WQD 7004 Programming for Data Science

Data Cleansing and Machine Learning algorithm application

Technical Analysis in Financial Data

Ng Kang Wei	WQD170068
Alireza Ghanbarzadeh	WQD170061
Khashayar Namsehchi	WQD170034
Kavish Punchoo	WQD170097

Contents

The data

The dataset comprised of 20 companies stock listed on New York Stock Exchange (NYSE). The 20 companies chosen here are all financial institutions. The dataset consisted of the weekly fluctuation in prices for each of the company stock since 2012 to 2017. Each of the stock data is stored in a CSV file. There are 20 financial institutions stock thus there are 20 CSV files. Each of the file contains 18 columns and 1260 rows.

The stock dataset just contains the stock code or symbol used by the companies in NYSE but not the company name. In order to get the company details for each of the financial institutions, there is another dataset that contains the details organized by the stock code. The companies' details are stored in 1 CSV file and it has 8 columns and 505 rows.

Purpose of dataset

The data can be used to analyze the past performance of the stocks and provide insights into the future performance of the stocks.

These are some of the analysis that can be done on the dataset to achieve the said purpose:

- Portfolio allocation
- Calculate risk and Sharpe ratio
- Calculate return based on prices
- Price prediction
- Technical indicators computation and analysis
- Price chart visualization
- Visualizing technical indicators such as moving averages
- Forecasting future returns using past data

Data Structure

This section explains the structure of the dataset. The stock data is a time series dataset that hold 5 years' worth of data from 30/11/2012 to 29/11/2017 in the form of weekly stock prices fluctuations.

The structure of the stock dataset is as follows:

Attribute	**Type**	**Explanation**
Date	String	Date in dd/mm/yy format
Financial Institution	String	Institution symbol
Location	String	Location (city) of the institution
High	Numeric	Highest price of the day
Low	Numeric	Lowest price of the day
Open	Numeric	The price of the share at the beginning of the trading day
Close	Numeric	The price of the share at the closing time of the day
Volume	Numeric	Number of shares traded on the trading day
Aggregated Data 2 days	Numeric	Means of open/close/high/low of 2 trading days
Aggregated Data 3 days	Numeric	Means of open/close/high/low of 3 trading days
Aggregated Data 5 days	Numeric	Means of open/close/high/low of 5 trading days
Number of employees	Numeric	Number of employees of an institution
Net change 0 (numeric)	Numeric	Net price change of current day
Net change 0 (nominal)	String	Flag the change as positive or negative
Net change 5 (numeric)	Numeric	Net price change of the past 5 days
Net change 5 (nominal)	String	Flag the change as positive or negative

Net change 25 (numeric)	Numeric	Net price change of the past 25 days
Net change 25 (nominal)	String	Flag the change as positive or negative

Besides the stock data, there is also a supplementary dataset. This supplementary dataset contains the company details. The stock dataset does not include the company name in the CSV file, just the stock code the company is listed on NYSE. The company dataset would provide more information on the company. The company dataset is stored in a single CSV file and contains 8 columns and 505 rows.

The structure of the company dataset is as follows:

Attribute	**Type**	**Explanation**
Ticker symbol	String	The stock code in NYSE
Security	String	Company name
SEC Filings	String	The type of filings used by the company
GISC Sector	String	The sector the company is in
GISC Sub Industry	String	The sub industry the company is in
Address of Headquarters	String	The address of the company headquarters
Date first added	String	The date of the company is added to NYSE
CIK	String	A unique code for the company

Even though the data in the CSV files are structured and cleaned, there might still be some cleansing, formatting and empty values management that need to be performed on the data before the data can be used for technical analysis.

Loading the dataset

Since the stock dataset is distributed on 20 CSV files. It would be time consuming and troublesome if it is loaded one by one. All the 20 stock CSV files are in the same directory and there are no other files within the directory. Therefore, the CSV files are loaded by checking the patterns of the files in the directory. The 20 files are loaded and merged into 1 single data frame after loading.

For the loading dataset code to work, the 20 stocks CSV files must be in a directory call ‘nyse-financial-stocks’ and the company details (securities) CSV files would be at the same level as the nyse-financial-stocks directory.

The stockCol and companyCol variable is a vector of column names to replace the existing column names in the dataset. The 20 CSV files and read into data frames and 20 of the data frames are merged into 1 data frame with the function rbind().

```
setwd('./nyse-financial-stocks/')
files <- list.files(pattern = "*.csv")
# The column names for the imported stock datasets
stockCol <- c("Date", "Stock", "Location", "High", "Low", "Open", "Close",
        "Aggregated data 2 days", "Aggregated data 3 days", "Aggregated data 5 days",
        "Volume", "Number of employees", "Net change 0-numeric", "Net change 0-nominal",
        "Net change 5-numeric", "Net change 5-nominal", "Net change 25-numeric",
        "Net change 25-nominal")
# Read the 20 source files and merge them into a single datasframe
stockDf <- do.call(rbind, lapply(files, function(x) read.csv(x, stringsAsFactors = FALSE,
                                        header = TRUE, col.names = stockCol)))

# The column names for the imported company dataset
companyCol <- c('Stock', 'Company', 'SEC filings', 'GICS Sector', 'GICS Sub Industry', 'HQ Address',
         'Date first added', 'CIK')
companyDetails <- read.csv('../securities.csv', stringsAsFactors = FALSE, col.names = companyCol)
```

Exploring Raw Data

After loading all the CSV files, the data would now be in the form of data frame. The data loaded need to be verified if it is loaded correctly and data of each of the columns are in the correct format.

A glimpse on the top 6 rows of the data frames is performed with the **head** command. The **dim** command shows the dimension of the data frames, since all the stock data is merged, now it has 25180 rows and 18 columns. The **summary** command would show the quartiles, the mean, median, the minimum and maximum values of all the columns of numerical values. The **View** command let the users view the data in a table format, it is useful to get a big picture view on the data.

The most important command in this stage is the **str** command. It shows the structure of the data and the types of the data for each column. From the output of this command, we found that some of the columns are not in the correct format, thus cleaning and conversion need to be performed on the data.

```
# Check the structure and dimension of the data
head(stockDf)
dim(stockDf)
str(stockDf)
summary(stockDf)

head(companyDetails)s
dim(companyDetails)
str(companyDetails)

# View the data to get a big picture
View(stockDf)
View(companyDetails)
```

The output of the str command before cleansing:

```
> str(stockDf)
'data.frame':	25180 obs. of  18 variables:
 $ Date                  : chr  "30/11/2012" "3/12/2012" "4/12/2012" "5/12/2012" ...
 $ Stock                 : chr  "AXP" "AXP" "AXP" "AXP" ...
 $ Location              : chr  "New York" "New York" "New York" "New York" ...
 $ High                  : num  56.1 56.5 56.2 56.7 56.3 ...
 $ Low                   : num  55.7 55.9 55.5 55.8 55.9 ...
 $ Open                  : num  55.9 56.1 55.9 55.9 56.3 ...
 $ Close                 : num  55.9 56 55.8 56.4 56.1 ...
 $ Aggregated.data.2.days: num  56 55.9 56.1 56.3 56.4 ...
 $ Aggregated.data.3.days: num  55.9 56.1 56.1 56.4 56.5 ...
 $ Aggregated.data.5.days: num  56 56.2 56.3 56.6 56.8 ...
 $ Volume                : num  5716100 4954600 3912300 5667500 3854400 ...
 $ Number.of.employees   : chr  "56,400" "56,400" "56,400" "56,400" ...
 $ Net.change.0.numeric  : num  0.04 -0.1 -0.06 0.51 -0.18 0.36 0.08 0.06 0.32 0.06 ...
 $ Net.change.0.nominal  : chr  "Positive" "Negative" "Negative" "Positive" ...
 $ Net.change.5.numeric  : num  0.26 0.52 0.85 1.17 1.37 1.47 -0.02 0.27 0.47 -0.87 ...
 $ Net.change.5.nominal  : chr  "Positive" "Positive" "Positive" "Positive" ...
 $ Net.change.25.numeric : num  4 4.11 4.36 4.9 4.94 4.96 4.09 3.62 3.39 2.12 ...
 $ Net.change.25.nominal : chr  "Positive" "Positive" "Positive" "Positive" ...

> str(companyDetails)
'data.frame':	505 obs. of  8 variables:
 $ Stock           : chr  "MMM" "ABT" "ABBV" "ACN" ...
 $ Company         : chr  "3M Company" "Abbott Laboratories" "AbbVie" "Accenture plc" ...
 $ SEC.filings     : chr  "reports" "reports" "reports" "reports" ...
 $ GICS.Sector     : chr  "Industrials" "Health Care" "Health Care" "Information Technology" ...
 $ GICS.Sub.Industry: chr  "Industrial Conglomerates" "Health Care Equipment" "Pharmaceuticals" ...
 $ HQ.Address      : chr  "St. Paul, Minnesota" "North Chicago, Illinois" "North Chicago, Illinois" ...
 $ Date.first.added : chr  "" "31/3/1964" "31/12/2012" "6/7/2011" ...
 $ CIK             : int  66740 1800 1551152 1467373 718877 1144215 796343 1158449 874761 1122304
```

