STA 6714 Final Project Report

Kyle Scott

11/12/2021

Question 1: Gamestop (GME) Stock Prices

a. Read in the relevant csv files from kaggle and display the data

```
library(readr)
gme <- read_csv("GME_stock.csv")


head(gme)
```

```
## # A tibble: 6 x 7
##   date       open_price high_price low_price close_price    volume adjclose_price
##   <date>          <dbl>      <dbl>     <dbl>       <dbl>     <dbl>           <dbl>
## 1 2021-01-28      265        483       112.        194.   58815800           194.
## 2 2021-01-27      355.       380       249         348.   93396700           348.
## 3 2021-01-26       88.6      150        80.2       148.  178588000           148.
## 4 2021-01-25       96.7      159.       61.1        76.8 177874000            76.8
## 5 2021-01-22       42.6       76.8      42.3        65.0 196784300            65.0
## 6 2021-01-21       39.2       44.8      37          43.0  57079800            43.0
```

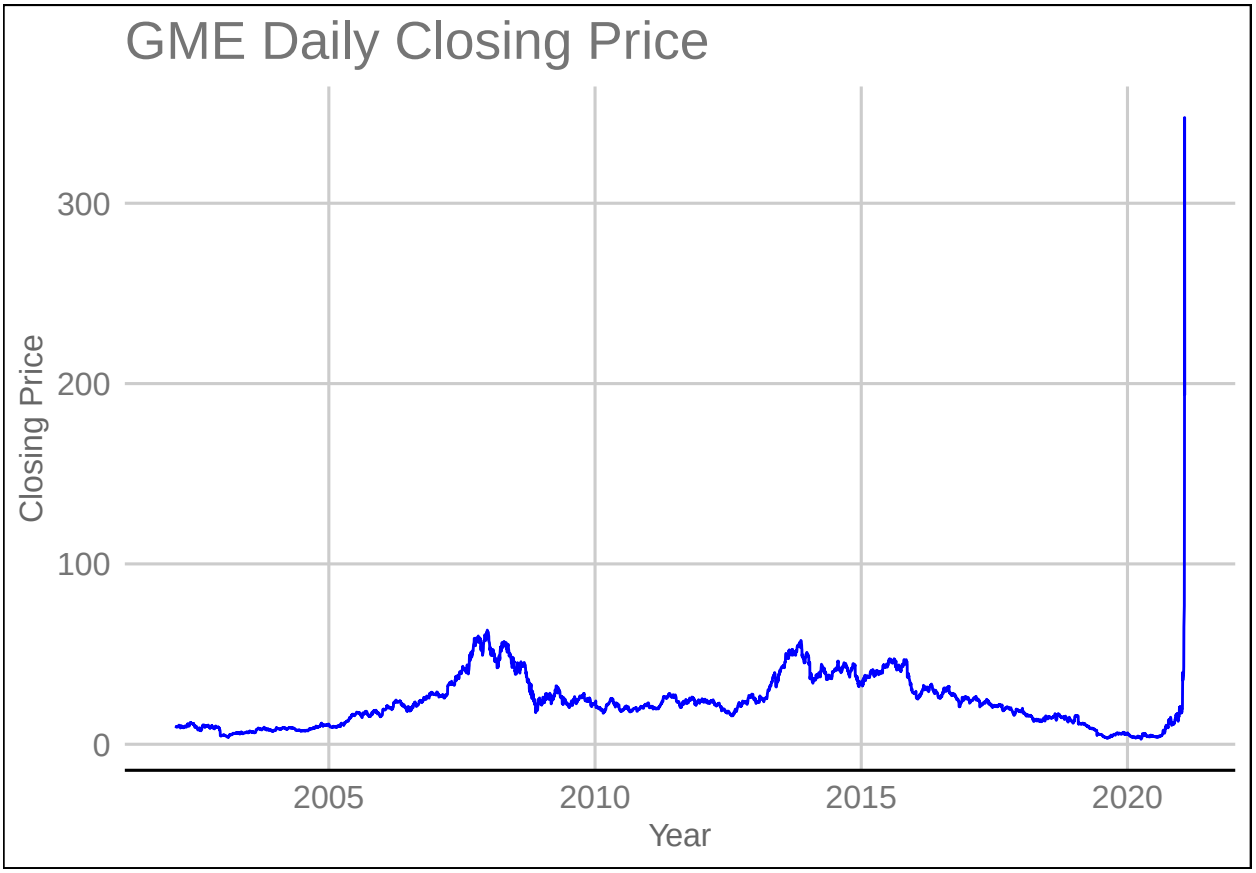
Plot the 'close_price

```
## We will need to read in the ggplot library to create our plot
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.2
```

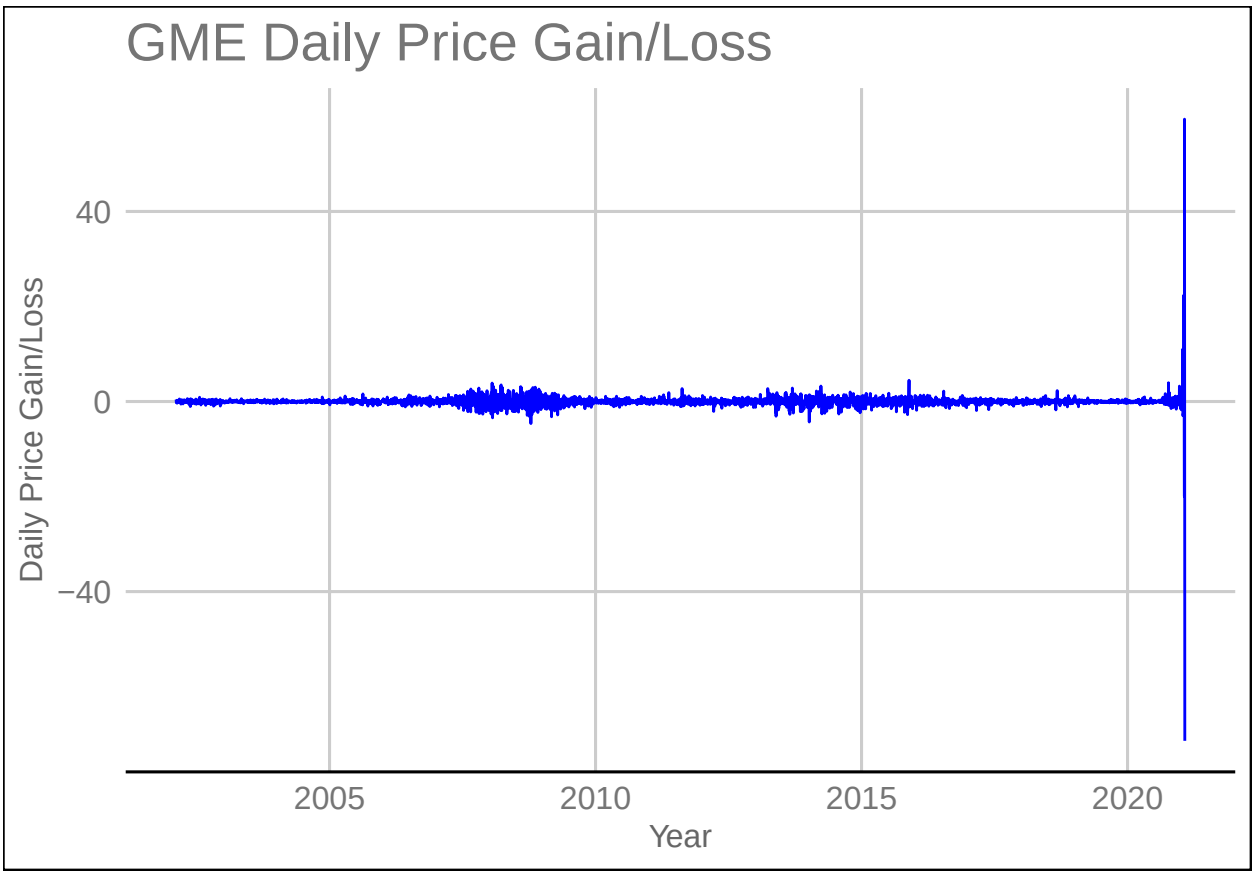
```
## Warning: replacing previous import 'vctrs::data_frame' by 'tibble::data_frame'
## when loading 'dplyr'
```

```
library(ggthemes)


## Create our initial ggplot, scaling the x axis as a date
gme_close <- ggplot(gme, aes(date, close_price)) +
  geom_line(color = 'blue') +
  scale_x_date("Year") +
  ylab("Closing Price") +
  ggtitle("GME Daily Closing Price") +
  theme_gdocs()


## Plot!
gme_close
```



b. Plot the difference between the 'open_price' and 'close_price' at each time point

```
## create a variable for the daily price gain
price_range <- gme$close_price - gme$open_price

## add the new variable to our dataframe
gme$price_range <- price_range

## Create our initial ggplot, scaling the x axis as a date
range_plot <- ggplot(gme, aes(date, price_range)) +
  geom_line(color = 'blue') +
  scale_x_date("Year") +
  ylab("Daily Price Gain/Loss") +
  ggtitle("GME Daily Price Gain/Loss") +
  theme_gdocs()

## Plot!
range_plot
```

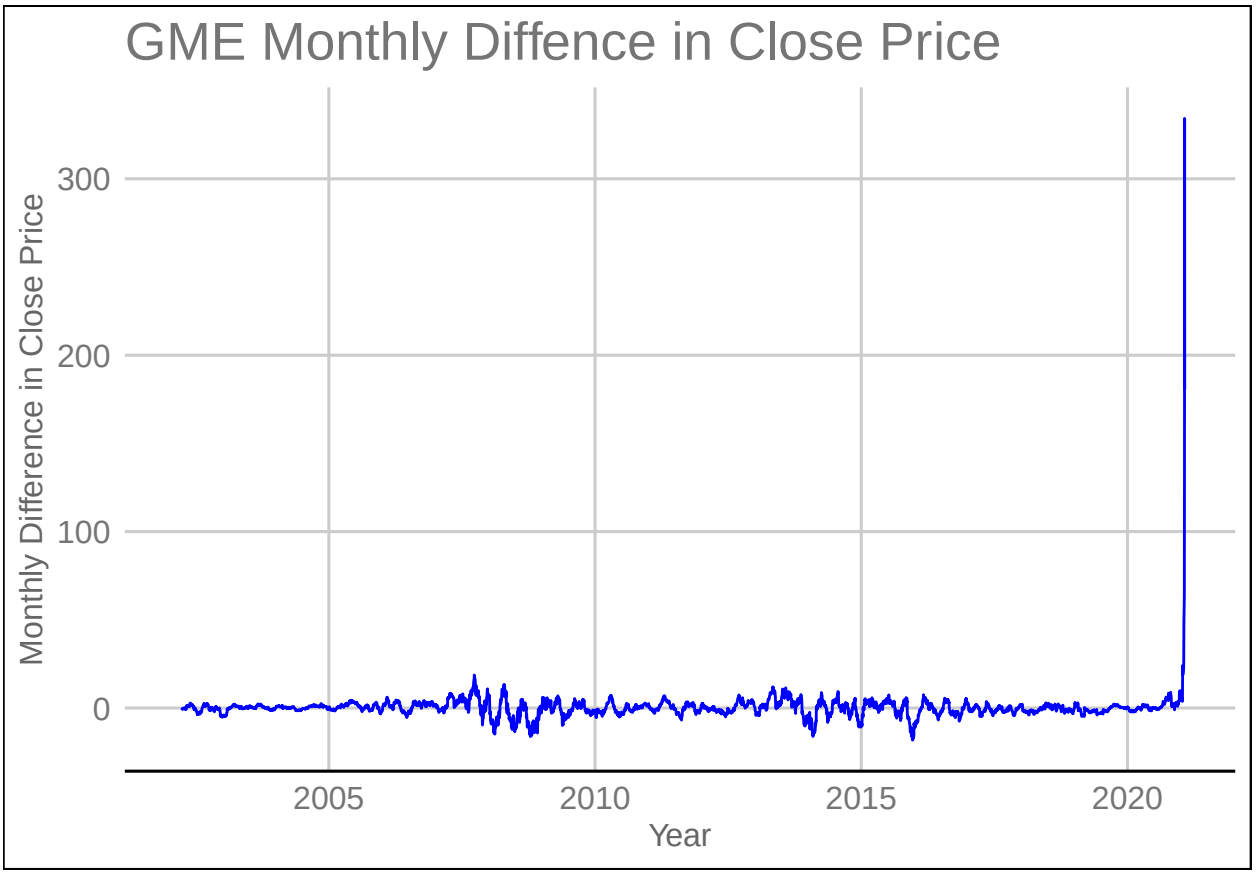


b. (cont.) Plot the difference between 'close_price' and 'close_price' where the prices are one month apart.

```
## for the purpose of simplicity, we will treat "one month" as a standard 30 days
close_1 <- gme$close_price[1:4743]
close_2 <- gme$close_price[31:4773]

## find the difference between close_1 and close_2
  ## this will give us the monthly difference in closing price
a <- rep(NA, 30)
monthly_close_diff <- close_1 - close_2
monthly_close_diff <- append(monthly_close_diff, a)

## add to our dataframe
gme$monthly_close_diff <- monthly_close_diff
```

