USING TIME SERIES TO PREDICT STOCK PRICE IN R (WEEK 2 ASSIGNMENT)

Course: ALY 6020

Name: Yuanying Li

1.INTRODUCTION

Predicting how the stock market will perform is one of the most difficult things to do. There are so many factors involved in the prediction – physical factors vs. physhological, rational and irrational behaviour, etc. All these aspects combine to make share prices volatile and very difficult to predict with a high degree of accuracy. The general research associated with the stock or share market is highly focusing on neither buy nor sell but it fails to address the dimensionality and expectancy of a new investor. The seasonal variance and steady flow of any index will help both existing and naive investors to understand and make a decision to invest in the stock/share market.

To solve these types of problems, the time series analysis will be the best tool for forecasting the trend or even future. Time series analysis will be the best tool for forecasting the trend or even future. The trend chart will provide adequate guidance for the investor.

In this project, we would use Time Series ARIMA model in R and forecast stocks price.

2. DATA AND PREPROCESSING

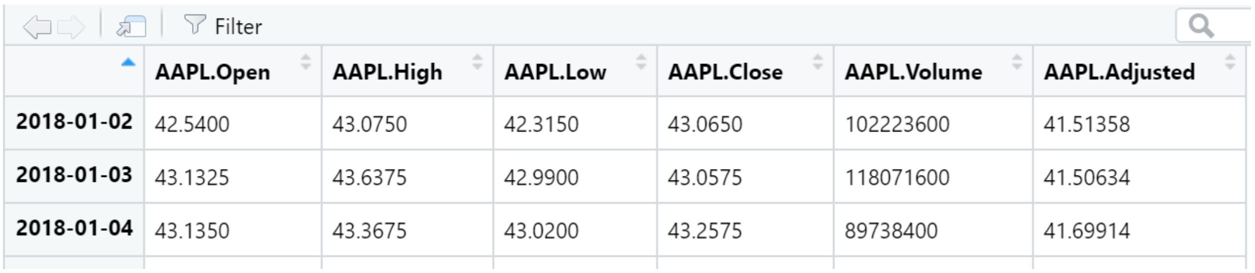
2.1 Exploratory Data Analysis

We will extract all the stock price data from Yahoo Finance, since the IPhone 12 would be release soon, so I choose the stock of APPLE to analyze the trend. Here is the overview of data set from 2018/01/01 – 2020/09/27. Also, we should drop the null data.

Code snippets:

```
# Pull APPLE INC stock price data from Yahoo finance
AAPL <- getSymbols('AAPL', from='2018-01-01', to='2020-09-27',
                   auto.assign = FALSE)
AAPL <- na.omit(AAPL)
View(AAPL)
str(AAPL)
```

Results:



	AAPL.Open	AAPL.High	AAPL.Low	AAPL.Close	AAPL.Volume	AAPL.Adjusted
2018-01-02	42.5400	43.0750	42.3150	43.0650	102223600	41.51358
2018-01-03	43.1325	43.6375	42.9900	43.0575	118071600	41.50634
2018-01-04	43.1350	43.3675	43.0200	43.2575	89738400	41.69914

```
An ‘xts’ object on 2018-01-02/2020-09-25 containing:
  Data: num [1:689, 1:6] 42.5 43.1 43.1 43.4 43.6 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:6] "AAPL.Open" "AAPL.High" "AAPL.Low" "AAPL.Close" ...
  Indexed by objects of class: [Date] TZ: UTC
  xts Attributes:
List of 2
 $ src    : chr "yahoo"
 $ updated: POSIXct[1:1], format: "2020-09-27 19:27:49"
```

Observations:

1. We only extract the date range from 2018 – 2020, so sample size is not that large (within 689), with 6 variables, we have dropped missing value in this dataset.

2. In this dataset, we just would like to keep close price to predict the final price using ARIMA model, so we would drop the rest of columns.