Data Cleansing

Format conversion

From the output of the str command on the data frames, we found that some of the columns of the data frames are not in the correct format. The **date** columns of both the data frames are in character (chr) form rather than the date format. The date format must be corrected for the analysis. To convert the strings into date format, we would import **lubridate** library. With a single command from lubridate directory, all the string in the date columns are now converted into Date format.

```
# Fix the date format in the dataset

library(lubridate)

library(dplyr)

stockDf$Date <- dmy(stockDf$Date)


companyDetails$Date.first.added <- dmy(companyDetails$Date.first.added)
```

Besides the date, the '**number of employees**' field in the stock data frame that is supposed to be numeric, but it is in character format. This is because there is comma within the number to increase human readability. This comma cause R to read the columns as string rather than number. In this scenario, we need to remove the comma in all the rows of the 'number of employees' column, then convert it into numerical format. The library **stringr** would be used for string replacement.

```
# Fix the number incorrectly represented as characters

library(stringr)

stockDf$Number.of.employees <- str_replace(stockDf$Number.of.employees, pattern = ',',
replacement = '')

stockDf$Number.of.employees <- as.numeric(stockDf$Number.of.employees)
```

After this conversion, all the data in the data frames are in the correct format. However, some of the columns in string format would be better if it is converted into factors. These columns only have a few types of values in it. The columns are the Net change 0 nominal, Net change 5 nominal and Net change 25 nominal columns in the stock data frame and the SEC filings, GICS Sector and GICS Sub Industry columns in the company details data frame.

After converting the few columns from string into factors, we check the levels of the factors. The levels contain some empty string, though this is not counted as NA, it would be better if we replace the empty string with a value. Thus, we replace the levels of the several factors.

```
# Fix the string that should be formatted as factors
stockDf$Net.change.0.nominal <- as.factor(stockDf$Net.change.0.nominal)
stockDf$Net.change.5.nominal <- as.factor(stockDf$Net.change.5.nominal)
stockDf$Net.change.25.nominal <- as.factor(stockDf$Net.change.25.nominal)

companyDetails$SEC.filings <- as.factor(companyDetails$SEC.filings)
companyDetails$GICS.Sector <- as.factor(companyDetails$GICS.Sector)
companyDetails$GICS.Sub.Industry <- as.factor(companyDetails$GICS.Sub.Industry)

levels(stockDf$Net.change.0.nominal)
levels(stockDf$Net.change.5.nominal)
levels(stockDf$Net.change.25.nominal)

levels(stockDf$Net.change.0.nominal) <- c("Unknown", "Equal", "Negative", "Positive")
levels(stockDf$Net.change.5.nominal) <- c("Unknown", "Equal", "Negative", "Positive")
levels(stockDf$Net.change.25.nominal) <- c("Unknown", "Equal", "Negative", "Positive")

levels(companyDetails$SEC.filings)
levels(companyDetails$GICS.Sector)
levels(companyDetails$GICS.Sub.Industry)
```

After all the data format conversion, all the columns now should be in the correct format. We check the structure of the data frame with the str command again, the output of the str command is as follows:

```
> str(stockDf)
'data.frame':	25180 obs. of  18 variables:
 $ Date                   : Date, format: "2012-11-30" "2012-12-03" "2012-12-04" "2012-12-05" ...
 $ Stock                  : chr  "AXP" "AXP" "AXP" "AXP" ...
 $ Location               : chr  "New York" "New York" "New York" "New York" ...
 $ High                   : num  56.1 56.5 56.2 56.7 56.3 ...
 $ Low                    : num  55.7 55.9 55.5 55.8 55.9 ...
 $ Open                   : num  55.9 56.1 55.9 55.9 56.3 ...
 $ Close                  : num  55.9 56 55.8 56.4 56.1 ...
 $ Aggregated.data.2.days: num  56 55.9 56.1 56.3 56.4 ...
 $ Aggregated.data.3.days: num  55.9 56.1 56.1 56.4 56.5 ...
 $ Aggregated.data.5.days: num  56 56.2 56.3 56.6 56.8 ...
 $ Volume                 : num  5716100 4954600 3912300 5667500 3854400 ...
 $ Number.of.employees    : num  56400 56400 56400 56400 56400 56400 56400 56400 56400...
 $ Net.change.0.numeric  : num  0.04 -0.1 -0.06 0.51 -0.18 0.36 0.08 0.06 0.32 0.06 ...
 $ Net.change.0.nominal  : Factor w/ 4 levels "Unknown","Equal",..: 4 3 3 4 3 4 4 4 4 4 ...
 $ Net.change.5.numeric  : num  0.26 0.52 0.85 1.17 1.37 1.47 -0.02 0.27 0.47 -0.87 ...
 $ Net.change.5.nominal  : Factor w/ 4 levels "Unknown","Equal",..: 4 4 4 4 4 4 2 4 4 3 ...
 $ Net.change.25.numeric : num  4 4.11 4.36 4.9 4.94 4.96 4.09 3.62 3.39 2.12 ...
 $ Net.change.25.nominal : Factor w/ 4 levels "Unknown","Equal",..: 4 4 4 4 4 4 4 4 4 4 ...

> str(companyDetails)
'data.frame':	505 obs. of  8 variables:
 $ Stock            : chr  "MMM" "ABT" "ABBV" "ACN" ...
 $ Company          : chr  "3M Company" "Abbott Laboratories" "AbbVie" "Accenture plc" ...
 $ SEC.filings      : Factor w/ 1 level "reports": 1 1 1 1 1 1 1 1 1 1 ...
 $ GICS.Sector      : Factor w/ 11 levels "Consumer Discretionary",..: 6 5 5 7 7 6 7 1 11 5 ...
 $ GICS.Sub.Industry: Factor w/ 124 levels "Advertising",..: 67 51 96 77 56 39 9 13 66 81 ...
 $ HQ.Address       : chr  "St. Paul, Minnesota" "North Chicago, Illinois" "Dublin, Ireland" ...
 $ Date.first.added : Date, format: NA "1964-03-31" "2012-12-31" "2011-07-06" ...
 $ CIK              : int  66740 1800 1551152 1467373 718877 1144215 796343 1158449 874761...
```

The date columns are now in the date format. The ‘number of employees’ column is now in the numeric format. The several columns with only a few different values are now in factors format.

NAs Management

Another section of data cleaning is managing the empty values or NA. First, we check if there is any NAs in our data frames.

```
# Check if there is NA in the dataset

sum(is.na(stockDf))

sum(is.na(companyDetails))
```

From the commands above, we found that there are 575 NA values in our stock data frame and 198 NA values in our company details data frame. Now we know that there are some NAs in our data frames but where is those NAs located? Which columns there are in? Other than go through the columns 1 by 1 with the function above, we can use the **sapply** function to check which of the column has NAs.

```
# Find out where is the NA

stockDf[!complete.cases(stockDf),]

sapply(stockDf, function (x) sum(is.na(x)))

nrow(stockDf[!complete.cases(stockDf),])
```

From the **nrow** function we get the number of rows that contain NA. This number is 495 which is lesser than 575, because some rows has more than 1 NA. The **View** function let us view the data in a tabular form, which could be easy for us to spot the NAs. From the view, we noticed that the NAs are in the column 'Net change 25 numeric' and 'Net change 5 numeric'. For some reasons, the values are not available in that few rows.

There are several ways to manage NA values. We can omit the rows that contain the NAs values with the function **na.omit()**, but that would make the data incomplete. Since the data is a time series, if some rows are missing then a fraction of the overall data would be missing. Therefore, omission is not the best way in this scenario. The other way to handle missing data or NA values is imputation which is replacing the missing values with dummy values. In this case, a dummy value of 0 would not be very meaningful. Other than 0 value, another replacement candidate is the mean of the column, replace the missing values with the mean. This would give us an approximate to the real value. After replacing all the missing values, we perform another check to verify all the NA values have been handled.

```
# Most of the NAs are from the column Net.change.25.numeric

sum(is.na(stockDf$Net.change.25.numeric))

# Fill in the NAs with the mean from the column

stockDf$Net.change.25.numeric[is.na(stockDf$Net.change.25.numeric)] <-
mean(stockDf$Net.change.25.numeric, na.rm=T)


# Another column with NAs

sum(is.na(stockDf$Net.change.5.numeric))

stockDf$Net.change.5.numeric[is.na(stockDf$Net.change.5.numeric)] <-
mean(stockDf$Net.change.5.numeric, na.rm=T)


# Removed all the NAs from the stock dataframe

sum(is.na(stockDf))
```

All the missing values in the stock data frame have been handled. Next, we handle the missing values in the company details data frame. Same step as before, we check the where is NA values.

```
# Find the NAs in the company dataframe

companyDetails[!complete.cases(companyDetails),]

sapply(companyDetails, function (x) sum(is.na(x)))

nrow(companyDetails[!complete.cases(companyDetails),])

sum(is.na(companyDetails$Date.first.added))
```

From the output of the code above, we found that there are 198 missing values in the company details data frame and all the missing values come from the date column.

