final_project

Arjun Raman, Alp Yel, Tanisha Pardashani, Charles Yu

12/3/2021

Part 1) Backward Returns

In this section we define a backward returns function that calculates $r_b(t,h)$ for a given range of h values and returns them as a data frame. We later use the function to calculate any range of backward returns needed.

Bret_calc Function

Since we will be calculating numerous different backward returns throughout the project, we will be automaticizing the process by defining a function that calculates backward returns for any given vector of h.

```
bret_calc <- function(back_h, data){
  assets_ids <- c(1,2,3)
  assets <- c("Asset_1","Asset_2","Asset_3")
  n_back_h <- length(back_h)

  # Repeat prices for Assets 1-3 n times times
  bret <- cbind(data[,paste(rep(assets,n_back_h))])

  #Get column names using a loop rather than manually
  bret_columns <- c()
  for (h in back_h) {
    for (i in assets_ids) {
      bret_columns <- c(bret_columns, paste(c("Asset_", i, "_BRet_", h), collapse = ""))
    }
  }
  colnames(bret) <- bret_columns

  #Pre_Allocate matrix for backward Prices
  BPrices <- data.frame(matrix(nrow = nrow(data), ncol = 0))

  #Calculate Backward Prices with condition max(t-h,1)
  for (h in back_h) {
    BPrices <- cbind(BPrices, data[c(rep(1, h), 1:(nrow(data) - h)), assets])
  }
  #Add in Column Names for Backward Prices
  BPrices_columns <- c()
  for (h in back_h) {
    for (i in assets) {
      BPrices_columns <- c(BPrices_columns, paste(c("Asset_", i, "_BPrice", h), collapse = ""))
    }
  }
  colnames(BPrices) <-BPrices_columns
```

```
  #Bret is simply [(s(t) - s(max(t-h,1)))]/s(max(t-h,1))
  bret <- (bret - BPrices)/BPrices
  return(bret)
}
```

Using the Bret function we wrote we will be calculating the 3,10, and 30 minute backward returns and write them into the *bret.csv*.

```
bret <- bret_calc(c(3,10,30), data)
#write.csv(bret, "bret.csv", row.names = FALSE)
```

Part 2) Backward Rolling Correlation

In this section we will be defining a rolling correlation function to calculate the pairwise 3-week backward rolling correlation between the assets

Roll_Corr Function

For tidyness and readability we will be writing a rolling correlation function.

```
roll_corr <- function(vec1, vec2, window_size){
  output <- rep(NA, length(v1))

  for (t_0 in 1:length(v1)){
    time_series <- max(t_0 -window_size,1):t0
    output[t_0] <- cor(vec1[time_series], vec2[time_series])
  }
  return(output)
}
```

Calculating Rolling Correlation

Using the rolling correlation function we defined we can calculate the pairwise 3 week rolling correlation between the assets.

```
#window_size <- 60*24*21
#corr <- data.frame(roll_corr(bret[, 1], bret[, 2], window_size,
#                   roll_corr(bret[, 1], bret[, 3], window_size,
#                   roll_corr(bret[, 2], bret[, 3], window_size,
#colnames(corr) <- c("Rho_1_2", "Rho_1_3", "Rho_2_3")
#write.csv(corr, "corr.csv", row.names = FALSE)
```

Given the slowness of loops in R, from this point on we will be using the roll_cor function from the *roll* library, implemented in C, for sake of time.

Part 3) Linear regression

In this section we use a linear model to predict the 10 minute forward returns of Asset 1 using the 3,10,30 minute backward returns of Assets 1,2,3 that we calculated in the preceding section.

Forward Returns

First we will be calculating the 10 minute forward returns of Asset 1 and combinig it with the backward returns data.

```r
fret10_Asset1_calc <-function(data){
  #Get Asset 1 Prices
  Asset_1_P <- data[, "Asset_1"]
  FPrices <- data[c(11:nrow(data), rep(nrow(data), 10)), 2]
  Asset_1_Fret <- (FPrices - Asset_1_P)/Asset_1_P
  colnames(Asset_1_Fret)<- "Asset_1_FRet_10"
  return(Asset_1_Fret)
}
Asset_1_Fret <- fret10_Asset1_calc(data)
fret_data <- cbind(bret, Asset_1_Fret)
```

Setting Training and Testing Data

Next we will be splitting the first 70% of our data as training and the remaining 30% as testing data.

```r
#Get training and testing set
train_size <- floor(nrow(bret) * 0.7)
fret_train <- fret_data[1:train_size, ]
fret_test <- fret_data[-(1:train_size), ]
```

Fitting the Linear Model

Now we will be fitting a linear model predicting 10 minute forward returns of Asset 1 using the 3,10,30 minute backward returns of assets 1,2,3. The summary of the model can be found below

```r
lm_fret <- lm(Asset_1_FRet_10 ~ ., data = fret_train)
summary(lm_fret)
```

```
##
## Call:
## lm(formula = Asset_1_FRet_10 ~ ., data = fret_train)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.147289 -0.000919  0.000010  0.000928  0.085484
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -1.036e-05  4.758e-06  -2.178 0.029391 *
## Asset_1_BRet_3   4.078e-02  4.077e-03  10.002  < 2e-16 ***
## Asset_2_BRet_3   2.591e-02  2.228e-03  11.626  < 2e-16 ***
## Asset_3_BRet_3   1.834e-02  2.247e-03   8.164 3.26e-16 ***
## Asset_1_BRet_10  1.708e-02  2.538e-03   6.730 1.70e-11 ***
## Asset_2_BRet_10 -5.405e-03  1.435e-03  -3.766 0.000166 ***
## Asset_3_BRet_10  3.418e-03  1.441e-03   2.372 0.017716 *
## Asset_1_BRet_30  6.252e-03  1.261e-03   4.958 7.11e-07 ***
## Asset_2_BRet_30  8.884e-03  7.609e-04  11.677  < 2e-16 ***
## Asset_3_BRet_30 -9.622e-04  7.295e-04  -1.319 0.187194
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.002882 on 366902 degrees of freedom
## Multiple R-squared:  0.004607,   Adjusted R-squared:  0.004583
## F-statistic: 188.7 on 9 and 366902 DF,  p-value: < 2.2e-16
```