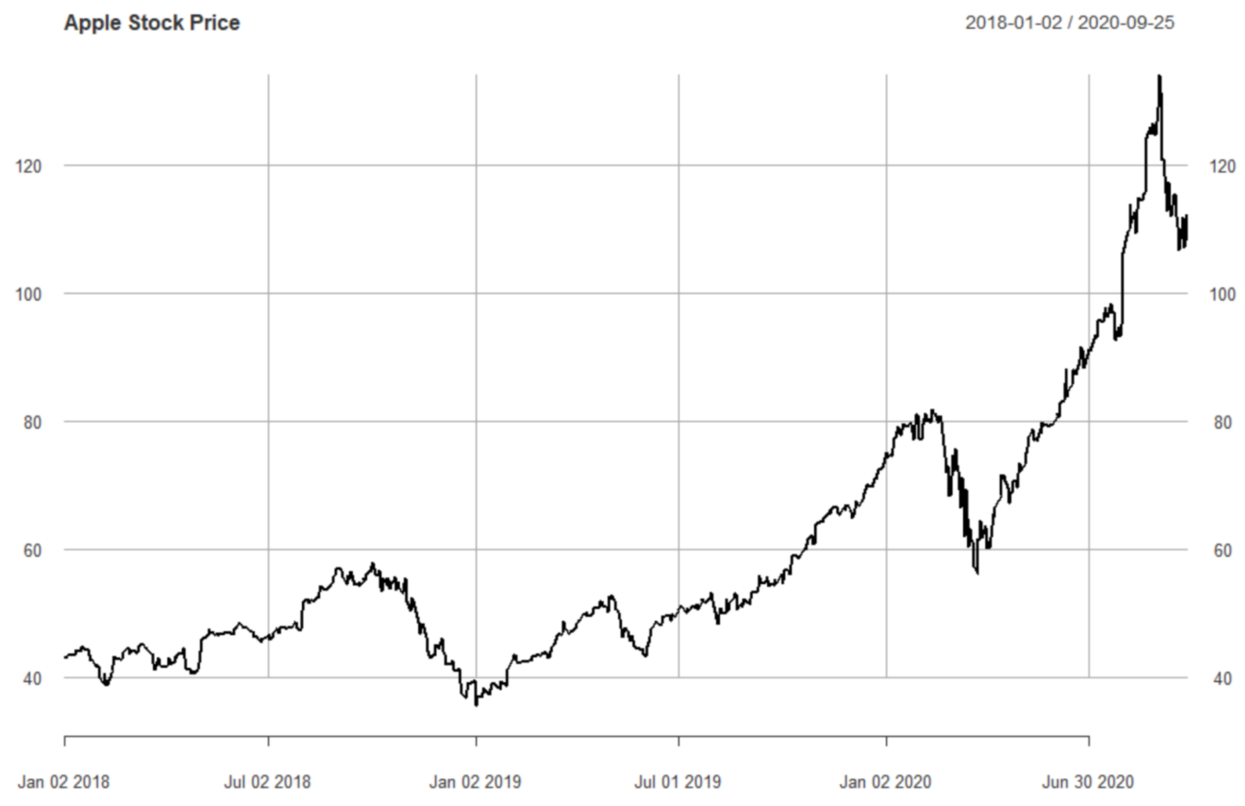
2.2 Data preprocessing

In this study we firstly focus on close price series, as we mentioned before, the others are not necessary to keep for predicting so that we could just drop it.

Code snippets:

```
# Select the relevant close price series and create plot line
stock_prices <- AAPL[,4]
plot(stock_prices, type='l', main='Apple Stock Price')
```

Results:



