M.Sc. COMPUTER SCIENCE

Staffordshire UNIVERSITY

DISSERTATION

Title

A Stock Market Prediction System, utilizing a combination of data mining algorithms on "gvol-candlesticks"© patterns and neural networks focused on an input smart pre-process system using technical indicators.

Module Leader	Author
PhD. Liolios Nik..	Nikitas Goumatianos

© *New term introduced by author*

Dissertation

A Stock Market Prediction System, utilizing a combination of data mining algorithms on "gvol-candlesticks"© patterns and neural networks focused on an input smart pre-process system using technical indicators.

The dissertation explores aspects and methods beyond technical analysis hoping to bring successful results in the stock market. Actually, it involves two different scientific approaches. The one approach is to discover unknown patterns that can be used for stock price predictions. And the other issue is to apply properly the technology of neural nets to predict the future of the stock prices.

The United States stock historical data used in this study starts from the year 1970 until May 2007. The sample selected for processing data is about 350 stocks.

Module leader

Professor: Liolios Nikolaos

Student

Goumatianos Nikitas

CONTENTS

Staffordshire UNIVERSITY

1. Introduction

1.1 Stock Market Prediction Systems

Many technologies have been developed in stock market prediction. Some of them have been used separately, other technologies have been combined together (Hybrid Technologies) in order to maximize the profits. Generally, all type of the investments can be divided into three main categories:

a) Those that involve studying of the Time Series Data of Indexes, stocks and commodities using special technologies (technical analysis, neural networks, etc),

b) those that analyze fundamental data and

c) those that predict the stock market by analyzing the influence of other markets, bonds, currencies etc (it is called “intermarket analysis”).

For the above categories there have been applied many technologies and methodologies in order to predict the stock market. Some of them have good results for specific type of stock markets or at specific periods of time or for specific stocks. But no one technique can guarantee absolute profitable results. This means that always occur errors during the prediction of the stock market. Which is the target? The target is to minimize the losses and maximize profits during a specific period of trading. The most well known technologies used for market prediction are the following:

- Technical Analysis. It involves identification of recurring patterns in historical price variation charts. It also uses technical indicators (functions) to predict the market. Its elements are normally limited to price (open, close, high, low), volume and open interested.
- Genetic Algorithms. Genetic Algorithms are problem solving techniques that possess an astonishing property – they solve problems by evolving solutions as nature does. They are generally used in combination with other technologies and can be applied in several ways in the financial field such as to optimize the inputs of a trading system, to construct rules etc.
- Neural networks. Neural Networks for prediction are used as special purpose indicators. Moreover, technical indicators can be used as inputs into neural network supposing that all values are normalized into the range of [0,1].
- Fuzzy Logic: It provides a methodology for reasoning using imprecise linguistic variables, rules and assertions.
- Fractal Analysis: It can analyze nonlinear systems and can be considered the main alternative for analyzing systems that defy development of predictive nonlinear

equations. Generally, most stock data variables show a fractal behaviour – if plotted without labels, it is difficult or impossible to decide which is the time scale.

- *Chaos & Nonlinear Dynamics*: Chaos theory is becoming an increasingly powerful tool for examining the way in which complex systems interact. According to chaos theory the observed price movements or asset returns are generated by a purely random process or by a process which includes a chaotic deterministic component.
- Fundamental Analysis: It analyzes fundamental data of various companies.
- Fast Fourier Transform: FFT can provide a mechanism for transforming the input data and as a result the indicators can perform better. Moreover, FFT can optimize the parameters of technical indicators (spectrum analysis), apply noise data removal and make cycle analysis.
- Data mining: It can be applied in various ways to predict the stock market.
- Pattern Recognition (Knowledge Representation). This technology is combined with data mining techniques.
- Elliot theory: It is a theory based on wave's patterns.
- Gann theory: Gann theory performs predictions of price movements on three premises: *Price*, *time* and *range* are the only three factors to consider. The markets are *cyclical* in nature and the markets are *geometric* in design and in function.
- Artificial multi-agents: They can demonstrate dynamic behaviours and have strong learning abilities. Also, they can build up an efficient portofolio management systems.

1.2 Concept of thesis's research

This thesis can be divided into two independent parts. The first part is to try to find patterns beyond the existing widely known candlesticks in order to predict the stock prices in the future and the second part is how to apply of neural nets in the field of the stock Market.

The first part of the work is focused on pattern recognition using data mining techniques. The initial motivation of work comes from a specific paper: "Mining for Profitable Patterns in the Stock Market", Yihua Philip Sheng – Southern Illinois University USA. This paper is attached as appendix "A". Although the basic idea comes from this paper, the algorithms as well as the range of data used are completely different. It actually discovers unknown patterns consist of two or three "gvol-candlesticks." This will be explained later.

The second part is the idea of utilizing neural networks to predict stock prices direction. The concentration in this area is focused on how to prepare the input values needed to train a neural net. It was selected values from technical indicators and trading systems based on technical indicators. In this case two key – words played a significant role to success: parameters optimization and adaptation. The first word optimization is to find the input parameters for each technical indicator that brings the best results. The word "adaptation" means that the optimization of each indicator is applied in each stock. As a result, each trading system parameters is depending on the stock. So, we developed a huge database which keeps and tells us in which stocks a specific trading system brings the best results.

1. 3. Background information

1.3.1 Mining for Profitable Patterns in the US Stock Market

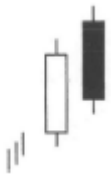
First it will be explained what is "gvol-candlesticks."

It is a proposed new type – an extension of candlesticks. In other words, gvol-candlesticks are candlesticks that include the following:

(i) gap information

(ii) Volume information

Generally, it is known that candlestick lines and charts are traditional Japanese charts whose individual lines look like candles, hence their name. The candlestick line is comprised of a real body and shadows. There have been published a lot of books which introduce patterns based on candlesticks that help to make profitable trades. For each formation it was given a defined name such as Dark Cloud Cover, Doji Star, etc.



Dark Cloud Cover formation

Doji Star formations

All these formations are known in the community of stock market traders and moreover the books present them in similar aspects. My question which may worth for further research includes four parts:

(i) Are there only the known formations of candlesticks or exist a lot of other hidden formations in the stock market?

(ii) Each stock market holds different type of patterns?

(iii) What will happen if we codify the formations of candlestick in more details?

(iv) What will happen if we take into account other important parameters such as volume and price gaps?

This work is focused on (i), (iii) and (iv) except (ii) which demands a lot of time of work. Actually, it was collected time series data from the stock market exchange of New York. These data comes from 350 stocks of US selected alphabetically from “A” to “LNG”. The number of Time Series Data is 1,561,023. All these data are inserted in a proper database (RDBMS SQL Server 2005) and was used proper data mining techniques. Also, it combined with a proper program in Microsoft C# which helped in discovering profitable patterns. The selected stocks and the days of starting and ending of collected data for each stock are depicted in appendix “B”. All results from this coursework are depicted in appendix “C”.

1.3.2 Stock market prediction by neural networks

There has recently been an avalanche of papers in which neural models are applied to financial and economical forecasting. In most cases surprisingly good results have been obtained. The novelty about neural networks lies in their ability to model non-linear processes with a few assumptions about the nature of the generating process. This is particularly useful in financial engineering applications where much is assumed and little is known about the nature of the processes determining directions of the stock market.

Although there are many types of neural network architectures, it will be chosen the back-propagation network because it is considered to be the most general and flexible for the purposes of the technical analysis. Identifying appropriate inputs is as equal important as the architecture design of the neural net. The main focus will be on the construction of a proper system for selecting, optimizing and normalizing the data inputs.

Designing a neural network to have minimal degrees of freedom is critical when attempting to predict stock markets. Degrees of freedom in a neural network are determined by the number of inputs, the number of hidden layers, the number of hidden nodes and the number of connection weighs.

The development of this prototype neural network required extensive experimentation, but the results sounds academically interested. This type of applied

methodology referring to input optimization is something new in this area. The optimum selector system which will be developed will select different technical indicators for each stock depending on their profits.

1.4 Ethics

For the completion of this dissertation it will be given both academic and professional respect to the related physical or legal persons involved, especially to the following:

- The University of Staffordshire
- ATEI Department of Computer Science Technology and Telecommunications.
- The faculty staff (supervisor, module leaders).
- The related companies whose software has been used.
- The research work of persons which has been used.

For this specific dissertation it should be given a respect referring to any piece of work or artifact. This is also important in the case of comparison with the performance of any other existing methodologies.

If the results from this research are very profitable, this may provoke the interest of commercial companies. It should be cleared that copyrights of this research should be protected and any possible usage outside the University of any piece of work must be given permission by author.

1.5 Deliverables

The deliverables are corresponding to the aims and objectives which are set distinctively and include:

- A literature report for this research.
- A report of this dissertation which will contain the construction of the neural network, methodologies and results of the data mining research for rule construction based on gvol-candlesticks patterns and the expert system.
- CD which contains:
- Software application executable of this research
- Source code of the application.
- Database design and backup time series data.
- Dissertation Report

- Literature Report

Depending on the success of this research it is programmed to be produced two papers with the following titles:

- Stock Market: The introduction of new type of candlesticks called “gvol-candlesticks” and the results of applying mining methods for profitable patterns based on them.
- Stock market prediction by utilizing neural networks focused on an input smart preprocess system using technical indicators.

Staffordshire UNIVERSITY

2. System Architecture Report

2.1 System Architecture Report

The system consists of two sub-systems:

a) Two main processing units

b) The output / results unit.

The “processing units” work independently and they implement two different technologies. The one as referred prior is the creation of the gvol-candlesticks patterns. Actually, the investor can select a group of stocks and then automatically create the patterns. The patterns which are in bit format (0 or 1) should be saved in the database for further process.

The second processing unit consist of the following sub-units:

- The smart pre-process input unit responsible for optimization and adaptation jobs. For each stock follows the parameters optimization process which bring the best profitable results for the specific stock. These parameters for each stock can be saved into database in order to be used later from neural nets.
- The next step is the neural net creation, training, testing and saving results into database. The neural net receives two groups of indicators: the one group consists of trading systems indicators which result in 1 (=buy) or 0 (=sell) signal. The other group consists of indicators that not produce signals.

The output / result unit produce what the stock price direction of the next day. For each specific stock is selected the appropriate neural net to produce a result – buy or sell signal. Additionally, a check for possible gvol-candlestick pattern can be used as advisor for the price direction for the next days.

The next figure 2.1 displays the architecture of the proposed system. A part of database contains the intelligence data:

- Data for optimized parameters of all indicators and each specific stock.
- Data for all hidden gvol-candlestick patterns
- Data of all neural testing results for each individual stock.

The main parts of the system and their relations are displayed in the diagram of the next page.

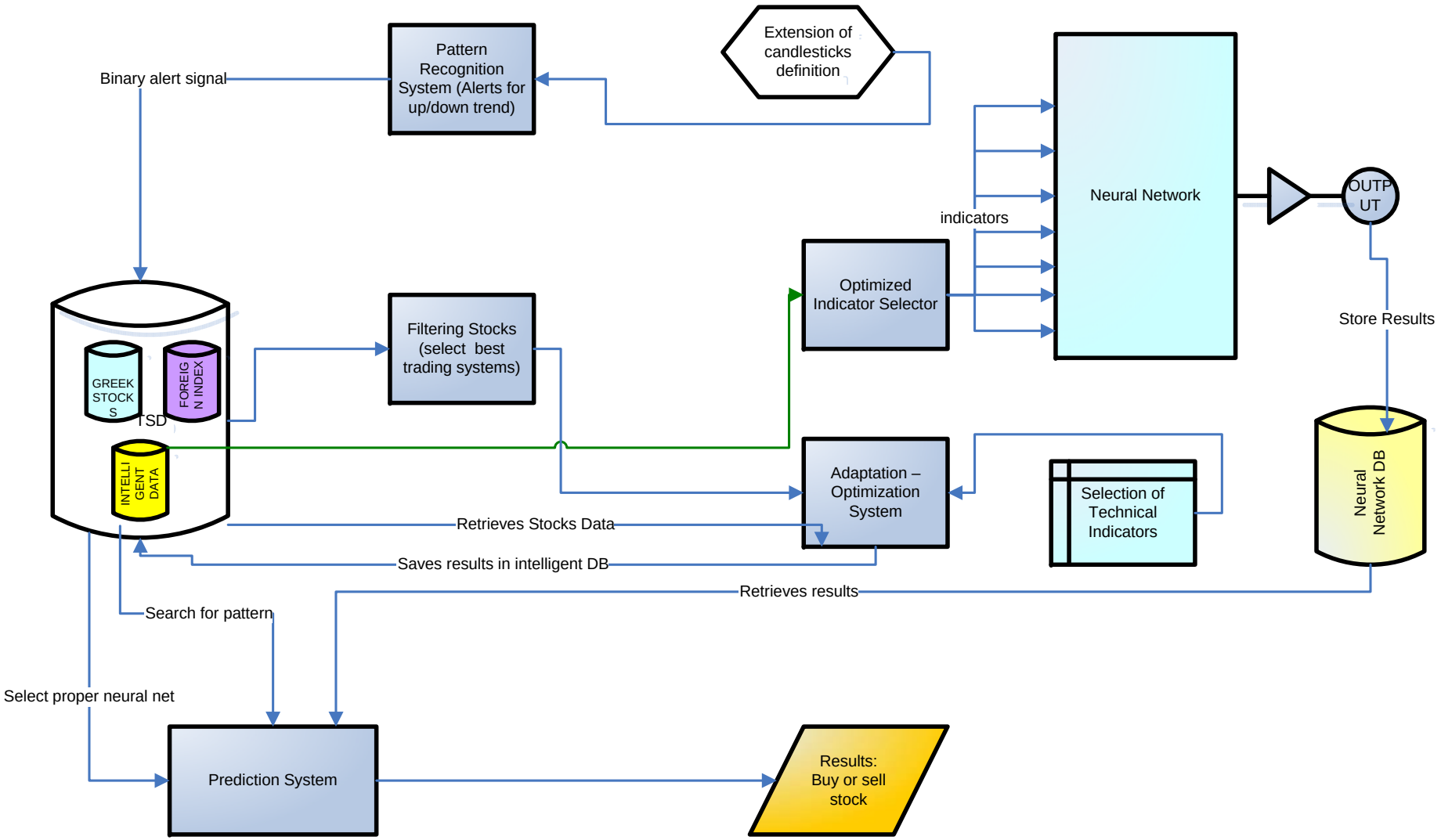


Figure 2.1: *System Architecture Report*

2.2 Database Diagram

The design was developed by using the Sybase Power Designer program. Details and the tables, fields and relations are displayed in the following pages:

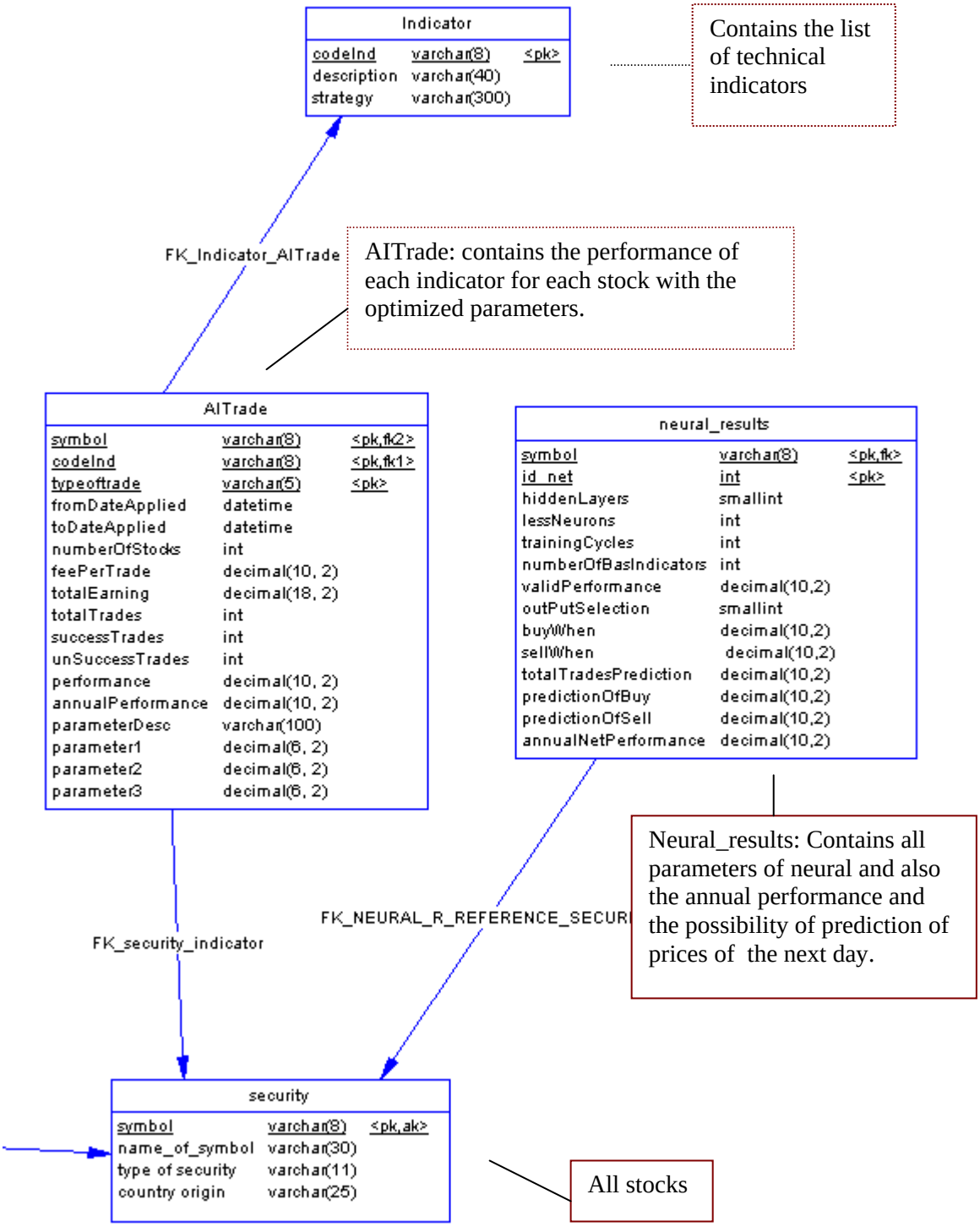


Figure 2.2.1: *Diagram part 1*

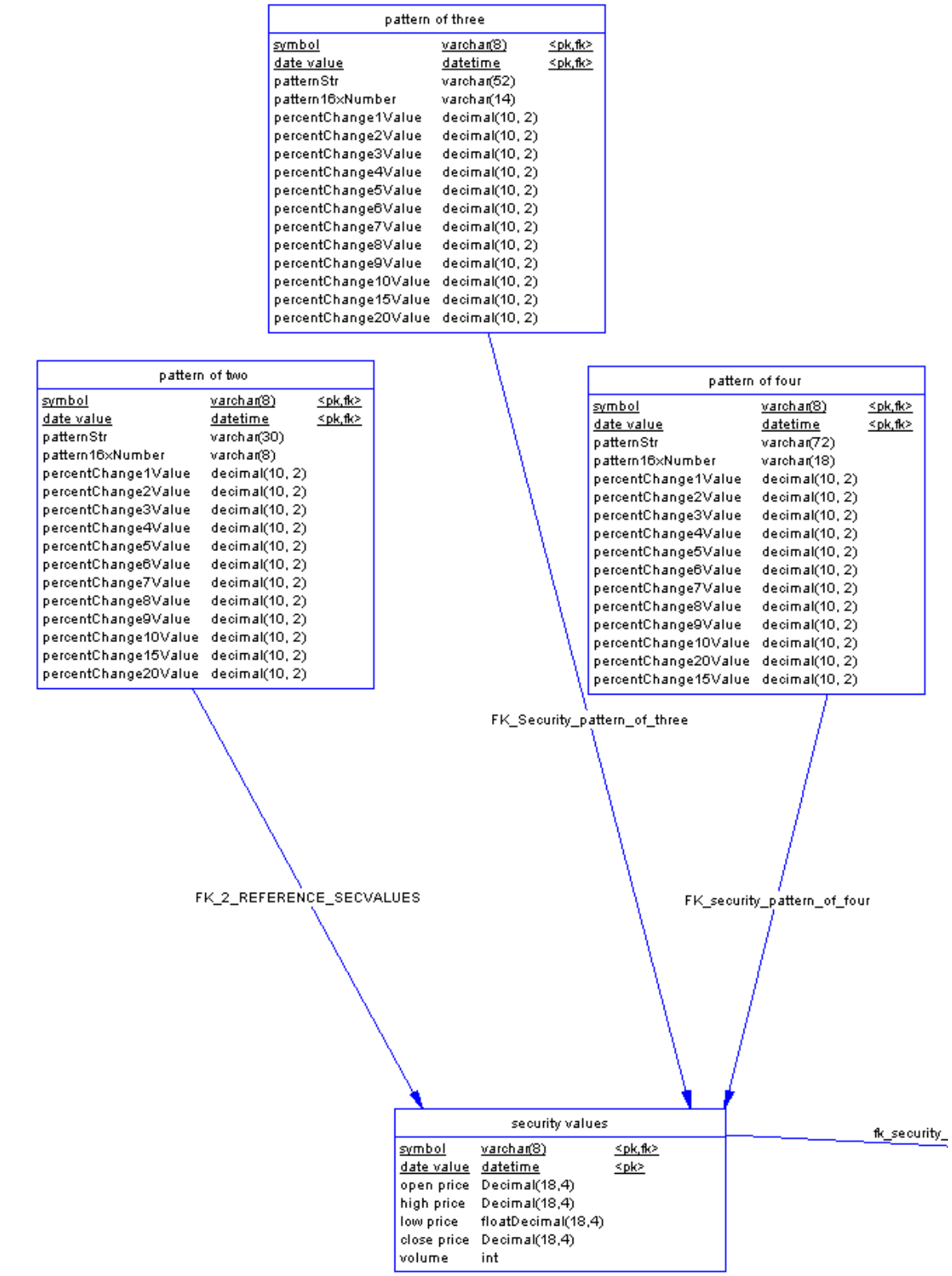


Figure 2.2.2: *Diagram part 2.* The security values (stocks daily data) and the patterns of 2, 3 and four gvol-candlesticks. Percent Change x Value means what happens in x days in comparing to the current day close price.

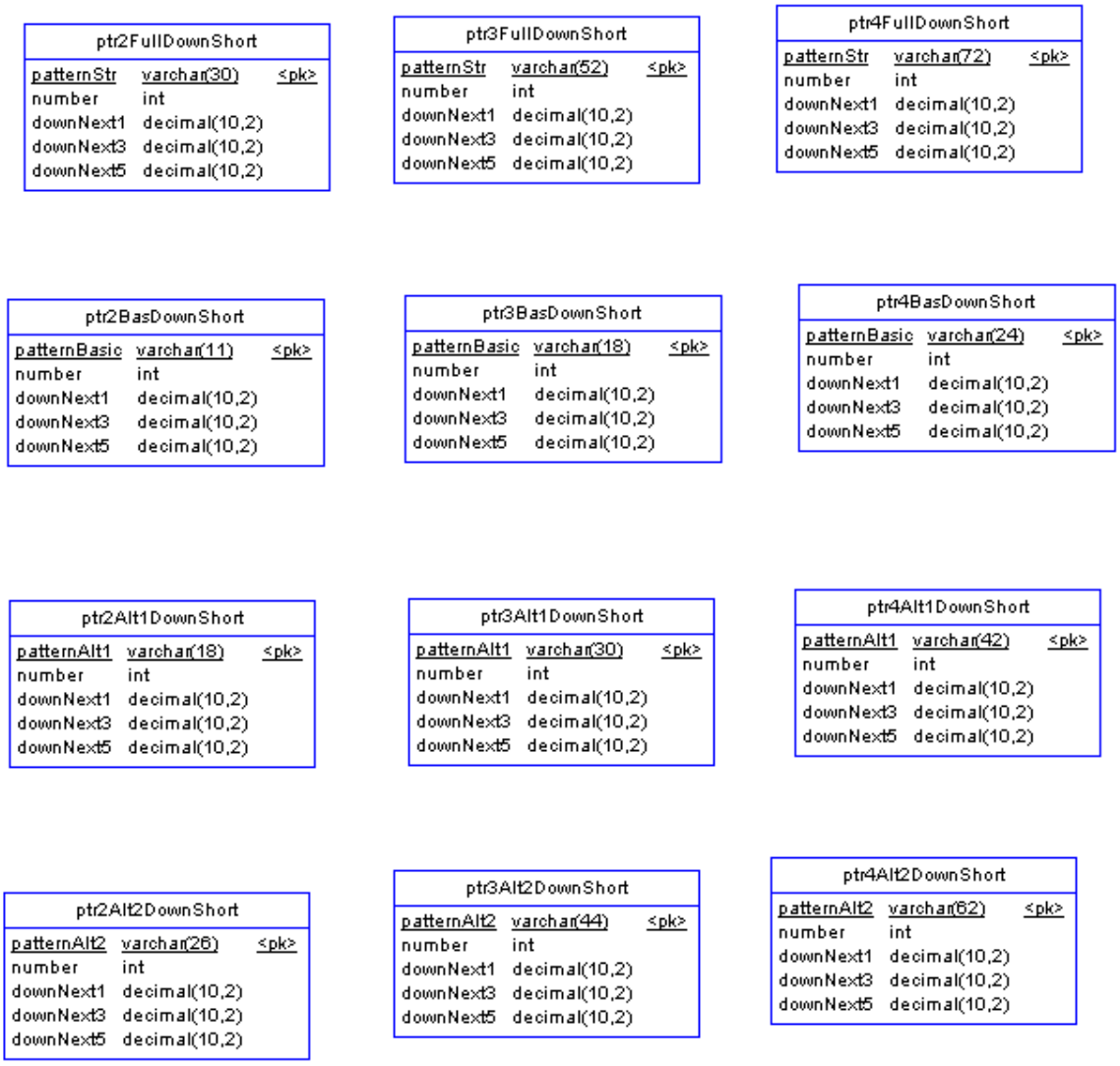


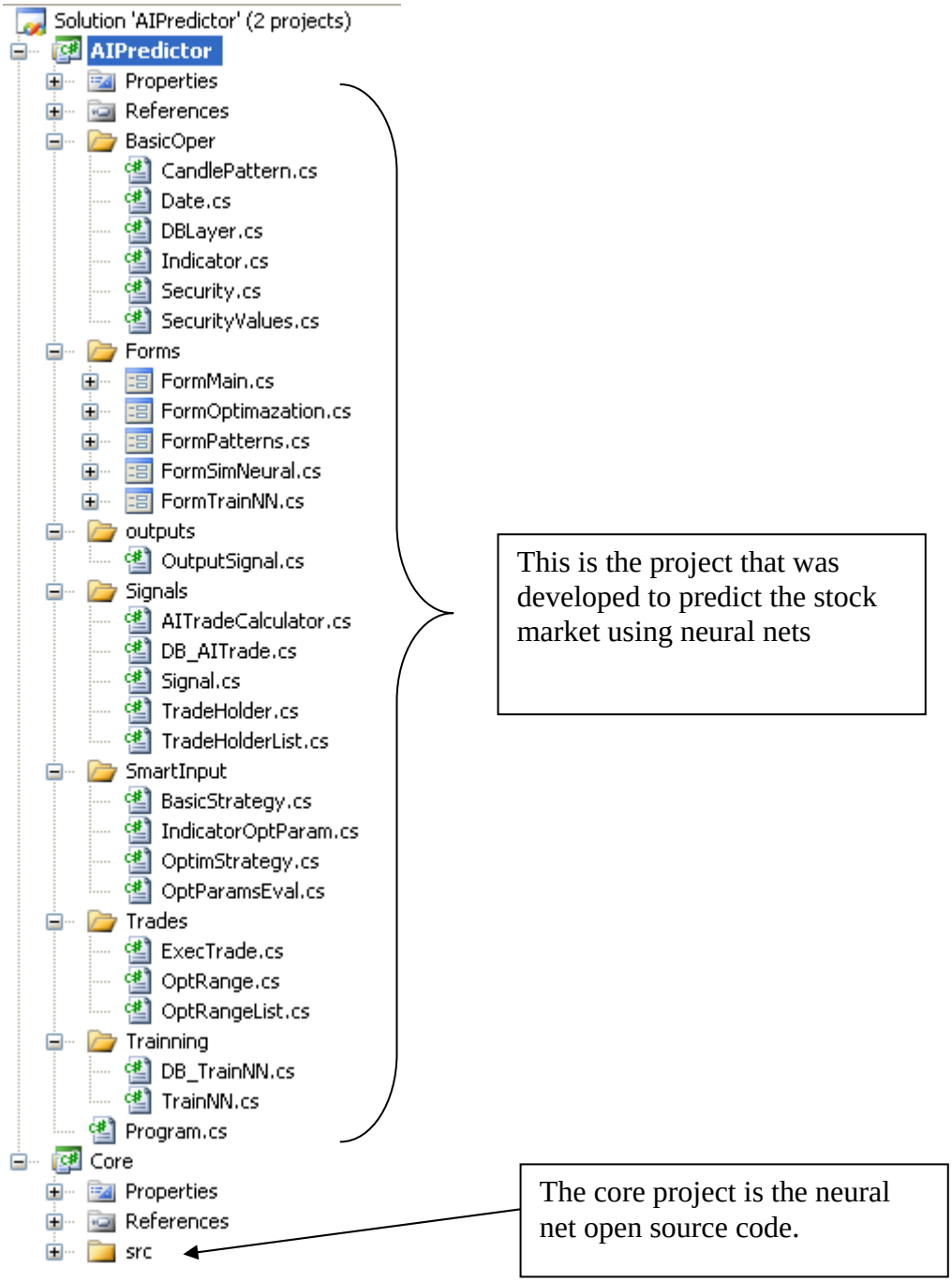
Figure 2.2.3: *Diagram part 3.*

There are displayed all summarized tables for all type of combinations of patterns.

Note:
There were created views and procedures which are in appendix.

2.3 Class Diagrams

The project was developed using Microsoft Visual Studio 2005. Classes are organized as the following picture:

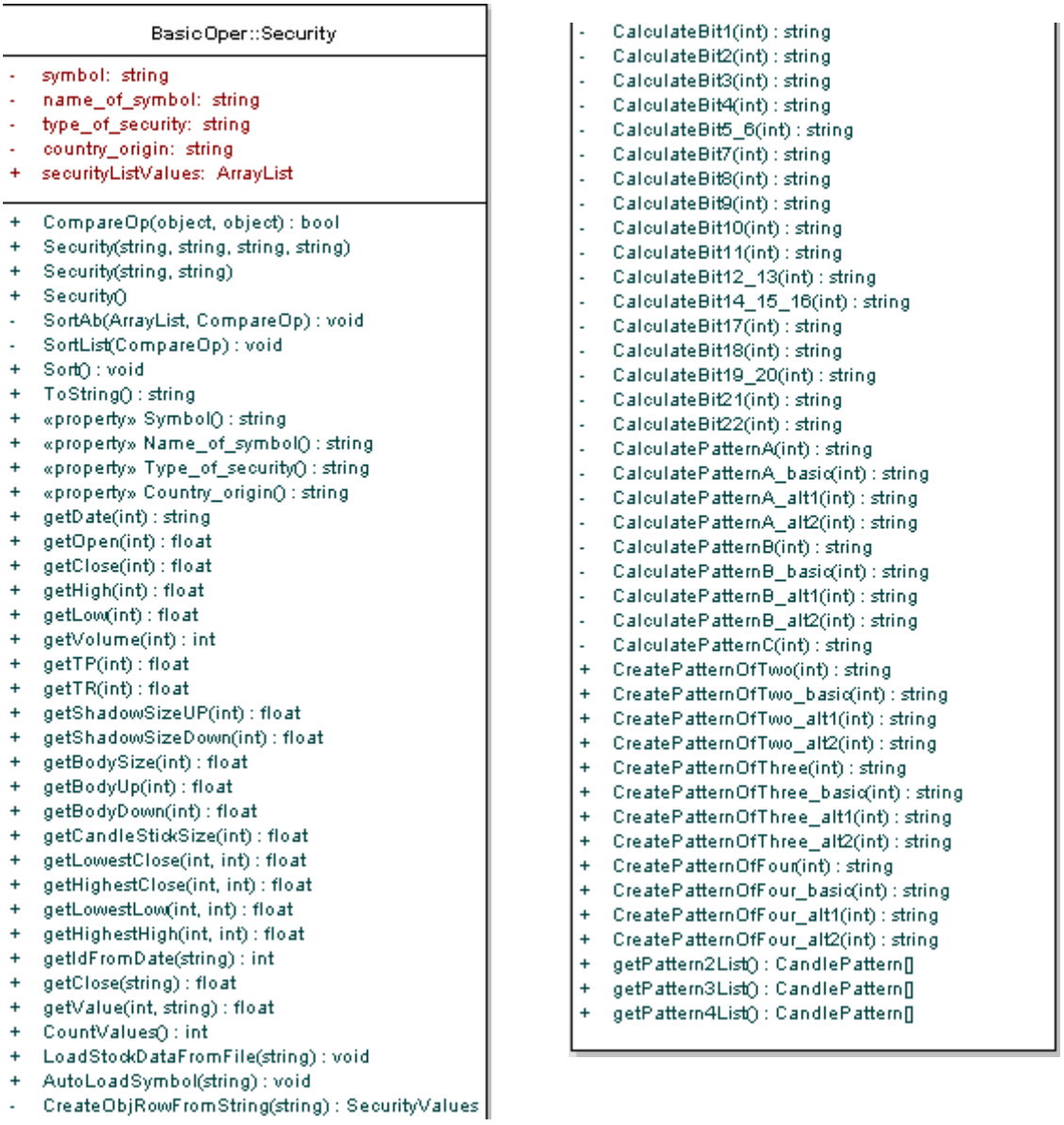


The next diagrams represent the details of the classes of the project.

Name space: BasicOper

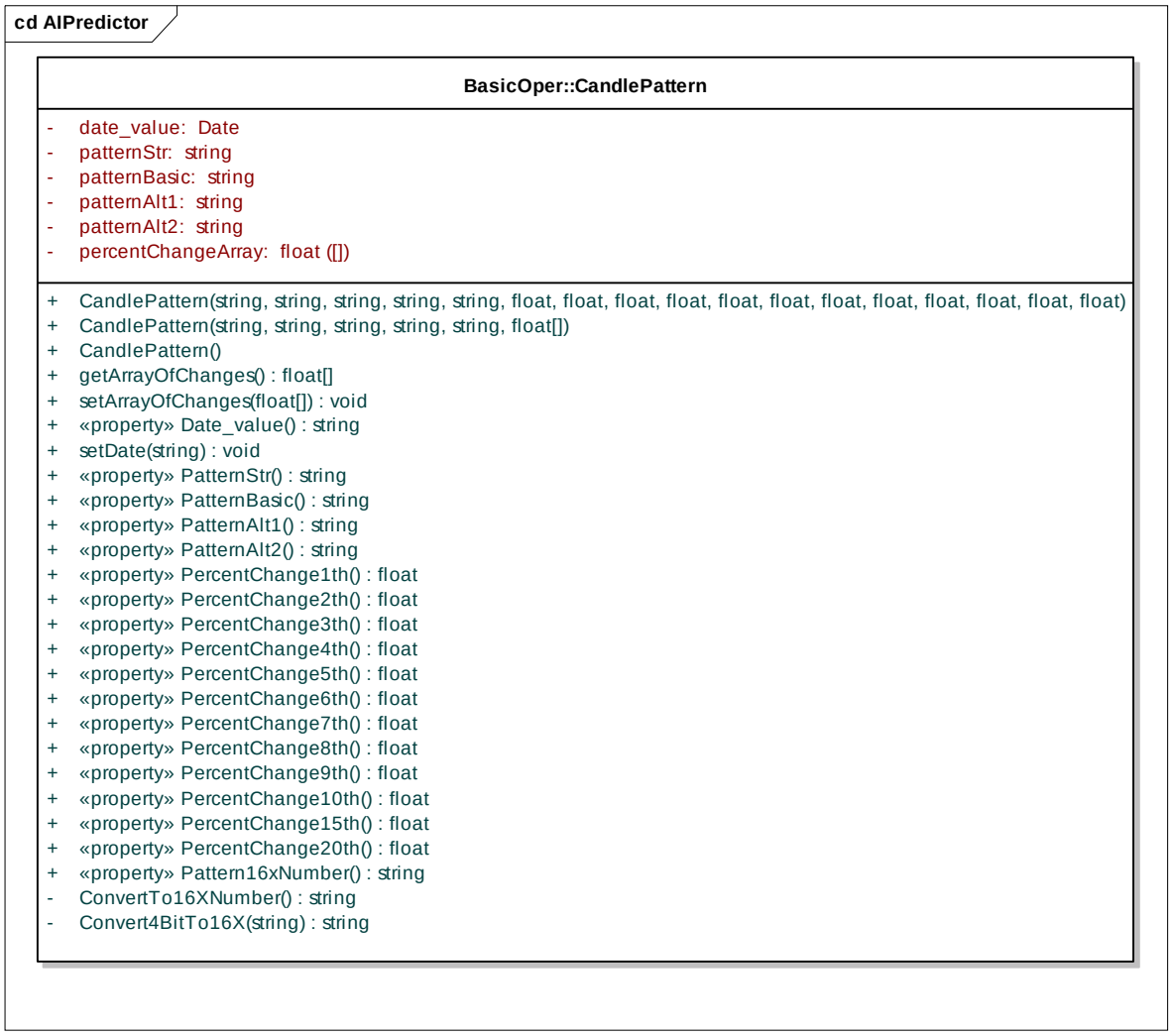
Class: Security

Purpose: It represents a stock including all detail values. Additionally, contains the codification system for creation of the patterns of gvol-candlesticks.



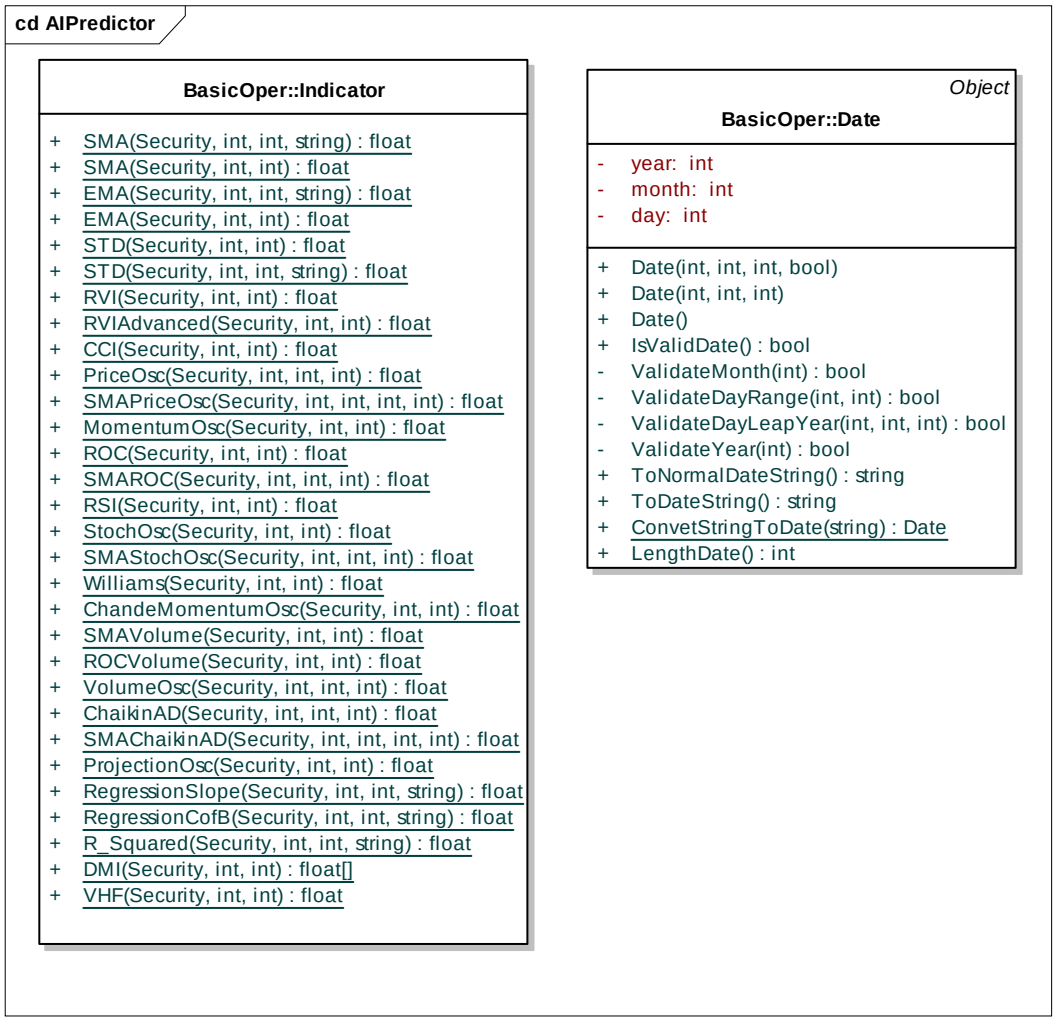
Class: CandlePattern

Purpose: `It is used for creation of gvol-candlestick patterns`



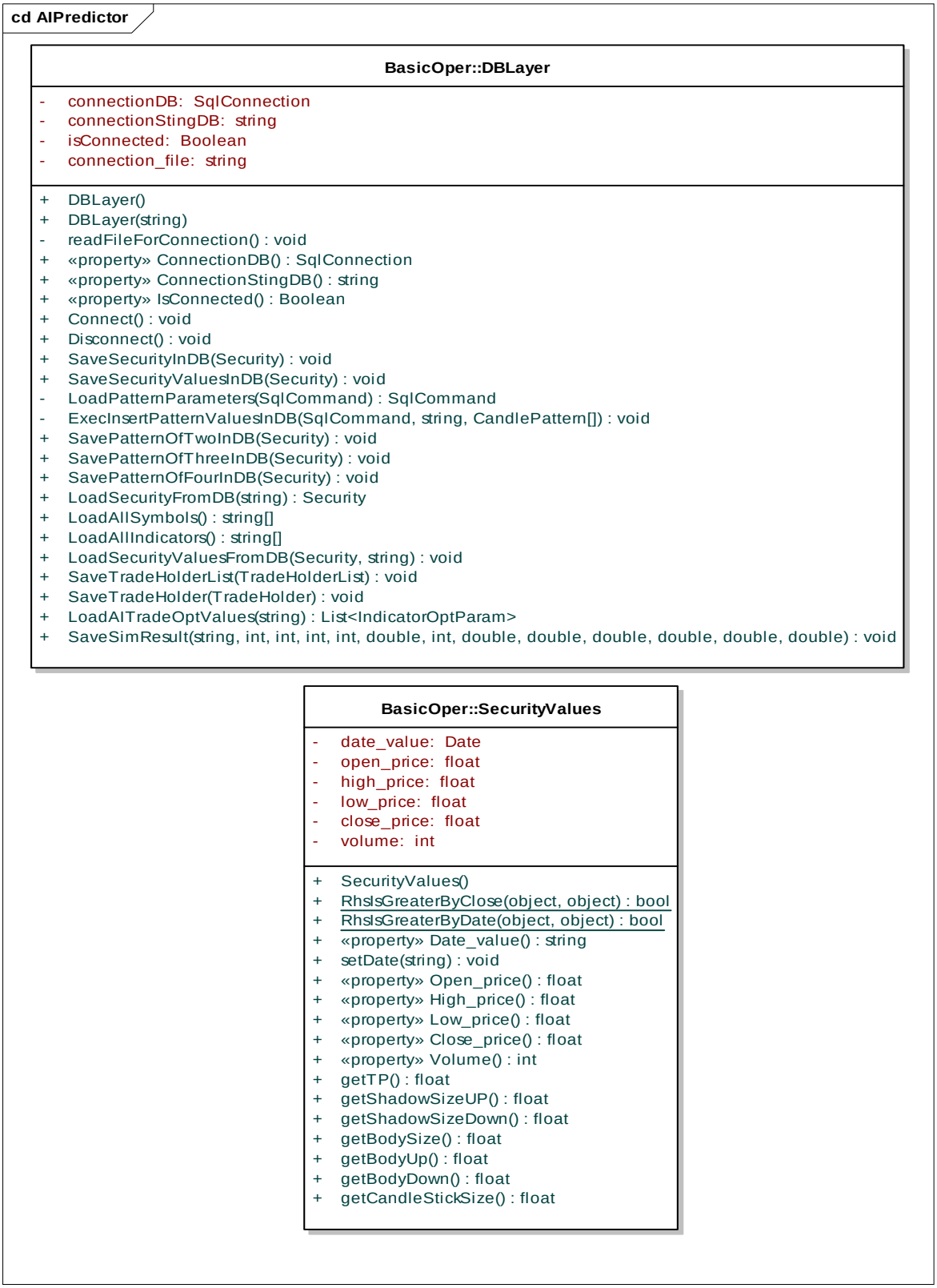
Class: Indicator
Purpose: All technical indicators used in this project

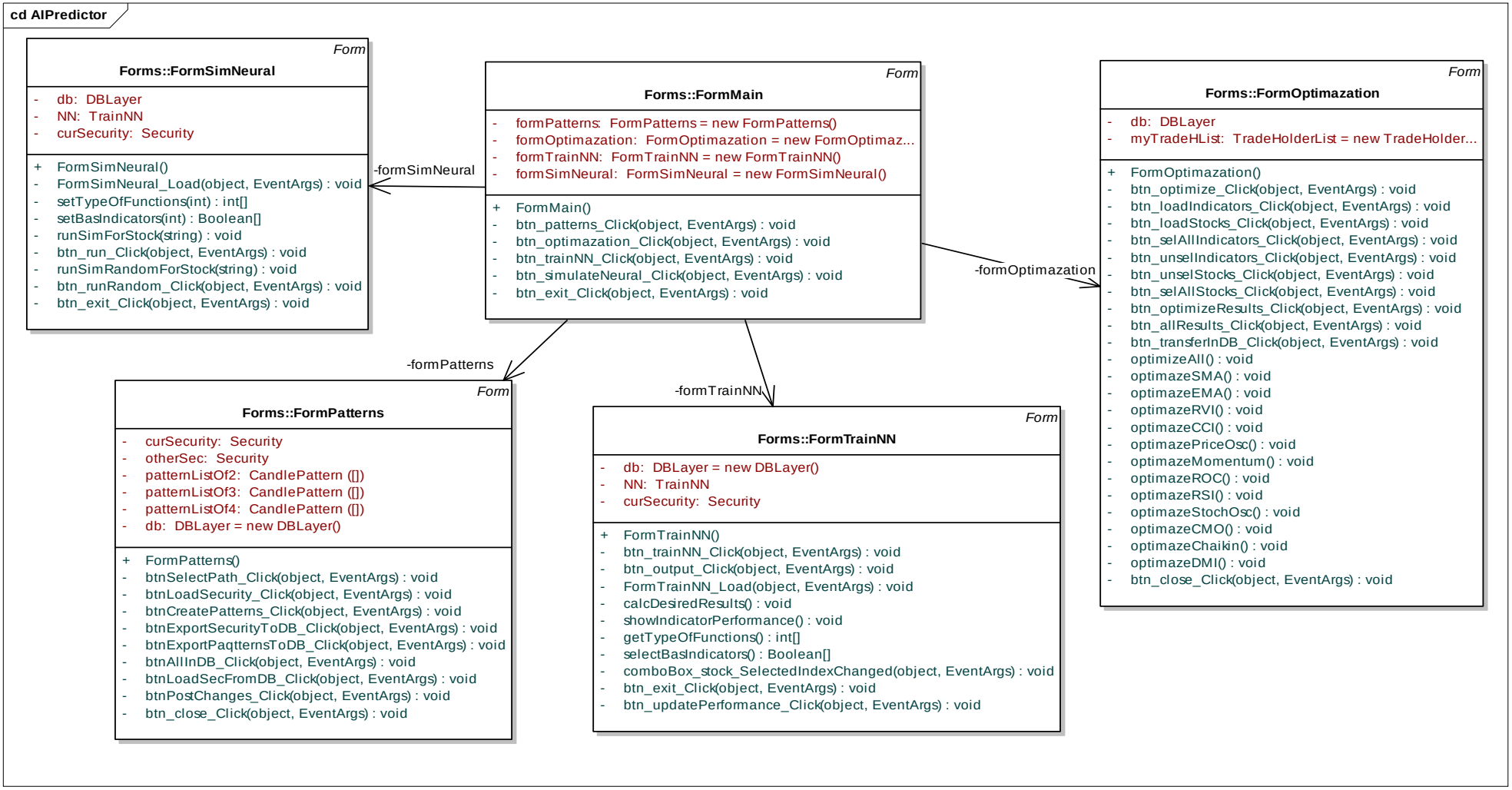
Class: Date
Purpose: It is used for representing the date object used in stocks



Class: DBlayer
Purpose: The handling of database of the project

Class: SecurityValues
Purpose: Handling the details of the stock values

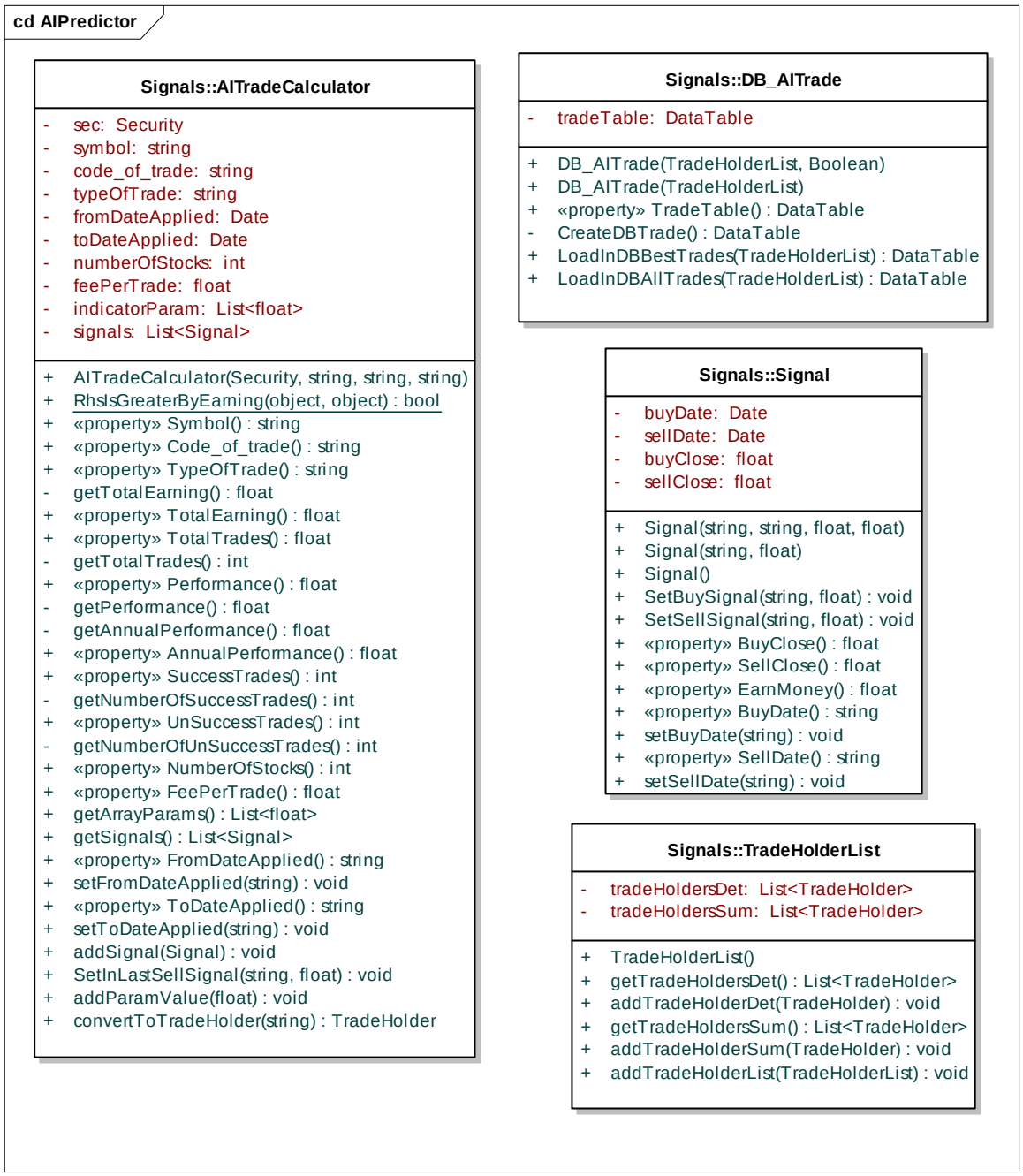




Class Diagram of Forms

Class: AITradeCalculator

Purpose: It calculates the performance of specific trading system.
It takes the security class and the signals and calculates the performance of the system



Class: DB_AITrade

Purpose: DataTable to hold the information of results of trading. Trading results can be displayed**.**

Class: Signal

Purpose **:** Object holder for signals producing by a trading system**.**

Class: TradeHoldeList

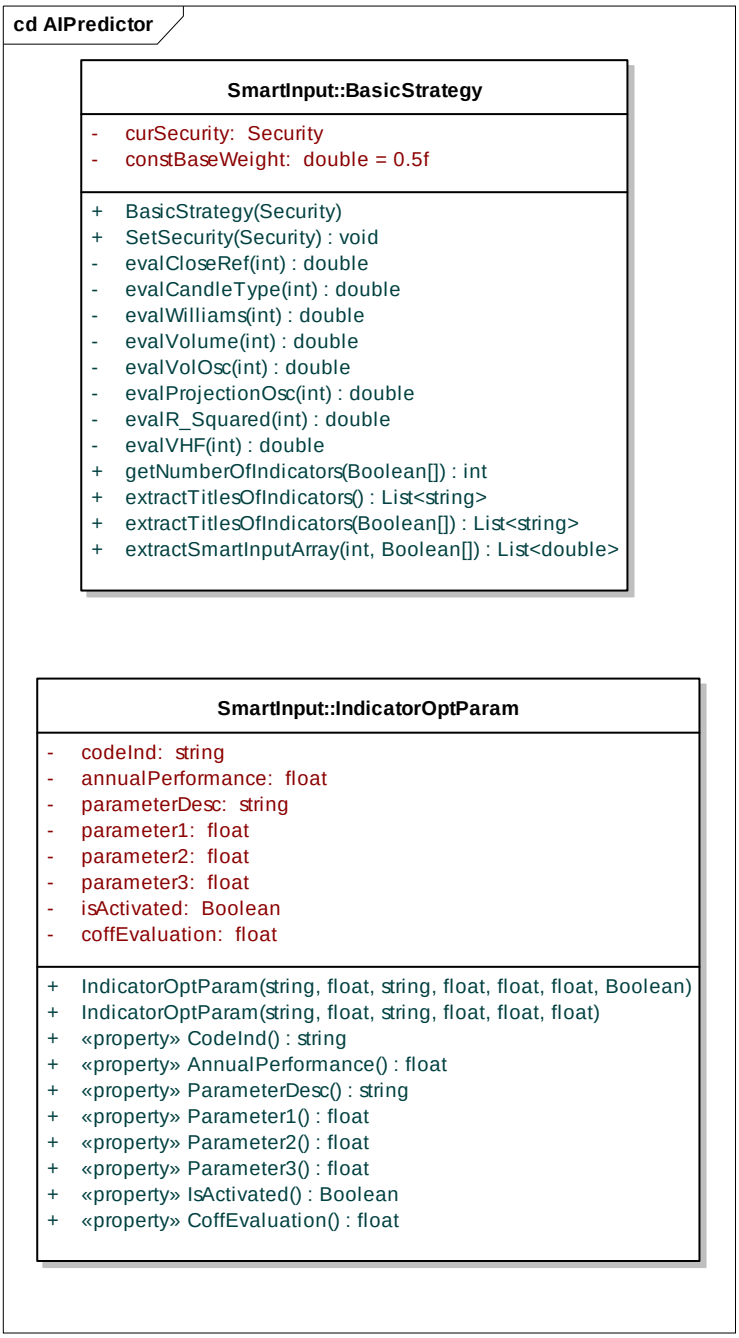
Purpose **:** Used for holding list of TradeHolder objects

Class: TradeHoldeList

Purpose **:** It is used to hold all relative information of a stock and specific trading system. This class is used for transfering data into db.

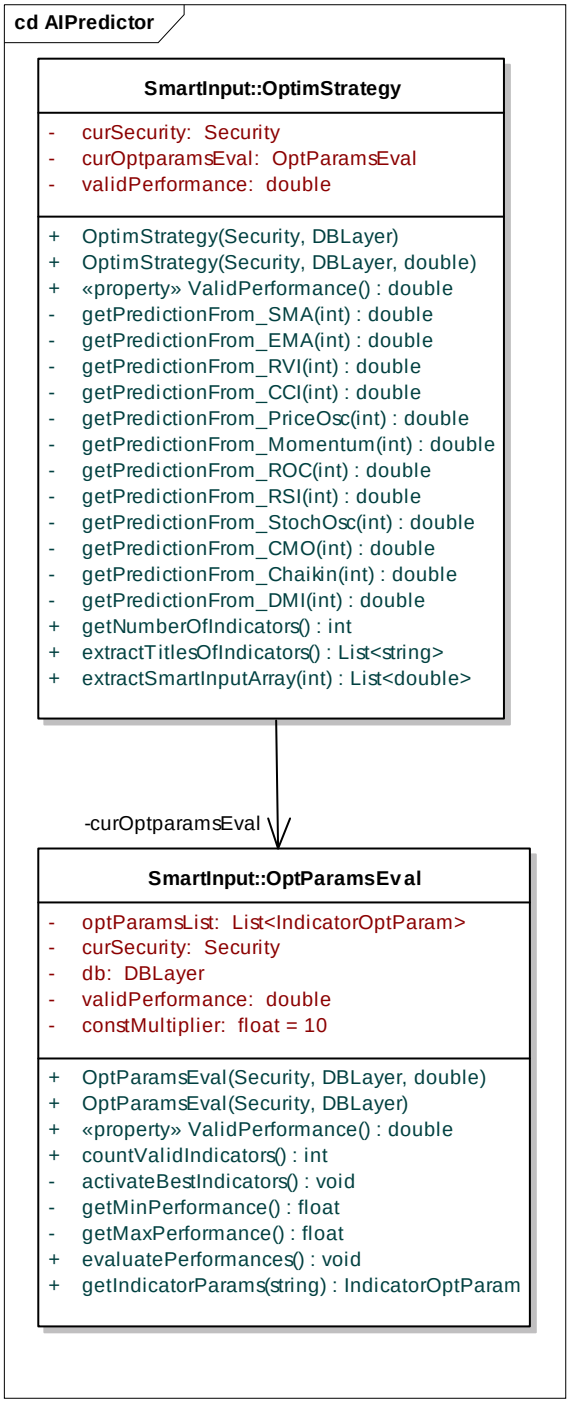
Class: BasicStrategy
Purpose: Evaluation of specific technical indicators that can not produce trading signals. The result (output) is to be usedas input to the neural net.

Class: IndicatorOptParam
Purpose: Holds the best parameters (input) values of a technical indicator.



