Final Report WQU Econometrics Week 7 Group Work Group 6A

Contents

Group Member (in alphabetical order):

- Dylan Thorne (dylan.thorne@gmail.com)
- Pariyat Limpitheeprakan (macchiato.me@gmail.com)
- Trai Torsricharoen (traitorsricharoen@gmail.com)
- YIU LEUNG CHENG (howard.yl.cheng.hk@gmail.com)

3.3.1 Algorithmic Trading

Introduction

To design our own algorithmic trading strategy in R. We decided our selections in this task as follows:

Number of assets in the strategy: SPY (Source:Yahoo)
Type of asset: stock
Timeframe: from `2007-03-01` to `2017-03-01`
Coding language: R, Excel (used supplmentarily for our analysis)
Model: Neural Networks (Package: nnet).

Forecasting stock prices has been a daunting task for many of the researchers and analysts. Thanks to recent breakthroughs of deep learning, applications such as deep neural network (Palaniappan, 2018), LSTM (Choudhury, 2019), or even GAN (Zhang, Zhong, Dong, Wang, & Wang, 2019) have been adopted into predicting future stock price in real-time manners. Therefore, we aim to try imitate such application. Our main idea in this project is to simply use neural network model to forcast the direction of stock price.

Our algorithmic trading implementation are described step-by-step as follow:

Load input time series data,

Firstly, we found following package useful in our implmementation:

```
require("quantmod") # useful quantitative financial modelling and trading framework in R
require(zoo) # for faciliating dealing with time series data
require(dplyr) # for faciliating data transformation
require(nnet) # for NN model
require(caret) # for misc statistics and fundamental machine learning model.
```

```
require(ggfortify) # for data visualization
require(magrittr) # for piping %>%
require(PerformanceAnalytics) # for calulate annual return and cummulative return
```

Then we load `SPY` from Yahoo Finance and preview and do some EDA on the time series.

```
options("getSymbols.warning4.0"=FALSE)
invisible(getSymbols("SPY", scr="yahoo",verbose = FALSE))
SPY500<- SPY[,"SPY.Close"]
autoplot(SPY500)
```

Imputation & Feature Engineering

We adopted Last Observation Carried Forward imputation method by `na.locf` in package zoo. Furthermore, we need to extract and derive some features (columns) in order to help our neural networks to learn to encompass variance of the time series. The derived features we chose include: rolling mean, rolling standard deviation, RSI, MACD, Bollinger Bands (used to measure stock's volatility and price levels). For the label of the output of the model, we leverage the lag return and label the output into 3 classes: `NoWhere`, `Up` and `Down`.

```
#fill NA with previous non-NA value

SPY500 <- na.locf(SPY500)
return <- Delt(SPY500)

average10<- rollapply(SPY500, 10, mean)
average20<-rollapply(SPY500, 20, mean)
```

```
std10<- rollapply(SPY500, 10, sd)
std20<- rollapply(SPY500, 20, sd)

rsi5<- RSI(SPY500,5,"SMA")
rsi14<- RSI(SPY500, 14, "SMA")

macd12269<- MACD(SPY500, 12, 26, 9, "SMA")
macd7205<- MACD(SPY500, 7, 20, 5, "SMA")

bollinger_bands<-BBands(SPY500,20,"SMA",2)

direction<- data.frame(matrix(NA,dim(SPY500)[1],1))

lagreturn<- (SPY500 - Lag(SPY500, 20))/Lag(SPY500, 20)

# Feature Engineer the label (multi-class classification)
direction[lagreturn>0.02] <- "Up"
direction[lagreturn< -0.02] <- "Down"
direction[lagreturn< 0.02 &lagreturn> -0.02] <- "NoWhere"

SPY500 <- cbind(SPY500, average10, average20, std10, std20, rsi5, rsi14, macd12269, macd7205, bollinger_
```

Split Train-Test-Validate Dataset

Next, we split train, test, validate dataset. Notice that we simply divide time series data into train, test and validate dataset by year similar to cross-validation method ("Cross-Validation strategies for Time Series forecasting [Tutorial]," 2019). More importantly, we need to normalize the train, test and validate dataset as well.

```
train_sdate<- "2007-03-01"
train_edate<- "2017-03-01"
vali_sdate<- "2017-03-02"
vali_edate<- "2018-03-02"
test_sdate<- "2018-03-03"
test_edate<- "2019-10-18"
trainrow<- which(index(SPY500) >= train_sdate& index(SPY500) <= train_edate)
valirow<- which(index(SPY500) >= vali_sdate& index(SPY500) <= vali_edate)
testrow<- which(index(SPY500) >= test_sdate& index(SPY500) <= test_edate)
train<- SPY500[trainrow,]
vali<- SPY500[valirow,]
test<- SPY500[testrow,]
trainme<-apply(train,2,mean)
trainstd<-apply(train,2,sd)
trainidn<- (matrix(1,dim(train)[1],dim(train)[2]))
valiidn<- (matrix(1,dim(vali)[1],dim(vali)[2]))
testidn<- (matrix(1,dim(test)[1],dim(test)[2]))
norm_train<- (train-t(trainme*t(trainidn)))/t(trainstd*t(trainidn))
norm_vali<- (vali-t(trainme*t(valiidn)))/t(trainstd*t(valiidn))
norm_test<- (test-t(trainme*t(testidn)))/t(trainstd*t(testidn))
traindir<- direction[trainrow,1]
validir<- direction[valirow,1]
testdir<- direction[testrow,1]
```

Train our neural network model

As we tuned the parameters, we found that we use 4 hidden layers and use default least-squares cross entropy function to calculate errors. Due to limitations of Rstudio Cloud, we limit max iteration to 100 iterations to train the model.

```
set.seed(1)
neural_network<- nnet(norm_train, class.ind(traindir), size=4, trace=T)
```

```
## # weights:  79
## initial  value 2292.477894
## iter  10 value 790.818756
## iter  20 value 558.454192
## iter  30 value 501.782827
## iter  40 value 479.945347
## iter  50 value 459.877532
## iter  60 value 437.096294
## iter  70 value 425.893461
## iter  80 value 419.107186
## iter  90 value 412.675545
## iter 100 value 410.010743
## final  value 410.010743
## stopped after 100 iterations
```

```
dim(norm_train)
```

```
## [1] 2519   15
```

Evaluate our model with validate and test dataset

To measure performance of our neural network model, we utilize confusion matrix to view model's accuracy, sensitivity, specificity, etc. See more detailed explaination about how to use these metric to evaluate predictive models in ("Measures of Predictive Models: Sensitivity and Specificity," 2018).

```
vali_pred<-predict(neural_network, norm_vali)
head(vali_pred)
```

```
##                    Down    NoWhere        Up
## 2017-03-02 0.0001502134 0.01622079 0.9883595
## 2017-03-03 0.0001619169 0.01865729 0.9865608
## 2017-03-06 0.0003117574 0.07004219 0.9456502
## 2017-03-07 0.0004460157 0.13300459 0.8912861
## 2017-03-08 0.0006771546 0.26387151 0.7709039
## 2017-03-09 0.0006230756 0.21195155 0.8139530
```

```
vali_pred_class<- data.frame(matrix(NA,dim(vali_pred)[1],1))
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"
vali_pred_class[is.na(vali_pred_class)]<- "NoWhere"

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