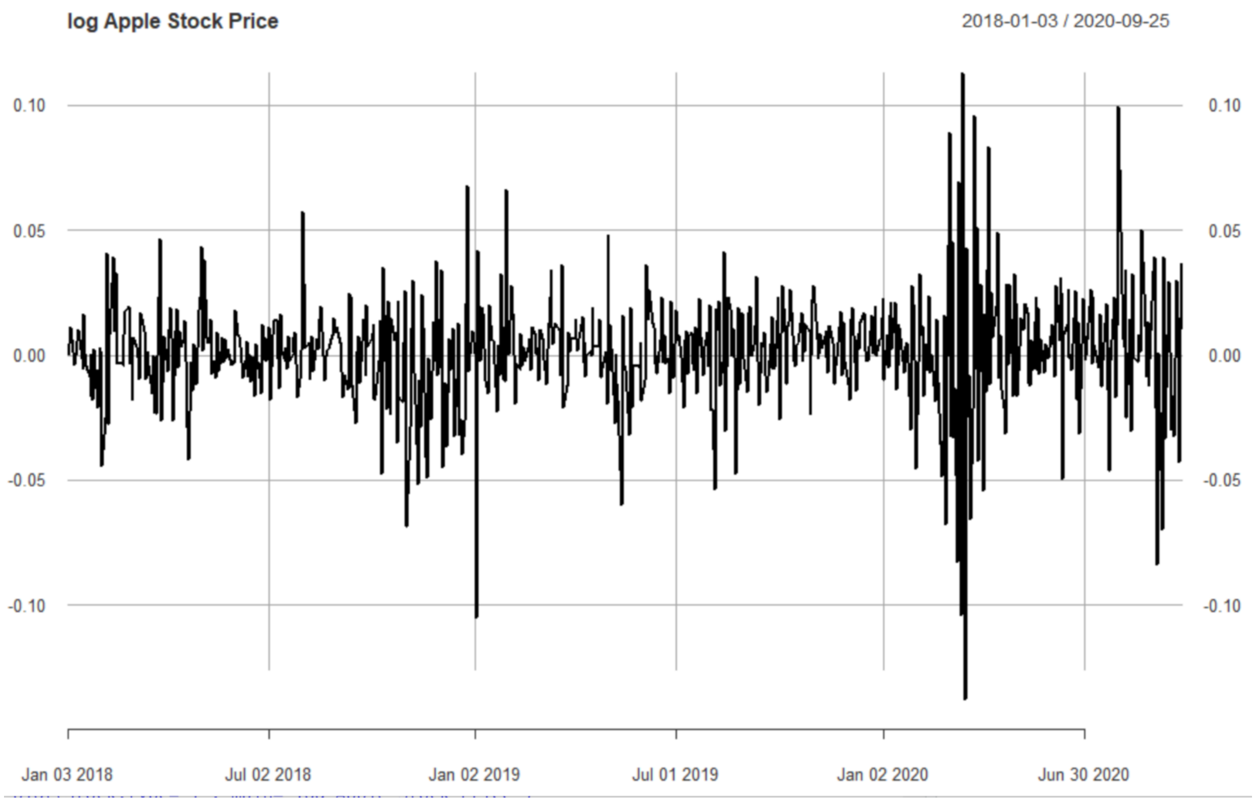
In the this step, we compute the logarithmic returns of the stock as we want the ARIMA model to forecast the log returns and not the stock price. We also plot the log return series using the plot function.

Code snippets:

```
# Compute the log returns for the stock
stock <- diff(log(stock_prices),lag=1)
stock <- stock[!is.na(stock)]
View(stock)

# Plot log returns
plot(stock,type='l', main='log Apple Stock Price')
```

Results:



2.4 Check stationary

Next, we call the ADF test on the returns series data to check for stationarity. The p-value of 0.01 from the ADF test tells us that the series is stationary. If the series were to be non-stationary, we would have first differenced the returns series to make it stationary.

Code snippets:

```
# Conduct ADF test on log returns series
print(adf.test(stock))
```

Result:

```
        Augmented Dickey-Fuller Test

data:  stock
Dickey-Fuller = -7.7797, Lag order = 8, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(stock) : p-value smaller than printed p-value
```

Observation:

1. we could see data set is stationary, so we don't need to differentiate the data set, otherwise, we need to make the data set be stationary.

3.BUILD THE MODEL

3.1 Use ARIMA Model

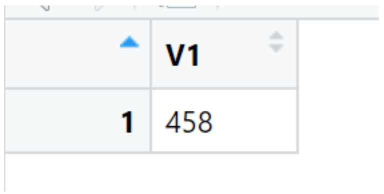
3.1.1 Split train dataset and test train

We fixed a breakpoint which will be used to split the returns dataset in two parts further down the code.

Code snippets:

```
# Split the dataset in two parts - training and testing
breakpoint <- floor(nrow(stock)*(2/3))
View(breakpoint)
```

Result:



	V1
1	458

Observation:

1. so the train data set is from row 1 to 458, the test data set is from row 459 to 687.

