Algorithmic Trading with Neural Networks

Tanishq Chauhan

Algorithmic Trading with Neural Networks

Given the coronavirus pandemic and the market volatily, there have been losses of billions of dollars worldwide. Where other services have come to a halt due to the imposed lockdowns, the pharmaceutical companies have gained traction in the race to develop the vaccine (and the cure as well) for the coronavirus. We attempt to model the reaction of the pharmaceutical firms to the coronavirus pandemic. Overall, we have a simple index holding strategy where we could theoretically invest in a portfolo rebalancing strategy for the DRG index. Such an exercise would however be out of scope for this project, and hence would be left for future discussions.

We do this with the following data:

Index: DRG (NYSR ARCA DRG)

Timeframe: '2020-01-01' to '2020-04-24'

Comparison Index: DJI

First we load the necessary libraries for our computation.

```r
library(quantmod)
```

```
## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: TTR

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```r
library(zoo)
library(stats)
library(tseries)
library(forecast)
library(fGarch)
```

```
## Loading required package: timeDate

## Loading required package: timeSeries

##
## Attaching package: 'timeSeries'

## The following object is masked from 'package:zoo':
##
##     time<-

## Loading required package: fBasics

##
## Attaching package: 'fBasics'

## The following object is masked from 'package:TTR':
##
##     volatility
```

```r
library(rugarch)
```

```
## Loading required package: parallel

##
## Attaching package: 'rugarch'

## The following object is masked from 'package:stats':
##
##     sigma
```

```r
library(nnet)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:timeSeries':
##
##     filter, lag

## The following objects are masked from 'package:xts':
##
##     first, last

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggfortify)
```

```
## Loading required package: ggplot2

## Registered S3 methods overwritten by 'ggfortify':
##   method                 from
##   autoplot.Arima         forecast
##   autoplot.acf           forecast
##   autoplot.ar            forecast
##   autoplot.bats          forecast
```

```
##   autoplot.decomposed.ts forecast
##   autoplot.ets           forecast
##   autoplot.forecast      forecast
##   autoplot.stl           forecast
##   autoplot.ts            forecast
##   fitted.ar              forecast
##   fortify.ts             forecast
##   residuals.ar           forecast
```

```
library(magrittr)
library(PerformanceAnalytics)
```

```
##
## Attaching package: 'PerformanceAnalytics'

## The following objects are masked from 'package:timeDate':
##
##     kurtosis, skewness

## The following object is masked from 'package:graphics':
##
##     legend
```

```
library()
```

We then import DRG index data from yahoo (please note that Yahoo has been inconsistent in providing appropriate data, hence Quandl or Tiingo would be much more viable for future computations). In addition, we import the DJI (Dow Jones Index) data to inspect the drawdown on the indices.

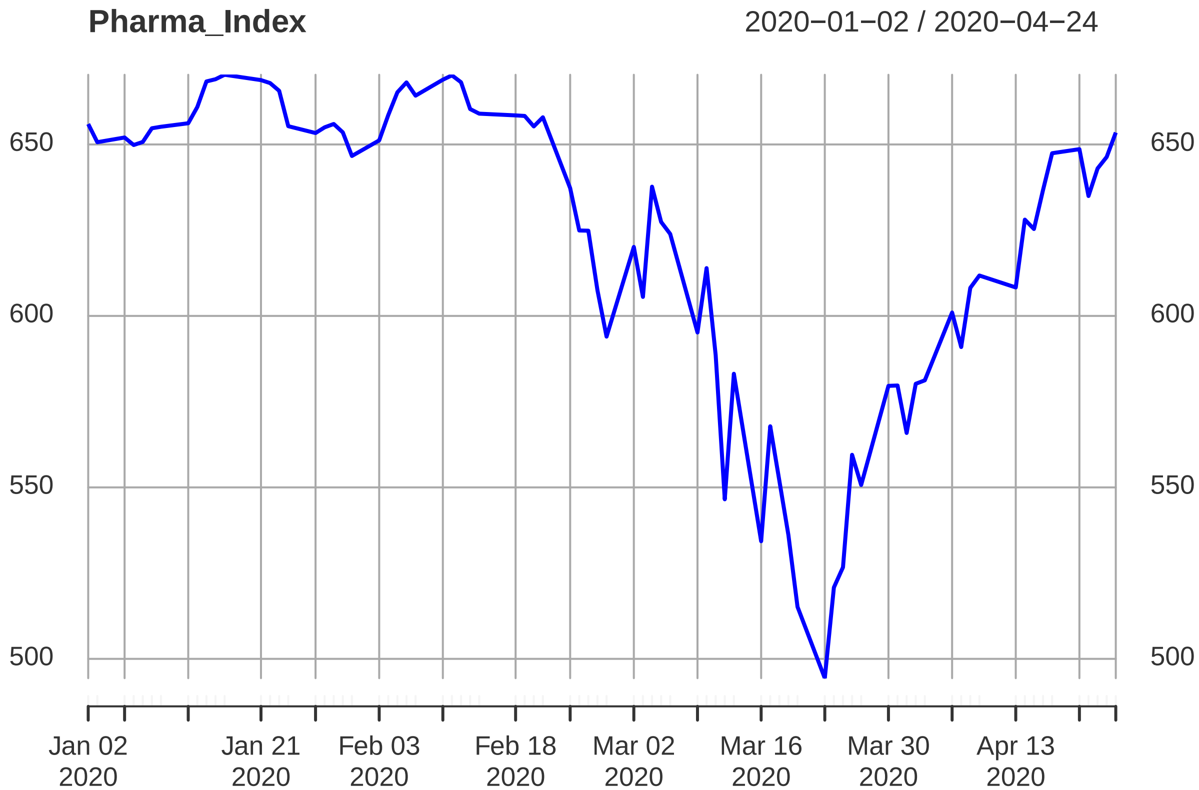
```
#DRG STANDS FOR NYSE ARCA PHARACEUTICAL INDEX
#Loading DRG data from Yahoo Finance
getSymbols("^DRG",src="yahoo", from = as.Date("2020-01-01"), to = as.Date("2020-04-25"),warnings=FALSE)
```

```
## '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: 'indexClass<-' is deprecated.
## Use 'tclass<-' instead.
## See help("Deprecated") and help("xts-deprecated").

## [1] "^DRG"
```

```
#Extracting DRG from Yahoo Finance - na.locf() can alter the matrix results
Pharma_Index<- (DRG[,"DRG.Close"])
head(Pharma_Index)
```

```
##            DRG.Close
## 2020-01-02    655.95
## 2020-01-03    650.66
## 2020-01-06    652.03
## 2020-01-07    649.83
## 2020-01-08    650.73
## 2020-01-09    654.72
```

```r
plot(Pharma_Index, col = "blue")
```




```r
#SPY STANDS FOR NYSE ARCA PHARACEUTICAL INDEX
#Loading SPY data from Yahoo Finance
getSymbols("^DJI",src="yahoo", from = as.Date("2020-01-01"), to = as.Date("2020-04-24"),warnings=FALSE)
```

```
## Warning: 'indexClass<-' is deprecated.
## Use 'tclass<-' instead.
## See help("Deprecated") and help("xts-deprecated").
```

```
## [1] "^DJI"
```

```r
#Extracting DJI from Yahoo Finance - na.locf() can alter the matrix results
DJIndex<- (DJI[,"DJI.Close"])
head(DJIndex)
```

```
##            DJI.Close
## 2020-01-02  28868.80
## 2020-01-03  28634.88
## 2020-01-06  28703.38
## 2020-01-07  28583.68
## 2020-01-08  28745.09
## 2020-01-09  28956.90
```