Class: OptimStrategy
Purpose: Produces the input values for the neural net. It normalize values (0 - 1). Signals for buy (long position) return 0.999 otherwise 0.001

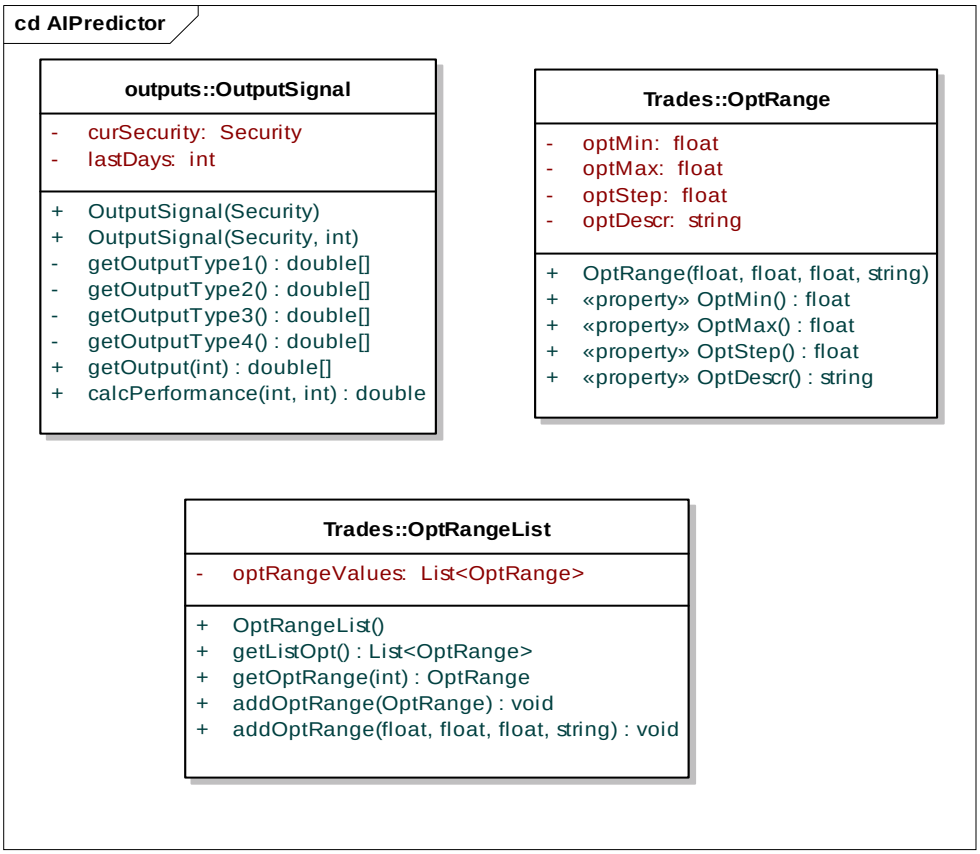
Class: OptParamsEval
Purpose: Used for loading all relative trading information about a stock. It actually loads from db all performance data of all trading systems referred to a specific stock.



Class: OutputSignal
Purpose: It is used to calculate the maximum results of hypothetical training. It should bring the best results for trading. It implements 4 differents methods of estimation.

Class: OptRangeList
Purpose: Holds the list of all parameters of an indicator

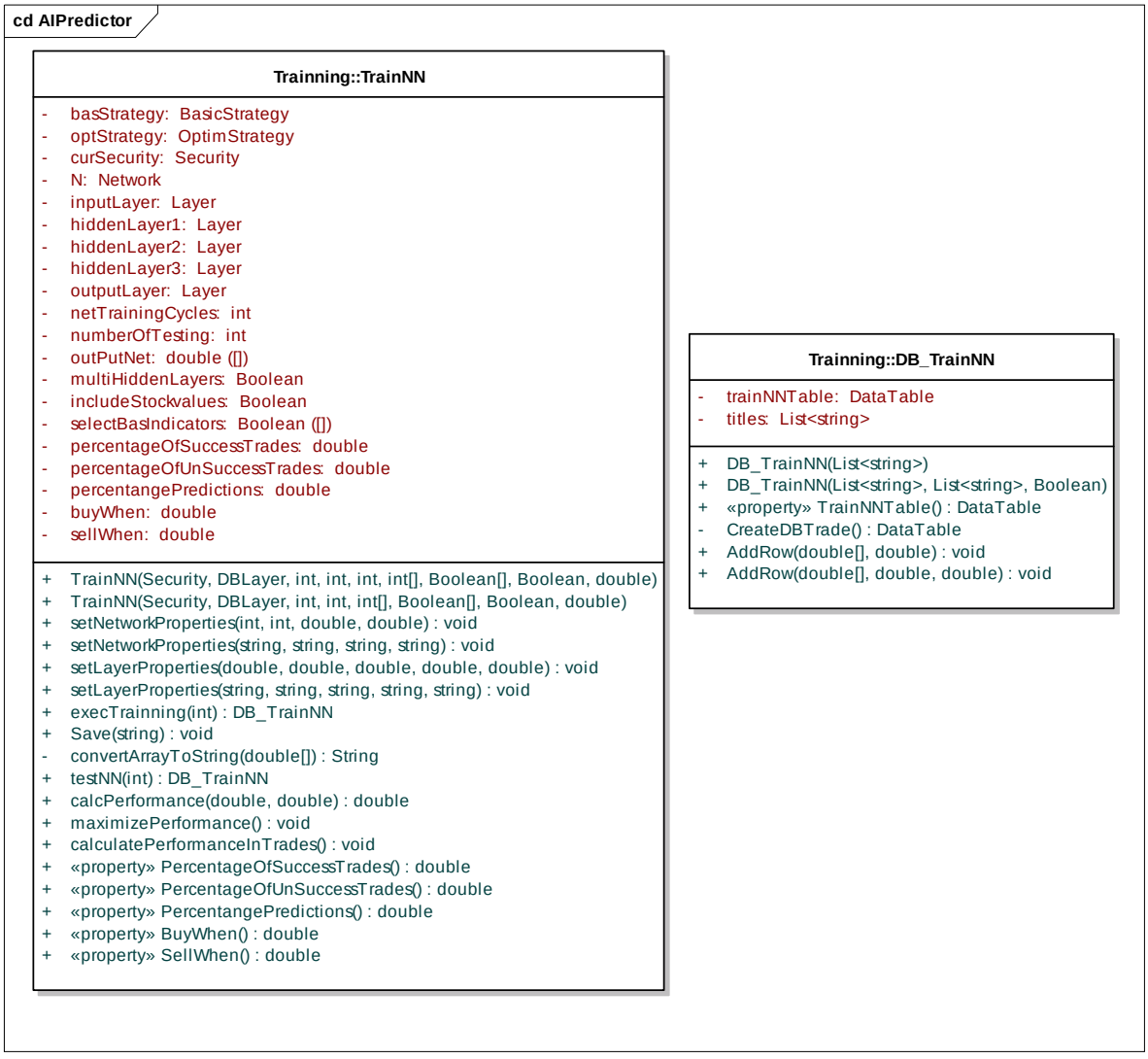
Class: OptRangeList
Purpose: Holds the list of all parameters of an indicator



Class: ExecTrade
Purpose: It contains all trading systems. All optimization methods for defining the best input parameters of indicators.

Class: TrainNN
Purpose: To train and test the neural net. It prepares the input data and gets the results. It includes a lot of customization.

Class: DB_TrainNN
Purpose: Holds the data the input and the output of the neural net in the form of a DataTable object in order to be able displayed in a dataGrid.



Staffordshire UNIVERSITY

3. Discovery for Hidden Patterns

3.1 Knowledge Representation Process

The key of the success of such work is being held by a proper knowledge representation. The proposed representation is not only based on K-Lines as in the paper (appendix A) but also in extension form of this (gvol-candlestick). According to the paper (appendix A) the knowledge representation is created by codifying information of K-Lines in a binary system. In my coursework it is followed similar way but the differences are the following:

(i) The holding information is mainly categorized in two parts. The part A holds information about the K-line point itself (without relationships with prior K-Lines). The part B holds only information about relationships. The advantage of this is that it can be easily translated into a program that can include patterns consist of any number of K-Lines (two, three, four, five…formations).

(ii) The holding information in my coursework is extended in much details comparing to the paper (appendix A).

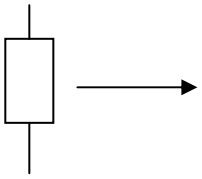
(iii) Additionally it includes information for gap and volume.

3.1.1 The Bit codification System

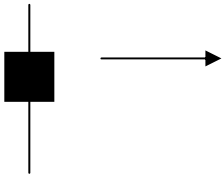
The proposed bit codification system is divided into three parts. The part A includes information related in one only gvol-candlestick. The parts “B and C” include information about the relationships among gvol-candlesticks.

Knowledge Representation: **Part B**-> The gvol-candlestick itself

Bit position 1: Define the type of candlestick, is it white or black?



Returns “1” (is white)

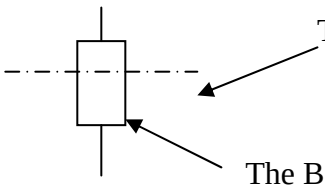


Returns “0” (is black)

Here is the code in C#:

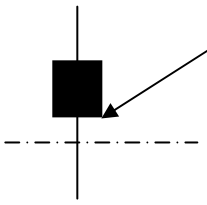
```
private string CalculateBit1(int id)
{
            if (getOpen(id) < getClose(id))
            {
                return "1";
            }
            else
            {
                return "0";
            }
 }
```

Bit position 2: Is the bottom of the body of the candlestick above of the mean of High and Low price? If yes, then return “1”, otherwise return “0”



The mean (middle of whole candlestick) of High and low.

The BodyDown is lower than the middle of the candlestick. So it returns “0”



The BodyDown is upper than the middle of the candlestick. So it returns “1”

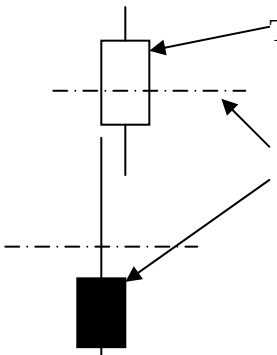
The mean of high and low (in the middle)

Here is the code in C#:

```
private string CalculateBit2(int id)
{
            if (getBodyDown(id) > (getHigh(id) + getLow(id)) / 2)
                return "1";
```

```
            else
                return "0";
 }
```

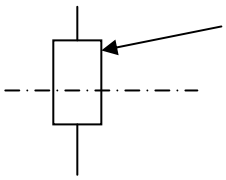
Bit position 3: The upper of the body of the candlestick is lower than the mean of High and Low price? If yes then return “1”, otherwise return “0”



Here is the code in C#:

```
private string CalculateBit3(int id)
{
            if (getBodyUp(id) < (getHigh(id) + getLow(id)) / 2)
                return "1";
            else
                return "0";
 }
```

Bit position 4: Where is the position of close value? If it is upper from the middle of the candlestick returns “1” otherwise returns “0”.



Here is the code in C#:

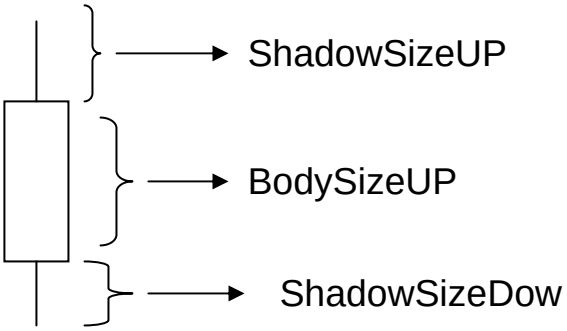
```
private string CalculateBit4(int id)
{
            if (getClose(id) > (getHigh(id) + getLow(id)) / 2)
                return "1";
            else
                return "0";
}
```

Bit positions 5 and 6: It presents the percentage of difference between open and close prices. If the close value is 3% lower (or more) than the open, then, it returns “00”. If close percentage change is between -3% and 0% it returns "01". If it is between 0% and 3% it returns “10” otherwise returns (>3%) “11”.

Here is the code in C#:

```
private string CalculateBit5_6(int id)
        {
            float val = 100*(getClose(id)-getOpen(id))/getClose(id);

            if (val <= -3)
               return "00";
            else if ( (val >-3) && (val < 0))
                return "01";
            else if ( (val >=0) && (val < 3))
                return "10";
            else
                return "11";
        }
```

Bit position 7: Compares the sum of Shandow up and Shadow down with the Bodysize.



Here the BodySize is greater than the sum of the ShadowUp+ShadowDown.

So, It returns “0”.

Here is the code in C#:

```
private string CalculateBit7(int id)
{
  if ((getShadowSizeUP(id) + getShadowSizeDown(id)) >

                                                getBodySize(id))
                return "1";
```

```
            else
                return "0";
}
```

Bit position 8: Now, it compares only the Shadow Size Up with the size of the Body.

Here is the code in C#:

```
private string CalculateBit8(int id)
{
            if ( getShadowSizeUP(id)  > getBodySize(id))
                return "1";
            else
                return "0";
}
```

Bit position 9: Now, it compares only the Shadow Size Down with the size of the Body.

Here is the code in C#:

```
private string CalculateBit9(int id)
{
            if ( getShadowSizeDown(id) > getBodySize(id))
                return "1";
            else
                return "0";
}
```

Bit position 10: It involves the volume. If the current volume is greater than the average volume of 20 days then return “1” otherwise return “0”.

Here is the code in C#:

```
private string CalculateBit10(int id)
{
            float sumVolAvg = 0.0f;

            // Calculate the average volume of 20 days:
            for (int i = 0; i < 20; i++)
            {
                sumVolAvg += getVolume(id - i);
```

```
            }

            //Finally, the average volume is:
            sumVolAvg = sumVolAvg / 20;

            if (getVolume(id) > sumVolAvg)
                return "1";
            else
                return "0";
 }
```

Knowledge Representation: **Part B**-> The relationships between two candlesticks (DAY 1 – DAY 2)

Bit position 1: Compares the close value with the previous ones. If the current close price is greater than the previous close day (stock price has increased) it returns “1” otherwise returns “0”.

Here is the code in C#:

```
private string CalculateBit11(int id)
{
            if (getClose(id) >= getClose(id-1))
                return "1";
            else
                return "0";
 }
```

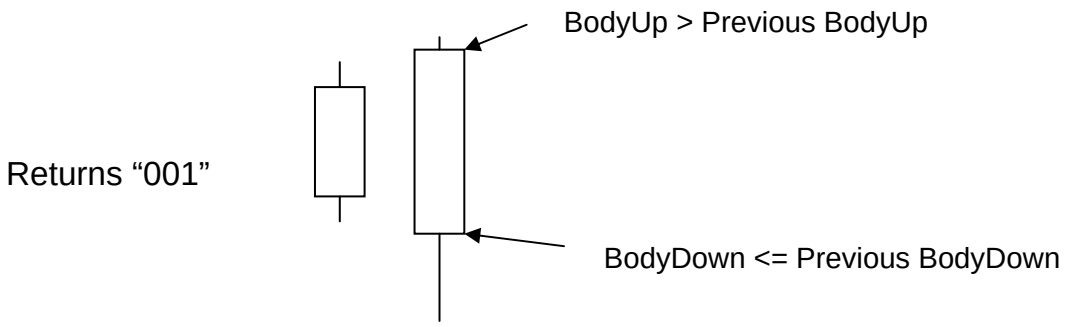
Bit positions 2 and 3: It calculates the percentage of close price. The range is divided by 4 regions:

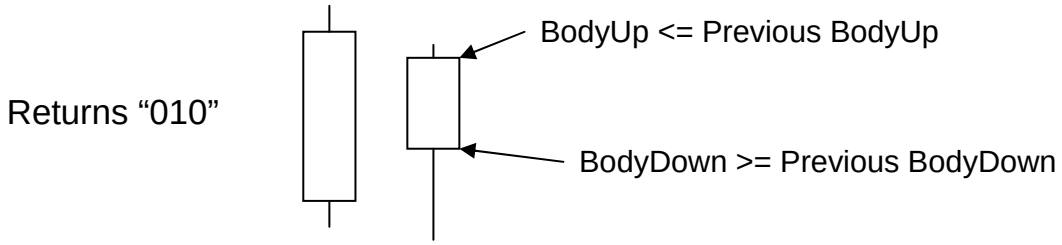
- Percentage change of close price is less than -3% , then it returns “00”
- Percentage change of close price is between -3% and 0% then it returns “01”
- Percentage change of close price is between 0% and 3% then it returns “10”
- Percentage change of close price is greater than 3% , then it returns “11”

Here is the code in C#:

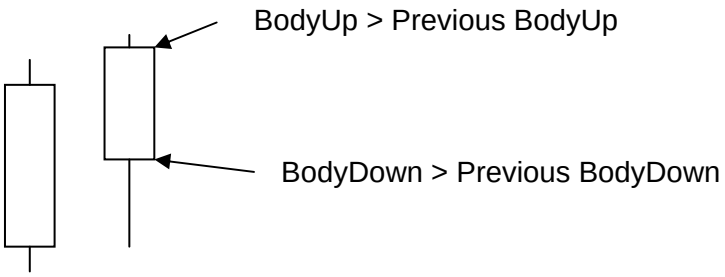
```
private string CalculateBit12_13(int id)
{
   float val = 100 * (getClose(id) - getClose(id - 1)) / getClose(id-1);

            if (val < -3)
                return "00";
            else if (val>=-3 && val < 0)
                return "01";
            else if (val >= 0 && val < 3)
                return "10";
            else
                return "11";
}
```

Bit positions 4,5 and 6: This involves the exact position between two candlesticks. The following figure represents the different positions with relative values:

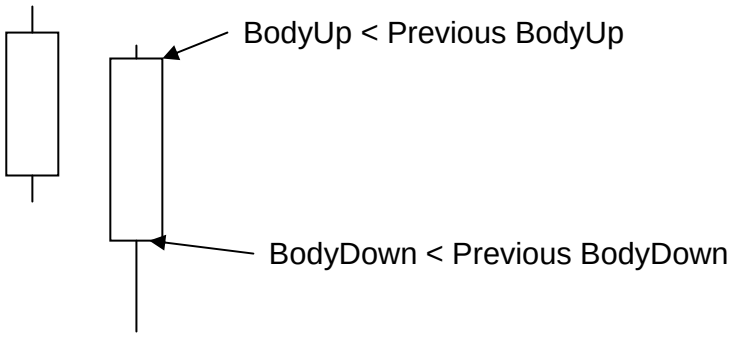




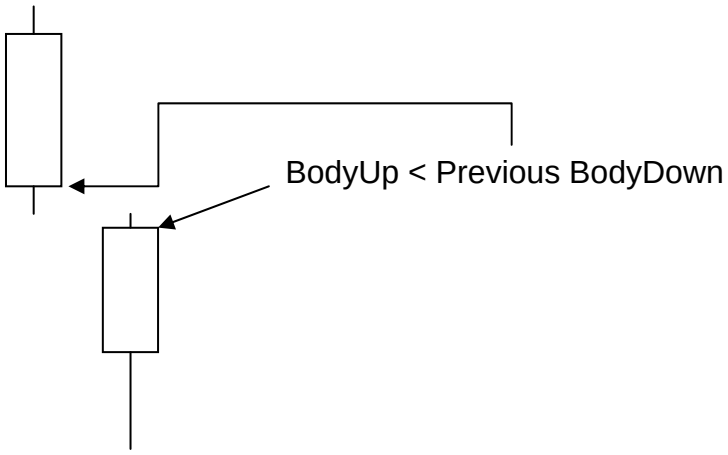
Returns “011”



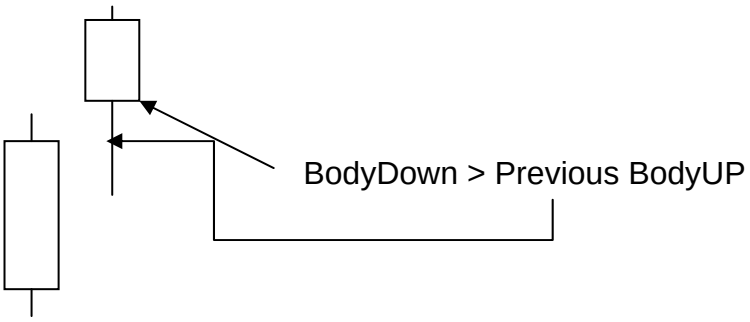
Returns “100”



Returns “101”



Returns “111”

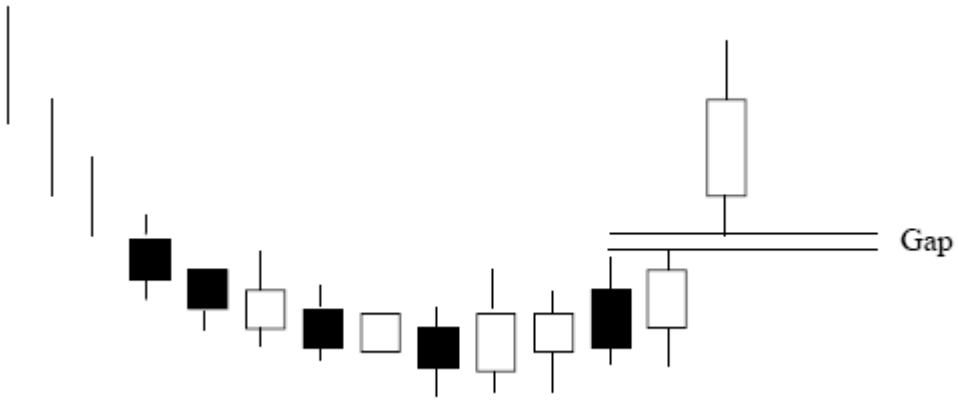


Here is the code in C#:

```
private string CalculateBit14_15_16(int id)
        {
            if ( (getBodyUp(id) > getBodyUp(id-1)) && (getBodyDown(id) <=
getBodyDown(id-1)) )
                return "001";
            else if ((getBodyUp(id) <= getBodyUp(id - 1)) &&
(getBodyDown(id) >= getBodyDown(id - 1)))
                return "010";
            else if ((getBodyUp(id) > getBodyUp(id - 1)) &&
(getBodyDown(id) > getBodyDown(id - 1)))
                return "011";
            else if ((getBodyUp(id) < getBodyUp(id - 1)) &&
(getBodyDown(id) < getBodyDown(id - 1)))
                return "100";
            else if (getBodyUp(id) <= getBodyDown(id - 1))
                return "101";
            else
                return "111";
        }
```

The next two bits represent the gaps.

"A price *gap* is formed when any traded item has a low that is above the prior day's high—an *upside gap*—or when the item's high is below the previous day's low creating a *downside gap*. Gaps are price areas where no trading took place from one session to the next.", as defined in book *Essential Technical Analysis – Tools and Techniques to Spot Market Trends, Leigh Steven.*

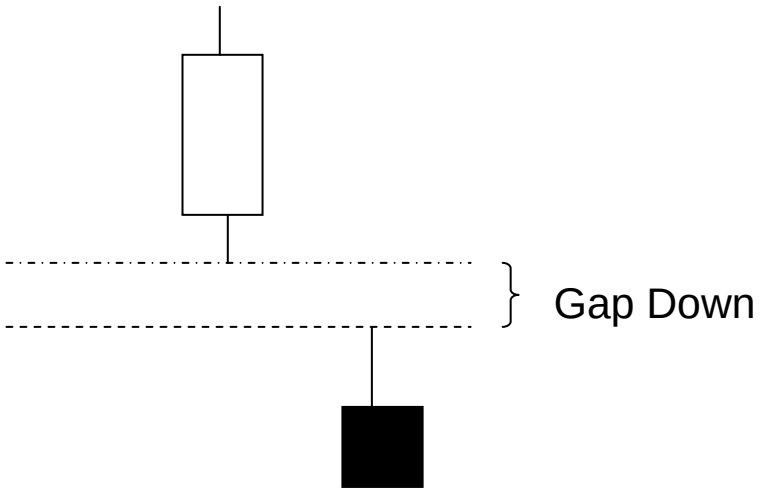


Bit position 7: Represents a Gap up as depicted in the above diagram. If gap up occurs, then, it returns “1”, otherwise returns “0”.

Here is the code in C#:

```
private string CalculateBit17(int id)
{
            if ( getLow(id) > getHigh(id - 1) )
                return "1";
            else
                return "0";
}
```

Bit Position 8: Represents a Gap down as depicted in the following diagram. If gap down occurs, then, it returns “1”, otherwise returns “0”.



Here is the code in C#:

```
private string CalculateBit18(int id)
{
            if (getHigh(id) < getLow(id - 1))
                return "1";
            else
                return "0";
}
```

Bit positions 9 and 10: It compares the current volume with the previous volume. If the percentage of change in volume is less -15% then it returns “10”. If the volume changes is in the range between -15% and 15, then it returns “00”. Finally, if the percentage change of volume is greater than 15%, it returns “11”.

Here is the code in C#:

```
private string CalculateBit19_20(int id)
{
   if (getVolume(id - 1) == 0)   // neutral result
            return "10";

   float val = 100 * (getVolume(id) - getVolume(id - 1))/getVolume(id - 1);

   if (val < -15)
               return "00";
       else if (val >= -15 && val <= 15)
               return "10";
       else
               return "11";
}
```

Knowledge Representation: Part C-> The relationships between three candlesticks (DAY 1 – DAY 2 – DAY 3)

Bit Position 1: It compares the current close price (DAY 3) with the day DAY 1 (two days ago). If current close price is greater than the second day (DAY 3) ago then return “1” otherwise “0”.

Here is the code in C#:

```
private string CalculateBit21(int id)
{
           if (getClose(id) >= getClose(id - 2))
               return "1";
           else
               return "0";
}
```

Bit Position 2: It compares the current close price (DAY 3) with the highest close price of the last 4 days including the current day.

Here is the code in C#:

```
private string CalculateBit22(int id)
{
            if ((getClose(id) == getHighestClose(id,4)) ||
            (getClose(id) == getLowestClose(id,4)))
                return "1";
            else
                return "0";
}
```

3.1.2 Developing Patterns based on the Codification System

First, the constructed codification components are three types:

- Component A: It contains all bits from “part A” of the previous paragraph – in other words, it includes all information corresponding to gvol-candlestick itself. So, the bit representation consists of 10 bits:
    - Bit 1 + Bit 2 + Bit 3 + Bit 4 + Bit 5_6 + Bit 7 + Bit 8 +Bit 9 + Bit 10
    - For example: For the value “0010110010” the bit in the third position (“1”) represents that the “BodyUp is lower than the middle of the candlestick.”

  Here is the code in C# corresponding to codification component A:

```
private string CalculatePatternA(int id)
{
     return CalculateBit1(id) + CalculateBit2(id) +
      CalculateBit3(id) + CalculateBit4(id) + CalculateBit5_6(id) +
      CalculateBit7(id) + CalculateBit8(id) + CalculateBit9(id) +
      CalculateBit10(id);
}
```

- Component B: It contains all bits from “part B” of the previous paragraph – in other words, it includes all information corresponding to the relationship between two gvol-candlesticks. So, the bit representation consists of 10 bits:
    - Bit 1 + Bit 2_3 + Bit 4_ 5_6 + Bit 7 + Bit 8 +Bit 9 + Bit 9_10
    - For example “0010110010” The bit in the second and third position (“01”) represents the that the “Percentage change of close price is between -3% and 0%”.

Here is the code in C#:

```
private string CalculatePatternB(int id)
{
    return CalculateBit11(id) + CalculateBit12_13(id) +
      CalculateBit14_15_16(id) + CalculateBit17(id) +
      CalculateBit18(id) + CalculateBit19_20(id);
}
```

- Component C: It contains only the bits of “part C” of the representation – in other words all information corresponding to the relation between three gvol-candlesticks. So, the bit representation consists of 2 bits:
    - Bit 1 + Bit 2
    - For example “01” The first bit (“0”) represents that the “current close price (DAY 1) is less than the DAY 3”.
    - Here is the code in C#:

```
private string CalculatePatternC(int id)
{
     return CalculateBit21(id) + CalculateBit22(id);
}
```

Finally, from the codification components we will construct the “pattern frames”. Pattern frames are the final destination as the result for the Knowledge Representation process. Based on them will be constructed millions of pattern frames corresponding to the collected time series of data. In other words, the “pattern frames” can be considered as a generator for translated Time Series Data into a proper component which contains specific information.

From the components A, B and C were constructed two types of pattern frames generators: Pattern Frame of Two and Pattern Frame of Three gvol-candlesticks

Pattern Frame of Two gvol-candlesticks

The pattern frame is formed as following:

(DAY 2: is the current day, DAY 1: is the previous day)

Calculation:

(Component A of DAY 1) + (Component A of DAY 2) + (Component B of DAY 2)

Here is the code in C#:

```
// It creates the pattern frame of 2 gvol-candles...10 bits
public string CreatePatternOfTwo(int id)
{
       return CalculatePatternA(id - 1) + CalculatePatternA(id) +
        CalculatePatternB(id);
}
```

Pattern Frame of Three gvol-candlesticks

(DAY 3: is the current day, DAY 2: is the previous of DAY 3, DAY 1: is the previous of DAY 2)

Calculation:

(Component A of DAY 1) + (Component A of DAY 2) + (Component A of DAY 3) + (Component B of DAY 2) + (Component B of DAY 3) + (Component C of DAY 3)

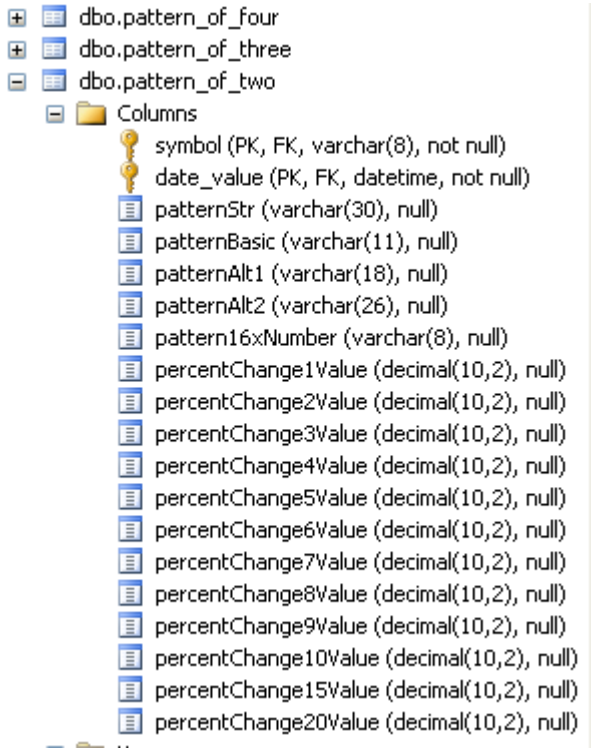
Here is the code in C#:

```
// It creates the pattern of 3 gvol-candles...length = 52 bits
public string CreatePatternOfThree(int id)
{
      return CalculatePatternA(id - 2) + CalculatePatternA(id - 1) +
      CalculatePatternA(id) +  CalculatePatternB(id - 1) +
      CalculatePatternB(id) + CalculatePatternC(id);
}
```

3.2 Details of Algorithm for Hidden Patterns

The algorithm is implementing as following:

STEP 1: From the program we create the data for all types of patterns: pattern of two gvol-candlesticks, pattern of three gvol-candlesticks and pattern of four gvol-candlesticks. These data are saved to the corresponding above tables (pattern_of_two, patterns_of_three, pattern_of_four). Therefore for each stock and each date we have one specific patterns of two, one specific pattern of three and one specific pattern of four.



Each table has the following data:

- The full pattern (patternStr: it has all possible information)
- Three alternative patterns which are simpler and hold less information (patternBasic, patternAlt1 and patternAlt2).
- It holds information about the changes of the close value of the price for the next 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15 and 20.

STEP 2: The next step is to group all type of patterns (patternStr ,patternBasic, patternAlt1 and patternAlt2) and referred to one of the three type of formations (pattern_of_two,

patterns_of_three, pattern_of_four). For example, let deal with the “patternStr” of pattern_of_two:

- First we group (categorize) the same patterns and count the number of their appearance according to their change (greater that zero, equals zero, less than zero). Actually, the implementation is in SQL Server. Here is the sql code for this specific pattern:
-

```
CREATE VIEW showTrendPtn2_all
AS

select count(a.patternStr) as number, a.patternStr,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange1Value>0 ) as upNext1,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange1Value=0 ) as equalNext1,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange1Value<0 ) as downNext1,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange3Value>0 ) as upNext3,

(select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange3Value=0 ) as equalNext3,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange3Value<0 ) as downNext3,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange5Value>0 ) as upNext5,

(select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange5Value=0 ) as equalNext5,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange5Value<0 ) as downNext5,
```

```
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange7Value>0 ) as upNext7,

(select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange7Value=0 ) as equalNext7,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange7Value<0 ) as downNext7,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange10Value>0 ) as upNext10,

(select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange10Value<0 ) as downNext10,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange15Value>0 ) as upNext15,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange15Value=0 ) as equalNext15,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange15Value<0 ) as downNext15


from pattern_of_two a  GROUP BY a.patternStr
```

After running the above query we can see the following display:

	number	patternStr	upNext1	equalNext1	downNext1	upNext3	equalNext3	downNext3
1	1	11011110110000000001000011001 0	0	1	0	0	1	
2	2	00000110000010101100110011000 0	1	0	1	1	0	1
3	199	00000000010000011001001100001 0	103	6	90	107	5	87
4	14	10101011011001101110110011001 0	11	1	2	9	0	5
5	7	10011010100101011010001001001 1	4	0	3	2	0	5
6	1	00100111110000001111000011100 00	0	0	1	1	0	0
7	2	00000000000101011110111010001 0	1	0	1	1	0	1
8	1	01011010100000100000000100011 0	1	0	0	0	1	0
9	2	00000110110010011100001001000 00	0	0	2	1	0	1
10	1	10001011101000101100110011000 0	0	0	1	0	0	1

Interpreting the above result , the third line corresponds to pattern that appeared 199 times and the next day (day 1) 103 times the % change close value was greater than zero, 6 times was no change and 90 times the % change in close value was negative.The next 3 columns represent what happened after 3 days, the next …

STEP 3: The next step is the above results to be presented as percentange. The sql code that implements this and for the specific above example is the following:

```
CREATE VIEW showTrendPtn2_Pall
AS
select      number, patternStr,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as
equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as
equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P


from showTrendPtn2_all
```

After running the above query we can see the following display:

	number	patternStr	upNext1P	equalNext1P	downNext1P	upNext3P	equalNext3P	downNext3P
1	70	01010110101001101110001100010	50.00	2.86	47.14	52.86	4.29	42.86
2	8	10101011010000011011001010010	25.00	0.00	75.00	50.00	12.50	37.50
3	4	00101011101000100001110011001	25.00	25.00	50.00	25.00	0.00	75.00
4	1	00000110100000011001000100010	0.00	0.00	100.00	0.00	0.00	100.00
5	3	11011011111001101001111011010	66.67	0.00	33.33	66.67	0.00	33.33
6	2	00100111011001101111111011001	50.00	0.00	50.00	50.00	50.00	0.00
7	1	10001011101001110000110011001	0.00	100.00	0.00	100.00	0.00	0.00
8	1	00000110101000110001110100011	0.00	0.00	100.00	0.00	0.00	100.00
9	1	00101011111001100000000100010	100.00	0.00	0.00	0.00	100.00	0.00
10	1	00000111010010011011100010010	0.00	100.00	0.00	0.00	100.00	0.00
11	22	11011011101010110100011000011	50.00	4.55	45.45	50.00	0.00	50.00
12	201	00000110100000000010001000011	48.26	3.98	47.76	53.23	1.99	44.78
13	5	00101011000010011011100111011	20.00	40.00	40.00	40.00	20.00	40.00
14	16	10101011000010011110001011001 1	75.00	12.50	12.50	56.25	0.00	43.75

STEP 4: The steps 2 and 3 are repeated for all type of patterns and formations.

STEP 5: The final step is to apply sorting and get the patterns that have the following:

- The number of repeats of each pattern is enough to create a rule.
- We apply various sorting in order to get these patterns that have large upNext value (and small downNext) or have large downNext value (and small upNext).

3.3 Results

The results seem to be important and need more paying attention for further improvement and then to be applied. All detail results are displayed at the appendix “C”. The appendix “C” contains four parts – one for buying and one for selling signals for each type of pattern frame.

Explanation of the columns:

Column “Frequency”: Depicts the occurrence of the specific pattern.

Column “Pattern of Two or Three”: It is the binary identification for pattern frame according to the codification rules have been explained.

Column “UP% NEXT 1”: It tells us what happens after one day (next day) the specific pattern occurred. The number depicts how many percent from these patterns drove the stock up (to increase). For example if “UP% NEXT 1” = 75 it means that the 75 % of these patterns had next day (close price) greater or equal to the previous day close price of the stock. In other words, there is a possibility of 0.75 or 75% to be the next day close price greater or equal to zero.

Column “UP% NEXT 10”: It tell us what happens after 10 days of the specific pattern occurred. The number depicts how many percent from these patterns drove the stock up (to increase) at the day +10. For example if “UP% NEXT 10” = 75 it means that the 75 % of these patterns had the close price at the 10th day (close price) greater or equal to the day of the pattern appeared. In other words, there is a possibility of 0.75 or 75% to be the day+10 close price greater or equal to zero.

Column “DOWN% NEXT 1”: It tell us what happens after one day (next day) the specific pattern occurred. The number depicts how many percent from these patterns drove the stock down (to decrease). For example if “DOWN % NEXT 1” = 75 it means that the 75 % of these patterns had next day (close price) less or equal to the previous day close price of the stock. In other words, there is a possibility of 0.75 or 75% to be the next day close price less or equal to zero.

Column “DOWN % NEXT 10”: It tell us what happens after 10 days of the specific pattern occurred. The number depicts how many percent from these patterns drove the stock down (to decrease) at the day +10. For example if “UP% NEXT 10” = 75 it means that the 75 % of these patterns had the close price at the 10th day (close price) less or equal to the day of the pattern appeared. In other words, there is a possibility of 0.75 or 75% to be the day+10 close price less or equal to zero.

NOTE: In this coursework we count only the event what happens after 1, 3, 5, 7, 10 and 15 days about the increasing and decreasing close price of stocks. In other words, there take place calculations about comparing the close prices:

NEXT 1 = (Close price of day of occurrence) – (Close price of day + 1)

NEXT 3 = (Close price of day of occurrence) – (Close price of day + 3)

…….

NEXT 15 = (Close price of day of occurrence) – (Close price of day + 15)

Therefore for trend up calculations if the result of (Close price of day of occurrence) – (Close price of day + 1) >= 0 it counts for DOWN, otherwise it counts for UP….

NOTE: The equal (=) plays some role in NEXT 1 day and very little for days next 3, 5, 7..

It was discovered a lot of complicated pattern frames and all results are in appendix “C”.

There were collecting more than 300,000 patterns frames and most of them were rejected due to the not significant results. The rule for rejection was “keep results only those have at least one column % of NEXT 1 or 3, 5, 7, 10 or..NEXT 15 greater than 65%. This key-point

was selected because the well known candlestick patterns very rarely can bring positive results more than 65%. These tests were examined by using Metastock Program.

3.4 Some Excellent Results:

RESULTS CATEGORY A: The following samples which are corresponding to up trend (buying signals) depict that after the occurrence of specific pattern for all next coming 10 dates (day+1, day+3, day+10) the possibility of positive change in close values is more than 70.59. In other words, these patterns trigger for strong positive (trend up) movements while keeping them at constant level for all 10 days.

Pattern Frame of two gvol-candlesticks:

PATTERN OF TWO	UP% NEXT 1 DAY	UP% NEXT 3 DAYS	UP% NEXT 10 DAYS
10011100000000100000000100000	94.12	88.24	70.59
01011010111101101010110101000 0	81.4	86.05	76.75
00101011000010101101111011101 1	81.08	72.97	75.68

Pattern Frame of three gvol-candlesticks:

PATTERN OF THREE	UP% NEXT 1 DAY	UP% NEXT 3 DAYS	UP% NEXT 15 DAYS
00101011010010101100001010110011001000001100100000 11	96.67	86.66	93.33
00101011000010101100001010110011001000001100100010 1111	92.59	83.33	85.19
00101011000010101100001010110011001000111100100000 11	83.33	83.33	76.19

RESULTS CATEGORY B: The following samples which are corresponding to up trend (buying signals) depict that after the occurrence of specific pattern the positive (uptrend) is keeping only for the first 3 days. At the 10th day there is a significant reverse in the market (downtrend). In other words, these patterns trigger for short period of positive (trend up).

Pattern Frame of two gvol-candlesticks:

PATTERN OF TWO	UP% NEXT 1	UP% NEXT 3	UP% NEXT 10
10101011100000011100001100001 0	80	62.86	40
11011010101101101110001010000 0	77.27	45.45	36.36
00100111000101011110110011001 1	70.59	58.82	47.06

Pattern Frame of two gvol-candlesticks:

PATTERN OF THREE	UP% NEXT 1	UP% NEXT 3	UP% NEXT 10
000001000100000100001001100000001100000011001000100 1	71.88	43.75	40.63
000001000010011000001001110001110001001111101100111 1	76.67	56.67	46.66
000001000001010110101001100000001100000011001100111 1	71.88	62.51	50.01

RESULTS CATEGORY C: The following samples which are corresponding to up trend (buying signals) depict that after the occurrence of specific pattern for all up coming dates (day+1, day+5, day+10) the possibility of positive change in close values is increasing gradually as the days pass. In other words, as the days passes the possibility of up trend always increases. Notice that the next day (day+1) is not good for up trend.

Pattern Frame of two gvol-candlesticks:

PATTERN OF TWO	UP% NEXT 1	UP% NEXT 5	UP% NEXT 10
000000000100000100000001000100	65.91	77.28	84.09
000001000001010111111100100010	54.84	70.97	83.87
010110111101011011111100100000	51.61	70.97	83.87

Especially in the pattern of three (following table), the possibility of next day (day+1) to be positive is less than 50% (36.67%, 38.09%, 47.5%) – very low. It alert us in advance that after 5 days there will be occur a reverse in the market.

Pattern Frame of three gvol-candlesticks:

PATTERN OF THREE	UP% NEXT 1	UP% NEXT 5	UP% NEXT 10
000001000001010110110000011000001100001100110000000 1	36.67	66.67	70
000001000000000011000000001000100110000110011000011 01	38.09	66.67	76.19
000001000000000010000001010111000110000001100100011 01	47.5	52.5	72.5

RESULTS CATEGORY D: The following samples which are corresponding to up trend (buying signals) depict that after the occurrence of specific pattern for all up coming dates (day+1, day+5, day+15) the possibility of positive change in close values is increasing gradually as the days pass. The importance in this case is the long term prediction. It tell us that the reverse will significantly start after 5 days and it will continue at list for 10 days more (until 15th day).

Pattern Frame of two gvol-candlesticks:

PATTERN OF TWO	UP% NEXT 1	UP% NEXT 5	UP% NEXT 15
000001000001010111111100100010	54.84	70.97	90.32
001001111001010110110011000111	59.38	71.88	84.38
010110111101011011111100100000	51.61	70.97	83.87

Pattern Frame of three gvol-candlesticks:

PATTERN OF THREE	UP% NEXT 1	UP% NEXT 5	UP% NEXT 15
000001000001010110110000011000001100001100110000000001	36.67	66.67	70
000000000110011000001001100000110100000011001100000011	45.94	54.06	78.38
000001100000000010001000001000000110000110011000000001	50.98	56.86	74.51

RESULTS CATEGORY E: The following samples which are corresponding to very strong down trend (selling signals) depict that after the occurrence of specific pattern for all up coming dates (day+1, day+5, day+15) the possibility of negative change in close values is very strong. The significance of this is the warning of bad market for a long period of days (15 days).

Pattern Frame of two gvol-candlesticks:

PATTERN OF TWO	DOWN% NEXT 1	DOWN% NEXT 5	DOWN% NEXT 15
000000000000001000001110100000	89.65	86.21	93.1
001001111000000100001100010010	80.77	88.46	69.23
000001000100001011100011000000	66.67	70.37	70.37

RESULTS CATEGORY F: The following samples which corresponding to very strong down trend (selling signals) depict that after the occurrence of specific pattern for the downtrend continues for about more 5 dates (until day+5). The significance of this is the warning for bad market for a long period of days is about to terminate and after 15 days will reverse for uptrend market.

Pattern Frame of three gvol-candlesticks:

PATTERN OF THREE	DOWN% NEXT 1	DOWN% NEXT 5	DOWN% NEXT 15
1001110001100111000100000000001110110010000011000011	80	80	45
0000010000110110101010011000001101000011110011000011	72	80	32
1001100000000001000011011010100011000010110010001001	73.92	78.26	39.13

RESULTS CATEGORY G: The following samples depict that a strong downtrend will occur after about 10 days.

Pattern Frame of three gvol-candlesticks:

PATTERN OF THREE	DOWN% NEXT 1	DOWN% NEXT 3	DOWN% NEXT 10
0000010001001001110000000100000011000000001100001101	39.13	30.44	73.91
1001100000101010110010011100011100110011111011001111	57.14	52.38	71.43
1001100001000001000100000100010010100011001100000001	58.34	58.33	70.83

Note: All details of results are collected in the appendix.

Staffordshire UNIVERSITY

4. Using Neural Nets in Stock Market

4.1 Constructing the Input System for the Neural Net

The most effort was on constructing a smart input system based on technical indicators and trading systems. The key words for this input system are “adaptation” and “optimization” . The target is not to construct a neural net that works with all stocks but each stock should have at least one proper neural net. Adaptation does not only applies to pair “stock-neural” but also to parameters of the trading system – stcock. It means that each trading system is applied in best efficient way. Parameters are optimized to produce the best profitable results.

Additionally, the input system consists of three data categories:

- Trading systems that produce signals (0 = sell, 1 = buy)
- Indicators that don’t produce signals but depict trend, volatility or overbought / oversold conditions.
- Stock values (open, close, high, low, volume)

Although, the network has the ability itself to normalize the input data (0,1), the normalization is done externally before feeding the neural. The trading systems produce signals with values 0.999 or 0.001. For the list of indicators which do not produce signals is applied proper algorithm to translate the value in a float number between 0 and 1. Stock values are not normalized externally.

What is required:

The first input part, which referred to the trading systems, needs to have the optimized parameters of the technical indicators. The values of these parameters should be stored in the database. For this, there is a special process used to optimize the input parameters for each indicator per stock. The other part (group) of technical indicators take default values in there input parameters.

The “Smart Input System” for the neural net is displayed in the next picture. There have been developed special algorithms that prepare, translate and normalize the data for input. The concept of optimized trading systems for each stock is to produce the most realible signals minimizing the input errors. Using large number of neurons in the input level does not mean that the results should be better.

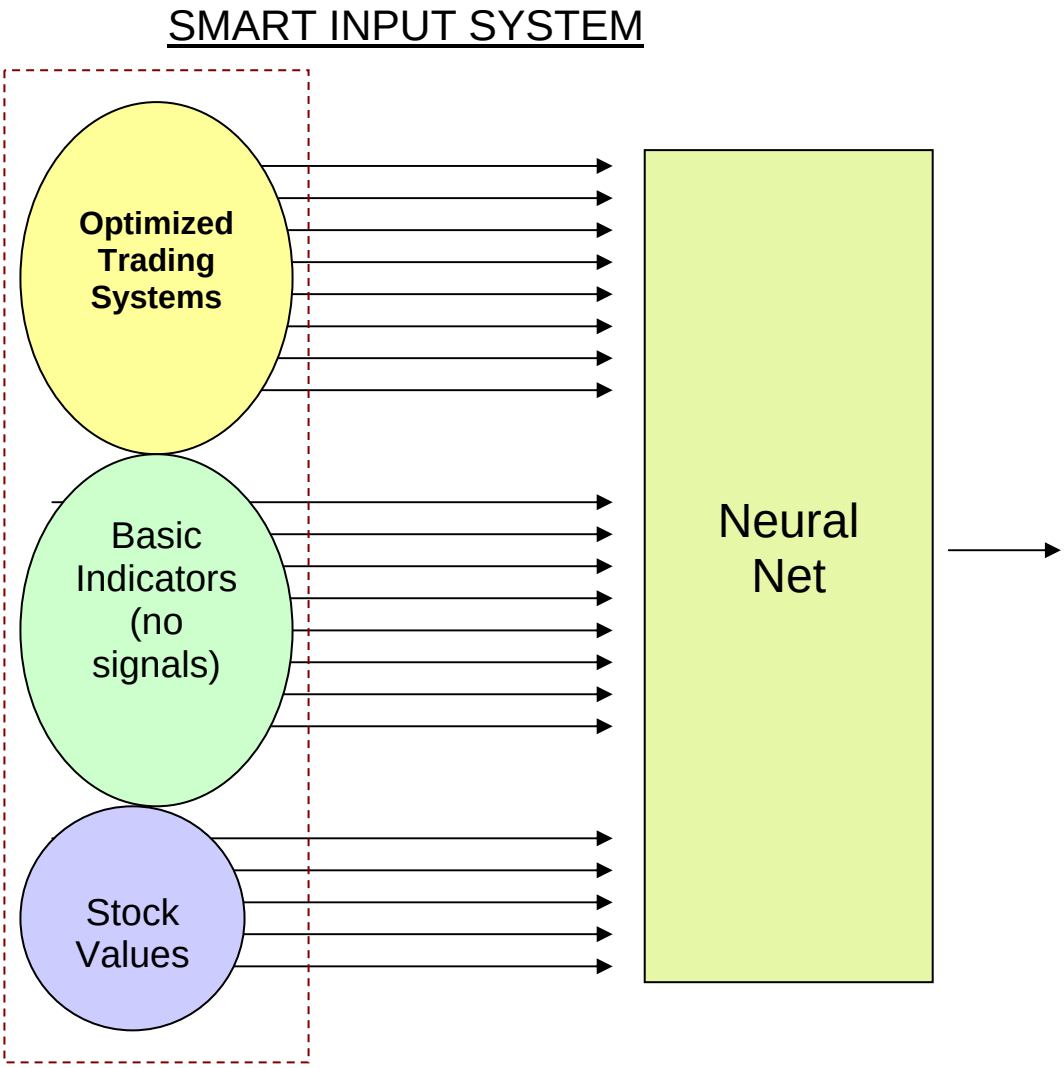


Figure 4.a: *The smart input system*

4.1.1 The optimized trading input system

It were developed 12 trading systems which their parameters have been optimized for each stock and saved to the database (table name: AITrade). The trading systems are the following:

i) Simple Moving Average (SMA) trading system. The input parameter is the the number of days used for calculation. The system examines the parameter in a range from 6 to 50 and keeps the value which brings the best profitable results. Buy (long position) when stock price (close value) is greater than its SMA.

ii) Expotential Moving Average (EMA) trading system which is a weighted moving average (most recent values are weight higher than the previous values). The input parameter is the the number of days used for calculation. The system examines the parameter in a range from 6 to 50 and keeps the value which brings the best profitable results. Buy (long position) when stock price (close value) is greater than its EMA.

iii) Commodity Channel Index (CCI) trading system. It takes three parameters which are needed to be optimized:

Parameters: number of days ago -> parameter range from 8 to 25
Buy x (when CCI > x)-> range of x: -150 to -50
Sell y (when CCI <y)->range of y: 50 to 150

iv) Chaikin A/D trading system. It takes three parameters which are needed to be optimized:

Parameters: number of short days ago -> parameter range from 2 to 5
number of long days ago -> parameter range from 8 to 14
Moving Average of Chaikin ->Days ago from 4 to 10
(crossover signal between Chaikin and its MA)

v) Chandle Momentum Oscilator (CMO) trading system. It takes three parameters which are needed to be optimized:

Parameters: number of days ago -> parameter range from 6 to 20
Buy x (when CMO > x)-> range of x: -60 to -40
Sell y (when CMO <y)->range of y: 40 to 60

vi) Directional Movement trading system. It takes one parameter: the number of days ago (optimization from 10 to 20 days ago). Positions should be taken by buying when the +DI rises above the –DI and selling when the +DI falls below the –DI..

vii) Momentum trading system. It takes three parameters which are needed to be optimized:

Parameters: number of days ago -> parameter range from 6 to 20
Buy x (when Momentum > x)-> range of x: 70 to 130
Sell y (when Momentum <y)-> range of y: 70 to 130

viii) Price Oscilator (PriceOsc) trading system. It takes three parameters which are needed to be optimized:

Parameters: number of short days ago -> parameter range from 5 to 15
number of long days ago -> parameter range from 16 to 40
Moving Average of PriceOsc ->Days ago from 4 to 15
(crossover signal between PriceOsc and its MA)

ix) Rate of Change (ROC) trading system. It takes three parameters which are needed to be optimized:

Parameters: number of days ago -> parameter range from 6 to 20
Buy x (when ROC > x)-> range of x: -20 to 20
Sell y (when ROC <y)-> range of y: -20 to 20

x) Relative Strenght Index (RSI) trading system. It takes three parameters which are needed to be optimized:

Parameters: number of days ago -> parameter range from 10 to 20
Buy x (when RSI > x)-> range of x: 20 to 40
Sell y (when RSI <y)-> range of y: 60 to 80

xi) Relative Volatility Index (RVI) trading system. It takes three parameters which are needed to be optimized:

Parameters: number of days ago -> parameter range from 8 to 20
Buy x (when RVI > x)-> range of x: 50 to 70
Sell y (when RVI <y)-> range of y: 30 to 50

xii) Stochastic Oscilator (StochOsc) trading system. It takes two parameters which are needed to be optimized:

Parameters: number of days ago -> parameter range from 10 to 20
Moving Average of StochOsc ->Days ago from 2 to 6
(crossover signal between StochOsc and its MA)

The formulas of the above trading systems are in the file: Indicator.cs (BasicOper namespace) which can be found in the appendixes. The table "AITrade" of the StockMarket database contains all information optimized results. Additionally this table holds and other information such as total trades, number of success trades, number of unsuccess trades, total Earning, performance %, annual performance % etc. For example if we run a query for a stock "A",

```
select * from AITrade where symbol = 'A'
```

we receive the results of all trading systems for the specific stock:

	symbol	codeInd	typeoftrade	fromDateAp...	toDateApplied	numberOfStocks	feePerTrade	totalEarning	totalTrades
1	A	CCI	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	30510.00	34
2	A	Chaikin	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	20370.01	205
3	A	CMO	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	28040.00	6
4	A	DMI	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	26080.00	52
5	A	EMA	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	34369.98	74
6	A	Momentum	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	34680.00	50
7	A	PriceOsc	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	23390.02	166
8	A	ROC	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	34680.00	50
9	A	RSI	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	21720.00	58
10	A	RVI	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	36590.00	21
11	A	SMA	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	34909.99	60
12	A	StochOsc	long	2003-01-02 ...	2007-05-04 ...	1000	10.00	9350.01	108

Also, there are additional fields (columns):

	symbol	codeInd	successTrades	unSuccessTrades	performance	annualPerformance
1	A	CCI	12	22	242.14	58.49
2	A	Chaikin	127	78	125.51	30.31
3	A	CMO	4	2	225.22	54.40
4	A	DMI	27	25	200.77	48.49
5	A	EMA	29	45	255.54	61.72
6	A	Momentum	26	24	270.94	65.44
7	A	PriceOsc	108	58	194.11	46.88
8	A	ROC	26	24	270.94	65.44
9	A	RSI	35	23	130.06	31.41
10	A	RVI	14	7	293.90	70.99
11	A	SMA	24	36	266.49	64.37
12	A	StochOsc	60	48	56.74	13.70

And the last fields included in AITrade table:

	symbol	codeInd	parameterDesc	parameter1	parameter2	parameter3
1	A	CCI	param 1: days of RVIAdvanced, param 2: Buy, param 3:Sell	25.00	-75.00	70.00
2	A	Chaikin	param 1: days of short MA, param 2: days of long MA, param 3:days for signal MA	3.00	13.00	7.00
3	A	CMO	param 1: days of CMO, param 2: Buy, param 3:Sell	28.00	-30.00	0.00
4	A	DMI	param 1: days back, param 2: Value for DX	11.00	18.00	0.00
5	A	EMA	parameter 1: days of MA	16.00	0.00	0.00
6	A	Momentum	param 1: days of Momentum, param 2: Buy, param 3:Sell	13.00	100.00	100.00
7	A	PriceOsc	param 1: days of short MA, param 2: days of long MA, param 3:days for signal MA	5.00	22.00	8.00
8	A	ROC	param 1: days of ROC, param 2: Buy, param 3:Sell	13.00	0.00	0.00
9	A	RSI	param 1: days of RSI, param 2: Buy, param 3:Sell	14.00	50.00	60.00
10	A	RVI	param 1: days of RVI, param 2: Buy, param 3:Sell	17.00	58.00	44.00
11	A	SMA	parameter 1: days of MA	16.00	0.00	0.00
12	A	StochOsc	param 1: days back, param 2:days for signal MA	14.00	6.00	0.00

4.1.2 The Input System for the Basic Indicators

This group includes indicators that do not produce signals for buy or sell. These indicators are the following:

i) Indicator about the stock close values of previous days. Generally, it compares the close values of the last 3 days and produces value between -0.37 and 0.37.

ii) Indicator about the candlestick types (formations).

iii) Williams Indicator. The return value of the indicator is translated to proper value between -0.49 to 0.49

iv) Volume evaluation. Includes comparisions between current and previous volume and comparisions between volume and moving average volume. The return value is between -0.44 to 0.44

v) Volume Oscilator. The evaluation depends on the value of the indicator. The return value is +0.45 when indicator value is greater than zero and -0.45 when indicator value is less than zero.

vi) Project Oscilator. There is evaluation process based on previous values of the indicator and current overbought / oversold status. The return value whould be +0.45 or -0.45.

vii) R-Squared. It does not take any place for further evaluation since the indicator counts the trend and returns values between 0 and 1.

viii) Vertical Horiozontal Filter (VHF). There is no further evaluation. The VHF determines whether prices are in a trending phase or a congestion phase.

4. 2 The Neural Net

It was tried various open source neurals. Finally, after a lot of consideration it was decided to use the "NeuronDotNet[1]" which is an open source platform proper to run AI applications on 'back propagation artificial neural networks'. This neural supports acyclic structure of layers, different types of activation functions, backpropagation algoritm using momentum term, weight decay and jitter, layers based on one-one connections between them, etc.

The training of the neural was repeated a lot of times (between 50 to 4000) per stock using different input neurons and different parameters of the neural. Actually, during running (training) it was used different values referred to the following:

i) *Input System*:

- It can be used the trading systems correspond to specific stock and produced annual performance (percentage) greater than a specific value (e.g. >= 10%). In other words we can automatiocally filter the trading systems and use only those that produce satisfied results.
- The basic indicators that do not produce signals can be selected according to user preferences.
- It can be also used as input to the the neural all various stock values (close, open, high, low, volume). The testing showed that using these

[1] Internet Source of Neural Network (NeuronDotNet) : http://neurondotnet.freehostia.com/index.html

values as additional input to the neural do not bring better results (more profitable).

ii) *Neural parameters.*

For this research it were used various parameters filling with different values. The customization of the neural includes the following:

- Capability to select one or three hidden layes.
- Possibility to select one of the four different type of activation functions correspond to each layer.
- Change the neural parameters referred to the following:
    - Jitter Epoch
    - Annealing Epoch
    - Learning Rate Increase Factor
    - Learning Rate Decrease Factor
    - Number of training cycles
    - Input layer learning rate
    - Input layer momentum
    - Hidden layer Momentum
    - Output layer Max Jitter

iii) *Neural Output system for training*.

What values should be used to the neural for the training is very important. For this purpose it were used four types of outputs which have been checked for producing profitable results. They are the following:

Output type 1: The signal depends on the next two days of close values. Enter into market only when the changes of the next two days are both positive (%change >0).

Output type 2: The same as previous but the dependency is only on the next change of close value. This output produce the best possible profitable results of all types of trading systems.

Output type 3: It is depend on the next two days of close values. Enter into the market only when the next day is positive (% close change value) or the next second close value greater than current close.

Output type 4: It depends on previous and next day close value. Enter into the market only when the current close value is greater than previous and the next day close value is greater that current close value.

The testing phase of the neural network produce results and they can be classified into two categories:

- Results referred to trading signals similar to the trading system of using indicators
- Results about the possibility of the direction of prices of the next day.

4. 3 The Results of Testing.

