# Main

Group 3

4/6/2018

```r
if(!require("ggplot2")){
  install.packages("ggplot2")
}

## Loading required package: ggplot2

library(ggplot2)
library("xts")

## Loading required package: zoo

##
## Attaching package: 'zoo'

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

library("dygraphs")
library("tseries")
library("forecast")
library("Metrics")

##
## Attaching package: 'Metrics'

## The following object is masked from 'package:forecast':
##
##     accuracy

library(tidyr)
library("dplyr")

##
## Attaching package: 'dplyr'

## 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
```

```
library("fifer")
```

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
# install.packages("PCAmixdata")
library("PCAmixdata")
# install.packages("FactoMineR")
library("FactoMineR")
# install.packages("dummies")
library("dummies")
```

```
## dummies-1.5.6 provided by Decision Patterns
```

## Step 0: Specify Directories and Source

```
# for macbook users
# setwd("~/Documents/GitHub/project-4-open-project-group3/doc")
source("../lib/functions.R")

# for PC users
# setwd("E:/GitHub/project-4-open-project-group3/doc")
# source("E:/GitHub/project-4-open-project-group3/lib/functions.R")
```

## Step 1: Setup Controls

## Step 2: Load Data and Make Features

## Step 2.1: Data prepocessing

- Note that test.csv only contains the first 50 time periods of prices and do not disclose the actual prices of the prices of t51-100. This is why we need to set aside a test set ourselves (using different test set from what other competitors used).
- Please download the data files using the link to the google drive. The data is too large to upload to the github.

```
###############################################################
#################### Stratified Sampling ######################
###############################################################

if(read_data){
  # Loading the data takes a long time so you could direcly load the Rdata.
  train <- read.csv("training.csv")
```

```r
  save(train, file = "train.Rdata")

  load("train.Rdata")

  # Since security 81 only has 300+ rows and is not enough for our stratified sampling, we eliminated these rows.
  train<-train[!train$security_id==81,]

  set.seed(1)
  # We used stratified sampling to get equal proportion of security_id's.
  sample <- train %>%
    group_by(security_id) %>%
    sample_n(size = round(50000/101))

  # Also get the unsampled set, so that we can stratified sample from the unsampled set.
  unsampled <- train[!train$row_id %in% sample$row_id,]

  test <- unsampled %>%
    group_by(security_id) %>%
    sample_n(size = round(10000/101))

  table(sample$security_id)
  table(test$security_id)

  sample_train <- sample %>%
    group_by(security_id) %>%
    sample_frac(size = 0.75)
  sample_test <- sample[!sample$row_id %in% sample_train$row_id,]