To handle the missing values in the date column, an empty string or 0 would not be sensible. Therefore, we replace the missing values of date with a date, a dummy value of date, 01-01-1970.

```
# Replace the NAs in the date column with a date
companyDetails$Date.first.added[is.na(companyDetails$Date.first.added)] <- dmy('01-01-1970')


# Removed all the NAs in the company data frame
sum(is.na(companyDetails))
```

After replacing the missing dates with a dummy date, we verify that there are no NA values in the data frame now. At this stage, the data has been cleaned and tidied. The next stage would be to prepare the data for analysis.

Preparing data for analysis

At this juncture, all the columns in the data frame is in the correct format and there are no missing values in the data frames. The next part would be to prepare the data for analysis, merge the 2 data frames into a single data frame, create new columns that would be needed for analysis, and removing the unnecessary columns from the data frame.

Sanity check

The following code check the number of unique stocks on the data frames. In order to merge the data frame, we need to know more about the attributes to merge them.

```
# Check unique stock
unique(stockDf$Stock)
length(unique(stockDf$Stock))

unique(companyDetails$Stock)
length(unique(companyDetails$Stock))
```

From the output of the code above, there are 20 unique stocks in the stock data frame, which is expected since we load 20 CSV files. In the company details data frame, there are 505 unique stocks. The stock data frame is our primary data and the company details is our supplementary data, we do not need all the information in the company details data frame, just the information about the 20 stocks.

Functions

Equipped with that knowledge, we create some functions what might be useful in the next part. The functions are to retrieve company details and company name with the stock code or ticker symbol.

```
# Function  to get company details with the stock code
getCompanyDetails <- function(tickerSym) {
  stock <- companyDetails[which(companyDetails$Stock==tickerSym),]
  return(stock)
}

getCompanyName <- function(tickerSym) {
  companyName <- companyDetails[which(companyDetails$Stock==tickerSym),]$Company
  return(companyName)
}

getCompanyDetails(tickerSym = 'AXP')
getCompanyDetails(tickerSym = 'BAC')
getCompanyDetails('C')

getCompanyName(tickerSym = 'AXP')
getCompanyName(tickerSym = 'ABT')
getCompanyName(tickerSym = 'ALB')
```

Merge the data frames

As mentioned before, the stock data frame is the primary data, the company data is the supplementary data. The merging of the data frame would be using the **left_join** function. We only want the company details of the companies in the stock data frame, not all the company details in the company details data frame.

The **left_join** function would join the 2 data frames based on a similar column name. In the beginning, while we import the dataset into R, we provide the column names for both data frames. There is a column with similar name in both data frame. If there is no common column names between the data frames, it would still be able to merge the data frame but it have to use another function called **merge()** which allows the user to specify the column name to be merged on in both data frames.

```
# Merge the 2 dataset with left join with the similar column Stock
stockData <- left_join(stockDf, companyDetails, by = c('Stock'))
colnames(stockData)
dim(stockData)
View(stockData)
```

With the command, the 2 data frames are merged. Now it is a data frame with dimension 25180 rows and 25 columns.

Out of the 25 columns, not all the columns are needed for the analysis. Only selected columns would be needed for further analysis. The **select()** command let us select the columns we want to keep from the data frame.

```
# Select the columns for further analysis
stockData <- select(stockData, c("Date", "Stock", "High", "Low", "Open", "Close",
                    "Volume", "Company", "GICS.Sector", "GICS.Sub.Industry",
                    "Net.change.0.numeric", "Net.change.0.nominal"))
colnames(stockData)
dim(stockData)
```

After the selection or removal, now the dimension of the data frame is 25180 rows and 12 columns.

Some of the column names of the data frame is long and would be inconvenient in the process of analysis later. The solution would be to rename the column of the data frame.

```
# Rename the columns to intuitive names
names(stockData) <- c("Date", "StockCode", "High", "Low", "Open", "Close", "Volume", "Company",
              "Sector", "SubIndustry", "NetChange", "NetChangeNominal")
colnames(stockData)
View(stockData)
```

Mutate new column

With the existing columns, we can create a new column that would be needed in the analysis session. The column we are going to create here is High-Low Difference column. This column is generated by using each row High value subtracts each row Low value. The High-Low Difference column would show the analysts the price gap between the highest price and lowest price of the traded shares. The function to generate the new column is the **mutate** function from the **dplyr** library.

```
# Create new columns from the existing columns

stockData <- stockData %>% mutate(HighLowDiff = High - Low)

colnames(stockData)

glimpse(stockData)
```

Now the data frame is complete. All the columns are in the correct format and in a tidy format. The data would be ready for the analysis stage next.

Exploratory Data Visualization

Before diving into the data analysis, we can visualize part of the clean data frame to get a visual or picture of the data. The way to do that is data visualization, we can plot some graphs from the data frame. Since there are 20 stocks in the data frame, it would be reasonable at this stage, just to plot a graph for just 1 stock at a time. In this case, function can be applied again.

The function would receive a stock code then it would subset the data frame, just focus on the stock selected. After that, visualize the closing price, the high-low difference and the volume of the stock. The first function show here is to select the time frame used to subset the stock data. Each of the plot function would call this selecting time frame function first before plotting the graphs.

```
# Function to get year range for the plot functions
getYear <- function(year) {
  vec <- c()
  if (year == '2013') {
    vec <- c('2013', '01-01-2013', '31-12-2013')
  } else if (year == '2014') {
    vec <- c('2014', '01-01-2014', '31-12-2014')
  } else if (year == '2015') {
    vec <- c('2015', '01-01-2015', '31-12-2015')
  } else if (year == '2016') {
    vec <- c('2016', '01-01-2016', '31-12-2016')
  } else if (year == '2017') {
    vec <- c('2017', '01-01-2017', '31-12-2017')
  } else {
    vec <- c()
  }
  return (vec)
}

# Closing price of a certain stock
plotClosePrice <- function(code, year='2017') {
  yearVec <- getYear(year)
  subStock <- stockData %>% filter(StockCode==code, Date >= dmy(yearVec[2]) & Date <=
dmy(yearVec[3]))
  ggplot(data = subStock, aes(x=Date, y=Close)) +
    geom_line(color='Blue') +
    ggtitle(paste('Closing Price for', getCompanyName(code))) +
    xlab(yearVec[1]) +
    ylab('Closing Price, $')
}

plotClosePrice(code="BAC", year='2016')

# High Low difference of a stock
plotHighLowDiff <- function(code, year='2017') {
  yearVec <- getYear(year)
  subStock <- stockData %>% filter(StockCode==code, Date >= dmy(yearVec[2]) & Date <=
dmy(yearVec[3]))
  ggplot(data=subStock, aes(x=Date, y=HighLowDiff)) +
    ggtitle(paste('High-Low Difference for', getCompanyName(code))) +
    geom_line(color='blue') +
    xlab(yearVec[1]) +
    ylab('High-Low Difference, $')
}

plotHighLowDiff(code='BAC', year='2016')
```

The graphs are as follows:

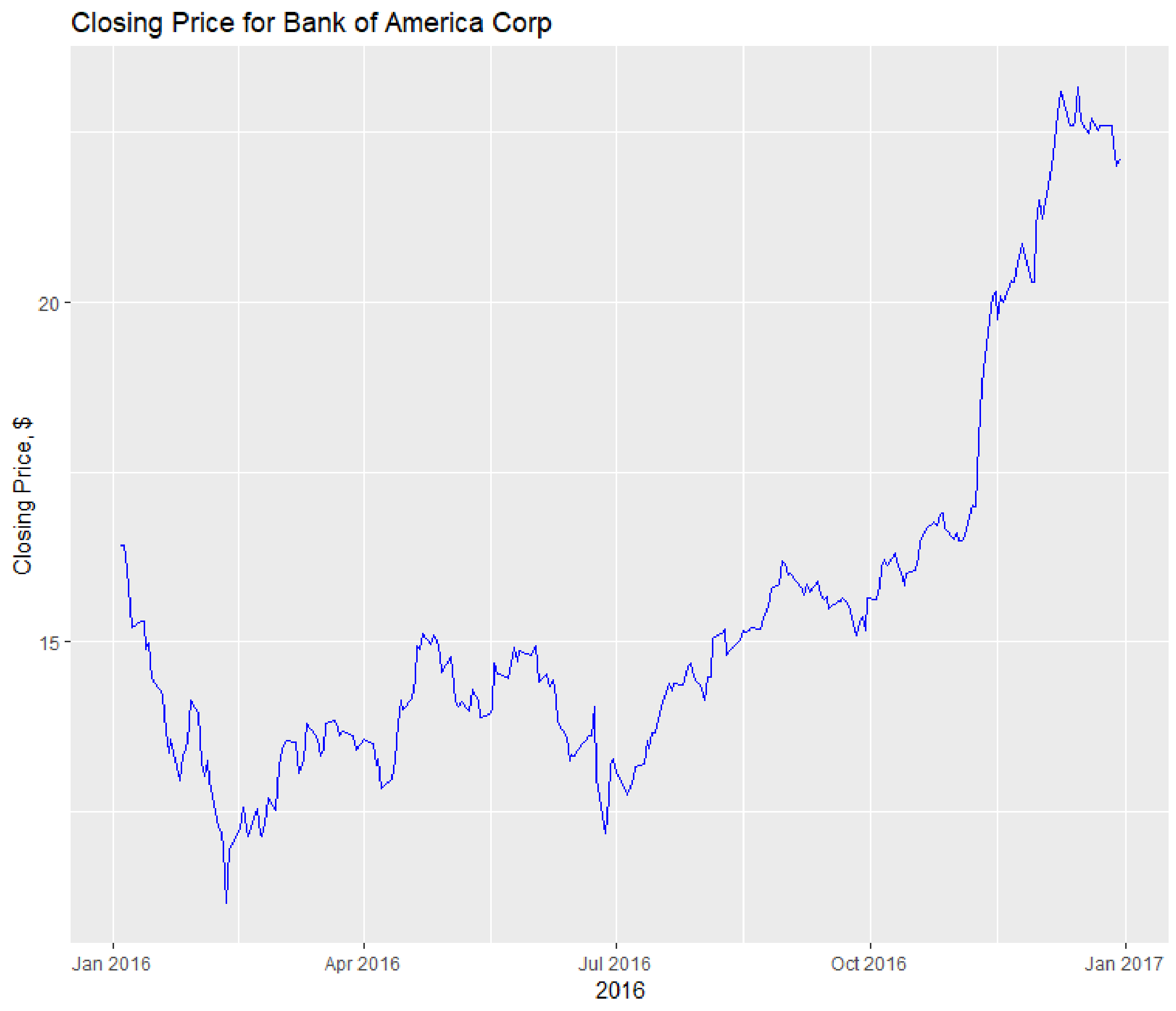

Figure 1 Closing price of Bank of America Corp

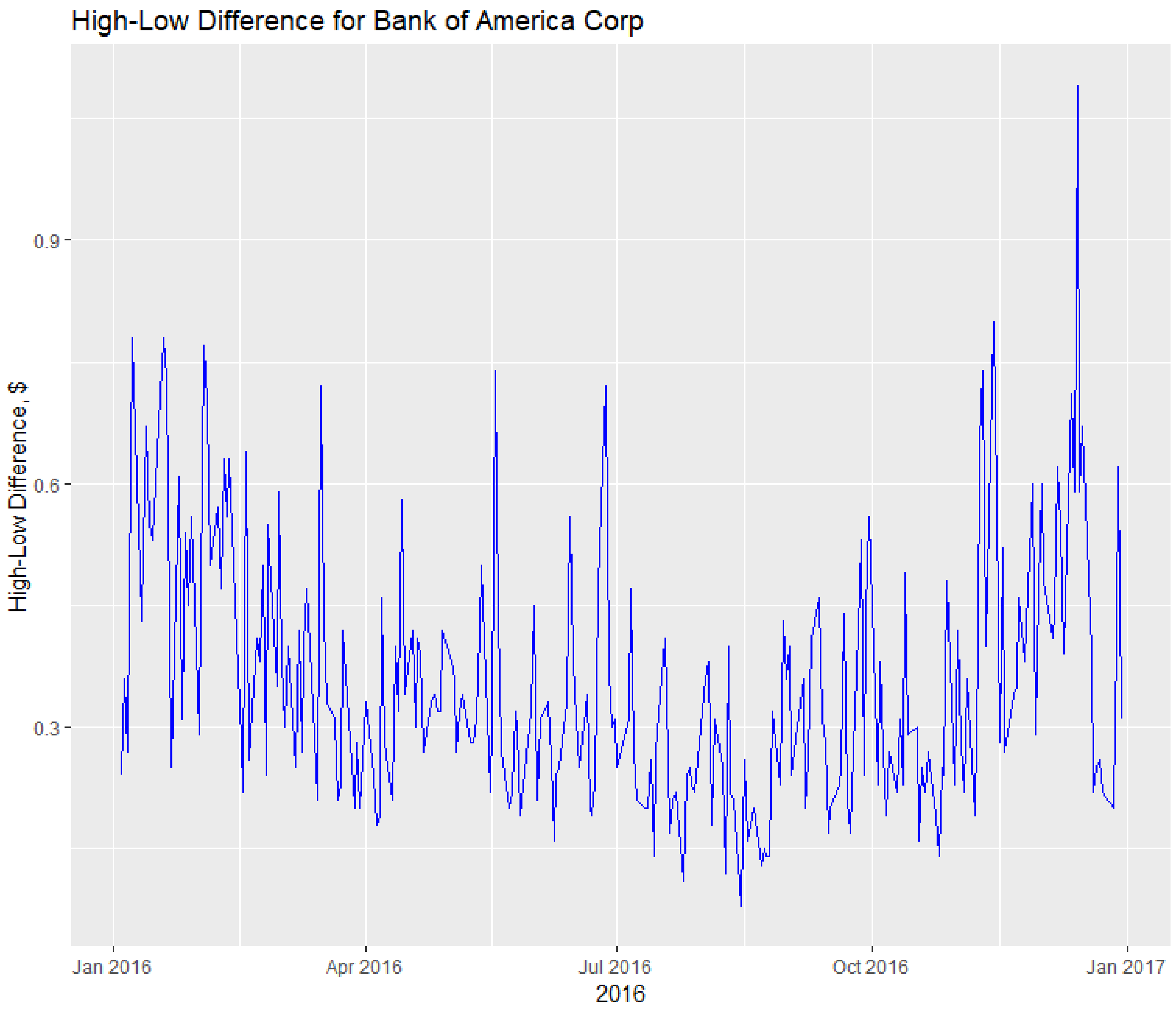

Figure 2 High-Low Difference for Bank of America Corp

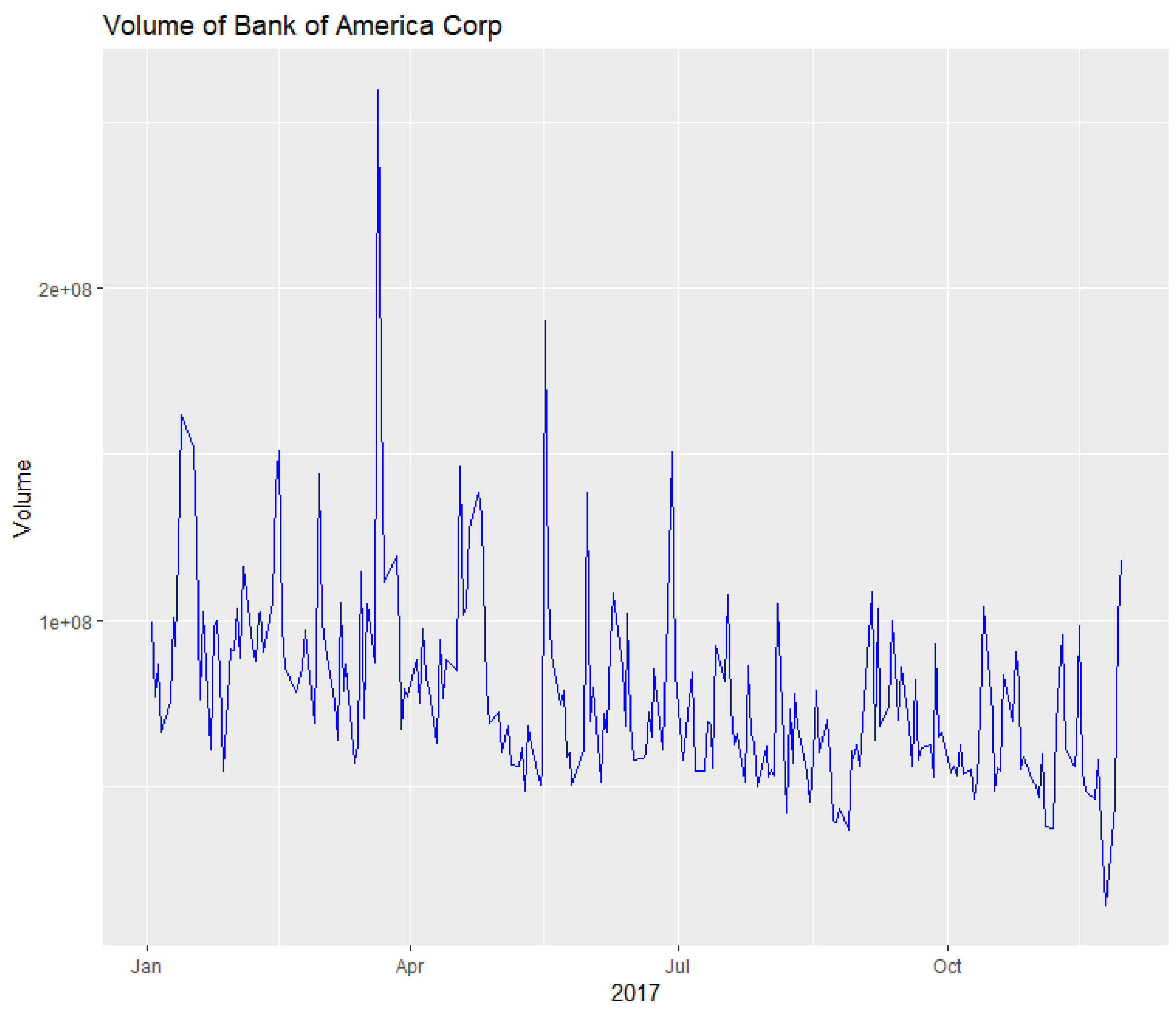

Figure 3 Volume of Bank of America Corp

The graphs above just shown the data for a single year. It would be easier to compare the year on year data side by side. First, all the graphs for each of the year is plotted and arrange in 2 columns to be shown in the plot area for ease of comparison.

```
# Comparing the year on year closing price
BACClo2013 <- plotClosePrice(code = 'BAC', year = '2013')
BACClo2014 <- plotClosePrice(code = 'BAC', year = '2014')
BACClo2015 <- plotClosePrice(code = 'BAC', year = '2015')
BACClo2016 <- plotClosePrice(code = 'BAC', year = '2016')
BACClo2017 <- plotClosePrice(code = 'BAC', year = '2017')
grid.arrange(BACClo2013, BACClo2014, BACClo2015, BACClo2016, BACClo2017, ncol=2)
```

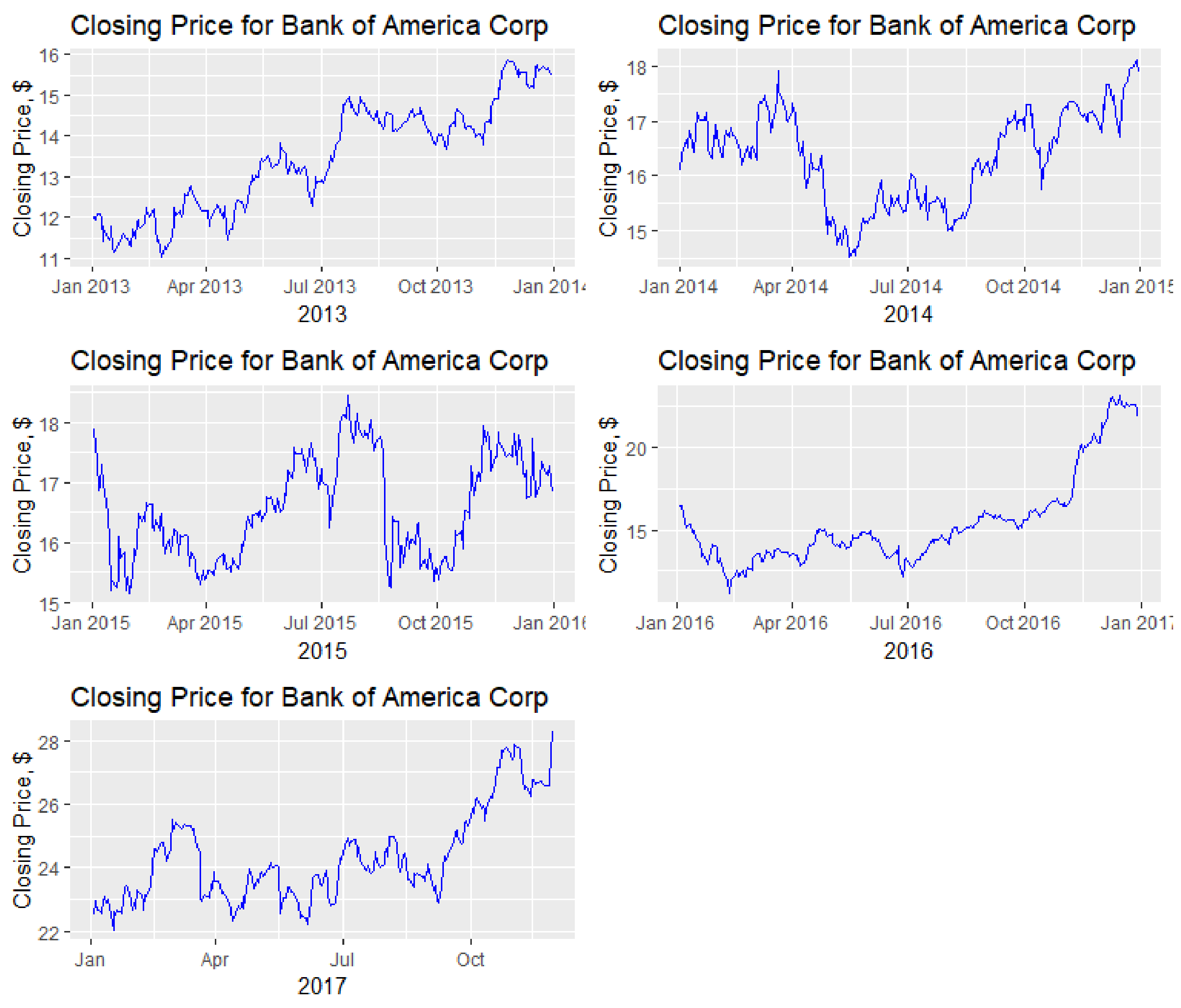

Figure 4 Closing price for Bank of America Corp year on year comparison