It were performed approximately 47,080 simulations (training and testing) of the neural net for 100 stocks (average 450 different neurals per stock). The results can be analyzed as following:

i) Generally, the annual performance of the neural testing for 100 stocks ranges from 9.5% to 131.21%. Running the following sql script:

```
select symbol,  count(*) as number_of_training, max(annualNetPerformance)
as annualNetPerformance, max(totalTradesPrediction) as
totalTradesPrediction from neural_results GROUP BY symbol ORDER BY
annualNetPerformance DESC
```

we receive the following result (this is sample – the first best 8):

SYMBOL	# NEURALS	Annual Neural % Profit	Prediction % of total trades
GS	502	131.21	57.86
CRS	502	85.28	55.85
DE	168	55.84	55.52
FLR	490	54.21	57.53
CMI	659	52.13	58.53
FFIV	87	48.57	55.18
ETR	167	45.77	57.53
DO	166	43.60	55.85

ii) Evaluation of the prediction of what will happen next day about stock value. The close value will increase or decrease? The results of the neural net testing range from 52.84% to 62.54%. Running the following sql script:

```
select symbol,  count(*) as number_of_training, max(annualNetPerformance)
as annualNetPerformance, max(totalTradesPrediction) as
totalTradesPrediction from neural_results GROUP BY symbol ORDER BY
totalTradesPrediction DESC
```

we receive the following result (this is sample – the first best 8):

SYMBOL	# NEURALS	Annual Neural % Profit	Prediction % of total trades
ABBI	787	11.45	62.54
BMS	121	9.64	61.54
ABK	2810	30.76	60.54
COST	305	21.23	60.20
EBAY	1306	15.60	60.20
FISV	3746	23.36	60.20
HLT	188	20.94	59.20
ABC	1879	20.72	59.20

iii) Concetration on neurals that produce better prediction signals either in buy or in sell. Running the following query which focues on those which have better results in buy signals:

```
select symbol, annualNetPerformance, totalTradesPrediction,
predictionOfBuy, predictionOfSell from neural_results where
predictionOfSell > 45 order by predictionOfBuy DESC
```

we receive the following result (this is sample – the first best 9):

SYMBOL	ANNUAL PERFORMANCE	Total % prediction	Prediction % on buy signals	Prediction % on sell signals
BMS	9.64	61.54	77.18	46.00
EBAY	7.08	59.20	73.19	47.20
FISV	16.68	58.86	71.90	45.21
FISV	17.73	59.53	71.90	46.58
FISV	16.25	58.53	71.24	45.21
FISV	18.04	60.20	71.24	48.63
ABC	17.48	59.20	70.44	46.43
EBAY	13.27	57.19	70.29	45.96
HLT	20.94	59.20	70.20	47.97

Additionally, running the following query which focuses on those which have better results in sell signals:

```
select symbol, annualNetPerformance, totalTradesPrediction,
predictionOfBuy, predictionOfSell from neural_results where
predictionOfBuy > 45 order by predictionOfSell DESC
```

we receive the following result (this is sample – all results are in the appendix):

SYMBOL	ANNUAL PERFORMANCE	Total % prediction	Prediction % on buy signals	Prediction % on sell signals
ABK	30.73	57.86	46.79	69.93
FISV	22.18	58.86	48.37	69.86
FISV	15.67	58.53	48.37	69.18
AIG	11.48	56.86	45.45	68.97
AIG	11.18	57.19	46.10	68.97
AIG	9.92	56.52	45.45	68.28
AIG	14.61	57.86	48.05	68.28
ABBI	9.27	58.53	45.16	68.00

Therefore, for each stock we can use different neural that produce better results either on buy or on sell directions.

iv) Comparison neural net performance with traditional technical techniques (trading systems which have been optimized to bring the maximum profitable results). The following query brings the stocks in which the neural net had better profitable performance comparing with the best profitable trading system (optimized).

```
select a.symbol, max(a.totalTradesPrediction) as totalTradesPredictionNet,
max(a.annualNetPerformance) as annualNetPerformance,
max(b.annualPerformance) as annualPerformanceFromTrading from
neural_results a, AITrade b where a.symbol = b.symbol  GROUP BY a.symbol
having  max(a.annualNetPerformance)> max(b.annualPerformance) order by
max(a.annualNetPerformance) DESC
```

we receive the following result (all results):

SYMBOL	Prediction % from Neural	Performance % from Neural	Best Perormance from Best Trading System	% Difference
GS	57.86	131.21	60.65	70.56
DE	55.52	55.84	46.56	9.28
ETR	57.53	45.77	45.47	0.30
GWW	58.53	32.67	32.40	0.27
ABK	60.54	30.76	25.05	5.71
FISV	60.20	23.36	22.27	1.09
FNM	55.18	23.18	16.85	6.33
AIV	53.85	22.98	21.28	1.70
DLX	54.85	21.16	15.38	5.78
AEE	58.86	16.25	13.45	2.80

BUD	56.19	13.96	8.68	5.28
LLY	58.19	13.64	12.08	1.56
IP	56.86	13.05	11.48	1.57
DWA	57.53	12.36	11.88	0.48
GCI	55.85	11.90	8.53	3.37
DTE	55.18	11.62	11.22	0.40
KFT	57.19	11.41	8.85	2.56
FITB	55.18	9.50	7.74	1.76

From the above results we can conclude that the neural produce better results for 18 stocks from total 100 (means 18%).

v) The next comparison is results between neural and the average of the performance of all trading systems (12 systems average performance) per stock. Running the following query:

```
select a.symbol, max(a.totalTradesPrediction) as totalTradesPredictionNet,
max(a.annualNetPerformance) as annualNetPerformance,
max(b.annualPerformance) as annualPerformanceFromTrading from
neural_results a, AITrade b
where a.symbol = b.symbol  GROUP BY a.symbol having
max(a.annualNetPerformance)> avg(b.annualPerformance)
order by max(a.annualNetPerformance) DESC
```

we receive the following result (all results):

SYMBOL	Prediction % from Neural	Performance % from Neural	Best Perormance % from Best Trading System	Average Perormance % from Best Trading System
GS	57.86	131.21	60.65	44.34
DE	55.52	55.84	46.56	37.07
FLR	57.53	54.21	67.67	47.25
ETR	57.53	45.77	45.47	32.43
AGN	56.52	39.21	39.34	25.40
BDK	56.19	35.43	51.32	34.54
GWW	58.53	32.67	32.40	18.65
ABK	60.54	30.76	25.05	20.02
DHR	56.86	26.45	38.70	25.55
FISV	60.20	23.36	22.27	10.65
DISH	57.53	23.28	33.28	21.53
FNM	55.18	23.18	16.85	5.99
AIV	53.85	22.98	21.28	16.62
DJ	57.86	22.91	22.94	18.49
COST	60.20	21.23	24.48	15.27
ABT	57.86	21.23	21.65	14.63

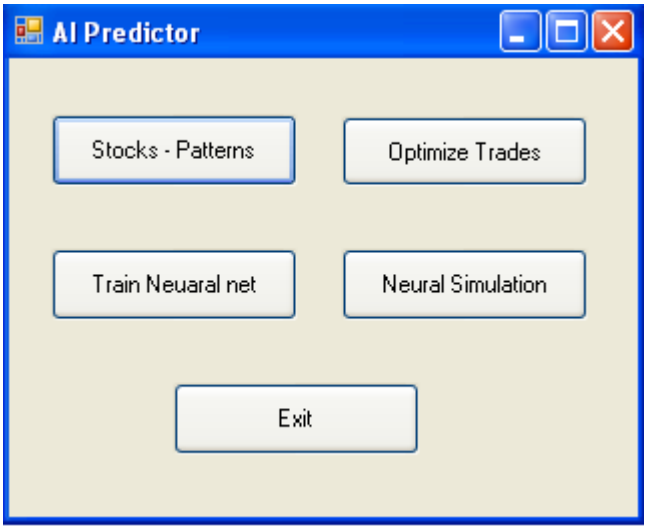
DLX	54.85	21.16	15.38	9.10
HOG	56.19	20.89	23.15	14.74
ABC	59.20	20.72	24.81	18.97
ADI	57.86	17.67	29.15	14.65
AEE	58.86	16.25	13.45	9.54
BMET	57.19	16.24	21.11	13.29
AIG	59.20	15.80	22.19	15.21
DD	54.85	15.28	16.40	12.32
EL	54.52	15.14	24.35	15.00
ACS	55.18	14.63	18.09	7.37
BUD	56.19	13.96	8.68	2.92
ADP	55.52	13.74	22.12	11.87
K	57.86	13.67	20.39	12.43
LLY	58.19	13.64	12.08	8.52
BBBY	57.19	13.46	17.59	8.32
IP	56.86	13.05	11.48	6.00
AFL	54.52	12.90	19.18	12.76
HNZ	57.19	12.59	15.77	11.01
DWA	57.53	12.36	11.88	5.17
BMY	55.52	12.11	16.45	10.24
GCI	55.85	11.90	8.53	-0.57
DTE	55.18	11.62	11.22	7.80
JNY	58.86	11.62	23.47	2.85
KFT	57.19	11.41	8.85	4.14
CTAS	55.85	11.19	12.77	7.09
BAC	56.19	10.80	17.19	10.40
FHN	54.85	9.53	13.62	5.31
FITB	55.18	9.50	7.74	0.34

The results are 44 stocks from total 100 stocks which means that these neural in 44% of stocks can produce better results comparing with the results of the average performance of the trading systems.

5. Running the application

5.1 Forms Navigation

On running the application it appears the main menu:



It includes four main categories:

(i) Stock – Patterns: It is used for inserting stock value data from a text file into database. And also it helps to create the gvol-candlestick patterns for each selected stock.

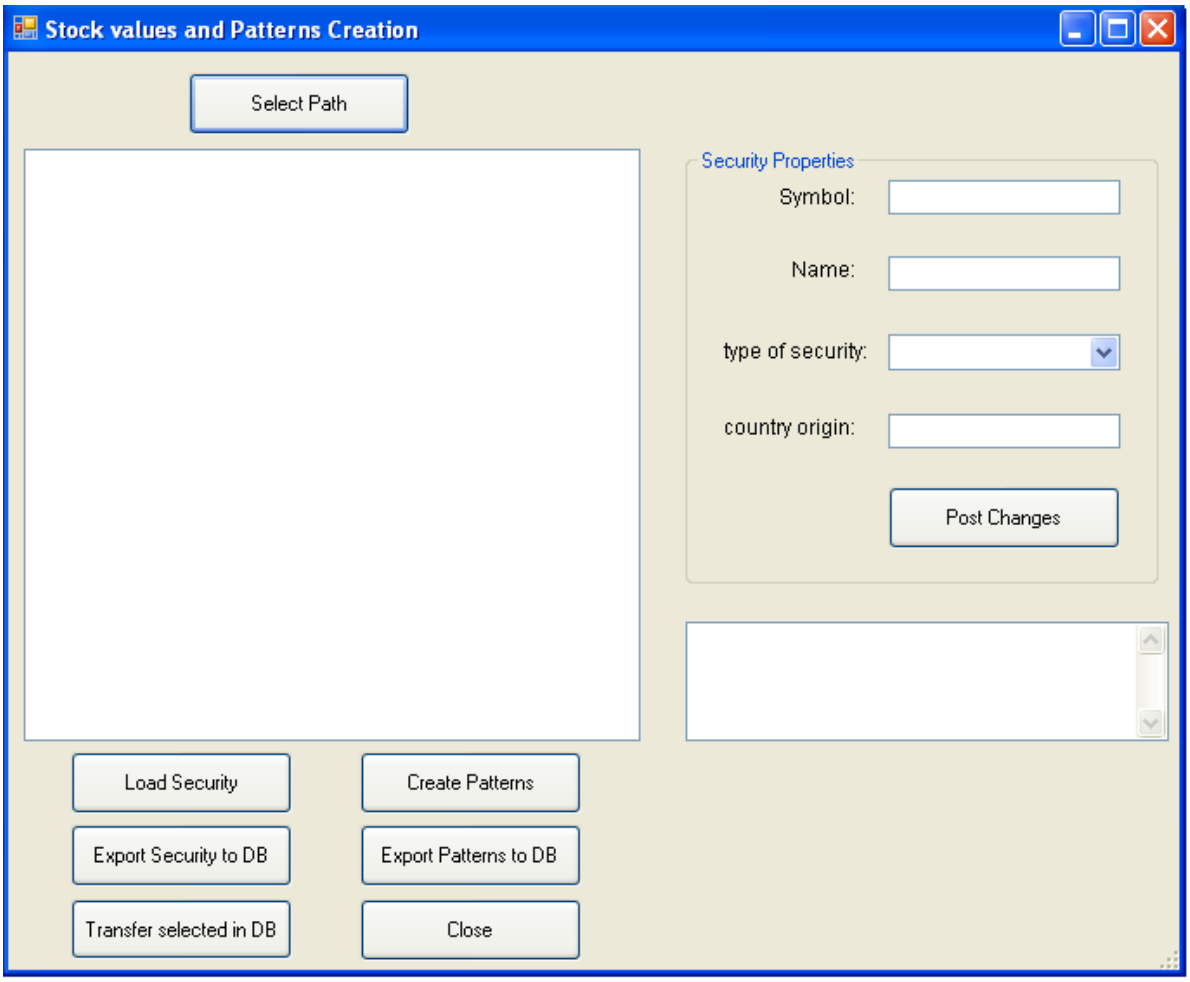
(ii) Optimize Trades: It is useful for optimize the input parameters of all indicators. Optimization means that this indicator brings the best results for a specific stock value.

(iii) Train Neural Net: It is used for training manually a neural net for a specific stock. Then, it can be tested and receive the performance. A lot of customization is available to the user.

(iv) Neural Simulation. It can be performed completely automatic training and testing of the neural nets. Customization is random and does not depend on the user.

5.2 Stocks – Patterns

When selecting this menu it is displayed the following form:



The first task is to create stocks if not have been created and then to load their price values. Stock values can be loaded automatically from files if they have specific format. The heading format is the following:

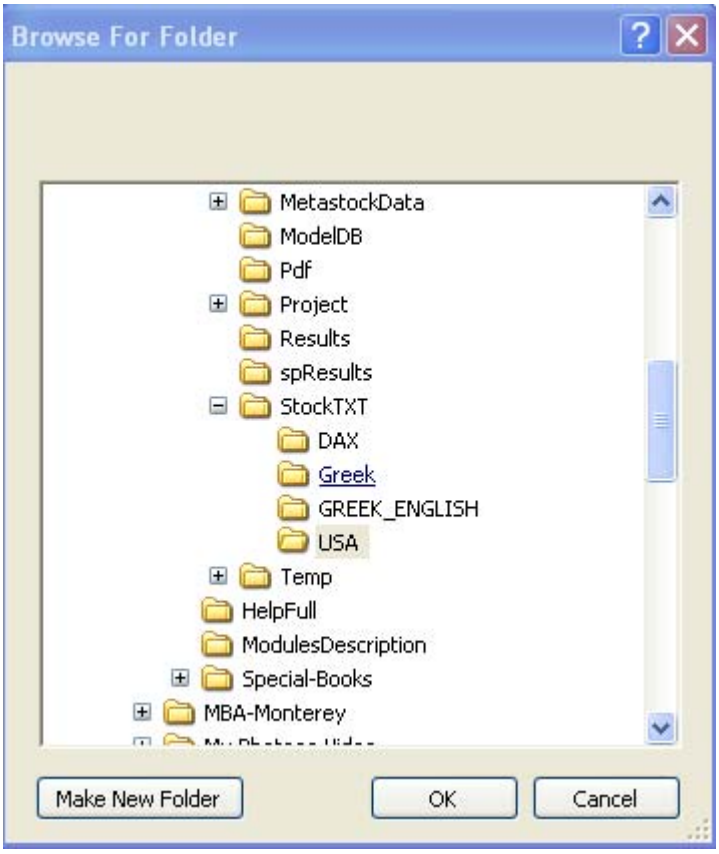
```
<TICKER>,<DTYYYYMMDD>,<OPEN>,<HIGH>,<LOW>,<CLOSE>,<VOL>
```

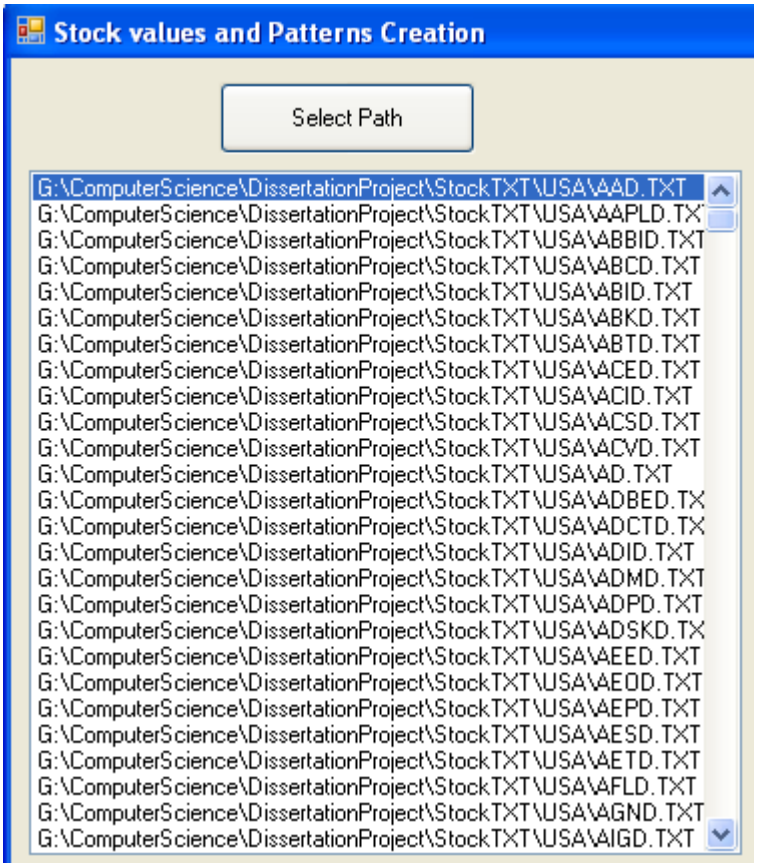
For example:

```
AA,20060927,14.3200,14.7400,14.3000,14.6400,361
```

Before loading security values, we should select the path that txt files

The following forms are displayed.:



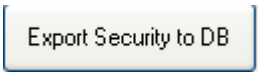


Then, we have two options about loading stock values from the files:

- To load one per time

- To load multiple stock values (the selected)

The first option 

load the selected security values into memory. In the right side of the displayed form we can see the symbol loaded. In this case we can fill the rest of the text input fields (Name, type of security and country origin) and then post updates. In order to transfer the data from memory to database we should press button 

.

The second option is to transfer the data of selected stock values directly to the database. The next operation of this form is to create and save patterns. Here, there are also two options: To creatre one pattern per time or to create patterns for all selected stocks.

When it is selected 

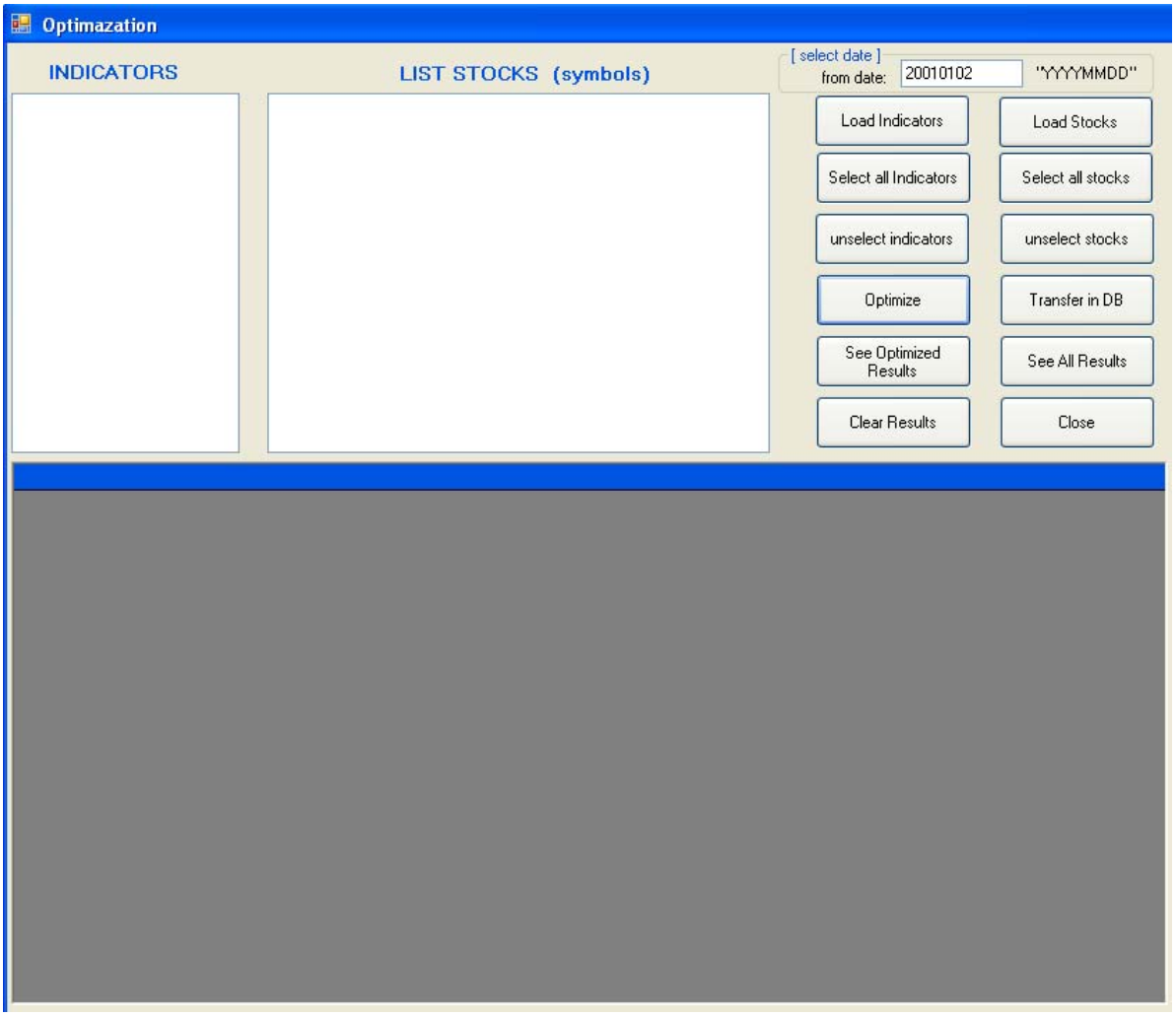
it means that previous we have load a security through button “Load Security”. The same is in case we are going to create patterns for all selected stocks 

. Before that, we should have created the stock values of selected stocks.

5.3 Optimazation

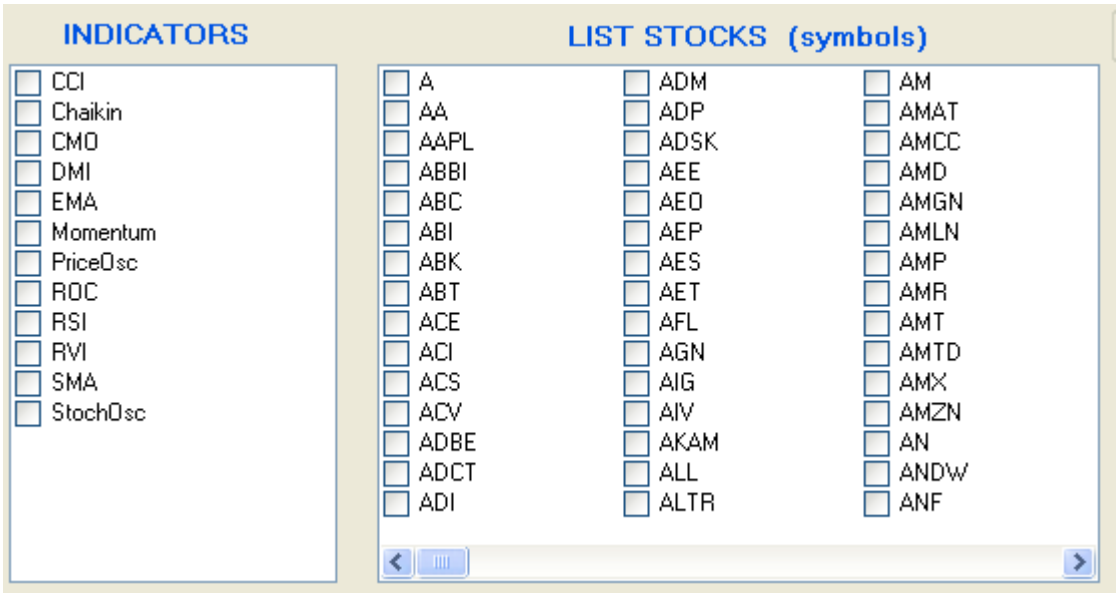
There have been developed specific trading systems that can produce signals for buying or selling (long position). These signals are dpendent on the values of the input indicators’ parameters. Here, the case is to find the specific values of the parameters for specific stocks that produce the best trading profits. The optimized parameter values for each trading system correspond to specific stock can be stored to be used later.

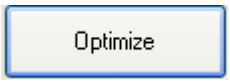
The following form is for optimization processes:



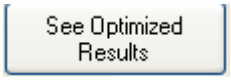
-First, we should load the indicators and the stocks from database (buttons: “Load Indicators” and “Load Stocks”)

- The next step is to select the indicators and the stocks that are to be optimized.



- The button 

executes the optimization process for each selected indicator to each selected stock.

- The button 

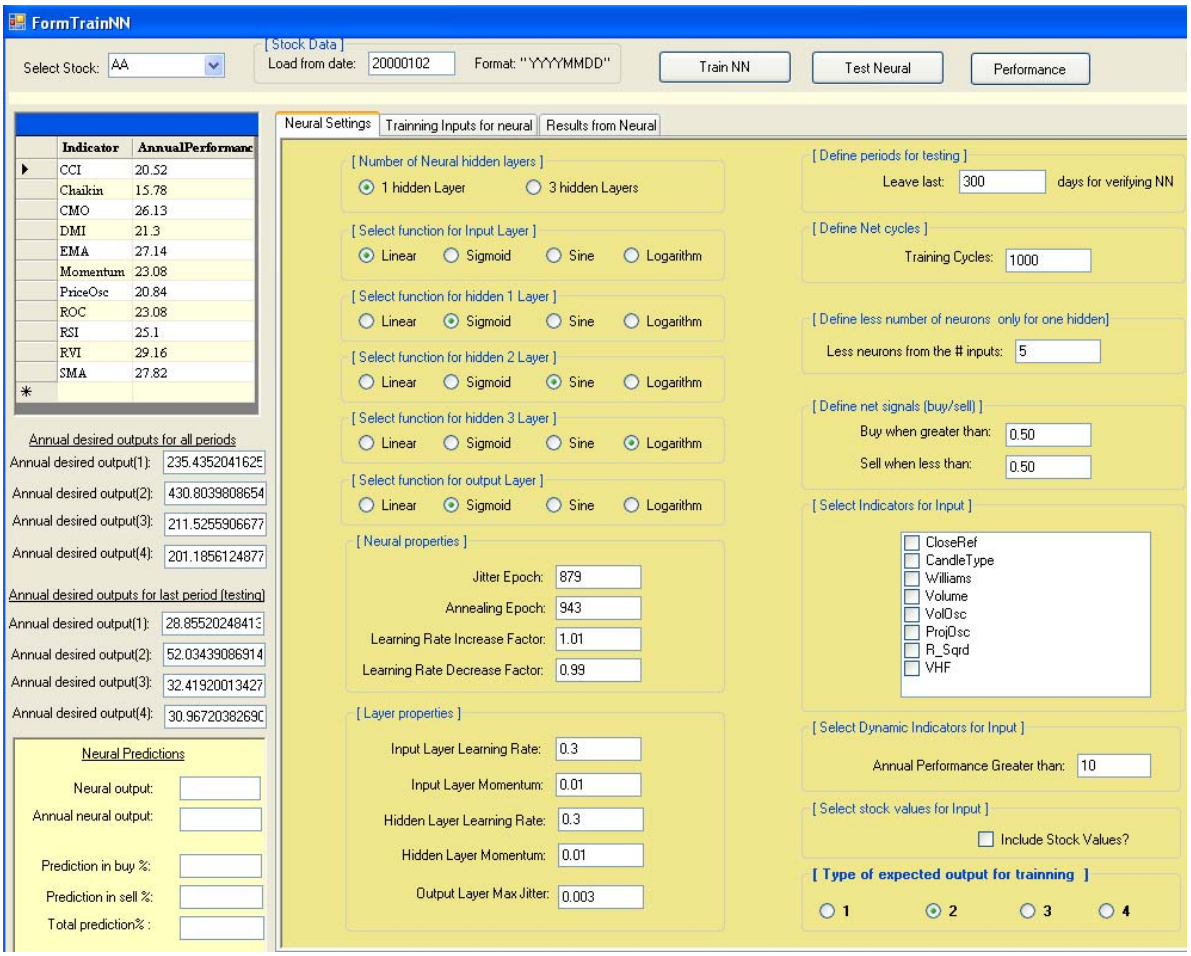
shows the results of best profitable performance.

- The button 

saves the optimized results into database.

5.4 Training the Neural Net

We can train the neural net for a specific stock and then test it to see the results.

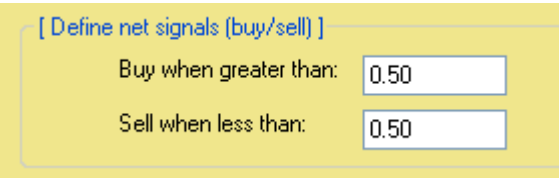


Training the neural for a specific stock:

- First, we should select a specific stock from the drop down list stoks.

- After selecting a specific stock, it appears the list of the indicators with their performance corresponds to selected stock.
- Also it appears two types of desired outputs: The one part is the desired outputs for all periods (from selected date to the latest date) and the second parts reffered to the latest periods defined in Neural Settings (Leave last...days for verifying the NN). It can be seen 4 type of outputs (type "2" is the best desired output).
- The next step is to define the setting for the neural net. Although there are default values they can be changed. The most important things in the Neural Settings page are the following:
    - Select the hidden layers (1 or 3)
    - Define the periods for testing the nueral
    - Select the basic indicators to be used as input to the neural. These type of indicators do not produce trading signals.
    - To select the expected output for training the neural.

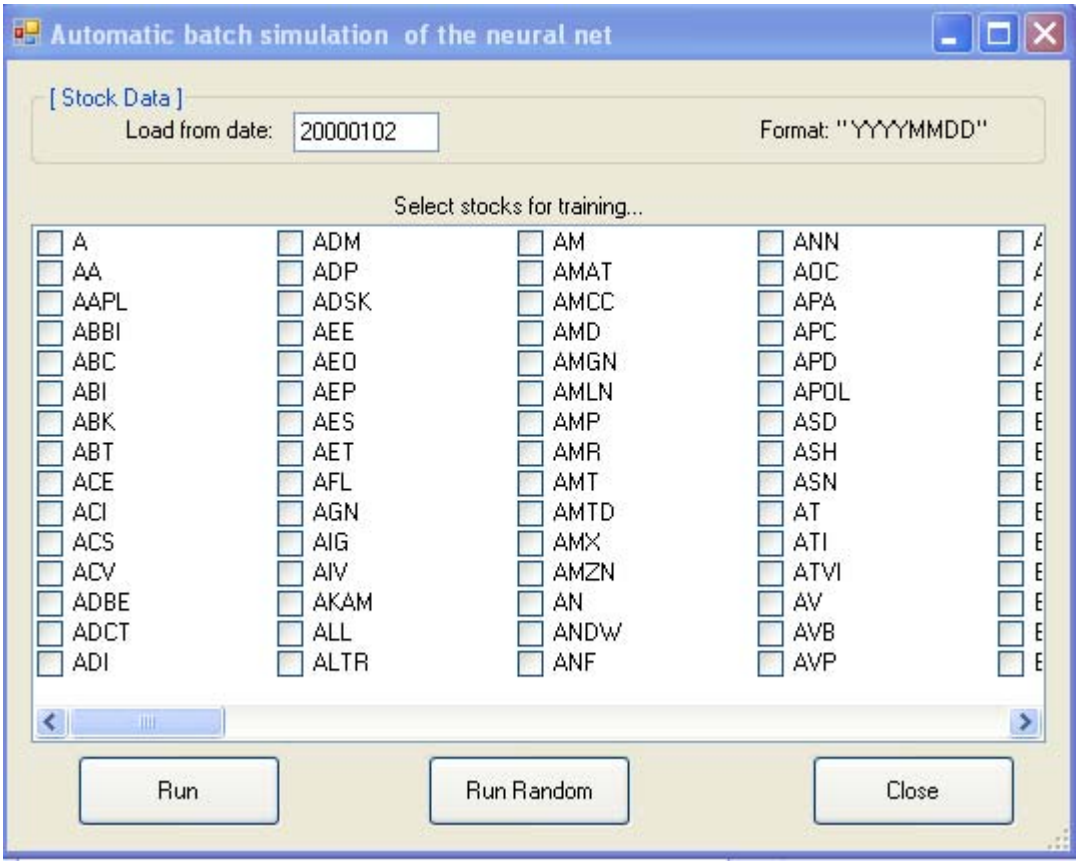
- Then, we train the neural by pressing the button "Train NN". The training inputs for the neural are displayed in the tabbed page "Traning Inputs for neural".
- To see the performance of the network we press the button "Test Neural".
- The results of neural testing are displayed in "Neural Predictions" (left bottom side)
- The "Performance" button used for testing the neural performance based on its output values (0, 1). The values for producing signals defined in the neural settings page as following:

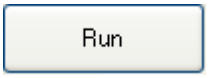


We can compare the performance of the network with the performance of various technical indicators and decide which type of trading system should be used for a specific stock.

5.5 Neural Automatic Running

The main task is to train the neural a hundrend of times for a specific stock and use the neural that brings the best results. These results are saved automatically into database. The following form is displayed. First, it must be selected the stocks for training and testing.



The first option is  button. This selection trains the neural for specific parameters which are some thousands different combinations of the neural parameters. All results are saved to the database.

The other option  , is similar to previous. The difference is this option combines random values for the parameters of the neural.

6. Conclusion

6.1 Conclusion

The results reveal us that there are enough more specials patterns beyond the well known candlesticks patterns which in some cases produce more accurate signals and referring to a long time of period ahead. The key of the success is finding a model for proper knowledge representation. Although this model it developed in short of period of time it needs more improvement. The expansion towards gvol-candlesticks helped enough for discovering new profitable patterns. Finally, the implementation of this model using proper software tools would led to high profitable results. Using special techniques for recognizing hidden patterns would be the base of an expert system for producing alerts systems.

On the other hand, using the neural nets for investment in the stock market can be benefiacial. What is found according to this research was the following:

- Neural nets produced in 13% of the total stocks better results comparing with the best traditional trading systems and 44% of the total stocks better results from the average of performance of the traditional trading systems. It means that they can work better in portion of stocks.
- The main focus should be on the input system of the neural.
- Neurals seems to have the ability to play the role of the advisor. It meens that they can be combined with trading traditional systems and help for entry or exit points.
- To get better results from the neurals is needed to repeat training a hundrends of times with using different parameters until get a satisfied result.
- According to this research it is has been shoowed that neurals can help investors in stock market. But the traditional technical trading systems it seems to be required.
- The better results can be produced by combining traditional trading systems and neural nets.

6.2 Future Improvements

There is a lot of room for future improvements. This research hopefully is the trigger for further significant improvements. Most important of them are the following:

(i) There is a room to extend the codification system in order to include more parameters, such as value come from technical indicators. For example, if RSI(14) > 70 return “0” (overbought) otherwise return ‘1’ (oversold).

(ii) It can be developed more pattern frame generators consisting more than three gvol-candlesticks.

(iii) It may be processed a huge amount of data from other stock markets and making comparisons among the markets. The disadvantage of this is the demand a lot of computer power (CPU). To be specific, in order to take these results (it was 1.5 million of Time Series Data and were generated more than 18 millions of patterns) were used simultaneously two computers working at CPU Power 100% for more than 30 hours.

(iv) It can be developed a proper computer program for searching and displaying graphically these pattern frames.

(v) The results of pattern recognition can be used in a proper simulated system for producing buying or selling signals.

(vi) There is room for further improvements referred to trading systems used as inputs to the neural nets.

8. Installation

8.1 DVD files

The DVD contains the following files:

- The project source files (/project)
- The report file (/thesisDoc)
- The database sql server file in zip format /database)
- The database design file in Sybase power builder (/dbDesign)
- The Excel files of patterns resuls (/patterns)
- The Sql scripts of database procedures and views (/sql)
- The intallation exe files (/RunFilesLocal & /RunFilesRemote)

8.2 Program Installation

In order to be able to run the program is needed to do the following:

- First, install the database. You need to have installed the SQL Server 2005 in the local or remote computer. From the Microsoft SQL Server Management Studio – go to Object Explorer -> Databases (right click) -> Select “Attach” and attach the database file.
- To run the application when Sql Server is locally installed (same computer), you need just run the application from directory / RunFilesLocal
- To run the application when Sql Server is remotely installed (different computers), before running the application from the directory **/ RunFilesRemote,** it is needed to make some modifications in the connection file: connectionDB.txt Open the file and there is the following connection string:

“Data Source=DESKNIKITAS;Initial Catalog=StockMarket;User ID=nikitas;password=0375”

It is needed three modifications: Change the DataSource (the name of the computer in the network, the user name and password for the remote connection).

9. Appendixes

List of appendixes:

- **Appendix A:** List of US stocks and information for the data used in this research
- **Appendix B:** Results of all pattern frames discovered
- **Appendix C:** General views and database procedures
- **Appendix D:** All views referred to hidden patterns
- **Appendix E:** All source code of the project

Appendix A: List of US stocks and information for the data used in this research

SELECTED STOCKS FOR COURSE WORK				
A/A	**Stock Symbol (universal)**	**Data From Date (mm/dd/yy)**	**Data to Date (mm/dd/yy)**	**Amount of Time Series Data**
1	A	11/18/99	5/4/07	1874
2	AA	1/2/85	5/4/07	5635
3	AAPL	9/7/84	5/4/07	5715
4	ABBI	12/14/01	5/4/07	1355
5	ABC	4/4/95	5/4/07	3036
6	ABI	5/6/99	5/4/07	2007
7	ABK	7/11/91	5/4/07	3985
8	ABT	4/6/83	5/4/07	6074
9	ACE	3/26/93	5/4/07	3552
10	ACI	8/12/88	5/4/07	4706
11	ACS	9/27/94	5/4/07	3174
12	ACV	7/19/84	5/4/07	5750
13	ADBE	8/14/86	5/4/07	5220
14	ADCT	9/10/84	5/4/07	5700
15	ADI	7/19/84	5/4/07	5743
16	ADM	4/6/83	5/4/07	6069
17	ADP	4/6/83	5/4/07	6074
18	ADSK	7/1/85	5/4/07	5504
19	AEE	1/2/98	5/4/07	2341
20	AEO	4/14/94	5/4/07	3283
21	AEP	1/2/70	5/4/07	9424
22	AES	6/26/91	5/4/07	3997
23	AET	1/3/77	5/4/07	7651
24	AFL	7/19/84	5/4/07	5743
25	AGN	6/22/89	5/4/07	4498
26	AIG	9/7/84	5/4/07	5712
27	AIV	7/22/94	5/4/07	3220
28	AKAM	10/29/99	5/4/07	1888
29	ALL	6/3/93	5/4/07	3507
30	ALTR	4/4/88	5/4/07	4774
31	AM	9/7/84	5/4/07	5709
32	AMAT	9/7/84	5/4/07	5716
33	AMCC	12/8/97	5/4/07	2353
34	AMD	3/21/83	5/4/07	6085
35	AMGN	9/7/84	5/4/07	5716
36	AMLN	1/17/92	5/4/07	3849
37	AMP	10/3/05	5/4/07	399
38	AMR	1/2/80	5/4/07	6894
39	AMT	6/5/98	5/4/07	2236
40	AMTD	3/4/97	5/4/07	2552
41	AMX	2/13/01	5/4/07	1561
42	AMZN	5/16/97	5/4/07	2498
43	AN	3/3/92	5/4/07	3806
44	ANDW	9/7/84	5/4/07	5708
45	ANF	9/26/96	5/4/07	2667
46	ANN	5/17/91	5/4/07	4018
47	AOC	9/7/84	5/4/07	5707
48	APA	12/31/81	5/4/07	6387

SELECTED STOCKS FOR COURSE WORK				
A/A	Stock Symbol (universal)	Data From Date (mm/dd/yy)	Data to Date (mm/dd/yy)	Amount of Time Series Data
49	APC	9/9/86	5/4/07	5203
50	APD	4/6/83	5/4/07	6069
51	APOL	1/4/95	5/4/07	3100
52	ASD	2/7/95	5/4/07	3075
53	ASH	4/6/83	5/4/07	6069
54	ASN	1/12/90	5/4/07	4364
55	AT	7/19/84	5/4/07	5743
56	ATI	11/30/99	5/4/07	1859
57	ATVI	10/25/93	5/4/07	3343
58	AV	9/19/00	5/4/07	1661
59	AVB	6/5/98	5/4/07	2242
60	AVP	3/3/83	5/4/07	6051
61	AVY	9/7/84	5/4/07	5716
62	AW	9/11/91	5/4/07	3926
63	AXP	12/26/80	5/4/07	6650
64	AYE	7/19/84	5/4/07	5750
65	AZO	4/2/91	5/4/07	4050
66	BA	11/6/75	5/4/07	7916
67	BAC	5/29/86	5/4/07	5274
68	BAX	12/31/81	5/4/07	6388
69	BBBY	6/5/92	5/4/07	3751
70	BBT	3/26/90	5/4/07	4306
71	BBY	4/19/85	5/4/07	5561
72	BC	12/31/81	5/4/07	6387
73	BCR	4/6/83	5/4/07	6069
74	BDK	12/31/81	5/4/07	6388
75	BDX	4/6/83	5/4/07	6069
76	BEAS	4/11/97	5/4/07	2530
77	BEN	9/23/83	5/4/07	5958
78	BF-B	9/7/84	5/4/07	5716
79	BHI	4/7/87	5/4/07	5057
80	BIG	11/5/87	5/4/07	4908
81	BIIB	9/17/91	5/4/07	3931
82	BJS	7/20/90	5/4/07	4223
83	BK	9/7/84	5/4/07	5708
84	BLL	9/7/84	5/4/07	5714
85	BMC	3/26/90	5/4/07	4305
86	BMET	9/7/84	5/4/07	5707
87	BMS	9/7/84	5/4/07	5691
88	BMY	10/5/81	5/4/07	6449
89	BNI	4/2/81	5/4/07	6504
90	BOL	1/4/82	5/4/07	6387
91	BOT	10/21/05	5/4/07	385

SELECTED STOCKS FOR COURSE WORK				
A/A	Stock Symbol (universal)	Data From Date (mm/dd/yy)	Data to Date (mm/dd/yy)	Amount of Time Series Data
90	BOL	1/4/82	5/4/07	6387
91	BOT	10/21/05	5/4/07	385
92	BP	1/3/77	5/4/07	7657
93	BRCD	5/25/99	5/4/07	1998
94	BRCM	4/17/98	5/4/07	2274
95	BRL	6/15/87	5/4/07	4963
96	BSC	10/30/85	5/4/07	5419
97	BSX	5/19/92	5/4/07	3763
98	BTU	5/22/01	5/4/07	1493
99	BUD	4/6/83	5/4/07	6069
100	BXP	6/18/97	5/4/07	2479
101	BYD	10/18/93	5/4/07	3407
102	BZH	2/23/94	5/4/07	3322
103	C	1/3/77	5/4/07	7657
104	CA	9/7/84	5/4/07	5709
105	CAG	9/7/84	5/4/07	5708
106	CAH	1/4/88	5/4/07	4877
107	CAL	7/19/93	5/4/07	3475
108	CAT	1/2/70	5/4/07	9426
109	CB	9/7/84	5/4/07	5708
110	CBE	1/4/82	5/4/07	6387
111	CBH	3/26/90	5/4/07	4260
112	CBS	1/3/06	5/4/07	336
113	CBSS	9/7/84	5/4/07	5704
114	CC	9/7/84	5/4/07	5708
115	CCE	11/24/86	5/4/07	5147
116	CCI	8/31/98	5/4/07	2182
117	CCK	12/18/84	5/4/07	5638
118	CCL	1/5/89	5/4/07	4622
119	CCU	3/17/92	5/4/07	3814
120	CDNS	3/26/90	5/4/07	4308
121	CDWC	5/27/93	5/4/07	3498
122	CEG	9/7/84	5/4/07	5715
123	CEGE	2/1/93	5/4/07	3577
124	CELG	3/26/90	5/4/07	4303
125	CEPH	4/25/91	5/4/07	4031
126	CERN	3/26/90	5/4/07	4314
127	CFC	11/5/87	5/4/07	4916
128	CG	2/1/02	5/4/07	1320
129	CHK	2/16/93	5/4/07	3561
130	CHKP	6/28/96	5/4/07	2730
131	CHRW	10/16/97	5/4/07	2393
132	CI	3/31/82	5/4/07	6331
133	CIEN	2/7/97	5/4/07	2573
134	CINF	3/26/90	5/4/07	4307
135	CIT	7/5/02	5/4/07	1216
136	CKFR	9/28/95	5/4/07	2913

SELECTED STOCKS FOR COURSE WORK				
A/A	Stock Symbol (universal)	Data From Date (mm/dd/yy)	Data to Date (mm/dd/yy)	Amount of Time Series Data
135	CIT	7/5/02	5/4/07	1216
136	CKFR	9/28/95	5/4/07	2913
137	CKP	11/12/86	5/4/07	5158
138	CL	4/6/82	5/4/07	6317
139	CLX	3/21/83	5/4/07	6087
140	CMA	3/26/90	5/4/07	4306
141	CMCSA	7/7/88	5/4/07	4741
142	CMCSK	3/26/90	5/4/07	4307
143	CME	12/24/02	5/4/07	1097
144	CMI	12/18/84	5/4/07	5636
145	CMS	12/31/84	5/4/07	5634
146	CNP	5/10/83	5/4/07	6045
147	CNX	4/30/99	5/4/07	2009
148	COF	11/16/94	5/4/07	3132
149	COH	10/6/00	5/4/07	1651
150	COL	7/3/01	5/4/07	1465
151	COP	1/4/82	5/4/07	6386
152	COST	7/9/86	5/4/07	5246
153	CPB	7/1/85	5/4/07	5503
154	CPWR	12/16/92	5/4/07	3616
155	CR	11/1/84	5/4/07	5669
156	CREE	2/9/93	5/4/07	3571
157	CRM	6/23/04	5/4/07	722
158	CRS	11/5/87	5/4/07	4916
159	CSC	12/31/81	5/4/07	6388
160	CSCO	3/26/90	5/4/07	4314
161	CSX	11/3/80	5/4/07	6687
162	CTAS	3/26/90	5/4/07	4307
163	CTL	11/5/87	5/4/07	4909
164	CTSH	6/19/98	5/4/07	2224
165	CTX	7/11/85	5/4/07	5501
166	CTXS	12/8/95	5/4/07	2862
167	CVC	3/17/92	5/4/07	3814
168	CVG	8/13/98	5/4/07	2194
169	CVH	4/17/91	5/4/07	4040
170	CVS	12/17/84	5/4/07	5638
171	CVX	11/4/71	5/4/07	8952
172	CY	4/5/89	5/4/07	4554
173	CZN	3/26/90	5/4/07	4286
174	D	10/3/84	5/4/07	5690
175	DD	5/19/76	5/4/07	7746
176	DDS	6/30/89	5/4/07	4492
177	DE	1/4/82	5/4/07	6387
178	DF	1/9/02	5/4/07	1338
179	DG	9/21/89	5/4/07	4435
180	DGX	12/26/96	5/4/07	2603
181	DHI	6/5/92	5/4/07	3758
182	DHR	11/5/87	5/4/07	4908
183	DIS	1/2/70	5/4/07	9319

SELECTED STOCKS FOR COURSE WORK				
A/A	Stock Symbol (universal)	Data From Date (mm/dd/yy)	Data to Date (mm/dd/yy)	Amount of Time Series Data
184	DISCA	7/8/05	5/4/07	459
185	DISH	6/21/95	5/4/07	2982
186	DJ	7/1/85	5/4/07	5503
187	DLTR	3/9/95	5/4/07	3054
188	DLX	7/23/87	5/4/07	4983
189	DNA	7/9/86	5/4/07	5230
190	DO	10/11/95	5/4/07	2905
191	DOV	7/1/85	5/4/07	5510
192	DOW	1/3/77	5/4/07	7655
193	DRI	5/17/95	5/4/07	3006
194	DTE	9/14/79	5/4/07	6967
195	DUK	4/6/83	5/4/07	6069
196	DWA	10/28/04	5/4/07	633
197	DYN	11/10/93	5/4/07	3395
198	EBAY	9/24/98	5/4/07	2165
199	ECL	1/5/88	5/4/07	4869
200	ED	5/7/81	5/4/07	6553
201	EDS	10/19/84	5/4/07	5685
202	EFX	11/5/87	5/4/07	4909
203	EIX	1/2/80	5/4/07	6898
204	EK	1/2/70	5/4/07	9426
205	EL	11/17/95	5/4/07	2878
206	ELN	3/17/92	5/4/07	3807
207	ELX	3/26/90	5/4/07	4314
208	EMC	12/16/88	5/4/07	4632
209	EMN	12/14/93	5/4/07	3370
210	EMR	1/4/82	5/4/07	6387
211	EOG	10/4/89	5/4/07	4431
212	EP	3/13/92	5/4/07	3809
213	EQ	5/18/06	5/4/07	242
214	EQR	8/12/93	5/4/07	3452
215	ERIC	9/28/89	5/4/07	4437
216	ERTS	3/26/90	5/4/07	4307
217	ETFC	8/16/96	5/4/07	2688
218	ETN	7/1/85	5/4/07	5508
219	ETR	12/31/81	5/4/07	6388
220	EXC	1/2/80	5/4/07	6898
221	EXP	4/12/94	5/4/07	3290
222	EXPD	3/26/90	5/4/07	4276
223	EXPE	7/21/05	5/4/07	450
224	EYE	7/8/02	5/4/07	1215
225	F	9/3/80	5/4/07	6657
226	FAST	3/26/90	5/4/07	4314
227	FCX	7/10/95	5/4/07	2970
228	FD	2/5/92	5/4/07	3835

SELECTED STOCKS FOR COURSE WORK				
A/A	Stock Symbol (universal)	Data From Date (mm/dd/yy)	Data to Date (mm/dd/yy)	Amount of Time Series Data
229	FDC	4/9/92	5/4/07	3790
230	FDO	11/5/87	5/4/07	4909
231	FE	11/10/97	5/4/07	2384
232	FFIV	6/4/99	5/4/07	1983
233	FHN	3/26/90	5/4/07	4307
234	FII	5/14/98	5/4/07	2251
235	FISV	3/26/90	5/4/07	4307
236	FITB	3/26/90	5/4/07	4307
237	FL	1/2/70	5/4/07	9425
238	FLEX	3/18/94	5/4/07	3297
239	FLR	12/22/00	5/4/07	1594
240	FNM	1/3/77	5/4/07	7649
241	FO	7/5/90	5/4/07	4237
242	FPL	6/10/83	5/4/07	6030
243	FRE	12/2/88	5/4/07	4636
244	FREE	12/16/05	5/4/07	346
245	FRX	4/19/88	5/4/07	4803
246	FTO	1/5/88	5/4/07	4870
247	GAS	10/22/84	5/4/07	5677
248	GCI	7/1/85	5/4/07	5503
249	GD	1/3/77	5/4/07	7649
250	GE	3/1/84	5/4/07	5847
251	GENZ	3/26/90	5/4/07	4307
252	GILD	1/22/92	5/4/07	3844
253	GIS	6/10/83	5/4/07	6023
254	GLW	12/31/81	5/4/07	6390
255	GM	1/2/70	5/4/07	9426
256	GNTX	3/26/90	5/4/07	4307
257	GNW	5/25/04	5/4/07	741
258	GOOG	8/19/04	5/4/07	682
259	GPC	4/6/83	5/4/07	6069
260	GPS	7/23/87	5/4/07	4990
261	GR	7/1/85	5/4/07	5503
262	GRMN	12/14/00	5/4/07	1601
263	GS	5/4/99	5/4/07	2006
264	GT	1/2/70	5/4/07	9419
265	GTW	12/9/93	5/4/07	3368
266	GWW	12/17/84	5/4/07	5638
267	HAL	12/31/81	5/4/07	6388
268	HAR	10/5/88	5/4/07	4640
269	HAS	12/18/84	5/4/07	5637
270	HBAN	3/26/90	5/4/07	4307
271	HCR	10/18/91	5/4/07	3910
272	HD	8/20/84	5/4/07	5728
273	HES	4/6/83	5/4/07	6070
274	HET	12/31/81	5/4/07	6388
275	HGSI	12/2/93	5/4/07	3373

SELECTED STOCKS FOR COURSE WORK				
A/A	Stock Symbol (universal)	Data From Date (mm/dd/yy)	Data to Date (mm/dd/yy)	Amount of Time Series Data
276	HIG	12/18/95	5/4/07	2857
277	HLT	4/6/83	5/4/07	6069
278	HLTH	2/11/99	5/4/07	2061
279	HMA	2/6/91	5/4/07	4079
280	HNZ	12/17/84	5/4/07	5638
281	HOG	11/5/87	5/4/07	4910
282	HON	1/2/70	5/4/07	9426
283	HOT	11/5/87	5/4/07	4898
284	HPC	1/4/82	5/4/07	6386
285	HPQ	1/5/70	5/4/07	9249
286	HRB	11/12/86	5/4/07	5164
287	HSIC	11/3/95	5/4/07	2886
288	HSP	5/3/04	5/4/07	757
289	HSY	7/1/85	5/4/07	5510
290	HUM	12/31/81	5/4/07	6388
291	IACI	1/19/93	5/4/07	3601
292	IBM	1/2/70	5/4/07	9426
293	ICE	11/16/05	5/4/07	367
294	IFF	12/31/81	5/4/07	6388
295	IGT	3/26/90	5/4/07	4307
296	IMCL	11/19/91	5/4/07	3889
297	INTC	7/9/86	5/4/07	5253
298	INTU	3/22/93	5/4/07	3551
299	IP	3/15/72	5/4/07	8855
300	IPG	11/5/87	5/4/07	4916
301	IR	7/1/85	5/4/07	5503
302	ISIL	3/13/00	5/4/07	1789
303	ISRG	6/23/00	5/4/07	1719
304	ITT	12/18/95	5/4/07	2862
305	ITW	11/5/87	5/4/07	4909
306	IVGN	2/26/99	5/4/07	2050
307	JBL	5/3/93	5/4/07	3515
308	JBLU	4/18/02	5/4/07	1271
309	JCI	3/27/85	5/4/07	5569
310	JCP	1/4/82	5/4/07	6387
311	JDSU	11/17/93	5/4/07	3390
312	JNJ	5/19/80	5/4/07	6804
313	JNPR	6/25/99	5/4/07	1976
314	JNS	6/26/00	5/4/07	1723
315	JNY	5/16/91	5/4/07	4018
316	JOE	3/10/92	5/4/07	3813
317	JOYG	6/15/01	5/4/07	1478
318	JPM	12/30/83	5/4/07	5889
319	JWN	7/9/86	5/4/07	5253
320	K	12/17/84	5/4/07	5645

SELECTED STOCKS FOR COURSE WORK				
A/A	Stock Symbol (universal)	Data From Date (mm/dd/yy)	Data to Date (mm/dd/yy)	Amount of Time Series Data
321	KBH	8/4/86	5/4/07	5235
322	KCI	3/2/04	5/4/07	800
323	KEY	11/5/87	5/4/07	4909
324	KFT	6/14/01	5/4/07	1477
325	KG	6/25/98	5/4/07	2226
326	KIM	11/22/91	5/4/07	3886
327	KKD	4/6/00	5/4/07	1775
328	KLAC	3/26/90	5/4/07	4307
329	KMB	12/17/84	5/4/07	5638
330	KMI	1/5/88	5/4/07	4862
331	KO	10/8/82	5/4/07	6195
332	KR	7/30/85	5/4/07	5483
333	KSE	6/1/98	5/4/07	2239
334	KSS	5/19/92	5/4/07	3763
335	LAMR	8/2/96	5/4/07	2692
336	LBTYA	6/3/04	5/4/07	735
337	LCC	9/29/05	5/4/07	401
338	LEG	11/5/87	5/4/07	4916
339	LEH	5/11/94	5/4/07	3264
340	LEN	11/5/87	5/4/07	4916
341	LH	3/29/90	5/4/07	4292
342	LIZ	7/9/86	5/4/07	5246
343	LLL	5/19/98	5/4/07	2248
344	LLTC	3/26/90	5/4/07	4307
345	LLY	1/4/82	5/4/07	6394
346	LM	11/5/87	5/4/07	4916
347	LMT	1/3/77	5/4/07	7657
348	LNC	10/5/84	5/4/07	5688
349	LNCR	3/19/92	5/4/07	3805
350	LNG	10/12/01	5/4/07	1397

NUMBER OF STOCKS: **350**

MIN DATE OF DATA FROM: **2 JAN 1970**

SUM OF DATA:
1,561,023

Appendix B: Results of all pattern frames discovered

BEST RESULTS OF PATTERN FRAME OF TWO FOR BUY SIGNALS							
FREQUNCY	PATTERN OF TWO	UP% NEXT 1	UP% NEXT 3	UP% NEXT 5	UP% NEXT 7	UP% NEXT 10	UP% NEXT 15
31	00000100000101011111110010010	54.84	70.97	70.97	80.65	83.87	90.32
32	00100111100101011011001100011	59.38	68.75	71.88	75	68.75	84.38
38	00101011000000000000000001010011	76.32	73.69	76.31	81.57	81.58	84.21
31	01011011110101101111110010000	51.61	54.84	70.97	74.19	83.87	83.87
31	11011011100101011110110011011	54.84	29.04	61.29	70.97	67.74	83.87
37	00101011000010101101111011011	81.08	72.97	72.97	70.27	75.68	83.79
37	00000000011001110001001100011	67.57	62.16	67.57	64.86	64.86	83.78
36	10101011100010011101110011011	50	58.34	50	52.78	61.11	83.33
35	00000000011010101111110100000	71.42	57.14	60	57.14	77.14	82.86
34	10011000000101011110001100010	50	47.06	52.94	35.29	52.94	82.35
33	11011010110010101111110010000	75.75	84.85	78.79	72.73	72.73	81.82
33	00100111101001101111110011011	57.58	66.67	69.7	72.73	81.82	81.82
44	00000110000101011011110010011	59.09	65.91	75	61.36	68.18	81.81
32	00101011010010011100001100000	59.38	75	68.76	59.38	81.25	81.25
37	01011010101010101101110101011	78.37	75.68	78.38	81.08	78.38	81.08
31	01011010100000011000001100011	67.74	67.75	67.75	77.42	77.42	80.65
31	01011010111010101100110101000	70.96	64.52	61.29	58.06	67.74	80.65
31	01010111110010101110001100000	58.06	61.29	58.06	70.97	77.42	80.64
36	10001000000000010000001100011	50	72.22	75	75	75	80.56
41	10011010000000011000110011000	60.98	60.97	58.53	53.66	63.42	80.49
50	10101011010010101111110010000	62	66	76	70	76	80
49	00101011010000010000001100000	65.31	57.14	69.39	67.35	61.22	79.59
44	00000000010000010000000100010	65.91	77.28	77.28	81.81	84.09	79.54
34	00000000010010101111001100011	58.82	64.7	64.71	73.53	76.47	79.41
43	10101011010010011101110011011	60.46	65.11	72.09	74.41	74.42	79.07
33	10101011000000001000000001000110	72.72	69.69	54.54	66.67	63.64	78.79
37	00100111000010101100001100000	78.38	75.67	83.79	81.09	72.98	78.38

1	BEST RESULTS OF PATTERN FRAME OF TWO FOR SELL SIGNALS							
2	FREQUNCY	PATTERN ALL	DOWN% NEXT 1	DOWN% NEXT 3	DOWN% NEXT 5	DOWN% NEXT 7	DOWN% NEXT 10	DOWN% NEXT 15
3	29	00000000000000010000011101000000	89.65	79.31	86.21	89.66	82.76	93.1
4	30	00000011101000001000000011000100	66.66	40	73.34	80	66.67	66.67
5	28	10011000000000001110111001110 11	64.29	71.43	71.43	78.57	60.71	67.86
6	26	00100111100000010000110001001 0	80.77	88.46	88.46	76.92	69.23	69.23
7	36	01011011111001110001111011000 0	63.89	72.23	72.22	75	63.89	66.67
8	28	10011100000010011110110011001 1	75	71.43	71.43	75	57.14	60.71
9	31	11011010100000010000001010001 0	64.52	58.06	48.39	74.2	64.52	45.16
10	27	00000111001001110000111011000 0	62.96	77.78	77.78	74.07	66.67	74.07
11	27	01010111100000010000110011000 0	33.33	55.56	77.78	74.07	66.67	48.15
12	26	01010111100101101110001100001 0	57.69	80.77	69.23	73.08	69.23	69.23
13	29	00000100001000101110110010000 0	55.17	55.17	68.97	72.42	58.62	48.28
14	28	11011011101001110000110001001 0	60.71	75	71.43	71.43	60.71	67.86
15	28	00100111111001110000110001000 0	64.29	60.71	53.57	71.43	53.57	57.14
16	28	10011010100000011001001100001 1	53.57	60.71	64.28	71.43	57.14	50
17	28	00000110001010101111110011001 1	60.71	57.14	67.86	71.43	75	53.57
18	28	00100111111010101100110100000 0	53.57	67.86	64.29	71.42	71.43	67.86
19	28	10101011000101011010001100011 1	64.29	67.86	67.86	71.42	60.71	60.71
20	31	10011010100010011110001010001 0	45.17	64.52	77.42	70.97	70.97	54.84
21	27	00000100010000101110001100000 0	66.67	66.67	70.37	70.37	55.56	70.37
22	27	00000111111001101001110001001 0	44.45	51.85	44.44	70.37	55.56	59.26
23	47	10011000011001100001011000010	61.71	65.96	70.22	70.22	55.32	55.32
24	30	11011010110000011111001100000 0	66.66	73.33	60	70	63.33	53.33
25	30	00000110011001101110110011000 0	63.34	56.67	63.33	70	46.67	50
26	36	00000100000101101010001100001 1	75	69.44	69.45	69.45	72.23	55.56
27	36	10011100010101101111001010001 1	63.89	66.67	69.45	69.45	66.67	55.56
28	26	11011010101001101001110011101 1	69.23	61.54	61.54	69.23	65.38	38.46
29	39	00000000001001101000111010000 0	43.59	58.97	48.72	69.23	64.1	51.28
30	26	10011011000101101110110011001 0	53.85	53.84	61.54	69.23	57.69	57.7
31	29	11011010110010011111001100001 0	48.28	68.97	62.07	68.97	55.17	65.52
32	29	10001011101001100001110011001 1	51.72	62.07	55.17	68.97	62.07	58.62
33	29	11011010111001110001110001000 0	62.06	62.06	62.07	68.97	62.07	55.17
34	32	01010110100101101010001100001 1	59.38	65.63	75.01	68.76	56.26	46.88
35	32	11011011111010101111110011001 0	53.13	46.88	56.25	68.76	62.51	56.26
36	32	11011010101001101110001100001 1	71.88	65.63	65.63	68.75	56.25	53.13
37	32	01011011101001101110110001001 1	68.75	62.5	62.51	68.75	65.63	68.75
38	25	00000111001001110000110001001 0	56	64	64	68	68	48

