Final Project

Shanie Talor, Arif Abd Aziz, Jon Girolamo

2023-06-02

Introduction

We compiled a data set, using features from a variety of sources, in order to build an algorithm that can predict the movement of gold prices on a month to month basis. Our data matrix contains features that theoretically have an effect on gold prices. These features include the federal funds rate, inflation month over month, stock market indices, consumer sentiment, consumer expectations, industry production indices related to gold, financial conditions, total bank reserves, and oil prices. Though this list of features is not comprehensive of all possible factors that affect the prices of gold, we believe they capture certain dynamics. In order to normalize some of these features since some grow continuously (such as bank reserves), we converted them to percent changes. Along with our features, we added lags for percent returns on gold from previous months as well as moving averages. This can capture time dynamics.

As for our main response variable, we are looking to predict the percent change in gold prices. We were able to make this into a quasi-forecasting problem by re-aligning percent changes in gold from our data. For example, the change in gold prices from January to February (Jan 1st to Jan 31st/Feb 1st) is usually assigned to the month of February. We assigned the percentage change to our January observation and not the month of February. Thus, we were able to forecast percent changes in gold prices. Along with predicting changes in gold, we decided to convert out response variable into a binary classifier for direction. We set the threshold of binary returns at 0.1% (any percent return above 0.1% will be assigned to a class of 1, and everything else to a class of 0).

By setting these two response variables, we can perform both regression and classification tasks in order to predict the percent returns on gold as well as their directions. We are conducting this project as if we are trying to sell an investment strategy (maximize returns and probabiity of success). Thus improving predictability is more important to us than interpretability. Most of the models we'll be running are highly non-liner and complex. To optimize these models, we'll be running grid searches of hyper parameters and comparing their performance through cross validation.

Load Data

```r
# Read csv
gold <- read.csv("gold.csv", sep = ",", header = TRUE)

# Read CSV with one extra parameter
sentiment <- read.csv("sentiment.csv", sep = ",", header = TRUE)

# Clean Data for Rows Missing
sentiment <- sentiment[-c(1:9, 543, 544), ]
```

```r
# Add sentiment parameter to main data set
gold$sentiment <- sentiment$sentiment

head(gold)
```

```
##         Date Average.Price PercChangeForc PercChangeLag1 PercChangeLag2
## 1 31/12/1978         207.8       9.384023             NA             NA
## 2 31/01/1979         227.3       8.095029       9.384023             NA
## 3 28/02/1979         245.7      -1.465201       8.095029       9.384023
## 4 30/03/1979         242.1      -1.197852      -1.465201       8.095029
## 5 30/04/1979         239.2       7.692308      -1.197852      -1.465201
## 6 31/05/1979         257.6       8.346273       7.692308      -1.197852
##   PercChangeLag3 PercChangeLag4 PercChangeLag5      X2MA     X3MA Inf.Rate.MoM
## 1             NA             NA             NA        NA       NA      0.93394
## 2             NA             NA             NA        NA       NA      1.07389
## 3             NA             NA             NA  8.739526       NA      0.39425
## 4       9.384023             NA             NA  3.314914 5.337950      1.02503
## 5       8.095029       9.384023             NA -1.331527 1.810658      0.88286
## 6      -1.465201       8.095029       9.384023  3.247228 1.676418      0.47022
##    Inf.L1  Inf.L2  Inf.L3  Inf.L4 Res.Change.Exc.Gold UM.Infl.Exp UM.Con.Sent
## 1      NA      NA      NA      NA           -13.10790         7.3        66.1
## 2 0.93394      NA      NA      NA            10.33653         7.8        72.1
## 3 1.07389 0.93394      NA      NA            27.18704         9.3        73.9
## 4 0.39425 1.07389 0.93394      NA            -1.45682         8.8        68.4
## 5 1.02503 0.39425 1.07389 0.93394            16.36438         9.7        66.0
## 6 0.88286 1.02503 0.39425 1.07389            -1.90551         9.8        68.1
##   Indus.Prod.Ind Nasdaq.Change.MoM   NFCI FedFundsRate FedFundsRateL1   Oil
## 1        18.8587           2.73927 1.8540        10.03             NA  9.47
## 2        18.1240           5.51792 1.3525        10.07          10.03  9.46
## 3        17.5308           0.67150 0.8250        10.06          10.07  9.69
## 4        17.2653           2.83544 0.4280        10.09          10.06  9.83
## 5        16.8854           4.18881 0.3200        10.01          10.09 10.33
## 6        17.0833          -1.57133 0.4350        10.24          10.01 10.71
##   PercChange.oil PPIJewelry PercChange.PPI sentiment
## 1             NA       57.3             NA 0.2487900
## 2     -0.1055966       58.7      2.4432810 0.2205821
## 3      2.4312896       62.0      5.6218058 0.2773207
## 4      1.4447884       62.6      0.9677419 0.2395667
## 5      5.0864700       62.8      0.3194888 0.2467593
## 6      3.6786060       64.9      3.3439490 0.2570759
```

```r
# Omit all rows that have NA Values: 6 Rows: 05/31/1979 to 03/31/2023
gold <- na.omit(gold)

# Add Binary Classifier Variable
gold$Binary.PercChange <- ifelse(gold$PercChangeForc > 0.1, 1, 0)

# Check if Features Are Numeric (Returns False, So Non-Numeric Variables)
all(sapply(gold, is.numeric))
```

```
## [1] FALSE
```

```r
# Final Check of Head
head(gold)
```

```
##          Date Average.Price PercChangeForc PercChangeLag1 PercChangeLag2
## 6  31/05/1979         257.6     8.34627329       7.692308      -1.197852
## 7  29/06/1979         279.1     5.58939448       8.346273       7.692308
## 8  31/07/1979         294.7     2.06990160       5.589394       8.346273
## 9  31/08/1979         300.8    18.05186170       2.069902       5.589394
## 10 28/09/1979         355.1    10.30695579      18.051862       2.069902
## 11 31/10/1979         391.7     0.07658923      10.306956      18.051862
##    PercChangeLag3 PercChangeLag4 PercChangeLag5      X2MA      X3MA
## 6       -1.465201       8.095029       9.384023  3.247228  1.676418
## 7       -1.197852      -1.465201       8.095029  8.019290  4.946910
## 8        7.692308      -1.197852      -1.465201  6.967834  7.209325
## 9        8.346273       7.692308      -1.197852  3.829648  5.335190
## 10       5.589394       8.346273       7.692308 10.060882  8.570386
## 11       2.069902       5.589394       8.346273 14.179409 10.142906
##    Inf.Rate.MoM  Inf.L1  Inf.L2  Inf.L3  Inf.L4 Res.Change.Exc.Gold UM.Infl.Exp
## 6       0.47022 0.88286 1.02503 0.39425 1.07389            -1.90551         9.8
## 7       1.32605 0.47022 0.88286 1.02503 0.39425             9.61407         9.9
## 8       1.08417 1.32605 0.47022 0.88286 1.02503            -9.13909         9.9
## 9       0.66002 1.08417 1.32605 0.47022 0.88286           -11.96978         9.9
## 10      1.85991 0.66002 1.08417 1.32605 0.47022             0.36875         9.6
## 11      1.13271 1.85991 0.66002 1.08417 1.32605           -16.58534         9.0
##    UM.Con.Sent Indus.Prod.Ind Nasdaq.Change.MoM   NFCI FedFundsRate
## 6         68.1        17.0833          -1.57133 0.4350        10.24
## 7         65.8        17.8613           3.26679 0.7100        10.29
## 8         60.4        17.6767           2.16821 1.0300        10.47
## 9         64.5        19.4906           5.84871 1.4160        10.94
## 10        66.7        20.2075           1.71662 1.9125        11.43
## 11        62.1        18.9242          -5.81565 2.3225        13.77
##    FedFundsRateL1   Oil PercChange.oil PPIJewelry PercChange.PPI sentiment
## 6           10.01 10.71       3.678606       64.9      3.3439490 0.2570759
## 7           10.24 11.70       9.243697       67.8      4.4684129 0.2287966
## 8           10.29 13.39      14.444444       70.0      3.2448378 0.2415483
## 9           10.47 14.00       4.555639       70.5      0.7142857 0.2487900
## 10          10.94 14.57       4.071429       76.5      8.5106383 0.2780379
## 11          11.43 15.06       3.363075       86.2     12.6797386 0.1899655
##    Binary.PercChange
## 6                  1
## 7                  1
## 8                  1
## 9                  1
## 10                 1
## 11                 0
```

We converted our percent change forecast variable into a binary classifier. The hurdle rate for our binary classifier ("Binary.PercChange") is set at 0.1%, so that edge cases that are very close to 0% (0%:0.0999%) are classified as non-investment opportunities. Percent changes in gold (month over month) greater than 0.1% will be considered "buy" opportunities, with a classifier value of 1.

Initial Analysis of Features

In order to better understand our features, we'll first look at the histograms of our features as well as the scatterplots between our features and our main response variables: monthly percent changes in gold for our regression problem and our binary classiifer for our classification problem.

Histogram for Loops

```
# Set Feature Space Excluding Lagged Variables and Other Miscellenous Feature
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(stats)
library(reshape2)
gold_histograms <- gold %>% select(-c("Date","PercChangeLag1","PercChangeLag2",
                                "PercChangeLag3", "PercChangeLag4",
                                "PercChangeLag5","Inf.L1","Inf.L2",
                                "Inf.L3","Inf.L4","FedFundsRateL1"))
gold_histograms <- as.data.frame(lapply(gold_histograms, as.numeric))
```

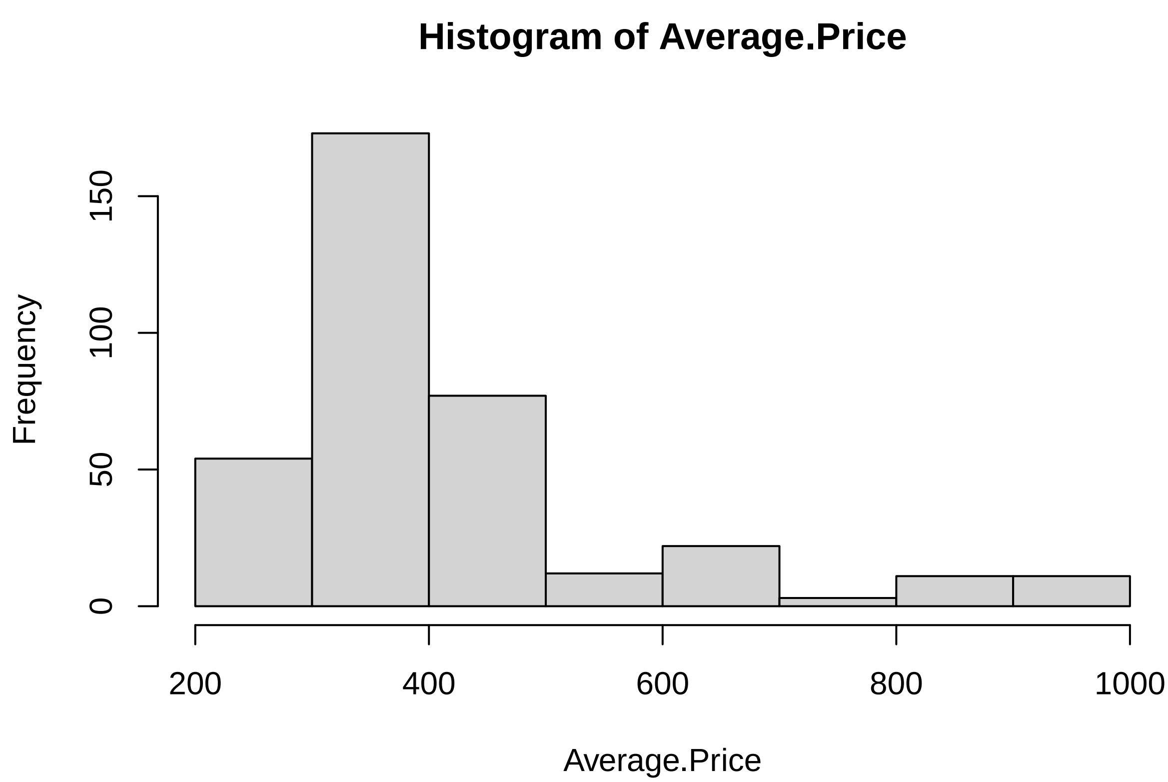
```
## Warning in lapply(gold_histograms, as.numeric): NAs introduced by coercion
```

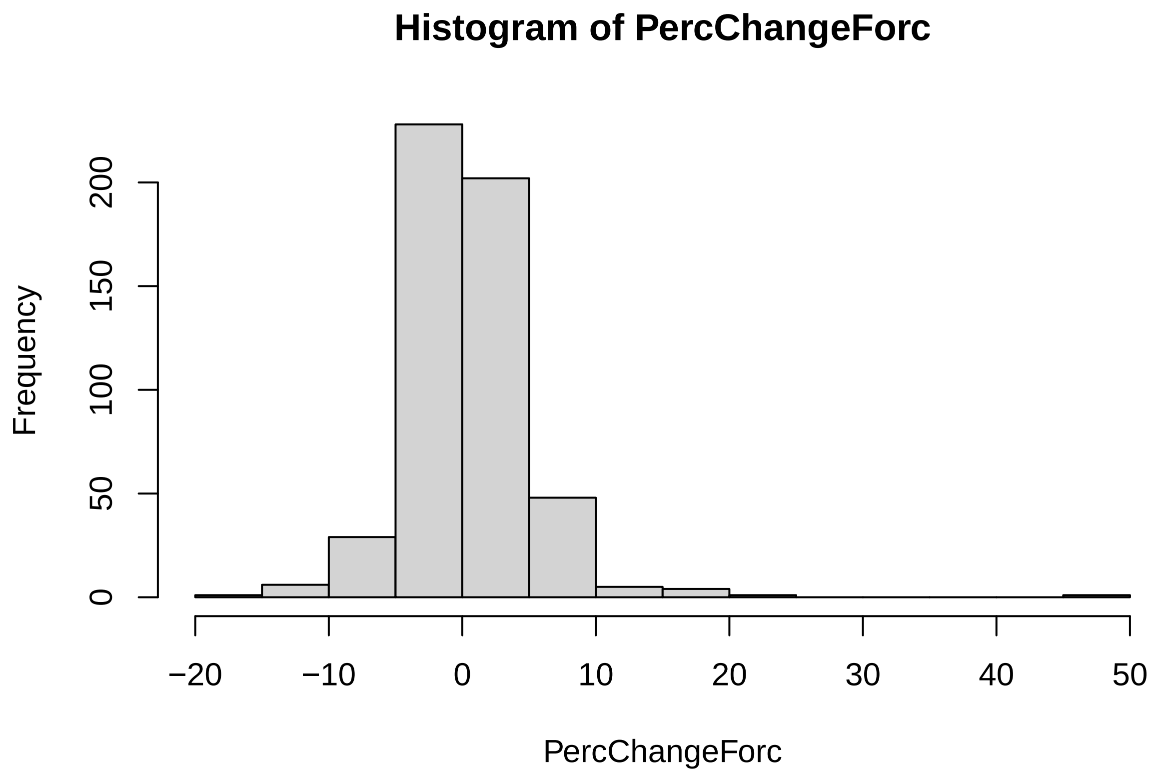
```
head(gold_histograms)
```

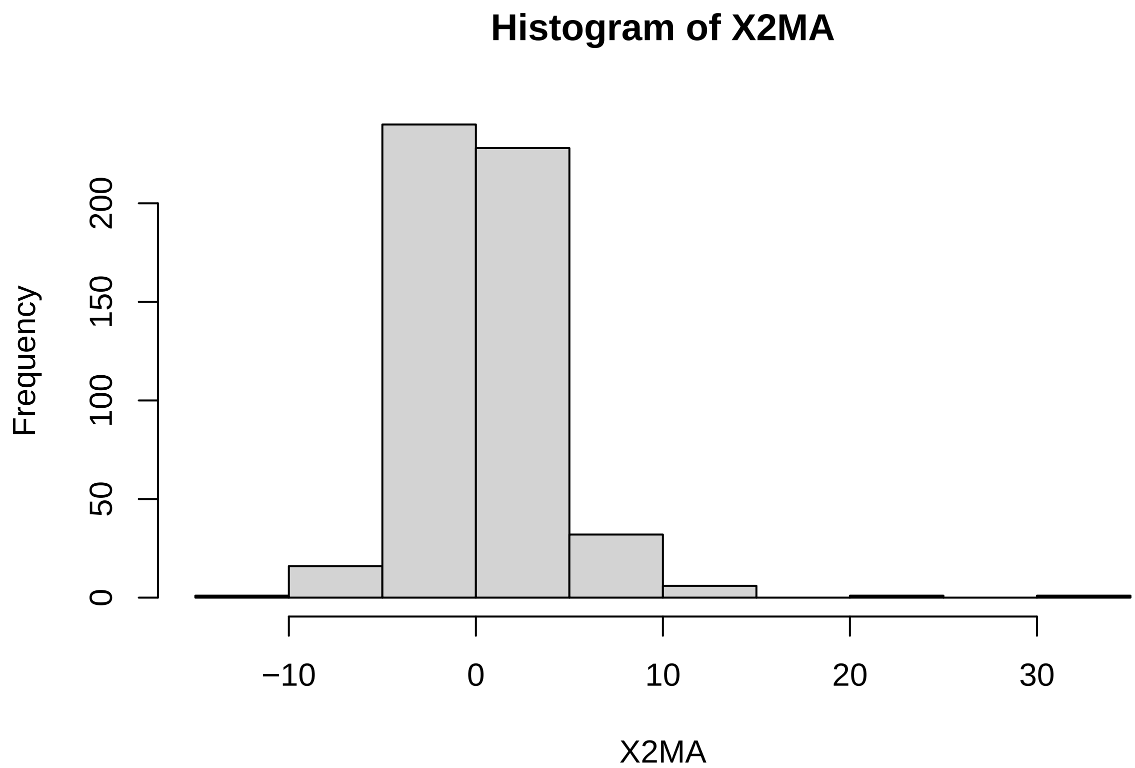
```
##   Average.Price PercChangeForc      X2MA      X3MA Inf.Rate.MoM
## 1         257.6     8.34627329  3.247228  1.676418      0.47022
## 2         279.1     5.58939448  8.019290  4.946910      1.32605
## 3         294.7     2.06990160  6.967834  7.209325      1.08417
## 4         300.8    18.05186170  3.829648  5.335190      0.66002
## 5         355.1    10.30695579 10.060882  8.570386      1.85991
## 6         391.7     0.07658923 14.179409 10.142906      1.13271
##   Res.Change.Exc.Gold UM.Infl.Exp UM.Con.Sent Indus.Prod.Ind Nasdaq.Change.MoM
## 1            -1.90551         9.8        68.1        17.0833          -1.57133
## 2             9.61407         9.9        65.8        17.8613           3.26679
## 3            -9.13909         9.9        60.4        17.6767           2.16821
## 4           -11.96978         9.9        64.5        19.4906           5.84871
## 5             0.36875         9.6        66.7        20.2075           1.71662
## 6           -16.58534         9.0        62.1        18.9242          -5.81565
```

```
##     NFCI FedFundsRate   Oil PercChange.oil PPIJewelry PercChange.PPI sentiment
## 1 0.4350        10.24 10.71       3.678606       64.9      3.3439490 0.2570759
## 2 0.7100        10.29 11.70       9.243697       67.8      4.4684129 0.2287966
## 3 1.0300        10.47 13.39      14.444444       70.0      3.2448378 0.2415483
## 4 1.4160        10.94 14.00       4.555639       70.5      0.7142857 0.2487900
## 5 1.9125        11.43 14.57       4.071429       76.5      8.5106383 0.2780379
## 6 2.3225        13.77 15.06       3.363075       86.2     12.6797386 0.1899655
##   Binary.PercChange
## 1                 1
## 2                 1
## 3                 1
## 4                 1
## 5                 1
## 6                 0
```

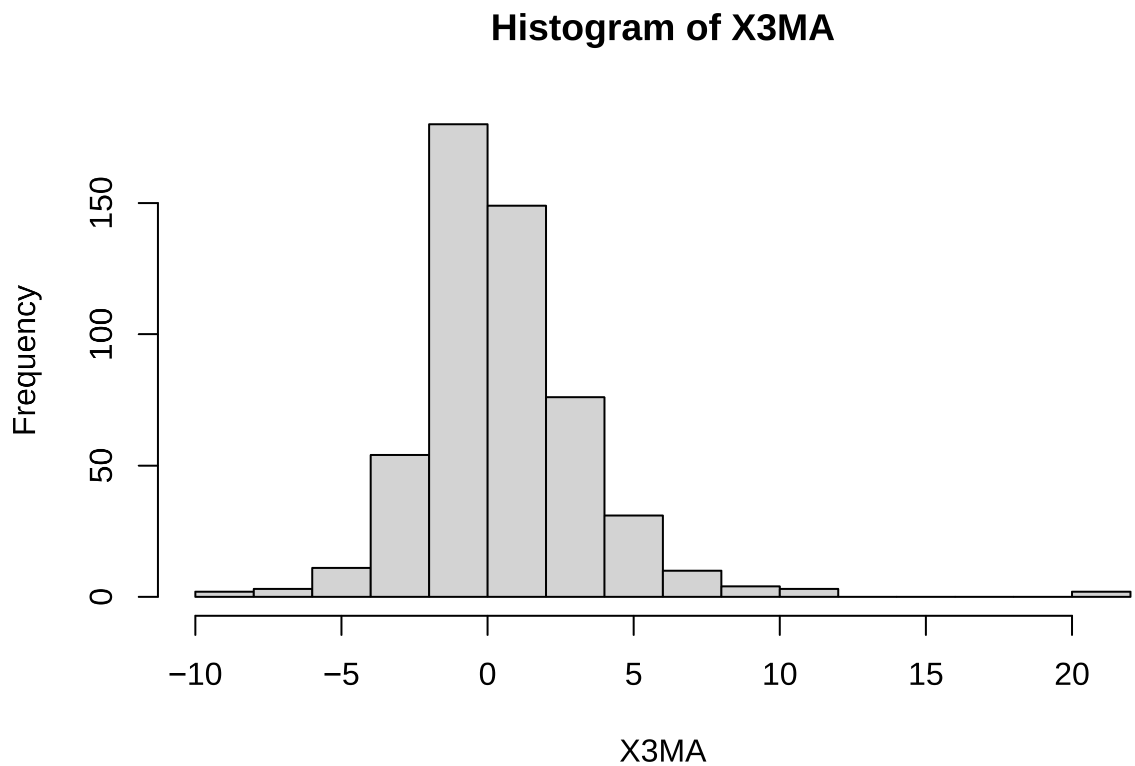
```r
# For Loop for Histograms
for (feature in names(gold_histograms)){
  hist(gold_histograms[[feature]], xlab = paste0(feature), ylab = "Frequency",
       main = paste0("Histogram of ",feature))
}
```

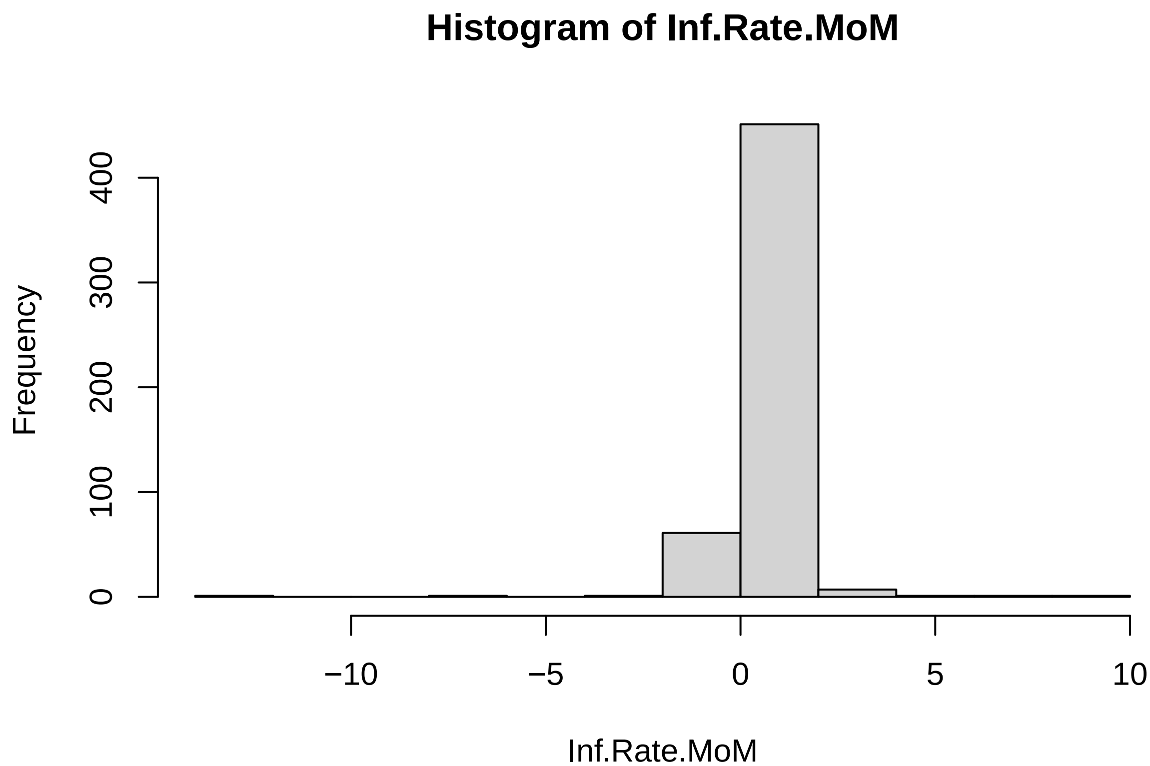


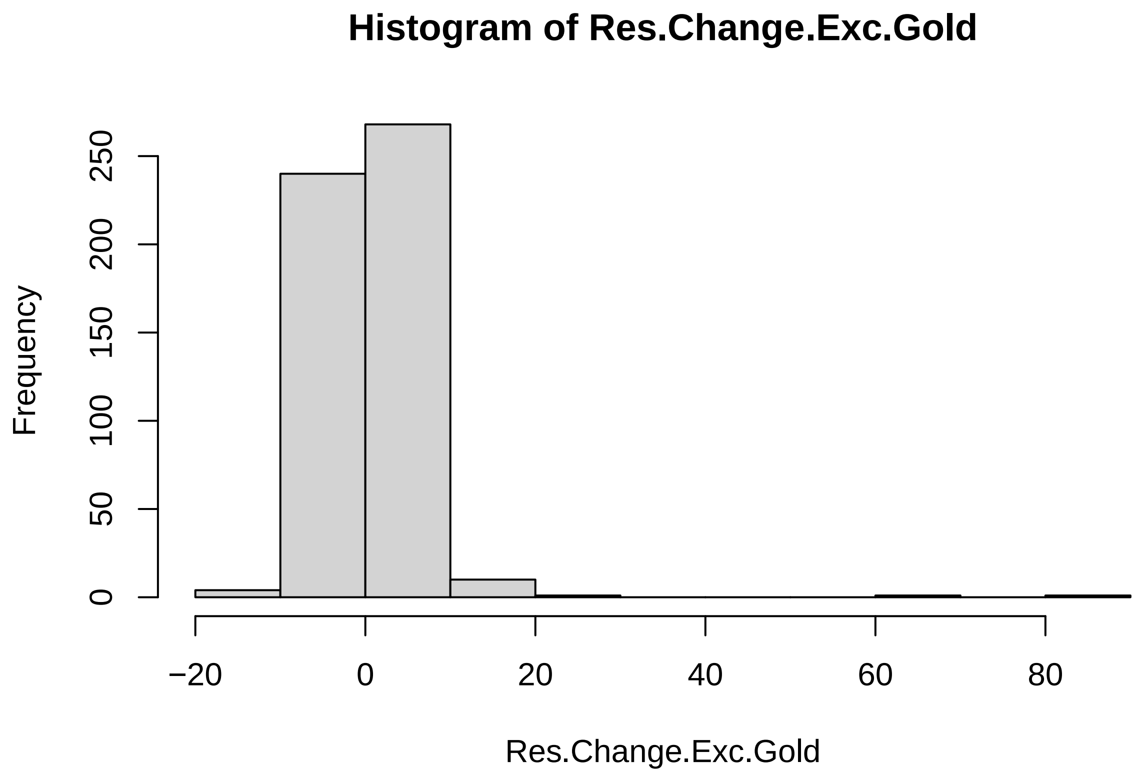

Histogram of PercChangeForc
Frequency
0
50
100
150
200
−20
−10
0
10
20
30
40
50
PercChangeForc

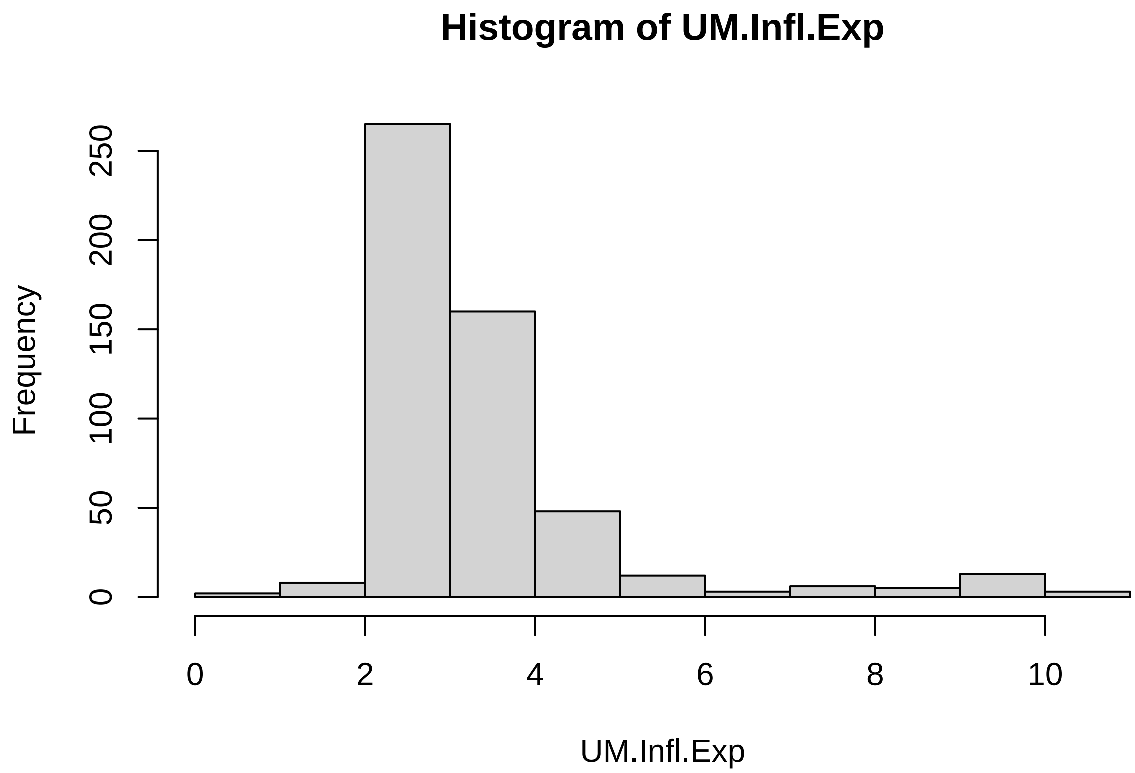


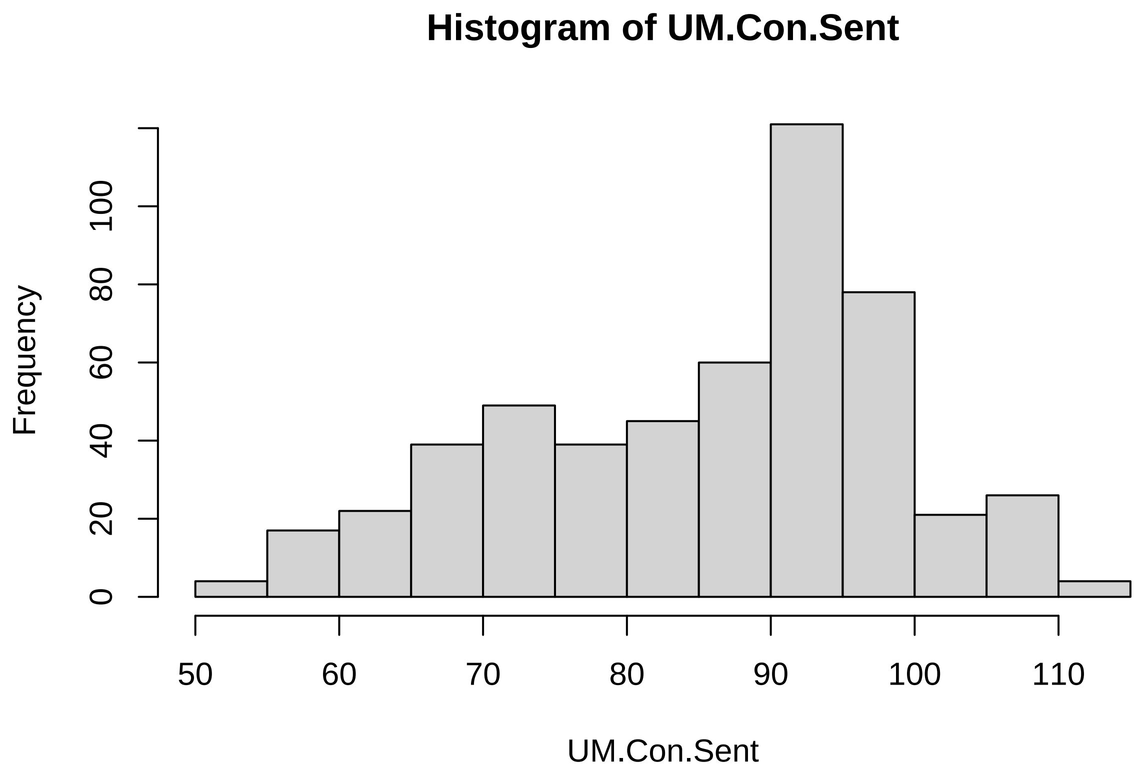
Histogram of X2MA
Frequency
0
50
100
150
200
−10
0
10
20
30
X2MA

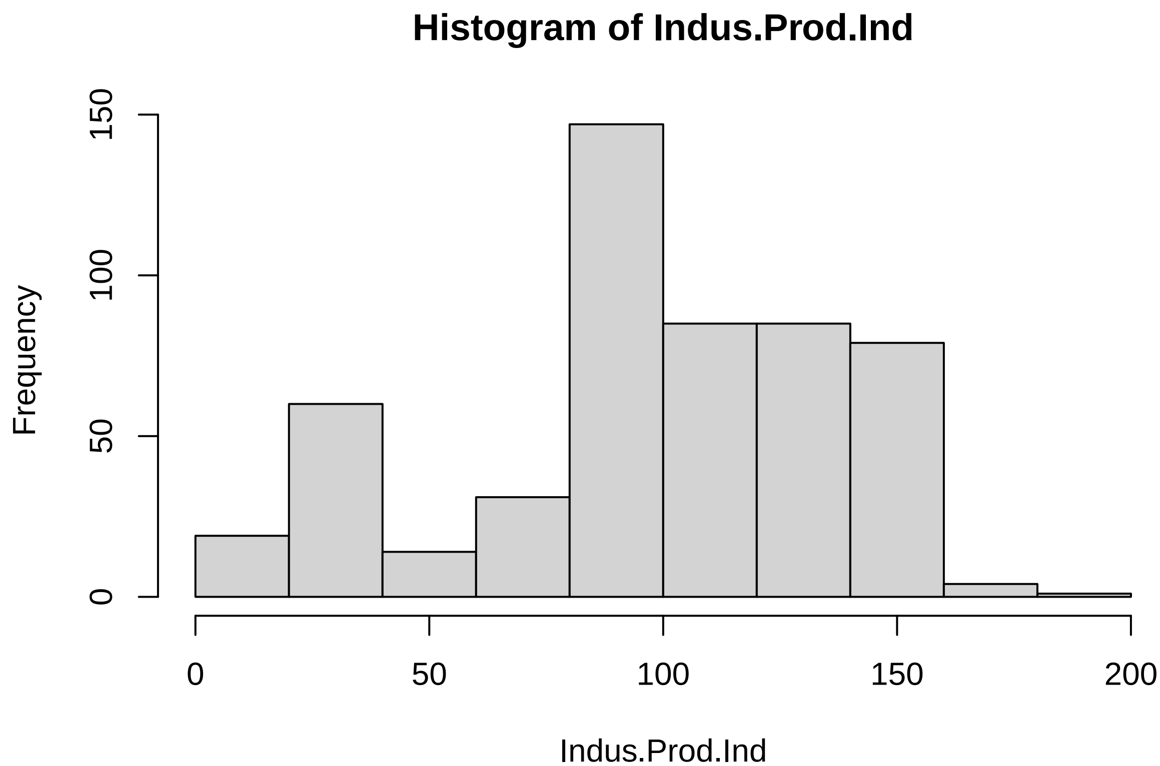

Histogram of X3MA
Frequency
0
50
100
150
−10
−5
0
5
10
15
20
X3MA

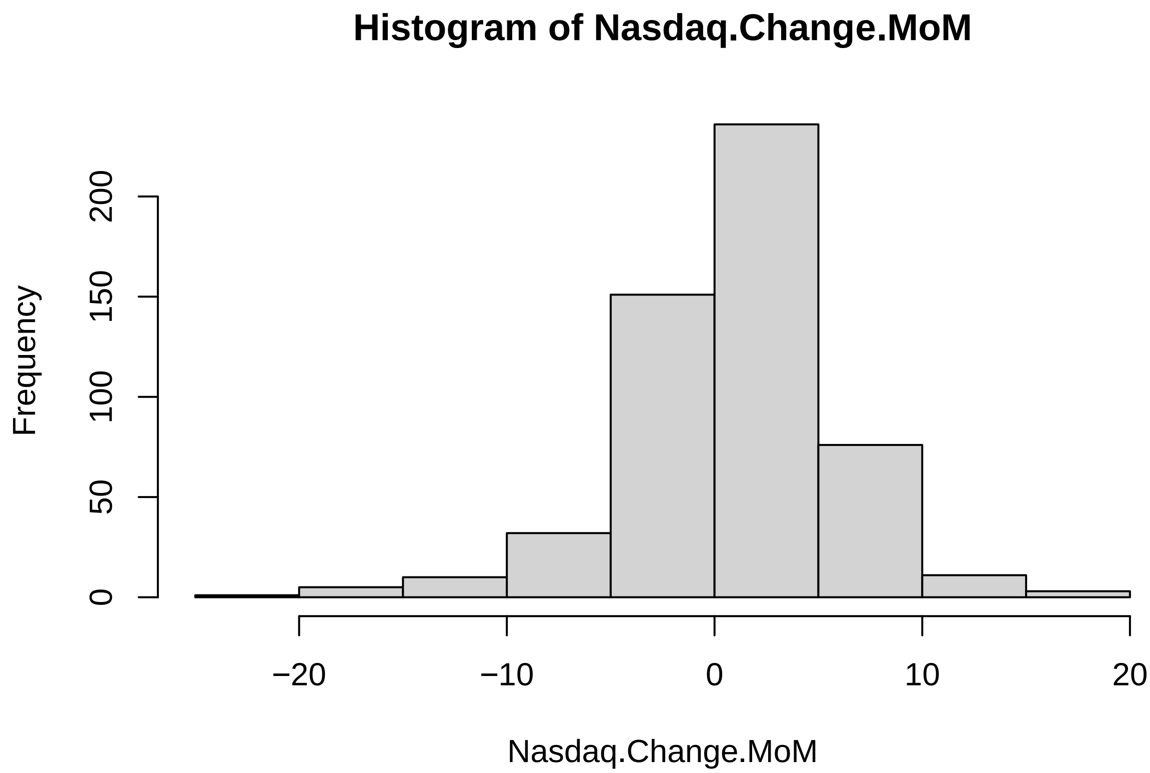

Histogram of Inf.Rate.MoM
Frequency
0
100
200
300
400
−10
−5
0
5
10
Inf.Rate.MoM

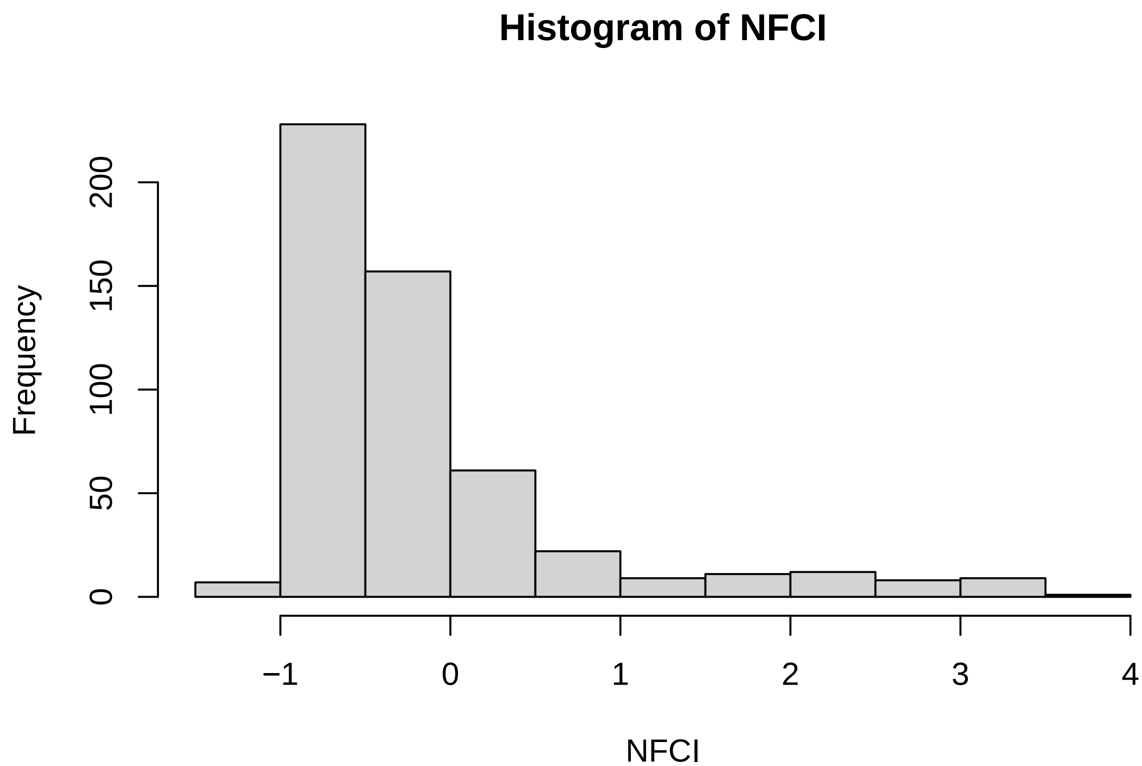

Histogram of Res.Change.Exc.Gold
Frequency
0
50
100
150
200
250
−20
0
20
40
60
80
Res.Change.Exc.Gold

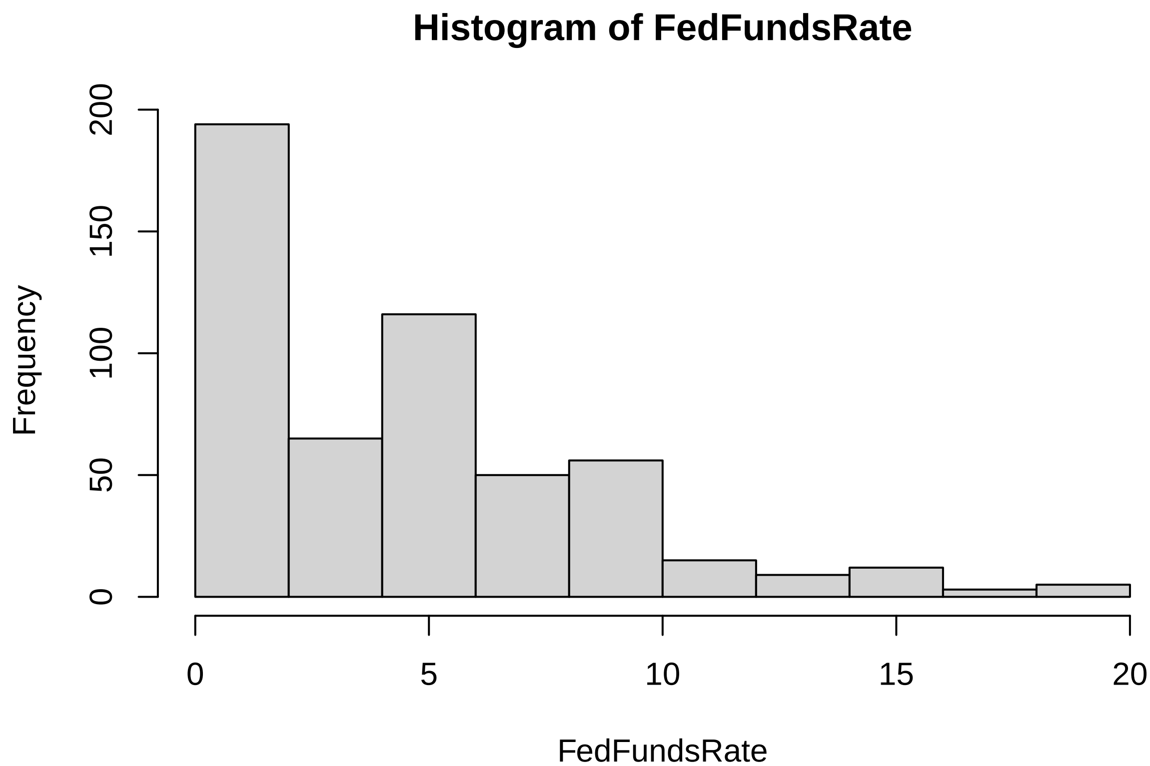

Histogram of UM.Infl.Exp
Frequency
0
50
100
150
200
250
0
2
4
6
8
10
UM.Infl.Exp


Histogram of UM.Con.Sent
Frequency
0
20
40
60
80
100
50
60
70
80
90
100
110
UM.Con.Sent

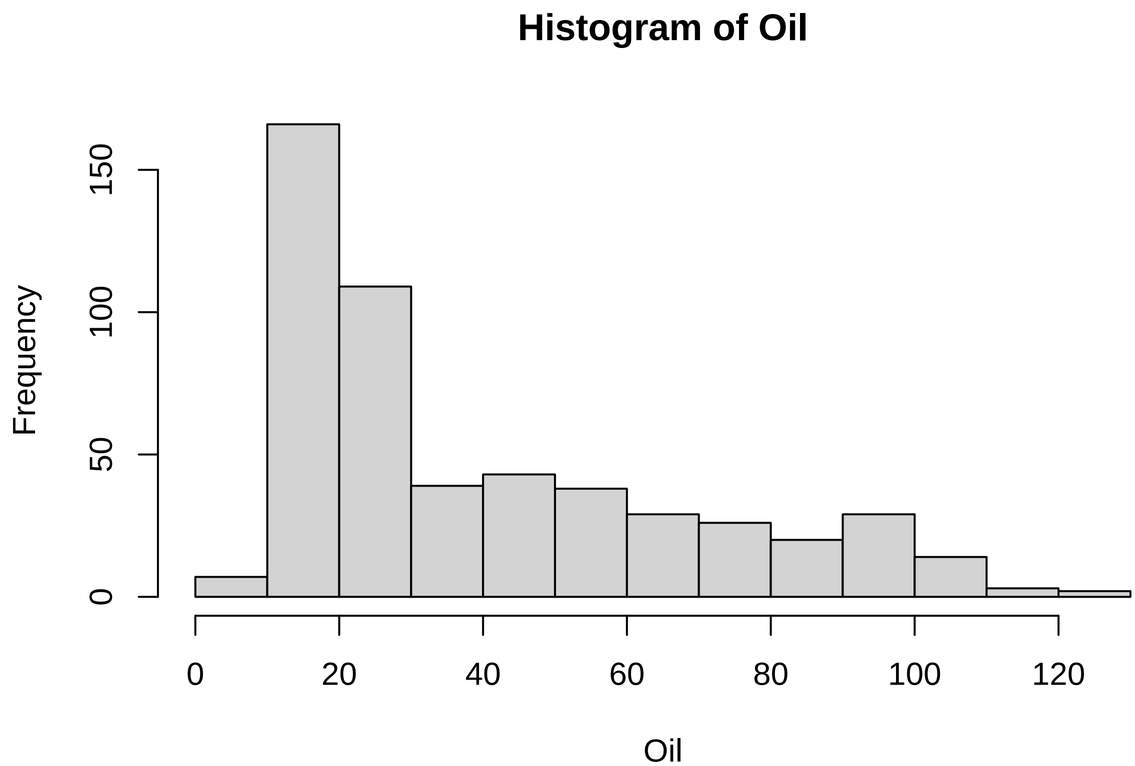

Histogram of Indus.Prod.Ind
Frequency
0
50
100
150
0
50
100
150
200
Indus.Prod.Ind


Histogram of Nasdaq.Change.MoM
Frequency
0
50
100
150
200
−20
−10
0
10
20
Nasdaq.Change.MoM