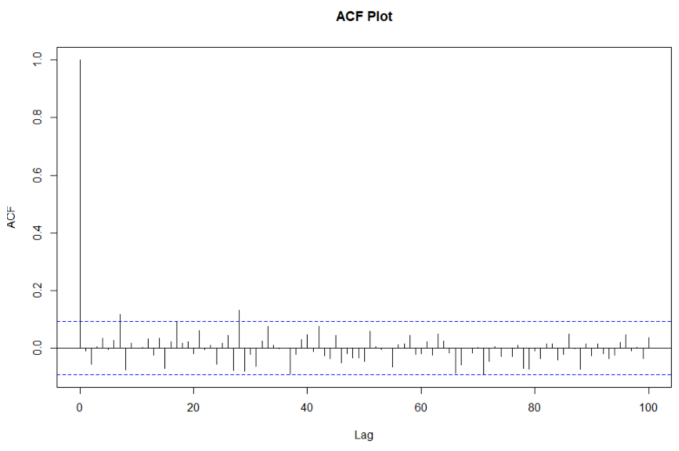
3.2 ACF AND PACF

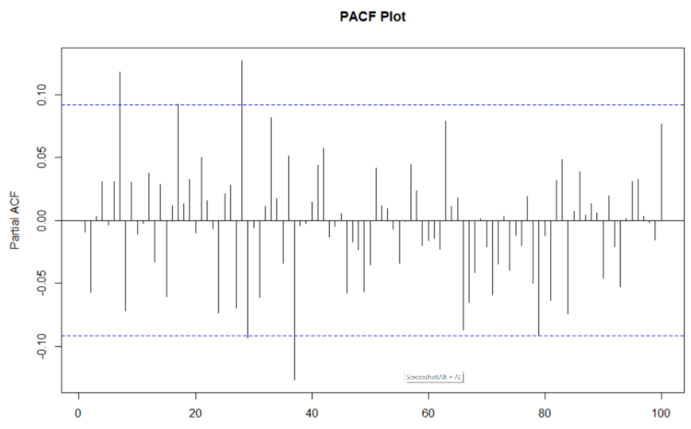
We truncate the original returns series till the breakpoint and call the ACF and PACF functions on this truncated series.

Code snippets:

```
# Apply the ACF and PACF functions
par(mfrow = c(1,1))
acf.stock = acf(stock[c(1:breakpoint),], main='ACF Plot', lag.max=100)
pacf.stock = pacf(stock[c(1:breakpoint),], main='PACF Plot', lag.max=100)
```

Result:





Observation:

1.We can observe these plots and arrive at the Autoregressive (AR) order and Moving Average (MA) order.

2. We know that for AR models, the ACF will dampen exponentially and the PACF plot will be used to identify the order (p) of the AR model. For MA models, the PACF will dampen exponentially and the ACF plot will be used to identify the order (q) of the MA model. From these plots let us select AR order = 2 and MA order = 2. Thus, our ARIMA parameters will be (2,0,2).

3.3 ACF AND PACF

Our objective is to forecast the entire returns series from breakpoint onwards. We will make use of the For Loop statement in R and within this loop, we will forecast returns for each data point from the test dataset.

In the code given below, we first initialize a series which will store the actual returns and another series to store the forecasted returns. In the For Loop, we first form the training dataset and the test dataset based on the dynamic breakpoint.

We call the ARIMA function on the training dataset for which the order specified is (2, 0, 2). We use this fitted model to forecast the next data point by using the forecast.Arima function. The function is set at 99% confidence level. One can use the confidence level argument to enhance the model. We will be using the forecasted point estimate from the model. The “h” argument in the forecast function indicates the number of values that we want to forecast, in this case, the next day returns.

We can use the summary function to confirm the results of the ARIMA model are within acceptable limits. In the last part, we append every forecasted return and the actual return to the forecasted returns series and the actual returns series respectively.