```
## Confusion Matrix and Statistics
```

```
##
##
##           Up NoWhere Down
##   Up       57       5    4
##   NoWhere  30     139    6
##   Down      0       2   10
##
## Overall Statistics
##
##                Accuracy : 0.8142
##                  95% CI : (0.7607, 0.8602)
##     No Information Rate : 0.5771
##     P-Value [Acc > NIR] : 9.120e-16
##
##                   Kappa : 0.6339
##
##  Mcnemar's Test P-Value : 2.676e-05
##
## Statistics by Class:
##
##                      Class: Up Class: NoWhere Class: Down
## Sensitivity             0.6552         0.9521     0.50000
## Specificity             0.9458         0.6636     0.99142
## Pos Pred Value          0.8636         0.7943     0.83333
## Neg Pred Value          0.8396         0.9103     0.95851
## Prevalence              0.3439         0.5771     0.07905
## Detection Rate          0.2253         0.5494     0.03953
## Detection Prevalence    0.2609         0.6917     0.04743
## Balanced Accuracy       0.8005         0.8078     0.74571
```

```
test_pred<- predict(neural_network, norm_test)
head(test_pred)
```

```
##                    Down    NoWhere        Up
## 2018-03-05 1.211824e-02 0.986720323 0.0103075
## 2018-03-06 2.500869e-03 0.818433435 0.1847361
## 2018-03-07 1.897507e-03 0.732761399 0.2805843
## 2018-03-08 3.679560e-04 0.097380484 0.9231653
## 2018-03-09 4.376541e-05 0.001232120 0.9991996
## 2018-03-12 4.388062e-05 0.001239012 0.9991950
```

```
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"
test_pred_class[is.na(test_pred_class)]<- "NoWhere"
u<- union(test_pred_class[,1],testdir)
t<-table(factor(test_pred_class[,1],u),factor(testdir,u))
confusionMatrix(t)
```

```
## Confusion Matrix and Statistics
##
##
##          NoWhere  Up Down
##   NoWhere     115  20   21
```

```
##   Up           29 145    2
##   Down          3   0   76
##
## Overall Statistics
##
##                Accuracy : 0.8175
##                  95% CI : (0.7767, 0.8537)
##     No Information Rate : 0.4015
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7175
##
##  Mcnemar's Test P-Value : 0.0006573
##
## Statistics by Class:
##
##                      Class: NoWhere Class: Up Class: Down
## Sensitivity                  0.7823    0.8788      0.7677
## Specificity                  0.8447    0.8740      0.9904
## Pos Pred Value               0.7372    0.8239      0.9620
## Neg Pred Value               0.8745    0.9149      0.9307
## Prevalence                   0.3577    0.4015      0.2409
## Detection Rate               0.2798    0.3528      0.1849
## Detection Prevalence         0.3796    0.4282      0.1922
## Balanced Accuracy            0.8135    0.8764      0.8790
```

As results shown above, accuracy of our model on validate and test dataset is quite high and statiscally significant.

Calculate returns, cumulative returns, standard deviation and forecasts

Now, we use predicted return from our model to calculate returns, cumulative returns, standard deviation (Sharpe Ratio of Return over StdDev).

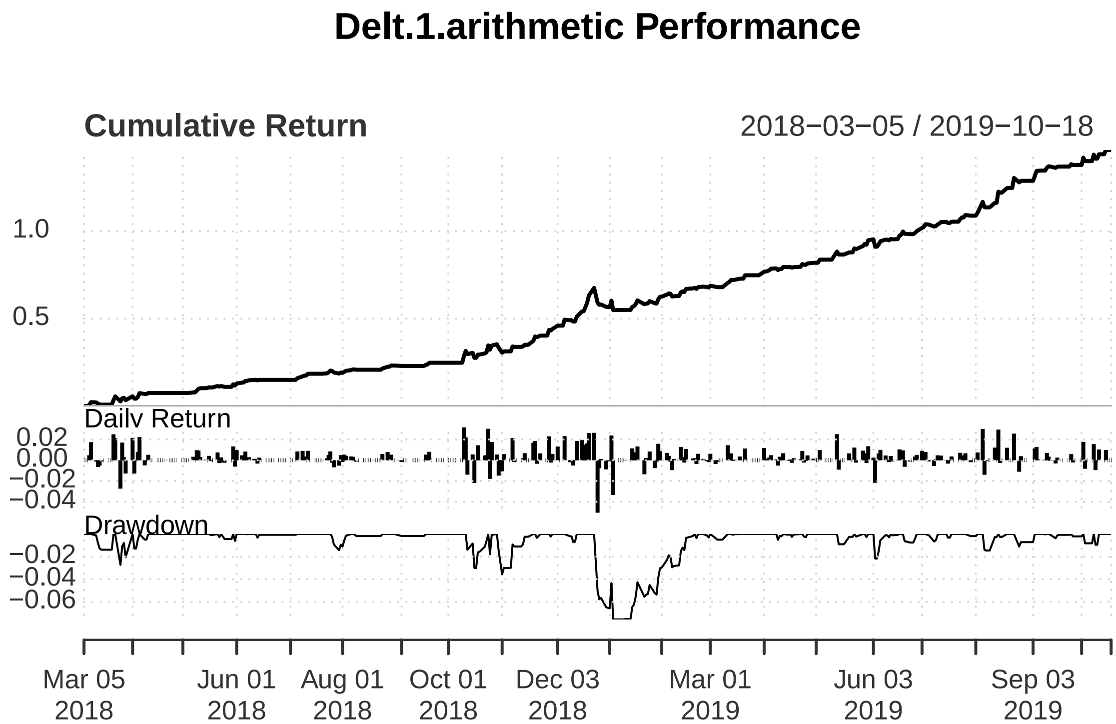
```
signal<-ifelse(test_pred_class=="Up",1,ifelse(test_pred_class=="Down",-1, 0))
test_return_SPY<- return[(index(return)>= test_sdate & index(return)<= test_edate), ]
test_return<- test_return_SPY*(signal)

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

```
##                   Delt.1.arithmetic
## Cumulative Return          1.463558
```

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

```
##                   Delt.1.arithmetic
## Annualized Return         0.7381301
```

```
charts.PerformanceSummary(test_return)
```




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

```
##     Delt.1.arithmetic
## VaR       -0.01006893
```

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

```
##                                    [,1]
## StdDev Sharpe (Rf=0%, p=95%): 0.2752382
```

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

```
##                                 Delt.1.arithmetic
## Annualized Sharpe Ratio (Rf=0%)          5.742213
```

3.3.2 Improve The Strategy

Implment Pair trading and Back-Testing in Excel

To improve the trading strategy, we revisit to do EDA (exploratory data analysis) and try to relate to toos and methodology stated in our contents in Econometrics course so that we could find something useful to improve our existing trading strategy. Firstly, we have tried implementing pair trading and backtesting by using Thai Stocks in Banking Sector, namely KBank and SCB (both of which are sourced from Yahoo Finance).

Pair trading is a streatgy for trading two higly correlated financial asset. Its main idea is that sometime the spread between two correlated asset is wider/narrower than usual. Then, trading opportunity exist. Back testing, however, is the test to check that the streatgy work well by test with the historical data. Our findings are as follows

Please refer to more details about our implementation in PairTrading.xlsx. Note that in the excel file, we implement IFS function inside.