From the comparison of year on year closing price, a pattern can be seen, the closing price at the end of the year is always higher than that of the closing price at the beginning of the year apart from 2015. It is safe to assume, the closing price at the end of year 2017 would be higher than that of the beginning of year 2017.

Comparison is also made on the volume of the stock.

```
# Comparing the year on year sales volume
BACVol2013 <- plotVolume(code = 'BAC', year = '2013')
BACVol2014 <- plotVolume(code = 'BAC', year = '2014')
BACVol2015 <- plotVolume(code = 'BAC', year = '2015')
BACVol2016 <- plotVolume(code = 'BAC', year = '2016')
BACVol2017 <- plotVolume(code = 'BAC', year = '2017')
grid.arrange(BACVol2013, BACVol2014, BACVol2015, BACVol2016, BACVol2017, ncol=2)
```

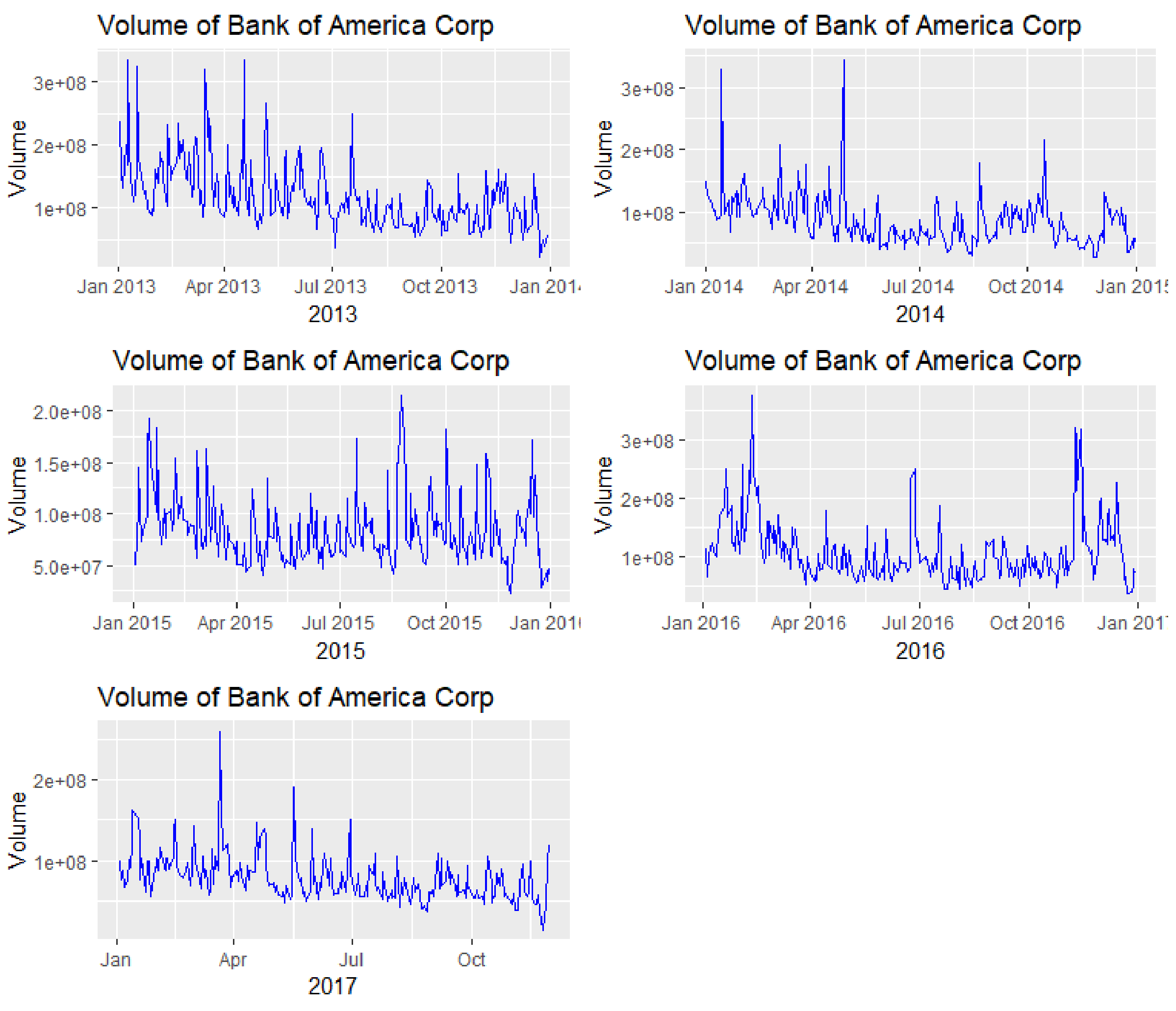

Figure 5 Volume of Bank of America Corp year on year comparison

The volume of a stock also can be compared year on year. From the graphs shown, the volume decreases at the end of the years. The same scenario could happen in 2017 as well.

