Stock Analysis using R

TCS Share

Stock and investments analysis is a theme that can be deeply explored in programming. This includes R language, which already has a big literature, packages and functions developed in this matter. In this post, we'll do a brief introduction to the subject using the packages quantmod and ggplot2.

Preparing the environment

```
rm(list=ls())
install.packages("quantmod")
install.packages("ggplot2")
library(quantmod)
library(ggplot2)
```

Selecting the Stock name and Duration

```
data <- getSymbols("TCS.NS", src = "yahoo", from = "2020-01-01",
                   to = "2021-06-01", auto.assign = FALSE)
data
```

Removing Null values

```
na.omit(data)
```

Price Visualization

Head & Tail Function

With the commands head() and tail() we can see the first and last 6 lines of the base. There are 6 columns with: opening price, maximum and minimum prices, closing price, volume of transactions and adjusted price.

```
head(data)
```

```
> head(data)
           TCS.NS.Open TCS.NS.High TCS.NS.Low TCS.NS.Close TCS.NS.Volume TCS.NS.Adjusted
2020-01-01     2168.00     2183.90    2154.00      2167.60       1354908        2112.478
2020-01-02     2179.95     2179.95    2149.20      2157.65       2380752        2102.780
2020-01-03     2164.00     2223.00    2164.00      2200.65       4655761        2144.687
2020-01-06     2205.00     2225.95    2187.90      2200.45       3023209        2144.492
2020-01-07     2200.50     2214.65    2183.80      2205.85       2429317        2149.755
2020-01-08     2205.00     2260.00    2202.05      2255.25       5197454        2197.898
```

```
tail(data)
```

```
> tail(data)
           TCS.NS.Open TCS.NS.High TCS.NS.Low TCS.NS.Close TCS.NS.Volume TCS.NS.Adjusted
2021-05-24     3081.50     3105.00    3072.00      3081.50       1652260         3066.50
2021-05-25     3092.00     3128.25    3082.10      3114.00       1841613         3114.00
2021-05-26     3120.00     3165.00    3103.80      3158.50       1923753         3158.50
2021-05-27     3161.95     3217.75    3161.80      3180.00       5959785         3180.00
2021-05-28     3189.50     3198.00    3135.65      3143.60       1763701         3143.60
2021-05-31     3150.00     3170.35    3128.60      3159.15       1652799         3159.15
```

Summary Function

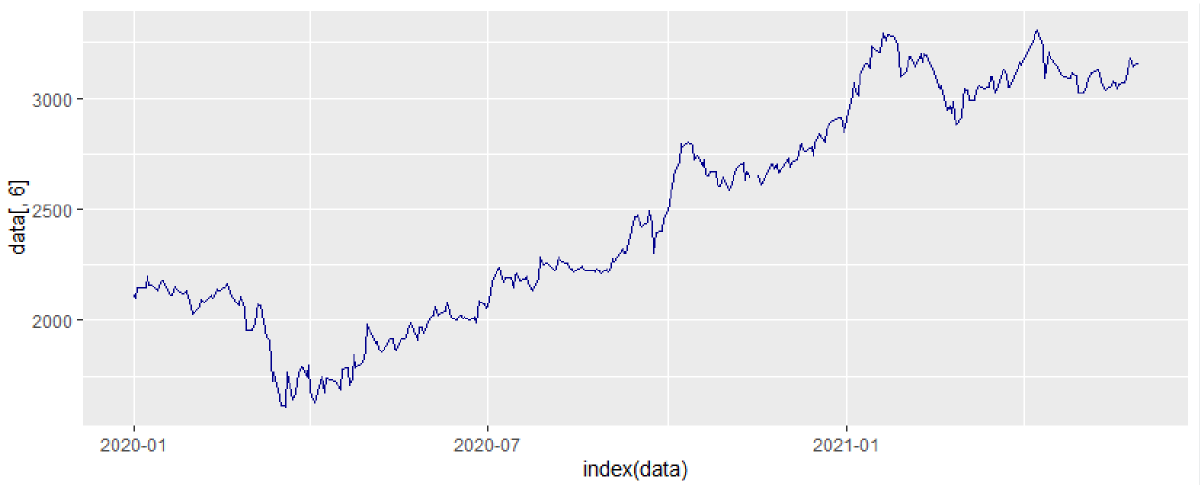
```
summary(data)
```

```
> summary(data)
     Index             TCS.NS.Open    TCS.NS.High     TCS.NS.Low    TCS.NS.Close  TCS.NS.Volume        TCS.NS.Adjusted
 Min.   :2020-01-01   Min.   :1560   Min.   :1685   Min.   :1506   Min.   :1636   Min.   : 1165882   Min.   :1610
 1st Qu.:2020-05-12   1st Qu.:2139   1st Qu.:2163   1st Qu.:2105   1st Qu.:2126   1st Qu.: 2561990   1st Qu.:2082
 Median :2020-09-14   Median :2352   Median :2415   Median :2329   Median :2361   Median : 3201190   Median :2335
 Mean   :2020-09-13   Mean   :2510   Mean   :2542   Mean   :2478   Mean   :2508   Mean   : 3788801   Mean   :2482
 3rd Qu.:2021-01-18   3rd Qu.:3024   3rd Qu.:3066   3rd Qu.:2996   3rd Qu.:3036   3rd Qu.: 4383744   3rd Qu.:3020
 Max.   :2021-05-31   Max.   :3354   Max.   :3354   Max.   :3308   Max.   :3322   Max.   :19839329   Max.   :3306
                      NA's   :1      NA's   :1      NA's   :1      NA's   :1      NA's   :1          NA's   :1
```