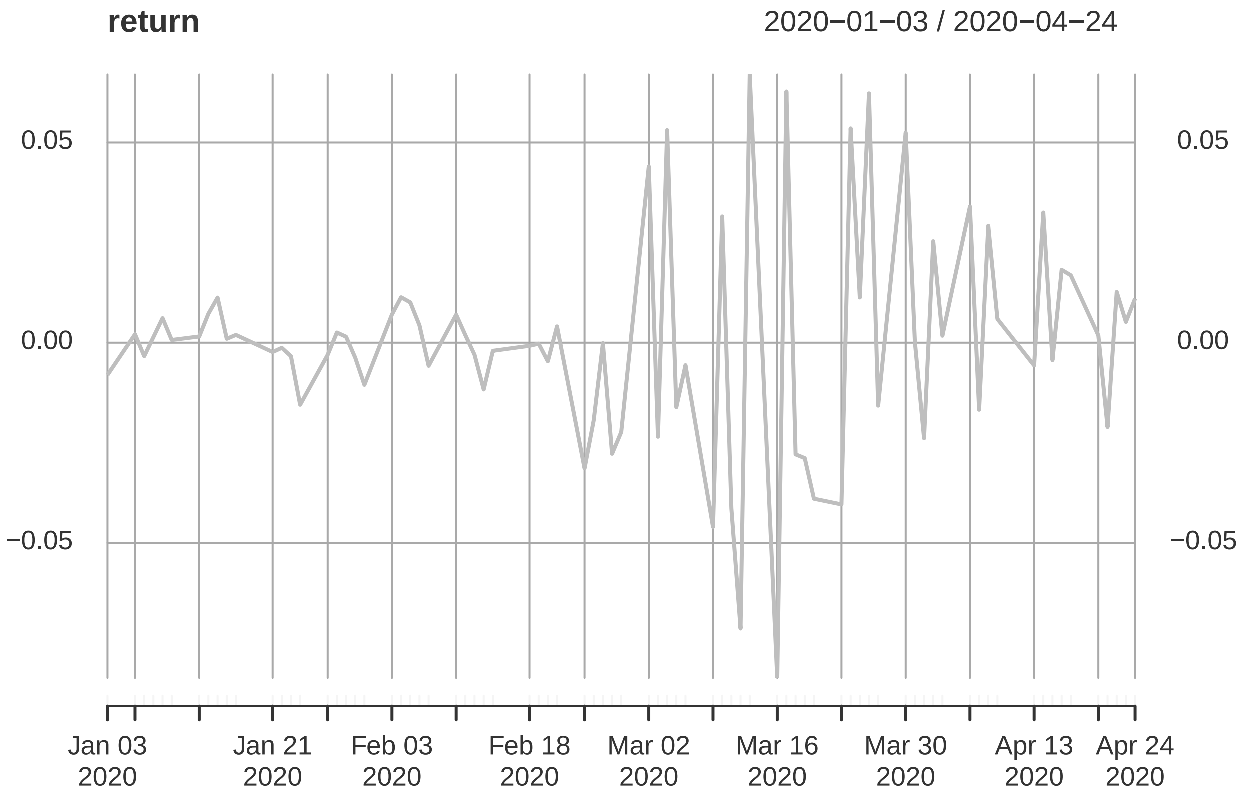
```r
plot(DJIndex, col = "blue")
```

Above, we can visually compare the market drawdown on the DRG and the DJI (Down Jones Index). The difference in the market capitalization of the two index is not the concern of this exercise, rather it is the quick recovery from the pandemic drawdown. Which is obvious as the DJI is an industrial average index and the DRG is a dedicated index comprising of pharmaceutical companies. According to the NYSE, "The NYSE Arca Pharmaceutical Index (DRG) is designed to represent a cross section of widely held, highly capitalized companies involved in various phases of the development, production, and marketing of pharmaceuticals. This index contains stocks and ADRs of some of the most highly capitalized companies in the pharmaceutical industry". The reason for doing so is to observe whether the trained Artificial Neural Network (ANN) we create are able to gauge any market inconsistency and any possibility for future rise

(NOTE: As of 2020-04-27, the DRG index has already recovered and is stable. This makes the exercise interesting as we can see whether the neural model can be trained to give accurate results for the given historical timeframe). For the DRG data, we can see there is already a major recovery (given that pharmaceutical sector is essential right now).

##Calulating technical indicators

```
#Implementing return
return <- Delt(Pharma_Index)
rows = nrow(return)
return <- return[2:rows]
plot(return, col = "grey")
```




```
#Implementing moving average
average10<- rollapply(Pharma_Index,10,mean)
#plot(average10)
average20<- rollapply(Pharma_Index,20,mean)
#plot(average20)
```

```
#Implementing standard deviation
std10<- rollapply(Pharma_Index,10,sd)
#plot(std10)
std20<- rollapply(Pharma_Index,20,sd)
#plot(std20)
```

```
#Implement RSI
rsi5<- RSI(Pharma_Index,5,"SMA")
#plot(rsi5)
rsi14<- RSI(Pharma_Index,14,"SMA")
#plot(rsi14)
```

```
#Implement MACD
macd12269<- MACD(Pharma_Index,12,26,9,"SMA")
#plot(macd12269)
macd7205<- MACD(Pharma_Index,7,20,5,"SMA")
#plot(macd7205)
```

```
#Implement Bollinger Bands
bollinger_bands<- BBands(Pharma_Index,20,"SMA",2)
#plot(bollinger_bands)
```

