In [ ]:
```
library(tidyverse)
```

Final Report Stats415

2.1 Backwards Returns ¶

In [ ]:
```
dataset <- read.csv("final_project_data.csv")
```

Here, we calculate returns by comparing the maximum value between the difference of row number and asset number with 1. We use this value to append returns to an empty vector. Finally, combine all vectors together, for each asset, into a dataframe.

In [ ]:
```
calc_returns1 <- function(which_asset, number) {

  x <- c()

  for(i in 1:nrow(dataset)) {

  max_val <- max((i-number),1)

  x[i] <- (dataset[,which_asset][i]-dataset[,which_asset][max_val])/(dataset[,
which_asset][max_val])
}

  return(x)
}
```

We use the calc_returns function to calculate returns for each asset.

In [ ]:
```
Asset_1_BRet_3 <- calc_returns1(2, 3)
Asset_1_BRet_10 <- calc_returns1(2, 10)
Asset_1_BRet_30 <- calc_returns1(2, 30)
```

In [ ]:
```
Asset_2_BRet_3 <- calc_returns1(3, 3)
Asset_2_BRet_10 <- calc_returns1(3, 10)
Asset_2_BRet_30 <- calc_returns1(3, 30)
```

In [ ]:
```
Asset_3_BRet_3 <- calc_returns1(4, 3)
Asset_3_BRet_10 <- calc_returns1(4, 10)
Asset_3_BRet_30 <- calc_returns1(4, 30)
```

Finally, we combine all vectors into a dataframe.

In [ ]:
```
xx <- cbind(Asset_1_BRet_3, Asset_1_BRet_10, Asset_1_BRet_30, Asset_2_BRet_3,
Asset_2_BRet_10, Asset_2_BRet_30, Asset_3_BRet_3, Asset_3_BRet_10, Asset_3_BRe
t_30)

#write.csv(xx,"D:\\bret.csv", row.names = FALSE)
```

2.2 Backwards Rolling Correlations

In [ ]:
```
data <- read.csv("bret-1.csv")
```

Using the w-min backwards correlation formula provided in the specifications, we get the maximum value between the difference of the row and three weeks and the integer 1. If 1 is the maximum value, then we calculate the correlation by indexing from 1 to i. If 1 is not the maximum value, then start from different index.

In [ ]:
```
three_weeks <- 21 * 24 * 60
three_weeks

correlations <- c()
for(i in 1:length(data$Asset_1_BRet_3)) {

  indexes <- max((i-three_weeks), 1)

  if(indexes == 1) {
    correlations <- c(correlations, round(cor(data$Asset_1_BRet_3[1:i], data$
Asset_2_BRet_3[1:i]),4))
  }

  else{

   correlations <- c(correlations, round(cor(data$Asset_1_BRet_3[indexes:i], d
ata$Asset_2_BRet_3[indexes:i]),4))
  }

}
```

In [ ]:
```
correlations1 <- c()
for(i in 1:length(data$Asset_1_BRet_3)) {

  indexes <- max((i-three_weeks), 1)

  if(indexes == 1) {
    correlations1 <- c(correlations1, round(cor(data$Asset_1_BRet_3[1:i],
data$Asset_3_BRet_3[1:i]),4))
  }

  else{

   correlations1 <- c(correlations1, round(cor(data$Asset_1_BRet_3[indexes:i],
data$Asset_3_BRet_3[indexes:i]),4))
  }

}
```

In [ ]:
```
correlations2 <- c()
for(i in 1:length(data$Asset_1_BRet_3)) {

  indexes <- max((i-three_weeks), 1)

  if(indexes == 1) {
    correlations2 <- c(correlations2, round(cor(data$Asset_2_BRet_3[1:i],
data$Asset_3_BRet_3[1:i]),4))
  }

  else{

   correlations2 <- c(correlations2, round(cor(data$Asset_2_BRet_3[indexes:i],
data$Asset_3_BRet_3[indexes:i]),4))
  }

}
```

Below are the backwards rolling correlations between assets 1-2,2-3,1-3, as each column.