  save(sample, file ="../data/sample.Rdata")
  save(test,file = "../data/test.Rdata")
  save(sample_train, file = "../data/sample_train.Rdata")
  save(sample_test, file = "../data/sample_test.Rdata")
}else{
  # If directly loading the data files
  load("../data/sample_train.Rdata")
  load("../data/sample_test.Rdata")
}
```

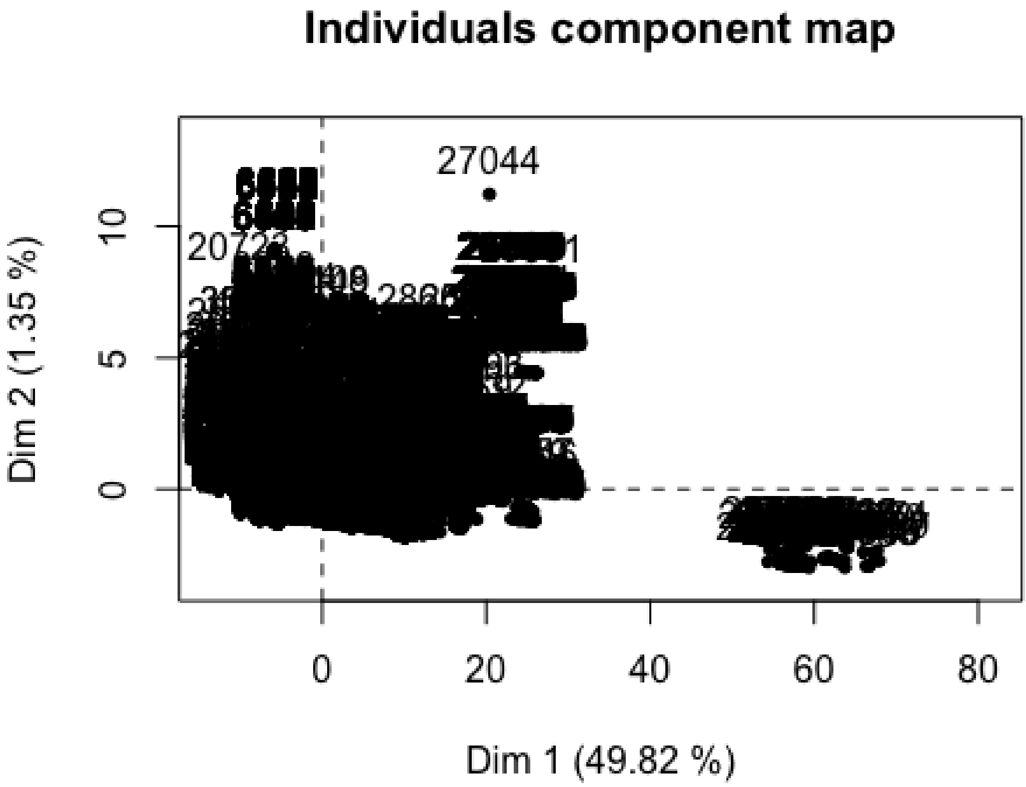
### Step 2.2: Baseline Feature

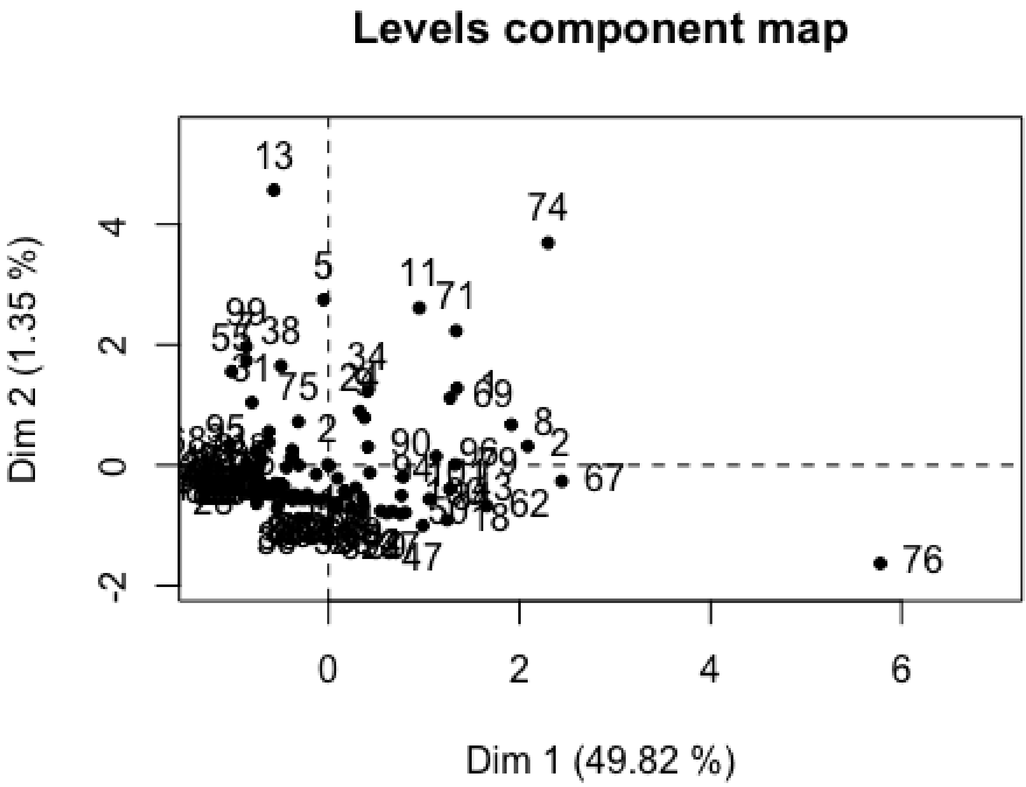
According to the discussion panel of the Kaggle competition, the most important columns in the data are: * trade_vwap * bid41 * bid50 * ask50 Therefore, our baseline feature is these four columns in the dataset.

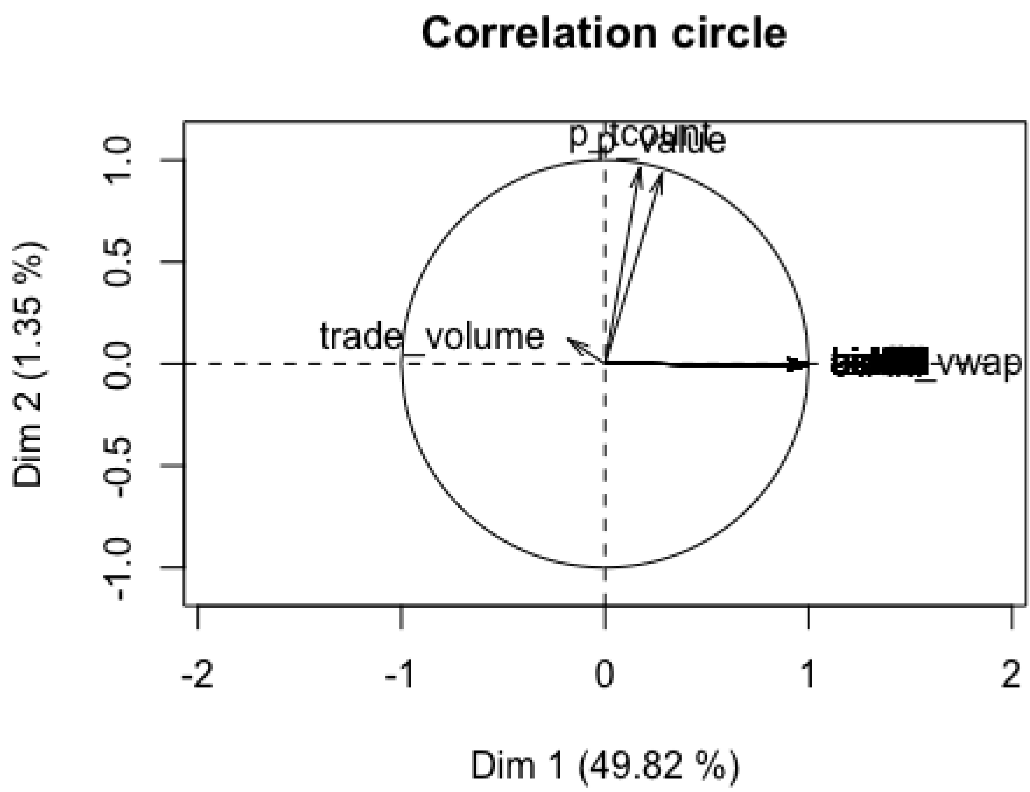
### Step 2.3: PCA Feature

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. * Note that PCA normally deals with all numerical data whereas in our case we have categorical data (e.g security_id) and binary data (e.g initiator). This requires us to perform a slightly different way of decomposing the dimensions. * I tried two packages to compute PCA for mixed data, one that automatically detects the categorical columns and one manually.

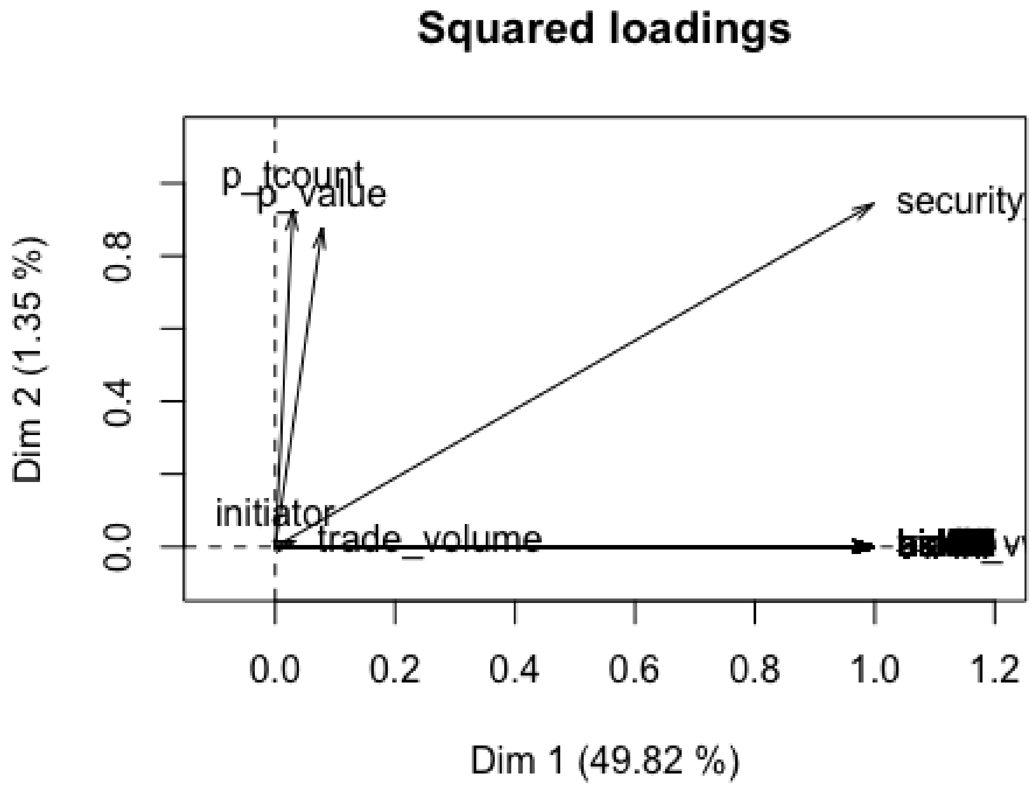
```
##############################################################
########################### PCA ##############################
##############################################################