BEST RESULTS OF PATTERN FRAME OF THREE FOR BUY SIGNALS							
FREQUNCY	PATTERN ALL	UP% NEXT 1	UP% NEXT 3	UP% NEXT 5	UP% NEXT 7	UP% NEXT 10	UP% NEXT 15
39	100110000000000000010000000001000100001100010000011 01	33.33	56.41	66.67	64.1	61.54	66.67
44	000001000010011000000000010000110011000000101100101 1	36.37	47.72	47.73	54.55	52.27	65.91
30	000001000001010110110000011000001100001100110000000 1	36.67	60	66.67	73.33	70	70
30	100110000010101011001001100000110011001011001100111 1	36.67	40	53.33	56.66	70	63.33
32	100110000110011000010000010001110011001000101100111 1	37.51	50.01	43.75	56.26	50	65.63
42	000001000000000110000000010001001100001100110000110 1	38.09	59.52	66.67	71.43	76.19	66.66
31	100110000110011000001010101100110011000011001100001 1	38.71	45.16	67.74	61.29	51.61	64.52
32	100110000011011010100000010000110011001000110000000 1	40.63	34.38	40.63	46.88	56.26	65.63
58	100110000000000100001001100000001100001111010000000 1	41.38	56.89	58.62	56.89	67.24	60.34
36	000000000100000100000000010000001100000000110000000 1	41.67	50	66.67	61.11	69.45	69.45
43	100110000010011000000000010001110011000000101100111 1	41.86	48.84	51.17	69.77	65.12	65.12
35	100110100010011000010000010000110011001100101100001 1	42.85	54.29	42.86	65.71	62.86	68.57
30	100110000000000100000000010001001100001100110001101	43.33	56.67	56.67	66.67	56.67	66.67
32	100111000010000010000000000001001010000000010000110 1	43.76	56.25	62.5	71.88	62.5	62.51
43	100110000100000100001001100000001011000011010000000 1	44.18	48.84	60.47	51.16	65.12	69.77
36	010101101000000100000000010000001100000000110000000 1	44.44	44.45	61.12	69.45	61.11	63.89
42	101010110000000100010000010000001001001100110000000 1	45.24	59.52	54.76	69.05	64.28	71.43
35	010101101000000100000000010000001100000000110001101	45.71	51.43	51.43	60	62.86	74.29
83	100110000000000100011001100000001100001111010000000 1	45.78	48.19	53.01	57.83	61.44	67.47
48	100110000010011000010000010001110011001100101000001 1	45.83	54.17	62.5	66.66	66.67	72.92
37	000000000110011000001001100000110100000011001100001 1	45.94	54.06	54.06	56.76	64.87	78.38
41	000001100000000100000000010000001100001000110000110 1	46.34	53.66	53.66	56.1	63.42	65.85
43	000001000000000100000000010000001100000011001000110 1	46.51	62.79	58.14	58.14	62.8	69.77
43	000001000010101011001001100000110010000011001100111 1	46.51	39.54	58.14	58.14	55.82	69.77
45	000001000000000100011001101110001100001111001000000 1	46.66	55.55	66.67	66.66	64.44	60
30	100110100000000100000000010000001100001000110000100 1	46.66	50	60	63.33	60	66.67
30	000001000000000100001001101110001100001011001000100 1	46.67	63.33	63.33	60	66.67	50
30	100111000101010110100000010000001010000000110000000 1	46.67	43.34	50	56.66	50	70
32	000001000011011010101001100000110010001011001100101 1	46.88	65.63	75.01	65.63	68.75	62.5
32	010101101000000100000000010000001100001100110000110 1	46.88	62.5	68.76	50.01	68.75	56.25
32	000001000010011000000000010001110100001000110000110 1	46.88	50	62.5	62.5	59.38	68.75
38	010101101100000100010000010000001100000000110000000 1	47.37	57.89	60.52	65.79	57.89	65.79
40	000001000000000100000001010111000110000001100100011 01	47.5	52.5	52.5	62.5	72.5	57.5
46	100110000110011000010000010000110011001100110000000 1	47.82	56.52	52.17	54.35	60.87	65.22
46	100110000000000100000000010000001001000000110000100 1	47.83	56.52	58.69	60.87	69.57	63.04
56	100110000100000100000000010001001001000000110000110 1	48.22	53.57	58.93	60.72	64.29	66.07
31	000001000000000100010000011110001100001111001000000 1	48.39	54.84	61.29	61.29	70.97	61.29
33	000001110000000100000000010000001100000000110000110 1	48.48	51.51	60.61	54.55	63.64	69.7
33	000001000010011000000000010000110001001000101100001 1	48.48	66.67	57.58	63.64	60.61	69.7
33	100110000010011000011101101110110011001111001100001 1	48.48	69.7	57.58	39.39	57.58	60.61

BEST RESULTS OF PATTERN FRAME OF THREE FOR SELL SIGNALS							
FREQUNCY	PATTERN ALL	DOWN% NEXT 1	DOWN% NEXT 3	DOWN% NEXT 5	DOWN% NEXT 7	DOWN% NEXT 10	DOWN% NEXT 15
24	000001000001010110100000000001001100001100010000 1101	37.5	66.67	50	66.67	70.83	66.67
26	100110000110011000001001100000110011001011001100 1011	38.47	53.85	69.23	46.15	46.15	38.46
23	000001000100100111000000010000001100000000110000 1101	39.13	30.44	47.83	52.18	73.91	56.52
38	110110111000000100000000010000001001001000110000 1001	39.47	60.52	65.79	57.89	55.26	55.26
20	101010111000000100000000010000001001001000110000 1001	40	70	65	55	55	65
25	100110100000000100000000010001001100001000110000 1101	40	48	60	68	72	52
22	100110000010011000001010101100110011001111001100 1111	40.91	54.55	59.09	59.09	54.55	68.18
24	100110000110011000001001100001110011000011001100 1011	41.67	58.34	54.17	62.5	70.83	62.5
37	100110000010011000001001101110110011000011001100 1111	43.25	56.76	62.16	67.57	54.05	45.95
23	100110000010101011011101101010110011001111001100 0011	43.48	56.52	60.87	65.22	56.52	60.87
23	100110000010101011010000010000110011001100101100 0011	43.48	60.87	56.52	47.83	69.57	43.48
29	110110101010011000001001100000110011001011001100 0011	44.83	58.62	65.52	51.72	48.28	44.83
22	100110000101010110100000010001001011000000110000 1101	45.45	81.82	59.1	50	59.1	36.36
22	100111000010011000001001110001110011000011101100 1111	45.46	54.55	54.55	63.64	72.73	81.82
26	100110000010011000000101011010110011001000101100 1011	46.16	61.54	61.54	73.08	53.85	53.85
30	010101101000000100010101011010001100001100110000 0001	46.66	50	53.34	60	66.67	66.67
19	100110000110011000001001100000110011000011001110 1111	47.06	64.7	73.53	52.94	50	50
20	100110000100000110010000010000001011001000110000 0001	47.83	47.83	56.52	43.48	43.48	69.57
25	100111000100000000000000000010000100000000100001 1101	48	44	52	32	48	72
22	100110000001010110100000010000001011001100110000 0001	48.14	55.56	70.37	66.67	48.14	48.15
33	100111000010011100001001110001111011001111101100 1111	48.48	60.61	66.67	42.42	51.51	45.45
24	010101101010011000000000010000110100000000110000 1101	50	70	65	75	85	80
25	000000000000000100001001110000001100000011101100 1111	50	60	70	75	70	75
26	100110000000000100011001100000001010001111000100 0011	50	76.92	69.23	65.38	65.39	65.38
27	100110000110011000011101101110110011001011001100 0011	50	60	75	80	65	65
28	100110110010011000011001100000110011001111001100 0011	50	81.82	63.64	63.64	68.18	63.64
29	000001000000000100001001100001001100001011001000 1111	50	60	55	55	75	60
30	000001000010011010001001100000110100001011001100 0011	50	68.19	59.09	63.64	72.73	59.09
31	010101101010011000011001100001110011001111001100 0011	50	66.67	70.83	70.83	58.34	58.34
32	100110000010011011101001100000110011001111001100 0011	50	59.09	54.55	68.18	68.18	54.55
33	100110000111011010110000010000110011000000110000 0001	50	65.39	65.39	69.23	53.85	53.85
34	001001110010011000001001100000110011001111001100 1011	50	73.08	57.69	73.08	65.39	53.85

Appendix C: General views and database procedures

PROCEDURE: AddSecurityValues	Source: generalViewsProc.sql

```
CREATE PROCEDURE AddSecurityValues @symbol varchar(8), @dateStr varchar(8),
                                        @open_price decimal(18, 4), @high_price
decimal(18, 4), @low_price decimal(18, 4),
@close_price decimal(18, 4), @volume int
AS
      DECLARE @theyear varchar(4), @themonth varchar(2), @thedate varchar(2),
@frmdate varchar(10)

    SET @theyear  = SUBSTRING(@dateStr, 1, 4);
    SET @themonth = SUBSTRING(@dateStr, 5, 2);
    SET @thedate  = SUBSTRING(@dateStr, 7, 2);
    SET @frmdate = @theyear+'.'+@themonth+'.'+@thedate;

      INSERT INTO security_values ( symbol, date_value, open_price, high_price,
low_price, close_price, volume )
    VALUES (@symbol, CONVERT(datetime, @frmdate, 102),  @open_price,
@high_price, @low_price, @close_price, @volume )
```

PROCEDURE: AddSecurity	Source: generalViewsProc.sql

```
CREATE PROCEDURE AddSecurity @symbol varchar(8), @name_of_symbol varchar(30),
@type_of_security varchar(11), @country_origin varchar(25)
AS
      INSERT INTO security ( symbol, name_of_symbol, type_of_security,
country_origin )
    VALUES (@symbol, @name_of_symbol, @type_of_security, @country_origin )
```

PROCEDURE: GetSecurity	Source: generalViewsProc.sql

```
CREATE PROCEDURE GetSecurity @symbol varchar(8)
AS
      SELECT * FROM security WHERE symbol = @symbol
```

PROCEDURE: GetSecurityValues	Source: generalViewsProc.sql

```
CREATE PROCEDURE GetSecurityValues @symbol varchar(8), @dateStr varchar(8)
```

```
AS
      DECLARE @theyear varchar(4), @themonth varchar(2), @thedate varchar(2),
@frmdate varchar(10)

    SET @theyear  = SUBSTRING(@dateStr, 1, 4);
    SET @themonth = SUBSTRING(@dateStr, 5, 2);
    SET @thedate  = SUBSTRING(@dateStr, 7, 2);
    SET @frmdate = @theyear+'.'+@themonth+'.'+@thedate;

      SELECT   CAST(YEAR(date_value) AS char(4)) + CAST(MONTH(date_value) AS
char(2)) + CAST(DAY(date_value) AS char(2)) AS dateStr,
             open_price, high_price, low_price, close_price, volume FROM
security_values
    WHERE symbol = @symbol and date_value >= CONVERT(datetime, @frmdate, 102)
ORDER BY date_value
```

PROCEDURE: GetAITradeValues	Source: generalViewsProc.sql

```
CREATE PROCEDURE GetAITradeValues @symbol varchar(8)
AS

      SELECT  codeInd, annualPerformance, parameterDesc, parameter1, parameter2,
parameter3 FROM AITrade
    WHERE symbol = @symbol
```

PROCEDURE: addNeuralResult	Source: generalViewsProc.sql

```
CREATE PROCEDURE addNeuralResult @symbol varchar(8), @hiddenLayers smallint,
@lessNeurons int,
                @trainingCycles int, @numberOfBasIndicators int,
@validPerformance decimal(10,2),
                @outPutSelection smallint, @buyWhen decimal(10,2), @sellWhen
decimal(10,2),
                        @totalTradesPrediction decimal(10,2), @predictionOfBuy
decimal(10,2),
                        @predictionOfSell decimal(10,2), @annualNetPerformance
decimal(10,2)
AS

DECLARE @ncount int

SELECT @ncount = count(*) from neural_results where symbol = @symbol;

INSERT INTO neural_results ( symbol, id_net, hiddenLayers, lessNeurons,
trainingCycles,
```

```
                    numberOfBasIndicators, validPerformance, outPutSelection,
buyWhen, sellWhen,
                    totalTradesPrediction, predictionOfBuy, predictionOfSell,
annualNetPerformance)
VALUES    ( @symbol, @ncount+1, @hiddenLayers, @lessNeurons, @trainingCycles,
                    @numberOfBasIndicators, @validPerformance, @outPutSelection,
@buyWhen, @sellWhen,
                    @totalTradesPrediction, @predictionOfBuy,@predictionOfSell,
@annualNetPerformance)
```

PROCEDURE: AITradeUpdate	Source: generalViewsProc.sql

```
CREATE PROCEDURE AITradeUpdate @symbol varchar(8), @codeInd
varchar(8),@typeoftrade varchar(5), @fromdateappliedStr varchar(8),
                                @todateappliedStr varchar(8), @numberofstocks
int, @feepertrade decimal(10, 0),
                                @totalearning decimal(18, 2), @totaltrades int,
@successtrades int, @unsuccesstrades int,
                                @performance decimal(10, 2), @annualperformance
decimal(10, 2), @parameterDesc varchar(100),
                                                @parameter1 decimal(6, 2),
@parameter2 decimal(6, 2),@parameter3 decimal(6, 2)
AS
      DECLARE @theyear1 varchar(4), @themonth1 varchar(2), @thedate1 varchar(2),
@frmdateFrom varchar(10),
                    @theyear2 varchar(4), @themonth2 varchar(2), @thedate2
varchar(2), @frmdateTo varchar(10), @ncount int

    SET @theyear1  = SUBSTRING(@fromdateappliedStr, 1, 4);
    SET @themonth1 = SUBSTRING(@fromdateappliedStr, 5, 2);
    SET @thedate1  = SUBSTRING(@fromdateappliedStr, 7, 2);
    SET @frmdateFrom = @theyear1+'.'+@themonth1+'.'+@thedate1;

    SET @theyear2  = SUBSTRING(@todateappliedStr, 1, 4);
    SET @themonth2 = SUBSTRING(@todateappliedStr, 5, 2);
    SET @thedate2  = SUBSTRING(@todateappliedStr, 7, 2);
    SET @frmdateTo = @theyear2+'.'+@themonth2+'.'+@thedate2;

    SELECT @ncount = count(*) from AITrade where symbol = @symbol and codeInd =
@codeInd and typeoftrade = @typeoftrade;

    if @ncount = 0

            INSERT INTO AITrade ( symbol, codeInd, typeoftrade, fromdateapplied,
todateapplied, numberofstocks, feepertrade,
                                totalearning, totaltrades, successtrades,
unsuccesstrades, performance, annualperformance,
                                parameterDesc, parameter1, parameter2, parameter3)
            VALUES                          ( @symbol, @codeInd, @typeoftrade,
CONVERT(datetime, @frmdateFrom, 102), CONVERT(datetime, @frmdateTo, 102),
```

```
                                                @numberofstocks, @feepertrade,
@totalearning, @totaltrades, @successtrades, @unsuccesstrades, @performance,
                                 @annualperformance, @parameterDesc, @parameter1,
@parameter2, @parameter3)
     else

            UPDATE AITrade
            SET fromdateapplied = CONVERT(datetime, @frmdateFrom, 102),
todateapplied = CONVERT(datetime, @frmdateTo, 102), numberofstocks =
@numberofstocks,
                feepertrade = @feepertrade, totalearning = @totalearning,
totaltrades = @totaltrades, successtrades = @successtrades,
                  unsuccesstrades = @unsuccesstrades, performance =
@performance, annualperformance = @annualperformance, parameterDesc =
@parameterDesc,
                  parameter1 = @parameter1,  parameter2 = @parameter2,
parameter3 = @parameter3
            WHERE symbol = @symbol and  codeInd = @codeInd and typeoftrade =
@typeoftrade;
```

PROCEDURE: CreatePatternOfTwo	Source: generalViewsProc.sql

```
CREATE PROCEDURE CreatePatternOfTwo @symbol varchar(8), @dateStr varchar(8),
@patternStr varchar(30),
                        @patternBasic varchar(11),@patternAlt1
varchar(18),@patternAlt2 varchar(26),@pattern16xNumber varchar(8),
                        @percentChange1Value decimal(10,2), @percentChange2Value
decimal(10,2),@percentChange3Value decimal(10,2),
                        @percentChange4Value decimal(10,2), @percentChange5Value
decimal(10,2),@percentChange6Value decimal(10,2),
                        @percentChange7Value decimal(10,2), @percentChange8Value
decimal(10,2),@percentChange9Value decimal(10,2),
                        @percentChange10Value
decimal(10,2),@percentChange15Value decimal(10,2), @percentChange20Value
decimal(10,2)
AS
      DECLARE @theyear varchar(4), @themonth varchar(2), @thedate varchar(2),
@frmdate varchar(10)

    SET @theyear  = SUBSTRING(@dateStr, 1, 4);
    SET @themonth = SUBSTRING(@dateStr, 5, 2);
    SET @thedate  = SUBSTRING(@dateStr, 7, 2);
    SET @frmdate = @theyear+'.'+@themonth+'.'+@thedate;

      INSERT INTO pattern_of_two ( symbol, date_value, patternStr, patternBasic,
patternAlt1, patternAlt2,pattern16xNumber, percentChange1Value,
percentChange2Value,
                                        percentChange3Value,
percentChange4Value, percentChange5Value, percentChange6Value,
                                                        percentChange7Value,
```

```
percentChange8Value, percentChange9Value, percentChange10Value,

      percentChange15Value, percentChange20Value )
    VALUES (@symbol, CONVERT(datetime, @frmdate, 102), @patternStr,
@patternBasic, @patternAlt1, @patternAlt2, @pattern16xNumber,
@percentChange1Value, @percentChange2Value,
                                        @percentChange3Value,
@percentChange4Value, @percentChange5Value, @percentChange6Value,

      @percentChange7Value, @percentChange8Value, @percentChange9Value,
@percentChange10Value,

      @percentChange15Value, @percentChange20Value )
```

PROCEDURE: CreatePatternOfThree	Source: generalViewsProc.sql

```
CREATE PROCEDURE CreatePatternOfThree @symbol varchar(8), @dateStr varchar(8),
@patternStr varchar(52),
                        @patternBasic varchar(18),@patternAlt1
varchar(30),@patternAlt2 varchar(40),@pattern16xNumber varchar(14),
                        @percentChange1Value decimal(10,2), @percentChange2Value
decimal(10,2),@percentChange3Value decimal(10,2),
                        @percentChange4Value decimal(10,2), @percentChange5Value
decimal(10,2),@percentChange6Value decimal(10,2),
                        @percentChange7Value decimal(10,2), @percentChange8Value
decimal(10,2),@percentChange9Value decimal(10,2),
                        @percentChange10Value
decimal(10,2),@percentChange15Value decimal(10,2), @percentChange20Value
decimal(10,2)
AS
      DECLARE @theyear varchar(4), @themonth varchar(2), @thedate varchar(2),
@frmdate varchar(10)

    SET @theyear  = SUBSTRING(@dateStr, 1, 4);
    SET @themonth = SUBSTRING(@dateStr, 5, 2);
    SET @thedate  = SUBSTRING(@dateStr, 7, 2);
    SET @frmdate = @theyear+'.'+@themonth+'.'+@thedate;

      INSERT INTO pattern_of_three ( symbol, date_value, patternStr,
patternBasic, patternAlt1, patternAlt2,pattern16xNumber, percentChange1Value,
percentChange2Value,
                                        percentChange3Value,
percentChange4Value, percentChange5Value, percentChange6Value,
                                                        percentChange7Value,
percentChange8Value, percentChange9Value, percentChange10Value,

      percentChange15Value, percentChange20Value )
    VALUES (@symbol, CONVERT(datetime, @frmdate, 102),
@patternStr,@patternBasic, @patternAlt1, @patternAlt2, @pattern16xNumber,
```

```
@percentChange1Value, @percentChange2Value,
                                        @percentChange3Value,
@percentChange4Value, @percentChange5Value, @percentChange6Value,

      @percentChange7Value, @percentChange8Value, @percentChange9Value,
@percentChange10Value,

      @percentChange15Value, @percentChange20Value )
```

PROCEDURE: CreatePatternOfFour	Source: generalViewsProc.sql

```
CREATE PROCEDURE CreatePatternOfFour @symbol varchar(8), @dateStr varchar(8),
@patternStr varchar(72),
                        @patternBasic varchar(24),@patternAlt1
varchar(42),@patternAlt2 varchar(62),@pattern16xNumber varchar(18),
                        @percentChange1Value decimal(10,2), @percentChange2Value
decimal(10,2),@percentChange3Value decimal(10,2),
                        @percentChange4Value decimal(10,2), @percentChange5Value
decimal(10,2),@percentChange6Value decimal(10,2),
                        @percentChange7Value decimal(10,2), @percentChange8Value
decimal(10,2),@percentChange9Value decimal(10,2),
                        @percentChange10Value
decimal(10,2),@percentChange15Value decimal(10,2), @percentChange20Value
decimal(10,2)
AS
      DECLARE @theyear varchar(4), @themonth varchar(2), @thedate varchar(2),
@frmdate varchar(10)

    SET @theyear  = SUBSTRING(@dateStr, 1, 4);
    SET @themonth = SUBSTRING(@dateStr, 5, 2);
    SET @thedate  = SUBSTRING(@dateStr, 7, 2);
    SET @frmdate = @theyear+'.'+@themonth+'.'+@thedate;

      INSERT INTO pattern_of_four ( symbol, date_value, patternStr,
patternBasic, patternAlt1, patternAlt2,pattern16xNumber, percentChange1Value,
percentChange2Value,
                                        percentChange3Value,
percentChange4Value, percentChange5Value, percentChange6Value,
                                                        percentChange7Value,
percentChange8Value, percentChange9Value, percentChange10Value,

      percentChange15Value, percentChange20Value )
    VALUES (@symbol, CONVERT(datetime, @frmdate, 102),
@patternStr,@patternBasic, @patternAlt1, @patternAlt2, @pattern16xNumber,
@percentChange1Value, @percentChange2Value,
                                        @percentChange3Value,
@percentChange4Value, @percentChange5Value, @percentChange6Value,

      @percentChange7Value, @percentChange8Value, @percentChange9Value,
```

```
@percentChange10Value,

      @percentChange15Value, @percentChange20Value )
```

PROCEDURE: UpdatePtnOfTwo	Source: generalViewsProc.sql

```
CREATE PROCEDURE UpdatePtnOfTwo @symbol varchar(8), @dateStr varchar(8)
AS
      DECLARE @theyear varchar(4), @themonth varchar(2), @thedate varchar(2),
@frmdate varchar(10)

    SET @theyear  = SUBSTRING(@dateStr, 1, 4);
    SET @themonth = SUBSTRING(@dateStr, 5, 2);
    SET @thedate  = SUBSTRING(@dateStr, 7, 2);
    SET @frmdate = @theyear+'.'+@themonth+'.'+@thedate;

      UPDATE pattern_of_two
      SET patternBasic = SUBSTRING(patternStr, 1, 1)+SUBSTRING(patternStr, 7,
3)+SUBSTRING(patternStr, 11, 1)+ SUBSTRING(patternStr, 17, 3)+
         SUBSTRING(patternStr, 21, 1)+ SUBSTRING(patternStr, 27, 2),
            patternAlt1 =  SUBSTRING(patternStr, 1, 3)+SUBSTRING(patternStr, 7,
3)+SUBSTRING(patternStr, 11, 3)+ SUBSTRING(patternStr, 17, 3)+
         SUBSTRING(patternStr, 21, 1)+ SUBSTRING(patternStr, 24, 5),
            patternAlt2 = SUBSTRING(patternStr, 1, 4)+SUBSTRING(patternStr, 7,
4)+SUBSTRING(patternStr, 11, 4)+ SUBSTRING(patternStr, 17, 4)+
         SUBSTRING(patternStr, 21, 10)
      FROM pattern_of_two
      WHERE symbol = @symbol and  date_value = CONVERT(datetime, @frmdate, 102);
```

PROCEDURE: UpdatePtnOfThree	Source: generalViewsProc.sql

```
CREATE PROCEDURE UpdatePtnOfThree @symbol varchar(8), @dateStr varchar(8)
AS
      DECLARE @theyear varchar(4), @themonth varchar(2), @thedate varchar(2),
@frmdate varchar(10)

    SET @theyear  = SUBSTRING(@dateStr, 1, 4);
    SET @themonth = SUBSTRING(@dateStr, 5, 2);
    SET @thedate  = SUBSTRING(@dateStr, 7, 2);
    SET @frmdate = @theyear+'.'+@themonth+'.'+@thedate;

      UPDATE pattern_of_three
      SET patternBasic = SUBSTRING(patternStr, 1, 1)+SUBSTRING(patternStr, 7,
3)+SUBSTRING(patternStr, 11, 1)+ SUBSTRING(patternStr, 17, 3)+
```

```
         SUBSTRING(patternStr, 21, 1)+SUBSTRING(patternStr, 27,
3)+SUBSTRING(patternStr, 31, 1)+ SUBSTRING(patternStr, 37, 2)+
       SUBSTRING(patternStr, 41, 1)+ SUBSTRING(patternStr, 47, 2),
            patternAlt1 =  SUBSTRING(patternStr, 1, 3)+SUBSTRING(patternStr, 7,
3)+SUBSTRING(patternStr, 11, 3)+ SUBSTRING(patternStr, 17, 3)+
         SUBSTRING(patternStr, 21, 3)+ SUBSTRING(patternStr, 27,
3)+SUBSTRING(patternStr, 31, 1)+ SUBSTRING(patternStr, 34, 5)+
       SUBSTRING(patternStr, 41, 1)+ SUBSTRING(patternStr, 44, 5),
            patternAlt2 = SUBSTRING(patternStr, 1, 4)+SUBSTRING(patternStr, 7,
4)+SUBSTRING(patternStr, 11, 4)+ SUBSTRING(patternStr, 17, 4)+
         SUBSTRING(patternStr, 21, 4)+ SUBSTRING(patternStr, 27,
4)+SUBSTRING(patternStr, 31, 10)+SUBSTRING(patternStr, 41, 10)
      FROM pattern_of_three
      WHERE symbol = @symbol and  date_value = CONVERT(datetime, @frmdate, 102);
```

PROCEDURE: UpdatePtnOfFour	Source: generalViewsProc.sql

```
CREATE PROCEDURE UpdatePtnOfFour @symbol varchar(8), @dateStr varchar(8)
AS
      DECLARE @theyear varchar(4), @themonth varchar(2), @thedate varchar(2),
@frmdate varchar(10)

    SET @theyear  = SUBSTRING(@dateStr, 1, 4);
    SET @themonth = SUBSTRING(@dateStr, 5, 2);
    SET @thedate  = SUBSTRING(@dateStr, 7, 2);
    SET @frmdate = @theyear+'.'+@themonth+'.'+@thedate;

      UPDATE pattern_of_four
      SET patternBasic = SUBSTRING(patternStr, 1, 1)+SUBSTRING(patternStr, 7,
3)+SUBSTRING(patternStr, 11, 1)+ SUBSTRING(patternStr, 17, 3)+
         SUBSTRING(patternStr, 21, 1)+SUBSTRING(patternStr, 27,
3)+SUBSTRING(patternStr, 31, 1)+SUBSTRING(patternStr, 37, 3)+
       SUBSTRING(patternStr, 41, 1)+ SUBSTRING(patternStr, 47, 2)+
SUBSTRING(patternStr, 51, 1)+ SUBSTRING(patternStr, 57, 2)+
       SUBSTRING(patternStr, 61, 1)+ SUBSTRING(patternStr, 67, 1),
            patternAlt1 =  SUBSTRING(patternStr, 1, 3)+SUBSTRING(patternStr, 7,
3)+SUBSTRING(patternStr, 11, 3)+ SUBSTRING(patternStr, 17, 3)+
         SUBSTRING(patternStr, 21, 3)+ SUBSTRING(patternStr, 27,
3)+SUBSTRING(patternStr, 31, 3)+ SUBSTRING(patternStr, 37, 3)+
       SUBSTRING(patternStr, 41, 1)+ SUBSTRING(patternStr, 44,
5)+SUBSTRING(patternStr, 51, 1)+ SUBSTRING(patternStr, 54, 5)+
       SUBSTRING(patternStr, 61, 1)+ SUBSTRING(patternStr, 54, 5),
            patternAlt2 = SUBSTRING(patternStr, 1, 4)+SUBSTRING(patternStr, 7,
4)+SUBSTRING(patternStr, 11, 4)+ SUBSTRING(patternStr, 17, 4)+
         SUBSTRING(patternStr, 21, 4)+ SUBSTRING(patternStr, 27,
4)+SUBSTRING(patternStr, 31, 4)+ SUBSTRING(patternStr, 37, 4)+
       SUBSTRING(patternStr, 41, 10)+SUBSTRING(patternStr, 51,
10)+SUBSTRING(patternStr, 61, 10)
```

```
      FROM pattern_of_four
      WHERE symbol = @symbol and  date_value = CONVERT(datetime, @frmdate, 102);
Go
```

View : Ptn_of_two_basic	Source: generalViewsProc.sql

```
CREATE VIEW Ptn_of_two_basic
AS
Select symbol, date_value, percentChange1Value, percentChange2Value,
percentChange3Value, percentChange4Value, percentChange5Value,
         percentChange6Value, percentChange7Value, percentChange8Value,
percentChange9Value, percentChange10Value, percentChange15Value,
         percentChange20Value, SUBSTRING(patternStr, 1, 1)+SUBSTRING(patternStr,
7, 3)+SUBSTRING(patternStr, 11, 1)+ SUBSTRING(patternStr, 17, 3)+
         SUBSTRING(patternStr, 21, 1)+ SUBSTRING(patternStr, 27, 2) AS
basic_ptn_of2
FROM pattern_of_two
```

View : Ptn_of_two_alt1	Source: generalViewsProc.sql

```
CREATE VIEW Ptn_of_two_alt1
AS
Select symbol, date_value, percentChange1Value, percentChange2Value,
percentChange3Value, percentChange4Value, percentChange5Value,
         percentChange6Value, percentChange7Value, percentChange8Value,
percentChange9Value, percentChange10Value, percentChange15Value,
         percentChange20Value, SUBSTRING(patternStr, 1, 3)+SUBSTRING(patternStr,
7, 3)+SUBSTRING(patternStr, 11, 3)+ SUBSTRING(patternStr, 17, 3)+
         SUBSTRING(patternStr, 21, 1)+ SUBSTRING(patternStr, 24, 5) AS
alt1_ptn_of2
FROM pattern_of_two
```

View : Ptn_of_two_alt2	Source: generalViewsProc.sql

```
CREATE VIEW Ptn_of_two_alt2
AS
Select symbol, date_value, percentChange1Value, percentChange2Value,
percentChange3Value, percentChange4Value, percentChange5Value,
         percentChange6Value, percentChange7Value, percentChange8Value,
percentChange9Value, percentChange10Value, percentChange15Value,
```

```
        percentChange20Value, SUBSTRING(patternStr, 1, 4)+SUBSTRING(patternStr,
7, 4)+SUBSTRING(patternStr, 11, 4)+ SUBSTRING(patternStr, 17, 4)+
        SUBSTRING(patternStr, 21, 10) AS alt2_ptn_of2
FROM pattern_of_two
```

View : Ptn_of_three_basic	Source: generalViewsProc.sql

```
CREATE VIEW Ptn_of_three_basic
AS
Select symbol, date_value, percentChange1Value, percentChange2Value,
percentChange3Value, percentChange4Value, percentChange5Value,
        percentChange6Value, percentChange7Value, percentChange8Value,
percentChange9Value, percentChange10Value, percentChange15Value,
        percentChange20Value, SUBSTRING(patternStr, 1, 1)+SUBSTRING(patternStr,
7, 3)+SUBSTRING(patternStr, 11, 1)+ SUBSTRING(patternStr, 17, 3)+
        SUBSTRING(patternStr, 21, 1)+SUBSTRING(patternStr, 27,
3)+SUBSTRING(patternStr, 31, 1)+ SUBSTRING(patternStr, 37, 2)+
      SUBSTRING(patternStr, 41, 1)+ SUBSTRING(patternStr, 47, 2) AS
basic_ptn_of3
FROM pattern_of_three
```

View : Ptn_of_three_alt1	Source: generalViewsProc.sql

```
CREATE VIEW Ptn_of_three_alt1
AS
Select symbol, date_value, percentChange1Value, percentChange2Value,
percentChange3Value, percentChange4Value, percentChange5Value,
        percentChange6Value, percentChange7Value, percentChange8Value,
percentChange9Value, percentChange10Value, percentChange15Value,
        percentChange20Value, SUBSTRING(patternStr, 1, 3)+SUBSTRING(patternStr,
7, 3)+SUBSTRING(patternStr, 11, 3)+ SUBSTRING(patternStr, 17, 3)+
        SUBSTRING(patternStr, 21, 3)+ SUBSTRING(patternStr, 27,
3)+SUBSTRING(patternStr, 31, 1)+ SUBSTRING(patternStr, 34, 5)+
      SUBSTRING(patternStr, 41, 1)+ SUBSTRING(patternStr, 44, 5) AS alt1_ptnof3
FROM pattern_of_three
```

View : Ptn_of_three_alt2	Source: generalViewsProc.sql

```
CREATE VIEW Ptn_of_three_alt2
AS
Select symbol, date_value, percentChange1Value, percentChange2Value,
percentChange3Value, percentChange4Value, percentChange5Value,
         percentChange6Value, percentChange7Value, percentChange8Value,
percentChange9Value, percentChange10Value, percentChange15Value,
         percentChange20Value, SUBSTRING(patternStr, 1, 4)+SUBSTRING(patternStr,
7, 4)+SUBSTRING(patternStr, 11, 4)+ SUBSTRING(patternStr, 17, 4)+
         SUBSTRING(patternStr, 21, 4)+ SUBSTRING(patternStr, 27,
4)+SUBSTRING(patternStr, 31, 10)+SUBSTRING(patternStr, 41, 10) AS alt2_ptn_of3
FROM pattern_of_three
```

View : Ptn_of_four_basic	Source: generalViewsProc.sql

```
CREATE VIEW Ptn_of_four_basic
AS
Select symbol, date_value, percentChange1Value, percentChange2Value,
percentChange3Value, percentChange4Value, percentChange5Value,
         percentChange6Value, percentChange7Value, percentChange8Value,
percentChange9Value, percentChange10Value, percentChange15Value,
         percentChange20Value, SUBSTRING(patternStr, 1, 1)+SUBSTRING(patternStr,
7, 3)+SUBSTRING(patternStr, 11, 1)+ SUBSTRING(patternStr, 17, 3)+
         SUBSTRING(patternStr, 21, 1)+SUBSTRING(patternStr, 27,
3)+SUBSTRING(patternStr, 31, 1)+SUBSTRING(patternStr, 37, 3)+
       SUBSTRING(patternStr, 41, 1)+ SUBSTRING(patternStr, 47, 2)+
SUBSTRING(patternStr, 51, 1)+ SUBSTRING(patternStr, 57, 2)+
       SUBSTRING(patternStr, 61, 1)+ SUBSTRING(patternStr, 67, 2) AS
basic_ptn_of4
FROM pattern_of_three
```

View :	Source: generalViewsProc.sql

```
CREATE VIEW Ptn_of_four_alt1
AS
Select symbol, date_value, percentChange1Value, percentChange2Value,
percentChange3Value, percentChange4Value, percentChange5Value,
         percentChange6Value, percentChange7Value, percentChange8Value,
percentChange9Value, percentChange10Value, percentChange15Value,
         percentChange20Value, SUBSTRING(patternStr, 1, 3)+SUBSTRING(patternStr,
```

```
7, 3)+SUBSTRING(patternStr, 11, 3)+ SUBSTRING(patternStr, 17, 3)+
         SUBSTRING(patternStr, 21, 3)+ SUBSTRING(patternStr, 27,
3)+SUBSTRING(patternStr, 31, 3)+ SUBSTRING(patternStr, 37, 3)+
       SUBSTRING(patternStr, 41, 1)+ SUBSTRING(patternStr, 44,
5)+SUBSTRING(patternStr, 51, 1)+ SUBSTRING(patternStr, 54, 5)+
       SUBSTRING(patternStr, 61, 1)+ SUBSTRING(patternStr, 54, 5) AS
alt1_ptn_of4
FROM pattern_of_four
```

View : `Ptn_of_four_alt2`	Source: generalViewsProc.sql

```
CREATE VIEW Ptn_of_four_alt2
AS
Select symbol, date_value, percentChange1Value, percentChange2Value,
percentChange3Value, percentChange4Value, percentChange5Value,
         percentChange6Value, percentChange7Value, percentChange8Value,
percentChange9Value, percentChange10Value, percentChange15Value,
         percentChange20Value, SUBSTRING(patternStr, 1, 4)+SUBSTRING(patternStr,
7, 4)+SUBSTRING(patternStr, 11, 4)+ SUBSTRING(patternStr, 17, 4)+
         SUBSTRING(patternStr, 21, 4)+ SUBSTRING(patternStr, 27,
4)+SUBSTRING(patternStr, 31, 4)+ SUBSTRING(patternStr, 37, 4)+
       SUBSTRING(patternStr, 41, 10)+SUBSTRING(patternStr, 51,
10)+SUBSTRING(patternStr, 61, 10) AS alt2_ptn_of4
FROM pattern_of_four
```

Appendix D: All views (sql scripts) used in patterns

View : showTrendPtn2_all	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn2_all
AS
select count(a.patternStr) as number, a.patternStr,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange1Value>0 ) as upNext1,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange1Value<0 ) as downNext1,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange3Value>0 ) as upNext3,
(select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange3Value<0 ) as downNext3,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange5Value>0 ) as upNext5,
(select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange5Value<0 ) as downNext5,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange7Value>0 ) as upNext7,
(select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange7Value<0 ) as downNext7,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange10Value>0 ) as upNext10,
(select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange10Value<0 ) as downNext10,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange15Value>0 ) as upNext15,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange15Value<0 ) as downNext15
from pattern_of_two a  GROUP BY a.patternStr
```

View : showTrendPtn2_Pall	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn2_Pall
AS
select      number, patternStr,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn2_all
```

View : showTrendPtn2_basic	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn2_basic
AS
select count(a.basic_ptn_of2) as number, a.basic_ptn_of2,

 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange1Value>0 ) as upNext1,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange1Value<0 ) as downNext1,
```

```
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange3Value>0 ) as upNext3,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange3Value<0 ) as downNext3,

 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange5Value>0 ) as upNext5,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange5Value<0 ) as downNext5,

 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange7Value>0 ) as upNext7,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange7Value<0 ) as downNext7,

 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange10Value>0 ) as upNext10,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange10Value<0 ) as downNext10,

 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange15Value>0 ) as upNext15,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.basic_ptn_of2) from Ptn_of_two_basic b where  a.basic_ptn_of2 =
b.basic_ptn_of2 and b.percentChange15Value<0 ) as downNext15

from Ptn_of_two_basic a  GROUP BY a.basic_ptn_of2
```

View : showTrendPtn2_Pbasic	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn2_Pbasic
AS
select      number, basic_ptn_of2,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
```

```
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn2_basic
```

View : showTrendPtn2_alt1	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn2_alt1
AS
select count(a.alt1_ptn_of2) as number, a.alt1_ptn_of2,

 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange1Value>0 ) as upNext1,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange1Value<0 ) as downNext1,

 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange3Value>0 ) as upNext3,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange3Value<0 ) as downNext3,

 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange5Value>0 ) as upNext5,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange5Value<0 ) as downNext5,

 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
```

```
b.alt1_ptn_of2 and b.percentChange7Value>0 ) as upNext7,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange7Value<0 ) as downNext7,

 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange10Value>0 ) as upNext10,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange10Value<0 ) as downNext10,

 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange15Value>0 ) as upNext15,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.alt1_ptn_of2) from Ptn_of_two_alt1 b where  a.alt1_ptn_of2 =
b.alt1_ptn_of2 and b.percentChange15Value<0 ) as downNext15

from Ptn_of_two_alt1 a  GROUP BY a.alt1_ptn_of2
```

View : showTrendPtn2_Palt1	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn2_Palt1
AS
select      number, alt1_ptn_of2,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
```

```
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn2_alt1
```

View : showTrendPtn2_alt2	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn2_alt2
AS
select count(a.alt2_ptn_of2) as number, a.alt2_ptn_of2,

 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange1Value>0 ) as upNext1,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange1Value<0 ) as downNext1,

 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange3Value>0 ) as upNext3,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange3Value<0 ) as downNext3,

 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange5Value>0 ) as upNext5,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange5Value<0 ) as downNext5,

 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange7Value>0 ) as upNext7,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange7Value<0 ) as downNext7,

 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange10Value>0 ) as upNext10,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange10Value<0 ) as downNext10,

 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange15Value>0 ) as upNext15,
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt1_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange15Value=0 ) as equalNext15,
```

```
 (select count(b.alt2_ptn_of2) from Ptn_of_two_alt2 b where  a.alt2_ptn_of2 =
b.alt2_ptn_of2 and b.percentChange15Value<0 ) as downNext15

from Ptn_of_two_alt2 a  GROUP BY a.alt2_ptn_of2
```

View : showTrendPtn2_Palt2	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn2_Palt2
AS
select      number, alt2_ptn_of2,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn2_alt2
```

View : showTrendPtn3_all	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn3_all
AS
select count(a.patternStr) as number, a.patternStr,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
```

```
b.patternStr and b.percentChange1Value>0 ) as upNext1,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange1Value<0 ) as downNext1,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange3Value>0 ) as upNext3,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange3Value<0 ) as downNext3,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange5Value>0 ) as upNext5,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange5Value<0 ) as downNext5,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange7Value>0 ) as upNext7,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange7Value<0 ) as downNext7,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange10Value>0 ) as upNext10,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange10Value<0 ) as downNext10,

 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange15Value>0 ) as upNext15,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.patternStr) from pattern_of_two b where  a.patternStr =
b.patternStr and b.percentChange15Value<0 ) as downNext15

from pattern_of_three a  GROUP BY a.patternStr
```

View : showTrendPtn3_Pall	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn3_Pall
AS
select      number, patternStr,
```

```
            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn3_all
```

View :	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn3_basic
AS
select count(a.basic_ptn_of3) as number, a.basic_ptn_of3,

 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange1Value>0 ) as upNext1,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange1Value<0 ) as downNext1,

 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange3Value>0 ) as upNext3,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange3Value<0 ) as downNext3,

 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange5Value>0 ) as upNext5,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
```

```
= b.basic_ptn_of3 and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange5Value<0 ) as downNext5,

 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange7Value>0 ) as upNext7,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange7Value<0 ) as downNext7,

 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange10Value>0 ) as upNext10,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange10Value<0 ) as downNext10,

 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange15Value>0 ) as upNext15,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.basic_ptn_of3) from Ptn_of_three_basic b where  a.basic_ptn_of3
= b.basic_ptn_of3 and b.percentChange15Value<0 ) as downNext15

from Ptn_of_three_basic a  GROUP BY a.basic_ptn_of3
```

View : showTrendPtn3_Pbasic	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn3_Pbasic
AS
select      number, basic_ptn_of3,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,
```

```
            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn3_basic
```

View : showTrendPtn3_alt1	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn3_alt1
AS
select count(a.alt1_ptn_of3) as number, a.alt1_ptn_of3,

 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange1Value>0 ) as upNext1,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange1Value<0 ) as downNext1,

 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange3Value>0 ) as upNext3,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange3Value<0 ) as downNext3,

 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange5Value>0 ) as upNext5,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange5Value<0 ) as downNext5,

 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange7Value>0 ) as upNext7,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange7Value<0 ) as downNext7,

 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange10Value>0 ) as upNext10,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
```

```
b.alt1_ptn_of3 and b.percentChange10Value<0 ) as downNext10,

 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange15Value>0 ) as upNext15,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.alt1_ptn_of3) from Ptn_of_three_alt1 b where  a.alt1_ptn_of3 =
b.alt1_ptn_of3 and b.percentChange15Value<0 ) as downNext15

from Ptn_of_three_alt1 a  GROUP BY a.alt1_ptn_of3
```

View : showTrendPtn3_Palt1	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn3_Palt1
AS
select      number, alt1_ptn_of3,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn3_alt1
```

View : showTrendPtn3_alt2	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn3_alt2
AS
select count(a.alt2_ptn_of3) as number, a.alt2_ptn_of3,

 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange1Value>0 ) as upNext1,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange1Value<0 ) as downNext1,

 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange3Value>0 ) as upNext3,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange3Value<0 ) as downNext3,

 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange5Value>0 ) as upNext5,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange5Value<0 ) as downNext5,

 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange7Value>0 ) as upNext7,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange7Value<0 ) as downNext7,

 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange10Value>0 ) as upNext10,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange10Value<0 ) as downNext10,

 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange15Value>0 ) as upNext15,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt1_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.alt2_ptn_of3) from Ptn_of_three_alt2 b where  a.alt2_ptn_of3 =
b.alt2_ptn_of3 and b.percentChange15Value<0 ) as downNext15

from Ptn_of_three_alt2 a  GROUP BY a.alt2_ptn_of3
```

View : showTrendPtn3_Palt2	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn3_Palt2
AS
select      number, alt2_ptn_of3,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn3_alt2
```

View : showTrendPtn4_all	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn4_all
AS
select count(a.patternStr) as number, a.patternStr,

 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange1Value>0 ) as upNext1,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange1Value<0 ) as downNext1,

 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
```

```
b.patternStr and b.percentChange3Value>0 ) as upNext3,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange3Value<0 ) as downNext3,

 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange5Value>0 ) as upNext5,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange5Value<0 ) as downNext5,

 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange7Value>0 ) as upNext7,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange7Value<0 ) as downNext7,

 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange10Value>0 ) as upNext10,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange10Value<0 ) as downNext10,

 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange15Value>0 ) as upNext15,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.patternStr) from pattern_of_four b where  a.patternStr =
b.patternStr and b.percentChange15Value<0 ) as downNext15

from pattern_of_four a  GROUP BY a.patternStr
go
```

View : showTrendPtn4_Pall	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn4_Pall
AS
select      number, patternStr,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,
```

```
            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn4_all
```

View : showTrendPtn4_basic	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn4_basic
AS
select count(a.basic_ptn_of4) as number, a.basic_ptn_of4,

 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange1Value>0 ) as upNext1,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange1Value<0 ) as downNext1,

 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange3Value>0 ) as upNext3,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange3Value<0 ) as downNext3,

 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange5Value>0 ) as upNext5,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange5Value<0 ) as downNext5,

 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange7Value>0 ) as upNext7,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange7Value=0 ) as equalNext7,
```

```
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange7Value<0 ) as downNext7,

 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange10Value>0 ) as upNext10,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange10Value<0 ) as downNext10,

 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange15Value>0 ) as upNext15,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.basic_ptn_of4) from Ptn_of_four_basic b where  a.basic_ptn_of4
= b.basic_ptn_of4 and b.percentChange15Value<0 ) as downNext15

from Ptn_of_four_basic a  GROUP BY a.basic_ptn_of4
```

View : showTrendPtn4_Pbasic	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn4_Pbasic
AS
select      number, basic_ptn_of4,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn4_basic
```

View :showTrendPtn4_Pbasic	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn4_Pbasic
AS
select      number, basic_ptn_of4,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn4_basic
```

View : showTrendPtn4_alt1	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn4_alt1
AS
select count(a.alt1_ptn_of4) as number, a.alt1_ptn_of4,

 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange1Value>0 ) as upNext1,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange1Value<0 ) as downNext1,
```

```
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange3Value>0 ) as upNext3,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange3Value<0 ) as downNext3,

 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange5Value>0 ) as upNext5,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange5Value<0 ) as downNext5,

 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange7Value>0 ) as upNext7,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange7Value<0 ) as downNext7,

 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange10Value>0 ) as upNext10,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange10Value<0 ) as downNext10,

 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange15Value>0 ) as upNext15,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.alt1_ptn_of4) from Ptn_of_four_alt1 b where  a.alt1_ptn_of4 =
b.alt1_ptn_of4 and b.percentChange15Value<0 ) as downNext15

from Ptn_of_four_alt1 a  GROUP BY a.alt1_ptn_of4
```

View : showTrendPtn4_Palt1	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn4_Palt1
AS
select      number, alt1_ptn_of4,

            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,
```

```
            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P

from showTrendPtn4_alt1
```

View : showTrendPtn4_alt2	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn4_alt2
AS
select count(a.alt2_ptn_of3) as number, a.alt2_ptn_of4,

 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange1Value>0 ) as upNext1,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange1Value=0 ) as equalNext1,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange1Value<0 ) as downNext1,

 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange3Value>0 ) as upNext3,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange3Value=0 ) as equalNext3,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange3Value<0 ) as downNext3,

 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange5Value>0 ) as upNext5,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange5Value=0 ) as equalNext5,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange5Value<0 ) as downNext5,

 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange7Value>0 ) as upNext7,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
```

```
b.alt2_ptn_of4 and b.percentChange7Value=0 ) as equalNext7,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange7Value<0 ) as downNext7,

 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange10Value>0 ) as upNext10,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange10Value=0 ) as equalNext10,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange10Value<0 ) as downNext10,

 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange15Value>0 ) as upNext15,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange15Value=0 ) as equalNext15,
 (select count(b.alt2_ptn_of4) from Ptn_of_four_alt2 b where  a.alt2_ptn_of4 =
b.alt2_ptn_of4 and b.percentChange15Value<0 ) as downNext15

from Ptn_of_four_alt2 a  GROUP BY a.alt2_ptn_of4
```

View : showTrendPtn4_Palt2	Source: generalViewsProc.sql

```
CREATE VIEW showTrendPtn4_Palt2
AS
select      number, alt2_ptn_of4,
            cast(100.0*upNext1/number AS decimal(6,2)) as upNext1P,
            cast(100.0*equalNext1/number AS decimal(6,2)) as equalNext1P,
            cast(100.0*downNext1/number AS decimal(6,2))as downNext1P,

            cast(100.0*upNext3/number AS decimal(6,2)) as upNext3P,
            cast(100.0*equalNext3/number AS decimal(6,2)) as equalNext3P,
            cast(100.0*downNext3/number AS decimal(6,2))as downNext3P,

            cast(100.0*upNext5/number AS decimal(6,2)) as upNext5P,
            cast(100.0*equalNext5/number AS decimal(6,2)) as equalNext5P,
            cast(100.0*downNext5/number AS decimal(6,2))as downNext5P,

            cast(100.0*upNext7/number AS decimal(6,2)) as upNext7P,
            cast(100.0*equalNext7/number AS decimal(6,2)) as equalNext7P,
            cast(100.0*downNext7/number AS decimal(6,2))as downNext7P,

            cast(100.0*upNext10/number AS decimal(6,2)) as upNext10P,
            cast(100.0*equalNext10/number AS decimal(6,2)) as equalNext10P,
            cast(100.0*downNext10/number AS decimal(6,2))as downNext10P,

            cast(100.0*upNext15/number AS decimal(6,2)) as upNext15P,
            cast(100.0*equalNext15/number AS decimal(6,2)) as equalNext15P,
            cast(100.0*downNext15/number AS decimal(6,2))as downNext15P
from showTrendPtn4_alt2
```

Appendix E: Project Source Code (Which has developed)

Package (namespace) : `BasicOper`	**Source: Date.cs**

```
/// <summary>
/// Summary: description for Date class
/// ---------------------------------------------------------------
/// It contains the Date class
/// ---------------------------------------------------------------
/// Purpose: It is used for representing the date object used in stocks
/// </summary>
/// ---------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;

namespace AIPredictor.BasicOper
{
    /// <summary>
    /// Date class definition
    /// It includes only date values (not time)
    /// </summary>
    public class Date : Object
    {
        private int year;   // max year 4 digits
        private int month;  // 1-12
        private int day;    // 1-31 based on month

        /// <summary>
        /// constructor confirms proper value for month;
        /// CheckDay to confirm proper value for day.
        /// and year
        /// </summary>
        /// <param name="theYear"></param>
        /// <param name="theMonth"></param>
        /// <param name="theDay"></param>

        /// <param name="lChecking"></param>
        public Date(int theYear, int theMonth, int theDay, bool lChecking)
        {
            year = theYear;
            month = theMonth;
            day = theDay;

            // the checking is performed only if lChecking is true
            if (lChecking)
            {
```

```
                if (!ValidateMonth(theMonth))
                    throw new Exception("Invalid Month!. It must be between 1
and 12.");

                if (!ValidateYear(theYear))
                    throw new Exception("Invalid Year!. It must be between 1 and
9999! ");

                if (!ValidateDayRange(theDay, theMonth))
                    throw new Exception("Invalid Day!" + "Month=" +
theMonth.ToString() + " Date=" + theDay.ToString() +
                        " It must be between 1 and 31 depending on month");

                if (!ValidateDayLeapYear(theDay, theMonth, theYear))
                    throw new Exception("Invalid Day " + theDay.ToString() + "
Month=" + theMonth.ToString() +
                        "for this year because is leap! for year:" +
year.ToString());
            }

        } // end Date constructor

        /// <summary>
        /// Constructor without checking
        /// </summary>
        /// <param name="theYear"></param>
        /// <param name="theMonth"></param>
        /// <param name="theDay"></param>

        public Date(int theYear, int theMonth, int theDay )
        {
            month = theMonth;
            day   = theDay;
            year  = theYear;

        } // end Date constructor

        /// <summary>
        /// Constructor with no value input
        /// </summary>
        public Date()
        {
            month = 1;
            day = 1;
            year = 1900;

        } // end Date constructor

        /// <summary>
        /// Validate the date value
        /// </summary>
        /// <returns></returns>
        public bool IsValidDate()
```

```
        {
            return ValidateMonth(this.month) && ValidateYear(this.year) &&
                   ValidateDayRange(this.month, this.day) &&
ValidateDayLeapYear(this.year, this.month, this.day);
        }

        /// <summary>
        /// Validate the month value
        /// </summary>
        /// <param name="inputMonth"></param>
        /// <returns></returns>
        private bool ValidateMonth(int inputMonth)
        {
            bool lflag = false;
            // validate month
            if (inputMonth > 0 && inputMonth <= 12)
                lflag = true;

            return lflag;
        }

        /// <summary>
        /// utility method confirms proper day value
        /// based on month and year
        /// </summary>
        /// <param name="theMonth"></param>
        /// <param name="theDay"></param>
        /// <returns></returns>
        private bool ValidateDayRange(int theMonth, int theDay)
        {
            int[] daysPerMonth = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
30, 31 };

            // check if day in range for month
            if (theDay < 1 || theDay > daysPerMonth[month])
            {
                return false;
            }
            return true;
        }

        /// <summary>
        /// Validate for leap year
        /// </summary>
        /// <param name="theDay"></param>
        /// <param name="theMonth"></param>
        /// <param name="theYear"></param>
        /// <returns></returns>
        private bool ValidateDayLeapYear(int theYear, int theMonth, int theDay)
        {
            // check for leap year
            if (theMonth == 2 && theDay == 29 && (theYear % 400 == 0 || (theYear
% 4 == 0 && theYear % 100 != 0)))
```

```
            {
                return false;
            }
            return true;
        }

        /// <summary>
        /// Validate for the year. It must be between 1 and 9999 value
        /// </summary>
        /// <param name="inputYear"></param>
        /// <returns></returns>
        private bool ValidateYear(int inputYear)
        {
            if ((inputYear < 0) || (inputYear > 9999))
            {
                return false;
            }
            return true;
        }

        // return date string as month/day/year
        /// <summary>
        /// It converts date to sting. The format is
        /// like 01/03/2005
        /// </summary>
        /// <returns></returns>
        public string ToNormalDateString()
        {
            string strMonth = month.ToString().Trim();
            string strDay = day.ToString().Trim();

            if (month < 10)
                strMonth = "0" + strMonth;

            if (day < 10)
                strDay = "0" + strDay;

            return strDay + "/" + strMonth + "/" + year.ToString();
        }

        // return date string as month/day/year
        /// <summary>
        /// It converts date to sting. The format is
        /// like 20051128
        /// </summary>
        /// <returns></returns>
        public string ToDateString()
        {
            string strMonth = month.ToString().Trim();
            string strDay = day.ToString().Trim();

            if (month < 10)
                strMonth = "0" + strMonth;
```

```
            if (day < 10)
                strDay = "0" + strDay;

            return year.ToString() + strMonth + strDay;
        }
        /// <summary>
        /// Converts the Date object to string
        /// </summary>
        /// <param name="strDate"></param>
        /// <returns></returns>
        public static Date ConvetStringToDate(string strDate)
        {
            if (strDate.Length == 8)

                return new Date(Int32.Parse(strDate.Substring(0, 4)),
Int32.Parse(strDate.Substring(4, 2)),
                    Int32.Parse(strDate.Substring(6, 2)));
            else
                return new Date();
        }

        public int LengthDate()
        {
            return 365 * year + 30 * month + day;
        }

    }  // end class Date
}
```