High-Low difference is compared year on year as well, with the following codes.

```
# Comparing High low differences
BACPri2013 <- plotHighLowDiff(code = 'BAC', year = '2013')
BACPri2014 <- plotHighLowDiff(code = 'BAC', year = '2014')
BACPri2015 <- plotHighLowDiff(code = 'BAC', year = '2015')
BACPri2016 <- plotHighLowDiff(code = 'BAC', year = '2016')
BACPri2017 <- plotHighLowDiff(code = 'BAC', year = '2017')
grid.arrange(BACPri2013, BACPri2014, BACPri2015, BACPri2016, BACPri2017, ncol=2)
```

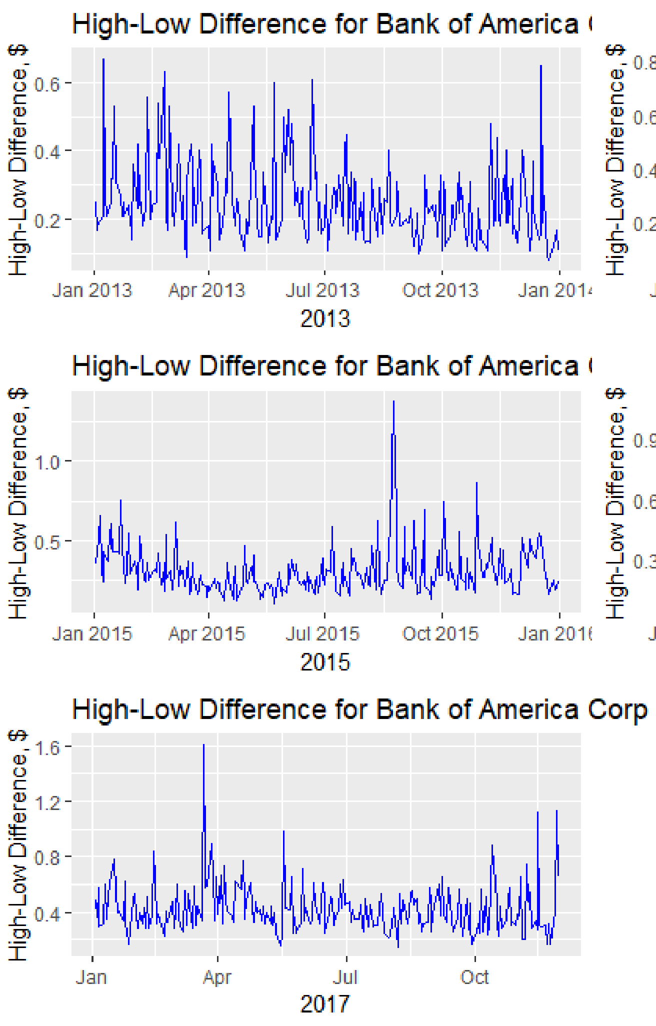

Figure 6 High-Low difference of Bank of America Corp year on year comparison

No apparent patterns can be found on the High-Low difference year on year comparison.

Besides graphs, the data can be explored to check the correlation between the variables. The correlation between the variables would prove to be useful for applying machine learning algorithms on the dataset. In this case, linear regression could be applied since most of the variables are continuous values.

```
# Exploring the correlation between variables
BACStock <- stockData %>% filter(StockCode == 'BAC')
cor(BACStock$Volume, BACStock$Close)
cor(BACStock$Volume, BACStock$High)
ggplot(BACStock, aes(x=High, y=Volume)) +
  geom_line(color='blue')

cor(BACStock$High, BACStock$Close)
cor(BACStock$Close, BACStock$Low)
cor(BACStock$High, BACStock$Low)
cor(BACStock$High, BACStock$Open)
cor(BACStock$Open, BACStock$Close)
cor(BACStock$Open, BACStock$High + BACStock$Low)
ggplot(BACStock, aes(x=Open, y=High)) +
  geom_point(color='blue') +
  geom_abline(color='red', size=1)
ggplot(BACStock, aes(x=Open, y=Low)) +
  geom_point(color='blue') +
  geom_abline(color='red', size=1)
ggplot(BACStock, aes(x=High, y=Low)) +
  geom_point(color='blue') +
  geom_abline(color='red', size=1)
# There is a corelation between High, Close, Low

Cstock <- stockData %>% filter(StockCode == 'C')
cor(Cstock$Close, Cstock$High)
cor(Cstock$Open, Cstock$High)
cor(Cstock$Open, Cstock$Close)
cor(Cstock$Open, Cstock$Low)

ggplot(Cstock, aes(x = Open, y = High)) +
  geom_point(color='blue') +
  geom_abline(color='red', size=1)
ggplot(Cstock, aes(x = Open, y = Close)) +
  geom_point(color='blue') +
  geom_abline(color='red', size=1)
```

It is found that the feature ‘Open’ has a high correlation value with ‘High’, ‘Low’, and ‘Close’. These correlations can be illustrated in a graph.

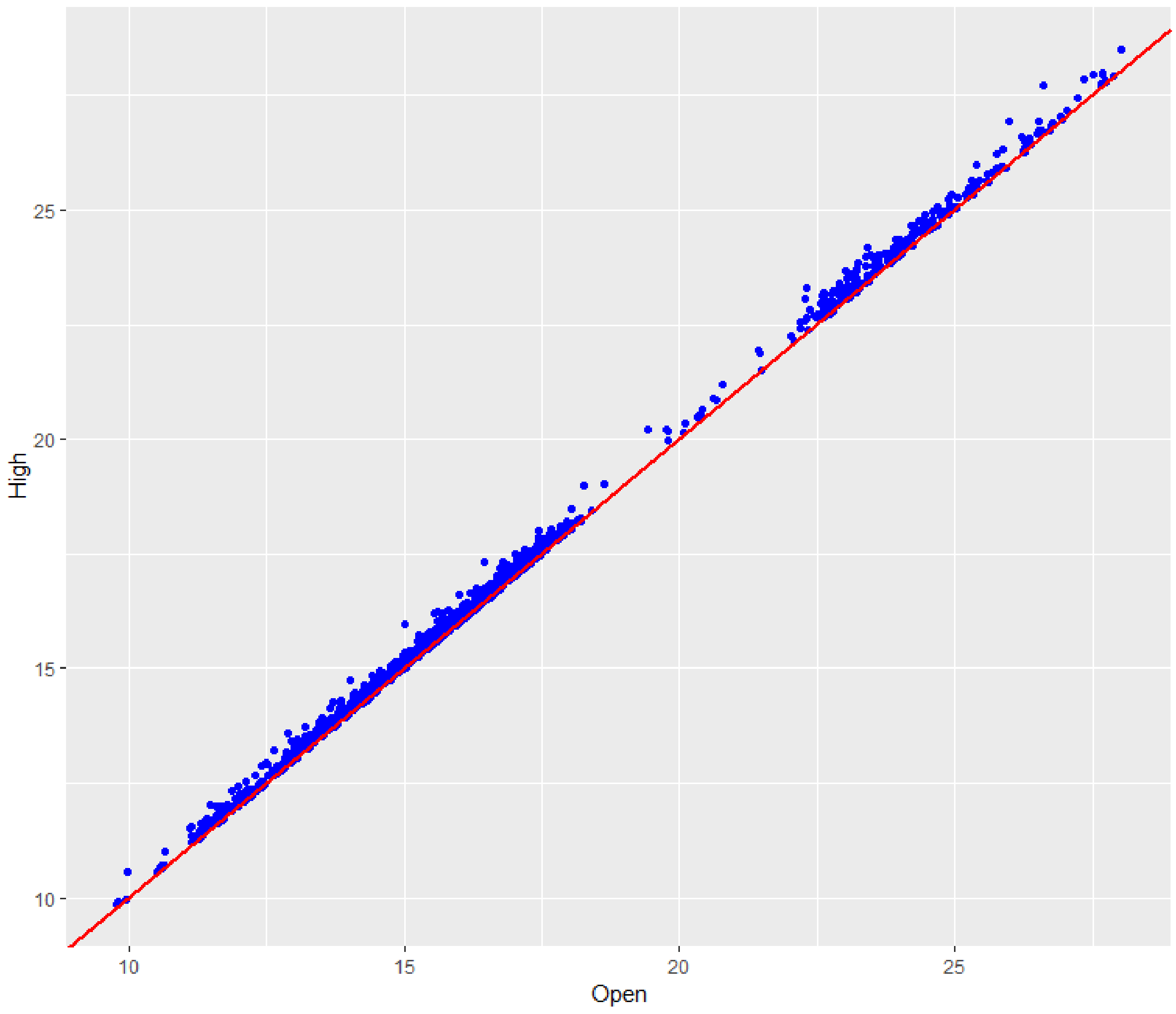

Figure 7 Correlation between 'Open' and 'High'