Significance of Backward Returns

Looking at the summary of our linear model fitted above, we observe that the 3,10,30 minute backward returns for Asset 1 are all highly significant. The same goes for Asset 2, both 3,10, and 30 minute backward returns are highly significant. However, on the other hand, for Asset 3 only the 3 minute backward returns are significant, 10 and 30 minute backward returns are not significant.

We conclude that 3,10, and 30 minute backward returns of Asset 1 and 2 are highly significant in predicting the 10 minute forward returns of Asset 1; however, only the 3 minute forward returns of Asset 3 are significant in predicting the 10 minute forward returns of Asset 1. The 10 minute backward returns of Asset 3 are significant only at $\alpha = 0.05$ level while 30 minute backward returns are not significant at all.

Predicted Values

To calculate the in-sample and out-sample corrolations we first calculate the predicted values for training and testing data.

```
pred_lm_train <- predict(lm_fret)
pred_lm_test <- predict(lm_fret, fret_test)
```

In-Sample Correlation

Using the training data and training returns we calculate the in-sample correlation as follows

```
in_sample_corr_lm <- cor(pred_lm_train, fret_train$Asset_1_FRet_10)
```

The in-sample correlation is 0.0678748

Out-Sample Correlation

Using the testing data and testing returns we calculate the out-sample correlation as follows

```
out_sample_corr_lm <- cor(pred_lm_test, fret_test$Asset_1_FRet_10)
```