Daily Price Graph

Now let's plot daily prices, using the Adjusted Price column, since it incorporates events like splits and dividends distribution, which can affect the series.

```
ggplot(data, aes(x = index(data),
                 y = data[,6])) + geom_line(color = "darkblue")
+ ggtitle("TCS daily prices series") + xlab("Date")
+ ylab("Price") + theme(plot.title = element_text(hjust = 0.5))
+ scale_x_date(date_labels = "%b %y", date_breaks = "6 months")
```



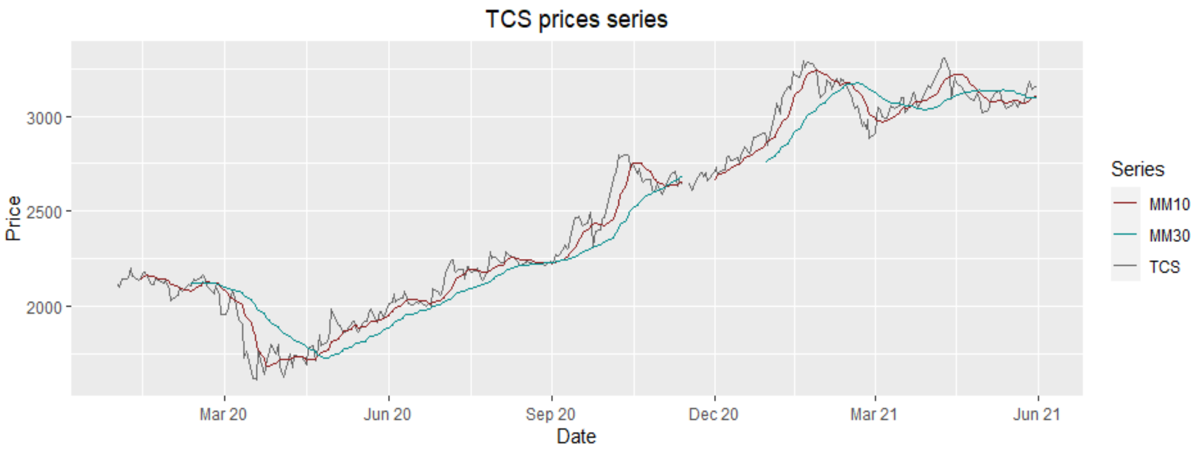
We created this graphic using the command ggplot.

Plotting 10 Day & 30 Day Moving Averages

```r
data_mm <- subset(data, index(data) >= "2020-01-01")

data_mm10 <- rollmean(data_mm[,6], 10, fill = list(NA, NULL, NA), align = "right")
data_mm30 <- rollmean(data_mm[,6], 30, fill = list(NA, NULL, NA), align = "right")

data_mm$mm10 <- coredata(data_mm10)
data_mm$mm30 <- coredata(data_mm30)
```

First we subset the base for data since 2020 using the function subset(). Then, we use the function rollmean(), which takes as argument: the series $(x_t)(x_t)$, in this case the adjusted price; the window of periods $(q)(q)$; an optional fill argument, that is used to complete the days where it's not possible to calculate the moving average, since the enough quantity of days hasn't passed

```r
ggplot(data_mm, aes(x = index(data_mm))) +
  geom_line(aes(y = data_mm[,6], color = "TCS")) + ggtitle("TCS prices series") +
  geom_line(aes(y = data_mm$mm10, color = "MM10")) +
  geom_line(aes(y = data_mm$mm30, color = "MM30")) + xlab("Date") + ylab("Price") +
  theme(plot.title = element_text(hjust = 0.5), panel.border = element_blank()) +
  scale_x_date(date_labels = "%b %y", date_breaks = "3 months") +
  scale_colour_manual("Series", values=c("TCS"="gray40", "MM10"="firebrick4", "MM30"="darkcyan"))
```



To create the graph, we plot the line of prices and the lines of moving averages.

Returns!