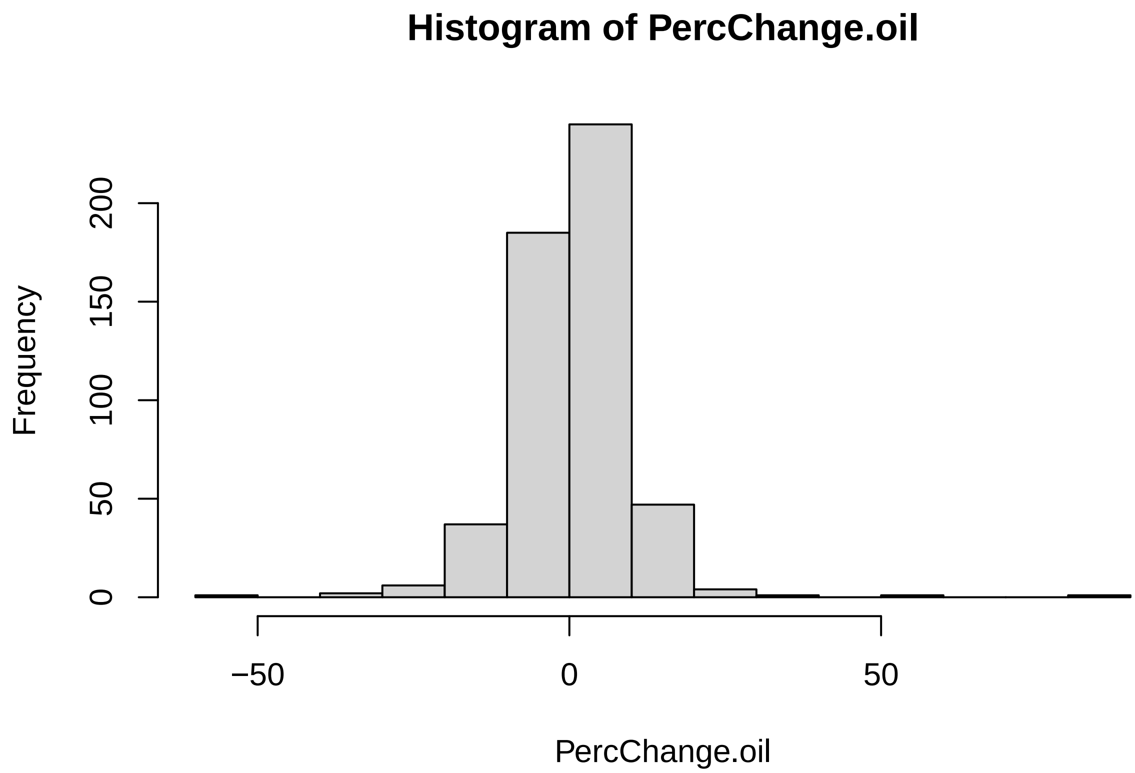

Histogram of NFCI
Frequency
0
50
100
150
200
−1
0
1
2
3
4
NFCI



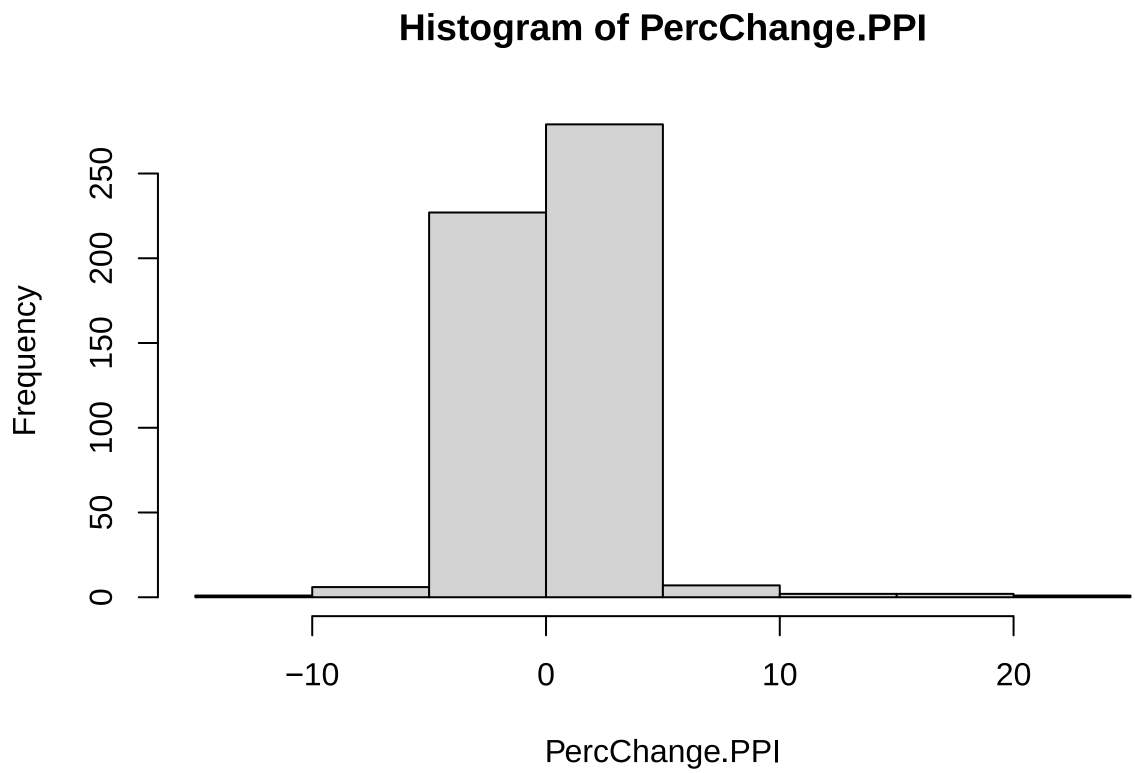
Histogram of FedFundsRate
Frequency
0
50
100
150
200
0
5
10
15
20
FedFundsRate



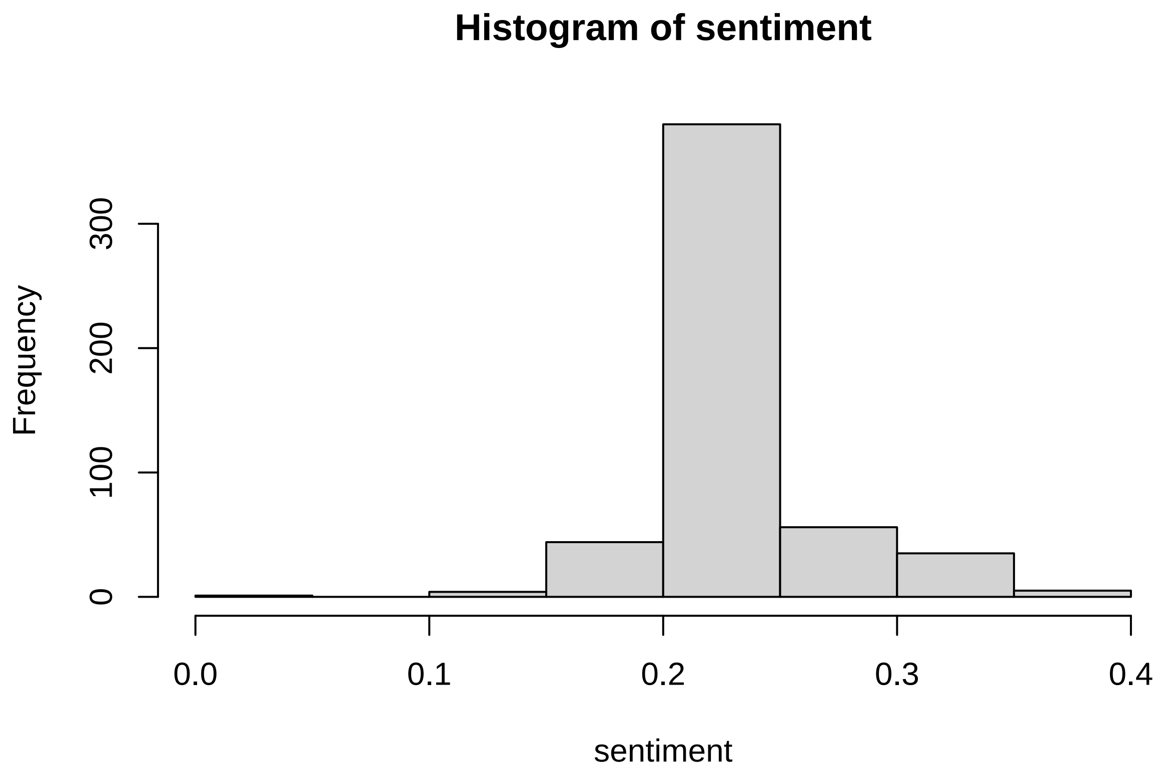
Histogram of Oil
Frequency
0
50
100
150
0
20
40
60
80
100
120
Oil

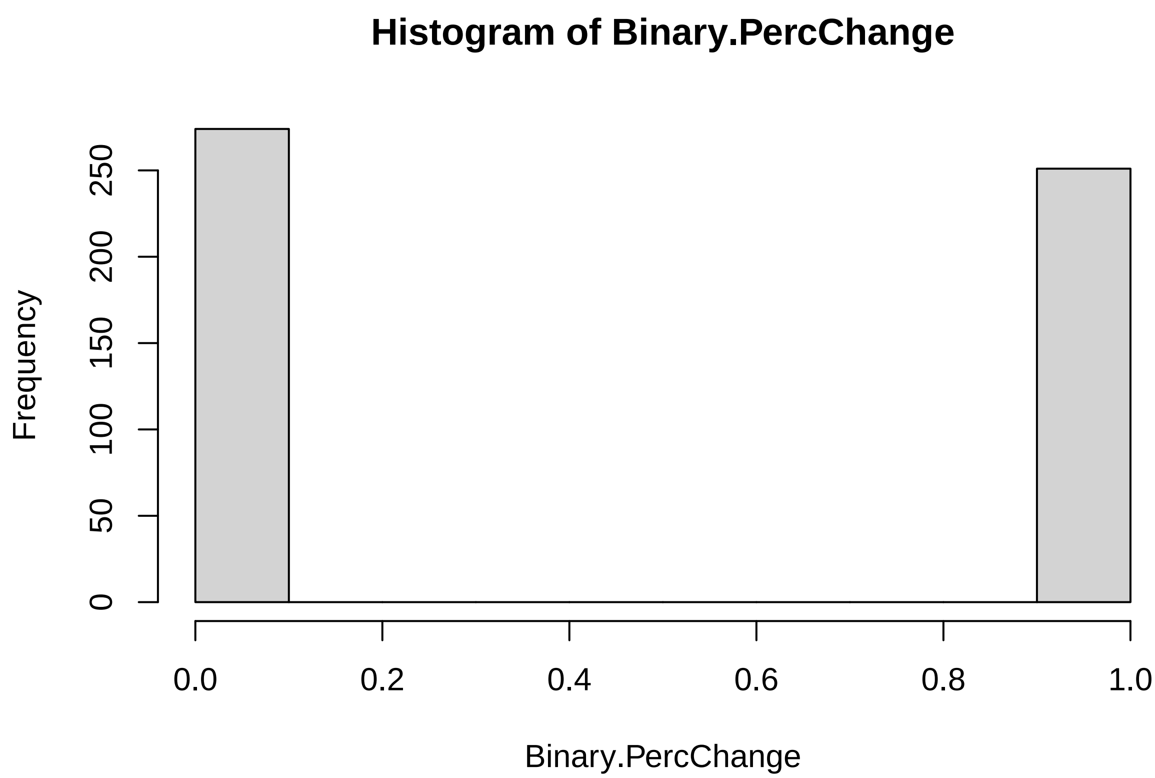

Histogram of PercChange.oil
Frequency
0
50
100
150
200
−50
0
50
PercChange.oil


Histogram of PPIJewelry
Frequency
0
50
100
150
200
100
150
200
250
300
PPIJewelry

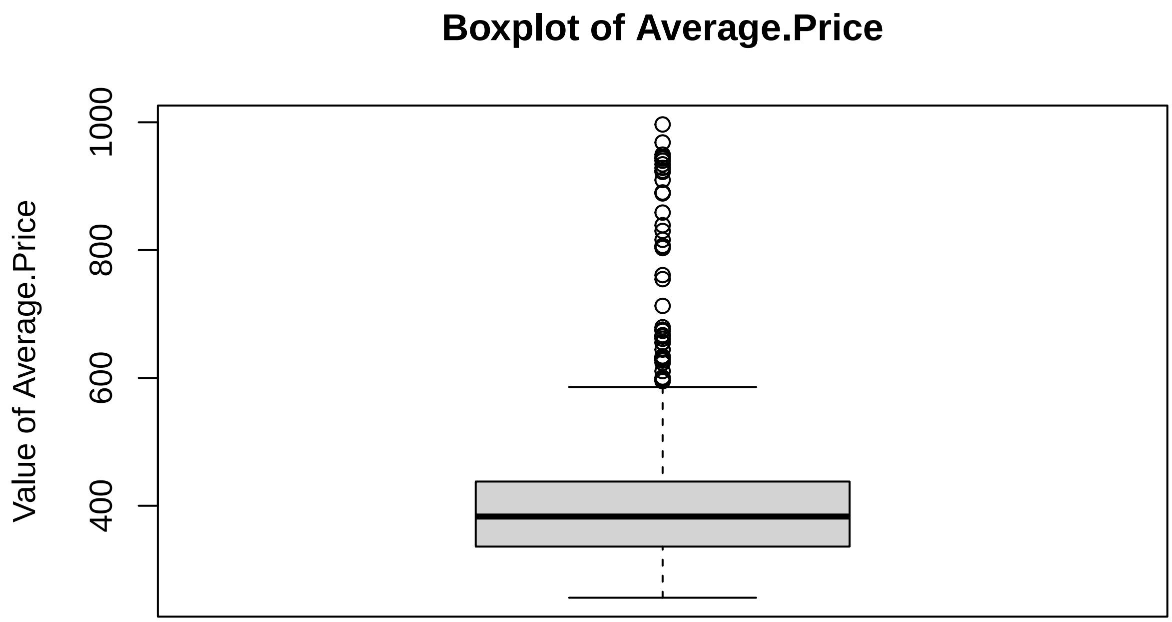


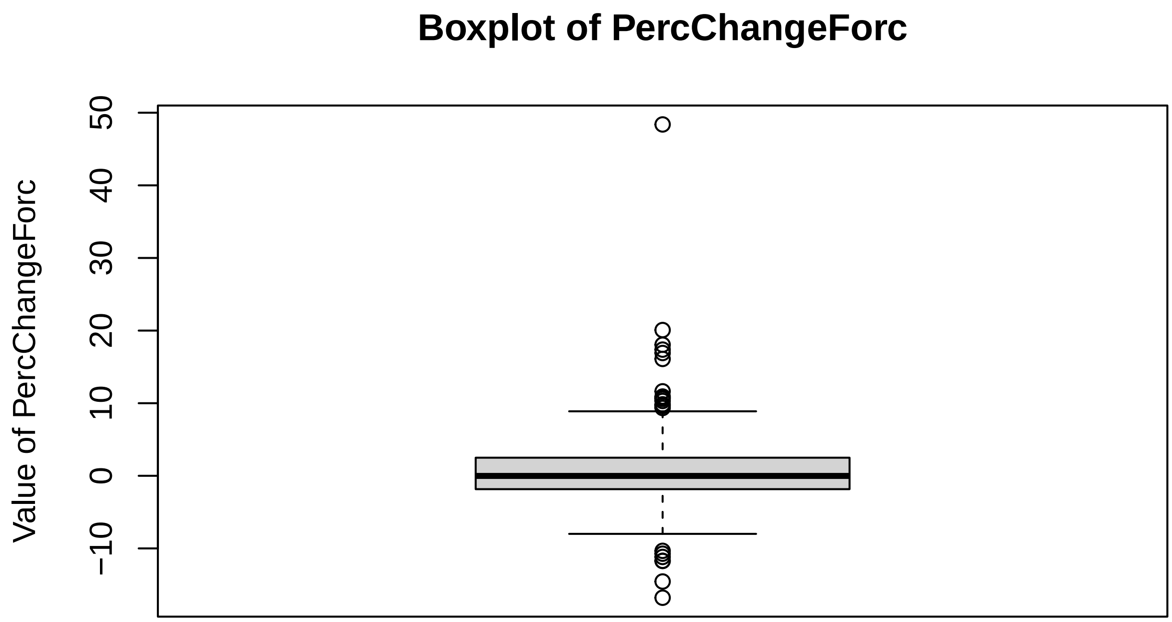
Histogram of PercChange.PPI
Frequency
0
50
100
150
200
250
−10
0
10
20
PercChange.PPI

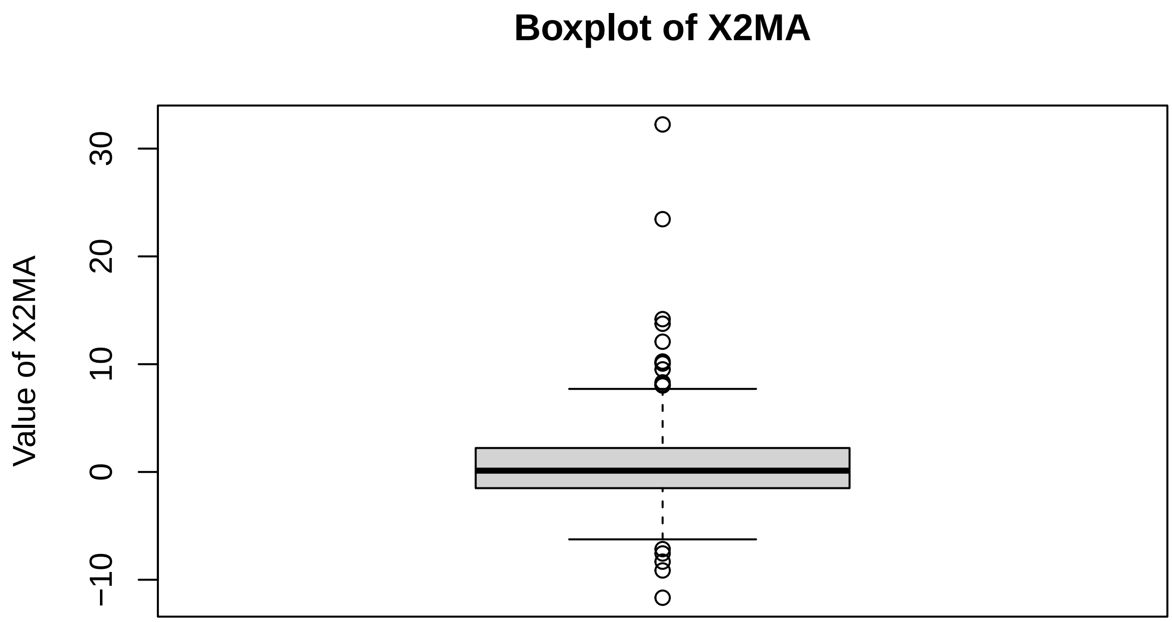

Histogram of sentiment
Frequency
0
100
200
300
0.0
0.1
0.2
0.3
0.4
sentiment

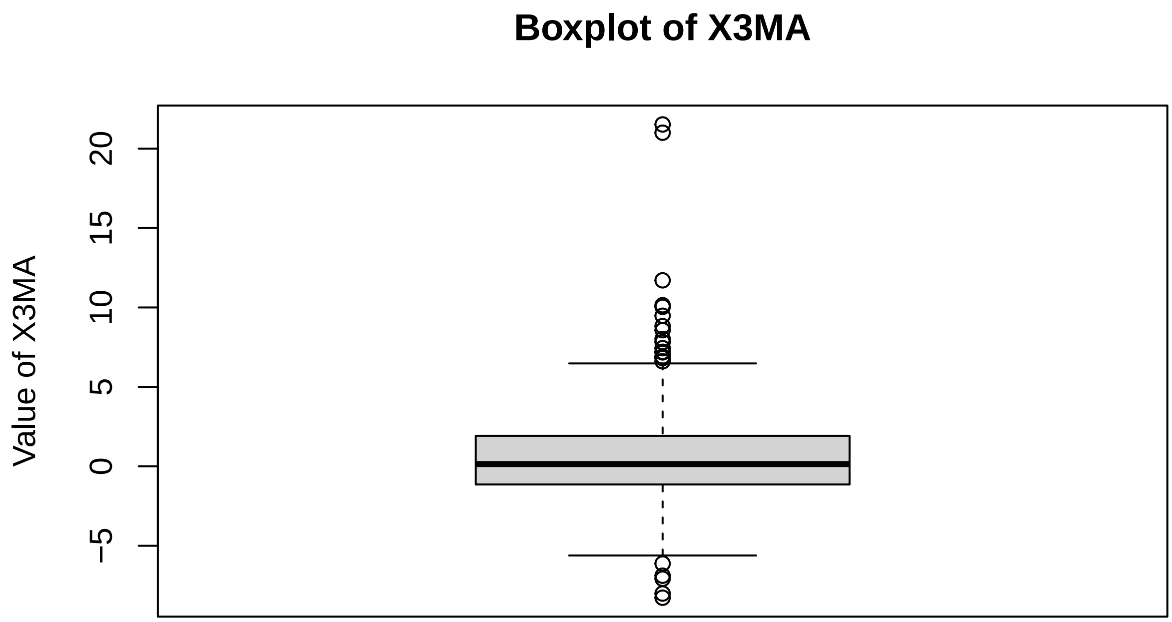


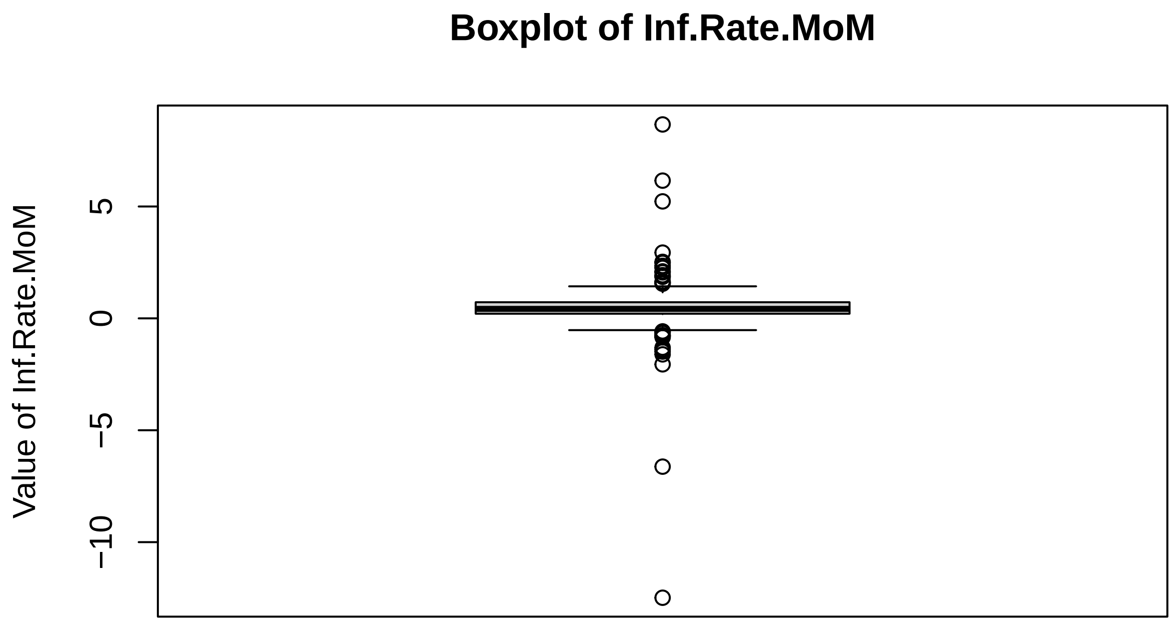

```
main.hist <- ggplot(gold, aes(x = PercChangeForc)) +
  geom_histogram(binwidth = 2, fill = "navy", color = "lightgrey") +
  labs(x = "Gold Price % Change", y = "Frequency") +
  ggtitle("Percent Change of Gold Price") +
  coord_cartesian(xlim = c(-20, 25))+
  theme_minimal()
# ggsave(file = "~/Desktop/main_hist.png", plot = main.hist, width = 10, height = 6, bg = "white")




# For Loop for Box Plots
for (feature in names(gold_histograms)){
  boxplot(gold_histograms[[feature]], ylab = paste0("Value of ",feature),
       main = paste0("Boxplot of ",feature))
}
```


Boxplot of Average.Price
Value of Average.Price
1000
800
600
400

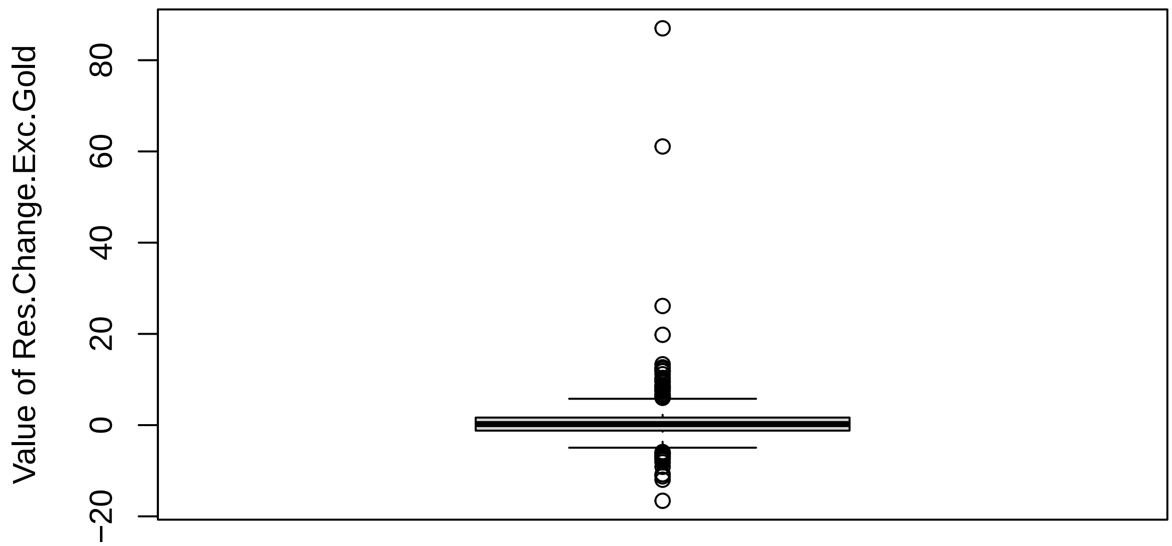

Boxplot of PercChangeForc
Value of PercChangeForc
50
40
30
20
10
0
−10

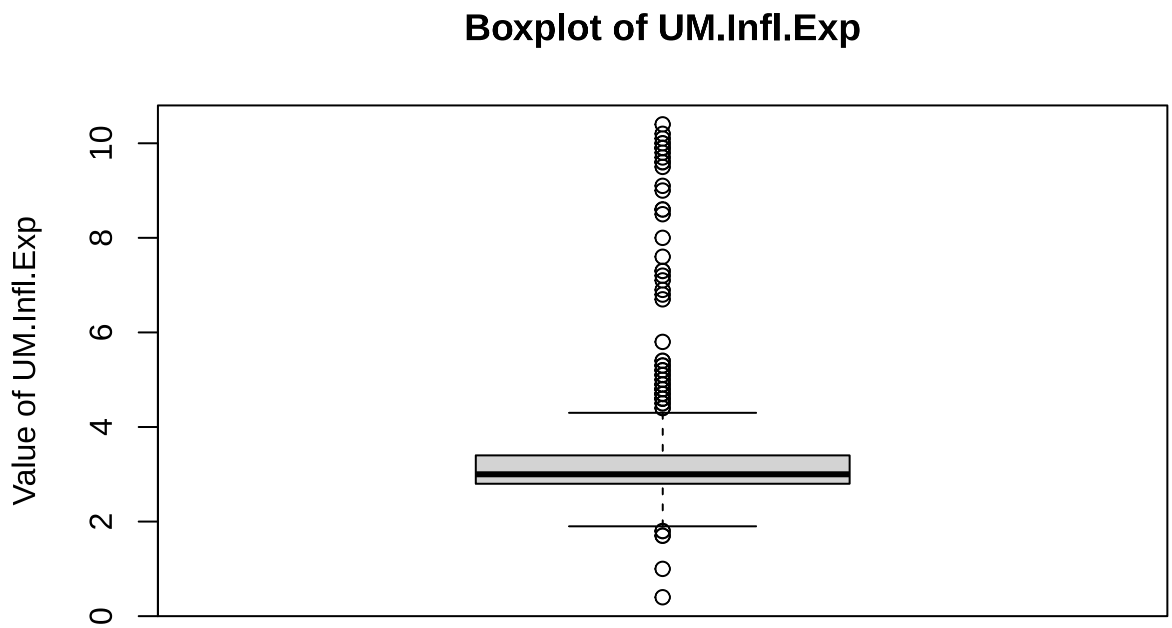

Boxplot of X2MA
Value of X2MA
30
20
10
0
−10

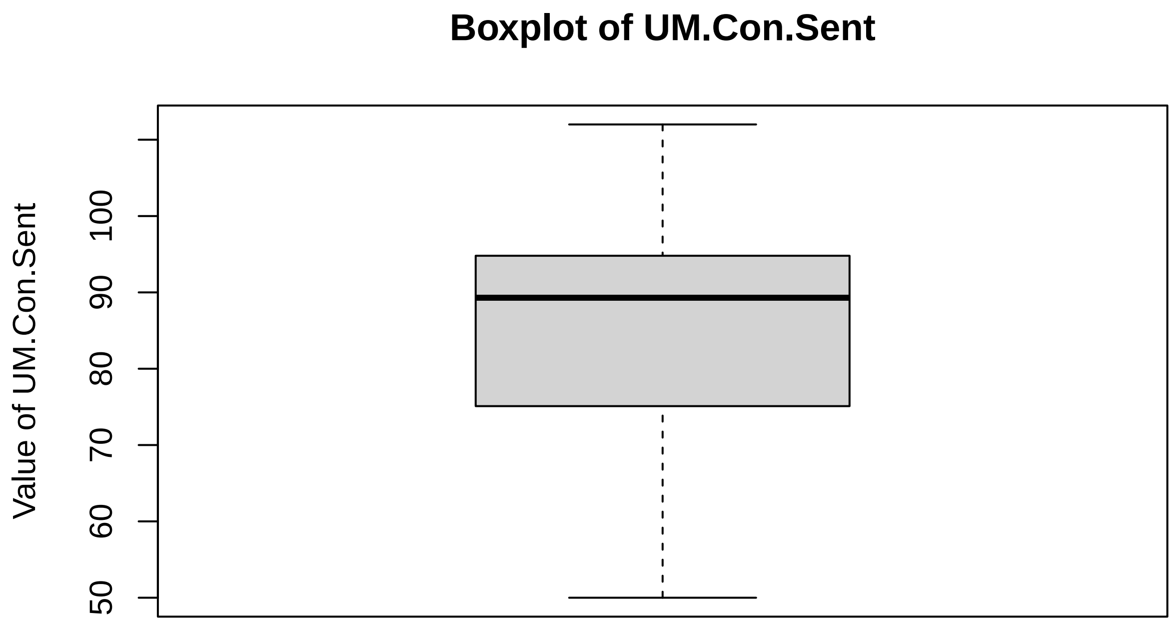

Boxplot of X3MA
Value of X3MA
−5
0
5
10
15
20

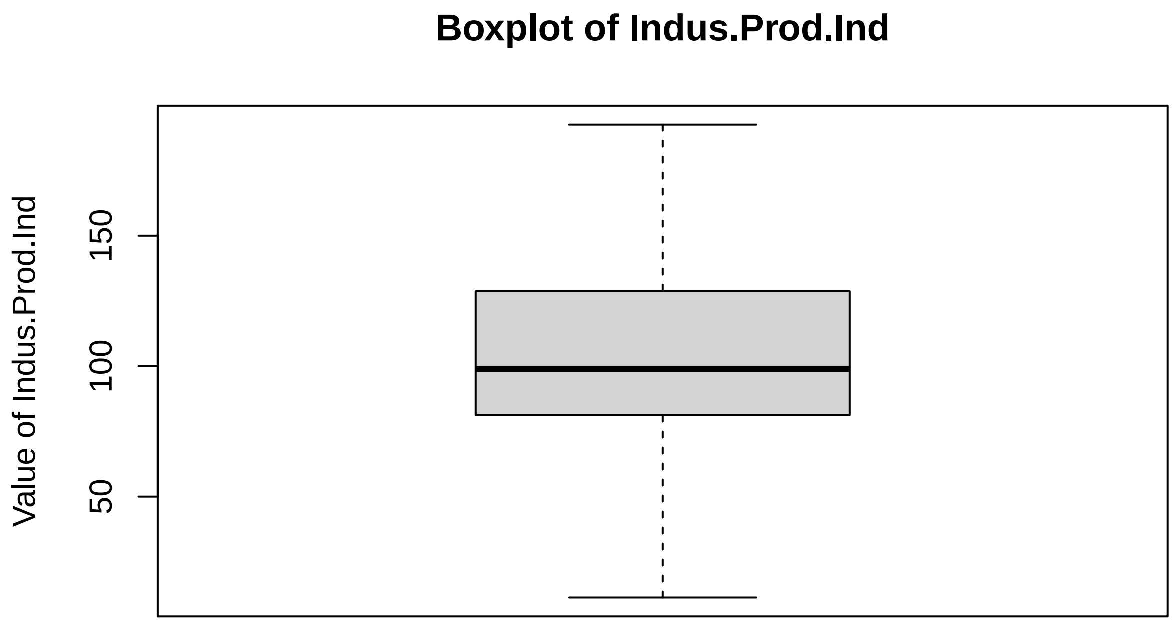

Boxplot of Inf.Rate.MoM
Value of Inf.Rate.MoM
5
0
−5
−10

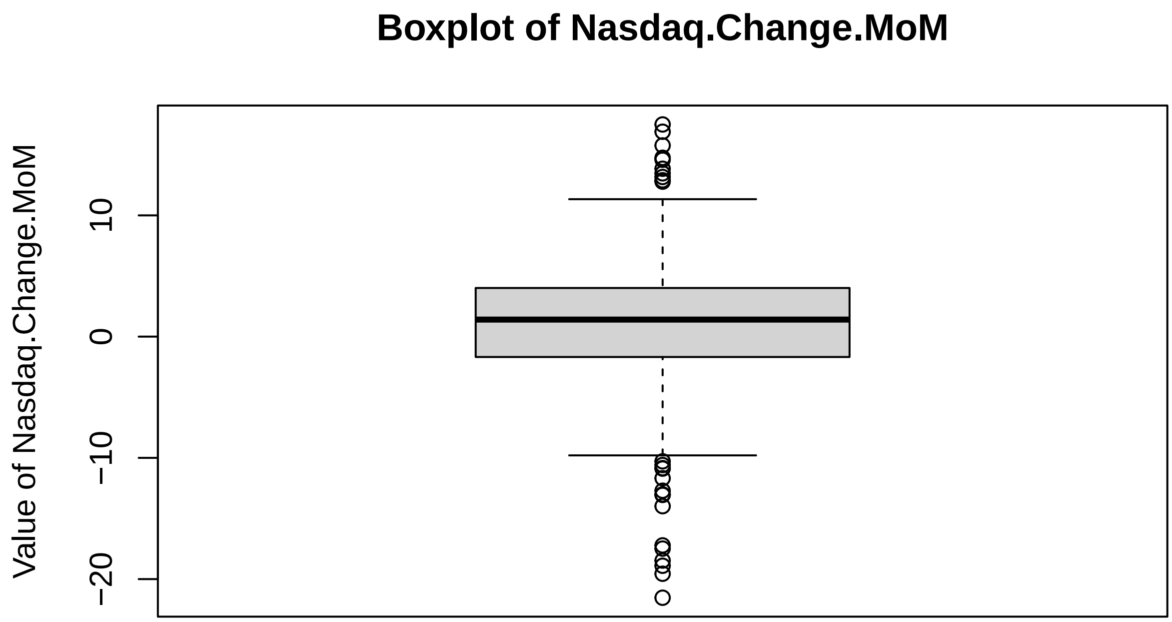
Boxplot of Res.Change.Exc.Gold

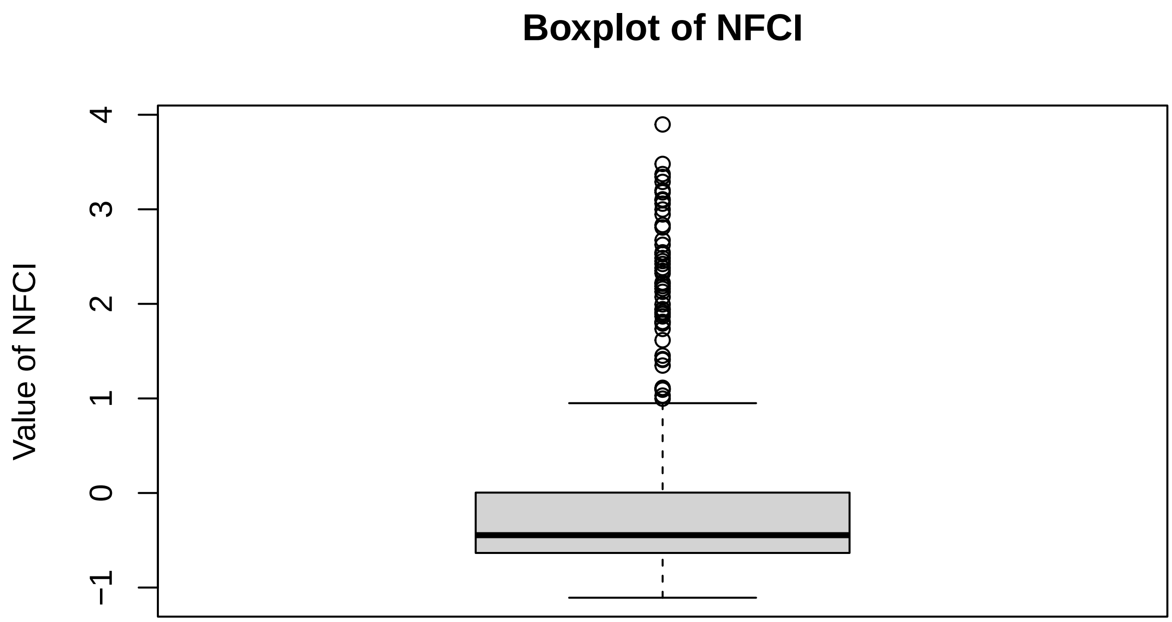


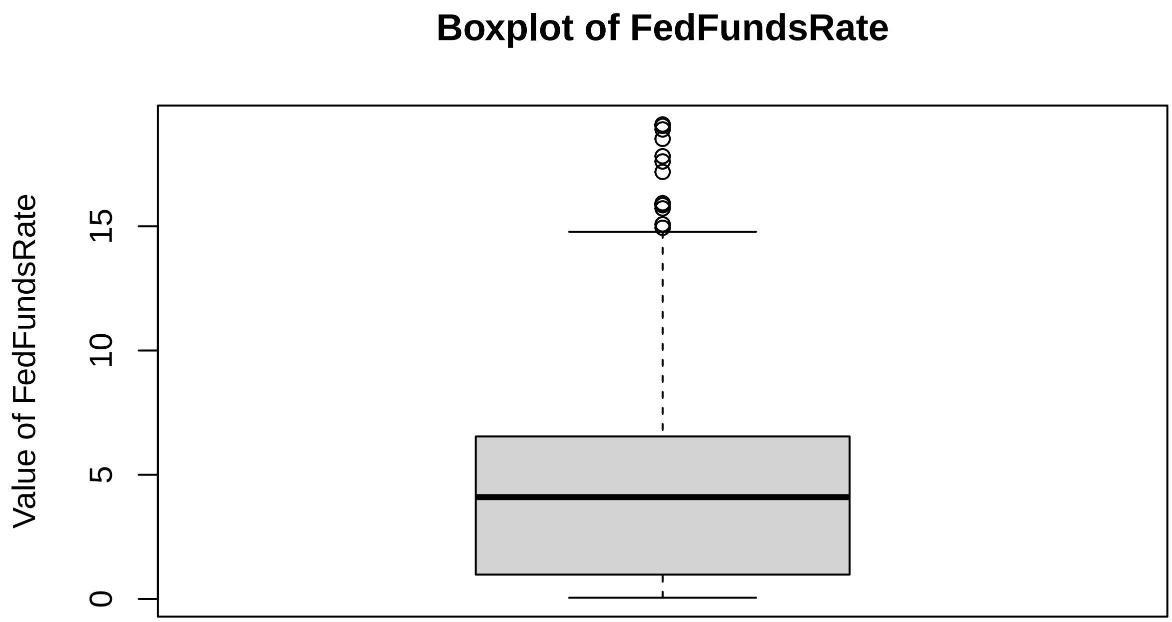

Boxplot of UM.Infl.Exp
Value of UM.Infl.Exp
0
2
4
6
8
10

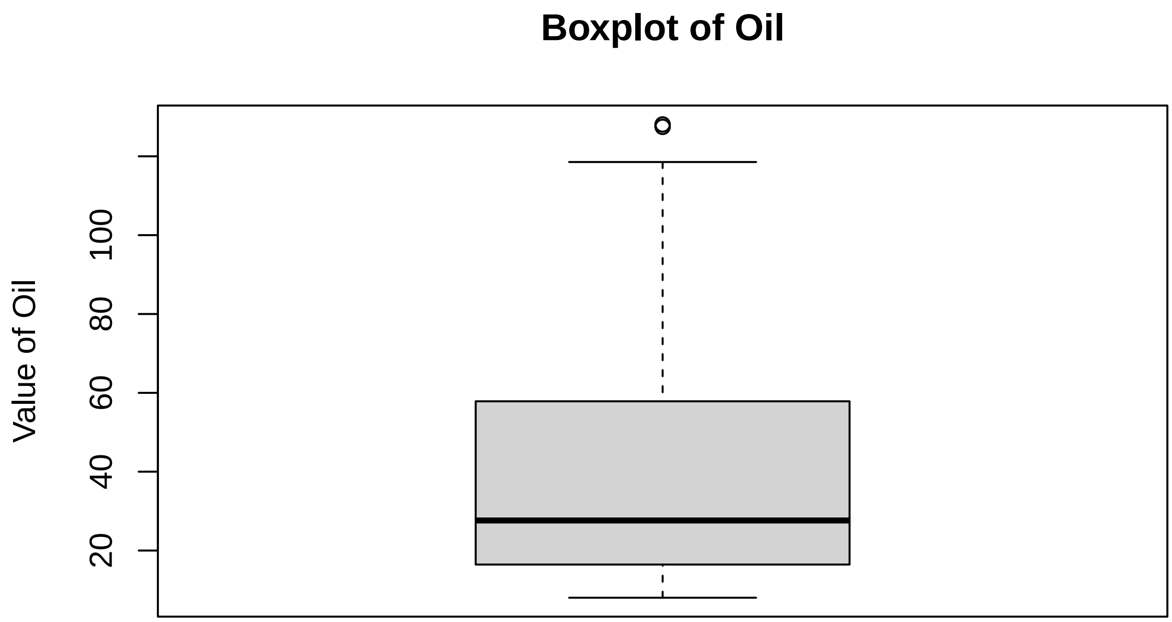

Boxplot of UM.Con.Sent
Value of UM.Con.Sent
50
60
70
80
90
100

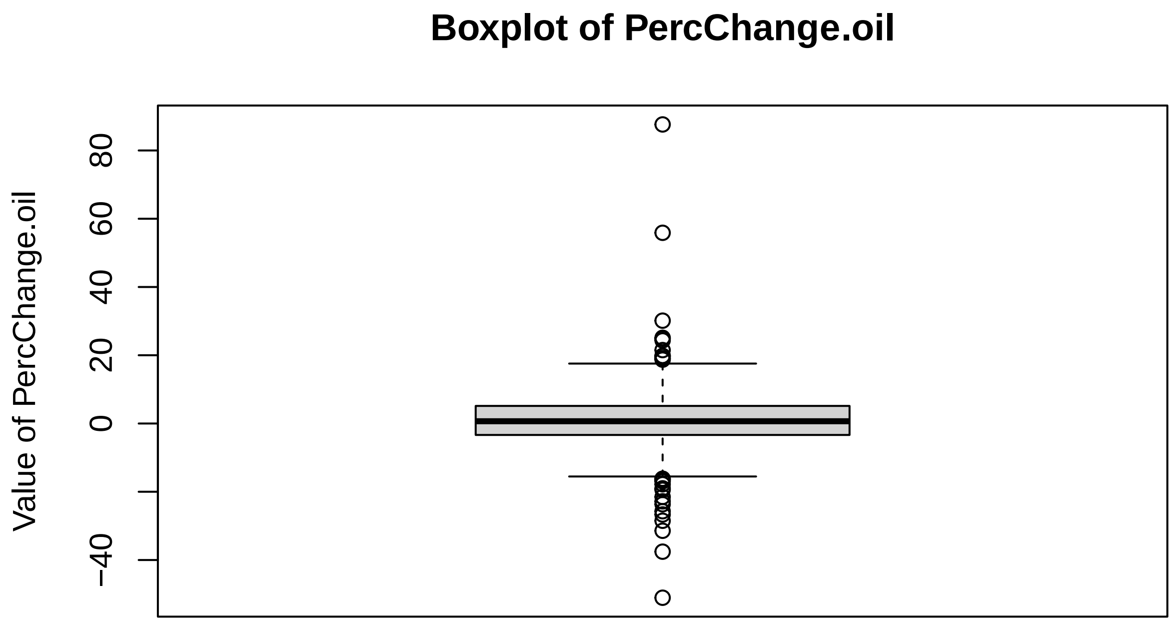

Boxplot of Indus.Prod.Ind
Value of Indus.Prod.Ind
150
100
50

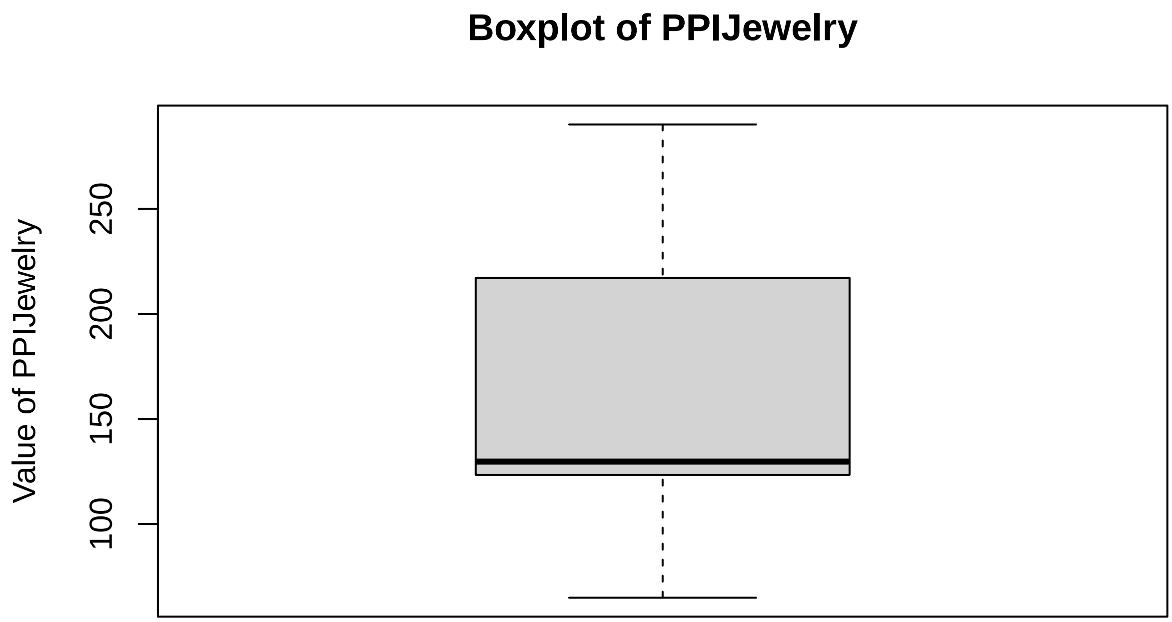

Boxplot of Nasdaq.Change.MoM
Value of Nasdaq.Change.MoM
10
0
−10
−20

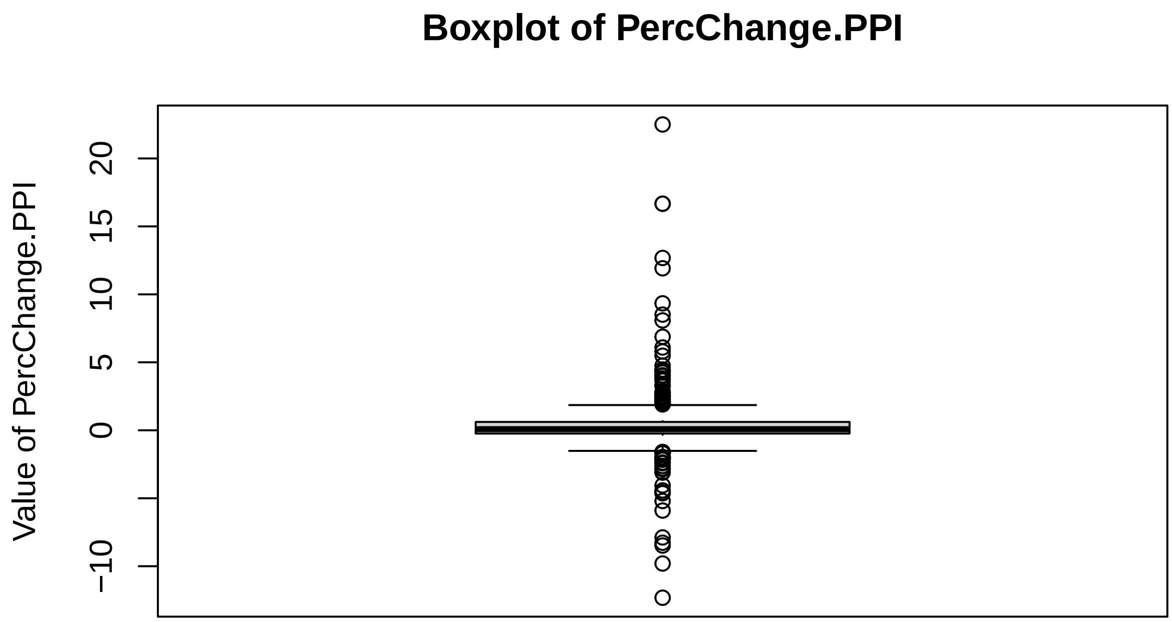

Boxplot of NFCI
Value of NFCI
4
3
2
1
0
−1

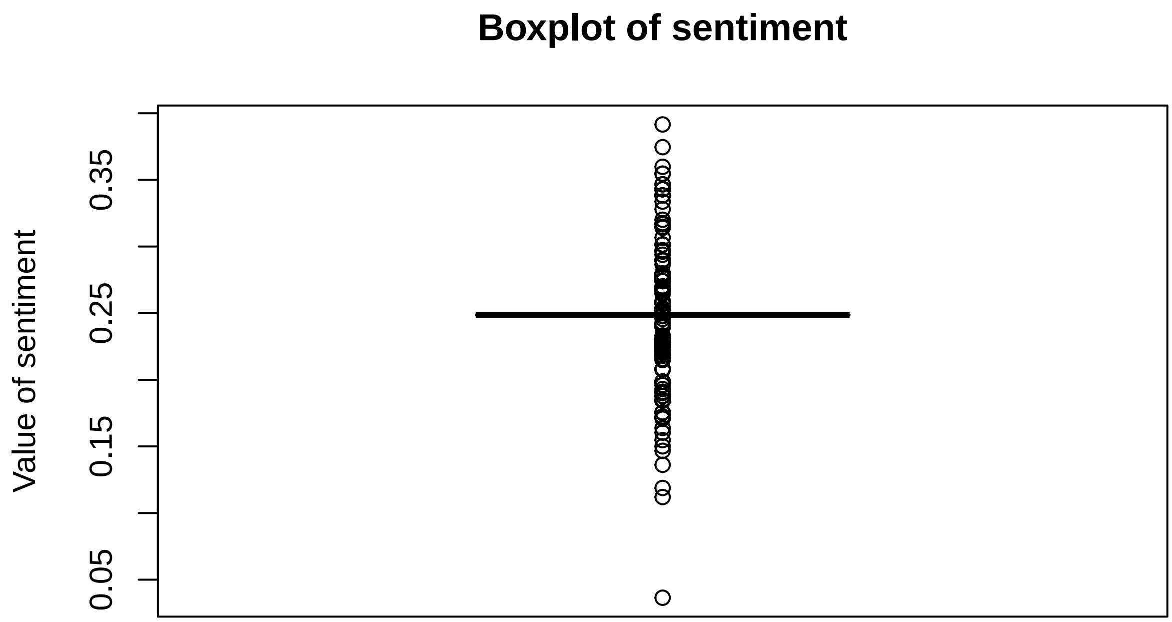

Boxplot of FedFundsRate
Value of FedFundsRate
0
5
10
15

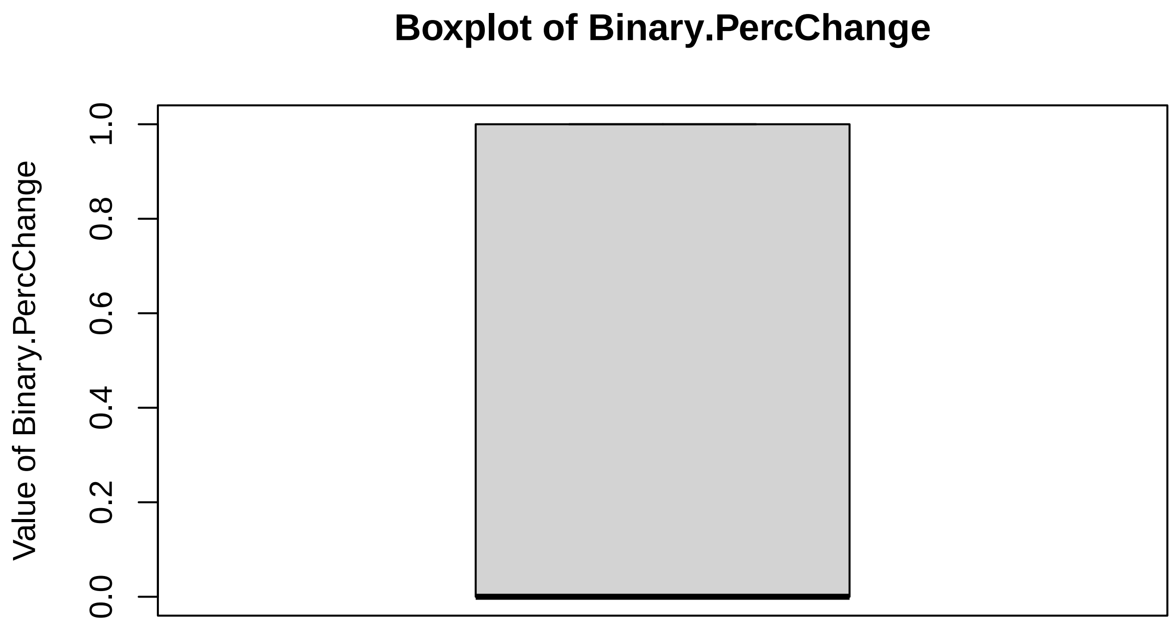

Boxplot of Oil
Value of Oil
20
40
60
80
100


Boxplot of PercChange.oil
Value of PercChange.oil
80
60
40
20
0
−40


Boxplot of PPIJewelry
Value of PPIJewelry
100
150
200
250


Boxplot of PercChange.PPI
Value of PercChange.PPI
20
15
10
5
0
−10


Boxplot of sentiment
Value of sentiment
0.05
0.15
0.25
0.35



The boxplots and histograms show that our variables are on average normally distributed. There are outliers in some of our features, such as percent changes in monthly prices and changes in bank reserves. However, we believe these outliers are still essential in predicting the movement of gold prices.