The out-sample correlation is 0.0406765

In-Sample Rolling Correlation Between Prediction and Actual

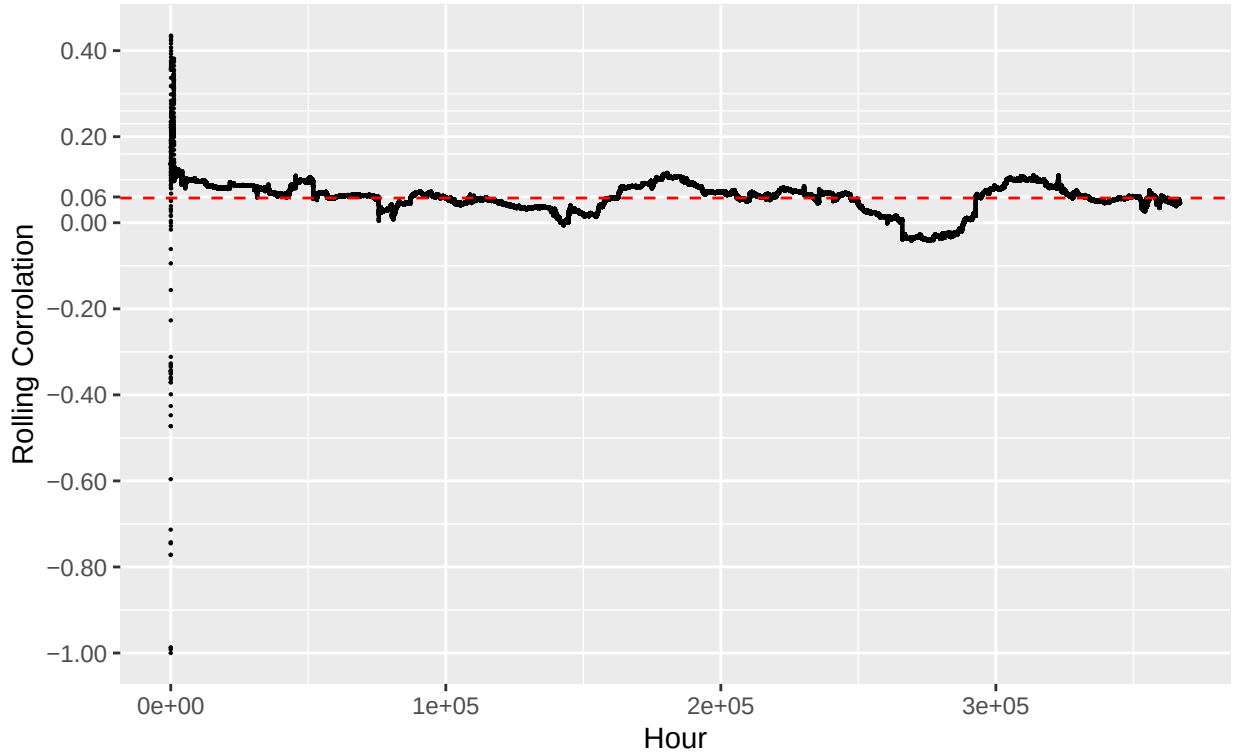
Now we will be calculating the 3 week backwards rolling correlation between the predicted forward returns and actual backwards returns using the training data

```
#Get the predicted values for test data
window_size <- 60*24*7*3 #hour,day,week,3

#Calculate 3 week rolling correlation between predicted and actual
lm_roll_corr_train <- roll_cor(pred_lm_train,
                               fret_train$Asset_1_FRet_10,
                               width = window_size,min_obs = 1)
lm_train_roll_corr_mean <- mean(lm_roll_corr_train, na.rm = T)
#Create Plot
ggplot() + geom_point(aes(x = index(lm_roll_corr_train),y = lm_roll_corr_train), size = 0.1) +
  geom_hline(yintercept = lm_train_roll_corr_mean , color = "red", linetype = "dashed") +
  scale_y_continuous(breaks = round(c(seq(-1,0.6, by = 0.2), lm_train_roll_corr_mean),2)) +
  labs(x = "Hour", y= "Rolling Corrolation",
       title = "Figure 3.1: In-Sample Rolling Correlation Between Predicted and Actual\n10 Hour Forward
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

Figure 3.1: In−Sample Rolling Correlation Between Predicted and Actual 10 Hour Forward Returns



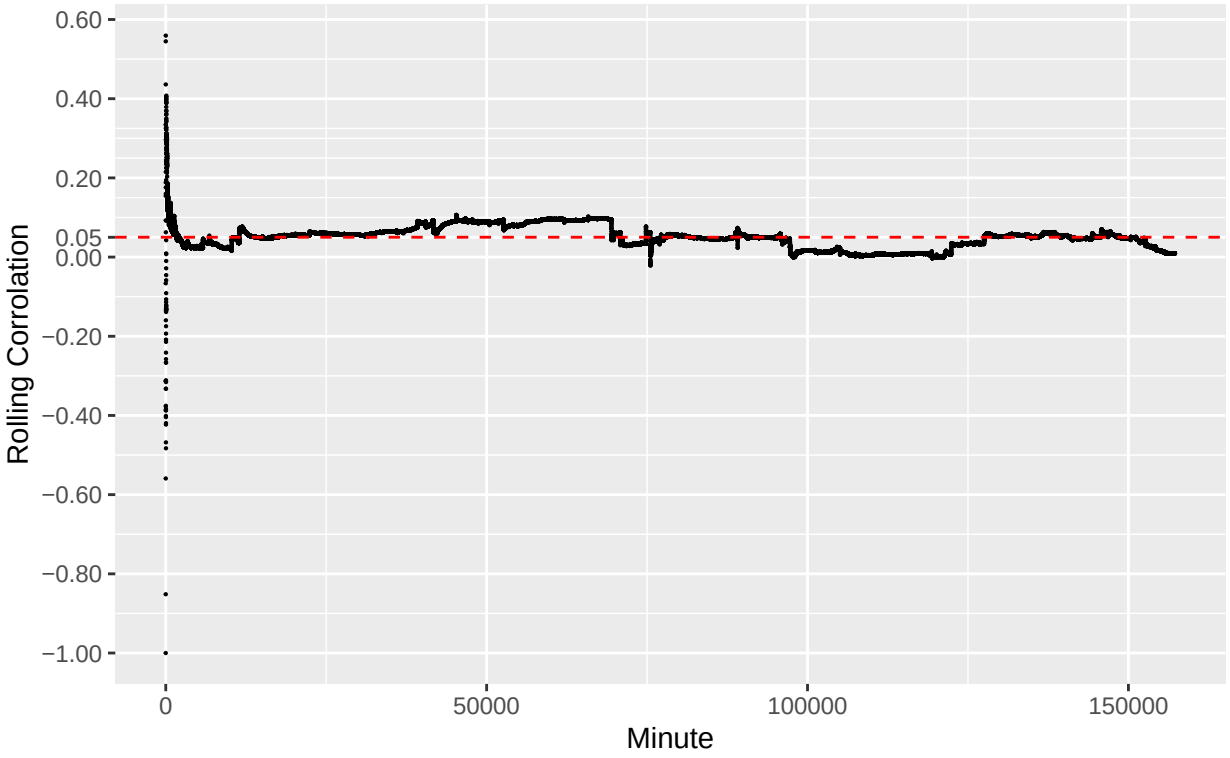
Out-Sample Rolling Corroleation Between Prediction and Actual

Now we will be calculating the 3 week backwards rolling correlation between the predicted forward returns and actual backwards returns using the testing data

```
window_size <- 60*24*7*3 #hour,day,week,3
#Calculate 3 week rolling corrolation between predicted and actual
lm_roll_corr_test <- roll_cor(pred_lm_test,
                              fret_test$Asset_1_FRet_10,
                              width = window_size,min_obs = 1)
lm_test_roll_corr_mean <- mean(lm_roll_corr_test, na.rm = T)
#Create Plot
ggplot() + geom_point(aes(x = index(lm_roll_corr_test),y = lm_roll_corr_test), size = 0.1) +
  geom_hline(yintercept = lm_test_roll_corr_mean , color = "red", linetype = "dashed") +
  scale_y_continuous(breaks = round(c(seq(-1,0.6, by = 0.2), lm_test_roll_corr_mean),2)) +
  labs(x = "Minute", y= "Rolling Corrolation",
       title = "Figure 3.2: Out-Sample Rolling Correlation Between Predicted and Actual\n10Hour Forward
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

Figure 3.2: Out−Sample Rolling Correlation Between Predicted and Actua
10Hour Forward Returns