Figure 1: Stock Price Comparison KBANK and SCB

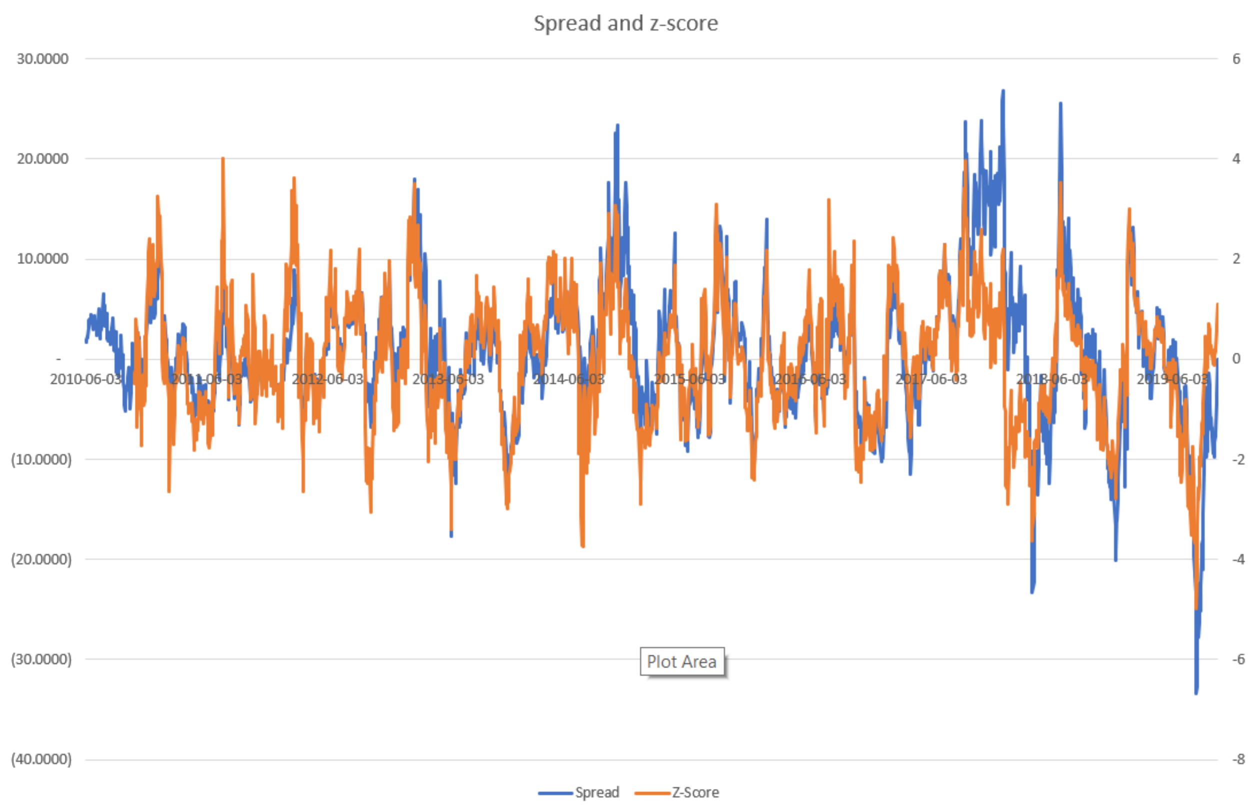


Figure 2: Spread and Z-Score

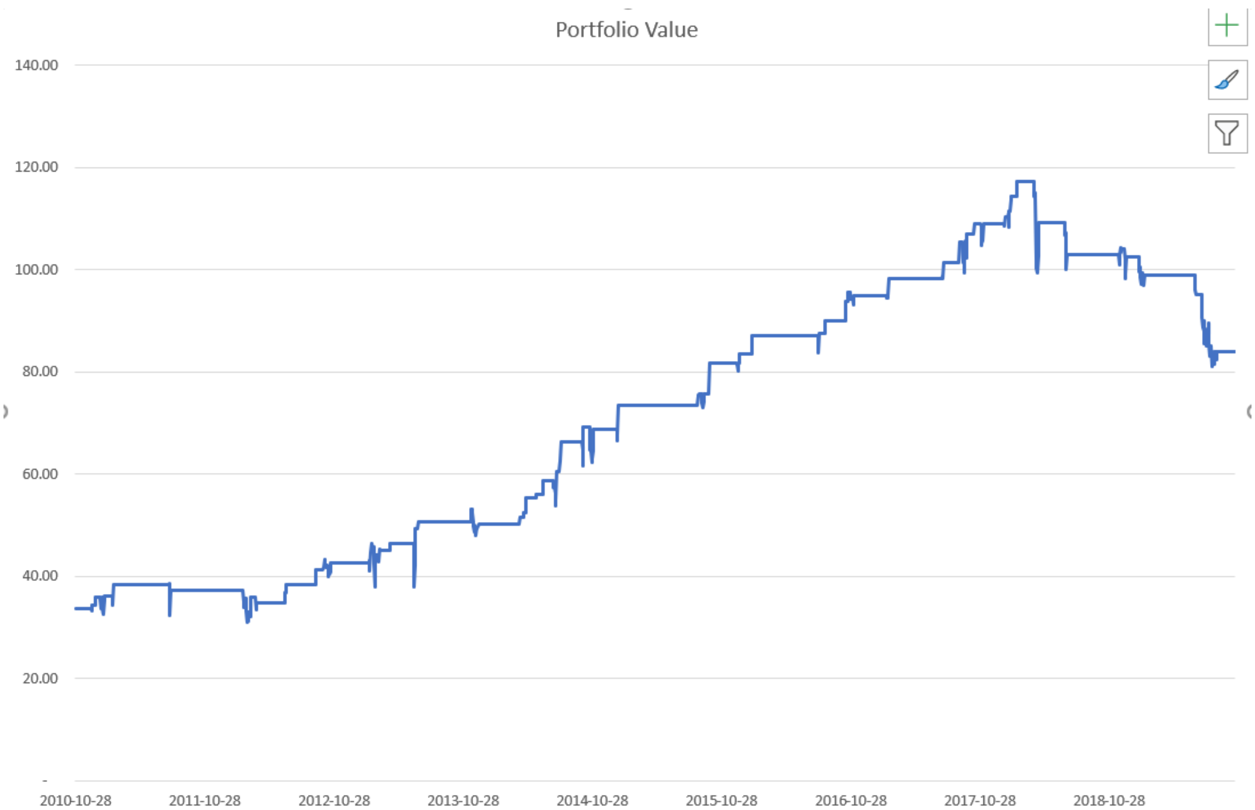


Figure 3: Cumulative Portfollio Value

Thanks to our research attempt above, we notice that to the stock price of `SPY` have been time oto time affected by several macro-economics unknown factors. To materialize and factor in such factors, we came up with an idea to incorporate data from Google trend which is, from our point of views, considered as a comprephensive output of those unknown factors. Initiated by this idea, we decided to improve our existing algorithmic trading strategy in [Section 3.3.1] by implmenting **Neural Network model incorporate with GARCH model and Google trend input**.

Implement Neural Network model incorporate with GARCH model and Google trend input

To start with, we have done some literature review: GARCH models and neural network has been adopted to forecast volatility (Lu, Que, & Cao, 2016) and conditional variance of stock returns (Arnerić, Poklepović, & Aljinović, 2014). For our this implementation, along with methodology stated in the aforementioned literature, we utilize combinative methods such as ACF plot, PDF plot, qq-plot and associated testing to determine parameters of our GARCH models.