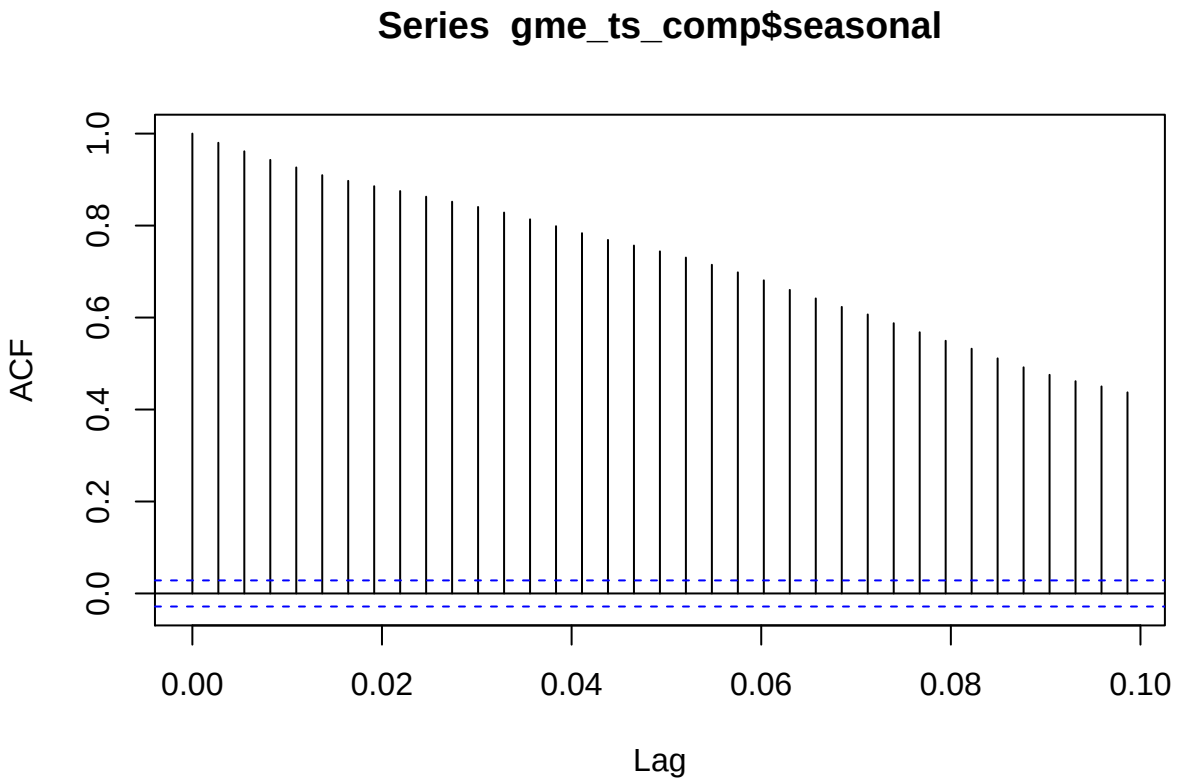
```r
## Create our initial ggplot, scaling the x axis as a date
monthly_close_plot <- ggplot(gme, aes(date, monthly_close_diff)) +
  geom_line(color = 'blue') +
  scale_x_date("Year") +
  ylab("Monthly Difference in Close Price") +
  ggtitle("GME Monthly Diffence in Close Price") +
  theme_gdocs()

## Plot!
monthly_close_plot
```

```
## Warning: Removed 30 row(s) containing missing values (geom_path).
```

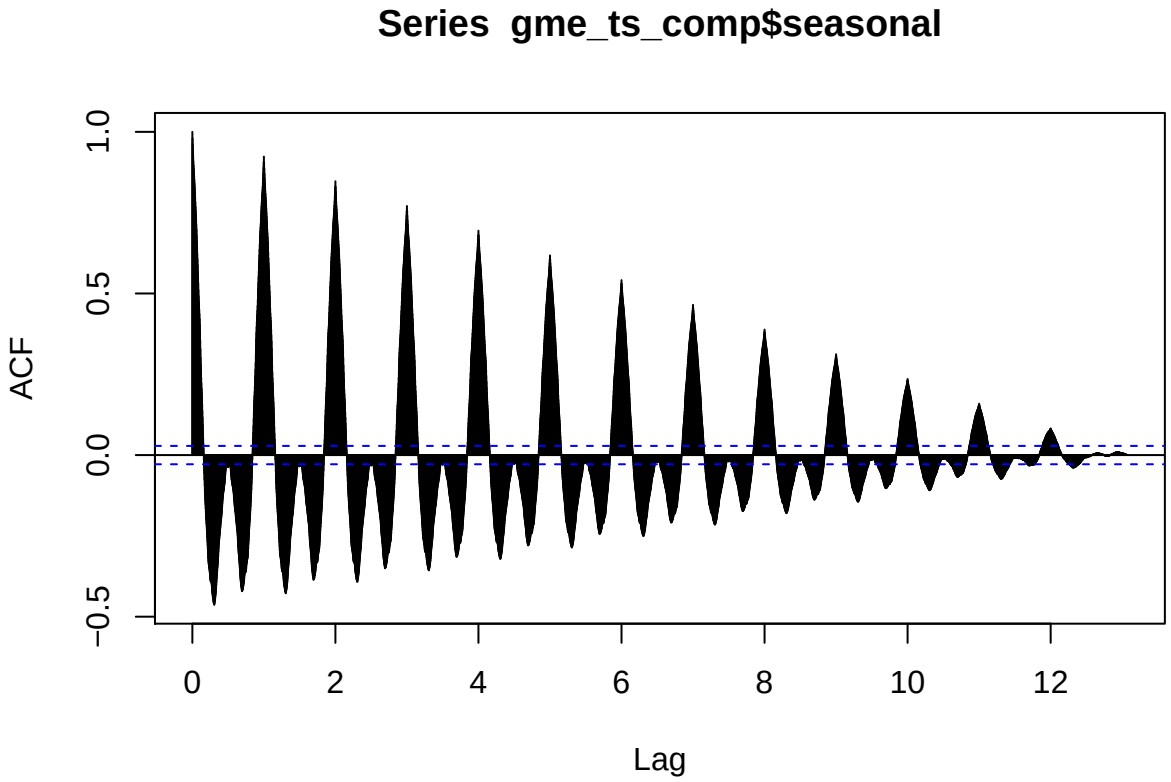


c. Plot the autocorrelation between the 'close_price' values and comment on the results

```
## create a time series for gme
gme_ts <- ts(gme$close_price, frequency = 365, start(2002, 44))
gme_ts_comp <- decompose(gme_ts)
## default acf plot
acf(gme_ts_comp$seasonal)
```



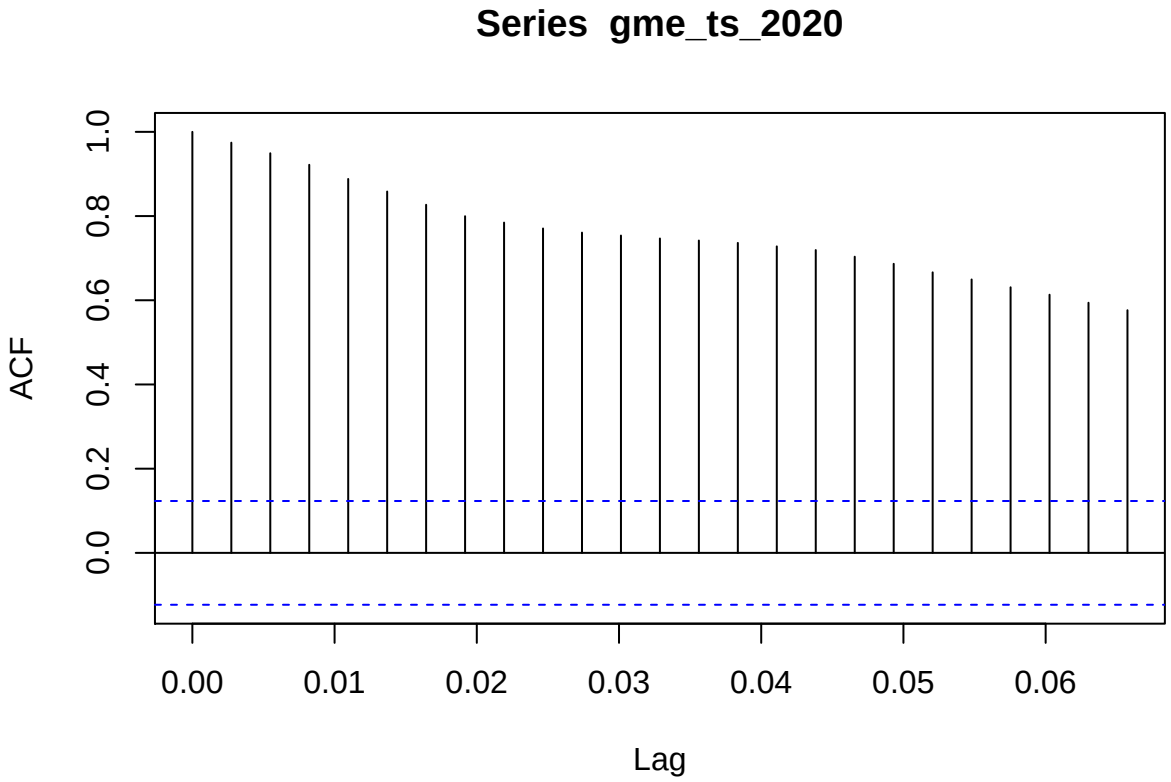

```
## increase lag max to show trends
acf(gme_ts_comp$seasonal, lag.max = 4800)
```



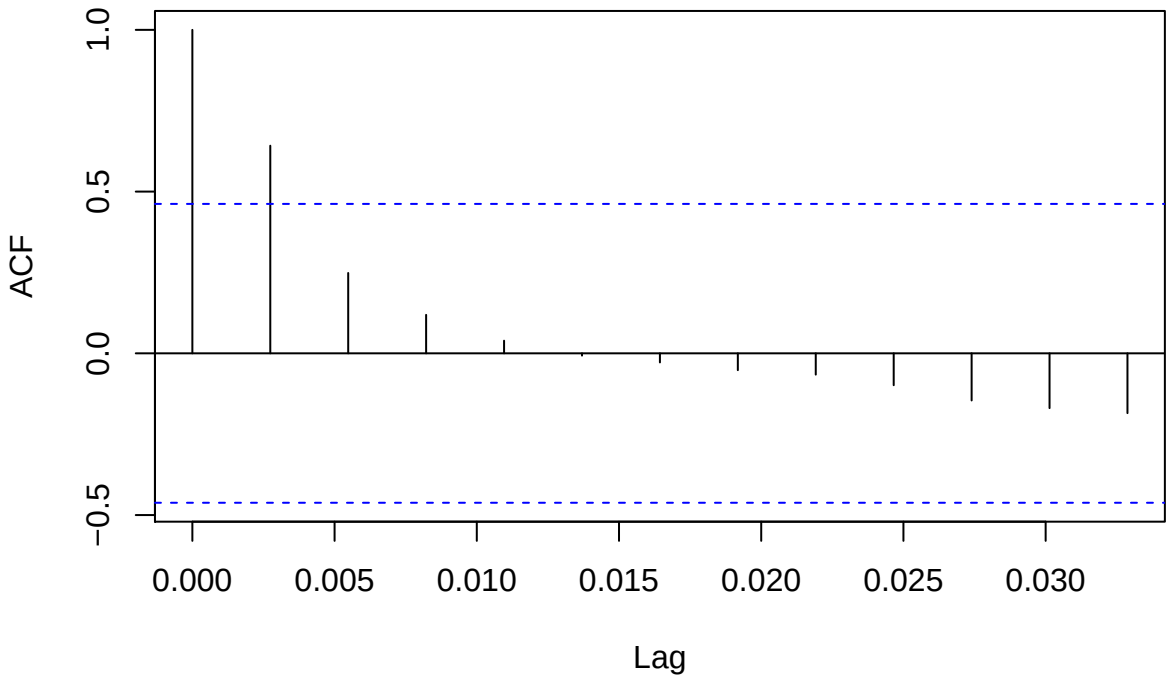
We cannot find a uniform seasonal pattern here using the ACF plot of GME. Because we are dealing with a stock, this isn't very surprising. For the most part, stocks move randomly with the current date/season having no impact on its movement. Stocks do release quarterly earnings which will impact the stock quite a lot, but there is no telling how much it will impact the stock in either direction.

c. (cont.) Plot the autocorrelation for 2021 and 2020 separately and comment on the difference

```
gme_2020 <- gme$close_price[19:271]
## create a time series for gme
gme_ts_2020 <- ts(gme_2020, frequency = 365, start(2020, 1))
## default acf plot
acf(gme_ts_2020)
```




```
gme_2021 <- gme$close_price[1:18]

## create a time series for gme

gme_ts_2021 <- ts(gme_2021, frequency = 365, start(2021, 1))

## default acf plot

acf(gme_ts_2021)
```



Based on our ACF plots here, we can see that in 2020, the close_price values were generally highly correlated with one another. Based on previous close prices, the next close price can be reasonably predicted. However, looking at the ACF plot for the beginning of 2021 (only have January data, so somewhat limited here in this study), we can see the ACF plot is all over the place and the values quickly drop to surround 0. This tells us that the closing prices in 2021 have much less bearing on the future closing prices than it did in 2021. This makes some sense when looking at our previous charts. In 2021, GME became an "internet meme stock" and tons of people began to buy into it causing wild spikes and drops. This clearly increases variability, meaning the previous closing prices wouldn't tell much about the future closes.