In [ ]:
```
Rho_1_2 <- correlations
Rho_1_3 <- correlations1
Rho_2_3 <- correlations2
zero <- c(0,0,0)
corr <- data.frame(Rho_1_2, Rho_2_3, Rho_1_3)
corr[1,] <- zero
corr
```

2.3 Forward Return Linear Regression

```
In [ ]: tbl1 = read_csv("final_project_data.csv")
        tbl2 = xx
        tbl = cbind(tbl1, tbl2)
        tbl$min10 = pmin((tbl$...1 + 10), 524160)
        tbl$rf10 = (tbl$Asset_1[tbl$min10] - tbl$Asset_1)/tbl$Asset_1
        #creating data to use and making a forward returns column
```

Splitting the data into training and testing, then making our linear regression model.

```
In [ ]: cutoff = floor(nrow(tbl)*.7)
```

```
In [ ]: train = tbl[1:cutoff,]
        test = tbl[(cutoff+1):nrow(tbl),]
        #splitting data
```

```
In [ ]: mod1 = lm(rf10 ~ Asset_1_BRet_3+Asset_1_BRet_10+Asset_1_BRet_30+Asset_2_BRet_3
        +Asset_2_BRet_10+Asset_2_BRet_30+Asset_3_BRet_3+Asset_3_BRet_10+Asset_3_BRet_3
        0, train)
```

```
In [ ]: summary(mod1)
        #forward returns of asset 1 with backwards returns as features
```

2-3, 2-30, and 3-3 are significant in predicting the forward returns. Now we observe the in-sample and out-sample correlations.

```
In [ ]: c(cor(predict(mod1, train), train$rf10), cor(predict(mod1, test), test$rf10
        ))
```

Now, we find the three-week backwards rolling correlation of the response variable and prediction.

```
In [ ]: three_weeks = 60*24*7
```

```
In [ ]: preds = predict(mod1,tbl)
```

In [ ]:
```
correlations <- c()
for(i in 1:nrow(tbl)) {

  indexes <- max((i-three_weeks), 1)

  if(indexes == 1) {
    correlations <- c(correlations, round(cor(tbl$rf10[1:i], preds[1:i]),4))
  }

  else{

   correlations <- c(correlations, round(cor(tbl$rf10[indexes:i], preds[indexe
s:i]),4))
  }

}
```

In [ ]:
```
ggplot() + geom_point(aes(tbl$...1, correlations)) + labs(x = "Time", y = "BR
C", main = "Backwards Rolling Correlation over time")
```

We see that the correlaton structure is relatively stationary over the year.

2.4 KNN

In [ ]:
```
library(FNN)
```

We split training and test data (using first 70% as training data and the last 30% as testing data)

In [ ]:
```
tbl1 = read_csv("final_project_data.csv")
tbl2 = xx
tbl = cbind(tbl1, tbl2)
minimum <- min((tbl$...1 + 10), 524160)

tbl$min10 = minimum
tbl$rf10 = (tbl$Asset_1[tbl$min10] - tbl$Asset_1)/tbl$Asset_1

cutoff = floor(nrow(tbl)*.7)

training = tbl[1:cutoff,]

testing = tbl[(cutoff+1):nrow(tbl),]
```

We remove unecessary columns from the training and test data.

```
In [ ]: X_train = training %>% dplyr::select(-...1, -Asset_1, -Asset_2, -Asset_3, -min
        10, -rf10)

        X_test = testing %>% dplyr::select(-...1, -Asset_1, -Asset_2, -Asset_3, -min10
        , -rf10)
```

We train the knn regression model for the below range of k.

```
In [ ]: k_range = c(5,25,125,625,1000)
        trainMSE = c() #creating null vector
        testMSE = c()
        for(i in 1:length(k_range)){
          knnTrain <- knn.reg(train = X_train, y = training$rf10,
          test = X_train,k = k_range[i])
          trainMSE[i] <- mean((training$rf10 - knnTrain$pred)^2)

          knnTest <- knn.reg(train = X_train, y = training$rf10,
          test = X_test, k = k_range[i])
          testMSE[i] <- mean((testing$rf10 - knnTest$pred)^2)
        }
```

We notice that, based on validation MSE, the best K for this dataset is 25.

```
In [ ]: plot(trainMSE ~ I(k_range), type = "b", lwd = 2, col = "blue",
        main = "Training MSE for KNN", xlab = "K", ylab = "trainMSE")
        # Add the test MSE
        plot(testMSE ~ I(k_range), type = "b", lwd = 2, col = "red", xlab = "K")
```