We have seen how the stock price has changed over time. Now we'll verify how the stock return has behaved in the same period. To do this, we first need to create a new object with the calculated returns, using the adjusted prices column:

```r
data_ret <- diff(log(data[,6]))
data_ret <- data_ret[-1,]
```

Opening & Closing Price Returns

```
Op(data) # will give returns of opening price
Cl(data) # will give returns of closing price
```

Returns of Different Periods

Another interesting possibility given by `quantmod` is the calculation of returns for different periods. For example, it's possible to calculate the returns by day, week, month, quarter and year, just by using the following commands:

```
dailyReturn(data) #getting daily returns

weeklyReturn(data) #getting weekly returns

monthlyReturn(data) #getting monthly returns

quarterlyReturn(data) #getting quaterly returns

yearlyReturn(data) #getting Yearly returns
```

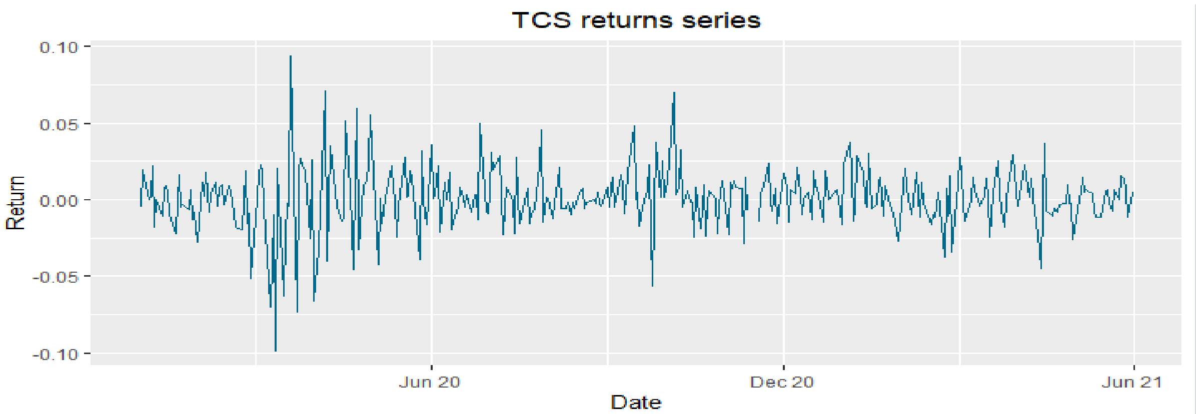
Returns Summary

```
summary(data_ret)
```

```
> summary(data_ret)
     Index               TCS.NS.Adjusted
 Min.   :2020-01-02   Min.   :-0.0988302
 1st Qu.:2020-05-13   1st Qu.:-0.0081572
 Median :2020-09-14   Median : 0.0009266
 Mean   :2020-09-14   Mean   : 0.0011465
 3rd Qu.:2021-01-18   3rd Qu.: 0.0101238
 Max.   :2021-05-31   Max.   : 0.0939009
                      NA's   :2
```

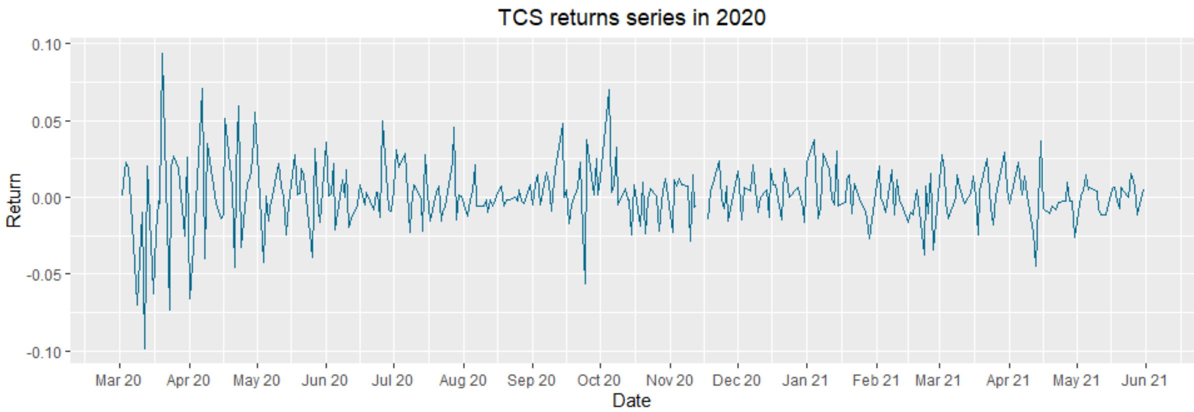
6 Months Returns Graph

```
ggplot(data_ret, aes(x = index(data_ret), y = data_ret)) +
  geom_line(color = "deepskyblue4") +
  ggtitle("TCS returns series") +
  xlab("Date") + ylab("Return") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_x_date(date_labels = "%b %y", date_breaks = "6 months")
```