d. Create a column for open_price quartiles and find a conditional probability

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.2
```

```
## condense the dataframe into only 2020/2021
gme_cond <- gme %>% slice(1:271)
```

```r
## add a column called quartile with the quartile values of each closing price
gme_cond <- within(gme_cond, quartile <- as.integer(cut(open_price, quantile(open_price, probs=0:4/4), inc

## show the head/tail of the new dataframe to show the quartile column
head(gme_cond$quartile)
```

```
## [1] 4 4 4 4 4 4
```

```r
tail(gme_cond$quartile)
```

```
## [1] 3 3 3 3 3 3
```

```r
## create 3 vectors for t, t-1, and t-2 respectively
time1 <- gme_cond$quartile[1:269]
time2 <- gme_cond$quartile[2:270]
time3 <- gme_cond$quartile[3:271]

## numerator of the conditional prob
num <- sum(time1 == 1 & time2 == 4 & time3 == 4)
num
```

```
## [1] 0
```

```r
## denominator of the conditional prob
den <- sum(time2 == time3 & time2 == 4)
den
```

```
## [1] 62
```

```r
## print conditional prob
num/den
```

```
## [1] 0
```

Because there is never an occurance where t-1 and t-2 were in the 4th quartile and t was in the 1st quantile, the conditional probability comes out to be zero. This isn't shocking because stocks rise and fall in a line, there are trends. For the stock to be in the 4th quantile for the two previous days and then fall to the 1st quantile, there would need to be a massive drop in the price of the stock. We can see here that this was never the case for GME and therefore we found the probability to be zero.

Question 2: Placement Data

0. Read in the data and show the head of the data

```
placement <- read_csv("Placement_Data_Full_Class.csv")
```

```
## Parsed with column specification:
## cols(
##   sl_no = col_double(),
##   gender = col_character(),
##   ssc_p = col_double(),
##   ssc_b = col_character(),
##   hsc_p = col_double(),
##   hsc_b = col_character(),
##   hsc_s = col_character(),
##   degree_p = col_double(),
##   degree_t = col_character(),
##   workex = col_character(),
##   etest_p = col_double(),
##   specialisation = col_character(),
##   mba_p = col_double(),
##   status = col_character(),
##   salary = col_double()
## )
```

```
head(placement)
```

```
## Warning: `...` is not empty.
##
```

```
## We detected these problematic arguments:
## * `needs_dots`
##
## These dots only exist to allow future extensions and should be empty.
## Did you misspecify an argument?
```

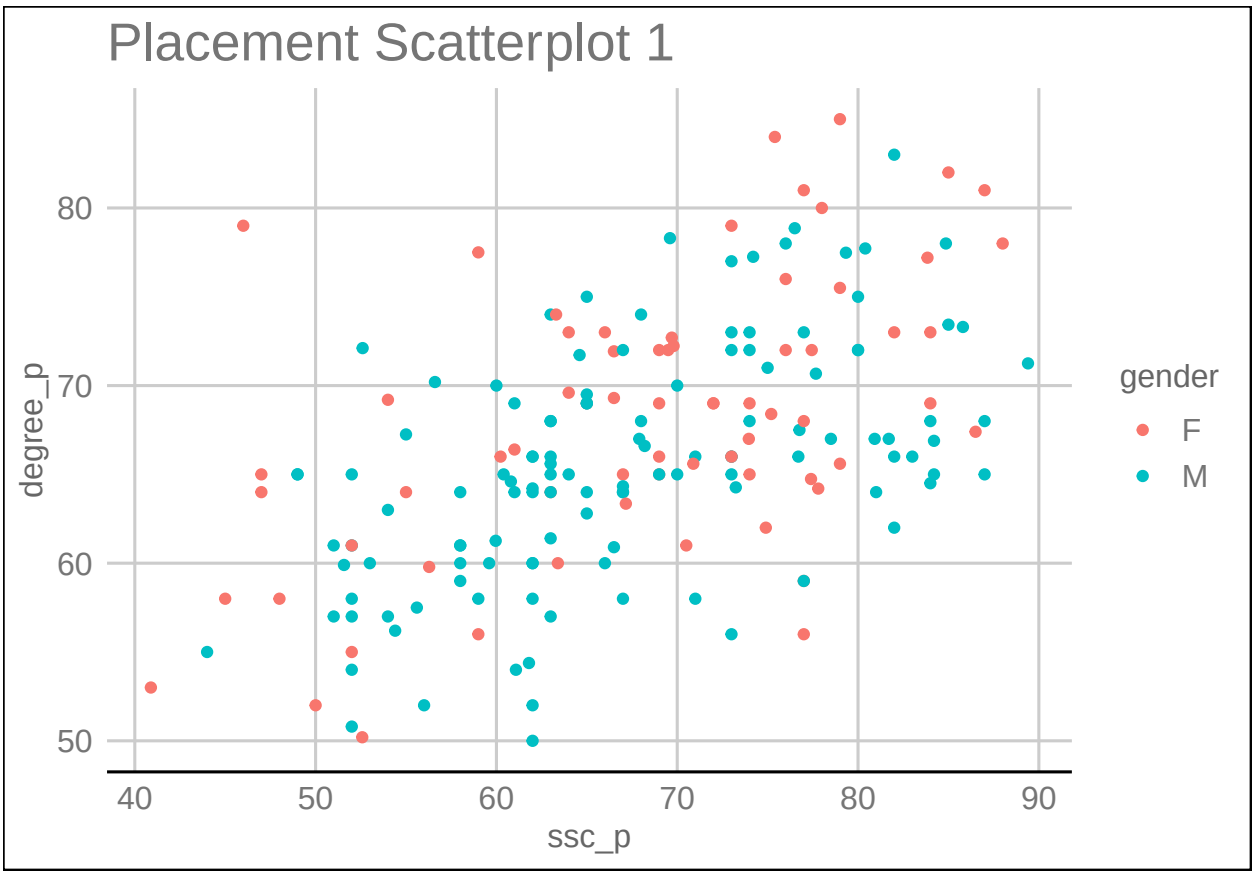
```
## # A tibble: 6 x 15
##   sl_no gender ssc_p ssc_b hsc_p hsc_b hsc_s degree_p degree_t workex etest_p
##   <dbl> <chr>  <dbl> <chr> <dbl> <chr> <chr>    <dbl> <chr>    <chr>    <dbl>
## 1     1 M       67   Othe~  91   Othe~ Comm~     58   Sci&Tech No        55
## 2     2 M       79.3 Cent~  78.3 Othe~ Scie~     77.5 Sci&Tech Yes       86.5
## 3     3 M       65   Cent~  68   Cent~ Arts      64   Comm&Mg~ No        75
## 4     4 M       56   Cent~  52   Cent~ Scie~     52   Sci&Tech No        66
## 5     5 M       85.8 Cent~  73.6 Cent~ Comm~     73.3 Comm&Mg~ No        96.8
## 6     6 M       55   Othe~  49.8 Othe~ Scie~     67.2 Sci&Tech Yes       55
## # ... with 4 more variables: specialisation <chr>, mba_p <dbl>, status <chr>,
## #   salary <dbl>
```

a. Separate the data set into testing and training

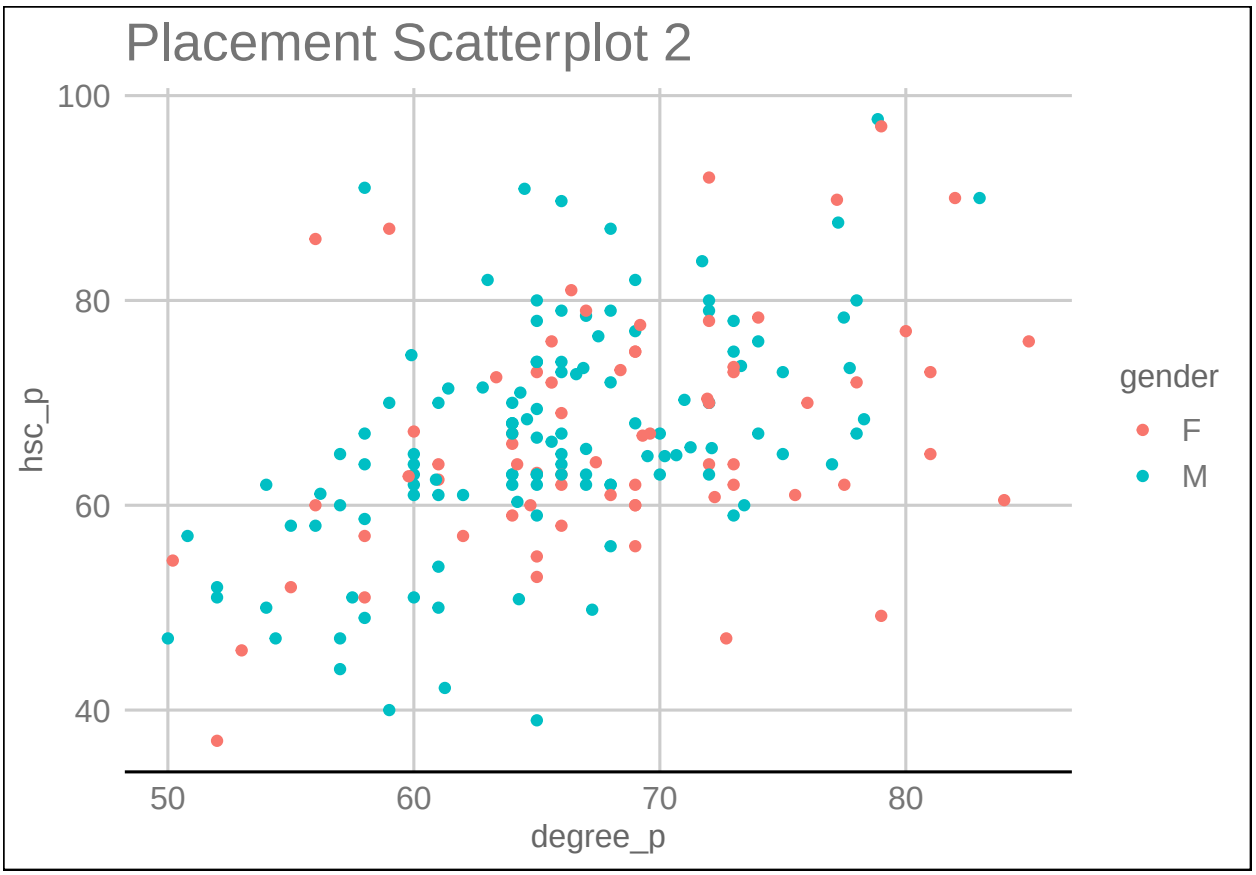
```
## create training set with first 190 data points
placement_train <- placement %>% slice(1:190)


## create testing set with final 25 data points
placement_test <- placement %>% slice(191:215)
```

a. (cont.) Produce scatter plots on the training set

```
## create ggplot figure
scatter1 <- ggplot(placement_train, aes(ssc_p, degree_p)) +
  geom_point(aes(color = gender)) +
  ggtitle("Placement Scatterplot 1") +
  theme_gdocs()


# Plot!
scatter1
```




```
## create ggplot figure
scatter2 <- ggplot(placement_train, aes(degree_p, hsc_p)) +
  geom_point(aes(color = gender)) +
  ggtitle("Placement Scatterplot 2") +
  theme_gdocs()

# Plot!
scatter2
```



b. Produce a clustering on the two scatterplots

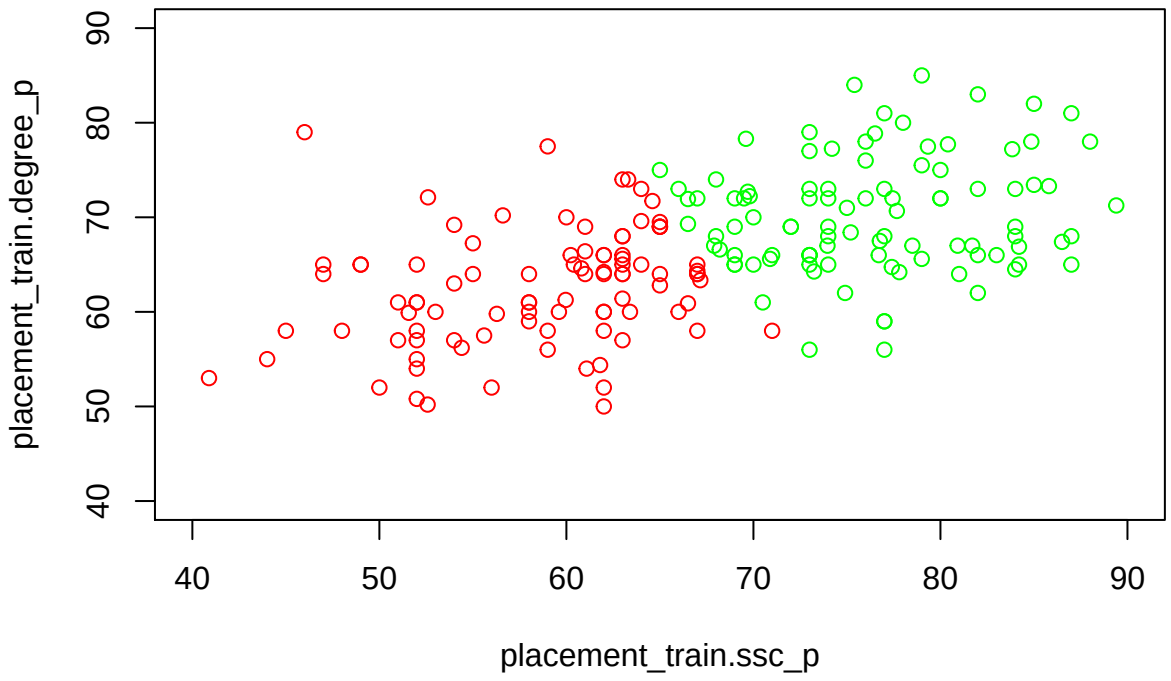
```
p_train_1 <- data.frame(placement_train$ssc_p, placement_train$degree_p)
cluster <- kmeans(p_train_1, 2)
p_train_1 <- cbind(p_train_1, cluster$cluster)
names(p_train_1)[3] <- "clusterNum"


clusternum1 = p_train_1[ which(p_train_1$clusterNum==1),1:2 ]


clusternum2 = p_train_1[ which(p_train_1$clusterNum==2),1:2 ]
```

```
cluster
```

```
## K-means clustering with 2 clusters of sizes 97, 93
##
## Cluster means:
##   placement_train.ssc_p placement_train.degree_p
## 1              76.32897                 70.28680
```