# I first tried automatically using PCA mix
train_clean <- sample_train[,1:207]
train_clean <- train_clean[,!(grepl("time",colnames(train_clean)) |
grepl("transtype",colnames(train_clean)))]
train_clean <- train_clean[,-1]
train_clean$security_id <- as.factor(train_clean$security_id)
X.quali <- train_clean %>%
  dplyr:: select(c(security_id,initiator))
X.quali <- apply(as.matrix(X.quali),2,as.character)
X.quanti <- train_clean[,-c(1,6)]
X.quanti <- apply(as.matrix(X.quanti),2,as.numeric)
X.quanti <- scale(X.quanti)

PCA <- PCAmix(X.quanti, X.quali, rename.level = F, graph = T)
```


Individuals component map
27044
20723
Dim 2 (1.35 %)
Dim 1 (49.82 %)
0
5
10
0
20
40
60
80

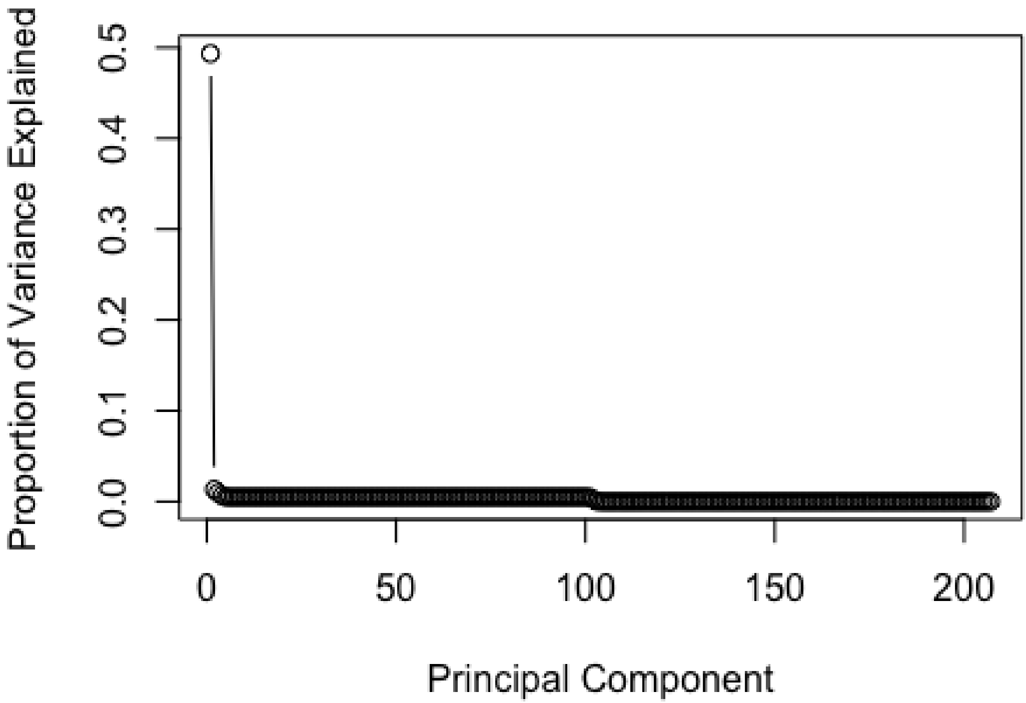

Levels component map
Dim 1 (49.82 %)
Dim 2 (1.35 %)
13
74
5
11
71
99
38
55
34
31
75
69
8
2
90
96
67
62
18
47
76
0
2
4
6
-2




```
## Warning in graphics::arrows(0, 0, res.pca$sqload[j, dim1], res.pca
## $sqload[j, : zero-length arrow is of indeterminate angle and so skipped
```




```
# we can see that p_tcount, p_value, trade_vwap and trade_volume are the most
important ones for dimension 2-5.


# Get the PCA new predictors as features for train and for test
train_pca <- dummy.data.frame(as.data.frame(train_clean), names =
c("security_id","initiator"))
prin_comp <- prcomp(train_pca, scale. = T)
prop_varex <- prin_comp$sdev ^ 2/sum(prin_comp$sdev ^ 2)
plot(prop_varex, xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     type = "b")
```




```
train_pca <- prin_comp$x[,c(1,2)]

test_clean <- sample_test[,1:207]
test_clean <- test_clean[,!(grepl("time",colnames(test_clean)) |
grepl("transtype",colnames(test_clean)))]
test_clean <- test_clean[,-1]
test_pca <- dummy.data.frame(as.data.frame(test_clean), names =
c("security_id","initiator"))
test_pca <- predict(prin_comp,test_pca)
test_pca <- test_pca[,c(1,2)]

save(train_pca, file = "../output/train_pca.Rdata")
save(test_pca, file = "../output/test_pca.Rdata")
```

### Step 2.4: Time Series Data Processing

In training set, every security has 371 rows. There're 101 securities in total, lack security 81. (1-80,82-102)

In test set, every security has 124 rows.

Every row has 100 bid prices and 100 ask prices. The latter 50 bid and ask prices are the response after liquidity shock. In training set we will contain that in the input of time series model but in the test set that's what we will predict.

We created a time series matrix by combining all the bid and ask price of rows belong to the same security into a long column. So the dimension of training matrix is (371*100) rows and (101*2) columns and the test matrix is (124*100) rows and (101*2) columns.

Functions: + data_to_ts

```
#################################################################
################## Making time series data ####################
#################################################################
if(process_data){

# initiate time series matrix
ts_train_matrix <- matrix(NA,nrow=37100,ncol=204)
ts_test_matrix <- matrix(NA,nrow=12400,ncol=204)

# transform data into time series matrix
ts_train_matrix <- data_to_ts(sample_train,ts_train_matrix)
ts_test_matrix <- data_to_ts(sample_test,ts_test_matrix)

# Check if all space on matrix is correctly filled in
sum(is.na(ts_train_matrix))
sum(is.na(ts_test_matrix))

# same time series matrix
save(ts_train_matrix, file = "../output/ts_train_matrix.Rdata")
save(ts_test_matrix,file="../output/ts_test_matrix.RData")
}else{
load("../output/ts_train_matrix.Rdata")
load("../output/ts_test_matrix.Rdata")
}
```

### Step 2.5: Lasso Data Matrix Creation

Limitations of lasso: impute data must be matrix. So I only use p_tcount, p_value, trade_vwap, trade_volume, 50 bid and ask prices as predictors, and this arrangement will lose information on initiator and transtype of per bid and ask. That's the trade-off.