Package (namespace) : BasicOper	**Source:** CandlePattern.cs

```
/// <summary>
/// Summary: description for CandlePattern class
/// -----------------------------------------------------------------
/// It contains the form class
/// -----------------------------------------------------------------
/// Purpose: It is used for creation of gvol-candlestick patterns
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;

namespace AIPredictor.BasicOper
{
    class CandlePattern
    {
        // private attributes
        private Date date_value;
        private string patternStr;
        private string patternBasic;
        private string patternAlt1;
        private string patternAlt2;
        private float[] percentChangeArray;

        /// <summary>
        /// Basic constractor method
        /// </summary>
        /// <param name="dateStr"></param>
        /// <param name="patternValue"></param>
        /// <param name="patternBasicValue"></param>
        /// <param name="patternAlt1Value"></param>
        /// <param name="patternAlt2Value"></param>
        /// <param name="percentChange1Value"></param>
        /// <param name="percentChange2Value"></param>
        /// <param name="percentChange3Value"></param>
        /// <param name="percentChange4Value"></param>
        /// <param name="percentChange5Value"></param>
        /// <param name="percentChange6Value"></param>
        /// <param name="percentChange7Value"></param>
        /// <param name="percentChange8Value"></param>
        /// <param name="percentChange9Value"></param>
        /// <param name="percentChange10Value"></param>
        /// <param name="percentChange15Value"></param>
        /// <param name="percentChange20Value"></param>
        public CandlePattern(string dateStr, string patternValue, string
patternBasicValue, string patternAlt1Value, string patternAlt2Value,
                                    float percentChange1Value, float
```

```
percentChange2Value,float percentChange3Value,
                                float percentChange4Value, float
percentChange5Value, float percentChange6Value, float percentChange7Value,
                                float percentChange8Value, float
percentChange9Value, float percentChange10Value, float percentChange15Value,
float percentChange20Value)
        {
            this.setDate(dateStr);
            this.patternStr            = patternValue;
            this.patternBasic = patternBasicValue;
            this.patternAlt1 = patternAlt1Value;
            this.patternAlt2 = patternAlt2Value;
            this.percentChangeArray = new float[12];

            this.percentChangeArray[0] = percentChange1Value;
            this.percentChangeArray[1] = percentChange2Value;
            this.percentChangeArray[2] = percentChange3Value;
            this.percentChangeArray[3] = percentChange4Value;
            this.percentChangeArray[4] = percentChange5Value;
            this.percentChangeArray[5] = percentChange6Value;
            this.percentChangeArray[6] = percentChange7Value;
            this.percentChangeArray[7] = percentChange8Value;
            this.percentChangeArray[8] = percentChange9Value;
            this.percentChangeArray[9] = percentChange10Value;
            this.percentChangeArray[10] = percentChange15Value;
            this.percentChangeArray[11] = percentChange20Value;
        }

        /// <summary>
        /// Another constractor method
        /// </summary>
        /// <param name="dateStr"></param>
        /// <param name="patternValue"></param>
        /// <param name="patternBasicValue"></param>
        /// <param name="patternAlt1Value"></param>
        /// <param name="patternAlt2Value"></param>
        /// <param name="percentChangeInput"></param>
        public CandlePattern(string dateStr, string patternValue, string
patternBasicValue, string patternAlt1Value, string patternAlt2Value, float[]
percentChangeInput)
        {
            this.setDate(dateStr);
            this.patternStr = patternValue;
            this.patternBasic = patternBasicValue;
            this.patternAlt1 = patternAlt1Value;
            this.patternAlt2 = patternAlt2Value;
            this.percentChangeArray = new float[12];
            for (int i = 0; i < 12; i++)
                percentChangeArray[i] = percentChangeInput[i];
        }

        /// <summary>
        /// Contractor without parameters
```

```
        /// </summary>
        public CandlePattern()
        {
            this.setDate("19000101");
            this.patternStr   = "";
            this.patternBasic = "";
            this.patternAlt1 = "";
            this.patternAlt2 = "";
            this.percentChangeArray = new float[12];

            for (int i = 0; i < 12; i++)
                percentChangeArray[i] = 0;
        }

        // public get Array of percent values
        public float[] getArrayOfChanges()
        {
            return percentChangeArray;
        }

        // public set Array of percent values
        public void setArrayOfChanges(float[] percentChangeInput)
        {
            for (int i = 0; i < 12; i++)
                percentChangeArray[i] = percentChangeInput[i];
        }

        // public property Date_value
        public string Date_value
        {
            get
            {
                return date_value.ToDateString();
            }
        }

        public void setDate(string strDate)
        {
            this.date_value = Date.ConvetStringToDate(strDate);
        }

        // public property PatternStr
        public string PatternStr
        {
            get
            {
                return patternStr;
            }
            set
            {
                patternStr = value;
            }
        }
```

```
        // public property PatternStr
        public string PatternBasic
        {
            get
            {
                return patternBasic;
            }
            set
            {
                patternBasic = value;
            }
        }

        // public property PatternAlt1
        public string PatternAlt1
        {
            get
            {
                return patternAlt1;
            }
            set
            {
                patternAlt1 = value;
            }
        }

        // public property PatternAlt2
        public string PatternAlt2
        {
            get
            {
                return patternAlt2;
            }
            set
            {
                patternAlt2 = value;
            }
        }

        // public property PercentChange1th
        public float PercentChange1th
        {
            get
            {
                return percentChangeArray[0];
            }
            set
            {
                percentChangeArray[0] = value;
            }
        }
```

```
        // public property PercentChange2th
        public float PercentChange2th
        {
            get
            {
                return percentChangeArray[1];
            }
            set
            {
                percentChangeArray[1] = value;
            }
        }
        // public property PercentChange3th
        public float PercentChange3th
        {
            get
            {
                return percentChangeArray[2];
            }
            set
            {
                percentChangeArray[2] = value;
            }
        }

        // public property PercentChange4th
        public float PercentChange4th
        {
            get
            {
                return percentChangeArray[3];
            }
            set
            {
                percentChangeArray[3] = value;
            }
        }

        // public property PercentChange5th
        public float PercentChange5th
        {
            get
            {
                return percentChangeArray[4];
            }
            set
            {
                percentChangeArray[4] = value;
            }
        }

        // public property PercentChange6th
        public float PercentChange6th
```

```
        {
            get
            {
                return percentChangeArray[5];
            }
            set
            {
                percentChangeArray[5] = value;
            }
        }

        // public property PercentChange7th
        public float PercentChange7th
        {
            get
            {
                return percentChangeArray[6];
            }
            set
            {
                percentChangeArray[6] = value;
            }
        }

        // public property PercentChange8th
        public float PercentChange8th
        {
            get
            {
                return percentChangeArray[7];
            }
            set
            {
                percentChangeArray[7] = value;
            }
        }


        // public property PercentChange9th
        public float PercentChange9th
        {
            get
            {
                return percentChangeArray[8];
            }
            set
            {
                percentChangeArray[8] = value;
            }
        }

        // public property PercentChange10th
        public float PercentChange10th
```

```
        {
            get
            {
                return percentChangeArray[9];
            }
            set
            {
                percentChangeArray[9] = value;
            }
        }

        // public property PercentChange15th
        public float PercentChange15th
        {
            get
            {
                return percentChangeArray[10];
            }
            set
            {
                percentChangeArray[10] = value;
            }
        }

        // public property PercentChange20th
        public float PercentChange20th
        {
            get
            {
                return percentChangeArray[11];
            }
            set
            {
                percentChangeArray[11] = value;
            }
        }

        public string Pattern16xNumber
        {
            get
            {
                return ConvertTo16XNumber();
            }
        }


        private string ConvertTo16XNumber()
        {
            int modOfDivision = this.PatternStr.Length % 4;
            string patternNormal = "";

            if (modOfDivision == 0)
                patternNormal = this.PatternStr;
```

```
            else if (modOfDivision == 3)
                patternNormal = "0" + this.PatternStr;
            else if (modOfDivision == 2)
                patternNormal = "00" + this.PatternStr;
            else
                patternNormal = "000" + this.PatternStr;

            string tmpPart = "";
            string retValue = "";

            for (int i = 0; i < patternNormal.Length; i++)
            {
                if (((i + 1) % 4 == 0) && i > 0)
                {
                    tmpPart = tmpPart + patternNormal[i];
                    retValue = retValue + Convert4BitTo16X(tmpPart);
                    tmpPart = "";
                }
                else
                {
                    tmpPart = tmpPart + patternNormal[i];
                }
            }
            return retValue;

        }

        private string Convert4BitTo16X(string Bit4Str)
        {
            int bit4Num = Int32.Parse(Bit4Str[3].ToString()) + 2 *
Int32.Parse(Bit4Str[2].ToString()) + 4 * Int32.Parse(Bit4Str[1].ToString()) + 8
* Int32.Parse(Bit4Str[0].ToString());
            if (bit4Num < 10)
                return bit4Num.ToString();
            else if (bit4Num == 10)
                return "A";
            else if (bit4Num == 11)
                return "B";
            else if (bit4Num == 12)
                return "C";
            else if (bit4Num == 13)
                return "D";
            else if (bit4Num == 14)
                return "E";
            else
                return "F";

        }
    }
}
```

Package (namespace) : BasicOper	**Source:** Security.cs

```
/// <summary>
/// Summary: description for Security class
/// ------------------------------------------------------------
/// It contains the Security class
/// ------------------------------------------------------------
/// Purpose: It represents a stock including all detail values
/// Additionally, contains the codification system for creation
/// of the patterns of gvol-candlesticks.
/// </summary>
/// ------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using System.IO;
using System.Data;
using System.Data.Sql;
using System.Data.SqlClient;

namespace AIPredictor.BasicOper
{
    class Security
    {
        private string symbol;
        private string name_of_symbol;
        private string type_of_security;
        private string country_origin;
        public ArrayList securityListValues;  //QuoteTimeSeries

        delegate bool CompareOp(object lhs, object rhs);

        /// <summary>
        /// Basic constractor method
        /// </summary>
        /// <param name="symbolStr"></param>
        /// <param name="name_of_symbolStr"></param>
        /// <param name="type_of_securityStr"></param>
        /// <param name="country_originStr"></param>
        public Security(string symbolStr, string name_of_symbolStr, string
type_of_securityStr, string country_originStr)
        {
            this.symbol = symbolStr;
            this.name_of_symbol = name_of_symbolStr;
            this.type_of_security = type_of_securityStr;
            this.country_origin = country_originStr;
            securityListValues = new ArrayList();
        }
```

```
        /// <summary>
        /// Another constractor method
        /// </summary>
        /// <param name="symbolStr"></param>
        /// <param name="country_originStr"></param>
        public Security(string symbolStr, string country_originStr)
        {
            this.symbol = symbolStr;
            this.name_of_symbol   = symbolStr;
            this.type_of_security = "stock";
            this.country_origin = country_originStr;
            securityListValues = new ArrayList();
        }

        /// <summary>
        /// no parameters
        /// </summary>
        public Security()
        {
            this.symbol = "noName";
            this.name_of_symbol = "noName";
            this.type_of_security = "stock";
            this.country_origin = "GREECE";
            securityListValues = new ArrayList();
        }

        /// <summary>
        /// sort method
        /// </summary>
        /// <param name="sortArray"></param>
        /// <param name="gtMethod"></param>
        private void SortAb(ArrayList sortArray, CompareOp gtMethod)
        {
            for (int i = 0; i < sortArray.Count; i++)
            {
                for (int j = i + 1; j < sortArray.Count; j++)
                {
                    if (gtMethod(sortArray[j], sortArray[i]))
                    {
                        SecurityValues temp = (SecurityValues)sortArray[i];
                        sortArray[i] = sortArray[j];
                        sortArray[j] = temp;
                    }
                }
            }
        }

        /// <summary>
        /// compare method
        /// </summary>
        /// <param name="gtMethod"></param>
        private void SortList( CompareOp gtMethod)
        {
```

```
            for (int i = 0; i < securityListValues.Count; i++)
            {
                for (int j = i + 1; j < securityListValues.Count; j++)
                {
                    if (gtMethod(securityListValues[j], securityListValues[i]))
                    {
                        SecurityValues temp =
(SecurityValues)securityListValues[i];
                        securityListValues[i] = securityListValues[j];
                        securityListValues[j] = temp;
                    }
                }
            }
        }

        /// <summary>
        /// implements the sort method
        /// </summary>
        public void Sort()
        {
            CompareOp secCompareOp = new
CompareOp(SecurityValues.RhsIsGreaterByClose);
            SortList(secCompareOp);
        }

        /// <summary>
        /// Converts numbers to string
        /// </summary>
        /// <returns></returns>
        public override string  ToString()
        {
            string outputText = "Symbol : "+ this.symbol + "\r\n";

            for (int i = 0; i < securityListValues.Count; i++)
            {
                outputText += ((SecurityValues)securityListValues[i]).Date_value
+
                              ((SecurityValues)securityListValues[i]).Open_price
+
                              ((SecurityValues)securityListValues[i]).High_price
+
                              ((SecurityValues)securityListValues[i]).Low_price
+

((SecurityValues)securityListValues[i]).Close_price +
                              ((SecurityValues)securityListValues[i]).Volume +
"\r\n";


            }
            return outputText;

        }
```

```
        // public property Symbol
        public string Symbol
        {
            get
            {
                return symbol;
            }
            set
            {
                symbol = value;
            }
        }

        // public property Name_of_symbol
        public string Name_of_symbol
        {
            get
            {
                return name_of_symbol;
            }
            set
            {
                name_of_symbol = value;
            }
        }

        // public property Type_of_security
        public string Type_of_security
        {
            get
            {
                return type_of_security;
            }
            set
            {
                type_of_security = value;
            }
        }

        // public property Country_origin
        public string Country_origin
        {
            get
            {
                return country_origin;
            }
            set
            {
                country_origin = value;
            }
        }
```

```
        public string getDate(int id)
        {
            try
            {
                return ((SecurityValues)securityListValues[id]).Date_value;
            }
            catch {
               return "";
            }
        }

        public float getOpen(int id)
        {
            try
            {
                return ((SecurityValues)securityListValues[id]).Open_price;
            }
            catch
            {
                return 0;
            }
        }

        public float getClose(int id)
        {
            try
            {
                return ((SecurityValues)securityListValues[id]).Close_price;
            }
            catch
            {
                return 0;
            }
        }

        public float getHigh(int id)
        {
            try
            {
                return ((SecurityValues)securityListValues[id]).High_price;
            }
            catch
            {
                return 0;
            }
        }

        public float getLow(int id)
        {
            try
            {
                return ((SecurityValues)securityListValues[id]).Low_price;
            }
```

```
            catch
            {
                return 0;
            }
        }

        public int getVolume(int id)
        {
            try
            {
                return ((SecurityValues)securityListValues[id]).Volume;
            }
            catch
            {
                return 0;
            }
        }

        /// <summary>
        /// return TP = Typical Price
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        public float getTP(int id)
        {
            return ((SecurityValues)securityListValues[id]).getTP();
        }

        /// <summary>
        /// True Range Value
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        public float getTR(int id)
        {

            return Math.Max(getHigh(id) - getLow(id),
                   Math.Max(getHigh(id) - getClose(id - 1), getClose(id - 1) -
getLow(id)));
        }

        public float getShadowSizeUP(int id)
        {
            return ((SecurityValues)securityListValues[id]).getShadowSizeUP();
        }

        public float getShadowSizeDown(int id)
        {
            return ((SecurityValues)securityListValues[id]).getShadowSizeDown();
        }

        public float getBodySize(int id)
        {
```

```
            return ((SecurityValues)securityListValues[id]).getBodySize();
        }

        public float getBodyUp(int id)
        {
          return ((SecurityValues)securityListValues[id]).getBodyUp();
        }

        public float getBodyDown(int id)
        {
            return ((SecurityValues)securityListValues[id]).getBodyDown();
        }

        public float getCandleStickSize(int id)
        {
            return
((SecurityValues)securityListValues[id]).getCandleStickSize();
        }

        public float getLowestClose(int id, int daysAgo)
        {
            float lowestClose = getClose(id);
            for (int i = 1; i < daysAgo; i++)
            {
                if (lowestClose > getClose(id - 1))
                    lowestClose = getClose(id - 1);
            }
            return lowestClose;
        }

        public float getHighestClose(int id, int daysAgo)
        {
            float highestClose = getClose(id);
            for (int i = 1; i < daysAgo; i++)
            {
                if (highestClose < getClose(id - 1))
                    highestClose = getClose(id - 1);
            }
            return highestClose;
        }

        public float getLowestLow(int id, int daysAgo)
        {
            float lowestLow = getLow(id);
            for (int i = 1; i < daysAgo; i++)
            {
                if (lowestLow > getLow(id - daysAgo))
                    lowestLow = getLow(id - daysAgo);
           }
           return lowestLow;
        }

        public float getHighestHigh(int id, int daysAgo)
```

```
        {
            float highestHigh = getHigh(id);
            for (int i = 1; i < daysAgo; i++)
            {
                if (highestHigh < getHigh(id - daysAgo))
                    highestHigh = getHigh(id - daysAgo);
            }
            return highestHigh;
        }

        public int getIdFromDate(string dateTgt)
        {
            int i = 0;
            Boolean isFound = false;
            int tgtId = -1;

            while (i < securityListValues.Count)
            {
                if
(((SecurityValues)securityListValues[i]).Date_value.CompareTo(dateTgt) == 0)
                {
                    tgtId = i;
                    isFound = true;
                }
                i += 1;
            }

            if (isFound)
                return tgtId;
            else
                return -1;
        }

        public float getClose(string dateTgt)
        {
            int id = getIdFromDate(dateTgt);
            if (id != -1)
                return getClose(id);
            else
                return -9999999999.0f;
        }

        public float getValue(int id, string typeValue)
        {
            if (typeValue.CompareTo("O") == 0)
            {
                return getOpen(id);
            }
            else if (typeValue.CompareTo("C") == 0)
            {
                return getClose(id);
            }
            else if (typeValue.CompareTo("H") == 0)
```

```
            {
                return getHigh(id);
            }
            else if (typeValue.CompareTo("L") == 0)
            {
                return getLow(id);
            }
            else if (typeValue.CompareTo("V") == 0)
            {
                return getVolume(id);
            }
            else
                return -1;  //error!
        }

        public int CountValues()
        {
            return securityListValues.Count;
        }

        public void LoadStockDataFromFile(string filename)
        {
            if (File.Exists(filename))
            {
                StreamReader myReader = File.OpenText(filename);
                SecurityValues currentSV;
                String curLine;
                myReader.ReadLine(); // ignore first row (it is the header)

                while ((curLine = myReader.ReadLine()) != null)
                {
                    currentSV = CreateObjRowFromString(curLine);
                    this.securityListValues.Add(currentSV);
                }

                myReader.Close();
            }
        }

        public void AutoLoadSymbol(string filename)
        {
            StreamReader myReader = File.OpenText(filename);
            String line;
            myReader.ReadLine(); // ignore first row (it is the header)

            // read next line
            if ((line = myReader.ReadLine()) != null)
            {
                int pos_No1 = line.IndexOf(',', 0);
                this.Symbol = line.Substring(0, pos_No1);
                this.name_of_symbol = this.Symbol;
            }
            else
```

```
            {
                this.Symbol = "No Name";
                this.name_of_symbol   = "No Name";
            }

            myReader.Close();
        }

        private SecurityValues CreateObjRowFromString(string line)
        {
            SecurityValues sv = new SecurityValues();
            string[] arValues = new string[7];
            for (int k = 0; k < 7; k++)
                arValues[k] = "";
            int j = 0;
            for (int i = 0; i < line.Length; i++)
            {
                if (line[i].CompareTo(',') == 0)
                    j++;
                else
                    arValues[j] = arValues[j] + line[i];
            }

            sv.setDate(arValues[1]);

            sv.Open_price = float.Parse(arValues[2]);

            sv.High_price = float.Parse(arValues[3]);

            sv.Low_price = float.Parse(arValues[4]);

            sv.Close_price = float.Parse(arValues[5]);

            sv.Volume = Int32.Parse(arValues[6]);

            return sv;
        }

        /// <summary>
        /// define white or black candlestick
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculateBit1(int id)
        {
            if (getOpen(id) < getClose(id))
            {
                return "1";
            }
            else
            {
                return "0";
            }
```

```
        }
        private string CalculateBit2(int id)
        {
            if (getBodyDown(id) > (getHigh(id) + getLow(id)) / 2)
                return "1";
            else
                return "0";
        }

        private string CalculateBit3(int id)
        {
            if (getBodyUp(id) < (getHigh(id) + getLow(id)) / 2)
                return "1";
            else
                return "0";
        }

        private string CalculateBit4(int id)
        {
            if (getClose(id) > (getHigh(id) + getLow(id)) / 2)
                return "1";
            else
                return "0";
        }

        /// <summary>
        /// Detail of Bit5_6
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculateBit5_6(int id)
        {
            float val = 100*(getClose(id)-getOpen(id))/getClose(id);

            if (val <= -3)
               return "00";
            else if ( (val >-3) && (val < 0))
                return "01";
            else if ( (val >=0) && (val < 3))
                return "10";
            else
                return "11";
        }

        private string CalculateBit7(int id)
        {
            if ((getShadowSizeUP(id) + getShadowSizeDown(id)) > getBodySize(id))
                return "1";
            else
                return "0";
        }

        private string CalculateBit8(int id)
```

```
        {
            if ( getShadowSizeUP(id)  > getBodySize(id))
                return "1";
            else
                return "0";
        }

        private string CalculateBit9(int id)
        {
            if ( getShadowSizeDown(id) > getBodySize(id))
                return "1";
            else
                return "0";
        }

        private string CalculateBit10(int id)
        {
            float sumVolAvg = 0.0f;
            for (int i = 0; i < 20; i++)
            {
                sumVolAvg += getVolume(id - i);
            }
            sumVolAvg = sumVolAvg / 20;

            if (getVolume(id) > sumVolAvg)
                return "1";
            else
                return "0";
        }

       // comparing Bits

        private string CalculateBit11(int id)
        {
            if (getClose(id) >= getClose(id-1))
                return "1";
            else
                return "0";
        }

        private string CalculateBit12_13(int id)
        {
            float val = 100 * (getClose(id) - getClose(id - 1)) / getClose(id -
1);

            if (val < -3)
                return "00";
            else if (val>=-3 && val < 0)
                return "01";
            else if (val >= 0 && val < 3)
                return "10";
            else
                return "11";
```

```
        }

        private string CalculateBit14_15_16(int id)
        {
            if ( (getBodyUp(id) > getBodyUp(id-1)) && (getBodyDown(id) <=
getBodyDown(id-1)) )
                return "001";
            else if ((getBodyUp(id) <= getBodyUp(id - 1)) && (getBodyDown(id) >=
getBodyDown(id - 1)))
                return "010";
            else if ((getBodyUp(id) > getBodyUp(id - 1)) && (getBodyDown(id) >
getBodyDown(id - 1)))
                return "011";
            else if ((getBodyUp(id) < getBodyUp(id - 1)) && (getBodyDown(id) <
getBodyDown(id - 1)))
                return "100";
            else if (getBodyUp(id) <= getBodyDown(id - 1))
                return "101";
            else
                return "111";
        }

        /// <summary>
        ///  Gap up
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculateBit17(int id)
        {
            if ( getLow(id) > getHigh(id - 1) )
                return "1";
            else
                return "0";
        }

        /// <summary>
        /// Gap down
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculateBit18(int id)
        {
            if (getHigh(id) < getLow(id - 1))
                return "1";
            else
                return "0";
        }

        private string CalculateBit19_20(int id)
        {
            if (getVolume(id - 1) == 0)   // neutral result
                return "10";
```

```
            float val = 100 * (getVolume(id) - getVolume(id - 1)) / getVolume(id
- 1);

            if (val < -15)
                return "00";
            else if (val >= -15 && val <= 15)
                return "10";
            else
                return "11";
        }

        private string CalculateBit21(int id)
        {
            if (getClose(id) >= getClose(id - 2))
                return "1";
            else
                return "0";
        }

        private string CalculateBit22(int id)
        {
            if ((getClose(id) == getHighestClose(id,4)) || (getClose(id) ==
getLowestClose(id,4)))
                return "1";
            else
                return "0";
        }

        /// <summary>
        /// Calculation of Pattern A ...length = 10 bits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculatePatternA(int id)
        {
            return CalculateBit1(id) + CalculateBit2(id) + CalculateBit3(id) +
                   CalculateBit4(id) + CalculateBit5_6(id) + CalculateBit7(id) +
                   CalculateBit8(id) + CalculateBit9(id) + CalculateBit10(id);
        }

        /// <summary>
        /// Calculation of Pattern A Basic...length = 4 bits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculatePatternA_basic(int id)
        {
            return CalculateBit1(id) + CalculateBit7(id) +CalculateBit8(id) +
CalculateBit9(id) ;
        }

        /// <summary>
        /// Calculation of Pattern A Alt1...length = 6 bits
```

```
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculatePatternA_alt1(int id)
        {
            return CalculateBit1(id) + CalculateBit2(id) + CalculateBit3(id) +
                   CalculateBit7(id) +CalculateBit8(id) + CalculateBit9(id);
        }

        /// <summary>
        /// Calculation of Pattern A Alt2...length = 8 bits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculatePatternA_alt2(int id)
        {
            return CalculateBit1(id) + CalculateBit2(id) + CalculateBit3(id) +
                   CalculateBit4(id)  + CalculateBit7(id) +
                   CalculateBit8(id) + CalculateBit9(id) + CalculateBit10(id);
        }

        /// <summary>
        /// Calculation of Pattern B ...length = 10 bits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculatePatternB(int id)
        {
            return CalculateBit11(id) + CalculateBit12_13(id) +
CalculateBit14_15_16(id) +
                   CalculateBit17(id) + CalculateBit18(id) +
CalculateBit19_20(id);
        }

        /// <summary>
        /// Calculation of Pattern B basic ...length = 3 bits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculatePatternB_basic(int id)
        {
            return CalculateBit11(id) + CalculateBit17(id) + CalculateBit18(id);
        }

        /// <summary>
        ///  Calculation of Pattern B alt1 ...length = 6 bits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculatePatternB_alt1(int id)
        {
            return CalculateBit11(id) + CalculateBit14_15_16(id) +
                   CalculateBit17(id) + CalculateBit18(id);
```

```
        }

        /// <summary>
        /// Calculation of Pattern B alt2...length = 10 bits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculatePatternB_alt2(int id)
        {
            return CalculatePatternB(id);
        }

        /// <summary>
        /// Calculation of Pattern C ...length = 2 bits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        private string CalculatePatternC(int id)
        {
            return CalculateBit21(id) + CalculateBit22(id);
        }

        /// <summary>
        /// It creates the pattern of 2 candles...length = 30 bits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        public string CreatePatternOfTwo(int id)
        {
            return CalculatePatternA(id - 1) + CalculatePatternA(id) +
CalculatePatternB(id);
        }

        public string CreatePatternOfTwo_basic(int id)
        {
            return CalculatePatternA_basic(id - 1) + CalculatePatternA_basic(id)
+ CalculatePatternB_basic(id);
        }

        public string CreatePatternOfTwo_alt1(int id)
        {
            return CalculatePatternA_alt1(id - 1) + CalculatePatternA_alt1(id) +
CalculatePatternB_alt1(id);
        }

        public string CreatePatternOfTwo_alt2(int id)
        {
            return CalculatePatternA_alt2(id - 1) + CalculatePatternA_alt2(id) +
CalculatePatternB_alt2(id);
        }

        /// <summary>
        /// It creates the pattern of 3 candles...length = 52 bits
```

```
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        public string CreatePatternOfThree(int id)
        {
            return CalculatePatternA(id - 2) + CalculatePatternA(id - 1) +
CalculatePatternA(id) +
                   CalculatePatternB(id - 1) + CalculatePatternB(id) +
CalculatePatternC(id);
        }

        public string CreatePatternOfThree_basic(int id)
        {
            return CalculatePatternA_basic(id - 2) + CalculatePatternA_basic(id
- 1) + CalculatePatternA_basic(id) +
                   CalculatePatternB_basic(id - 1) +
CalculatePatternB_basic(id);
        }

        public string CreatePatternOfThree_alt1(int id)
        {
            return CalculatePatternA_alt1(id - 2) + CalculatePatternA_alt1(id -
1) + CalculatePatternA_alt1(id) +
                   CalculatePatternB_alt1(id - 1) + CalculatePatternB_alt1(id);
        }

        public string CreatePatternOfThree_alt2(int id)
        {
            return CalculatePatternA_alt2(id - 2) + CalculatePatternA_alt2(id -
1) + CalculatePatternA_alt2(id) +
                   CalculatePatternB_alt2(id - 1) + CalculatePatternB_alt2(id);
        }

        /// <summary>
        /// It creates the pattern of 2 candles...length = 72 bits
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        public string CreatePatternOfFour(int id)
        {
            return CalculatePatternA(id - 3) + CalculatePatternA(id - 2) +
CalculatePatternA(id - 1) + CalculatePatternA(id) +
                   CalculatePatternB(id - 2) + CalculatePatternB(id - 1) +
CalculatePatternB(id) + CalculatePatternC(id);
        }

        public string CreatePatternOfFour_basic(int id)
        {
            return CalculatePatternA_basic(id - 3) + CalculatePatternA_basic(id
- 2) + CalculatePatternA_basic(id - 1) + CalculatePatternA_basic(id) +
                   CalculatePatternB_basic(id - 2) + CalculatePatternB_basic(id
- 1) + CalculatePatternB_basic(id);
        }
```

```
        public string CreatePatternOfFour_alt1(int id)
        {
            return CalculatePatternA_alt1(id - 3) + CalculatePatternA_alt1(id -
2) + CalculatePatternA_alt1(id - 1) + CalculatePatternA_alt1(id) +
                   CalculatePatternB_alt1(id - 2) + CalculatePatternB_alt1(id -
1) + CalculatePatternB_alt1(id);
        }

        public string CreatePatternOfFour_alt2(int id)
        {
            return CalculatePatternA_alt2(id - 3) + CalculatePatternA_alt2(id -
2) + CalculatePatternA_alt2(id - 1) + CalculatePatternA_alt2(id) +
                   CalculatePatternB_alt2(id - 2) + CalculatePatternB_alt2(id -
1) + CalculatePatternB_alt2(id);
        }

        public CandlePattern[] getPattern2List()
        {
            int nlength = securityListValues.Count - 23;

            CandlePattern[] patternListOf2 = new CandlePattern[nlength];
            float[] percentChangeAr = new float[12];

            for (int i = 3; i <= securityListValues.Count - 21; i++)
            {
                CandlePattern curPattern = new CandlePattern();
                curPattern.setDate(this.getDate(i));
                curPattern.PatternStr = CreatePatternOfTwo(i);
                curPattern.PatternBasic = CreatePatternOfTwo_basic(i);
                curPattern.PatternAlt1 = CreatePatternOfTwo_alt1(i);
                curPattern.PatternAlt2 = CreatePatternOfTwo_alt2(i);

                for (int j = 0; j < 10; j++)
                {
                    percentChangeAr[j] = 100 * (getClose(j + i + 1) -
getClose(i)) / getClose(i);
                }

                percentChangeAr[10] = 100 * (getClose(i + 15) - getClose(i)) /
getClose(i);
                percentChangeAr[11] = 100 * (getClose(i + 20) - getClose(i)) /
getClose(i);

                curPattern.setArrayOfChanges(percentChangeAr);

                patternListOf2[i - 3] = curPattern;
                curPattern = null;
            }
            return patternListOf2;
        }

        public CandlePattern[] getPattern3List()
```

```
        {
            int nlength = securityListValues.Count - 23;

            CandlePattern[] patternListOf3 = new CandlePattern[nlength];
            float[] percentChangeAr = new float[12];

            for (int i = 3; i <= securityListValues.Count - 21; i++)
            {
                CandlePattern curPattern = new CandlePattern();
                curPattern.setDate(this.getDate(i));
                curPattern.PatternStr = CreatePatternOfThree(i);
                curPattern.PatternBasic = CreatePatternOfThree_basic(i);
                curPattern.PatternAlt1 = CreatePatternOfThree_alt1(i);
                curPattern.PatternAlt2 = CreatePatternOfThree_alt2(i);

                for (int j = 0; j < 10; j++)
                {
                    percentChangeAr[j] = 100 * (getClose(j + i + 1) -
getClose(i)) / getClose(i);
                }

                percentChangeAr[10] = 100 * (getClose(i + 15) - getClose(i)) /
getClose(i);
                percentChangeAr[11] = 100 * (getClose(i + 20) - getClose(i)) /
getClose(i);

                curPattern.setArrayOfChanges(percentChangeAr);

                patternListOf3[i - 3] = curPattern;
                curPattern = null;
            }
            return patternListOf3;
        }

        public CandlePattern[] getPattern4List()
        {
            int nlength = securityListValues.Count - 23;

            CandlePattern[] patternListOf4 = new CandlePattern[nlength];
            float[] percentChangeAr = new float[12];

            for (int i = 3; i <= securityListValues.Count - 21; i++)
            {
                CandlePattern curPattern = new CandlePattern();
                curPattern.setDate(this.getDate(i));
                curPattern.PatternStr = CreatePatternOfFour(i);
                curPattern.PatternBasic = CreatePatternOfFour_basic(i);
                curPattern.PatternAlt1 = CreatePatternOfFour_alt1(i);
                curPattern.PatternAlt2 = CreatePatternOfFour_alt2(i);

                for (int j = 0; j < 10; j++)
                {
                    percentChangeAr[j] = 100 * (getClose(j + i + 1) -
```

```
getClose(i)) / getClose(i);
                }

                percentChangeAr[10] = 100 * (getClose(i + 15) - getClose(i)) /
getClose(i);
                percentChangeAr[11] = 100 * (getClose(i + 20) - getClose(i)) /
getClose(i);

                curPattern.setArrayOfChanges(percentChangeAr);

                patternListOf4[i - 3] = curPattern;
                curPattern = null;
            }
            return patternListOf4;
        }
    }

}
```

Package (namespace) : BasicOper	**Source:** SecurityValues.cs

```
/// <summary>
/// Summary: description for SecurityValues class
/// -----------------------------------------------------------------
/// It contains the SecurityValues class
/// -----------------------------------------------------------------
/// Purpose: It contains all detail stock values values
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;

namespace AIPredictor.BasicOper
{

    class SecurityValues
    {
        // private attributes
        private Date date_value;
        private float open_price;
        private float high_price;
        private float low_price;
        private float close_price;
        private int volume;

        /// <summary>
        /// Constractor method
        /// </summary>
        public SecurityValues()
        {
            this.setDate("19000101");
            this.Open_price    = 0;
            this.High_price    = 0;
            this.Low_price     = 0;
            this.Close_price   = 0;
            this.Volume        = 0;
        }

        /// <summary>
        /// Used for comparison - sort by close value
        /// </summary>
        /// <param name="lhs"></param>
        /// <param name="rhs"></param>
        /// <returns></returns>
        public static bool RhsIsGreaterByClose(object lhs, object rhs)
        {
            SecurityValues secValuesLhs = (SecurityValues)lhs;
```

```
            SecurityValues secValuesRhs = (SecurityValues)rhs;
            return (secValuesRhs.Close_price > secValuesLhs.Close_price) ? true
: false;

        }

        /// <summary>
        /// Used for comparison - sort by date value
        /// </summary>
        /// <param name="lhs"></param>
        /// <param name="rhs"></param>
        /// <returns></returns>
        public static bool RhsIsGreaterByDate(object lhs, object rhs)
        {
            SecurityValues secValuesLhs = (SecurityValues)lhs;
            SecurityValues secValuesRhs = (SecurityValues)rhs;
            return (secValuesRhs.date_value.LengthDate() >
secValuesRhs.date_value.LengthDate()) ? true : false;
        }


        // public property Date_value
        public string Date_value
        {
            get
            {
                return date_value.ToDateString();
            }
        }

        public void setDate(string strDate)
        {
            this.date_value = Date.ConvetStringToDate(strDate);
        }

        // public property Open_price
        public float Open_price
        {
            get
            {
                return open_price;
            }
            set
            {
                open_price = value;
            }
        }

        // public property High_price
        public float High_price
        {
            get
            {
```

```
                return high_price;
            }
            set
            {
                high_price = value;
            }
        }

        // public property low_price
        public float Low_price
        {
            get
            {
                return low_price;
            }
            set
            {
                low_price = value;
            }
        }

        // public property Close_price
        public float Close_price
        {
            get
            {
                return close_price;
            }
            set
            {
                close_price = value;
            }
        }

        // public property Volume
        public int Volume
        {
            get
            {
                return volume;
            }
            set
            {
                volume = value;
            }
        }

        /// <summary>
        /// return TP = Typical Price
        /// </summary>
        /// <returns></returns>
        public float getTP()
        {
```

```
            return (High_price + Low_price + Close_price) / 3;
        }

        public float getShadowSizeUP()
        {
            return High_price - Math.Max(Close_price, open_price);

        }

        public float getShadowSizeDown()
        {
            return Math.Min(Close_price, open_price) - Low_price;

        }

        public float getBodySize()
        {
            return Math.Abs(Close_price - open_price);

        }

        public float getBodyUp()
        {
            return Math.Max(Close_price, open_price);

        }

        public float getBodyDown()
        {
            return Math.Min(Close_price, open_price);

        }

        public float getCandleStickSize()
        {
            return High_price - Low_price;
        }

    }
}
```

Package (namespace) : BasicOper	**Source:** DBLayer.cs

```
/// <summary>
/// Summary: description for DBLayer class
/// -----------------------------------------------------------------
/// It contains the DBLayer class
/// -----------------------------------------------------------------
/// Purpose: Basic class for establishing connections to the database (SQL
Server)
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using System.IO;
using System.Data;
using System.Data.Sql;
using System.Data.SqlClient;
using AIPredictor.Signals;
using AIPredictor.SmartInput;

namespace AIPredictor.BasicOper
{
    class DBLayer
    {
        private SqlConnection connectionDB;
        private string connectionStingDB;
        private Boolean isConnected;
        private string connection_file;

        /// <summary>
        /// basic constractor method -
        /// It reads the connection string from text file
        /// </summary>
        public DBLayer()
        {
            readFileForConnection();
            this.connectionDB = new SqlConnection();
            this.connectionDB.ConnectionString = connection_file;
            this.isConnected = false;
        }

        /// <summary>
        /// Another contractor method (takes parameter for string connection
        /// </summary>
        /// <param name="conString"></param>
        public DBLayer(string conString)
        {
```

```
            this.connectionDB = new SqlConnection();
            this.connectionDB.ConnectionString = conString;
            this.isConnected = false;
        }

        /// <summary>
        /// reads the connection string from specific file: connectionDB.txt
        /// </summary>
        private void readFileForConnection()
        {
            connection_file = "";

            // determine whether fileName is a file
            if (File.Exists("connectionDB.txt"))
            {
                StreamReader myReader = File.OpenText("connectionDB.txt");
                string inputStr = "";
                while ((inputStr = myReader.ReadLine()) != null)
                {
                    connection_file = connection_file + inputStr;
                }
                myReader.Close();
            }
            else  // local connection - SQL DB is installed in the same computer
                connection_file = "data source =  (Local);Initial
Catalog=StockMarket;Integrated Security=True";

        }

        // public property ConnectionDB
        public SqlConnection ConnectionDB
        {
            get
            {
                return connectionDB;
            }
            set
            {
                connectionDB = value;
            }
        }

        // public property ConnectionStingDB
        public string ConnectionStingDB
        {
            get
            {
                return connectionStingDB;
            }
            set
            {
                connectionStingDB = value;
            }
```

```
        }

        // public property IsConnected
        public Boolean IsConnected
        {
            get
            {
                return isConnected;
            }
        }

        public void Connect()
        {
            try
            {
                connectionDB.Open();
                isConnected = true;
            }
            catch (Exception ex)
            {
                System.Console.Out.Write("Error: " + ex.Message);
                isConnected = false;
            }
        }

        public void Disconnect()
        {
            try
            {
                connectionDB.Close();
                isConnected = false;
            }
            catch (Exception ex)
            {
                System.Console.Out.Write("Error: " + ex.Message);
            }
        }

        /// <summary>
        /// Saves the object of Security (stock) in DB
        /// It calls specific database procedure
        /// </summary>
        /// <param name="sec"></param>
        public void SaveSecurityInDB(Security sec)
        {
            SqlCommand myCommand = new SqlCommand("AddSecurity", ConnectionDB);
            myCommand.CommandType = CommandType.StoredProcedure;

            SqlParameter myParam;

            myParam = new SqlParameter("@symbol", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);
```

```
            myParam = new SqlParameter("@name_of_symbol", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@type_of_security", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@country_origin", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myCommand.Parameters[0].Value = sec.Symbol;
            myCommand.Parameters[1].Value = sec.Name_of_symbol;
            myCommand.Parameters[2].Value = sec.Type_of_security;
            myCommand.Parameters[3].Value = sec.Country_origin;
            myCommand.ExecuteNonQuery();
        }

        /// <summary>
        /// Saves the object detail values of Security (stock) in DB
        /// It calls specific database procedure
        /// </summary>
        /// <param name="sec"></param>
        public void SaveSecurityValuesInDB(Security sec)
        {
            SqlCommand myCommand = new SqlCommand("AddSecurityValues",
ConnectionDB);
            myCommand.CommandType = CommandType.StoredProcedure;

            SqlParameter myParam;

            myParam = new SqlParameter("@symbol", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@dateStr", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@open_price", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@high_price", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@low_price", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);
```

```
            myParam = new SqlParameter("@close_price", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@volume", SqlDbType.Int);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            for (int i = 0; i<sec.securityListValues.Count;i++)
            {
                myCommand.Parameters[0].Value = sec.Symbol;
                myCommand.Parameters[1].Value = sec.getDate(i);
                myCommand.Parameters[2].Value = sec.getOpen(i);
                myCommand.Parameters[3].Value = sec.getHigh(i);
                myCommand.Parameters[4].Value = sec.getLow(i);
                myCommand.Parameters[5].Value = sec.getClose(i);
                myCommand.Parameters[6].Value = sec.getVolume(i);

                myCommand.ExecuteNonQuery();
            }
        }

        /// <summary>
        /// Load Pattern Parameters
        /// </summary>
        /// <param name="myCommand"></param>
        /// <returns></returns>
        private SqlCommand LoadPatternParameters(SqlCommand myCommand)
        {
            SqlParameter myParam;

            myParam = new SqlParameter("@symbol", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@dateStr", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@patternStr", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@patternBasic", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@patternAlt1", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@patternAlt2", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
```

```
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@pattern16xNumber", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange1Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange2Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange3Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange4Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange5Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange6Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange7Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange8Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange9Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange10Value",
SqlDbType.Decimal);
```

```
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange15Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@percentChange20Value",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            return myCommand;
        }

        /// <summary>
        /// Insert Pattern values in DB
        /// </summary>
        /// <param name="myCommand"></param>
        /// <param name="symbolStr"></param>
        /// <param name="patternList"></param>
        private void ExecInsertPatternValuesInDB(SqlCommand myCommand, string
symbolStr, CandlePattern[] patternList)
        {
            for (int i = 0; i < patternList.Length; i++)
            {
                myCommand.Parameters[0].Value  = symbolStr;
                myCommand.Parameters[1].Value  = patternList[i].Date_value;
                myCommand.Parameters[2].Value  = patternList[i].PatternStr;
                myCommand.Parameters[3].Value = patternList[i].PatternBasic;
                myCommand.Parameters[4].Value = patternList[i].PatternAlt1;
                myCommand.Parameters[5].Value = patternList[i].PatternAlt2;
                myCommand.Parameters[6].Value = patternList[i].Pattern16xNumber;
                myCommand.Parameters[7].Value = patternList[i].PercentChange1th;
                myCommand.Parameters[8].Value  =
patternList[i].PercentChange2th;
                myCommand.Parameters[9].Value  =
patternList[i].PercentChange3th;
                myCommand.Parameters[10].Value =
patternList[i].PercentChange4th;
                myCommand.Parameters[11].Value =
patternList[i].PercentChange5th;
                myCommand.Parameters[12].Value =
patternList[i].PercentChange6th;
                myCommand.Parameters[130].Value =
patternList[i].PercentChange7th;
                myCommand.Parameters[14].Value =
patternList[i].PercentChange8th;
                myCommand.Parameters[15].Value =
patternList[i].PercentChange9th;
                myCommand.Parameters[16].Value =
patternList[i].PercentChange10th;
```

```
                myCommand.Parameters[17].Value =
patternList[i].PercentChange15th;
                myCommand.Parameters[18].Value =
patternList[i].PercentChange20th;

                myCommand.ExecuteNonQuery();
            }
        }

        /// <summary>
        /// Save Pattern Of Two In DB
        /// </summary>
        /// <param name="sec"></param>
        public void SavePatternOfTwoInDB(Security sec)
        {
            SqlCommand myCommand = new SqlCommand("CreatePatternOfTwo",
ConnectionDB);
            myCommand.CommandType = CommandType.StoredProcedure;

            myCommand = LoadPatternParameters(myCommand);

            ExecInsertPatternValuesInDB(myCommand, sec.Symbol,
sec.getPattern2List());
        }

        /// <summary>
        /// Save Pattern Of Three In DB
        /// </summary>
        /// <param name="sec"></param>
        public void SavePatternOfThreeInDB(Security sec)
        {
            SqlCommand myCommand = new SqlCommand("CreatePatternOfThree",
ConnectionDB);
            myCommand.CommandType = CommandType.StoredProcedure;

            myCommand = LoadPatternParameters(myCommand);

            ExecInsertPatternValuesInDB(myCommand, sec.Symbol,
sec.getPattern3List());
        }

        /// <summary>
        /// Save Pattern Of Four In DB
        /// </summary>
        /// <param name="sec"></param>
        public void SavePatternOfFourInDB(Security sec)
        {
            SqlCommand myCommand = new SqlCommand("CreatePatternOfFour",
ConnectionDB);
            myCommand.CommandType = CommandType.StoredProcedure;

            myCommand = LoadPatternParameters(myCommand);
```

```
            ExecInsertPatternValuesInDB(myCommand, sec.Symbol,
sec.getPattern4List());
        }

        /// <summary>
        /// Load Security From DB into the class Security
        /// </summary>
        /// <param name="symbol_tgt"></param>
        /// <returns></returns>
        public Security LoadSecurityFromDB(string symbol_tgt)
        {
            Security mySecurity = null;

            DataSet mySQLDS = new DataSet();
            SqlCommand myCommand = new SqlCommand("GetSecurity", ConnectionDB);
            myCommand.CommandType = CommandType.StoredProcedure;

            SqlParameter symbolParam = new SqlParameter("@symbol",
SqlDbType.VarChar);
            symbolParam.Direction = ParameterDirection.Input;
            symbolParam.Value = symbol_tgt;
            myCommand.Parameters.Add(symbolParam);

            SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter(myCommand);
            mySqlDataAdapter.Fill(mySQLDS);

            if (mySQLDS.Tables[0].Rows.Count > 0)
            {
                mySecurity = new Security(
mySQLDS.Tables[0].Rows[0][0].ToString(),
mySQLDS.Tables[0].Rows[0][1].ToString(),

mySQLDS.Tables[0].Rows[0][2].ToString(),
mySQLDS.Tables[0].Rows[0][3].ToString());
            }
            return mySecurity;
        }

        /// <summary>
        /// Load all symbols of stocks from Db
        /// </summary>
        /// <returns></returns>
        public string[] LoadAllSymbols()
        {
            string[] myListSymbols = null;

            DataSet mySQLDS = new DataSet();
            SqlCommand myCommand = new SqlCommand("Select symbol from
dbo.security order by symbol", ConnectionDB);
            myCommand.CommandType = CommandType.Text;

            SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter(myCommand);
            mySqlDataAdapter.Fill(mySQLDS);
```

```
            if (mySQLDS.Tables[0].Rows.Count > 0)
            {
                myListSymbols = new string[mySQLDS.Tables[0].Rows.Count];
                for (int i = 0; i < mySQLDS.Tables[0].Rows.Count; i++)
                {
                    myListSymbols[i] = mySQLDS.Tables[0].Rows[i][0].ToString();
                }
            }
            else
            {
                myListSymbols = new string[1];
                myListSymbols[0] = "error!";
            }
            return myListSymbols;
        }

        /// <summary>
        /// Load all indicators contained in DB
        /// </summary>
        /// <returns></returns>
        public string[] LoadAllIndicators()
        {
            string[] myListInd = null;

            DataSet mySQLDS = new DataSet();
            SqlCommand myCommand = new SqlCommand("Select codeInd from
dbo.indicator", ConnectionDB);
            myCommand.CommandType = CommandType.Text;

            SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter(myCommand);
            mySqlDataAdapter.Fill(mySQLDS);

            if (mySQLDS.Tables[0].Rows.Count > 0)
            {
                myListInd = new string[mySQLDS.Tables[0].Rows.Count];
                for (int i = 0; i < mySQLDS.Tables[0].Rows.Count; i++)
                {
                    myListInd[i] = mySQLDS.Tables[0].Rows[i][0].ToString();
                }
            }
            else
            {
                myListInd = new string[1];
                myListInd[0] = "error!";
            }
            return myListInd;
        }

        /// <summary>
        /// LoadSecurity Values From DB
        /// </summary>
        /// <param name="sec"></param>
```

```
        /// <param name="date_valueStr"></param>
        public void LoadSecurityValuesFromDB(Security sec, string date_valueStr)
        {
            DataSet mySQLDS = new DataSet();
            SqlCommand myCommand = new SqlCommand("GetSecurityValues",
ConnectionDB);
            myCommand.CommandType = CommandType.StoredProcedure;

            SqlParameter symbolParam = new SqlParameter("@symbol",
SqlDbType.VarChar);
            symbolParam.Direction = ParameterDirection.Input;
            symbolParam.Value = sec.Symbol;
            myCommand.Parameters.Add(symbolParam);

            SqlParameter date_valueParam = new SqlParameter("@dateStr",
SqlDbType.VarChar);
            date_valueParam.Direction = ParameterDirection.Input;
            date_valueParam.Value = date_valueStr;
            myCommand.Parameters.Add(date_valueParam);

            SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter(myCommand);
            mySqlDataAdapter.Fill(mySQLDS);

            if (mySQLDS.Tables[0].Rows.Count > 0)
            {
                for (int i = 0; i < mySQLDS.Tables[0].Rows.Count; i++)
                {
                    SecurityValues secValues = new SecurityValues();
                    secValues.setDate(mySQLDS.Tables[0].Rows[i][0].ToString());
                    secValues.Open_price  =
float.Parse(mySQLDS.Tables[0].Rows[i][1].ToString());
                    secValues.High_price  =
float.Parse(mySQLDS.Tables[0].Rows[i][2].ToString());
                    secValues.Low_price   =
float.Parse(mySQLDS.Tables[0].Rows[i][3].ToString());
                    secValues.Close_price =
float.Parse(mySQLDS.Tables[0].Rows[i][4].ToString());
                    secValues.Volume =
Int32.Parse(mySQLDS.Tables[0].Rows[i][5].ToString());

                    sec.securityListValues.Add(secValues);

                    secValues = null;
                }
            }

        }

        /// <summary>
        /// Save TradeHolderList class in DB
        /// </summary>
        /// <param name="holderList"></param>
        public void SaveTradeHolderList(TradeHolderList holderList)
```

```
        {
            for (int i = 0; i < holderList.getTradeHoldersSum().Count; i++)
            {
                SaveTradeHolder(holderList.getTradeHoldersSum()[i]);
            }
        }

        /// <summary>
        /// Save TradeHolder class in DB
        /// </summary>
        /// <param name="holder"></param>
        public void SaveTradeHolder(TradeHolder holder)
        {
            SqlCommand myCommand = new SqlCommand("AITradeUpdate",
ConnectionDB);
            myCommand.CommandType = CommandType.StoredProcedure;

            SqlParameter myParam;

            myParam = new SqlParameter("@symbol", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);


            myParam = new SqlParameter("@codeInd", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@typeoftrade", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@fromdateappliedStr",
SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@todateappliedStr", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@numberofstocks", SqlDbType.Int);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@feepertrade", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@totalearning", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);
```

```
            myParam = new SqlParameter("@totaltrades", SqlDbType.Int);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@successtrades", SqlDbType.Int);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@unsuccesstrades", SqlDbType.Int);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@performance", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@annualperformance", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@parameterDesc", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@parameter1", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@parameter2", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@parameter3", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myCommand.Parameters[0].Value = holder.Symbol;
            myCommand.Parameters[1].Value = holder.Code_of_trade;
            myCommand.Parameters[2].Value = holder.TypeOfTrade;
            myCommand.Parameters[3].Value = holder.FromDateApplied;
            myCommand.Parameters[4].Value = holder.ToDateApplied;
            myCommand.Parameters[5].Value = holder.NumberOfStocks;
            myCommand.Parameters[6].Value = holder.FeePerTrade;
            myCommand.Parameters[7].Value = holder.TotalEarning;
            myCommand.Parameters[8].Value = holder.TotalTrades;
            myCommand.Parameters[9].Value = holder.SuccessTrades;
            myCommand.Parameters[10].Value = holder.UnSuccessTrades;
            myCommand.Parameters[11].Value = holder.Performance;
            myCommand.Parameters[12].Value = holder.AnnualPerformance;
            myCommand.Parameters[13].Value = holder.ParameterDesc;
            myCommand.Parameters[14].Value = holder.Parameter1;
            myCommand.Parameters[15].Value = holder.Parameter2;
            myCommand.Parameters[16].Value = holder.Parameter3;
```

```
            myCommand.ExecuteNonQuery();
        }

        /// <summary>
        /// Load  optimized parameters of all indicators referred
        /// specific stock (symbol)
        /// </summary>
        /// <param name="tgtSymbol"></param>
        /// <returns></returns>
        public List<IndicatorOptParam> LoadAITradeOptValues(string tgtSymbol)
        {
            DataSet mySQLDS = new DataSet();
            SqlCommand myCommand = new SqlCommand("GetAITradeValues",
ConnectionDB);
            myCommand.CommandType = CommandType.StoredProcedure;

            SqlParameter symbolParam = new SqlParameter("@symbol",
SqlDbType.VarChar);
            symbolParam.Direction = ParameterDirection.Input;
            symbolParam.Value = tgtSymbol;
            myCommand.Parameters.Add(symbolParam);

            SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter(myCommand);
            mySqlDataAdapter.Fill(mySQLDS);

            List<IndicatorOptParam> myListOptParams = new
List<IndicatorOptParam>();
            IndicatorOptParam curIndOptParam;

            if (mySQLDS.Tables[0].Rows.Count > 0)
            {
                for (int i = 0; i < mySQLDS.Tables[0].Rows.Count; i++)
                {
                    curIndOptParam = new
IndicatorOptParam(mySQLDS.Tables[0].Rows[i][0].ToString(),
//codeInd

float.Parse(mySQLDS.Tables[0].Rows[i][1].ToString()), //annualPerformance

mySQLDS.Tables[0].Rows[i][2].ToString(),              //parameterDesc

float.Parse(mySQLDS.Tables[0].Rows[i][3].ToString()),  //parameter1

float.Parse(mySQLDS.Tables[0].Rows[i][4].ToString()),  //parameter1

float.Parse(mySQLDS.Tables[0].Rows[i][5].ToString()) );  //parameter1

                    myListOptParams.Add(curIndOptParam);
                    curIndOptParam = null;
                }
            }
```

```
            return myListOptParams;
        }


        /// <summary>
        /// Save results of the neural net training
        /// </summary>
        /// <param name="symbol_"></param>
        /// <param name="hiddenLayers_"></param>
        /// <param name="lessNeurons_"></param>
        /// <param name="trainingCycles_"></param>
        /// <param name="numberOfBasIndicators_"></param>
        /// <param name="validPerformance_"></param>
        /// <param name="outPutSelection_"></param>
        /// <param name="buyWhen_"></param>
        /// <param name="sellWhen_"></param>
        /// <param name="totalTradesPrediction_"></param>
        /// <param name="predictionOfBuy_"></param>
        /// <param name="predictionOfSell_"></param>
        /// <param name="annualNetPerformance_"></param>
        public void SaveSimResult(string symbol_, int hiddenLayers_, int
lessNeurons_,
                  int trainingCycles_, int numberOfBasIndicators_, double
validPerformance_,
                  int outPutSelection_, double buyWhen_, double sellWhen_,
                          double totalTradesPrediction_, double predictionOfBuy_,
                          double predictionOfSell_, double annualNetPerformance_)
        {
            SqlCommand myCommand = new SqlCommand("addNeuralResult",
ConnectionDB);
            myCommand.CommandType = CommandType.StoredProcedure;

            SqlParameter myParam;

            myParam = new SqlParameter("@symbol", SqlDbType.VarChar);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@hiddenLayers", SqlDbType.SmallInt);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@lessNeurons", SqlDbType.Int);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@trainingCycles", SqlDbType.Int);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@numberOfBasIndicators", SqlDbType.Int);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);
```

```
            myParam = new SqlParameter("@validPerformance", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@outPutSelection", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@buyWhen", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@sellWhen", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@totalTradesPrediction",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@predictionOfBuy", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@predictionOfSell", SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myParam = new SqlParameter("@annualNetPerformance",
SqlDbType.Decimal);
            myParam.Direction = ParameterDirection.Input;
            myCommand.Parameters.Add(myParam);

            myCommand.Parameters[0].Value = symbol_;
            myCommand.Parameters[1].Value = hiddenLayers_;
            myCommand.Parameters[2].Value = lessNeurons_;
            myCommand.Parameters[3].Value = trainingCycles_;
            myCommand.Parameters[4].Value = numberOfBasIndicators_;
            myCommand.Parameters[5].Value = validPerformance_;
            myCommand.Parameters[6].Value = outPutSelection_;
            myCommand.Parameters[7].Value = buyWhen_;
            myCommand.Parameters[8].Value = sellWhen_;
            myCommand.Parameters[9].Value = totalTradesPrediction_;
            myCommand.Parameters[10].Value = predictionOfBuy_;
            myCommand.Parameters[11].Value = predictionOfSell_;
            myCommand.Parameters[12].Value = annualNetPerformance_;

            myCommand.ExecuteNonQuery();
        }
    }
}
```