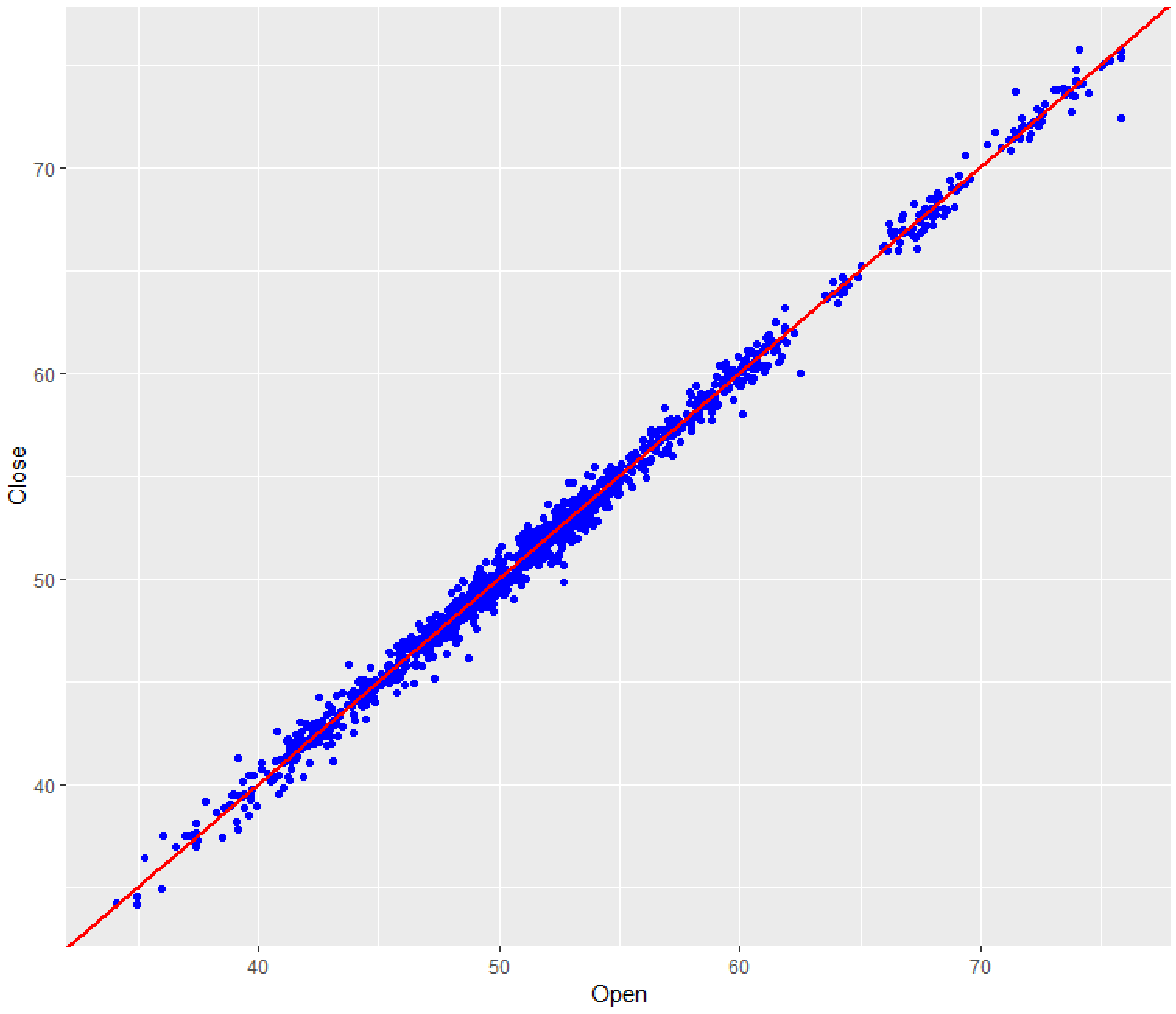

Figure 8 Correlation between 'Open' and 'Close'

Linear Regression

After getting the correlation between the features, one of the features can be used to predict or determine the value of another feature, in other words, linear regression. Since it is found that ‘Open’ and ‘High’ have a high correlation value. The linear regression apply in this part would use the ‘Open’ feature to predict the value of ‘High’. The dataset can be divided into years. The training data would be the dataset of year 2013 to 2016. The testing set would be the dataset of year 2017.

First, the dataset must be filtered by the stock code and the year. Then divide the dataset into training set and testing set based on the years. Feature scaling is not applied because the range of both the values are similar.

After the dataset is divided, a regression model is trained with the training set, specifying the formula, using ‘Open’ to predict ‘High’. After the model is trained, it is used to predict the ‘High’ value in the test set. The predicted value is shown on the graph as a line and the exact values are shown on the graphs as dots.

The linear regression model is evaluated by calculating the mean squared error and root mean squared error. The root mean squared error would show how much the prediction differ from the exact values. From the result, RMSE value is a small value, 0.13 and 0.38 for Bank of America Corp and Citibank Group respectively, thus it means that the regression model is a good fit for the data.

```
############ Linear Regression ###############
# Linear Regression with BAC stock
# Getting the training and testing dataset
BACStockTrain <- stockData %>% filter(StockCode == 'BAC', Date < dmy('01-01-2017')) %>%
  select(StockCode, High, Low, Open, Close)
BACStockTest <- stockData %>% filter(StockCode == 'BAC', Date >= dmy('01-01-2017')) %>%
  select(StockCode, High, Low, Open, Close)

regressionBac <- lm(formula = High ~ Open, data=BACStockTrain)

yPredBac <- predict(regressionBac, newdata = BACStockTest)

summary(regressionBac)

# Residual sum of squares
rss <- c(crossprod(regressionBac$residuals))
rss
# Mean squared error
mse <- rss / length(regressionBac$residuals)
mse
# Root mean squared error
rmse <- sqrt(mse)
rmse
# Pearson estimated residual variance
sig2 <- rss / regressionBac$df.residual
sig2

# Plot with training set
ggplot() +
  geom_point(aes(x = BACStockTrain$Open, y = BACStockTrain$High), color='blue') +
  geom_line(aes(x = BACStockTrain$Open, y=predict(regressionBac, newdata =
BACStockTrain)), color='red',size=1) +
  labs(title = 'Open vs High (training set)', x = 'Open', y='High')

# Plot with test set
ggplot() +
  geom_point(aes(x = BACStockTest$Open, y = BACStockTest$High), colour = 'blue') +
  geom_line(aes(x = BACStockTest$Open, y = yPredBac), colour = 'red', size=1) +
  labs(title = 'Prediction of High price using Open price (test set)', x='Open', y='High')
```

```
# Linear regression with C stock
CstockTrain <- stockData %>% filter(StockCode == 'C', Date < dmy('01-01-2017')) %>%
  select(StockCode, High, Low, Open, Close)
CstockTest <- stockData %>% filter(StockCode == 'C', Date >= dmy('01-01-2017')) %>%
  select(StockCode, High, Low, Open, Close)

regressionC <- lm(formula = High ~ Open, data=CstockTrain)

yPredC <- predict(regressionC, newdata = CstockTest)

summary(regressionC)

# Residual sum of squares
rss <- c(crossprod(regressionC$residuals))
rss
# Mean squared error
mse <- rss / length(regressionC$residuals)
mse
# Root mean squared error
rmse <- sqrt(mse)
rmse
# Pearson estimated residual variance
sig2 <- rss / regressionC$df.residual
sig2

ggplot() +
  geom_point(aes(x = CstockTest$Open, y=CstockTest$High), color='blue') +
  geom_line(aes(x = CstockTest$Open, y=yPredC), color='red', size=1) +
  labs(title='Prediction of High price with Open price', x='Open', y='High')
```