```
## 2              58.55215                  62.48065
##
## Clustering vector:
##   [1] 2 1 2 2 1 2 2 1 1 2 2 1 2 1 2 2 2 2 2 2 2 1 1 1 1 2 1 2 1 2 2 2 2 1 2 1 2
##  [38] 1 1 1 1 1 2 1 1 1 1 2 2 2 1 2 2 1 1 2 2 1 1 2 1 1 1 2 1 2 1 1 1 1 1 1 1 2
##  [75] 2 2 1 2 1 1 1 1 2 1 1 1 2 2 1 1 1 2 2 2 2 1 1 1 1 2 2 2 1 1 1 2 2 1 2 2 1
## [112] 2 2 1 2 1 1 1 1 2 2 2 1 1 2 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 2 2 1 1 2 1 2 1
## [149] 1 2 2 1 1 2 2 2 1 2 2 2 1 2 1 2 2 2 2 1 2 2 2 1 1 2 1 2 2 1 1 1 2 2 2 2 2
## [186] 1 2 1 2 2
##
## Within cluster sum of squares by cluster:
## [1] 6904.123 7083.795
##  (between_SS / total_SS =  56.1 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
plot(clusternum1,col="green", xlim=c(40,90), ylim=c(40,90))
points(clusternum2,col="red")
```

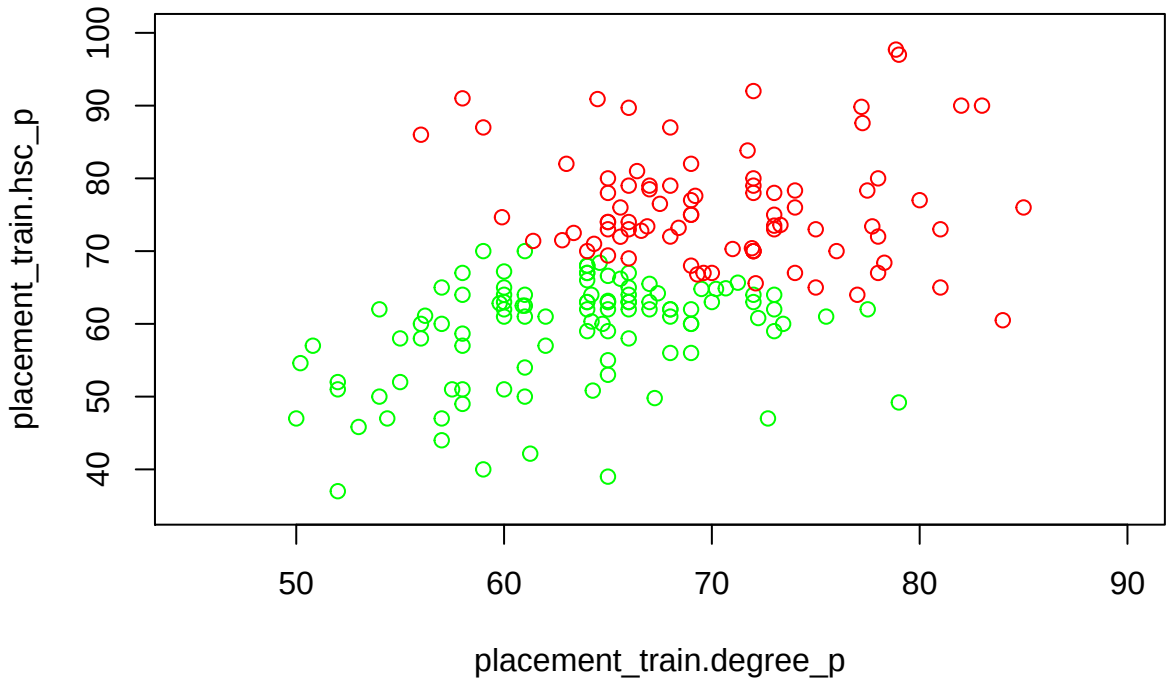


```r
p_train_2 <- data.frame(placement_train$degree_p, placement_train$hsc_p)
cluster <- kmeans(p_train_2, 2)
p_train_2 <- cbind(p_train_2, cluster$cluster)
names(p_train_2)[3] <- "clusterNum"

clusternum1 = p_train_2[ which(p_train_2$clusterNum==1),1:2 ]

clusternum2 = p_train_2[ which(p_train_2$clusterNum==2),1:2 ]
```

```r
cluster
```

```
## K-means clustering with 2 clusters of sizes 107, 83
##
## Cluster means:
##   placement_train.degree_p placement_train.hsc_p
## 1                 63.24561              59.26701
## 2                 70.61735              76.08590
##
## Clustering vector:
##   [1] 2 2 1 1 2 1 1 1 2 1 1 2 1 2 1 2 1 1 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2 2 1 2 1
##  [38] 2 1 1 2 1 1 2 2 1 2 1 1 1 2 1 1 2 1 1 2 2 1 2 2 2 1 2 2 1 2 2 1 2 1 2 2 2
##  [75] 1 1 2 2 2 1 1 1 2 2 1 2 1 1 1 2 2 1 2 1 1 2 2 1 2 2 1 2 1 2 1 1 1 2 2 1 2
## [112] 1 1 2 2 1 2 2 2 1 1 2 2 1 2 2 1 1 2 2 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 2
## [149] 2 1 1 2 2 1 1 2 2 1 1 1 2 1 2 1 2 2 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 2 1
## [186] 2 1 1 1 2
##
## Within cluster sum of squares by cluster:
## [1] 9400.704 8155.186
##  (between_SS / total_SS =  47.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
plot(clusternum1,col="green", xlim=c(45,90), ylim=c(35,100))
points(clusternum2,col="red")
```

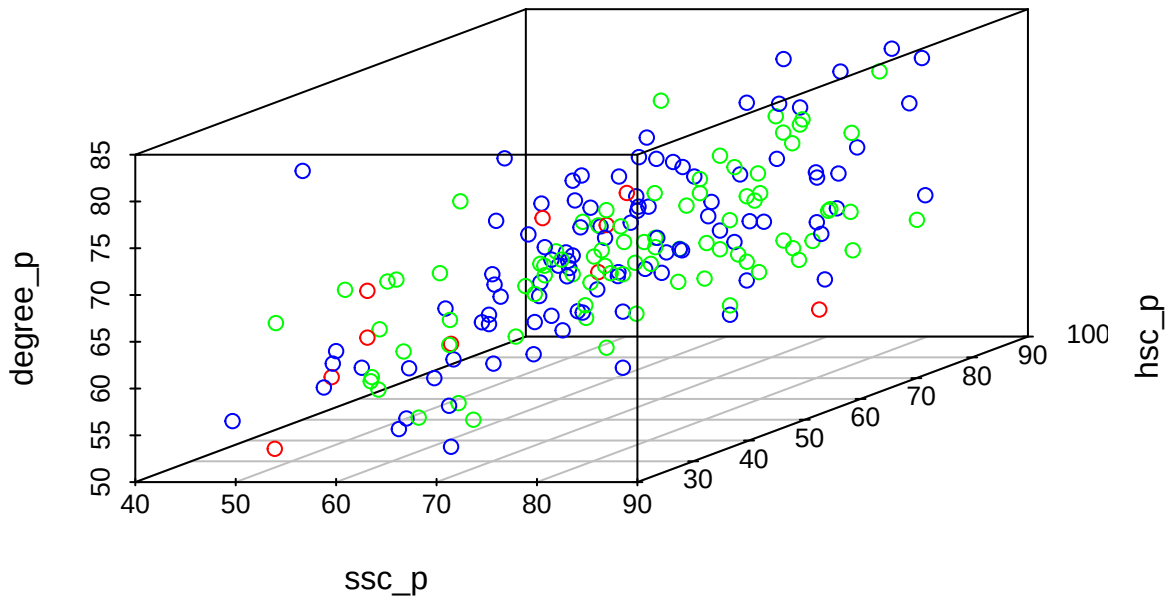


Considering that both scatterplots show no sign of separation by gender simply by looking at the plot, it is hard to imagine that simple clustering like this would be able to create much of a useful divide. With that said, I would pick the first scatterplot as the one that produced the better clustering. With both accuracies below 60%, neither did a good job but the first scatterplot did display a higher accuracy by about 10%.

c. Using T_SNE, perform a clustering with 3 variables.

```
p_train_3 <- data.frame('ssc_p' = placement_train$ssc_p, 'hsc_p' = placement_train$hsc_p,
                        'degree_p' = placement_train$degree_p,
                        'hsc_s' = placement_train$hsc_s)
```

```
library(scatterplot3d)
colors <- c("red", "blue", "green")
colors <- colors[as.numeric(p_train_3$hsc_s)]
scatterplot3d(p_train_3[,1:3], color=colors)
```

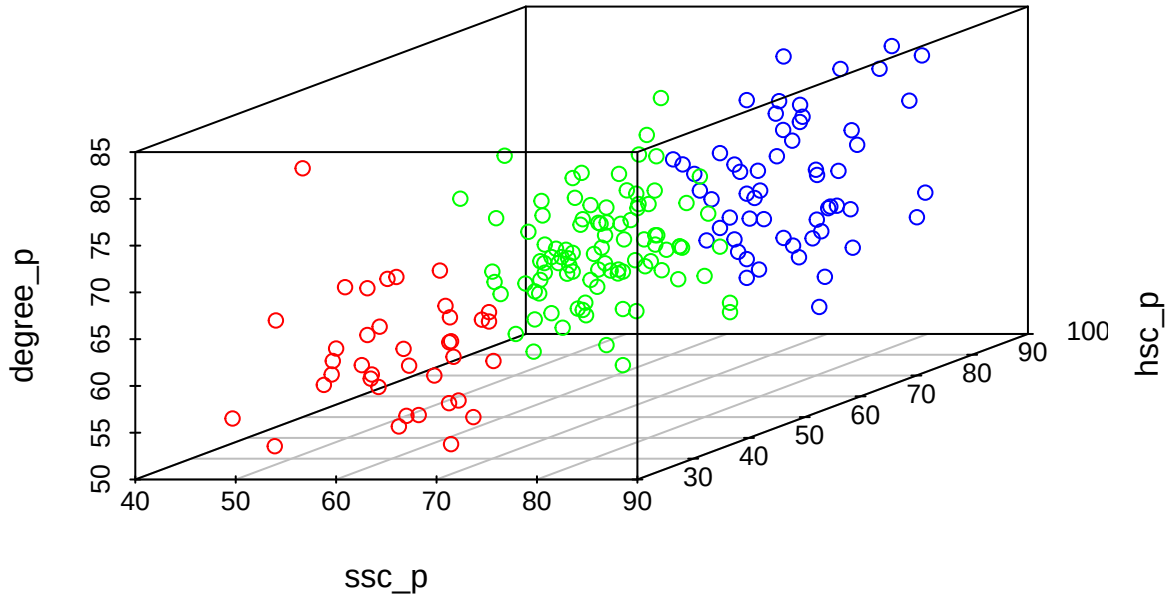



```
kmeans.fit <- kmeans(p_train_3[,c(-4)], 3)
kmeans.fit
```

```
## K-means clustering with 3 clusters of sizes 38, 57, 95
##
## Cluster means:
##      ssc_p    hsc_p degree_p
## 1 53.16289 53.19842 58.45737
## 2 79.06158 77.08842 71.42982
## 3 66.55326 65.69600 66.69095
##
## Clustering vector:
##   [1] 2 2 3 1 2 1 1 2 2 3 1 3 1 2 1 3 3 3 3 3 3 2 3 3 2 1 2 3 2 3 3 3 3 2 1 2 1
##  [38] 2 3 2 2 3 1 2 2 3 3 3 3 1 2 1 1 2 3 3 3 2 3 3 2 2 2 3 2 1 2 2 3 2 3 2 2 2
##  [75] 3 3 3 3 2 3 3 2 3 2 3 2 3 1 3 2 2 1 3 1 3 2 2 3 3 3 1 3 3 2 3 3 1 2 3 1 3
## [112] 1 1 2 3 3 3 2 2 3 1 3 3 3 3 2 3 3 2 2 3 3 3 3 2 3 1 3 2 3 3 3 2 3 1 2 3 3
## [149] 2 1 3 3 3 1 1 3 2 3 3 1 2 1 2 3 3 3 3 3 1 1 3 2 3 1 3 3 1 2 3 3 3 3 1 3 1
## [186] 2 1 3 1 3
##
## Within cluster sum of squares by cluster:
## [1]  4307.968  8515.076 10117.786
##  (between_SS / total_SS =  58.6 %)
##
```

```
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
colors <- c("red", "blue", "green")
colors <- colors[as.numeric(kmeans.fit$cluster)]
scatterplot3d(p_train_3[,1:3], color=colors)
```



With an accuracy of 58.6%, this was not very effective. This is especially evident because not all clusters were the same size within our raw data, so our clustering model accounted far too much for the red label.

Question 3

Due to my lack of experience with Decision Trees and Random Forests in R, I will do this question using Python. This also gives me a chance to showcase an additional language.

Question 4

a. Gather the data from Mongo and bring it into R

```
# set up mongodb through R - start mongodb in the terminal first


#install.packages("devtools")
library(devtools)
#install_github(repo = "mongosoup/rmongodb")
#install.packages("rmongodb")
library(rmongodb)
```

```
# connect to mongodb and ensure that it is connected
mongo <- mongo.create(host = "localhost")
mongo.is.connected(mongo)
```

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

We need to setup our mongo database, which I have done in the terminal by running the following:

mongoimport –type csv -d stockDB -c ethereumCSV –headerline ETHUSD.csv

mongoimport –type csv -d stockDB -c googleCSV –headerline GOOG.csv

mongoimport –type csv -d stockDB -c gmeCSV –headerline GME_stock.csv

mongoimport –type csv -d stockDB -c dogeCSV –headerline DOGE-USD.csv

The -d and -c flags are very important here because these flags determine which database and collection the csv will go in. Once we need to pull data from the csv, we will need both of these names.