Package (namespace) : BasicOper	**Source:** Indicator.cs

```
/// <summary>
/// Summary: description for Indicator class
/// -----------------------------------------------------------
/// It contains the Indicator class
/// -----------------------------------------------------------
/// Purpose: All technical indicators used in this project
/// </summary>
/// -----------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;

namespace AIPredictor.BasicOper
{
    class Indicator
    {


        /* Category A: Volatility Indicators
         * Volatility is a general term used to describe the magnitude of day-
to-day fluctuations in prices
         * (independent of direction). Generally, changes in volatility tend to
lead to changes in prices.
         */

        /// <summary>
        /// 1a. Indicator: Simple Moving Average. It is tested - OK.
        /// It calculates close or high or low values depending on the input
        /// parameter 'typeValue'
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesMA"></param>
        /// <param name="typeValue"></param>
        /// <returns></returns>
        public static float SMA(Security sec, int id, int timesMA, string
typeValue)
        {
            float sumValue = 0;

            for (int i = 0; i < timesMA; i++)
            {
                sumValue = sumValue + sec.getValue(id - i, typeValue);
            }

            return sumValue / timesMA;
        }
```

```
        /// <summary>
        /// 1b. Indicator: Simple Moving Average. It is tested - OK.
        /// By default, it calculates close values.
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesMA"></param>
        /// <returns></returns>
        public static float SMA(Security sec, int id, int timesMA)
        {
            float sumValue = 0;

            for (int i = 0; i < timesMA; i++)
            {
                sumValue = sumValue + sec.getClose(id - i);
            }
            return sumValue / timesMA;
        }

        /// <summary>
        /// 2a.Indicator: Exponential Moving Average Calculation. It is tested -
OK.
        /// It calculates close or high or low values depending on the input
        /// parameter 'typeValue'
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="periodsAgo"></param>
        /// <returns></returns>
        public static float EMA(Security sec, int id, int periodsAgo, string
typeValue)
        {
            float a = 2 / (1 + periodsAgo);  // a = smoothing variable
            float retEMA = 0.0f;

            for (int xPeriods = 1; xPeriods <= periodsAgo; xPeriods++)
            {
                retEMA = retEMA + (float)Math.Pow(a, xPeriods) * sec.getValue(id
- xPeriods + 1, typeValue);
            }

            return retEMA;
        }

        /// <summary>
        /// 2b.Indicator: Exponential Moving Average Calculation. It is tested -
OK.
        /// The Exponential Moving Average is a weighted moving average where
the most recent values are
        /// weighted higher than the previous values.
        /// </summary>
        /// <param name="sec"></param>
```

```
        /// <param name="id"></param>
        /// <param name="periodsAgo"></param>
        /// <returns></returns>
        public static float EMA(Security sec, int id, int periodsAgo)
        {
            float a = (float)Decimal.Divide(2, 1 + periodsAgo);
            float part1 = 0.0f;
            float part2 = 0.0f;

            for (int xPeriods = 0; xPeriods < periodsAgo; xPeriods++)
            {
                part1 = part1 + (float)Math.Pow(1 - a, xPeriods) *
sec.getClose(id - xPeriods);
                part2 = part2 + (float)Math.Pow(1 - a, xPeriods);
            }
            return part1 / part2;
        }

        /// <summary>
        /// 3a. Standard Deviation of Close values. It is tested - OK.
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float STD(Security sec, int id, int timesAgo)
        {
            float SMAValue = SMA(sec, id, timesAgo);
            float retValue = 0.0f;

            for (int i = 0; i < timesAgo; i++)
            {
                retValue = retValue + (sec.getClose(id - i) - SMAValue) *
(sec.getClose(id - i) - SMAValue);
            }
            retValue = retValue / timesAgo;
            return (float)Math.Sqrt(retValue);
        }

        /// <summary>
        /// 3b. Standard Deviation for every type value (O, C, H, L, V). It is
tested - OK.
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float STD(Security sec, int id, int timesAgo, string
typeValue)
        {
            float SMAValue = SMA(sec, id, timesAgo, typeValue);
            float retValue = 0.0f;
```

```
            for (int i = 0; i < timesAgo; i++)
            {
                retValue = retValue + (sec.getValue(id - i, typeValue) -
SMAValue) * (sec.getValue(id - i, typeValue) - SMAValue);
            }
            retValue = retValue / timesAgo;
            return (float)Math.Sqrt(retValue);
        }


        /// <summary>
        /// 4. Relative Volatility Index (RVI)
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float RVI(Security sec, int id, int timesAgo)
        {
            float UPValues   = 0.0f;
            float DownValues = 0.0f;

            for (int i = 0; i < 14; i++)
            {
                if (sec.getClose(id - i) > sec.getClose(id - i - 1))
                {
                    UPValues = UPValues + STD(sec, id - i, timesAgo);
                }
                else if (sec.getClose(id - i)< sec.getClose(id - i - 1))
                {
                    DownValues = DownValues + STD(sec, id - i, timesAgo);
                }
            }

                UPValues = UPValues / 14;
                DownValues = DownValues / 14;

            return 100*UPValues / (UPValues + DownValues);
        }

        /// <summary>
        /// 5. Relative Volatility Index Advanced (RVIAdvanced)
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float RVIAdvanced(Security sec, int id, int timesAgo)
        {
            float UPHValues = 0.0f;
            float DownHValues = 0.0f;
            float UPLValues = 0.0f;
            float DownLValues = 0.0f;
```

```
            float RVIH = 0.0f;
            float RVIL = 0.0f;

            // calculate UPHValues & DownHValues
            for (int i = 0; i < 14; i++)
            {
                if (sec.getValue(id - i, "H") > sec.getValue(id - i - 1, "H"))
                {
                    UPHValues = UPHValues + STD(sec, id - i, timesAgo, "H");
                }
                else if (sec.getValue(id - i, "H") < sec.getValue(id - i - 1,
"H"))
                {
                    DownHValues = DownHValues + STD(sec, id - i, timesAgo, "H");
                }
            }
            // Finally Values
            UPHValues = UPHValues / 14;
            DownHValues = DownHValues / 14;

            // calculate UPLValues & DownLValues
            for (int i = 0; i < 14; i++)
            {
                if (sec.getValue(id - i, "L") > sec.getValue(id - i - 1, "L"))
                {
                    UPLValues = UPLValues + STD(sec, id - i, timesAgo, "L");
                }
                else if (sec.getValue(id - i, "L") < sec.getValue(id - i - 1,
"L"))
                {
                    DownLValues = DownLValues + STD(sec, id - i, timesAgo);
                }
            }
            // Finally Values
            UPLValues = UPLValues / 14;
            DownLValues = DownLValues / 14;

            RVIH = UPHValues / (UPHValues + DownHValues);
            RVIL = UPLValues / (UPLValues + DownLValues);

            return 100*(RVIH + RVIL)/ 2;
        }

        /// <summary>
        /// 6.Indicator: Commodity Channel Index (CCI)
        /// From oversold levels, a buy signal might be given when the CCI moves
back above -100.
        /// From overbought levels, a sell signal might be given when the CCI
moved back below +100.
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
```

```
        /// <returns></returns>
        public static float CCI(Security sec, int id, int timesAgo)
        {
            float ma_closeValues = SMA(sec, id, timesAgo);
            float D = 0.0f;
            for (int i = 0; i < timesAgo; i++)
            {
                D = D + Math.Abs(sec.getTP(id - i) - ma_closeValues);
            }

            // finally D=
            D = D / timesAgo;

            return (float) ((sec.getTP(id) - ma_closeValues) / (0.015 * D));
        }

        /* Category B: Momentum Indicators
         * Momentum is a general term used to describe the speed at which prices
move over a given time period.
         * Generally, changes in momentum tend to lead to changes in prices.
         */

        /// <summary>
        /// 7.Indicator: Price Oscillator
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="shortMAAvr"></param>
        /// <param name="longMAAvr"></param>
        /// <returns></returns>
        public static float PriceOsc(Security sec, int id, int shortMAAvr, int
longMAAvr)
        {
            return 100 * ((SMA(sec, id, shortMAAvr) - SMA(sec, id, longMAAvr)) /
SMA(sec, id, longMAAvr));
        }

        /// <summary>
        /// Moving Average of Price Oscillator.
        /// It is used for bying or selling signals.
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="shortAvr"></param>
        /// <param name="longAvr"></param>
        /// <param name="timesMA"></param>
        /// <returns></returns>
        public static float SMAPriceOsc(Security sec, int id, int shortAvr, int
longAvr, int timesMA)
        {
            float sumValue = 0;
            for (int i = 0; i < timesMA; i++)
            {
```

```
                sumValue = sumValue + PriceOsc(sec, id, shortAvr, longAvr);
            }
            return sumValue / timesMA;
        }

        /// <summary>
        /// 8.Indicator: MomentumOsc
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float MomentumOsc(Security sec, int id, int timesAgo)
        {
            return 100 * (sec.getClose(id) / sec.getClose(id - timesAgo));
        }

        /// <summary>
        /// 9.Indicator ROC (Rate of Change)
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float ROC(Security sec, int id, int timesAgo)
        {
            return 100 * ((sec.getClose(id) - sec.getClose(id - timesAgo)) /
sec.getClose(id - timesAgo));
        }

        /// <summary>
        /// Moving Average of ROC.
        /// It is used for bying or selling signals.
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <param name="timesMA"></param>
        /// <returns></returns>
        public static float SMAROC(Security sec, int id, int timesAgo, int
timesMA)
        {
            float sumValue = 0;
            for (int i = 0; i < timesMA; i++)
            {
                sumValue = sumValue + ROC(sec, id, timesAgo);
            }
            return sumValue / timesMA;
        }

        /// <summary>
        /// 10.Indicator: Relative Strength Index (RSI)
        /// </summary>
```

```
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float RSI(Security sec, int id, int timesAgo)
        {
            int ncountUpdays = 0;
            int ncountDowndays = 0;
            float sumValueUpdays = 0.0f;
            float sumValueDowndays = 0.0f;

            for (int i = 0; i < timesAgo; i++)
            {
                if (sec.getClose(id - i) > sec.getClose(id - i - 1))
                {
                    ncountUpdays++;
                    sumValueUpdays = sumValueUpdays + sec.getClose(id - i);
                }
                else
                {
                    ncountDowndays++;
                    sumValueDowndays = sumValueDowndays + sec.getClose(id - i);
                }
            }

            float RS = 0.0f;
            if (ncountUpdays == 0)
                RS = 00.0f;
            else if (ncountDowndays == 0)
                RS = 99.0f;
            else
                RS = (sumValueUpdays / ncountUpdays) / (sumValueDowndays /
ncountDowndays);

            return 100 - (100 / (1 + RS));
        }


        /// <summary>
        /// 11.Indicator: Stochastic Oscillator %K
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float StochOsc(Security sec, int id, int timesAgo)
        {
            return 100 * ((sec.getClose(id) - sec.getLowestLow(id, timesAgo)) /
                   (sec.getHighestHigh(id, timesAgo) - sec.getLowestLow(id,
timesAgo)));
        }

        /// <summary>
```

```
        /// Moving Average of StochRSI %D
        /// It is used for bying or selling signals.
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <param name="timesMA"></param>
        /// <returns></returns>
        public static float SMAStochOsc(Security sec, int id, int timesAgo, int
timesMA)
        {
            float sumValue = 0;
            for (int i = 0; i < timesMA; i++)
            {
                sumValue = sumValue + StochOsc(sec, id, timesAgo);
            }
            return sumValue / timesMA;
        }

        /// <summary>
        /// 12.Indicator: Williams %R
        /// Once a security becomes overbought or oversold,
        /// traders should wait for a signal that a price reversal has occurred.
        /// One method might be to wait for Williams %R to cross above or below
-50 for confirmation.
        /// Williams %R = (( HH - Today's Close ) / ( HH - LL )) * -100
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float Williams(Security sec, int id, int timesAgo)
        {
            return (-100) * ((sec.getHighestHigh(id, timesAgo) -
sec.getClose(id)) /
                        (sec.getHighestHigh(id, timesAgo) - sec.getLowestLow(id,
timesAgo)));
        }



        /// <summary>
        /// 13.Indicator Chande Momentum Oscillator
        /// Bullish signals are generated when the oscillator crosses above the
signal -50,
        /// and bearish signals are generated when the oscillator crosses down
through the signal +50
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float ChandeMomentumOsc(Security sec, int id, int
```

```
timesAgo)
        {
            float sumUp = 0.0f;
            float sumDown = 0.0f;

            for (int i = 0; i < timesAgo; i++)
            {
                if (sec.getClose(id - i) > sec.getClose(id - i - 1))
                {
                    sumUp = sumUp + sec.getClose(id - i);
                }
                else
                {
                    sumDown = sumDown + sec.getClose(id - i);
                }
            }

            return 100 * (sumUp - sumDown) / (sumUp + sumDown);
        }

        /* Category C: Market Strength Indicators (Volume Indicators)
         * Each indicator incorporate volume, which is the basic ingredient to
measurement of market strength.
         * Generally higher volume levels indicate more participants and
therefore more strength.
         */

        /// <summary>
        /// 14. Indicator: Moving Average of volume
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesMA"></param>
        /// <returns></returns>
        public static float SMAVolume(Security sec, int id, int timesMA)
        {
            float sumValue = 0;
            for (int i = 0; i < timesMA; i++)
            {
                sumValue = sumValue + sec.getVolume(id - i);
            }
            return sumValue / timesMA;
        }

        /// <summary>
        /// 16.Indicator: Rate of change volume
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float ROCVolume(Security sec, int id, int timesAgo)
        {
```

```
            return 100 * (sec.getVolume(id) - sec.getVolume(id - timesAgo)) /
sec.getVolume(id - timesAgo);
        }

        /// <summary>
        /// 15.Indicator: Volume Oscillator
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="shortMAAvr"></param>
        /// <param name="longMAAvr"></param>
        /// <returns></returns>
        public static float VolumeOsc(Security sec, int id, int shortMAAvr, int
longMAAvr)
        {
            return 100 * ((SMAVolume(sec, id, longMAAvr) - SMAVolume(sec, id,
shortMAAvr)) / SMAVolume(sec, id, longMAAvr));
        }

        /// <summary>
        /// 16. Indicator: Chaikin A/D
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="shortMAAvr"></param>
        /// <param name="longMAAvr"></param>
        /// <returns></returns>
        public static float ChaikinAD(Security sec, int id, int shortMAAvr, int
longMAAvr)
        {
            float chakin_short = 0.0f;
            float chakin_long  = 0.0f;

            for (int i = 0; i < shortMAAvr; i++)
            {
                chakin_short += sec.getVolume(id - i) * (2 * sec.getClose(id -
i) - sec.getLow(id - i) - sec.getHigh(id - i)) /

(sec.getHigh(id - i) - sec.getLow(id - i));
            }
            chakin_short = chakin_short / shortMAAvr;

            for (int i = 0; i < longMAAvr; i++)
            {
                chakin_long += sec.getVolume(id - i) * (2 * sec.getClose(id - i)
- sec.getLow(id - i) - sec.getHigh(id - i)) /

(sec.getHigh(id - i) - sec.getLow(id - i));
            }
            chakin_long = chakin_long / longMAAvr;

            // finally
            return (float) (chakin_short - chakin_long);
```

```
        }

        public static float SMAChaikinAD(Security sec, int id, int shortAvr, int
longAvr, int timesMA)
        {
            float sumValue = 0;
            for (int i = 0; i < timesMA; i++)
            {
                sumValue = sumValue + ChaikinAD(sec, id, shortAvr, longAvr);
            }
            return sumValue / timesMA;
        }

        /* Category D: Support and Resistance Indicators
         * A common occurrence is for prices to repeatedly rise or fall to a
certain level and then reverse.
         * This phenomenon (attributed to basic supply and demand) is called
support and resistance.
         */

        /// <summary>
        /// 17.Indicator: Projection Oscillator.
        /// Developed by Mel Widner, Ph.D., the Projection Oscillator is a by-
product of his Projection Bands.
        /// The Projection Oscillator is basically a slope-adjusted Stochastic.
Where the Stochastic Oscillator
        /// shows the relationship of the current price to its minimum and
maximum prices over a recent time period,
        /// the Projection Oscillator shows the same thing, but the minimum and
maximum prices are adjusted up/down
        /// by the slope of the price’s regression line.
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float ProjectionOsc(Security sec, int id, int timesAgo)
        {
            float slopeL = RegressionSlope(sec, id, timesAgo, "L");
            float slopeH = RegressionSlope(sec, id, timesAgo, "H");

            float minLBand = sec.getLow(id);
            float maxHBand = sec.getHigh(id);

            for (int i = 1; i < timesAgo; i++)
            {
                minLBand = Math.Min(minLBand, sec.getLow(id - i) + i * slopeL);
            }

            for (int i = 1; i < timesAgo; i++)
            {
                maxHBand = Math.Max(maxHBand, sec.getHigh(id - i) + i * slopeH);
            }
```

```
            return 100 * (sec.getClose(id) - minLBand) / (maxHBand - minLBand);
        }


        /* Category E: Trend Indicators
         * Trend is a term used to describe the persistence of prices to move in
one direction.
         */

        /// <summary>
        /// 18.Indicator: Regression Slope A, see ( Y = AX+B )
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <param name="typeValue"></param>
        /// <returns></returns>
        public static float RegressionSlope(Security sec, int id, int timesAgo,
string typeValue)
        {
            float avgX = (timesAgo+1)/2;
            float avgY = SMA(sec, id, timesAgo,typeValue);
            float sumA = 0.0f;
            float sumB = 0.0f;

            for (int i = 0; i < timesAgo; i++)
            {
                sumA = sumA + (i + 1 - avgX) * (sec.getValue(id - i, typeValue)
- avgY);
                sumB = sumA + (i + 1 - avgX) * (i + 1 - avgX);
            }

            return sumA / sumB;
        }

        /// <summary>
        /// Regression variable B , see ( Y = AX+B -> B = Y-AX )
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <param name="typeValue"></param>
        /// <returns></returns>
        public static float RegressionCofB(Security sec, int id, int timesAgo,
string typeValue)
        {
            float avgX = (timesAgo + 1) / 2;
            float avgY = SMA(sec, id, timesAgo, typeValue);

            return avgY - RegressionSlope(sec, id, timesAgo, typeValue) * avgX;
        }
```

```
        /// <summary>
        /// 19.Indicator: r2 (r-squared)
        /// if r2 is close to 1 indicates that shape is linear else
        /// if r2 is close to 0 indicates that shape is not linear
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <param name="typeValue"></param>
        /// <returns></returns>
        public static float R_Squared(Security sec, int id, int timesAgo, string
typeValue)
        {
            float avgX = (timesAgo + 1) / 2;
            float avgY = SMA(sec, id, timesAgo, typeValue);
            float sumA = 0.0f;
            float sumB = 0.0f;
            float sumC = 0.0f;

            for (int i = 0; i < timesAgo; i++)
            {
                sumA = sumA + (i + 1 - avgX) * (sec.getValue(id - i, typeValue)
- avgY);
                sumB = sumB + (i + 1 - avgX) * (i + 1 - avgX);
                sumC = sumC + (sec.getValue(id - i, typeValue) - avgY) *
(sec.getValue(id - i, typeValue) - avgY);
            }

            return (sumA * sumA) / (sumB * sumC);
        }

        /// <summary>
        /// 20,21,22.Indicators: DI-, DI+, DX
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns>It returns an array of three values representing the 3
indicators </returns>
        public static float[] DMI(Security sec, int id, int timesAgo)
        {
            float[] retValue = new float[3];

            float dm_plus  = 0.0f;
            float dm_minus = 0.0f;
            float tr = 0.0f;

            for (int i = 0; i < timesAgo; i++)
            {
                if (sec.getLow(id - i) < sec.getLow(id - i - 1))
                {
                    dm_minus += (sec.getLow(id - i - 1) - sec.getLow(id - i));
                }
```

```
                if (sec.getHigh(id - i) > sec.getHigh(id - i - 1))
                {
                    dm_plus += (sec.getHigh(id - i) - sec.getHigh(id - i - 1));
                }

                tr += sec.getTR(id-i);
            }

            retValue[0] = 100 * dm_minus / tr;
            retValue[1] = 100 * dm_plus / tr;
            retValue[2] = 100 * (retValue[1] - retValue[0]) / (retValue[0] +
retValue[1]);

            return retValue;
        }

        /// <summary>
        /// 23. Indicator: Vertical Horizontal Filter
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="id"></param>
        /// <param name="timesAgo"></param>
        /// <returns></returns>
        public static float VHF(Security sec, int id, int timesAgo)
        {
            float sumDiff = 0.0f;

            for (int i = 0; i < timesAgo; i++)
            {
                sumDiff += Math.Abs(sec.getClose(id - i) - sec.getClose(id - i -
1));
            }

            return (sec.getHighestClose(id, timesAgo) - sec.getLowestClose(id,
timesAgo)) / sumDiff;
        }
    }
}
```

Package (namespace) : Forms	**Source:** FormMain.cs

```
/// <summary>
/// Summary: description for FormMain class
/// ---------------------------------------------------------------
/// It contains the form class
/// ---------------------------------------------------------------
/// Purpose: Display main menu
/// </summary>
/// ---------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;


namespace AIPredictor.Forms
{
    public partial class FormMain : Form
    {
        //forms to be called from button clicks
        private FormPatterns formPatterns = new FormPatterns();

        private FormOptimazation formOptimazation = new FormOptimazation();

        private FormTrainNN formTrainNN = new FormTrainNN();

        private FormSimNeural formSimNeural = new FormSimNeural();

        /// <summary>
        /// The main constractor class for the menu form
        /// </summary>
        public FormMain()
        {
            InitializeComponent();
        }

        /// <summary>
        /// calls the form for creation the patterns
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_patterns_Click(object sender, EventArgs e)
        {
            formPatterns.ShowDialog();
        }
```

```
        /// <summary>
        /// calls the form for optimazation of trading systems
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_optimazation_Click(object sender, EventArgs e)
        {
            formOptimazation.ShowDialog();
        }

        /// <summary>
        /// Calls the form for tranning and testing the neural net
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_trainNN_Click(object sender, EventArgs e)
        {
            formTrainNN.ShowDialog();
        }

        /// <summary>
        /// calls automating procedure for training the neural net
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_simulateNeural_Click(object sender, EventArgs e)
        {
            formSimNeural.ShowDialog();
        }

        /// <summary>
        /// Exit from application
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_exit_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

Package (namespace) : Forms	**Source:** FormOptimization.cs

```
/// <summary>
/// Summary: description for FormOptimazation class
/// -----------------------------------------------------------------
/// It contains the form class
/// -----------------------------------------------------------------
/// Purpose: It is to optimize the input parameters of the techical indicators
///          for each stock separately. The parameters that produce the best
rewsults are saved
///          into sql server database
/// </summary>
/// -----------------------------------------------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using AIPredictor.Signals;
using AIPredictor.Trades;
using AIPredictor.BasicOper;

namespace AIPredictor.Forms
{
    public partial class FormOptimazation : Form
    {
        private DBLayer db;
        private TradeHolderList myTradeHList = new TradeHolderList();

        /// <summary>
        /// The main contractor class
        /// </summary>
        public FormOptimazation()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Optimizes the parameters of selected indicators and stocks
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_optimize_Click(object sender, EventArgs e)
        {
           db = new DBLayer();
           db.Connect();
           optimizeAll();
```

```
        }

        /// <summary>
        /// Loads the available technical indicators for optimazation
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_loadIndicators_Click(object sender, EventArgs e)
        {
            db = new DBLayer();
            db.Connect();

            listboxIndicators.Items.Clear();
            string[] myIndicators = db.LoadAllIndicators();
            for (int i = 0; i < myIndicators.Length; i++)
            {
                listboxIndicators.Items.Add(myIndicators[i]);
            }
        }

        /// <summary>
        /// Loads the available stocks from DB
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_loadStocks_Click(object sender, EventArgs e)
        {
            db = new DBLayer();
            db.Connect();

            listboxStocks.Items.Clear();
            string[] mystocks = db.LoadAllSymbols();
            for (int i = 0; i < mystocks.Length; i++)
            {
                listboxStocks.Items.Add(mystocks[i]);
            }
        }

        /// <summary>
        ///  Select all indicators
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_selAllIndicators_Click(object sender, EventArgs e)
        {
           for (int i = 0; i < listboxIndicators.Items.Count; i++)
            {
                listboxIndicators.SetItemChecked(i,true);
            }
        }

        /// <summary>
        /// unselect all indicators
```

```
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_unselIndicators_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < listboxIndicators.Items.Count; i++)
            {
                listboxIndicators.SetItemChecked(i, false);
            }
        }

        /// <summary>
        /// unselect all stocks
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_unselStocks_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < listboxStocks.Items.Count; i++)
            {
                listboxStocks.SetItemChecked(i, false);
            }
        }

        /// <summary>
        /// Select all stocks
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_selAllStocks_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < listboxStocks.Items.Count; i++)
            {
                listboxStocks.SetItemChecked(i, true);
            }
        }

        /// <summary>
        /// Optimazation and display of the results
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_optimizeResults_Click(object sender, EventArgs e)
        {
            DB_AITrade myAIDB = null;

            // Create DB_AITrade
            myAIDB = new DB_AITrade(myTradeHList);
            dataGridTrade.DataSource = myAIDB.TradeTable;
            dataGridTrade.ReadOnly = true;
        }

        /// <summary>
```

```
        /// See all results
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_allResults_Click(object sender, EventArgs e)
        {
            DB_AITrade myAIDB = null;

            // Create DB_AITrade
            myAIDB = new DB_AITrade(myTradeHList, true);
            dataGridTrade.DataSource = myAIDB.TradeTable;
            dataGridTrade.ReadOnly = true;
        }  // end of method

        /// <summary>
        /// Save data of optimized indicators in DB
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_transferInDB_Click(object sender, EventArgs e)
        {
            db.SaveTradeHolderList(myTradeHList);
        }

        /// <summary>
        /// Optimize the selected indicators
        /// </summary>
        private void optimizeAll()
        {
            if (listboxStocks.CheckedItems.Count > 0)
            {
                if (listboxIndicators.CheckedItems.Contains("SMA") == true)
                    optimazeSMA();

                if (listboxIndicators.CheckedItems.Contains("EMA") == true)
                    optimazeEMA();

                if (listboxIndicators.CheckedItems.Contains("RVI") == true)
                    optimazeRVI();

                if (listboxIndicators.CheckedItems.Contains("CCI") == true)
                    optimazeCCI();

                if (listboxIndicators.CheckedItems.Contains("PriceOsc") == true)
                    optimazePriceOsc();

                if (listboxIndicators.CheckedItems.Contains("Momentum") == true)
                    optimazeMomentum();

                if (listboxIndicators.CheckedItems.Contains("ROC") == true)
                    optimazeROC();

                if (listboxIndicators.CheckedItems.Contains("RSI") == true)
```

```
                    optimazeRSI();

                if (listboxIndicators.CheckedItems.Contains("StochOsc") == true)
                    optimazeStochOsc();

                if (listboxIndicators.CheckedItems.Contains("CMO") == true)
                    optimazeCMO();

                if (listboxIndicators.CheckedItems.Contains("Chaikin") == true)
                    optimazeChaikin();

                if (listboxIndicators.CheckedItems.Contains("DMI") == true)
                    optimazeDMI();

            }
            else
                MessageBox.Show("You have not selected stocks!", "Try again",
                            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }

        /// <summary>
        /// optimize SMA
        /// </summary>
        private void optimazeSMA()
        {
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;
            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(10, 30, 1, "parameter periods of days");

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                 symbolVal = listboxStocks.CheckedItems[i].ToString();

                 curSecurity = db.LoadSecurityFromDB(symbolVal);
                 db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                 execTd = new ExecTrade(curSecurity);
                 holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "SMA",
"long", optlist);

                 myTradeHList.addTradeHolderList(holdTrades);
            }
        }

        /// <summary>
        /// optimize EMA
        /// </summary>
        private void optimazeEMA()
        {
            string symbolVal = "";
            Security curSecurity;
```

```
            ExecTrade execTd;
            TradeHolderList holdTrades;
            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(10, 30, 1, "parameter periods of days");

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "EMA",
"long", optlist);

                myTradeHList.addTradeHolderList(holdTrades);
            }
        }

        /// <summary>
        /// optimize RVI
        /// </summary>
        private void optimazeRVI()
        {
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;

            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(8, 20, 1, "parameter periods of days");
            optlist.addOptRange(38, 70, 2, "parameter for buy signal");
            optlist.addOptRange(28, 60, 2, "parameter for sell signal");

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "RVI",
"long", optlist);

                myTradeHList.addTradeHolderList(holdTrades);
            }
        }

        /// <summary>
        /// optimize CCI
        /// </summary>
        private void optimazeCCI()
        {
```

```
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;

            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(8, 25, 1, "parameter periods of days");      //
should be 8 to 25 step 2
            optlist.addOptRange(-150, -70, 5, "parameter for buy signal");   //
should be -150 to -50 step 10
            optlist.addOptRange(70, 130, 5, "parameter for sell signal");    //
should be 50 to 150 step 10

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "CCI",
"long", optlist);

                myTradeHList.addTradeHolderList(holdTrades);
            }
        }

        /// <summary>
        /// optimize Price Oscilator
        /// </summary>
        private void optimazePriceOsc()
        {
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;

            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(5, 12, 1, "parameter periods of short days");
// should be 5 to 15 step 1
            optlist.addOptRange(15, 30, 1, "parameter periods of long days");
// should be 16 to 40 step 2
            optlist.addOptRange(4, 8, 1, "parameter periods of days for signal
MA");    // should be 4 to 10 step 1

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "PriceOsc",
```

```
"long", optlist);

                myTradeHList.addTradeHolderList(holdTrades);
            }

        }

        /// <summary>
        /// optimize Momentum
        /// </summary>
        private void optimazeMomentum()
        {
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;

            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(8, 25, 1, "parameter periods of days"); //
should be 6 to 20 step 2
            optlist.addOptRange(70, 130, 5, "parameter for buy signal");  //
should be 70 to 130 step 5
            optlist.addOptRange(70, 130, 5, "parameter for sell signal"); //
should be 70 to 130 step 5

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "Momentum",
"long", optlist);

                myTradeHList.addTradeHolderList(holdTrades);
            }
        }

        /// <summary>
        /// optimize ROC
        /// </summary>
        private void optimazeROC()
        {
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;

            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(8, 20, 1, "parameter periods of days"); //
should be 10 to 20 step 2
            optlist.addOptRange(-20, 20, 5, "parameter for buy signal");  //
```

```
should be -20 to 20 step 5
            optlist.addOptRange(-20, 20, 5, "parameter for sell signal"); //
should be -20 to 20 step 5

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "ROC",
"long", optlist);

                myTradeHList.addTradeHolderList(holdTrades);
            }
        }

        /// <summary>
        /// optimize RSI
        /// </summary>
        private void optimazeRSI()
        {
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;

            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(10, 20, 1, "parameter periods of days"); //
should be 10 to 20 step 2
            optlist.addOptRange(35, 50, 5, "parameter for buy signal");  //
should be 20 to 30 step 5
            optlist.addOptRange(60, 80, 5, "parameter for sell signal"); //
should be 60 to 80 step 5

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "RSI",
"long", optlist);

                myTradeHList.addTradeHolderList(holdTrades);
            }
        }

        /// <summary>
        /// optimize Stochastic Oscilator
        /// </summary>
```

```
        private void optimazeStochOsc()
        {
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;

            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(5, 15, 1, "parameter periods of days");
// should be 8 to 20 step 1
            optlist.addOptRange(3, 6, 1, "parameter periods of days for signal
MA");    // should be 4 to 10 step 1

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "StochOsc",
"long", optlist);

                myTradeHList.addTradeHolderList(holdTrades);
            }

        }

        /// <summary>
        /// optimize CMO
        /// </summary>
        private void optimazeCMO()
        {
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;

            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(6, 30, 1, "parameter periods of days"); //
should be 8 to 30 step 2
            optlist.addOptRange(-65, -30, 5, "parameter for buy signal");  //
should be -60 to -40 step 5
            optlist.addOptRange(0, 60, 5, "parameter for sell signal"); //
should be 4 0 to 60 step 5

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
```

```
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "CMO",
"long", optlist);

                myTradeHList.addTradeHolderList(holdTrades);
            }
        }

        /// <summary>
        ///  optimize Chaikin
        /// </summary>
        private void optimazeChaikin()
        {
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;

            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(2, 5, 1, "parameter periods of short days");
// should be 2 to 5 step 1
            optlist.addOptRange(8, 14, 1, "parameter periods of long days");
// should be 8 to 14 step 1
            optlist.addOptRange(4, 7, 1, "parameter periods of days for signal
MA");    // should be 4 to 8 step 1

            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "Chaikin",
"long", optlist);
                myTradeHList.addTradeHolderList(holdTrades);
            }
        }

        /// <summary>
        /// optimize DMI
        /// </summary>
        private void optimazeDMI()
        {
            string symbolVal = "";
            Security curSecurity;
            ExecTrade execTd;
            TradeHolderList holdTrades;

            OptRangeList optlist = new OptRangeList();
            optlist.addOptRange(10, 20, 1, "parameter periods of short days");
// should be 10 to 20 step 1
            optlist.addOptRange(18, 25, 1, "Value compared with DX");
// should be 24 step 1
```

```
                for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                symbolVal = listboxStocks.CheckedItems[i].ToString();

                curSecurity = db.LoadSecurityFromDB(symbolVal);
                db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
                execTd = new ExecTrade(curSecurity);
                holdTrades = execTd.OptimizeTrade(txtFromDate.Text, "DMI",
"long", optlist);
                myTradeHList.addTradeHolderList(holdTrades);
            }

        }

        /// <summary>
        /// Close form window
        /// </summary>
        private void btn_close_Click(object sender, EventArgs e)
        {
            Close();
        }

    }
}
```

Package (namespace) : Forms	**Source:** FormPatterns.cs

```
/// <summary>
/// Summary: description for FormPatterns class
/// -----------------------------------------------------------
/// It contains the form class
/// -----------------------------------------------------------
/// Purpose: 1. Load stock values from text file and transfer into DB
///          2. Creation of all type of patterns
/// </summary>
/// -----------------------------------------------------------

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.IO;
using System.Windows.Forms;
using AIPredictor.BasicOper;
using AIPredictor.Signals;

namespace AIPredictor.Forms
{
    public partial class FormPatterns : Form
    {
        private Security curSecurity, otherSec;
        private CandlePattern[] patternListOf2;
        private CandlePattern[] patternListOf3;
        private CandlePattern[] patternListOf4;

        private DBLayer db = new DBLayer();

        /// <summary>
        /// Constractor class
        /// </summary>
        public FormPatterns()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Select the path which contains the text files of stocks
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnSelectPath_Click(object sender, EventArgs e)
        {
            if (folderBrowserDlg.ShowDialog() == DialogResult.OK)
```

```
            {
                // determine whether fileName is a directory
                if (Directory.Exists(folderBrowserDlg.SelectedPath))
                {
                    // array for directories
                    string[] fileList;

                    // obtain file/directory list of specified directory
                    fileList =
Directory.GetFiles(folderBrowserDlg.SelectedPath);
                    // output directoryList contents
                    for (int i = 0; i < fileList.Length; i++)
                        listBoxSecurities.Items.Add(fileList[i]);
                }
            }
        }

        /// <summary>
        /// Load security from a text file
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnLoadSecurity_Click(object sender, EventArgs e)
        {
            curSecurity = new Security();

curSecurity.LoadStockDataFromFile(listBoxSecurities.SelectedItem.ToString());

curSecurity.AutoLoadSymbol(listBoxSecurities.SelectedItem.ToString());

            txtSymbol.Text = curSecurity.Symbol;
            txtName.Text = curSecurity.Name_of_symbol;
            txtCountry.Text = "US";

        }

        /// <summary>
        /// Create all type of patterns
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnCreatePatterns_Click(object sender, EventArgs e)
        {
            patternListOf2 = curSecurity.getPattern2List();
            patternListOf3 = curSecurity.getPattern3List();
            patternListOf4 = curSecurity.getPattern4List();
        }

        /// <summary>
        /// Save Security object values into Db
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
```

```
        private void btnExportSecurityToDB_Click(object sender, EventArgs e)
        {
            db.Connect();
            db.SaveSecurityInDB(curSecurity);
            db.SaveSecurityValuesInDB(curSecurity);
            db.Disconnect();
        }

        /// <summary>
        /// Save all created patterns into Db
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnExportPaqtternsToDB_Click(object sender, EventArgs e)
        {
            db.Connect();
            db.SavePatternOfTwoInDB(curSecurity);
            db.SavePatternOfThreeInDB(curSecurity);
            db.SavePatternOfFourInDB(curSecurity);
            db.Disconnect();
        }

        /// <summary>
        /// Save all selected stocks and their data into db
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnAllInDB_Click(object sender, EventArgs e)
        {
            db.Connect();

            for (int i = 0; i < listBoxSecurities.SelectedItems.Count; i++)
            {
                curSecurity = new Security();

curSecurity.LoadStockDataFromFile(listBoxSecurities.SelectedItems[i].ToString())
;

curSecurity.AutoLoadSymbol(listBoxSecurities.SelectedItems[i].ToString());
                curSecurity.Name_of_symbol = curSecurity.Symbol;
                curSecurity.Type_of_security = "stock";
                curSecurity.Country_origin = "USA";

                txtSymbol.Text = curSecurity.Symbol;
                txtName.Text = curSecurity.Name_of_symbol;
                txtCountry.Text = curSecurity.Country_origin;

                db.SaveSecurityInDB(curSecurity);
                db.SaveSecurityValuesInDB(curSecurity);

                db.SavePatternOfTwoInDB(curSecurity);
                db.SavePatternOfThreeInDB(curSecurity);
                db.SavePatternOfFourInDB(curSecurity);
```

```
                textBoxMsg.Text = "finish the " +
listBoxSecurities.SelectedItems[i].ToString();
            }

            db.Disconnect();
        }

        /// <summary>
        /// Loads security from Db
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnLoadSecFromDB_Click(object sender, EventArgs e)
        {
            db.Connect();
            otherSec = db.LoadSecurityFromDB("DIS");
            txtSymbol.Text = otherSec.Symbol;
            txtName.Text = otherSec.Name_of_symbol;
            txtCountry.Text = otherSec.Country_origin;

            db.LoadSecurityValuesFromDB(otherSec, "20000101");
            db.Disconnect();
        }

        /// <summary>
        /// Save the current changes of the specific displayed stock
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnPostChanges_Click(object sender, EventArgs e)
        {
            curSecurity.Symbol = txtSymbol.Text;
            curSecurity.Name_of_symbol = txtName.Text;
            curSecurity.Type_of_security =
(string)comboTypeOfSecurity.SelectedItem;
            curSecurity.Country_origin = txtCountry.Text;
        }

        /// <summary>
        /// Exit from form
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_close_Click(object sender, EventArgs e)
        {
            Close();
        }

    }
}
```

Package (namespace) : Forms	**Source:** FormSimNeural.cs

```
/// <summary>
/// Summary: description for FormSimNeural class
/// -----------------------------------------------------------------
/// It contains the FormSimNeural class
/// -----------------------------------------------------------------
/// Purpose: It is used to train automatically the neural net. All parameteres
///          are changed in random.
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using AIPredictor.BasicOper;
using AIPredictor.Trainning;
using AIPredictor.outputs;
using AIPredictor.SmartInput;

namespace AIPredictor.Forms
{
    public partial class FormSimNeural : Form
    {
        private DBLayer db;
        private TrainNN NN;
        private Security curSecurity;

        /// <summary>
        /// The constractor method
        /// </summary>
        public FormSimNeural()
        {
            InitializeComponent();
            db = new DBLayer();
        }

        /// <summary>
        /// Loads all available stocks from db
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void FormSimNeural_Load(object sender, EventArgs e)
        {
            db.Connect();
```

```
            listboxStocks.Items.Clear();
            string[] mystocks = db.LoadAllSymbols();
            for (int i = 0; i < mystocks.Length; i++)
            {
                listboxStocks.Items.Add(mystocks[i]);
            }
        }

        /// <summary>
        /// customize the neurons
        /// </summary>
        /// <param name="f"></param>
        /// <returns></returns>
        private int[] setTypeOfFunctions(int f)
        {
            int[] listFunctions = new int[5];

            if (f == 1)
            {
                listFunctions[0] = 1; // inputLinear
                listFunctions[1] = 2; // hid1Sigmoid
                listFunctions[2] = 2; // hid2Sigmoid
                listFunctions[3] = 2; // hid3Sigmoid
                listFunctions[4] = 3; // outputSine
            }
            else if (f == 2)
            {
                listFunctions[0] = 1; // inputLinear
                listFunctions[1] = 2; // hid1Sigmoid
                listFunctions[2] = 2; // hid2Sigmoid
                listFunctions[3] = 2; // hid3Sigmoid
                listFunctions[4] = 4; // outputLog
            }
            else if (f == 3)
            {
                listFunctions[0] = 2; // inputSigmoid
                listFunctions[1] = 2; // hid1Sigmoid
                listFunctions[2] = 2; // hid2Sigmoid
                listFunctions[3] = 3; // hid3Sine
                listFunctions[4] = 2; // outputSigmoid
            }
            else if (f == 4)
            {
                listFunctions[0] = 1; // inputLinear
                listFunctions[1] = 3; // hid1Sine
                listFunctions[2] = 3; // hid2Sine
                listFunctions[3] = 2; // hid3Sigmoid
                listFunctions[4] = 2; // outputSigmoid
            }
            else
            {
                listFunctions[0] = 2; // inputSigmoid
                listFunctions[1] = 2; // hid1Sigmoid
```

```
                listFunctions[2] = 2; // hid2Sigmoid
                listFunctions[3] = 2; // hid3Sigmoid
                listFunctions[4] = 2; // outputSigmoid
            }

            return listFunctions;
        }

        /// <summary>
        /// Select the basic indicators as input for the neural
        /// </summary>
        /// <param name="p"></param>
        /// <returns></returns>
        private Boolean[] setBasIndicators(int p)
        {
            Boolean[] listFunctions = new Boolean[8];

            if (p == 0)
            {
                for (int i = 0; i <= 7; i++)
                    listFunctions[i] = true;
            }
            else if (p == 1)
            {
                listFunctions[0] = true; // CloseRef
                listFunctions[1] = true; // CandleType
                listFunctions[2] = false; // Williams
                listFunctions[3] = false; // Volume
                listFunctions[4] = true; // VolOsc
                listFunctions[5] = true; // ProjOsc
                listFunctions[6] = true; // R_Sqrd
                listFunctions[7] = false; // VHF
            }
            else if (p == 2)
            {
                listFunctions[0] = true; // CloseRef
                listFunctions[1] = false; // CandleType
                listFunctions[2] = false; // Williams
                listFunctions[3] = false; // Volume
                listFunctions[4] = true; // VolOsc
                listFunctions[5] = true; // ProjOsc
                listFunctions[6] = false; // R_Sqrd
                listFunctions[7] = false; // VHF
            }
            else
            {
                listFunctions[0] = true; // CloseRef
                listFunctions[1] = false; // CandleType
                listFunctions[2] = false; // Williams
                listFunctions[3] = false; // Volume
                listFunctions[4] = true; // VolOsc
                listFunctions[5] = true; // ProjOsc
                listFunctions[6] = false; // R_Sqrd
```

```
                listFunctions[7] = true; // VHF
            }

            return listFunctions;
        }

        /// <summary>
        /// Train the neural. Repeat tranning.
        /// Customization is comopletely automatic and random
        /// </summary>
        /// <param name="symbol_str"></param>
        private void runSimForStock(string symbol_str)
        {
            curSecurity = db.LoadSecurityFromDB(symbol_str);
            db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
            int hiddenLayers = 0;
            int numberOfBasInd = 0;
            int countAllRepeats = 0;

            for (int i = 0; i < 2; i++)  // represents hidden layers
            {
                for (int j = 2; j < 6; j++) // less Neurons
                {
                    for (int k = 100; k <= 3000; k = k * 3) //training Cycles
                    {
                        for (int p = 0; p < 3; p++) // number Of Basic
Indicators
                        {
                            for (int h = 10; h < 31; h = h + 10) // valid
Performance
                            {
                                for (int g = 1; g < 5; g++)  // out Put
Selection
                                {
                                    for (int f = 1; f < 7; f++)  // repeat again
for getting better results
                                    {
                                        countAllRepeats = countAllRepeats + 1;
                                        if (i == 0) // 1 hidden layer
                                        {
                                            hiddenLayers = 1; // 1 hidden Layer
                                            NN = new TrainNN(curSecurity, db,
300, k, j, setTypeOfFunctions(f), setBasIndicators(p), false,

Convert.ToDouble(h));
                                        }
                                        else
                                        {
                                            hiddenLayers = 3;  // 3 hidden
Layers
                                            NN = new TrainNN(curSecurity, db,
300, k, setTypeOfFunctions(f), setBasIndicators(p), false,
```

```
Convert.ToDouble(h));
                                        }

                                        if (f < 3)
                                        {
                                            NN.setNetworkProperties(870, 943,
1.01, 0.99);
                                            NN.setLayerProperties(0.3, 0.01,
0.3, 0.01, 0.003);
                                        }
                                        else
                                        {
                                            NN.setNetworkProperties(870, 943,
1.015, 0.98);
                                            NN.setLayerProperties(0.28, 0.01,
0.29, 0.01, 0.003);
                                        }

                                        NN.execTrainning(g);

                                        NN.Save(curSecurity.Name_of_symbol +
countAllRepeats.ToString()+".net");

                                        NN.testNN(g);
                                        NN.maximizePerformance();

                                        if (p==0)
                                            numberOfBasInd = 8;
                                        else if (p==1)
                                            numberOfBasInd = 5;
                                        else if (p == 2)
                                            numberOfBasInd = 3;
                                        else
                                            numberOfBasInd = 4;

                                        db.SaveSimResult(symbol_str,
hiddenLayers, j, k, numberOfBasInd, Convert.ToDouble(h), g,
                                            NN.BuyWhen, NN.SellWhen,
NN.PercentangePredictions, NN.PercentageOfSuccessTrades,
                                            NN.PercentageOfUnSuccessTrades, 0.88
* NN.calcPerformance(NN.BuyWhen, NN.SellWhen)); // 264.0 / 300 periods = 0.88;


                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

        /// <summary>
```

```
        /// run training for the selected stocks
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_run_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                string symbolVal = listboxStocks.CheckedItems[i].ToString();
                runSimForStock(symbolVal);

            }
        }

        /// <summary>
        /// Automitic training. The resuls are saved into db
        /// </summary>
        /// <param name="symbol_str"></param>
        private void runSimRandomForStock(string symbol_str)
        {
            curSecurity = db.LoadSecurityFromDB(symbol_str);
            db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);
            int hiddenLayers = 0;
            int numberOfBasInd = 0;
            int countAllRepeats = 0;
            int countTotalTradesPrediction = 0;
            int countAnnualNetPerformance = 0;
            double maxTotalTradesPrediction = 0;
            double maxAnnualNetPerformance = 0;
            Random rd = new Random();
            Boolean continueLoop = true;

            // selection number of hidden neurons
            while (continueLoop)
            {
                countAllRepeats = countAllRepeats + 1;
                countTotalTradesPrediction = countTotalTradesPrediction + 1;
                countAnnualNetPerformance = countAnnualNetPerformance + 1;

                // selection number of hidden neurons
                int neuronsNum = rd.Next(10);
                int trainingCycles = rd.Next(80, 1200);
                int p = rd.Next(0, 4);
                int lessNeurons = rd.Next(7);
                double validPerf = Convert.ToDouble(rd.Next(5, 35));
                int g = rd.Next(1, 5);

                if (neuronsNum < 8)  // select 1 hidden layer
                {
                    hiddenLayers = 1;
                    NN = new TrainNN(curSecurity, db, 300, trainingCycles,
lessNeurons, setTypeOfFunctions(rd.Next(1, 6)),
                                     setBasIndicators(p), false, validPerf);
```

```
                }
                else // 3 hidden Layers
                {
                    hiddenLayers = 3;
                    NN = new TrainNN(curSecurity, db, 300, trainingCycles,
setTypeOfFunctions(rd.Next(1, 6)), setBasIndicators(p), false,
                                                                validPerf);
                }

                int f = rd.Next(5);
                if (f < 3)
                {
                    NN.setNetworkProperties(rd.Next(700, 900), rd.Next(850,
1000), 1.01, 0.99);
                    NN.setLayerProperties(0.3, 0.01, 0.3, 0.01, 0.003);
                }
                else
                {
                    NN.setNetworkProperties(rd.Next(700, 900), rd.Next(850,
1000), 1.015, 0.98);
                    NN.setLayerProperties(0.28, 0.01, 0.29, 0.01, 0.003);
                }

                NN.execTrainning(g);

                NN.testNN(g);
                NN.maximizePerformance();

                if (p == 0)
                    numberOfBasInd = 8;
                else if (p == 1)
                    numberOfBasInd = 5;
                else if (p==2)
                    numberOfBasInd = 3;
                else
                    numberOfBasInd = 4;

                if (NN.PercentangePredictions > maxTotalTradesPrediction)
                {
                    maxTotalTradesPrediction = NN.PercentangePredictions;
                    countTotalTradesPrediction = 0;
                }

                if (NN.calcPerformance(NN.BuyWhen, NN.SellWhen) >
maxAnnualNetPerformance)
                {
                    maxAnnualNetPerformance = NN.PercentangePredictions;
                    countAnnualNetPerformance = 0;
                }

                //save neural to file
                NN.Save(curSecurity.Name_of_symbol + countAllRepeats.ToString()
+ ".net");
```

```
                // save to database
                db.SaveSimResult(symbol_str, hiddenLayers, lessNeurons,
trainingCycles, numberOfBasInd, validPerf, g,
                    NN.BuyWhen, NN.SellWhen, NN.PercentangePredictions,
NN.PercentageOfSuccessTrades,
                    NN.PercentageOfUnSuccessTrades, 0.88 *
NN.calcPerformance(NN.BuyWhen, NN.SellWhen)); // 264.0 / 300 periods = 0.88;

                //exit conditions
                if ((countTotalTradesPrediction > 50) &&
(countAnnualNetPerformance > 50))
                {
                    continueLoop = false;  //no further improvement of results
                }

                if (countAllRepeats > 501)
                {
                    continueLoop = false;  // stop It's enough
                }
            }
        }

        /// <summary>
        /// random execution of automatic training
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_runRandom_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < listboxStocks.CheckedItems.Count; i++)
            {
                string symbolVal = listboxStocks.CheckedItems[i].ToString();
                runSimRandomForStock(symbolVal);

            }
        }

        /// <summary>
        /// exit
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_exit_Click(object sender, EventArgs e)
        {
            Close();
        }


    }
}
```

Package (namespace) : Forms	**Source:** FormTrainNN.cs

```
/// <summary>
/// Summary: description for FormSimNeural class
/// -----------------------------------------------------------------
/// It contains the FormSimNeural class
/// -----------------------------------------------------------------
/// Purpose: It is used to train the neural net. It contains a lot of
parameteres
///          that can be changed. It will create about 6912 different neurals.
///          Also, after tranning you can test the
///          performance of the network.
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using AIPredictor.BasicOper;
using AIPredictor.Trainning;
using AIPredictor.outputs;
using AIPredictor.SmartInput;


namespace AIPredictor.Forms
{
    public partial class FormTrainNN : Form
    {
        private DBLayer db = new DBLayer();
        private TrainNN NN;
        private Security curSecurity;

        /// <summary>
        /// Basic contractor method
        /// </summary>
        public FormTrainNN()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Train the neural. Repeat tranning.
        /// It will create about 6912 different neurals. Some of them
        /// have excellent results. This procedures is used for optimazation
        /// of the neural. We should select the neural produced the best resuls
```

```
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_trainNN_Click(object sender, EventArgs e)
        {
            db.Connect();
            DB_TrainNN trainInputsDB = null;

            int ntypeOutput = 0;

            if (radioBut1.Checked)
                ntypeOutput = 1;
            else if (radioBut2.Checked)
                ntypeOutput = 2;
            else if (radioBut3.Checked)
                ntypeOutput = 3;
            else
                ntypeOutput = 4;

            string symbolVal =
comboBox_stock.Items[comboBox_stock.SelectedIndex].ToString();
            curSecurity = db.LoadSecurityFromDB(symbolVal);
            db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);

            // Two general type of neurals (1 hidden or 3 hiddenlayers)
            if (radioLayers1.Checked)
            {
                // 1 hidden Layer Boolean[] selBasIndicators, Boolean
stockValues, double nAnnualPerformance
                NN = new TrainNN(curSecurity, db,
Convert.ToInt32(txt_lastdaysTest.Text),
Convert.ToInt32(txt_numberOfCycles.Text),
                                 Convert.ToInt32(txt_lessDim.Text),
getTypeOfFunctions(), selectBasIndicators(), checkBox_stocks.Checked,
                                 Convert.ToDouble(txt_annualLimit.Text));
            }
            else
            {
                // 3 hidden Layers
                NN = new TrainNN(curSecurity, db,
Convert.ToInt32(txt_lastdaysTest.Text),
Convert.ToInt32(txt_numberOfCycles.Text),
                                 getTypeOfFunctions(), selectBasIndicators(),
checkBox_stocks.Checked,
                                 Convert.ToDouble(txt_annualLimit.Text));
            }

            NN.setNetworkProperties(txt_jitter.Text, txt_annealing.Text,
txt_increaseFact.Text, txt_decreaseFact.Text);
            NN.setLayerProperties(txt_inputLearningRate.Text, txt_inputMom.Text,
txt_hidLearningRate.Text, txt_hidMom.Text, txt_maxOutputJitter.Text);
```

```
            trainInputsDB = NN.execTrainning(ntypeOutput);

            dataGridTrade.DataSource = trainInputsDB.TrainNNTable;
            dataGridTrade.ReadOnly = true;
        }

        /// <summary>
        /// Test the neural net performance. It should be first train the neural
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>

        private void btn_output_Click(object sender, EventArgs e)
        {
            db.Connect();
            DB_TrainNN testOutputsDB = null;

            int ntypeOutput = 0;
            if (radioBut1.Checked)
                ntypeOutput = 1;
            else if (radioBut2.Checked)
                ntypeOutput = 2;
            else if (radioBut3.Checked)
                ntypeOutput = 3;
            else
                ntypeOutput = 4;

            string symbolVal =
comboBox_stock.Items[comboBox_stock.SelectedIndex].ToString();
            curSecurity = db.LoadSecurityFromDB(symbolVal);
            db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);

            testOutputsDB = NN.testNN(ntypeOutput);

            dataGridResults.DataSource = testOutputsDB.TrainNNTable;
            dataGridResults.ReadOnly = true;

            btn_updatePerformance_Click(sender, e);

            txt_predBuy.Text = NN.PercentageOfSuccessTrades.ToString();
            txt_predSell.Text = NN.PercentageOfUnSuccessTrades.ToString();
            txt_predTotal.Text = NN.PercentangePredictions.ToString();
        }

        /// <summary>
        /// Load all stocks
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void FormTrainNN_Load(object sender, EventArgs e)
        {
            db.Connect();
```

```
            comboBox_stock.Items.Clear();
            string[] mystocks = db.LoadAllSymbols();
            for (int i = 0; i < mystocks.Length; i++)
            {
                comboBox_stock.Items.Add(mystocks[i]);
            }
        }

        /// <summary>
        /// Displays the best performance results independent of the indicators.
        /// The performance is based on specific 4 methods (outputs)
        /// </summary>
        private void calcDesiredResults()
        {
            string symbolVal =
comboBox_stock.Items[comboBox_stock.SelectedIndex].ToString();
            curSecurity = db.LoadSecurityFromDB(symbolVal);
            db.LoadSecurityValuesFromDB(curSecurity, txtFromDate.Text);


            OutputSignal s = new OutputSignal(curSecurity,
Convert.ToInt32(txt_lastdaysTest.Text));
            Double ratioAnnual = 264.0 /
Convert.ToDouble(txt_lastdaysTest.Text);

            // all periods are taken into account
            txt_desOutput1.Text = (s.calcPerformance(1, 0) *
ratioAnnual).ToString();
            txt_desOutput2.Text = (s.calcPerformance(2, 0) *
ratioAnnual).ToString();
            txt_desOutput3.Text = (s.calcPerformance(3, 0) *
ratioAnnual).ToString();
            txt_desOutput4.Text = (s.calcPerformance(4, 0) *
ratioAnnual).ToString();

            // only last periods are taken into account
            txt_desOutputLast1.Text = (s.calcPerformance(1, 2) *
ratioAnnual).ToString();
            txt_desOutputLast2.Text = (s.calcPerformance(2, 2) *
ratioAnnual).ToString();
            txt_desOutputLast3.Text = (s.calcPerformance(3, 2) *
ratioAnnual).ToString();
            txt_desOutputLast4.Text = (s.calcPerformance(4, 2) *
ratioAnnual).ToString();
        }

        /// <summary>
        /// Displays the performance of technical indicators
        /// for the specific selected stock (symbol)
        /// </summary>
        private void showIndicatorPerformance()
        {
            List<IndicatorOptParam> myListOptParams = new
```

```
List<IndicatorOptParam>();

            db.Connect();
            myListOptParams =
db.LoadAITradeOptValues(comboBox_stock.Items[comboBox_stock.SelectedIndex].ToStr
ing());

            DataTable indicatorTable = new DataTable("IndicatorsPerf");
            indicatorTable.Columns.Add("Indicator");
            indicatorTable.Columns.Add("AnnualPerformance");

            DataRow fileRow;
            for (int i = 0; i < myListOptParams.Count - 1; i++)
            {
                fileRow = indicatorTable.NewRow();
                // Fill row with data
                fileRow["Indicator"] = myListOptParams[i].CodeInd;
                fileRow["AnnualPerformance"] =
myListOptParams[i].AnnualPerformance;

                indicatorTable.Rows.Add(fileRow);
            }

            dataGridInd.DataSource = indicatorTable;
        }

        /// <summary>
        /// customization of function of neurons
        /// </summary>
        /// <returns></returns>
        private int[] getTypeOfFunctions()
        {
            int[] listFunctions = new int[5];

            // set up input Layer
            if (radio_inputLinear.Checked)
                listFunctions[0] = 1;
            else if (radio_inputSigmoid.Checked)
                listFunctions[0] = 2;
            else if (radio_inputSine.Checked)
                listFunctions[0] = 3;
            else
                listFunctions[0] = 4;

            // set up hidden 1 Layer
            if (radio_hid1Linear.Checked)
                listFunctions[1] = 1;
            else if (radio_hid1Sigmoid.Checked)
                listFunctions[1] = 2;
            else if (radio_hid1Sine.Checked)
                listFunctions[1] = 3;
            else
                listFunctions[1] = 4;
```

```
            // set up hidden 2 Layer
            if (radio_hid2Linear.Checked)
                listFunctions[2] = 1;
            else if (radio_hid2Sigmoid.Checked)
                listFunctions[2] = 2;
            else if (radio_hid2Sine.Checked)
                listFunctions[2] = 3;
            else
                listFunctions[2] = 4;

            // set up hidden 3 Layer
            if (radio_hid3Linear.Checked)
                listFunctions[3] = 1;
            else if (radio_hid3Sigmoid.Checked)
                listFunctions[3] = 2;
            else if (radio_hid3Sine.Checked)
                listFunctions[3] = 3;
            else
                listFunctions[3] = 4;

            // set up output Layer
            if (radio_outputLinear.Checked)
                listFunctions[4] = 1;
            else if (radio_outputSigmoid.Checked)
                listFunctions[4] = 2;
            else if (radio_outputSine.Checked)
                listFunctions[4] = 3;
            else
                listFunctions[4] = 4;

            return listFunctions;
        }

        /// <summary>
        /// Returns an array representing which basic
        /// indicators will be used as inputs
        /// </summary>
        /// <returns></returns>
        private Boolean[] selectBasIndicators()
        {
            Boolean[] listFunctions = new Boolean[8];
            for (int i = 0; i <= 7; i++)
                listFunctions[i] = false;

            if (checkedListIndicators.CheckedItems.Contains("CloseRef") == true)
                listFunctions[0] = true;

            if (checkedListIndicators.CheckedItems.Contains("CandleType") ==
true)
                listFunctions[1] = true;
            if (checkedListIndicators.CheckedItems.Contains("Williams") == true)
                listFunctions[2] = true;
```

```
            if (checkedListIndicators.CheckedItems.Contains("Volume") == true)
                listFunctions[3] = true;

            if (checkedListIndicators.CheckedItems.Contains("VolOsc") == true)
                listFunctions[4] = true;

            if (checkedListIndicators.CheckedItems.Contains("ProjOsc") == true)
                listFunctions[5] = true;

            if (checkedListIndicators.CheckedItems.Contains("R_Sqrd") == true)
                listFunctions[6] = true;

            if (checkedListIndicators.CheckedItems.Contains("VHF") == true)
                listFunctions[7] = true;

            return listFunctions;
        }

        private void comboBox_stock_SelectedIndexChanged(object sender,
EventArgs e)
        {
            calcDesiredResults();
            showIndicatorPerformance();
        }
       /// <summary>
       /// Exit
       /// </summary>
       /// <param name="sender"></param>
       /// <param name="e"></param>
        private void btn_exit_Click(object sender, EventArgs e)
        {
            Close();
        }
        /// <summary>
        /// Displays the performance of the neural based on selected values
        /// which define the points for buy or sell
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btn_updatePerformance_Click(object sender, EventArgs e)
        {
            Double ratioAnnual = 264.0 /
Convert.ToDouble(txt_lastdaysTest.Text);
            double netPerf =
NN.calcPerformance(Convert.ToDouble(txt_thresBuy.Text),
Convert.ToDouble(txt_thresSell.Text));
            txt_netoutput.Text = netPerf.ToString();
            txt_netDesiredoutput.Text = (netPerf * ratioAnnual).ToString();
        }
    }
```

Package (namespace) : output	**Source:** `OutputSignal.cs`

```
/// <summary>
/// Summary: description for OutputSignal class
/// -----------------------------------------------------------
/// It contains the OutputSignal class
/// -----------------------------------------------------------
/// Purpose: It is used to calculate the maximum results
/// of hypothetical training. It should bring the best results
/// for trading. It implements 4 differents methods of estimation.
/// </summary>
/// -----------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using AIPredictor.BasicOper;

namespace AIPredictor.outputs
{
    class OutputSignal
    {
        private Security curSecurity;
        private int lastDays;

        /// <summary>
        /// Constractor method
        /// </summary>
        /// <param name="sec"></param>
        public OutputSignal(Security sec)
        {
            this.curSecurity = sec;
            this.lastDays = 0;
        }

        /// <summary>
        /// constractor method. Additionally defines the last data
        /// to calculate the max performance
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="lastDaysIn"></param>
        public OutputSignal(Security sec, int lastDaysIn)
        {
            this.curSecurity = sec;
            this.lastDays = lastDaysIn;
        }
        /// <summary>
        /// The signal is dependend on the next two days
        /// </summary>
        /// <returns></returns>
```

```
        private double[] getOutputType1()
        {
            int nCount = curSecurity.CountValues();
            double[] retArray = new double[nCount];
            for (int i = 0; i < nCount - 1; i++)
                retArray[i] = 0;

            for (int i = 1; i < nCount-2; i++)
            {
                if ((curSecurity.getClose(i) > curSecurity.getClose(i - 1)) &&
(curSecurity.getClose(i + 1) > curSecurity.getClose(i)))
                {
                    retArray[i - 1] = 1;
                    retArray[i] = 1;
                    retArray[i + 1] = 1;
                }
            }
            return retArray;
        }

        /// <summary>
        ///  It simulates what happens next day (stock will increase or not)
        /// </summary>
        /// <returns></returns>
        private double[] getOutputType2()
        {
            int nCount = curSecurity.CountValues();
            double[] retArray = new double[nCount];
            for (int i = 0; i < nCount - 1; i++)
                retArray[i] = 0;

            for (int i = 0; i < nCount-1; i++)
            {
                if (curSecurity.getClose(i) < curSecurity.getClose(i+1))
                {
                    retArray[i] = 1;
                }
            }

            return retArray;
        }

        private double[] getOutputType3()
        {
            int nCount = curSecurity.CountValues();
            double[] retArray = new double[nCount];

            for (int i = 0;i<nCount-1;i++)
                retArray[i] = 0; // default at beginning