Our implmentations are as follows. Since R script below are similar to our implmentation in Section 3.3.1, additionally required explanation will be provided:

```
require(stats) # for Garch model
require(tseries) # for Garch model
require(forecast) # for Garch model
require(fGarch) # for Garch model
require("rugarch") # for Garch model
```

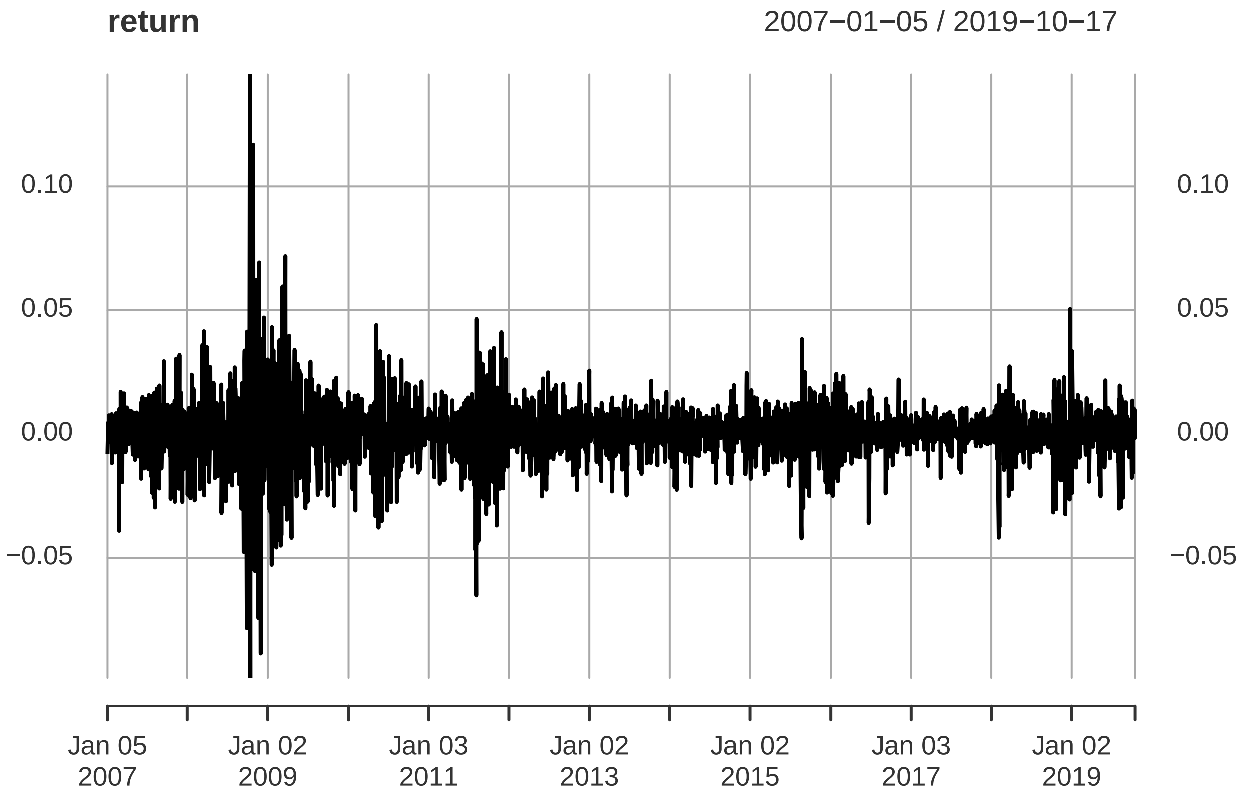
```
getSymbols("SPY", scr="yahoo",from = as.Date("2007-01-04"), to = as.Date("2019-10-18"),warnings=FALSE)
```

```
## [1] "SPY"
```

```
SPY500<- SPY[,"SPY.Close"]
head(SPY500)
```

```
##            SPY.Close
## 2007-01-04    141.67
## 2007-01-05    140.54
## 2007-01-08    141.19
## 2007-01-09    141.07
## 2007-01-10    141.54
## 2007-01-11    142.16
```

```
#Imputation
#fill NA with previous non-NA value
SPY500 <- na.locf(SPY500)
return <- Delt(SPY500)
rows = nrow(return)
return <- return[2:rows]
plot(return)
```




```r
#Feature Engineering
#technical analysis indicators
average10<- rollapply(SPY500, 10, mean)
average20<-rollapply(SPY500, 20, mean)
std10<- rollapply(SPY500, 10, sd)
std20<- rollapply(SPY500, 20, sd)
rsi5<- RSI(SPY500,5,"SMA")
rsi14<- RSI(SPY500, 14, "SMA")
macd12269<- MACD(SPY500, 12, 26, 9, "SMA")
macd7205<- MACD(SPY500, 7, 20, 5, "SMA")
bollinger_bands<-BBands(SPY500,20,"SMA",2)
direction<- data.frame(matrix(NA,dim(SPY500)[1],1))
lagreturn<- (SPY500 - Lag(SPY500, 20))/Lag(SPY500, 20)
direction[lagreturn>0.02] <- "Up"
direction[lagreturn< -0.02] <- "Down"
direction[lagreturn< 0.02 &lagreturn> -0.02] <- "NoWhere"
```

Now, in this step, we begin determining parameters of our GARCH model.

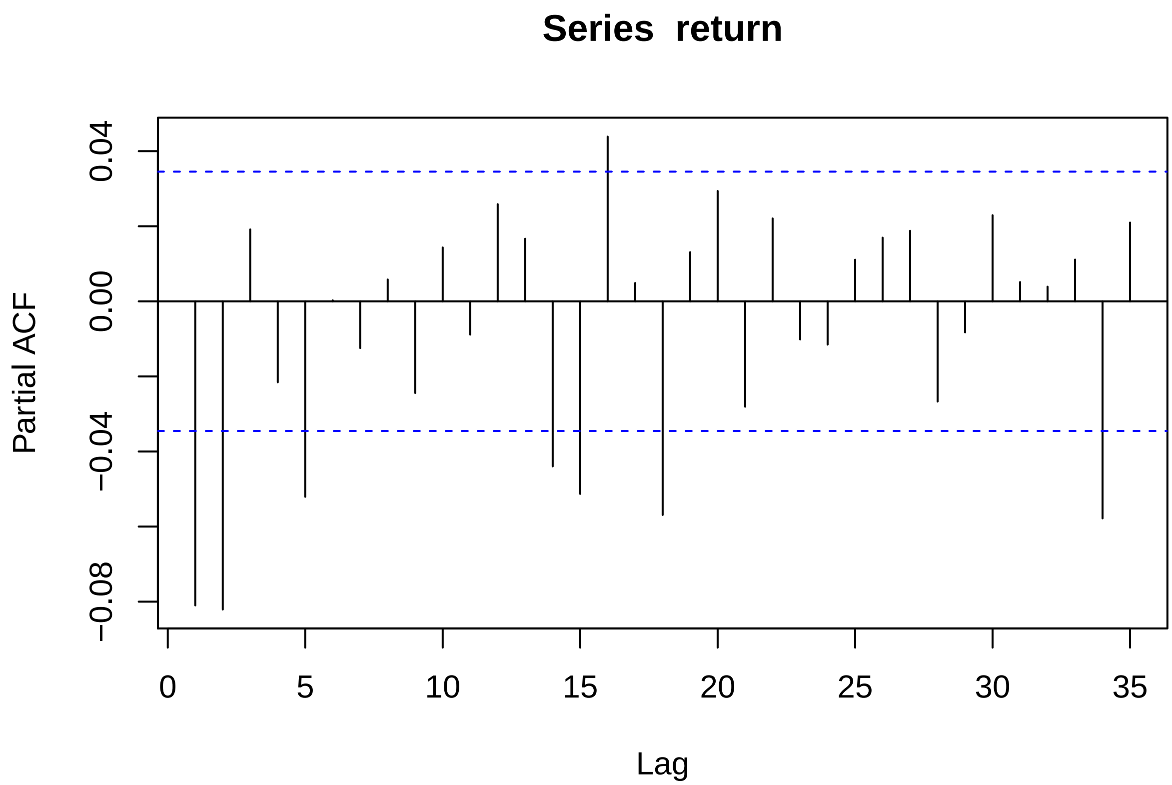
```r
#GARCH Model

#adf test suggesting stationarity
adf.test(return)
```

```
## Warning in adf.test(return): p-value smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  return
## Dickey-Fuller = -16.192, Lag order = 14, p-value = 0.01
```