Generate directions: 1. Return over the last 20 days > 2 % -> Up direction 2. Return over the last 20 days is between -2 % and 2 % -> Nowhere 3. Return over the last 20 days < - 2 % -> Down direction

```r
#Generate a data frame named direction which consists of NA and a
#number of rows the same as the number of rows in Pharma_Index and one column
direction<- data.frame(matrix(NA,dim(Pharma_Index)[1],1) )
#Calculate the return over the last 20 days (20 is not a fixed value
#in quantitative finance. You can chose any value you consider)
lagreturn<- (Pharma_Index - Lag(Pharma_Index,20)) / Lag(Pharma_Index,20)
#Indicate Up, Down and Nowhere directions
direction[lagreturn> 0.02] <- "Up"
direction[lagreturn< -0.02] <- "Down"
direction[lagreturn< 0.02 &lagreturn> - 0.02] <- "NoWhere"
```

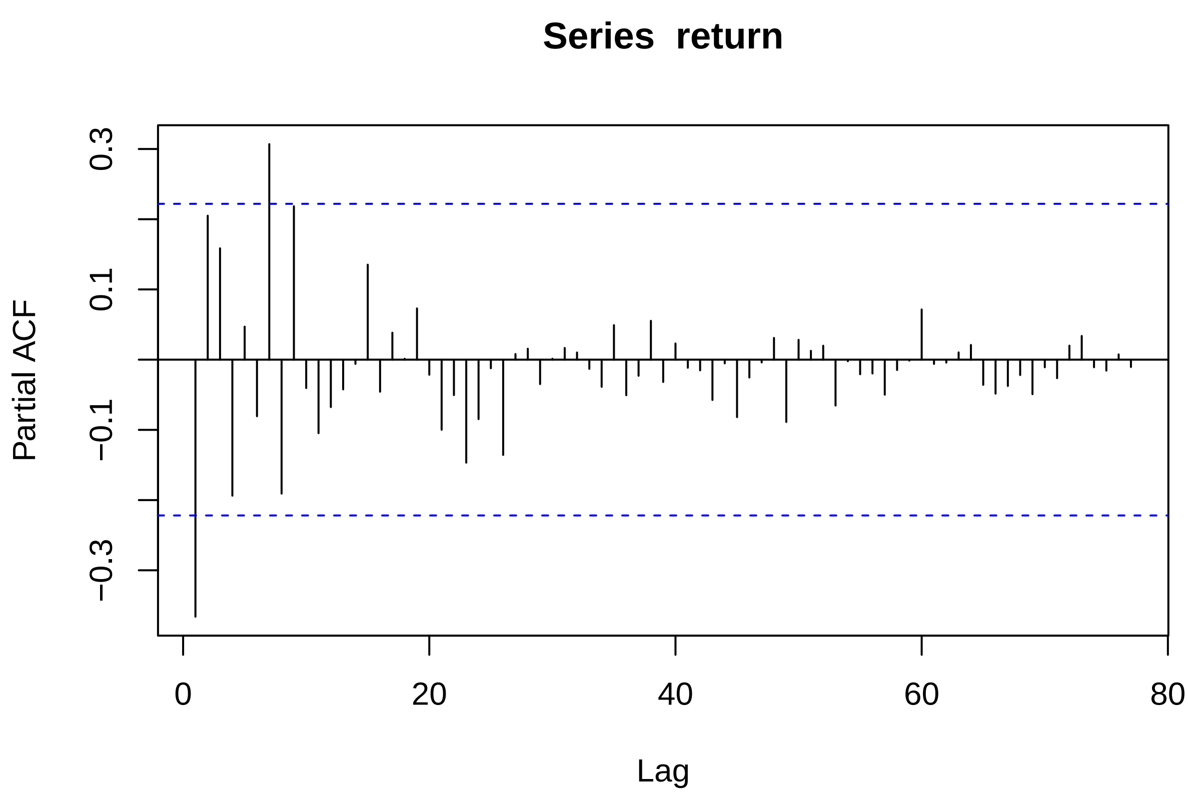
We perform a preliminary Dickey-Fuller test for the stationarity of the returns.

```r
#______________________________________________________________________________________________________
# BEGIN DETERMINING PARAMETERS OF THE CHOSEN MODE (say GARCH)
#First we determine stationarity using the ADF test
adf.test(return)
```

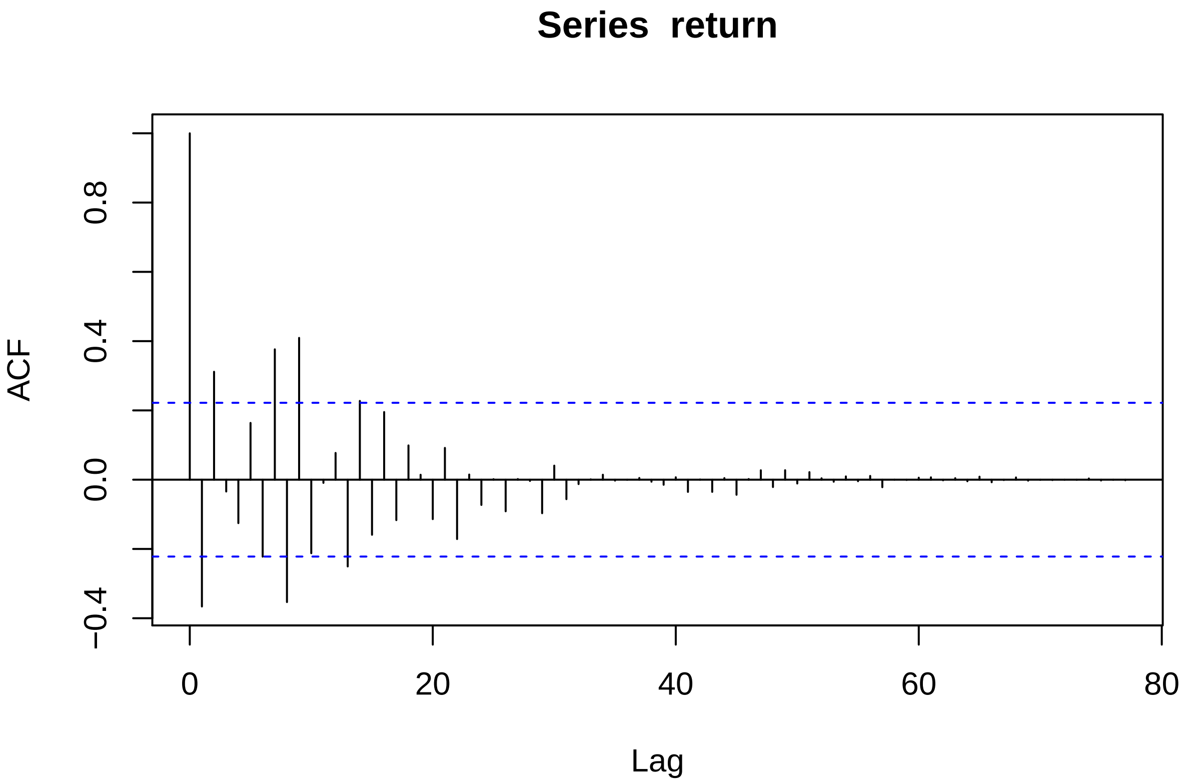
```
##
##  Augmented Dickey-Fuller Test
##
## data:  return
## Dickey-Fuller = -3.7372, Lag order = 4, p-value = 0.02716
## alternative hypothesis: stationary
```

