Machine Learning applied to the Stocks Market - R lang

by Carmel Shvartzman

2021

1. Introduction / Executive summary

Artificial Intelligence is referred as systems that *perceive its environment and take actions that maximize its chance of achieving its goals.* Artificial Intelligence applications include advanced web search engines, human speech understanding, self-driving cars, automated decision-making, Recommendation Systems and *Machine Learning Algorithms.*

The present paper presents an application of **Machine Learning algorithms** to the Stocks Market. For this sake, Technical Analysis features will be used. Stocks prices Technical Analysis is a trading discipline employed to evaluate investments and identify trading opportunities by analyzing statistical trends gathered from trading activity, such as price movement and volume [1]. In this project, different Machine Learning methods, such as Neural Networks, Random Forests, Support Vector Machines and Naive Bayes, will be applied to the Stocks Market daily prices, in order to predict the movement of the stocks the next day.

The algorithms used here, will be applied to stocks from two important companies, Apple Inc and Visa Inc. Apple is the largest information technology company by revenue and, since January 2021, the world's most valuable company. Visa facilitates electronic funds transfers throughout the world, most commonly through the "Visa" branded credit cards, debit cards and prepaid cards.

For the development of the algorithms, the statistic R language was used, since it is a language specifically created for statistics, by statisticians[2]. It includes a full battery of Machine Learning algorithms[3], and its graphics support is outstanding[4].

The steps taken in order to build this Machine Learning project, are depicted in the present Report, which includes several sections:

At first, in the **Data Analysis Section**, after loading the stocks prices from Yahoo Finance, an exploratory analysis of the data will be done, to get insights that would help in the development of the Models. Because we'll successively use data from different companies, specifically Apple Inc and Visa Inc, we'll have to split this **Data Analysis Section** twice, to show the data loaded. In this sections, which will be spread inside the models, the dataset will be analyzed, data exploration will be performed, and visualization of the data will be thoroughly rendered.

Following the exploratory section, in the **Modeling Section**, six different models will be developed, to attain better predictions. These models will start from two Support Vector Machines (SVM) models, which will use two different kernels, the Polynomial kernel and the Sigmoid Kernel, and, passing through a Naive Bayes algorithm, the models will progressively become more complicated, with the utilization of a Random Forests Model, and finally, a Neural Network Model, developed using resilient back-propagation with weight backtracking.

Neural networks are Machine Learning Supervised Algorithms analogous to the human nervous system, made up of interconnected nodes called *neurons*, which are used as interconnected information processing units. Similarly to the human nervous system, inside a Neural Network the information flows through interconnected units called Neurons, in a analogous manner to the information passage through neurons in animals. In this project, 20 neural networks will be compared, using two different software providers: the Caret package, and the Neuralnet package.

The algorithms developed on this project will use several Stocks Market trading technical analysis indicators, such as Reversal Candles and Moving Averages. A Reversal is when a prices trend change to the opposite direction. Using the notation known as Japanese Candles, throughout the project we'll call the stocks prices "candles" The Reversal Candle known as "Doji", representing indecision in the market, meaning that the battle between sellers and buyers have not yet seem to commit in either direction, will be used in two of its forms: the Dragonfly Doji, considered a bullish (UP) reversal pattern, and the Gravestone Doji, a bearish(DOWN) reversal pattern.

Also, the algorithms used on the present project, will compute on Moving Averages. Considered trend following indicators, Moving averages show the trend of the market. They are useful in order to follow the prices trend. On the present project, we will consider both the relative position of the stocks prices to the Moving Averages, and also the crossings occurred between the Moving Averages themselves.

While training the Machine Learning models, at least two formulas will be used: we'll use both a narrow number of the fields performing an univariant training, and we'll be using other formulas including all the predictors.

The predictors created will be normalized and scaled, to be sure that each variable contributes equally to the analysis. Afterwards, factorization will be performed, in order to comply with training requisites.

The prices dataset will be partitioned in Train set and Test Validation, containing the required continuity between the yesterday-today-tomorrow candles, because this is what the training algorithms will try to research and reveal: the patterns of continuity between consecutive candles. Therefore, we cannot just take prices hazardously and insert them in both training set and test set. Two methods of data partitioning will be used: we'll perform split holdout method to test the Models, and also K-fold Cross Validation, taking different chunks of data to validate, but always continuous data.

All along this process of building the models, their corresponding Accuracies will be computed, and the misclassifications committed will also be assessed. We'll perform Z-value, P-value, and AIC Analysis of the results, and a ROC Curve will be output. In addition to the accuracy and missclassification calculations, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and confusion matrix will be computed, for checking performance achievements. Also, the revenue money predicted return will be calculated for some of the models, as an example of how can be computed the income obtained by applying the algorithms.

And after all the six models had been validated against the Validation datasets, a **Results Section**, will present the modeling results and will discuss the model performance.

Also, a **Conclusion Section** will give a summary of this Report, its limitations and future work that could be done.

References

1. Hayes, Adam. 2021 . "Technical Analysis" . Investopedia.com
2. Rafael A. Irizarry (2021), Data Analysis and Prediction Algorithms with R https://rafalab.github.io/dsbook/introduction-to-productivity-tools.html
3. https://data-flair.training/blogs/pros-and-cons-of-r-programming-language/
4. https://techvidvan.com/tutorials/pros-and-cons-of-r/

2. Methods/Analysis Section - Visa Inc. Case

This Section explains the process and techniques used, performs data exploration and data visualization, summarizes the insights gained, and addresses the necessity of performing data cleaning. After the analysis of the data, comes a thorough explanation on the modeling approaches to the issue.

2.1 Techniques for Data Loading and Preparation

2.1.1 Loading Visa Stocks prices from Yahoo Finance

Two known Stocks will be downloaded and used for the analysis: Apple and Visa stocks. Here the Visa Stocks prices will be loaded and used for training and testing some of the Machine Learning models.

```
# Trying to download the Visa stocks data from Yahoo:
# the data will be intended to be downloaded 10 times:
i <- 1
maxTries <- 10
while (i < maxTries) {

  try(
    getSymbols("V",
         from ="2017-01-01",
         to = "2020-12-31",
         src =  "yahoo",
         adjust =  TRUE)
  )
  if (exists("V")) {
    break
  }
   i <- i + 1
}
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
## Warning in read.table(file = file, header = header, sep = sep,
## quote = quote, : incomplete final line found by readTableHeader
## on 'https://query1.finance.yahoo.com/v7/finance/download/V?
## period1=-2208988800&period2=1638835200&interval=1d&events=split&crumb=eBSm4.CeONz'
```

```
## Warning in read.table(file = file, header = header, sep = sep,
## quote = quote, : incomplete final line found by readTableHeader
## on 'https://query2.finance.yahoo.com/v7/finance/download/V?
## period1=-2208988800&period2=1638835200&interval=1d&events=split&crumb=eBSm4.CeONz'
```

```
# Create a dataframe with the downloaded data:
price <- as.data.frame(V)
price$Date <- as.Date(rownames(price))
head(price)
```

```
##              V.Open   V.High    V.Low  V.Close V.Volume V.Adjusted       Date
## 2017-01-03 76.72118 77.84141 76.45817 77.44202 13785200   76.96923 2017-01-03
```

```
## 2017-01-04 77.38358 78.18235 77.30565 78.07520  8033100   77.59854 2017-01-04
## 2017-01-05 78.26027 79.41947 78.14339 78.99086  7805100   78.50861 2017-01-05
## 2017-01-06 79.19543 80.35462 78.84475 80.08187  8898300   79.59296 2017-01-06
## 2017-01-09 80.15006 80.31565 79.53637 79.63378  7305300   79.14760 2017-01-09
## 2017-01-10 79.58507 79.77990 79.02983 79.20517  6017500   78.72161 2017-01-10
```

```
names(price) <- c("Open","High","Low","Close","Volume","Adjusted","Date")

# Discarding the columns that are not needed:
price <- price %>% select(c(Date,Open,High,Low,Close))
price<-xts(price[, -1, drop=FALSE],  order.by=price[,1])
```

2.2 Data Analysis Exploration & Modeling

In this section, we analyze the data provided and the relations between the predictors.

2.2.1 Preliminar Observations

At this stage, we take a look at the loaded dataset:

```
# Prices dataset glimpse:
nrow(price)
```

```
## [1] 1006
```

```
str(`price`)
```

```
## An 'xts' object on 2017-01-03/2020-12-30 containing:
##   Data: num [1:1006, 1:4] 76.7 77.4 78.3 79.2 80.2 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:4] "Open" "High" "Low" "Close"
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
class(price)
```

```
## [1] "xts" "zoo"
```

```
head(price)
```

```
##                Open     High      Low    Close
## 2017-01-03 76.72118 77.84141 76.45817 77.44202
## 2017-01-04 77.38358 78.18235 77.30565 78.07520
## 2017-01-05 78.26027 79.41947 78.14339 78.99086
## 2017-01-06 79.19543 80.35462 78.84475 80.08187
## 2017-01-09 80.15006 80.31565 79.53637 79.63378
## 2017-01-10 79.58507 79.77990 79.02983 79.20517
```

```
length(price)
```

```
## [1] 4024
```

```
ncol(price)
```

```
## [1] 4
```

At first glance, we can see that the data.frame is in tidy format, which means that each row represents only one observation, being the column names the prices features.

2.2.2 Features Analysis

2.2.2.1 - Analysis of the Stock Prices

Let's see some general data on the prices:

```
summary(price$Close)
```

```
##       Index                Close        
##  Min.   :2017-01-03   Min.   : 77.44  
##  1st Qu.:2018-01-02   1st Qu.:112.22  
##  Median :2019-01-02   Median :141.96  
##  Mean   :2019-01-01   Mean   :146.24  
##  3rd Qu.:2020-01-01   3rd Qu.:179.03  
##  Max.   :2020-12-30   Max.   :218.36
```

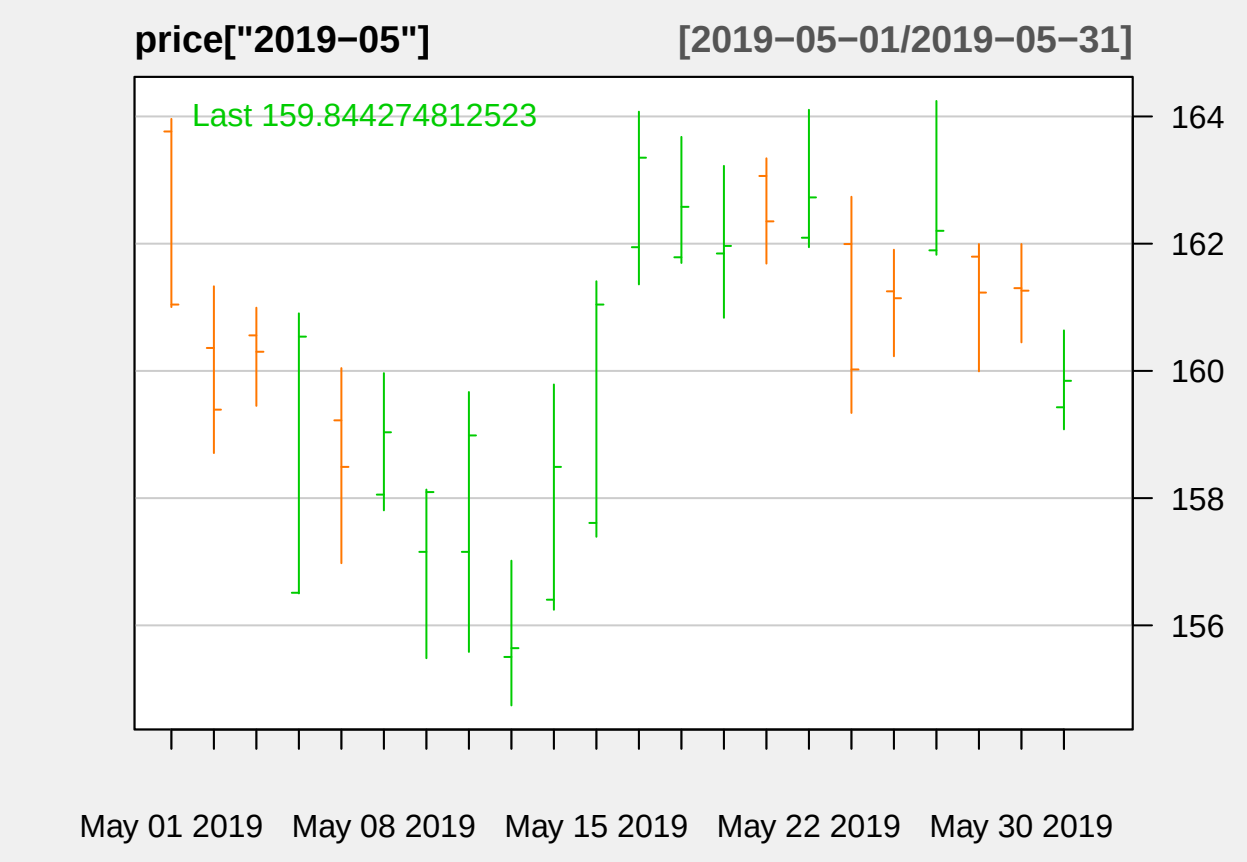
The mean closing price of the Visa stock is 146.24, the Minimal price is 77.44, and the maximal closing price is 218.36 usd.

Creating a custom Theme for be used on the graphs GGPLOT2:

```
# Creating a custom Theme to be used on graphs :
neuralTheme <- theme_fivethirtyeight() +
  theme(panel.background = element_rect(fill = "aliceblue"),
        text = element_text( face = "italic",
                             colour = "black", size = 8,
                             lineheight = 0.9,  hjust = 0.5,
                             vjust = 0.5, angle = 0,
                             #margin = margin(),
                             debug = FALSE),
        axis.title = element_text(colour = "gray" ))
```

To get a first approach to the Technical Trading Analysis features, the following graph shows the Visa prices of May 2019, represented as Bar icons, which place greater emphasis on the closing price of the stock in relation to the previous day closing prices.

```
barChart(price['2019-05'], theme = "white") + neuralTheme
```

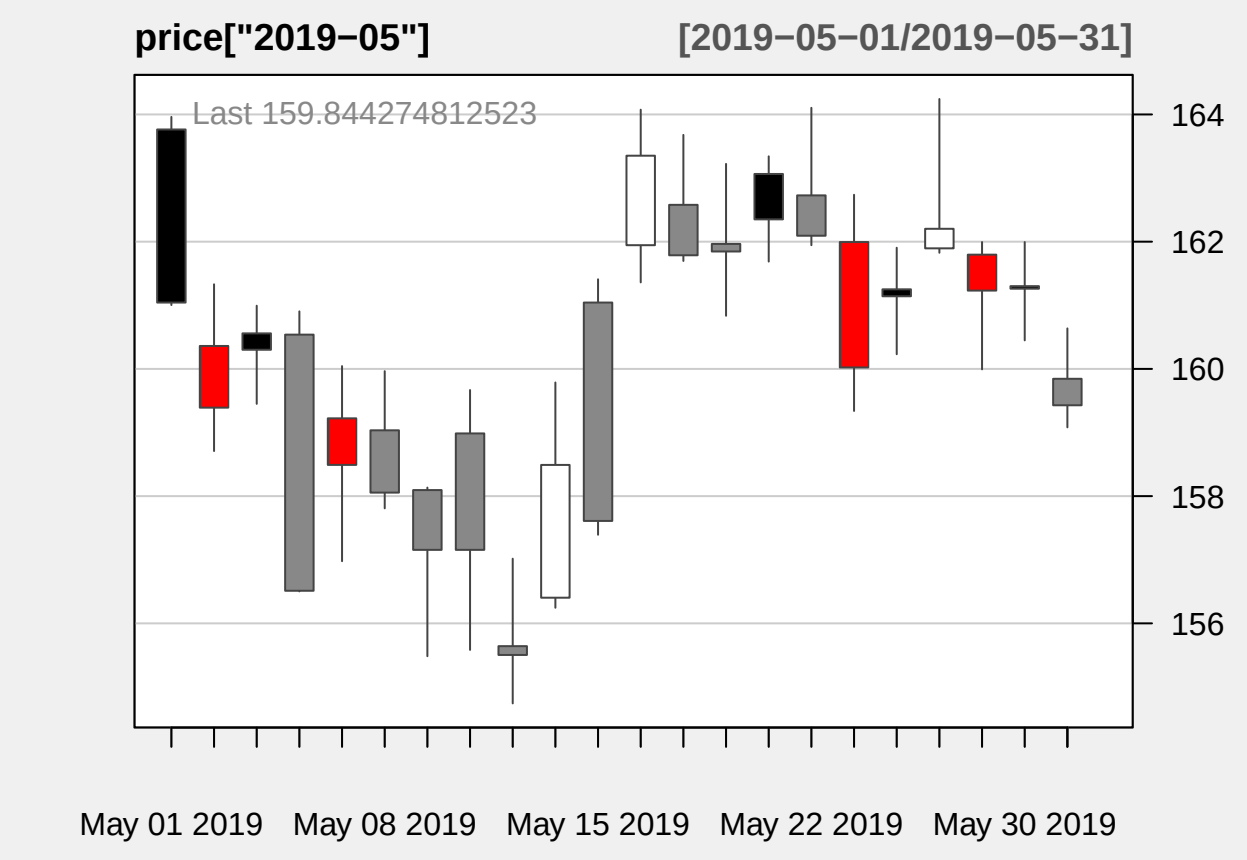



```
## NULL
```

However, the candlestick charts place the highest importance of the close as it relates to the open of the same day, which can be seen on the following charts, which show the same data for Visa stocks, using Japanese Candles, and two different formats:

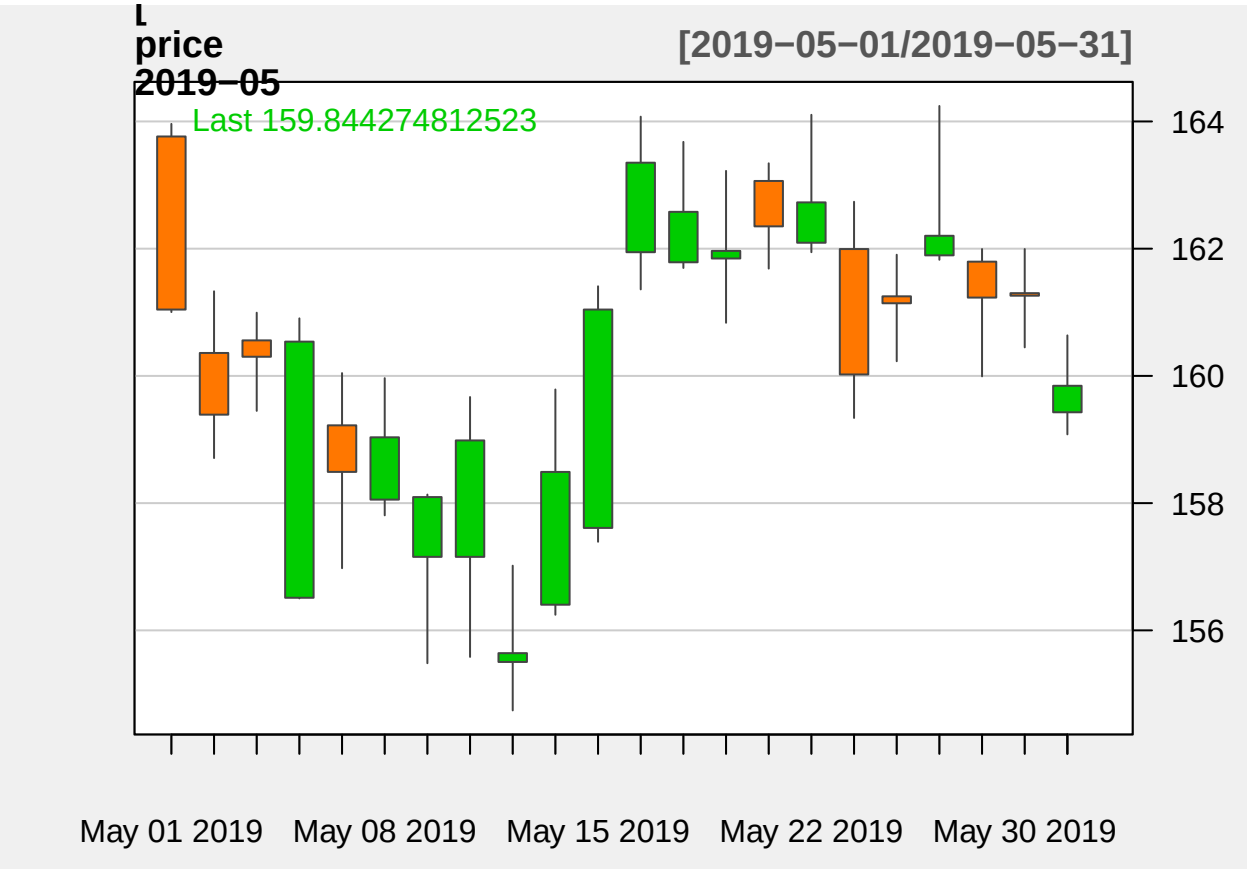
Another two ways of seeing the Ratings distribution :

```
candleChart(price['2019-05'],multi.col=TRUE,theme="white") + neuralTheme
```




```
## NULL
```

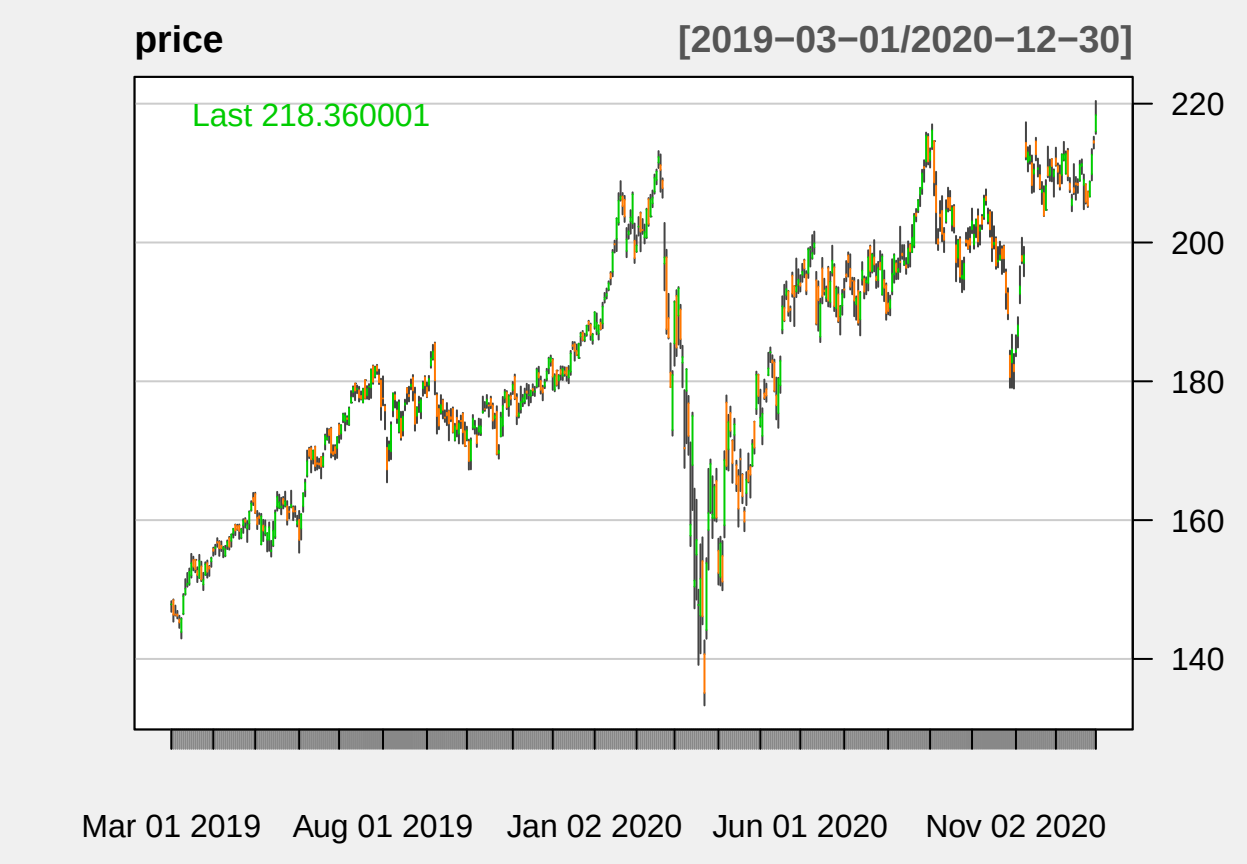
```
chartSeries(price['2019-05'], theme = "white")  + neuralTheme
```




```
## NULL
```

When representing longer periods of data, the candlesticks charts can become quite difficult to follow. This two charts show the prices for the entire period of stock prices loaded for Visa :

```
chartSeries(price, subset = "2019-03::",theme = "white") + neuralTheme
```

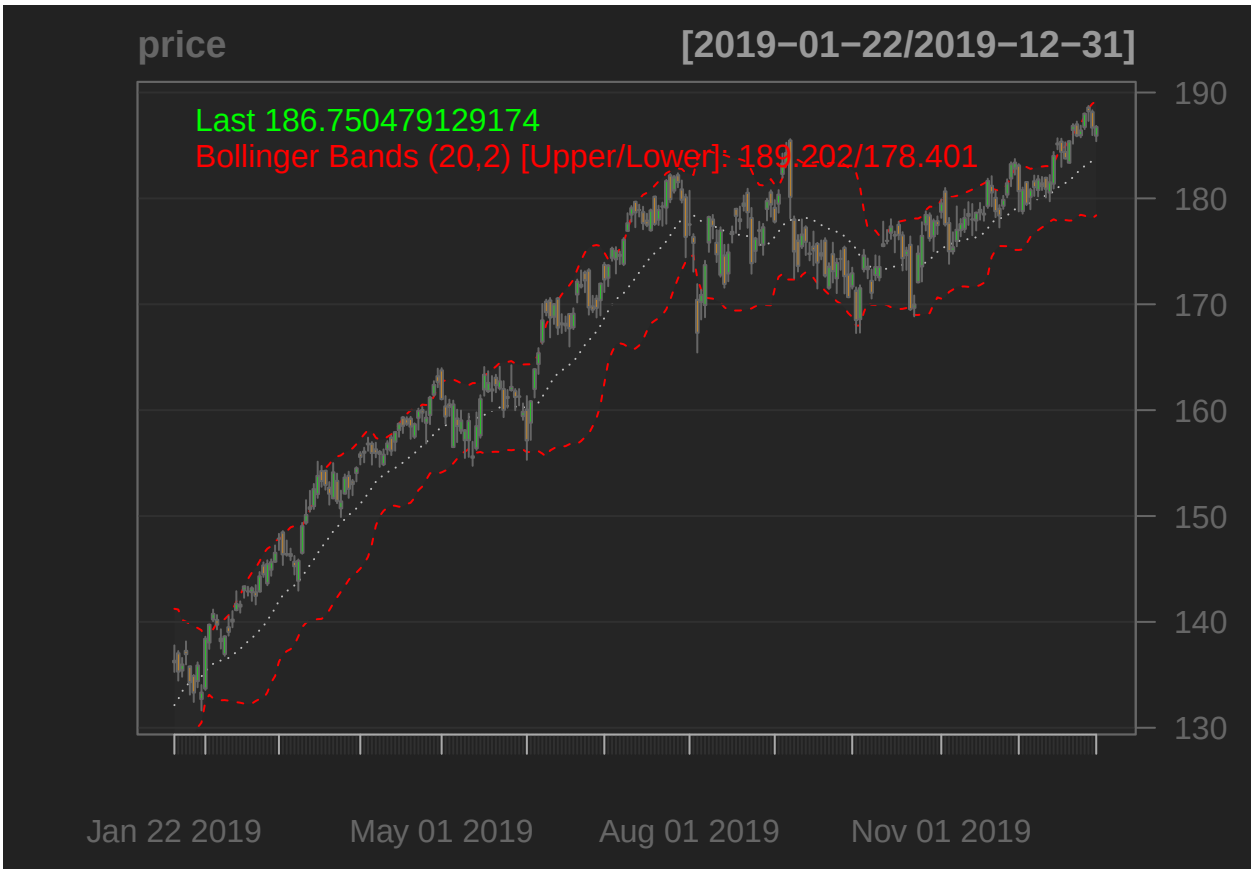



```
## NULL
```

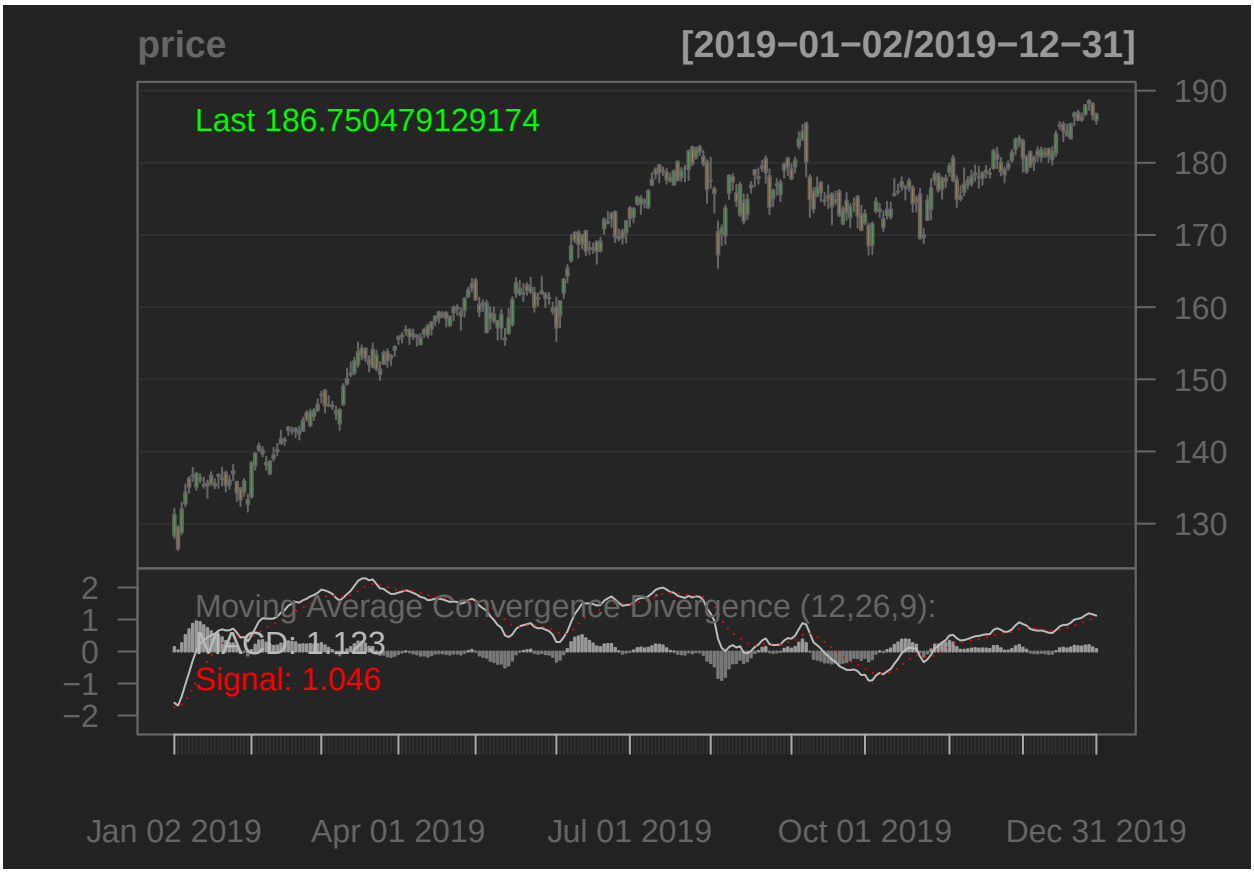
```
chart_Series(price, subset = "2019/")
```

As before, here we will take a look at the data, using some Technical Analysis Indicators, such as Bollinger Bands and Moving Average Convergence Divergence (MACD). The following plots represent these indicators applied to the data that we have loaded from Yahoo right now:

```
chartSeries(price, TA="addBBands(n=20)", subset="2019-01-21::2019-12-31")
```



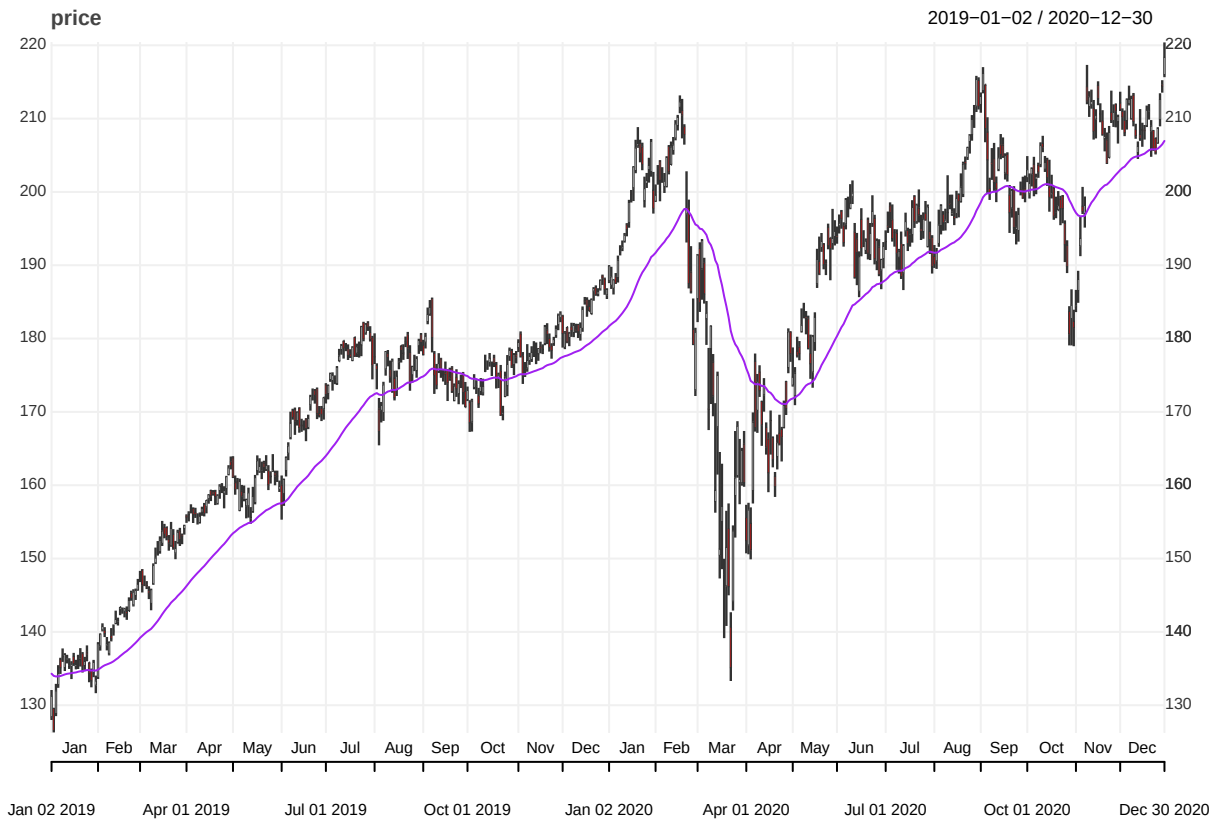

```
candleChart(price, TA=c(addMACD(),addVo()), subset = '2019') + neuralTheme
```



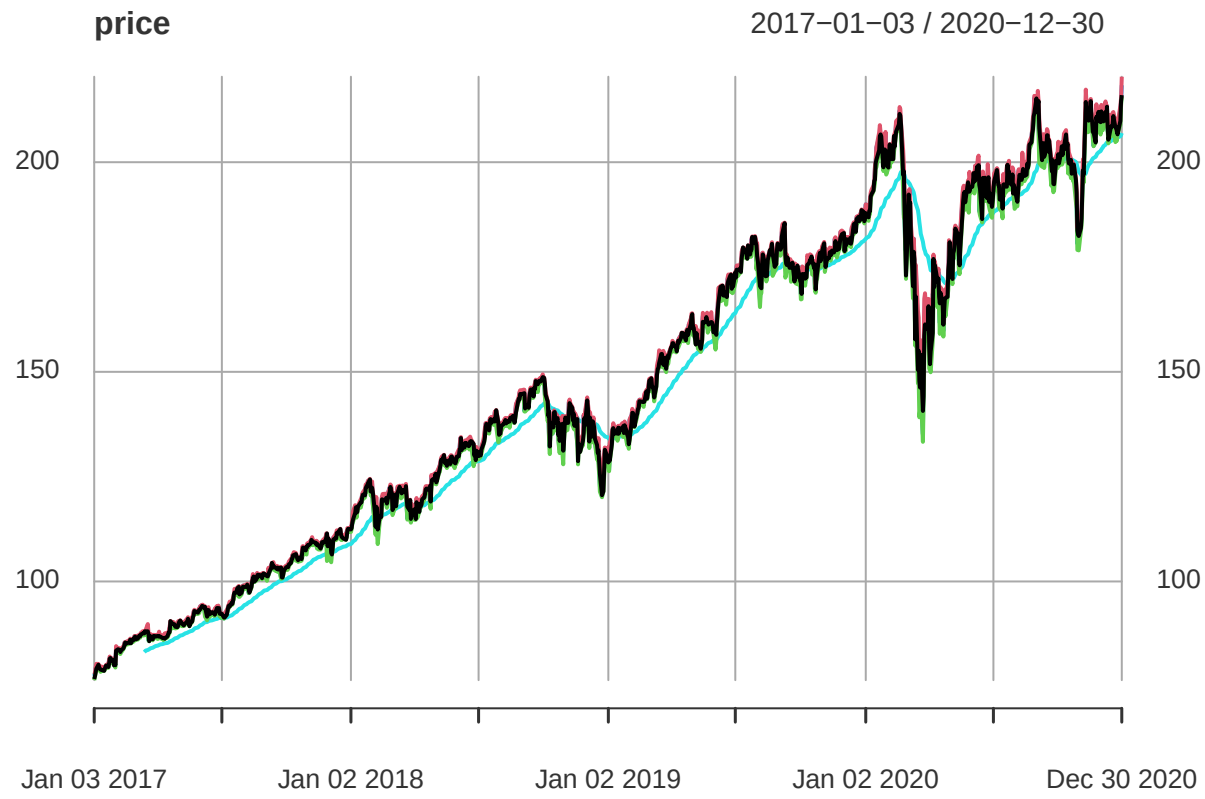

```
## NULL
```

Another Technical Analysis indicator, which we'll use on this project, is the Moving Average indicator, that looks as follows, when applied to the stocks from Visa:

```
price$EMA <- EMA(Cl(price), n = 50)
add_TA(price$EMA, on = 1, col = "purple")
```


price
2019−01−02 / 2020−12−30
220
210
200
190
180
170
160
150
140
130
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan 02 2019
Apr 01 2019
Jul 01 2019
Oct 01 2019
Jan 02 2020
Apr 01 2020
Jul 01 2020
Oct 01 2020
Dec 30 2020

```
plot(price)
```



2.2.3 Data Cleaning, Preparation and Partition

2.2.3.1 Creating the predictors to train and test the Models

In this section, the predictors for the Models will be calculated, using the stocks prices of Visa, downloaded from Yahoo:

```
library(tidyverse)

# This predictor will express the current candle type:
todayCandle<-data.frame(ifelse(price$Close > price$Open,"BULL","BEAR"))
# After creating this field, more predictors will be created using this column:

# This predictor will express the previous candle type: For this purpose,
# the "lag" function will be used:
yesterdayCandle<-data.frame(lag(todayCandle$Close,n=1))

# This predictor will express the next (future) candle type: the "lead"
# function will be used:
tomorrowCandle<-data.frame(lead(todayCandle$Close, n = 1))
```

A Reversal is when a trend change to the opposite direction, which can be positive (“bullish”, represented by green candles, called “Bull” candles) or negative (“bearish”, or red “Bear” candles). Bullish means the prices are going UP, while Bearish is prices going DOWN. When the trend is going to change direction, the appearance of the candle usually changes too, some kind of pattern is formed. Usually those patterns or

reversal candles, reveal that the trend is about to change direction, giving to investors a chance to enter the market opening positions, i.e. buying or selling stocks.