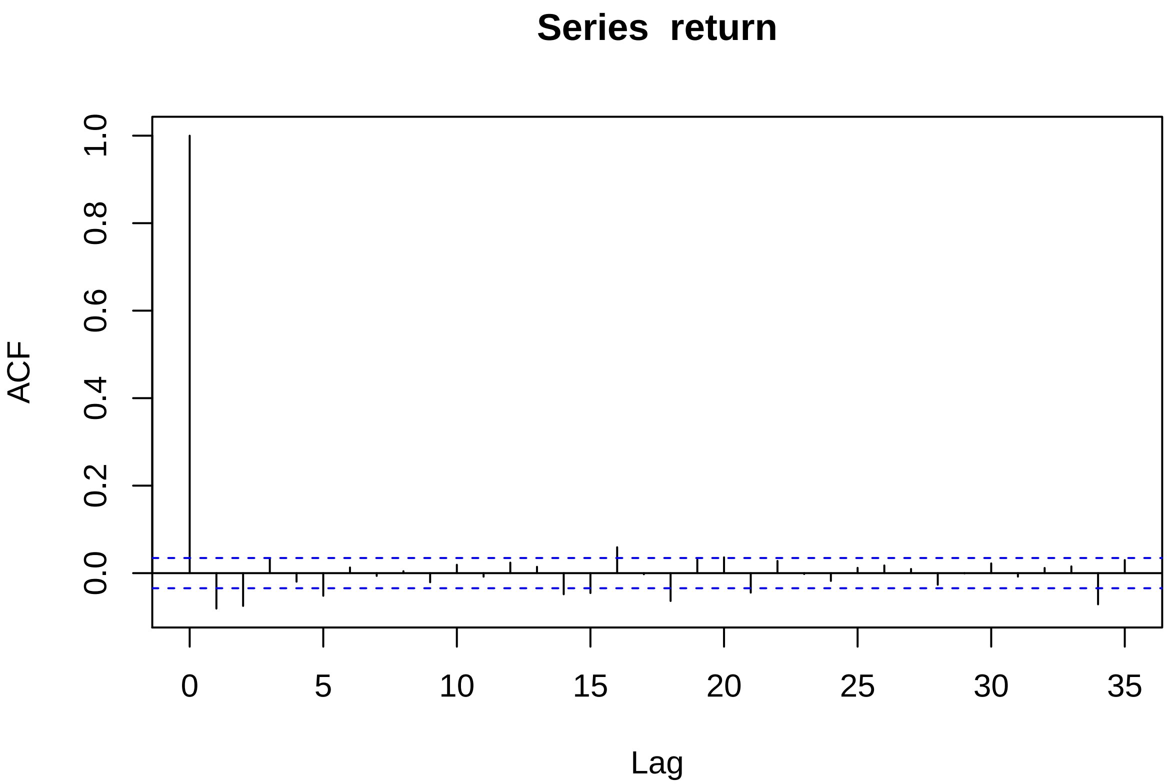
```
## alternative hypothesis: stationary
```

Firstly, we use ADF to test and found `return` is stationary. Next, we plot both PACF and ACF.

```r
#PACF plot suggests significant spike through lag 2.
pacf(return)
```



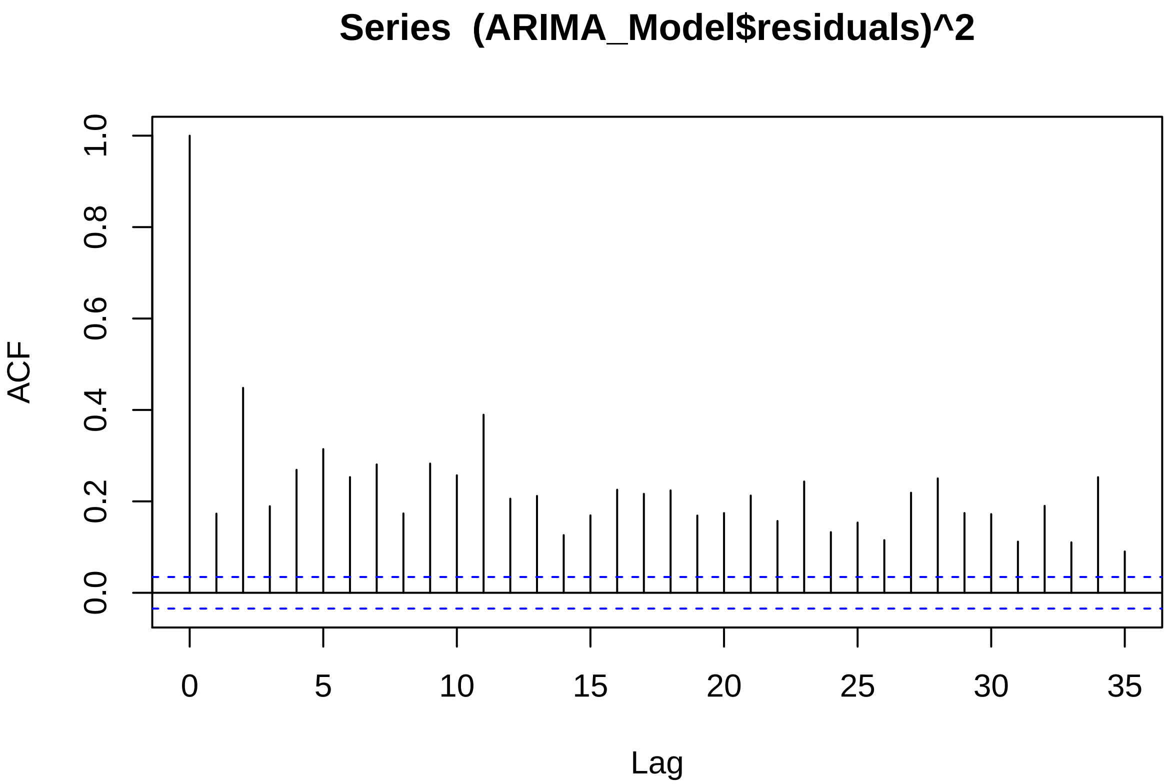

```r
#ACF plot shows exponential decay. Thus, it can be deduced AR(2) model.
acf(return)
```



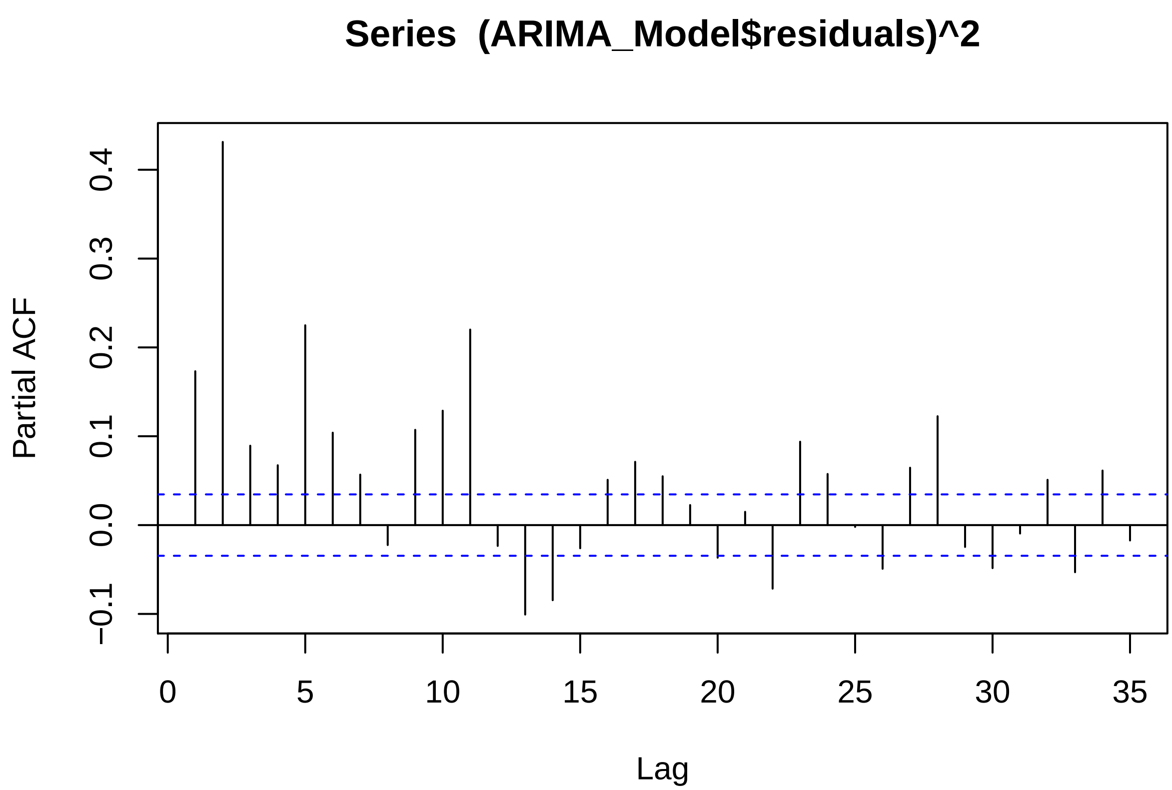
Now we try with ARIMA(2,0,0) and determin square residuals:

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

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




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



Finally, we try fitting model with ARMA(2,0) and GARCH(11,0)

```r
model <- garchFit(formula = ~ arma(2,0) + garch(11,0) , data = timeseries, trace = F)
summary(model)
```

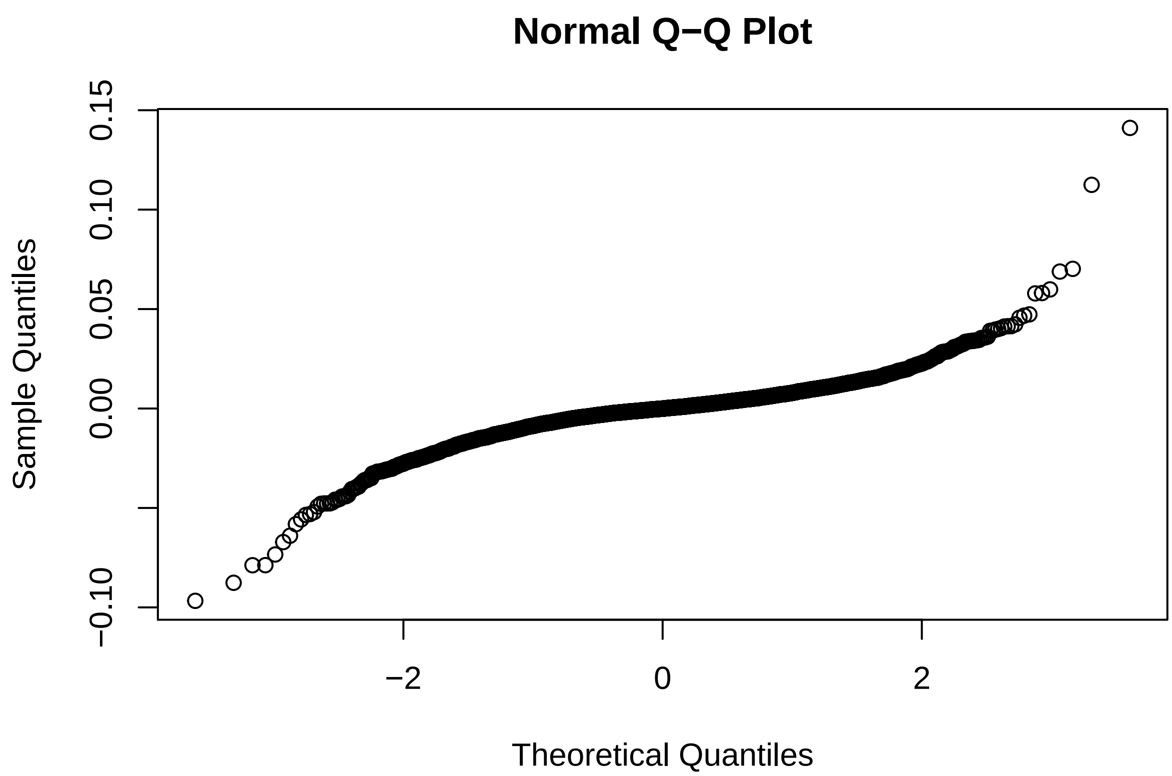
```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = ~arma(2, 0) + garch(11, 0), data = timeseries,
##     trace = F)
##
## Mean and Variance Equation:
##  data ~ arma(2, 0) + garch(11, 0)
## <environment: 0x11dc9d68>
##  [data = timeseries]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##          mu          ar1          ar2        omega       alpha1
##  8.1551e-04  -6.5797e-02  -2.4323e-02   1.8515e-05   7.6807e-02
##      alpha2       alpha3       alpha4       alpha5       alpha6
##  1.5374e-01   9.8210e-02   1.3536e-01   6.3122e-02   5.9403e-02
##      alpha7       alpha8       alpha9      alpha10      alpha11
##  5.4562e-02   5.9989e-02   5.9964e-02   6.7037e-02   4.2801e-02
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##           Estimate  Std. Error  t value Pr(>|t|)
## mu       8.155e-04   1.360e-04    5.997 2.01e-09 ***
## ar1     -6.580e-02   1.836e-02   -3.583 0.000339 ***
## ar2     -2.432e-02   1.936e-02   -1.256 0.209051
## omega    1.852e-05   1.747e-06   10.595  < 2e-16 ***
## alpha1   7.681e-02   1.794e-02    4.280 1.87e-05 ***
## alpha2   1.537e-01   2.475e-02    6.212 5.22e-10 ***
## alpha3   9.821e-02   2.234e-02    4.396 1.11e-05 ***
## alpha4   1.354e-01   2.603e-02    5.200 2.00e-07 ***
## alpha5   6.312e-02   1.846e-02    3.419 0.000628 ***
## alpha6   5.940e-02   1.861e-02    3.191 0.001416 **
## alpha7   5.456e-02   1.725e-02    3.162 0.001565 **
## alpha8   5.999e-02   1.885e-02    3.183 0.001460 **
## alpha9   5.996e-02   2.092e-02    2.867 0.004149 **
## alpha10  6.704e-02   1.907e-02    3.516 0.000438 ***
## alpha11  4.280e-02   1.716e-02    2.494 0.012641 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  10521.29    normalized:  3.268496
##
## Description:
```

```
##  Tue Oct 22 20:39:44 2019 by user:
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  766.633   0
##  Shapiro-Wilk Test  R    W      0.9749061 0
##  Ljung-Box Test     R    Q(10)  12.21964  0.2706257
##  Ljung-Box Test     R    Q(15)  23.61286  0.07196621
##  Ljung-Box Test     R    Q(20)  27.28147  0.1275291
##  Ljung-Box Test     R^2  Q(10)  3.15208   0.977615
##  Ljung-Box Test     R^2  Q(15)  8.047753  0.9218539
##  Ljung-Box Test     R^2  Q(20)  10.68428  0.9540028
##  LM Arch Test       R    TR^2   5.127157  0.9535963
##
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -6.527673 -6.499356 -6.527716 -6.517524
```

```
res = residuals(model)
```

And then we plot qq-plot of residual between GARCH model and actual data:

```
qqnorm(res)
```




```
garch11_spec <- ugarchspec(variance.model = list(garchOrder = c(11, 0)),mean.model = list(armaOrder = c
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-->warning: solver failer to converge.
```

```
garch11_fit
```

