Main

Group 3

4/6/2018

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")  
 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")  
}

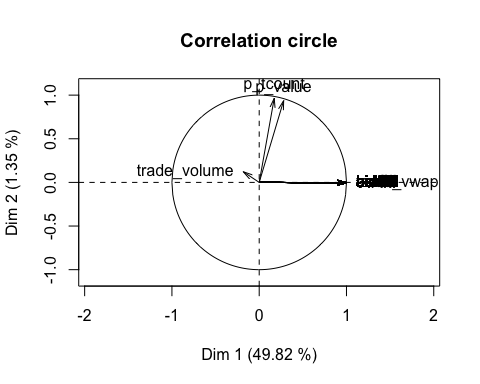
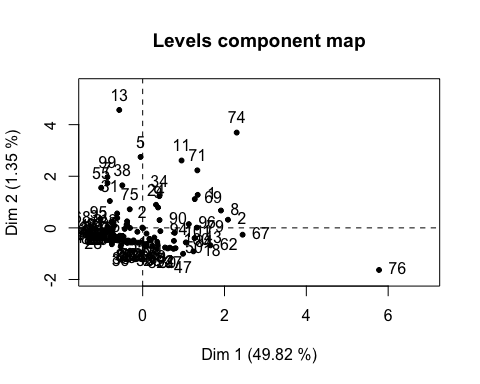
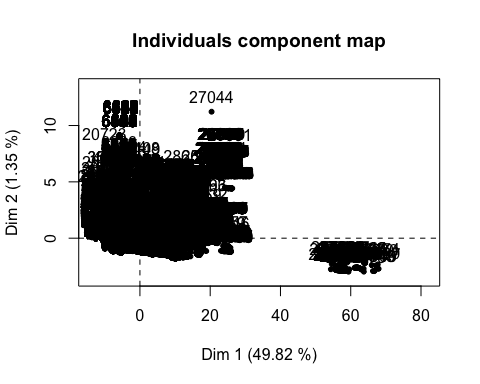
### Step 2.2: Baseline Feature

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.

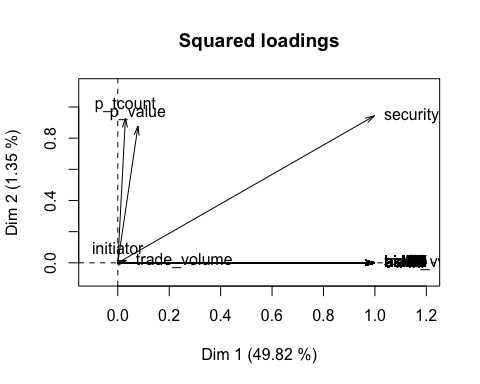
### Step 2.3: PCA Feature

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.

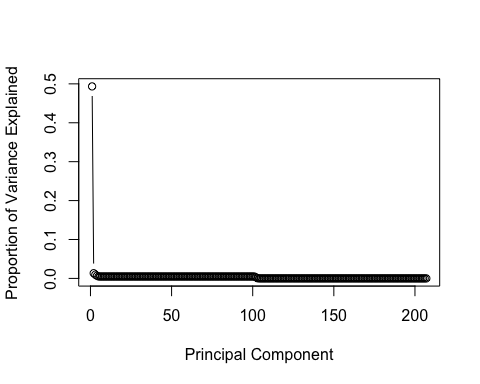
###############################################################  
############################ PCA ##############################  
###############################################################  
  
  
  
# I first tried automatically using PCA mix  
train\_clean <- sample\_train[,1:207]  
train\_clean <- 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)



## 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] <- 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

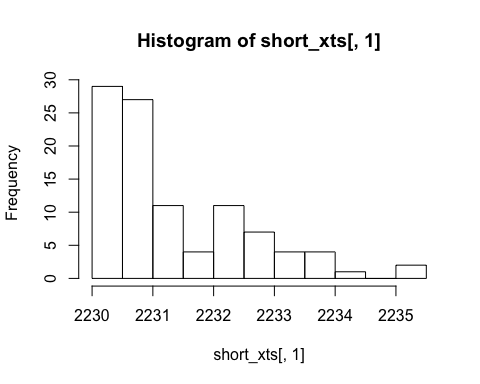
################################# Short Time Series ###############################  
  