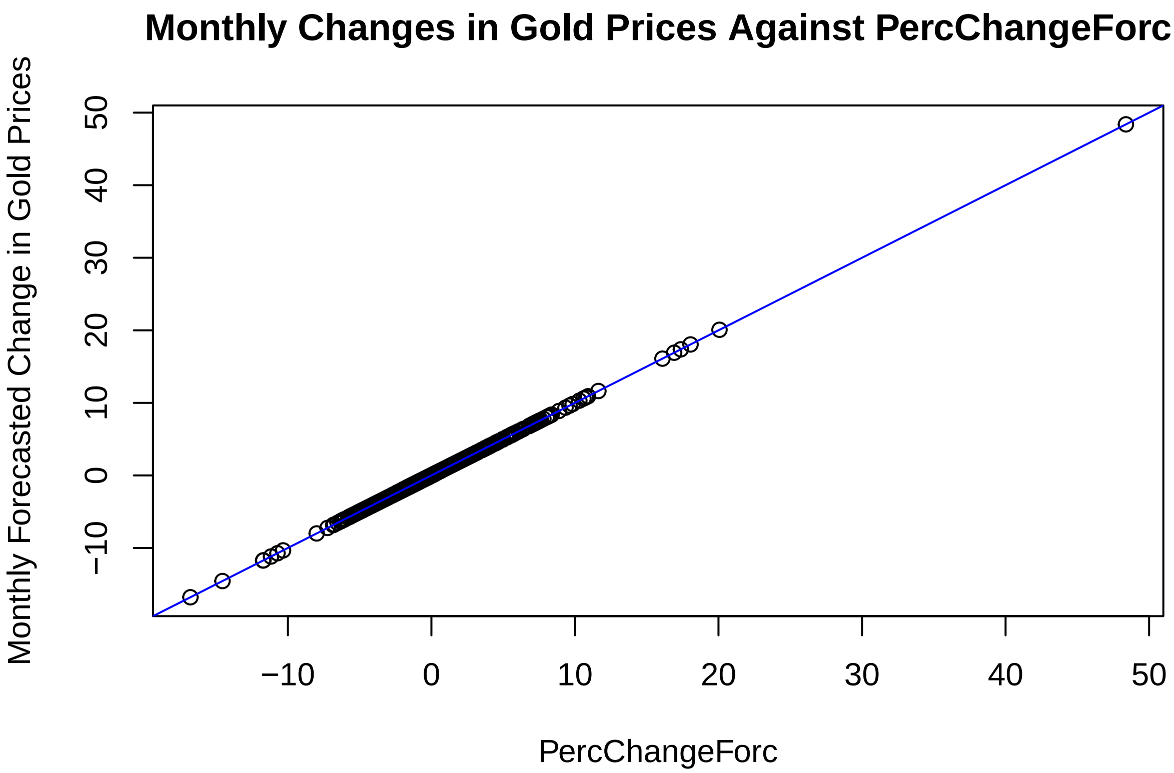
```
# library(ggplot2)
# library(gridExtra)

hist_plots <- list()

for (feature in names(gold_histograms)) {
  # Create a temporary data frame with a single column for the current feature
  temp_df <- data.frame(x = gold_histograms[[feature]])

  hist_plot <- ggplot(temp_df, aes(x = x)) +
    geom_histogram(binwidth = 2, fill = "navy", color = "lightgrey") +
    labs(x = feature, y = "Frequency") +
    ggtitle(paste0("Histogram of ", feature)) +
    theme_minimal()
  hist_plots[[feature]] <- hist_plot
}

# grid <- do.call(grid.arrange, c(hist_plots, ncol = 6))

# ggsave(file = "~/Desktop/histogram_grid.png", plot = grid, width = 31.5, height = 14.4, bg = "white")
```

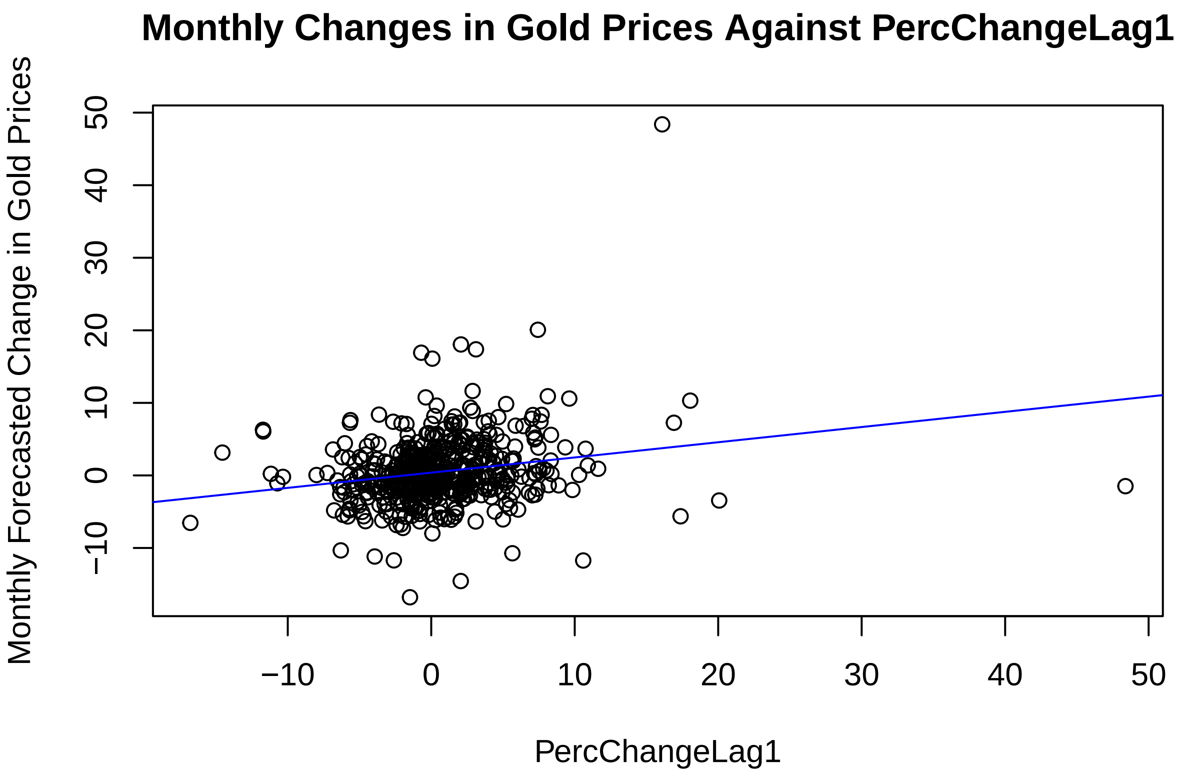
Correlation Values

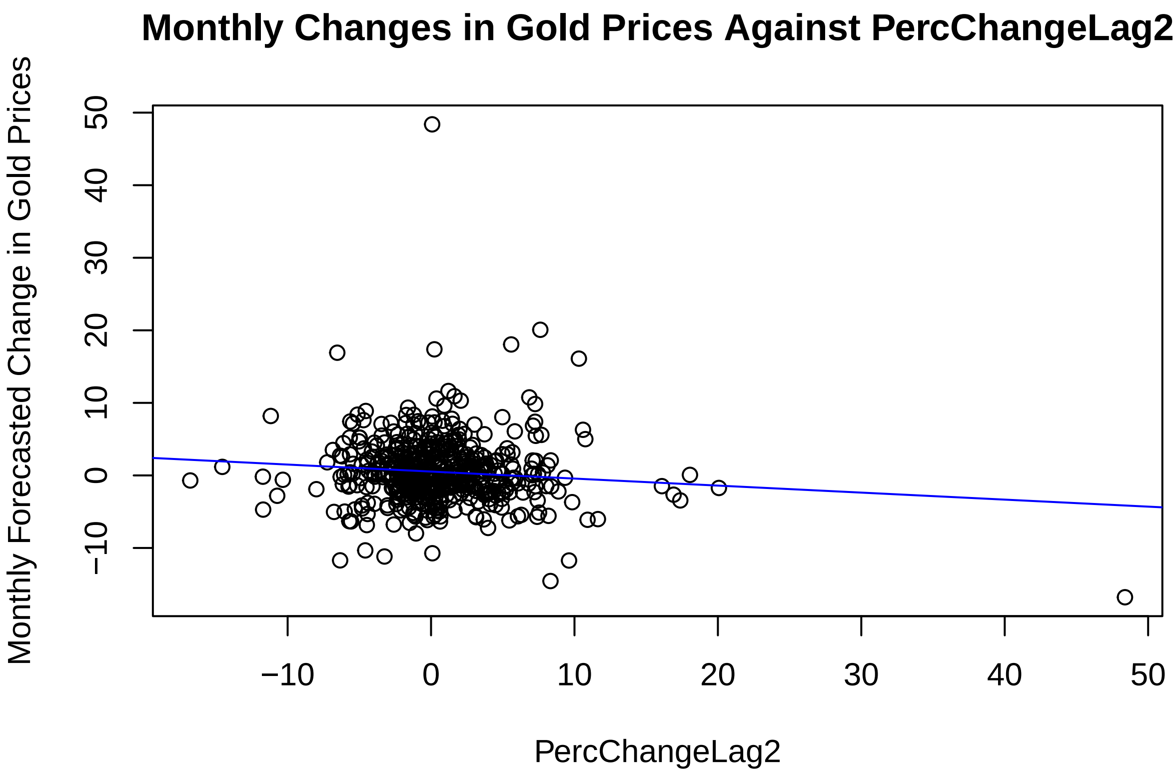
```r
# Data Clean for Scatter Plots
set.seed(490782)
gold_cor <- gold %>% select(-c("Date","Average.Price"))
gold_cor <- as.data.frame(lapply(gold_cor, as.numeric))
head(gold_cor)
```

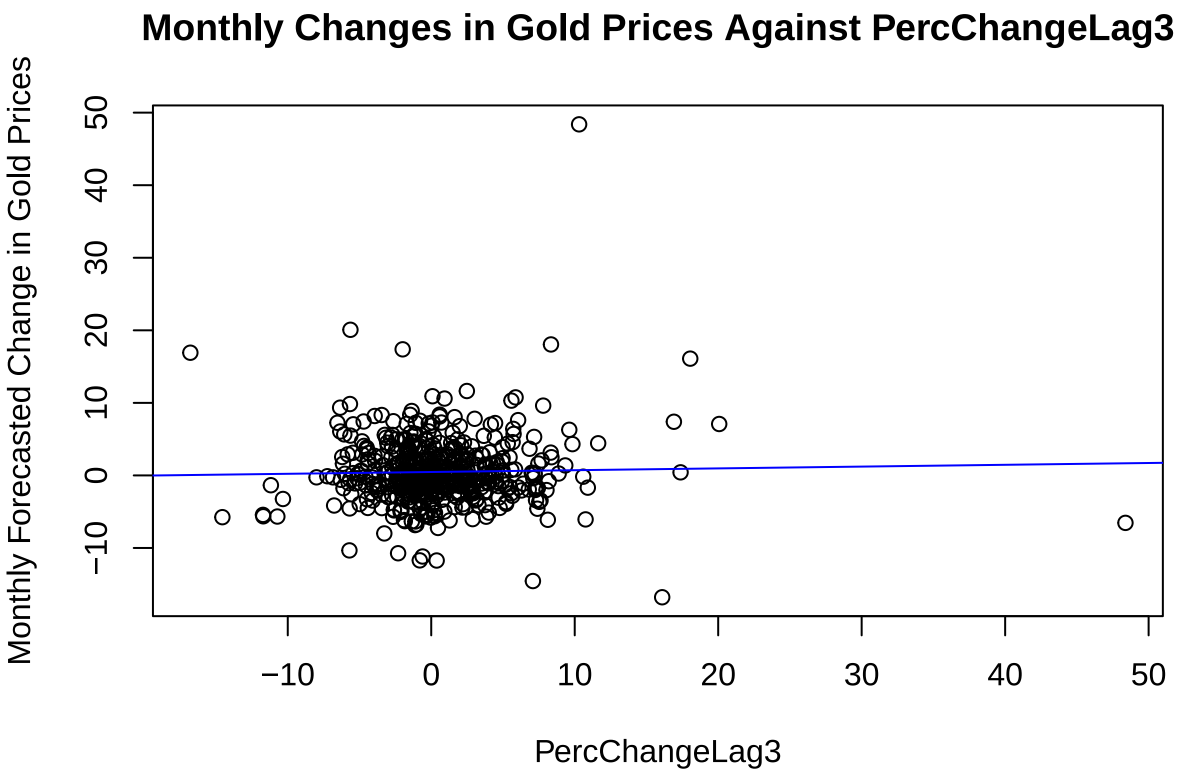
```
##   PercChangeForc PercChangeLag1 PercChangeLag2 PercChangeLag3 PercChangeLag4
## 1     8.34627329       7.692308      -1.197852      -1.465201       8.095029
## 2     5.58939448       8.346273       7.692308      -1.197852      -1.465201
## 3     2.06990160       5.589394       8.346273       7.692308      -1.197852
## 4    18.05186170       2.069902       5.589394       8.346273       7.692308
## 5    10.30695579      18.051862       2.069902       5.589394       8.346273
## 6     0.07658923      10.306956      18.051862       2.069902       5.589394
##   PercChangeLag5      X2MA      X3MA Inf.Rate.MoM  Inf.L1  Inf.L2  Inf.L3
## 1       9.384023  3.247228  1.676418      0.47022 0.88286 1.02503 0.39425
## 2       8.095029  8.019290  4.946910      1.32605 0.47022 0.88286 1.02503
## 3      -1.465201  6.967834  7.209325      1.08417 1.32605 0.47022 0.88286
## 4      -1.197852  3.829648  5.335190      0.66002 1.08417 1.32605 0.47022
## 5       7.692308 10.060882  8.570386      1.85991 0.66002 1.08417 1.32605
## 6       8.346273 14.179409 10.142906      1.13271 1.85991 0.66002 1.08417
##    Inf.L4 Res.Change.Exc.Gold UM.Infl.Exp UM.Con.Sent Indus.Prod.Ind
## 1 1.07389            -1.90551         9.8        68.1        17.0833
## 2 0.39425             9.61407         9.9        65.8        17.8613
## 3 1.02503            -9.13909         9.9        60.4        17.6767
## 4 0.88286           -11.96978         9.9        64.5        19.4906
## 5 0.47022             0.36875         9.6        66.7        20.2075
## 6 1.32605           -16.58534         9.0        62.1        18.9242
##   Nasdaq.Change.MoM   NFCI FedFundsRate FedFundsRateL1   Oil PercChange.oil
## 1          -1.57133 0.4350        10.24          10.01 10.71       3.678606
## 2           3.26679 0.7100        10.29          10.24 11.70       9.243697
## 3           2.16821 1.0300        10.47          10.29 13.39      14.444444
## 4           5.84871 1.4160        10.94          10.47 14.00       4.555639
## 5           1.71662 1.9125        11.43          10.94 14.57       4.071429
## 6          -5.81565 2.3225        13.77          11.43 15.06       3.363075
##   PPIJewelry PercChange.PPI sentiment Binary.PercChange
## 1       64.9      3.3439490 0.2570759                 1
## 2       67.8      4.4684129 0.2287966                 1
## 3       70.0      3.2448378 0.2415483                 1
## 4       70.5      0.7142857 0.2487900                 1
## 5       76.5      8.5106383 0.2780379                 1
## 6       86.2     12.6797386 0.1899655                 0
```

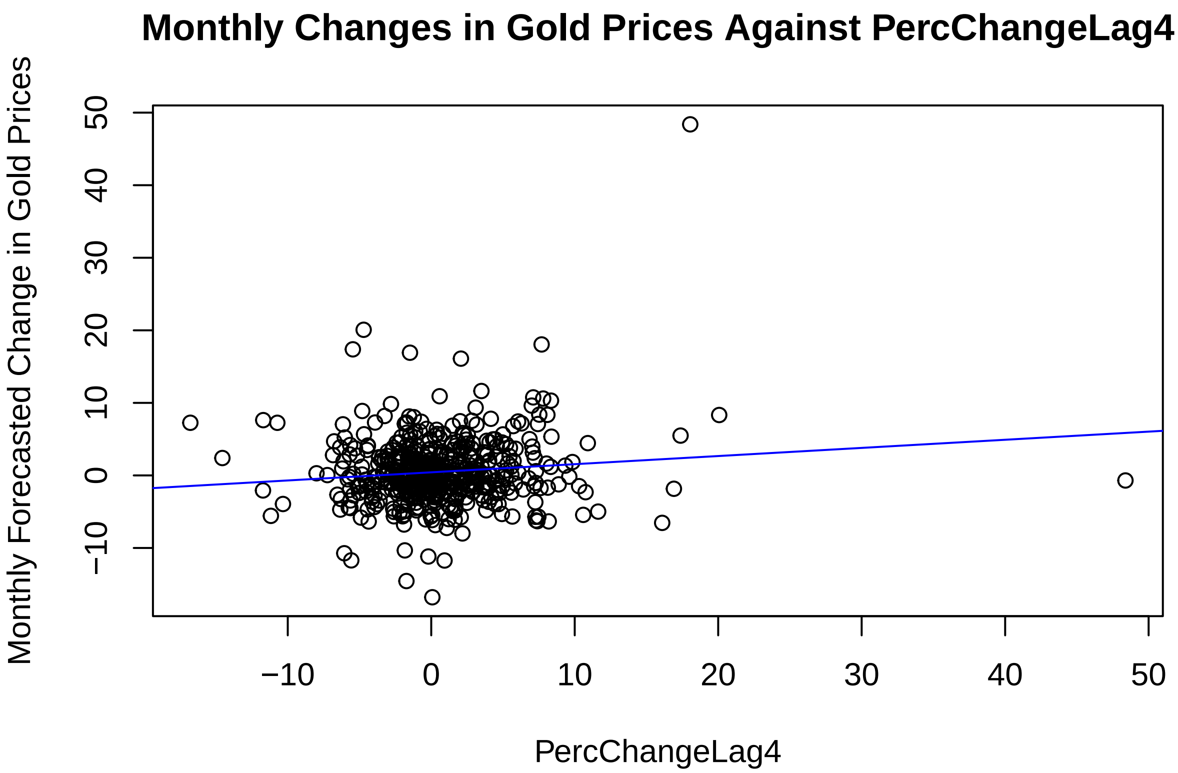
```r
# For Loop for Scatter Plots
for (feature in names(gold_cor)[!names(gold_cor) %in% c("Binary.PercChange")]){
  plot(gold_cor[[feature]], gold_cor$PercChangeForc, type = "p",
       xlab = paste(feature), ylab = "Monthly Forecasted Change in Gold Prices",
       main = paste0("Monthly Changes in Gold Prices Against ",feature))
  line <- lm(gold_cor$PercChangeForc ~ gold_cor[[feature]])
  abline(line, col = "blue")
}
```

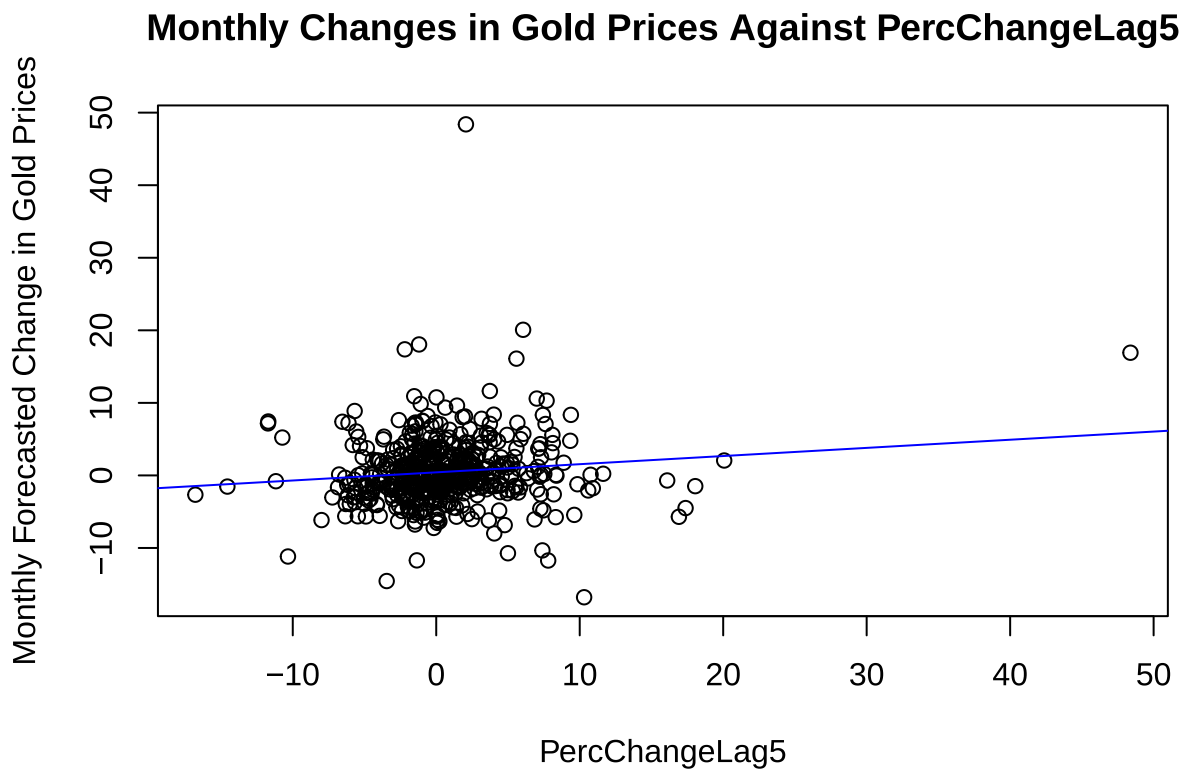

Monthly Changes in Gold Prices Against PercChangeForc
Monthly Forecasted Change in Gold Prices
PercChangeForc
−10
0
10
20
30
40
50

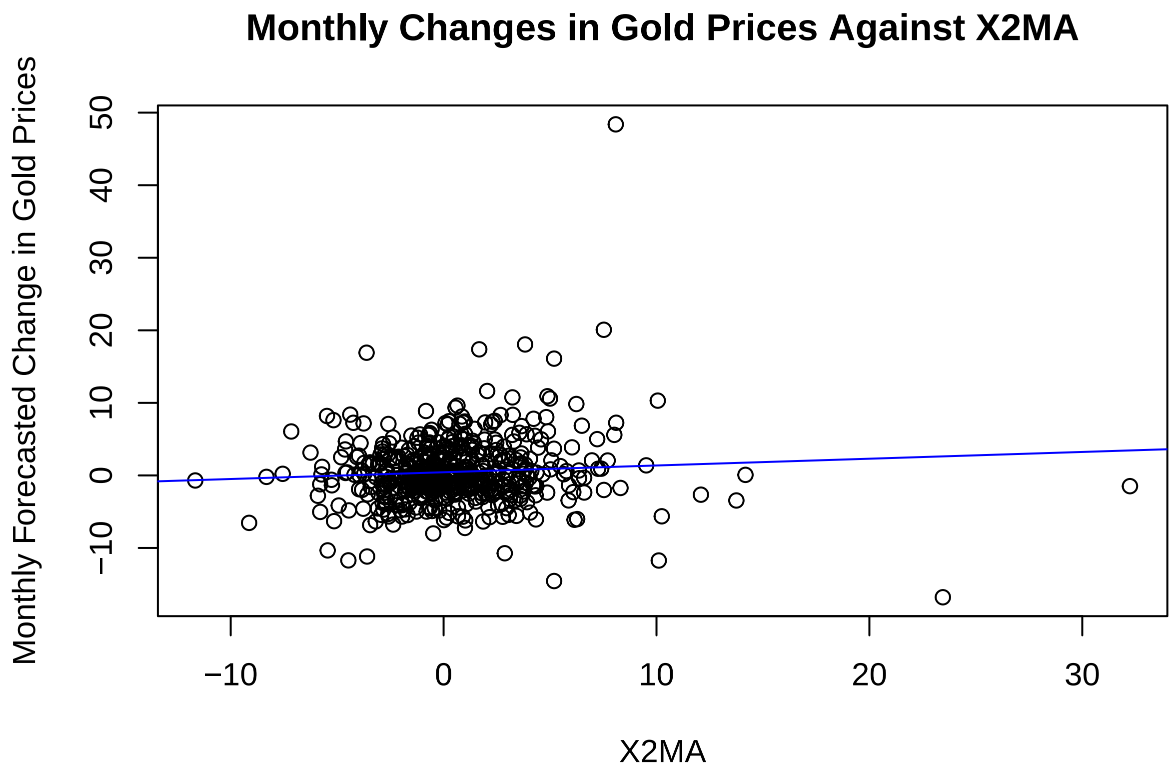

Monthly Changes in Gold Prices Against PercChangeLag1
Monthly Forecasted Change in Gold Prices
−10
0
10
20
30
40
50
−10
0
10
20
30
40
50
PercChangeLag1

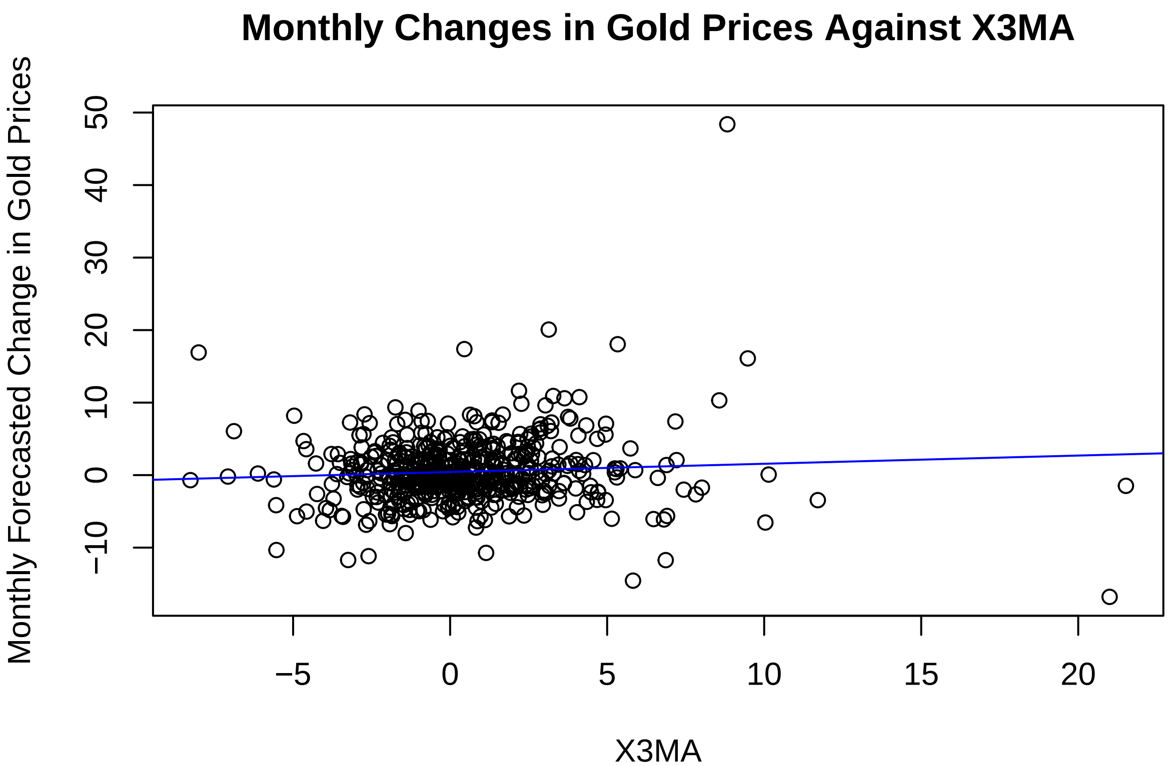

Monthly Changes in Gold Prices Against PercChangeLag2
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−10
0
10
20
30
40
50
PercChangeLag2

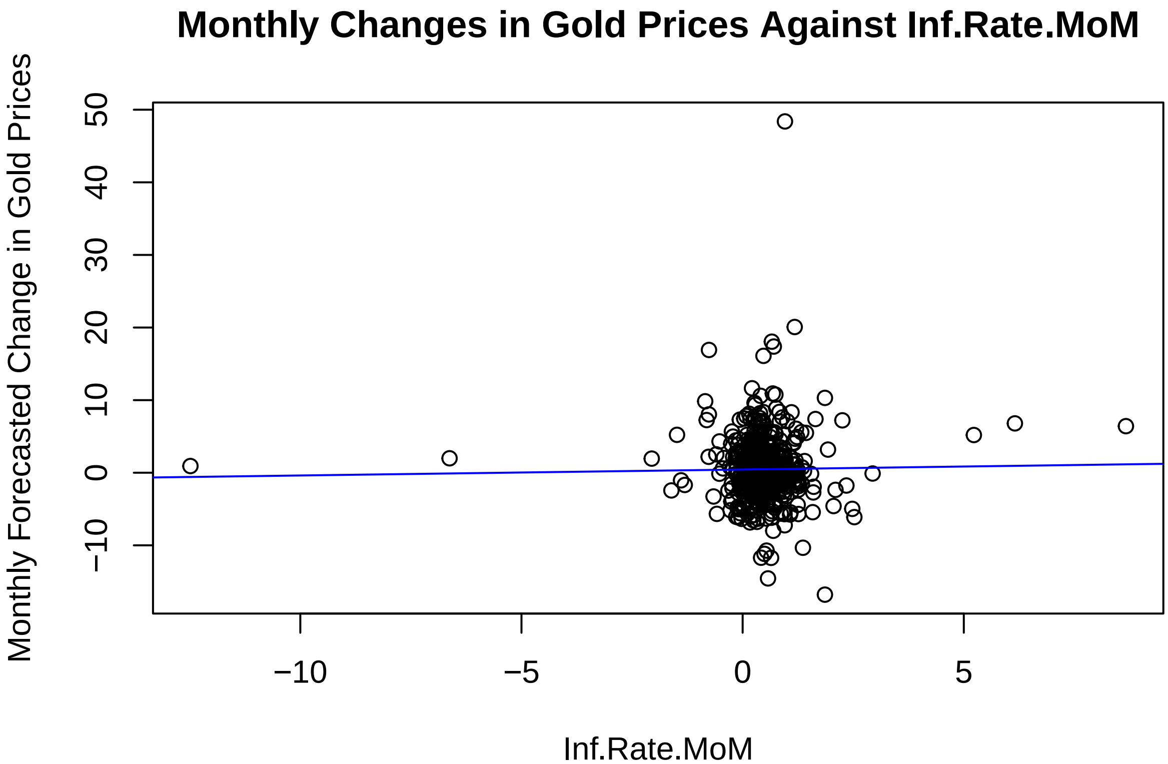

Monthly Changes in Gold Prices Against PercChangeLag3
Monthly Forecasted Change in Gold Prices
PercChangeLag3
−10
0
10
20
30
40
50
−10
0
10
20
30
40
50


Monthly Changes in Gold Prices Against PercChangeLag4
Monthly Forecasted Change in Gold Prices
−10
0
10
20
30
40
50
−10
0
10
20
30
40
50
PercChangeLag4

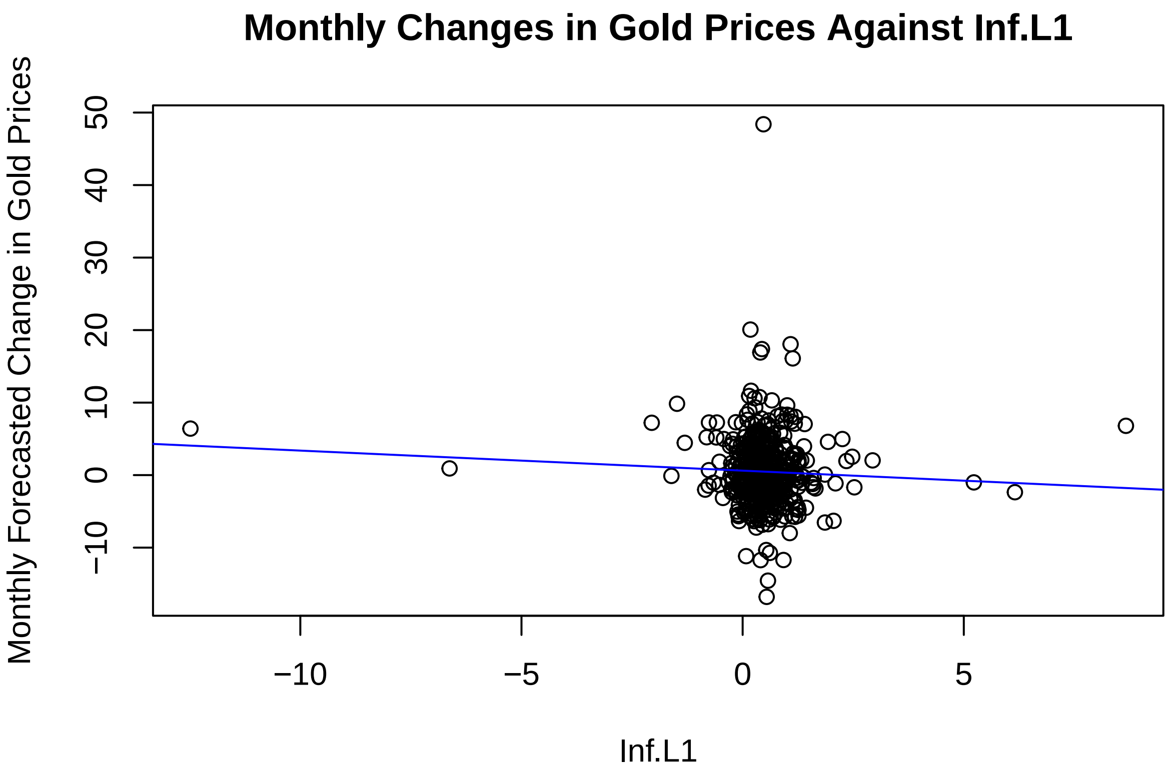

Monthly Changes in Gold Prices Against PercChangeLag5
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−10
0
10
20
30
40
50
PercChangeLag5

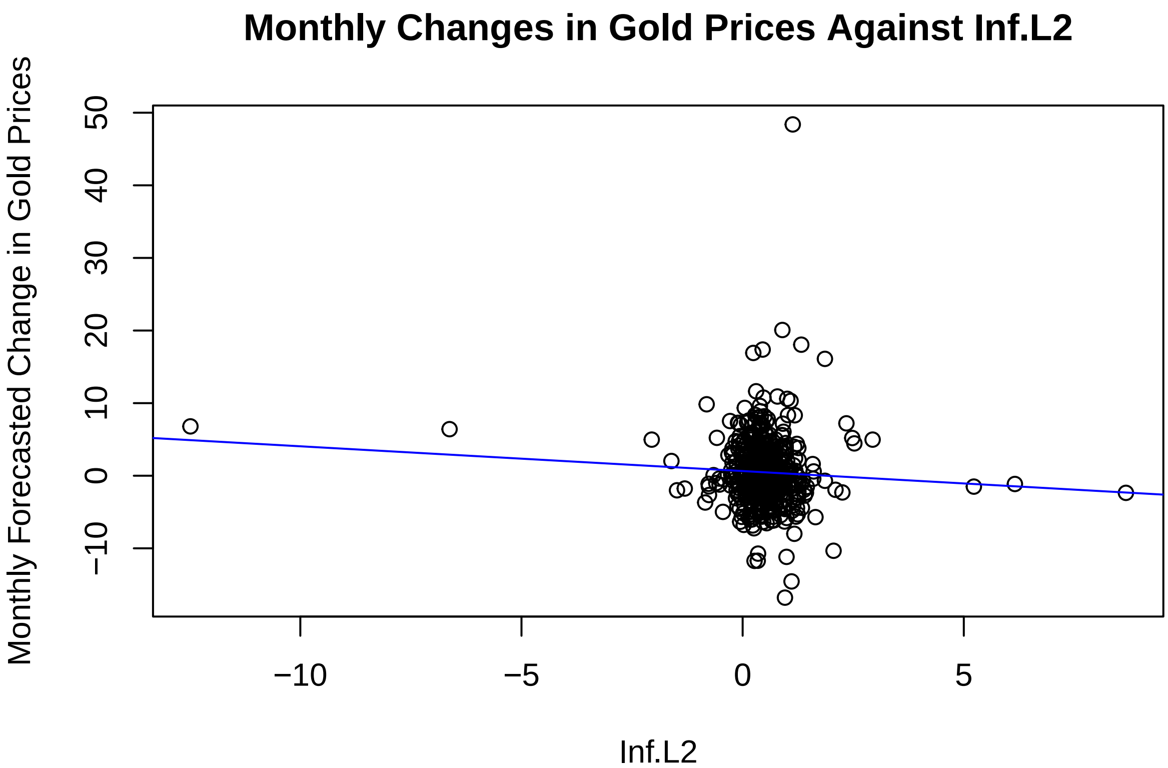

Monthly Changes in Gold Prices Against X2MA
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−10
0
10
20
30
X2MA


Monthly Changes in Gold Prices Against X3MA
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−5
0
5
10
15
20
X3MA

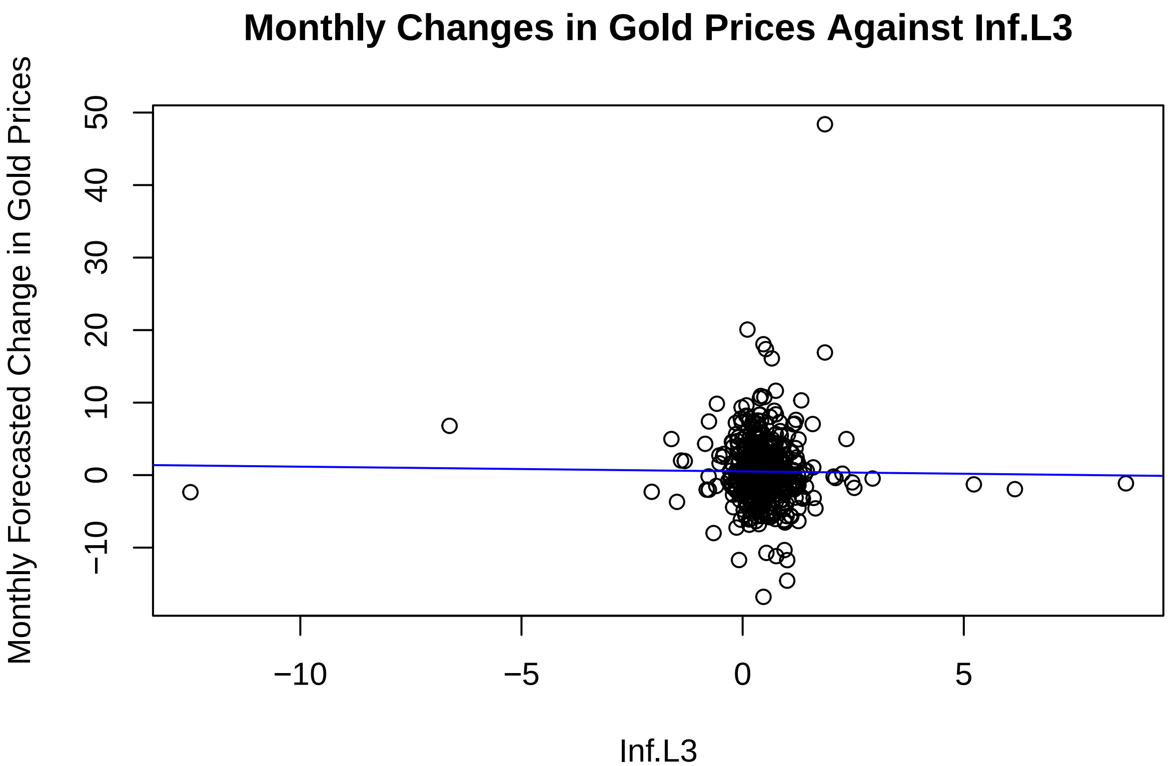


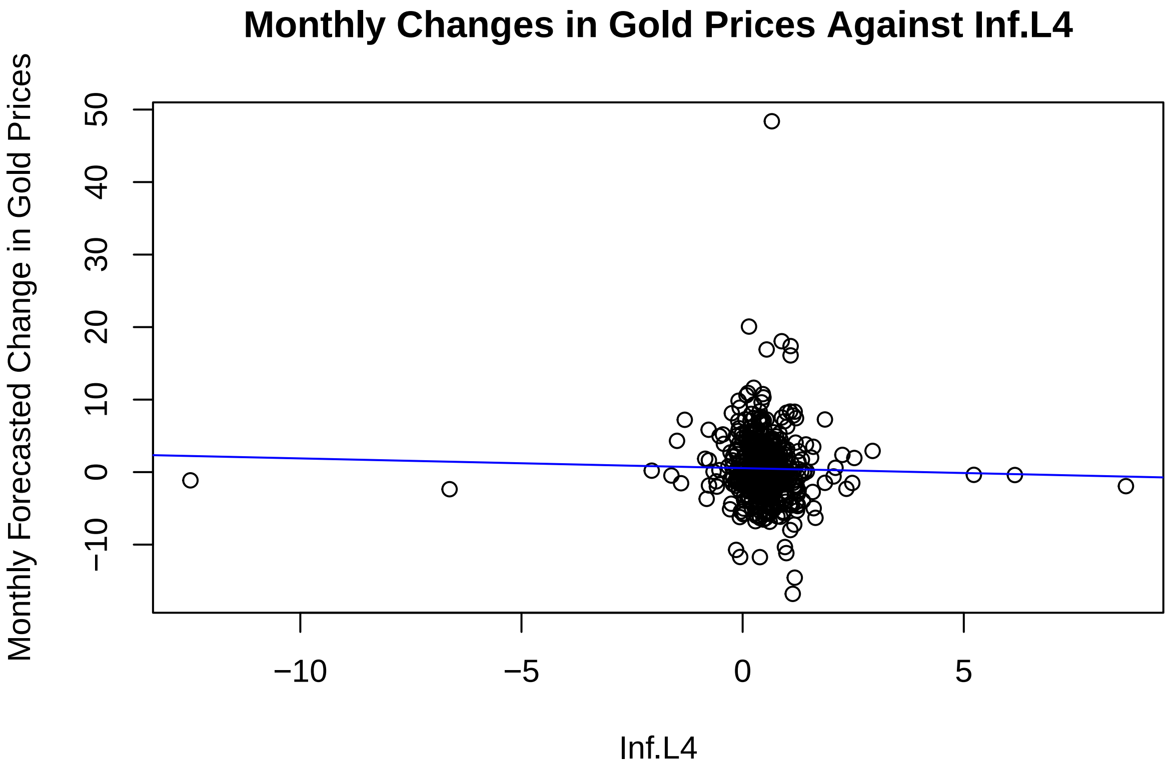
Monthly Changes in Gold Prices Against Inf.Rate.MoM
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−10
−5
0
5
Inf.Rate.MoM


Monthly Changes in Gold Prices Against Inf.L1
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−10
−5
0
5
Inf.L1