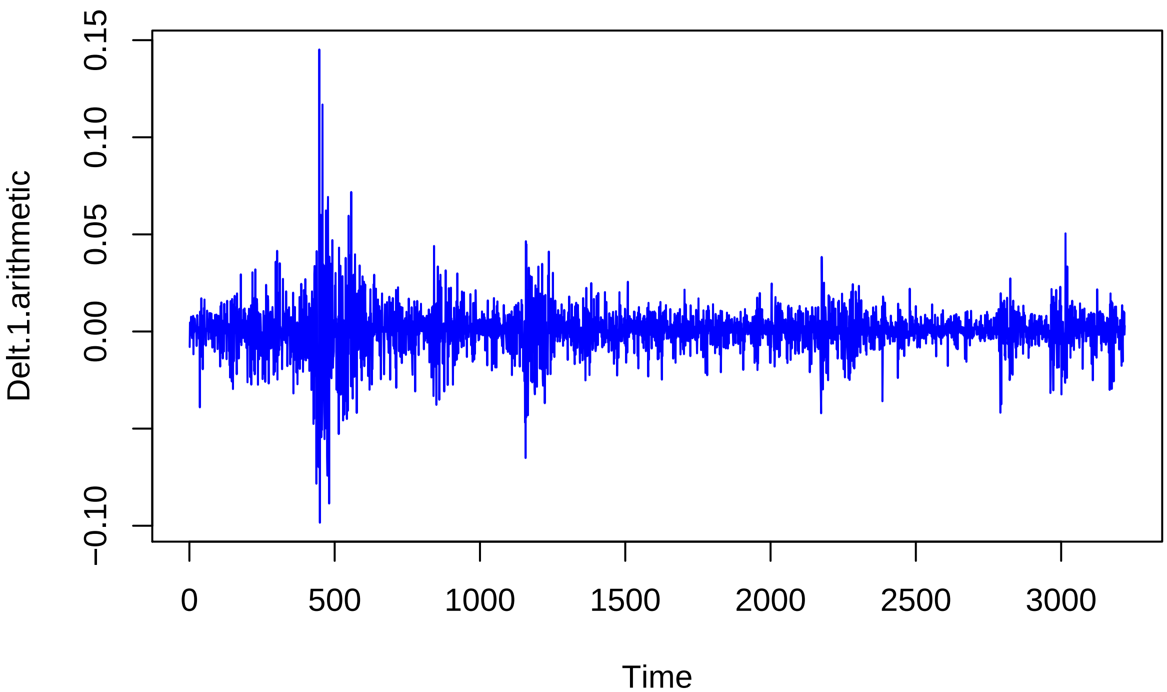
```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(11,0)
## Mean Model   : ARFIMA(2,0,0)
## Distribution : norm
##
## Convergence Problem:
## Solver Message:
```

```
garch11_fit@fit$fitted.values
```

```
## NULL
```

Now we visualize how well GARCH itself fit the data:

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



Here, after we fitting the GARCH model, the output is applied with rolling window function and combine into input dataset for training in neural network model.

```
#binding closing price and technical analysis indicators into a variable SPY500
SPY500 <- cbind(SPY500[2:nrow(SPY500)], average10[2:nrow(average10)], average20[2:nrow(average20)], std

#integrate GARCH model rolling window prediction output into variable
SPY500 <- cbind(SPY500,garch11_fit@fit$fitted.values)
```

In this step, we incoporate Google Trend data into our GARCH model. Google trend data is extracted from

Google Trend website. Yet, the data retrieved from there is in monthly basis. So, we just transformed those data into daily basis with excel: - Recession_gtrends.csv
- Expansion_gtrends.csv

We combine existing input data and newly read Google Trend data:

```
#Import Google trend data regarding trend of recession and expansion
recessiondata<-read.csv("Recession_gtrends.csv",header=F)$V2
expansiondata<-read.csv("Expansion_gtrends.csv",header=F)$V2
#integrate Google trend data into variable
SPY500 <- cbind(SPY500,recessiondata,expansiondata)
```

And then we begin the same process as int

```
#indicate end and start dates for train, validating and testing period
train_sdate<- "2007-03-01"
train_edate<- "2017-03-01"
vali_sdate<- "2017-03-02"
vali_edate<- "2018-03-02"
test_sdate<- "2018-03-03"
test_edate<- "2019-10-18"
```

```
#constructing data ranges for the three datasets
trainrow<- which(index(SPY500) >= train_sdate& index(SPY500) <= train_edate)
valirow<- which(index(SPY500) >= vali_sdate& index(SPY500) <= vali_edate)
testrow<- which(index(SPY500) >= test_sdate& index(SPY500) <= test_edate)
```

```
#extract data fpr training, validating and testing periods
train<- SPY500[trainrow,]
vali<- SPY500[valirow,]
test<- SPY500[testrow,]
trainme<-apply(train,2,mean)
trainstd<-apply(train,2,sd)

#training, validating and testing data dimensions
trainidn<- (matrix(1,dim(train)[1],dim(train)[2]))
valiidn<- (matrix(1,dim(vali)[1],dim(vali)[2]))
testidn<- (matrix(1,dim(test)[1],dim(test)[2]))

#normalize the three datasets
norm_train<- (train-t(trainme*t(trainidn)))/t(trainstd*t(trainidn))
norm_vali<- (vali-t(trainme*t(valiidn)))/t(trainstd*t(valiidn))
norm_test<- (test-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]
```

Additionally, we improve our neural network by hyper-parameter tuning (Boyle, 2019), i.e. increase iterations and lower hidden layers and set weight decay in order to mitigate overfitting.

```
#implement NN
require(nnet)
set.seed(1)
neural_network<- nnet(norm_train, class.ind(traindir), maxit = 400, size=2,decay=0.01, trace=T)
```

```
## # weights:  45
```

```
## initial  value 2130.092492
## iter  10 value 846.259706
## iter  20 value 678.733246
## iter  30 value 626.653792
## iter  40 value 595.455987
## iter  50 value 581.778184
## iter  60 value 576.075851
## iter  70 value 575.338375
## iter  80 value 575.136773
## iter  90 value 575.027682
## iter 100 value 575.018434
## iter 110 value 575.012858
## iter 120 value 574.997176
## iter 130 value 574.983195
## final  value 574.983143
## converged
```

```r
#obtain data dimension
dim(norm_train)
```

```
## [1] 2519   17
```

```r
#make prediction
vali_pred<-predict(neural_network, norm_vali)
head(vali_pred)
```

```
##                    Down    NoWhere        Up
## 2017-03-02 0.0009435159 0.03518121 0.9603901
## 2017-03-03 0.0009406432 0.03510830 0.9605271
## 2017-03-06 0.0009992822 0.04065603 0.9549907
## 2017-03-07 0.0010610733 0.04727113 0.9485857
## 2017-03-08 0.0018645513 0.14349999 0.8531217
## 2017-03-09 0.0021667185 0.19592629 0.8055405
```