There are plenty of different candle patterns which are interpreted as reversal candles.

The algorithm used on this project will research the reversal candle known as "Doji": representing indecision in the market, this reversal candle means that the battle between sellers and buyers have not yet seem to commit in either direction.

There are several types of Doji candles, but for this research, we'll use only three of them:

1. the Standard Doji: is a single candlestick where the Open and Close prices are pretty much the same.
2. the Dragonfly Doji: considered a bullish (UP) reversal pattern, is a candle where the price closes about the same as the opening price, and have a long lower "shadow" (that means, the price went downwards, and later remonted), telling that the market is testing to find out where "support" or demand is, or in other words, the sellers were selling but found a strong opposition made by the stock buyers.
3. Gravestone Doji: the opposite of the Dragonfly Doji, and considered a bearish (DOWN) reversal pattern, is a candle where the price closes about the same as the opening price, with a long upper "shadow", meaning that the price went upwards, but the buyers found a strong opposition made by the stock sellers. Investors see this pattern as a time to sell, or exit the market.

As a first step, at the present stage of this development, we'll use only the Standard Doji candle. Further on this work, we'll use also other types of Doji candles.

```
# Researching Reversal Candles: Gravestone Doji and Dragonfly Doji candles:
# According to the prices of the stock, the size of the Doji Reversal candle will
# be fixed to 0.01 $ :
dojiSize <- 0.01


# Creating a predictor containing both Gravestone and Dragonfly Doji:
reversalCandle<-data.frame(ifelse(abs(price$Close - price$Open)<dojiSize,"YES","NO"))

# Checking the quantity of Reversal candles on the dataset:
countOfDoji<-length(reversalCandle[reversalCandle$Close == "YES",])
countOfDoji
```

```
## [1] 16
```

On this calculation, we choose as Reversal Candles, the ones that have a "shadow" or tail bigger than the average size of all candles on data, that means, that the difference between the High and Low prices, or the opposite, between Low and High, is greater than the average gap between Open-Close prices. Also, "reversalCandle" candles with no shadows are taken into account.

The algorithms used on the present project, will compute on Moving Averages. Considered trend following indicators, Moving averages show the trend of the market. They are useful in order to follow the prices trend.

Moving averages are the average prices for a certain past number of periods, days on this case. The software computes them to reduce price fluctuations or noise on the charts. This way, the moving averages filter relative small fluctuations in the prices, in order to watch the price trends without those fluctuations.

There are four different types of moving averages:Simple averages(SMAs), Exponential Moving Averages (EMAs), Smoothed moving averages (SMMAs), and Linear weighted moving averages(LWAs). Also, if the

moving average is calculated using shorter time periods, i.e. a fast-moving average, it will be quite responsive to recent price changes. However, if the moving average uses a longer time period to calculate the average, it will be a slow moving average, therefore less responsive, and its results will be affected by some kind of lag. That means, it will be less "noise", but the responsiveness will be notoriously slow.

A moving average with a longer period represents the long term trends. A shorter period moving average represents the short term trend. For example, a 200-period moving average shows the long term trend direction, a 50 period shows a medium trend, and a 5 period represents the shortest trend direction. In scalping trading, a five period moving average is usually used, while for intraday trading will be used a 14 periods moving average. Short term traders use a 21-period moving average, Medium-term will use a 50 day EMA, and long term traders use 200 periods EMAs.

The SMA and EMA are calculated differently. The computation for the EMA will make it quicker to react to price changes, while the SMA will react slower. That is because in case of an EMA, each point included in the average decreases in weight over time, such as the weight of a new data point on would drop to just 6.67% of its initial weight value after five new points had been added.

In this section, will be used Exponential Moving Averages (EMAs), for daily periods of 20 and 7 days respectively. After calculating the Moving Averages, we will compute both the relative position of the current candle to the two EMAs, and also the crossings between them:

```
ema7 <- EMA(price$Close, n = 7)
ema20 <- EMA(price$Close, n = 20)

# Using the above two EMAs, we can calculate whether the stocks price is
# actually above or below the moving averages:
relativePositionToEMA7<-data.frame(ifelse(price$Close > ema7,"ABOVE","BELOW"))
relativePositionToEMA20<-data.frame(ifelse(price$Close > ema20,"ABOVE","BELOW"))

# And also, we can calculate the crossings between the two moving averages:
movingAveragesCrossing <-data.frame(ifelse(ema7 > ema20,"ABOVE","BELOW"))


# The following predictor reflects the size of the candle, which will be
# measured according to the absolute difference between the opening-closing
# prices:
candleSize <-data.frame(abs(price$Close - price$Open))
head(candleSize)
```

```
##                Close
## 2017-01-03 0.7208420
## 2017-01-04 0.6916206
## 2017-01-05 0.7305851
## 2017-01-06 0.8864394
## 2017-01-09 0.5162792
## 2017-01-10 0.3799033
```

```
# The following field will be useful to calculate our stocks yield in case
# the prediction was correct:
revenue <- lead(candleSize$Close, n = 1)


## Creating a Data frame with the predictors as columns:
dsPredictors<-data.frame(todayCandle,
                         yesterdayCandle,
```

```
                  reversalCandle,
                  relativePositionToEMA7,
                  relativePositionToEMA20,
                  movingAveragesCrossing ,
                  candleSize,revenue,
                  tomorrowCandle)

names(dsPredictors)<-c("todayCandle",
                       "yesterdayCandle",
                  "reversalCandle",
                  "relativePositionToEMA7",
                  "relativePositionToEMA20",
                  "movingAveragesCrossing" ,
                  "candleSize",
                  "revenue",
                  "tomorrowCandle")

# Taking a glimpse at the dataset:
head(dsPredictors,20)
```

```
##            todayCandle yesterdayCandle reversalCandle relativePositionToEMA7
## 2017-01-03        BULL            <NA>             NO                    <NA>
## 2017-01-04        BULL            BULL             NO                    <NA>
## 2017-01-05        BULL            BULL             NO                    <NA>
## 2017-01-06        BULL            BULL             NO                    <NA>
## 2017-01-09        BEAR            BULL             NO                    <NA>
## 2017-01-10        BEAR            BEAR             NO                    <NA>
## 2017-01-11        BULL            BEAR             NO                   ABOVE
## 2017-01-12        BULL            BULL             NO                   ABOVE
## 2017-01-13        BULL            BULL             NO                   BELOW
## 2017-01-17        BULL            BULL             NO                   ABOVE
## 2017-01-18        BULL            BULL             NO                   ABOVE
## 2017-01-19        BEAR            BULL             NO                   ABOVE
## 2017-01-20        BEAR            BEAR             NO                   ABOVE
## 2017-01-23        BULL            BEAR             NO                   ABOVE
## 2017-01-24        BULL            BULL             NO                   ABOVE
## 2017-01-25        BULL            BULL             NO                   ABOVE
## 2017-01-26        BEAR            BULL             NO                   ABOVE
## 2017-01-27        BULL            BEAR             NO                   ABOVE
## 2017-01-30        BULL            BULL             NO                   ABOVE
## 2017-01-31        BULL            BULL             NO                   BELOW
##            relativePositionToEMA20 movingAveragesCrossing candleSize    revenue
## 2017-01-03                    <NA>                    <NA> 0.72084204 0.69162058
## 2017-01-04                    <NA>                    <NA> 0.69162058 0.73058512
## 2017-01-05                    <NA>                    <NA> 0.73058512 0.88643939
## 2017-01-06                    <NA>                    <NA> 0.88643939 0.51627918
## 2017-01-09                    <NA>                    <NA> 0.51627918 0.37990329
## 2017-01-10                    <NA>                    <NA> 0.37990329 0.53576535
## 2017-01-11                    <NA>                    <NA> 0.53576535 0.31171632
## 2017-01-12                    <NA>                    <NA> 0.31171632 0.14611800
## 2017-01-13                    <NA>                    <NA> 0.14611800 0.48705675
## 2017-01-17                    <NA>                    <NA> 0.48705675 0.31171535
## 2017-01-18                    <NA>                    <NA> 0.31171535 0.05844486
```

```
## 2017-01-19                   <NA>                   <NA> 0.05844486 0.24352837
## 2017-01-20                   <NA>                   <NA> 0.24352837 0.43835594
## 2017-01-23                   <NA>                   <NA> 0.43835594 0.80851518
## 2017-01-24                   <NA>                   <NA> 0.80851518 0.10715346
## 2017-01-25                   <NA>                   <NA> 0.10715346 0.66239718
## 2017-01-26                   <NA>                   <NA> 0.66239718 0.43834815
## 2017-01-27                   <NA>                   <NA> 0.43834815 0.22404221
## 2017-01-30                   <NA>                   <NA> 0.22404221 0.56498778
## 2017-01-31                  ABOVE                  ABOVE 0.56498778 0.44809221
##            tomorrowCandle
## 2017-01-03           BULL
## 2017-01-04           BULL
## 2017-01-05           BULL
## 2017-01-06           BEAR
## 2017-01-09           BEAR
## 2017-01-10           BULL
## 2017-01-11           BULL
## 2017-01-12           BULL
## 2017-01-13           BULL
## 2017-01-17           BULL
## 2017-01-18           BEAR
## 2017-01-19           BEAR
## 2017-01-20           BULL
## 2017-01-23           BULL
## 2017-01-24           BULL
## 2017-01-25           BEAR
## 2017-01-26           BULL
## 2017-01-27           BULL
## 2017-01-30           BULL
## 2017-01-31           BEAR
```

```
head(yesterdayCandle,20)
```

```
##    lag.todayCandle.Close..n...1.
## 1                            <NA>
## 2                            BULL
## 3                            BULL
## 4                            BULL
## 5                            BULL
## 6                            BEAR
## 7                            BEAR
## 8                            BULL
## 9                            BULL
## 10                           BULL
## 11                           BULL
## 12                           BULL
## 13                           BEAR
## 14                           BEAR
## 15                           BULL
## 16                           BULL
## 17                           BULL
## 18                           BEAR
## 19                           BULL
## 20                           BULL
```

```
head(todayCandle,20)
```

```
##            Close
## 2017-01-03  BULL
## 2017-01-04  BULL
## 2017-01-05  BULL
## 2017-01-06  BULL
## 2017-01-09  BEAR
## 2017-01-10  BEAR
## 2017-01-11  BULL
## 2017-01-12  BULL
## 2017-01-13  BULL
## 2017-01-17  BULL
## 2017-01-18  BULL
## 2017-01-19  BEAR
## 2017-01-20  BEAR
## 2017-01-23  BULL
## 2017-01-24  BULL
## 2017-01-25  BULL
## 2017-01-26  BEAR
## 2017-01-27  BULL
## 2017-01-30  BULL
## 2017-01-31  BULL
```

```
head(tomorrowCandle,20)
```

```
##    lead.todayCandle.Close..n...1.
## 1                            BULL
## 2                            BULL
## 3                            BULL
## 4                            BEAR
## 5                            BEAR
## 6                            BULL
## 7                            BULL
## 8                            BULL
## 9                            BULL
## 10                           BULL
## 11                           BEAR
## 12                           BEAR
## 13                           BULL
## 14                           BULL
## 15                           BULL
## 16                           BEAR
## 17                           BULL
## 18                           BULL
## 19                           BULL
## 20                           BEAR
```

2.2.3.2 Data Cleaning

The calculations performed previously, have produced NAs on the data, and in this section we'll clean it:

```r
# Checking whether are there any NAs ? :
apply(price,2, function(x)sum(is.na(x)))
```

```
##  Open  High   Low Close   EMA
##     0     0     0     0    49
```

```r
# The original dataset loaded had no NAs, and the only field with NAs is the
# one that was created on code, "EMA", with the purpose of calculating the EMAs.
# This problem will be reflected on the dataframe with the predictors that
# was created right now:
apply(dsPredictors,2, function(x)sum(is.na(x)))
```

```
##             todayCandle         yesterdayCandle          reversalCandle
##                       0                       1                       0
##  relativePositionToEMA7 relativePositionToEMA20  movingAveragesCrossing
##                       6                      19                      19
##              candleSize                 revenue          tomorrowCandle
##                       0                       1                       1
```

```r
# Indeed, six columns with NAs can be detected.


# Eliminating the NAs caused by the EMAs computations:
dsPredictors<-slice(dsPredictors,21:length(dsPredictors$reversalCandle))
length(dsPredictors)
```

```
## [1] 9
```

```r
apply(dsPredictors,2, function(x)sum(is.na(x)))
```

```
##             todayCandle         yesterdayCandle          reversalCandle
##                       0                       0                       0
##  relativePositionToEMA7 relativePositionToEMA20  movingAveragesCrossing
##                       0                       0                       0
##              candleSize                 revenue          tomorrowCandle
##                       0                       1                       1
```

Now the NAs had been cleaned from the dataframe. The only fields with a NA (only one NA on the field), are the two columns which refers to the future, "revenue" and "tomorrowCandle". That means, the "lead" function writes NA because the final day does not exist.

2.2.3.3 Data Partition

The dataset will be partitioned in Train set (66% of the data) and a test validation dataset (33% of the price candles). This partition must contain the required continuity between the yesterday-today-tomorrow candles, because this is what the training algorithm will try to research and reveal: the patterns of continuity between consecutive candles. Therefore, we cannot just take prices hazardously and insert them in both training set and test set.

The data will be partitioned in a Train set, used to the Machine Learning algorithms training, and a Test dataset, used to validate the predictions made by using the Models. In this case, we will make predictions using a proportion of 2 to 1 between training data to test data:

```
trainRange<-1:200
testRange<-201:300

# After defining the ranges, the two datasets, Train and Test, are created:
train<-dsPredictors[trainRange,]
test<-dsPredictors[testRange,]
```

2.3 Modeling Section

In this section we develop 3 Models in order to predict Stocks Market prices for the Visa data obtained from Yahoo:

1. First the Formula to be used on the Models will be defined.
2. Second, creation of the Support Vector Machines (SVM) Model
3. Third, creation of the Naive Bayes Model
4. Fourth, creation of the Random Forests Model

2.3.1 Defining the Formulas

First, we define the formula to be used on the Models calculation. Will we use all the columns as predictors? Or we'll use a narrow number of the fields to the computations? The predictions to be made are those concerning to the stocks prices direction tomorrow, i.e., the "tomorrowCandle". Therefore, taking apart this column, we'll use all other fields in order to perform the predictions.

```
# The predictions target is the field "tomorrowCandle":
goalToPredict<-"tomorrowCandle"

# Therefore, all other fields are taken as predictors:
allPredictors<-c("todayCandle","yesterdayCandle",
                   "reversalCandle","relativePositionToEMA7",
                 "relativePositionToEMA20","movingAveragesCrossing" ,
                   "candleSize") # ,"revenue"



# Creating the predictions formula:
allPredictors<-paste(allPredictors, collapse = "+" )
predictionsFormula<-as.formula(paste(goalToPredict,"~",allPredictors, sep=""))
```

When calculating the accuracy of the models, we will both compute the errors found and also we'll want to calculate the profit rendered by the models. That's why we'll make a function to compute the yields, using the following logic: if the prediction was accurate, we'll earn the difference between the opening and the closing prices of the stocks tomorrow, i.e. the "revenue" field. This logic is expressed by the following function:

```
stocksYield <- function(dataframe,predictionMade){
  # predictionMade represents the prediction made by the machine learning model:
  dataframe$predictionMade <- predictionMade
  dataframe$prediReturn <- ifelse(
```

```
    dataframe$tomorrowCandle != dataframe$predictionMade,
    -dataframe$revenue,dataframe$revenue)
  dataframe$cumReturn <- cumsum(dataframe$prediReturn )
  return (dataframe)
}
```

2.3.2 Support Vector Machines (SVM) Model

In this section, a Support Vector Machines (SVM) Model is developed. Support Vector Machines are one of the most robust prediction Supervised Machine Learning models, which perform data analysis for classification and regression problems. They were developed at the AT&T Bell Laboratories, and are based on statistical learning frameworks or VC theory proposed by Vapnik (1982, 1995), Chervonenkis (1974), and Corinna Cortes [1].

The Support Vector Machine algorithm translates each data item as a point in an n-dimensional space, where n is the number of predictors, with the value of each feature being the value of a particular coordinate. Based on that, it performs classification by finding the hyper-plane that differentiates the two classes the best.

The objective of the Support Vector Machines is to find a hyperplane in an N-predictors space, that distinctly classifies the data points.

In other words, the "support vectors" are simply the coordinates of the data, and the Support Vector Machine algorithm researches the frontier that best segregates the two classes, called hyper-plane-line.

The maximum-margin hyperplane algorithm proposed by Vapnik in 1963, used a linear classifier. This was improved in 1992, when Bernhard Boser, Isabelle Guyon and Vladimir Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes [2].

On the algorithms of the present project, two different kernels will be used: the Polynomial kernel and the Sigmoid Kernel.

References

1. Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks" . Machine Learning. 20 (3): 273–297

2. Boser, Bernhard E.; Guyon, Isabelle M.; Vapnik, Vladimir N. (1992). "A training algorithm for optimal margin classifiers". Proceedings of the fifth annual workshop on Computational learning theory – COLT '92. p. 144

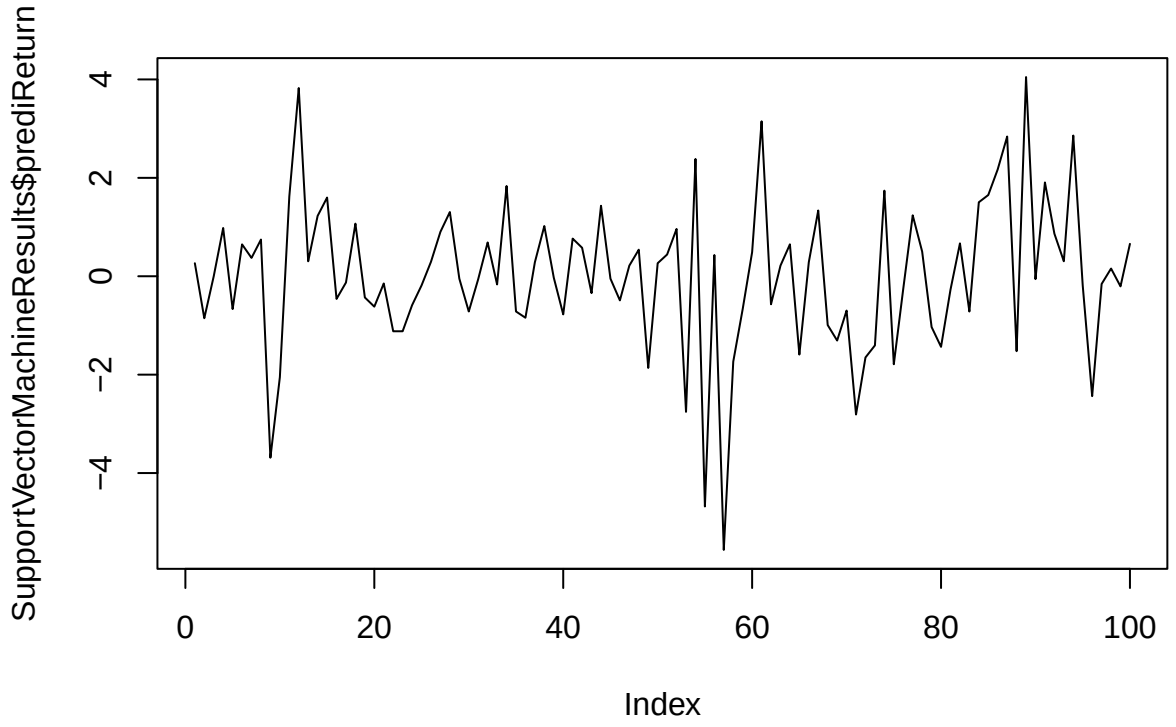
2.3.2.1 Using the Polynomial Kernel

As stated before, on the present project, two different kernels will be used: the Polynomial kernel and the Sigmoid Kernel.

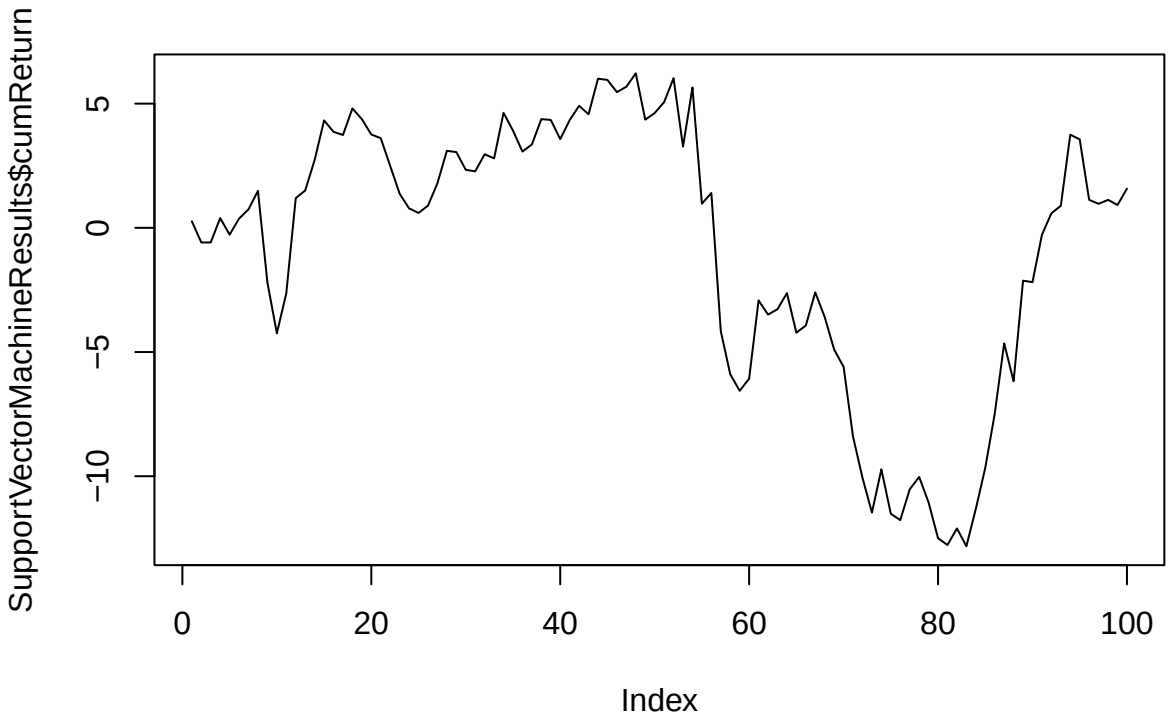
First, we will make predictions using the Polynomial Model, then we will perform the predictions by using the Sigmoid Model.

```
# Loading the Support Vector Machine software library:
library(e1071)


set.seed(42)
# Creating the Support Vector Machine model using Polynomial Kernel:
SupportVectorMachineModel <- svm(factor(tomorrowCandle) ~ todayCandle +
    yesterdayCandle + reversalCandle + relativePositionToEMA7 +
```

```
      relativePositionToEMA20 +
    movingAveragesCrossing + candleSize, data = train , kernel = "polynomial")


# Making the predictions by using the Support Vector Machine Model:
SupportVectorMachinePredictions <- predict(SupportVectorMachineModel, test)


# Calculating the profit results:
SupportVectorMachineResults <- stocksYield(test, SupportVectorMachinePredictions)


plot(SupportVectorMachineResults$prediReturn, type = "l")
```




```
plot(SupportVectorMachineResults$cumReturn, type = "l")
```




```
# The plots express respectively  predicted revenue and cumulative revenue, and
# can be seen that the results are not conveniently good.

# Computing the confusion matrix:
confusionmatrix.svm <- table(SupportVectorMachineResults$tomorrowCandle,
                             SupportVectorMachineResults$predictionMade)
print(confusionmatrix.svm)
```

```
##
##        BEAR BULL
##   BEAR   14   32
##   BULL   19   35
```

```
# Calculating accuracy of the Support Vector Machine Model:
SupportVectorMachineErrors <- mean(SupportVectorMachineResults$tomorrowCandle !=
                    SupportVectorMachineResults$predictionMade)
print(paste("Accuracy", 1 - SupportVectorMachineErrors))
```

```
## [1] "Accuracy 0.49"
```

```
results <- data.frame(Model="Stocks Trading System expectations",
                      Accuracy =  0.5100)

1 - SupportVectorMachineErrors -> totalAccuracy
```

```
results <- results %>%
  add_row(Model="   First Model : Support Vector Machine - Polynomial kernel",
                               Accuracy = 1 - SupportVectorMachineErrors
                               )
results
```

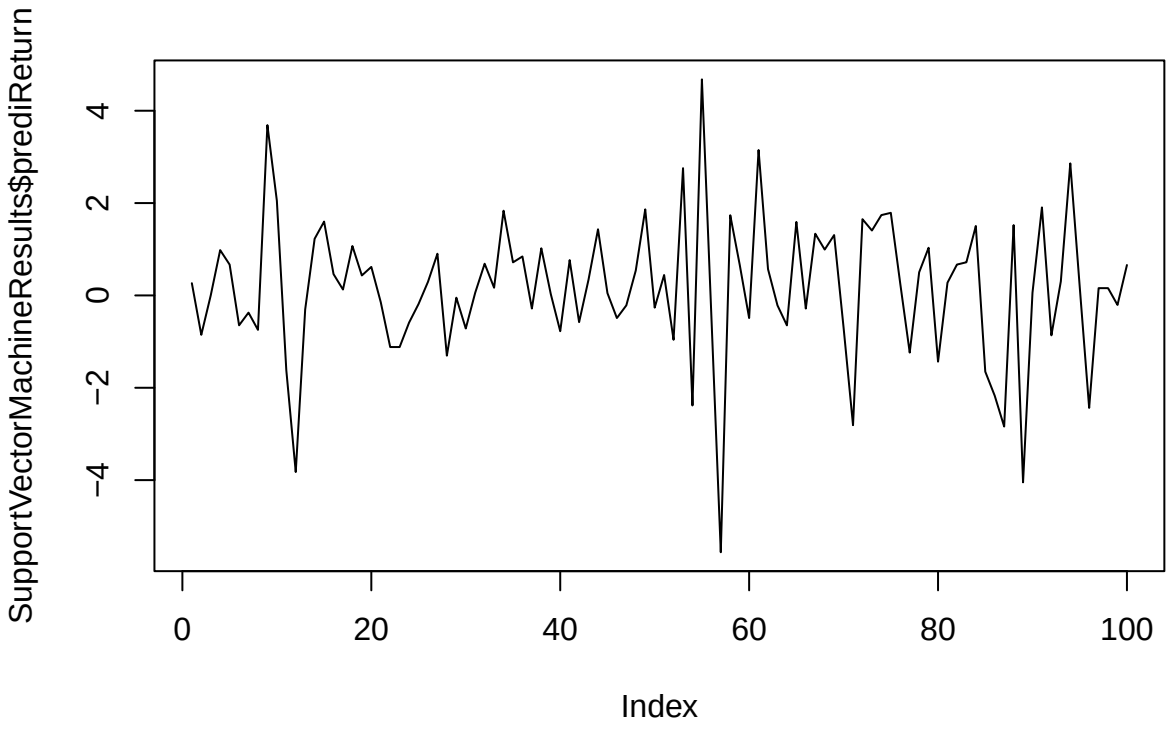
```
##                                                          Model Accuracy
## 1                          Stocks Trading System expectations     0.51
## 2    First Model : Support Vector Machine - Polynomial kernel     0.49
```

The present model, Support Vector Machine based on the Polynomial kernel, does not achieve the goals of the project, which were of at least an accuracy of fifty one percent.

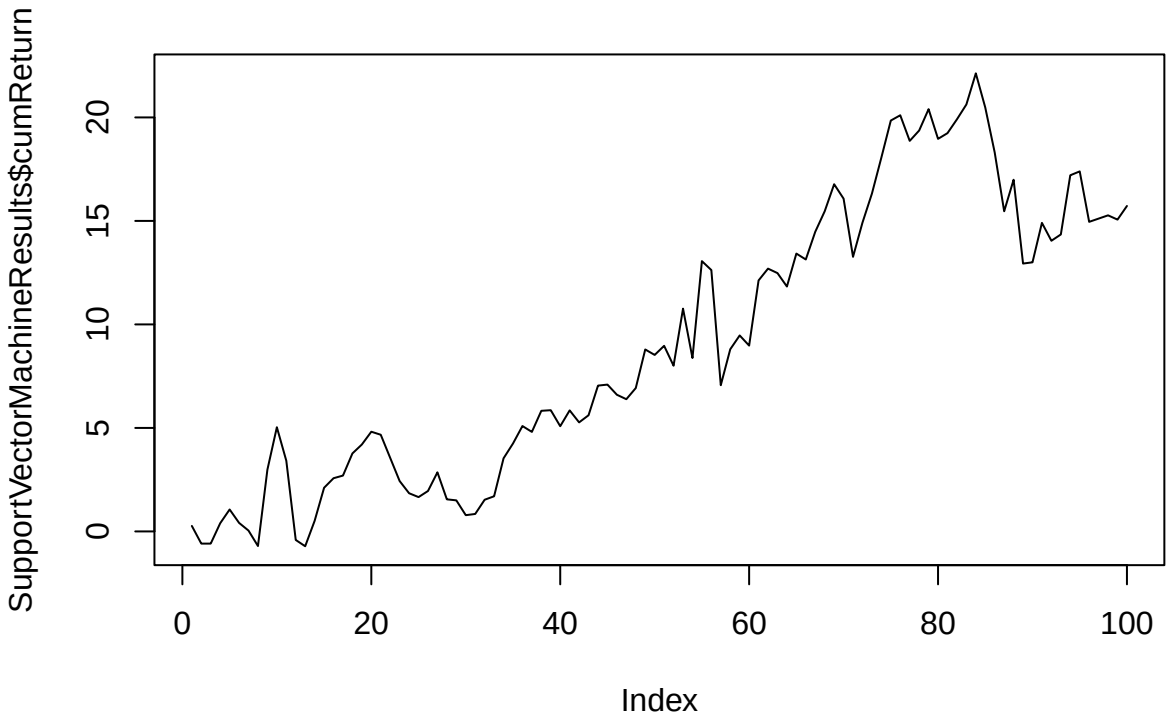
2.3.2.2 Using the Sigmoid Kernel

Now perform the predictions using the Sigmoid Kernel Model:

```
set.seed(42)
# Creating the Support Vector Machine model using Sigmoid Kernel:
SupportVectorMachineModel <- svm(factor(tomorrowCandle) ~ todayCandle +
    yesterdayCandle + reversalCandle + relativePositionToEMA7 + relativePositionToEMA20 +
    movingAveragesCrossing + candleSize, data = train , kernel = "sigmoid")



# Making the predictions by using the SVM Model:
SupportVectorMachinePredictions <- predict(SupportVectorMachineModel, test)


# Calculating the profit results:
SupportVectorMachineResults <- stocksYield(test, SupportVectorMachinePredictions)


# Showing the revenues produced:
plot(SupportVectorMachineResults$prediReturn, type = "l")
```




```
# Cumulative revenue:
plot(SupportVectorMachineResults$cumReturn, type = "l")
```




```
# The plots above express respectively  predicted revenue and cumulative revenue,
# and can be seen an improvement of the results.

# Computing and showing the confusion matrix:
confusionmatrix.svm <- table(SupportVectorMachineResults$tomorrowCandle,
                             SupportVectorMachineResults$predictionMade)
print(confusionmatrix.svm)
```

```
##
##        BEAR BULL
##   BEAR   18   28
##   BULL   13   41
```

```
# Calculating accuracy of the Support Vector Machine Model:
SupportVectorMachineErrors <- mean(SupportVectorMachineResults$tomorrowCandle !=
                                     SupportVectorMachineResults$predictionMade)
print(paste("Accuracy", 1 - SupportVectorMachineErrors))
```

```
## [1] "Accuracy 0.59"
```

```
(1 - SupportVectorMachineErrors) + totalAccuracy -> totalAccuracy

# Adding the accuracy to the Results table:
results <- results %>%
```

```
  add_row(Model="   Second Model : Support Vector Machine - Sigmoid kernel",
                                Accuracy = 1 - SupportVectorMachineErrors
                                )
results
```

```
##                                                        Model Accuracy
## 1                           Stocks Trading System expectations     0.51
## 2    First Model : Support Vector Machine - Polynomial kernel     0.49
## 3      Second Model : Support Vector Machine - Sigmoid kernel     0.59
```

The table expresses an improvement on accuracy, obtained from applying the Sigmoid kernel to the Support Vector Machine.