Now let's take a small look at the stock returns in 2020:

```r
data_ret17 <- subset(data_ret, index(data_ret) > "2020-03-01")

ggplot(data_ret17, aes(x = index(data_ret17), y = data_ret17)) +
  geom_line(color = "deepskyblue4") +
  ggtitle("TCS returns series in 2020") + xlab("Date") + ylab("Return") +
  theme(plot.title = element_text(hjust = 0.5)) + scale_x_date(date_labels = "%b %y", date_breaks = "1 months")
```



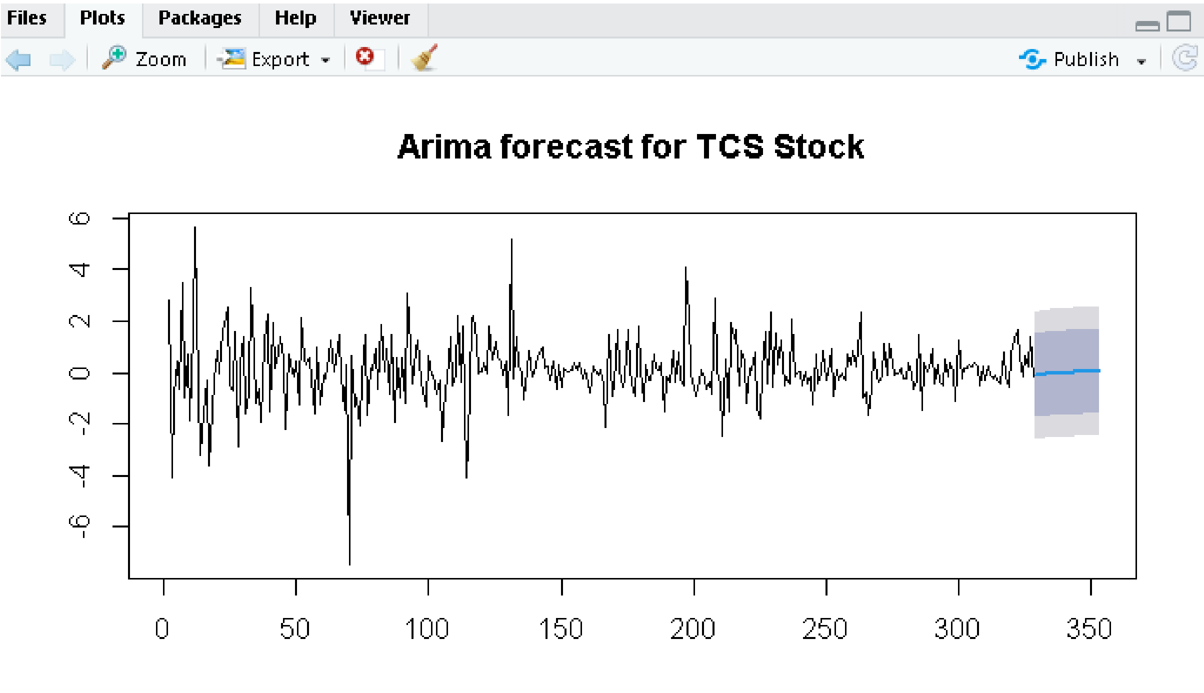
Arima Model For Prediction

adf.test() is the function that allows us to perform the **Augmented Dicky-Fuller test**.

It is used to adjust the randomness present in our time series data.ARIMA stands for auto-regressive integrated moving average and is specified by three order parameters, which are:- d, p, q.

```r
chartSeries(TCS.NS, subset = 'last 12 months', type = 1)
addBBands()
library(tseries, quietly = T)
adf.test(data$TCS.NS.Adjusted)

ret_TCS.NS <- 100*diff(log(TCS.NS$TCS.NS.Adjusted[2274:2638]))

library(forecast, quietly = T)

TCS.NS_ret_train <- ret_TCS.NS[1:(0.9*length(ret_TCS.NS))]

TCS.NS_ret_test <- ret_TCS.NS[(0.9*length(ret_TCS.NS)+1):length(ret_TCS.NS)]

fit <- Arima(TCS.NS_ret_train, order = c(2,0,2))

preds <- predict(fit, n.ahead = (length(ret_TCS.NS) - (0.9*length(ret_TCS.NS))))$pred

test_forecast <- forecast(fit,h = 25)

plot(test_forecast, main = "Arima forecast for TCS Stock")
```

Prediction Graph



Calculating Accuracy

```
accuracy(preds, TCS.NS_ret_test)
```

```
                 ME     RMSE       MAE      MPE     MAPE
Test set 0.03784516 1.006537 0.7222298 111.3656 111.3656
>
```