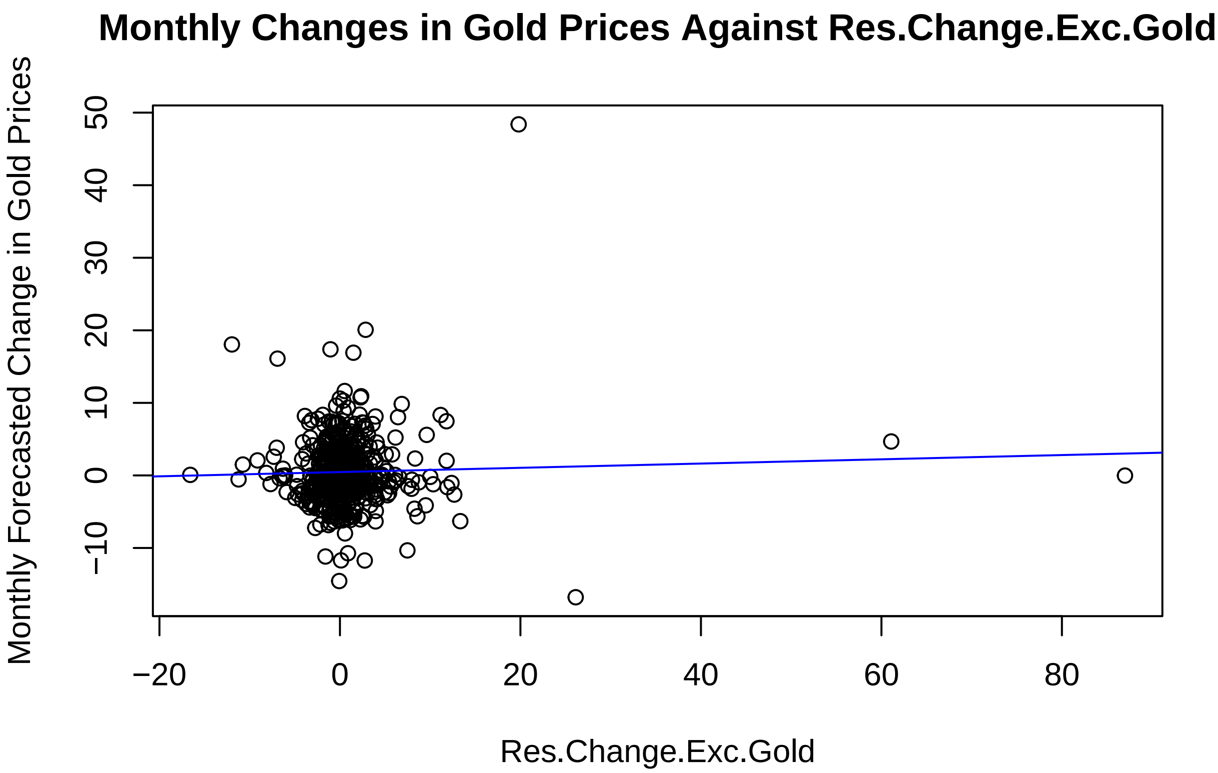


Monthly Changes in Gold Prices Against Inf.L2
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−10
−5
0
5
Inf.L2

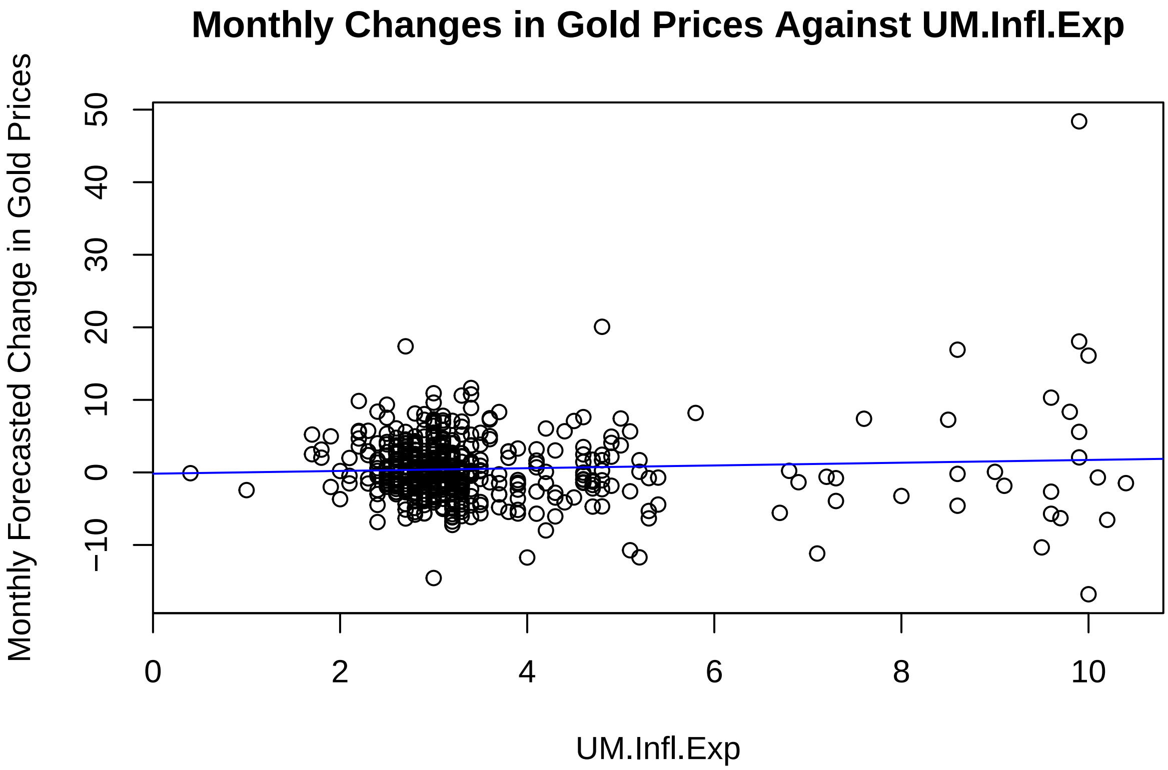

Monthly Changes in Gold Prices Against Inf.L3
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−10
−5
0
5
Inf.L3

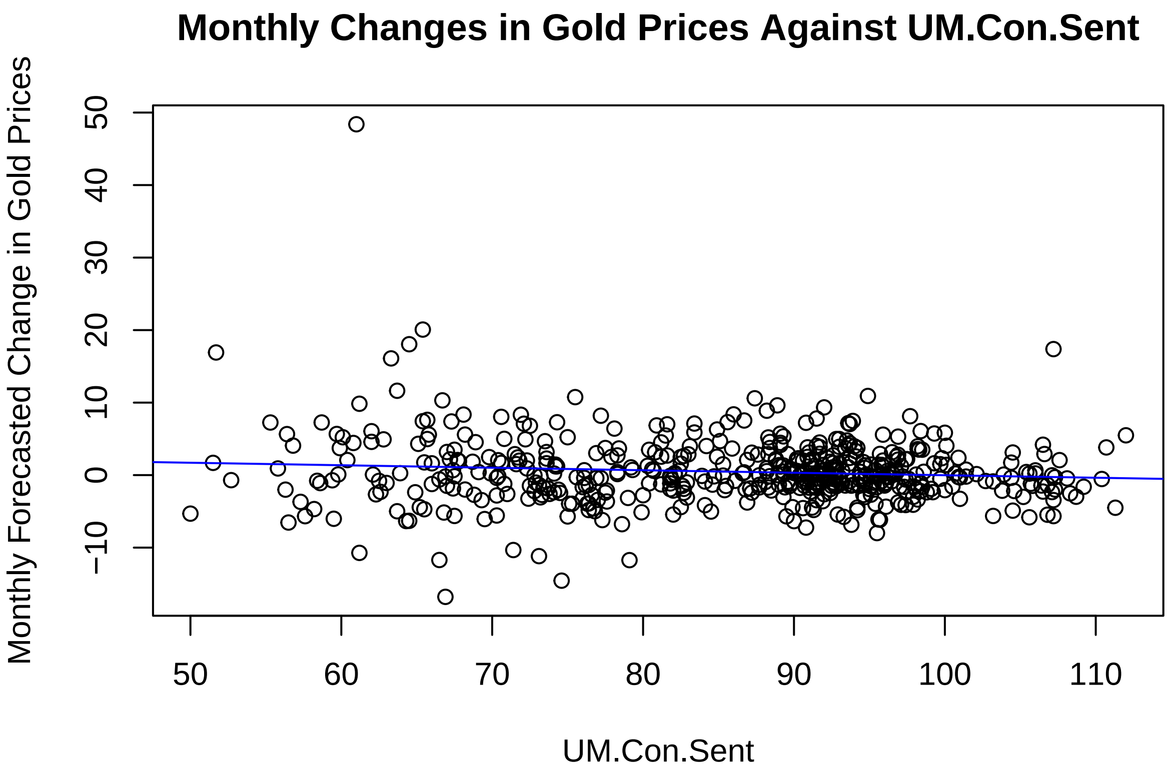


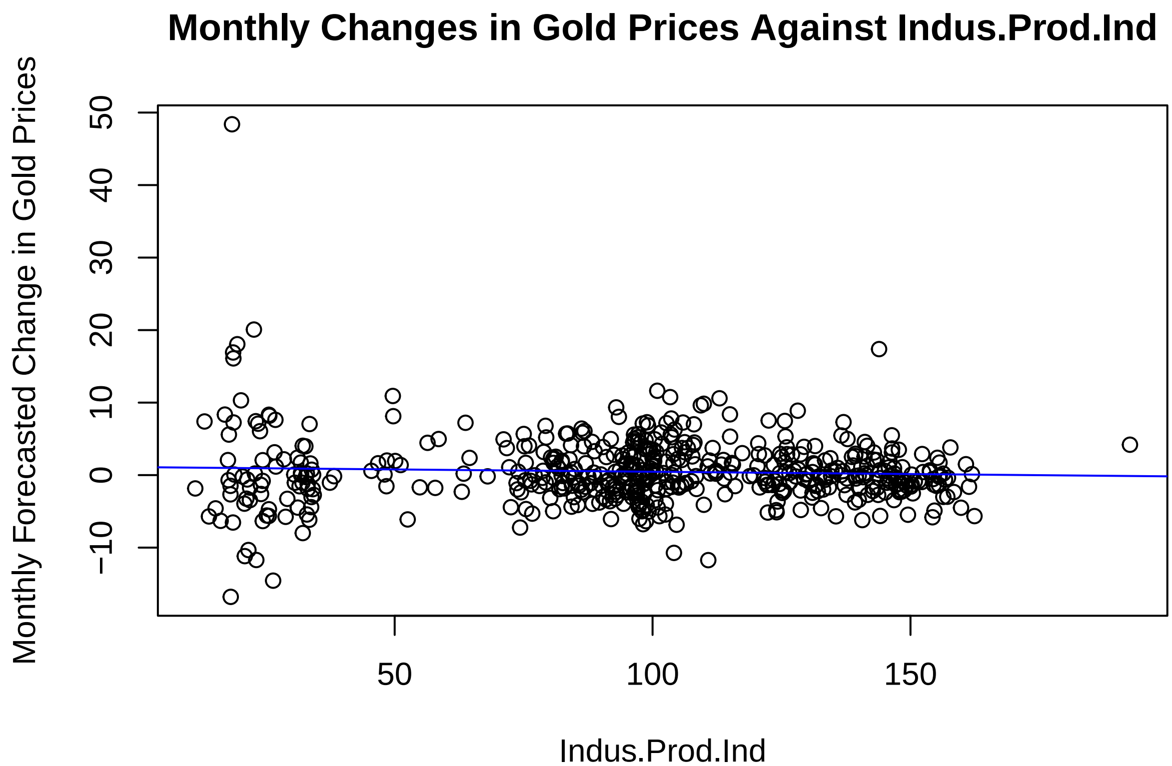
Monthly Changes in Gold Prices Against Inf.L4
Monthly Forecasted Change in Gold Prices
−10
0
10
20
30
40
50
−10
−5
0
5
Inf.L4



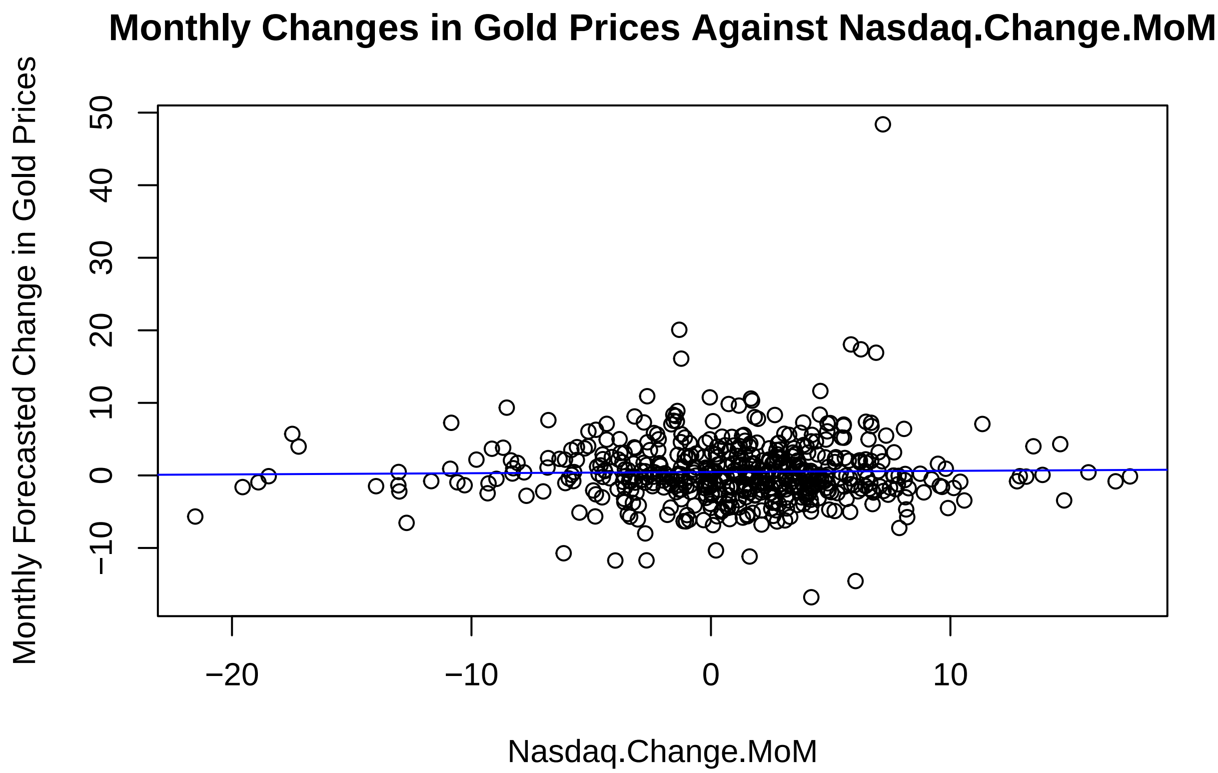
Monthly Changes in Gold Prices Against Res.Change.Exc.Gold
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−20
0
20
40
60
80
Res.Change.Exc.Gold

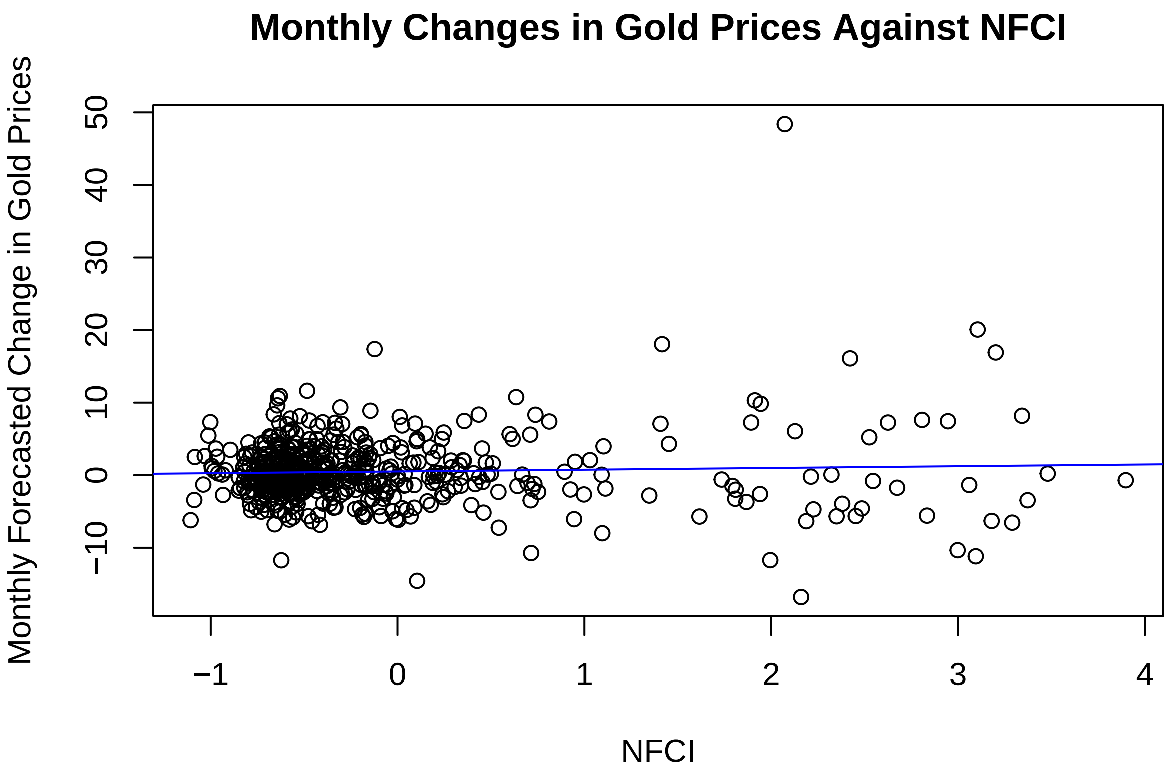

Monthly Changes in Gold Prices Against UM.Infl.Exp
Monthly Forecasted Change in Gold Prices
−10
0
10
20
30
40
50
0
2
4
6
8
10
UM.Infl.Exp


Monthly Changes in Gold Prices Against UM.Con.Sent
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
50
60
70
80
90
100
110
UM.Con.Sent

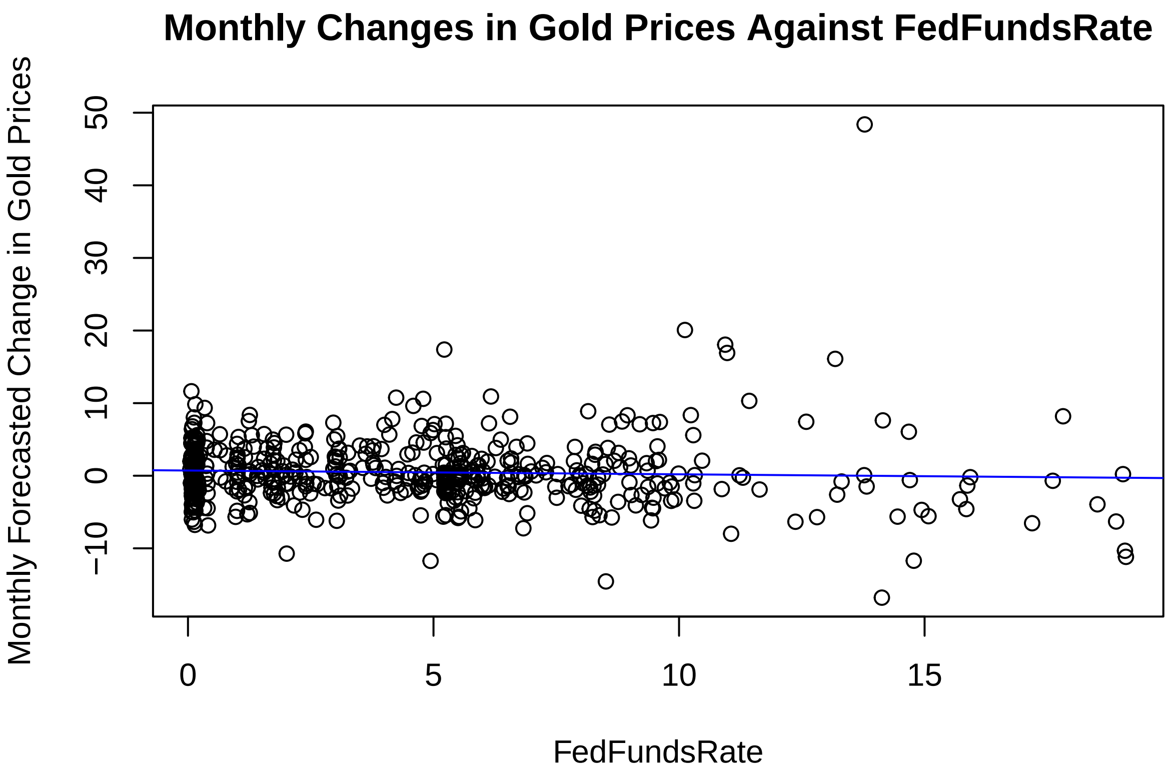


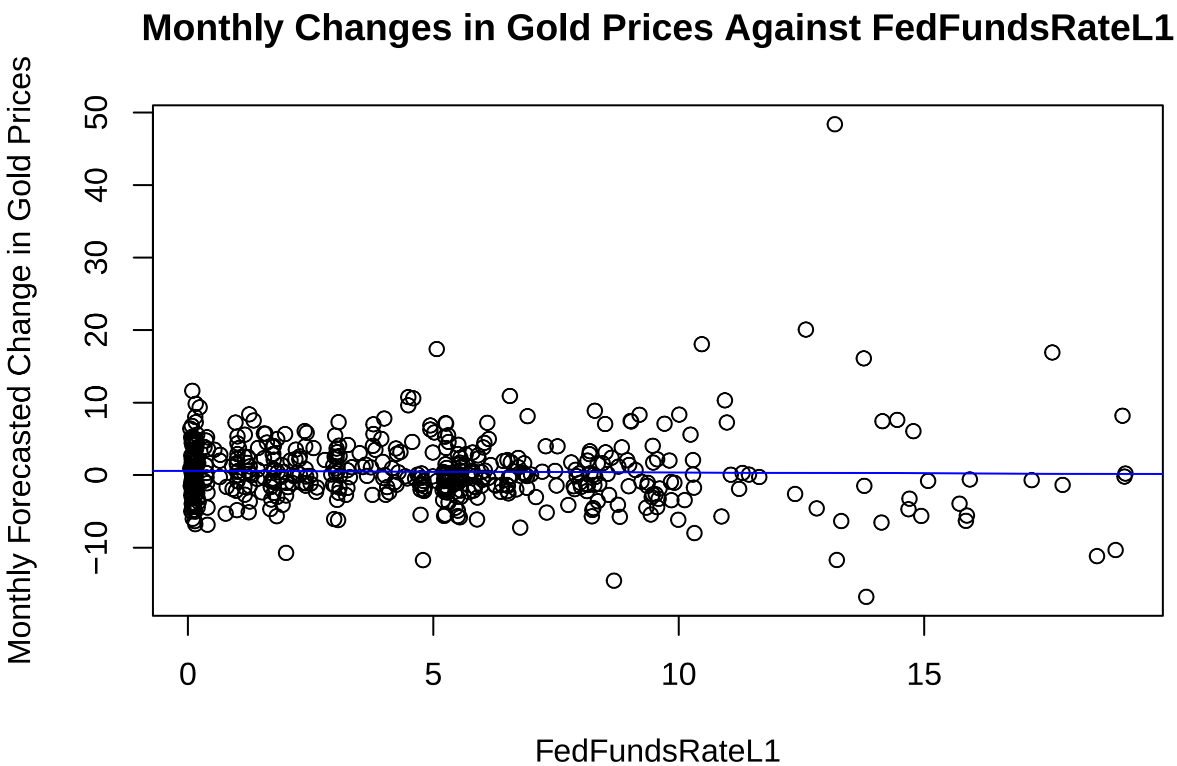
Monthly Changes in Gold Prices Against Indus.Prod.Ind
Monthly Forecasted Change in Gold Prices
−10
0
10
20
30
40
50
50
100
150
Indus.Prod.Ind

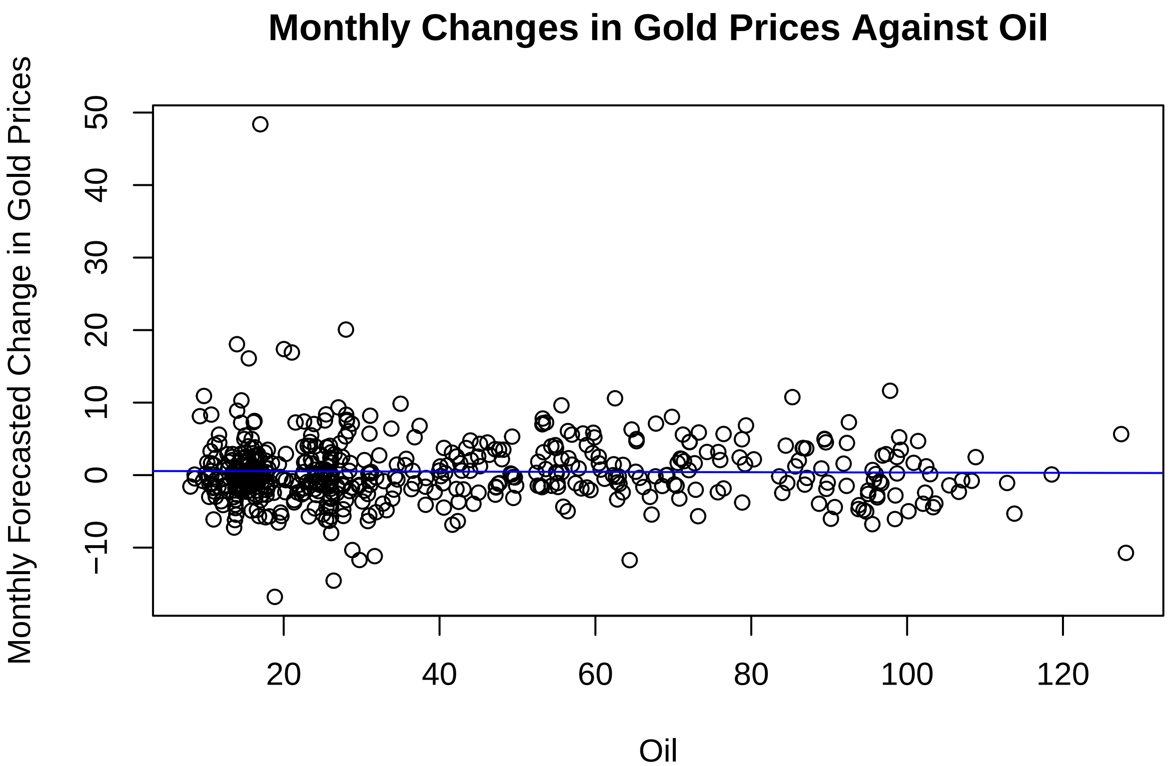

Monthly Changes in Gold Prices Against Nasdaq.Change.MoM
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−20
−10
0
10
Nasdaq.Change.MoM

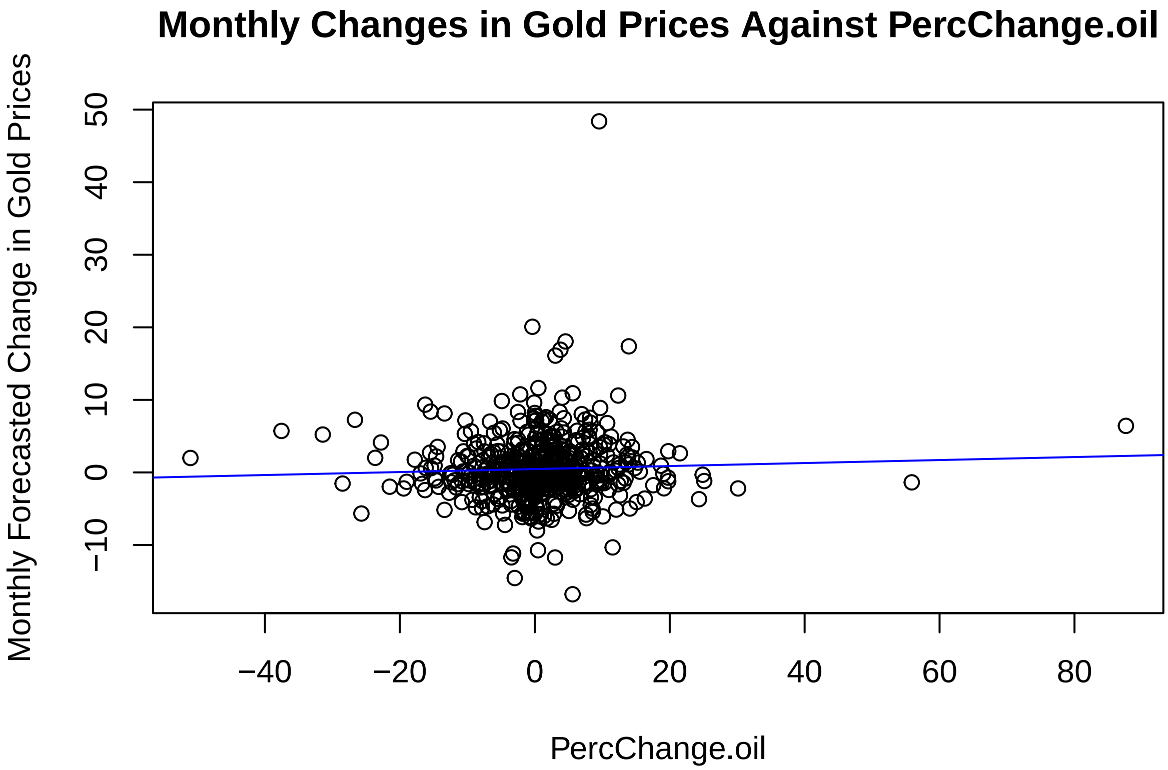


Monthly Changes in Gold Prices Against NFCI
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−1
0
1
2
3
4
NFCI

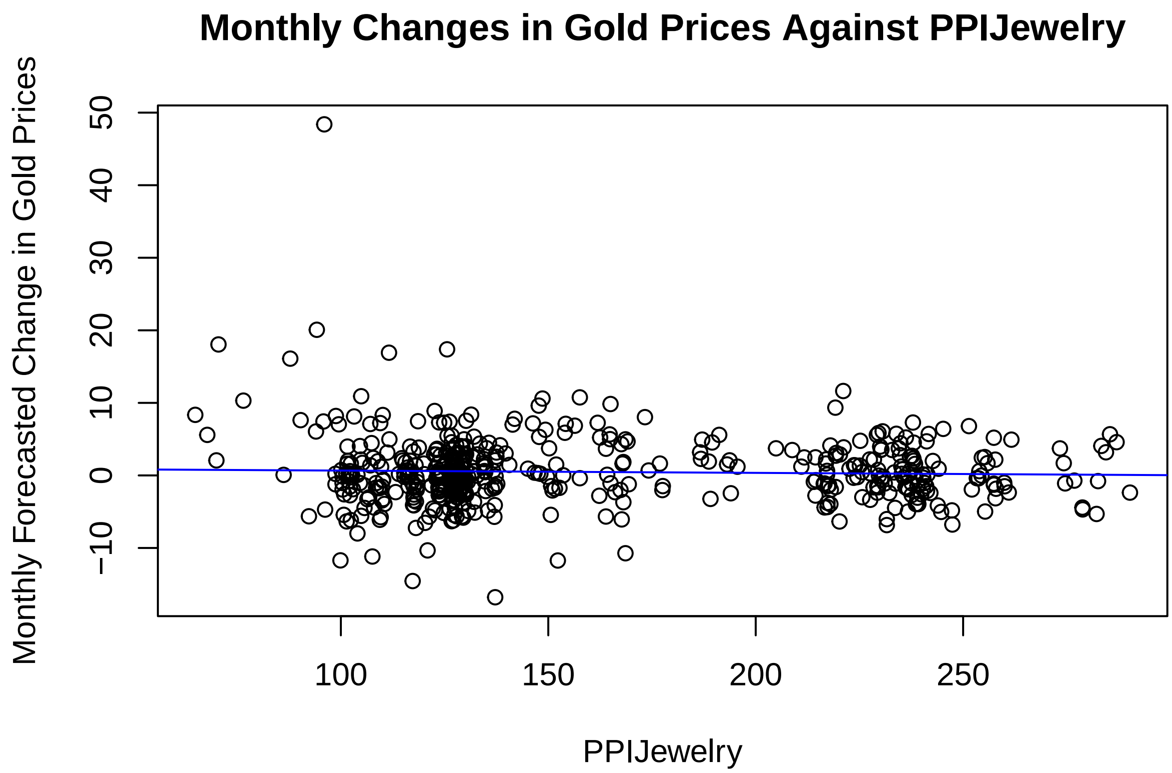

Monthly Changes in Gold Prices Against FedFundsRate
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
0
5
10
15
FedFundsRate


Monthly Changes in Gold Prices Against FedFundsRateL1
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
0
5
10
15
FedFundsRateL1

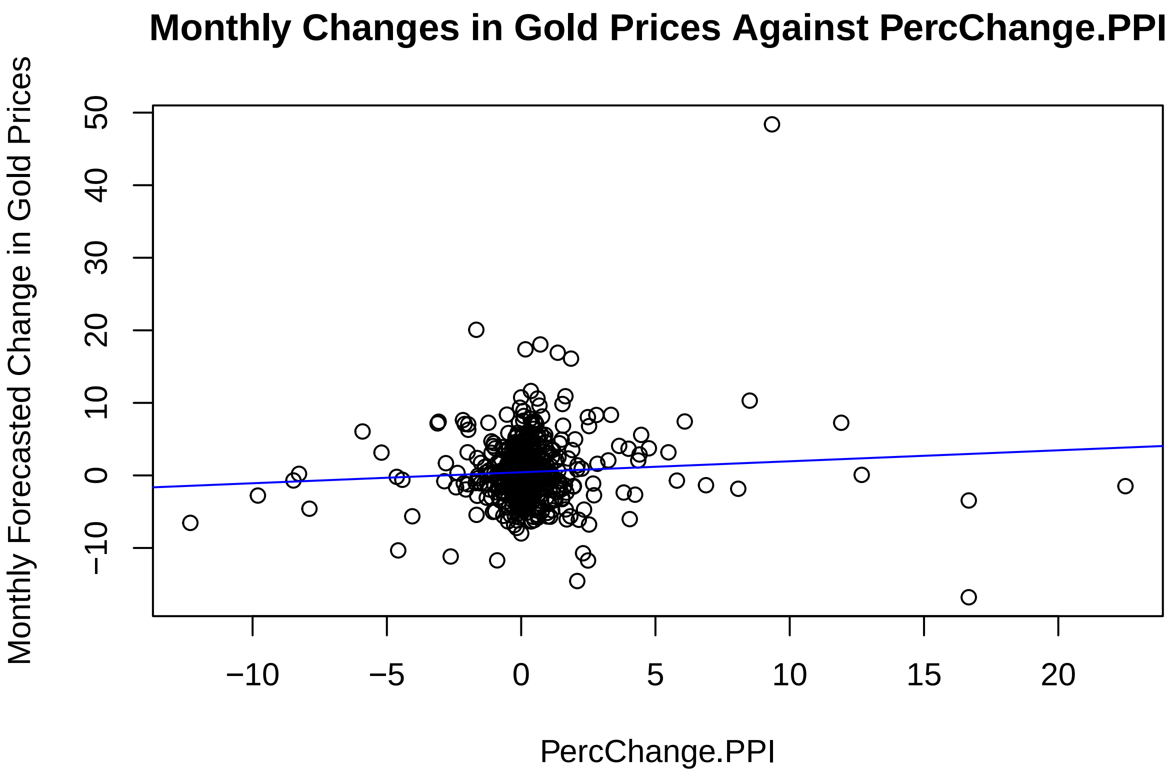

Monthly Changes in Gold Prices Against Oil
Monthly Forecasted Change in Gold Prices
Oil
50
40
30
20
10
0
−10
20
40
60
80
100
120



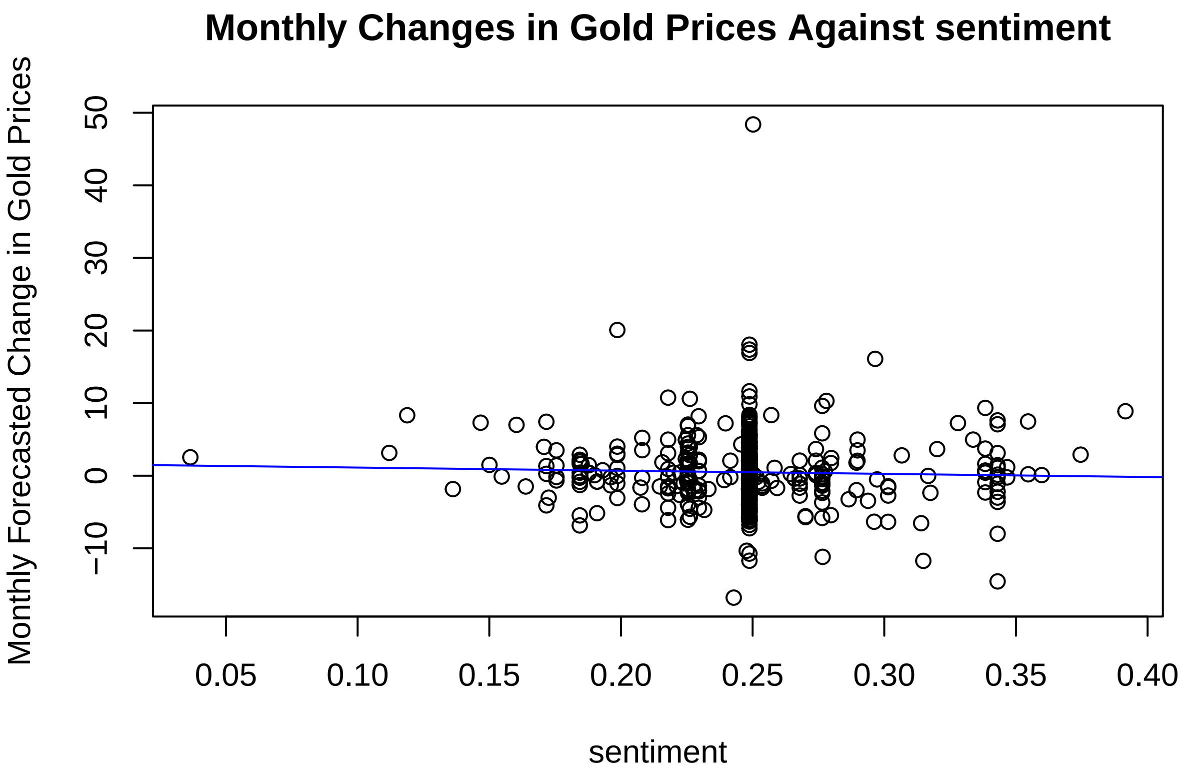
Monthly Changes in Gold Prices Against PercChange.oil
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
−40
−20
0
20
40
60
80
PercChange.oil



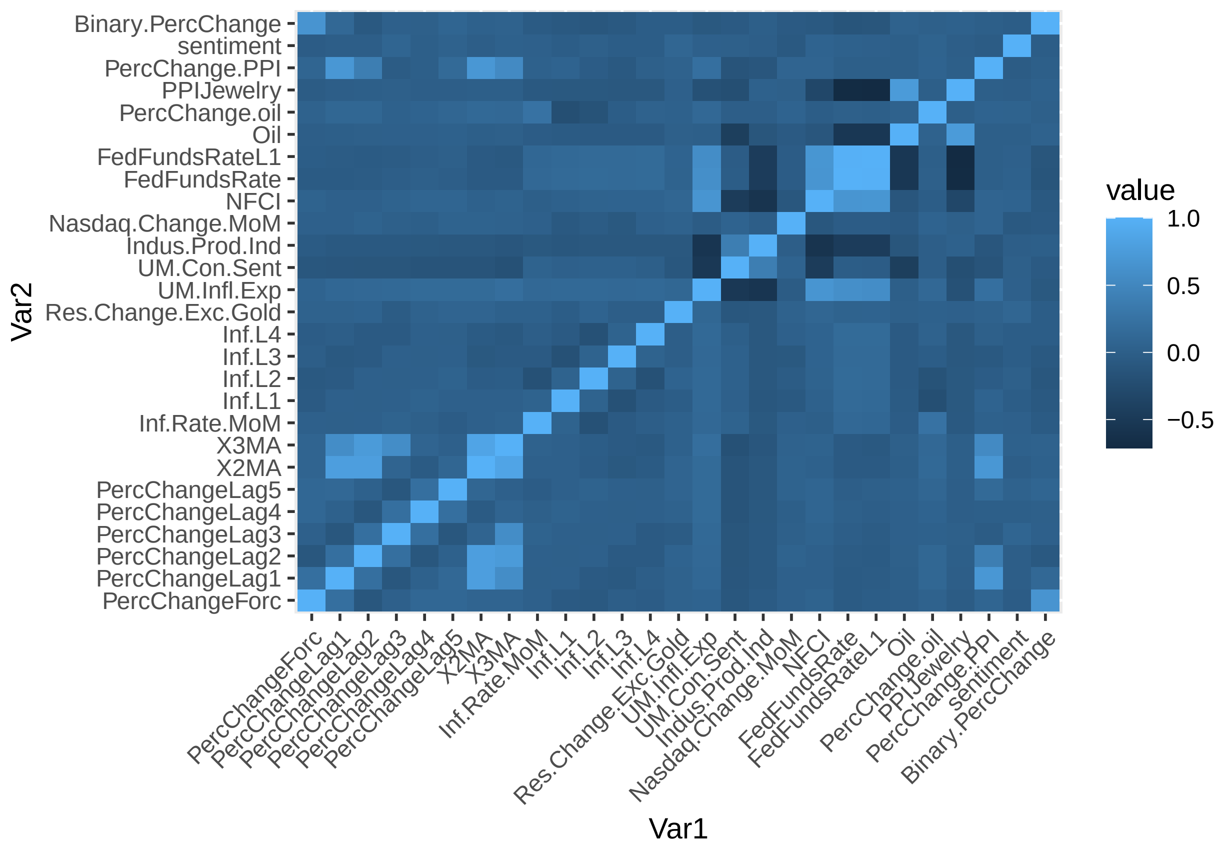
Monthly Changes in Gold Prices Against PPIJewelry
Monthly Forecasted Change in Gold Prices
50
40
30
20
10
0
−10
100
150
200
250
PPIJewelry



Monthly Changes in Gold Prices Against PercChange.PPI
Monthly Forecasted Change in Gold Prices
PercChange.PPI
−10
−5
0
5
10
15
20
−10
0
10
20
30
40
50




```r
# Heat Map
cor_matrix <- round(cor(gold_cor),2)
melted_cor_gold <- melt(cor_matrix)
ggplot(data = melted_cor_gold, aes(x = Var1, y = Var2, fill = value))+
  geom_tile()+theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Our scatterplots and heat maps show that there are some pockets of correlation between features in our data set. Looking at our main response variables (PercChangeForc & Binary.PercChange), there are no visibly strong correlations between them and our various features. However, we believe a combination and enhancement of these features through the various algorithms we'll be looking at will allow us to best predict both response variables - in a regression and classification setting.

Initial Data Cleaning

```r
# Gold Data for Regression Problem:
head(gold)
```

```
##          Date Average.Price PercChangeForc PercChangeLag1 PercChangeLag2
## 6  31/05/1979         257.6     8.34627329       7.692308      -1.197852
## 7  29/06/1979         279.1     5.58939448       8.346273       7.692308
## 8  31/07/1979         294.7     2.06990160       5.589394       8.346273
## 9  31/08/1979         300.8    18.05186170       2.069902       5.589394
## 10 28/09/1979         355.1    10.30695579      18.051862       2.069902
## 11 31/10/1979         391.7     0.07658923      10.306956      18.051862
##    PercChangeLag3 PercChangeLag4 PercChangeLag5     X2MA     X3MA
## 6       -1.465201       8.095029       9.384023 3.247228 1.676418
## 7       -1.197852      -1.465201       8.095029 8.019290 4.946910
## 8        7.692308      -1.197852      -1.465201 6.967834 7.209325
## 9        8.346273       7.692308      -1.197852 3.829648 5.335190
```

```
## 10       5.589394       8.346273       7.692308 10.060882  8.570386
## 11       2.069902       5.589394       8.346273 14.179409 10.142906
##    Inf.Rate.MoM  Inf.L1  Inf.L2  Inf.L3  Inf.L4 Res.Change.Exc.Gold UM.Infl.Exp
## 6       0.47022 0.88286 1.02503 0.39425 1.07389            -1.90551         9.8
## 7       1.32605 0.47022 0.88286 1.02503 0.39425             9.61407         9.9
## 8       1.08417 1.32605 0.47022 0.88286 1.02503            -9.13909         9.9
## 9       0.66002 1.08417 1.32605 0.47022 0.88286           -11.96978         9.9
## 10      1.85991 0.66002 1.08417 1.32605 0.47022             0.36875         9.6
## 11      1.13271 1.85991 0.66002 1.08417 1.32605           -16.58534         9.0
##    UM.Con.Sent Indus.Prod.Ind Nasdaq.Change.MoM   NFCI FedFundsRate
## 6         68.1        17.0833          -1.57133 0.4350        10.24
## 7         65.8        17.8613           3.26679 0.7100        10.29
## 8         60.4        17.6767           2.16821 1.0300        10.47
## 9         64.5        19.4906           5.84871 1.4160        10.94
## 10        66.7        20.2075           1.71662 1.9125        11.43
## 11        62.1        18.9242          -5.81565 2.3225        13.77
##    FedFundsRateL1   Oil PercChange.oil PPIJewelry PercChange.PPI sentiment
## 6           10.01 10.71       3.678606       64.9      3.3439490 0.2570759
## 7           10.24 11.70       9.243697       67.8      4.4684129 0.2287966
## 8           10.29 13.39      14.444444       70.0      3.2448378 0.2415483
## 9           10.47 14.00       4.555639       70.5      0.7142857 0.2487900
## 10          10.94 14.57       4.071429       76.5      8.5106383 0.2780379
## 11          11.43 15.06       3.363075       86.2     12.6797386 0.1899655
##    Binary.PercChange
## 6                  1
## 7                  1
## 8                  1
## 9                  1
## 10                 1
## 11                 0
```

```r
gold.r <- gold %>% select(-c("Date","Average.Price","Binary.PercChange","Oil",
                             "PPIJewelry"))

# Gold Data for Classification Problem:
gold.c <- gold %>% select(-c("Date","Average.Price","PercChangeForc","Oil",
                             "PPIJewelry"))

# Convert Data to Numeric
gold.r <- as.data.frame(lapply(gold.r, as.numeric))
gold.c <- as.data.frame(lapply(gold.c, as.numeric))

# Data Check & Numeric Check
all(sapply(gold.r, is.numeric))
```

```
## [1] TRUE
```

```r
all(sapply(gold.c, is.numeric))
```

```
## [1] TRUE
```

```
head(gold.r)
```

```
##   PercChangeForc PercChangeLag1 PercChangeLag2 PercChangeLag3 PercChangeLag4
## 1     8.34627329       7.692308      -1.197852      -1.465201       8.095029
## 2     5.58939448       8.346273       7.692308      -1.197852      -1.465201
## 3     2.06990160       5.589394       8.346273       7.692308      -1.197852
## 4    18.05186170       2.069902       5.589394       8.346273       7.692308
## 5    10.30695579      18.051862       2.069902       5.589394       8.346273
## 6     0.07658923      10.306956      18.051862       2.069902       5.589394
##   PercChangeLag5      X2MA      X3MA Inf.Rate.MoM  Inf.L1  Inf.L2  Inf.L3
## 1       9.384023  3.247228  1.676418      0.47022 0.88286 1.02503 0.39425
## 2       8.095029  8.019290  4.946910      1.32605 0.47022 0.88286 1.02503
## 3      -1.465201  6.967834  7.209325      1.08417 1.32605 0.47022 0.88286
## 4      -1.197852  3.829648  5.335190      0.66002 1.08417 1.32605 0.47022
## 5       7.692308 10.060882  8.570386      1.85991 0.66002 1.08417 1.32605
## 6       8.346273 14.179409 10.142906      1.13271 1.85991 0.66002 1.08417
##    Inf.L4 Res.Change.Exc.Gold UM.Infl.Exp UM.Con.Sent Indus.Prod.Ind
## 1 1.07389            -1.90551         9.8        68.1        17.0833
## 2 0.39425             9.61407         9.9        65.8        17.8613
## 3 1.02503            -9.13909         9.9        60.4        17.6767
## 4 0.88286           -11.96978         9.9        64.5        19.4906
## 5 0.47022             0.36875         9.6        66.7        20.2075
## 6 1.32605           -16.58534         9.0        62.1        18.9242
##   Nasdaq.Change.MoM   NFCI FedFundsRate FedFundsRateL1 PercChange.oil
## 1          -1.57133 0.4350        10.24          10.01       3.678606
## 2           3.26679 0.7100        10.29          10.24       9.243697
## 3           2.16821 1.0300        10.47          10.29      14.444444
## 4           5.84871 1.4160        10.94          10.47       4.555639
## 5           1.71662 1.9125        11.43          10.94       4.071429
## 6          -5.81565 2.3225        13.77          11.43       3.363075
##   PercChange.PPI sentiment
## 1      3.3439490 0.2570759
## 2      4.4684129 0.2287966
## 3      3.2448378 0.2415483
## 4      0.7142857 0.2487900
## 5      8.5106383 0.2780379
## 6     12.6797386 0.1899655
```

```
head(gold.c)
```

```
##   PercChangeLag1 PercChangeLag2 PercChangeLag3 PercChangeLag4 PercChangeLag5
## 1       7.692308      -1.197852      -1.465201       8.095029       9.384023
## 2       8.346273       7.692308      -1.197852      -1.465201       8.095029
## 3       5.589394       8.346273       7.692308      -1.197852      -1.465201
## 4       2.069902       5.589394       8.346273       7.692308      -1.197852
## 5      18.051862       2.069902       5.589394       8.346273       7.692308
## 6      10.306956      18.051862       2.069902       5.589394       8.346273
##        X2MA      X3MA Inf.Rate.MoM  Inf.L1  Inf.L2  Inf.L3  Inf.L4
## 1  3.247228  1.676418      0.47022 0.88286 1.02503 0.39425 1.07389
## 2  8.019290  4.946910      1.32605 0.47022 0.88286 1.02503 0.39425
## 3  6.967834  7.209325      1.08417 1.32605 0.47022 0.88286 1.02503
## 4  3.829648  5.335190      0.66002 1.08417 1.32605 0.47022 0.88286
```