2.3.3 Naive Bayes Model

The Naive Bayes algorithm is a supervised Machine Learning algorithm, based on the Bayes Theorem [1] and used for solving classification problems. It predicts on the basis of the probability of an event, therefore is considered a probabilistic classifier.

The Naive Bayes algorithm is called Naive because it makes the assumption that the occurrence of a determined feature is independent of the existence of other features.

For example, if classifying fruits, each one is identified on the bases of taste, color, shape, etc. Therefore, a red, quasi-spherical, and sweet fruit, will be recognized as an apple. Hence each feature contributes individually to identify what kind of fruit this is, without depending on each other.

The Naive Bayes algorithm belongs to a family of simple "probabilistic classifiers" based on applying Bayes Theorem, along with strong independence assumptions between the predictors[2], and, when coupled with kernel density estimation, they can achieve higher accuracy levels[3].

For this project, being a problem of classification, since we try to predict whether the prices will be incremented or will fall tomorrow, the Naive Bayes algorithm can be very useful to make the predictions.

References

1. Irizarry, Rafael. 2021. "Introduction to Data Science - Data Analysis and Prediction Algorithms with R". Naive Bayes - 31.7.1
2. McCallum, Andrew. 2019. "Graphical Models, Lecture2: Bayesian Network Representation" .
3. Hastie, Trevor. 2001. "The elements of statistical learning : data mining, inference, and prediction : with 200 full-color illustrations".


```
# Loading the Naive Bayes software library:
library("naivebayes")
```

```
## naivebayes 0.9.7 loaded
```

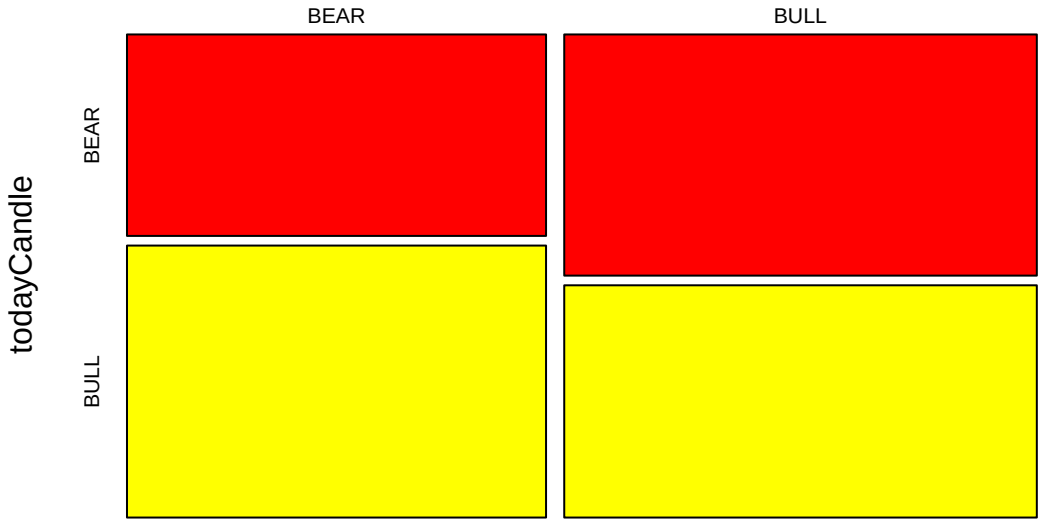
```
##
## Attaching package: 'naivebayes'
```

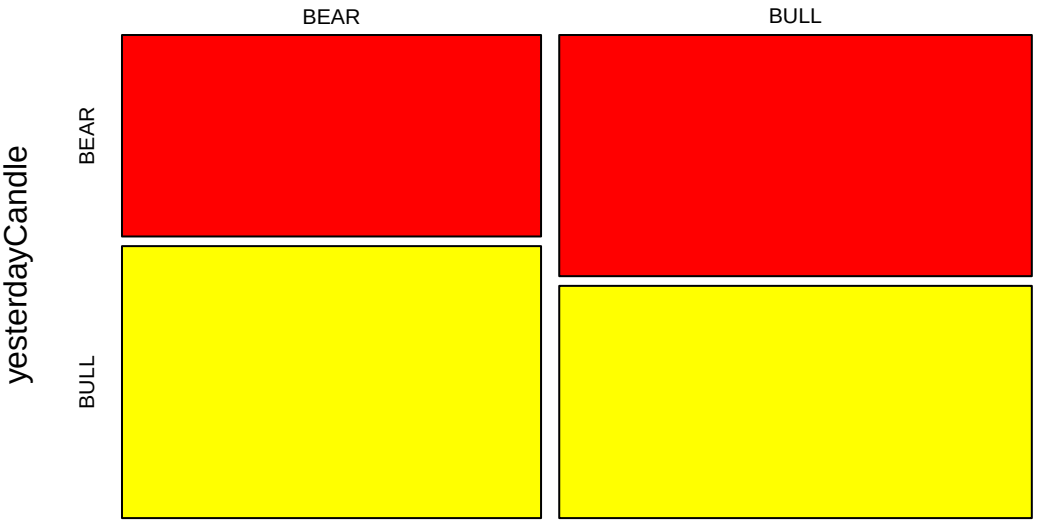
```
## The following object is masked from 'package:data.table':
##
##     tables
```

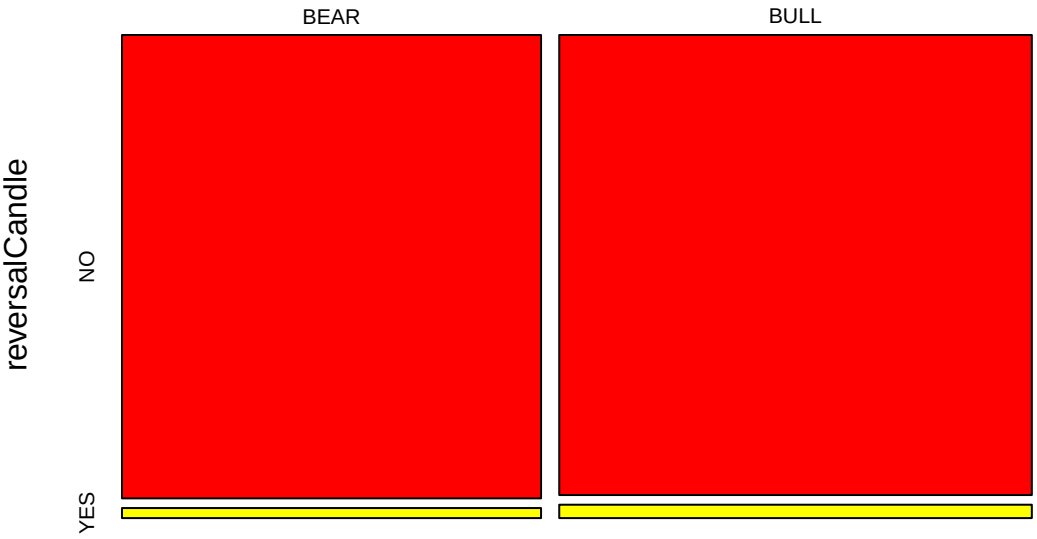
```
# Creating the Naive Bayes Model:
set.seed(42)
NaiveBayesModel <- naive_bayes(predictionsFormula, data = train)
```

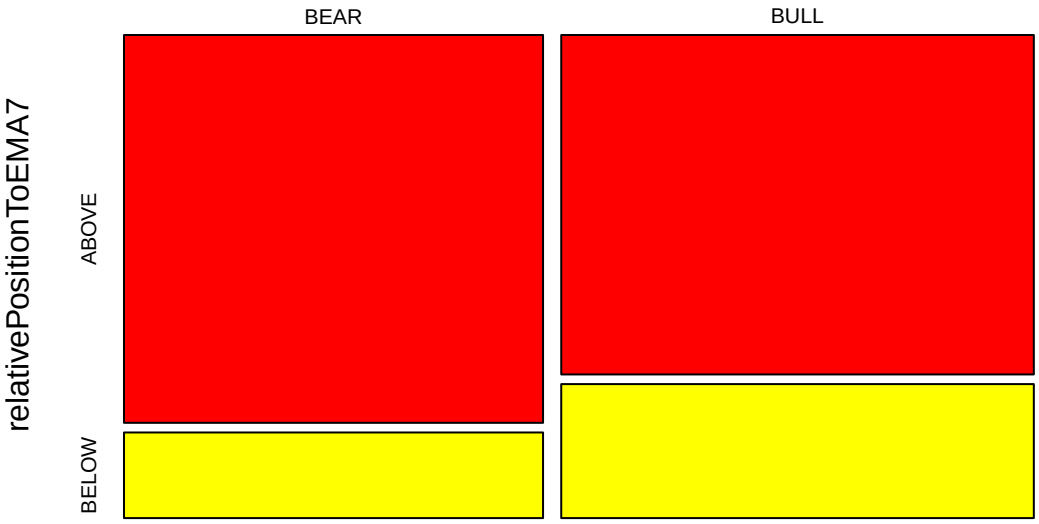
The following graphs show the predictive power of the different predictors, according to the Naive Bayes Model:

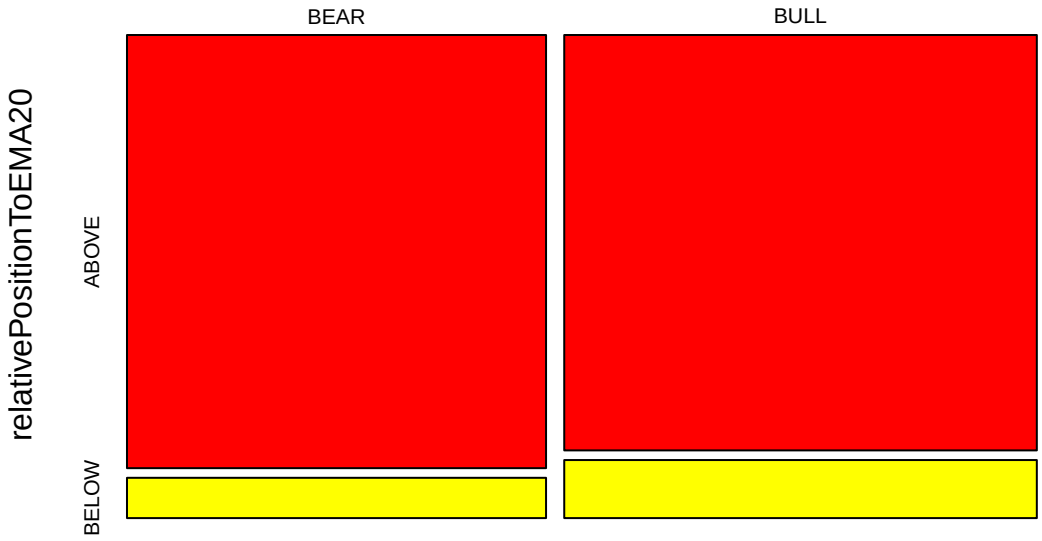
```
plot(NaiveBayesModel)
```

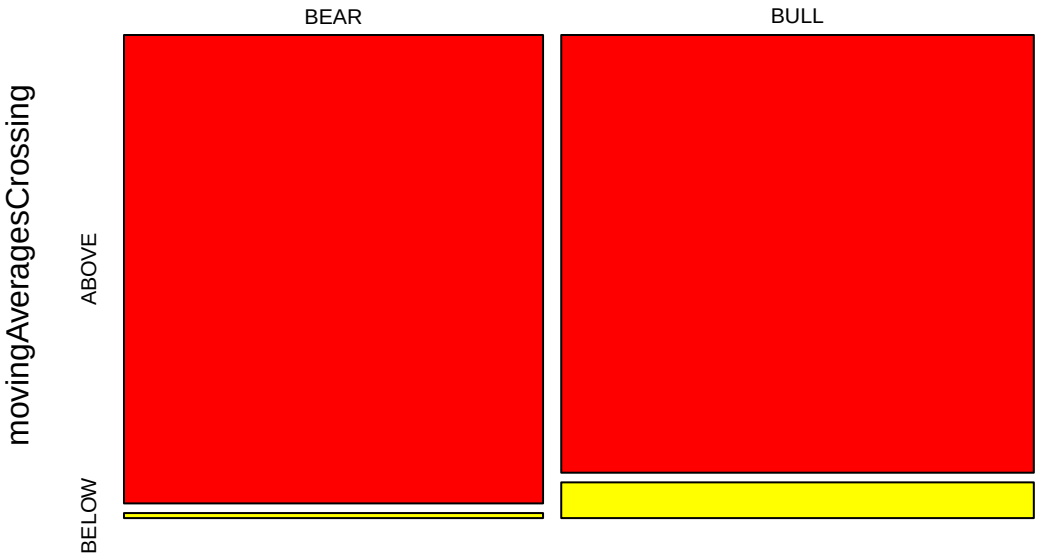


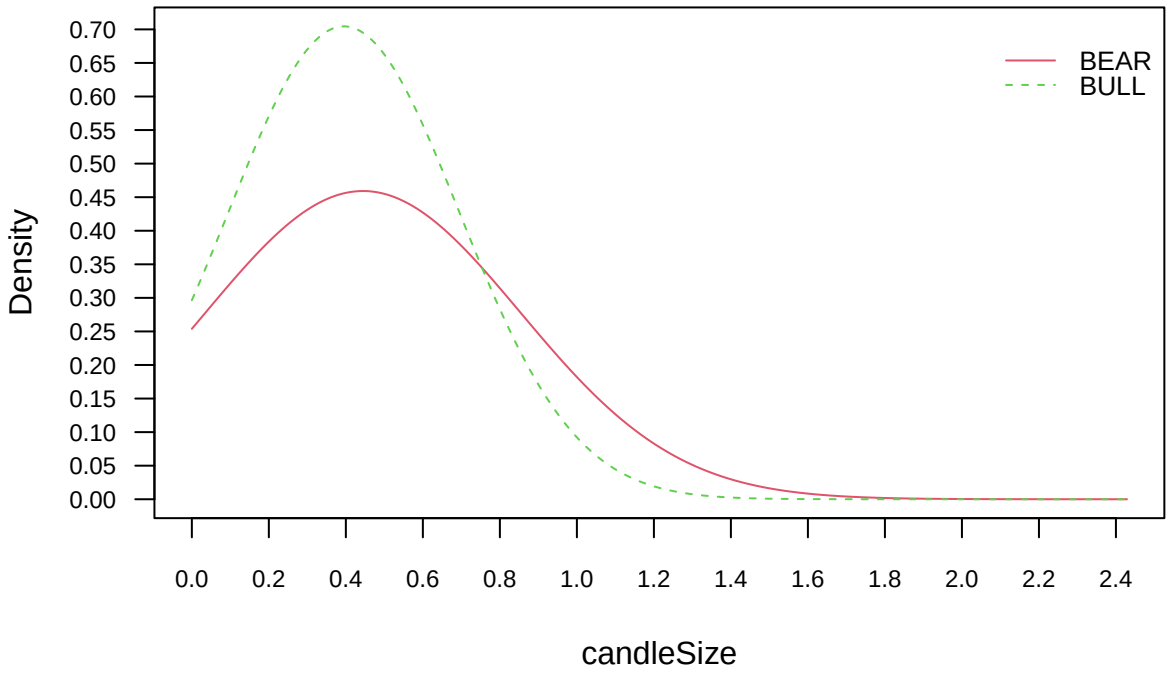

BEAR
BULL
yesterdayCandle
BEAR
BULL


BEAR
BULL
reversalCandle
NO
YES


BEAR
BULL
relativePositionToEMA7
ABOVE
BELOW


BEAR
BULL
relativePositionToEMA20
ABOVE
BELOW


movingAveragesCrossing
ABOVE
BELOW
BEAR
BULL

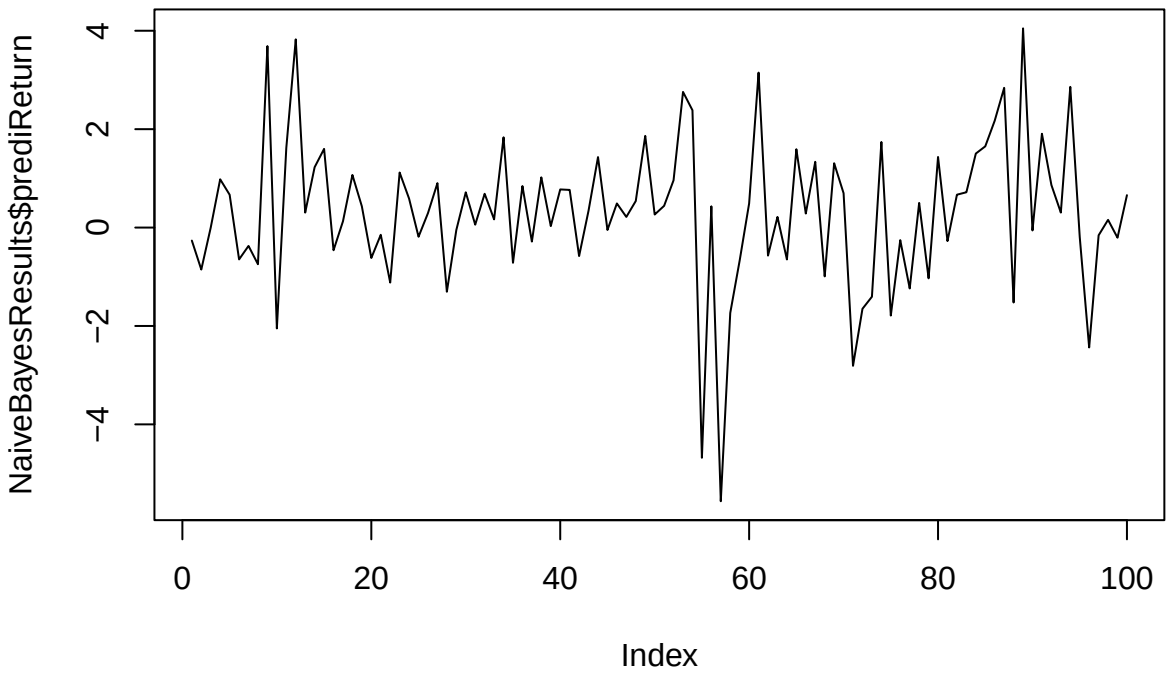


From the graphs we can extrapolate some useful indications. It seems that, if Today's candle is Bull, there are less probabilities that tomorrow's candle will be Bull again, that means, that the price of the stock may stop increasing. And on the contrary, if today's candle is Bear, tomorrow there will be more chances that continue to be Bear. The same occurs with Yesterday's candle. A more interesting thing happens with the predictor "RelativePositionToEMA7". It seems that, if today the price closes above the Moving Average, there are good chances that tomorrow prices will go down. Also, if the 7 days Moving Average is above the 20 days long EMA, there are bigger chances that tomorrow's candle will be Bear, i.e. prices would go down.

```
# Making the predictions to guess the Next Day candle direction:
NaiveBayesPredictions <- predict(NaiveBayesModel, test)
```

```
## Warning: predict.naive_bayes(): more features in the newdata are provided as
## there are probability tables in the object. Calculation is performed based on
## features to be found in the tables.
```

```
NaiveBayesResults <- stocksYield(test, NaiveBayesPredictions)

# Showing the Prediction results:
plot(NaiveBayesResults$prediReturn, type = "l")
```




```
# Cumulative revenue:
plot(NaiveBayesResults$cumReturn, type = "l")
```


```
# The plots above show again an improvement of the results for
# respectively predicted revenue and cumulative revenue.


# Computing and showing the confusion matrix:
confusionmatrix.nb <- table(NaiveBayesResults$tomorrowCandle,
                            NaiveBayesResults$predictionMade)
print(confusionmatrix.nb)
```

```
##
##        BEAR BULL
##   BEAR   35   11
##   BULL   28   26
```

```
# Calculating accuracy of the Naive Bayes Model:
NaiveBayesErrors <- mean(NaiveBayesResults$tomorrowCandle !=
                   NaiveBayesResults$predictionMade)
print(paste("Accuracy", 1 - NaiveBayesErrors))
```

```
## [1] "Accuracy 0.61"
```

```
(1 - NaiveBayesErrors) + totalAccuracy -> totalAccuracy

# Adding the accuracy to the Results table:
```

```
results <- results %>%
  add_row(Model="   Third Model : Naive Bayes Model",
                                Accuracy = 1 - NaiveBayesErrors
                                )
results
```

```
##                                                      Model Accuracy
## 1                          Stocks Trading System expectations     0.51
## 2    First Model : Support Vector Machine - Polynomial kernel     0.49
## 3      Second Model : Support Vector Machine - Sigmoid kernel     0.59
## 4                              Third Model : Naive Bayes Model     0.61
```

We can see an improvement of the accuracy obtained.

2.3.4 Random Forest Model

Random Forests are Supervised Machine Learning algorithms used mainly for classification problems, although they can also solve regression ones. In Machine Learning, the Random Forest algorithm and method is also known as the Random Forest Classifier, and is formed by a random collection of Decision Trees. Basically, the Random Forest algorithm creates a random sample of multiple decision trees and merges them together in order to obtain more accurate predictions. Usually, the more trees are in the Random Forest, the more accurate will be the prediction.

The Random Forests are aggregations of several Decision Trees, a Machine Learning method used to solve both regression and classification problems. They operate by assembling a multitude of Decision Trees at training time.

Random Forests have features that help to improve prediction performance and reduce instability, like bootstrap aggregation or bagging, which by generating many predictors, form a final prediction based on the average prediction of all the different trees used by the method. Also, to be sure that those trees are not the same one, bootstrap is used to induce randomness. That is why they are called Random Forests: the bootstrap makes the individual trees randomly different, and the combination of the different trees becomes the forest [1].

For classification problems, the output of the Random Forest is the one selected by most of the Decision Trees. However, for regression problems, the average prediction of the individual Decision Trees is returned [2].

Another advantage of Random Forests is, they succeeds to correct the Decision Trees tendency to overfit their training set[3].

An substantial improvement can be attained by making the estimate smoother, changing the parameter that controls the minimum number of data points in the nodes of the Decision Trees, so the larger this minimum, the smoother the final estimate will be [4].

On the algorithm used in the present project, the number of data points in the nodes of the tree, will be 500.

References

1. Irizarry, Rafael. 2021. “Introduction to Data Science - Data Analysis and Prediction Algorithms with R”. Random forests - 31.11
2. Ho TK. 1998. “The Random Subspace Method for Constructing Decision Forests” IEEE Transactions on Pattern Analysis and Machine Intelligence. 20
3. Irizarry, Rafael. 2021. “Introduction to Data Science - Data Analysis and Prediction Algorithms with R”. Random forests - 31.12

4. Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome. 2008. “The Elements of Statistical Learning”

```
# Loading the Random Forests software library:
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.1.1
```

```
## randomForest 4.6-14
```

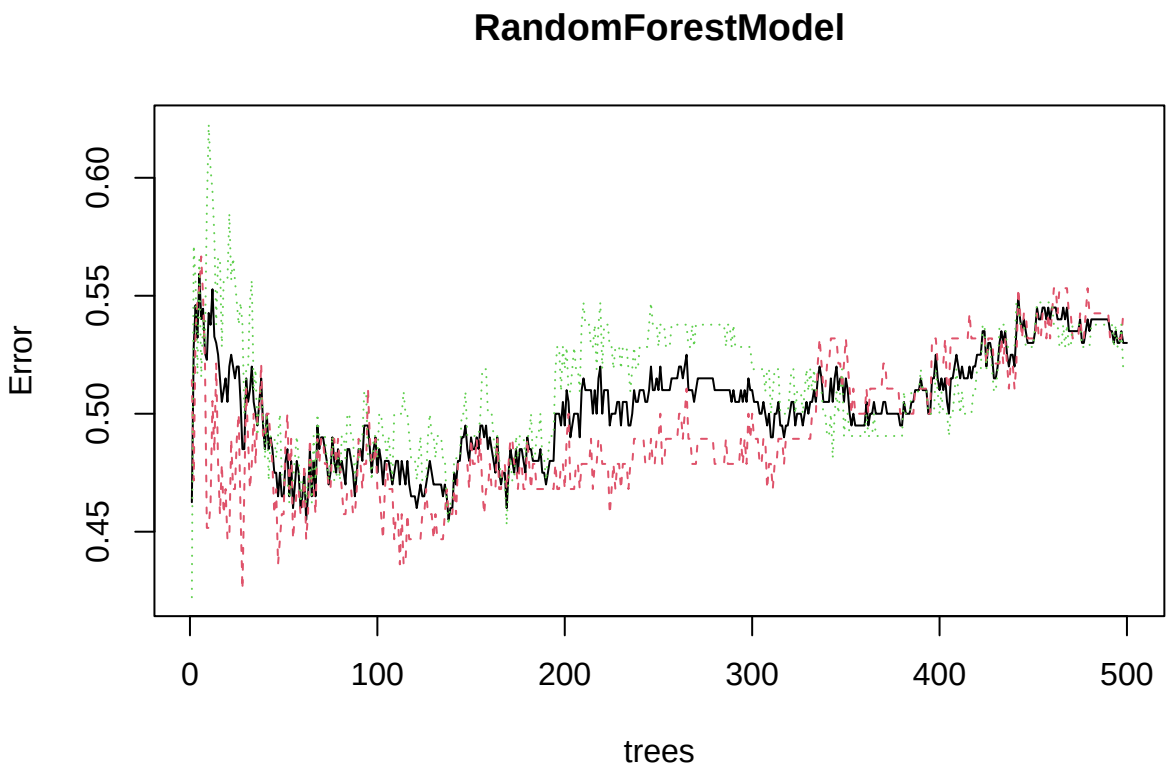
```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

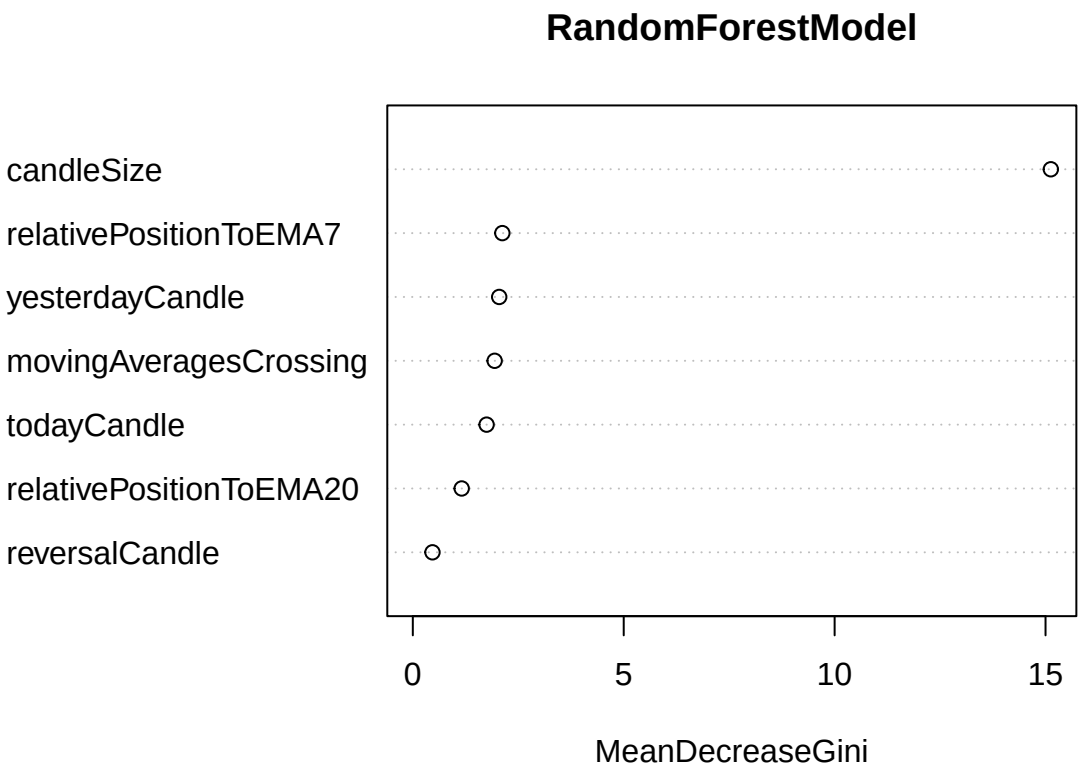
```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

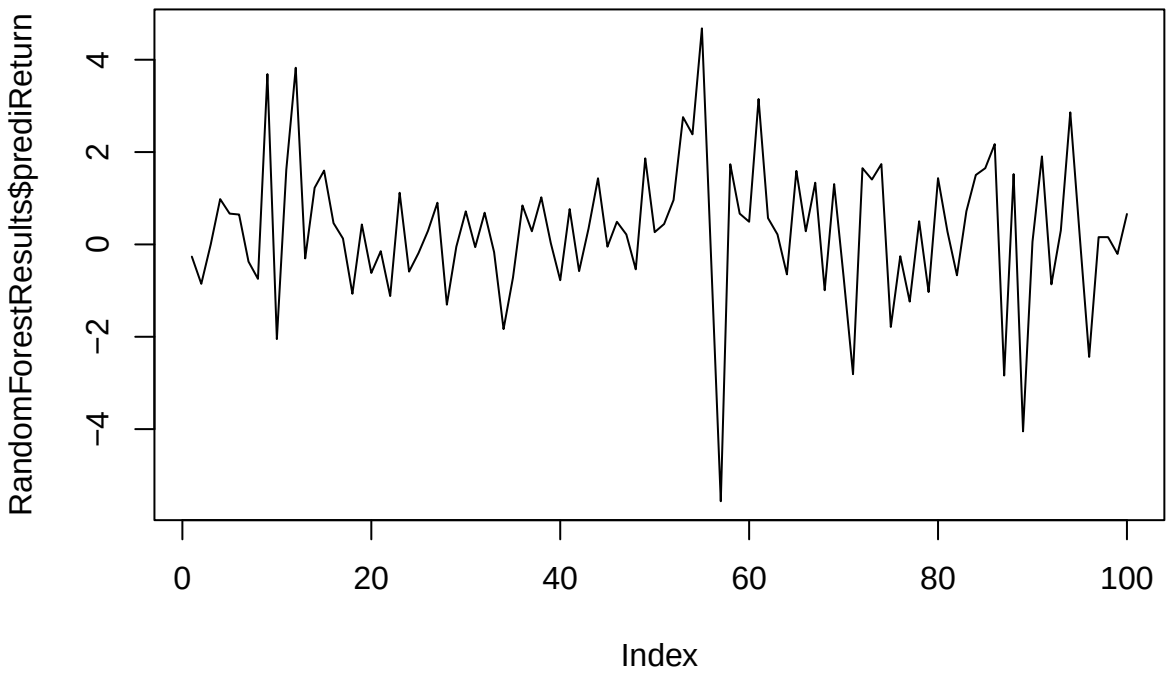
```
set.seed(42)
# Creating the Random Forests model:
RandomForestModel <- randomForest(factor(tomorrowCandle) ~ todayCandle +
    yesterdayCandle + reversalCandle + relativePositionToEMA7 +
      relativePositionToEMA20 +
    movingAveragesCrossing + candleSize, data = train, ntree=500 )
# As discussed previously, incrementing the "ntree" parameter will get better
# results at the risk of overfitting the model.

# Showing the model:
plot(RandomForestModel)
```




```
varImpPlot(RandomForestModel)
```




```r
# Making the predictions:
RandomForestPredictions <- predict(RandomForestModel, test)


# Calculating the predicted revenues income returns:
RandomForestResults <- stocksYield(test, RandomForestPredictions)

# Showing the Prediction results:
plot(RandomForestResults$prediReturn, type = "l")
```




```
# Cumulative revenue:
plot(RandomForestResults$cumReturn, type = "l")
```


```
# Computing and showing the confusion matrix:
confusionmatrix.rf <- table(RandomForestResults$tomorrowCandle,
                            RandomForestResults$predictionMade)
print(confusionmatrix.rf)
```

```
##
##         BEAR BULL
##   BEAR    26   20
##   BULL    18   36
```

```
# Calculating accuracy of the Random Forest Model:
RandomForestErrors <- mean(RandomForestResults$tomorrowCandle !=
                  RandomForestResults$predictionMade)
print(paste("Accuracy", 1 - RandomForestErrors))
```

```
## [1] "Accuracy 0.62"
```

```
(1 - RandomForestErrors) + totalAccuracy -> totalAccuracy


# Adding the accuracy to the Results table:
results <- results %>%
  add_row(Model="   Fourth Model : Random Forest Model",
                               Accuracy = 1 - RandomForestErrors
```

```
                                )
results
```

```
##                                                        Model Accuracy
## 1                          Stocks Trading System expectations     0.51
## 2    First Model : Support Vector Machine - Polynomial kernel     0.49
## 3      Second Model : Support Vector Machine - Sigmoid kernel     0.59
## 4                             Third Model : Naive Bayes Model     0.61
## 5                          Fourth Model : Random Forest Model     0.62
```

This Fourth Model, the Random Forest Model, shows again an improvement of the accuracy obtained.