```
  # initiate training matrix
  train.mat <- matrix(NA,nrow=37471,ncol=204)
  train.mat[,1:4] <- as.matrix(sample_train[,3:6])
  train.mat[,105:204] <- as.matrix(sample_train[,208:307])

  for (i in 1:50) {
    train.mat[,(2*i+3)] <- as.matrix(sample_train[,(6+i*4)])
    train.mat[,(2*i+4)] <- as.matrix(sample_train[,(7+i*4)])
  }
```

```
  # initiate test matrix
  test.mat <- matrix(NA,nrow=12524,ncol=204)
  test.mat[,1:4] <- as.matrix(sample_test[,3:6])
  test.mat[,105:204] <- as.matrix(sample_test[,208:307])

  for (i in 1:50) {
    test.mat[,(2*i+3)] <- as.matrix(sample_test[,(6+i*4)])
    test.mat[,(2*i+4)] <- as.matrix(sample_test[,(7+i*4)])
  }

  # initiate prediction matrix
  pred.mat <- test.mat
  pred.mat[,105:204] <- NA
```

## Step 3: Data Visualization

### Step 3.1: Visualization for panel data

```
if(run_visual)
{
  #Creates the vwap visuals
  sample_train$security_id <- factor(sample_train$security_id)
  vwap_train <- ggplot(sample_train, aes(x = security_id, y = trade_vwap,
color = trade_vwap)) + geom_point() +     scale_color_gradient(low =
"lightgreen", high = "darkblue") +
    theme(axis.text.x=element_blank(),
          axis.ticks.x=element_blank())
  vwap_train
  ggsave("../figs/vwap_train.png")


  vwap_test <- ggplot(sample_test, aes(x = security_id, y = trade_vwap, color
= trade_vwap)) + geom_point() +     scale_color_gradient(low = "lightgreen",
high = "darkblue") +
    theme(axis.text.x=element_blank(),
          axis.ticks.x=element_blank())
  vwap_test
  ggsave("../figs/vwap_test.png")

  #Creates the price visuals for security 25, 50, 75, and 100
  subset_train <- sample_train[sample_train$security_id == 25 |
sample_train$security_id == 50 | sample_train$security_id == 75 |
sample_train$security_id == 100,]
  subset_test <- sample_test[sample_test$security_id == 25 |
sample_test$security_id == 50 |
                               sample_test$security_id == 75 |
sample_test$security_id == 100,]

  #Dataframes separated to bid and ask prices
```

```
  train_price_bid <- data.frame(NA, nrow = 400, ncol = 3)
  colnames(train_price_bid) <- c("Security_Id", "Event_t", "Price")
  train_price_ask <- data.frame(NA, nrow = 400, ncol = 3)
  colnames(train_price_ask) <- c("Security_Id", "Event_t", "Price")

  #Temp dataframes to be added to the main dataframes.
  train_price_bid_temp <- data.frame(NA, nrow = 100, ncol = 3)
  colnames(train_price_bid_temp) <- c("Security_Id", "Event_t", "Price")

  train_price_ask_temp <- data.frame(NA, nrow = 100, ncol = 3)
  colnames(train_price_ask_temp) <- c("Security_Id", "Event_t", "Price")

  #Security 25
  for(i in 1:50)
  {
    train_price_bid[i, 1] <- 25
    train_price_bid[i, 2] <- i
    train_price_bid[i, 3] <- colMeans(subset_train[subset_train$security_id
== 25, 6 + i*4])
    train_price_ask[i, 1] <- 25
    train_price_ask[i, 2] <- i
    train_price_ask[i, 3] <- colMeans(subset_train[subset_train$security_id
== 25, 7 + i*4])
  }
  for(i in 51:100)
  {
    train_price_bid[i, 1] <- 25
    train_price_bid[i, 2] <- i
    train_price_bid[i, 3] <- colMeans(subset_train[subset_train$security_id
== 25, 106 + i*2])
    train_price_ask[i, 1] <- 25
    train_price_ask[i, 2] <- i
    train_price_ask[i, 3] <- colMeans(subset_train[subset_train$security_id
== 25, 107 + i*2])
  }

  #Security 50
  for(i in 1:50)
  {
    train_price_bid_temp[i, 1] <- 50
    train_price_bid_temp[i, 2] <- i
    train_price_bid_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 50, 6 + i*4])
    train_price_ask_temp[i, 1] <- 50
    train_price_ask_temp[i, 2] <- i
    train_price_ask_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 50, 7 + i*4])

  }
```

```
  for(i in 51:100)
  {
    train_price_bid_temp[i, 1] <- 50
    train_price_bid_temp[i, 2] <- i
    train_price_bid_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 50, 106 + i*2])
    train_price_ask_temp[i, 1] <- 50
    train_price_ask_temp[i, 2] <- i
    train_price_ask_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 50, 107 + i*2])
  }

  train_price_bid <- rbind(train_price_bid, train_price_bid_temp)
  train_price_ask <- rbind(train_price_ask, train_price_ask_temp)

  #Security 75
  for(i in 1:50)
  {
    train_price_bid_temp[i, 1] <- 75
    train_price_bid_temp[i, 2] <- i
    train_price_bid_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 75, 6 + i*4])
    train_price_ask_temp[i, 1] <- 75
    train_price_ask_temp[i, 2] <- i
    train_price_ask_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 75, 7 + i*4])

  }
  for(i in 51:100)
  {
    train_price_bid_temp[i, 1] <- 75
    train_price_bid_temp[i, 2] <- i
    train_price_bid_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 75, 106 + i*2])
    train_price_ask_temp[i, 1] <- 75
    train_price_ask_temp[i, 2] <- i
    train_price_ask_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 75, 107 + i*2])
  }

  train_price_bid <- rbind(train_price_bid, train_price_bid_temp)
  train_price_ask <- rbind(train_price_ask, train_price_ask_temp)

  #Security 100
  for(i in 1:50)
  {
    train_price_bid_temp[i, 1] <- 100
    train_price_bid_temp[i, 2] <- i
    train_price_bid_temp[i, 3] <-
```

```r
colMeans(subset_train[subset_train$security_id == 100, 6 + i*4])
    train_price_ask_temp[i, 1] <- 100
    train_price_ask_temp[i, 2] <- i
    train_price_ask_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 100, 7 + i*4])