```
# confirm that our database was created, we should see 'config' and our new stockDB
mongo.get.databases(mongo)
```

```
## [1] "config"      "countriesDB" "stockDB"
```

```
eth_date_vec <- c()
ethereum_bson <- mongo.find(mongo, ns = "stockDB.ethereumCSV")
while(mongo.cursor.next(ethereum_bson)) {
    tmp_bson <- mongo.cursor.value(ethereum_bson)
    tmp_json <- mongo.bson.to.list(tmp_bson)
    tmp_date <- tmp_json$Date
    eth_date_vec <- append(eth_date_vec,tmp_date)
}
```

```
google_date_vec <- c()
google_bson <- mongo.find(mongo, ns = "stockDB.googleCSV")
while(mongo.cursor.next(google_bson)) {
    tmp_bson <- mongo.cursor.value(google_bson)
    tmp_json <- mongo.bson.to.list(tmp_bson)
    tmp_date <- tmp_json$Date
    google_date_vec <- append(google_date_vec,tmp_date)
}
```

```
gme_date_vec <- c()
gme_bson <- mongo.find(mongo, ns = "stockDB.gmeCSV")
while(mongo.cursor.next(gme_bson)) {
    tmp_bson <- mongo.cursor.value(gme_bson)
    tmp_json <- mongo.bson.to.list(tmp_bson)
    tmp_date <- tmp_json$date
    gme_date_vec <- append(gme_date_vec,tmp_date)
}
```

```
eth_open_vec <- c()
ethereum_bson <- mongo.find(mongo, ns = "stockDB.ethereumCSV")
while(mongo.cursor.next(ethereum_bson)) {
    tmp_bson <- mongo.cursor.value(ethereum_bson)
    tmp_json <- mongo.bson.to.list(tmp_bson)
    tmp_open <- tmp_json$Open
    eth_open_vec <- append(eth_open_vec,tmp_open)
}
```

```
google_open_vec <- c()
google_bson <- mongo.find(mongo, ns = "stockDB.googleCSV")
while(mongo.cursor.next(google_bson)) {
    tmp_bson <- mongo.cursor.value(google_bson)
    tmp_json <- mongo.bson.to.list(tmp_bson)
    tmp_open <- tmp_json$Open
    google_open_vec <- append(google_open_vec,tmp_open)
}
```

```
gme_open_vec <- c()
gme_bson <- mongo.find(mongo, ns = "stockDB.gmeCSV")
while(mongo.cursor.next(gme_bson)) {
    tmp_bson <- mongo.cursor.value(gme_bson)
    tmp_json <- mongo.bson.to.list(tmp_bson)
    tmp_open <- tmp_json$open_price
    gme_open_vec <- append(gme_open_vec,tmp_open)
}
```

```
doge_date_vec <- c()
doge_bson <- mongo.find(mongo, ns = "stockDB.dogeCSV")
while(mongo.cursor.next(doge_bson)) {
    tmp_bson <- mongo.cursor.value(doge_bson)
    tmp_json <- mongo.bson.to.list(tmp_bson)
    tmp_date <- tmp_json$Date
    doge_date_vec <- append(doge_date_vec,tmp_date)
}
```

```
doge_open_vec <- c()
doge_bson <- mongo.find(mongo, ns = "stockDB.dogeCSV")
while(mongo.cursor.next(doge_bson)) {
    tmp_bson <- mongo.cursor.value(doge_bson)
    tmp_json <- mongo.bson.to.list(tmp_bson)
    tmp_open <- tmp_json$Open
    doge_open_vec <- append(doge_open_vec,tmp_open)
}
```

```
## create our dataframes from the info we pulled from Mongo
e <- data.frame("date" = eth_date_vec, "open" = eth_open_vec)
go <- data.frame("date" = google_date_vec, "open" = google_open_vec)
gm <- data.frame("date" = gme_date_vec, "open" = gme_open_vec)
d <- data.frame("date" = doge_date_vec, "open" = doge_open_vec)
e$date <- as.Date(e$date)
go$date <- as.Date(go$date)
gm$date <- as.Date(gm$date)
d$date <- as.Date(d$date)
## sort data by dates
ethereum <- e[order(as.Date(e$date, format="%Y-%m-%d")),]
google <- go[order(as.Date(go$date, format="%Y-%m-%d")),]
gme <- gm[order(as.Date(gm$date, format="%Y-%m-%d")),]
doge <- d[order(as.Date(d$date, format="%Y-%m-%d")),]
```

```
## slice the data into a common three year period
  ## start: 01-03-2017
  ## end: 12-31-2019


ethereum <- ethereum[ethereum$date >= "2017-01-03" & ethereum$date <= "2019-12-31", ]
google <- google[google$date >= "2017-01-03" & google$date <= "2019-12-31", ]
gme <- gme[gme$date >= "2017-01-03" & gme$date <= "2019-12-31", ]
doge <- doge[doge$date >= "2017-01-03" & doge$date <= "2019-12-31", ]
```

```
head(ethereum)
```

```
##           date     open
## 516 2017-01-03  8.37458
## 517 2017-01-04  9.70929
## 518 2017-01-05 11.28680
## 519 2017-01-06 10.28580
## 520 2017-01-07 10.24090
## 521 2017-01-08  9.87258
```

```
head(google)
```

```
##            date   open
```

```
## 3116 2017-01-03 778.81
## 3117 2017-01-04 788.36
## 3118 2017-01-05 786.08
## 3119 2017-01-06 795.26
## 3120 2017-01-09 806.40
## 3121 2017-01-10 807.86
```

```
head(gme)
```

```
##            date  open
## 1025 2017-01-03 25.44
## 1024 2017-01-04 25.58
## 1023 2017-01-05 25.56
## 1022 2017-01-06 25.15
## 1021 2017-01-09 24.60
## 1020 2017-01-10 24.52
```

```
head(doge)
```

```
##           date     open
## 840 2017-01-03 0.000224
## 841 2017-01-04 0.000227
## 842 2017-01-05 0.000233
## 843 2017-01-06 0.000228
## 844 2017-01-07 0.000222
## 845 2017-01-08 0.000224
```

b. Plot the moving average for each dataset and find the MSE. Which MA is the best choice?

```
library("TTR")
```

```
## Warning: package 'TTR' was built under R version 3.6.2
```

```
eth_ts <- ts(ethereum$open)


week <- as.vector(SMA(eth_ts, n=7))
mse <- (week - ethereum$open)^2
```

```
sum(na.omit(mse))
```

```
## [1] 1248156
```

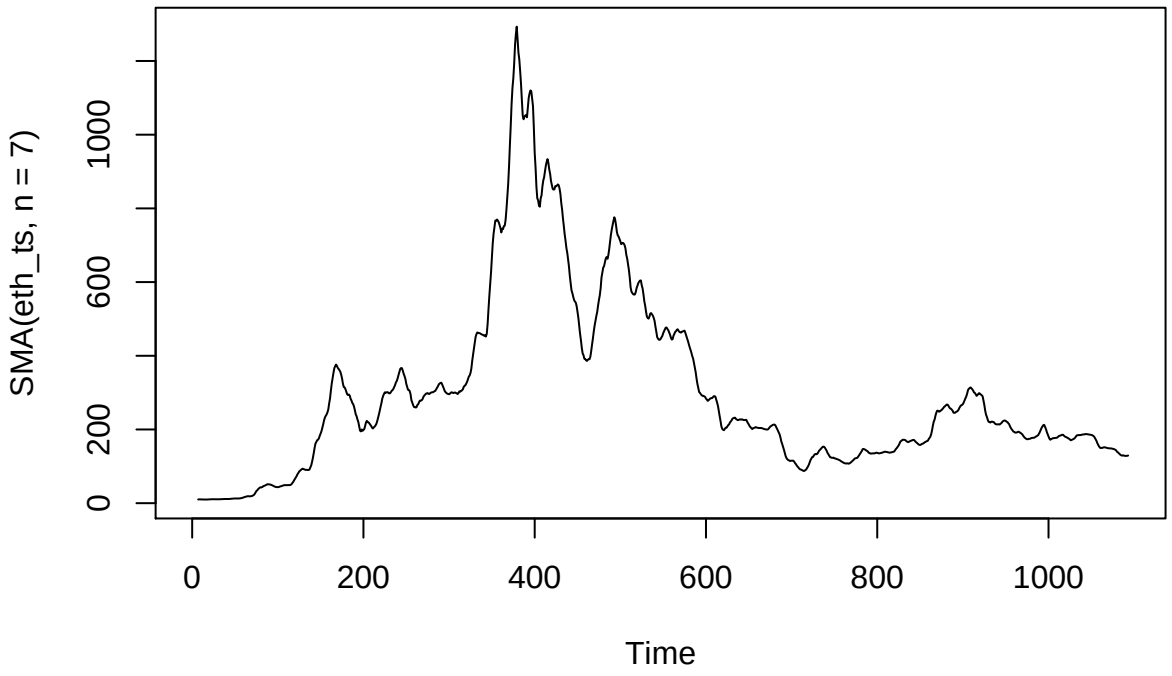
```
month <- as.vector(SMA(eth_ts, n=30))
mse <- (month - ethereum$open)^2
sum(na.omit(mse))
```

```
## [1] 7274145
```

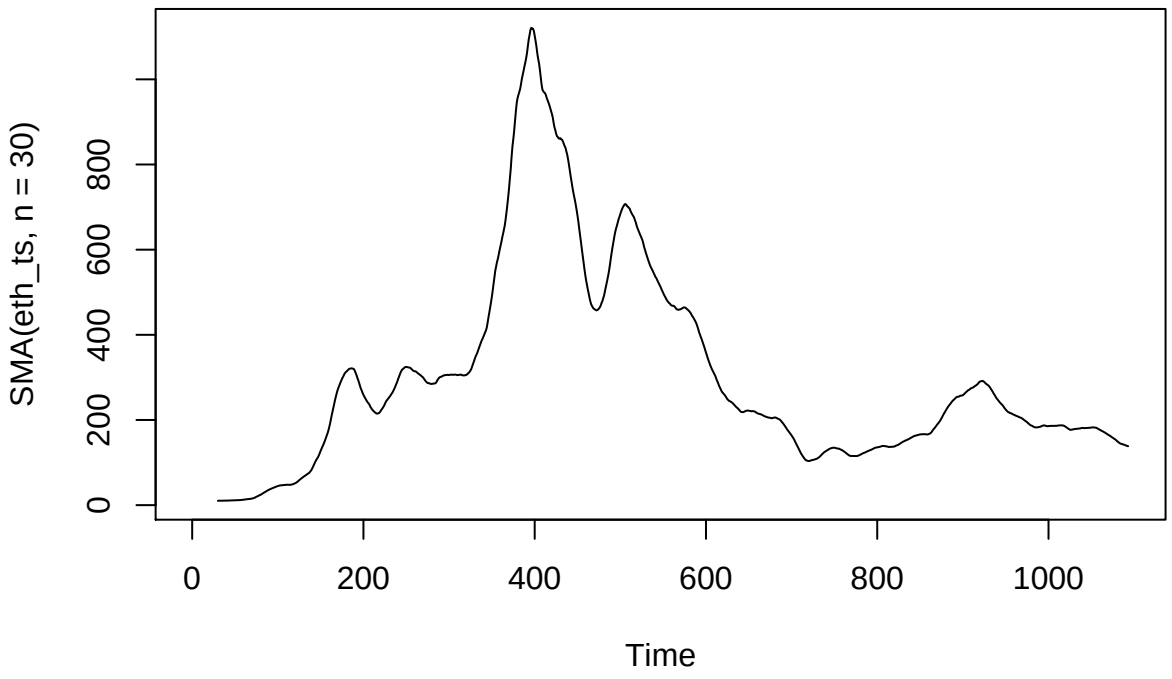
```
three_month <- as.vector(SMA(eth_ts, n=90))
mse <- (three_month - ethereum$open)^2
sum(na.omit(mse))
```