In and Out Sample Rolling Correlation in Same Plot

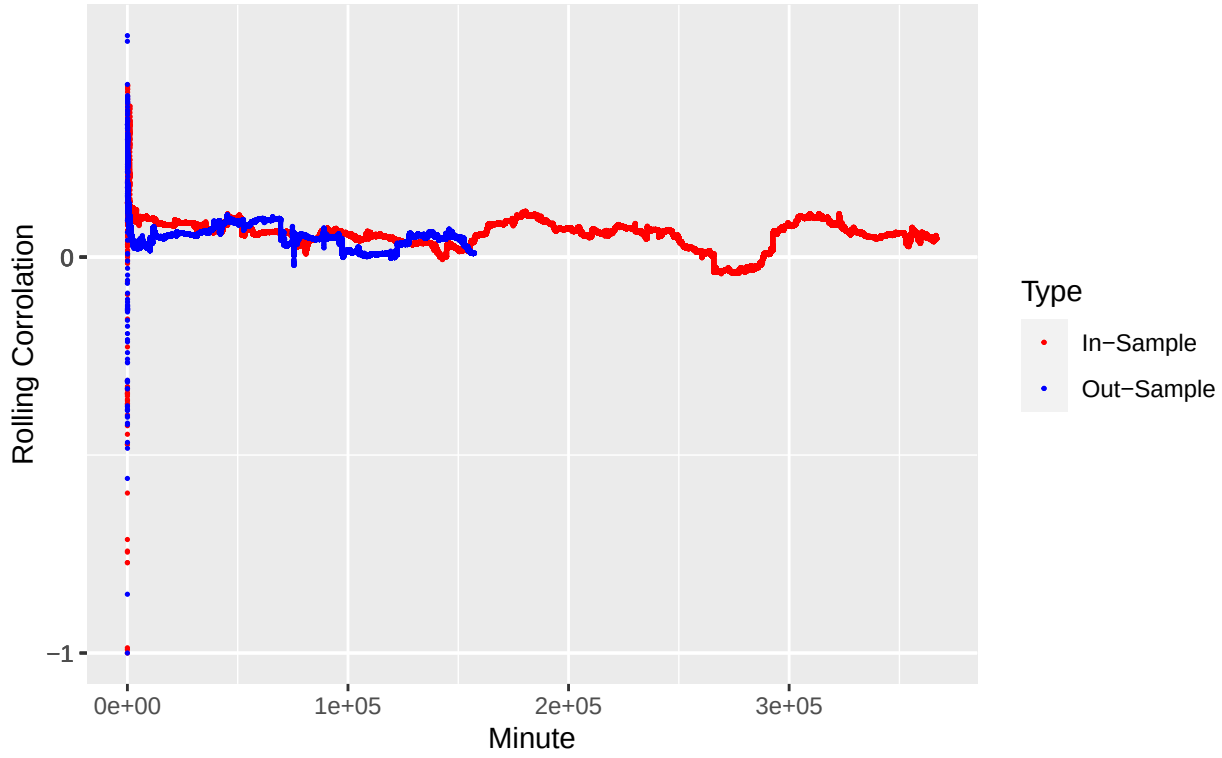
Finally we will be plotting both In-Sample and Out-Sample Correlation in the same plot

```
ggplot() + geom_point(aes(x = index(lm_roll_corr_train),y = lm_roll_corr_train,
                          color = "In-Sample"), size = 0.25) +
  geom_point(aes(x = index(lm_roll_corr_test),y = lm_roll_corr_test,
                 color = "Out-Sample"), size = 0.25) +
  scale_y_continuous(breaks = round(c(seq(-1,0.6, by = 0.2),2))) +
  scale_color_manual(values = c("In-Sample" = "Red", "Out-Sample" = "Blue")) +
  labs(x = "Minute", y= "Rolling Corrolation", colour = "Type",
       title = "Figure 3.3: In and Out Sample Rolling Correlation Between Predicted\nand Actual 10 Hour
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

Figure 3.3: In and Out Sample Rolling Correlation Between Predicted and Actual 10 Hour Forward Returns for



Comments on the Correlation Structure

Looking at the plots of 3 week backward In-Sample and Out-Sample rolling correlation between the predicted and actual 10 minute forward returns, we observe that the correlation structure is not stationary across the year

Part 4: KNN

In this section we will be using K-Nearest Neighbors to predict the 10 Minute Forward Returns of Asset 1. Unlike most cases we will not be standardizing our data since all of our predictors are in form of returns therefore have the same units and the scale of these returns gives us information about the volatility of the assets returns therefore we will not be standardizing the returns

Set up

For set-up we will be splitting the first 70% of our data for training and remaining 30% for testing as well as preallocating vecotrs for the MSE and defining the range of K values we will be fitting the model.

```
train_size <- floor(nrow(bret)*0.7)
train_ids <- c(1:train_size)
train_bret <- bret[train_ids,]
test_bret <- bret[-train_ids,]


k_range <- c(5, 25, 125, 625, 1000)
```

```
train_mse_knn = rep(NA, length(k_range)) #Pre-Allocate Vector
test_mse_knn <- rep(NA, length(k_range)) #Pre-Allocate Vector
```

Fitting KNN Models

We will be fitting KNN with $K \in \{5, 25, 125, 625, 1000\}$ number of nearest neighbors and saving these models at each step so we don't have to fit them every time. We will be commenting this section out once all the models are fitted.

```
#for(i in 1:length(k_range)){
#  knn_train <- knn.reg(train = train_bret,
#                       y = Asset_1_FRet10,
#                       test = train_bret,
#                       k = k_range[i])
#  knn_test<- knn.reg(train = train_bret,
#                       y = Asset_1_FRet10,
#                       test = test_bret,
#                       k = k_range[i])
#
#  train_file_name <- paste("KNN_Train_K_", k_range[i],".RData",sep = "")
#  save(knn_train,file = train_file_name)
#  test_file_name <- paste("KNN_Test_K_", k_range[i],".RData",sep = "")
#  save(knn_test,file = test_file_name)
#  }
```

Using Validation Approach to Calculate Optimal

Using the models we fitted and saved above, we will be using validation MSE to calculate the optimal number of nearest neighbors.