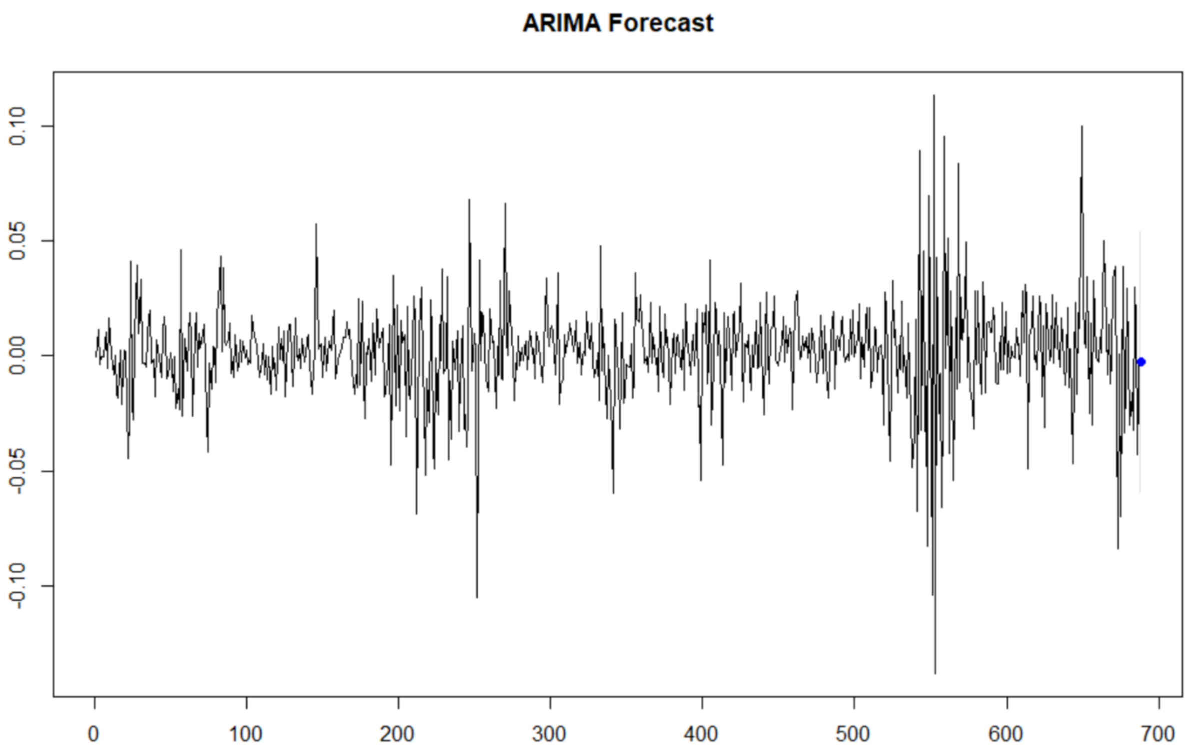
3.3 Train the model

```
# Initialzing a dataframe for the forecasted return series
forecasted_series <- data.frame(Forecasted = numeric())

for (b in breakpoint:(nrow(stock)-1)) {

  stock_train <- stock[1:b, ]
  View(stock_train)
  stock_test <- stock[(b+1):nrow(stock), ]
  View(stock_test)
  # Summary of the ARIMA model using the determined (p,d,q) parameters
  fit <- arima(stock_train, order = c(2, 0, 2),include.mean=FALSE)
  summary(fit)
  # plotting a acf plot of the residuals
  acf(fit$residuals,main="Residuals plot")
  # Forecasting the log returns
  arima.forecast <- forecast(fit, h = 1,level=99)
  summary(arima.forecast)
  # plotting the forecast
  par(mfrow=c(1,1))
  plot(arima.forecast, main = "ARIMA Forecast")
  # Creating a series of forecasted returns for the forecasted period
```

Result:



```
Model Information:

Call:
arima(x = stock_train, order = c(2, 0, 2), include.mean = FALSE)

Coefficients:
         ar1     ar2      ma1     ma2
       -0.07  0.0337  -0.0781  0.0075
s.e.     NaN     NaN      NaN  0.0450

sigma^2 estimated as 0.0004828:  log likelihood = 1648.11,  aic = -3286.22

Error measures:
                      ME       RMSE        MAE  MPE MAPE      MASE         ACF1
Training set 0.001491902 0.02197269 0.01492944 -Inf  Inf 0.6645805 -0.005173922

Forecasts:
    Point Forecast       Lo 99     Hi 99
688   -0.002725989 -0.05932388 0.0538719
           AAPL.Close
2020-09-24     108.22
           AAPL.Close
2020-09-25     112.28
```

Observation:

1. check the results of the ARIMA model for a sample data point from the test dataset.

From the coefficients obtained, the return equation can be written as:

$$\mathbf{Y_t = -0.07}_{(t-1)}\ \mathbf{-}\ \boldsymbol{0.0337}\mathbf{Y}_{(t-2)} \mathbf{- 0.0781\varepsilon}_{(t-1)} \mathbf{+ 0.0075\ \varepsilon}_{(t-2)}$$

2.The standard error is given for the coefficients, and this needs to be within the acceptable limits. The Akaike information criterion (AIC) score is a good indicator of the ARIMA model accuracy. Lower the AIC score better the model. We can also view the ACF plot of the residuals; a good ARIMA model will have its autocorrelations below the threshold limit. The forecasted point return is -0.002725989, which is given in the last row of the output.