```
## [1] 23143297
```

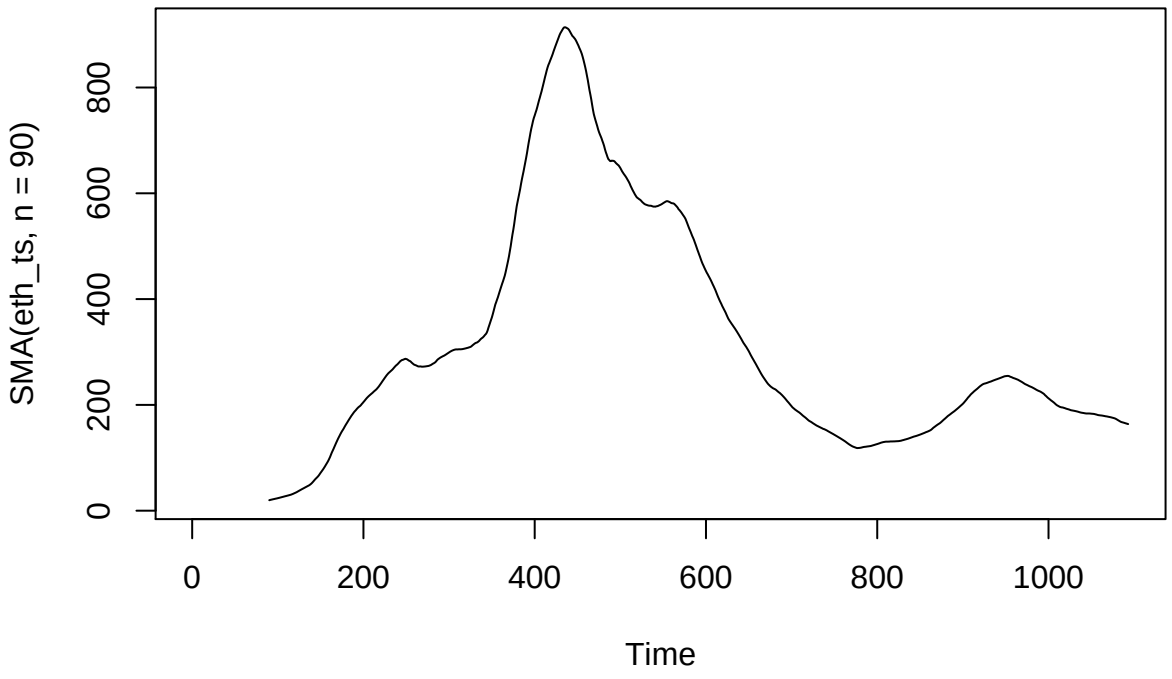
```
plot.ts(SMA(eth_ts, n=7))
```




```
plot.ts(SMA(eth_ts, n=30))
```




```
plot.ts(SMA(eth_ts, n=90))
```



We can see that the MSE was minimized under the 7 day moving average for ethereum. We will repeat for the other 3.

```
google_ts <- ts(google$open)


week <- as.vector(SMA(google_ts, n=5))
mse <- (week - google$open)^2
```

```
sum(na.omit(mse))
```

```
## [1] 213869.4
```

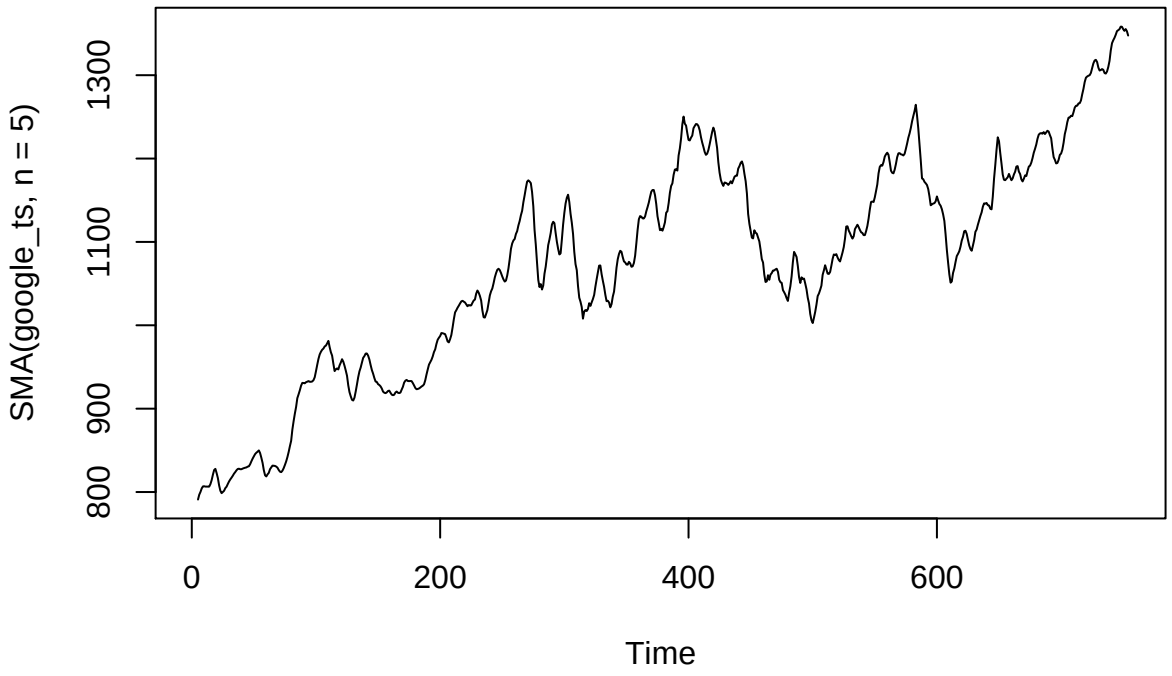
```
month <- as.vector(SMA(google_ts, n=20))
mse <- (month - google$open)^2
sum(na.omit(mse))
```

```
## [1] 810730.5
```

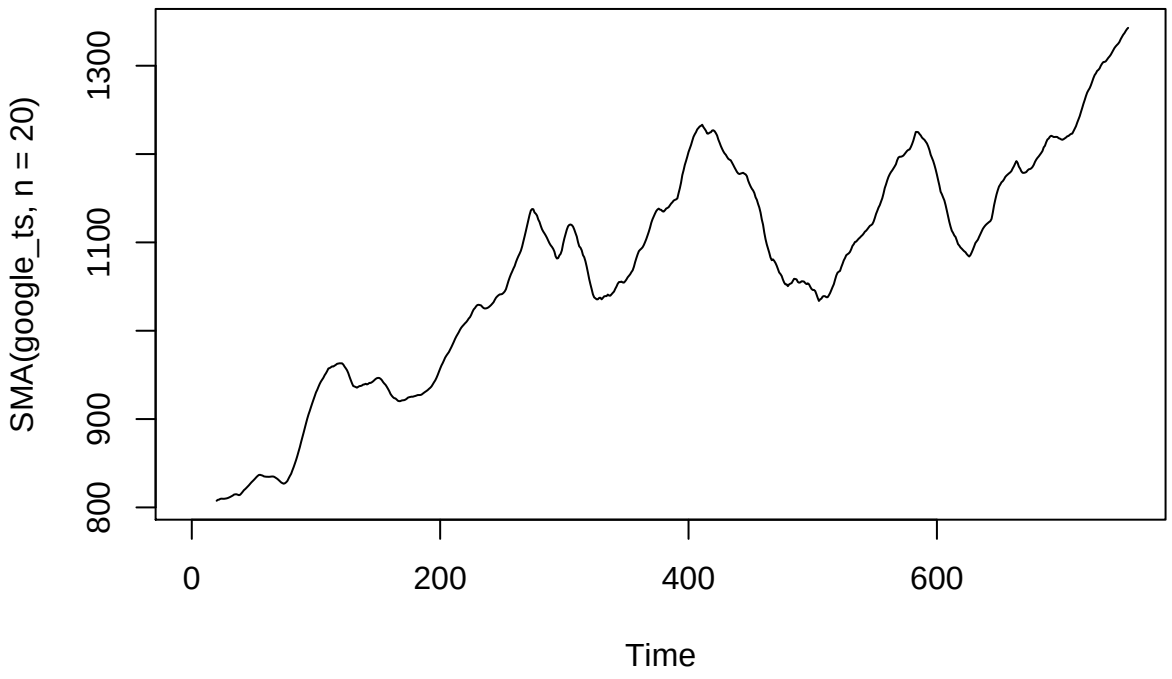
```
three_month <- as.vector(SMA(google_ts, n=60))
mse <- (three_month - google$open)^2
sum(na.omit(mse))
```

```
## [1] 2465235
```

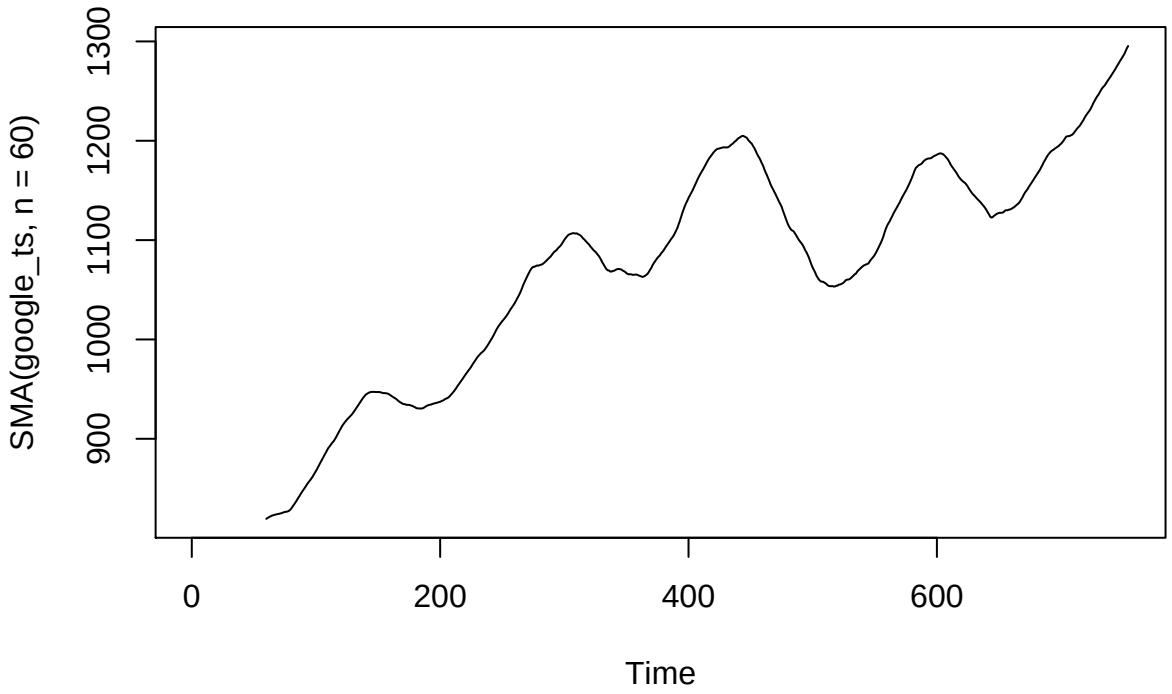
```
plot.ts(SMA(google_ts, n=5))
```




```
plot.ts(SMA(google_ts, n=20))
```




```
plot.ts(SMA(google_ts, n=60))
```



First I want to say that for Google (and GME when we get there), I am using 5 days for a week, 20 days for a month, and 60 days for 3 months because stocks (not cryptos), are only on the open market during weekdays. There is no data for Saturday and Sunday. We can see that once again our MSE is minimized for the 5 day moving average.