2.3.5 Accuracy & Performance Comparison - Visa Inc. Case

We could by now perform a comparison between the models developed until now, calculating both their accuracy and the projected revenue that they would produce. The prices periods used to compute the models were the following:

```
print(paste("The Train dataset prices were computed between the dates ",
            rownames(train[1,]), " and " ,
            rownames(train[length(train$reversalCandle),])))
```

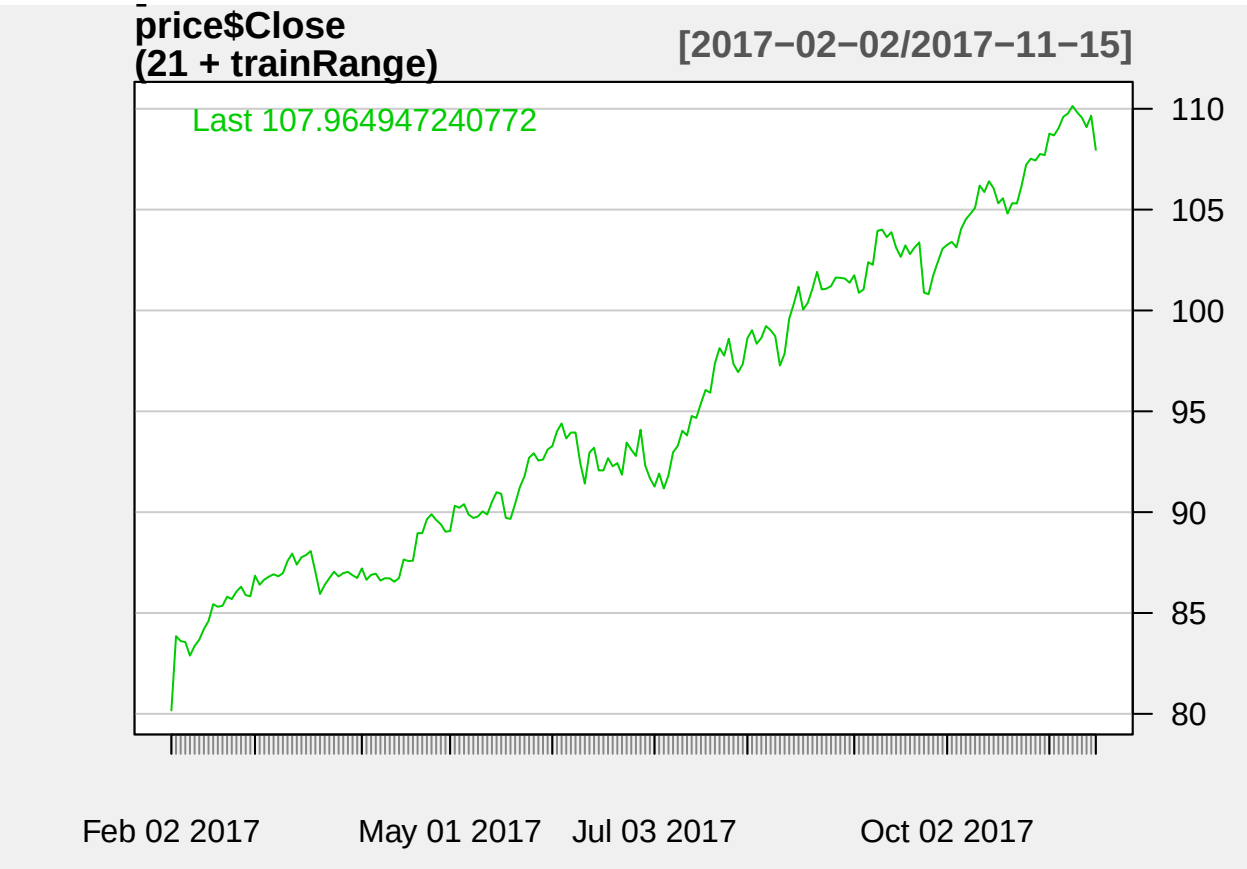
```
## [1] "The Train dataset prices were computed between the dates  2017-02-01  and  2017-11-14"
```

```
print(paste("The Train dataset prices were computed between the dates ",
            rownames(test[1,]), " and " ,
            rownames(test[length(test$reversalCandle),])))
```

```
## [1] "The Train dataset prices were computed between the dates  2017-11-15  and  2018-04-11"
```

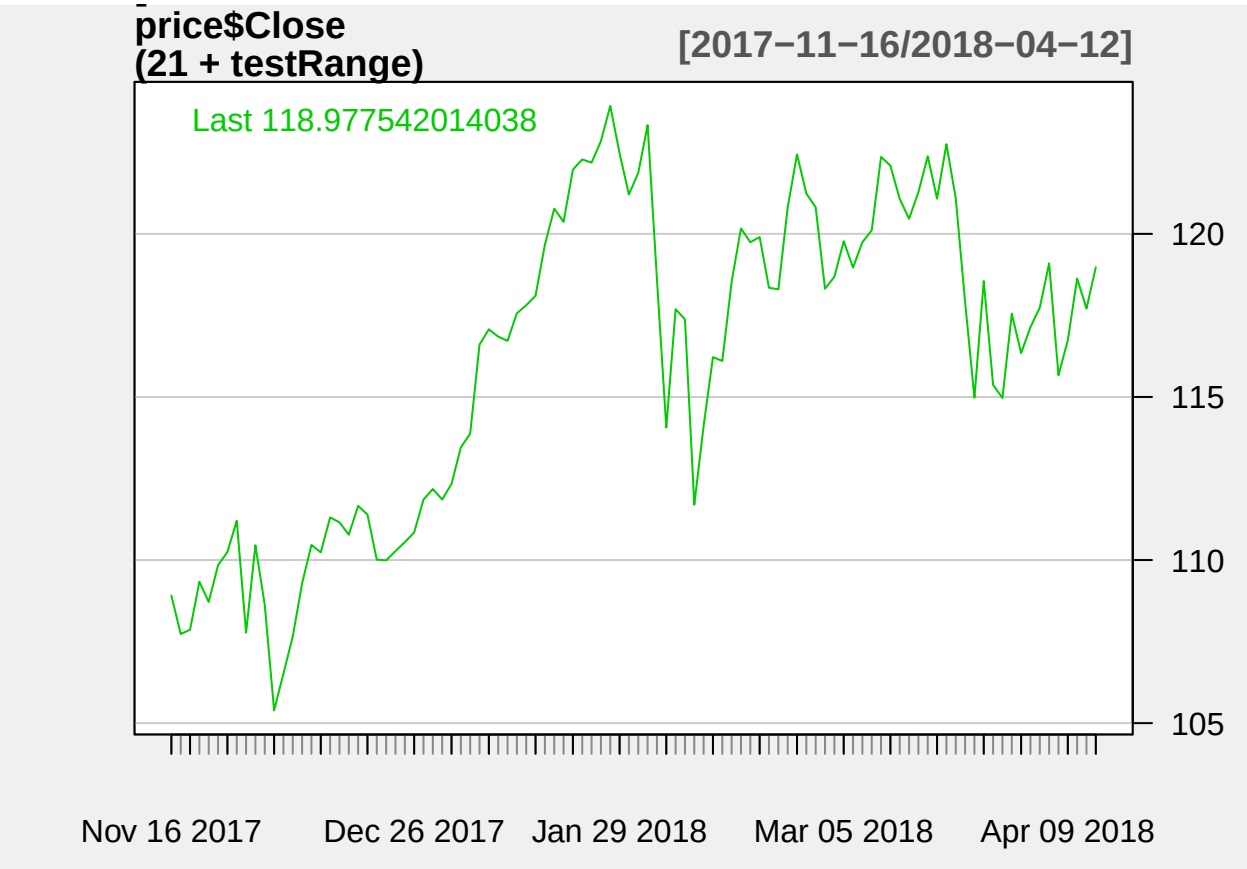
And the corresponding prices for those dates are the following:

```
# Plotting the training and testing sets :
chartSeries(price$Close[(21+trainRange),], theme = "white") + neuralTheme
```




```
## NULL
```

```
chartSeries(price$Close[(21+testRange),], theme = "white") + neuralTheme
```

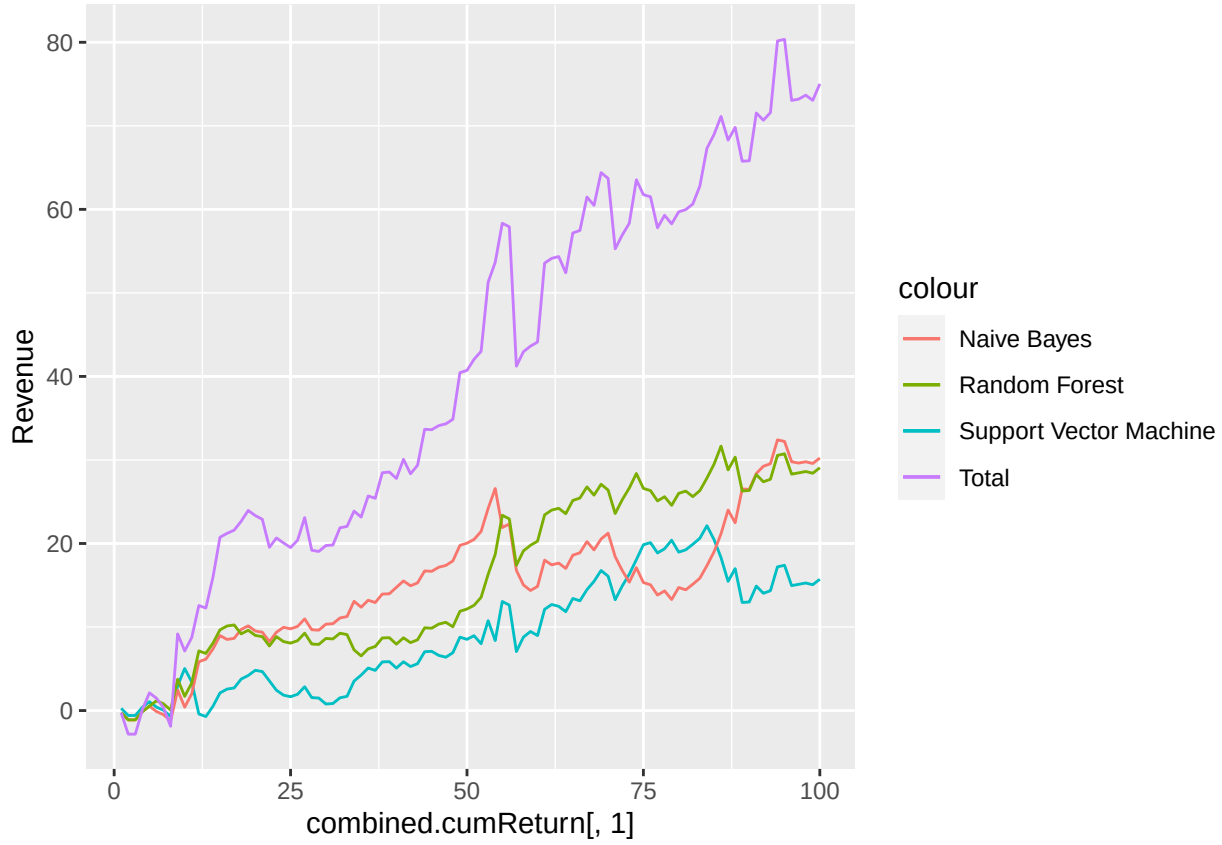



```
## NULL
```

The revenues produced by the previous three models developed, are the following:

```
library(ggplot2)
NaiveBayesRevenue <- NaiveBayesResults$cumReturn
SupportVectorMachineRevenue <- SupportVectorMachineResults$cumReturn
RandomForestRevenue <- RandomForestResults$cumReturn
total.cumReturn <- NaiveBayesRevenue + SupportVectorMachineRevenue +
  RandomForestRevenue

names(NaiveBayesRevenue) <- "cumreturn"
names(SupportVectorMachineRevenue) <- "cumreturn"
names(RandomForestRevenue) <- "cumreturn"


# Transform the vector into a data.frame
combined.cumReturn <- data.frame(c(1:length(NaiveBayesRevenue)),NaiveBayesRevenue,
                                 SupportVectorMachineRevenue,RandomForestRevenue,
                                 total.cumReturn)

# Plot all cumulative revenues for all three models:
print(ggplot(combined.cumReturn) +
        geom_line(aes(combined.cumReturn[,1],
                      SupportVectorMachineRevenue,
```

```
                    colour = "Support Vector Machine")) +
        geom_line(aes(combined.cumReturn[,1], NaiveBayesRevenue,
                    colour = "Naive Bayes")) +
        geom_line(aes(combined.cumReturn[,1], RandomForestRevenue,
                    colour = "Random Forest")) +
        geom_line(aes(combined.cumReturn[,1], total.cumReturn,
                    colour = "Total")) +
        ylab("Revenue")

        )
```



Cleaning up RAM memory:

```
rm( price, train, test, candleSize, combined.cumReturn,
    ema20, ema7, movingAveragesCrossing, RandomForestModel,
    relativePositionToEMA20, relativePositionToEMA7, reversalCandle,
    todayCandle, yesterdayCandle, SupportVectorMachineResults,
    dsPredictors, NaiveBayesModel, NaiveBayesResults, neuralTheme,
    RandomForestResults, tomorrowCandle, SupportVectorMachineModel, V)
gc(verbose = getOption("verbose"), reset = FALSE, full = TRUE)
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2664988 142.4    5032495 268.8  3772894 201.5
## Vcells  4580300  35.0   10146329  77.5  8388608  64.0
```

3. Methods/Analysis Section - Apple Inc. Case

Through the following sections, a Neural Network Model will be developed, exploring market stocks from another famous firm, Apple, and enhancing the predictors used previously. Accordingly, different computing software will be used, and compared. We will base our research upon two R Neural Network software packages, Caret and Neuralnet, and compare the results obtained.

A Neural Network is a Machine Learning Supervised Algorithm and can be viewed as analogous to the human nervous system, since it is made up of interconnected nodes called neurons, which are used as interconnected information processing units. Instead of analyzing data in a linear manner, the Neural Network works as a parallel processing of information, which allows it to deal with non-linearity. The Neural Networks allows to infer meaning and to detect patterns inside complex data sets.

Similarly to the human nervous system, inside a Neural Network the information flows through interconnected units called Neurons, in a analogous manner to the information passage through neurons in alive animals. The Neural Network consists in layers of neurons: the first one receives the raw input, processes it and passes the processed information to the hidden layers, which in turn pass the processed information to the last layer which output the results.

A great advantage of Neural Network is, being adaptive in nature, it can learn from the input provided. The simplest one-layer neural network, is called Perceptron, and processes its input using a weighted summation and an activation function. It counts with a learning algorithm that optimize the weights in the summation processor. But the Perceptron model is very limited because of its inability to deal with non-linearity. This kind of problem is confronted by using multilayered Neural Networks, which can solve non-linear problems. The input layer passes the information to the hidden layer, which in turn do the same towards the output layer.

The number of layers and neurons is crucial in the Neural Network capability of learning. A Neural Network using no hidden layers, is only capable of representing linear separable functions or decisions. Using a single hidden layer of neurons, a Neural Network can approximate any function that contains a continuous mapping from one finite space to another, while if composed of two hidden layers, it can represent an arbitrary decision boundary with rational activation functions and can approximate any smooth mapping to any accuracy. However, determining the number of hidden layers is only a part of the problem, since we need also to determine how many neurons will be in each of the hidden layers.

Too few neurons in the hidden layers will cause underfitting, meaning that the Neural Network wouldn't adequately train a complicated data set. But the opposite, too many neurons in the hidden layers may result in overfitting, meaning that the limited amount of information in the input, is not enough to train the information processing capacity of the system, i.e. all of the neurons in the hidden layers. Also, the large number of neurons augments the time needed to train the network, so much that it would be impossible to adequately train the neural network. That's why there are some rules to determine the number of neurons on the system: neurons should be less than twice the size of the input layer, should be between the number of nodes on the input layer and the size of the output layer, and should be more or less 2/3 of the size of the input layer + the output layer.

In this project, because we have seven nodes in the input layer, and two on the output layer, we choose accordingly seven neurons, placed inside only one hidden layer.

Another way of determining the number of neurons in the hidden layers, is applying the following formula: neurons ~ sqrt(input layer nodes * output layer nodes)

Using the above formula, and because we have seven input nodes and two output nodes, we could state the following: neurons = sqrt(7 * 2) = sqrt(14) = 3.74 ~ 4 Nonetheless, we found more accurate using four neurons as for this second method of determining the number of hidden nodes.

Also, concerning the number of layers, should be took into account that a single hidden layer in neural networks is capable of performing universal approximation[1]. The Universal Approximation Theorem states that the results of the first layer of a Neural Network can approximate any well-behaved function[2]. Such a function can also be approximated by a network of greater depth by using the same construction for the

first layer and approximating the identity function with later layers. The first version of this theorem was proposed by Cybenko [3] for sigmoid activation functions, and later Hornik expanded this showing that the multilayer architecture itself allows neural networks the potential of being universal approximators[4].

The connections between neurons are weighted and those weights are optimized using a learning rule. The learning rules can be used in conjunction with several algorithms: a. back-propagation; b. resilient back-propagation using the sign (increment/decrement) of the error between iterations, thus with or without weight backtracking; c. modified globally convergent algorithm (grprop). In our project we use resilient back-propagation with weight backtracking. The learning rule is used to calculate the error at the output unit. This error is backpropagated to all the units such that the error at each unit is proportional to the contribution of that unit towards the total error at the output unit.
The errors at each unit are then used to optimize the weight at each connection.

References 1. Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314

2. Pinkus, Allan (January 1999). "Approximation theory of the MLP model in neural networks". Acta Numerica. 8: 143–195.
3. Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314
4. Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks, 4(2), 251–257.

3.1 Techniques for Data Loading and Preparation

The data on which is based this Neural Network research, is different from the previously used. This time, there will be downloaded stocks prices from Apple. That's why on this section we explore the newly loaded dataset. As a first necessary step, we install the required software and download the stocks prices from Yahoo Finance.

3.1.1 Loading Apple Stocks prices from Yahoo Finance

Here the Apple Stock prices will be loaded and used for training and testing the Machine Learning Neural Network model. We will try to download the data from Yahoo, performing 10 attempts:

```
i <- 1
maxTries <- 10
while (i < maxTries) {

  try(
    getSymbols("AAPL",
          from ="2017-01-01",
          to = "2020-12-31",
          src =  "yahoo",
          adjust =  TRUE)
  )
  if (exists("AAPL")) {
    break
  }
   i <- i + 1
}
```

```r
# Create a dataframe with the downloaded data:
APPLEPrices <- as.data.frame(AAPL)
APPLEPrices$Date <- as.Date(rownames(APPLEPrices))
head(APPLEPrices)
```

```
##            AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted
## 2017-01-03  6.854525  6.885897 6.792964   6.875242   115127600      27.33247
## 2017-01-04  6.857484  6.896552 6.851565   6.867547    84472400      27.30188
## 2017-01-05  6.861628  6.917269 6.855116   6.902471    88774400      27.44072
## 2017-01-06  6.912534  6.994220 6.894184   6.979422   127007600      27.74663
## 2017-01-09  6.981789  7.069395 6.981197   7.043349   134247600      28.00078
## 2017-01-10  7.030327  7.066435 7.002507   7.050453    97848400      28.02902
##                  Date
## 2017-01-03 2017-01-03
## 2017-01-04 2017-01-04
## 2017-01-05 2017-01-05
## 2017-01-06 2017-01-06
## 2017-01-09 2017-01-09
## 2017-01-10 2017-01-10
```

```r
names(APPLEPrices) <- c("Open","High","Low","Close","Volume","Adjusted","Date")

# Discarding the columns that are not needed:
APPLEPrices <- APPLEPrices %>% select(c(Date,Open,High,Low,Close))
APPLEPrices<-xts(APPLEPrices[, -1, drop=FALSE],  order.by=APPLEPrices[,1])
```

3.2 Data Analysis Exploration & Modeling

3.2.1 Preliminar Observations

In this section, we analyze the data provided and the relations between the predictors. At this stage, we take a look at the loaded dataset:

```r
# Prices dataset glimpse:
nrow(APPLEPrices)
```

```
## [1] 1006
```

```r
str(`APPLEPrices`)
```

```
## An 'xts' object on 2017-01-03/2020-12-30 containing:
##   Data: num [1:1006, 1:4] 6.85 6.86 6.86 6.91 6.98 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:4] "Open" "High" "Low" "Close"
##   Indexed by objects of class: [Date] TZ: UTC
##   xts Attributes:
##  NULL
```

```
class(APPLEPrices)
```

```
## [1] "xts" "zoo"
```

```
head(APPLEPrices)
```

```
##                Open     High      Low    Close
## 2017-01-03 6.854525 6.885897 6.792964 6.875242
## 2017-01-04 6.857484 6.896552 6.851565 6.867547
## 2017-01-05 6.861628 6.917269 6.855116 6.902471
## 2017-01-06 6.912534 6.994220 6.894184 6.979422
## 2017-01-09 6.981789 7.069395 6.981197 7.043349
## 2017-01-10 7.030327 7.066435 7.002507 7.050453
```

```
length(APPLEPrices)
```

```
## [1] 4024
```

```
ncol(APPLEPrices)
```

```
## [1] 4
```

At first glance, we can see that the Apple dataframe is in tidy format, i.e. each row represents only one observation, being the column names the prices features.

3.2.2 Features Analysis

3.2.2.1 - Analysis of the Stock Prices Let's see some general data on the prices:

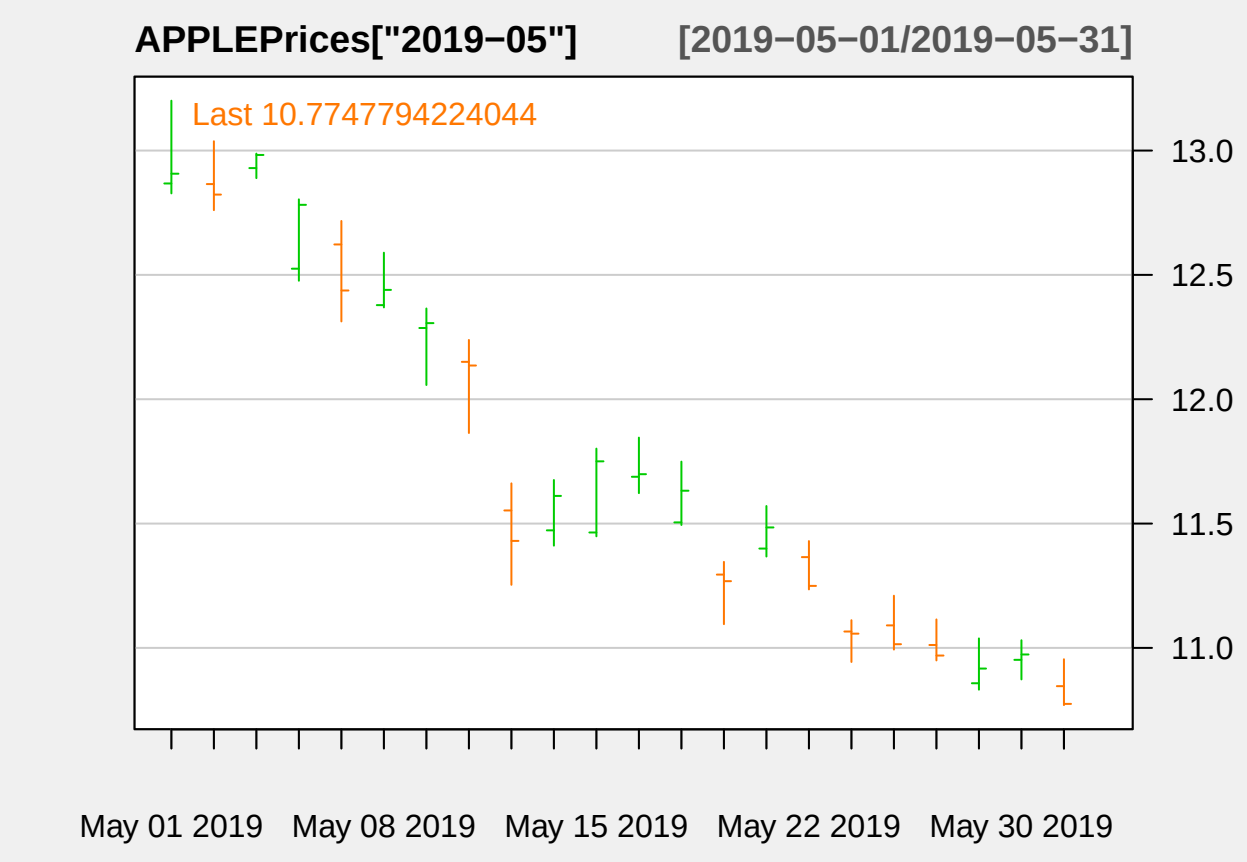
```
summary(APPLEPrices$Close)
```

```
##      Index               Close
##  Min.   :2017-01-03   Min.   :  6.868
##  1st Qu.:2018-01-02   1st Qu.: 10.003
##  Median :2019-01-02   Median : 11.732
##  Mean   :2019-01-01   Mean   : 21.792
##  3rd Qu.:2020-01-01   3rd Qu.: 16.538
##  Max.   :2020-12-30   Max.   :136.690
```

The mean closing price of the Visa stock is 21.792, the Minimal price is 6.868, and the maximal closing price is 136.690 usd.

To get a first approach to the Technical Trading Analysis features, the following graph shows the Apple prices as of May 2019, represented by Bar icons, which place greater emphasis on the closing price of the stock in relation to the previous day closing prices. Also, we create a new theme for Neural Networks.

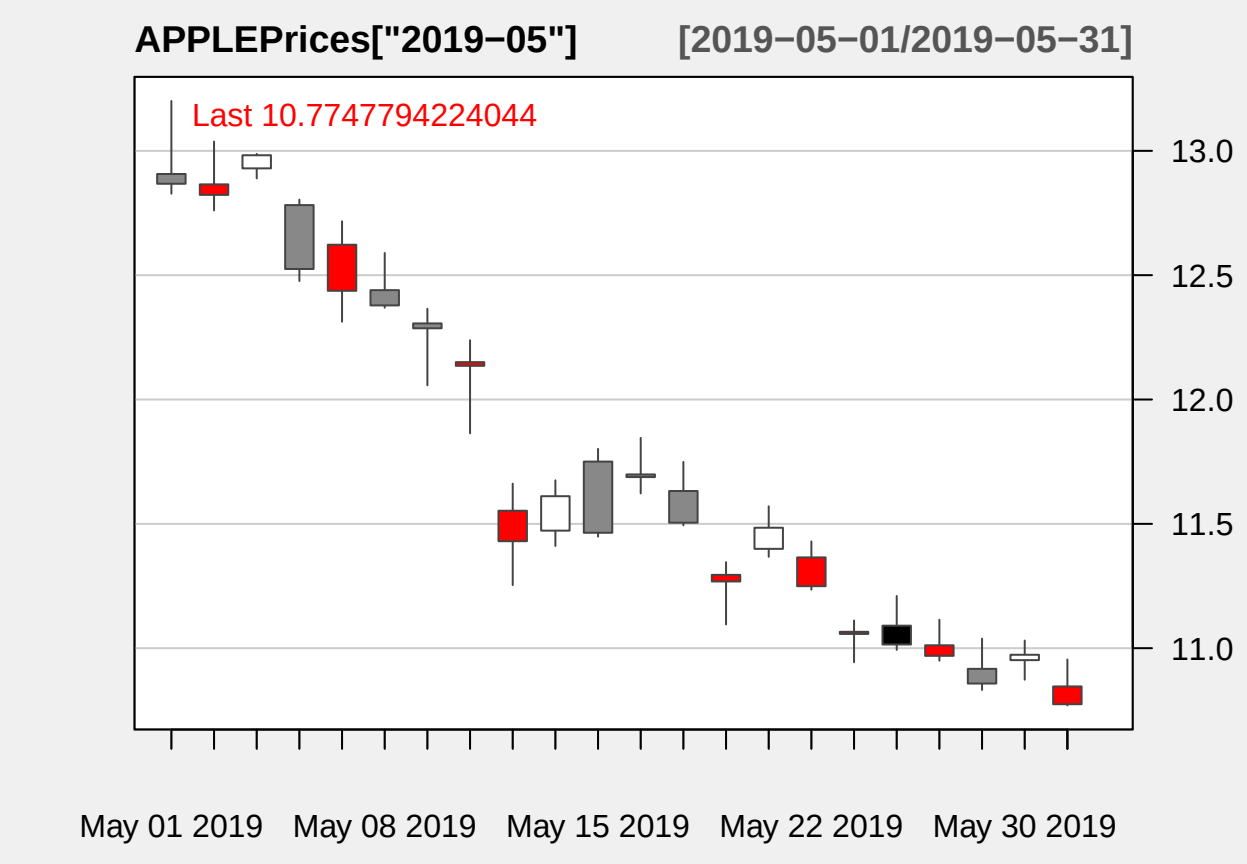
```
# Creating a custom Theme for be used on graphs :
neuralTheme <- theme_fivethirtyeight() +
  theme(panel.background = element_rect(fill = "aliceblue"),
        text = element_text( face = "italic",
                             colour = "black", size = 8,
                             lineheight = 0.9,  hjust = 0.5,
                             vjust = 0.5, angle = 0,
                             #margin = margin(),
                             debug = FALSE),
        axis.title = element_text(colour = "gray" ))

barChart(APPLEPrices['2019-05'], theme = "white") + neuralTheme
```




```
## NULL
```

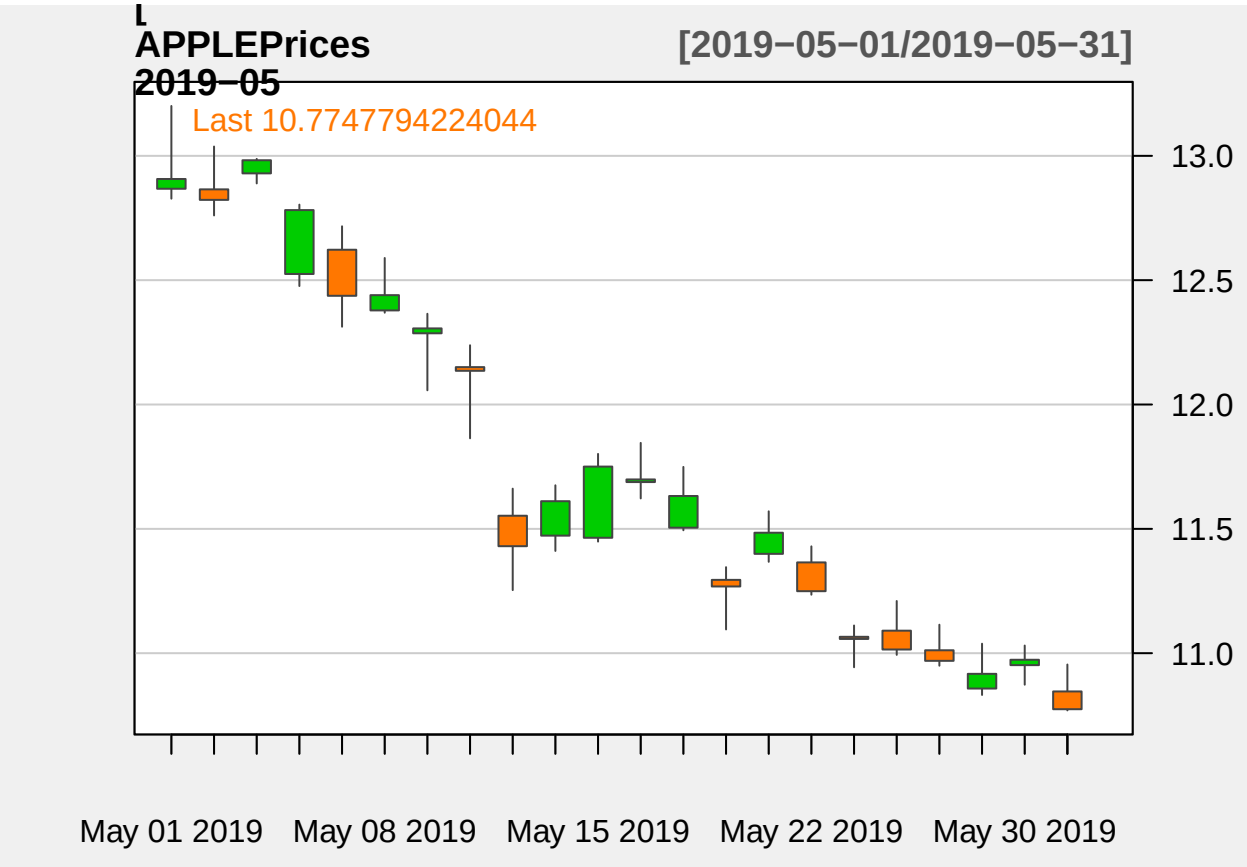
Bar charts are pretty well informative, however, the candlestick charts place the highest importance to the close prices as it relates to the open of the same day, which can be seen on the following charts, which show the same data for the Apple stocks, this time using Japanese Candles, in two different formats. Another two ways of seeing the Apple prices :

```
candleChart(APPLEPrices['2019-05'],multi.col=TRUE,theme="white") + neuralTheme
```




```
## NULL
```

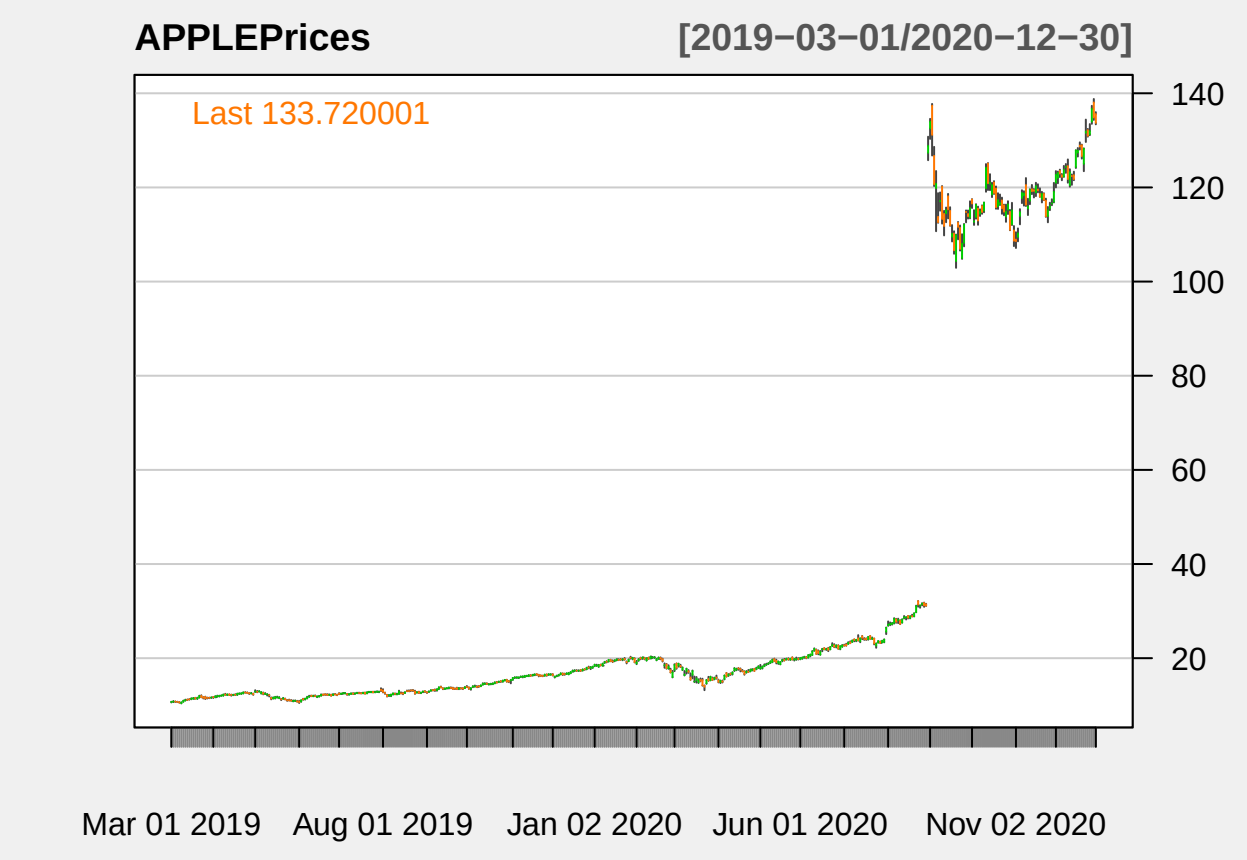
```
chartSeries(APPLEPrices['2019-05'], theme = "white")  + neuralTheme
```




```
## NULL
```

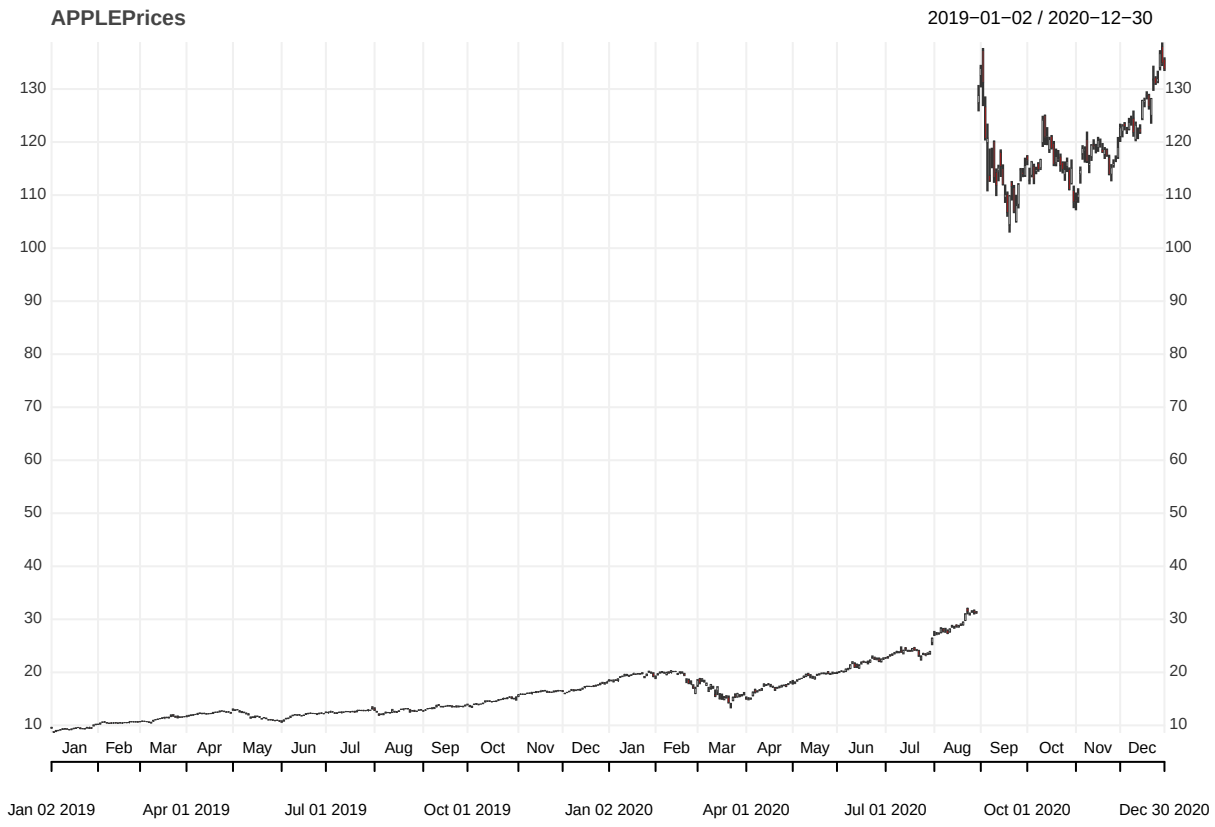
When representing longer periods of data, the candlesticks charts can become quite difficult to follow. This two charts show the prices for the entire period of stock prices loaded for Visa :

```
chartSeries(APPLEPrices, subset = "2019-03::",theme = "white") + neuralTheme
```