```
for (i in 1:length(k_range)) {
  train_file_name <- paste("KNN_Train_K_", k_range[i],".RData",sep = "")
  test_file_name <- paste("KNN_Test_K_", k_range[i],".RData",sep = "")

  load(train_file_name)
  load(test_file_name)

  train_mse_knn[i] <- mean((Asset_1_Fret$Asset_1_FRet_10 - knn_train$pred)^2, na.rm = T)
  test_mse_knn[i] <- mean((Asset_1_Fret$Asset_1_FRet_10 - knn_test$pred)^2, na.rm = T)
  rm(knn_test, knn_train) #remove the model to be loaded again
}
optimal_k_index <- which.min(test_mse_knn)
optimal_k <- k_range[optimal_k_index]
```

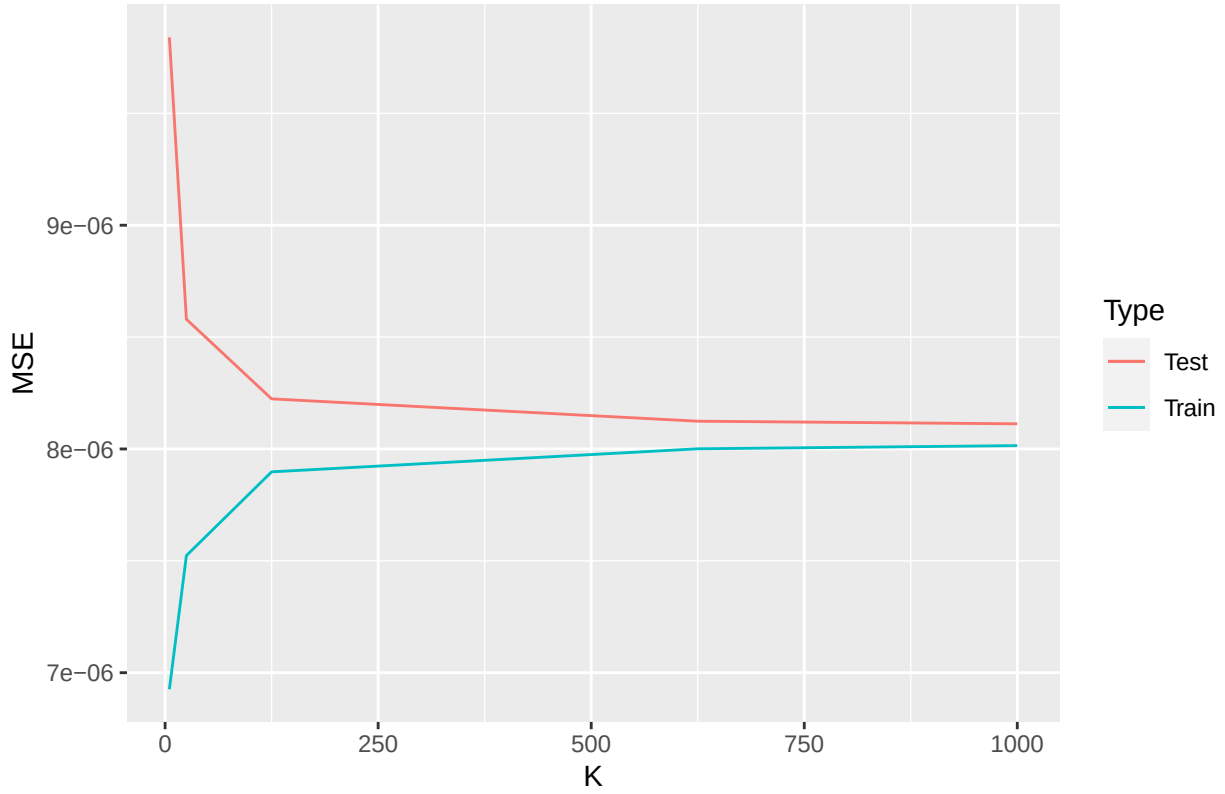
The lowest mean squared error occurs at K= 1000.

Trainig and Testing MSE as a Function of K

```
KNN_errors <- data.frame(k_range,train_mse_knn,test_mse_knn)

KNN_errors %>% ggplot() + geom_line(aes(k_range, test_mse_knn, color = "Test")) +
  geom_line(aes(k_range, train_mse_knn, color = "Train")) +
  labs(x = "K", y = "MSE", color = "Type",title = "Figure 4: MSE as a Function of K")
```

Figure 4: MSE as a Function of K



Predictions

Now that we know the optimal number of nearest neighbors, we can call that model of the optimal K we have saved and use them to calculate predictions for testing and training data.

```
load("KNN_Train_K_1000.RData")
KNN_1000_train_preds <- knn_train$pred

load("KNN_Test_K_1000.RData")
KNN_1000_test_preds <- knn_test$pred
```

In-Sample Correlation

We will be calculating the in-sample correlation using

```
KNN_train_cor <-cor(KNN_1000_train_preds,fret_train$Asset_1_FRet_10)
```

The in-sample correlation is 0.1181981

Out-Sample Correlation

We will be calculating the out-sample correlation as follows

```
KNN_test_cor <- cor(KNN_1000_test_preds,fret_test$Asset_1_FRet_10)
```

The out-sample correlation is 0.0432636

Part 5: Ridge and LASSO Regression

In this part, we will fit Ridge and LASSO regressions on a wide array of backward returns within the previous day to predict the 10 minute forward returns with the optimal penalty term λ decided by validation MSE.

Set-Up

Setting Up Data

First we will setting up data by calculating 1 day backward returns and splitting up the data into training and testing data. As last part the first 70% of our data goes into training data and the last 30% goes to testing.