We generate the prediction for the entire year.

```
In [ ]: knnTrain1 <- knn.reg(train = X_train, y = training$rf10,
          test = X_train,k = 25)

        knnTest1 <- knn.reg(train = X_train, y = training$rf10,
          test = X_test, k = 25)

        prediction <- c(knnTrain1$pred, knnTest1$pred)
```

The in-sample and out of sample correlation are displayed respectively.

```
In [ ]: cor(X_train, knnTrain1$pred)

        cor(X_test, knnTest1$pred)
```

2.5 Lasso and Ridge Regressions

```
In [ ]: library(glmnet)
        tbl1 = read_csv("final_project_data.csv")
        tbl2 = xx
        tbl = cbind(tbl1, tbl2)
```

Here, we are making columns for backwards returns of various times.

```
In [ ]: tbl$min10 = pmin((tbl$...1 + 10), nrow(tbl))
        tbl$rf10 = (tbl$Asset_1[tbl$min10] - tbl$Asset_1)/tbl$Asset_1
        tbl$max3 = pmax((tbl$...1 - 3), 1)
        tbl$rb3 = (tbl$Asset_1 - tbl$Asset_1[tbl$max3])/tbl$Asset_1[tbl$max3]
        tbl$max10 = pmax((tbl$...1 - 10), 1)
        tbl$rb10 = (tbl$Asset_1 - tbl$Asset_1[tbl$max10])/tbl$Asset_1[tbl$max10]
        tbl$max30 = pmax((tbl$...1 - 30), 1)
        tbl$rb30 = (tbl$Asset_1 - tbl$Asset_1[tbl$max30])/tbl$Asset_1[tbl$max30]
        tbl$max60 = pmax((tbl$...1 - 60), 1)
        tbl$rb60 = (tbl$Asset_1 - tbl$Asset_1[tbl$max60])/tbl$Asset_1[tbl$max60]
        tbl$max120 = pmax((tbl$...1 - 120), 1)
        tbl$rb120 = (tbl$Asset_1 - tbl$Asset_1[tbl$max120])/tbl$Asset_1[tbl$max120]
        tbl$max180 = pmax((tbl$...1 - 180), 1)
        tbl$rb180 = (tbl$Asset_1 - tbl$Asset_1[tbl$max180])/tbl$Asset_1[tbl$max180]
        tbl$max240 = pmax((tbl$...1 - 240), 1)
        tbl$rb240 = (tbl$Asset_1 - tbl$Asset_1[tbl$max240])/tbl$Asset_1[tbl$max240]
        tbl$max360 = pmax((tbl$...1 - 360), 1)
        tbl$rb360 = (tbl$Asset_1 - tbl$Asset_1[tbl$max360])/tbl$Asset_1[tbl$max360]
        tbl$max480 = pmax((tbl$...1 - 480), 1)
        tbl$rb480 = (tbl$Asset_1 - tbl$Asset_1[tbl$max480])/tbl$Asset_1[tbl$max480]
        tbl$max600 = pmax((tbl$...1 - 600), 1)
        tbl$rb600 = (tbl$Asset_1 - tbl$Asset_1[tbl$max600])/tbl$Asset_1[tbl$max600]
        tbl$max720 = pmax((tbl$...1 - 720), 1)
        tbl$rb720 = (tbl$Asset_1 - tbl$Asset_1[tbl$max720])/tbl$Asset_1[tbl$max720]
        tbl$max960 = pmax((tbl$...1 - 960), 1)
        tbl$rb960 = (tbl$Asset_1 - tbl$Asset_1[tbl$max960])/tbl$Asset_1[tbl$max960]
        tbl$max1200 = pmax((tbl$...1 - 1200), 1)
        tbl$rb1200 = (tbl$Asset_1 - tbl$Asset_1[tbl$max1200])/tbl$Asset_1[tbl$max120
        0]
        tbl$max1440 = pmax((tbl$...1 - 1440), 1)
        tbl$rb1440 = (tbl$Asset_1 - tbl$Asset_1[tbl$max1440])/tbl$Asset_1[tbl$max144
        0]
```

We split the data and run a ridge regression model, plotting lambda vs coefficients.