  }
  for(i in 51:100)
  {
    train_price_bid_temp[i, 1] <- 100
    train_price_bid_temp[i, 2] <- i
    train_price_bid_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 100, 106 + i*2])
    train_price_ask_temp[i, 1] <- 100
    train_price_ask_temp[i, 2] <- i
    train_price_ask_temp[i, 3] <-
colMeans(subset_train[subset_train$security_id == 100, 107 + i*2])
  }

  train_price_bid <- rbind(train_price_bid, train_price_bid_temp)
  train_price_ask <- rbind(train_price_ask, train_price_ask_temp)



  train_price_bid_plot <- ggplot(train_price_bid, aes(x = Event_t, y = Price,
color = factor(Security_Id))) + geom_line() +
    theme(axis.text.x=element_blank(),
          axis.ticks.x=element_blank()) + facet_wrap( ~ Security_Id, ncol =
2, scales = "free") + guides(color = guide_legend(title = "Security ID"))

  train_price_ask_plot <- ggplot(train_price_ask, aes(x = Event_t, y = Price,
color = factor(Security_Id))) + geom_line() +
    theme(axis.text.x=element_blank(),
          axis.ticks.x=element_blank()) + facet_wrap( ~ Security_Id, ncol =
2, scales = "free") + guides(color = guide_legend(title = "Security ID"))

  train_price_bid_plot
  ggsave("../figs/price_bid_train.png")
  train_price_ask_plot
  ggsave("../figs/price_ask_train.png")
}
```

```
## Saving 5 x 4 in image
## Saving 5 x 4 in image
## Saving 5 x 4 in image
## Saving 5 x 4 in image
```

### Step 3.2: Time series visualization

- It can be seen that the time series is non-stationary,non-seasonal and not normal if we just use single row as a time series.
- After we convert to a long time series, we get stationary, seasonal and normal time series

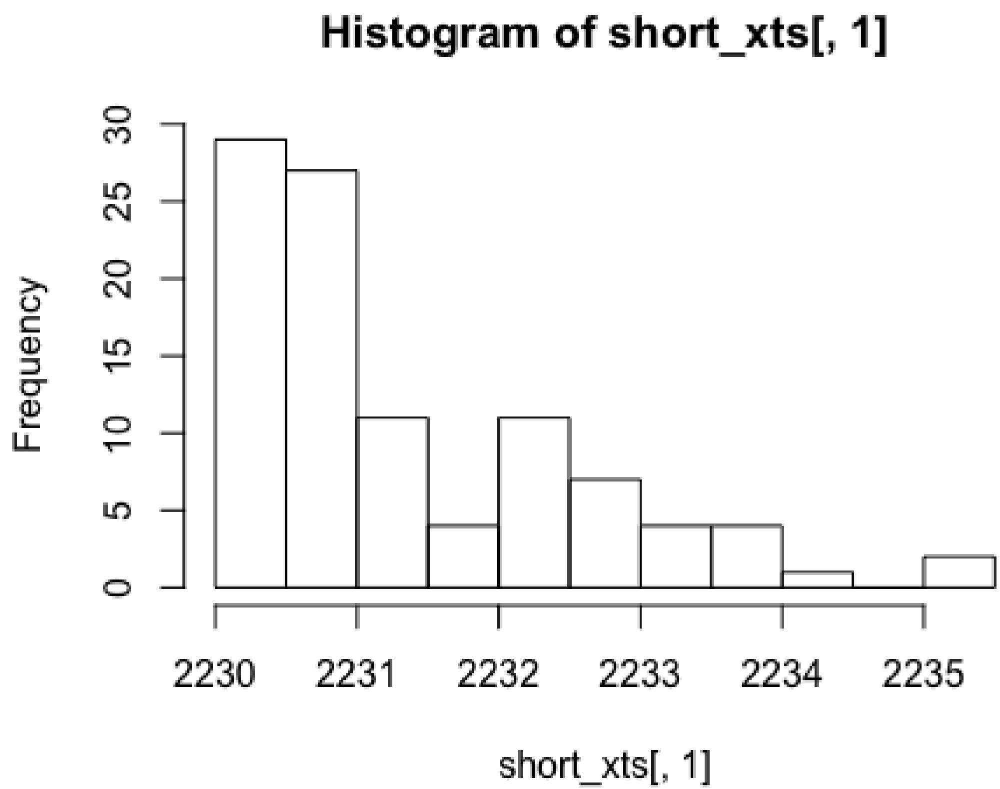
```
################################## Short Time Series
################################

  #This is what we tried to do at first
load("../output/short_mat.Rdata")
times <- seq(as.Date("2017-05-01"),length=100,by="days")
short_xts <- xts(short_mat,order.by = times)
dygraph(short_xts[,1:2],main = 'Short version') %>%
  dyRangeSelector()  %>%
  dyOptions(axisLineWidth = 1.5, fillGraph = FALSE, drawGrid = T,
rightGap=50)
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs().
If it is installed, please make sure the phantomjs executable can be found
via the PATH variable.
```

```
# Doing ARIMA prediction for the first time series - short version
# Check by Dickey-fuller test, we can not reject the null hypothesis
adf.test(short_xts[,1], "stationary")
```

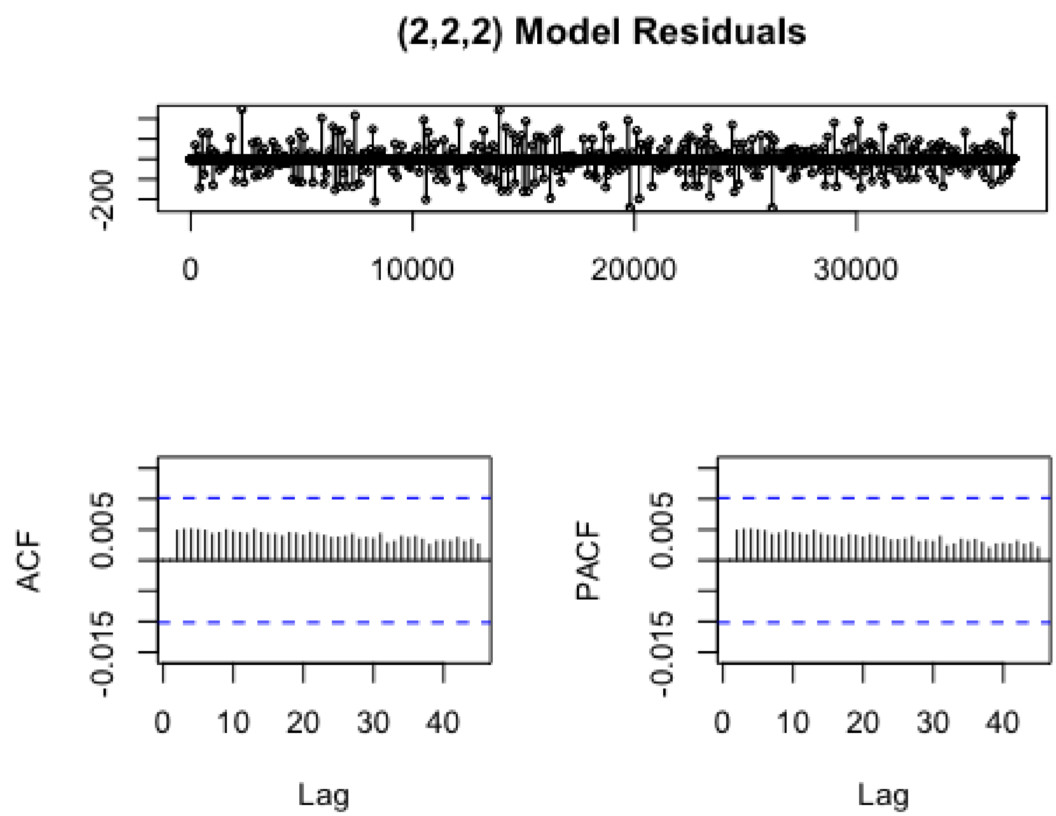
```
##
##  Augmented Dickey-Fuller Test
##
## data:  short_xts[, 1]
## Dickey-Fuller = -0.3068, Lag order = 4, p-value = 0.9888
## alternative hypothesis: stationary
```