```
# Caclulate Backward returns
brets_1day <- bret_calc(c(3, 10, 30, 60, 120, 180, 240, 360, 480, 600, 720, 960, 1200, 1440), data)

#Combine it with 10Min Forward Returns
brets_1day_Asset1_Fret <- cbind(brets_1day,Asset_1_Fret)

#Set training size
train_size <- floor(nrow(brets_1day) * 0.7)
#Split training and testing data
train_ids <- c(1:train_size)
brets_1day_Asset1_Fret_train <- brets_1day_Asset1_Fret[train_ids, ]
brets_1day_Asset1_Fret_test <- brets_1day_Asset1_Fret[-train_ids, ]

#Setup our data into a model matrix without intercept
X_train <- model.matrix(data = brets_1day_Asset1_Fret_train,
                        Asset_1_FRet_10 ~.)[,-1]
y_train <- brets_1day_Asset1_Fret_train$Asset_1_FRet_10
X_test <- model.matrix(data = brets_1day_Asset1_Fret_test,
                       Asset_1_FRet_10 ~ .)[,-1]
y_test <- brets_1day_Asset1_Fret_test$Asset_1_FRet_10
```

Defining a MSE Function for Ridge/Lasso

In order to do our cross-valudation more easily, we will be defining a function that calculates the MSE for Ridge and LASSO with penalty term λ as a parameter.

```
MSE_Ridge_Lasso <- function(model,lambda, data, outcome){
  #Predict the data based on model
  predicted <- predict(model,s = lambda, newx = data, type = "response")
  #(Predicted - outcome)squared
  Squared_Error <- (predicted - outcome)^2
  #return the mean of MSE
  return(mean(Squared_Error))
}
```

Ridge

In this section we will be fitting a Ridge Regression over a wide variety of penalty terms, finding the optimal penatly term by validation approach

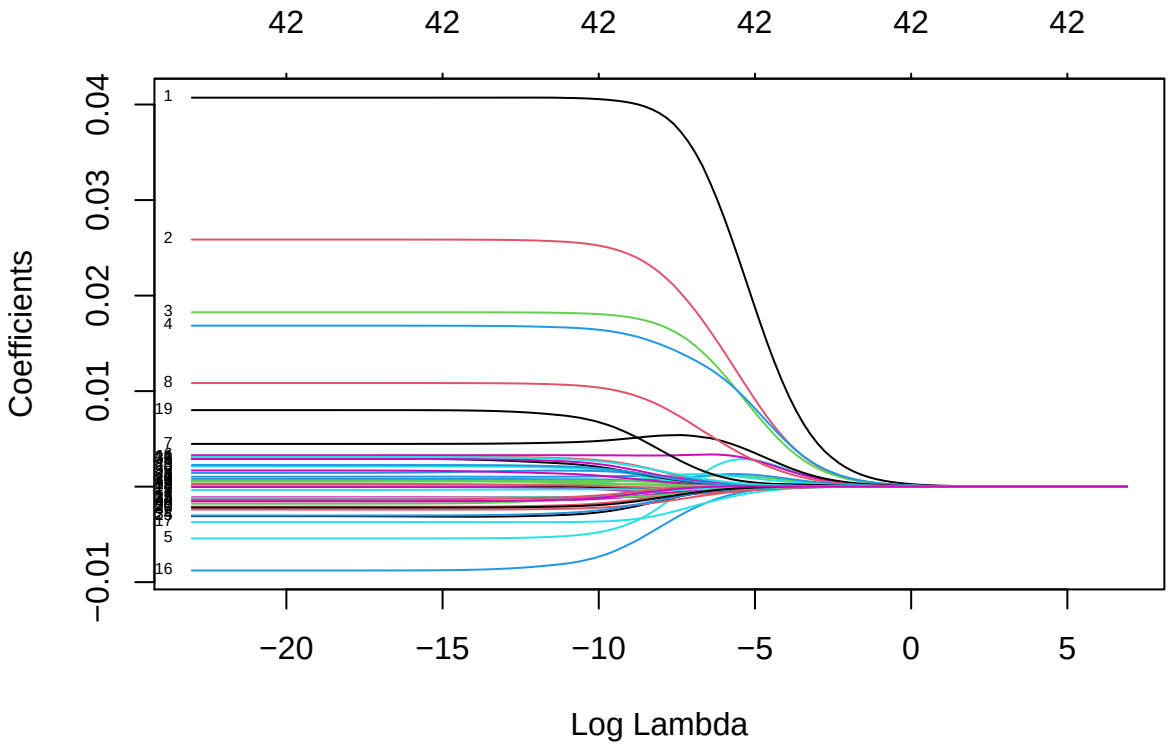
Fitting Ridge Regression

Now that we have done the pre-processing, we will be fitting a ridge model over a sequence of lambda ranging between 10^{-10} and 10^3. Then, we use calculate the optimal λ using the validation set.

```
grid <- 10^seq(-10, 3, by = 0.1)
ridge <- glmnet(X_train, y_train, alpha = 0, lambda = grid)
```

Variable Trace plot for Ridge can be seen below

```
plot(ridge, xvar = "lambda", label = TRUE)
```



Finding Optimal Lambda for Ridge

After fitting Ridge Model, we will calculate the optimal penalty term λ for our model using validation MSE. To do so we will calculate test MSE for all λ in our grid and select the one that produces the smallest testing MSE.