We can see that our data is stationary. To determine the order of the model, let us plot the ACF and the PACF plots for the returns.

```r
pacf(return, lag.max = 150)
```



```r
acf(return, lag.max = 100)
```



We use the `auto.arima` function to let the inbuild R package pick the appropriate model given the historical data.

```r
auto.arima(return)
```

```
## Series: return
## ARIMA(0,0,2) with zero mean
##
## Coefficients:
##           ma1     ma2
##       -0.2511  0.3854
## s.e.   0.1042  0.1093
##
## sigma^2 estimated as 0.0005853:  log likelihood=180.45
## AIC=-354.9   AICc=-354.58   BIC=-347.83
```

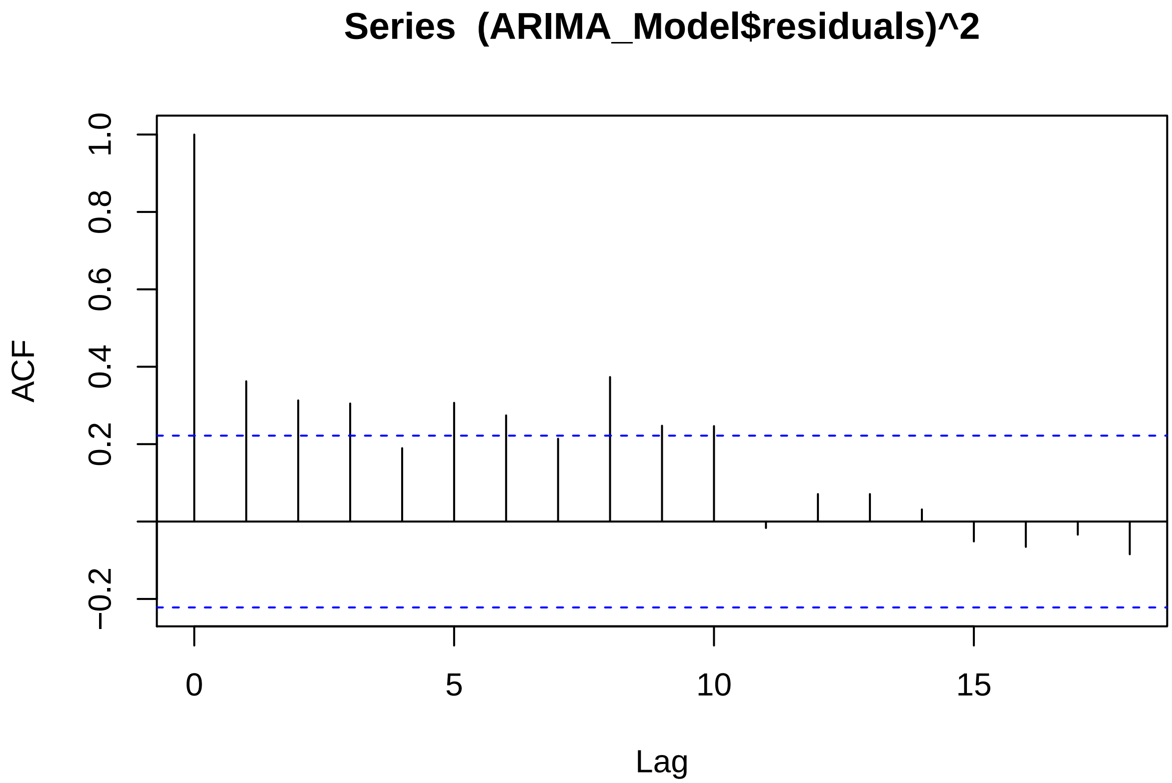
Let us construct an ARIMA model with the based on the `auto.arima` determined values

```r
lengthOfReturns<-length(return)
timeseries <- ts(return)
ARIMA_Model <- arima(window(timeseries,1,lengthOfReturns), order=c(2,0,1), method = "ML")
summary(ARIMA_Model)
```

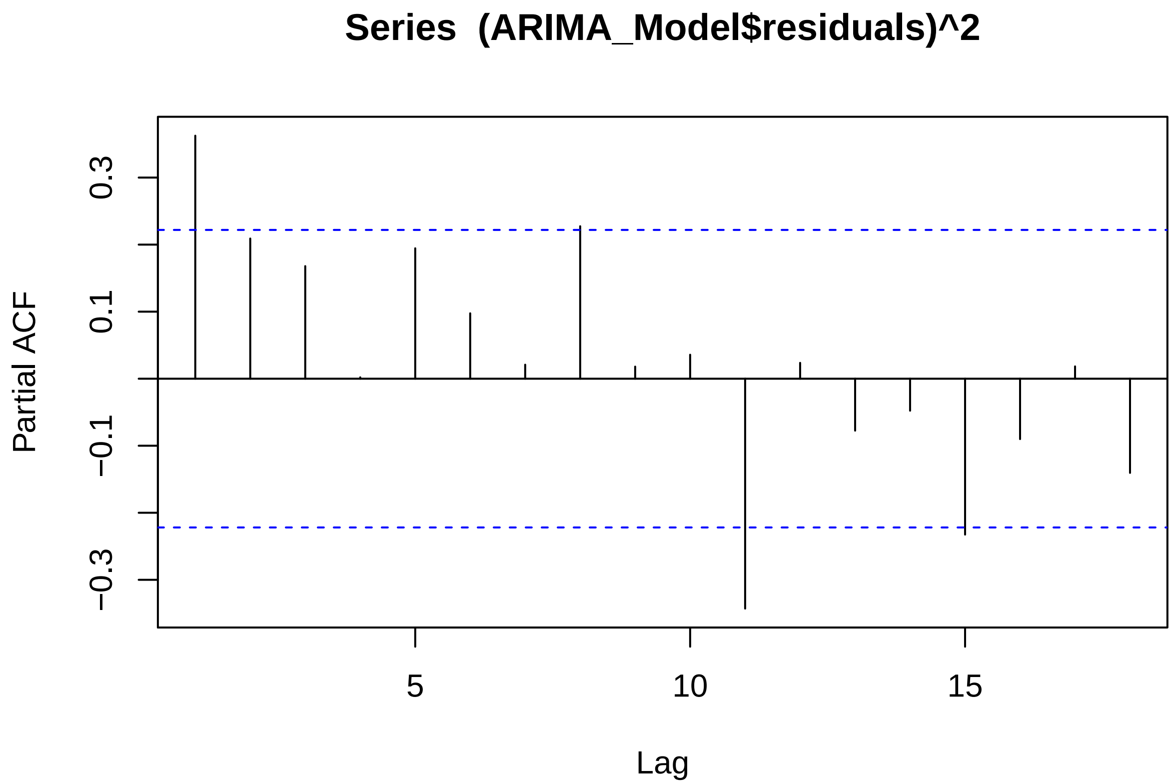
```
##
## Call:
## arima(x = window(timeseries, 1, lengthOfReturns), order = c(2, 0, 1), method = "ML")
##
## Coefficients:
##           ar1     ar2      ma1  intercept
```

```
##       -0.0290  0.3046  -0.2681     0.0003
## s.e.   0.3144  0.1398   0.3189     0.0028
##
## sigma^2 estimated as 0.0005848:  log likelihood = 179.52,  aic = -349.05
##
## Training set error measures:
##                         ME       RMSE        MAE      MPE     MAPE      MASE
## Training set -3.047437e-06 0.02418327 0.01680396 199.4112 233.0118 0.5940237
##                     ACF1
## Training set -0.01813885
```

```r
acf((ARIMA_Model$residuals)^2)
```




```r
pacf((ARIMA_Model$residuals)^2)
```