```
# Problem: data is not normal, in fact extremely skewed
hist(short_xts[,1])
```

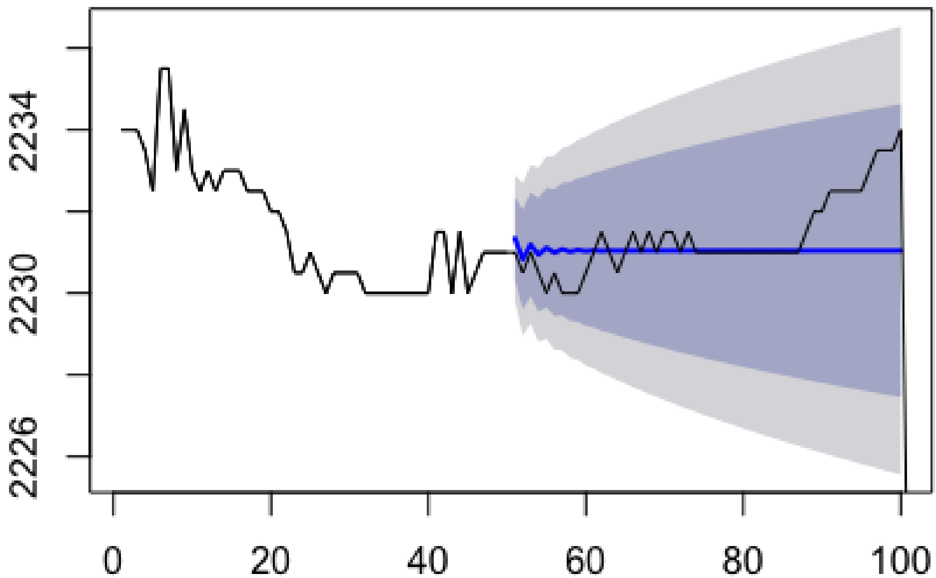



```
################################## Long Time Series
################################
times <- seq(as.Date("2017-05-01"),length=37100,by="days")
train_xts <- xts(ts_train_matrix,order.by = times)
dygraph(train_xts[,1:2],main = 'Long version') %>%
  dyRangeSelector()  %>%
  dyOptions(axisLineWidth = 1.5, fillGraph = FALSE, drawGrid = T,
rightGap=50)

# Doing ARIMA prediction for the first time series
# Check by Dickey-fuller test, we reject the null hypothesis
adf.test(train_xts[,1], "stationary")

## Warning in adf.test(train_xts[, 1], "stationary"): p-value smaller than
## printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  train_xts[, 1]
## Dickey-Fuller = -14.797, Lag order = 33, p-value = 0.01
## alternative hypothesis: stationary
```

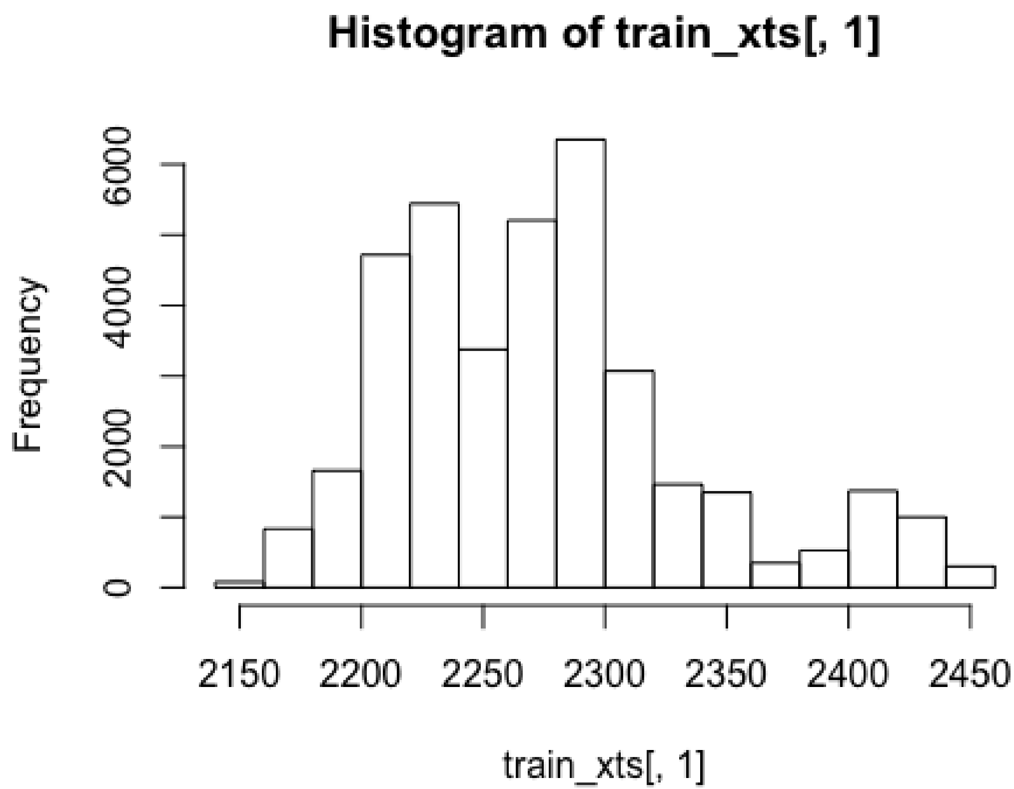
```r
# auto.arima automatically searches for the optimal p and q to be used in the ARIMA model
fit <- auto.arima(train_xts[,1], seasonal = F, max.p = 10, max.q = 10)

tsdisplay(residuals(fit), lag.max=45, main='(2,2,2) Model Residuals')
```




```r
# See how the model would perform on the 51-100 time interval
hold <- window(ts(train_xts[,1]), start=51)

fit_first_half = auto.arima(train_xts[1:50,1])

fcast_second_half <- forecast(fit_first_half,h=50)
plot(fcast_second_half, main=" ")
lines(ts(train_xts[,1]))
```