```
# Calculate MSE for all lambda in our grid
test_error_lambda <- rep(NA,length(grid))
for (i in 1:length(grid)){
  test_error_lambda[i] <- MSE_Ridge_Lasso(ridge, grid[i], X_test, y_test)
}
#Find the index of the optimal lambda
optimal_lambda_id <- which.min(test_error_lambda)
#Use the index to find optimal lambda
optimal_lambda <- grid[optimal_lambda_id]
```

The penalty term λ that minimizes testing MSE is 0.0039811

In-Sample Corrolation

For the optimal penalty term λ, we calculate the predicted values and use them to calculate the in-sample correlation

```
#Get Predicted Values
ridge_train_pred_frets <- predict(object = ridge, newx = X_train,
                                  s = optimal_lambda,type = "response")
#Get In-Sample Corrolation
In_Sample_Cor_Ridge <- cor(ridge_train_pred_frets,y_train)
```

We calculate In-Sample Corrolation to be 0.074

Out-Sample Corrolation

For the optimal penalty term λ, we calculate the predicted values and use them to calculate the out-sample correlation

```
#Get Predicted Values
ridge_test_pred_frets <- predict(object = ridge, newx = X_test,
                                  s = optimal_lambda,type = "response")

#Get Out of Sample Corrolation
Out_Sample_Cor_Ridge <- cor(ridge_test_pred_frets,y_test)
```

We Calculate Out-Sample Correlation is 0.038

LASSO

In this section we will be fitting a Ridge Regression over a wide variety of penalty terms, finding the optimal penatly term by validation approach

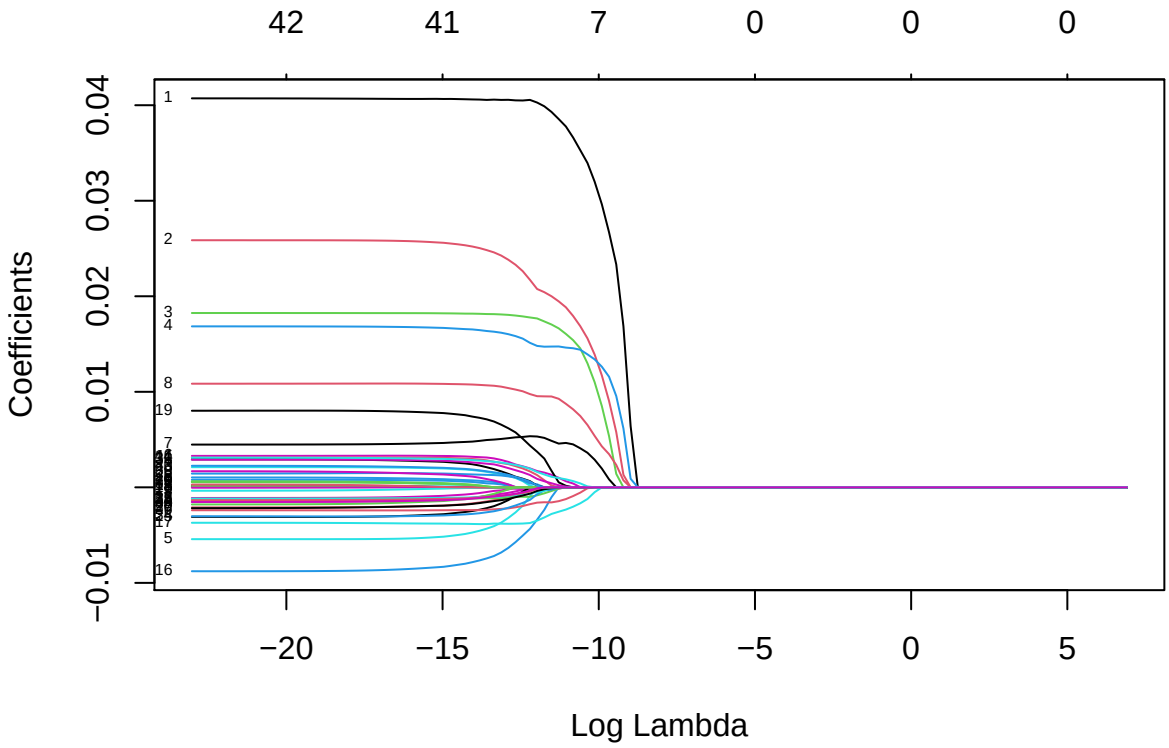
Fitting LASSO Regression

We will be fitting a LASSO model over a sequence of lambda ranging between 10^{-10} and 10^3. Then, we use calculate the optimal λ using the validation set

```
#Set a grid of values for Lasso
grid <- 10^seq(-10, 3, by = 0.1)
lasso <- glmnet(X_train, y_train, alpha = 1, lambda = grid)
```

Variable Trace plot for Lasso can be seen below

```
plot(lasso, xvar = "lambda", label = TRUE)
```



Finding Optimal Lambda for LASSO

After fitting LASSO Model, we will calculate the optimal penalty term λ for our model using validation MSE. To do so we will calculate test MSE for all λ in our grid and select the one that produces the smallest testing MSE.