```
## 5 10.060882  8.570386      1.85991 0.66002 1.08417 1.32605 0.47022
## 6 14.179409 10.142906      1.13271 1.85991 0.66002 1.08417 1.32605
##   Res.Change.Exc.Gold UM.Infl.Exp UM.Con.Sent Indus.Prod.Ind Nasdaq.Change.MoM
## 1            -1.90551         9.8        68.1        17.0833          -1.57133
## 2             9.61407         9.9        65.8        17.8613           3.26679
## 3            -9.13909         9.9        60.4        17.6767           2.16821
## 4           -11.96978         9.9        64.5        19.4906           5.84871
## 5             0.36875         9.6        66.7        20.2075           1.71662
## 6           -16.58534         9.0        62.1        18.9242          -5.81565
##     NFCI FedFundsRate FedFundsRateL1 PercChange.oil PercChange.PPI sentiment
## 1 0.4350        10.24          10.01       3.678606      3.3439490 0.2570759
## 2 0.7100        10.29          10.24       9.243697      4.4684129 0.2287966
## 3 1.0300        10.47          10.29      14.444444      3.2448378 0.2415483
## 4 1.4160        10.94          10.47       4.555639      0.7142857 0.2487900
## 5 1.9125        11.43          10.94       4.071429      8.5106383 0.2780379
## 6 2.3225        13.77          11.43       3.363075     12.6797386 0.1899655
##   Binary.PercChange
## 1                 1
## 2                 1
## 3                 1
## 4                 1
## 5                 1
## 6                 0
```

Feature Selection: Boruta Algorithm

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine

##  1. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  2. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest
```

```
##  3. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  4. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  5. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  6. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  7. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  8. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  9. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  10. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  11. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  12. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest
```

```
## After 12 iterations, +2.8 secs:

##  confirmed 1 attribute: PercChangeLag1;

##  rejected 8 attributes: Inf.L2, Inf.Rate.MoM, Nasdaq.Change.MoM, PercChange.oil, PercChangeLag3 and :

##  still have 14 attributes left.

##  13. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  14. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  15. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  16. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

## After 16 iterations, +3.6 secs:

##  confirmed 1 attribute: FedFundsRate;

##  rejected 3 attributes: Inf.L1, Inf.L3, UM.Con.Sent;

##  still have 10 attributes left.

##  17. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  18. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  19. run of importance source...
```

```
## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  20. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  21. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  22. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  23. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  24. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  25. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  26. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

## After 26 iterations, +5.3 secs:

##  confirmed 2 attributes: X2MA, X3MA;

##  still have 8 attributes left.

##  27. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest
```

```
##  28. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  29. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  30. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  31. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  32. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  33. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  34. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

## After 34 iterations, +6.7 secs:

##  confirmed 1 attribute: PercChangeLag2;

##  still have 7 attributes left.

##  35. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  36. run of importance source...
```

```
## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  37. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  38. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  39. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  40. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

## After 40 iterations, +7.8 secs:

##  confirmed 1 attribute: FedFundsRateL1;

##  still have 6 attributes left.

##  41. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  42. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  43. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  44. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest
```

```
##  45. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  46. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  47. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  48. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  49. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  50. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  51. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

## After 51 iterations, +9.7 secs:

##  rejected 1 attribute: Inf.L4;

##  still have 5 attributes left.

##  52. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  53. run of importance source...
```

```
## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

## After 53 iterations, +10 secs:

##  confirmed 2 attributes: Indus.Prod.Ind, NFCI;

##  still have 3 attributes left.

##  54. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  55. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  56. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  57. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  58. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  59. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  60. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  61. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest
```

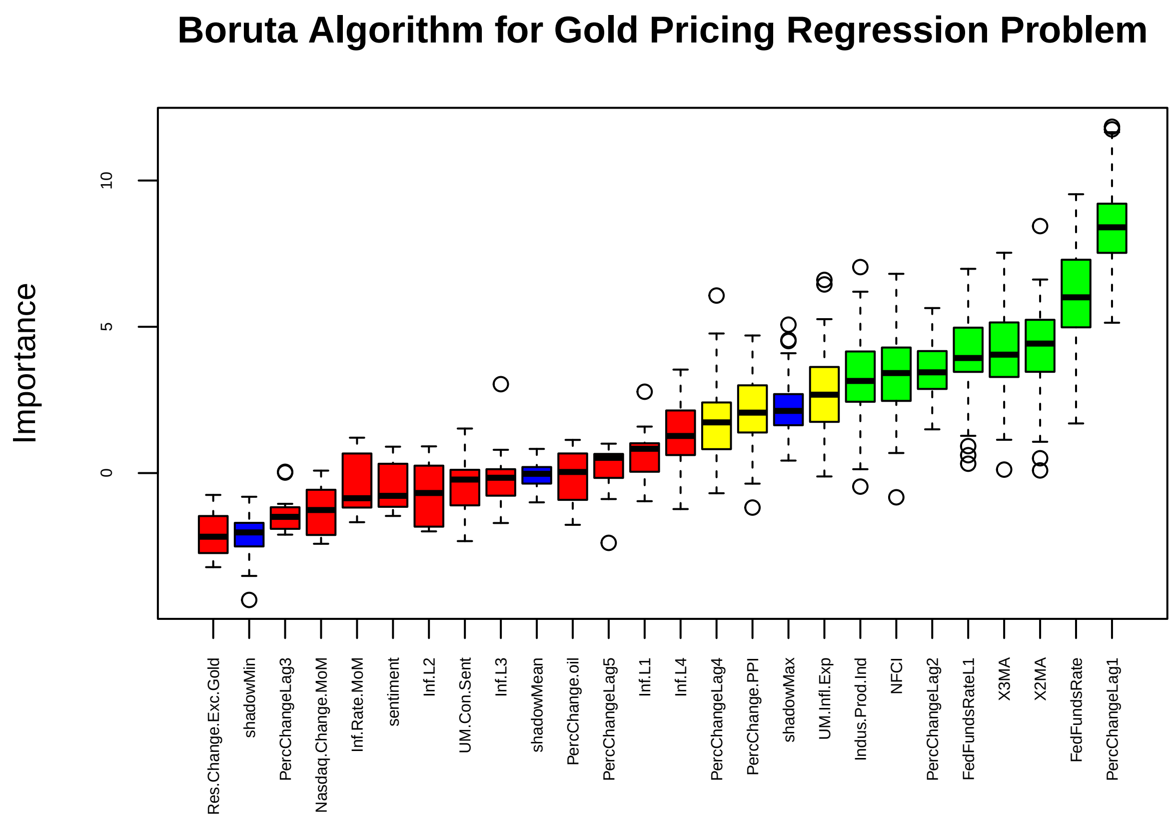
```
##  62. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  63. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  64. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  65. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  66. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  67. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  68. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  69. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  70. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  71. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest
```

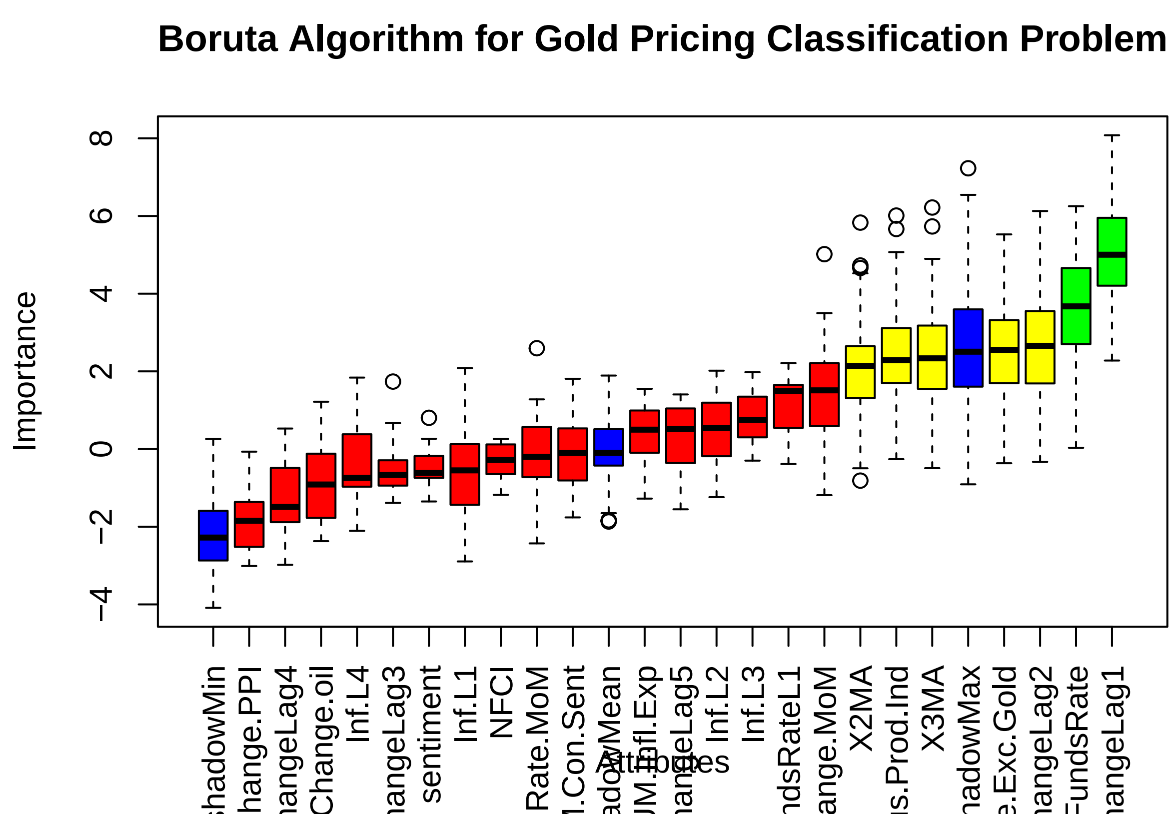
```
##  72. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  73. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  74. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  75. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  76. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  77. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  78. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  79. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  80. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  81. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest
```

```
##  82. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  83. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  84. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  85. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  86. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  87. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  88. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  89. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  90. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  91. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest
```

```
##  92. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  93. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  94. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  95. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  96. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  97. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  98. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest

##  99. run of importance source...

## Warning in ranger::ranger(data = x, dependent.variable.name =
## "shadow.Boruta.decision", : Unused arguments: randomForest
```

Plot Boruta Algorithm





```
##      PercChangeLag1      PercChangeLag2      PercChangeLag3      PercChangeLag4
##           Confirmed           Confirmed            Rejected           Tentative
##      PercChangeLag5                X2MA                X3MA        Inf.Rate.MoM
##            Rejected           Confirmed           Confirmed            Rejected
##              Inf.L1              Inf.L2              Inf.L3              Inf.L4
##            Rejected            Rejected            Rejected            Rejected
## Res.Change.Exc.Gold         UM.Infl.Exp         UM.Con.Sent      Indus.Prod.Ind
##            Rejected           Tentative            Rejected           Confirmed
##    Nasdaq.Change.MoM                NFCI        FedFundsRate      FedFundsRateL1
##            Rejected           Confirmed           Confirmed           Confirmed
##      PercChange.oil      PercChange.PPI           sentiment
##            Rejected           Tentative            Rejected
## Levels: Tentative Confirmed Rejected
```

```
##      PercChangeLag1      PercChangeLag2      PercChangeLag3      PercChangeLag4
##           Confirmed           Tentative            Rejected            Rejected
##      PercChangeLag5                X2MA                X3MA        Inf.Rate.MoM
##            Rejected           Tentative           Tentative            Rejected
##              Inf.L1              Inf.L2              Inf.L3              Inf.L4
##            Rejected            Rejected            Rejected            Rejected
## Res.Change.Exc.Gold         UM.Infl.Exp         UM.Con.Sent      Indus.Prod.Ind
##           Tentative            Rejected            Rejected           Tentative
##    Nasdaq.Change.MoM                NFCI        FedFundsRate      FedFundsRateL1
##            Rejected            Rejected           Confirmed            Rejected
##      PercChange.oil      PercChange.PPI           sentiment
##            Rejected            Rejected            Rejected
## Levels: Tentative Confirmed Rejected
```

Create Five Fold Cross Validation

```r
library(caret)
```

```
## Loading required package: lattice
```

```r
set.seed(123)
folds <- createFolds(gold.r$PercChangeForc, k = 5, returnTrain = TRUE)
```

Regression

OLS

```r
set.seed(123)
# Run linear regression
lin.reg <- lm(PercChangeForc~., data = gold.r)

summary(lin.reg)
```

```
##
## Call:
## lm(formula = PercChangeForc ~ ., data = gold.r)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.168   -2.325   -0.288    2.061   42.308
##
## Coefficients: (2 not defined because of singularities)
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          0.4651653  2.4984810   0.186  0.85238
## PercChangeLag1       0.2701653  0.0579685   4.661 4.04e-06 ***
## PercChangeLag2      -0.1477146  0.0472258  -3.128  0.00186 **
## PercChangeLag3       0.1061077  0.0459875   2.307  0.02144 *
## PercChangeLag4       0.0734674  0.0446749   1.644  0.10070
## PercChangeLag5       0.0448495  0.0443649   1.011  0.31254
## X2MA                        NA         NA      NA       NA
## X3MA                        NA         NA      NA       NA
## Inf.Rate.MoM         0.0669039  0.2175405   0.308  0.75855
## Inf.L1              -0.2733998  0.2214626  -1.235  0.21759
## Inf.L2              -0.2782493  0.2213102  -1.257  0.20924
## Inf.L3              -0.1364338  0.2134525  -0.639  0.52300
## Inf.L4              -0.2192522  0.2123104  -1.033  0.30224
## Res.Change.Exc.Gold  0.0194529  0.0338006   0.576  0.56520
## UM.Infl.Exp          0.4406176  0.2316589   1.902  0.05774 .
## UM.Con.Sent          0.0132122  0.0218922   0.604  0.54644
## Indus.Prod.Ind      -0.0046561  0.0064421  -0.723  0.47016
## Nasdaq.Change.MoM    0.0041910  0.0382214   0.110  0.91273
## NFCI                -0.0179472  0.3975370  -0.045  0.96401
## FedFundsRate        -1.9597395  0.3940275  -4.974 9.03e-07 ***
## FedFundsRateL1       1.8441132  0.3862064   4.775 2.36e-06 ***
## PercChange.oil      -0.0002575  0.0221643  -0.012  0.99074
## PercChange.PPI      -0.1911220  0.1242824  -1.538  0.12473
## sentiment           -5.7286221  5.2997459  -1.081  0.28025
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.378 on 503 degrees of freedom
## Multiple R-squared:  0.1466, Adjusted R-squared:  0.111
## F-statistic: 4.114 on 21 and 503 DF,  p-value: 5.054e-09
```

```r
# Create empty vector
lin.reg.mse <- vector()

# CV loop for linear regression
for(i in 1:5){

  training.i <- folds[[i]]
  testing.i <- setdiff(1:length(gold.r$PercChangeForc), training.i)
  gold.test <- gold.r[testing.i,]
  gold.train <- gold.r[training.i,]

  # Model
  lin.reg <- lm(PercChangeForc~., data = gold.train)
```

```r
  # Model pred
  lin.reg.pred <- predict(lin.reg, newdata = gold.test)

  # Store MSE vals
  lin.reg.mse[i] <- mean((lin.reg.pred-gold.test$PercChangeForc)^2)

}
```

```
## Warning in predict.lm(lin.reg, newdata = gold.test): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(lin.reg, newdata = gold.test): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(lin.reg, newdata = gold.test): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(lin.reg, newdata = gold.test): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(lin.reg, newdata = gold.test): prediction from a
## rank-deficient fit may be misleading
```

```r
lin.reg.mse
```

```
## [1] 20.21816 22.19575 31.15362 19.19955 12.08847
```

```r
mean(lin.reg.mse)
```

```
## [1] 20.97111
```

Ridge & Lasso

```r
# Load required library
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-7
```

```r
# Convert the data to matrix format
X <- as.matrix(gold.r[, -1]) # takes out response variable
y <- as.matrix(gold.r$PercChangeForc)

# Create a matrix of cross-validation folds

# Initialize vectors to store MSEs
ridgeMSEs <- c()
```

```r
lassoMSEs <- c()

# Perform ridge and lasso regression with cross-validation
for (i in 1:length(folds)) {
  # Split the data into training and testing sets based on the current fold
  trainData <- X[folds[[i]], ] # training and testing are switched here
  trainLabels <- y[folds[[i]]]
  testData <- X[-folds[[i]], ]
  testLabels <- y[-folds[[i]]]

  # Perform ridge regression
  ridgeModel <- cv.glmnet(trainData, trainLabels, alpha = 0)
  ridgePredictions <- predict(ridgeModel, newx = testData, s = "lambda.min")
  ridgeMSE <- mean((testLabels - ridgePredictions)^2)
  ridgeMSEs <- c(ridgeMSEs, ridgeMSE)

  # Perform lasso regression
  lassoModel <- cv.glmnet(trainData, trainLabels, alpha = 1)
  lassoPredictions <- predict(lassoModel, newx = testData, s = "lambda.min")
  lassoMSE <- mean((testLabels - lassoPredictions)^2)
  lassoMSEs <- c(lassoMSEs, lassoMSE)
}

# Find the best ridge model
bestRidgeIndex <- which.min(ridgeMSEs)
bestRidgeMSE <- ridgeMSEs[bestRidgeIndex]
bestRidgeModel <- cv.glmnet(X, y, alpha = 0)
bestRidgePredictions <- predict(bestRidgeModel, newx = X, s = "lambda.min")

# Find the best lasso model
bestLassoIndex <- which.min(lassoMSEs)
bestLassoMSE <- lassoMSEs[bestLassoIndex]
bestLassoModel <- cv.glmnet(X, y, alpha = 1)
bestLassoPredictions <- predict(bestLassoModel, newx = X, s = "lambda.min")

# Print the MSE for the best ridge and lasso models
print(paste("Best Ridge Regression MSE:", bestRidgeMSE))
```

```
## [1] "Best Ridge Regression MSE: 13.6035373189784"
```

```r
print(paste("Best Lasso Regression MSE:", bestLassoMSE))
```

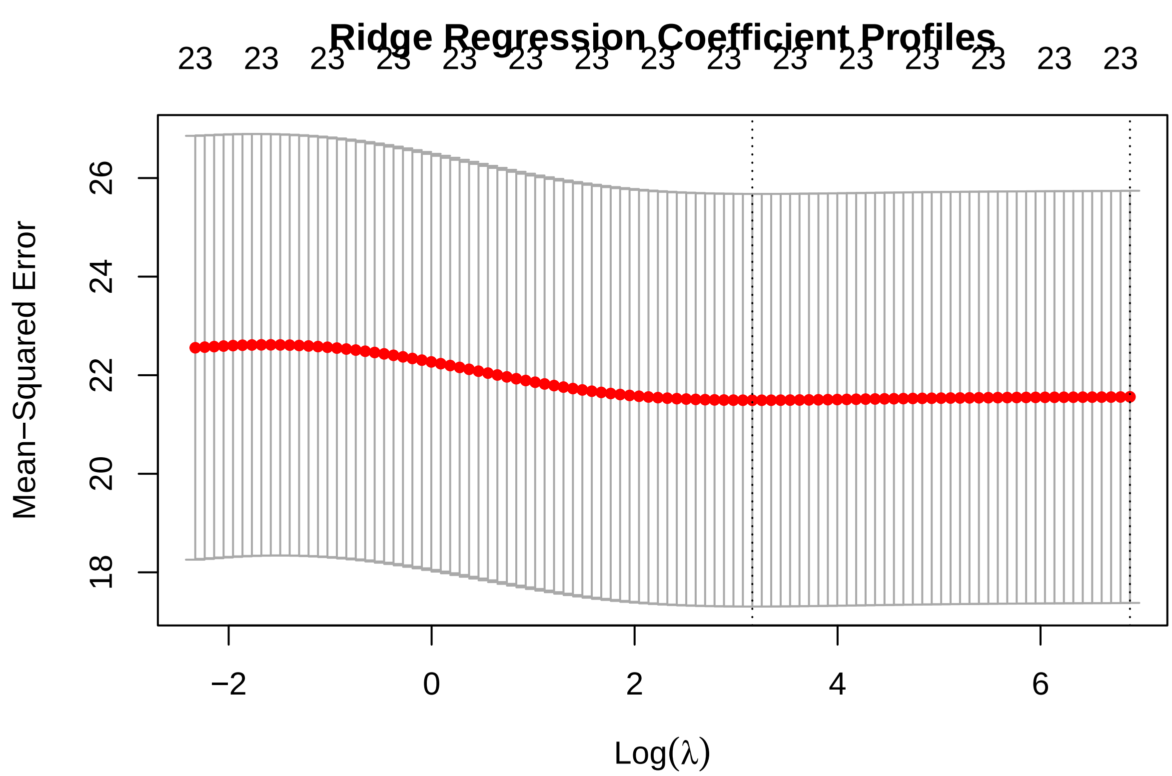
```
## [1] "Best Lasso Regression MSE: 13.378310512051"
```

```r
plot(bestRidgeModel, xvar = "lambda", label = TRUE)
```

```
## Warning in plot.window(...): "xvar" is not a graphical parameter

## Warning in plot.window(...): "label" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "xvar" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "label" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "xvar" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "label" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "xvar" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "label" is not a
## graphical parameter

## Warning in box(...): "xvar" is not a graphical parameter

## Warning in box(...): "label" is not a graphical parameter

## Warning in title(...): "xvar" is not a graphical parameter

## Warning in title(...): "label" is not a graphical parameter
```

```r
title("Ridge Regression Coefficient Profiles")
```



```r
# Plot the coefficient profiles for lasso regression
plot(bestLassoModel, xvar = "lambda", label = TRUE)
```

```
## Warning in plot.window(...): "xvar" is not a graphical parameter
```

```
## Warning in plot.window(...): "label" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "xvar" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "label" is not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "xvar" is not a
## graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "label" is not a
## graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "xvar" is not a
## graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "label" is not a
## graphical parameter
```

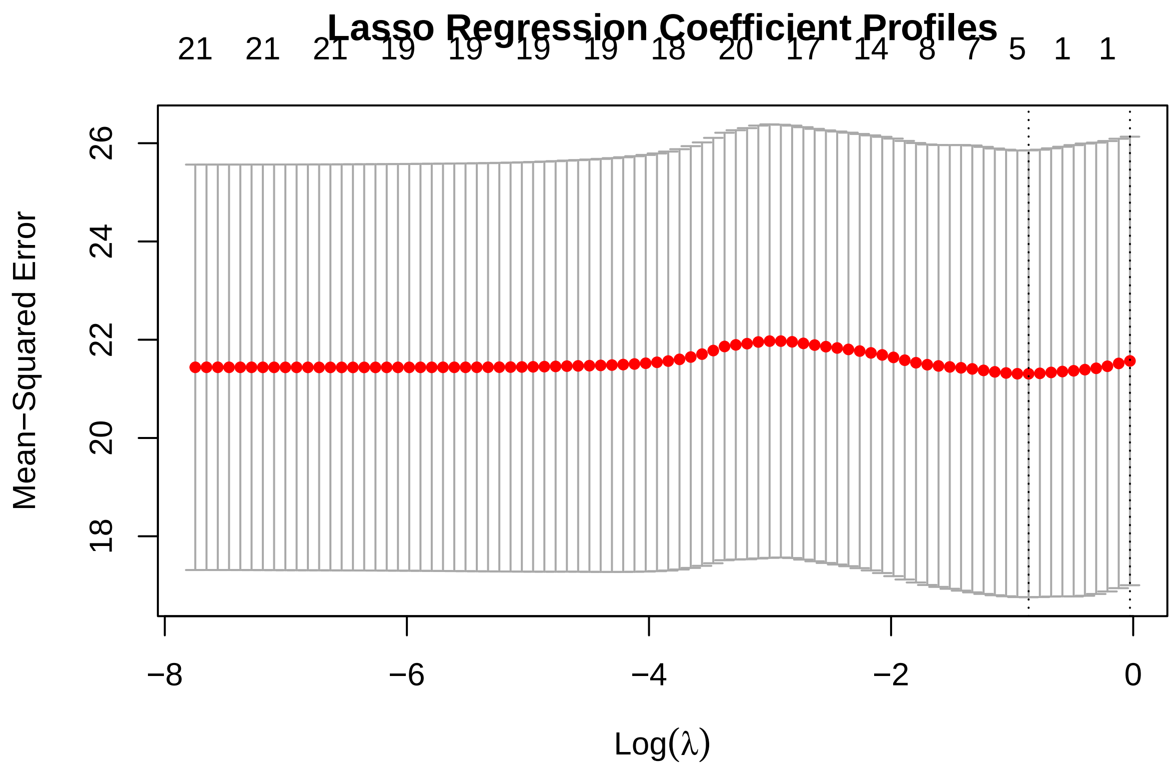
```
## Warning in box(...): "xvar" is not a graphical parameter
```

```
## Warning in box(...): "label" is not a graphical parameter
```

```
## Warning in title(...): "xvar" is not a graphical parameter
```

```
## Warning in title(...): "label" is not a graphical parameter
```

```r
title("Lasso Regression Coefficient Profiles")
```



Principal Components Regression

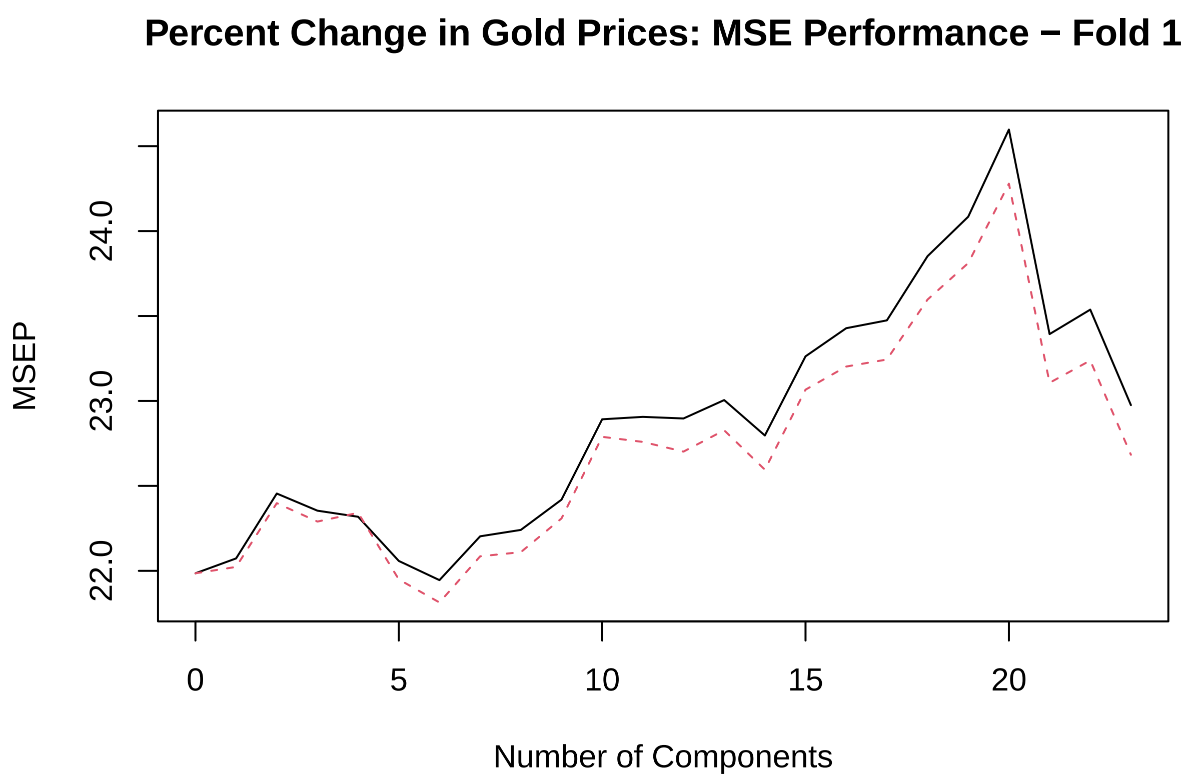
For principal components regression, we'll optimize for the number of components in our regression by minimizing mean squared errors in predicting forecast percent change. We'll use a validation plot to see what number of components minimizes errors - in this case MSEP scores.

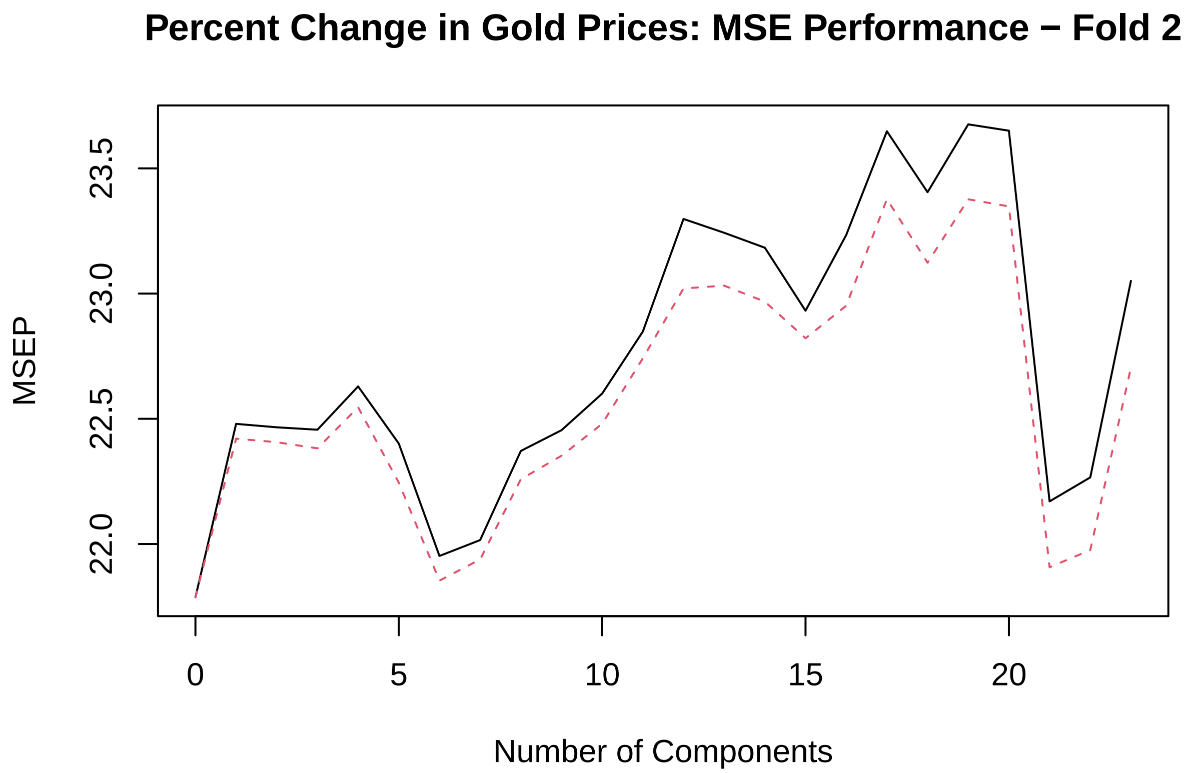
```
# Check Validation Plots for Principal Components Regression
library(pls)
```

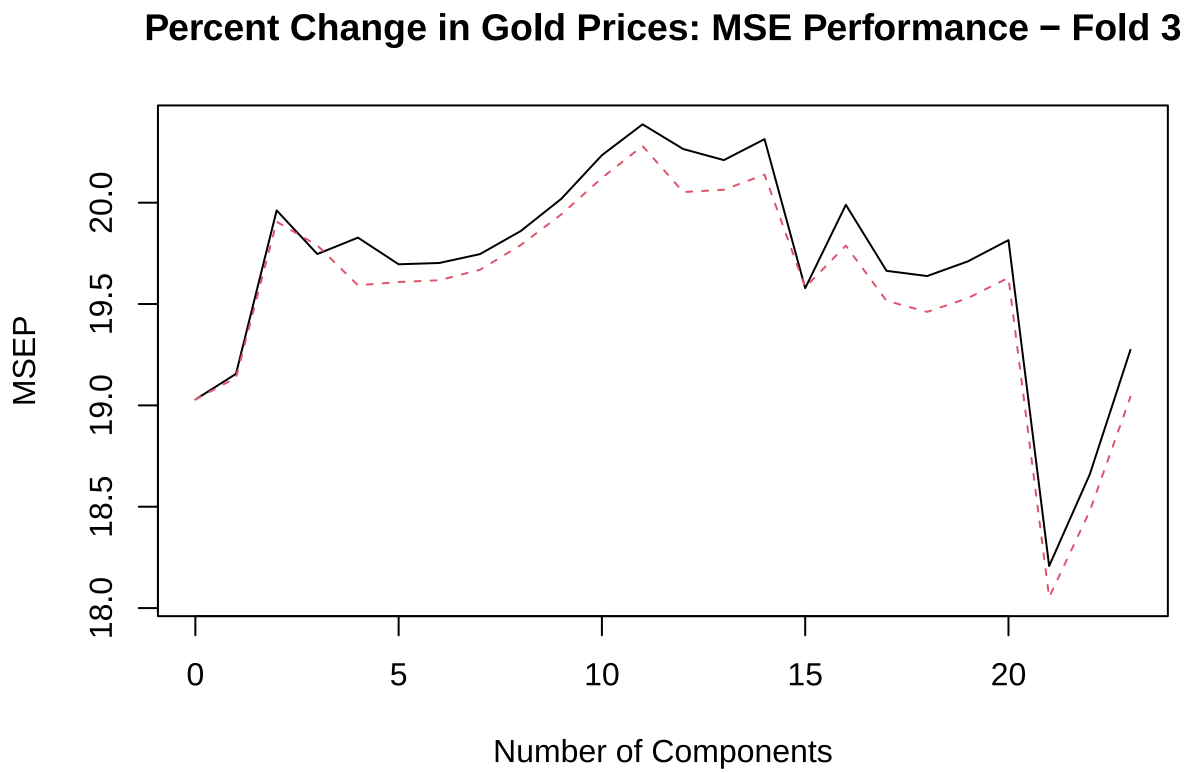
```
##
## Attaching package: 'pls'

## The following object is masked from 'package:caret':
##
##     R2

## The following object is masked from 'package:stats':
##
##     loadings
```

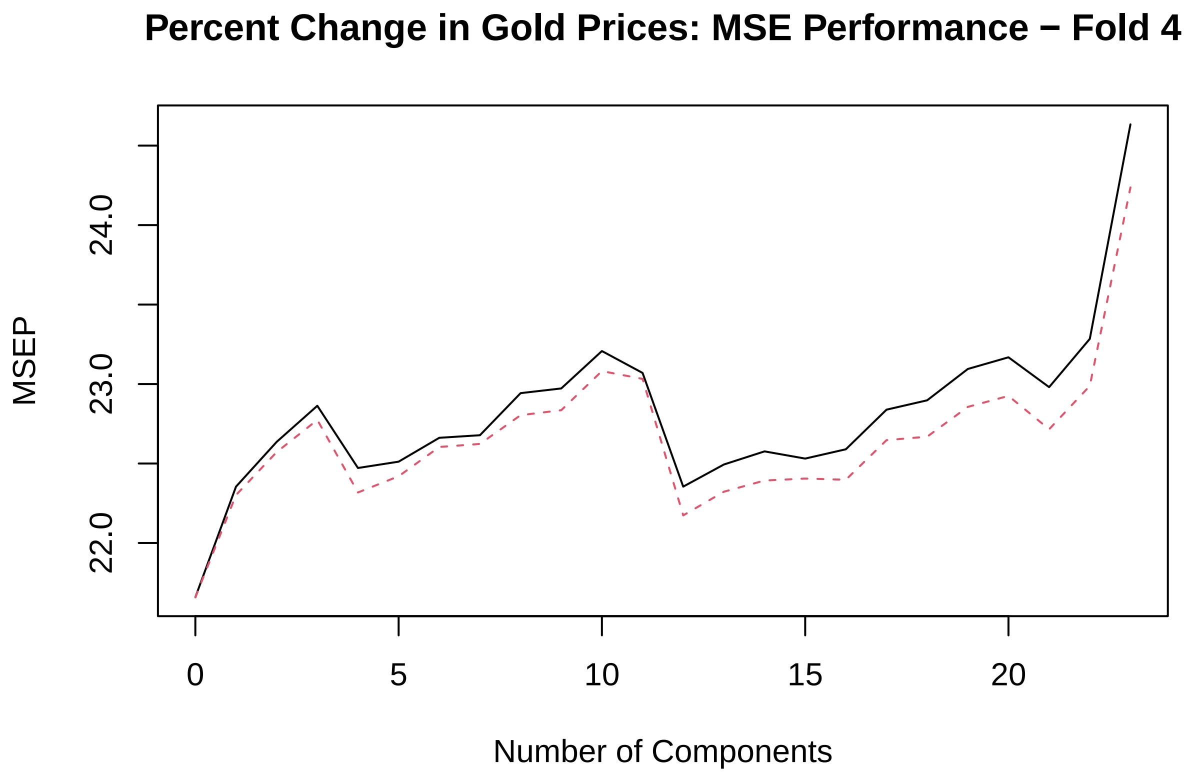
```
for (i in 1:length(folds)){
  # Set Folds
  gold.r.train <- gold.r[folds[[i]],]
  gold.r.test <- gold.r[-folds[[i]],]
```

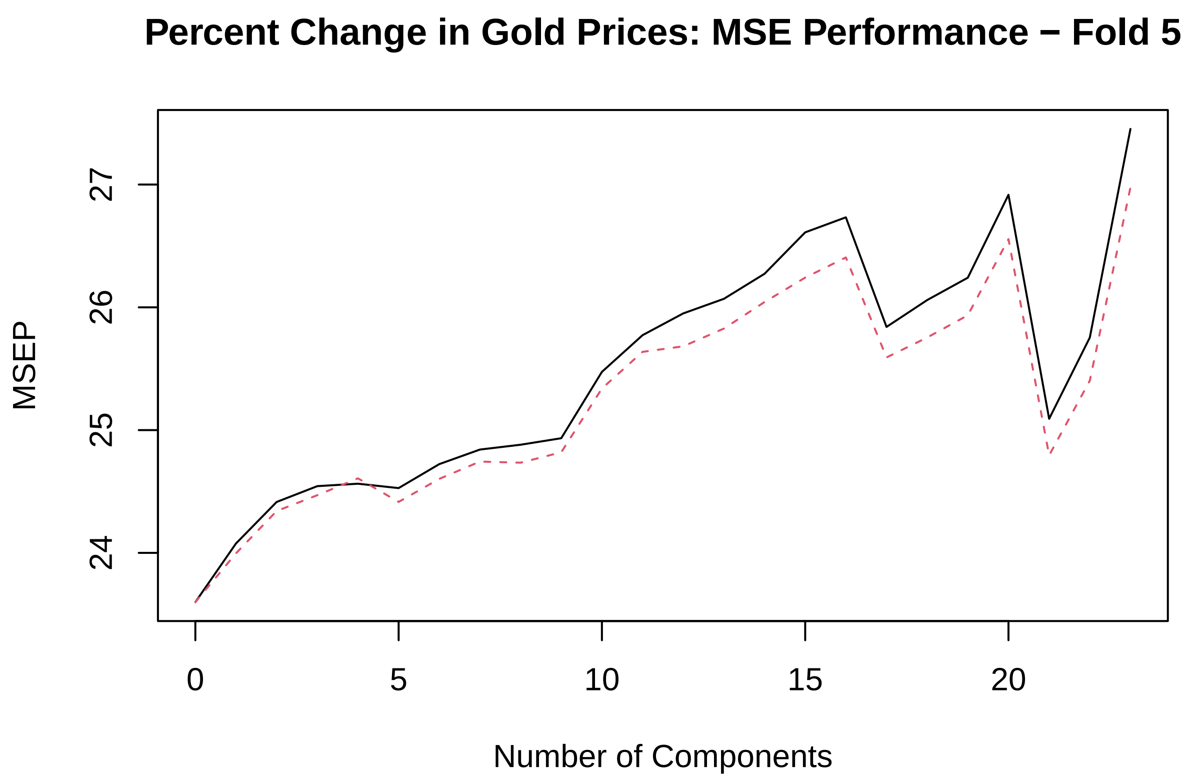
```
  pcr.mdl.r <- pcr(PercChangeForc ~., data = gold.r.train, scale = TRUE,
                   validation = "CV")
  validationplot(pcr.mdl.r, val.type = "MSEP", main = paste0("Percent Change in Gold Prices: MSE Perfor
}
```




Percent Change in Gold Prices: MSE Performance − Fold 2
MSEP
22.0
22.5
23.0
23.5
0
5
10
15
20
Number of Components


Percent Change in Gold Prices: MSE Performance − Fold 3
MSEP
18.0
18.5
19.0
19.5
20.0
0
5
10
15
20
Number of Components


Percent Change in Gold Prices: MSE Performance − Fold 4
MSEP
22.0
23.0
24.0
0
5
10
15
20
Number of Components




```r
# Set Up Data Frame for Receiving Error Vectors
components.r <- 1:(length(gold.r)-1)
pcr.err.df <- data.frame(components.r)

# Fit PCR Model on Five Fold Cross Validation
for (i in 1:length(folds)){
  # Set Folds
  train_index <- folds[[i]]
  gold.r.train <- gold.r[train_index,]
  gold.r.test <- gold.r[-train_index,]

  # Set Error Vector to Capture MSE Scores
  mse.pcr.vector <- vector()

  # For Loop on Multiple Components
  for (j in 1:(length(gold.r)-1)){

  # Fit Regression Model
  pcr.mdl <- pcr(PercChangeForc ~., data = gold.r.train, scale = TRUE)

  # Predict Using PCR
  pcr.predict.r <- predict(pcr.mdl, gold.r.test, ncomp = j)

  # MSE Calculation & Input
  mse.pcr.vector[j] <- mean((gold.r.test$PercChangeForc-pcr.predict.r)^2)
  }
```

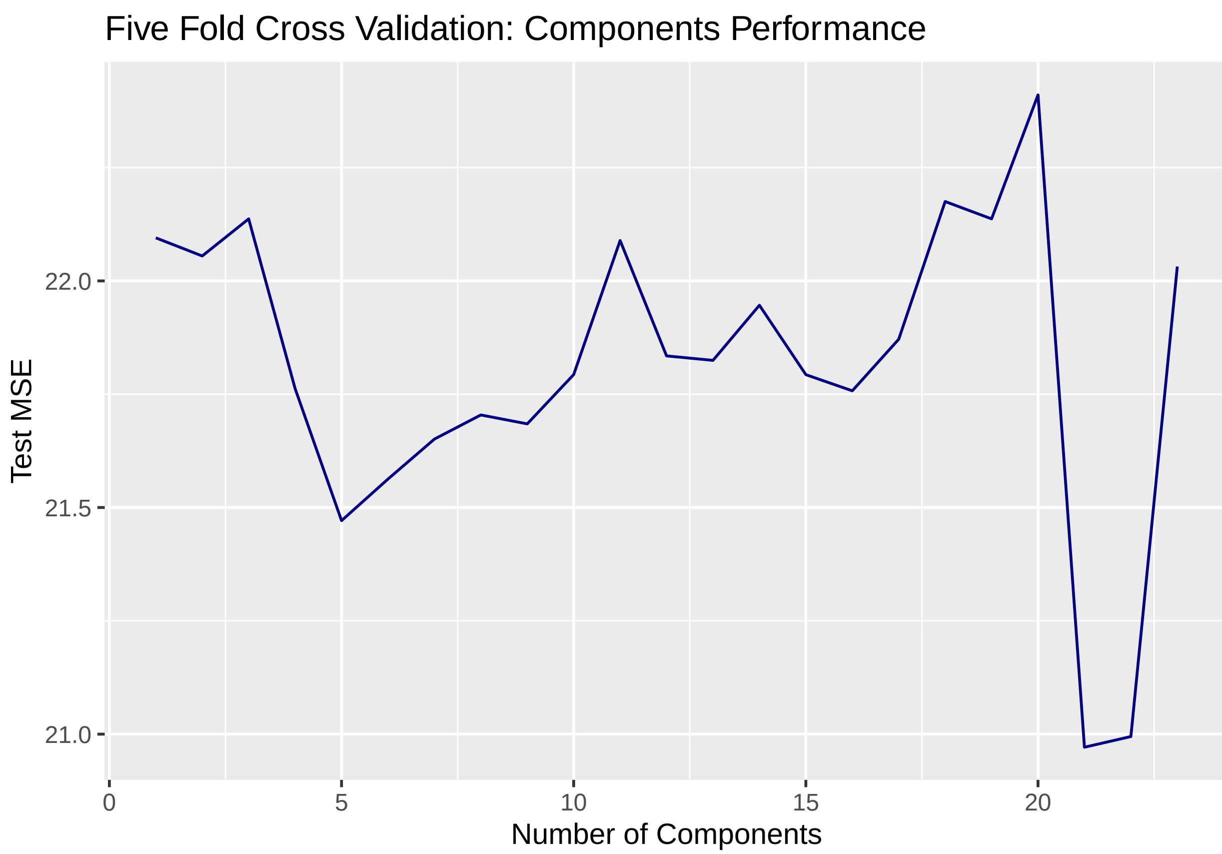
```
  # Input Vector of MSE Values for Each Fold into Data Frame
  pcr.err.df <- cbind(pcr.err.df, mse.pcr.vector)
}

# Clean Data Frame
pcr.err.df$Means <- rowMeans(pcr.err.df[,-1])
colnames(pcr.err.df) <- c("Components","Fold 1","Fold 2","Fold 3", "Fold 4",
                          "Fold 5", "MSE Means")
print(pcr.err.df)
```

```
##    Components   Fold 1   Fold 2   Fold 3   Fold 4   Fold 5 MSE Means
## 1           1 21.71432 20.85342 32.47207 20.82757 14.60775  22.09503
## 2           2 21.61494 21.36228 32.05071 20.83733 14.40957  22.05496
## 3           3 21.71607 21.69754 32.19757 20.66807 14.40522  22.13689
## 4           4 21.72765 21.40838 30.58070 20.68240 14.41327  21.76248
## 5           5 21.11530 21.67707 30.29664 20.68318 13.58305  21.47105
## 6           6 22.08328 21.65311 30.34001 20.39006 13.34744  21.56278
## 7           7 22.08285 21.56652 30.49019 20.71381 13.40140  21.65096
## 8           8 21.82257 21.90039 30.38292 20.75233 13.66288  21.70422
## 9           9 21.71543 21.94631 30.28980 20.72151 13.74993  21.68460
## 10         10 22.04022 21.92963 30.71793 20.62698 13.65182  21.79331
## 11         11 22.56539 22.20276 31.26069 20.66733 13.74952  22.08914
## 12         12 21.51560 22.29751 29.98515 22.81650 12.55782  21.83452
## 13         13 21.38684 22.46961 29.93464 22.81441 12.51786  21.82467
## 14         14 21.91138 22.76365 30.07512 22.40067 12.58072  21.94631
## 15         15 21.56696 22.73061 30.12681 22.31029 12.23104  21.79314
## 16         16 21.60597 22.69006 29.57958 21.78462 13.12727  21.75750
## 17         17 22.28950 22.56039 30.40149 21.81156 12.29350  21.87129
## 18         18 21.63750 22.68808 31.45826 22.25827 12.83351  22.17512
## 19         19 21.82613 22.57803 31.35752 22.13850 12.78410  22.13686
## 20         20 22.31650 22.95660 31.35686 22.52884 12.89321  22.41040
## 21         21 20.21816 22.19575 31.15362 19.19955 12.08847  20.97111
## 22         22 20.44378 22.41981 30.99758 19.07033 12.04181  20.99466
## 23         23 25.64551 22.53415 30.82628 19.06358 12.08931  22.03176
```

```
# Locate Minimium MSE in Components
min.pcr.error.rn <- which(pcr.err.df$`MSE Means` == min(pcr.err.df$`MSE Means`))
print(pcr.err.df[min.pcr.error.rn,])
```

```
##    Components   Fold 1   Fold 2   Fold 3   Fold 4   Fold 5 MSE Means
## 21         21 20.21816 22.19575 31.15362 19.19955 12.08847  20.97111
```

```
# Additional Plots for Presentation: Along with MSE Plots
ggplot(data = pcr.err.df, aes(x = Components, y = `MSE Means`))+
  geom_line(color = "navy")+labs(x = "Number of Components", y = "Test MSE",
       title = "Five Fold Cross Validation: Components Performance")
```



After running five fold cross validation, we find that setting 21 components in our principal components regression optimally minimizes mean squared error. From running cross validation, our model returns an average mean squared error of **20.97**.