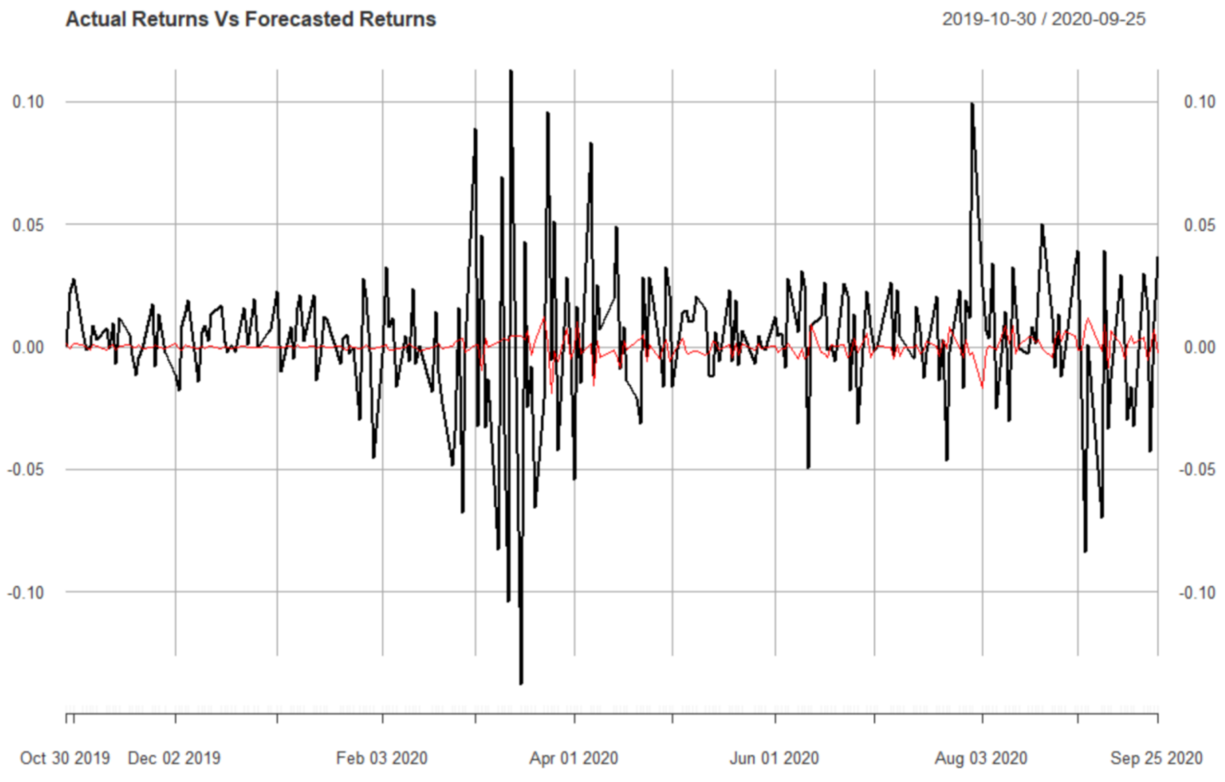
3.5 Evaluation Model

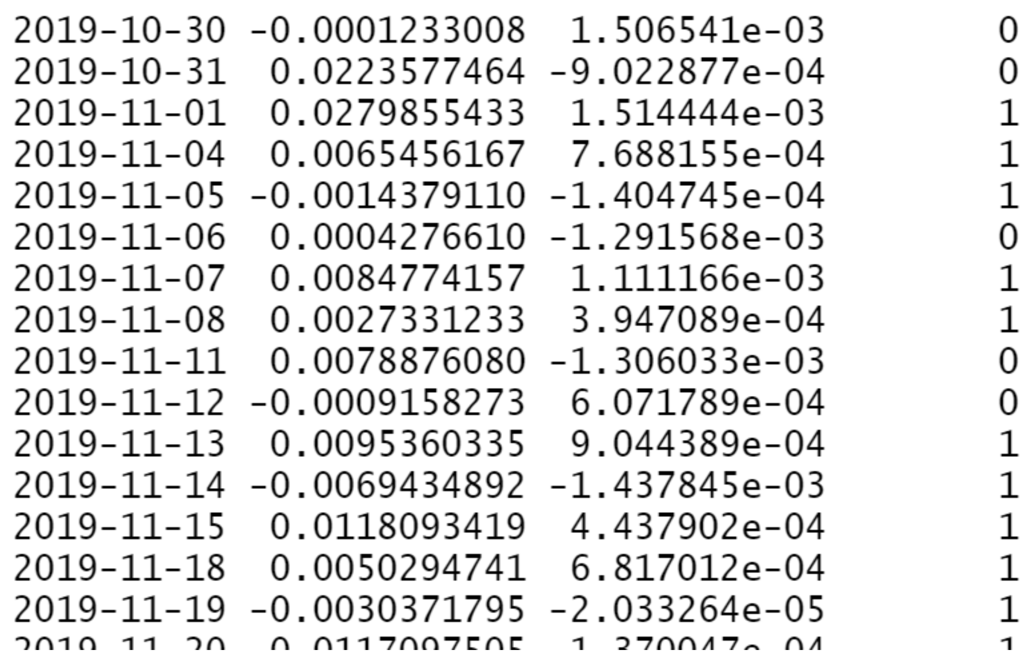
After using a classification, evaluation is one of the important parts to do, Let us check the accuracy of the ARIMA model by comparing the forecasted returns versus the actual returns.