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

#check forecast accuracy

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

```
## Confusion Matrix and Statistics
##
##
##            Up NoWhere Down
##   Up       63      22    0
##   NoWhere  25     123    5
##   Down      0       1   14
##
## Overall Statistics
```

```
##
##                Accuracy : 0.7905
##                  95% CI : (0.7351, 0.839)
##     No Information Rate : 0.5771
##     P-Value [Acc > NIR] : 6.531e-13
##
##                   Kappa : 0.6045
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Up Class: NoWhere Class: Down
## Sensitivity             0.7159         0.8425     0.73684
## Specificity             0.8667         0.7196     0.99573
## Pos Pred Value          0.7412         0.8039     0.93333
## Neg Pred Value          0.8512         0.7700     0.97899
## Prevalence              0.3478         0.5771     0.07510
## Detection Rate          0.2490         0.4862     0.05534
## Detection Prevalence    0.3360         0.6047     0.05929
## Balanced Accuracy       0.7913         0.7810     0.86628
```

```
#check accuracy on testing data
test_pred<- predict(neural_network, norm_test)
head(test_pred)
```

```
##                    Down    NoWhere         Up
## 2018-03-05 0.1331463854 0.81686162 0.02770999
## 2018-03-06 0.0834495589 0.84423493 0.03642278
## 2018-03-07 0.0518665617 0.85322263 0.05074193
## 2018-03-08 0.0063045943 0.59766518 0.38376283
## 2018-03-09 0.0013694508 0.06039272 0.92864521
## 2018-03-12 0.0009021542 0.03092867 0.96449337
```

```
#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"
test_pred_class[is.na(test_pred_class)]<- "NoWhere"
```

```
#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
##
##
##           NoWhere  Up Down
##   NoWhere     107  17   22
##   Up           31 148    0
##   Down          7   0   78
##
## Overall Statistics
##
```

```
##                Accuracy : 0.8122
##                  95% CI : (0.771, 0.8488)
##     No Information Rate : 0.4024
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7101
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: NoWhere Class: Up Class: Down
## Sensitivity                  0.7379    0.8970      0.7800
## Specificity                  0.8528    0.8735      0.9774
## Pos Pred Value               0.7329    0.8268      0.9176
## Neg Pred Value               0.8561    0.9264      0.9323
## Prevalence                   0.3537    0.4024      0.2439
## Detection Rate               0.2610    0.3610      0.1902
## Detection Prevalence         0.3561    0.4366      0.2073
## Balanced Accuracy            0.7954    0.8852      0.8787
```

```
#generate trade signals using the same pattern as human psychology
signal<-ifelse(test_pred_class=="Up",1,ifelse(test_pred_class=="Down",-1, 0))

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

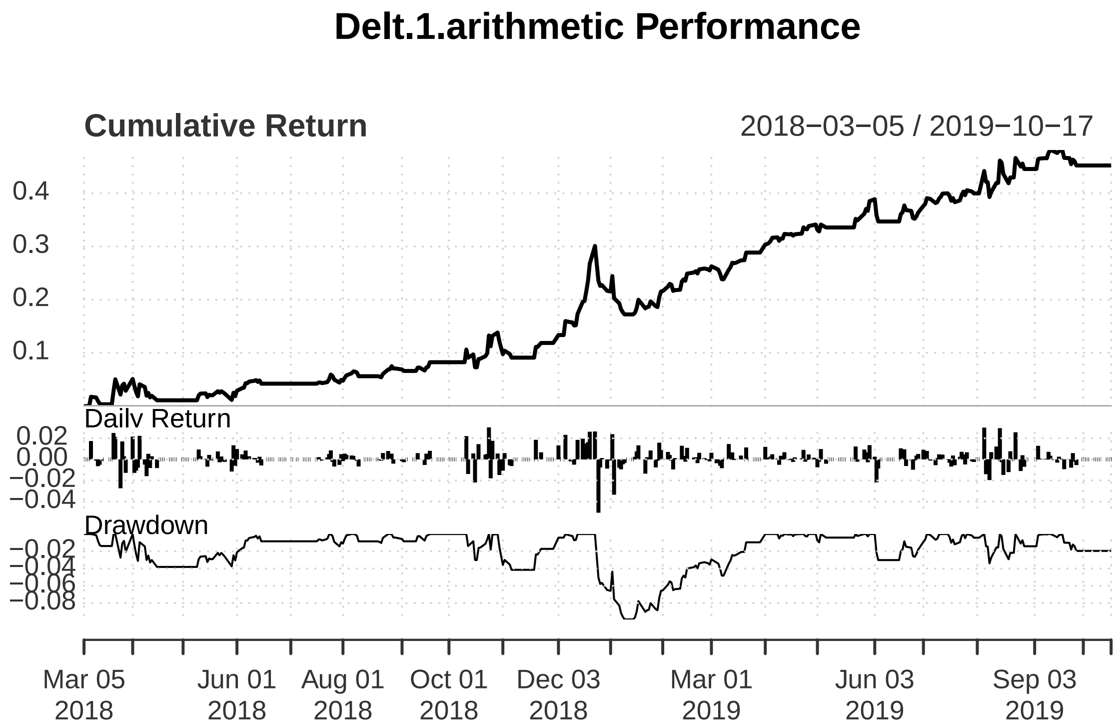
```
#calculate cummulative return
cumm_return<- Return.cumulative(test_return)
cumm_return
```

```
##                   Delt.1.arithmetic
## Cumulative Return         0.4522507
```

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

```
##                   Delt.1.arithmetic
## Annualized Return         0.2577557
```

```
charts.PerformanceSummary(test_return)
```




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

```
##     Delt.1.arithmetic
## VaR       -0.01137543
```

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

```
##                                    [,1]
## StdDev Sharpe (Rf=0%, p=95%): 0.1178555
```

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

```
##                                 Delt.1.arithmetic
## Annualized Sharpe Ratio (Rf=0%)          2.030721
```

Conclusion

In this group work project, adopted methodologies from several articles, we have implemented algorithmic trading strategy based upon neural network predictive model. Upon including, Google Trend data and GARCH model, we also devise a new neural network model. As the new neural model's performance shown, although the new neural network model does notsignificantly outperform the former model in terms of accuracy, sensitivity and specificity, the new model prediction can produce stable returns as it gives lower standard deviation of Sharpe ratio. Apart from that, this implementation proves that we can incoporate several techniques from traditional time-series technical anaysis and advanced machine learning techniques into practical use for predictive model development.

Future Work

Our group would like to try adapt a methodology used to predict power usage by LSTM ("How to Develop Multi-Step LSTM Time Series Forecasting Models for Power Usage," 2019) into our use case scenario in this project.

References

Palaniappan, V. (2018, November 21). Neural Networks to Predict the Market. Retrieved from https://towardsdatascience.com/neural-networks-to-predict-the-market-c4861b649371

Choudhury, A. (2019, May 6). Stock Market Prediction by Recurrent Neural Network on LSTM Model. Retrieved from https://blog.usejournal.com/stock-market-prediction-by-recurrent-neural-network-on-lstm-model-56de700bff68

Zhang, K., Zhong, G., Dong, J., Wang, S., & Wang, Y. (2019). Stock Market Prediction Based on Generative Adversarial Network. Procedia Computer Science, 147, 400-406. doi:10.1016/j.procs.2019.01.256

Cross-Validation strategies for Time Series forecasting [Tutorial]. (2019, May 6). Retrieved from https://hub.packtpub.com/cross-validation-strategies-for-time-series-forecasting-tutorial/

Measures of Predictive Models: Sensitivity and Specificity. (2018, January 5). Retrieved from https://www.theanalysisfactor.com/sensitivity-and-specificity/

Arnerić, J., Poklepović, T., & Aljinović, Z. (2014). GARCH based artificial neural networks in forecasting conditional variance of stock returns. Croatian Operational Research Review, 5(2), 329-343. doi:10.17535/crorr.2014.0017

Lu, X., Que, D., & Cao, G. (2016). Volatility Forecast Based on the Hybrid Artificial Neural Network and GARCH-type Models. Procedia Computer Science, 91, 1044-1049. doi:10.1016/j.procs.2016.07.145

Boyle, T. (2019, February 16). Hyperparameter Tuning. Retrieved from https://towardsdatascience.com/hyperparameter-tuning-c5619e7e6624

How to Develop Multi-Step LSTM Time Series Forecasting Models for Power Usage. (2019, August 5). Retrieved from https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-househ