```
# Check: data is normal
hist(train_xts[,1])
```




### Step 4: Create Models

### Step 4.1: Model fitting

**Note that since the models are too large, we moved it to the google drive, together with the data files.**

```
if(create_models)
{
  if(run_LR)
  {

    LR_models <- create_LR_models(sample_train)
    save(LR_models, file = "../output/LR_models.RData")
  }
  if(run_LR_PCA)
  {
    load("../output/train_pca.RData")
    load("../output/test_pca.RData")
    LR_PCA_models <- create_LR_PCA_models(sample_train, train_pca)
    save(LR_PCA_models, file = "../output/LR_PCA_models.RData")
  }
  if(run_RF)
```

```
  {
    RF_models <- rf_train(sample_train)
    save(RF_models, file = "../output/RF_models.RData")
  }
  if(run_GBM){
    gbm_model <- gbm_train(sample_train)
    save(gbm_model, file = "../output/GBM_model.RData")
  }
  if(run_SVM){
    svm_model <- svm_train(sample_train)
    save(svm_model, file = "../output/SVM_model.RData")
  }
  if(run_GBM)
  {
    load("../output/train_pca.RData")
    load("../output/test_pca.RData")
    gbm_pca <- gbm_train_PCA(train_pca)
    save(gbm_pca, file = "../output/GBM_PCA.RData")
  }
  if(run_SVM)
  {
    load("../output/train_pca.RData")
    load("../output/test_pca.RData")
    svm_pca <- svm_train_PCA(train_pca)
    save(svm_pca, file = "../output/SVM_PCA.RData")
  }
}
if(load_model){

  load("../data/GBM_model.Rdata")
  load("../data/SVM_model.Rdata")
  load("../data/GBM_PCA.Rdata")
  load("../data/SVM_PCA.Rdata")
  load("../data/RF_models.Rdata")
  load("../output/LR_models.Rdata")
  load("../output/LR_PCA_models.Rdata")

}
```

### Step 4.2: Time series ARIMA modeling and prediction

We wrote model fitting and predicting in the same for loop and didn't save the models (there are too many of them).

Function: + predict_ts

```
if(create_models){

  # copy a new test_matrix to calculate prediction rmse.
  ts_pred_matrix <- ts_test_matrix
```

```
  dim(ts_pred_matrix)

  # turn the 50-100 response chunks into NA for every 100 rows.
  for (i in 1:124) {
    ts_pred_matrix[((i-1)*100+51):(i*100),] <- NA
  }

  # prediction
  ts_pred_matrix <- predict_ts(ts_train_matrix,ts_pred_matrix)

  # check if all space on matrix is correctly filled in
  sum(is.na(ts_pred_matrix))

  save(ts_pred_matrix,file="../output/ts_pred_matrix.RData")
}else{
  load("../output/ts_pred_matrix.Rdata")
}
```

### Step 4.3: Lasso modeling and prediction

We wrote model fitting and predicting in the same for loop and didn't save the models (can't afford to save them, it's too large).

```
if(create_models){

  # run prediction on lasso model
  pred.mat <- predict_lasso(train.mat,pred.mat)

  # check if the matrix has been filled in correctly
  sum(is.na(pred.mat))

  # save prediction matrix
  save(pred.mat,file="../output/lasso_pred_matrix.RData")
}else{
  load("../output/lasso_pred_matrix.Rdata")
}
```

## Step 5: Make Predictions

```
if(make_pred)
{
  if(run_LR)
  {
    load("../output/LR_models.RData")

    LR_predictions <- make_LR_predictions(LR_models, sample_test)
    save(LR_predictions, file = "../output/LR_predictions.RData")
  }
  if(run_LR_PCA)
  {
```

```
    load("../output/LR_PCA_models.RData")
    LR_PCA_predictions <- make_LR_PCA_predictions(LR_PCA_models, sample_test,
test_pca)
    save(LR_PCA_predictions, file = "../output/LR_PCA_predictions.RData")

  }
  if(run_RF){
    load("../output/RF_models.Rdata")
    RF_predict <- test_data(RF_models,sample_test)
    save(RF_predict, file = "../output/RF_predict.Rdata")
  }
  if(run_GBM){
  gbm_pre <- make_GBM_predictions(gbm_model, sample_test)
  save(gbm_pre, file = "../output/gbm_pre.Rdata")
  }
  if(run_SVM){
  svm_pre <- make_svm_predictions(svm_model, sample_test)
  save(svm_pre, file = "../output/svm_pre.Rdata")
}
}
if(load_model){
    load("../output/LR_predictions.RData")
    load("../output/LR_PCA_predictions.RData")
    load("../output/RF_predict.Rdata")
    load("../output/svm_pre.Rdata")
    load("../output/gbm_pre.Rdata")

}
```

## Step 6: Calculate RMSE

```
if(calc_RMSE)
{
  if(run_LR)
  {
    load("../output/LR_predictions.RData")
    LR_RMSE <- evaluate_RMSE(LR_predictions, sample_test)
    save(LR_RMSE, file = "../output/LR_RMSE.RData")
  }
  if(run_LR_PCA)
  {
    load("../output/LR_PCA_predictions.RData")
    LR_PCA_RMSE <- evaluate_RMSE(LR_PCA_predictions, sample_test)
    save(LR_PCA_RMSE, file = "../output/LR_PCA_RMSE.RData")
  }
  if(run_RF){
    # 2.03
    load("../output/RF_models.Rdata")
    test_data_actual <- test_data_acutal(sample_test)
  }
  if(run_SVM){
```

```
      #951.846

    load("../output/svm_pre.Rdata")
  rmse_svm <- evalution(svm_pre,sample_test[,208:307])
  }
  if(run_GBM){
        #954. 634

    load("../output/gbm_pre.Rdata")
  rmse_gbm <- evalution(gbm_pre,sample_test[,208:307])
  }