 #This is what we tried to do at first   
load("../output/short\_mat.Rdata")  
times <- seq(as.Date("2017-05-01"),length=100,by="days")  
short\_xts <- xts(short\_mat,order.by = times)  
dygraph(short\_xts[,1:2],main = '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])



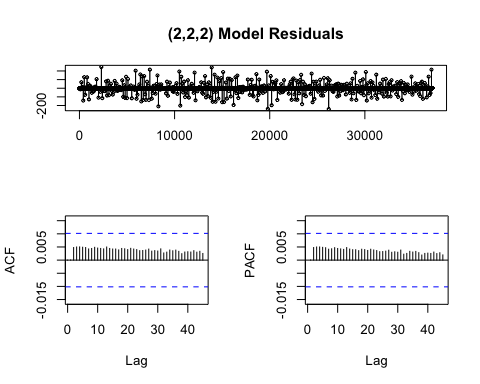
################################# Long Time Series ###############################  
times <- seq(as.Date("2017-05-01"),length=37100,by="days")  
train\_xts <- xts(ts\_train\_matrix,order.by = times)  
dygraph(train\_xts[,1:2],main = 'Long version') %>%  
 dyRangeSelector() %>%   
 dyOptions(axisLineWidth = 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")

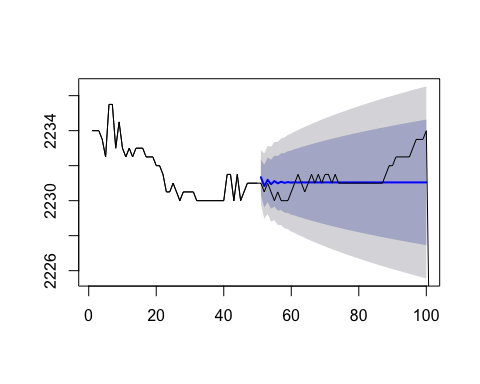
## Warning in adf.test(train\_xts[, 1], "stationary"): p-value smaller than  
## printed p-value

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

# 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')



# 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])  
 }  
   
}

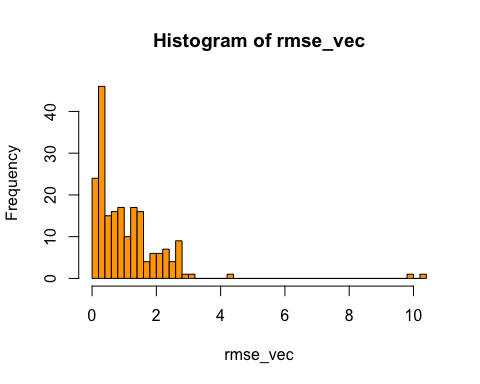
### Step 7: Summarize RMSE

### 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

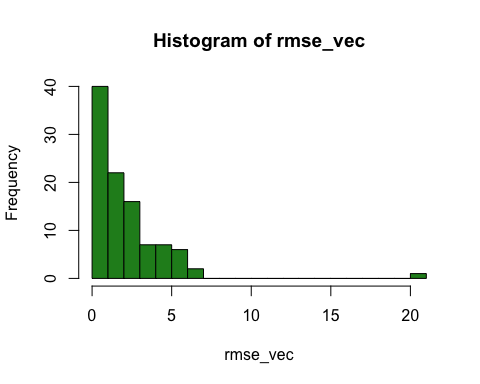
if(sum\_RMSE)  
{  
 if(run\_LR)  
 {  
 load("../output/LR\_RMSE.RData")  
 cat("RMSE for LR Model =", LR\_RMSE, "\n")  
 }  
 if(run\_LR\_PCA)  
 {  
 load("../output/LR\_PCA\_RMSE.RData")  
 cat("RMSE for LR\_PCA Model =", LR\_PCA\_RMSE, "\n")  
 }  
   
}

### 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