```
#Cross Validate Using Test data
test_error_lambda <- rep(NA,length(grid))
for (i in 1:length(grid)){
  test_error_lambda[i] <- MSE_Ridge_Lasso(lasso, grid[i], X_test, y_test)
}
optimal_lambda_id <- which.min(test_error_lambda)
optimal_lambda <- grid[optimal_lambda_id]
```

The penalty term λ that minimizes testing MSE is 3.9810717×10^{-5}

Predictors Chosen by LASSO

The predictors chosen by the LASSO model can be seen below

```
predict(lasso, optimal_lambda,newx = X_train,type = "coefficient")
```

```
## 43 x 1 sparse Matrix of class "dgCMatrix"
##                                s1
## (Intercept)       -1.091053e-05
## Asset_1_BRet_3     3.199301e-02
## Asset_2_BRet_3     1.389708e-02
## Asset_3_BRet_3     1.100086e-02
```

```
## Asset_1_BRet_10    1.339646e-02
## Asset_2_BRet_10    .
## Asset_3_BRet_10    .
## Asset_1_BRet_30    2.891877e-03
## Asset_2_BRet_30    5.381886e-03
## Asset_3_BRet_30    .
## Asset_1_BRet_60    .
## Asset_2_BRet_60    .
## Asset_3_BRet_60    .
## Asset_1_BRet_120   .
## Asset_2_BRet_120   .
## Asset_3_BRet_120   .
## Asset_1_BRet_180   .
## Asset_2_BRet_180  -5.063523e-04
## Asset_3_BRet_180   .
## Asset_1_BRet_240   .
## Asset_2_BRet_240   .
## Asset_3_BRet_240   .
## Asset_1_BRet_360   .
## Asset_2_BRet_360   .
## Asset_3_BRet_360   .
## Asset_1_BRet_480   .
## Asset_2_BRet_480   .
## Asset_3_BRet_480   .
## Asset_1_BRet_600   .
## Asset_2_BRet_600   .
## Asset_3_BRet_600   .
## Asset_1_BRet_720   .
## Asset_2_BRet_720   .
## Asset_3_BRet_720   .
## Asset_1_BRet_960   .
## Asset_2_BRet_960   .
## Asset_3_BRet_960   .
## Asset_1_BRet_1200  .
## Asset_2_BRet_1200  .
## Asset_3_BRet_1200  .
## Asset_1_BRet_1440  .
## Asset_2_BRet_1440  .
## Asset_3_BRet_1440  .
```

In-Sample Corrolation

For the optimal penalty term λ, we calculate the predicted values and use them to calculate the in-sample correlation

```
#Get Predicted Values
lasso_train_pred_frets <- predict(object = lasso, newx = X_train,
                                  s = optimal_lambda,type = "response")
#Get In-Sample Corrolation
In_Sample_Cor_lasso <- cor(lasso_train_pred_frets,y_train)
```

We calculate In-Sample Corrolation to be 0.0687

Out-Sample Corrolation

For the optimal penalty term λ, we calculate the predicted values and use them to calculate the out-sample correlation

```
#Get Predicted Values
lasso_test_pred_frets <- predict(object = lasso, newx = X_test,
                                 s = optimal_lambda,type = "response")

#Get Out of Sample Corrolation
Out_Sample_Cor_lasso <- cor(lasso_test_pred_frets,y_test)
```

We calculate Out-Sample Correlation is 0.0393

Part 6: Principal Component Regression

In the final section we will be fitting a Principal Component Regression predicting the the 10 Minute Forward returns of Asset 1 using a array of backward returns within the previous day.

Set-Up

We will be using the same data we prepared for Ridge and LASSO in the preceding section

Principal Component Regression MSE Function

In order to make our code cleaner, we will be defining a function that calculates the Mean Squared Error of a given PCR model for a number of principal components

```
MSE_PCR <- function(model, n_pc, data, outcome){
  predicted <- predict(model, ncomp= n_pc, newdata = data)
  #(Predicted - outcome)squared
  Squared_Error <- (predicted - outcome)^2
  #return the mean of MSE
  return(mean(Squared_Error))
}
```

Fitting PCR Regression

We will be fitting a PCR regression without scaling the data. The decision to not scale the data is same as KNN, all of our data is in form of returns (same scale) and the thus the variability of our data gives us information about the assests volatility.

```
pcr_model <- pcr(Asset_1_FRet_10 ~ ., data = brets_1day_Asset1_Fret_train, scale = F, validation = "non
```

Pick Optimal Number of Principal Components Using Validation

We will be selecting the optimal number of principal components for the model using validation appraoch. Using the *MSE_PCR* function we defined above we can calculate the MSE for each number of principal compnents.

```
#Number of predictors - 1 for response
n_principal_comp <- ncol(brets_1day_Asset1_Fret_train) -1
#Pre Allocate Vector
test_error_ncomp <- rep(NA, n_principal_comp)
for (i in 1:n_principal_comp){
  test_error_ncomp[i] <- MSE_PCR(pcr_model,n_pc = i,
```

```
                                  brets_1day_Asset1_Fret_test, y_test)
}
optimal_ncomp <- which.min(test_error_ncomp)
```

In-Sample Correlation

Using the optimal number of principal components calculated above, we calculate the in-sample correlation as follows

```
pcr_train_preds <- predict(pcr_model,newdata = X_train, ncomp = optimal_ncomp, )
pcr_train_cor <- cor(pcr_train_preds,y_train)
```

The in-sample correlation is 0.082619

Out-Sample Correlation

Using the optimal number of principal components calculated above, we calculate the out-sample correlation as follows

```
pcr_test_preds <- predict(pcr_model,newdata = X_test, ncomp = optimal_ncomp, )
pcr_test_cor <- cor(pcr_test_preds,y_test)
```

The out-sample correlation is 0.0390737