In [ ]:
```
cutoff = floor(nrow(tbl)*.7)
train_id = 2:cutoff
train = tbl[2:cutoff,]
test = tbl[(cutoff+1):nrow(tbl),]
#splitting data
Y = tbl$rf10
X = model.matrix(rf10 ~ rb3 + rb10 + rb30 + rb60 + rb120 + rb240 + rb360 + rb4
80 + rb600 + rb720 + rb960 + rb1200 + rb1440, tbl)[,-1]
grid = 10^seq(5, -8, length=100)
ridge.mod = glmnet(X[train_id,], Y[train_id], alpha=0, lambda=grid)
plot(ridge.mod, xvar = "lambda", label = TRUE)
```

Now, we minimize MSE to find the optimal lambda.

In [ ]:
```
ridge.pred_test = predict(ridge.mod, newx=X[-train_id,])
which.min(apply(ridge.pred_test, 2, function (x) mean((x - Y[-train_id])^2)))
# findng which lambda has the lowest validation mse by index
```

In [ ]:
```
(bestlam = grid[55])
# finding optimal lambda
```

We find the in and out of sample correlations.

In [ ]:
```
ridge.pred_train = predict(ridge.mod,s = bestlam, newx=X[train_id,])
ridge.pred_test = predict(ridge.mod,s = bestlam, newx=X[-train_id,])
c(cor(Y[train_id], ridge.pred_train), cor(Y[-train_id], ridge.pred_test))
#in sample and out sample correlation
```

Finally, we repeat the exact same process for a lasso model.

In [ ]:
```
lasso.mod = glmnet(X[train_id,], Y[train_id], alpha=1, lambda=grid)
plot(lasso.mod, xvar = "lambda", label = TRUE)
```

In [ ]:
```
lasso.pred_test = predict(lasso.mod, newx=X[-train_id,])
which.min(apply(lasso.pred_test, 2, function (x) mean((x - Y[-train_id])^2)))
# index of lambda w lowest mse
```

In [ ]:
```
(bestlam = grid[71])
#lambda with lowest mse
```

In [ ]:
```
lasso.pred_train = predict(lasso.mod,s = bestlam, newx=X[train_id,])
lasso.pred_test = predict(lasso.mod,s = bestlam, newx=X[-train_id,])
c(cor(Y[train_id], lasso.pred_train), cor(Y[-train_id], lasso.pred_test))
# in sample and out sample correlation
```

2.6 Principle component regression (PCR)

The pls package is loaded in order to do do principal component regression (PCR)

In [ ]:
```
library(pls)
tbl1 = read_csv("final_project_data.csv")
tbl2 = xx
tbl = cbind(tbl1, tbl2)
```