Random Forest + Bagging

```
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
set.seed(123)

# mtry.vals <- seq(1, 22, by = 1)
#
# bag.mse <- vector()
# mod.df.r <- data.frame(mtry.vals)
#
# for (i in 1:5){
#
#   training.i <- folds[[i]]
#   testing.i <- setdiff(1:length(gold.r$PercChangeForc), training.i)
#   gold.test <- gold.r[testing.i,]
#   gold.train <- gold.r[training.i,]
#
```

```
#
#   for (j in seq_along(mtry.vals)){
#
#   mod <- randomForest(PercChangeForc~., data = gold.train, mtry = mtry.vals[j], importance = TRUE)
#
#   mod.pred <- predict(mod, newdata = gold.test)
#
#   bag.mse[j] <- mean((mod.pred-gold.test$PercChangeForc)^2)
#
#   }
#   # Append our MSE Vector into Our Data Frame (Memory)
#   mod.df.r <- cbind(mod.df.r, bag.mse)
#
# }
# mod.df.r
#
# mod.df.r$mean_mse <- rowMeans(mod.df.r[, -1])
#
# # Optimal bagging mtry
# opt.mtry <- mod.df.r[which.min(mod.df.r$mean_mse), ]
# opt.mtry
#
# # plot(randomForest(PercChangeForc~., data = gold.train, mtry = 1, importance = TRUE))
#
# # Optimal Random Forest MSE: mtry = 1, MSE = 21.21
#
# # Bagging MSE: MSE = 25.70529


# Optimize Random Forest Model for Parameters
library(tree)
set.seed(75849)

# Set Grid of Parameters for Random Forests
rf.c.grid <- expand.grid(mtry = 1:22, ntree = c(100,200,300,400,500,1000,1500))
nrow(rf.c.grid)
```

```
## [1] 154
```

```
# Set Up Data Frame to Input Values from Manual Grid Search
rf.perf.df <- data.frame(rf.c.grid[,1]) # First Column is mtry
rf.perf.df <- cbind(rf.perf.df, rf.c.grid[,2]) # Second Column is number of trees

# Manual For Loop Grid Search
for (i in 1:length(folds)){
  # Set Folds
  train_index <- folds[[i]]
  gold.r.train <- gold.r[train_index,]
  gold.r.test <- gold.r[-train_index,]

  # Set Up Empty Vector to Input MSE for Each Fold
  rf_mse_vec <- vector()
```

```r
  # Now Set Up Grid Search For Loop Using rf.perf.df Parameter Grid
  for (j in 1:nrow(rf.perf.df)){
    rf.model <- randomForest(PercChangeForc ~.,
                             data = gold.r.train, mtry = rf.perf.df[j,1],
                             ntree = rf.perf.df[j,2])

    # Predict on Testing Set
    rf.predictions <- predict(rf.model, newdata = gold.r.test)

    # Calculate MSE
    rf.mse <- mean((rf.predictions-gold.r.test$PercChangeForc)^2)

    # Input into Vector
    rf_mse_vec[j] <- rf.mse
  }

  # Put Error Vector into RF Data Frame to Measure MSE
  rf.perf.df <- cbind(rf.perf.df, rf_mse_vec)

  # Run This Back Up Again for Five Folds, Data Frame Has Five Columns
}

# Clean Up Data Frame
rf.perf.df$Means <- rowMeans(rf.perf.df[,-c(1,2)])
colnames(rf.perf.df) <- c("Mtry Parameters", "Number of Trees", "Fold 1", "Fold 2",
                          "Fold 3", "Fold 4", "Fold 5", "MSE")
print(rf.perf.df)
```

```
##     Mtry Parameters Number of Trees   Fold 1   Fold 2   Fold 3   Fold 4
## 1                 1             100 23.43578 21.72790 28.36185 20.51760
## 2                 2             100 23.37892 21.98201 27.27879 22.10163
## 3                 3             100 22.95229 21.19556 27.54655 20.66961
## 4                 4             100 23.47850 23.30421 27.97688 21.52767
## 5                 5             100 24.43947 21.97471 28.11313 22.17098
## 6                 6             100 22.96888 21.63678 25.87410 21.57446
## 7                 7             100 26.24996 22.87398 25.62521 21.24058
## 8                 8             100 24.92213 22.55355 27.22020 21.10783
## 9                 9             100 26.94493 24.91034 25.50737 21.27473
## 10               10             100 26.83316 25.98373 28.44968 20.76634
## 11               11             100 26.91941 22.18633 27.36632 22.17185
## 12               12             100 25.24883 23.93946 28.18558 21.22019
## 13               13             100 28.91388 26.42209 26.05249 20.69330
## 14               14             100 28.60420 28.01625 26.40340 22.01810
## 15               15             100 28.89978 29.53045 26.69467 21.69264
## 16               16             100 28.49441 26.47695 25.90523 22.60414
## 17               17             100 28.60463 30.00713 26.62098 22.37535
## 18               18             100 30.48852 28.54921 26.56270 22.27602
## 19               19             100 26.96952 35.45242 26.01291 22.15213
## 20               20             100 32.59285 32.69000 26.31717 21.38315
## 21               21             100 33.80918 35.20048 26.75645 22.16230
## 22               22             100 35.72882 35.17784 28.89830 22.89413
## 23                1             200 22.23483 21.58639 29.53554 20.67840
## 24                2             200 24.15022 21.60892 26.96458 20.75153
```

```
## 25                  3                200 23.98764 22.01009 27.28236 20.21671
## 26                  4                200 23.19813 21.84486 27.15318 20.51186
## 27                  5                200 25.87969 22.83296 26.97558 20.64793
## 28                  6                200 25.50937 23.15797 26.45450 21.58845
## 29                  7                200 26.44728 22.62728 26.80794 21.24331
## 30                  8                200 27.34065 21.96286 26.84199 22.16510
## 31                  9                200 25.47642 23.93708 26.30984 21.26606
## 32                 10                200 27.54949 24.76130 26.56690 22.09701
## 33                 11                200 26.79084 23.96671 26.79412 21.57119
## 34                 12                200 28.25772 26.80693 26.37434 21.80515
## 35                 13                200 29.70309 25.02324 26.84765 21.61609
## 36                 14                200 25.94297 28.58393 26.59917 21.63757
## 37                 15                200 27.52157 28.06073 25.99779 21.62470
## 38                 16                200 26.81051 27.77319 26.22844 22.78297
## 39                 17                200 29.63159 28.84164 25.48410 21.23023
## 40                 18                200 30.37683 30.93037 27.02257 22.67939
## 41                 19                200 30.43350 30.54949 27.49845 22.18329
## 42                 20                200 29.76345 29.81355 26.82743 22.60876
## 43                 21                200 30.26375 32.46027 25.16884 23.04679
## 44                 22                200 32.03689 31.60361 26.21259 21.83426
## 45                  1                300 22.90771 21.48174 28.77842 20.51597
## 46                  2                300 23.57164 21.80866 27.64548 20.43297
## 47                  3                300 24.20176 21.92143 27.55421 20.65670
## 48                  4                300 24.03312 21.53952 27.67321 20.89352
## 49                  5                300 25.74351 22.09968 26.92329 21.20432
## 50                  6                300 25.36379 22.54913 26.59694 20.97592
## 51                  7                300 27.02710 22.82049 26.41288 21.51849
## 52                  8                300 27.88707 23.52449 25.89022 21.28162
## 53                  9                300 23.95341 23.94330 25.15975 21.35130
## 54                 10                300 26.15975 25.16889 26.32510 21.52974
## 55                 11                300 25.27338 25.05657 26.81426 22.26320
## 56                 12                300 26.87714 25.28049 27.01331 21.35389
## 57                 13                300 28.47627 26.33110 26.50004 22.03892
## 58                 14                300 28.17903 27.79539 26.42699 22.39276
## 59                 15                300 28.81802 26.92708 26.51268 21.90053
## 60                 16                300 30.36139 28.75936 25.55909 21.49187
## 61                 17                300 27.97268 27.26290 25.93325 22.04787
## 62                 18                300 29.05324 30.53970 26.88838 21.85355
## 63                 19                300 29.62359 30.88581 26.43094 22.27803
## 64                 20                300 29.73568 30.21767 27.37745 22.19369
## 65                 21                300 31.56447 33.87210 25.96773 21.69414
## 66                 22                300 31.68165 36.15541 27.10377 21.81293
## 67                  1                400 22.61303 21.19352 28.84727 20.76184
## 68                  2                400 24.01377 21.10351 27.52213 20.88377
## 69                  3                400 24.45379 21.55988 26.56139 20.67393
## 70                  4                400 25.01156 21.51482 27.19503 20.98508
## 71                  5                400 24.71300 22.30285 27.05393 21.06204
## 72                  6                400 24.89762 22.68241 27.08811 20.90267
## 73                  7                400 25.98422 22.93960 26.37228 20.85251
## 74                  8                400 26.84832 23.40620 26.61386 21.25399
## 75                  9                400 25.95002 23.60123 26.99309 21.34927
## 76                 10                400 26.55056 24.37453 27.09581 21.39877
## 77                 11                400 27.70268 25.37536 25.71914 21.77278
## 78                 12                400 27.06382 25.14367 26.91749 21.51911
```

```
## 79                13            400 27.90064 26.86537 26.63314 21.69076
## 80                14            400 27.51248 27.88268 25.78084 22.08669
## 81                15            400 29.02021 27.85892 26.34248 21.82225
## 82                16            400 28.29226 27.68996 26.63517 21.64948
## 83                17            400 28.84616 28.00751 24.99085 21.64854
## 84                18            400 31.04886 31.14552 26.58432 21.78187
## 85                19            400 30.05657 30.70452 26.43536 22.35513
## 86                20            400 29.38571 31.75221 25.78681 22.13079
## 87                21            400 29.87321 33.67800 25.93366 22.06902
## 88                22            400 31.68601 34.03516 26.41039 21.81853
## 89                 1            500 23.21494 21.05764 29.24293 20.60561
## 90                 2            500 24.18538 21.41850 28.28017 20.63453
## 91                 3            500 24.22236 21.66985 26.93518 21.25737
## 92                 4            500 24.78209 21.50559 26.98845 21.18913
## 93                 5            500 23.81648 22.42975 27.10325 21.07043
## 94                 6            500 25.19075 22.96929 26.90265 21.84099
## 95                 7            500 24.72787 22.91619 26.35369 21.33009
## 96                 8            500 26.32406 23.34286 25.93228 21.40379
## 97                 9            500 26.10931 24.29679 26.92887 21.58437
## 98                10            500 27.31328 23.86289 26.29206 21.64611
## 99                11            500 26.75454 23.96021 26.56475 21.56613
## 100               12            500 27.48241 25.22607 26.51800 21.72618
## 101               13            500 27.60941 25.62081 26.58935 21.75352
## 102               14            500 28.73396 27.46178 26.56002 21.72847
## 103               15            500 29.26028 28.66901 26.87760 21.63284
## 104               16            500 28.75412 28.46012 26.49024 21.85138
## 105               17            500 27.87997 28.54066 26.18474 21.46123
## 106               18            500 29.47299 29.61714 26.46200 22.06449
## 107               19            500 30.00555 31.56567 25.89305 21.73955
## 108               20            500 30.50175 29.03155 26.41906 22.40180
## 109               21            500 30.46325 32.78063 26.01977 22.19433
## 110               22            500 31.86857 34.92730 26.20249 22.04211
## 111                1           1000 22.45187 21.40900 29.14986 20.41149
## 112                2           1000 23.22785 21.47789 27.90599 20.59658
## 113                3           1000 24.72576 21.59159 27.10963 21.14367
## 114                4           1000 24.87639 21.89843 26.92943 20.89749
## 115                5           1000 25.10864 22.17500 26.78373 20.94700
## 116                6           1000 25.74626 22.56908 26.52286 20.99043
## 117                7           1000 26.22461 22.90860 26.60502 21.06689
## 118                8           1000 26.30307 23.23217 26.65461 21.21951
## 119                9           1000 26.09607 23.53273 26.45997 21.38072
## 120               10           1000 27.22849 24.17994 26.38406 21.54461
## 121               11           1000 26.95055 24.70597 26.33634 21.78739
## 122               12           1000 27.31275 25.03717 26.28634 21.41101
## 123               13           1000 26.51303 26.59559 26.07313 21.85763
## 124               14           1000 28.96892 27.14614 26.18811 21.57256
## 125               15           1000 28.67397 27.77120 26.63898 21.74811
## 126               16           1000 28.61982 28.45896 26.23876 21.79531
## 127               17           1000 28.35847 29.21175 26.32142 21.81635
## 128               18           1000 30.17360 30.16232 26.00066 22.14487
## 129               19           1000 30.29397 32.80072 26.27916 22.09789
## 130               20           1000 29.65841 30.68528 26.03017 22.24858
## 131               21           1000 30.92919 31.91386 26.61031 21.58480
## 132               22           1000 30.13375 32.95049 26.34445 22.12153
```

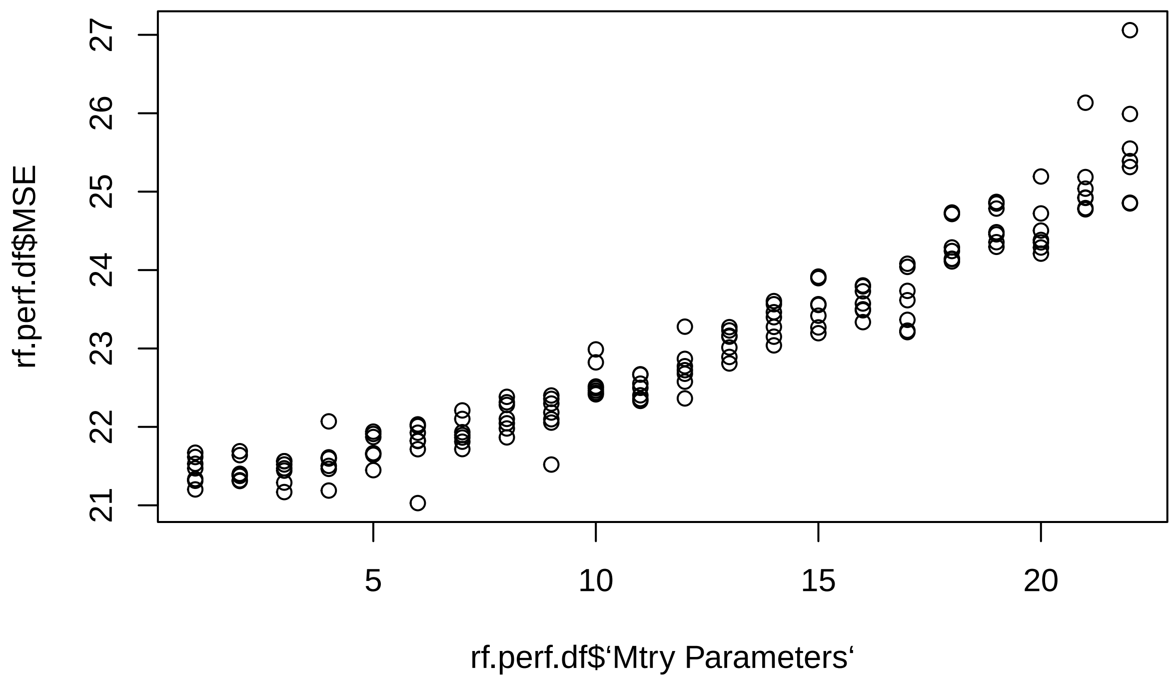
```
## 133               1             1500 22.27941 21.21905 28.53529 20.56768
## 134               2             1500 23.73977 21.43323 27.41216 20.87227
## 135               3             1500 23.81735 21.61505 27.38179 21.16394
## 136               4             1500 24.44340 21.73191 27.04518 20.80682
## 137               5             1500 24.93524 22.17932 26.76290 21.17850
## 138               6             1500 25.61241 22.51316 26.90749 21.40746
## 139               7             1500 25.72110 22.78493 26.29751 21.20519
## 140               8             1500 26.06561 22.92595 26.53358 21.14757
## 141               9             1500 26.53720 23.99120 26.11944 21.64832
## 142              10             1500 26.71438 24.24400 26.44902 21.58087
## 143              11             1500 27.19910 24.92741 26.75621 21.53314
## 144              12             1500 27.80891 25.67845 26.59040 21.40927
## 145              13             1500 27.40922 26.55730 26.43643 21.72232
## 146              14             1500 27.15690 26.96384 26.29329 21.81571
## 147              15             1500 28.20155 26.88520 26.17104 21.78308
## 148              16             1500 29.35250 28.44051 26.37707 22.02789
## 149              17             1500 29.24044 29.63748 26.51761 21.89643
## 150              18             1500 29.70867 29.63401 26.25690 22.25143
## 151              19             1500 29.50800 31.17535 26.09143 21.89435
## 152              20             1500 30.41111 31.83086 26.57378 21.91705
## 153              21             1500 31.44108 32.90931 26.19396 21.83586
## 154              22             1500 31.77536 33.48950 26.18206 22.45148
##       Fold 5      MSE
## 1   14.31147 21.67092
## 2   13.70834 21.68994
## 3   13.47829 21.16846
## 4   14.06413 22.07028
## 5   13.00030 21.93972
## 6   13.08717 21.02828
## 7   13.49144 21.89624
## 8   13.52692 21.86613
## 9   13.14943 22.35736
## 10  12.90681 22.98795
## 11  13.37098 22.40298
## 12  13.22074 22.36296
## 13  13.67368 23.15109
## 14  12.97897 23.60418
## 15  12.76615 23.91674
## 16  14.01686 23.49951
## 17  12.79786 24.08119
## 18  12.67347 24.10998
## 19  13.65413 24.84822
## 20  12.98280 25.19319
## 21  12.74216 26.13412
## 22  12.59698 27.05921
## 23  14.04434 21.61590
## 24  13.12042 21.31913
## 25  13.71424 21.44221
## 26  13.23193 21.18799
## 27  13.21189 21.90961
## 28  13.45074 22.03221
## 29  13.38081 22.10132
## 30  13.60607 22.38334
## 31  13.28312 22.05450
```

```
## 32  13.13905 22.82275
## 33  12.53444 22.33146
## 34  13.14771 23.27837
## 35  12.96154 23.23032
## 36  12.98772 23.15027
## 37  13.13779 23.26852
## 38  13.08568 23.33616
## 39  12.89041 23.61559
## 40  12.56425 24.71468
## 41  13.25656 24.78426
## 42  12.90841 24.38432
## 43  13.01355 24.79064
## 44  12.56149 24.84977
## 45  13.67170 21.47111
## 46  13.47939 21.38763
## 47  13.47934 21.56269
## 48  13.37410 21.50269
## 49  13.38275 21.87071
## 50  13.08914 21.71498
## 51  13.27123 22.21004
## 52  12.98628 22.31394
## 53  13.18967 21.51949
## 54  13.28014 22.49272
## 55  13.08296 22.49808
## 56  13.09140 22.72325
## 57  13.01441 23.27215
## 58  13.02745 23.56432
## 59  12.93401 23.41846
## 60  12.78708 23.79176
## 61  12.82572 23.20848
## 62  13.10764 24.28850
## 63  13.06010 24.45570
## 64  12.99270 24.50344
## 65  12.83194 25.18608
## 66  13.19763 25.99028
## 67  13.14700 21.31253
## 68  13.35013 21.37466
## 69  13.19749 21.28929
## 70  13.27485 21.59627
## 71  13.19387 21.66514
## 72  13.52638 21.81944
## 73  13.17003 21.86373
## 74  13.27663 22.27980
## 75  13.02151 22.18303
## 76  13.15747 22.51543
## 77  12.78656 22.67130
## 78  12.74367 22.67755
## 79  12.72301 23.16258
## 80  13.11496 23.27553
## 81  12.73155 23.55508
## 82  13.16823 23.48702
## 83  12.64489 23.22759
## 84  13.10963 24.73404
## 85  12.86295 24.48291
```

```
## 86  12.70829 24.35276
## 87  13.06673 24.92412
## 88  12.99940 25.38990
## 89  13.52780 21.52978
## 90  13.68045 21.63981
## 91  13.51892 21.52074
## 92  13.53969 21.60099
## 93  12.82077 21.44814
## 94  13.14461 22.00966
## 95  13.25648 21.71686
## 96  13.22513 22.04562
## 97  13.08267 22.40040
## 98  12.96165 22.41520
## 99  12.92882 22.35489
## 100 12.91577 22.77368
## 101 12.88634 22.89189
## 102 12.82884 23.46262
## 103 13.04243 23.89643
## 104 13.08998 23.72917
## 105 12.75976 23.36527
## 106 13.10560 24.14444
## 107 12.57339 24.35544
## 108 12.69203 24.20924
## 109 13.14482 24.92056
## 110 12.70626 25.54935
## 111 13.25297 21.33504
## 112 13.34000 21.30966
## 113 13.24210 21.56255
## 114 13.45719 21.61179
## 115 13.20774 21.64442
## 116 13.29689 21.82510
## 117 12.85721 21.93247
## 118 13.10177 22.10223
## 119 13.01847 22.09759
## 120 12.97702 22.46283
## 121 12.97847 22.55175
## 122 12.83869 22.57719
## 123 13.00358 22.80859
## 124 13.11763 23.39867
## 125 12.99284 23.56502
## 126 12.75543 23.57366
## 127 12.96516 23.73463
## 128 12.73308 24.24291
## 129 12.87588 24.86953
## 130 12.81592 24.28767
## 131 12.83530 24.77469
## 132 12.74508 24.85906
## 133 13.41083 21.20245
## 134 13.55303 21.40209
## 135 13.38157 21.47194
## 136 13.29750 21.46496
## 137 13.24999 21.66119
## 138 13.18788 21.92568
## 139 13.04715 21.81117
```

```
## 140 13.22566 21.97967
## 141 13.19324 22.29788
## 142 13.18701 22.43506
## 143 12.91575 22.66632
## 144 12.84938 22.86728
## 145 12.93872 23.01280
## 146 12.96710 23.03937
## 147 12.93405 23.19498
## 148 12.82397 23.80439
## 149 12.91122 24.04064
## 150 12.84904 24.14001
## 151 12.81297 24.29642
## 152 12.87819 24.72220
## 153 12.80989 25.03802
## 154 12.67128 25.31393
```

```r
plot(rf.perf.df$`Mtry Parameters`, rf.perf.df$MSE)
```




```r
# Find the line with the lowest MSE
lowest_mse_line <- which.min(rf.perf.df$MSE)

print(rf.perf.df[lowest_mse_line, ])
```

```
##   Mtry Parameters Number of Trees   Fold 1   Fold 2  Fold 3   Fold 4   Fold 5
## 6               6             100 22.96888 21.63678 25.8741 21.57446 13.08717
```

```
##        MSE
## 6 21.02828
```

```r
# OPTIMAL RANDOM FOREST PARAMETERS:
# MTRY = 3
# Number of Trees = 300
# Corresponding MSE = 21.05649


rf.perf.df[rf.perf.df$`Mtry Parameters`==22,]
```

```
##     Mtry Parameters Number of Trees   Fold 1   Fold 2   Fold 3   Fold 4
## 22               22             100 35.72882 35.17784 28.89830 22.89413
## 44               22             200 32.03689 31.60361 26.21259 21.83426
## 66               22             300 31.68165 36.15541 27.10377 21.81293
## 88               22             400 31.68601 34.03516 26.41039 21.81853
## 110              22             500 31.86857 34.92730 26.20249 22.04211
## 132              22            1000 30.13375 32.95049 26.34445 22.12153
## 154              22            1500 31.77536 33.48950 26.18206 22.45148
##       Fold 5      MSE
## 22  12.59698 27.05921
## 44  12.56149 24.84977
## 66  13.19763 25.99028
## 88  12.99940 25.38990
## 110 12.70626 25.54935
## 132 12.74508 24.85906
## 154 12.67128 25.31393
```

```r
# OPTIMAL BAGGING TREE COUNT:
# 500 trees had the lowest MSE of all the mtry = 22 iterations
# Corresponding MSE = 24.44521


# Grid Plot of MSE Performance through Five Fold
reg.tree.mse.plot <- ggplot(rf.perf.df, aes(x = `Mtry Parameters`, y = `Number of Trees`,
                       size = `MSE`))+geom_point(col = "navy")+
  labs(x = "Number of Variables Sampled (Mtry)",
       title = "Regression Random Forest Performance")+
  theme_classic()


# ggsave(file = "~/Desktop/reg_plot.png", plot = reg.tree.mse.plot, width = 9, height = 6, bg = "white")
```

Boosting

```r
set.seed(123)
library(caret)
library(gbm)

# Train Function with Grid of Parameters
control <- trainControl(method = "cv", number = 5)
```

```r
# Create Parameter Grid
parameters.r <- expand.grid(n.trees = c(100,200,500,1000,1500),
                            interaction.depth = c(5,10,15,20,50,100),
                            shrinkage = c(0,0.001,0.005,0.01,0.03,0.05),
                            n.minobsinnode = c(5))

# Fit A Boosting (GBM) Model
# features.r <- gold.r[, !colnames(gold.r) %in% "PercChangeForc"]
#
# trained.gbm.fit.r <- train(x = features.r, y = gold.r$PercChangeForc,
#                            method = "gbm", trControl = control,
#                            tuneGrid = parameters.r)
#
# trained.gbm.fit.r$bestTune

# Output:

# n.trees = 100
#
# interaction.depth = 5
#
# shrinkage = 0.01
#
# n.minobsinnode = 5




# Run Boosted Model with optimal Parameters to calculate CV MSE:

boost.mse <- vector()

for (i in 1:5){

  training.i <- folds[[i]]
  testing.i <- setdiff(1:length(gold.r$PercChangeForc), training.i)
  gold.test <- gold.r[testing.i,]
  gold.train <- gold.r[training.i,]

  gold.r.boost <- gbm(PercChangeForc ~ ., data = gold.train, distribution = "gaussian",
                 n.trees = 100, interaction.depth = 5,
                 shrinkage = 0.01)

  # Made preds
  yhat <- predict(gold.r.boost, newdata = gold.test, n.trees = 100)

  # MSE calc
  boost.mse[i] <- mean((yhat - gold.test$PercChangeForc)^2)
}

boost.mse
```

```
## [1] 20.51685 21.93923 29.78842 21.47160 12.94886
```

```
mean(boost.mse)
```

```
## [1] 21.33299
```

Neural Network

Disclaimer: The neural network models ran on one of our markdown files, but can't on this one. In our final presentation, we added our findings from our neural network model. We won't run it in this document because it won't knit.

```
# library(keras)
# library(tensorflow)
# library(reticulate)
# # Define the parameter grid
# layers_grid <- c(1, 2, 3)  # Different numbers of layers
# neurons_grid <- c(32, 64, 128)  # Different numbers of neurons
#
# # Initialize variables to store the best configuration and MSE
# best_layers <- NULL
# best_neurons <- NULL
# best_mse <- Inf
#
# # Perform grid search
# for (layers in layers_grid) {
#   for (neurons in neurons_grid) {
#     # Create the sequential model
#     model <- keras_model_sequential()
#     model %>%
#       layer_dense(units = neurons, activation = "relu", input_shape = ncol(gold.r) - 1)
#     for (i in seq(layers - 1)) {
#       model %>%
#         layer_dense(units = neurons, activation = "relu")
#     }
#     model %>%
#       layer_dense(units = 1)
#
#     # Compile the model
#     model %>% compile(
#       loss = "mean_squared_error",
#       optimizer = "adam",
#       metrics = c("mse")
#     )
#
#     # Train the model
#     history <- model %>% fit(
#       x = as.matrix(gold.r[, -ncol(gold.r)]),
#       y = as.matrix(gold.r$PercChangeForc),
#       epochs = 10,
#       batch_size = 32,
#       validation_split = 0.2
#     )
#
```

```
#     # Calculate the MSE
#     mse <- history$metrics$val_mse[length(history$metrics$val_mse)]
#
#     # Check if the current configuration is the best so far
#     if (mse < best_mse) {
#       best_layers <- layers
#       best_neurons <- neurons
#       best_mse <- mse
#       best_history <- history
#
#     }
#   }
# }
#
# # Print the best configuration and MSE
# print(paste("Best Layers:", best_layers))
# print(paste("Best Neurons:", best_neurons))
# print(paste("Best MSE:", best_mse))
#
# # Plot the training history of the best model
# plot(best_history$metrics$loss, type = "l", col = "blue", xlab = "Epoch", ylab = "Loss",
#      main = "Training History - Best Model")
# lines(best_history$metrics$val_loss, col = "red")
# legend("topright", legend = c("Training Loss", "Validation Loss"), col = c("blue", "red"), lty = 1)
```

Classification Problem

Preliminary boundary visuals

```
library(ggplot2)
# install.packages("gridExtra")
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:randomForest':
##
##     combine

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

```
# Classification plots of important variables according to Baruta

plot1 <- ggplot(gold.c, aes(x = PercChangeLag1, y = FedFundsRate, color = factor(Binary.PercChange))) +
  geom_point() +
  labs(x = "Gold Price % Change", y = "Federal Funds Rate",color = "Binary.PercChange") +
```

```r
  scale_color_manual(values = c("navy", "salmon")) +
  theme_linedraw() +
  theme(legend.position = "none")

plot2 <- ggplot(gold.c, aes(x = UM.Infl.Exp, y = Res.Change.Exc.Gold, color = factor(Binary.PercChange)
  geom_point() +
  labs(x = "Inflation Expectation", y = "US Bank Reserves (less gold)", color = "Binary.PercChange") +
  scale_color_manual(values = c("navy", "salmon")) +
  theme_linedraw() +
  theme(legend.position = "none")

plot3 <- ggplot(gold.c, aes(x = X2MA, y = X3MA, color = factor(Binary.PercChange))) +
  geom_point() +
  labs(x = "2 Month Moving Average Gold Price % Change", y = "2 Month Moving Average Gold Price % Chang
  scale_color_manual(values = c("navy", "salmon")) +
  theme_linedraw() +
  theme(legend.position = "none")

plot4 <- ggplot(gold.c, aes(x = NFCI, y = Indus.Prod.Ind, color = factor(Binary.PercChange))) +
  geom_point() +
  labs(x = "Market Sentiment Index", y = "Industrial Production",color = "Binary.PercChange") +
  scale_color_manual(values = c("navy", "salmon")) +
  theme_linedraw() +
  theme(legend.position = "none")

# grid.plot <- grid.arrange(plot1, plot2, plot3, plot4, nrow = 2, ncol = 2)
# grid.plot

# ggsave(file = "~/Desktop/grid_plot.png", plot = grid.plot, width = 10, height = 6, bg = "white")
```

LDA/QDA

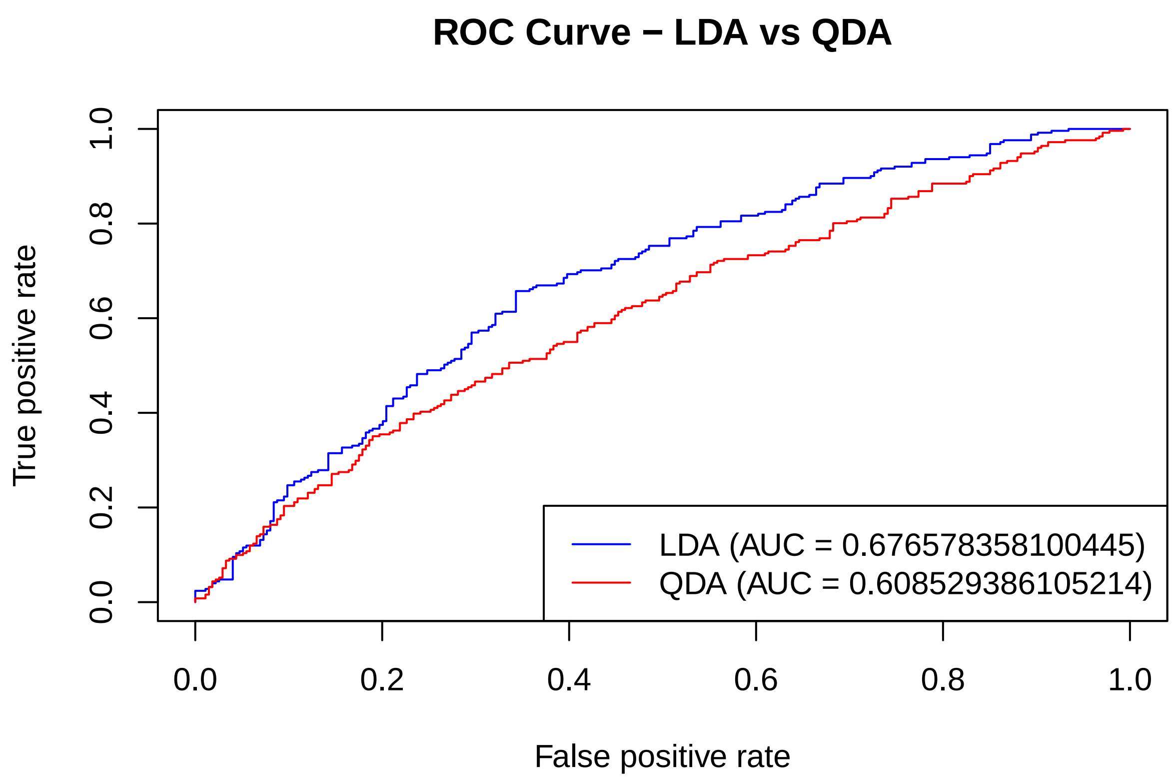
```r
# Load required libraries
library(ROCR)
library(ggplot2)
library(MASS)
```

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

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

```r
# Fit LDA model
ldaModel <- lda(as.factor(Binary.PercChange) ~ ., data = gold.c)
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
# Fit QDA model using selected variables
qdaFeatures <- c("PercChangeLag1", "Indus.Prod.Ind", "FedFundsRate")
qdaModel <- qda(as.factor(Binary.PercChange) ~ ., data = gold.c[, c(qdaFeatures, "Binary.PercChange")])

# Compute predicted probabilities
ldaPred <- predict(ldaModel, newdata = gold.c)$posterior[, 2]  # Use class 1 posterior probability
qdaPred <- predict(qdaModel, newdata = gold.c[, qdaFeatures])$posterior[, 2]  # Use class 1 posterior p

# Create a prediction object for LDA
ldaPrediction <- prediction(ldaPred, gold.c$Binary.PercChange)

# Create a prediction object for QDA
qdaPrediction <- prediction(qdaPred, gold.c$Binary.PercChange)

# Create ROC curve for LDA
ldaROC <- performance(ldaPrediction, "tpr", "fpr")
ldaAUC <- performance(ldaPrediction, "auc")@y.values[[1]]

# Create ROC curve for QDA
qdaROC <- performance(qdaPrediction, "tpr", "fpr")
qdaAUC <- performance(qdaPrediction, "auc")@y.values[[1]]

# Plot ROC curves
plot(ldaROC, col = "blue", main = "ROC Curve - LDA vs QDA")
plot(qdaROC, col = "red", add = TRUE)
legend("bottomright", legend = c(paste0("LDA (AUC = ", ldaAUC, ")"), paste0("QDA (AUC = ", qdaAUC, ")"))
```

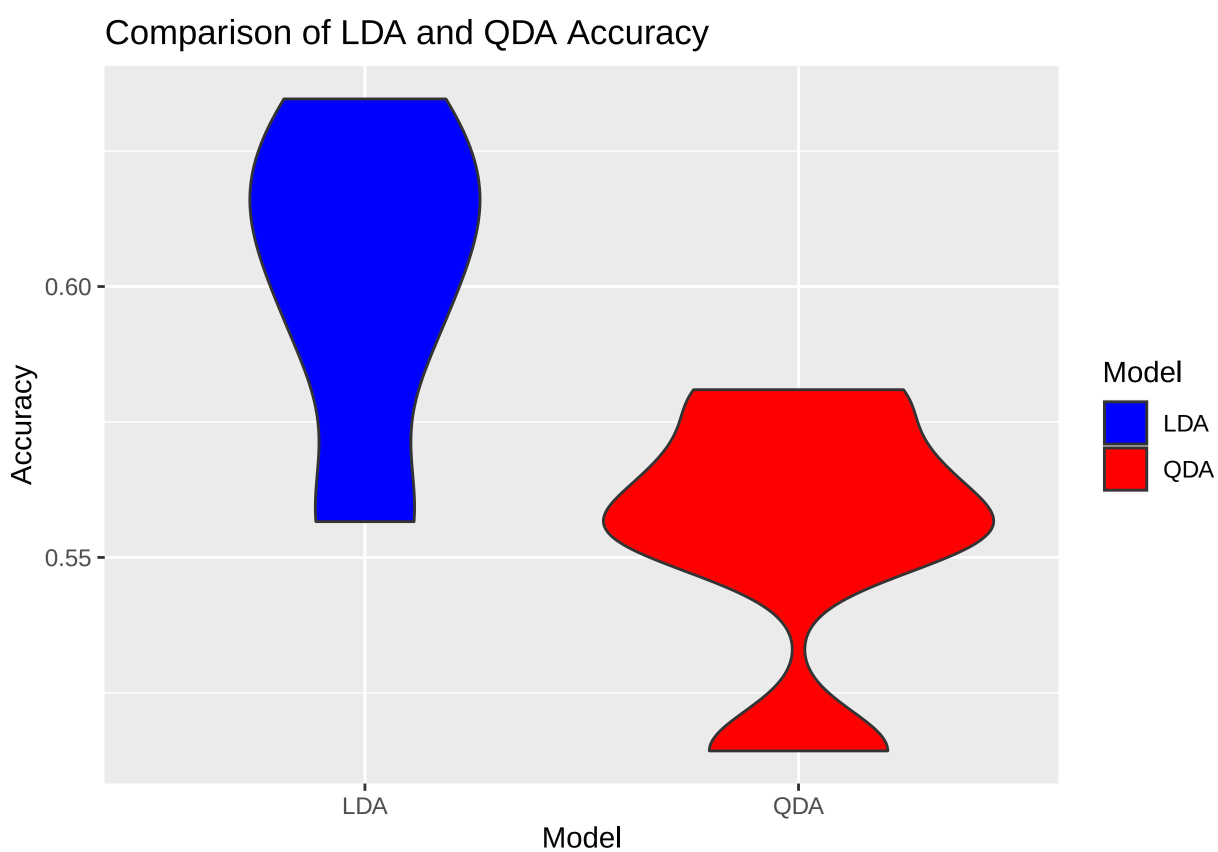



```r
# Initialize vectors to store accuracy scores
ldaAccuracies <- c()
qdaAccuracies <- c()

# Perform cross-validation
for (i in 1:length(folds)) {
  # Split the data into training and testing sets based on the current fold
  trainData <- gold.c[folds[[i]], ]
  testData <- gold.c[-folds[[i]], ]

  # Perform LDA
  ldaModel <- lda(as.factor(Binary.PercChange) ~ ., data = trainData)
  ldaPredictions <- predict(ldaModel, newdata = testData)$class

  # Calculate accuracy for LDA
  ldaAccuracy <- sum(ldaPredictions == testData$Binary.PercChange) / length(testData$Binary.PercChange)
  ldaAccuracies <- c(ldaAccuracies, ldaAccuracy)

  # Perform QDA using selected variables
  qdaFeatures <- c("PercChangeLag1", "Indus.Prod.Ind", "FedFundsRate")
  qdaModel <- qda(as.factor(Binary.PercChange) ~ ., data = trainData[, c(qdaFeatures, "Binary.PercChang
  qdaPredictions <- predict(qdaModel, newdata = testData[, qdaFeatures])$class

  # Calculate accuracy for QDA
  qdaAccuracy <- sum(qdaPredictions == testData$Binary.PercChange) / length(testData$Binary.PercChange)
  qdaAccuracies <- c(qdaAccuracies, qdaAccuracy)
```

```
}
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear

## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
# Create a data frame with accuracy scores
accuracyData <- data.frame(Model = rep(c("LDA", "QDA"), each = length(folds)),
                           Accuracy = c(ldaAccuracies, qdaAccuracies))

# Plot the violin plot
ggplot(accuracyData, aes(x = Model, y = Accuracy, fill = Model)) +
  geom_violin() +
  xlab("Model") +
  ylab("Accuracy") +
  ggtitle("Comparison of LDA and QDA Accuracy") +
  scale_fill_manual(values = c("blue", "red"))
```



```r
# Calculate mean accuracy scores
ldaMeanAccuracy <- mean(ldaAccuracies)
qdaMeanAccuracy <- mean(qdaAccuracies)

# Print the mean accuracy scores
print(paste("LDA Mean Accuracy:", ldaMeanAccuracy))
```

```
## [1] "LDA Mean Accuracy: 0.602053355449582"
```

```r
print(paste("QDA Mean Accuracy:", qdaMeanAccuracy))
```

```
## [1] "QDA Mean Accuracy: 0.554306102702329"
```

KNN: Using Optimal Train

```r
# Optimize Model for Best K Parameters Using Train Function
library(class)
set.seed(75849)
ctrl <- trainControl(method = "cv", number = 5) # Five Fold CV

# Parameter Grid Search
k_values <- expand.grid(k = 1:100)

# Train the Model on Grid Set Above - Parameter Grid Search
knn_optimize <- train(as.factor(Binary.PercChange) ~., data = gold.c,
                      method = "knn", trControl = ctrl, tuneGrid = k_values)
knn_optimize$bestTune
```

```
##     k
## 13 13
```

```r
# Optimize Models (More Manually)
knn_vector_value <- seq(3,51,2)
knn_perf_df <- data.frame(knn_vector_value)

for (i in 1:length(folds)){
  # Set Folds
  train_index <- folds[[i]]
  gold.c.train <- gold.c[train_index,]
  gold.c.test <- gold.c[-train_index,]

  # Set Inputs for KNN Function With Each Fold
  train_features <- gold.c.train[,-length(gold.c.train)]
  test_features <- gold.c.test[,-length(gold.c.train)]
  train_class <- gold.c.train$Binary.PercChange

  # Set Up Vector To Input Error Scores
  knn_error <- vector() # This vector will be wiped out with each fold run

  # Set Up Inner For Loop for KNN Factors, 3-51 (That Would be 50 Parameters)
```

```
  for (j in seq(3,51,2)){
    # Run Model
    knn.mdl <- knn(train_features, test_features, train_class, k = j)

    # Get Accuracy Score
    knn.accuracy <- mean(knn.mdl == gold.c.test$Binary.PercChange)

    # Get Error Score & Input
    knn.error <- 1-knn.accuracy
    knn_error <- append(knn_error, knn.error)
  }
  # Bind to Data Frame (Creating Five Columns for Five Folds)
  knn_perf_df <- cbind(knn_perf_df, knn_error)
}

# Clean Data Frame
knn_perf_df$KNNmeans <- rowMeans(knn_perf_df[,-1])
colnames(knn_perf_df) <- c("K Parameter", "Fold 1", "Fold 2", "Fold 3", "Fold 4",
                           "Fold 5", "Error Means")
print(knn_perf_df)
```

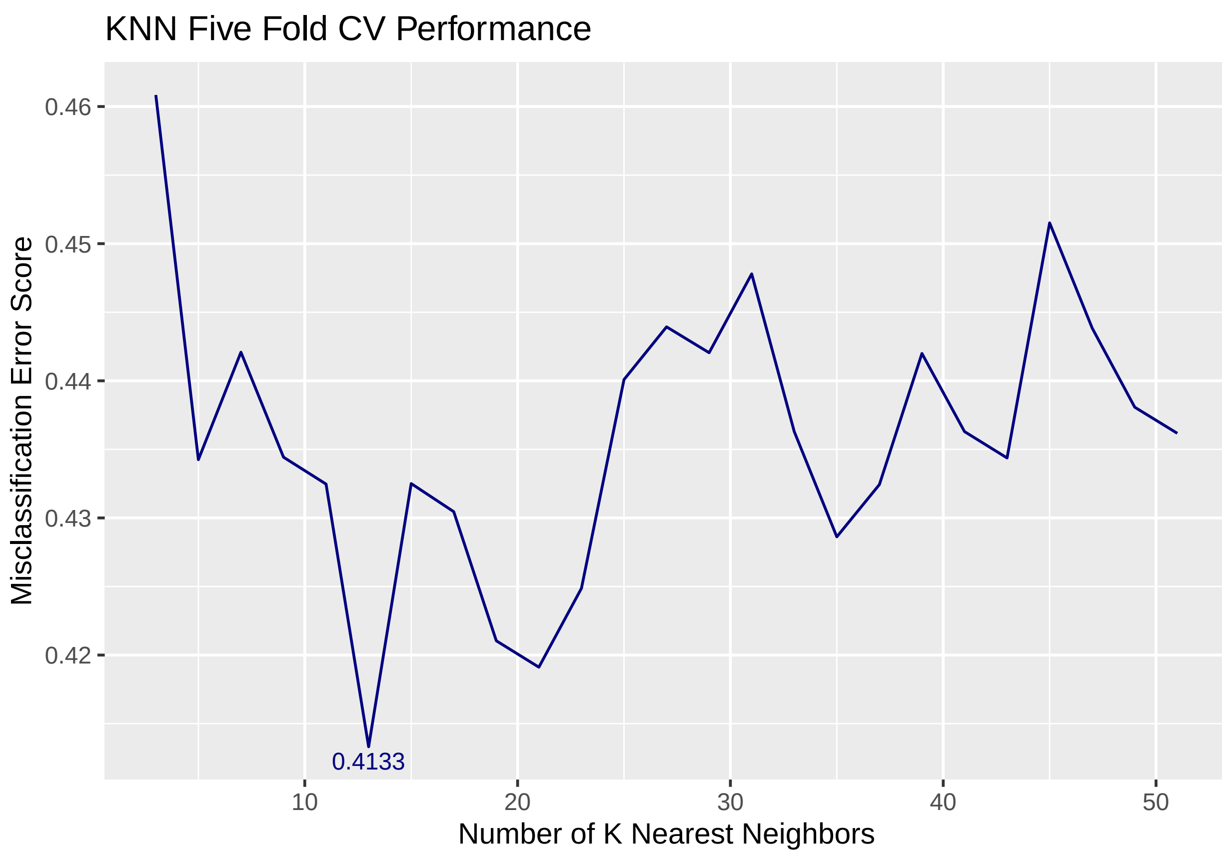
```
##    K Parameter    Fold 1    Fold 2    Fold 3    Fold 4    Fold 5 Error Means
## 1            3 0.4326923 0.4761905 0.4905660 0.4285714 0.4761905   0.4608421
## 2            5 0.4326923 0.4380952 0.4528302 0.4000000 0.4476190   0.4342474
## 3            7 0.5096154 0.4380952 0.4150943 0.4285714 0.4190476   0.4420848
## 4            9 0.4711538 0.4285714 0.3962264 0.4380952 0.4380952   0.4344284
## 5           11 0.4519231 0.4095238 0.4056604 0.4952381 0.4000000   0.4324691
## 6           13 0.4230769 0.4285714 0.4339623 0.4285714 0.3523810   0.4133126
## 7           15 0.4903846 0.4666667 0.4245283 0.4095238 0.3714286   0.4325064
## 8           17 0.4134615 0.4761905 0.4245283 0.4571429 0.3809524   0.4304551
## 9           19 0.4326923 0.4761905 0.3867925 0.4476190 0.3619048   0.4210398
## 10          21 0.4326923 0.4571429 0.3962264 0.4571429 0.3523810   0.4191171
## 11          23 0.4326923 0.4857143 0.3773585 0.4571429 0.3714286   0.4248673
## 12          25 0.4326923 0.5047619 0.3867925 0.4571429 0.4190476   0.4400874
## 13          27 0.4711538 0.4857143 0.4056604 0.4571429 0.4000000   0.4439343
## 14          29 0.4807692 0.4952381 0.4056604 0.4761905 0.3523810   0.4420478
## 15          31 0.4903846 0.4761905 0.3962264 0.4952381 0.3809524   0.4477984
## 16          33 0.4326923 0.4857143 0.3773585 0.4857143 0.4000000   0.4362959
## 17          35 0.4326923 0.4666667 0.4056604 0.4761905 0.3619048   0.4286229
## 18          37 0.4326923 0.4857143 0.4056604 0.4761905 0.3619048   0.4324324
## 19          39 0.4711538 0.4761905 0.4245283 0.4952381 0.3428571   0.4419936
## 20          41 0.4615385 0.4571429 0.4056604 0.4952381 0.3619048   0.4362969
## 21          43 0.4423077 0.4571429 0.3962264 0.4952381 0.3809524   0.4343735
## 22          45 0.4615385 0.4857143 0.4150943 0.5047619 0.3904762   0.4515170
## 23          47 0.4326923 0.4857143 0.4150943 0.4857143 0.4000000   0.4438430
## 24          49 0.4230769 0.4571429 0.4339623 0.4857143 0.3904762   0.4380745
## 25          51 0.4326923 0.4666667 0.4433962 0.4666667 0.3714286   0.4361701
```

```
# Find Minimum Error for K Parameter
knn_min_rn <- which.min(knn_perf_df$`Error Means`)
knn_perf_df[knn_min_rn,]
```

```
##   K Parameter    Fold 1    Fold 2    Fold 3    Fold 4    Fold 5 Error Means
```

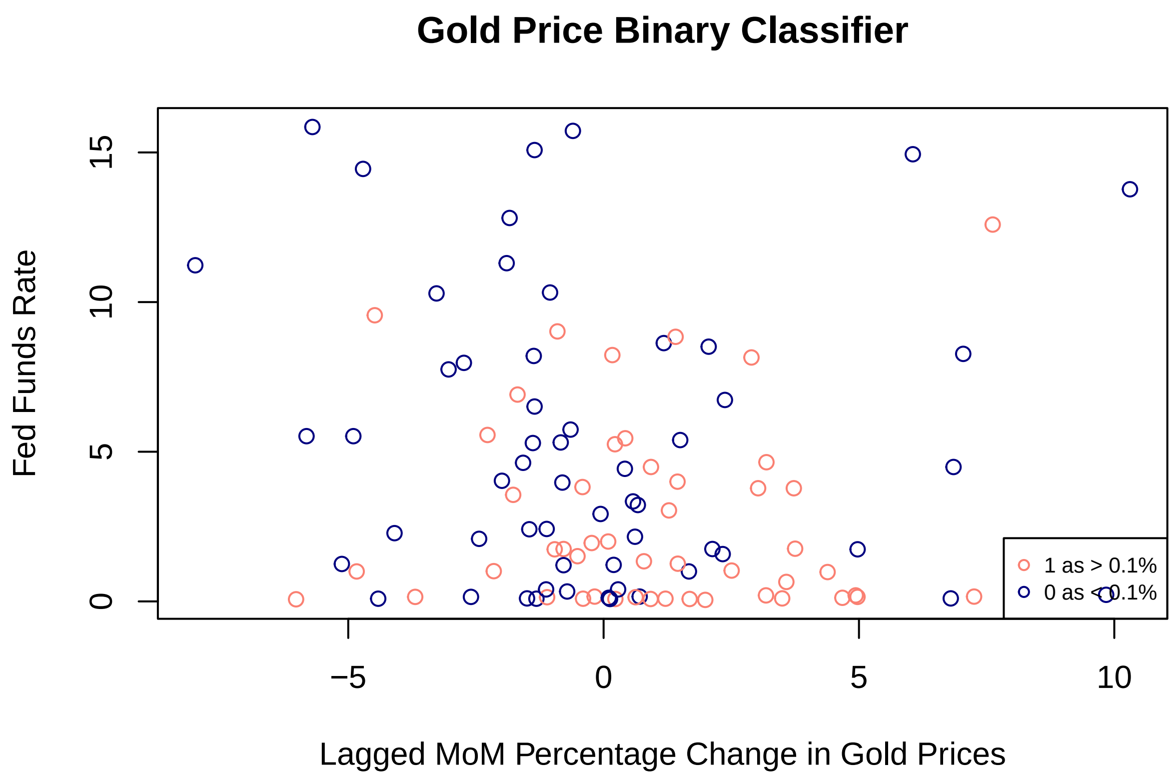
```
## 6          13 0.4230769 0.4285714 0.4339623 0.4285714 0.352381   0.4133126
```