            Boolean isInMarket = false;

            for (int i = 1; i < nCount - 2; i++)
```

```
            {
                if (isInMarket)  // in the market
                {
                    if (curSecurity.getClose(i+1) > curSecurity.getClose(i))
                    {
                        retArray[i] = 1;
                    }
                    else
                    {
                        if (curSecurity.getClose(i + 2) -
curSecurity.getClose(i) > 0)
                        {
                            retArray[i] = 1;
                        }
                        else
                        {
                            isInMarket = false;
                        }
                    }
                }
                else  // out of market
                {
                    if (curSecurity.getClose(i) > curSecurity.getClose(i - 1))
                    {
                        // check next day
                        if (curSecurity.getClose(i + 1) >
curSecurity.getClose(i))
                        {
                            if (curSecurity.getClose(i + 2) -
curSecurity.getClose(i) > 0)
                            {
                                retArray[i] = 1;
                                isInMarket = true;
                            }
                        }
                    }
                }
            }

            return retArray;
        }

        private double[] getOutputType4()
        {
            int nCount = curSecurity.CountValues();
            double[] retArray = new double[nCount];
            for (int i = 0; i < nCount - 1; i++)
                retArray[i] = 0;

            for (int i = 1; i < nCount - 1; i++)
            {
                if (curSecurity.getClose(i) > curSecurity.getClose(i - 1))
                {
```

```
                if (curSecurity.getClose(i+1) > curSecurity.getClose(i))
                retArray[i] = 1;
            }
        }

        return retArray;
    }

    public double[] getOutput(int typeOfOutput)
    {
        if (typeOfOutput == 1)
            return getOutputType1();
        else if (typeOfOutput == 2)
            return getOutputType2();
        else if (typeOfOutput == 3)
            return getOutputType3();
        else
            return getOutputType4();
    }

    public double calcPerformance(int typeOfOutput, int typeOfRange)
    {
        double[] signalList;

        if (typeOfOutput == 1)
            signalList = getOutputType1();
        else if (typeOfOutput == 2)
            signalList = getOutputType2();
        else if (typeOfOutput == 3)
            signalList = getOutputType3();
        else
            signalList = getOutputType4();

        int nCount = curSecurity.CountValues();

        int startCounting = 1;
        int endCounting = nCount - 1;

        if (typeOfRange == 1)
        {
            endCounting = endCounting - lastDays;
        }
        else if (typeOfRange == 2)
        {
            startCounting = endCounting - lastDays;
        }

        int startType = 0;  // 0 = out of market, 1 = exec position
        int posEntrance = -1;
        int posExit = -1;

        double win_loss = 0; // per stock
```

```
            for (int i = startCounting; i < endCounting; i++)
            {
                if ((signalList[i] == 1) && (signalList[i - 1] == 0))
                {
                    posEntrance = i;
                }

                if ((signalList[i] == 0) && (signalList[i - 1] == 1))
                {
                    posExit = i;
                    startType = 1;
                }

                if (startType == 1)
                {
                    startType = 0;  // reset
                    win_loss = win_loss + (curSecurity.getClose(posExit) -
curSecurity.getClose(posEntrance));
                }
            }

            return win_loss;
        }


    }
}
```

Package (namespace) : `Signal`	**Source:** `Signal.cs`

```
/// <summary>
/// Summary: description for Signals class
/// ---------------------------------------------------------------
/// It contains the Signals class
/// ---------------------------------------------------------------
/// Purpose: Object holder for signals producing by a trading system
/// </summary>
/// ---------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using AIPredictor.BasicOper;

namespace AIPredictor.Signals
{
    class Signal
    {
        private Date buyDate;
        private Date sellDate;
```

```
        private float buyClose;
        private float sellClose;

        /// <summary>
        /// Constractor method
        /// </summary>
        /// <param name="buyDateStr"></param>
        /// <param name="sellDateStr"></param>
        /// <param name="buyCloseValue"></param>
        /// <param name="sellCloseValue"></param>
        public Signal(string buyDateStr, string sellDateStr, float
buyCloseValue, float sellCloseValue)
        {
            setBuyDate(buyDateStr);
            setSellDate(sellDateStr);
            this.buyClose  = buyCloseValue;
            this.sellClose = sellCloseValue;
        }

        /// <summary>
        /// Constractor method
        /// </summary>
        /// <param name="buyDateStr"></param>
        /// <param name="buyCloseValue"></param>
        public Signal(string buyDateStr, float buyCloseValue)
        {
            setBuyDate(buyDateStr);
            setSellDate("");
            this.buyClose = buyCloseValue;
            this.sellClose = 0;
        }

        /// <summary>
        /// default values
        /// </summary>
        public Signal()
        {
            setBuyDate("");
            setSellDate("");
            this.buyClose  = 0;
            this.sellClose = 0;
        }

        public void SetBuySignal(string buyDateStr, float buyCloseValue)
        {
            setBuyDate(buyDateStr);
            this.buyClose = buyCloseValue;
        }

        public void SetSellSignal(string sellDateStr, float sellCloseValue)
        {
            setSellDate(sellDateStr);
            this.sellClose = sellCloseValue;
```

```
        }

        // public property BuyClose
        public float BuyClose
        {
            get
            {
                return buyClose;
            }
            set
            {
                buyClose = value;
            }
        }

        // public property SellClose
        public float SellClose
        {
            get
            {
                return sellClose;
            }
            set
            {
                sellClose = value;
            }
        }

        // public property EarnMoney
        public float EarnMoney
        {
            get
            {
                return sellClose - buyClose;
            }
        }

        // public property BuyDate
        public string BuyDate
        {
            get
            {
                return buyDate.ToDateString();
            }
        }

        /// <summary>
        /// Set buyDate value
        /// </summary>
        /// <param name="strDate"></param>
        public void setBuyDate(string strDate)
        {
            this.buyDate = Date.ConvetStringToDate(strDate);
```

```
        }

        // public property SellDate
        public string SellDate
        {
            get
            {
                return sellDate.ToDateString();
            }
        }

        /// <summary>
        /// Set sellDate value
        /// </summary>
        /// <param name="strDate"></param>
        public void setSellDate(string strDate)
        {
            this.sellDate = Date.ConvetStringToDate(strDate);
        }

    }
}
```

Package (namespace) : Signal	**Source:** DB_AITrade.cs

```
/// <summary>
/// Summary: description for tradeTable class
/// -----------------------------------------------------------
/// It contains the tradeTable class
/// -----------------------------------------------------------
/// Purpose: DataTable to hold the information of results of trading.
///          Trading results can be displayed
/// </summary>
/// -----------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Collections;

namespace AIPredictor.Signals
{
    class DB_AITrade
    {
        // The private property is a DataTable Object
        private DataTable tradeTable;

        /// <summary>
        /// constractor class
        /// </summary>
        /// <param name="myTradeHList"></param>
        /// <param name="loadAllDetails"></param>
        public DB_AITrade(TradeHolderList myTradeHList, Boolean loadAllDetails)
        {
            if (loadAllDetails)
                this.tradeTable = LoadInDBAllTrades(myTradeHList);
            else
                this.tradeTable = LoadInDBBestTrades(myTradeHList);
        }

        public DB_AITrade(TradeHolderList myTradeHList)
        {
            this.tradeTable = LoadInDBBestTrades(myTradeHList);
        }

        // The public property TradeTable
        public DataTable TradeTable
        {
            get
            {
                return tradeTable;
            }
```

```
            set
            {
                tradeTable = value;
            }
        }

        private DataTable CreateDBTrade()
        {
            DataTable trTable = new DataTable("tradeDB");

            trTable.Columns.Add("symbol");
            trTable.Columns.Add("code of Indicator");
            trTable.Columns.Add("type of trade");
            trTable.Columns.Add("from Date Applied");
            trTable.Columns.Add("to Date Applied");
            trTable.Columns.Add("number Of Stocks");
            trTable.Columns.Add("fee Per Trade");
            trTable.Columns.Add("total Earning");
            trTable.Columns.Add("total Trades");
            trTable.Columns.Add("success Trades");
            trTable.Columns.Add("unSuccess Trades");
            trTable.Columns.Add("performance");
            trTable.Columns.Add("annual Performance");
            trTable.Columns.Add("parameter Desc");
            trTable.Columns.Add("parameter 1");
            trTable.Columns.Add("parameter 2");
            trTable.Columns.Add("parameter 3");

            return trTable;
        }

        public DataTable LoadInDBBestTrades(TradeHolderList myTradeHList)
        {
            DataTable trTable = CreateDBTrade();
            DataRow fileRow;

            for (int i = 0; i < myTradeHList.getTradeHoldersSum().Count; i++)
            {

                // Add blank row
                fileRow = trTable.NewRow();

                // Fill row with data

                fileRow["symbol"]               =
myTradeHList.getTradeHoldersSum()[i].Symbol;
                fileRow["code of Indicator"]    =
myTradeHList.getTradeHoldersSum()[i].Code_of_trade;
                fileRow["from Date Applied"]    =
myTradeHList.getTradeHoldersSum()[i].FromDateApplied;
                fileRow["to Date Applied"]      =
myTradeHList.getTradeHoldersSum()[i].ToDateApplied;
                fileRow["total Earning"]        =
```

```
myTradeHList.getTradeHoldersSum()[i].TotalEarning;
                fileRow["total Trades"]         =
myTradeHList.getTradeHoldersSum()[i].TotalTrades;
                fileRow["success Trades"]       =
myTradeHList.getTradeHoldersSum()[i].SuccessTrades;
                fileRow["unSuccess Trades"]     =
myTradeHList.getTradeHoldersSum()[i].UnSuccessTrades;
                fileRow["performance"]          =
myTradeHList.getTradeHoldersSum()[i].Performance;
                fileRow["annual Performance"]   =
myTradeHList.getTradeHoldersSum()[i].AnnualPerformance;
                fileRow["parameter Desc"]       =
myTradeHList.getTradeHoldersSum()[i].ParameterDesc;
                fileRow["parameter 1"]          =
myTradeHList.getTradeHoldersSum()[i].Parameter1;
                fileRow["parameter 2"]          =
myTradeHList.getTradeHoldersSum()[i].Parameter2;
                fileRow["parameter 3"]          =
myTradeHList.getTradeHoldersSum()[i].Parameter3;
                fileRow["number Of Stocks"]     =
myTradeHList.getTradeHoldersSum()[i].NumberOfStocks;
                fileRow["fee Per Trade"]        =
myTradeHList.getTradeHoldersSum()[i].FeePerTrade;
                fileRow["type of trade"]        =
myTradeHList.getTradeHoldersSum()[i].TypeOfTrade;
                trTable.Rows.Add(fileRow);
            }
            return trTable;
        }

        public DataTable LoadInDBAllTrades(TradeHolderList myTradeHList)
        {
            DataTable trTable = CreateDBTrade();
            DataRow fileRow;

            for (int i = 0; i < myTradeHList.getTradeHoldersDet().Count; i++)
            {

                // Add blank row
                fileRow = trTable.NewRow();

                // Fill row with data

                fileRow["symbol"]               =
myTradeHList.getTradeHoldersDet()[i].Symbol;
                fileRow["code of Indicator"]    =
myTradeHList.getTradeHoldersDet()[i].Code_of_trade;
                fileRow["from Date Applied"]    =
myTradeHList.getTradeHoldersDet()[i].FromDateApplied;
                fileRow["to Date Applied"]      =
myTradeHList.getTradeHoldersDet()[i].ToDateApplied;
                fileRow["total Earning"]        =
myTradeHList.getTradeHoldersDet()[i].TotalEarning;
```

```
                fileRow["total Trades"]         =
myTradeHList.getTradeHoldersDet()[i].TotalTrades;
                fileRow["success Trades"]       =
myTradeHList.getTradeHoldersDet()[i].SuccessTrades;
                fileRow["unSuccess Trades"]     =
myTradeHList.getTradeHoldersDet()[i].UnSuccessTrades;
                fileRow["performance"]          =
myTradeHList.getTradeHoldersDet()[i].Performance;
                fileRow["annual Performance"]   =
myTradeHList.getTradeHoldersDet()[i].AnnualPerformance;
                fileRow["parameter Desc"]       =
myTradeHList.getTradeHoldersDet()[i].ParameterDesc;
                fileRow["parameter 1"]          =
myTradeHList.getTradeHoldersDet()[i].Parameter1;
                fileRow["parameter 2"]          =
myTradeHList.getTradeHoldersDet()[i].Parameter2;
                fileRow["parameter 3"]          =
myTradeHList.getTradeHoldersDet()[i].Parameter3;
                fileRow["number Of Stocks"]     =
myTradeHList.getTradeHoldersDet()[i].NumberOfStocks;
                fileRow["fee Per Trade"]        =
myTradeHList.getTradeHoldersDet()[i].FeePerTrade;
                fileRow["type of trade"]        =
myTradeHList.getTradeHoldersDet()[i].TypeOfTrade;
                trTable.Rows.Add(fileRow);
            }
            return trTable;
        }
    }
}
```

Package (namespace) : `Signal`	**Source:** `AITradeCalculator.cs`

```
/// <summary>
/// Summary: description for AITradeCalculator class
/// -----------------------------------------------------------------
/// It contains the AITradeCalculator class
/// -----------------------------------------------------------------
/// Purpose: It calculates the performance of specific trading system.
///          It takes the security class and the signals and calculates
///          the performance of the system
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using AIPredictor.BasicOper;
using AIPredictor.Trades;

namespace AIPredictor.Signals
{
    class AITradeCalculator
    {
        private Security sec;
        private string symbol;
        private string code_of_trade;
        private string typeOfTrade;
        private Date fromDateApplied;
        private Date toDateApplied;
        private int numberOfStocks;
        private float feePerTrade;

        private List<float> indicatorParam;
        private List<Signal> signals;

        /// <summary>
        /// Constractor method
        /// </summary>
        /// <param name="s"></param>
        /// <param name="code_of_tradeStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="strFromDateApplied"></param>
        public AITradeCalculator(Security s, string code_of_tradeStr, string
typeOfTradeStr, string strFromDateApplied)
        {
            this.sec = s;
            this.symbol = s.Symbol;
            this.code_of_trade = code_of_tradeStr;
            this.typeOfTrade = typeOfTradeStr;
```

```
            setFromDateApplied(strFromDateApplied);
            setToDateApplied(s.getDate(s.CountValues() - 1));

            this.numberOfStocks = 1000;
            this.feePerTrade = 10;

            this.signals = new List<Signal>();
            this.indicatorParam = new List<float>();
        }

        public static bool RhsIsGreaterByEarning(object lhs, object rhs)
        {
            AITradeCalculator secValuesLhs = (AITradeCalculator)lhs;
            AITradeCalculator secValuesRhs = (AITradeCalculator)rhs;
            return (secValuesRhs.TotalEarning < secValuesLhs.TotalEarning) ?
true : false;
        }


        // public property Symbol
        public string Symbol
        {
            get
            {
                return symbol;
            }
            set
            {
                symbol = value;
            }
        }

        // public property Code_of_trade
        public string Code_of_trade
        {
            get
            {
                return code_of_trade;
            }
            set
            {
                code_of_trade = value;
            }
        }

        // public property TypeOfTrade
        public string TypeOfTrade
        {
            get
            {
                return typeOfTrade;
            }
            set
```

```
            {
                typeOfTrade = value;
            }
        }

        private float getTotalEarning()
        {
            float sumEarn = 0.0f;
            for (int i = 0; i < signals.Count; i++)
            {
                sumEarn += signals[i].EarnMoney;
            }

            sumEarn = numberOfStocks * sumEarn - this.signals.Count *
feePerTrade;

            return sumEarn;
        }

        // public property TotalEarning
        public float TotalEarning
        {
            get
            {
                return this.getTotalEarning();
            }
        }

        // public property TotalTrades
        public float TotalTrades
        {
            get
            {
                return this.getTotalTrades();
            }
        }

        private int getTotalTrades()
        {
            return this.signals.Count;
        }

        // public property Performance
        public float Performance
        {
            get
            {
                return this.getPerformance();
            }
        }

        private float getPerformance()
        {
```

```
            float initialInv = 0;
            float performanceInv = 0;

            if (this.signals.Count > 0)
            {
                initialInv = this.signals[0].BuyClose * numberOfStocks;
                performanceInv = 100 * getTotalEarning() / initialInv;
            }
            return performanceInv;
        }

        private float getAnnualPerformance()
        {
            int startingDay = sec.getIdFromDate(this.FromDateApplied);
            int allDays = sec.securityListValues.Count;

            int tradingDays = allDays - startingDay + 1;

            return  (float) (Performance * (264.0 / (tradingDays+0.0)));

        }

        public float AnnualPerformance
        {
            get
            {
                return this.getAnnualPerformance();
            }
        }


        // public property SuccessTrades
        public int SuccessTrades
        {
            get
            {
                return this.getNumberOfSuccessTrades();
            }
        }

        private int getNumberOfSuccessTrades()
        {
            int ncount = 0;
            for (int i = 0; i<getTotalTrades(); i++)
            {
                if (this.signals[i].EarnMoney > 0)
                    ncount++;
            }
            return ncount;
        }

        // public property UnSuccessTrades
        public int UnSuccessTrades
```

```
        {
            get
            {
                return this.getNumberOfUnSuccessTrades();
            }
        }
        private int getNumberOfUnSuccessTrades()
        {
            int ncount = 0;
            for (int i = 0; i < getTotalTrades(); i++)
            {
                if (this.signals[i].EarnMoney <= 0)
                    ncount++;
            }
            return ncount;
        }

        // public property NumberOfStocks
        public int NumberOfStocks
        {
            get
            {
                return numberOfStocks;
            }
            set
            {
                numberOfStocks = value;
            }
        }

        // public property FeePerTrade
        public float FeePerTrade
        {
            get
            {
                return feePerTrade;
            }
            set
            {
                feePerTrade = value;
            }
        }

        public List<float> getArrayParams()
        {
            return indicatorParam;
        }

        public List<Signal> getSignals()
        {
            return signals;
        }
```

```
        // public property FromDateApplied
        public string FromDateApplied
        {
            get
            {
                return fromDateApplied.ToDateString();
            }
        }

        /// <summary>
        /// Set fromDateApplied value
        /// </summary>
        /// <param name="strDate"></param>
        public void setFromDateApplied(string strDate)
        {
            this.fromDateApplied = Date.ConvetStringToDate(strDate);
        }

        // public property ToDateApplied
        public string ToDateApplied
        {
            get
            {
                return toDateApplied.ToDateString();
            }
        }

        /// <summary>
        /// Set ToDateApplied value
        /// </summary>
        /// <param name="strDate"></param>
        public void setToDateApplied(string strDate)
        {
            this.toDateApplied = Date.ConvetStringToDate(strDate);
        }

        public void addSignal(Signal s)
        {
            signals.Add(s);
        }

        public void SetInLastSellSignal(string sellDateStr, float
sellCloseValue)
        {
            signals[signals.Count - 1].SetSellSignal(sellDateStr,
sellCloseValue);
        }

        public void addParamValue(float paramValue)
        {
            indicatorParam.Add(paramValue);
        }
```

```
        public TradeHolder convertToTradeHolder(string parameterDescVal)
        {
            TradeHolder myHolder = new TradeHolder();
            myHolder.Symbol             = this.Symbol;
            myHolder.Code_of_trade      = this.Code_of_trade;
            myHolder.TypeOfTrade        = this.TypeOfTrade;
            myHolder.FromDateApplied    = this.FromDateApplied;
            myHolder.ToDateApplied      = this.ToDateApplied;
            myHolder.NumberOfStocks     = this.NumberOfStocks;
            myHolder.FeePerTrade        = this.FeePerTrade;
            myHolder.TotalEarning       = this.TotalEarning;
            myHolder.TotalTrades        = this.TotalTrades;
            myHolder.SuccessTrades      = this.SuccessTrades;
            myHolder.UnSuccessTrades    = this.UnSuccessTrades;
            myHolder.Performance        = this.Performance;
            myHolder.AnnualPerformance  = this.AnnualPerformance;
            myHolder.ParameterDesc      = parameterDescVal;

            if (indicatorParam.Count > 0)
                myHolder.Parameter1 = this.indicatorParam[0];

            if (indicatorParam.Count > 1)
                myHolder.Parameter2 = this.indicatorParam[1];

            if (indicatorParam.Count > 2)
                myHolder.Parameter3 = this.indicatorParam[2];

            myHolder.setSignals(this.getSignals());

            return myHolder;

        }

    }
}
```

Package (namespace) : `Signal`	**Source:** `TradeHolder.cs`

```
/// <summary>
/// Summary: description for TradeHolder class
/// -----------------------------------------------------------------
/// It contains the form TradeHolder
/// -----------------------------------------------------------------
/// Purpose: It is used to hold all relative information of a stock
/// and specific trading system. This class is used for transfering
/// data into db.
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;

namespace AIPredictor.Signals
{
    class TradeHolder
    {
        private string symbol;
        private string code_of_trade;
        private string typeOfTrade;
        private string fromDateApplied;
        private string toDateApplied;
        private int numberOfStocks;
        private float feePerTrade;
        private float totalEarning;
        private float totalTrades;
        private int successTrades;
        private int unSuccessTrades;
        private float performance;
        private float annualPerformance;
        private string parameterDesc;
        private float parameter1;
        private float parameter2;
        private float parameter3;

        private List<Signal> signals;

        /// <summary>
        /// Constractor method
        /// </summary>
        /// <param name="symbolVal"></param>
        /// <param name="code_of_tradeVal"></param>
        /// <param name="typeOfTradeVal"></param>
        /// <param name="fromDateAppliedVal"></param>
        /// <param name="toDateAppliedVal"></param>
        /// <param name="numberOfStocksVal"></param>
```

```
        /// <param name="feePerTradeVal"></param>
        /// <param name="totalEarningVal"></param>
        /// <param name="totalTradesVal"></param>
        /// <param name="successTradesVal"></param>
        /// <param name="unSuccessTradesVal"></param>
        /// <param name="performanceVal"></param>
        /// <param name="annualPerformanceVal"></param>
        /// <param name="parameterDescVal"></param>
        /// <param name="parameter1Val"></param>
        /// <param name="parameter2Val"></param>
        /// <param name="parameter3Val"></param>
        public TradeHolder(string symbolVal, string code_of_tradeVal, string
typeOfTradeVal, string fromDateAppliedVal,
                           string toDateAppliedVal, int numberOfStocksVal, float
feePerTradeVal, float totalEarningVal,
                           float totalTradesVal, int successTradesVal, int
unSuccessTradesVal, float performanceVal,
                           float annualPerformanceVal, string parameterDescVal,
float parameter1Val, float parameter2Val, float parameter3Val)
        {
            this.symbol = symbolVal;
            this.code_of_trade = code_of_tradeVal;
            this.typeOfTrade = typeOfTradeVal;
            this.fromDateApplied = fromDateAppliedVal;
            this.toDateApplied = toDateAppliedVal;
            this.numberOfStocks = numberOfStocksVal;
            this.feePerTrade = feePerTradeVal;
            this.totalEarning = totalEarningVal;
            this.successTrades = successTradesVal;
            this.unSuccessTrades = unSuccessTradesVal;
            this.performance = performanceVal;
            this.annualPerformance = annualPerformanceVal;
            this.parameterDesc = parameterDescVal;
            this.parameter1 = parameter1Val;
            this.parameter2 = parameter2Val;
            this.parameter3 = parameter3Val;

            this.signals = new List<Signal>();
        }

        /// <summary>
        /// Another constractor method
        /// </summary>
        /// <param name="symbolVal"></param>
        /// <param name="code_of_tradeVal"></param>
        /// <param name="typeOfTradeVal"></param>
        /// <param name="fromDateAppliedVal"></param>
        /// <param name="toDateAppliedVal"></param>
        /// <param name="numberOfStocksVal"></param>
        /// <param name="feePerTradeVal"></param>
        /// <param name="totalEarningVal"></param>
        /// <param name="totalTradesVal"></param>
        /// <param name="successTradesVal"></param>
```

```
        /// <param name="unSuccessTradesVal"></param>
        /// <param name="performanceVal"></param>
        /// <param name="annualPerformanceVal"></param>
        /// <param name="parameterDescVal"></param>
        /// <param name="parameter1Val"></param>
        /// <param name="parameter2Val"></param>
        public TradeHolder(string symbolVal, string code_of_tradeVal, string
typeOfTradeVal, string fromDateAppliedVal,
                           string toDateAppliedVal, int numberOfStocksVal,
float feePerTradeVal, float totalEarningVal,
                           float totalTradesVal, int successTradesVal, int
unSuccessTradesVal, float performanceVal,
                           float annualPerformanceVal, string parameterDescVal,
float parameter1Val, float parameter2Val)
        {
            this.symbol = symbolVal;
            this.code_of_trade = code_of_tradeVal;
            this.typeOfTrade = typeOfTradeVal;
            this.fromDateApplied = fromDateAppliedVal;
            this.toDateApplied = toDateAppliedVal;
            this.numberOfStocks = numberOfStocksVal;
            this.feePerTrade = feePerTradeVal;
            this.totalEarning = totalEarningVal;
            this.successTrades = successTradesVal;
            this.unSuccessTrades = unSuccessTradesVal;
            this.performance = performanceVal;
            this.annualPerformance = annualPerformanceVal;
            this.parameterDesc = parameterDescVal;
            this.parameter1 = parameter1Val;
            this.parameter2 = parameter2Val;
            this.parameter3 = 0;

            this.signals = new List<Signal>();
        }

        /// <summary>
        /// Another constractor method
        /// </summary>
        /// <param name="symbolVal"></param>
        /// <param name="code_of_tradeVal"></param>
        /// <param name="typeOfTradeVal"></param>
        /// <param name="fromDateAppliedVal"></param>
        /// <param name="toDateAppliedVal"></param>
        /// <param name="numberOfStocksVal"></param>
        /// <param name="feePerTradeVal"></param>
        /// <param name="totalEarningVal"></param>
        /// <param name="totalTradesVal"></param>
        /// <param name="successTradesVal"></param>
        /// <param name="unSuccessTradesVal"></param>
        /// <param name="performanceVal"></param>
        /// <param name="annualPerformanceVal"></param>
        /// <param name="parameterDescVal"></param>
        /// <param name="parameter1Val"></param>
```

```
        public TradeHolder(string symbolVal, string code_of_tradeVal, string
typeOfTradeVal, string fromDateAppliedVal,
                    string toDateAppliedVal, int numberOfStocksVal, float
feePerTradeVal, float totalEarningVal,
                    float totalTradesVal, int successTradesVal, int
unSuccessTradesVal, float performanceVal,
                    float annualPerformanceVal, string parameterDescVal, float
parameter1Val)
        {
            this.symbol = symbolVal;
            this.code_of_trade = code_of_tradeVal;
            this.typeOfTrade = typeOfTradeVal;
            this.fromDateApplied = fromDateAppliedVal;
            this.toDateApplied = toDateAppliedVal;
            this.numberOfStocks = numberOfStocksVal;
            this.feePerTrade = feePerTradeVal;
            this.totalEarning = totalEarningVal;
            this.successTrades = successTradesVal;
            this.unSuccessTrades = unSuccessTradesVal;
            this.performance = performanceVal;
            this.annualPerformance = annualPerformanceVal;
            this.parameterDesc = parameterDescVal;
            this.parameter1 = parameter1Val;
            this.parameter2 = 0;
            this.parameter3 = 0;

            this.signals = new List<Signal>();
        }

        /// <summary>
        /// Constractor method  - default data values
        /// </summary>
        public TradeHolder()
        {
            this.symbol = "";
            this.code_of_trade = "";
            this.typeOfTrade = "";
            this.fromDateApplied = "";
            this.toDateApplied = "";
            this.numberOfStocks = 0;
            this.feePerTrade = 0;
            this.totalEarning = 0;
            this.successTrades = 0;
            this.unSuccessTrades = 0;
            this.performance = 0;
            this.annualPerformance = 0;
            this.parameterDesc = "default";
            this.parameter1 = 0;
            this.parameter2 = 0;
            this.parameter3 = 0;

            this.signals = new List<Signal>();
        }
```

```
            // public property Symbol
        public string Symbol
        {
            get
            {
                return symbol;
            }
            set
            {
                symbol = value;
            }
        }

        // public property Code_of_trade
        public string Code_of_trade
        {
            get
            {
                return code_of_trade;
            }
            set
            {
                code_of_trade = value;
            }
        }

        // public property TypeOfTrade
        public string TypeOfTrade
        {
            get
            {
                return typeOfTrade;
            }
            set
            {
                typeOfTrade = value;
            }
        }

        // public property FromDateApplied
        public string FromDateApplied
        {
            get
            {
                return fromDateApplied;
            }
            set
            {
                fromDateApplied = value;
            }
        }

        // public property ToDateApplied
```

```
        public string ToDateApplied
        {
            get
            {
                return toDateApplied;
            }
            set
            {
                toDateApplied = value;
            }
        }

        // public property NumberOfStocks
        public int NumberOfStocks
        {
            get
            {
                return numberOfStocks;
            }
            set
            {
                numberOfStocks = value;
            }
        }

        // public property FeePerTrade
        public float FeePerTrade
        {
            get
            {
                return feePerTrade;
            }
            set
            {
                feePerTrade = value;
            }
        }

        // public property TotalEarning
        public float TotalEarning
        {
            get
            {
                return totalEarning;
            }
            set
            {
                totalEarning = value;
            }
        }

        // public property TotalTrades
        public float TotalTrades
```

```
        {
            get
            {
                return totalTrades;
            }
            set
            {
                totalTrades = value;
            }
        }

        // public property SuccessTrades
        public int SuccessTrades
        {
            get
            {
                return successTrades;
            }
            set
            {
                successTrades = value;
            }
        }

        // public property UnSuccessTrades
        public int UnSuccessTrades
        {
            get
            {
                return unSuccessTrades;
            }
            set
            {
                unSuccessTrades = value;
            }
        }


        // public property Performance
        public float Performance
        {
            get
            {
                return performance;
            }
            set
            {
                performance = value;
            }
        }


        public float AnnualPerformance
```

```
        {
            get
            {
                return annualPerformance;
            }
            set
            {
                annualPerformance = value;
            }
        }

        public string ParameterDesc
        {
            get
            {
                return parameterDesc;
            }
            set
            {
                parameterDesc = value;
            }
        }

        public float Parameter1
        {
            get
            {
                return parameter1;
            }
            set
            {
                parameter1 = value;
            }
        }

        public float Parameter2
        {
            get
            {
                return parameter2;
            }
            set
            {
                parameter2 = value;
            }
        }

        public float Parameter3
        {
            get
            {
                return parameter3;
            }
```

```
            set
            {
                parameter3 = value;
            }
        }

        public List<Signal> getSignals()
        {
            return signals;
        }

        public void setSignals(List<Signal> s)
        {
            signals = s;
        }
    }
}
```

Package (namespace) : `Signal`	**Source:** `TradeHolderList.cs`

```
/// <summary>
/// Summary: description for TradeHolderList class
/// -----------------------------------------------------------------
/// It contains the TradeHolderList class
/// -----------------------------------------------------------------
/// Purpose: Used for holding list of TradeHolder objects
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;

namespace AIPredictor.Signals
{
    class TradeHolderList
    {
        private List<TradeHolder> tradeHoldersDet;
        private List<TradeHolder> tradeHoldersSum;

        /// <summary>
        /// Constractor Class
        /// </summary>
        public TradeHolderList()
        {
            this.tradeHoldersDet = new List<TradeHolder>();
            this.tradeHoldersSum = new List<TradeHolder>();
        }

        public List<TradeHolder> getTradeHoldersDet()
        {
            return tradeHoldersDet;
        }

        public void addTradeHolderDet(TradeHolder t)
        {
            tradeHoldersDet.Add(t);
        }

        public List<TradeHolder> getTradeHoldersSum()
        {
            return tradeHoldersSum;
        }

        public void addTradeHolderSum(TradeHolder t)
        {
            tradeHoldersSum.Add(t);
        }
```

```
        public void addTradeHolderList(TradeHolderList tList)
        {
            for (int i = 0; i < tList.getTradeHoldersDet().Count; i++)
            {
                this.addTradeHolderDet(tList.getTradeHoldersDet()[i]);
            }

            for (int i = 0; i < tList.getTradeHoldersSum().Count; i++)
            {
                this.addTradeHolderSum(tList.getTradeHoldersSum()[i]);
            }
        }


    }
}
```

Package (namespace) : SmartInput	**Source:** BasicStrategy.cs

```
/// <summary>
/// Summary: description for BasicStrategy class
/// -----------------------------------------------------------
/// It contains the BasicStrategy class
/// -----------------------------------------------------------
/// Purpose: Evaluation of specific technical indicators that can not
///          produce trading signals. The result (output) is to be used
///          as input to the neural net
/// </summary>
/// -----------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using AIPredictor.BasicOper;

namespace AIPredictor.SmartInput
{
    class BasicStrategy
    {
        private Security curSecurity;
        private const double constBaseWeight = 0.5f;

        /// <summary>
        /// Constractor method
        /// </summary>
        /// <param name="sec"></param>
        public BasicStrategy(Security sec)
        {
            this.curSecurity = sec;
        }

        public void SetSecurity(Security newSec)
        {
            this.curSecurity = newSec;
        }

        /// <summary>
        /// Evaluation referring to close values
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double evalCloseRef(int i)
        {
            double curEvaluator = constBaseWeight;

            if (curSecurity.getClose(i) > curSecurity.getClose(i-1))
```

```
            {
                curEvaluator = curEvaluator + 0.22;
            }
            else
            {
                curEvaluator = curEvaluator - 0.22;
            }

            if (curSecurity.getClose(i) > curSecurity.getClose(i-2))
            {
                curEvaluator = curEvaluator + 0.15;
            }
            else
            {
                curEvaluator = curEvaluator - 0.15;
            }

            if (curSecurity.getClose(i - 1) > curSecurity.getClose(i - 2))
            {
                curEvaluator = curEvaluator + 0.12;
            }
            else
            {
                curEvaluator = curEvaluator - 0.12;
            }

            return curEvaluator;
        }

        /// <summary>
        /// Evaluation referring to candle formation
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double evalCandleType(int i)
        {
            double curEvaluator = constBaseWeight;

            if (curSecurity.getClose(i) > curSecurity.getOpen(i))
            {
                curEvaluator = curEvaluator + 0.25;
            }
            else
            {
                curEvaluator = curEvaluator - 0.25;
            }

            if (curSecurity.getClose(i) >
(curSecurity.getHigh(i)+curSecurity.getLow(i))/2)
            {
                curEvaluator = curEvaluator + 0.2;
            }
            else
```

```
            {
                curEvaluator = curEvaluator - 0.2;
            }

            return curEvaluator;
        }

        /// <summary>
        /// Evaluation referring to Williams indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double evalWilliams(int i)
        {
            double curEvaluator = constBaseWeight;
            float williamsVal = Indicator.Williams(curSecurity,i,14);
            float williamsValRef1 = Indicator.Williams(curSecurity, i-1, 14);
            float williamsValRef2 = Indicator.Williams(curSecurity, i-2, 14);

            if (williamsVal > williamsValRef1)
            {
                curEvaluator = curEvaluator + 0.17;
            }
            else
            {
                curEvaluator = curEvaluator - 0.17;
            }

            if (williamsValRef2 > williamsValRef1)
            {
                curEvaluator = curEvaluator + 0.12;
            }
            else
            {
                curEvaluator = curEvaluator - 0.12;
            }

            if (williamsVal<-70)
                curEvaluator = curEvaluator + 0.20;

            if (williamsVal > -30)
                curEvaluator = curEvaluator - 0.20;


            return curEvaluator;
        }

        /// <summary>
        /// Evaluation referring to volume
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double evalVolume(int i)
```

```
        {
            double curEvaluator = constBaseWeight;

            Boolean volIncresingWithSMA = curSecurity.getVolume(i) >
Indicator.SMAVolume(curSecurity, i, 10);
            Boolean volIncresingWithPrev = curSecurity.getVolume(i) >
curSecurity.getVolume(i-1);
            Boolean CloseIncresing = curSecurity.getClose(i) >
curSecurity.getClose(i - 1);

            if (volIncresingWithSMA)
            {
                if (CloseIncresing)
                {
                    curEvaluator = curEvaluator + 0.22;
                }
                else
                {
                    curEvaluator = curEvaluator -  0.22;
                }
            }
            else  // decreasing comparing with volume SMA
            {
                if (CloseIncresing)
                {
                    curEvaluator = curEvaluator + 0.12;
                }
                else
                {
                    curEvaluator = curEvaluator -  0.12;
                }
            }

            if (volIncresingWithPrev)
            {
                if (CloseIncresing)
                {
                    curEvaluator = curEvaluator + 0.22;
                }
                else
                {
                    curEvaluator = curEvaluator - 0.22;
                }
            }
            else  // decreasing comparing with volume SMA
            {
                if (CloseIncresing)
                {
                    curEvaluator = curEvaluator + 0.12;
                }
                else
                {
                    curEvaluator = curEvaluator - 0.12;
```

```
                }
            }

            return curEvaluator;
        }

        /// <summary>
        /// Evaluation referring to Volume Osc indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double evalVolOsc(int i)
        {
            double curEvaluator = constBaseWeight;

            if (Indicator.VolumeOsc(curSecurity, i, 5, 15) > 0)
            {
                curEvaluator = curEvaluator + 0.45;
            }
            else
            {
                curEvaluator = curEvaluator - 0.45;
            }

            return curEvaluator;
        }

        /// <summary>
        /// Evaluation referring to Projection Osc indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double evalProjectionOsc(int i)
        {
            double curEvaluator = constBaseWeight;

            float ProjOscVal = Indicator.ProjectionOsc(curSecurity, i, 15);
            float ProjOscValRef1 = Indicator.ProjectionOsc(curSecurity, i - 1,
15);
            float ProjOscValRef2 = Indicator.ProjectionOsc(curSecurity, i - 2,
15);


            if ((ProjOscVal < ProjOscValRef1) && (ProjOscValRef1 <
ProjOscValRef2) && (ProjOscVal < 80) && ((ProjOscValRef2 > 80) ||
(ProjOscValRef1 > 80)))
            {
                curEvaluator = curEvaluator - 0.45;
            }

            if ((ProjOscVal > ProjOscValRef1) && (ProjOscValRef1 >
ProjOscValRef2) && (ProjOscVal > 20) && ((ProjOscValRef2 < 20) ||
(ProjOscValRef1 < 20)))
```

```
            {
                curEvaluator = curEvaluator + 0.45;
            }

            return curEvaluator;
        }

        /// <summary>
        /// Evaluation referring to R-Squared indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double evalR_Squared(int i)
        {
            return Indicator.R_Squared(curSecurity, i, 15, "C");
        }

        /// <summary>
        /// Evaluation referring to VHF indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double evalVHF(int i)
        {
            return Indicator.VHF(curSecurity, i, 15);
        }

        /// <summary>
        /// The number of indicators to extract
        /// </summary>
        /// <param name="selectionList"></param>
        /// <returns></returns>
        public int getNumberOfIndicators(Boolean[] selectionList)
        {
            int ncount = 0;
            for (int i = 0; i < selectionList.Length; i++)
                if (selectionList[i])
                    ncount = ncount + 1;

            return ncount;
        }

        /// <summary>
        /// returns the titles of indicators
        /// </summary>
        /// <returns></returns>
        public List<string> extractTitlesOfIndicators()
        {
            List<string> titlesArray = new List<string>();

            titlesArray.Add("CloseRef");
            titlesArray.Add("CandleType");
            titlesArray.Add("Williams");
```

```
            titlesArray.Add("Volume");
            titlesArray.Add("VolOsc");
            titlesArray.Add("ProjOsc");
            titlesArray.Add("R_Sqrd");
            titlesArray.Add("VHF");

            return titlesArray;
        }

        /// <summary>
        /// Returns only the selected titles of indicators
        /// </summary>
        /// <param name="selectionList"></param>
        /// <returns></returns>
        public List<string> extractTitlesOfIndicators(Boolean[] selectionList)
        {
            List<string> titlesArray = new List<string>();

            if (selectionList[0])
                titlesArray.Add("CloseRef");

            if (selectionList[1])
                titlesArray.Add("CandleType");

            if (selectionList[2])
                titlesArray.Add("Williams");

            if (selectionList[3])
                titlesArray.Add("Volume");

            if (selectionList[4])
                titlesArray.Add("VolOsc");

            if (selectionList[5])
                titlesArray.Add("ProjOsc");

            if (selectionList[6])
                titlesArray.Add("R_Sqrd");

            if (selectionList[7])
                titlesArray.Add("VHF");

            return titlesArray;
        }

        /// <summary>
        /// Returns the selected values of indicators
        /// </summary>
        /// <param name="i"></param>
        /// <param name="selectionList"></param>
        /// <returns></returns>
        public List<double> extractSmartInputArray(int i, Boolean[]
selectionList)
```

```
        {
            List<double> smartArray = new List<double>();

            if (selectionList[0])
                smartArray.Add(evalCloseRef(i));

            if (selectionList[1])
                smartArray.Add(evalCandleType(i));

            if (selectionList[2])
                smartArray.Add(evalWilliams(i));

            if (selectionList[3])
                smartArray.Add(evalVolume(i));

            if (selectionList[4])
                smartArray.Add(evalVolOsc(i));

            if (selectionList[5])
                smartArray.Add(evalProjectionOsc(i));

            if (selectionList[6])
                smartArray.Add(evalR_Squared(i));

            if (selectionList[7])
                smartArray.Add(evalVHF(i));

            return smartArray;
        }
    }
}
```

Package (namespace) : SmartInput	**Source:** IndicatorOptParam.cs

```
/// <summary>
/// Summary: description for IndicatorOptParam class
/// -----------------------------------------------------------------
/// It contains the IndicatorOptParam class
/// -----------------------------------------------------------------
/// Purpose: Holds the best parameters (input) values of
///          a technical indicator
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;

namespace AIPredictor.SmartInput
{
    class IndicatorOptParam
    {
        private string codeInd;
        private float annualPerformance;
        private string parameterDesc;
        private float parameter1;
        private float parameter2;
        private float parameter3;
        private Boolean isActivated;
        private float coffEvaluation;

        /// <summary>
        /// constractor method
        /// </summary>
        /// <param name="codeIndVal"></param>
        /// <param name="annualPerformanceVal"></param>
        /// <param name="parameterDescVal"></param>
        /// <param name="parameter1Val"></param>
        /// <param name="parameter2Val"></param>
        /// <param name="parameter3Val"></param>
        /// <param name="isActivatedVal"></param>
        public IndicatorOptParam(string codeIndVal, float annualPerformanceVal,
string parameterDescVal,
                                 float parameter1Val, float parameter2Val, float
parameter3Val, Boolean isActivatedVal)
        {
            this.codeInd = codeIndVal;
            this.annualPerformance = annualPerformanceVal;
            this.parameterDesc = parameterDescVal;
            this.parameter1 = parameter1Val;
            this.parameter2 = parameter2Val;
            this.parameter3 = parameter3Val;
```

```
            this.isActivated = isActivatedVal;
        }

        /// <summary>
        /// Another constractor method
        /// </summary>
        /// <param name="codeIndVal"></param>
        /// <param name="annualPerformanceVal"></param>
        /// <param name="parameterDescVal"></param>
        /// <param name="parameter1Val"></param>
        /// <param name="parameter2Val"></param>
        /// <param name="parameter3Val"></param>
        public IndicatorOptParam(string codeIndVal, float annualPerformanceVal,
string parameterDescVal,
                                 float parameter1Val, float parameter2Val, float
parameter3Val)
        {
            this.codeInd = codeIndVal;
            this.annualPerformance = annualPerformanceVal;
            this.parameterDesc = parameterDescVal;
            this.parameter1 = parameter1Val;
            this.parameter2 = parameter2Val;
            this.parameter3 = parameter3Val;
            this.isActivated = false;
            this.coffEvaluation = 0;
        }

        // public property CodeInd
        public string CodeInd
        {
            get
            {
                return codeInd;
            }
            set
            {
                codeInd = value;
            }
        }

        public float AnnualPerformance
        {
            get
            {
                return annualPerformance;
            }
            set
            {
                annualPerformance = value;
            }
        }

        public string ParameterDesc
```

```
        {
            get
            {
                return parameterDesc;
            }
            set
            {
                parameterDesc = value;
            }
        }

        public float Parameter1
        {
            get
            {
                return parameter1;
            }
            set
            {
                parameter1 = value;
            }
        }

        public float Parameter2
        {
            get
            {
                return parameter2;
            }
            set
            {
                parameter2 = value;
            }
        }

        public float Parameter3
        {
            get
            {
                return parameter3;
            }
            set
            {
                parameter3 = value;
            }
        }

        public Boolean IsActivated
        {
            get
            {
                return isActivated;
            }
```

```
            set
            {
                isActivated = value;
            }
        }

        public float CoffEvaluation
        {
            get
            {
                return coffEvaluation;
            }
            set
            {
                coffEvaluation = value;
            }
        }

    }
}
```

Package (namespace) : SmartInput	**Source:** OptimStrategy.cs

```
/// <summary>
/// Summary: description for OptimStrategy class
/// -----------------------------------------------------------
/// It contains the OptimStrategy class
/// -----------------------------------------------------------
/// Purpose: Produces the input values for the neural net.
///          It normalize values (0 - 1). Signals for buy (long position)
///          return 0.999 otherwise 0.001
/// </summary>
/// -----------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using AIPredictor.BasicOper;

namespace AIPredictor.SmartInput
{
    class OptimStrategy
    {
        private Security curSecurity;
        private OptParamsEval curOptparamsEval;
        private double validPerformance;

        /// <summary>
        /// Constractor method
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="mydb"></param>
        public OptimStrategy(Security sec, DBLayer mydb)
        {
            this.curSecurity = sec;
            this.validPerformance = 10;
            this.curOptparamsEval = new OptParamsEval(curSecurity, mydb,
validPerformance);

        }

        /// <summary>
        /// Constractor method based on valid performance
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="mydb"></param>
        /// <param name="validPerf"></param>
        public OptimStrategy(Security sec, DBLayer mydb, double validPerf)
        {
            this.curSecurity = sec;
            this.validPerformance = validPerf;
```

```
            this.curOptparamsEval = new OptParamsEval(curSecurity, mydb,
validPerf);

        }

        /// <summary>
        /// The indicators that will be used are depended on their
        /// performance. Only above this performance will be used!
        /// </summary>
        public double ValidPerformance
        {
            get
            {
                return validPerformance;
            }
            set
            {
                validPerformance = value;
                curOptparamsEval.ValidPerformance = this.validPerformance;
            }
        }

        /// <summary>
        /// Get the signal value of SMA indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_SMA(int i)
        {
            int optPeriods =
(int)curOptparamsEval.getIndicatorParams("SMA").Parameter1;
            if (curSecurity.getClose(i) > Indicator.SMA(curSecurity, i,
optPeriods))
            {
                return 0.999;
            }
            else
            {
                return 0.001;
            }
        }

        /// <summary>
        /// Get the signal value of EMA indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_EMA(int i)
        {
            int optPeriods =
(int)curOptparamsEval.getIndicatorParams("EMA").Parameter1;
            if (curSecurity.getClose(i) > Indicator.EMA(curSecurity, i,
optPeriods))
```

```
            {
                return 0.999;
            }
            else
            {
                return 0.001;
            }
        }

        /// <summary>
        /// Get the signal value of RVI indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_RVI(int i)
        {
            int optPeriods =
(int)curOptparamsEval.getIndicatorParams("RVI").Parameter1;
            float optBuyX =
curOptparamsEval.getIndicatorParams("RVI").Parameter2;
            float optSellY =
curOptparamsEval.getIndicatorParams("RVI").Parameter3;

            float curRVI = Indicator.RVI(curSecurity,i,optPeriods);

            if (curRVI > optBuyX)
            {
                return 0.999;
            }
            else if (curRVI < optSellY)
            {
                return 0.001;
            }
            else return 0.500;
        }

        /// <summary>
        /// Get the signal value of CCI indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_CCI(int i)
        {
            int optPeriods =
(int)curOptparamsEval.getIndicatorParams("CCI").Parameter1;
            float optBuyX =
curOptparamsEval.getIndicatorParams("CCI").Parameter2;
            float optSellY =
curOptparamsEval.getIndicatorParams("CCI").Parameter3;

            float curCCI = Indicator.CCI(curSecurity, i, optPeriods);

            if (curCCI > optBuyX)
```

```
            {
                return 0.999;
            }
            else if (curCCI < optSellY)
            {
                return 0.001;
            }
            else return 0.500;
        }

        /// <summary>
        /// Get the signal value of Price Osc indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_PriceOsc(int i)
        {
            int shortMA =
(int)curOptparamsEval.getIndicatorParams("PriceOsc").Parameter1;
            int longMA =
(int)curOptparamsEval.getIndicatorParams("PriceOsc").Parameter2;
            int signalMA =
(int)curOptparamsEval.getIndicatorParams("PriceOsc").Parameter3;

            if (Indicator.PriceOsc(curSecurity, i, shortMA, longMA) >
Indicator.SMAPriceOsc(curSecurity, i, shortMA, longMA, signalMA))
            {
                return 0.999;
            }
            else
            {
                return 0.001;
            }
        }

        /// <summary>
        /// Get the signal value of Momentum indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_Momentum(int i)
        {
            int optPeriods =
(int)curOptparamsEval.getIndicatorParams("Momentum").Parameter1;
            float optBuyX =
curOptparamsEval.getIndicatorParams("Momentum").Parameter2;
            float optSellY =
curOptparamsEval.getIndicatorParams("Momentum").Parameter3;

            float curMomentum = Indicator.MomentumOsc(curSecurity, i,
optPeriods);

            if (curMomentum > optBuyX)
```

```
            {
                return 0.999;
            }
            else if (curMomentum < optSellY)
            {
                return 0.001;
            }
            else return 0.500;
        }

        /// <summary>
        /// Get the signal value of ROC indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_ROC(int i)
        {
            int optPeriods =
(int)curOptparamsEval.getIndicatorParams("ROC").Parameter1;
            float optBuyX =
curOptparamsEval.getIndicatorParams("ROC").Parameter2;
            float optSellY =
curOptparamsEval.getIndicatorParams("ROC").Parameter3;

            float curROC = Indicator.ROC(curSecurity, i, optPeriods);

            if (curROC > optBuyX)
            {
                return 0.999;
            }
            else if (curROC < optSellY)
            {
                return 0.001;
            }
            else return 0.500;
        }

        /// <summary>
        /// Get the signal value of RSI indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_RSI(int i)
        {
            int optPeriods =
(int)curOptparamsEval.getIndicatorParams("RSI").Parameter1;
            float optBuyX =
curOptparamsEval.getIndicatorParams("RSI").Parameter2;
            float optSellY =
curOptparamsEval.getIndicatorParams("RSI").Parameter3;

            float curRSI = Indicator.RSI(curSecurity, i, optPeriods);
            float curRSIPrev1 = Indicator.RSI(curSecurity, i-1, optPeriods);
```

```
            if ((curRSI > optBuyX) && (curRSIPrev1 < curRSI))
            {
                return 0.999;
            }
            else if ((curRSI < optSellY) && (curRSIPrev1 > curRSI))
            {
                return 0.001;
            }
            else
                return 0.500;
        }

        /// <summary>
        /// Get the signal value of Stochastic indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_StochOsc(int i)
        {
            int xPeriods =
(int)curOptparamsEval.getIndicatorParams("StochOsc").Parameter1;
            int signalMA =
(int)curOptparamsEval.getIndicatorParams("StochOsc").Parameter2;

            if (Indicator.StochOsc(curSecurity, i, xPeriods) >
Indicator.SMAStochOsc(curSecurity, i, xPeriods, signalMA))
            {
                return 0.999;
            }
            else
            {
                return 0.001;
            }
        }

        /// <summary>
        /// Get the signal value of CMO indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_CMO(int i)
        {
            int optPeriods =
(int)curOptparamsEval.getIndicatorParams("CMO").Parameter1;
            float optBuyX =
curOptparamsEval.getIndicatorParams("CMO").Parameter2;
            float optSellY =
curOptparamsEval.getIndicatorParams("CMO").Parameter3;

            float curCMO = Indicator.ChandeMomentumOsc(curSecurity, i,
optPeriods);
```

```
            if (curCMO > optBuyX)
            {
                return 0.999;
            }
            else if (curCMO < optSellY)
            {
                return 0.001;
            }
            else return 0.500;
        }

        /// <summary>
        /// Get the signal value of Chaikin indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_Chaikin(int i)
        {
            int shortMA =
(int)curOptparamsEval.getIndicatorParams("Chaikin").Parameter1;
            int longMA =
(int)curOptparamsEval.getIndicatorParams("Chaikin").Parameter2;
            int signalMA =
(int)curOptparamsEval.getIndicatorParams("Chaikin").Parameter3;

            if (Indicator.ChaikinAD(curSecurity, i, shortMA, longMA) >
Indicator.SMAChaikinAD(curSecurity, i, shortMA, longMA, signalMA))
            {
                return 0.999;
            }
            else
            {
                return 0.001;
            }
        }

        /// <summary>
        /// Get the signal value of DMI indicator
        /// </summary>
        /// <param name="i"></param>
        /// <returns></returns>
        private double getPredictionFrom_DMI(int i)
        {
            int optPeriods =
(int)curOptparamsEval.getIndicatorParams("DMI").Parameter1;
            int signalDX =
(int)curOptparamsEval.getIndicatorParams("DMI").Parameter2;

            float[] listDMI = Indicator.DMI(curSecurity, i, optPeriods);
            Boolean curCalcStatus = (listDMI[1] > listDMI[0]);

            if (curCalcStatus)
            {
```

```
                if (listDMI[2] < signalDX)
                    curCalcStatus = false;
            }

            if (curCalcStatus)
            {
                return 0.999;
            }
            else
            {
                return 0.001;
            }
        }

        /// <summary>
        /// Get the number of valid indicators
        /// </summary>
        /// <returns></returns>
        public int getNumberOfIndicators()
        {
            return curOptparamsEval.countValidIndicators();
        }

        /// <summary>
        /// Extract the titles of valid indicators
        /// </summary>
        /// <returns></returns>
        public List<string> extractTitlesOfIndicators()
        {
            List<string> titlesArray = new List<string>();

            if (curOptparamsEval.getIndicatorParams("SMA").IsActivated)
                titlesArray.Add("SMA");

            if (curOptparamsEval.getIndicatorParams("EMA").IsActivated)
                titlesArray.Add("EMA");

            if (curOptparamsEval.getIndicatorParams("RVI").IsActivated)
                titlesArray.Add("RVI");

            if (curOptparamsEval.getIndicatorParams("CCI").IsActivated)
                titlesArray.Add("CCI");

            if (curOptparamsEval.getIndicatorParams("PriceOsc").IsActivated)
                titlesArray.Add("PriceOsc");

            if (curOptparamsEval.getIndicatorParams("Momentum").IsActivated)
                titlesArray.Add("Mom");

            if (curOptparamsEval.getIndicatorParams("ROC").IsActivated)
                titlesArray.Add("ROC");

            if (curOptparamsEval.getIndicatorParams("RSI").IsActivated)
```

```
                titlesArray.Add("RSI");

            if (curOptparamsEval.getIndicatorParams("StochOsc").IsActivated)
                titlesArray.Add("StochOsc");

            if (curOptparamsEval.getIndicatorParams("CMO").IsActivated)
                titlesArray.Add("CMO");

            if (curOptparamsEval.getIndicatorParams("Chaikin").IsActivated)
                titlesArray.Add("Chaikin");

            if (curOptparamsEval.getIndicatorParams("DMI").IsActivated)
                titlesArray.Add("DMI");

            return titlesArray;
        }
        /// <summary>
        /// Extract the values of valid indicators
        /// </summary>
        public List<double> extractSmartInputArray(int i)
        {
            List<double> smartArray = new List<double>();

            if (curOptparamsEval.getIndicatorParams("SMA").IsActivated)
                smartArray.Add( getPredictionFrom_SMA(i) );

            if (curOptparamsEval.getIndicatorParams("EMA").IsActivated)
                smartArray.Add(getPredictionFrom_EMA(i));

            if (curOptparamsEval.getIndicatorParams("RVI").IsActivated)
                smartArray.Add( getPredictionFrom_RVI(i));

            if (curOptparamsEval.getIndicatorParams("CCI").IsActivated)
                smartArray.Add( getPredictionFrom_CCI(i));

            if (curOptparamsEval.getIndicatorParams("PriceOsc").IsActivated)
                smartArray.Add( getPredictionFrom_PriceOsc(i));

            if (curOptparamsEval.getIndicatorParams("Momentum").IsActivated)
                smartArray.Add( getPredictionFrom_Momentum(i));

            if (curOptparamsEval.getIndicatorParams("ROC").IsActivated)
                smartArray.Add( getPredictionFrom_ROC(i));

            if (curOptparamsEval.getIndicatorParams("RSI").IsActivated)
                smartArray.Add( getPredictionFrom_RSI(i));

            if (curOptparamsEval.getIndicatorParams("StochOsc").IsActivated)
                smartArray.Add( getPredictionFrom_StochOsc(i));

            if (curOptparamsEval.getIndicatorParams("CMO").IsActivated)
                smartArray.Add( getPredictionFrom_CMO(i));
```

```
            if (curOptparamsEval.getIndicatorParams("Chaikin").IsActivated)
                smartArray.Add( getPredictionFrom_Chaikin(i));

            if (curOptparamsEval.getIndicatorParams("DMI").IsActivated)
                smartArray.Add(getPredictionFrom_DMI(i));

            return smartArray;
        }
    }
}
```

Package (namespace) : SmartInput	**Source:** OptParamsEval.cs

```
/// <summary>
/// Summary: description for OptParamsEval class
/// -----------------------------------------------------------------
/// It contains the OptParamsEval class. This class is used by OptimStrategy.
/// -----------------------------------------------------------------
/// Purpose: Used for loading all relative trading information about a stock.
///          It actually loads from db all performance data of all trading
systems
///          referred to a specific stock.
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Collections;
using System.Text;
using System.Data;
using System.Data.Sql;
using System.Data.SqlClient;
using AIPredictor.BasicOper;


namespace AIPredictor.SmartInput
{
    class OptParamsEval
    {
        private List<IndicatorOptParam> optParamsList;
        private Security curSecurity;
        private DBLayer db;
        private double validPerformance;
        private const float constMultiplier = 10;

        /// <summary>
        /// constractor method
        /// </summary>
        /// <param name="curSec"></param>
        /// <param name="mydb"></param>
        /// <param name="validPerformanceIn"></param>
        public OptParamsEval(Security curSec, DBLayer mydb, double
validPerformanceIn)
        {
            this.curSecurity = curSec;
            this.db = mydb;
            this.validPerformance = validPerformanceIn;
            this.optParamsList =
this.db.LoadAITradeOptValues(curSecurity.Symbol);
            this.activateBestIndicators();
            this.evaluatePerformances();
```

```
        }

        /// <summary>
        /// Constractor method - default value for property validPerformanceIn
        /// </summary>
        /// <param name="curSec"></param>
        /// <param name="mydb"></param>
        public OptParamsEval(Security curSec, DBLayer mydb)
        {
            this.curSecurity = curSec;
            this.db = mydb;
            this.validPerformance = 10;
            this.optParamsList =
this.db.LoadAITradeOptValues(curSecurity.Symbol);
            this.activateBestIndicators();
            this.evaluatePerformances();
        }

        /// <summary>
        /// The indicators that will be used are depended on their
        /// performance. Only above this performance will be used!
        /// </summary>
        public double ValidPerformance
        {
            get
            {
                return validPerformance;
            }
            set
            {
                validPerformance = value;
            }

        }

        /// <summary>
        /// How many indicators will be used that have above this performance
        /// </summary>
        /// <returns></returns>
        public int countValidIndicators()
        {
            int ncount = 0;
            for (int i = 0; i < optParamsList.Count; i++)
            {
                if (optParamsList[i].AnnualPerformance >= validPerformance)
                {
                    ncount++;
                }
            }
            return ncount;
        }

        /// <summary>
```

```
        /// Activate the best indicators
        /// </summary>
        private void activateBestIndicators()
        {
            for (int i = 0; i < optParamsList.Count; i++)
            {
                if (optParamsList[i].AnnualPerformance >= validPerformance)
                {
                    optParamsList[i].IsActivated = true;
                }
                else
                {
                    optParamsList[i].IsActivated = false;
                }
            }
        }

        /// <summary>
        /// The min performance of all indicators
        /// </summary>
        /// <returns></returns>
        private float getMinPerformance()
        {
            float minValue = 1000f;
            for (int i = 0; i < optParamsList.Count; i++)
            {
                if (optParamsList[i].AnnualPerformance < minValue)
                {
                    minValue = optParamsList[i].AnnualPerformance;
                }
            }
            return minValue;
        }