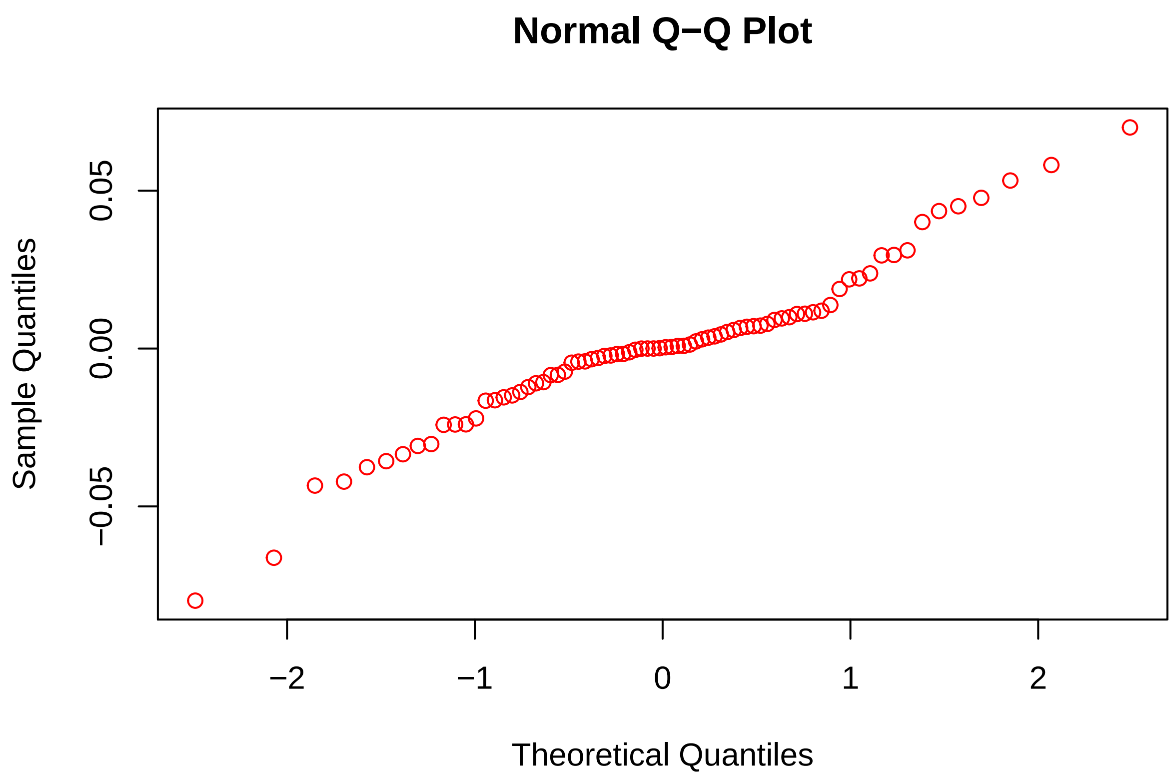



```
model <- garchFit(formula = ~ arma(2,0) + garch(1,1) , data = timeseries, trace = F)
summary(model)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = ~arma(2, 0) + garch(1, 1), data = timeseries,
##     trace = F)
##
## Mean and Variance Equation:
##  data ~ arma(2, 0) + garch(1, 1)
## <environment: 0x7fa59fe27178>
##  [data = timeseries]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##          mu          ar1          ar2        omega       alpha1        beta1
## -3.2656e-05  -6.4603e-02   1.8382e-01   2.9153e-05   4.2409e-01   5.4139e-01
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##          Estimate  Std. Error  t value Pr(>|t|)
## mu     -3.266e-05   1.710e-03   -0.019 0.984766
```

```
## ar1    -6.460e-02   1.108e-01   -0.583 0.559782
## ar2     1.838e-01   9.853e-02    1.866 0.062099 .
## omega   2.915e-05   1.451e-05    2.009 0.044534 *
## alpha1  4.241e-01   1.949e-01    2.176 0.029573 *
## beta1   5.414e-01   1.542e-01    3.510 0.000448 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  202.6473    normalized:  2.598042
##
## Description:
##  Thu Apr 30 09:29:53 2020 by user:
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  7.524374  0.02323287
##  Shapiro-Wilk Test  R    W      0.9772681 0.1767051
##  Ljung-Box Test     R    Q(10)  9.650959  0.4716307
##  Ljung-Box Test     R    Q(15)  13.6906   0.5491126
##  Ljung-Box Test     R    Q(20)  17.19782  0.6400937
##  Ljung-Box Test     R^2  Q(10)  8.285409  0.6009804
##  Ljung-Box Test     R^2  Q(15)  16.8321   0.3290002
##  Ljung-Box Test     R^2  Q(20)  21.64866  0.3598754
##  LM Arch Test       R    TR^2   6.185186  0.9064604
##
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -5.042237 -4.860952 -5.052983 -4.969665
```

```r
res = residuals(model)
qqnorm(res, col = "red")
```




```
garch11_spec <- ugarchspec(variance.model = list(garchOrder = c(1, 1)),mean.model = list(armaOrder = c(2
garch11_fit<-ugarchfit(spec=garch11_spec,solver.control = list(tol = 1e-12), data=timeseries)
```

```
## Warning in .sgarchfit(spec = spec, data = data, out.sample = out.sample, :
## ugarchfit-->waring: using less than 100 data
##  points for estimation
```

```
garch11_fit
```