```
### Additional Plots for Presentation ###

# Five Fold CV Error Score Against Number of K Nearest Neighbors
knn.point <- data.frame(`K Parameter` = 13, `Error Means` = 0.4133)

knn.ffcv.plot <- ggplot(knn_perf_df, aes(x = `K Parameter`, y = `Error Means`))+
         geom_line(color = "navy")+
         labs(x = "Number of K Nearest Neighbors",
              y = "Misclassification Error Score",
              title = "KNN Five Fold CV Performance")+
  geom_text(data = knn.point,
            aes(x = K.Parameter, y = Error.Means, label = Error.Means),
             vjust = 1.3, color = "navy", size = 3)
knn.ffcv.plot
```

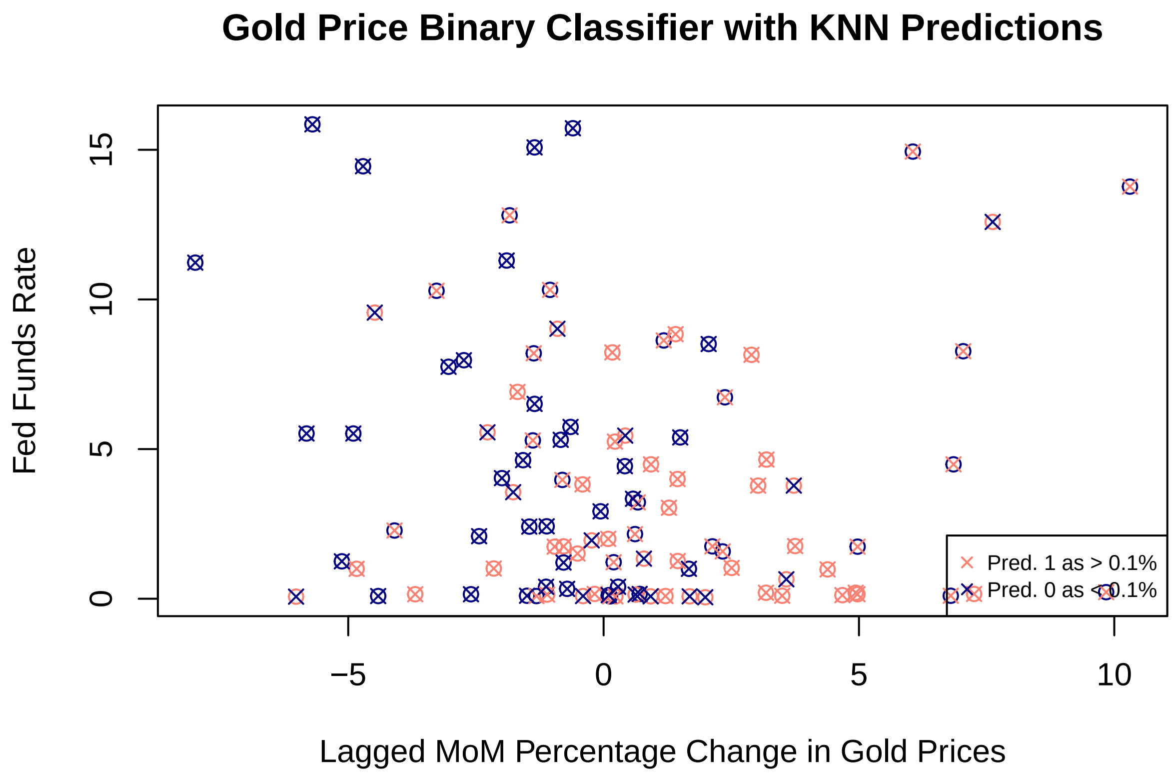



```
# ggsave(file = "~/Desktop/knnffcv.png", plot = knn.ffcv.plot, width = 10,
#         height = 6, bg = "white")

# Create a Scatter Plot for Fifth Fold Test Points
plot(gold.c.test$PercChangeLag1, gold.c.test$FedFundsRate,
     col = ifelse(gold.c.test$Binary.PercChange == 1, "salmon","navy"),
     pch = 1, xlab = "Lagged MoM Percentage Change in Gold Prices",
     ylab = "Fed Funds Rate", main = "Gold Price Binary Classifier")
```

```
legend("bottomright", legend = c("1 as > 0.1%", "0 as < 0.1%"),
       pch = c(1,1), col = c("salmon", "navy"), cex = 0.7)
```




```
# Create Scatter Plot with Fifth Fold Test Points & KNN Predictions
plot(gold.c.test$PercChangeLag1, gold.c.test$FedFundsRate,
     col = ifelse(gold.c.test$Binary.PercChange == 1, "salmon","navy"),
     pch = 1, xlab = "Lagged MoM Percentage Change in Gold Prices",
     ylab = "Fed Funds Rate", main = "Gold Price Binary Classifier with KNN Predictions")
points(gold.c.test$PercChangeLag1, gold.c.test$FedFundsRate,
       col = ifelse(knn.mdl == 1, "salmon","navy"), pch = 4)
legend("bottomright", legend = c("Pred. 1 as > 0.1%", "Pred. 0 as < 0.1%"),
       pch = c(4,4), col = c("salmon", "navy"), cex = 0.7)
```



Both our train hyper-grid search function and manual grid search show that the optimal number for K parameters (the number of neighboring observations) is 13. In other words, 13 observations closest to our observation of interest will be checked to see whether they identify with class 1 (above 0.1% gold price growth) or class 0 (below 0.1% gold price growth). The majority of these 13 classes will be assigned to the observation of interest. Using this algorithm, we get a mis-classification error (on average) of **41.33%**. This is below 50% and is considered better than random chance. Our manual fit is also similar to running an optimized KNN model on five fold cross validation, so we don't have to run that again. The mis-classification error of 41.33%, thus, is a comparable number to other approaches (as we run five fold cross validation for them).

Random Forest & Bagging

```
# Optimize Random Forest Model for Parameters Using Train Function
library(tree)
set.seed(75849)
ctrl <- trainControl(method = "cv", number = 5)

# Set Grid of Parameters for Random Forests
rf.c.grid <- expand.grid(mtry = 1:22, ntree = c(100,200,300,400,500,1000,1500))
nrow(rf.c.grid)
```

```
## [1] 154
```

```r
# Set Up Data Frame to Input Values from Manual Grid Search
rf.perf.df <- data.frame(rf.c.grid[,1]) # First Column is mtry
rf.perf.df <- cbind(rf.perf.df, rf.c.grid[,2]) # Second Column is number of trees

# Manual For Loop Grid Search Since Train Function Refuses to Work
for (i in 1:length(folds)){
  # Set Folds
  train_index <- folds[[i]]
  gold.c.train <- gold.c[train_index,]
  gold.c.test <- gold.c[-train_index,]

  # Set Up Empty Vector to Input Misclassificaiton Errors for Each Fold
  rf_error_vec <- vector()

  # Now Set Up Grid Search For Loop Using rf.perf.df Parameter Grid
  for (j in 1:nrow(rf.perf.df)){ # Will Havet to Change Just Testing
    # Fit the Model, Mtry is the first column of grid, Ntrees is second column
    rf.model <- randomForest(as.factor(Binary.PercChange) ~.,
                             data = gold.c.train, mtry = rf.perf.df[j,1],
                             ntree = rf.perf.df[j,2])

    # Predict on Testing Set, Response is a Class Variable
    rf.predictions <- predict(rf.model, newdata = gold.c.test, type = "class")

    # Calculate Accuracy Score
    rf.accuracy <- mean(rf.predictions == gold.c.test$Binary.PercChange)

    # Calculate Error Scores and Input into Vector
    rf.error <- 1-rf.accuracy
    rf_error_vec[j] <- rf.error
  }

  # Put Error Vector into RF Data Frame to Measure Error Scores
  rf.perf.df <- cbind(rf.perf.df, rf_error_vec)

  # Run This Back Up Again for Five Folds, Data Frame Has Five Columns
}

# Clean Up Data Frame
rf.perf.df$Means <- rowMeans(rf.perf.df[,-c(1,2)])
colnames(rf.perf.df) <- c("Mtry Parameters", "Number of Trees", "Fold 1", "Fold 2",
                          "Fold 3", "Fold 4", "Fold 5", "FF CV Errors")
print(rf.perf.df)
```

```
##     Mtry Parameters Number of Trees    Fold 1    Fold 2    Fold 3    Fold 4
## 1                 1             100 0.4711538 0.4095238 0.4433962 0.4952381
## 2                 2             100 0.4711538 0.4857143 0.4528302 0.4380952
## 3                 3             100 0.4423077 0.4666667 0.4150943 0.4857143
## 4                 4             100 0.4423077 0.4571429 0.5000000 0.4857143
## 5                 5             100 0.4615385 0.4571429 0.4528302 0.4380952
## 6                 6             100 0.5192308 0.4380952 0.4811321 0.4761905
## 7                 7             100 0.4807692 0.4571429 0.4433962 0.4857143
## 8                 8             100 0.4615385 0.4571429 0.4150943 0.4285714
```

```
## 9             9              100 0.4326923 0.4666667 0.4716981 0.4857143
## 10           10              100 0.4615385 0.4380952 0.4433962 0.4952381
## 11           11              100 0.4519231 0.4761905 0.4811321 0.4952381
## 12           12              100 0.4807692 0.4666667 0.4716981 0.4761905
## 13           13              100 0.5096154 0.4666667 0.4622642 0.4380952
## 14           14              100 0.4903846 0.4761905 0.5000000 0.5047619
## 15           15              100 0.4615385 0.4380952 0.4528302 0.4952381
## 16           16              100 0.4230769 0.4761905 0.5188679 0.4380952
## 17           17              100 0.4326923 0.4952381 0.4433962 0.5333333
## 18           18              100 0.4711538 0.4761905 0.4716981 0.5142857
## 19           19              100 0.5096154 0.4666667 0.4811321 0.4571429
## 20           20              100 0.4807692 0.4380952 0.4150943 0.4666667
## 21           21              100 0.4903846 0.4761905 0.4056604 0.5142857
## 22           22              100 0.4615385 0.4380952 0.4716981 0.4190476
## 23            1              200 0.4519231 0.4380952 0.4433962 0.4476190
## 24            2              200 0.4134615 0.4380952 0.4622642 0.4761905
## 25            3              200 0.4615385 0.4476190 0.4905660 0.4571429
## 26            4              200 0.4615385 0.4285714 0.4339623 0.4571429
## 27            5              200 0.4615385 0.4476190 0.4433962 0.4952381
## 28            6              200 0.4615385 0.4380952 0.4811321 0.5238095
## 29            7              200 0.4423077 0.4095238 0.4716981 0.4571429
## 30            8              200 0.4134615 0.4476190 0.4622642 0.4761905
## 31            9              200 0.4519231 0.4285714 0.4716981 0.4571429
## 32           10              200 0.4423077 0.4380952 0.4433962 0.4857143
## 33           11              200 0.4711538 0.4952381 0.4811321 0.4952381
## 34           12              200 0.4326923 0.4666667 0.5000000 0.4857143
## 35           13              200 0.5192308 0.4761905 0.4339623 0.4571429
## 36           14              200 0.5096154 0.4666667 0.4528302 0.4666667
## 37           15              200 0.4519231 0.4476190 0.4716981 0.4190476
## 38           16              200 0.4519231 0.4857143 0.4339623 0.4761905
## 39           17              200 0.4807692 0.4761905 0.4245283 0.4761905
## 40           18              200 0.3942308 0.4666667 0.4245283 0.4571429
## 41           19              200 0.4711538 0.4761905 0.4905660 0.4476190
## 42           20              200 0.4711538 0.4476190 0.4528302 0.4571429
## 43           21              200 0.4711538 0.4666667 0.4905660 0.4666667
## 44           22              200 0.4807692 0.4380952 0.4716981 0.4476190
## 45            1              300 0.4519231 0.4476190 0.4716981 0.4380952
## 46            2              300 0.4326923 0.4190476 0.4339623 0.4571429
## 47            3              300 0.4519231 0.4666667 0.4622642 0.4761905
## 48            4              300 0.4711538 0.4571429 0.4811321 0.4666667
## 49            5              300 0.4807692 0.4380952 0.4150943 0.4761905
## 50            6              300 0.4615385 0.4380952 0.4528302 0.4761905
## 51            7              300 0.4615385 0.4285714 0.4339623 0.4571429
## 52            8              300 0.4615385 0.4095238 0.4339623 0.4761905
## 53            9              300 0.4807692 0.4761905 0.4150943 0.4285714
## 54           10              300 0.4807692 0.4857143 0.4433962 0.4571429
## 55           11              300 0.4615385 0.4666667 0.4433962 0.4761905
## 56           12              300 0.4711538 0.4666667 0.4716981 0.4476190
## 57           13              300 0.4038462 0.4666667 0.4056604 0.4761905
## 58           14              300 0.4807692 0.4857143 0.4433962 0.4952381
## 59           15              300 0.4326923 0.4476190 0.4905660 0.4761905
## 60           16              300 0.4615385 0.4857143 0.4528302 0.4476190
## 61           17              300 0.4423077 0.4285714 0.4433962 0.4571429
## 62           18              300 0.4519231 0.4857143 0.4433962 0.4952381
```

```
## 63                19                 300 0.4807692 0.4857143 0.4811321 0.5142857
## 64                20                 300 0.4423077 0.4666667 0.4622642 0.4761905
## 65                21                 300 0.4326923 0.4571429 0.5188679 0.4857143
## 66                22                 300 0.4615385 0.5047619 0.4622642 0.4857143
## 67                 1                 400 0.4903846 0.4000000 0.4622642 0.4285714
## 68                 2                 400 0.4038462 0.4095238 0.4433962 0.4857143
## 69                 3                 400 0.5000000 0.4666667 0.4150943 0.4666667
## 70                 4                 400 0.4711538 0.4095238 0.4245283 0.4380952
## 71                 5                 400 0.4903846 0.4476190 0.4622642 0.4571429
## 72                 6                 400 0.4711538 0.4571429 0.5000000 0.4666667
## 73                 7                 400 0.4326923 0.4571429 0.4433962 0.4761905
## 74                 8                 400 0.4519231 0.4476190 0.4433962 0.4857143
## 75                 9                 400 0.4519231 0.4571429 0.4528302 0.4952381
## 76                10                 400 0.4711538 0.4761905 0.5000000 0.4571429
## 77                11                 400 0.4423077 0.4571429 0.4716981 0.4857143
## 78                12                 400 0.4423077 0.4380952 0.4905660 0.4857143
## 79                13                 400 0.4038462 0.4476190 0.4433962 0.4761905
## 80                14                 400 0.4711538 0.4666667 0.4528302 0.5047619
## 81                15                 400 0.4807692 0.4761905 0.4716981 0.4857143
## 82                16                 400 0.4807692 0.4476190 0.4339623 0.4666667
## 83                17                 400 0.4134615 0.4952381 0.4811321 0.4476190
## 84                18                 400 0.4326923 0.4857143 0.4905660 0.4952381
## 85                19                 400 0.4519231 0.4857143 0.4528302 0.4571429
## 86                20                 400 0.4134615 0.4571429 0.4433962 0.4380952
## 87                21                 400 0.4519231 0.4190476 0.4716981 0.4666667
## 88                22                 400 0.4326923 0.4761905 0.4811321 0.4857143
## 89                 1                 500 0.4615385 0.4190476 0.4433962 0.4666667
## 90                 2                 500 0.4326923 0.4571429 0.4056604 0.4095238
## 91                 3                 500 0.4423077 0.4476190 0.4245283 0.4571429
## 92                 4                 500 0.4807692 0.4476190 0.4245283 0.4476190
## 93                 5                 500 0.4903846 0.4476190 0.4339623 0.4380952
## 94                 6                 500 0.4615385 0.4285714 0.4622642 0.4761905
## 95                 7                 500 0.4711538 0.4476190 0.4528302 0.4666667
## 96                 8                 500 0.4615385 0.4571429 0.4716981 0.4666667
## 97                 9                 500 0.4711538 0.4476190 0.4245283 0.4761905
## 98                10                 500 0.4615385 0.4666667 0.4433962 0.4761905
## 99                11                 500 0.5000000 0.4571429 0.4811321 0.4571429
## 100               12                 500 0.4423077 0.4571429 0.4433962 0.4952381
## 101               13                 500 0.4230769 0.4571429 0.4528302 0.4761905
## 102               14                 500 0.4230769 0.4285714 0.4622642 0.4761905
## 103               15                 500 0.5000000 0.4571429 0.4622642 0.4761905
## 104               16                 500 0.4423077 0.4761905 0.4811321 0.4857143
## 105               17                 500 0.5096154 0.4476190 0.4716981 0.4952381
## 106               18                 500 0.4615385 0.4761905 0.4716981 0.4857143
## 107               19                 500 0.4230769 0.4761905 0.4622642 0.4761905
## 108               20                 500 0.4615385 0.4666667 0.4433962 0.4952381
## 109               21                 500 0.5000000 0.4857143 0.4433962 0.4952381
## 110               22                 500 0.4615385 0.4666667 0.4339623 0.5142857
## 111                1                1000 0.4711538 0.4095238 0.4622642 0.4666667
## 112                2                1000 0.4519231 0.4380952 0.4622642 0.4857143
## 113                3                1000 0.4230769 0.4285714 0.4150943 0.4952381
## 114                4                1000 0.4615385 0.4190476 0.4622642 0.4857143
## 115                5                1000 0.4711538 0.4285714 0.4716981 0.4571429
## 116                6                1000 0.4615385 0.4285714 0.4622642 0.4571429
```

```
## 117               7            1000 0.4615385 0.4666667 0.4339623 0.4761905
## 118               8            1000 0.4326923 0.4476190 0.4622642 0.4857143
## 119               9            1000 0.4615385 0.4666667 0.4622642 0.5047619
## 120              10            1000 0.4807692 0.4666667 0.4811321 0.4571429
## 121              11            1000 0.4423077 0.4666667 0.4433962 0.4666667
## 122              12            1000 0.4615385 0.4666667 0.4528302 0.4857143
## 123              13            1000 0.4711538 0.4761905 0.4622642 0.4571429
## 124              14            1000 0.4326923 0.4666667 0.4528302 0.4857143
## 125              15            1000 0.4903846 0.4571429 0.4245283 0.4857143
## 126              16            1000 0.4423077 0.4761905 0.4905660 0.4571429
## 127              17            1000 0.5000000 0.4380952 0.4811321 0.4857143
## 128              18            1000 0.4903846 0.4666667 0.4622642 0.5047619
## 129              19            1000 0.4711538 0.4857143 0.4433962 0.4761905
## 130              20            1000 0.4326923 0.4571429 0.4811321 0.4761905
## 131              21            1000 0.4615385 0.4666667 0.4905660 0.4761905
## 132              22            1000 0.4711538 0.4857143 0.4905660 0.4952381
## 133               1            1500 0.4423077 0.4000000 0.4339623 0.4761905
## 134               2            1500 0.4423077 0.4190476 0.4716981 0.4952381
## 135               3            1500 0.4807692 0.4190476 0.4433962 0.4666667
## 136               4            1500 0.4519231 0.4285714 0.4433962 0.4761905
## 137               5            1500 0.4615385 0.4380952 0.4528302 0.4761905
## 138               6            1500 0.4423077 0.4476190 0.4245283 0.4761905
## 139               7            1500 0.4711538 0.4380952 0.4528302 0.4761905
## 140               8            1500 0.4615385 0.4380952 0.4339623 0.4761905
## 141               9            1500 0.4230769 0.4666667 0.4339623 0.4571429
## 142              10            1500 0.4711538 0.4761905 0.4622642 0.4857143
## 143              11            1500 0.4807692 0.4761905 0.5000000 0.4857143
## 144              12            1500 0.4711538 0.4571429 0.4716981 0.4857143
## 145              13            1500 0.4519231 0.4666667 0.4811321 0.4380952
## 146              14            1500 0.4615385 0.4666667 0.4622642 0.4761905
## 147              15            1500 0.4807692 0.4666667 0.4622642 0.4761905
## 148              16            1500 0.4615385 0.4761905 0.4905660 0.4761905
## 149              17            1500 0.4615385 0.4761905 0.4622642 0.4761905
## 150              18            1500 0.4615385 0.4761905 0.4433962 0.4666667
## 151              19            1500 0.4615385 0.4666667 0.4622642 0.4761905
## 152              20            1500 0.4711538 0.4761905 0.4716981 0.4666667
## 153              21            1500 0.4711538 0.4761905 0.5000000 0.4761905
## 154              22            1500 0.4711538 0.4761905 0.5000000 0.4666667
##        Fold 5 FF CV Errors
## 1   0.4095238    0.4457672
## 2   0.4095238    0.4514635
## 3   0.4285714    0.4476709
## 4   0.4380952    0.4646520
## 5   0.4571429    0.4533499
## 6   0.4095238    0.4648345
## 7   0.4476190    0.4629283
## 8   0.4476190    0.4419932
## 9   0.4285714    0.4570686
## 10  0.4285714    0.4533679
## 11  0.4000000    0.4608967
## 12  0.4000000    0.4590649
## 13  0.4857143    0.4724711
## 14  0.4476190    0.4837912
## 15  0.4761905    0.4647785
```

```
## 16  0.4380952    0.4588652
## 17  0.4285714    0.4666463
## 18  0.4857143    0.4838085
## 19  0.4857143    0.4800543
## 20  0.3904762    0.4382203
## 21  0.4380952    0.4649233
## 22  0.4476190    0.4475997
## 23  0.3428571    0.4247781
## 24  0.4190476    0.4418118
## 25  0.4476190    0.4608971
## 26  0.4190476    0.4400525
## 27  0.4095238    0.4514631
## 28  0.3809524    0.4571055
## 29  0.4285714    0.4418488
## 30  0.4285714    0.4456213
## 31  0.4000000    0.4418671
## 32  0.4190476    0.4457122
## 33  0.4190476    0.4723619
## 34  0.4380952    0.4646337
## 35  0.3714286    0.4515910
## 36  0.3619048    0.4515367
## 37  0.3904762    0.4361528
## 38  0.4095238    0.4514628
## 39  0.4380952    0.4591547
## 40  0.3714286    0.4227994
## 41  0.4380952    0.4647249
## 42  0.4476190    0.4552730
## 43  0.4285714    0.4647249
## 44  0.4285714    0.4533506
## 45  0.4380952    0.4494861
## 46  0.4095238    0.4304738
## 47  0.3523810    0.4418851
## 48  0.4380952    0.4628381
## 49  0.3523810    0.4325060
## 50  0.3714286    0.4400166
## 51  0.4380952    0.4438620
## 52  0.4190476    0.4400525
## 53  0.3809524    0.4363156
## 54  0.4095238    0.4553093
## 55  0.4380952    0.4571774
## 56  0.3904762    0.4495228
## 57  0.4190476    0.4342823
## 58  0.3904762    0.4591188
## 59  0.4380952    0.4570326
## 60  0.4380952    0.4571594
## 61  0.4571429    0.4457122
## 62  0.4095238    0.4571591
## 63  0.4190476    0.4761898
## 64  0.4666667    0.4628191
## 65  0.4190476    0.4626930
## 66  0.4571429    0.4742843
## 67  0.4095238    0.4381488
## 68  0.4190476    0.4323056
## 69  0.3809524    0.4458760
```

```
## 70  0.4000000    0.4286602
## 71  0.3619048    0.4438631
## 72  0.4571429    0.4704212
## 73  0.4190476    0.4456939
## 74  0.3809524    0.4419210
## 75  0.3904762    0.4495221
## 76  0.3714286    0.4551832
## 77  0.4190476    0.4551821
## 78  0.3809524    0.4475271
## 79  0.4000000    0.4342104
## 80  0.4285714    0.4647968
## 81  0.4095238    0.4647792
## 82  0.4285714    0.4515177
## 83  0.4380952    0.4551092
## 84  0.4285714    0.4665564
## 85  0.4666667    0.4628554
## 86  0.4095238    0.4323239
## 87  0.4380952    0.4494861
## 88  0.4285714    0.4608601
## 89  0.3809524    0.4343203
## 90  0.4285714    0.4267182
## 91  0.3904762    0.4324148
## 92  0.4000000    0.4401071
## 93  0.4000000    0.4420122
## 94  0.3904762    0.4438081
## 95  0.4095238    0.4495587
## 96  0.4380952    0.4590283
## 97  0.4285714    0.4496126
## 98  0.3904762    0.4476536
## 99  0.4095238    0.4609883
## 100 0.4285714    0.4533313
## 101 0.4285714    0.4475624
## 102 0.4095238    0.4399254
## 103 0.4000000    0.4591195
## 104 0.4095238    0.4589737
## 105 0.3904762    0.4629294
## 106 0.4476190    0.4685521
## 107 0.3904762    0.4456396
## 108 0.4666667    0.4667012
## 109 0.4476190    0.4743935
## 110 0.4190476    0.4591001
## 111 0.4190476    0.4457312
## 112 0.3809524    0.4437898
## 113 0.4000000    0.4323962
## 114 0.4095238    0.4476177
## 115 0.4095238    0.4476180
## 116 0.3904762    0.4399986
## 117 0.4000000    0.4476716
## 118 0.4000000    0.4456580
## 119 0.3904762    0.4571415
## 120 0.4285714    0.4628565
## 121 0.4285714    0.4495217
## 122 0.4380952    0.4609690
## 123 0.4095238    0.4552550
```

```
## 124 0.4380952    0.4551997
## 125 0.4095238    0.4534588
## 126 0.4380952    0.4608605
## 127 0.4095238    0.4628931
## 128 0.4285714    0.4705298
## 129 0.4190476    0.4591005
## 130 0.4571429    0.4608601
## 131 0.4666667    0.4723257
## 132 0.4095238    0.4704392
## 133 0.3904762    0.4285873
## 134 0.3904762    0.4437535
## 135 0.3904762    0.4400712
## 136 0.3809524    0.4362067
## 137 0.3904762    0.4438261
## 138 0.4285714    0.4438434
## 139 0.3809524    0.4438444
## 140 0.4190476    0.4457668
## 141 0.3809524    0.4323602
## 142 0.4190476    0.4628741
## 143 0.4095238    0.4704396
## 144 0.4095238    0.4590466
## 145 0.4190476    0.4513729
## 146 0.3904762    0.4514272
## 147 0.4190476    0.4609876
## 148 0.4095238    0.4628019
## 149 0.4000000    0.4552367
## 150 0.4190476    0.4533679
## 151 0.3904762    0.4514272
## 152 0.4190476    0.4609513
## 153 0.4095238    0.4666117
## 154 0.4095238    0.4647070
```

```
# Find Bagging Misclassification Errors
rf.perf.df[rf.perf.df$`Mtry Parameters` == 22,]
```

```
##     Mtry Parameters Number of Trees    Fold 1    Fold 2    Fold 3    Fold 4
## 22               22             100 0.4615385 0.4380952 0.4716981 0.4190476
## 44               22             200 0.4807692 0.4380952 0.4716981 0.4476190
## 66               22             300 0.4615385 0.5047619 0.4622642 0.4857143
## 88               22             400 0.4326923 0.4761905 0.4811321 0.4857143
## 110              22             500 0.4615385 0.4666667 0.4339623 0.5142857
## 132              22            1000 0.4711538 0.4857143 0.4905660 0.4952381
## 154              22            1500 0.4711538 0.4761905 0.5000000 0.4666667
##        Fold 5 FF CV Errors
## 22  0.4476190    0.4475997
## 44  0.4285714    0.4533506
## 66  0.4571429    0.4742843
## 88  0.4285714    0.4608601
## 110 0.4190476    0.4591001
## 132 0.4095238    0.4704392
## 154 0.4095238    0.4647070
```

```r
# Find Minimum Misclassification Error
rf_min_rn <- which.min(rf.perf.df$`FF CV Errors`)
rf.perf.df[rf_min_rn,]
```

```
##    Mtry Parameters Number of Trees    Fold 1    Fold 2    Fold 3    Fold 4
## 40              18             200 0.3942308 0.4666667 0.4245283 0.4571429
##       Fold 5 FF CV Errors
## 40 0.3714286    0.4227994
```

```r
### Additional Plots for Presentation ###

# Variable Importance
rf.mdl.pres.try <- randomForest(as.factor(Binary.PercChange) ~.,
                               data = gold.c, mtry = 18,
                               ntree = 200, importance = TRUE)
# varImpPlot(rf.mdl.pres, cex = 0.55, main = "Random Forest Variable Importance")

# Grid Plot of MSE Performance through Five Fold
classif.rf.plot <- ggplot(rf.perf.df, aes(x = `Mtry Parameters`, y = `Number of Trees`,size = `FF CV Er
  labs(x = "Number of Variables Sampled (Mtry)",
       title = "Classification Random Forest Performance")

# ggsave(file = "~/Desktop/classifrfplot.png", plot = classif.rf.plot, width = 10, height = 6, bg = "wh
```

Since our train control function was not working, we decided to run a manual parameter grid search for number of trees (ntrees) and number of features to consider for each tree (mtry). This like the manual knn parameter search is equivalent to testing the random forest model's performance on our five cross validation folds. Thus, the results from this grid search are comparable to other classificaiton models we ran. After running random forests and bagging algorithms on our data set, we find that a random forest with parameters (mtry = 18 and number of trees = 200) is the most optimal in minimizing misclassification errors. It returned a misclassification error of **42.28%** which is slightly higher than our KNN model but still less than 50%. This shows that our model performs better than random chance when it comes to classifying the direction of gold prices month over month, given our 0.1% hurdle rate. Looking at our bagging algorithms (mtry = 22, where we use all predictors in our feature space), their misclassification errors, on average, fell in the range **44.76%-47.42%**. The bagging algorithms were not optimal in our classification setting.

Boosting

```r
# Conduct Parameter Search for Gradient Boosting Model Using Train Function
set.seed(75849)
ctrl <- trainControl(method = "cv", number = 5) # Five Fold CV

# Create Parameter Grid
parameters.gbm.c <- expand.grid(n.trees = c(100,200,500,1000,1500),
                                interaction.depth = c(5,10,15,20,50,100),
                                shrinkage = c(0.001,0.01,0.05,0.075,0.1),
                                n.minobsinnode = c(5,10,15,20))

# Fit A Boosting GBM Model for Classification
features.c <- gold.c[, !colnames(gold.c) %in% "Binary.PercChange"]
```

```r
# trained.gbm.fit.c <- train(x = features.c, y = as.factor(gold.c$Binary.PercChange),
#                            method = "gbm", trControl = ctrl,
#                            tuneGrid = parameters.gbm.c)

# trained.gbm.fit.c$bestTune

### Train Function Results ###
# Number of Trees: 200
# Interaction Depth: 10
# Shrinkage: 0.075
# Minimum Number of Observations: 15
```

After running a parameter search on our data using the train function, the best tuning parameters for our classification gradient boosting model are 200 for the number of trees, 10 for the interaction depth, 0.075 for the shrinkage parameter, and 15 for the minimum number of observations in a node. We'll use these parameters to calculate the five fold cross validation misclassification error. This will allow us to compare model performance.

```r
# Set Misclassification Error Vector
gbm.c.error <- vector()

# Five Fold Cross Validation with Best Parameters
for (i in 1:length(folds)){
  # Set Folds
  train_index <- folds[[i]]
  gold.c.train <- gold.c[train_index,]
  gold.c.test <- gold.c[-train_index,]

  # Fit the Model with Optimal Parameters Listed Above
  gbm.mdl.c <- gbm(Binary.PercChange ~., data = gold.c.train,
                   distribution = "bernoulli", shrinkage = 0.075,
                   n.trees = 200, interaction.depth = 10,
                   n.minobsinnode = 15)

  # Predict the Training Model on Testing Data
  gbm.predict <- predict(gbm.mdl.c, newdata = gold.c.test, type = "response")

  # Convert Response Prediction to Classifier
  gbm.class <- ifelse(gbm.predict > 0.5, 1,0)

  # Accuracy Scores
  accuracy <- mean(gbm.class == gold.c.test$Binary.PercChange)

  # Error Scores & Input
  err <- 1-accuracy
  gbm.c.error <- append(gbm.c.error,err)
}
```

```
## Using 200 trees...
##
## Using 200 trees...
##
## Using 200 trees...
```

```
##
## Using 200 trees...
##
## Using 200 trees...
```

```
# Print Error Vector & Retrieve Five Fold Cross Validation Mean
print(gbm.c.error)
```

```
## [1] 0.3846154 0.4285714 0.4622642 0.4952381 0.4476190
```

```
mean(gbm.c.error)
```

```
## [1] 0.4436616
```

After running five fold cross validation on our classification gradient boosting model, we received a mean misclassification error of **43.06%**. This is better than our classification random forest model, but it does not perform as well as our KNN model. Regardless, our misclassification error is below 50% and is better than random chance.

Support Vector Machine

For support vector machines, there are two main subsets of models that we'd like to explore in our classification task. The first is running support vector machines with polynomial kernels, and the second is running support vector machines with radial kernels. When running support vector machines with polynomial kernels, we'll vary for cost and degree - the latter represents the maximum polynomial transformation of our predictors. For support vector machines with radial kernels, we'll vary for cost and gamma. We'll use the tune function to run a parameter grid search for cost and gamma/degree. If the tune function does not work, we'll run a for loop to run the grid search. In order to compare our support vector machine results to other classification models, we'll run five fold cross-validation.

```
### Polynomial Kernel: Tune Function ###
library(e1071)

# Create Polynomial Grid: First Column is Cost, Second Column is Degree
svm.param.c.poly <- expand.grid(cost = c(0.1,0.5,1,2,5,8,10,100),
                                degree = c(2,3,4,5))

# Create Data Frame to Store Misclassification Errors
svm.poly.df <- data.frame(svm.param.c.poly)

# For Loop Grid Search Using Five Fold Cross Validation
for (i in 1:length(folds)){
  # Set Folds
  train_index <- folds[[i]]
  gold.c.train <- gold.c[train_index,]
  gold.c.test <- gold.c[-train_index,]

  # Set Error Vector to Input Misclassification Errors for Each Fold
  svm.error.vec <- vector()

  # Now Begin Fitting Model with Each Fold, Varying Parameters
```

```
  for (j in 1:nrow(svm.poly.df)){

    # Fit Model utih Varying Parameters: First Column = Cost, Second = Degree
    svm.mdl <- svm(as.factor(Binary.PercChange) ~., data = gold.c.train,
                   cost = svm.poly.df[j,1], degree = svm.poly.df[j,2])

    # Predictions of Class (Binary Classifier)
    svm.predictions <- predict(svm.mdl, newdata = gold.c.test)

    # Accuracy Score
    svm.accuracy <- mean(svm.predictions == gold.c.test$Binary.PercChange)

    # Error Score & Input Into Vector
    svm.error <- 1-svm.accuracy
    svm.error.vec[j] <- svm.error

    # Input Error into Data Frame in Outermost (Fold) Loop
  }

  # Input Error Vector into Data Frame
  svm.poly.df <- cbind(svm.poly.df, svm.error.vec)

  # Error Vector Gets Washed Out At Start of New Loop
}

# Clean Data Frame
svm.poly.df$Means <- rowMeans(svm.poly.df[,-c(1,2)])
colnames(svm.poly.df) <- c("Cost Parameter", "Degree Parameter","Fold 1",
                           "Fold 2", "Fold 3", "Fold 4", "Fold 5",
                           "Average Error")
print(svm.poly.df)
```

```
##    Cost Parameter Degree Parameter    Fold 1    Fold 2    Fold 3    Fold 4
## 1             0.1                2 0.4903846 0.4952381 0.4339623 0.4857143
## 2             0.5                2 0.4519231 0.4380952 0.5094340 0.4095238
## 3             1.0                2 0.4615385 0.4571429 0.4811321 0.4476190
## 4             2.0                2 0.4903846 0.4476190 0.4433962 0.4000000
## 5             5.0                2 0.4326923 0.4380952 0.4433962 0.4190476
## 6             8.0                2 0.4615385 0.4190476 0.4622642 0.4666667
## 7            10.0                2 0.4134615 0.4285714 0.4433962 0.4761905
## 8           100.0                2 0.4038462 0.4095238 0.3867925 0.4476190
## 9             0.1                3 0.4903846 0.4952381 0.4339623 0.4857143
## 10            0.5                3 0.4519231 0.4380952 0.5094340 0.4095238
## 11            1.0                3 0.4615385 0.4571429 0.4811321 0.4476190
## 12            2.0                3 0.4903846 0.4476190 0.4433962 0.4000000
## 13            5.0                3 0.4326923 0.4380952 0.4433962 0.4190476
## 14            8.0                3 0.4615385 0.4190476 0.4622642 0.4666667
## 15           10.0                3 0.4134615 0.4285714 0.4433962 0.4761905
## 16          100.0                3 0.4038462 0.4095238 0.3867925 0.4476190
## 17            0.1                4 0.4903846 0.4952381 0.4339623 0.4857143
## 18            0.5                4 0.4519231 0.4380952 0.5094340 0.4095238
## 19            1.0                4 0.4615385 0.4571429 0.4811321 0.4476190
## 20            2.0                4 0.4903846 0.4476190 0.4433962 0.4000000
```

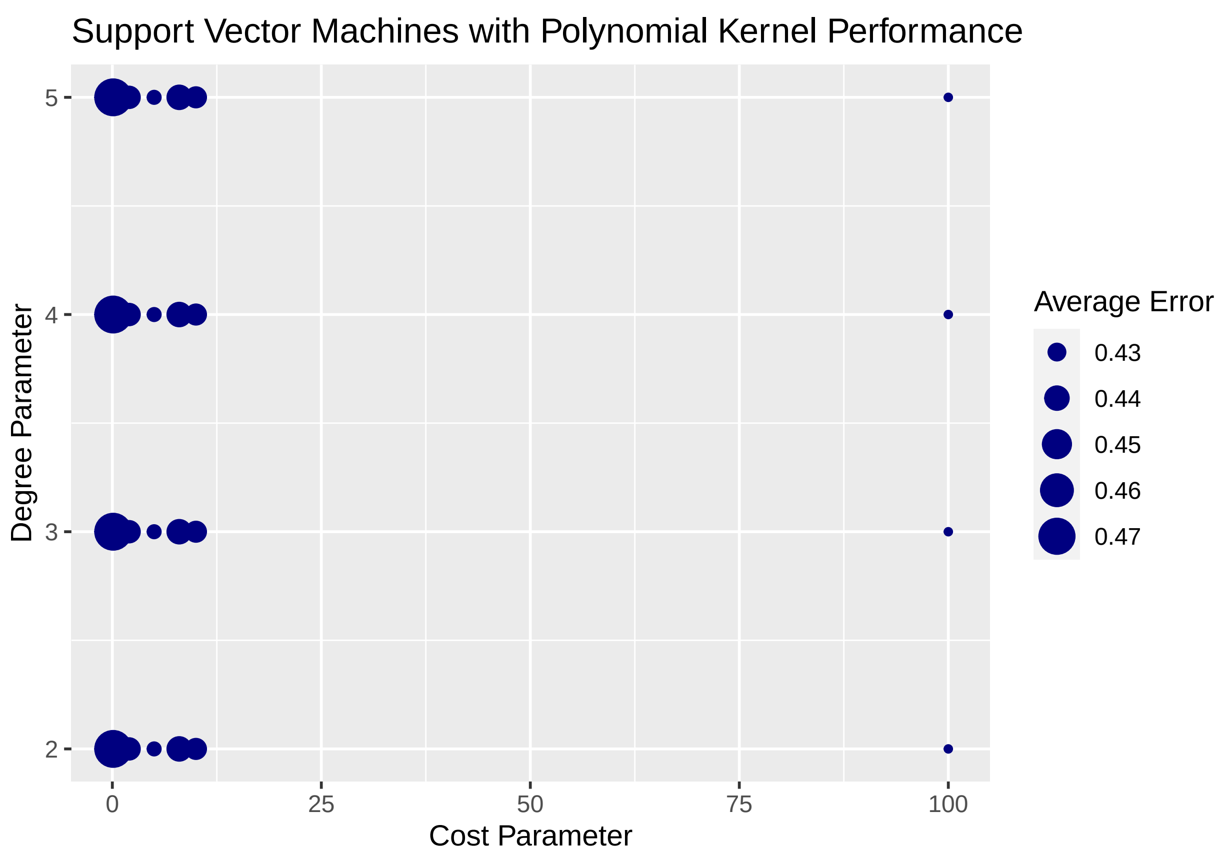
```
## 21            5.0                4 0.4326923 0.4380952 0.4433962 0.4190476
## 22            8.0                4 0.4615385 0.4190476 0.4622642 0.4666667
## 23           10.0                4 0.4134615 0.4285714 0.4433962 0.4761905
## 24          100.0                4 0.4038462 0.4095238 0.3867925 0.4476190
## 25            0.1                5 0.4903846 0.4952381 0.4339623 0.4857143
## 26            0.5                5 0.4519231 0.4380952 0.5094340 0.4095238
## 27            1.0                5 0.4615385 0.4571429 0.4811321 0.4476190
## 28            2.0                5 0.4903846 0.4476190 0.4433962 0.4000000
## 29            5.0                5 0.4326923 0.4380952 0.4433962 0.4190476
## 30            8.0                5 0.4615385 0.4190476 0.4622642 0.4666667
## 31           10.0                5 0.4134615 0.4285714 0.4433962 0.4761905
## 32          100.0                5 0.4038462 0.4095238 0.3867925 0.4476190
##       Fold 5 Average Error
## 1  0.4571429     0.4724884
## 2  0.4000000     0.4417952
## 3  0.3714286     0.4437722
## 4  0.4000000     0.4362800
## 5  0.4000000     0.4266463
## 6  0.3904762     0.4399986
## 7  0.4095238     0.4342287
## 8  0.4761905     0.4247944
## 9  0.4571429     0.4724884
## 10 0.4000000     0.4417952
## 11 0.3714286     0.4437722
## 12 0.4000000     0.4362800
## 13 0.4000000     0.4266463
## 14 0.3904762     0.4399986
## 15 0.4095238     0.4342287
## 16 0.4761905     0.4247944
## 17 0.4571429     0.4724884
## 18 0.4000000     0.4417952
## 19 0.3714286     0.4437722
## 20 0.4000000     0.4362800
## 21 0.4000000     0.4266463
## 22 0.3904762     0.4399986
## 23 0.4095238     0.4342287
## 24 0.4761905     0.4247944
## 25 0.4571429     0.4724884
## 26 0.4000000     0.4417952
## 27 0.3714286     0.4437722
## 28 0.4000000     0.4362800
## 29 0.4000000     0.4266463
## 30 0.3904762     0.4399986
## 31 0.4095238     0.4342287
## 32 0.4761905     0.4247944
```

```r
# Find Minimized Misclassification Error and Optimal Parameters for Polynomial Kernel
svm.poly.min <- which(svm.poly.df$`Average Error` == min(svm.poly.df$`Average Error`))
print(svm.poly.df[svm.poly.min,])
```

```
##    Cost Parameter Degree Parameter    Fold 1    Fold 2    Fold 3   Fold 4
## 8             100                2 0.4038462 0.4095238 0.3867925 0.447619
## 16            100                3 0.4038462 0.4095238 0.3867925 0.447619
## 24            100                4 0.4038462 0.4095238 0.3867925 0.447619
```

```
## 32            100                5 0.4038462 0.4095238 0.3867925 0.447619
##       Fold 5 Average Error
## 8  0.4761905     0.4247944
## 16 0.4761905     0.4247944
## 24 0.4761905     0.4247944
## 32 0.4761905     0.4247944
```

```
### Additional Plots for Presentation ###

# Grid Search Plot
svm.plot.poly <- ggplot(svm.poly.df, aes(x = `Cost Parameter`, y = `Degree Parameter`,
                       size = `Average Error`))+
  geom_point(col = "navy")+labs(x = "Cost Parameter", y = "Degree Parameter",
      title = "Support Vector Machines with Polynomial Kernel Performance")
svm.plot.poly
```




```
# ggsave(file = "~/Desktop/svmplotpoly.png", plot = svm.plot.poly, width = 10, height = 6, bg = "white")

print(svm.poly.df)
```

```
##    Cost Parameter Degree Parameter    Fold 1    Fold 2    Fold 3    Fold 4
## 1             0.1                2 0.4903846 0.4952381 0.4339623 0.4857143
## 2             0.5                2 0.4519231 0.4380952 0.5094340 0.4095238
## 3             1.0                2 0.4615385 0.4571429 0.4811321 0.4476190
## 4             2.0                2 0.4903846 0.4476190 0.4433962 0.4000000
```

```
## 5            5.0                    2 0.4326923 0.4380952 0.4433962 0.4190476
## 6            8.0                    2 0.4615385 0.4190476 0.4622642 0.4666667
## 7           10.0                    2 0.4134615 0.4285714 0.4433962 0.4761905
## 8          100.0                    2 0.4038462 0.4095238 0.3867925 0.4476190
## 9            0.1                    3 0.4903846 0.4952381 0.4339623 0.4857143
## 10           0.5                    3 0.4519231 0.4380952 0.5094340 0.4095238
## 11           1.0                    3 0.4615385 0.4571429 0.4811321 0.4476190
## 12           2.0                    3 0.4903846 0.4476190 0.4433962 0.4000000
## 13           5.0                    3 0.4326923 0.4380952 0.4433962 0.4190476
## 14           8.0                    3 0.4615385 0.4190476 0.4622642 0.4666667
## 15          10.0                    3 0.4134615 0.4285714 0.4433962 0.4761905
## 16         100.0                    3 0.4038462 0.4095238 0.3867925 0.4476190
## 17           0.1                    4 0.4903846 0.4952381 0.4339623 0.4857143
## 18           0.5                    4 0.4519231 0.4380952 0.5094340 0.4095238
## 19           1.0                    4 0.4615385 0.4571429 0.4811321 0.4476190
## 20           2.0                    4 0.4903846 0.4476190 0.4433962 0.4000000
## 21           5.0                    4 0.4326923 0.4380952 0.4433962 0.4190476
## 22           8.0                    4 0.4615385 0.4190476 0.4622642 0.4666667
## 23          10.0                    4 0.4134615 0.4285714 0.4433962 0.4761905
## 24         100.0                    4 0.4038462 0.4095238 0.3867925 0.4476190
## 25           0.1                    5 0.4903846 0.4952381 0.4339623 0.4857143
## 26           0.5                    5 0.4519231 0.4380952 0.5094340 0.4095238
## 27           1.0                    5 0.4615385 0.4571429 0.4811321 0.4476190
## 28           2.0                    5 0.4903846 0.4476190 0.4433962 0.4000000
## 29           5.0                    5 0.4326923 0.4380952 0.4433962 0.4190476
## 30           8.0                    5 0.4615385 0.4190476 0.4622642 0.4666667
## 31          10.0                    5 0.4134615 0.4285714 0.4433962 0.4761905
## 32         100.0                    5 0.4038462 0.4095238 0.3867925 0.4476190
##       Fold 5 Average Error
## 1  0.4571429     0.4724884
## 2  0.4000000     0.4417952
## 3  0.3714286     0.4437722
## 4  0.4000000     0.4362800
## 5  0.4000000     0.4266463
## 6  0.3904762     0.4399986
## 7  0.4095238     0.4342287
## 8  0.4761905     0.4247944
## 9  0.4571429     0.4724884
## 10 0.4000000     0.4417952
## 11 0.3714286     0.4437722
## 12 0.4000000     0.4362800
## 13 0.4000000     0.4266463
## 14 0.3904762     0.4399986
## 15 0.4095238     0.4342287
## 16 0.4761905     0.4247944
## 17 0.4571429     0.4724884
## 18 0.4000000     0.4417952
## 19 0.3714286     0.4437722
## 20 0.4000000     0.4362800
## 21 0.4000000     0.4266463
## 22 0.3904762     0.4399986
## 23 0.4095238     0.4342287
## 24 0.4761905     0.4247944
## 25 0.4571429     0.4724884
```

```
## 26 0.4000000      0.4417952
## 27 0.3714286      0.4437722
## 28 0.4000000      0.4362800
## 29 0.4000000      0.4266463
## 30 0.3904762      0.4399986
## 31 0.4095238      0.4342287
## 32 0.4761905      0.4247944
```

We performed a grid search for the most optimal cost and degree parameters within polynomial kernels and found that the most optimal parameters are a cost value of 100 and a degree value of 2-5. The cost parameter of 100 seemed to be very good at minimizing misclassification error, regardless of polynomial degree. After running five fold cross validation with our grid search, we find that a cost parameter of 100 returned a misclassification error of **42.48%**. The support vector machine model with polynomial kernel outperforms our tree models (gradient boosting, random forests, and bagging) but still underperforms our KNN model.