In [ ]:
```
tbl$...1 <- tbl$X1
```

In [ ]:
```
tbl <- tbl %>% select(-X1)
```

In [ ]:
```
tbl$min10 = pmin((tbl$...1 + 10), nrow(tbl))
tbl$rf10 = (tbl$Asset_1[tbl$min10] - tbl$Asset_1)/tbl$Asset_1
tbl$max3 = pmax((tbl$...1 - 3), nrow(tbl))
tbl$rb3 = (tbl$Asset_1 - tbl$Asset_1[tbl$max3])/tbl$Asset_1[tbl$max3]
tbl$max10 = pmax((tbl$...1 - 10), 1)
tbl$rb10 = (tbl$Asset_1 - tbl$Asset_1[tbl$max10])/tbl$Asset_1[tbl$max10]
tbl$max30 = pmax((tbl$...1 - 30), 1)
tbl$rb30 = (tbl$Asset_1 - tbl$Asset_1[tbl$max30])/tbl$Asset_1[tbl$max30]
tbl$max60 = pmax((tbl$...1 - 60), 1)
tbl$rb60 = (tbl$Asset_1 - tbl$Asset_1[tbl$max60])/tbl$Asset_1[tbl$max60]
tbl$max120 = pmax((tbl$...1 - 120), 1)
tbl$rb120 = (tbl$Asset_1 - tbl$Asset_1[tbl$max120])/tbl$Asset_1[tbl$max120]
tbl$max180 = pmax((tbl$...1 - 180), 1)
tbl$rb180 = (tbl$Asset_1 - tbl$Asset_1[tbl$max180])/tbl$Asset_1[tbl$max180]
tbl$max240 = pmax((tbl$...1 - 240), 1)
tbl$rb240 = (tbl$Asset_1 - tbl$Asset_1[tbl$max240])/tbl$Asset_1[tbl$max240]
tbl$max360 = pmax((tbl$...1 - 360), 1)
tbl$rb360 = (tbl$Asset_1 - tbl$Asset_1[tbl$max360])/tbl$Asset_1[tbl$max360]
tbl$max480 = pmax((tbl$...1 - 480), 1)
tbl$rb480 = (tbl$Asset_1 - tbl$Asset_1[tbl$max480])/tbl$Asset_1[tbl$max480]
tbl$max600 = pmax((tbl$...1 - 600), 1)
tbl$rb600 = (tbl$Asset_1 - tbl$Asset_1[tbl$max600])/tbl$Asset_1[tbl$max600]
tbl$max720 = pmax((tbl$...1 - 720), 1)
tbl$rb720 = (tbl$Asset_1 - tbl$Asset_1[tbl$max720])/tbl$Asset_1[tbl$max720]
tbl$max960 = pmax((tbl$...1 - 960), 1)
tbl$rb960 = (tbl$Asset_1 - tbl$Asset_1[tbl$max960])/tbl$Asset_1[tbl$max960]
tbl$max1200 = pmax((tbl$...1 - 1200), 1)
tbl$rb1200 = (tbl$Asset_1 - tbl$Asset_1[tbl$max1200])/tbl$Asset_1[tbl$max120
0]
tbl$max1440 = pmax((tbl$...1 - 1440), 1)
tbl$rb1440 = (tbl$Asset_1 - tbl$Asset_1[tbl$max1440])/tbl$Asset_1[tbl$max144
0]
#creating all the backwards returns and forward returns
```

In [ ]:
```
cutoff <- floor(nrow(tbl)*.7)
train_id <- 2:cutoff
train <- tbl[2:cutoff,]
test <- tbl[(cutoff+1):nrow(tbl),]
#splitting data
```

set.seed is used to generate a sequence of random numbers. The pcr function is used to get the PCR model using rb3 + rb10 + rb30 + rb60 + rb120 + rb240 + rb360 + rb480 + rb600 + rb720 + rb960 + rb1200 + rb1440 as predictors for the response variable rf10. The scale argument in the pcr function is to indicate that the data should be scaled. The validation argument is set to "CV" so that a 10-fold cross-validation error is performed for each value of k

```
In [ ]: set.seed(1)
        hitPCR_model <- pcr(rf10 ~ rb3 + rb10 + rb30 + rb60 + rb120 + rb240 + rb360 +
        rb480 + rb600 + rb720 + rb960 + rb1200 + rb1440,
                      data = train, scale = TRUE, validation = "CV")
```

From the summary of the model, the section under "VALIDATION" provides the 10-fold CV root MSE for each k. From the summary, the smallest cross-validation error occurs when k = 11 so when 11 components are used.

```
In [ ]: summary(hitPCR_model)
```

The validationplot, it is also seen that the smallest cross-validation error is when k = 11 so when 11 components are used. This suggests that a model that uses a smaller number of components, 11 vs 13, is sufficient.

```
In [ ]: validationplot(hitPCR_model, val.type = "MSEP", legendpos = "topright")
```

To find the test and training error, MSE is calculated using prediction first and then the mean. In prediction the argument ncomp = 11 comes from the smaller model where k = 11 minimizes the CV-MSE.

```
In [ ]: hitPCR.pred <- predict(hitPCR_model, test, ncomp = 11)
        PCRTestMSE <- mean((hitPCR.pred - test[, "rf10"])^2)
        PCRTestMSE
        hitPCR_pred_train <- predict(hitPCR_model, train, ncomp = 11)
        PCRTrainMSE <- mean((hitPCR_pred_train - train[, "rf10"])^2)
        PCRTrainMSE
```

The cor function is used to report the correlation between the preiction and true response from the in-sample and out-of-sample.

```
In [ ]: c(cor(train$rf10, hitPCR_pred_train), cor(test$rf10, hitPCR.pred))
```

```
In [ ]:
```