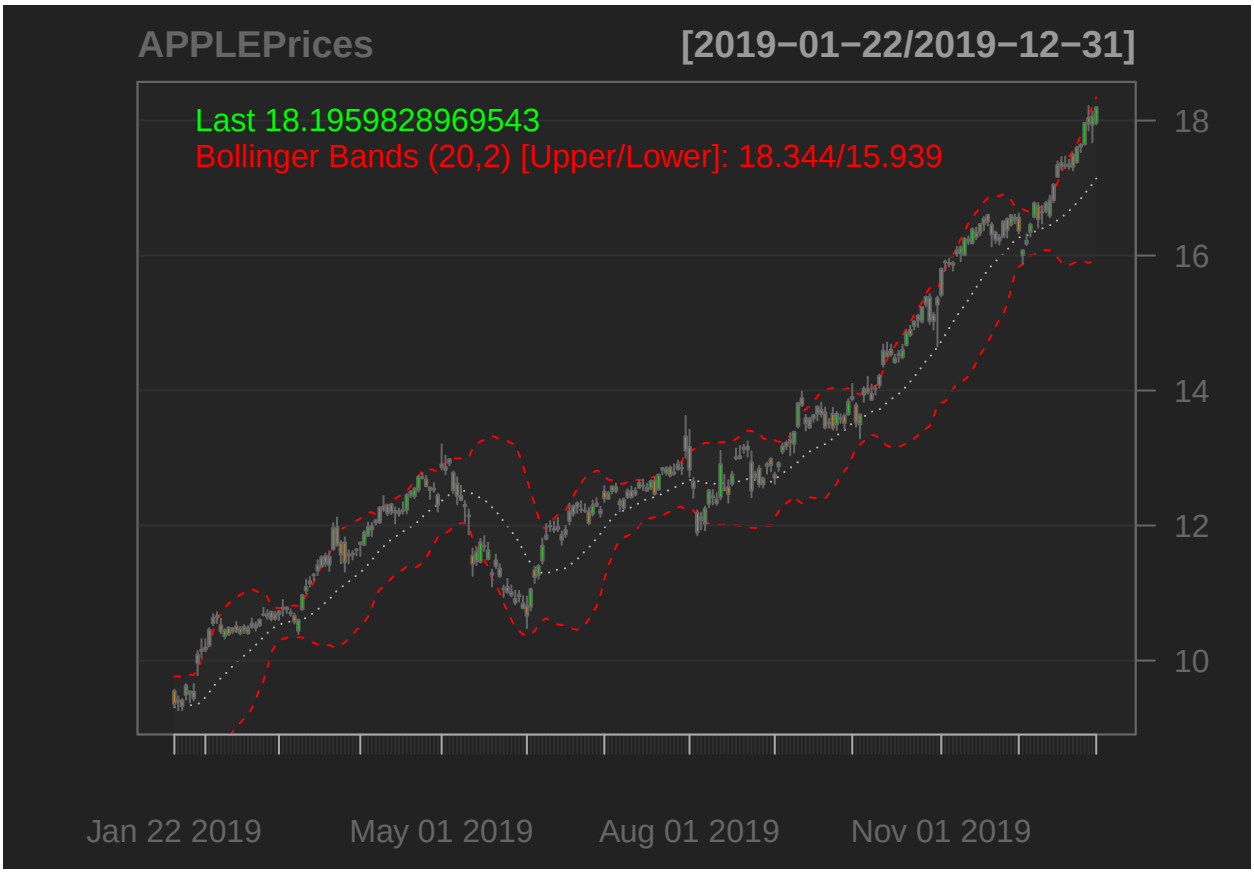

```
## NULL
```

```
chart_Series(APPLEPrices, subset = "2019/")
```

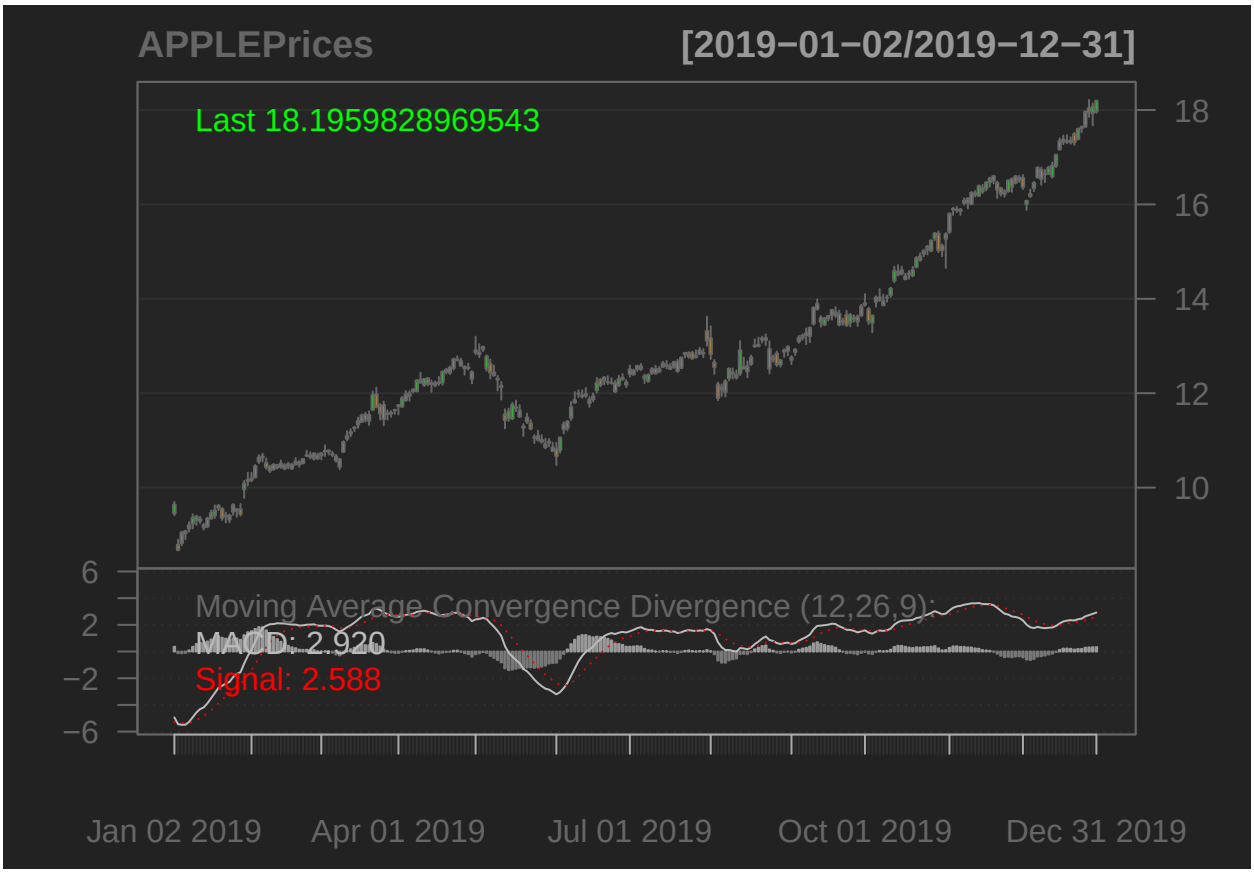


The Technical Analysis usually builds upon Indicators, or mathematical computations, such as Bollinger Bands, MACD Moving Averages, etc. The following plots represents some of those indicators applied to the data that we have loaded from Yahoo right now:

```
chartSeries(APPLEPrices, TA="addBBands(n=20)", subset="2019-01-21::2019-12-31")
```




```
candleChart(APPLEPrices, TA=c(addMACD(),addVo()), subset = '2019') + neuralTheme
```




```
## NULL
```

3.2.3 Data Cleaning, Preparation and Partition

3.2.3.1 Creating the predictors for the Model

The next step, before proceeding to the partition of the data in training and test datasets, is to create the columns that will be used for the computations.

The algorithms used on the present section, will compute on Moving Averages. As previously explained, the Moving Averages are trend following indicators, that show the trends of the market. They are useful in order to follow the prices trends.

The Moving averages are means for a certain past number of trading days, and this way is possible to reduce the price fluctuations (noise) also known as "Volatility". We could say that the moving averages filter fluctuations in the prices, which in turn allows us to perceive the prices trends without the "noise".

Between the four different types of moving averages, on this work we'll use Exponential Moving Averages (EMAs).

As stated before, there exists some kind of trade-off concerning to the periods of time upon which the Moving Averages are calculated.

If the moving averages are computed on short time periods, they are called "fast-moving" averages, and will be highly responsive to the price changes. However, if the moving average uses a longer time period to calculate the average, it will be a slow moving average, therefore less responsive, and its results will be affected by some kind of lag. That means, it will be less "noise", but the responsiveness will be notoriously slow.

Some kind of compromise should be attained, between responsiveness and noise avoiding.

On previous steps of the present research, moving averages with a short periods, specifically seven days and twenty days, were used as predictors. They represent the short term trends.

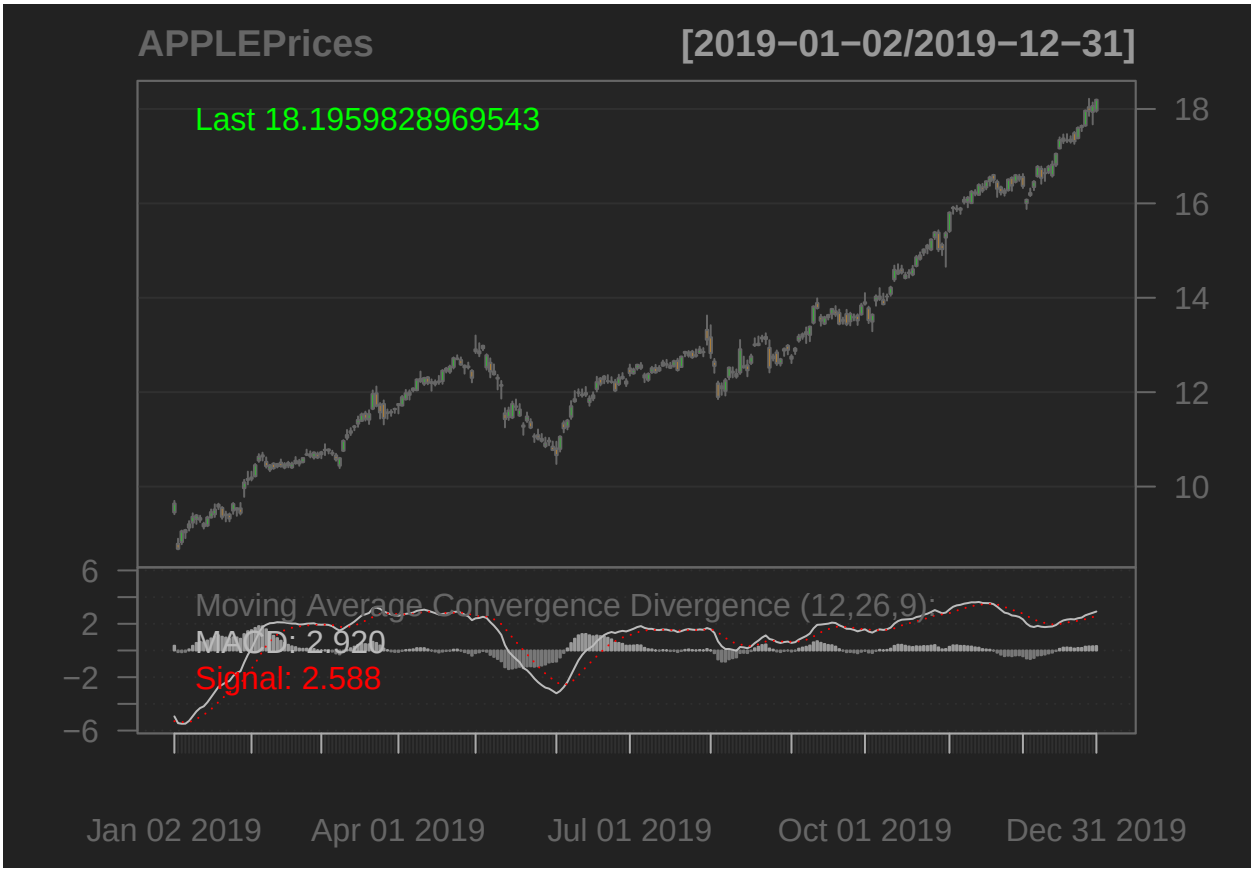
On the present step, we'll use a fifty days moving average period, which shows a medium trend. Short term traders use a 21-period moving average, Medium-term will use a 50 day EMA, and long term traders use 200 periods EMAs.

We'll use the computation for the Moving Average known as "exponential", which make it quicker to react to the price changes, where every point included in the moving average decreases in weight over time.

Therefore, on this point of this project, Exponential Moving Averages for daily periods of 20 and 50 days will be used respectively.

Because for this algorithm, an Exponential Moving Average (EMA) of 50 days interval times will be used, such an indicator is shown here, to get an intuitive view of its utilization:

```
APPLEPrices$EMA <- EMA(Cl(APPLEPrices), n = 50)
candleChart(APPLEPrices, TA=c(addMACD(),addVo()), subset = '2019')
```




```
plot(APPLEPrices)
```

Next, we will compute the predictors for the Models, using the stocks prices of Apple, downloaded from Yahoo.

The following predictor will express the current day candle type:

```
todaysCandle<-data.frame(ifelse(APPLEPrices$Close > APPLEPrices$Open,"BULL","BEAR"))
```

After creating this field, more predictors will be created using the created column. The following predictor will express the previous candle type. For this purpose, the "lag" function will be used:

```
yesterdaysCandle<-data.frame(lag(todaysCandle$Close,n=1))
```

This predictor will express the next (future) candle type: the "lead" function will be used:

```
tomorrowsCandle<-data.frame(lead(todaysCandle$Close, n = 1))
```

As stated above, a "Reversal" is when a trend change to the opposite direction. When the trend is going to change direction, the appearance of the candle usually changes too, revealing that the trend is about to change direction.

The algorithm used on this project will research the reversal candle known as "Doji", which represents indecision in the market.

As explained before, on the present work we'll use three types of Doji candles:

1. the Standard Doji: is a single candlestick where the Open and Close prices are pretty much the same.

2. the Dragonfly Doji: considered a bullish (UP) reversal pattern, is a candle where the price closes about the same as the opening price, and have a long lower "shadow" (that means, the price went downwards, and later remonted), telling that the market is testing to find out where "support" or demand is, or in other words, the sellers were selling but found a strong opposition made by the stock buyers.

3. Gravestone Doji: the opposite of the Dragonfly Doji, and considered a bearish (DOWN) reversal pattern, is a candle where the price closes about the same as the opening price, with a long upper "shadow", meaning that the price went upwards, but the buyers found a strong opposition made by the stock sellers. Investors see this pattern as a time to sell, or exit the market.

Now, in addition to the previously researched Standard Doji candle, we will also research both Gravestone Doji and Dragonfly Doji candles. According to the prices of the stock, the size of the Doji Reversal candle will be fixed to 0.01 usd :

```
dojiSize <- 0.01

# Calculation of the required size of the Doji candles:
today.size <-data.frame(abs(APPLEPrices$Close - APPLEPrices$Open))
size_avg <- mean(today.size$Close)
```

Now we create a predictor containing both Gravestone and Dragonfly Dojis:

```
enhancedReversalCandle<-data.frame(ifelse(

  (abs(APPLEPrices$Close - APPLEPrices$Open)<dojiSize)
  &
  (
                          ((APPLEPrices$High - APPLEPrices$Low) >
                             size_avg)
                          |
                            ((APPLEPrices$Low - APPLEPrices$High) >
                             size_avg)
   )
                          ,"YES","NO")
  )
```

On this calculation, we choose as enhancedReversalCandle the ones that have a "shadow" or tail bigger than the average size of all candles on data, that means, that the difference between the High and Low prices, or the opposite, between Low and High, is greater than the average gap between Open-Close prices.

Checking the quantity of Reversal candles on the dataset:

```
countOfDoji<-length(enhancedReversalCandle[enhancedReversalCandle$Close == "YES",])
countOfDoji
```

```
## [1] 5
```

As explained above, during an uptrend, prices tend to stay above the moving averages. And accordingly, in a downtrend, prices tend to stay below the moving average. The position of the moving average relative to the price indicates the trend. The next three predictors will express the Exponential Moving Averages (EMA) for 20 days and 50 days respectively, and its crosses:

```
movingAverage20 <- EMA(APPLEPrices$Close, n = 20)
ema20 <- EMA(APPLEPrices$Close, n = 50)
priceRelativeToEMA20<-data.frame(
  ifelse(APPLEPrices$Close > movingAverage20,"ABOVE","BELOW"))
priceRelativeToEMA50<-data.frame(ifelse(APPLEPrices$Close > ema20,"ABOVE","BELOW"))
```

Now there is calculated the cross-over of those two EMAs:

```
EMA20_50Crossings <-data.frame(ifelse(movingAverage20 > ema20,"ABOVE","BELOW"))
```

The following predictor will reflect the size of the candles:

```
candleSizeNN <-data.frame(abs(APPLEPrices$Close - APPLEPrices$Open))
```

The following predictor will be useful to calculate the income or revenue that will be obtained in case that the predictions were correct:

```
revenueNN <- lead(candleSizeNN$Close, n = 1)


## Next, a dataframe with the FEATURES as columns is created:
dsData<-data.frame(todaysCandle,
                   yesterdaysCandle,
                   enhancedReversalCandle,
                   priceRelativeToEMA20,
                   priceRelativeToEMA50,
                   EMA20_50Crossings ,
                   candleSizeNN,
                   revenueNN,
                   tomorrowsCandle)

names(dsData)<-c("todaysCandle",
                 "yesterdaysCandle",
                 "enhancedReversalCandle",
                 "priceRelativeToEMA20",
                 "priceRelativeToEMA50",
                 "EMA20_50Crossings" ,
                 "candleSizeNN",
                 "revenueNN",
                 "tomorrowsCandle")

# A quick glimpse of the dataframe:
head(dsData,20)
```

```
##            todaysCandle yesterdaysCandle enhancedReversalCandle
## 2017-01-03         BULL             <NA>                     NO
## 2017-01-04         BULL             BULL                     NO
## 2017-01-05         BULL             BULL                     NO
## 2017-01-06         BULL             BULL                     NO
## 2017-01-09         BULL             BULL                     NO
## 2017-01-10         BULL             BULL                     NO
## 2017-01-11         BULL             BULL                     NO
```

```
## 2017-01-12         BULL             BULL                    NO
## 2017-01-13         BEAR             BULL                    NO
## 2017-01-17         BULL             BEAR                    NO
## 2017-01-18         BEAR             BULL                    NO
## 2017-01-19         BULL             BEAR                    NO
## 2017-01-20         BEAR             BULL                    NO
## 2017-01-23         BULL             BEAR                    NO
## 2017-01-24         BULL             BULL                    NO
## 2017-01-25         BULL             BULL                    NO
## 2017-01-26         BULL             BULL                    NO
## 2017-01-27         BEAR             BULL                    NO
## 2017-01-30         BULL             BEAR                    NO
## 2017-01-31         BULL             BULL                    NO
##            priceRelativeToEMA20 priceRelativeToEMA50 EMA20_50Crossings
## 2017-01-03                 <NA>                 <NA>              <NA>
## 2017-01-04                 <NA>                 <NA>              <NA>
## 2017-01-05                 <NA>                 <NA>              <NA>
## 2017-01-06                 <NA>                 <NA>              <NA>
## 2017-01-09                 <NA>                 <NA>              <NA>
## 2017-01-10                 <NA>                 <NA>              <NA>
## 2017-01-11                 <NA>                 <NA>              <NA>
## 2017-01-12                 <NA>                 <NA>              <NA>
## 2017-01-13                 <NA>                 <NA>              <NA>
## 2017-01-17                 <NA>                 <NA>              <NA>
## 2017-01-18                 <NA>                 <NA>              <NA>
## 2017-01-19                 <NA>                 <NA>              <NA>
## 2017-01-20                 <NA>                 <NA>              <NA>
## 2017-01-23                 <NA>                 <NA>              <NA>
## 2017-01-24                 <NA>                 <NA>              <NA>
## 2017-01-25                 <NA>                 <NA>              <NA>
## 2017-01-26                 <NA>                 <NA>              <NA>
## 2017-01-27                 <NA>                 <NA>              <NA>
## 2017-01-30                 <NA>                 <NA>              <NA>
## 2017-01-31                ABOVE                 <NA>              <NA>
##            candleSizeNN    revenueNN tomorrowsCandle
## 2017-01-03 0.0207172385 0.0100625370            BULL
## 2017-01-04 0.0100625370 0.0408430228            BULL
## 2017-01-05 0.0408430228 0.0668880857            BULL
## 2017-01-06 0.0668880857 0.0615604981            BULL
## 2017-01-09 0.0615604981 0.0201257842            BULL
## 2017-01-10 0.0201257842 0.0597849513            BULL
## 2017-01-11 0.0597849513 0.0207174753            BULL
## 2017-01-12 0.0207174753 0.0041434951            BEAR
## 2017-01-13 0.0041434951 0.0982602626            BULL
## 2017-01-17 0.0982602626 0.0005921646            BEAR
## 2017-01-18 0.0005921646 0.0224932589            BULL
## 2017-01-19 0.0224932589 0.0266365172            BEAR
## 2017-01-20 0.0266365172 0.0047354229            BULL
## 2017-01-23 0.0047354229 0.0248607336            BULL
## 2017-01-24 0.0248607336 0.0864212317            BULL
## 2017-01-25 0.0864212317 0.0159822892            BULL
## 2017-01-26 0.0159822892 0.0112468662            BEAR
## 2017-01-27 0.0112468662 0.0414347139            BULL
## 2017-01-30 0.0414347139 0.0118385573            BULL
```

```
## 2017-01-31 0.0118385573 0.1018115930            BULL
```

```
head(yesterdaysCandle,20)
```

```
##    lag.todaysCandle.Close..n...1.
## 1                            <NA>
## 2                            BULL
## 3                            BULL
## 4                            BULL
## 5                            BULL
## 6                            BULL
## 7                            BULL
## 8                            BULL
## 9                            BULL
## 10                           BEAR
## 11                           BULL
## 12                           BEAR
## 13                           BULL
## 14                           BEAR
## 15                           BULL
## 16                           BULL
## 17                           BULL
## 18                           BULL
## 19                           BEAR
## 20                           BULL
```

```
head(todaysCandle,20)
```

```
##            Close
## 2017-01-03  BULL
## 2017-01-04  BULL
## 2017-01-05  BULL
## 2017-01-06  BULL
## 2017-01-09  BULL
## 2017-01-10  BULL
## 2017-01-11  BULL
## 2017-01-12  BULL
## 2017-01-13  BEAR
## 2017-01-17  BULL
## 2017-01-18  BEAR
## 2017-01-19  BULL
## 2017-01-20  BEAR
## 2017-01-23  BULL
## 2017-01-24  BULL
## 2017-01-25  BULL
## 2017-01-26  BULL
## 2017-01-27  BEAR
## 2017-01-30  BULL
## 2017-01-31  BULL
```

```
head(tomorrowsCandle,20)
```

```
##    lead.todaysCandle.Close..n...1.
## 1                             BULL
## 2                             BULL
## 3                             BULL
## 4                             BULL
## 5                             BULL
## 6                             BULL
## 7                             BULL
## 8                             BEAR
## 9                             BULL
## 10                            BEAR
## 11                            BULL
## 12                            BEAR
## 13                            BULL
## 14                            BULL
## 15                            BULL
## 16                            BULL
## 17                            BEAR
## 18                            BULL
## 19                            BULL
## 20                            BULL
```

3.2.3.2 Data Cleaning

The calculations performed previously, have produced NAs on the data, and in this section we'll clean it. Checking whether are there any NAs ? :

```
apply(APPLEPrices,2, function(x)sum(is.na(x)))
```

```
##  Open  High   Low Close   EMA
##     0     0     0     0    49
```

The original dataset loaded had no NAs, and the only field with NAs is the one that was created on code, with the purpose of calculating the EMAs. This problem will be reflected on the dataframe with the predictors that was created right now:

```
apply(dsData,2, function(x)sum(is.na(x)))
```

```
##           todaysCandle       yesterdaysCandle enhancedReversalCandle
##                      0                      1                      0
##   priceRelativeToEMA20   priceRelativeToEMA50      EMA20_50Crossings
##                     19                     49                     49
##           candleSizeNN              revenueNN        tomorrowsCandle
##                      0                      1                      1
```

Indeed, six columns with NAs can be detected. Eliminating the NAs produced by the EMAs computations:

```
dsData<-slice(dsData,51:length(dsData$enhancedReversalCandle))
nrow(dsData)
```

```
## [1] 956
```

```
apply(dsData,2, function(x)sum(is.na(x)))
```

```
##           todaysCandle       yesterdaysCandle enhancedReversalCandle
##                      0                      0                      0
##   priceRelativeToEMA20   priceRelativeToEMA50      EMA20_50Crossings
##                      0                      0                      0
##           candleSizeNN              revenueNN        tomorrowsCandle
##                      0                      1                      1
```

Now the NAs had been cleaned from the dataframe. The only fields with a NA (only one NA on the field), are the two columns which refers to the future. That means, the "lead" function writes NA because the final day does not exist.

For the training purposes, not the entire dataset will be used, but a portion of it, which in the following section will be partitioned in two dataframes: a training set and a test set. In a further section, the training dataset, that will be called "Train", will again be partitioned several times in training-test sets, each time consisting of different sizes. This big dataset "Train" will be created now as follows:

```
predictorsDataset<-dsData[1:950,]
```

3.2.3.3 Predictors Analysis for the Neural Network

In this section, we will specifically analyze the relevance of the predictors, i.e. will they be significant or not for the construction of the models.

The predictor "candleSizeNN" is a quantitative variable. We'll check whether this independent variable is significant in predicting the next day candle, using the Roc curve:

```
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.1.1
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following object is masked from 'package:colorspace':
##
##     coords
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
c <- roc(predictorsDataset$tomorrowsCandle ~ predictorsDataset$candleSizeNN)
```

```
## Setting levels: control = BEAR, case = BULL
```

```
## Setting direction: controls < cases
```

```
c
```

```
##
## Call:
## roc.formula(formula = predictorsDataset$tomorrowsCandle ~ predictorsDataset$candleSizeNN)
##
## Data: predictorsDataset$candleSizeNN in 443 controls (predictorsDataset$tomorrowsCandle BEAR) < 507 c
## Area under the curve: 0.5397
```

A determined variable is considered a good predictor if its area under the curve is bigger than 0.70. The area under the curve is : 0.5454
This means that this variable is not a good predictor of the next day candle.

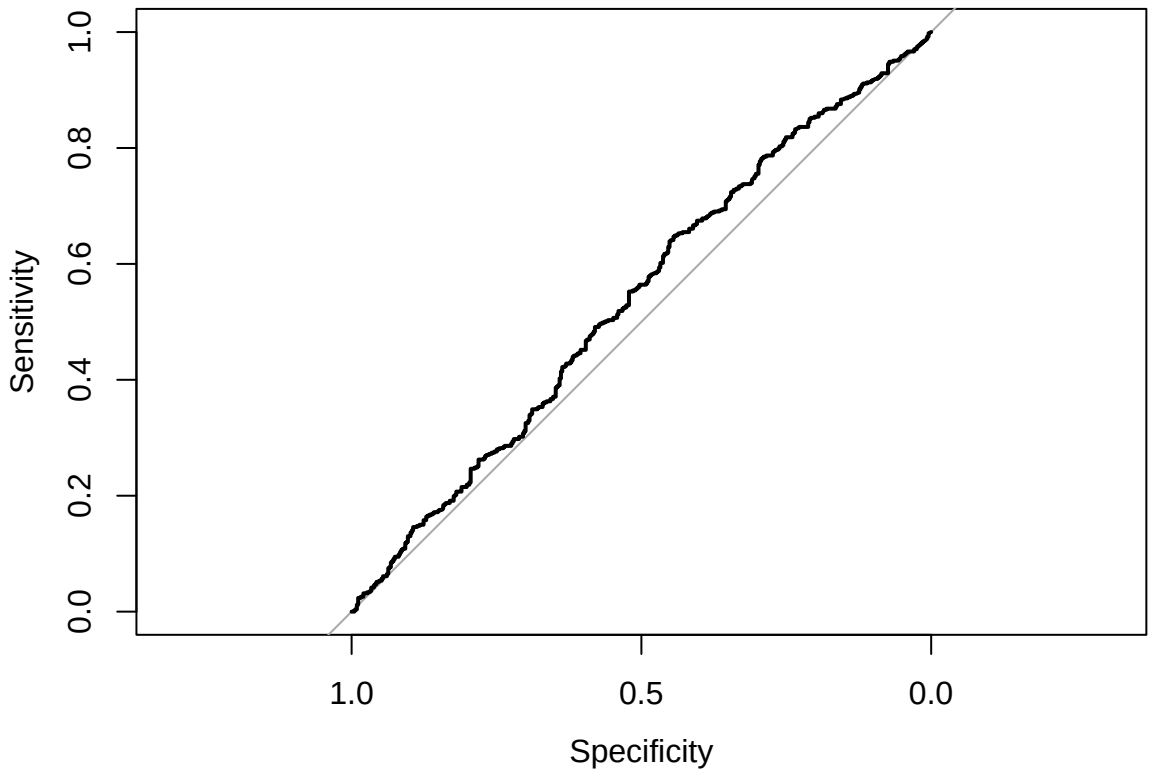
Now we check its confidence interval:

```
ci.auc(c)
```

```
## 95% CI: 0.5029-0.5765 (DeLong)
```

The variable is considered a good predictor if its CI does not include values below 0.70. In our case, the candleSizeNN variable is entirely under 0.70, which means that it is not a good predictor. This is the Roc curve for this quantitative variable:

```
plot(c)
```



We can see that the area under the curve is very small: if it would be a big area, it could be considered a good predictor, but it is not the case.

We want to know if every variable value has influence on the dependent variable, i.e. "next day candle". If that's not the case, we would discard that value. For that sake, we use XTABS function:

```
xtabs(~ tomorrowsCandle + todaysCandle,predictorsDataset)
```

```
##                todaysCandle
## tomorrowsCandle BEAR BULL
##            BEAR  193  250
##            BULL  251  256
```

```
xtabs(~ tomorrowsCandle + yesterdaysCandle,predictorsDataset)
```

```
##                yesterdaysCandle
## tomorrowsCandle BEAR BULL
##            BEAR  202  241
##            BULL  242  265
```

```
xtabs(~ tomorrowsCandle + enhancedReversalCandle,predictorsDataset)
```

```
##                enhancedReversalCandle
## tomorrowsCandle  NO YES
##            BEAR 441   2
##            BULL 504   3
```

```
xtabs(~ tomorrowsCandle + priceRelativeToEMA20,predictorsDataset)
```

```
##                priceRelativeToEMA20
## tomorrowsCandle ABOVE BELOW
##            BEAR   328   115
##            BULL   360   147
```

```
xtabs(~ tomorrowsCandle +  EMA20_50Crossings ,predictorsDataset)
```

```
##                EMA20_50Crossings
## tomorrowsCandle ABOVE BELOW
##            BEAR   368    75
##            BULL   408    99
```

We conclude that next day's BULL and BEAR candles are both represented by all the independent variables values.

3.2.3.4 Formulas for the Neural Network

First, we define the formula to be used on the Neural Network Models computation. Will we use all the columns as predictors? Or we'll use a narrow number of the fields to the computations? The predictions to be made are those concerning to the stocks prices direction tomorrow, i.e., the "tomorrowCandle".

We will define three Neural Network models: the first one including all independent variables, the second one will include two independent predictors, and the third one will allow us to perform an univariant Logistic Regression, i.e. a regression with only one predictor:

Defining the dependent variable:

```
dependent <-"tomorrowsCandle"
```

Discarding the 2 unnecessary columns (and perhaps discarding also the "enhancedReversalCandle" field, if no such type of candles were detected):

```
independent <- names(predictorsDataset)
if (countOfDoji > 0) {
  independent <- independent[!independent %in%
                             c("revenueNN","tomorrowsCandle")]
} else {
  independent <- independent[!independent %in%
                             c("revenueNN","tomorrowsCandle",
                               "enhancedReversalCandle")]
}


independent <- paste( independent, collapse = " + ")
```

Next, we define the Formulas:

```
# Model #1 : this is the preferred model: Using all predictors for the Formula:
formula1<-as.formula(paste(dependent,"~",independent))

# Model #2 : using only two predictors:
predictors2 <-c("priceRelativeToEMA20","candleSizeNN" )
features<-paste(predictors2, collapse = "+" )
formula2<-as.formula(paste(dependent,"~",features, sep=""))

# Model #3 : Univariant Logistic Regression or regression with only one predictor:
predictors3 <-c("todaysCandle")
features<-paste(predictors3, collapse = "+" )
formula3<-as.formula(paste(dependent,"~",features, sep=""))
```

3.2.3.5 Data Normalization of the dataset

In this section will be used a Neural Network, and since the dependent variable is definitely binary, diatomic, because its values can only be "BULL" or "BEAR", we'll eventually use the activation function "Logistic".

The neuralnet() function that we are going to use, only takes numeric values as input. Therefore we must perform some kind of data cleaning and preparation in order to create our Neural Network.

We normalize this variables to be sure that each variable contributes equally to the analysis. The normalization of the data will be performed using Min-Max Normalization: (X – min(X)) / (max(X) – min(X))

First, we'll normalize the variable predictorsDataset$candleSizeNN:it should represent the size of the candle. Therefore, we should convert its double values to categorical ones, which will grade from 1 to 4: from the smaller candles to the bigger ones.