Code snippets:

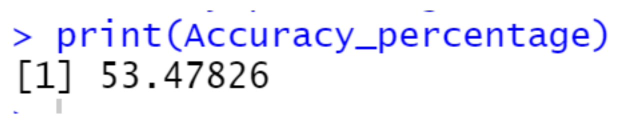
```
# Adjust the length of the Actual return series
Actual_series = Actual_series[-1]
View(Actual_series)
# Create a time series object of the forecasted series
forecasted_series = xts(forecasted_series,index(Actual_series))
# Create a plot of the two return series - Actual versus Forecasted
plot(Actual_series,type='l',main='Actual Returns Vs Forecasted Returns')
lines(forecasted_series,lwd=1.5,col='red')
legend('bottomright',c("Actual","Forecasted"),lty=c(1,1),lwd=c(1.5,1.5),c
# Create a table for the accuracy of the forecast
comparsion = merge(Actual_series,forecasted_series)
comparsion$Accuracy = sign(comparsion$Actual_series)==sign(comparsion$For
print(comparsion)
# Compute the accuracy percentage metric
Accuracy_percentage = sum(comparsion$Accuracy == 1)*100/length(comparsion
print(Accuracy_percentage)
```

Result:





```
2019-10-30 -0.0001233008  1.506541e-03         0
2019-10-31  0.0223577464 -9.022877e-04         0
2019-11-01  0.0279855433  1.514444e-03         1
2019-11-04  0.0065456167  7.688155e-04         1
2019-11-05 -0.0014379110 -1.404745e-04         1
2019-11-06  0.0004276610 -1.291568e-03         0
2019-11-07  0.0084774157  1.111166e-03         1
2019-11-08  0.0027331233  3.947089e-04         1
2019-11-11  0.0078876080 -1.306033e-03         0
2019-11-12 -0.0009158273  6.071789e-04         0
2019-11-13  0.0095360335  9.044389e-04         1
2019-11-14 -0.0069434892 -1.437845e-03         1
2019-11-15  0.0118093419  4.437902e-04         1
2019-11-18  0.0050294741  6.817012e-04         1
2019-11-19 -0.0030371795 -2.033264e-05         1
```



```
> print(Accuracy_percentage)
[1] 53.47826
```

Observation:

1. If the sign of the forecasted return equals the sign of the actual returns we have assigned it a positive accuracy score. In the plot Actual Return VS Forecasted Returns, the red line represent predict values, the black line is true price, in this plot, we could see the trend of prediction is similar to real stock price.

2. The accuracy percentage of the ARIMA model comes to around 53% which looks like a decent number. One can try running the model for other possible combinations of (p,d,q) or instead use the auto.arima function which selects the best optimal parameters to run the ARIMA model.

4.CONCLUSION

In this project, We used Time Series above to predict the trend of stock price and the accuracy of model is 55%, in the future, we might try another model to see whether we can improve the performance and reduce the number of values that have been incorrectly predicted.

5.REFERENCE

1. Time Series - Stock Price Predictions - Part 1

https://www.kaggle.com/viswanathanc/time-series-stock-price-predictions-part-1/comments

2. Candy Production: Time Series Analysis

https://www.kaggle.com/goldens/candy-production-time-series-analysis