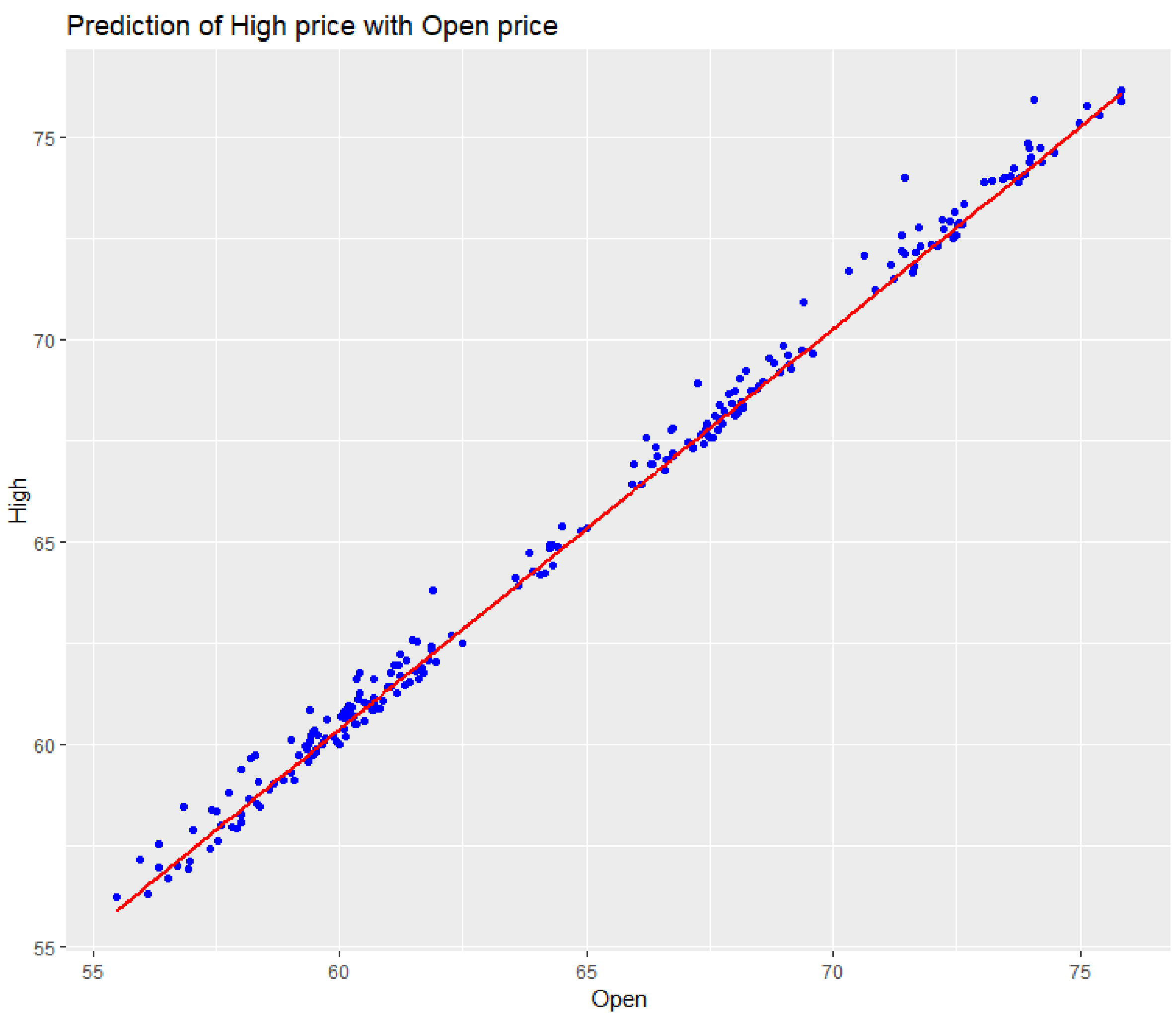

Figure 9 Prediction of the High price using the Open price

Arima

As described previously, the dataset is a time series, having observations of different variables bound by a time factor, in this case, the ‘date’ variable. Forecasting methods may be applied to the time series dataset in an attempt to predict future observations based on historical data. The ARIMA model will be implemented to predict the ‘high’ prices of the ‘BAC’ stock.

The ARIMA model, which stands for Autoregressive Integrated Moving Average, is one of the most popular models to perform forecasting. The main application of this model is the short-term forecasting of events based on at least 40 historical data points. The model works best when applied to data exhibiting stable and consistent pattern over time. This property of consistent pattern over time is known as stationarity. Stationarity of the dataset is verified as shown below through the use of the Partial Autocorrelation and Cross-Correlation Function Estimation Test and the Augmented Dickey-Fuller Test. Both of these tests conclude that the dataset is stationary; the ARIMA model can thus be applied.

```
library(MASS)
library(forecast)
library(tseries)

full = stockData %>% filter(StockCode == "BAC") %>% dplyr::select(Date,High)

# predicting only for 2 days ahead
train = full[1:1257,] # 1:1000
test = full[1258:1259,] # 1001:1259

# Autocorrelation And Cross-Correlation Function Estimation
acf(train$High,lag.max = 20)

# Partial Autocorrelation And Cross-Correlation Function Estimation
# test indicates data is stationary, time series data is predictable to some degree
pacf(train$High,lag.max = 20)

diffstock = diff(train$High,1)
adf.test(train$High)

# Augmented Dickey-Fuller Test to test if model is stationary
adf.test(diffstock)
```

In this case, the ARIMA model is trained using the entire dataset except the latest two observations. The last two observations will be to test the accuracy of the ARIMA model. The forecasted 'high' price had a mean error of 4.01%.

```
pricearima = ts(train$High, start = c(2012,11,30), frequency = 365)

# auto arima automatically tunes the model
fitStock = auto.arima(pricearima)

fitStock
ggplot2::autoplot(pricearima,main="High Price Values for BAC")

forecastedValues = forecast(fitStock, h=2) #259
forecastedValues

# higher level view of the time series + forecast
ggplot2::autoplot(forecast(fitStock, h=2), include = 50)
# closer look at the time series + forecast
ggplot2::autoplot(forecast(fitStock, h=2), include = 10)

# evaluating the mean percentage error of the forecasted prices
finalForecastedValues = as.numeric(forecastedValues$mean)

compare = data.frame(test$High,finalForecastedValues)
columns = c("ActualPrice","ForecastedPrice")
names(compare) = columns
percentage_error = ((compare$ActualPrice - compare$ForecastedPrice)/compare$ActualPrice)
percentage_error
mean(percentage_error)
```

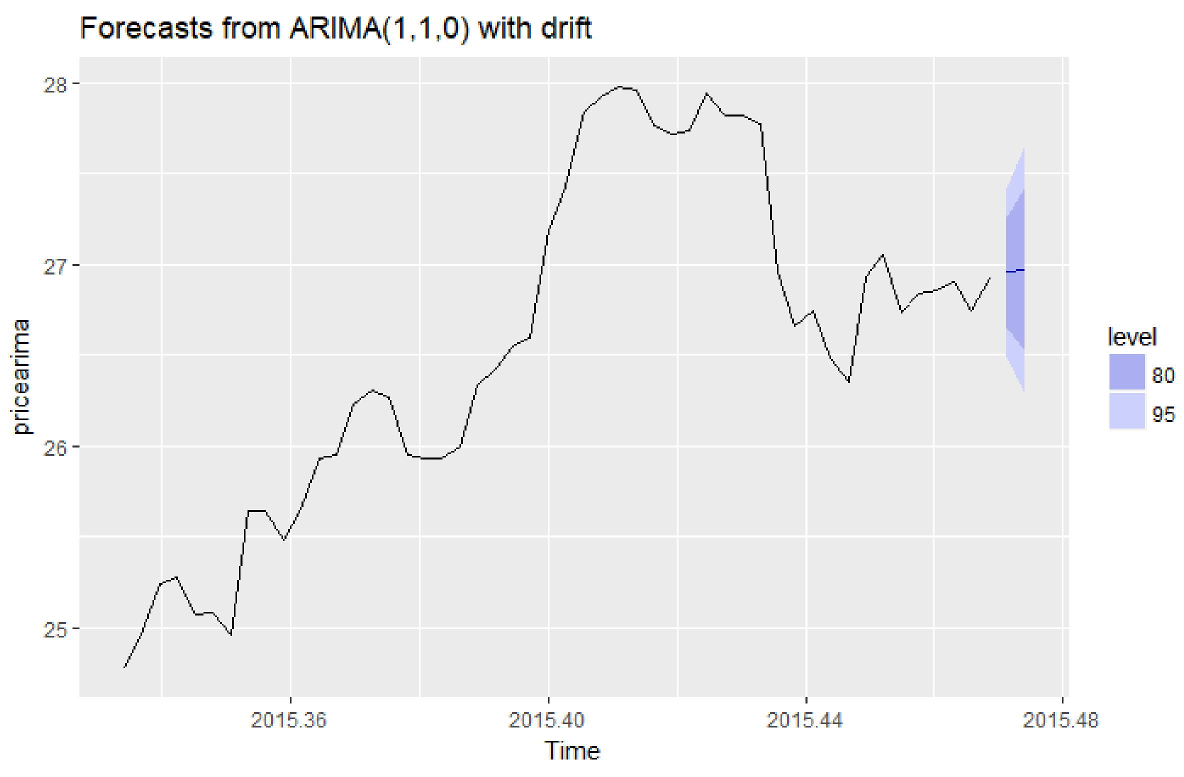

Figure 10 Forecast results of ARIMA model