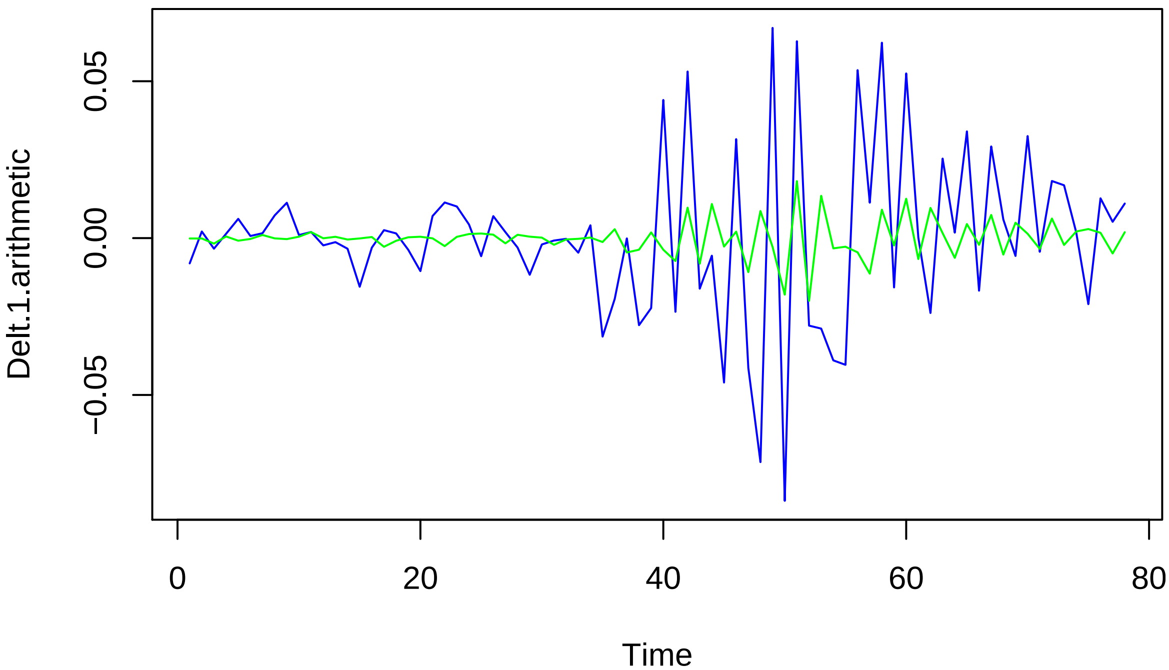
```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(2,0,0)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error   t value Pr(>|t|)
## mu     -0.000123    0.001952 -0.062916 0.949833
## ar1    -0.069609    0.111182 -0.626086 0.531258
## ar2     0.185604    0.099177  1.871441 0.061284
## omega   0.000030    0.000015  1.926201 0.054079
## alpha1  0.437952    0.207732  2.108251 0.035009
## beta1   0.544141    0.158898  3.424478 0.000616
##
```

```
## Robust Standard Errors:
##         Estimate  Std. Error   t value Pr(>|t|)
## mu     -0.000123    0.002304 -0.053312 0.957484
## ar1    -0.069609    0.091005 -0.764895 0.444334
## ar2     0.185604    0.064770  2.865576 0.004163
## omega   0.000030    0.000015  1.937875 0.052638
## alpha1  0.437952    0.195946  2.235062 0.025413
## beta1   0.544141    0.145729  3.733937 0.000189
##
## LogLikelihood : 201.6495
##
## Information Criteria
## ------------------------------------
##
## Akaike       -5.0167
## Bayes        -4.8354
## Shibata      -5.0274
## Hannan-Quinn -4.9441
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.0871  0.7679
## Lag[2*(p+q)+(p+q)-1][5]    0.4929  1.0000
## Lag[4*(p+q)+(p+q)-1][9]    1.8847  0.9889
## d.o.f=2
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                      0.216  0.6421
## Lag[2*(p+q)+(p+q)-1][5]     1.403  0.7639
## Lag[4*(p+q)+(p+q)-1][9]     3.120  0.7390
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]     0.338 0.500 2.000  0.5610
## ARCH Lag[5]     2.777 1.440 1.667  0.3238
## ARCH Lag[7]     3.349 2.315 1.543  0.4502
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  1.9186
## Individual Statistics:
## mu     0.19024
## ar1    0.08213
## ar2    0.03265
## omega  0.12416
## alpha1 0.14583
## beta1  0.32286
##
```

```
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.49 1.68 2.12
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value   prob sig
## Sign Bias           0.2312 0.8178
## Negative Sign Bias  0.6278 0.5321
## Positive Sign Bias  0.8613 0.3919
## Joint Effect        2.9049 0.4065
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     13.28       0.8238
## 2    30     23.54       0.7513
## 3    40     25.59       0.9516
## 4    50     34.82       0.9371
##
##
## Elapsed time : 0.105211
```

```
GFIT<- garch11_fit@fit$fitted.values
```

```
plot(timeseries, type="l", col="blue") + lines(garch11_fit@fit$fitted.values, col="green")
```




```
## integer(0)
```

```
#binding closing price and technical analysis indicators into a variable Pharma_Index_Bind

Pharma_Index_Bind <- cbind(Pharma_Index[2:nrow(Pharma_Index)], average10[2:nrow(average10)], average20[2
```

```
#integrate GARCH model rolling window prediction output into variable
Pharma_Index_model <- cbind(Pharma_Index_Bind,garch11_fit@fit$fitted.values)
```

Including Google trends data into the GARCH model

From the article "Quantifying Trading Behavior in Financial Markets Using Google Trends" it is showed that we can trade efficiently if we take into consideration the frequency of certain key words in Google searches. *[T. Preis, H. S. Moat & H.E. Stanley. (2013)]*