Take a glimpse of the data:

```
str(predictorsDataset$candleSizeNN  )
```

```
##  num [1:950] 0.00178 0.06004 0.06302 0.13495 0.09334 ...
```

```
min(predictorsDataset$candleSizeNN )
```

```
## [1] 0
```

```
max(predictorsDataset$candleSizeNN )
```

```
## [1] 6.858171
```

```
class(predictorsDataset$candleSizeNN)
```

```
## [1] "numeric"
```

```
# Normalization of the predictorsDataset$candleSizeNN field:
predictorsDataset$candleSizeNN <-
  (predictorsDataset$candleSizeNN - min(predictorsDataset$candleSizeNN)) /
  (max(predictorsDataset$candleSizeNN)  - min(predictorsDataset$candleSizeNN ))


# Creating a Candle Size variable:
m <-mean(predictorsDataset$candleSizeNN )
length(predictorsDataset$candleSizeNN[predictorsDataset$candleSizeNN > m])
```

```
## [1] 178
```

```
length(predictorsDataset$candleSizeNN[predictorsDataset$candleSizeNN <= m])
```

```
## [1] 772
```

```
candleSizeNN <- data.frame(
                                    ifelse(predictorsDataset$candleSizeNN > (m*1.5),4,
                                    ifelse(predictorsDataset$candleSizeNN > m,3,
                                    ifelse(predictorsDataset$candleSizeNN > (m/2),2,
                                    ifelse(predictorsDataset$candleSizeNN >= 0,1,3
                                           )
                                    )
                                    )
                                    )

                    )
names(predictorsDataset$candleSizeNN)<-c("candleSizeNN")
```

Normalization of the variable “candleSizeNN” :

We’ll be using Min-Max Normalization: (X – min(X)) / (max(X) – min(X)):

```
str(predictorsDataset)
```

```
## 'data.frame':    950 obs. of  9 variables:
##  $ todaysCandle          : chr  "BEAR" "BEAR" "BULL" "BEAR" ...
##  $ yesterdaysCandle      : chr  "BULL" "BEAR" "BEAR" "BULL" ...
##  $ enhancedReversalCandle: chr  "NO" "NO" "NO" "NO" ...
##  $ priceRelativeToEMA20  : chr  "ABOVE" "ABOVE" "ABOVE" "ABOVE" ...
##  $ priceRelativeToEMA50  : chr  "ABOVE" "ABOVE" "ABOVE" "ABOVE" ...
##  $ EMA20_50Crossings     : chr  "ABOVE" "ABOVE" "ABOVE" "ABOVE" ...
##  $ candleSizeNN          : num  0.00026 0.00876 0.00919 0.01968 0.01361 ...
##  $ revenueNN             : num  0.06 0.063 0.135 0.0933 0.0202 ...
##  $ tomorrowsCandle       : chr  "BEAR" "BULL" "BEAR" "BULL" ...
```

```
min(candleSizeNN)
```

```
## [1] 1
```

```
max(candleSizeNN)
```

```
## [1] 4
```

```
candleSizeNN <- (candleSizeNN - min(candleSizeNN)) /
  (max(candleSizeNN) - min(candleSizeNN))

names( candleSizeNN)<-c("candleSizeNN")
```

Normalization of the variable "revenueNN" : doing the same as before, for the revenueNN predictor:

```
str(predictorsDataset)
```

```
## 'data.frame':    950 obs. of  9 variables:
##  $ todaysCandle          : chr  "BEAR" "BEAR" "BULL" "BEAR" ...
##  $ yesterdaysCandle      : chr  "BULL" "BEAR" "BEAR" "BULL" ...
##  $ enhancedReversalCandle: chr  "NO" "NO" "NO" "NO" ...
##  $ priceRelativeToEMA20  : chr  "ABOVE" "ABOVE" "ABOVE" "ABOVE" ...
##  $ priceRelativeToEMA50  : chr  "ABOVE" "ABOVE" "ABOVE" "ABOVE" ...
##  $ EMA20_50Crossings     : chr  "ABOVE" "ABOVE" "ABOVE" "ABOVE" ...
##  $ candleSizeNN          : num  0.00026 0.00876 0.00919 0.01968 0.01361 ...
##  $ revenueNN             : num  0.06 0.063 0.135 0.0933 0.0202 ...
##  $ tomorrowsCandle       : chr  "BEAR" "BULL" "BEAR" "BULL" ...
```

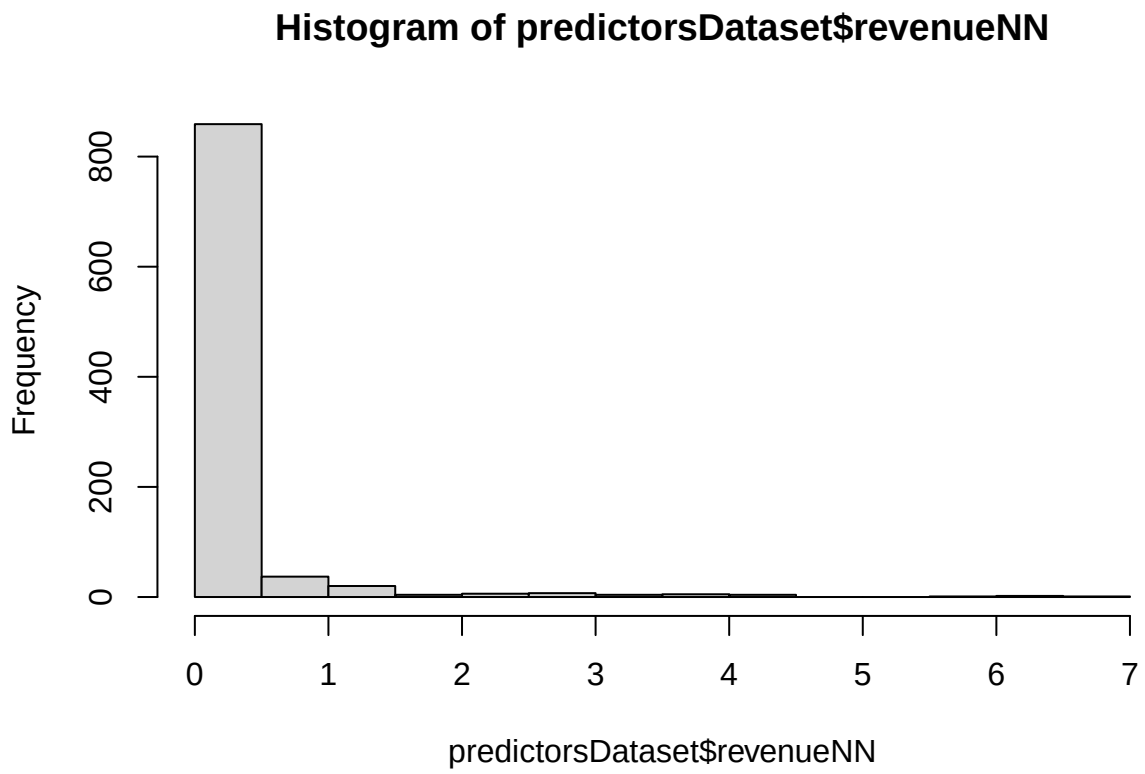
```
min(predictorsDataset $revenueNN)
```

```
## [1] 0
```

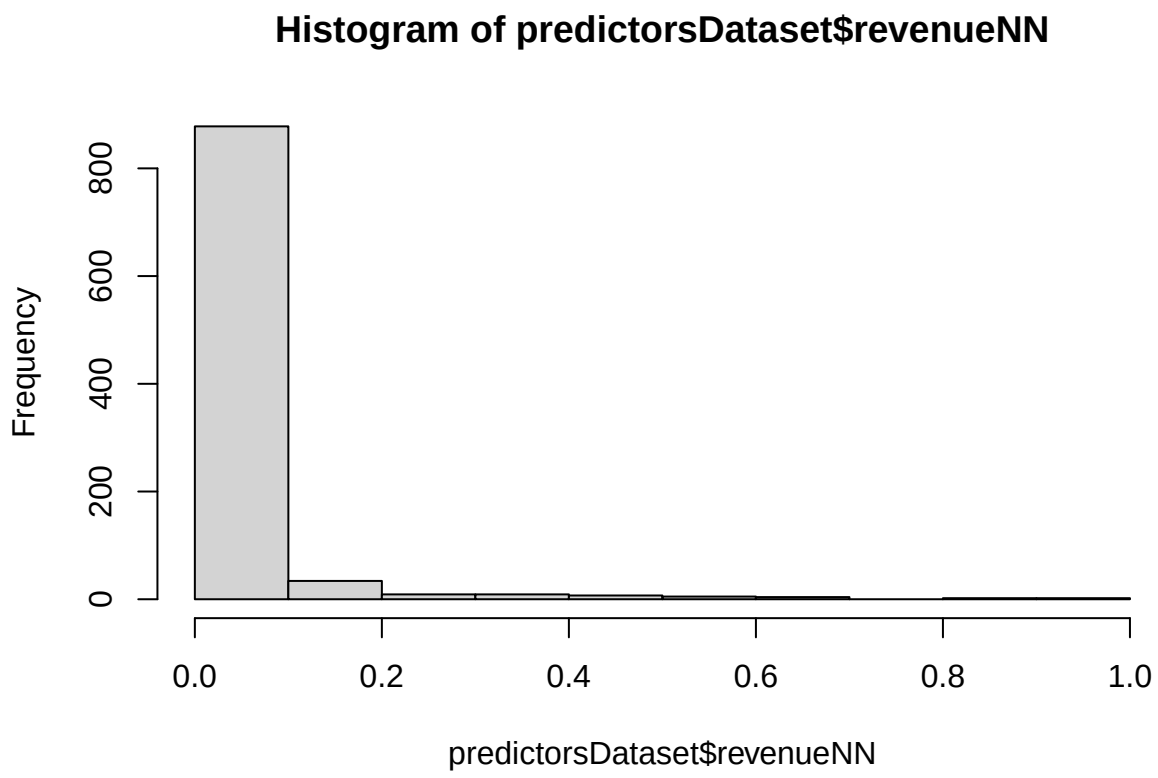
```
max((predictorsDataset $revenueNN))
```

```
## [1] 6.858171
```

```
hist(predictorsDataset $ revenueNN)
```



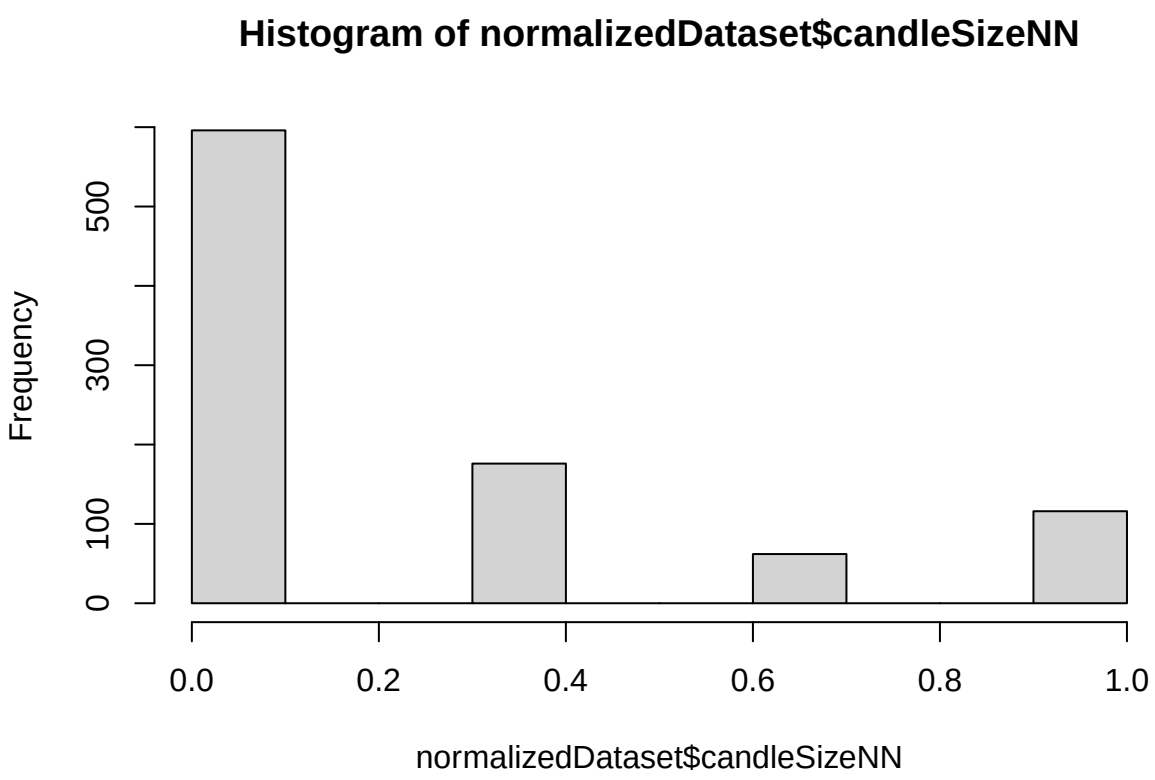

```
predictorsDataset$revenueNN <- (predictorsDataset$revenueNN - min(predictorsDataset$revenueNN)) /
  (max(predictorsDataset$revenueNN) - min(predictorsDataset$revenueNN))

# Show the results:
hist(predictorsDataset$revenueNN)
```



Making all fields numeric: as stated above, Neural Networks only take numeric values as input for the computations:

```
yesterdaysCandle<-data.frame(
  ifelse(predictorsDataset$yesterdaysCandle == "BULL","1","0"))
todaysCandle<-data.frame(
  ifelse(predictorsDataset$todaysCandle == "BULL","1","0"))
tomorrowsCandle <-data.frame(
  ifelse(predictorsDataset$tomorrowsCandle == "BULL","1","0"))

enhancedReversalCandle <- data.frame(
  ifelse(predictorsDataset$enhancedReversalCandle == "YES","1","0"))
priceRelativeToEMA20 <-data.frame(
  ifelse(predictorsDataset$priceRelativeToEMA20 == "ABOVE","1","0"))
priceRelativeToEMA50 <-data.frame(
  ifelse(predictorsDataset$priceRelativeToEMA50 == "ABOVE","1","0"))
EMA20_50Crossings <-data.frame(
  ifelse(predictorsDataset$EMA20_50Crossings == "ABOVE","1","0"))
revenueNN <-as.numeric(predictorsDataset$revenueNN )


## Create a dataframe with the predictors as columns:
normalizedDataset<-data.frame(todaysCandle,yesterdaysCandle,
                    enhancedReversalCandle,
                  priceRelativeToEMA20,
                  priceRelativeToEMA50,
```

```
                  EMA20_50Crossings ,
                    candleSizeNN,revenueNN,
                    tomorrowsCandle)

names(normalizedDataset)<-c("todaysCandle","yesterdaysCandle",
                    "enhancedReversalCandle",
                "priceRelativeToEMA20",
                "priceRelativeToEMA50",
                "EMA20_50Crossings" ,
                "candleSizeNN",
                "revenueNN",
                "tomorrowsCandle")


normalizedDataset$yesterdaysCandle <-
  as.numeric(normalizedDataset$yesterdaysCandle)
normalizedDataset$todaysCandle<- as.numeric(normalizedDataset$todaysCandle)


normalizedDataset$enhancedReversalCandle <-
  as.numeric(normalizedDataset$enhancedReversalCandle)
normalizedDataset$priceRelativeToEMA20 <-
  as.numeric(normalizedDataset$priceRelativeToEMA20)
normalizedDataset$priceRelativeToEMA50 <-
  as.numeric(normalizedDataset$priceRelativeToEMA50)
normalizedDataset$EMA20_50Crossings <-
  as.numeric(normalizedDataset$EMA20_50Crossings )
normalizedDataset$revenueNN <-as.numeric(normalizedDataset$revenueNN )

# Show the results:
hist(normalizedDataset $ candleSizeNN)
```



Factoring of the Output Dependent Variable: now we transform the dependent variable to a factor:

```
normalizedDataset$tomorrowsCandle <- factor(normalizedDataset$tomorrowsCandle,
                                levels = c(0,1),
                                labels = c(0,1))
```

Next, we proceed to Normalizing & Scaling all the Predictors. As stated before, we must normalize the variables in order to avoid some predictors to weight more than others. The scaling of data is essential because otherwise a variable may have a bigger impact on the prediction, only because of its scale. Using non-scaled data may produce erroneous results.

Actually we want each variable to contribute equally to the analysis. To standardize the rest of the variables, we use the "scale" function. We do not scale the output variable, i.e. the dependent variable "tomorrowsCandle" : so we take only 8 independent variables to scale:

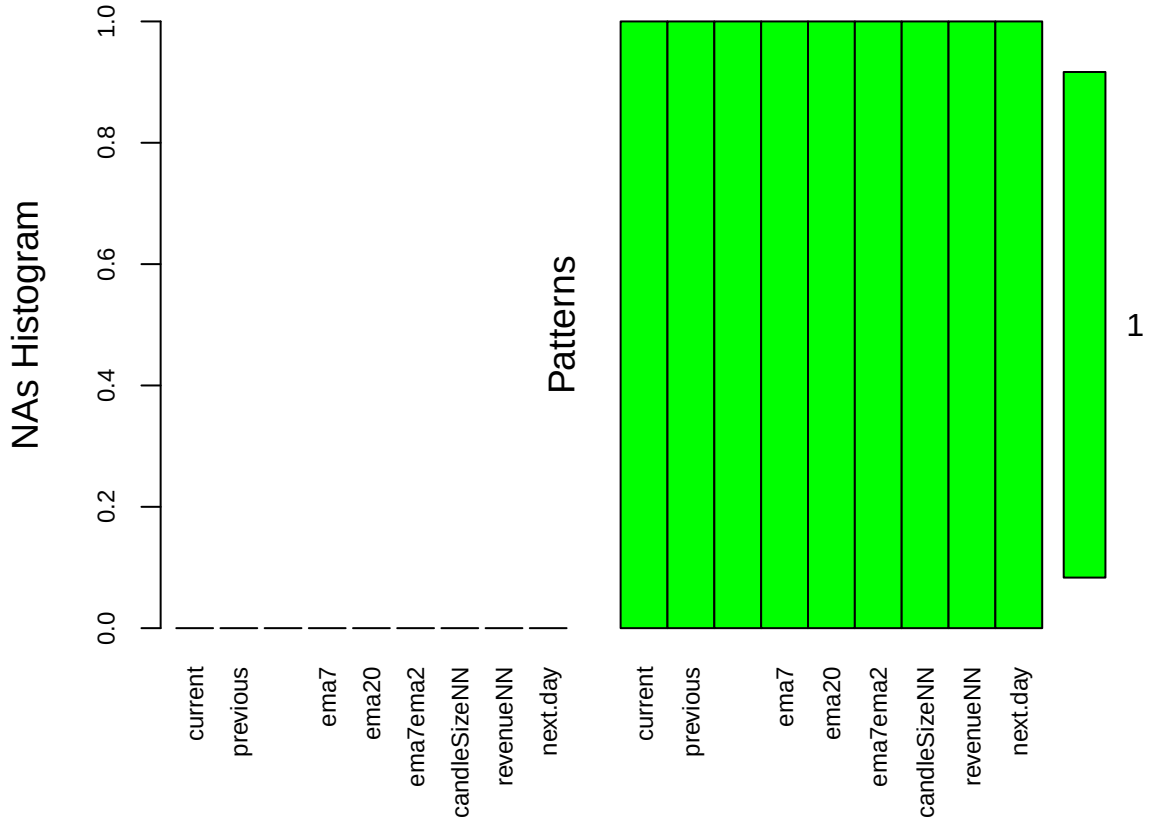
```
i = 1
for (i in 1:(length(normalizedDataset) - 2)) {
    temp <- scale(normalizedDataset[,i])
    apply(temp[,1,drop=F],1, function(x)sum(is.na(x))) -> NAs
    if (NAs[length(NAs)] > 0) {
     print(paste("There were ",NAs[length(NAs)]," NAs detected.") )

  } else {
    normalizedDataset[,i] <- temp[,1]
  }
}
```

Check for the maximal and minimal values, in order to centering the scale of the columns:

```
subds <- subset(normalizedDataset, select =  -c(tomorrowsCandle))
if (countOfDoji == 0) {
  subds <- subset(normalizedDataset, select =
                    -c(tomorrowsCandle,enhancedReversalCandle))
}
apply(subds, 2, max) -> maximums
apply(subds, 2, min) -> minimums

data <- as.data.frame(scale(subds,
                            center = minimums, scale = maximums - minimums))

# Check whether the "enhancedReversalCandle" predictor is relevant:
if (countOfDoji == 0) {
  data$enhancedReversalCandle <- normalizedDataset$enhancedReversalCandle
}
data$tomorrowsCandle <- normalizedDataset$tomorrowsCandle
normalizedDataset <- data
```

As the final preparations before the Neural Network computations, we check the data to verify whether or not there are NAs inside the predictors:

```
# Checking for the behavior of NAs in the predictors:

aggr(normalizedDataset,
     col = c("green","red"),  # graphic colors
     numbers = TRUE,          # proportions represented by numbers
     sortVars = TRUE, # sort the variables according to their count of NAs
     labels = c("current","previous",
                "enhancedReversalCandle","ema7","ema20","ema7ema2" ,
                "candleSizeNN","revenueNN",
                "next.day"),
     cex.axis = 0.75, # the width of the bars
     gap = 1, # distance between the graphs
     ylab = c("NAs Histogram","Patterns")) +
     neuralTheme
```




```
##
##  Variables sorted by number of missings:
##                 Variable Count
##                  current     0
##                 previous     0
##  enhancedReversalCandle     0
##                     ema7     0
##                    ema20     0
##                 ema7ema2     0
##              candleSizeNN     0
##                 revenueNN     0
##                 next.day     0

## NULL
```

On the "Patterns" part of the graph, it can be seen that all the predictors reach the maximum of "1.0", as expected. From the graph we can conclude that all the variables are complete, that means, there are not NAs at all.

Results Data exploration:

```
table(normalizedDataset$todaysCandle)
```

```
##
##   0   1
## 444 506
```

```
table(normalizedDataset$yesterdaysCandle)
```

```
##
##   0   1
## 444 506
```

```
table(normalizedDataset$enhancedReversalCandle)
```

```
##
##   0   1
## 945   5
```

```
table(normalizedDataset$priceRelativeToEMA20)
```

```
##
##   0   1
## 262 688
```

```
table(normalizedDataset$priceRelativeToEMA50)
```

```
##
##   0   1
## 194 756
```

```
table(normalizedDataset$EMA20_50Crossings)
```

```
##
##   0   1
## 174 776
```

```
table(normalizedDataset$candleSizeNN)
```

```
##
##                   0 0.333333333333333 0.666666666666667                   1
##                 596                 176                  62                 116
```

3.2.3.6 Data Partition of the dataset

In order to divide the dataset in two sets, Train set and Test set, we cannot use the "Sample" function, because we need to ensure a continuity of the stream of prices from a day to the next. An example of the utilization of the "Sample()" function, would be the following:

```
sample(1:nrow(dsData), round(0.75 * nrow(dsData))) -> index
```

What we'll do instead, is to partition the data in Train and Test, but keeping the dates continuity stream, that means, the logic continuation between successive candles. We'll take 75% of the dataset for creating the Train data frame, from the initial date until some date approximately at 75% from the beginning, and take the rest of the continuous stream of prices, until the end:

```
trainRange<-1:200
testRange<-201:300

dfNN1 <- normalizedDataset[trainRange,]
testNN1<-normalizedDataset[testRange,]
```

3.3 Neural Networks Modeling Section

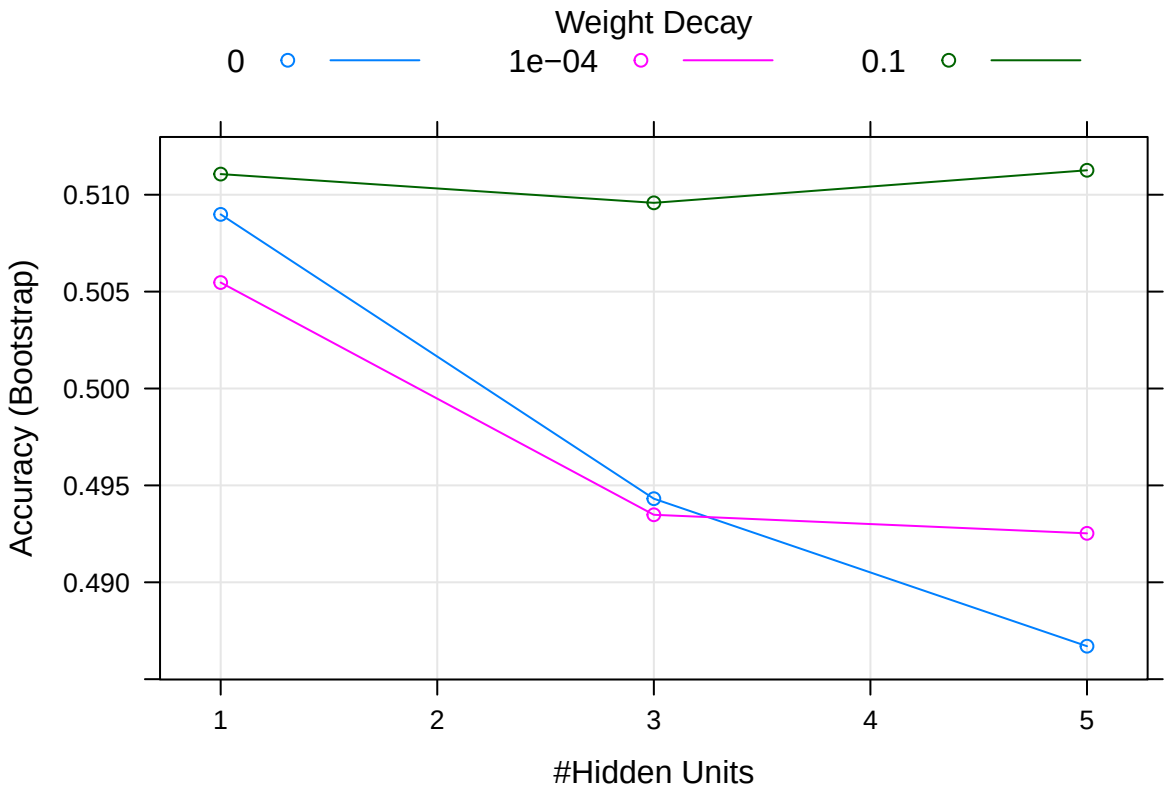
In this section we develop two Models in order to predict Stocks Market prices for the Apple data obtained from Yahoo. The first one using the Caret package, and the second one using the Neuralnet package. Then, we'll decide which one gives the best results, and choose that one to proceed with our research.

After the construction of the Models, the RMSE will be computed, on the validation dataset. This method of validation, the training-test split method, is in fact the simplest form of cross validation, known as holdout method.
A limitation of the holdout method is the variance of performance evaluation metrics, in our case, the RMSE, which can be highly based on the data selected to create the training and the test set. This dependence of performance on test-training split, reduces the variance of performance metrics, and can be considered a bias. For that reason, in the following section, a more elaborated way of validation will be performed on the Models.

3.3.1 Neural Network Caret Model

Training the Caret Neural Networks function:

```
set.seed(42)
nncaret <- caret::train(tomorrowsCandle~.,
                  dfNN1,
                  method = "nnet",
                  trace = FALSE)

# Show the analysis of the calculations:
plot(nncaret)
```



From the plot we can see that by using one neurons layer the algorithm obtains an accuracy of ~51%, while if using 5 layers, the accuracy descends to 49%

```
str(testNN1)
```

```
## 'data.frame':    100 obs. of  9 variables:
##  $ todaysCandle          : num  0 1 0 1 1 0 0 1 1 1 ...
##  $ yesterdaysCandle      : num  1 0 1 0 1 1 0 0 1 1 ...
##  $ enhancedReversalCandle: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ priceRelativeToEMA20  : num  0 1 1 1 1 1 1 1 1 1 ...
##  $ priceRelativeToEMA50  : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ EMA20_50Crossings     : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ candleSizeNN          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ revenueNN             : num  0.01842 0.00263 0.0043 0.01368 0 ...
##  $ tomorrowsCandle       : Factor w/ 2 levels "0","1": 2 1 2 2 1 1 2 2 2 1 ...
```

```
# Make the predictions:
caretPred <- predict(nncaret,testNN1)

confusionmatrix.CARET <- table(
  prediction= caretPred,   actual=testNN1$tomorrowsCandle)
print(confusionmatrix.CARET)
```

```
##           actual
## prediction  0  1
```

```
##           0 28 24
##           1 22 26
```

```
# Calculating misclassification:
misc <- 1 - sum(diag(confusionmatrix.CARET)) / sum(confusionmatrix.CARET)
misc
```

```
## [1] 0.46
```

```
print(paste("Caret Misclassification percentage = ",(( misc) * 100),"%") )
```

```
## [1] "Caret Misclassification percentage =  46 %"
```

```
#calculating accuracy:
lr.error <- mean(testNN1$tomorrowsCandle != caretPred )

print(paste("Caret Neural Network Accuracy = ",(( 1 - lr.error) * 100),"%") )
```

```
## [1] "Caret Neural Network Accuracy =  54 %"
```

3.3.2 Neural Network Neuralnet Model

Now we perform the training of the Neuralnet Neural Networks function. The dependency between independent and dependent variables is not linear, therefore we use linear.output with value "false".

For being relevant, the Accuracy of the Model must be bigger than 50%. Here we apply the Neural Network algorithm, using the following parameters:

1. Hidden: this expresses the number of inner neurons inside the Neural Net: i.e. = c(3,5) tells to use a first hidden layer of 3 neurons, then a second hidden layer of 5 neurons. After extended exploration, we choose a net of #### 7 neurons.
2. Threshold: this means that the iterations of the model will stop if the error is smaller than 5% (threshold = 0.05)
3. Algorithm: Neural Networks can use several algorithms: a. Backprop: uses back-propagation; b. Rprop+/- : is resilient back-propagation using the sign (increment/decrement) of the error between iterations, thus with or without weight backtracking; Sag and Slr: use the modified globally convergent algorithm (grprop). We use "rprop+".
4. act.fct: is the function that is used for smoothing the result of the cross product of the covariate or neurons and the weights. We used 'logistic' instead of 'tanh'.
5. Lifesign: specifies how much the function will print during the calculation of the neural network. 'none', 'minimal' or 'full'.
6. Rep:represents the number of repetitions for the neural network's training. We choose to train the model 10 times.
7. Stepmax: tells the number of steps before stopping the neural network's training process.
8. linear.output: output neurons regulation: FALSE states that act.fct should be applied to the output neurons.

```
set.seed(42)


nn <- neuralnet(formula1, data = dfNN1,
                hidden = 4,
                threshold = 0.05,
                algorithm = "rprop+", # sag slr backprop rprop-
                act.fct = "logistic",
                lifesign = "none",
                rep = 10,
                stepmax = 100000,
                linear.output = F)
```

For more information about Neuralnet, type : ?neuralnet

We won't use the Formulas #2 and #3 because, while tested, the percentage of accuracy for the Neural Network Model descended.

Now we can see the "nn$result.matrix" results, in order to check the results obtained from the Neural Network training.

After checking the nn$result.matrix, could be noticed that the "rep" (repetition) producing the smallest error, was the number 9, therefore we choose that "rep" which had the smaller error:

In our case, it's the model 9 between 10 models, so we use it, sending the value "9" to the parameter "rep":

```
compTRAIN <- neuralnet::compute(nn,  within(dfNN1,rm(tomorrowsCandle)), rep = 9)

# For more information about the repetitions obtained, uncomment :
#   nn$result.matrix
```

To satisfy curiosity, we proceed to checking the model against the Training dataset. We also select a threshold that defines which results correspond to which kind of candles, bear or bull:

```
min(compTRAIN$net.result)
```

```
## [1] 0.05450897
```

```
max(compTRAIN$net.result)
```

```
## [1] 0.9554959
```

```
thresholdTRAIN <- mean(compTRAIN$net.result)
thresholdTRAIN <- 0.5
predTRAIN <- ifelse(compTRAIN$net.result >= thresholdTRAIN,"1","0")

# Computing the Confusion Matrix:

confusionmatrix.TRAIN <- table(
  prediction= predTRAIN[,2] , actual=dfNN1$tomorrowsCandle)
print(confusionmatrix.TRAIN)
```

```
##            actual
## prediction  0  1
##          0 71 41
##          1 31 57
```

```
# Calculating misclassification:
misc <- 1 - sum(diag(confusionmatrix.TRAIN)) / sum(confusionmatrix.TRAIN)
misc
```

```
## [1] 0.36
```

```
print(paste("Misclassification percentage = ",(( misc) * 100),"%") )
```

```
## [1] "Misclassification percentage =  36 %"
```

```
print(paste("Accuracy percentage = ", (( 1 - misc) * 100),"%") )
```

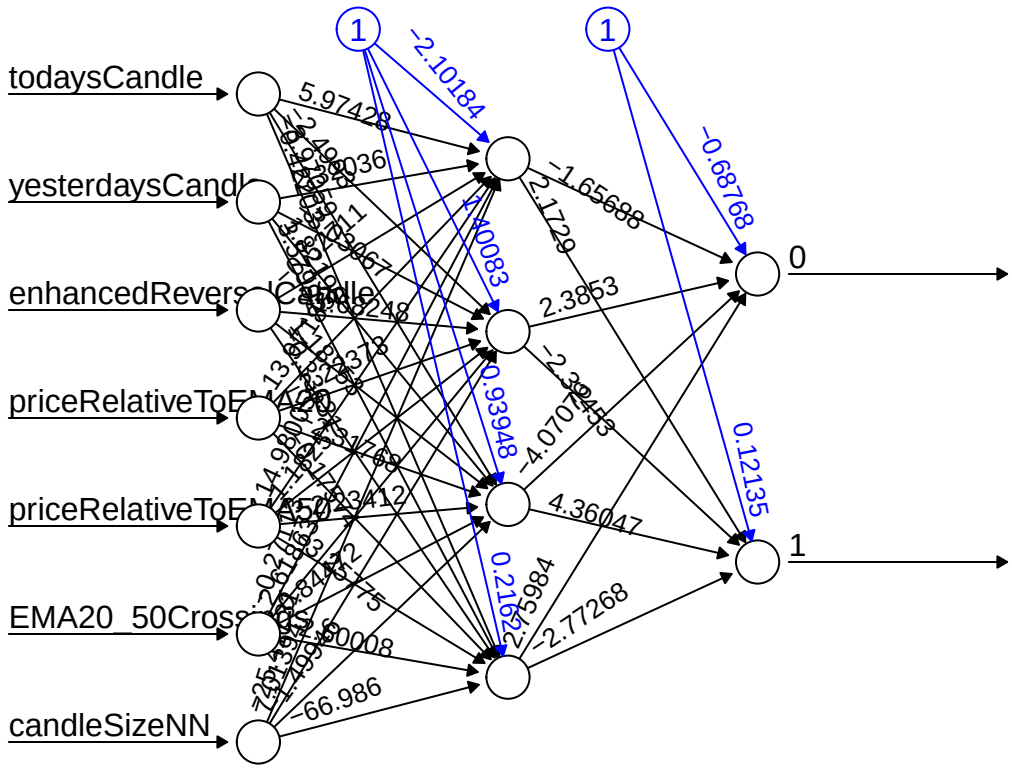
```
## [1] "Accuracy percentage =  64 %"
```

```
#calculating accuracy
lr.error <- mean(dfNN1$tomorrowsCandle != predTRAIN[,2] )

print(paste("TRAIN SET Accuracy = ",(( 1 - lr.error) * 100),"%") )
```

```
## [1] "TRAIN SET Accuracy =  64 %"
```

Show the Neural Network for the repetition number "9":

```
plot(nn, rep = 9)
```



3.3.3 Validation by training-test Split Holdout Method

Now we proceed to checking the model against the Test dataset. In this step, we perform Validation against the Test dataset :

```
comp <- neuralnet::compute(nn,within( testNN1,rm(tomorrowsCandle)), rep = 9)
```

Select a threshold that defines which results correspond to which kind of candles, bear or bull:

```
min(comp$net.result)
```

```
## [1] 0.05859895
```

```
max(comp$net.result)
```

```
## [1] 0.9250869
```

```
threshold <- mean(comp$net.result)
pred <- ifelse(comp$net.result >= threshold,"1","0")
```

To perceive the results of the Neural Network predictions against the Validation dataset, we show the confusion matrix:

```
confusionmatrix.lr <- table(prediction= pred[,2] ,
                            actual=testNN1$tomorrowsCandle)
print(confusionmatrix.lr)
```

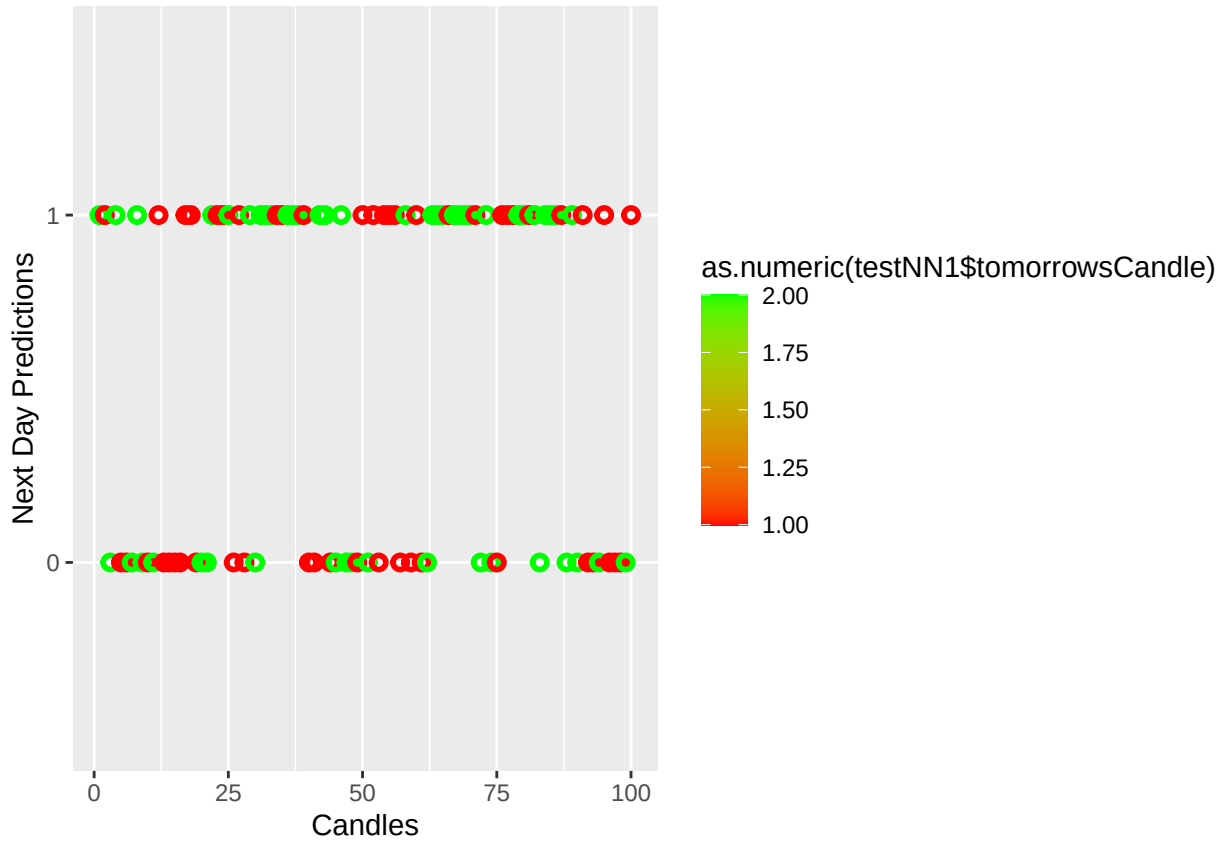
```
##            actual
## prediction  0  1
##          0 24 19
##          1 26 31
```

The confusion matrix means, 24 BEAR predictions were accurate, but 19 weren't. And 31 BULL predictions were right, and 26 were not.