        /// <summary>
        /// The max performance of all indicators
        /// </summary>
        /// <returns></returns>
        private float getMaxPerformance()
        {
            float maxValue = -1000;
            for (int i = 0; i < optParamsList.Count; i++)
            {
                if (optParamsList[i].AnnualPerformance > maxValue)
                {
                    maxValue = optParamsList[i].AnnualPerformance;
                }
            }
            return maxValue;
        }

        public void evaluatePerformances()
        {
```

```
            float minPerformance = getMinPerformance();
            float maxPerformance = getMaxPerformance();

            for (int i = 0; i < optParamsList.Count; i++)
            {
                if (optParamsList[i].AnnualPerformance <= validPerformance)
                {
                    optParamsList[i].CoffEvaluation = 0;
                }
                else
                {
                    optParamsList[i].CoffEvaluation =
                        constMultiplier * (optParamsList[i].AnnualPerformance -
minPerformance) / (maxPerformance - minPerformance);
                }
            }
        }

        public IndicatorOptParam getIndicatorParams(string codeIndTgt)
        {
            for (int i = 0; i < optParamsList.Count; i++)
            {
                if (optParamsList[i].CodeInd.CompareTo(codeIndTgt) == 0)
                {
                    return optParamsList[i];
                }
            }
            return null;
        }

    }
}
```

Package (namespace) : Trades	**Source:** `OptRange.cs`

```
/// <summary>
/// Summary: description for OptRange class
/// -----------------------------------------------------------------
/// It contains the OptRange class
/// -----------------------------------------------------------------
/// Purpose: Holds the information of range of a parameter of an indicator
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;

namespace AIPredictor.Trades
{
    class OptRange
    {
        private float optMin;
        private float optMax;
        private float optStep;
        private string optDescr;

        /// <summary>
        /// Constractor method
        /// </summary>
        /// <param name="optMinValue"></param>
        /// <param name="optMaxValue"></param>
        /// <param name="optStepValue"></param>
        /// <param name="optDescrValue"></param>
        public OptRange(float optMinValue, float optMaxValue, float
optStepValue, string optDescrValue)
        {
            this.optMin   = optMinValue;
            this.optMax   = optMaxValue;
            this.optStep  = optStepValue;
            this.optDescr = optDescrValue;
        }

        // public property OptMin
        public float OptMin
        {
            get
            {
                return optMin;
            }
            set
            {
                optMin = value;
```

```
            }
        }

        // public property OptMax
        public float OptMax
        {
            get
            {
                return optMax;
            }
            set
            {
                optMax = value;
            }
        }

        // public property optStep
        public float OptStep
        {
            get
            {
                return optStep;
            }
            set
            {
                optStep = value;
            }
        }

        // public property OptDescr
        public string OptDescr
        {
            get
            {
                return optDescr;
            }
            set
            {
                optDescr = value;
            }
        }

    }
}
```

Package (namespace) : Trades	**Source:** OptRangeList.cs

```
/// <summary>
/// Summary: description for OptRangeList class
/// -----------------------------------------------------------
/// It contains the OptRangeList class
/// -----------------------------------------------------------
/// Purpose: Holds the list of all parameters of an indicator
/// </summary>
/// -----------------------------------------------------------
using System;
using System.Collections.Generic;
using System.Text;
namespace AIPredictor.Trades
{
    class OptRangeList
    {
        private List<OptRange> optRangeValues;

        /// <summary>
        /// Constractor method
        /// </summary>
        public OptRangeList()
        {
            this.optRangeValues = new List<OptRange>();
        }
        public List<OptRange> getListOpt()
        {
            return optRangeValues;
        }
        public OptRange getOptRange(int id)
        {
            return optRangeValues[id];
        }
        public void addOptRange(OptRange opt)
        {
            optRangeValues.Add(opt);
        }
        public void addOptRange(float optMinValue, float optMaxValue, float
optStepValue, string optDescrValue )
        {
            OptRange opt = new OptRange(optMinValue, optMaxValue, optStepValue,
optDescrValue);
            optRangeValues.Add(opt);
        }
    }
}
```

Package (namespace) : Trades	**Source:** ExecTrade.cs

```
/// <summary>
/// Summary: description for ExecTrade class
/// -----------------------------------------------------------
/// It contains the ExecTrade class
/// -----------------------------------------------------------
/// Purpose: It contains all trading systems. All optimization
///          methods for defining the best input parameters of indicators.
/// </summary>
/// -----------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using AIPredictor.BasicOper;
using AIPredictor.Signals;

namespace AIPredictor.Trades
{
    class ExecTrade
    {

        delegate bool CompareOp(object lhs, object rhs);

        private Security security;

        /// <summary>
        /// The constractor class
        /// </summary>
        /// <param name="sec"></param>
        public ExecTrade(Security sec)
        {
            this.security = sec;
        }

        /// <summary>
        /// Sorts the training results come from optimazation
        /// </summary>
        /// <param name="slArray"></param>
        /// <param name="gtMethod"></param>
        /// <returns></returns>
        private AITradeCalculator[] SortTrade(AITradeCalculator[] slArray,
CompareOp gtMethod)
        {
            for (int i = 0; i < slArray.Length; i++)
            {
                for (int j = i + 1; j < slArray.Length; j++)
                {
                    if (gtMethod(slArray[j], slArray[i]))
                    {
```

```
                        AITradeCalculator temp = slArray[i];
                        slArray[i] = slArray[j];
                        slArray[j] = temp;
                    }
                }
            }
            return slArray;
        }

        public AITradeCalculator[]  Sort(AITradeCalculator[] slArray)
        {
            CompareOp secCompareOp = new
CompareOp(AITradeCalculator.RhsIsGreaterByEarning);
            return SortTrade(slArray, secCompareOp);
        }

        /// <summary>
        /// Performs calculations of specific trading systems.
        /// RVI indicator trading is not included. Results are loaded into
object
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="listParams"></param>
        /// <returns></returns>
        public AITradeCalculator MakeTrade(string fromDateStr, string
nameOfTrade, string typeOfTradeStr, List<float> listParams)
        {
            if (nameOfTrade.CompareTo("SMA") == 0)
            {
                return Trade_SMA(fromDateStr, typeOfTradeStr,
(int)listParams[0]);
            }
            else if (nameOfTrade.CompareTo("EMA") == 0)
            {
                return Trade_EMA(fromDateStr, typeOfTradeStr,
(int)listParams[0]);
            }
            else if (nameOfTrade.CompareTo("CCI") == 0)
            {
                return Trade_CCI(fromDateStr, typeOfTradeStr,
(int)listParams[0], (int)listParams[1], (int)listParams[2]);
            }
            else if (nameOfTrade.CompareTo("PriceOsc") == 0)
            {
                return Trade_PriceOsc(fromDateStr, typeOfTradeStr,
(int)listParams[0], (int)listParams[1], (int)listParams[2]);
            }
            else if (nameOfTrade.CompareTo("Momentum") == 0)
            {
                return Trade_Momentum(fromDateStr, typeOfTradeStr,
(int)listParams[0], (int)listParams[1], (int)listParams[2]);
```

```
            }
            else if (nameOfTrade.CompareTo("ROC") == 0)
            {
                return Trade_ROC(fromDateStr, typeOfTradeStr,
(int)listParams[0], (int)listParams[1], (int)listParams[2]);
            }
            else if (nameOfTrade.CompareTo("RSI") == 0)
            {
                return Trade_RSI(fromDateStr, typeOfTradeStr,
(int)listParams[0], (int)listParams[1], (int)listParams[2]);
            }
            else if (nameOfTrade.CompareTo("StochOsc") == 0)
            {
                return Trade_StochOsc(fromDateStr, typeOfTradeStr,
(int)listParams[0], (int)listParams[1]);
            }
            else if (nameOfTrade.CompareTo("CMO") == 0)
            {
                return Trade_CMO(fromDateStr, typeOfTradeStr,
(int)listParams[0], (int)listParams[1], (int)listParams[2]);
            }
            else if (nameOfTrade.CompareTo("Chaikin") == 0)
            {
                return Trade_Chaikin(fromDateStr, typeOfTradeStr,
(int)listParams[0], (int)listParams[1], (int)listParams[2]);
            }
            else if (nameOfTrade.CompareTo("DMI") == 0)
            {
                return Trade_DMI(fromDateStr, typeOfTradeStr,
(int)listParams[0], (int)listParams[1]);
            }
            else

                return null;
        }

        /// <summary>
        /// Perform trade only for RVI indicator
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="listParams"></param>
        /// <param name="RVIListValues"></param>
        /// <returns></returns>
        public AITradeCalculator MakeTrade(string fromDateStr, string
nameOfTrade, string typeOfTradeStr, List<float> listParams, float[]
RVIListValues)
        {
            return Trade_RVI(fromDateStr, typeOfTradeStr, (int)listParams[0],
(int)listParams[1], (int)listParams[2], RVIListValues);
        }
```

```
        /// <summary>
        /// Optimazation Results
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList OptimizeTrade(string fromDateStr, string
nameOfTrade, string typeOfTradeStr, OptRangeList optlist)
        {
            if (nameOfTrade.CompareTo("SMA") == 0)
            {
                return Optimize_SMA(fromDateStr, nameOfTrade, typeOfTradeStr,
optlist);
            }
            else if (nameOfTrade.CompareTo("EMA") == 0)
            {
                return Optimize_EMA(fromDateStr, nameOfTrade, typeOfTradeStr,
optlist);
            }
            else if (nameOfTrade.CompareTo("RVI") == 0)
            {
                return Optimize_RVI(fromDateStr, nameOfTrade, typeOfTradeStr,
optlist);
            }
            else if (nameOfTrade.CompareTo("CCI") == 0)
            {
                return Optimize_CCI(fromDateStr, nameOfTrade, typeOfTradeStr,
optlist);
            }
            else if (nameOfTrade.CompareTo("PriceOsc") == 0)
            {
                return Optimize_PriceOsc(fromDateStr, nameOfTrade,
typeOfTradeStr, optlist);
            }
            else if (nameOfTrade.CompareTo("Momentum") == 0)
            {
                return Optimize_Momentum(fromDateStr, nameOfTrade,
typeOfTradeStr, optlist);
            }
            else if (nameOfTrade.CompareTo("ROC") == 0)
            {
                return Optimize_ROC(fromDateStr, nameOfTrade, typeOfTradeStr,
optlist);
            }
            else if (nameOfTrade.CompareTo("RSI") == 0)
            {
                return Optimize_RSI(fromDateStr, nameOfTrade, typeOfTradeStr,
optlist);
            }
            else if (nameOfTrade.CompareTo("StochOsc") == 0)
            {
```

```
                return Optimize_StochOsc(fromDateStr, nameOfTrade,
typeOfTradeStr, optlist);
            }
            else if (nameOfTrade.CompareTo("CMO") == 0)
            {
                return Optimize_CMO(fromDateStr, nameOfTrade, typeOfTradeStr,
optlist);
            }
            else if (nameOfTrade.CompareTo("Chaikin") == 0)
            {
                return Optimize_Chaikin(fromDateStr, nameOfTrade,
typeOfTradeStr, optlist);
            }
            else if (nameOfTrade.CompareTo("DMI") == 0)
            {
                return Optimize_DMI(fromDateStr, nameOfTrade, typeOfTradeStr,
optlist);
            }
            else
                return null;
        }

        /// <summary>
        /// Convertion of classes AITradeCalculator to TradeHolderList
        /// </summary>
        /// <param name="alTC"></param>
        /// <param name="desc"></param>
        /// <returns></returns>
        public TradeHolderList convertClasses(AITradeCalculator[] alTC, string
desc)
        {
            AITradeCalculator[] slTradeSorted = Sort(alTC);

            AITradeCalculator[] slBest10Trades;

            if (slTradeSorted.Length > 10)
                slBest10Trades = new AITradeCalculator[10];
            else
                slBest10Trades = new AITradeCalculator[slTradeSorted.Length];

            for (int i = 0; i < slBest10Trades.Length; i++)
            {
                slBest10Trades[i] = slTradeSorted[i];
            }

            TradeHolderList THList = new TradeHolderList();

            for (int i = 0; i < slBest10Trades.Length; i++)
            {

THList.addTradeHolderDet(slBest10Trades[i].convertToTradeHolder(desc));
            }
```

```
THList.addTradeHolderSum(slBest10Trades[0].convertToTradeHolder(desc));

            return THList;
        }

        /// <summary>
        /// Get results of trading for SMA by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="xPeriods"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_SMA(string fromDateStr, string
typeOfTradeStr, int xPeriods)
        {
            AITradeCalculator sl = new AITradeCalculator(security, "SMA",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(xPeriods);

            // start id
            int idStarting = xPeriods + 1;
            Boolean curCalcStatus  = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy  = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {
                curCalcStatus = (security.getClose(i) > Indicator.SMA(security,
i, xPeriods));
                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy  = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length>0))
            {
```

```
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }
            return sl;
        }

        /// <summary>
        /// Get results of trading for EMA by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="xPeriods"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_EMA(string fromDateStr, string
typeOfTradeStr, int xPeriods)
        {
            AITradeCalculator sl = new AITradeCalculator(security, "EMA",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(xPeriods);

            // start id
            int idStarting = xPeriods + 1;
            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {
                curCalcStatus = (security.getClose(i) > Indicator.EMA(security,
i, xPeriods));
                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
```

```
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
        }

        /// <summary>
        /// Get results of trading for RVI by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="xPeriods"></param>
        /// <param name="buyX"></param>
        /// <param name="sellY"></param>
        /// <param name="RVIListValues"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_RVI(string fromDateStr, string
typeOfTradeStr, int xPeriods, int buyX, int sellY, float[] RVIListValues)
        {
            AITradeCalculator sl = new AITradeCalculator(security, "RVI",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(xPeriods);
            sl.addParamValue(buyX);
            sl.addParamValue(sellY);

            // start id
            int idStarting = xPeriods;

            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {
                if (RVIListValues[i - idStarting] > buyX)
                    curCalcStatus = true;
                else if (RVIListValues[i - idStarting] < sellY)
                    curCalcStatus = false;

                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
```

```
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
        }

        /// <summary>
        /// Get results of trading for CCI by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="xPeriods"></param>
        /// <param name="buyX"></param>
        /// <param name="sellY"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_CCI(string fromDateStr, string
typeOfTradeStr, int xPeriods, int buyX, int sellY)
        {
            AITradeCalculator sl = new AITradeCalculator(security, "CCI",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(xPeriods);
            sl.addParamValue(buyX);
            sl.addParamValue(sellY);

            // start id
            int idStarting = xPeriods + 1;
//security.getIdFromDate(fromDateStr);

            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {

                if (Indicator.CCI(security, i, xPeriods) > buyX)
                    curCalcStatus = true;
                else if (Indicator.CCI(security, i, xPeriods) < sellY)
                    curCalcStatus = false;

                if (curCalcStatus != prevCalcStatus)
```

```
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
        }

        /// <summary>
        /// Get results of trading for Price Osc by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="shortMA"></param>
        /// <param name="longMA"></param>
        /// <param name="signalMA"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_PriceOsc(string fromDateStr, string
typeOfTradeStr, int shortMA, int longMA, int signalMA)
        {

            AITradeCalculator sl = new AITradeCalculator(security, "PriceOsc",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(shortMA);
            sl.addParamValue(longMA);
            sl.addParamValue(signalMA);

            // start id
            int idStarting = longMA + 1;
            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy = "";
```

```
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {
                curCalcStatus = (Indicator.PriceOsc(security, i, shortMA,
longMA) > Indicator.SMAPriceOsc(security, i, shortMA, longMA, signalMA));

                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
        }

        /// <summary>
        /// Get results of trading for Momentum by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="xPeriods"></param>
        /// <param name="buyX"></param>
        /// <param name="sellY"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_Momentum(string fromDateStr, string
typeOfTradeStr, int xPeriods, int buyX, int sellY)
        {
            AITradeCalculator sl = new AITradeCalculator(security, "Momentum",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(xPeriods);
            sl.addParamValue(buyX);
            sl.addParamValue(sellY);
```

```
            // start id
            int idStarting = xPeriods + 1;

            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {

                if (Indicator.MomentumOsc(security, i, xPeriods) > buyX)
                    curCalcStatus = true;
                else if (Indicator.MomentumOsc(security, i, xPeriods) < sellY)
                    curCalcStatus = false;

                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
        }

        /// <summary>
        /// Get results of trading for ROC by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="xPeriods"></param>
        /// <param name="buyX"></param>
        /// <param name="sellY"></param>
```

```
        /// <returns></returns>
        public AITradeCalculator Trade_ROC(string fromDateStr, string
typeOfTradeStr, int xPeriods, int buyX, int sellY)
        {
            AITradeCalculator sl = new AITradeCalculator(security, "ROC",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(xPeriods);
            sl.addParamValue(buyX);
            sl.addParamValue(sellY);

            // start id
            int idStarting = xPeriods + 1;

            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {

                if (Indicator.ROC(security, i, xPeriods) > buyX)
                    curCalcStatus = true;
                else if (Indicator.ROC(security, i, xPeriods) < sellY)
                    curCalcStatus = false;

                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
```

```
        }

        /// <summary>
        /// Get results of trading for RSI by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="xPeriods"></param>
        /// <param name="buyX"></param>
        /// <param name="sellY"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_RSI(string fromDateStr, string
typeOfTradeStr, int xPeriods, int buyX, int sellY)
        {
            AITradeCalculator sl = new AITradeCalculator(security, "RSI",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(xPeriods);
            sl.addParamValue(buyX);
            sl.addParamValue(sellY);

            // start id
            int idStarting = xPeriods + 1;

            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {

                if (Indicator.RSI(security, i, xPeriods) > buyX)
                    curCalcStatus = true;
                else if (Indicator.RSI(security, i, xPeriods) < sellY)
                    curCalcStatus = false;

                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
```

```
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
        }

        /// <summary>
        /// Get results of trading for StochOsc by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="xPeriods"></param>
        /// <param name="signalMA"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_StochOsc(string fromDateStr, string
typeOfTradeStr, int xPeriods, int signalMA)
        {

            AITradeCalculator sl = new AITradeCalculator(security, "StochOsc",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(xPeriods);
            sl.addParamValue(signalMA);


            // start id
            int idStarting = xPeriods + 1;
            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {
                curCalcStatus = (Indicator.StochOsc(security, i, xPeriods) >
Indicator.SMAStochOsc(security, i, xPeriods, signalMA));

                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
```

```
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
        }

        /// <summary>
        /// Get results of trading for CMO by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="xPeriods"></param>
        /// <param name="buyX"></param>
        /// <param name="sellY"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_CMO(string fromDateStr, string
typeOfTradeStr, int xPeriods, int buyX, int sellY)
        {
            AITradeCalculator sl = new AITradeCalculator(security, "CMO",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(xPeriods);
            sl.addParamValue(buyX);
            sl.addParamValue(sellY);

            // start id
            int idStarting = xPeriods + 1;

            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {

                if (Indicator.ChandeMomentumOsc(security, i, xPeriods) > buyX)
                    curCalcStatus = true;
                else if (Indicator.ChandeMomentumOsc(security, i, xPeriods) <
sellY)
                    curCalcStatus = false;
```

```
                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
        }

        /// <summary>
        /// Get results of trading for Chaikin by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="shortMA"></param>
        /// <param name="longMA"></param>
        /// <param name="signalMA"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_Chaikin(string fromDateStr, string
typeOfTradeStr, int shortMA, int longMA, int signalMA)
        {

            AITradeCalculator sl = new AITradeCalculator(security, "Chaikin",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(shortMA);
            sl.addParamValue(longMA);
            sl.addParamValue(signalMA);

            // start id
            int idStarting = longMA + 1;
            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
```

```
            string dateBuy = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {
                curCalcStatus = (Indicator.ChaikinAD(security, i, shortMA,
longMA) > Indicator.SMAChaikinAD(security, i, shortMA, longMA, signalMA));

                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
        }

        /// <summary>
        /// Get results of trading for DMI by applying specific parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="typeOfTradeStr"></param>
        /// <param name="xPeriods"></param>
        /// <param name="signalDX"></param>
        /// <returns></returns>
        public AITradeCalculator Trade_DMI(string fromDateStr, string
typeOfTradeStr, int xPeriods,  int signalDX)
        {

            AITradeCalculator sl = new AITradeCalculator(security, "DMI",
typeOfTradeStr, fromDateStr);
            sl.addParamValue(xPeriods);
            sl.addParamValue(signalDX);
```

```
            float[] listDMI = new float[3];

            // start id
            int idStarting = xPeriods + 1;
            Boolean curCalcStatus = false;   // false = sell signal     true =
buy signal
            Boolean prevCalcStatus = false;
            string dateBuy = "";
            string dateSell = "";
            for (int i = idStarting; i < security.CountValues(); i++)
            {
                listDMI = Indicator.DMI(security, i, xPeriods);
                curCalcStatus = (listDMI[1] > listDMI[0]);

                if (curCalcStatus)
                {
                    if (listDMI[2] < signalDX)
                        curCalcStatus = false;
                }

                if (curCalcStatus != prevCalcStatus)
                {
                    if (curCalcStatus)
                    {
                        prevCalcStatus = curCalcStatus;
                        dateBuy = security.getDate(i);
                        Signal s = new Signal(dateBuy, security.getClose(i));
                        sl.addSignal(s);
                        dateSell = "";
                    }
                    else
                    {
                        dateSell = security.getDate(i);
                        sl.SetInLastSellSignal(dateSell, security.getClose(i));
                        prevCalcStatus = curCalcStatus;
                        dateBuy = "";
                    }
                }
            }
            // final
            if ((dateSell.Length == 0) && (dateBuy.Length > 0))
            {
                sl.SetInLastSellSignal(security.getDate(security.CountValues() -
1), security.getClose(security.CountValues() - 1));
            }

            return sl;
        }

        /// <summary>
        /// Optimize trading for SMA by applying various parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
```

```
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_SMA(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;
            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                dim++;
            }
            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);
            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
           {
               listParams[0] = (float)i;
               slTrades[i - (int)optlist.getOptRange(0).OptMin] =
MakeTrade(fromDateStr, "SMA", typeOfTradeStrStr, listParams);
           }

            return convertClasses(slTrades, "parameter 1: days of MA");
        }

        /// <summary>
        /// Optimize trading for EMA by applying various parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_EMA(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;
            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                dim++;
            }

            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);
            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
```

```
                listParams[0] = (float)i;
                slTrades[i - (int)optlist.getOptRange(0).OptMin] =
MakeTrade(fromDateStr, "EMA", typeOfTradeStrStr, listParams);
            }

            return convertClasses(slTrades, "parameter 1: days of MA");
        }

        /// <summary>
        /// Optimize trading for RVI by applying various parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_RVI(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        dim++;
                    }
                }
            }

            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);  // xPeriods
            listParams.Add(0.0f);  // buy signal
            listParams.Add(0.0f);  // sell signal
            float[] RVIListValues;
            int ncounter = 0;
            int dim1 = 0;
            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                dim1 = security.CountValues() - i;
                RVIListValues = new float[dim1];
                for (int k = 0; k < dim1; k++)
                    RVIListValues[k] = Indicator.RVI(security, k + (int)i,
(int)i);
```

```
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        listParams[0] = (float)i;
                        listParams[1] = (float)j;
                        listParams[2] = (float)k;
                        slTrades[ncounter] = MakeTrade(fromDateStr, "RVI",
typeOfTradeStrStr, listParams, RVIListValues);
                        ncounter++;
                    }
                }
            }

            return convertClasses(slTrades, "param 1: days of RVI, param 2: Buy,
param 3:Sell"); ;
        }

        /// <summary>
        /// Optimize trading for CCI by applying various parameters
        ///
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_CCI(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        dim++;
                    }
                }
            }

            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);  // xPeriods
            listParams.Add(0.0f);  // buy signal
```

```
            listParams.Add(0.0f);  // sell signal

            int ncounter = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        listParams[0] = (float)i;
                        listParams[1] = (float)j;
                        listParams[2] = (float)k;
                        slTrades[ncounter] = MakeTrade(fromDateStr, "CCI",
typeOfTradeStrStr, listParams);
                        ncounter++;
                    }
                }
            }

            return convertClasses(slTrades, "param 1: days of CCI, param 2: Buy,
param 3:Sell"); ;
        }

        /// <summary>
        /// Optimize trading for Price Osc by applying various parameters
        ///
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_PriceOsc(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        dim++;
                    }
                }
```

```
            }

            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);  // xPeriods
            listParams.Add(0.0f);  // buy signal
            listParams.Add(0.0f);  // sell signal

            int ncounter = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        listParams[0] = (float)i;
                        listParams[1] = (float)j;
                        listParams[2] = (float)k;
                        slTrades[ncounter] = MakeTrade(fromDateStr, "PriceOsc",
typeOfTradeStrStr, listParams);
                        ncounter++;
                    }
                }
            }

            return convertClasses(slTrades, "param 1: days of short MA, param 2:
days of long MA, param 3:days for signal MA"); ;
        }

        /// <summary>
        /// Optimize trading for Momentum by applying various parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_Momentum(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
```

```
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        dim++;
                    }
                }
            }

            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);  // xPeriods
            listParams.Add(0.0f);  // buy signal
            listParams.Add(0.0f);  // sell signal

            int ncounter = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        listParams[0] = (float)i;
                        listParams[1] = (float)j;
                        listParams[2] = (float)k;
                        slTrades[ncounter] = MakeTrade(fromDateStr, "Momentum",
typeOfTradeStrStr, listParams);
                        ncounter++;
                    }
                }
            }

            return convertClasses(slTrades, "param 1: days of Momentum, param 2:
Buy, param 3:Sell"); ;
        }

        /// <summary>
        /// Optimize trading for ROC by applying various parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_ROC(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;
```

```
            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        dim++;
                    }
                }
            }

            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);  // xPeriods
            listParams.Add(0.0f);  // buy signal
            listParams.Add(0.0f);  // sell signal

            int ncounter = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        listParams[0] = (float)i;
                        listParams[1] = (float)j;
                        listParams[2] = (float)k;
                        slTrades[ncounter] = MakeTrade(fromDateStr, "ROC",
typeOfTradeStrStr, listParams);
                        ncounter++;
                    }
                }
            }

            return convertClasses(slTrades, "param 1: days of ROC, param 2: Buy,
param 3:Sell"); ;
        }

        /// <summary>
        /// Optimize trading for RSI by applying various parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
```

```
        /// <returns></returns>
        public TradeHolderList Optimize_RSI(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        dim++;
                    }
                }
            }

            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);  // xPeriods
            listParams.Add(0.0f);  // buy signal
            listParams.Add(0.0f);  // sell signal

            int ncounter = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        listParams[0] = (float)i;
                        listParams[1] = (float)j;
                        listParams[2] = (float)k;
                        slTrades[ncounter] = MakeTrade(fromDateStr, "RSI",
typeOfTradeStrStr, listParams);
                        ncounter++;
                    }
                }
            }

            return convertClasses(slTrades, "param 1: days of RSI, param 2: Buy,
param 3:Sell"); ;
        }

        /// <summary>
```

```
        /// Optimize trading for Stochastic by applying various parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_StochOsc(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                     dim++;
                }
            }

            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);  // xPeriods
            listParams.Add(0.0f);  //  signal MA

            int ncounter = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                        listParams[0] = (float)i;
                        listParams[1] = (float)j;

                        slTrades[ncounter] = MakeTrade(fromDateStr, "StochOsc",
typeOfTradeStrStr, listParams);
                        ncounter++;
                }
            }

            return convertClasses(slTrades, "param 1: days back, param 2:days
for signal MA"); ;
        }

        /// <summary>
        /// Optimize trading for CMO by applying various parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
```

```
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_CMO(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        dim++;
                    }
                }
            }

            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);  // xPeriods
            listParams.Add(0.0f);  // buy signal
            listParams.Add(0.0f);  // sell signal

            int ncounter = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        listParams[0] = (float)i;
                        listParams[1] = (float)j;
                        listParams[2] = (float)k;
                        slTrades[ncounter] = MakeTrade(fromDateStr, "CMO",
typeOfTradeStrStr, listParams);
                        ncounter++;
                    }
                }
            }

            return convertClasses(slTrades, "param 1: days of CMO, param 2: Buy,
param 3:Sell"); ;
```

```
        }

        /// <summary>
        /// Optimize trading for Chaikin by applying various parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_Chaikin(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        dim++;
                    }
                }
            }

            AITradeCalculator[] slTrades = new AITradeCalculator[dim];

            List<float> listParams = new List<float>();
            listParams.Add(0.0f);  // short days
            listParams.Add(0.0f);  // long days
            listParams.Add(0.0f);  // days MA signal
            int ncounter = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    for (int k = (int)optlist.getOptRange(2).OptMin; k <=
(int)optlist.getOptRange(2).OptMax; k = k + (int)optlist.getOptRange(2).OptStep)
                    {
                        listParams[0] = (float)i;
                        listParams[1] = (float)j;
                        listParams[2] = (float)k;
                        slTrades[ncounter] = MakeTrade(fromDateStr, "Chaikin",
typeOfTradeStrStr, listParams);
                        ncounter++;
                    }
```

```
                }
            }

            return convertClasses(slTrades, "param 1: days of short MA, param 2:
days of long MA, param 3:days for signal MA"); ;
        }
        /// <summary>
        /// Optimize trading for DMI by applying various parameters
        /// </summary>
        /// <param name="fromDateStr"></param>
        /// <param name="nameOfTrade"></param>
        /// <param name="typeOfTradeStrStr"></param>
        /// <param name="optlist"></param>
        /// <returns></returns>
        public TradeHolderList Optimize_DMI(string fromDateStr, string
nameOfTrade, string typeOfTradeStrStr, OptRangeList optlist)
        {
            int dim = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    dim++;
                }
            }
            AITradeCalculator[] slTrades = new AITradeCalculator[dim];
            List<float> listParams = new List<float>();
            listParams.Add(0.0f);  // xPeriods
            listParams.Add(0.0f);  //  signal DX
            int ncounter = 0;

            for (int i = (int)optlist.getOptRange(0).OptMin; i <=
(int)optlist.getOptRange(0).OptMax; i = i + (int)optlist.getOptRange(0).OptStep)
            {
                for (int j = (int)optlist.getOptRange(1).OptMin; j <=
(int)optlist.getOptRange(1).OptMax; j = j + (int)optlist.getOptRange(1).OptStep)
                {
                    listParams[0] = (float)i;
                    listParams[1] = (float)j;

                    slTrades[ncounter] = MakeTrade(fromDateStr, "DMI",
typeOfTradeStrStr, listParams);
                    ncounter++;
                }
            }
            return convertClasses(slTrades, "param 1: days back, param 2: Value
for DX"); ;
        }
    }
}
```

Package (namespace) : Trainning	**Source:** DB_TrainNN.cs

```
/// <summary>
/// Summary: description for DB_TrainNN class
/// -----------------------------------------------------------------
/// It contains the DB_TrainNN class
/// -----------------------------------------------------------------
/// Purpose: Holds the data the input and the output of the neural net in the
///          form of a DataTable object in order to be able displayed
///          in a dataGrid
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Collections;

namespace AIPredictor.Trainning
{
    class DB_TrainNN
    {
        // The private property is a DataTable Object
        private DataTable trainNNTable;
        private List<string> titles;

        /// <summary>
        /// Constractor method
        /// </summary>
        /// <param name="myTitles"></param>
        public DB_TrainNN(List<string> myTitles)
        {
            this.titles = myTitles;

            this.trainNNTable = CreateDBTrade();
        }

        /// <summary>
        /// Another constractor method
        /// </summary>
        /// <param name="myTitle1"></param>
        /// <param name="myTitle2"></param>
        /// <param name="isIncludeStockValues"></param>
        public DB_TrainNN(List<string> myTitle1, List<string> myTitle2, Boolean
isIncludeStockValues)
        {
            this.titles = new List<string>();

            if (isIncludeStockValues)
```

```
            {
                this.titles.Add("Open");
                this.titles.Add("Close");
                this.titles.Add("DayDifference");
            }

            for (int i = 0; i < myTitle1.Count; i++)
                this.titles.Add(myTitle1[i]);

            for (int i = 0; i < myTitle2.Count; i++)
                this.titles.Add(myTitle2[i]);

            this.trainNNTable = CreateDBTrade();
        }


        // The public property TrainNNTable
        public DataTable TrainNNTable
        {
            get
            {
                return trainNNTable;
            }
            set
            {
                trainNNTable = value;
            }
        }

        /// <summary>
        /// Creates the DataTable based on titles given
        /// </summary>
        /// <returns></returns>
        private DataTable CreateDBTrade()
        {
            DataTable nnTable = new DataTable("NNdb");

            for (int i = 0; i < titles.Count;i++)
                nnTable.Columns.Add(titles[i]);

            nnTable.Columns.Add("OutPut");
            nnTable.Columns.Add("Neural");
            return nnTable;
        }

        /// <summary>
        /// Adds raw data
        /// </summary>
        /// <param name="myRowIn"></param>
        /// <param name="myCellOut"></param>
        public void AddRow(double[] myRowIn, double myCellOut)
        {
```

```
            DataRow fileRow;

            // Add blank row
                fileRow = trainNNTable.NewRow();

                for (int i = 0; i < myRowIn.Length; i++)
            {
                // Fill row with data
                fileRow[titles[i]] = myRowIn[i];
            }
            fileRow["OutPut"] = myCellOut;
            fileRow["Neural"] = 0;
            trainNNTable.Rows.Add(fileRow);

        }
        /// <summary>
        /// Adds raw data included the output value of the neural net
        /// </summary>
        /// <param name="myRowIn"></param>
        /// <param name="myCellOut"></param>
        /// <param name="myNeuralOut"></param>
        public void AddRow(double[] myRowIn, double myCellOut, double
myNeuralOut)
        {

            DataRow fileRow;

            // Add blank row
            fileRow = trainNNTable.NewRow();

            for (int i = 0; i < myRowIn.Length; i++)
            {
                // Fill row with data
                fileRow[titles[i]] = myRowIn[i];
            }
            fileRow["OutPut"] = myCellOut;
            fileRow["Neural"] = myNeuralOut;
            trainNNTable.Rows.Add(fileRow);

        }

    }
}
```

Package (namespace) : Training	**Source:** **TrainNN.cs**

```
/// <summary>
/// Summary: description for TrainNN class
/// -----------------------------------------------------------------
/// It contains the TrainNN class
/// -----------------------------------------------------------------
/// Purpose: To train and test the neural net. It prepares the input data
///          and gets the results. It includes a lot of customization.
/// </summary>
/// -----------------------------------------------------------------

using System;
using System.Collections.Generic;
using System.Text;
using AIPredictor.SmartInput;
using AIPredictor.BasicOper;
using AIPredictor.outputs;
using NeuronDotNet.Core;
using NeuronDotNet.Core.Layers;
using NeuronDotNet.Core.Connections;

namespace AIPredictor.Trainning
{
    class TrainNN
    {
        private BasicStrategy basStrategy;
        private OptimStrategy optStrategy;
        private Security curSecurity;

        private Network N;

        private Layer inputLayer;
        private Layer hiddenLayer1;
        private Layer hiddenLayer2;
        private Layer hiddenLayer3;
        private Layer outputLayer;

        private int netTrainingCycles;

        private int numberOfTesting;

        private double[] outPutNet;

        private Boolean multiHiddenLayers;
        private Boolean includeStockvalues;
        private Boolean[] selectBasIndicators;

        private double percentageOfSuccessTrades;
        private double percentageOfUnSuccessTrades;
```

```
        private double percentangePredictions;

        private double buyWhen;
        private double sellWhen;

        /// <summary>
        /// Creates one hidden layer (no complexity)
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="mydb"></param>
        /// <param name="numberOfTestingValue"></param>
        /// <param name="numberOfCycles"></param>
        /// <param name="dimLessForlayer"></param>
        /// <param name="typeOfFunctions"></param>
        public TrainNN(Security sec, DBLayer mydb, int numberOfTestingValue, int
numberOfCycles, int dimLessForlayer,
                       int[] typeOfFunctions, Boolean[] selBasIndicators,
Boolean stockValues, double nAnnualPerformance)
        {
            this.basStrategy = new BasicStrategy(sec);
            this.optStrategy = new OptimStrategy(sec, mydb, nAnnualPerformance);
            this.curSecurity = sec;
            this.numberOfTesting = numberOfTestingValue;
            this.netTrainingCycles = numberOfCycles;
            this.multiHiddenLayers = false;
            this.includeStockvalues = stockValues;
            this.selectBasIndicators = selBasIndicators;
            this.buyWhen = 0.50;
            this.sellWhen = 0.50;

            // calculate the dimension of input array
            int dim = basStrategy.getNumberOfIndicators(selectBasIndicators) +
optStrategy.getNumberOfIndicators();
            if (includeStockvalues)
                dim = dim + 3;

            this.outPutNet = new double[curSecurity.CountValues()];
            for (int i = 0; i < curSecurity.CountValues(); i++)
                outPutNet[i] = 0;


            if (typeOfFunctions[0] == 1)
                inputLayer = new LinearLayer(dim);
            else if (typeOfFunctions[0] == 2)
                inputLayer = new SigmoidLayer(dim);
            else if (typeOfFunctions[0] == 3)
                inputLayer = new SineLayer(dim);
            else
                inputLayer = new LogarithmLayer(dim);

            int newDim = dim - dimLessForlayer;
            if (newDim < 4)
                newDim = 4;
```

```
            if (typeOfFunctions[1] == 1)
                hiddenLayer1 = new LinearLayer(newDim);
            else if (typeOfFunctions[1] == 2)
                hiddenLayer1 = new SigmoidLayer(newDim);
            else if (typeOfFunctions[1] == 3)
                hiddenLayer1 = new SineLayer(newDim);
            else
                hiddenLayer1 = new LogarithmLayer(newDim);

            if (typeOfFunctions[4] == 1)
                outputLayer = new LinearLayer(1);
            else if (typeOfFunctions[4] == 2)
                outputLayer = new SigmoidLayer(1);
            else if (typeOfFunctions[4] == 3)
                outputLayer = new SineLayer(1);
            else
                outputLayer = new LogarithmLayer(1);

            ConnectionFactory.EstablishCompleteConnection(inputLayer,
hiddenLayer1);
            ConnectionFactory.EstablishCompleteConnection(hiddenLayer1,
outputLayer);

            N = new Network(inputLayer);
            N.Initialize();
            N.TrainingCycles = netTrainingCycles;

            this.percentageOfSuccessTrades = 0;
            this.percentageOfUnSuccessTrades = 0;
            this.percentangePredictions = 0;

        }

        /// <summary>
        /// Creates 3 hidden Layers (a lot of complexity)
        /// </summary>
        /// <param name="sec"></param>
        /// <param name="mydb"></param>
        /// <param name="numberOfTestingValue"></param>
        /// <param name="numberOfCycles"></param>
        /// <param name="typeOfFunctions"></param>
        public TrainNN(Security sec, DBLayer mydb, int numberOfTestingValue, int
numberOfCycles, int[] typeOfFunctions,
                       Boolean[] selBasIndicators, Boolean stockValues, double
nAnnualPerformance)
        {
            this.basStrategy = new BasicStrategy(sec);
            this.optStrategy = new OptimStrategy(sec, mydb, nAnnualPerformance);
            this.curSecurity = sec;
            this.numberOfTesting = numberOfTestingValue;
            this.netTrainingCycles = numberOfCycles;
            this.multiHiddenLayers = true;
```

```
            this.includeStockvalues = stockValues;
            this.selectBasIndicators = selBasIndicators;
            this.buyWhen = 0.50;
            this.sellWhen = 0.50;

            // calculate the dimension of input array
            int dim = basStrategy.getNumberOfIndicators(selectBasIndicators) +
optStrategy.getNumberOfIndicators();

            if (includeStockvalues)
                dim = dim + 3;

            this.outPutNet = new double[curSecurity.CountValues()];
            for (int i = 0; i < curSecurity.CountValues(); i++)
                outPutNet[i] = 0;

            if (typeOfFunctions[0] == 1)
                inputLayer = new LinearLayer(dim);
            else if (typeOfFunctions[0] == 2)
                inputLayer = new SigmoidLayer(dim);
            else if (typeOfFunctions[0] == 3)
                inputLayer = new SineLayer(dim);
            else
                inputLayer = new LogarithmLayer(dim);

            if (typeOfFunctions[1] == 1)
                hiddenLayer1 = new LinearLayer(dim);
            else if (typeOfFunctions[1] == 2)
                hiddenLayer1 = new SigmoidLayer(dim);
            else if (typeOfFunctions[1] == 3)
                hiddenLayer1 = new SineLayer(dim);
            else
                hiddenLayer1 = new LogarithmLayer(dim);

            if (typeOfFunctions[2] == 1)
                hiddenLayer2 = new LinearLayer(dim);
            else if (typeOfFunctions[2] == 2)
                hiddenLayer2 = new SigmoidLayer(dim);
            else if (typeOfFunctions[2] == 3)
                hiddenLayer2 = new SineLayer(dim);
            else
                hiddenLayer2 = new LogarithmLayer(dim);

            if (typeOfFunctions[3] == 1)
                hiddenLayer3 = new LinearLayer(2 * dim);
            else if (typeOfFunctions[3] == 2)
                hiddenLayer3 = new SigmoidLayer(2 * dim);
            else if (typeOfFunctions[3] == 3)
                hiddenLayer3 = new SineLayer(2 * dim);
            else
                hiddenLayer3 = new LogarithmLayer(2 * dim);

            if (typeOfFunctions[4] == 1)
```

```
                outputLayer = new LinearLayer(1);
            else if (typeOfFunctions[4] == 2)
                outputLayer = new SigmoidLayer(1);
            else if (typeOfFunctions[4] == 3)
                outputLayer = new SineLayer(1);
            else
                outputLayer = new LogarithmLayer(1);


             ConnectionFactory.EstablishOneOneConnection(inputLayer,
hiddenLayer1);
             ConnectionFactory.EstablishOneOneConnection(inputLayer,
hiddenLayer2);
             ConnectionFactory.EstablishCompleteConnection(hiddenLayer1,
hiddenLayer3);
             ConnectionFactory.EstablishCompleteConnection(hiddenLayer2,
hiddenLayer3);
             ConnectionFactory.EstablishCompleteConnection(hiddenLayer3,
outputLayer);

            N = new Network(inputLayer);
            N.Initialize();
            N.TrainingCycles = netTrainingCycles;

            this.percentageOfSuccessTrades = 0;
            this.percentageOfUnSuccessTrades = 0;
            this.percentangePredictions = 0;
        }

        /// <summary>
        /// Set specific parameters referred to all neural
        /// </summary>
        /// <param name="jitter"></param>
        /// <param name="annealing"></param>
        /// <param name="lRateIncrease"></param>
        /// <param name="lRateDecrease"></param>
        public void setNetworkProperties(int jitter, int annealing, double
lRateIncrease, double lRateDecrease)
        {
            N.JitterEpoch = jitter;
            N.AnnealingEpoch = annealing;
            N.LearningRateIncreaseFactor = lRateIncrease;
            N.LearningRateDecreaseFactor = lRateDecrease;
        }

        /// <summary>
        /// Set specific parameters referred to all neural (values in string
format)
        /// </summary>
        /// <param name="jitter"></param>
        /// <param name="annealing"></param>
        /// <param name="lRateIncrease"></param>
        /// <param name="lRateDecrease"></param>
```

```
        public void setNetworkProperties(string jitter, string annealing, string
lRateIncrease, string lRateDecrease)
        {
            N.JitterEpoch = Convert.ToInt32(jitter);
            N.AnnealingEpoch = Convert.ToInt32(annealing);
            N.LearningRateIncreaseFactor = Convert.ToDouble(lRateIncrease);
            N.LearningRateDecreaseFactor = Convert.ToDouble(lRateDecrease);
        }

        /// <summary>
        /// Set neuron properties
        /// </summary>
        /// <param name="InpuLearningRate"></param>
        /// <param name="InputMomentum"></param>
        /// <param name="HiddenLearningRate"></param>
        /// <param name="HiddenMomentum"></param>
        /// <param name="OutputMaxJitter"></param>
        public void setLayerProperties(double InpuLearningRate, double
InputMomentum, double HiddenLearningRate,
                                       double HiddenMomentum, double
OutputMaxJitter )
        {
            inputLayer.LearningRate = InpuLearningRate;
            inputLayer.Momentum = InputMomentum;
            hiddenLayer1.LearningRate = HiddenLearningRate;
            hiddenLayer1.Momentum = HiddenMomentum;
            outputLayer.MaxJitter = OutputMaxJitter;

            if (multiHiddenLayers)
            {
                hiddenLayer2.LearningRate = HiddenLearningRate;
                hiddenLayer2.Momentum = HiddenMomentum;
                hiddenLayer3.LearningRate = HiddenLearningRate;
                hiddenLayer3.Momentum = HiddenMomentum;
            }
        }

        /// <summary>
        /// Set neuron properties (values in string format)
        /// </summary>
        /// <param name="InpuLearningRate"></param>
        /// <param name="InputMomentum"></param>
        /// <param name="HiddenLearningRate"></param>
        /// <param name="HiddenMomentum"></param>
        /// <param name="OutputMaxJitter"></param>
        public void setLayerProperties(string InpuLearningRate, string
InputMomentum, string HiddenLearningRate,
                                       string HiddenMomentum, string
OutputMaxJitter)
        {
            inputLayer.LearningRate = Convert.ToDouble(InpuLearningRate);
            inputLayer.Momentum = Convert.ToDouble(InputMomentum);
            hiddenLayer1.LearningRate = Convert.ToDouble(HiddenLearningRate);
```

```
            hiddenLayer1.Momentum = Convert.ToDouble(HiddenMomentum);
            outputLayer.MaxJitter = Convert.ToDouble(OutputMaxJitter);

            if (multiHiddenLayers)
            {
                hiddenLayer2.LearningRate =
Convert.ToDouble(HiddenLearningRate);
                hiddenLayer2.Momentum = Convert.ToDouble(HiddenLearningRate);
                hiddenLayer3.LearningRate =
Convert.ToDouble(HiddenLearningRate);
                hiddenLayer3.Momentum = Convert.ToDouble(HiddenLearningRate);
            }
        }

        /// <summary>
        /// Perform train to the neural
        /// </summary>
        /// <param name="typeOfOutput"></param>
        /// <returns></returns>
        public DB_TrainNN execTrainning(int typeOfOutput)
        {

            // construct input and output arrays
            int nLastTrainID = curSecurity.CountValues() - numberOfTesting;

            DB_TrainNN trainDB = new
DB_TrainNN(basStrategy.extractTitlesOfIndicators(selectBasIndicators),
optStrategy.extractTitlesOfIndicators(), includeStockvalues);

            OutputSignal outSignal = new OutputSignal(curSecurity,
numberOfTesting);

            List<double> basArray;
            List<double> optArray;


            double[] inputAll;

            double[] outputArray = outSignal.getOutput(typeOfOutput);
            double[] outputElement = new double[1];

            int dim = 0;
            int extraDim = 0;
            if (includeStockvalues)
                extraDim = 3;


            for (int i = 50; i < nLastTrainID; i++)
            {
                basArray = basStrategy.extractSmartInputArray(i,
selectBasIndicators);
                optArray = optStrategy.extractSmartInputArray(i);
```

```
                dim = basArray.Count + optArray.Count + extraDim;

                inputAll = new double[dim];

                if (includeStockvalues)
                {
                    inputAll[0] = curSecurity.getOpen(i);
                    inputAll[1] = curSecurity.getClose(i);
                    inputAll[2] = (curSecurity.getHigh(i) -
curSecurity.getLow(i));
                }

                for (int j = 0; j < basArray.Count; j++)
                    inputAll[j + extraDim] = basArray[j];

                for (int j = 0; j < optArray.Count; j++)
                    inputAll[j + basArray.Count + extraDim] = optArray[j];

                outputElement[0] = outputArray[i];

                N.TrainingSamples.Add(new TrainingSample(inputAll,
outputElement));
                trainDB.AddRow(inputAll, outputElement[0]);

            }

            N.Learn();


            return trainDB;
        }

        /// <summary>
        /// Save results of training in a file to be used later
        /// </summary>
        /// <param name="filename"></param>
        public void Save(string filename)
        {
            N.Save(filename);
        }

        /// <summary>
        /// utility: converts array to string
        /// </summary>
        /// <param name="ar"></param>
        /// <returns></returns>
        private String convertArrayToString(double[] ar)
        {
            String strAr = "";

            for (int j = 0; j < ar.Length; j++)
                strAr = strAr + Math.Round(ar[j],1).ToString() + "," ;
```

```
            return strAr;
        }

        /// <summary>
        /// Test the neural
        /// </summary>
        /// <param name="typeOfOutput"></param>
        /// <returns></returns>
        public DB_TrainNN testNN(int typeOfOutput)
        {

              // construct input and output arrays
              int nLastTrainID = curSecurity.CountValues() - numberOfTesting;

              DB_TrainNN trainDB = new
DB_TrainNN(basStrategy.extractTitlesOfIndicators(selectBasIndicators),
optStrategy.extractTitlesOfIndicators(), includeStockvalues);

              OutputSignal outSignal = new OutputSignal(curSecurity,
numberOfTesting);

              List<double> basArray;
              List<double> optArray;


              double[] inputAll;

              double[] outputArray = outSignal.getOutput(typeOfOutput);
              double[] outputElement = new double[1];
              double[] outputNeural = new double[1];

              int dim = 0;
              int extraDim = 0;
              if (includeStockvalues)
                  extraDim = 3;

              for (int i = nLastTrainID; i < curSecurity.CountValues(); i++)
              {
                  basArray = basStrategy.extractSmartInputArray(i,
selectBasIndicators);
                  optArray = optStrategy.extractSmartInputArray(i);

                  dim = basArray.Count + optArray.Count + extraDim;

                  inputAll = new double[dim];

                  if (includeStockvalues)
                  {
                      inputAll[0] = curSecurity.getOpen(i);
                      inputAll[1] = curSecurity.getClose(i);
                      inputAll[2] = (curSecurity.getHigh(i) -
curSecurity.getLow(i));
                  }
```

```
                for (int j = 0; j < basArray.Count; j++)
                    inputAll[j] = basArray[j];

                for (int j = 0; j < optArray.Count; j++)
                    inputAll[j + basArray.Count] = optArray[j];

                outputElement[0] = outputArray[i];


                outputNeural = N.Run(inputAll);

                trainDB.AddRow(inputAll, outputElement[0], outputNeural[0]);

                outPutNet[i] = outputNeural[0];

            }

            calculatePerformanceInTrades();

            return trainDB;

        }

        /// <summary>
        /// Calculate the performance of the neural based on specific values
        /// </summary>
        /// <param name="thresForBuy"></param>
        /// <param name="thresForSell"></param>
        /// <returns></returns>
        public double calcPerformance(double thresForBuy, double thresForSell)
        {
            double[] signalList = new double[curSecurity.CountValues()];

            //init values all signals = sell
            signalList[0] = 0;
            for (int i = 1; i < curSecurity.CountValues(); i++)
            {
                if (outPutNet[i] > thresForBuy)
                    signalList[i] = 1;
                else if ((outPutNet[i] >= thresForSell) && (signalList[i - 1] ==
1))
                    signalList[i] = 1;
                else
                    signalList[i] = 0;
            }

            int endCounting = curSecurity.CountValues()-1;
            int startCounting = endCounting - numberOfTesting;

            int startType = 0;  // 0 = out of market, 1 = exec position
            int posEntrance = -1;
            int posExit = -1;
```

```
            double win_loss = 0; // per stock

            for (int i = startCounting; i < endCounting; i++)
            {
                if ((signalList[i] == 1) && (signalList[i - 1] == 0))
                {
                    posEntrance = i;
                }

                if ((signalList[i] == 0) && (signalList[i - 1] == 1))
                {
                    posExit = i;
                    startType = 1;
                }

                if (startType == 1)
                {
                    startType = 0;  // reset
                    win_loss = win_loss + (curSecurity.getClose(posExit) -
curSecurity.getClose(posEntrance));
                    posEntrance = -1; // reset
                }
            }

            // close last position
            if (posEntrance > 0)
            {
                win_loss = win_loss + (curSecurity.getClose(endCounting) -
curSecurity.getClose(posEntrance));
            }

            return win_loss;
        }

        /// <summary>
        /// maximize the performance of the neural by finding the best values
        /// for buy and sell. Normally if the result is greater than 0.5 is buy
signal
        /// and if less than 0.5 is sell signal. These values are adjusted for
better
        /// performance.
        /// </summary>
        public void maximizePerformance()
        {
            double tmpStart = 0.35;

            double maxPerformance = -1000;
            double tempPerformance = -1000;

            double stepIncr = 0.01;

            for (int i = 1; i <= 40; i++)
```

```
            {
                for (int j = i; j >= 1; j--)
                {
                    tempPerformance = calcPerformance(tmpStart + stepIncr *
(double)i, tmpStart + stepIncr * (double)j);

                    if (tempPerformance > maxPerformance)
                    {
                        maxPerformance = tempPerformance;
                        this.buyWhen = tmpStart + stepIncr * (double)i;
                        this.sellWhen = tmpStart + stepIncr * (double)j;
                    }
                }

            }
        }

        /// <summary>
        /// Calculations is based on type output 2 which determines what will
happen next day
        /// </summary>
        public void calculatePerformanceInTrades()
        {
            OutputSignal outSignal = new OutputSignal(curSecurity,
numberOfTesting);
            double[] outputArray = outSignal.getOutput(2);
            int nLastTrainID = curSecurity.CountValues() - numberOfTesting;

            int ncountSuccess = 0;
            int ncountAllSuccess = 0;
            int ncountUnSuccess = 0;
            int ncountAllUnSuccess = 0;
            int ncountPredictions = 0;
            double tmpPrediction = -1;

            // calculate % success
            for (int i = nLastTrainID + 1; i < curSecurity.CountValues(); i++)
            {
                if (outputArray[i] == 1)  // check neural output
                {
                    ncountAllSuccess = ncountAllSuccess + 1;
                    if (outPutNet[i] > 0.5)
                        ncountSuccess = ncountSuccess + 1;
                }
                else
                {
                    ncountAllUnSuccess = ncountAllUnSuccess + 1;
                    if (outPutNet[i] < 0.5)
                        ncountUnSuccess = ncountUnSuccess + 1;
                }

                if (outPutNet[i] > 0.5)
                    tmpPrediction = 1;
```

```
                else
                    tmpPrediction = 0;

                if (tmpPrediction == outputArray[i])
                    ncountPredictions = ncountPredictions +1;

                tmpPrediction = -1;
            }

            percentageOfSuccessTrades = 100 * ((double)ncountSuccess /
(double)ncountAllSuccess);
            percentageOfUnSuccessTrades = 100 * ((double)ncountUnSuccess /
(double)ncountAllUnSuccess);
            percentangePredictions = 100 * ((double)ncountPredictions /
(double)(curSecurity.CountValues() - nLastTrainID - 1));
        }


        public double PercentageOfSuccessTrades
        {
            get
            {
                return percentageOfSuccessTrades;
            }
            set
            {
                percentageOfSuccessTrades = value;
            }
        }

        public double PercentageOfUnSuccessTrades
        {
            get
            {
                return percentageOfUnSuccessTrades;
            }
            set
            {
                percentageOfUnSuccessTrades = value;
            }
        }

        public double PercentangePredictions
        {
            get
            {
                return percentangePredictions;
            }
            set
            {
                percentangePredictions = value;
            }
        }
```

```
        public double BuyWhen
        {
            get
            {
                return buyWhen;
            }
            set
            {
                buyWhen = value;
            }
        }

        public double SellWhen
        {
            get
            {
                return sellWhen;
            }
            set
            {
                sellWhen = value;
            }
        }
    }
}
```

10. References

A. Books:

1. Artificial Intelligence in the Capital Markets, Roy S. Freedman, Robert A. Klein & Jess Lederman
2. Trend Forecasting with Technical Analysis, Louis B. Mendelsohn
3. Technical Analysis Explained, Third Edition, martin J. Pring
4. Chaos and order in the capital Markets, Second Edition, “The bible of market chaologists”, Edgar E. Peters
5. Neural Networks in the Capital Markets, Apostolos-Paul Refenes
6. Training on the Edge, Neural, Geentic, and Fuzzy Systems for chaotic Financial Markets, Guido J. Deboeck
7. Fractal Market Analysis, Applying Chaos Theory to Investment & Economics, Edgar E.Peters
8. Technical Analysis From A to Z, Steven B. Achelis
9. Essential Technical Analysis – Tools and Techniques to Spot Market Trends, Leigh Stevens
10. Candlestick Charting – Timeless Techniques for Trading Stocks and Futures, Gregory L. Morris
11. “Big Profit Patterns Using Candlestick Signals and Gaps, Copyright by Stephen W. Bigalow 2002, Published by the Candlestick Forum LLC.
12. Japanese Candlestick Charting Techniques – A Contemporary Guide to the Ancient Investment Techniques of the Far East – STEVE NISON
13. JOHN J. MURPHY, Intermarket Technical Analysis – Trading Strategies for the Global Stock, Bond, Commodity and Currency markets.
14. Trading Chaos – Applying Expert Techniques to Maximize your Profits, Bill Williams, PhD
15. Market Neural Strategies, Bruce I. Jacobs & Kenneth N. Levy editors
16. Modeling Financial Markets – Using Visual Basic and Databases to Create Pricing, Trading, and Risk Management – Benjamin Van Vliet and Robert Hendry
17. Technical Analysis of the Financial Markets – A comprehensive guide to trading methods and Applications – John J. Murphy
18. Trading Systems and Methods – Third Edition Perry J. Kaufman
19. Pattern Classification – Second Edition – Richard O. Duda, Peter E. Hart, David G. Stork

20. Neural Network Design, Martin T. Hagan, Howard B. Domuth, Mark Beale
21. Handbook of Neural Network Signal Processing, YU HEN HU JENQ-NENG HWANG
22. C++ Neural Networks and Fuzzy Logic, Valluru B. Rao, IDG Books Worldwide,Inc.

B. Published Papers

1. “Mastering Short-Term Trading Through Technical Analysis” with Alan Farley
2. “Recent Developments of Self-Organising Modeling in Prediction and Analysis of Stock Market”, Ivakhnenko, A.G.
3. “Mining for Profitable Patterns in the Stock Market”, Yihua Philip Sheng – Southern Illinois University USA
4. “Prediction of the Next Stock Price using Neural Network for Data Mining”, Hiroshi Takaho, Takayuki Arai, Tsuyoshi Otake and Mamoru Tanaka
5. “Exploring the Fuzzy nature of Technical Patterns of U.S. Stock Market”, Ming Dong, Xu-Shen Zhou
6. “A Hybrid Time Lagged Network for Predicting Stock Prices”, SC Hui, M.T. Yap and P. Prakash
7. “Stock Market Prediction Using Artificial Neural Networks”, Birgul Egeli, Meltem Ozturan, Bertan Badur

C. Stock & Commodities CD Version 8. Articles

1. Testing Exit Strategies by Jeffrey Owen Katz, PhD and Donna L. McCormick
2. Genetic Algorithms and Rule Based Systems, by Jeffrey Owen Katz, PhD and Donna L. McCormick
3. Developing Systems with a Rule-based Approach, by Jeffrey Owen Katz, PhD and Donna L. McCormick
4. A Genetic Algorithm System For Predicting the OEX, by Deniz Yuret and Michael de la Maza
5. Trading Fuzzy Patterns by Marge Sherald
6. Fuzzy Expert Systems, by J.F. Derry

7. Using Fuzzy Logic in Expert Systems, by James F. Derry.
8. Neural Nets in Technical Analysis, by Yin Lung Shih
9. A Hybrid System For Market Timing, by Mark B. Fishman and Dean S. Barr
10. A Neural Network System For Reliable Trading Signals, by Marlowe D. Cassetti
11. Neural Net Input Optimization, by Kyle M. Druey
12. Designing A Personal Neural Net Trading System, by James Stakelum
13. Developing Neural Network Forecasters For Trading, by Jeffrey Owen Katz, PhD
14. Trading Neural Networks, by Lou Mendelsohn
15. Preprocessing Data For Neural Networks, by Lou Mendelsohn
16. Using Neural Networks For Financial Forecasting, by Lou Mendelsohn
17. Neural networks: A Trading Perspective, by Carot H. Halquist & George F. Schmoll, III
18. Optimizing Momentum, by Anthony W. Warren, Ph.D
19. Trading The E-Mini using the Endpoint Fast Fourier Transform, by Denis Meyers
20. The Endpoint Fast Fourier Transform, by Denis Meyers

D. Web sites

1. www.traders.com/ "Stock & Commodities Traders' magazine"
2. http://www.equis.com/ "Metastock software product"
3. http://www.guppytraders.com/ "Metastock Formulas"
4. http://www.meta-formula.com/index.html "Articles and Formulas"
5. http://trader.online.pl "Metastock Zone"