```
### Radial Kernel: Grid Search ###

# Create Radial Grid: First Column is Cost, Second Column is Degree
svm.param.c.radial <- expand.grid(cost = c(0.1,0.5,1,2,5,8,10),
                                  gamma = c(0.1,0.5,1,2,5,8,10))

# Create Data Frame to Store Misclassification Errors
svm.radial.df <- data.frame(svm.param.c.radial)

# For Loop Grid Search Using Five Fold Cross Validation
for (i in 1:length(folds)){
  # Set Folds
  train_index <- folds[[i]]
  gold.c.train <- gold.c[train_index,]
  gold.c.test <- gold.c[-train_index,]

  # Set Error Vector to Input Misclassification Errors for Each Fold
  svm.error.vec <- vector()

  # Now Begin Fitting Model with Each Fold, Varying Parameters
  for (j in 1:nrow(svm.radial.df)){

    # Fit Model wtih Varying Parameters: First Column = Cost, Second = Degree
    svm.mdl <- svm(as.factor(Binary.PercChange) ~., data = gold.c.train,
                   kernel = "radial",
                   cost = svm.radial.df[j,1], gamma = svm.radial.df[j,2])

    # Predictions of Class (Binary Classifier)
    svm.predictions <- predict(svm.mdl, newdata = gold.c.test)

    # Accuracy Score
    svm.accuracy <- mean(svm.predictions == gold.c.test$Binary.PercChange)

    # Error Score & Input Into Vector
    svm.error <- 1-svm.accuracy
    svm.error.vec[j] <- svm.error

    # Input Error into Data Frame in Outermost (Fold) Loop
```

```r
  }

  # Input Error Vector into Data Frame
  svm.radial.df <- cbind(svm.radial.df, svm.error.vec)

  # Error Vector Gets Washed Out At Start of New Loop
}

# Clean Data Frame
svm.radial.df$Means <- rowMeans(svm.radial.df[,-c(1,2)])
colnames(svm.radial.df) <- c("Cost Parameter", "Gamma Parameter","Fold 1",
                             "Fold 2", "Fold 3", "Fold 4", "Fold 5",
                             "Average Error")
print(svm.radial.df)
```

```
##    Cost Parameter Gamma Parameter    Fold 1    Fold 2    Fold 3    Fold 4
## 1             0.1             0.1 0.4903846 0.4952381 0.4622642 0.4857143
## 2             0.5             0.1 0.4711538 0.4285714 0.4528302 0.4571429
## 3             1.0             0.1 0.4423077 0.4285714 0.4622642 0.4095238
## 4             2.0             0.1 0.4134615 0.4380952 0.4528302 0.4571429
## 5             5.0             0.1 0.4038462 0.4666667 0.4245283 0.4857143
## 6             8.0             0.1 0.4038462 0.4190476 0.4433962 0.4476190
## 7            10.0             0.1 0.4038462 0.4285714 0.4339623 0.4666667
## 8             0.1             0.5 0.4903846 0.4952381 0.4622642 0.4857143
## 9             0.5             0.5 0.4903846 0.4952381 0.4622642 0.4857143
## 10            1.0             0.5 0.4519231 0.4666667 0.3962264 0.4190476
## 11            2.0             0.5 0.4615385 0.4857143 0.4056604 0.4476190
## 12            5.0             0.5 0.4615385 0.4761905 0.4150943 0.4380952
## 13            8.0             0.5 0.4615385 0.4761905 0.4150943 0.4380952
## 14           10.0             0.5 0.4615385 0.4761905 0.4150943 0.4380952
## 15            0.1             1.0 0.4903846 0.4952381 0.4622642 0.4857143
## 16            0.5             1.0 0.4903846 0.4952381 0.4622642 0.4857143
## 17            1.0             1.0 0.4807692 0.4666667 0.4433962 0.4857143
## 18            2.0             1.0 0.4903846 0.5047619 0.3962264 0.4666667
## 19            5.0             1.0 0.4903846 0.5047619 0.3962264 0.4666667
## 20            8.0             1.0 0.4903846 0.5047619 0.3962264 0.4666667
## 21           10.0             1.0 0.4903846 0.5047619 0.3962264 0.4666667
## 22            0.1             2.0 0.4903846 0.4952381 0.4622642 0.4857143
## 23            0.5             2.0 0.4903846 0.4952381 0.4622642 0.4857143
## 24            1.0             2.0 0.4903846 0.4952381 0.4622642 0.4857143
## 25            2.0             2.0 0.5000000 0.4952381 0.4622642 0.4857143
## 26            5.0             2.0 0.5000000 0.4952381 0.4622642 0.4857143
## 27            8.0             2.0 0.5000000 0.4952381 0.4622642 0.4857143
## 28           10.0             2.0 0.5000000 0.4952381 0.4622642 0.4857143
## 29            0.1             5.0 0.4903846 0.4952381 0.4622642 0.4857143
## 30            0.5             5.0 0.4903846 0.4952381 0.4622642 0.4857143
## 31            1.0             5.0 0.4903846 0.4952381 0.4622642 0.4857143
## 32            2.0             5.0 0.4903846 0.4952381 0.4622642 0.4857143
## 33            5.0             5.0 0.4903846 0.4952381 0.4622642 0.4857143
## 34            8.0             5.0 0.4903846 0.4952381 0.4622642 0.4857143
## 35           10.0             5.0 0.4903846 0.4952381 0.4622642 0.4857143
## 36            0.1             8.0 0.4903846 0.4952381 0.4622642 0.4857143
## 37            0.5             8.0 0.4903846 0.4952381 0.4622642 0.4857143
```

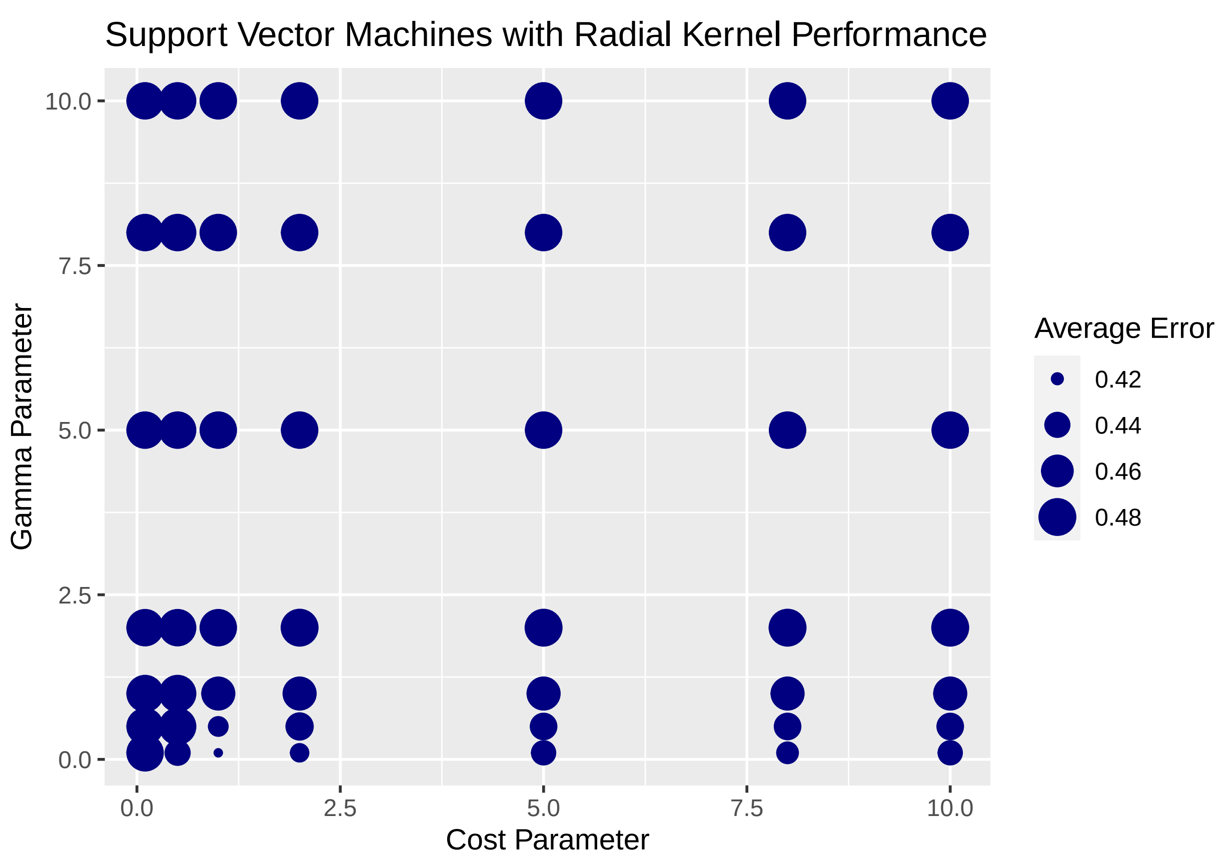
```
## 38           1.0            8.0 0.4903846 0.4952381 0.4622642 0.4857143
## 39           2.0            8.0 0.4903846 0.4952381 0.4622642 0.4857143
## 40           5.0            8.0 0.4903846 0.4952381 0.4622642 0.4857143
## 41           8.0            8.0 0.4903846 0.4952381 0.4622642 0.4857143
## 42          10.0            8.0 0.4903846 0.4952381 0.4622642 0.4857143
## 43           0.1           10.0 0.4903846 0.4952381 0.4622642 0.4857143
## 44           0.5           10.0 0.4903846 0.4952381 0.4622642 0.4857143
## 45           1.0           10.0 0.4903846 0.4952381 0.4622642 0.4857143
## 46           2.0           10.0 0.4903846 0.4952381 0.4622642 0.4857143
## 47           5.0           10.0 0.4903846 0.4952381 0.4622642 0.4857143
## 48           8.0           10.0 0.4903846 0.4952381 0.4622642 0.4857143
## 49          10.0           10.0 0.4903846 0.4952381 0.4622642 0.4857143
##       Fold 5 Average Error
## 1  0.4571429     0.4781488
## 2  0.3904762     0.4400349
## 3  0.3523810     0.4190096
## 4  0.3714286     0.4265917
## 5  0.4095238     0.4380558
## 6  0.4476190     0.4323056
## 7  0.4571429     0.4380379
## 8  0.4571429     0.4781488
## 9  0.4571429     0.4781488
## 10 0.4095238     0.4286775
## 11 0.4285714     0.4458207
## 12 0.4285714     0.4438980
## 13 0.4285714     0.4438980
## 14 0.4285714     0.4438980
## 15 0.4571429     0.4781488
## 16 0.4571429     0.4781488
## 17 0.4476190     0.4648331
## 18 0.4666667     0.4649413
## 19 0.4666667     0.4649413
## 20 0.4666667     0.4649413
## 21 0.4666667     0.4649413
## 22 0.4571429     0.4781488
## 23 0.4571429     0.4781488
## 24 0.4571429     0.4781488
## 25 0.4571429     0.4800719
## 26 0.4571429     0.4800719
## 27 0.4571429     0.4800719
## 28 0.4571429     0.4800719
## 29 0.4571429     0.4781488
## 30 0.4571429     0.4781488
## 31 0.4571429     0.4781488
## 32 0.4571429     0.4781488
## 33 0.4571429     0.4781488
## 34 0.4571429     0.4781488
## 35 0.4571429     0.4781488
## 36 0.4571429     0.4781488
## 37 0.4571429     0.4781488
## 38 0.4571429     0.4781488
## 39 0.4571429     0.4781488
## 40 0.4571429     0.4781488
## 41 0.4571429     0.4781488
```

```
## 42 0.4571429      0.4781488
## 43 0.4571429      0.4781488
## 44 0.4571429      0.4781488
## 45 0.4571429      0.4781488
## 46 0.4571429      0.4781488
## 47 0.4571429      0.4781488
## 48 0.4571429      0.4781488
## 49 0.4571429      0.4781488
```

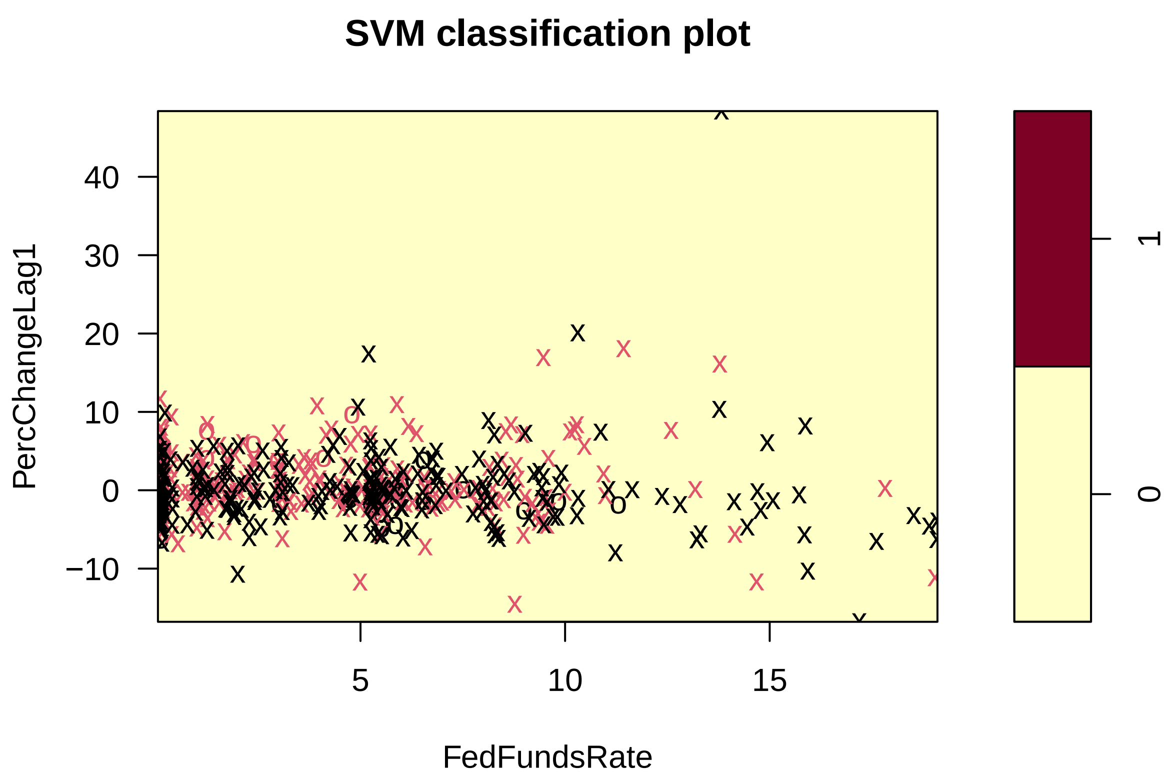
```
# Find Minimized Misclassification Error and Optimal Parameters for Polynomial Kernel
svm.radial.min <- which(svm.radial.df$`Average Error` == min(svm.radial.df$`Average Error`))
print(svm.radial.df[svm.radial.min,])
```

```
##   Cost Parameter Gamma Parameter    Fold 1    Fold 2    Fold 3    Fold 4
## 3              1             0.1 0.4423077 0.4285714 0.4622642 0.4095238
##     Fold 5 Average Error
## 3 0.352381     0.4190096
```

```
# SVM Plots: Maybe Don't Use This
svm.plot.radial <- ggplot(svm.radial.df, aes(x = `Cost Parameter`, y = `Gamma Parameter`,
                       size = `Average Error`))+
  geom_point(col = "navy")+labs(x = "Cost Parameter", y = "Gamma Parameter",
      title = "Support Vector Machines with Radial Kernel Performance")

svm.plot.radial
```



```r
# Save Plot
# ggsave(file = "~/Desktop/svmplotradial.png", plot = svm.plot.radial, width = 10, height = 6, bg = "wh

svm.radial.full <- svm(as.factor(Binary.PercChange) ~., data = gold.c,
                       kernel = "radial", cost = 1, gamma = 0.1)

plot(svm.radial.full, gold.c, PercChangeLag1 ~ FedFundsRate,
     xlab = "Fed Funds Rate (Monthly Average)", ylab = "Gold Price Lagged Percent Change")
```



After performing a grid search on cost and gamma parameters for our support vector machine with radial kernels, we find that the most optimal parameter values are a cost parameter of 1 and a gamma parameter of 0.1. These parameter values return a misclassification error of **41.90%**. This edges our polynomial kernels, but it still slightly underperforms our KNN model - though it is the closest among the classification models to the KNN model.

Neural Network

Disclaimer: The neural network models ran on one of our markdown files, but can't on this one. In our final presentation, we added our findings from our neural network model. We won't run it in this document because it won't knit.

```r
# #py_install("tensorflow")
# library(keras)
# #reticulate::install_miniconda()
# reticulate::py_install("tensorflow")
```

```
# reticulate::py_install("keras")
#
# # Define the parameter grid
# layers_grid <- c(1, 2, 3)  # Different numbers of layers
# neurons_grid <- c(32, 64, 128)  # Different numbers of neurons
#
# # Initialize variables to store the best configuration and accuracy
# best_layers <- NULL
# best_neurons <- NULL
# best_accuracy <- 0
#
# # Perform grid search
# for (layers in layers_grid) {
#   for (neurons in neurons_grid) {
#     # Create the sequential model
#     model <- keras_model_sequential()
#     model %>%
#       layer_dense(units = neurons, activation = "relu", input_shape = ncol(gold.c) - 1)
#     for (i in seq(layers - 1)) {
#       model %>%
#         layer_dense(units = neurons, activation = "relu")
#     }
#     model %>%
#       layer_dense(units = 1, activation = "sigmoid")
#
#     # Compile the model
#     model %>% compile(
#       loss = "binary_crossentropy",
#       optimizer = "adam",
#       metrics = c("accuracy")
#     )
#
#     # Train the model
#     history <- model %>% fit(
#       x = as.matrix(gold.c[, -ncol(gold.c)]),
#       y = as.matrix(gold.c$Binary.PercChange),
#       epochs = 10,
#       batch_size = 32,
#       validation_split = 0.2
#     )
#
#     # Calculate the accuracy
#     accuracy <- history$metrics$val_accuracy[length(history$metrics$val_accuracy)]
#
#     # Check if the current configuration is the best so far
#     if (accuracy > best_accuracy) {
#       best_layers <- layers
#       best_neurons <- neurons
#       best_accuracy <- accuracy
#       best_history <- history
#
#     }
#   }
```

```
# }
#
# # Print the best configuration and accuracy
# print(paste("Best Layers:", best_layers))
# print(paste("Best Neurons:", best_neurons))
# print(paste("Best Accuracy:", best_accuracy))
#
# # Plot the training history of the best model
# plot(best_history$metrics$accuracy, type = "l", col = "blue", xlab = "Epoch", ylab = "Accuracy",
#      main = "Training History - Best Model")
# lines(best_history$metrics$val_accuracy, col = "red")
# legend("bottomright", legend = c("Training Accuracy", "Validation Accuracy"), col = c("blue", "red"),
#
#
# library(pROC)
#
# # Predict probabilities
# y_pred <- model %>% predict(as.matrix(gold.c[, -ncol(gold.c)]))
#
# # Compute ROC curve
# roc_data <- roc(gold.c$Binary.PercChange, y_pred)
#
# # Plot ROC curve
# plot(roc_data, main = "ROC Curve", print.auc = TRUE)
```

ROC Graph for Classification Models

```
# Data Frame for Predicted Probability Values & Actual Values
empty_vector <- 1:nrow(gold.c)
roc.df <- data.frame(empty_vector)

# Vectors for Predicted Models/Actual observations: Run This Before Each Fold
gold.c.classes <- vector()
lda.pred.vector <- vector()
qda.pred.vector <- vector()
knn.pred.vector <- vector()
rf.pred.vector <- vector()
boost.pred.vector <- vector()
svm.pred.vector <- vector()

# Fitting the Models with Five Folds
for (i in 1:length(folds)){
  # Set Folds
  train_index <- folds[[i]]
  gold.c.train <- gold.c[train_index,]
  gold.c.test <- gold.c[-train_index,]

  # Append Actual Observations to a Vector
  gold.c.classes <- append(gold.c.classes, gold.c.test$Binary.PercChange)

  # Set Inputs for KNN Function With Each Fold
  train_features <- gold.c.train[,-length(gold.c.train)]
```

```r
  test_features <- gold.c.test[,-length(gold.c.train)]
  train_class <- gold.c.train$Binary.PercChange
  qdaFeatures <- c("PercChangeLag1", "Indus.Prod.Ind", "FedFundsRate")

  # Fit Various Models to Folds
  knn.mdl <- knn(train_features, test_features, train_class, k = 13, prob = TRUE)
  rf.model <- randomForest(as.factor(Binary.PercChange) ~.,
                           data = gold.c.train, mtry = 18,
                           ntree = 200)
  gbm.mdl.c <- gbm(Binary.PercChange ~., data = gold.c.train,
                  distribution = "bernoulli", shrinkage = 0.075,
                  n.trees = 200, interaction.depth = 10,
                  n.minobsinnode = 15)
  svm.mdl <- svm(as.factor(Binary.PercChange) ~., data = gold.c.train,
                kernel = "radial", cost = 1, gamma = 0.1, prob = TRUE)
  ldaModel <- lda(as.factor(Binary.PercChange) ~ ., data = gold.c.train)
  qdaModel <- qda(as.factor(Binary.PercChange) ~ ., data =
                   gold.c.train[,c(qdaFeatures, "Binary.PercChange")])


  # Predict on Various Models
  knn.pred <- attr(knn.mdl, "prob")
  rf.pred <- predict(rf.model, gold.c.test, type = "prob")
  gbm.pred <- predict(gbm.mdl.c, gold.c.test, type = "response")
  svm.pred <- predict(svm.mdl, gold.c.test, prob = TRUE)
  ldaPredictions <- predict(ldaModel, newdata = gold.c.test)$posterior[,2]
  qdaPredictions <- predict(qdaModel, newdata = gold.c.test[, qdaFeatures])$posterior[,2]

  # Append Predictions to Predict Vectors
  knn.pred.vector <- append(knn.pred.vector, knn.pred)
  rf.pred.vector <- append(rf.pred.vector, rf.pred[,2])
  boost.pred.vector <- append(boost.pred.vector, gbm.pred)
  svm.pred.vector <- append(svm.pred.vector, svm.pred)
  lda.pred.vector <- append(lda.pred.vector, ldaPredictions)
  qda.pred.vector <- append(qda.pred.vector, qdaPredictions)

}
```

```
## Warning in lda.default(x, grouping, ...): variables are collinear

## Using 200 trees...

## Warning in lda.default(x, grouping, ...): variables are collinear

## Using 200 trees...

## Warning in lda.default(x, grouping, ...): variables are collinear

## Using 200 trees...

## Warning in lda.default(x, grouping, ...): variables are collinear
```

```
## Using 200 trees...

## Warning in lda.default(x, grouping, ...): variables are collinear

## Using 200 trees...
```

```
# ROC Curves
library(ROCR)
roc_knn <- pROC::roc(gold.c.classes, knn.pred.vector)
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

```
roc_rf <- pROC::roc(gold.c.classes, rf.pred.vector)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc_gbm <- pROC::roc(gold.c.classes, boost.pred.vector)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
roc_svm <- pROC::roc(gold.c.classes, svm.pred.vector)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

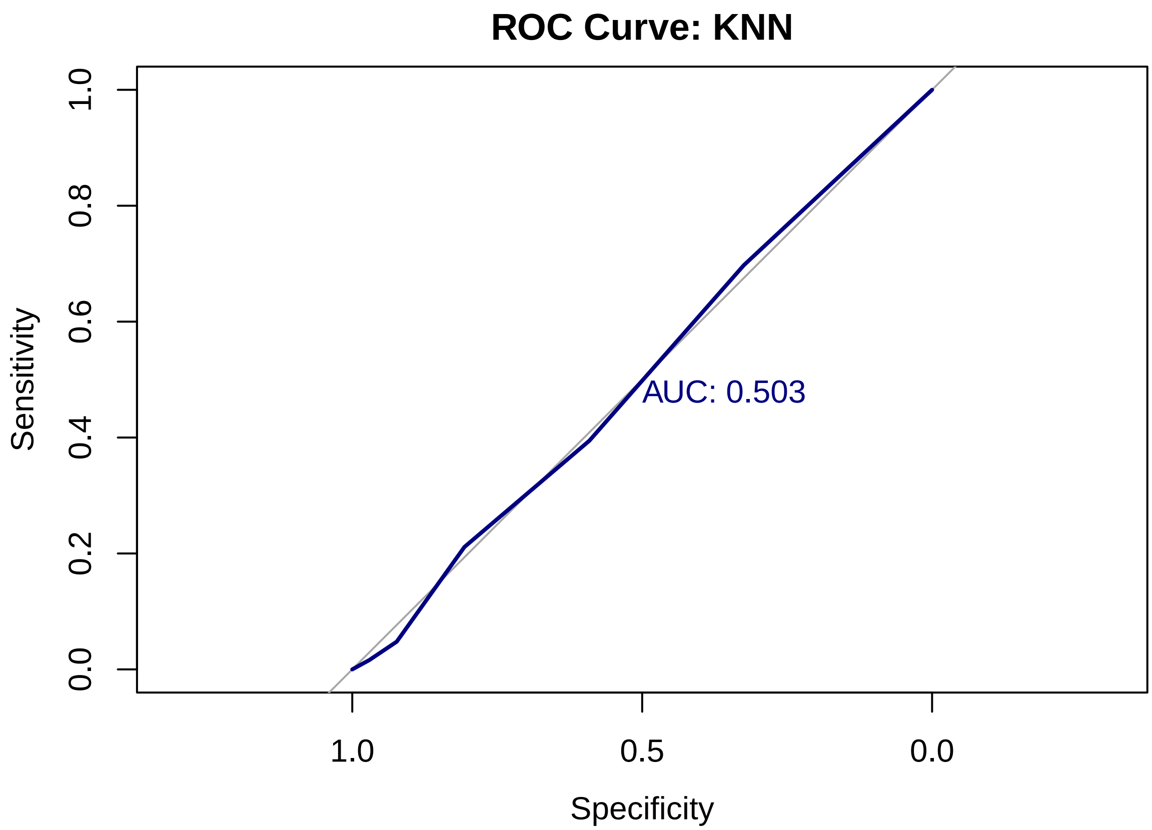
```
roc_lda <- pROC::roc(gold.c.classes, lda.pred.vector)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

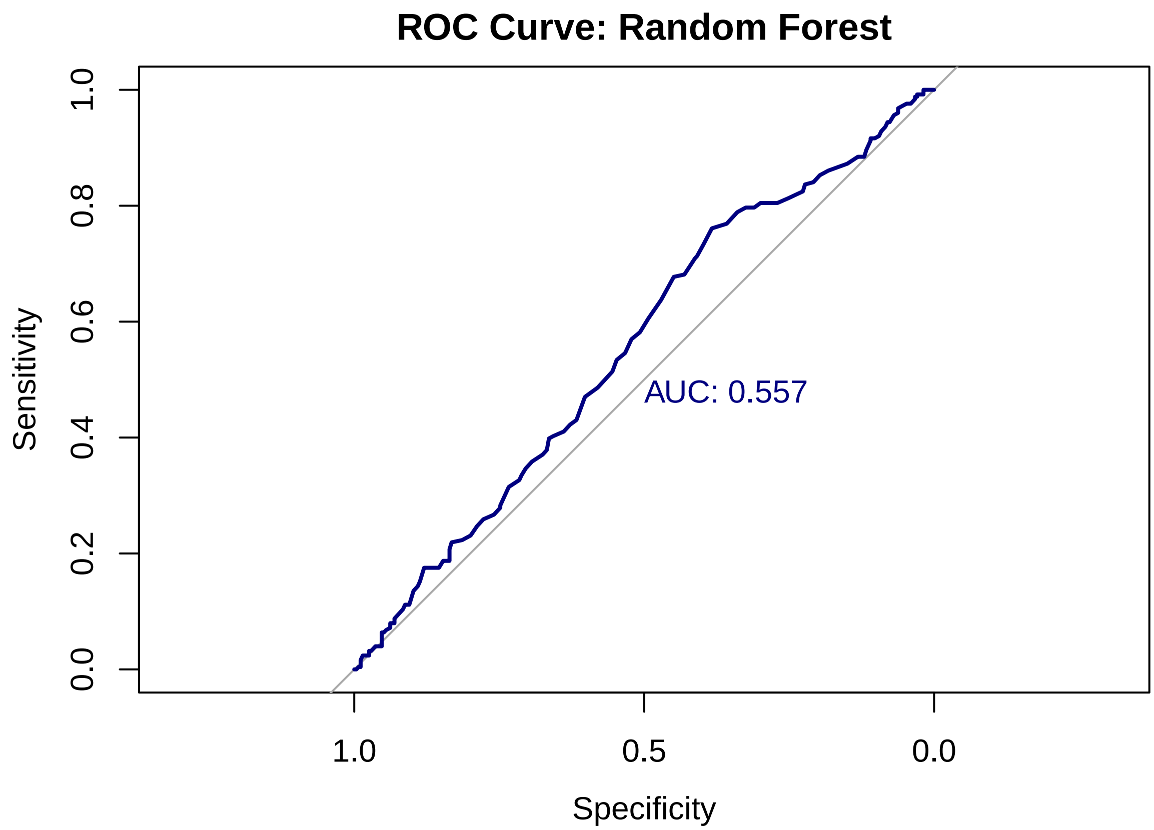
```
roc_qda <- pROC::roc(gold.c.classes, qda.pred.vector)
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

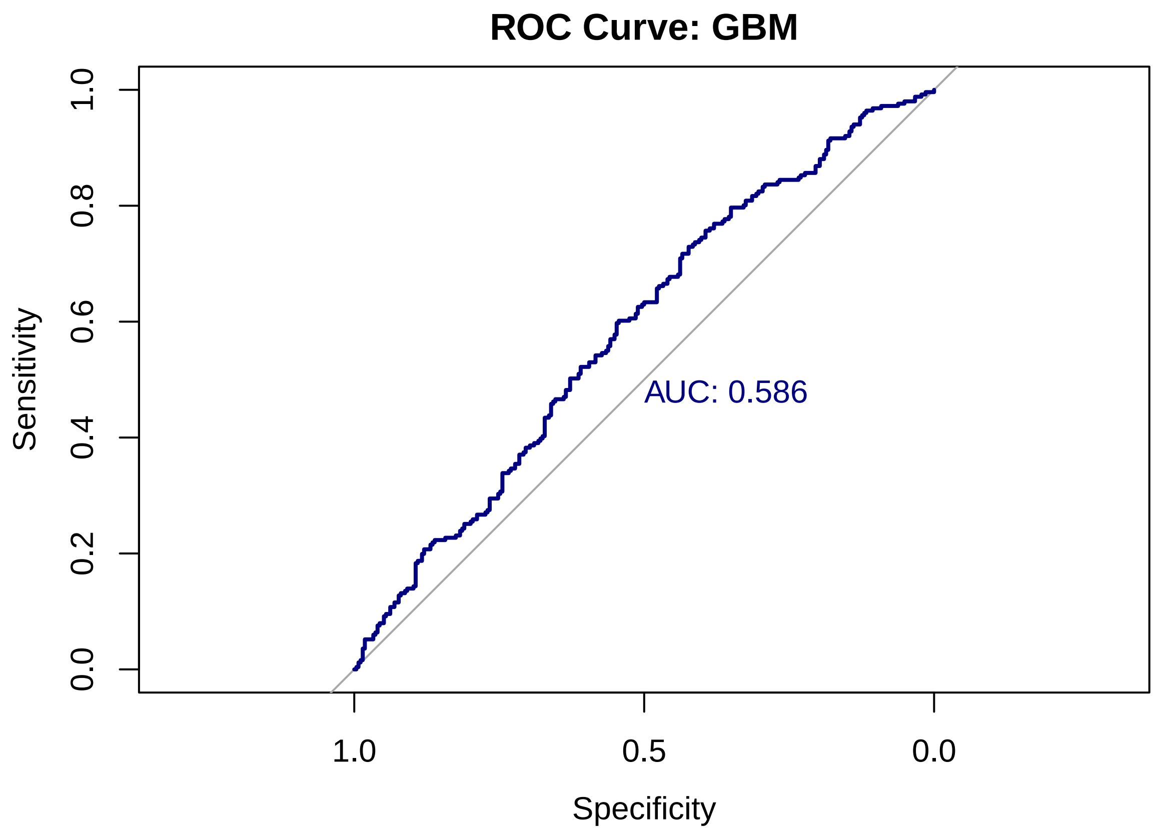
```
# Individual Plots
plot(roc_knn, main = "ROC Curve: KNN", print.auc = TRUE, col = "navy")
```



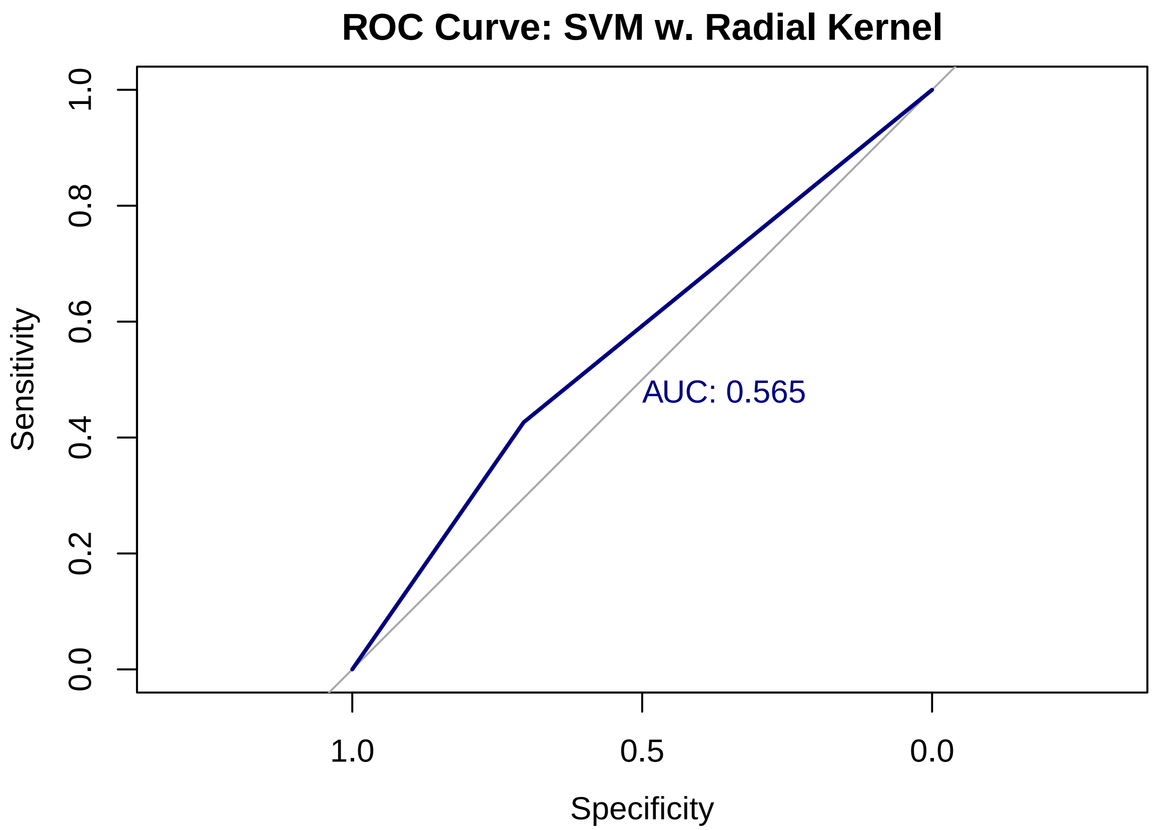

```
plot(roc_rf, main = "ROC Curve: Random Forest", print.auc = TRUE, col = "navy")
```



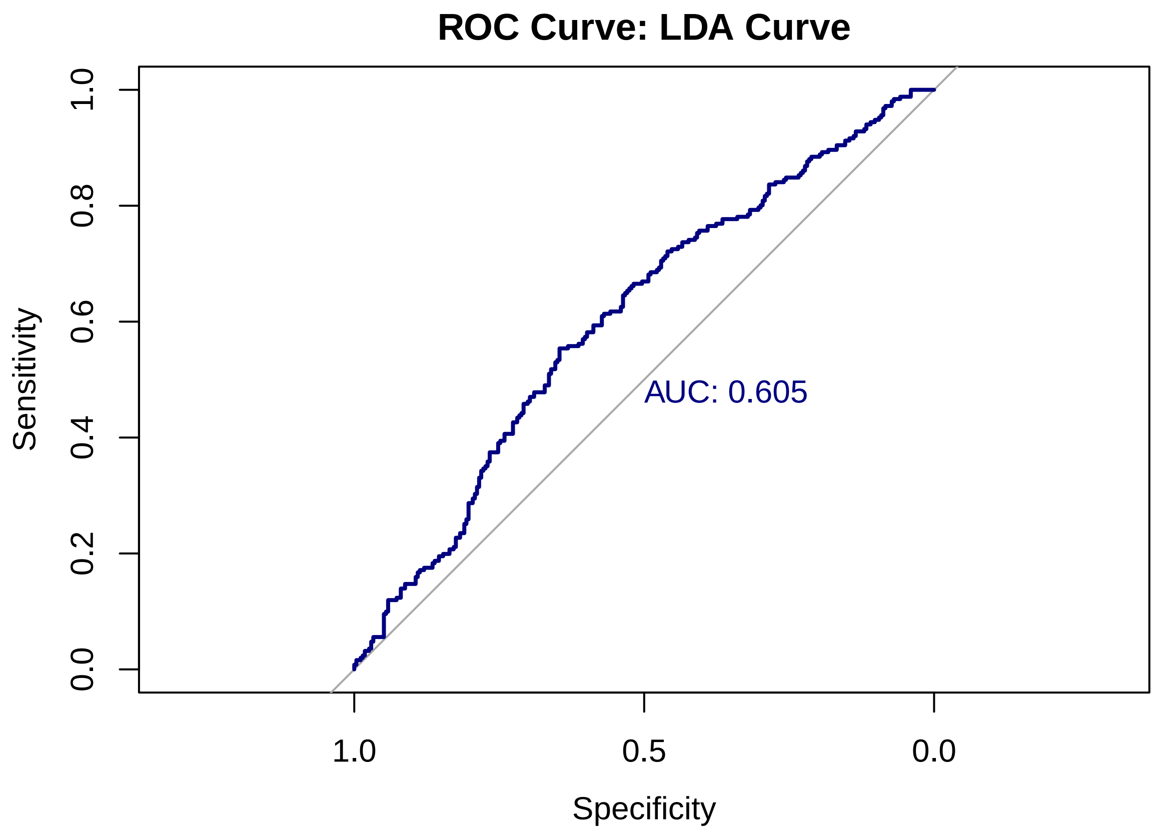

```
plot(roc_gbm, main = "ROC Curve: GBM", print.auc = TRUE, col = "navy")
```



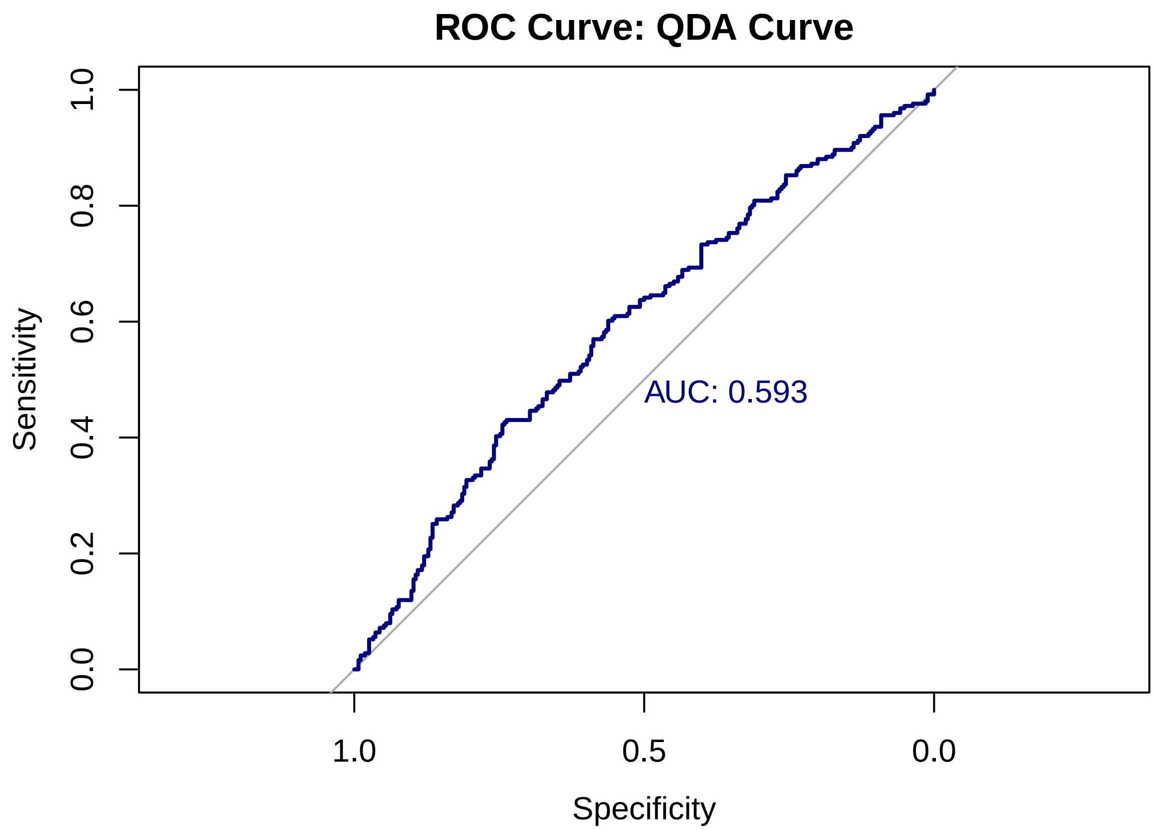

```
plot(roc_svm, main = "ROC Curve: SVM w. Radial Kernel", print.auc = TRUE, col = "navy")
```



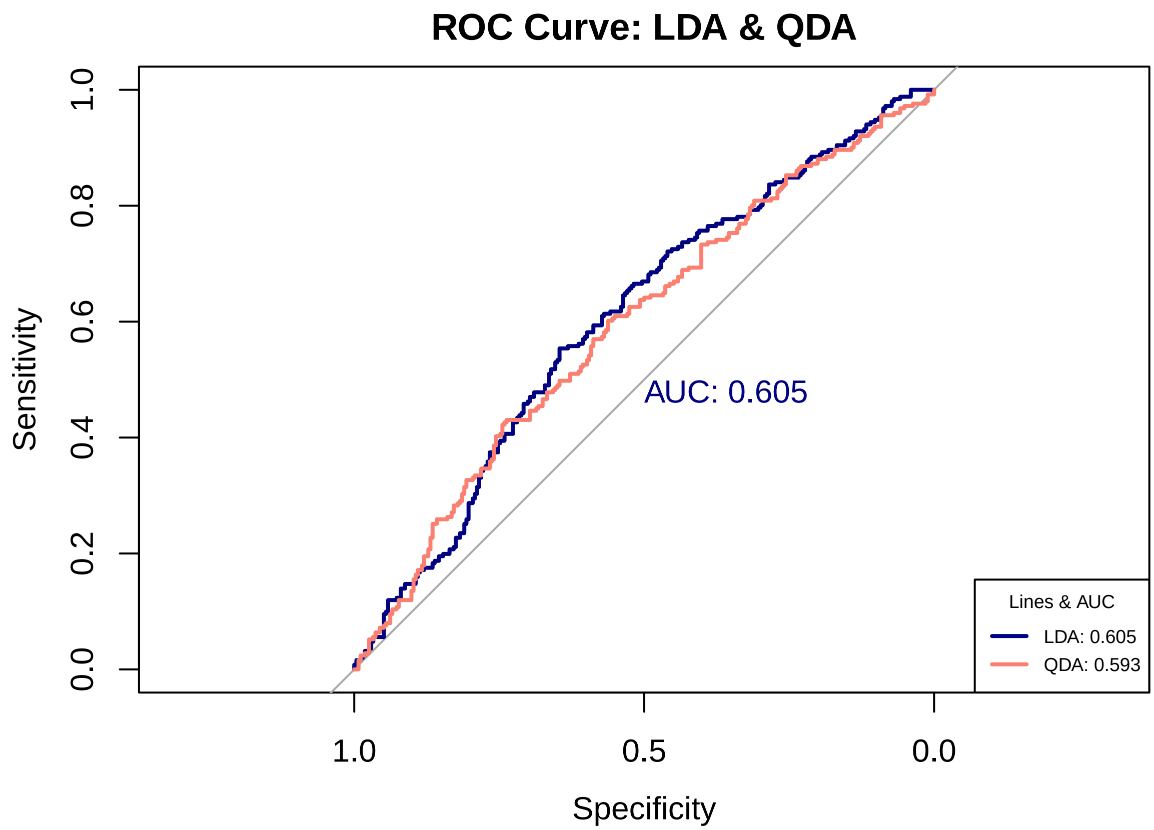

```
plot(roc_lda, main = "ROC Curve: LDA Curve", print.auc = TRUE, col = "navy")
```



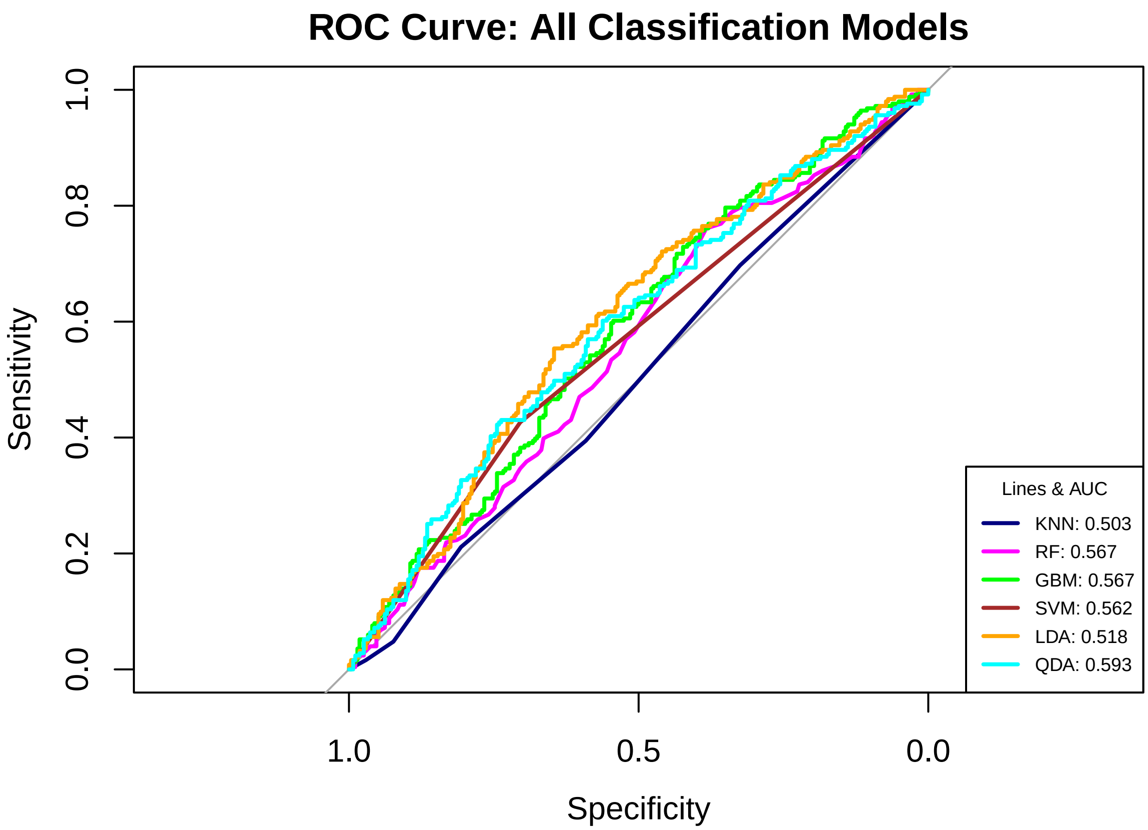

```
plot(roc_qda, main = "ROC Curve: QDA Curve", print.auc = TRUE, col = "navy")
```




```r
# LDA/QDA
plot(roc_lda, main = "ROC Curve: LDA & QDA", print.auc = TRUE, col = "navy")
lines(roc_qda, col = "salmon")
legend("bottomright", legend = c("LDA: 0.605", "QDA: 0.593"),
       col = c("navy","salmon"),  lwd = 2, cex = 0.6,
       lty = c(1, 1), title = "Lines & AUC")
```




```r
# All Together Now
plot(roc_knn, main = "ROC Curve: All Classification Models", col = "navy")
lines(roc_rf, col = "magenta")
lines(roc_gbm, col = "green")
lines(roc_svm, col = "brown")
lines(roc_lda, col = "orange")
lines(roc_qda, col = "cyan")
legend("bottomright", legend = c("KNN: 0.503", "RF: 0.567","GBM: 0.567",
                                 "SVM: 0.562", "LDA: 0.518", "QDA: 0.593"), # Put AUCs in here
       col = c("navy", "magenta","green","brown","orange","cyan"),
       lwd = 2, cex = 0.6,
       lty = c(1, 1), title = "Lines & AUC")
```




```
pROC::auc(roc_knn)
```

```
## Area under the curve: 0.5031
```

```
pROC::auc(roc_rf)
```

```
## Area under the curve: 0.557
```

```
pROC::auc(roc_gbm)
```

```
## Area under the curve: 0.5863
```

```
pROC::auc(roc_svm)
```

```
## Area under the curve: 0.5653
```

Conclusion:

Most of our models were highly non-linear and uninterpretable in our regression and classification tasks. Overall our neural network models outperformed the other models in RMSE (regression) and overall accuracy (classification). Our regression results were not deployable. On average, our models predicted returns that were about 3-4 percentage points off observed values. This could be very costly for investors we pitch our algorithm to. As for predicting the direction of gold prices on a month to month basis, our machine learnings algorithms outperformed classification benchmarks (dominant class proportions, random chance classification, etc.). The ROC curves of our classification algorithms returned deployable AUC values which tested our alogrithms on different threshold values.

In conclusion, though we do not recommend our algorithms for predicting the returns of gold prices, we will look into further bettering our algorithms for predicting the direction of gold prices. One potential future project could be looking into creating a “majority vote” decision between multiple different machine learnings models in order to classify return direction.