Plotting the confusion matrix:

```
ind <- seq(1:100)
ggplot(testNN1,aes(ind, pred[,2] )) +
  geom_point(aes(color=as.numeric(testNN1$tomorrowsCandle)), alpha=10,
             shape = 1,stroke = 1.5) +
             scale_color_gradient(low = "red", high = "green") +
  theme_gray() +
  ylab("Next Day Predictions") +
  xlab("Candles")
```

```
## Warning: Use of 'testNN1$tomorrowsCandle' is discouraged. Use 'tomorrowsCandle'
## instead.
```



The red circles represent BEAR candles predictions, and ideally they should be situated below the 0.51 line. On the contrary, green circles are BULL candles predictions, and they should be bigger than 0.51, hence situated above the line. We can see from the graph, the quantity of misplaced circles. representing miscalculations.

Calculating the accuracy:

```
lr.error <- mean(testNN1$tomorrowsCandle != pred[,2] )


testNN1 %>% mutate(Prediction = pred[,2]) %>%
  mutate(Success = Prediction == testNN1$tomorrowsCandle)  %>%
  select(Prediction,tomorrowsCandle, Success, everything()) -> result

head(result,20)
```

```
##     Prediction tomorrowsCandle Success todaysCandle yesterdaysCandle
## 201          1               1    TRUE            0                1
## 202          1               0   FALSE            1                0
## 203          0               1   FALSE            0                1
## 204          1               1    TRUE            1                0
## 205          0               0    TRUE            1                1
## 206          0               0    TRUE            0                1
## 207          0               1   FALSE            0                0
## 208          1               1    TRUE            1                0
## 209          0               1   FALSE            1                1
## 210          0               0    TRUE            1                1
## 211          0               1   FALSE            0                1
## 212          1               0   FALSE            1                0
## 213          0               0    TRUE            0                1
## 214          0               0    TRUE            0                0
## 215          0               0    TRUE            0                0
## 216          0               0    TRUE            0                0
## 217          1               0   FALSE            0                0
## 218          1               0   FALSE            0                0
## 219          0               0    TRUE            0                0
## 220          0               1   FALSE            0                0
##     enhancedReversalCandle priceRelativeToEMA20 priceRelativeToEMA50
## 201                      0                    0                    1
## 202                      0                    1                    1
## 203                      0                    1                    1
## 204                      0                    1                    1
## 205                      0                    1                    1
## 206                      0                    1                    1
## 207                      0                    1                    1
## 208                      0                    1                    1
## 209                      0                    1                    1
## 210                      0                    1                    1
## 211                      0                    1                    1
## 212                      0                    1                    1
## 213                      0                    1                    1
## 214                      0                    1                    1
## 215                      0                    1                    1
## 216                      0                    1                    1
```

```
## 217                     0                   0                   1
## 218                     0                   0                   0
## 219                     0                   0                   0
## 220                     0                   0                   0
##     EMA20_50Crossings candleSizeNN   revenueNN
## 201                 1    0.0000000 0.018416471
## 202                 1    0.0000000 0.002630970
## 203                 1    0.0000000 0.004297263
## 204                 1    0.0000000 0.013680824
## 205                 1    0.0000000 0.000000000
## 206                 1    0.0000000 0.001929387
## 207                 1    0.0000000 0.009909748
## 208                 1    0.0000000 0.006051184
## 209                 1    0.0000000 0.007980536
## 210                 1    0.0000000 0.014996221
## 211                 1    0.0000000 0.025870996
## 212                 1    0.3333333 0.000964676
## 213                 1    0.0000000 0.001315397
## 214                 1    0.0000000 0.002630970
## 215                 1    0.0000000 0.002280248
## 216                 1    0.0000000 0.026572438
## 217                 1    0.3333333 0.029817223
## 218                 1    0.3333333 0.004297228
## 219                 1    0.0000000 0.019293485
## 220                 1    0.0000000 0.012628486
```

```
mean(result$Success, na.rm = T) * 100 -> percentSuccess

paste("The percentage of accuracy for the Neural Network Model is:",
      percentSuccess,"%")
```

```
## [1] "The percentage of accuracy for the Neural Network Model is: 55 %"
```

This Model has an Accuracy of 55 %, that means that in 55% of the times, the prediction is correct.

Yet another way of computing accuracy, is by calculating misclassification:

```
misc <- 1 - sum(diag(confusionmatrix.lr)) / sum(confusionmatrix.lr)
misc
```

```
## [1] 0.45
```

```
print(paste("Misclassification percentage = ",(( misc) * 100),"%") )
```

```
## [1] "Misclassification percentage =  45 %"
```

```
print(paste("Accuracy percentage = ", (( 1 - misc) * 100),"%") )
```

```
## [1] "Accuracy percentage =  55 %"
```

```
#calculating accuracy
lr.error <- mean(testNN1$tomorrowsCandle != pred[,2] )

print(paste("Accuracy = ",(( 1 - lr.error) * 100),"%") )
```

```
## [1] "Accuracy =  55 %"
```

Next we proceed to the denormalization of the field revenueNN. # We need de-normalize this column, in order to calculate the real return # for the model:

```
min(testNN1$revenueNN)
```

```
## [1] 0
```

```
max((testNN1$revenueNN))
```

```
## [1] 0.07191221
```

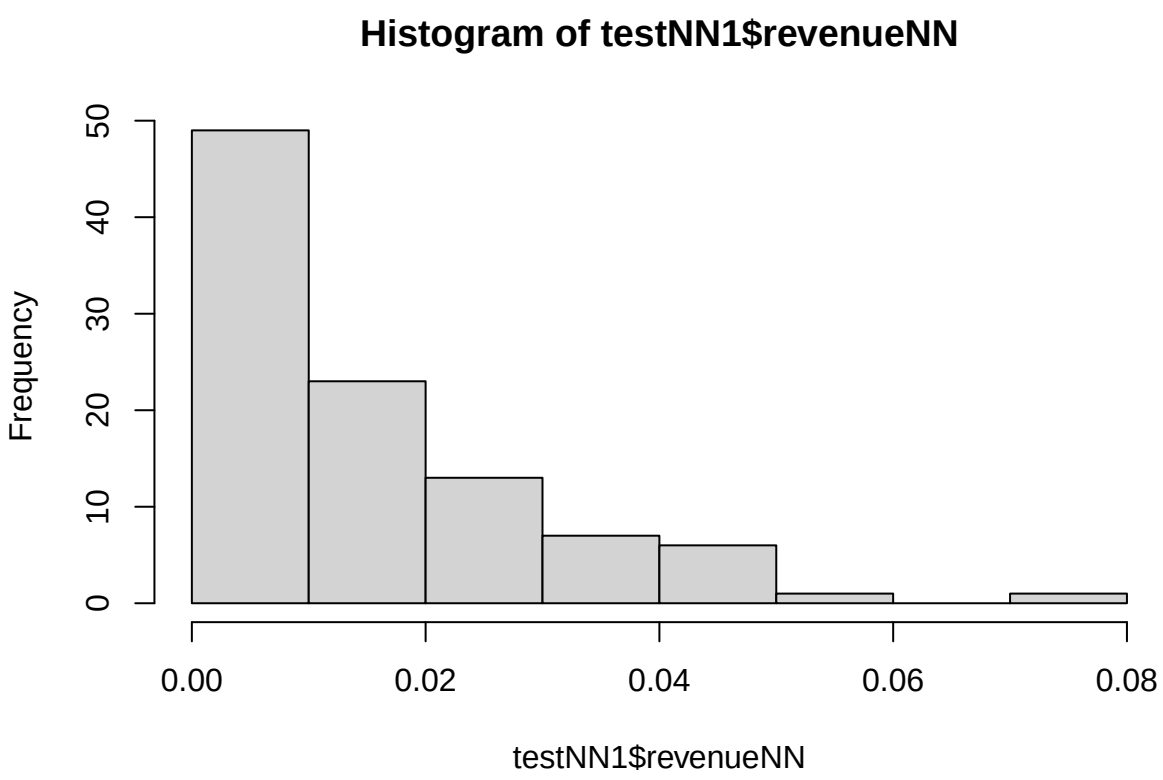
```
hist(testNN1$ revenueNN)


min(testNN1$revenueNN)
```

```
## [1] 0
```

```
max((testNN1$revenueNN))
```

```
## [1] 0.07191221
```

```
hist(testNN1$ revenueNN)
```




```
# Create a function to calculate the revenue:
predictedReturn <- function(df,pred){

  df$pred <- pred
  df$prediReturn <- ifelse(df$tomorrowsCandle != df$pred, -df$revenueNN,df$revenueNN)
  df$cumReturn <- cumsum(df$prediReturn )
  return (df)
}

# results in usd :
testReturns <- predictedReturn(testNN1, pred[,2])
tail(testReturns$cumReturn)
```

```
## [1] 0.2841635 0.2919423 0.2952129 0.3059971 0.2882298 0.2827492
```

```
head(testReturns)
```

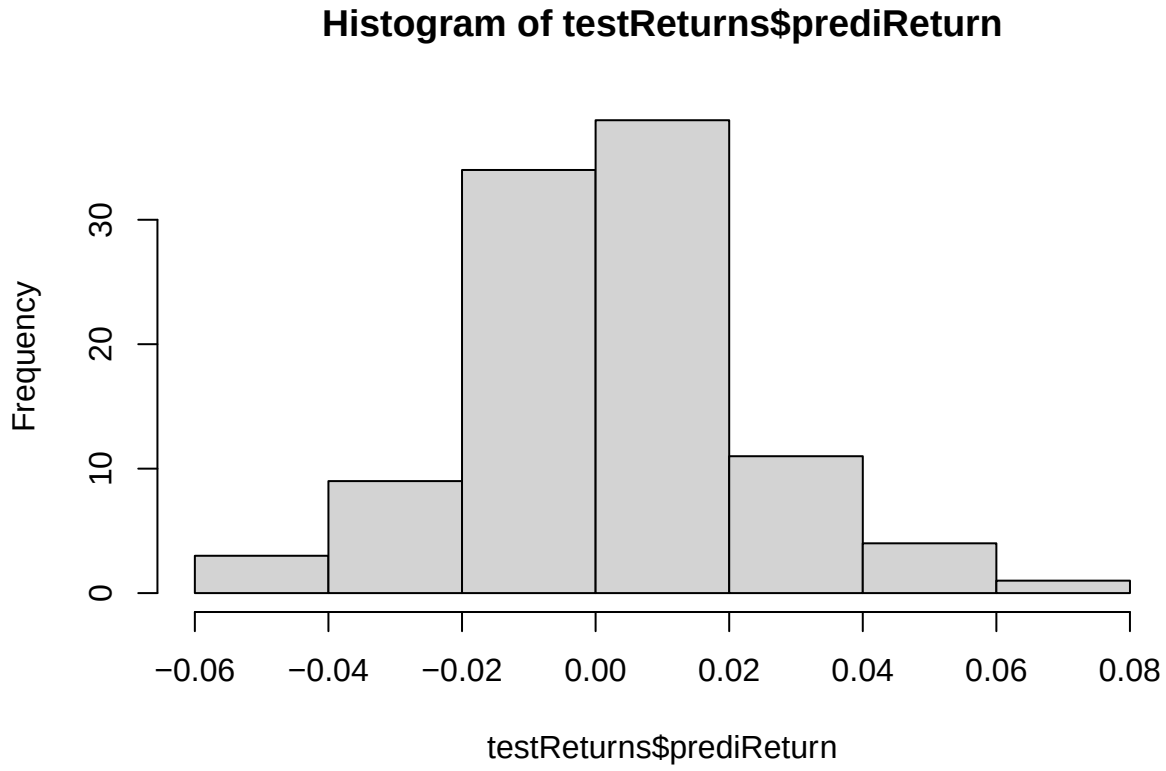
```
##     todaysCandle yesterdaysCandle enhancedReversalCandle priceRelativeToEMA20
## 201            0                1                       0                    0
## 202            1                0                       0                    1
## 203            0                1                       0                    1
## 204            1                0                       0                    1
## 205            1                1                       0                    1
## 206            0                1                       0                    1
##     priceRelativeToEMA50 EMA20_50Crossings candleSizeNN   revenueNN
```

```
## 201                    1                 1           0 0.018416471
## 202                    1                 1           0 0.002630970
## 203                    1                 1           0 0.004297263
## 204                    1                 1           0 0.013680824
## 205                    1                 1           0 0.000000000
## 206                    1                 1           0 0.001929387
##     tomorrowsCandle pred  prediRreturn  cumReturn
## 201               1    1  0.018416471 0.01841647
## 202               0    1 -0.002630970 0.01578550
## 203               1    0 -0.004297263 0.01148824
## 204               1    1  0.013680824 0.02516906
## 205               0    0  0.000000000 0.02516906
## 206               0    0  0.001929387 0.02709845
```

```
sum(testReturns$prediReturn)
```

```
## [1] 0.2827492
```

```
hist(testNN1 $ revenueNN)
hist(testReturns$prediReturn)
```



Next we calculate the Quadratic Error, of the difference between the actual next.day.return (from Test dataframe) and the predicted next day return:

```
qe <- sum((testReturns$prediReturn - testNN1 $ revenueNN)^2)/nrow(testNN1)
qe
```

```
## [1] 0.0005956818
```

Along with the Accuracy of the system, we calculate the RMSE:

```
str(testReturns$pred)
```

```
##  chr [1:100] "1" "1" "0" "1" "0" "0" "0" "1" "0" "0" "0" "1" "0" "0" "0" ...
```

```
# This column represents the "BULL" candle:
str(pred[,2])
```

```
##  Named chr [1:100] "1" "1" "0" "1" "0" "0" "0" "1" "0" "0" "0" "1" "0" "0" ...
##  - attr(*, "names")= chr [1:100] "201" "202" "203" "204" ...
```

```
str(as.numeric(testReturns$pred))
```

```
##  num [1:100] 1 1 0 1 0 0 0 1 0 0 ...
```

```
str(as.numeric(pred[,2]))
```

```
##  num [1:100] 1 1 0 1 0 0 0 1 0 0 ...
```

```
# Calculate RMSE:
RMSE.NN = (sum((as.numeric(pred[,2]) -
                  as.numeric(testReturns$tomorrowsCandle))^2)
                  / nrow(testReturns)) ^ 0.5

# Show the RMSE:
RMSE.NN
```

```
## [1] 1.144552
```

The RMSE of the Model is not impressive, because it is greater than 1. However, this model performs better than the previous models researched.

```
paste("The percentage of accuracy for the Neural Network Model is:",
      percentSuccess,"%")
```

```
## [1] "The percentage of accuracy for the Neural Network Model is: 55 %"
```

3.3.4 Cross validation by training-test K-fold Cross Validation Method

In the previous sections were built two Neural Network Models, the first one using the Caret library, and the second one using NeuralNet. After the construction of the Models, the RMSE was computed, using the validation dataset created with the training-test split Holdout method.

To overcome the limitations of the Holdout method, i.e. the data bias produced by selecting the training-validation datasets, its influence on the RMSE, and the variance of performance evaluation metrics caused, in the present section, a more elaborated way of validation will be performed on the Models.

The cross validation technique used in this section, will be the K-fold Cross Validation. This method can be viewed as a recurring Holdout method, in which most of the data is partitioned into k equal subsets, and each time a subset is assigned as training set, while others are used for validating the model.

This way, most of the data gets a chance to be both in training set and in the test set, and therefore, the dependence of the Models performance on test-training split sets is dramatically reduced, and consequently, also the variance of performance metrics (RMSE and MAE) are decreased. The thorough case of K-fold Cross Validation is when k is equal to the total number of data points, meaning that the predictive model is trained over all the data points except the last one data point, which will be the validation set. This method of leaving one data point as test set is known as "leave-one-out cross validation".

In order to compare the models performances on cross-validation, we add two functions for Model Evaluation Definitions: MAE & RMSE. First we design the Mean Absolute Error (MAE), which is the average of the squared error that is used as the loss function, the square of the difference between the predicted and actual target variables:

```
MeanAbsoluteError <- function(data = NULL, predictions = NULL){
  mean(abs((as.numeric(data$tomorrowsCandle)) - (as.numeric(predictions))))
}
```

Second, we define the RMSE - Root Mean Squared Error: RMSE is the square root of the Mean Squared Error (MSE)

```
RMSE <- function(data = NULL, predictions = NULL){
    (sum((as.numeric(predictions) -
                  as.numeric(data$tomorrowsCandle))^2)
                  / nrow(data)) ^ 0.5

}
```

3.3.4.1 K-fold Cross Validation Loop

Creating a dataframe that contains all RMSE-MAE-Accuracy results for the two Models (Caret and Neuralnet) iterations:

```
crossValidationResults <- data.frame(Model="Trading System expectations",
                      RMSE=  0.8000,
                      MeanAbsoluteError = 0.500,
                      Accuracy = 51)



# Initializing variables:
set.seed(42)
```

```
k = 7 # 7 no. of iterations according to dataset size
sizeTrain <- 200
sizeTest <- 100
dsSize <- nrow(normalizedDataset)
RMSE.NN = NULL
dfRMSECARET <- as.data.frame(1.300)
dfMAECARET <- as.data.frame(0.500)
dfACCCARET <- as.data.frame(50)
dfRMSENN <- as.data.frame(1.300)
dfMAENN <- as.data.frame(0.500)
dfACCNN <- as.data.frame(50)

# Dataframes used for computing averages after the cross-validation:
dfCaret = data.frame(x = numeric(), y = numeric(),z=numeric())
names(dfCaret)<-c("ValidationLoop","Accuracy","RMSE")
dfNeuralnet = data.frame(x = numeric(), y = numeric(),z=numeric())
names(dfNeuralnet)<-c("ValidationLoop","Accuracy","RMSE")

# Fit neural network models within cross-validation "for" loop:
for (i in 0:(k - 1)) {

        offset <- ifelse(i == 0,1,0)
        startTrain <- i * 100 + offset
        startTest <- startTrain + sizeTrain + 1
        endTrain <- startTrain + sizeTrain
        endTest <- startTest + sizeTest
        rangeTrain <- startTrain:endTrain
        rangeTest <- startTest:endTest

        tempTrain <- normalizedDataset[rangeTrain,]

        tempTest <- normalizedDataset[rangeTest,]



 #1 : using Caret Neural Network function:
set.seed(42)
nncaret <- caret::train(tomorrowsCandle~.,
                  tempTrain,
                  method = "nnet",
                  trace = FALSE)

# Make the predictions:
caretPred <- predict(nncaret,tempTest)

confusionmatrix.CARET <- table(
  prediction= caretPred,   actual=tempTest$tomorrowsCandle)

# Calculating misclassification:
misc <- 1 - sum(diag(confusionmatrix.CARET)) / sum(confusionmatrix.CARET)
#calculating accuracy
lr.error <- mean(tempTest$tomorrowsCandle != caretPred )
accuracy <- (( 1 - lr.error) * 100)
```

```
# Calculating the RMSE:
RMSE.NN   <-  RMSE(tempTest, caretPred)
MAE <- MeanAbsoluteError(tempTest, caretPred)

# Adding the Caret RMSE to the results data frame:
paste("Caret Model #",i + 1," validation: ",startTrain,":",
      startTrain + sizeTrain," // ",
      startTest,":",startTest + sizeTest," "
      ) -> modelCaret

crossValidationResults <- crossValidationResults %>% add_row(Model=modelCaret,
                                RMSE=RMSE.NN,
                                MeanAbsoluteError = MAE,
                                Accuracy  = accuracy
                                  )

dfRMSECARET[i + 1] <- RMSE.NN
dfMAECARET[i + 1] <- MAE
dfACCCARET[i + 1] <- accuracy

dfCaret <-rbind(dfCaret,
                    data.frame(ValidationLoop=i + 1,
                                Accuracy=accuracy,
                                RMSE=RMSE.NN))


 #2 : using NeuralNet Neural Network function:

set.seed(42)
nnCrossValidation <- neuralnet(formula1, data = tempTrain,
                hidden = 4,
                threshold = 0.05,
                algorithm = "rprop+", # sag slr backprop rprop-
                act.fct = "logistic",
                lifesign = "none",
                rep = 1,
                stepmax = 100000,
                linear.output = F)


# Make prediction on TEST dataset:
compCV <- neuralnet::compute(nnCrossValidation,
                           within( tempTest,rm(tomorrowsCandle)),
                           rep = 1)

threshold <- mean(compCV$net.result)
pred <- ifelse(compCV$net.result >= threshold,"1","0")

#confusion matrix
confusionmatrix.neuralnet <- table(prediction= pred[,2] ,
                            actual=tempTest$tomorrowsCandle)
```

```
#calculating accuracy:
# Using pred[,2] because that means the "1" ("BULL") candle:
lr.error <- mean(tempTest$tomorrowsCandle != pred[,2] )
accuracy <- (( 1 - lr.error) * 100)


tempTest %>% mutate(Prediction = pred[,2]) %>%
  mutate(Success = Prediction == tempTest$tomorrowsCandle)  %>%
  select(Prediction,tomorrowsCandle, Success, everything()) -> res

mean(res$Success, na.rm = T) * 100 -> percentSuccess


# Calculate RMSE:
RMSE.NN   <-  RMSE(tempTest, pred[,2])
MAE <- MeanAbsoluteError(tempTest, pred[,2])

# Adding the Caret RMSE to the results data frame:
paste("Neuralnet Model #",i + 1," validation: ",startTrain,":",
      startTrain + sizeTrain," // ",
      startTest,":",startTest + sizeTest," "
      ) -> modelNN

crossValidationResults <- crossValidationResults %>% add_row(Model=modelNN,
                                RMSE=RMSE.NN,
                                MeanAbsoluteError = MAE,
                                Accuracy  = accuracy
                                  )

dfRMSENN[i + 1] <- RMSE.NN
dfMAENN[i + 1] <- MAE
dfACCNN[i + 1] <- accuracy


dfNeuralnet <-rbind(dfNeuralnet,
                    data.frame(ValidationLoop=i + 1,
                                Accuracy=accuracy,
                                RMSE=RMSE.NN))

} # END of the Cross-Validation For loop


mean(as.numeric( dfACCCARET)) / 100 -> caretMean
mean(as.numeric( dfACCNN)) / 100 -> neuralnetMean

# Adding the averages to the results data frame:
crossValidationResults <- crossValidationResults %>% add_row(
  Model="  Caret Model averages: RMSE - MAE - Accuracy",
                                RMSE=mean(as.numeric(dfRMSECARET)),
                                MeanAbsoluteError = mean(as.numeric(dfMAECARET)),
                                Accuracy  = mean(as.numeric( dfACCCARET))
                                  )
```

```
# Adding the RMSE to the results data frame:
crossValidationResults <- crossValidationResults %>% add_row(
  Model="  NeuralNet Model averages: RMSE - MAE - Accuracy",
                                RMSE=mean(as.numeric(dfRMSENN)),
                                MeanAbsoluteError = mean(as.numeric(dfMAENN)),
                                Accuracy  = mean(as.numeric( dfACCNN))
                                  )
```

3.3.5 Accuracy & Performance Comparison - Apple Inc. Case

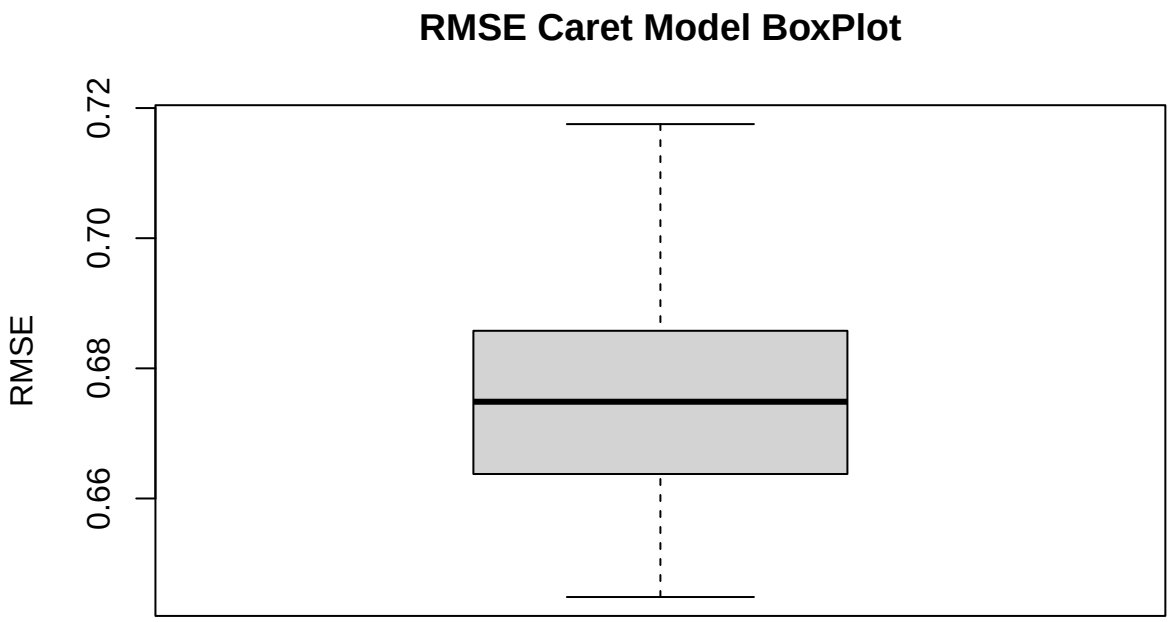
After running the two models on the cross-validation loop, and adding the results to a dataframe, we can check the results:

```
crossValidationResults
```

```
##                                                                 Model      RMSE
## 1                                    Trading System expectations 0.8000000
## 2        Caret Model # 1  validation:  1 : 201  //  202 : 302   0.6748671
## 3    Neuralnet Model # 1  validation:  1 : 201  //  202 : 302   1.1815412
## 4      Caret Model # 2  validation:  100 : 300  //  301 : 401   0.6893820
## 5  Neuralnet Model # 2  validation:  100 : 300  //  301 : 401   1.3570498
## 6      Caret Model # 3  validation:  200 : 400  //  401 : 501   0.7175315
## 7  Neuralnet Model # 3  validation:  200 : 400  //  401 : 501   1.0244535
## 8      Caret Model # 4  validation:  300 : 500  //  501 : 601   0.6674912
## 9  Neuralnet Model # 4  validation:  300 : 500  //  501 : 601   1.0624090
## 10     Caret Model # 5  validation:  400 : 600  //  601 : 701   0.6821631
## 11 Neuralnet Model # 5  validation:  400 : 600  //  601 : 701   1.1646612
## 12     Caret Model # 6  validation:  500 : 700  //  701 : 801   0.6448578
## 13 Neuralnet Model # 6  validation:  500 : 700  //  701 : 801   1.0388493
## 14     Caret Model # 7  validation:  600 : 800  //  801 : 901   0.6600330
## 15 Neuralnet Model # 7  validation:  600 : 800  //  801 : 901   1.1213500
## 16                  Caret Model averages: RMSE - MAE - Accuracy 0.6766180
## 17              NeuralNet Model averages: RMSE - MAE - Accuracy 1.1357591
##    MeanAbsoluteError Accuracy
## 1          0.5000000 51.00000
## 2          0.4554455 54.45545
## 3          0.9801980 56.43564
## 4          0.4752475 52.47525
## 5          1.1881188 53.46535
## 6          0.5148515 48.51485
## 7          0.7524752 45.54455
## 8          0.4455446 55.44554
## 9          0.8118812 49.50495
## 10         0.4653465 53.46535
## 11         0.9405941 52.47525
## 12         0.4158416 58.41584
## 13         0.8217822 56.43564
## 14         0.4356436 56.43564
## 15         0.9207921 58.41584
## 16         0.4582744 54.17256
## 17         0.9165488 53.18246
```

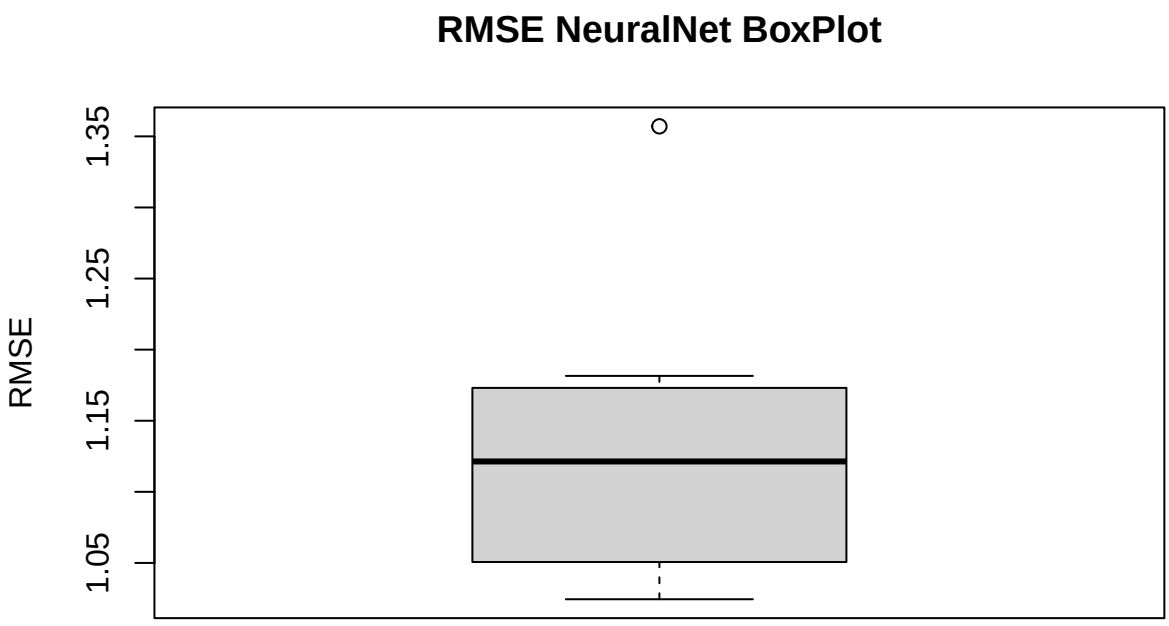
Also, we show a visualization of the Neural Network Models validation results :

```
boxplot((as.numeric(dfRMSECARET)), ylab = "RMSE", main = "RMSE Caret Model BoxPlot")
```



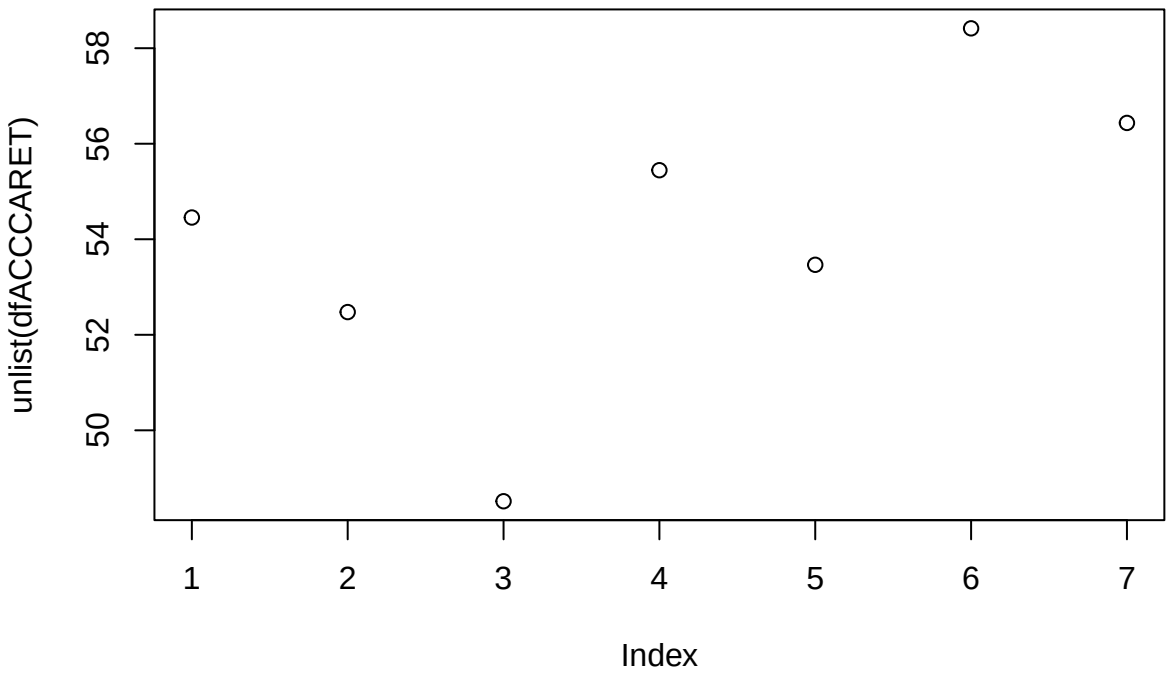
This boxplot, representing the results for the Caret Neural Network Model, shows that the median RMSE across the 7 samples when changing the training set and the validation set, is about 0.6700, and that the RMSE varies between approx. 0.6600 and 0.7100. Also, the boxplot expresses an approximately normal distribution of the results. The 50% of the results, spread between 0.6600 and 0.6900.

```
boxplot((as.numeric(dfRMSENN)), ylab = "RMSE", main = "RMSE NeuralNet BoxPlot")
```