}
```

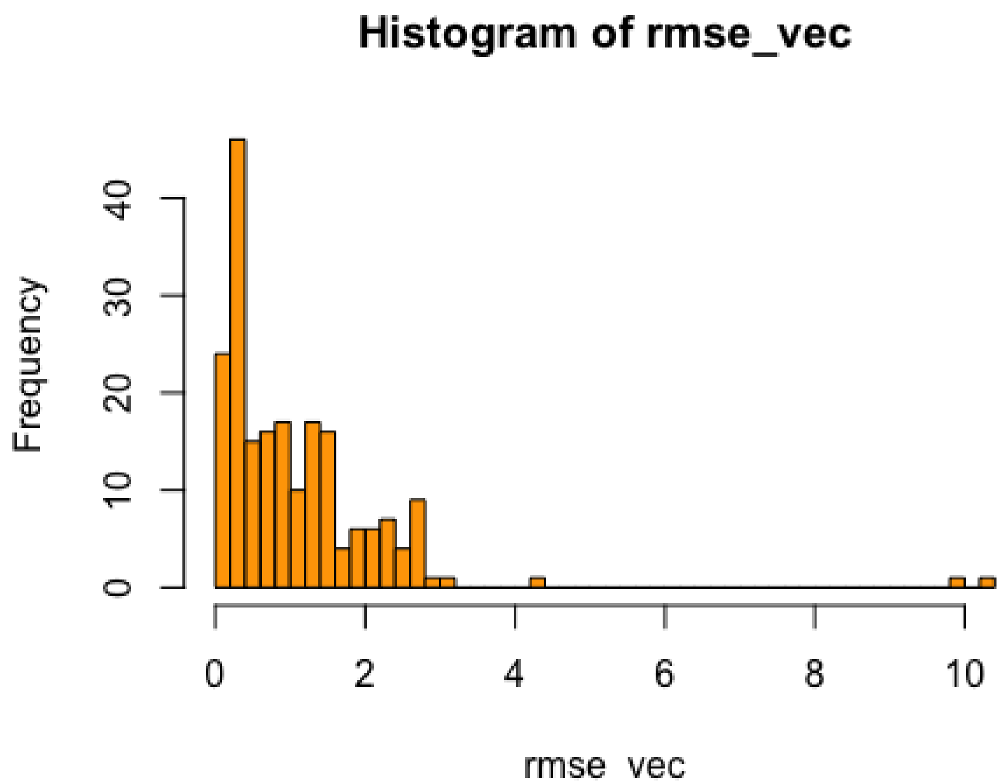
## Step 7: Summarize RMSE

### Step 7.1: RMSE for time series

calculate rmse for each column to plot a histogram of rmse to see the distribution.

It's obvious that the majority of rmse for each security is below 2. There're some outliers that impact our result. Here we tried to remove the outliers and calculate the mean rmse overall and it's around 0.96.

```
rmse_vec <- rep(NA,202)

for (i in 1:202) {
  rmse_vec[i] <- rmse(pred_matrix[,i],ts_test_matrix[,i])
}

hist(rmse_vec,breaks=40,col="orange",border="black")
```




```
# remove the outlier to see the overall rmse
mean(rmse_vec[which(rmse_vec<3)])

## [1] 0.959165
```

### Step 7.2: RMSE for Linear Regression

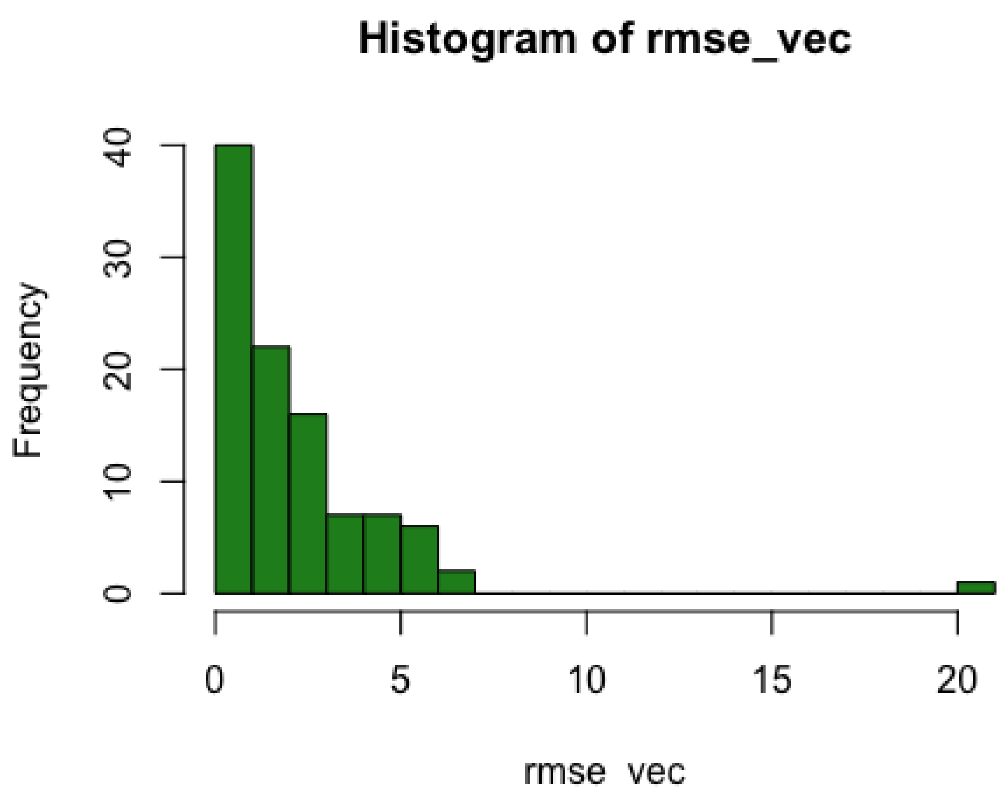
```
if(sum_RMSE)
{
  if(run_LR)
  {
    load("../output/LR_RMSE.RData")
    cat("RMSE for LR Model =", LR_RMSE, "\n")
  }
  if(run_LR_PCA)
  {
    load("../output/LR_PCA_RMSE.RData")
    cat("RMSE for LR_PCA Model =", LR_PCA_RMSE, "\n")
  }

}
```

### Step 7.3: RMSE for Lasso

calculate rmse for each security to plot a histogram of rmse to see the distribution.

It's obvious that the majority of rmse for each security is below 5. There're some outliers that impact our result. Here we tried to remove the outliers and calculate the mean rmse overall and it's around 1.94.

```
# initiate a rmse vector to store the rmse for every security
rmse_vec <- rep(NA,101)

for (i in 1:101) {
  rmse_vec[i] <- rmse(pred.mat[(124*(i-1)+1):(124*i),],test.mat[(124*(i-
1)+1):(124*i),])
}

hist(rmse_vec,breaks=20,col="forestgreen",border="black")
```




```
# remove the outlier to see the overall rmse
mean(rmse_vec[which(rmse_vec<7)])

## [1] 1.942743
```