Inline with our topic of interest, we download google trends data regarding the term "Coronavirus". The timeframe of this data will be the same as our pharamceutical index data. We bind this trends data with our above `Pharma_Index_Model` and analyze further.

```
#______________________________________________________________________________________________________
#ADD GOOGLE TREND DATA HERE AND INTEGRATE INTO Pharma_Index using cbind
#Creating Pharma_Index dataset using cbind function
#Closing price and indicators are put into one variable called Pharma_Index
#For adjusting the trend dates with those of the Market trading dates, we extracted the Dates from colu
#write.csv(as.data.frame(DATAFRAMENAME[,1]), "PATHNAME\\Filename.csv", row.names = TRUE)

coronavirus_trend <- read.csv("Coronavirus.csv", header=F)$V2
#Bind with Pharma_Index_Model
Pharma_Index_coronavirus <- na.locf(cbind(Pharma_Index_model$DRG.Close ,coronavirus_trend))
```

Dividing neural network dataset in three parts: 1. Training dataset -> training the neural network 2. Validating dataset -> validating the estimated parameters 3. Testing dataset -> measure the accuracy of the prediction

The choice of timeperiods is crucial here as we would like the see the drawdown impact of the coronavirus pandemic and whether the model can forecast any reasonable scenario. Given the coronavirus trends data, we know that the first instance of coronavirus Google searches were at the start of the january 2020. So our timeframe should be from `2020-01-01` to `2020-04-14`

```
#Indicate end and start dates for train, validating and testing period
train_sdate<- "2020-01-03"
train_edate<- "2020-04-23"
vali_sdate<- "2020-01-03"
vali_edate<- "2020-04-23"
test_sdate<- "2020-01-03"
test_edate<- "2020-04-23"
```

Constructing date ranges for the three datasets

```
#Generate row numbers where the date is greater than and equal to
#start date and less than equal to the end date. Which() function is used
trainrow<- which(index(Pharma_Index) >= train_sdate& index(Pharma_Index) <= train_edate)
valirow<- which(index(Pharma_Index) >= vali_sdate& index(Pharma_Index) <= vali_edate)
testrow<- which(index(Pharma_Index) >= test_sdate& index(Pharma_Index) <= test_edate)
```

Extract data for training, validating and testing periods

```
#Extracting data for training, validating and testing periods
trainDRG<- Pharma_Index[trainrow,]
valiDRG<- Pharma_Index[valirow,]
testDRG<- Pharma_Index[testrow,]
```

```
#Calculate mean and standard deviation of the training data
trainme<- apply(trainDRG,2,mean)
trainstd<- apply(trainDRG,2,sd)
```

```
#Create three matrices of dimensions equal to the Training, validating and testing data dimensions
trainidn<- (matrix(1,dim(trainDRG)[1],dim(trainDRG)[ 2]))
valiidn<- (matrix(1,dim(valiDRG)[1],dim(valiDRG)[2] ))
testidn<- (matrix(1,dim(testDRG)[1],dim(testDRG)[2] ))
```

```
#Normalize the three datasets.
#T() function is used for matrix transposing.
norm_trainDRG<- (trainDRG - t(trainme*t(trainidn))) /t(trainstd*t(trainidn))
norm_valiDRG<- (valiDRG - t(trainme*t(valiidn))) / t(trainstd*t(valiidn))
norm_testDRG<- (testDRG - t(trainme*t(testidn))) / t(trainstd*t(testidn))
```

```
#Define training, validating and testing period
traindir<- direction[trainrow,1]
validir<- direction[valirow,1]
testdir<- direction[testrow,1]
```

Implement the Artificial Neural Network

```
library(nnet)
```

```
#Implement ANN
#The neural network starts with random weights and will provide different results.
#Seed() function will allow the same output every time
set.seed(1)
```

```
#Implement ANN
#Parameter 1 -> All normalized columns
#Parameter 2 -> Target vector
#Parameter 3 -> number of hidden layers (4 in this case)
#Parameter 4 -> Output is indicated at the end (trace=T) or not (trace=F)
neural_network<- nnet(norm_trainDRG,class.ind(traindir),size=4,trace=T)
```

```
## # weights:  23
## initial  value 68.282999
## iter  10 value 32.571622
## iter  20 value 32.263153
## iter  30 value 32.070744
## iter  40 value 31.755650
## iter  50 value 31.448994
## final  value 31.448813
## converged
```

```
neural_network
```

```
## a 1-4-3 network with 23 weights
## options were -
```

```r
#Obtain data dimension
dim(norm_trainDRG)
```

```
## [1] 77  1
```

```r
#Make predictions
vali_pred<- predict(neural_network,norm_valiDRG)
head(vali_pred)
```

```
##            Down NoWhere Up
## 2020-01-03    0       0  0
## 2020-01-06    0       0  0
## 2020-01-07    0       0  0
## 2020-01-08    0       0  0
## 2020-01-09    0       0  0
## 2020-01-10    0       0  0
```

```r
#Calculate the predicted direction using the information obtained above.
vali_pred_class<- data.frame(matrix(NA,dim(vali_pred)[1],1))

#The next lines are used for checking the condition.
vali_pred_class[vali_pred[,"Down"] > 0.5,1] <- "Down"
vali_pred_class[vali_pred[,"NoWhere"] > 0.5,1] <- "NoWhere"
vali_pred_class[vali_pred[,"Up"] > 0.5,1] <- "Up"
```

```r
#Check forecasts accuracy
#Load caret library
#Use confusionMatrix() over the predicted class and original class for the validating dataset.
library(caret)
```

```
## Loading required package: lattice
```

```r
u<- union(vali_pred_class[,1],validir)
t<-table(factor(vali_pred_class[,1],u),factor(validir,u))
confusionMatrix(t)
```

```
## Confusion Matrix and Statistics
##
##
##           Down NoWhere Up
##   Down      31       1  7
##   NoWhere    0       0  0
##   Up         0       0  0
##
## Overall Statistics
##
##                Accuracy : 0.7949
##                  95% CI : (0.6354, 0.907)
##     No Information Rate : 0.7949
##     P-Value [Acc > NIR] : 0.5932
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: Down Class: NoWhere Class: Up
## Sensitivity               1.0000        0.00000    0.0000
## Specificity               0.0000        1.00000    1.0000
## Pos Pred Value            0.7949            NaN       NaN
## Neg Pred Value               NaN        0.97436    0.8205
## Prevalence                0.7949        0.02564    0.1795
## Detection Rate            0.7949        0.00000    0.0000
## Detection Prevalence      1.0000        0.00000    0.0000
## Balanced Accuracy         0.5000        0.50000    0.5000
```

```
#Check the accuracy on testing data
test_pred <- predict(neural_network,norm_testDRG)
head(test_pred)
```

```
##            Down NoWhere Up
## 2020-01-03    0       0  0
## 2020-01-06    0       0  0
## 2020-01-07    0       0  0
## 2020-01-08    0       0  0
## 2020-01-09    0       0  0
## 2020-01-10    0       0  0
```

```
#Indicate the classes for the testing data
test_pred_class<- data.frame(matrix(NA,dim(test_pred)[1],1) )
test_pred_class[test_pred[,"Down"] > 0.5,1] <- "Down"
test_pred_class[test_pred[,"NoWhere"] > 0.5,1] <- "NoWhere"
test_pred_class[test_pred[,"Up"] > 0.5,1] <- "Up"
#Check the accuracy of the forecasts
u<- union(test_pred_class[,1],testdir)
t<-table(factor(test_pred_class[,1],u),factor(testdir,u))
confusionMatrix(t)
```

```
## Confusion Matrix and Statistics
##
##
##           Down NoWhere Up
##   Down      31       1  7
##   NoWhere    0       0  0
##   Up         0       0  0
##
## Overall Statistics
##
##                Accuracy : 0.7949
##                  95% CI : (0.6354, 0.907)
##     No Information Rate : 0.7949
##     P-Value [Acc > NIR] : 0.5932
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Down Class: NoWhere Class: Up
```

```
## Sensitivity             1.0000       0.00000    0.0000
## Specificity             0.0000       1.00000    1.0000
## Pos Pred Value          0.7949           NaN       NaN
## Neg Pred Value             NaN       0.97436    0.8205
## Prevalence              0.7949       0.02564    0.1795
## Detection Rate          0.7949       0.00000    0.0000
## Detection Prevalence    1.0000       0.00000    0.0000
## Balanced Accuracy       0.5000       0.50000    0.5000
```

```r
#Generate trade signals
signal<- ifelse(test_pred_class =="Up",0.01,ifelse(test_pred_class =="Down",-0.01,0))
head(signal)
```