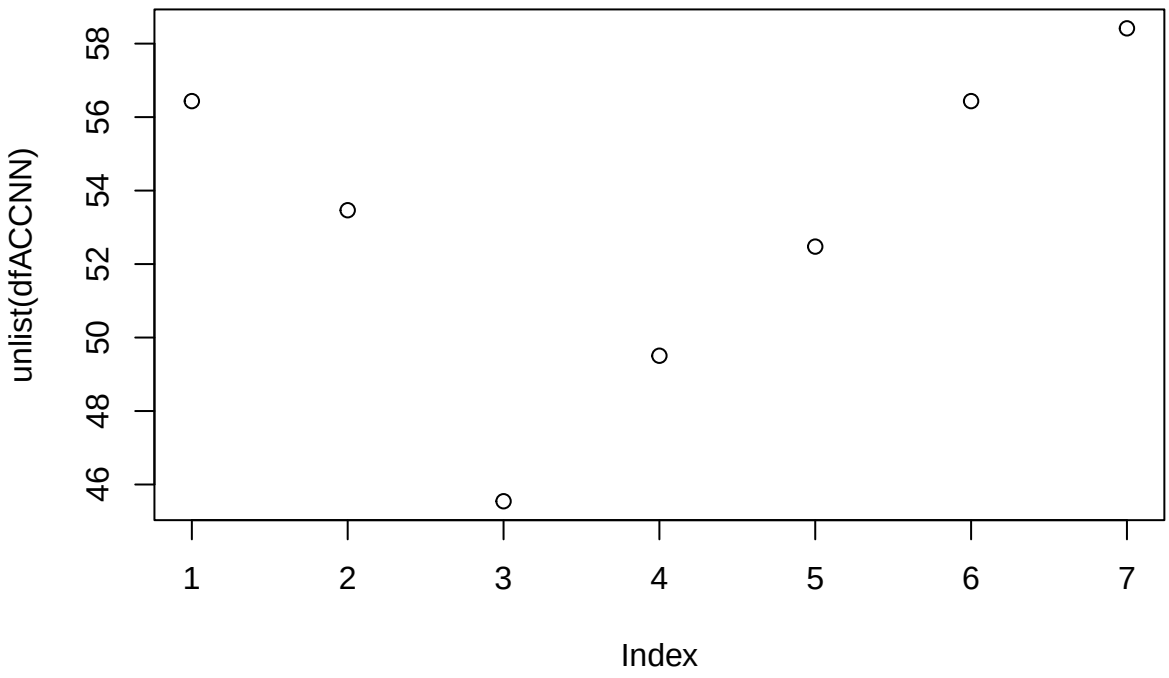


This graph refers to the NeuralNet Neural Network Model. The above boxplot shows that the median RMSE across the 7 samples when changing the training set and the validation set, is about 1.13000, and that the RMSE varies between approx. 1.0200 and 1.18000. The distribution is mostly on the region of the RMSE 1.1000.

```
plot( unlist(dfACCCARET))
```

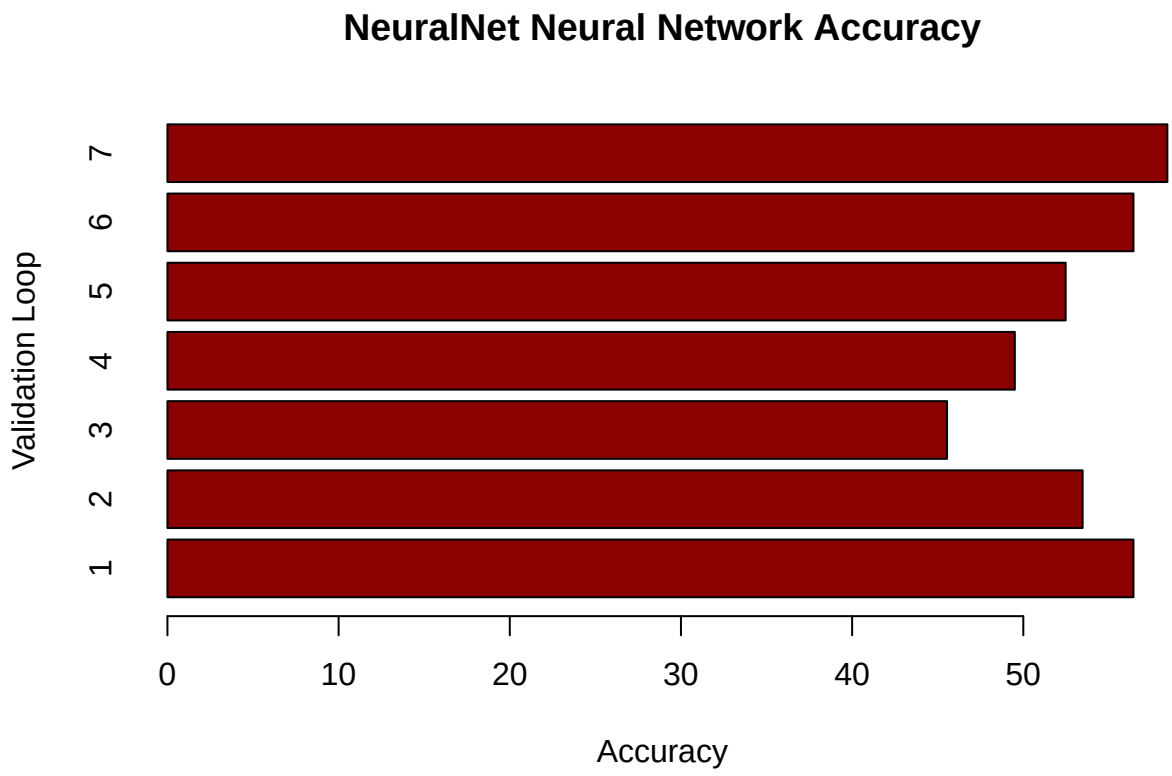



```
plot( unlist(dfACCNN))
```



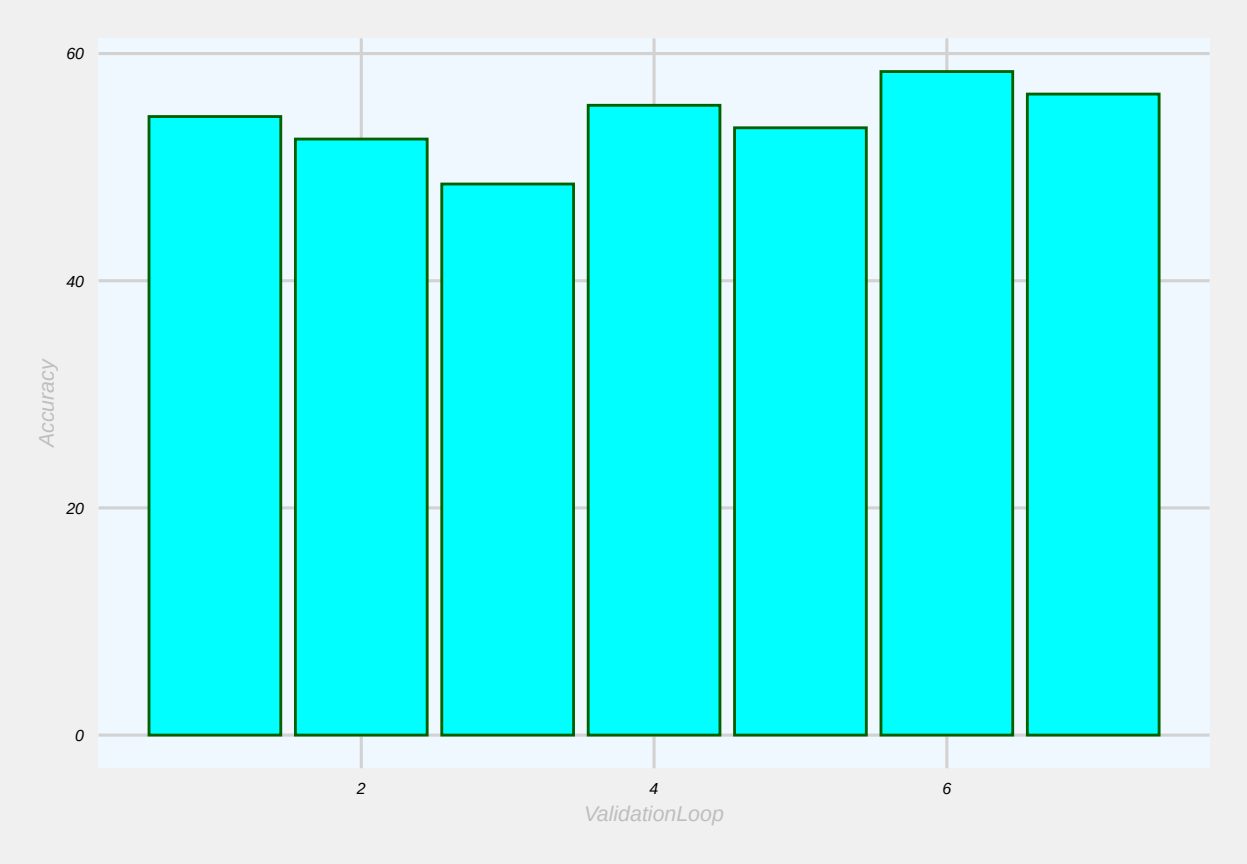
The accuracy results for the Caret Models are above 50% in most of the cases.

```
x <- as.matrix( dfACCNN)
barplot(x,
main = "NeuralNet Neural Network Accuracy",
xlab = "Accuracy",
ylab = "Validation Loop",
names.arg = c("1", "2", "3", "4", "5", "6", "7"),
col = "darkred",
horiz = TRUE)
```

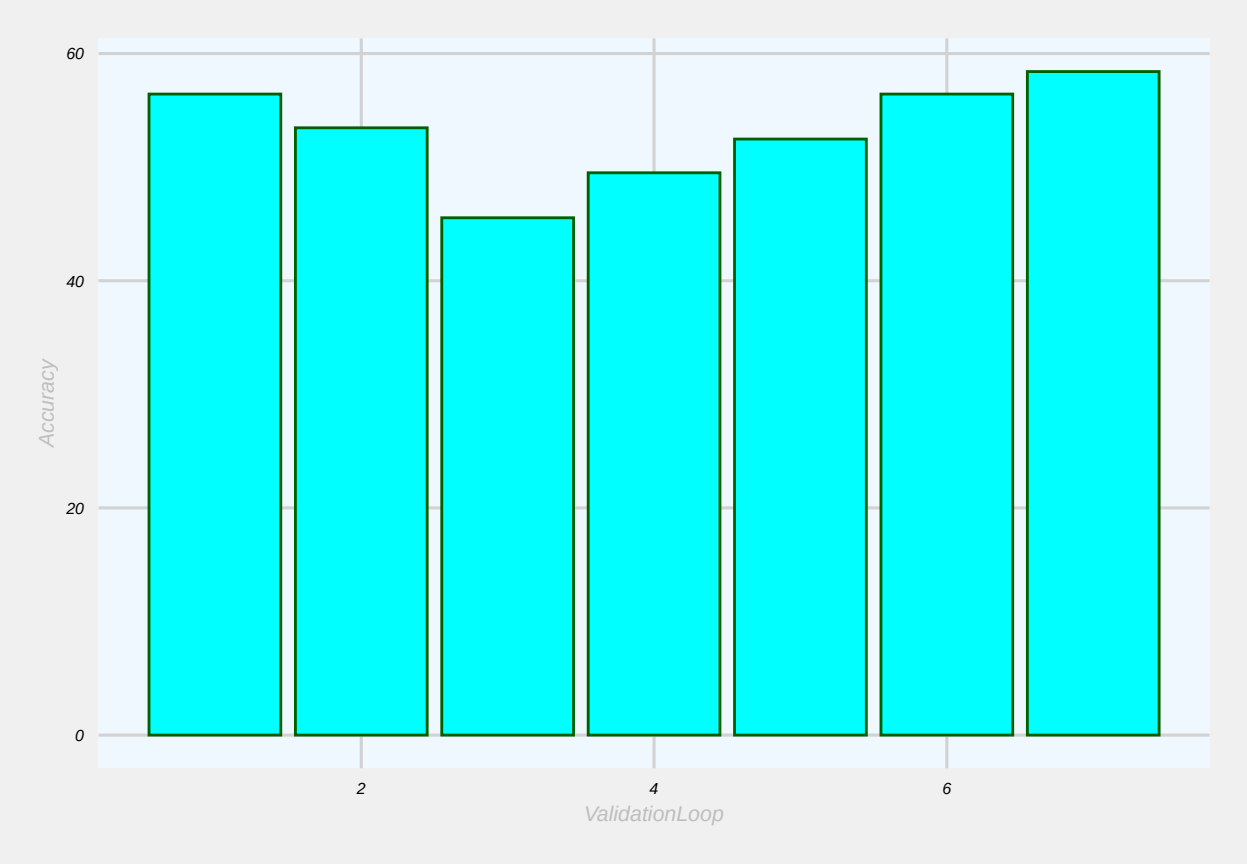


Comparing accuracy for the two models:

```
ggplot(dfCaret, aes(x = ValidationLoop, y = Accuracy      )) +  # bar plot
  geom_col(size = 0.5,fill="cyan",colour="#006000") +
  neuralTheme
```

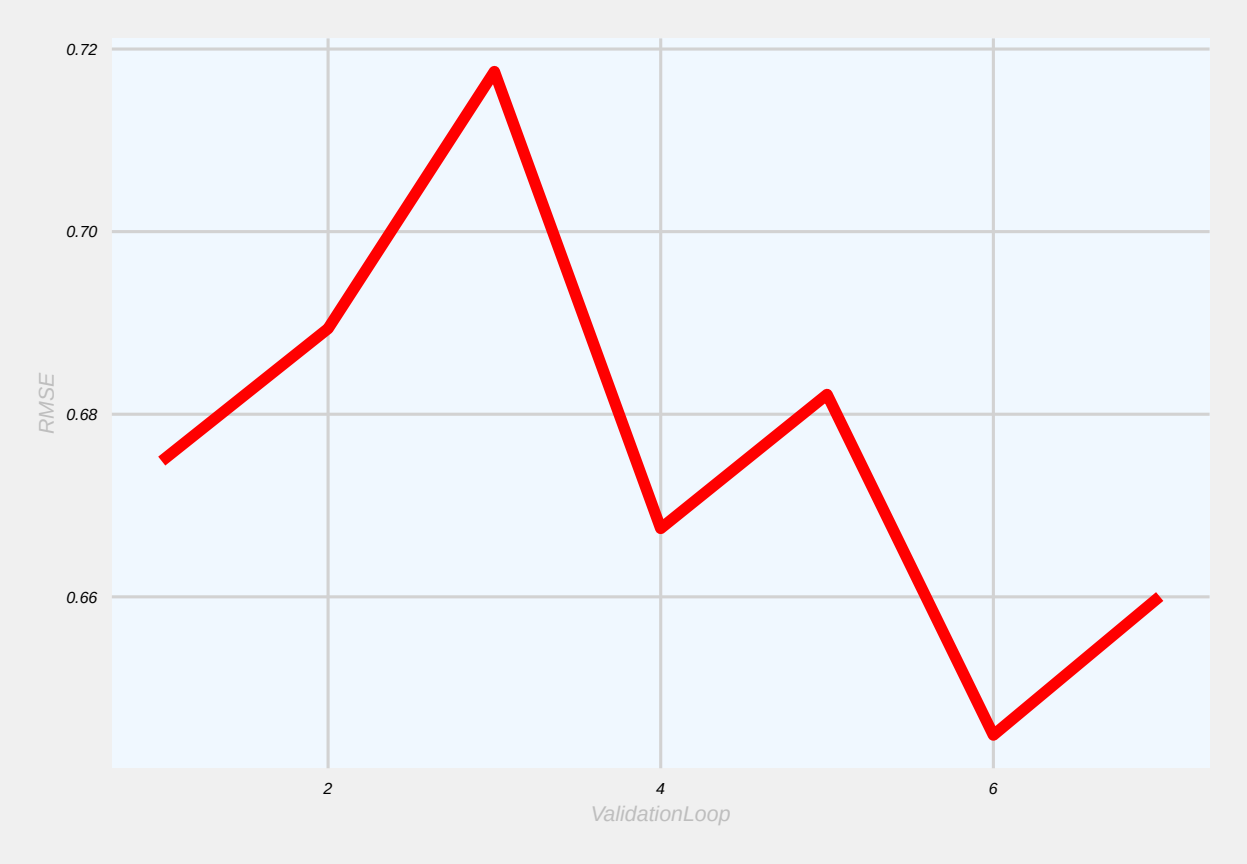



```
ggplot(dfNeuralnet, aes(x = ValidationLoop, y = Accuracy      )) +  # bar plot
  geom_col(size = 0.5,fill="cyan",colour="#006000")  +
  neuralTheme
```

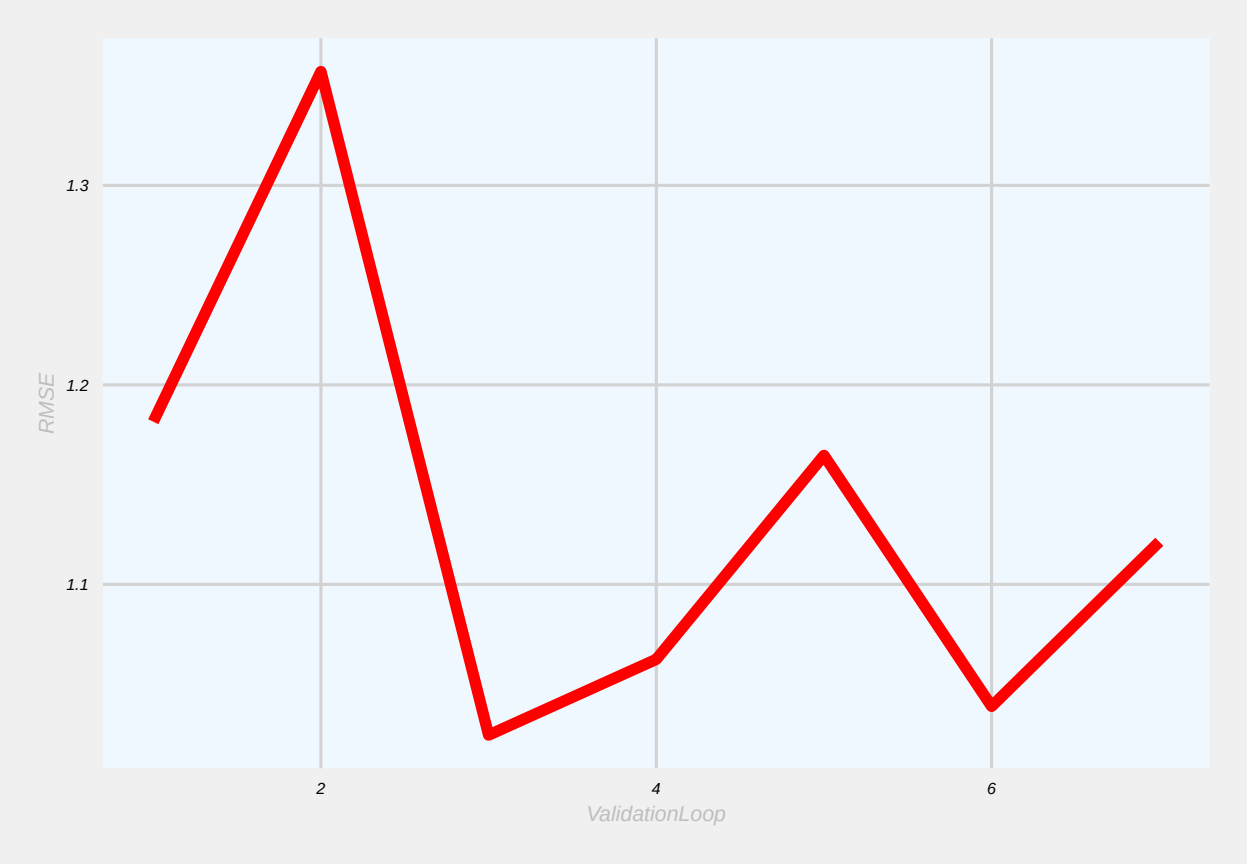


Comparing RMSE for the two models:

```
ggplot(dfCaret, aes(x = ValidationLoop, y = RMSE      )) +    # line plot
  geom_line(size = 2 , color="red", group = 1) +
  neuralTheme
```

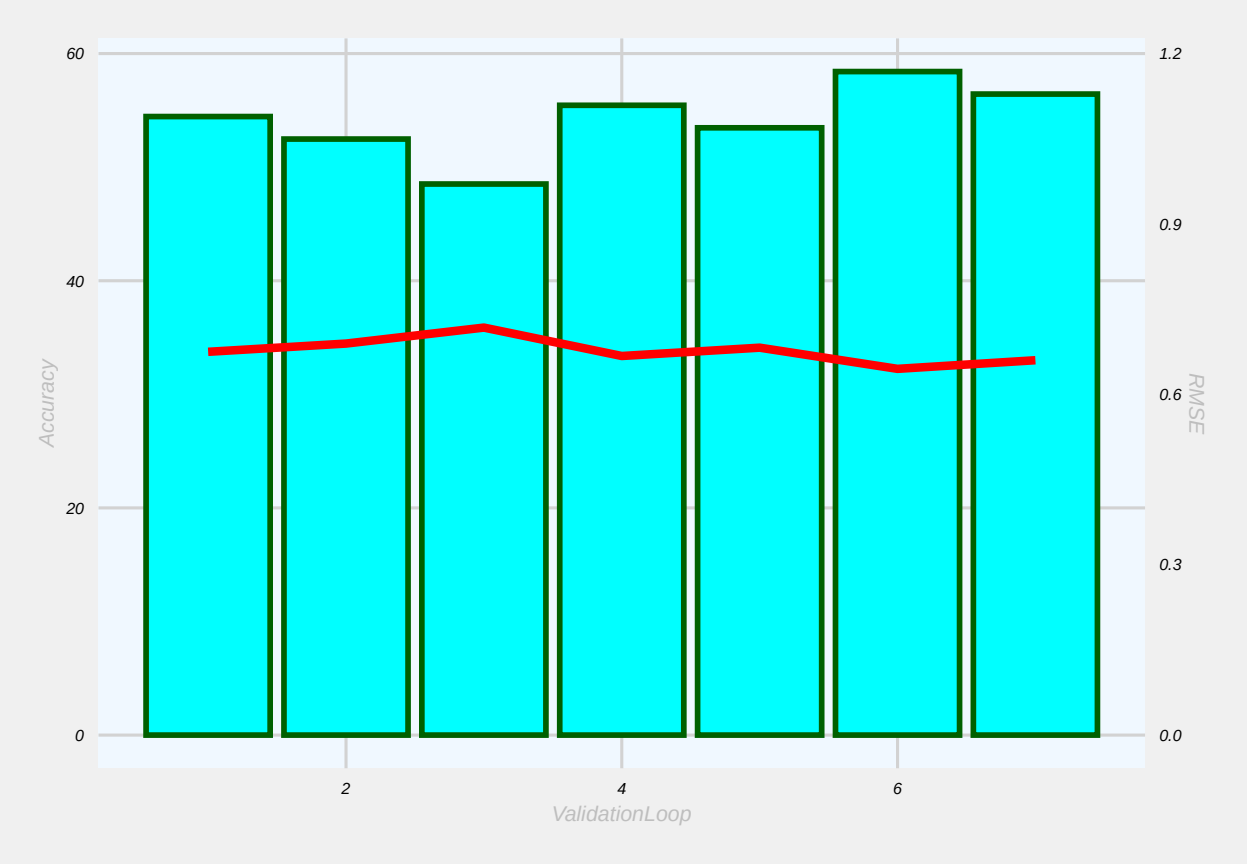



```
ggplot(dfNeuralnet, aes(x = ValidationLoop, y = RMSE      )) +    # line plot
  geom_line(size = 2 , color="red", group = 1) +
  neuralTheme
```



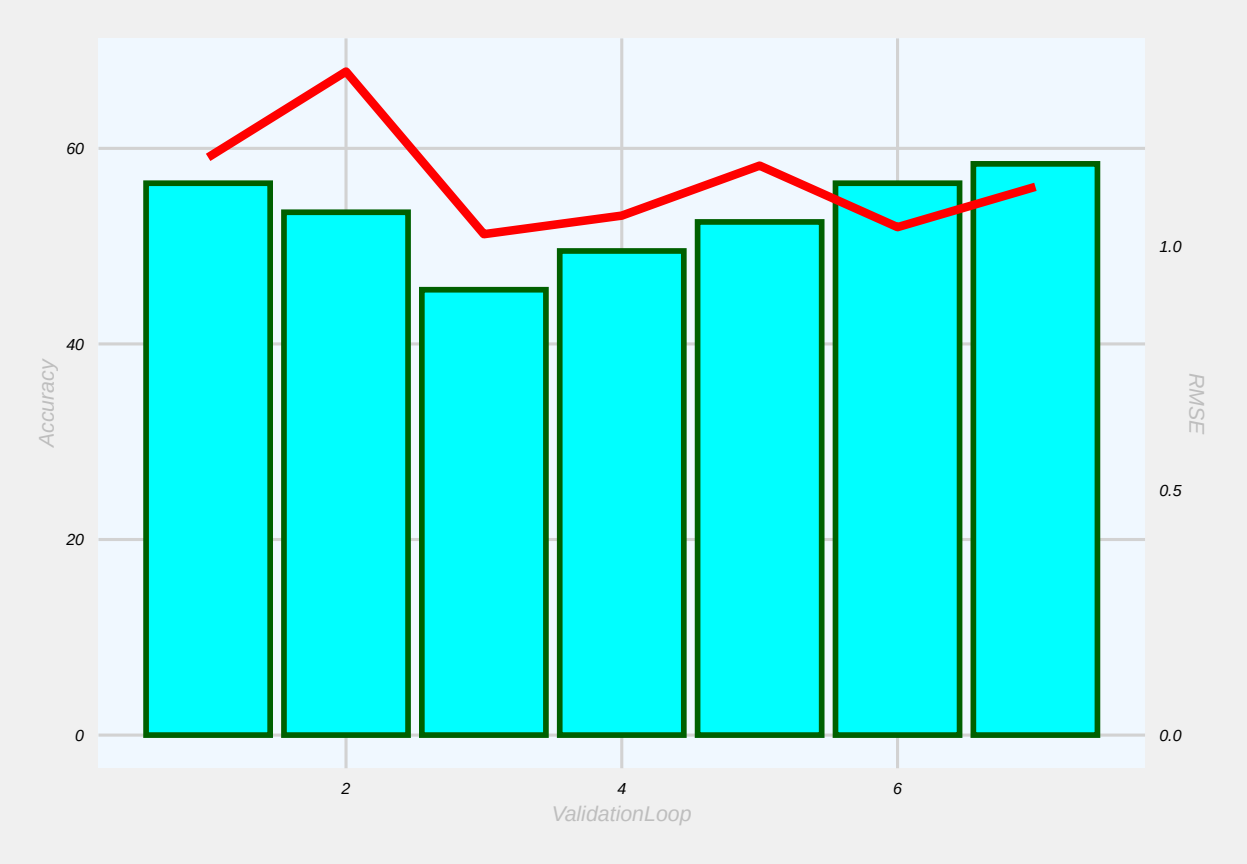
Comparing both accuracy and RMSE for the two models: Caret:

```
ggplot(dfCaret) +
  geom_col(aes(x = ValidationLoop, y = Accuracy), size = 1
           ,fill="cyan",colour="#006000") + # , color = "darkblue", fill = "white"
  geom_line(aes(x = ValidationLoop, y = 50*RMSE),
            size = 1.5 , color="red", group = 1) +
  scale_y_continuous(sec.axis = sec_axis(~./50, name = "RMSE")) +
  neuralTheme
```



Neuralnet:

```
ggplot(dfNeuralnet) +
  geom_col(aes(x = ValidationLoop, y = Accuracy), size = 1
           ,fill="cyan",colour="#006000") + # , color = "darkblue", fill = "white"
  geom_line(aes(x = ValidationLoop, y = 50*RMSE),
            size = 1.5 , color="red", group = 1) +
  scale_y_continuous(sec.axis = sec_axis(~./50, name = "RMSE")) +
  neuralTheme
```



From the comparison of the two graphs, can be drawn that the RMSEs for the NeuralNet Models are more unequally distributed than those of the Caret Models. There are also big differences between the accuracy of the models.

4. Results Section

This section presents the modeling results and discusses the models performances, that have been tested using the validation sets.

4.1 Final Evaluation

When starting the development of the present project, we fixed a minimal threshold in order to evaluate the Models performances. That minimum would be .51, i.e. 51 percent. If the performance of a Model surpasses that minimum, we could state that the specific model has attained the required goal. At this stage, we can compare the results obtained by the different Models, and draw conclusions.

4.2 Results Conclusions & Model Performance Discussion

While testing the different models against the corresponding Validation datasets, specifically when performing Cross validation by training-test K-fold Cross Validation Method, we got plenty of results measured by accuracy, RMSE and MAE standards.

The Neural Network Model rendered several results that were averaged in two mean global results, one for the Caret model and another for the Neuralnet model. These two averages will now be added to the Results

Table, and compared with the results obtained from the Naive Bayes model, the Support Vector Machine model, and the Random Forest model.

4.2.1 Final Results Table

The following are the final results for all of the models built, trained and validated against the validation datasets:

```
caretMean + totalAccuracy -> totalAccuracy
results <- results %>%
  add_row(Model="   Fifth Model : Neural Network Caret Model Accuracy",
                               Accuracy = caretMean
                               )


neuralnetMean + totalAccuracy -> totalAccuracy
results <- results %>%
  add_row(Model="   Sixth Model : Neural Network NeuralNet Model Accuracy",
                               Accuracy = neuralnetMean
                               )



# Calculating the mean accuracy and adding it to the Results table:
results <- results %>%
  add_row(Model="   Mean Accuracy",
                               Accuracy = totalAccuracy / 6
                               )

# Printing the accuracies:
results
```

```
##                                                    Model  Accuracy
## 1                          Stocks Trading System expectations 0.5100000
## 2    First Model : Support Vector Machine - Polynomial kernel 0.4900000
## 3      Second Model : Support Vector Machine - Sigmoid kernel 0.5900000
## 4                            Third Model : Naive Bayes Model 0.6100000
## 5                          Fourth Model : Random Forest Model 0.6200000
## 6           Fifth Model : Neural Network Caret Model Accuracy 0.5417256
## 7       Sixth Model : Neural Network NeuralNet Model Accuracy 0.5318246
## 8                                              Mean Accuracy 0.5639250
```

The above table shows the results obtained from the Models. Starting from the Stocks Trading System expectations, which were the size of 51 percent, or .51, we tested the models against the validation data, to find the performances done.

The first Model, Support Vector Machine, using the Polynomial kernel, rendered an accuracy of 49 percent, which is smaller than the required goals of 51 percent.

The second Model, Support Vector Machine using the Sigmoid kernel, enhanced the performance of SVM to a result of .59, i.e. 8 percent bigger than the Stocks Trading System expectations of .51.

The third Model was the Naive Bayes Model, and attained more that expected, 61 percent.

The fourth Model, the Random Forest Model, performed far better than the previous results, and also than the expectations.

Finally, two Neural Network models were developed and tested using the method called K-fold Cross Validation. These models, which we called Caret Model and NeuralNet Model, produced an Accuracy of 0.5417256 and 0.5318246 respectively, which are also better than the Stocks Trading System expectations.

The Neural Network Model, using the Caret software package, rendered an accuracy of more than 54%.

A Mean Accuracy of more than 56% was finally obtained.

5. Conclusions

This section presents a brief summary of the report, its limitations and future work to be done.

5.1 Brief Summary of the Report

In order to present you this Machine Learning Algorithms System applied to the Stocks Market, this paper includes several sections progressively displayed from the data analysis to the performances obtained.

At first, after data loading, it performs a thorough analysis of the data. This allows to produce insights that helped in the design of the Project, such as the different features to be turned into predictors, the techniques used for data cleaning and normalization, and the formulas to be used on the models.

Following the exploratory section, comes the Models building section, which tries six different models to reach a better Machine Learning System performance. These models started from two Support Vector Machines (SVM) models, which used two different kernels, the Polynomial kernel and the Sigmoid Kernel, and, passing through a Naive Bayes Model, they progressively became more complicated, with the utilization of a Random Forests Model, and finally, a Neural Network Model, developed using resilient back-propagation with weight backtracking.

After training the different models, it stand to reason that the two best models were the Random Forest Model, which reached an accuracy of 62%, and the Naive Bayes Model, with an accuracy of 61%.

However, the performances rendered by the Neural Network Models, which were solidly tested against several validation sets, were fairly good, better than the Stocks Trading System expectations. These models achieved the desired performance, with an RMSE and MAE pretty good, at least in the Caret Model average case, which produced an RMSE of 0.6766180 and a Mean Absolute Error (MAE) of 0.4582744.

The Neural Network Model, using the Caret software package, rendered an accuracy of more than 54%.

All Models put together, a Mean Accuracy of more than 56% was attained, which represents a good improvement over the project goals fixed at the beginning.

In short, can be stated that on a whole the Predictive Models yield a fairly good accuracy, better than the Stocks Trading System expectations.

5.2 Limitations

The Models developed and the Data Analysis performed, were somehow limited by the home laptop available to run this Machine Learning techniques, because of the required amount of memory needed.

Although the Stocks Trading System succeeded on achieving a good accuracy, the hardware limitations didn't allow for a very more thorough analysis, such as the one which could be performed on many years of daily stocks prices data. In addition to that, while developing the System, the need of doing Cross validation by training-test K-fold Method, applied over many more years, appeared many times. The completion of such a project was postponed because of the previously referred hardware limitations.

Another issue refers to the need of necessary research concerning the enormously diverse techniques regarding Technical Trading. The vast number of different methods and Technical Indicators, that could be applied to this research, limited definitely its scope, and open the need of new computations in order to render more appropriate new predictions.

5.3 Future Work

Despite the fairly good results obtained while measuring the Stocks Market Trading System accuracy, better results could be obtained by adding different features to the predictors, or by thorough research of their impact on the system performance.

While rendering the prices data graphs, we used some Technical Analysis Indicators, specifically Bollinger Bands and Moving Average Convergence Divergence (MACD), which for example could been also used as predictors. There are many more Trading Technical Analysis Indicators, such as the Stochastic Oscillator, which represents oversold markets and overbought markets, the Relative Strength Index (RSI), Fibonacci Retracements, the Ichimoku Cloud, which identifies support and resistance levels and also estimates price momentum, the Average Directional Index (ADX), which illustrates the strength of a price trend, and so on. Every one of these Indicators can eventually be used as predictors for the System, which in turn could produce more robust predictions.

By accessing stronger hardware, enhancements like doing Cross-validation by training-test K-fold Method, applied over many years, and of course, the training of the System on many years of daily stocks prices data, could be certainly added, leading to very more thorough analysis and to a more powerful Trading System, which eventually could be designed and developed.

5.4 References

Irizarry, Rafael. 2021. “Data Analysis and Prediction Algorithms with R” https://rafalab.github.io/dsbook/introduction-to-productivity-tools.html

Cortes, Corinna; Vapnik, Vladimir N. (1995). “Support-vector networks” . Machine Learning. 20 (3): 273–297

Boser, Bernhard E.; Guyon, Isabelle M.; Vapnik, Vladimir N. (1992). “A training algorithm for optimal margin classifiers”. Proceedings of the fifth annual workshop on Computational learning theory – COLT ’92. p. 144

Irizarry, Rafael. 2021. “Introduction to Data Science - Data Analysis and Prediction Algorithms with R”. Naive Bayes - 31.7.1

McCallum, Andrew. 2019. “Graphical Models, Lecture2: Bayesian Network Representation” .

Hastie, Trevor. 2001. “The elements of statistical learning : data mining, inference, and prediction : with 200 full-color illustrations”.

Irizarry, Rafael. 2021. “Introduction to Data Science - Data Analysis and Prediction Algorithms with R”. Random forests - 31.11

Ho TK. 1998. “The Random Subspace Method for Constructing Decision Forests” IEEE Transactions on Pattern Analysis and Machine Intelligence. 20

Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome. 2008. “The Elements of Statistical Learning”

Cybenko, G. .1989. “Approximations by superpositions of sigmoidal functions”, Mathematics of Control, Signals, and Systems, 2 (4), 303-314

Pinkus, Allan .1999. “Approximation theory of the MLP model in neural networks”. Acta Numerica. 8: 143–195.

Hornik, Kurt .1991. “Approximation Capabilities of Multilayer Feedforward Networks”, Neural Networks, 4(2), 251–257.

Hayes, Adam. 2021 . “Technical Analysis” . Investopedia.com

https://data-flair.training/blogs/pros-and-cons-of-r-programming-language/

https://techvidvan.com/tutorials/pros-and-cons-of-r/