```
gme_ts <- ts(gme$open)


week <- as.vector(SMA(gme_ts, n=5))
```

```
mse <- (week - gme$open)^2
sum(na.omit(mse))
```

```
## [1] 172.6898
```

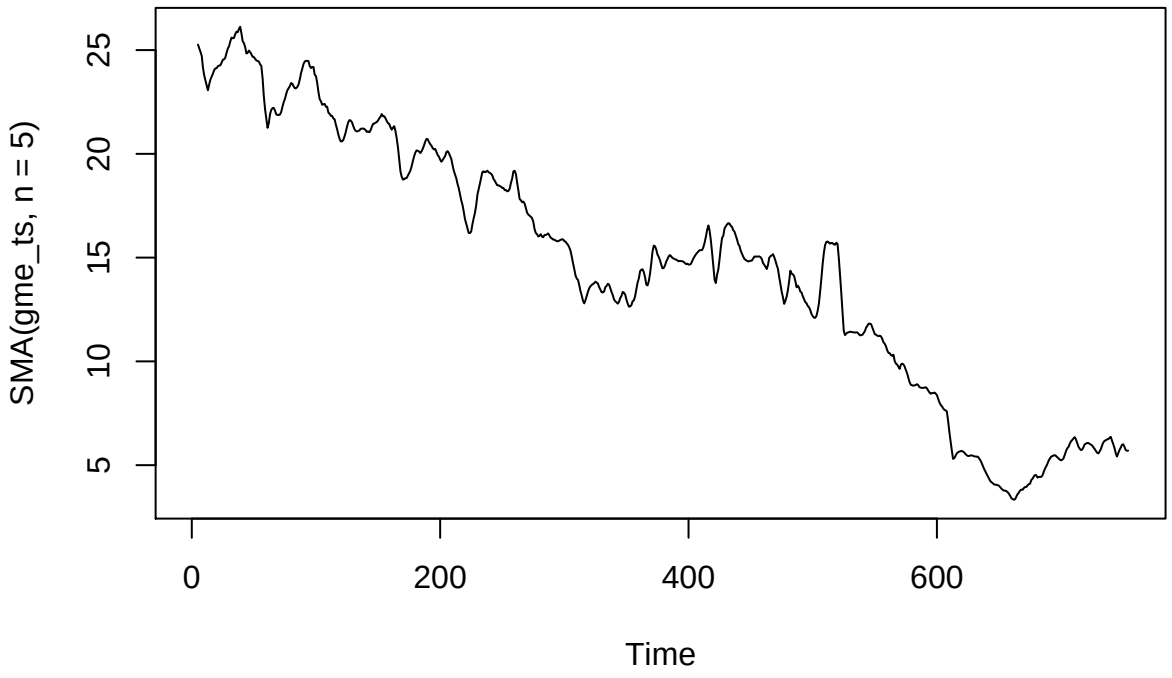
```
month <- as.vector(SMA(gme_ts, n=20))
mse <- (month - gme$open)^2
sum(na.omit(mse))
```

```
## [1] 704.1467
```

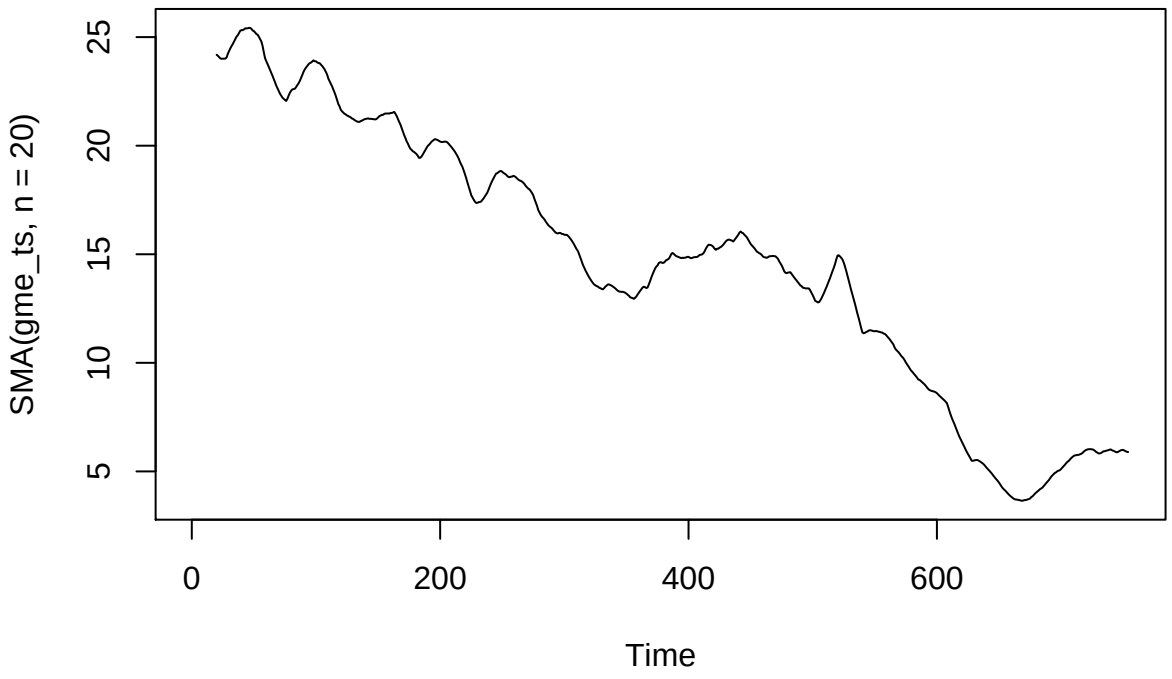
```
three_month <- as.vector(SMA(gme_ts, n=60))
mse <- (three_month - gme$open)^2
sum(na.omit(mse))
```

```
## [1] 1587.242
```

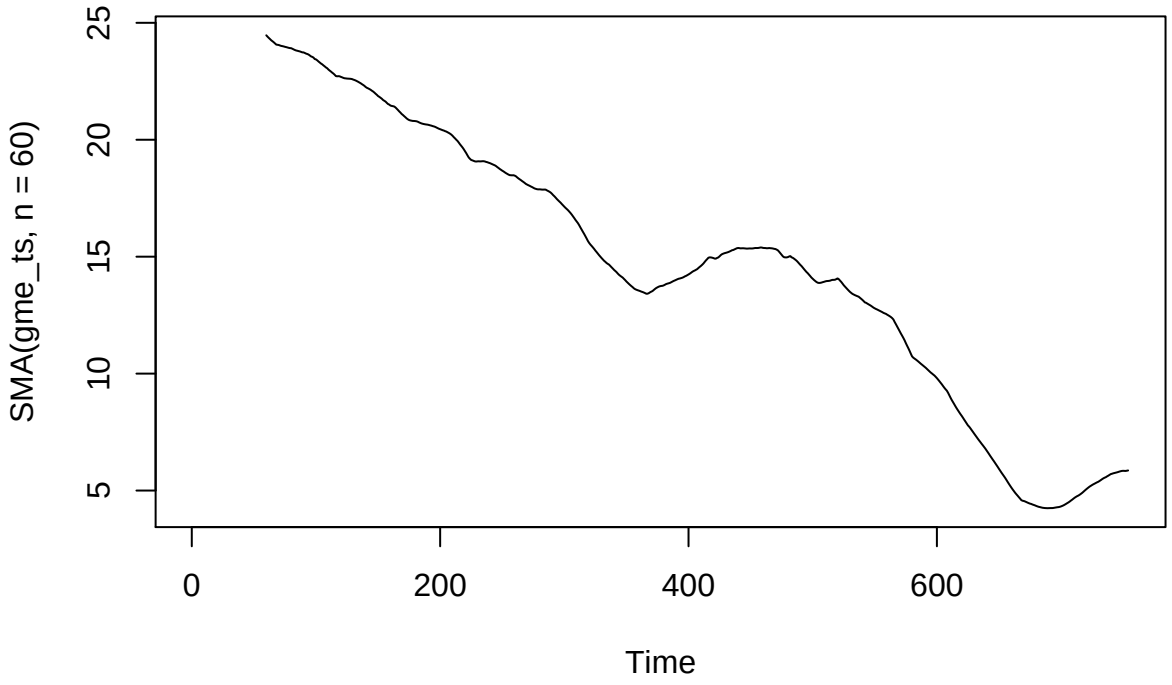
```
plot.ts(SMA(gme_ts, n=5))
```




```
plot.ts(SMA(gme_ts, n=20))
```




```
plot.ts(SMA(gme_ts, n=60))
```



Same result. 5 day moving average minimizes MSE here.

```
doge_ts <- ts(doge$open)


week <- as.vector(SMA(doge_ts, n=7))
mse <- (week - doge$open)^2
sum(na.omit(mse))
```

```
## [1] 0.0002296515
```

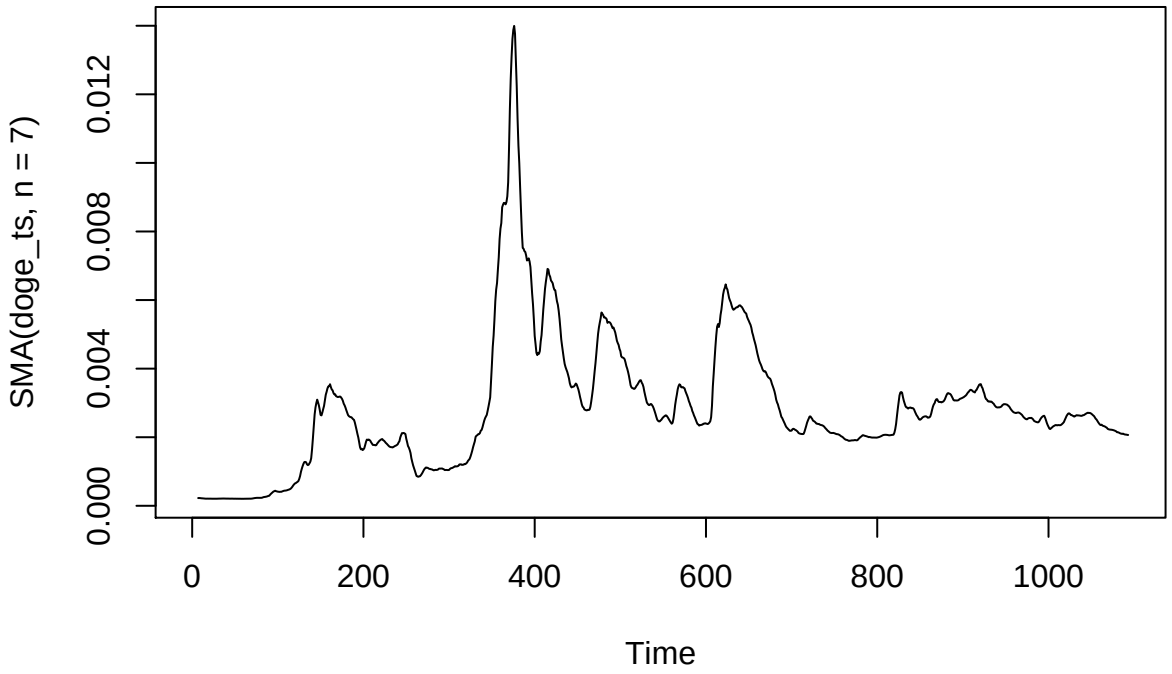
```
month <- as.vector(SMA(doge_ts, n=30))
mse <- (month - doge$open)^2
sum(na.omit(mse))
```

```
## [1] 0.001318933
```

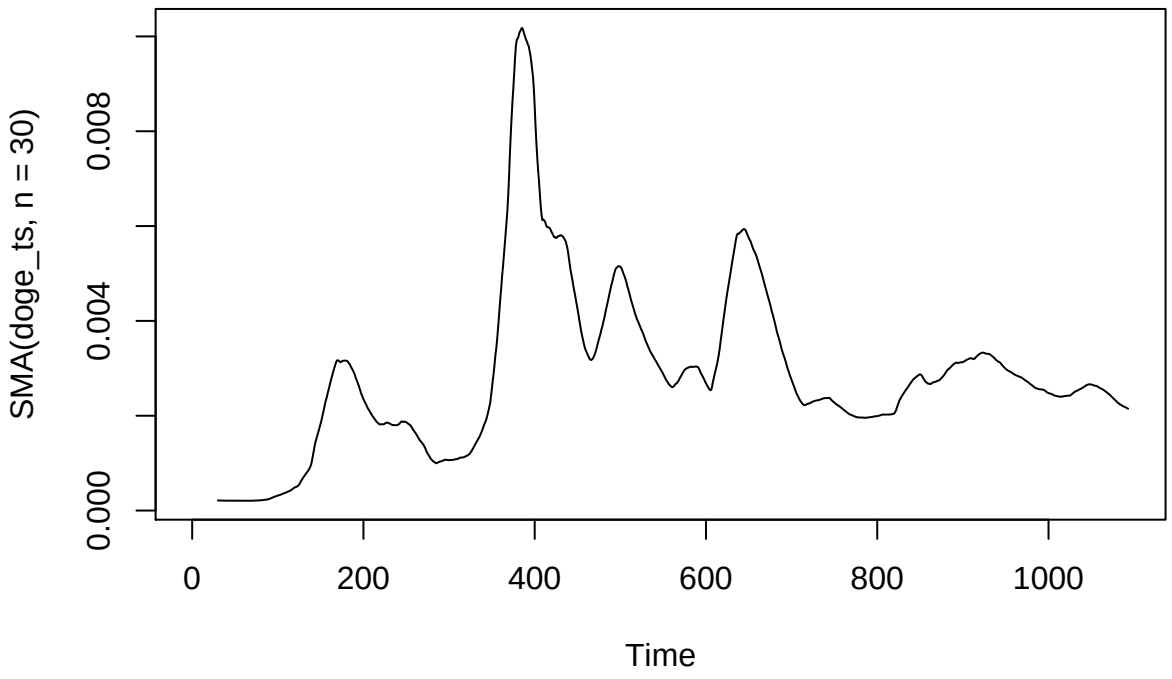
```
three_month <- as.vector(SMA(doge_ts, n=90))
mse <- (three_month - doge$open)^2
sum(na.omit(mse))
```