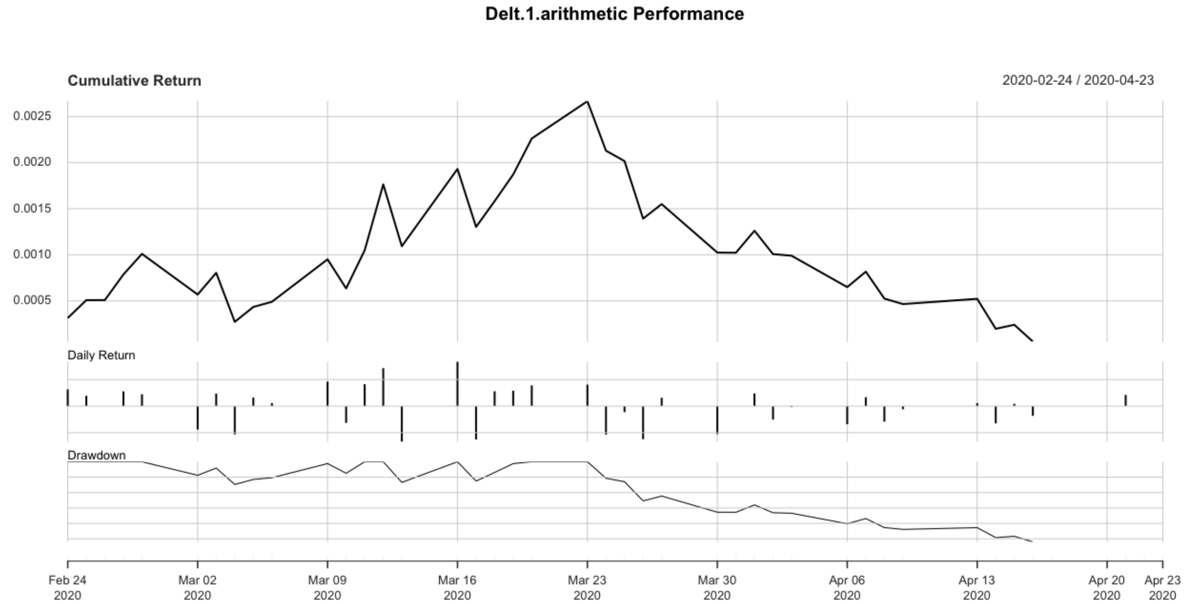
```
##      matrix.NA..dim.test_pred..1...1.
## [1,]                               NA
## [2,]                               NA
## [3,]                               NA
## [4,]                               NA
## [5,]                               NA
## [6,]                               NA
```

```r
test_return_DRG<- return[(index(return)>= test_sdate & index(return)<= test_edate), ]
test_return<- test_return_DRG*(signal)

#calculate cummulative return
cumm_return<- Return.cumulative(test_return)
cumm_return
```

```
##                    Delt.1.arithmetic
## Cumulative Return       0.0002692915
```

```r
#calculate annual return
annual_return<- Return.annualized(test_return)
annual_return
```

```
##                    Delt.1.arithmetic
## Annualized Return        0.001741318
```




```r
VaR(test_return, p=0.95)
```

```
##     Delt.1.arithmetic
## VaR     -0.0006170195
```

```r
SharpeRatio(as.ts(test_return), Rf = 0, p=0.95, FUN = "StdDev")
```

```
##                                     [,1]
## StdDev Sharpe (Rf=0%, p=95%): 0.01856136
```

```r
SharpeRatio.annualized(test_return, Rf=0)
```

```
##                                 Delt.1.arithmetic
## Annualized Sharpe Ratio (Rf=0%)         0.2920001
```

Compiled Results

The above neural network trained on the DRG - DJI and coronavirus Google trends data, provides the following output.

1. `a 1-4-3 network with 23 weights options were -`
2. `The first layer (input layer) -> 1  The second layer (hidden layer) -> 4  Third layer (output layer) -> 3 Weighted parameters -> 23`
3. `Dim(norm_traindji)` function provides the following output [1] 71 1
4. Number of columns = number of neurons in the input layer = number of input data features = 1

Conclusion

The project demonstrated a simple artificial neural network which trains data and tests it, making the stock data to combine with trend data. There are however more efficient methods available in R. Such neural networks can be made to use the machine GPU (Graphics Processing Unit) along with the CPU, and produce deep learning results. Popular programs with deeplearning libraries which can be installed in R are Tensorflow, Keras, and Pytorch.

References

- The NYSE Arca Pharmaceutical Index (DRG), NYSE. https://www.nyse.com/publicdocs/nyse/indices/nyse_arca_pharmaceutical_index.pdf
- Preis T., Moat H., Stanley E. Quantifying Trading Behavior in Financial Markets Using Google Trends, Scientific Reports. (2013). https://www.nature.com/articles/srep01684
- DeGroot M., Schervish M. Probability and Statistics, Fourth Edition (2014)