```
## [1] 0.003037907
```

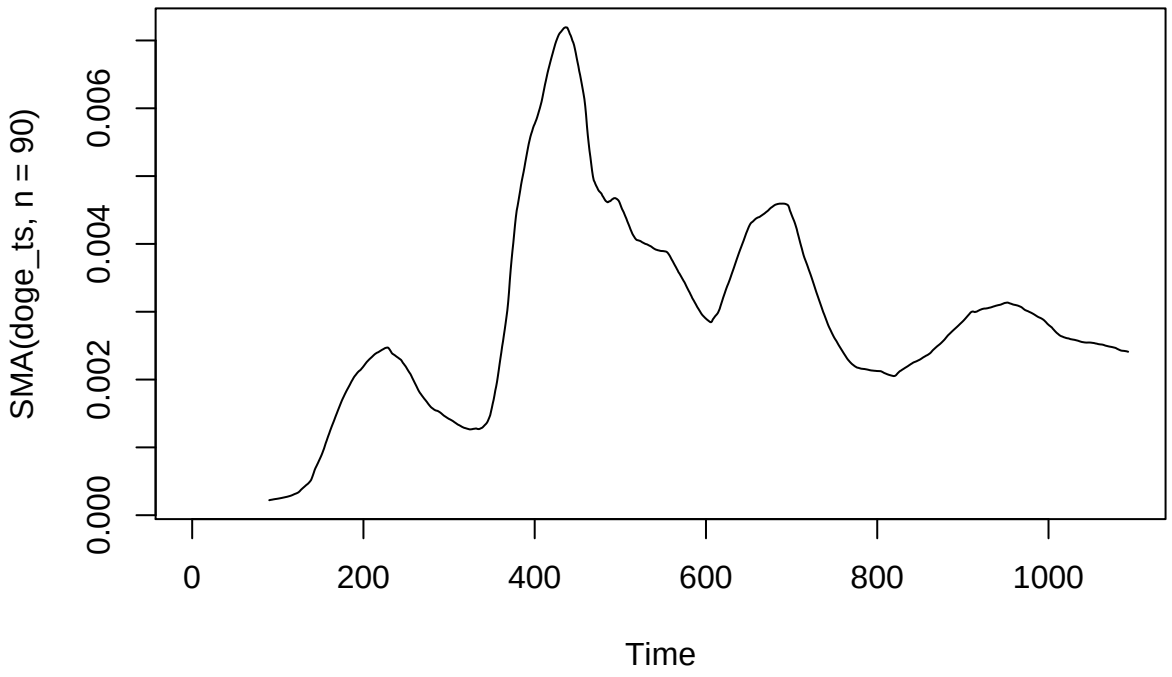
```
plot.ts(SMA(doge_ts, n=7))
```




```
plot.ts(SMA(doge_ts, n=30))
```




```
plot.ts(SMA(doge_ts, n=90))
```



Once again, 7 day moving average has minimized MSE. I can confidently state that using the moving average for 1 week is a better overall time window for this type of data.

c. Create a new column in the data for if with the first standard deviation of the mean or above or below it. BECAUSE Cryptos have data on the weekends, we will need to remove these prices so that all of our datasets cover the exact same dates. If we do not make this change, our conditional probability will calculated using the wrong context of dates.

```
## make sure ethereum and doge now have the proper dates and weekends cut off
common_dates <- as.Date(intersect(ethereum$date, gme$date), origin = '1970-01-01')
ethereum <- subset(ethereum, date %in% common_dates)
doge <- subset(doge, date %in% common_dates)
```

```
ethereum$stdev <- NA
google$stdev <- NA
gme$stdev <- NA
doge$stdev <- NA
ethereum$mean <- NA
google$mean <- NA
gme$mean <- NA
doge$mean <- NA

## insert the SD into stdev column for each dataset for the previous 30 days
x = 31
while (x < 755){
  ethereum$stdev[x] <- sd(ethereum$open[(x-30):x])
  google$stdev[x] <- sd(google$open[(x-30):x])
  gme$stdev[x] <- sd(gme$open[(x-30):x])
  doge$stdev[x] <- sd(doge$open[(x-30):x])
  ethereum$mean[x] <- mean(ethereum$open[(x-30):x])
  google$mean[x] <- mean(google$open[(x-30):x])
  gme$mean[x] <- mean(gme$open[(x-30):x])
  doge$mean[x] <- mean(doge$open[(x-30):x])
  x = x + 1
}
```

```
ethereum$stdev_class <- NA
x = 31
while (x < 755){
  if (ethereum$open[x] > (ethereum$mean[x] + ethereum$stdev[x])){
    ethereum$stdev_class[x] <- 'high'
  }
  else if(ethereum$open[x] < (ethereum$mean[x] - ethereum$stdev[x])){
    ethereum$stdev_class[x] <- 'low'
  }
  else{
    ethereum$stdev_class[x] <- 'medium'
  }
  x = x + 1
}
```

```
google$stdev_class <- NA
x = 31
while (x < 755){
  if (google$open[x] > (google$mean[x] + google$stdev[x])){
    google$stdev_class[x] <- 'high'
  }
  else if(google$open[x] < (google$mean[x] - google$stdev[x])){
    google$stdev_class[x] <- 'low'
  }
  else{
    google$stdev_class[x] <- 'medium'
  }
  x = x + 1
}
```

```
gme$stdev_class <- NA
x = 31
while (x < 755){
  if (gme$open[x] > (gme$mean[x] + gme$stdev[x])){
    gme$stdev_class[x] <- 'high'
```

```
  }
  else if(gme$open[x] < (gme$mean[x] - gme$stdev[x])){
    gme$stdev_class[x] <- 'low'
  }
  else{
    gme$stdev_class[x] <- 'medium'
  }
  x = x + 1
}
```

```
doge$stdev_class <- NA
x = 31
while (x < 755){
  if (doge$open[x] > (doge$mean[x] + doge$stdev[x])){
    doge$stdev_class[x] <- 'high'
  }
  else if(doge$open[x] < (doge$mean[x] - doge$stdev[x])){
    doge$stdev_class[x] <- 'low'
  }
  else{
    doge$stdev_class[x] <- 'medium'
  }
  x = x + 1
}
```

c. (cont.) Find the conditional probability

```
## create vectors for t, t-1, t-2, and t-3
eth_t3 <- ethereum$stdev_class[31:751]
eth_t2 <- ethereum$stdev_class[32:752]
eth_t1 <- ethereum$stdev_class[33:753]
eth_t <- ethereum$stdev_class[34:754]

# find the numerator of the conditional probability
num <- sum(eth_t == 'high' & eth_t1 == 'low' & eth_t2 == 'low' & eth_t3 == 'low')
```

```
num
```

```
## [1] 0
```

```
# find the denominator of the conditional probability
den <- sum(eth_t1 == 'low' & eth_t2 == 'low' & eth_t3 == 'low')
den
```

```
## [1] 141
```

```
# divide for conditional probability
num/den
```

```
## [1] 0
```

Based on our calculations, if there are three straight days in the low category, the probability of the next day being in the high category is zero.

```
gme_t1 <- gme$stdev_class[31:753]
goog_t1 <-google$stdev_class[31:753]
doge_t1 <- doge$stdev_class[31:753]
eth_t <- ethereum$stdev_class[32:754]


# find the numerator of the conditional probability
num <- sum(eth_t == 'high' & gme_t1 == 'low' & goog_t1 == 'low' & doge_t1 == 'low')
num
```

```
## [1] 0
```

```
# find the denominator of the conditional probability
den <- sum(gme_t1 == 'low' & goog_t1 == 'low' & doge_t1 == 'low')
den
```

```
## [1] 22
```

```
# divide for conditional probability
num/den
```

```
## [1] 0
```

Based on our calculations, if the previous day produced low prices for GME, Google, and Doge, the probability of Ethereum being in the high price category is once again zero.

This does make sense when you think about how rare that particular sequence of events should be, but it is curious

that yet another probability is zero.

I want to at least show that my data is not screwed up and that a different conditional probability would yield a non-zero result, so I will show the probability of ethereum yielding the high category given the previous three time periods were medium.

Just an example for show, not a real question

```
## create vectors for t, t-1, t-2, and t-3
eth_t3 <- ethereum$stdev_class[31:751]
eth_t2 <- ethereum$stdev_class[32:752]
eth_t1 <- ethereum$stdev_class[33:753]
eth_t <- ethereum$stdev_class[34:754]


# find the numerator of the conditional probability
num <- sum(eth_t == 'high' & eth_t1 == 'medium' & eth_t2 == 'medium' & eth_t3 == 'medium')
num
```

```
## [1] 14
```

```
# find the denominator of the conditional probability
den <- sum(eth_t1 == 'medium' & eth_t2 == 'medium' & eth_t3 == 'medium')
den
```

```
## [1] 234
```

```
# divide for conditional probability
num/den
```

```
## [1] 0.05982906
```

Just for show, we can see that if the previous three categories of ethereum were medium, then the probability of the next day being high would be 6%

d. Fit a multiple regression model for ethereum based upon its price for the previous 4 days. I will still use the data without the weekend datapoints as Question 5 will require us to use the Google data. This is for simplicities sake and comparison reasons.

```r
eth_t4 <- ethereum$open[1:750]
eth_t3 <- ethereum$open[2:751]
eth_t2 <- ethereum$open[3:752]
eth_t1 <- ethereum$open[4:753]
eth_t <- ethereum$open[5:754]

ethereum_model <- lm(eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4)
summary(ethereum_model)
```

```
##
## Call:
## lm(formula = eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -211.979   -6.104   -1.940    6.251  196.374
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.75369    1.61323   1.707  0.08825 .
## eth_t1       1.00662    0.03640  27.657  < 2e-16 ***
## eth_t2       0.06689    0.05180   1.291  0.19703
## eth_t3       0.03201    0.05181   0.618  0.53686
## eth_t4      -0.11435    0.03638  -3.144  0.00174 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27.87 on 745 degrees of freedom
## Multiple R-squared:  0.9869, Adjusted R-squared:  0.9868
## F-statistic: 1.4e+04 on 4 and 745 DF,  p-value: < 2.2e-16
```

d. (cont.) add in the doge parameters and compare the models

```r
eth_t4 <- ethereum$open[1:750]
eth_t3 <- ethereum$open[2:751]
eth_t2 <- ethereum$open[3:752]
eth_t1 <- ethereum$open[4:753]
eth_t <- ethereum$open[5:754]

doge_t4 <- doge$open[1:750]
doge_t3 <- doge$open[2:751]
doge_t2 <- doge$open[3:752]
doge_t1 <- doge$open[4:753]

ethereum_and_doge_model <- lm(eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4 +
                                doge_t1 + doge_t2 + doge_t3 + doge_t4)
summary(ethereum_and_doge_model)
```

```
##
## Call:
## lm(formula = eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4 + doge_t1 +
##     doge_t2 + doge_t3 + doge_t4)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -182.106   -6.398   -1.129    6.143  190.193
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.364e+00  1.780e+00   0.766  0.44390
## eth_t1       9.420e-01  4.122e-02  22.855  < 2e-16 ***
## eth_t2       2.987e-02  5.675e-02   0.526  0.59880
## eth_t3       1.707e-01  5.684e-02   3.004  0.00276 **
## eth_t4      -1.560e-01  3.989e-02  -3.911  0.00010 ***
## doge_t1      7.548e+03  2.782e+03   2.713  0.00682 **
## doge_t2      9.789e+03  3.960e+03   2.472  0.01367 *
## doge_t3     -2.116e+04  3.961e+03  -5.341 1.23e-07 ***
```

```
## doge_t4      4.783e+03  2.834e+03   1.688  0.09187 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27.02 on 741 degrees of freedom
## Multiple R-squared:  0.9877, Adjusted R-squared:  0.9876
## F-statistic:  7453 on 8 and 741 DF,  p-value: < 2.2e-16
```

attempt to find redundancies in our model and remove them

```
ethereum_and_doge_model_remove_redundant <- lm(eth_t ~ eth_t1 + eth_t3 + eth_t4 +
                                doge_t1 + doge_t2 + doge_t3)
summary(ethereum_and_doge_model_remove_redundant)
```

```
##
## Call:
## lm(formula = eth_t ~ eth_t1 + eth_t3 + eth_t4 + doge_t1 + doge_t2 +
##     doge_t3)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -183.016   -6.376   -1.137    6.262  191.699
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.434e+00  1.782e+00   0.805 0.421092
## eth_t1       9.463e-01  2.844e-02  33.273  < 2e-16 ***
## eth_t3       1.673e-01  4.707e-02   3.555 0.000402 ***
## eth_t4      -1.260e-01  3.570e-02  -3.529 0.000442 ***
## doge_t1      7.494e+03  2.552e+03   2.936 0.003422 **
## doge_t2      1.023e+04  3.528e+03   2.899 0.003852 **
## doge_t3     -1.689e+04  2.673e+03  -6.321 4.47e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 27.05 on 743 degrees of freedom
## Multiple R-squared:  0.9877, Adjusted R-squared:  0.9876
## F-statistic:  9918 on 6 and 743 DF,  p-value: < 2.2e-16
```

The fit from ethereum only to ethereum and doge together in the same model did improve the fit of our model, but the improvement was more or less negligible. If we were looking for absolute simplicity, there would be no need to include the data on doge into our model.

We can find some redundant parameters, by removing some parameters that were not significant at the five percent level, we managed to keep our R^2 value exactly the same. The RSE value increased only .03 with this change. We removed eth_t2 and doge_t4.

Question 5: Code and Results Used in Presentation

```
eth_t9 <- ethereum$open[1:745]
eth_t8 <- ethereum$open[2:746]
eth_t7 <- ethereum$open[3:747]
eth_t6 <- ethereum$open[4:748]
eth_t5 <- ethereum$open[5:749]
eth_t4 <- ethereum$open[6:750]
eth_t3 <- ethereum$open[7:751]
eth_t2 <- ethereum$open[8:752]
eth_t1 <- ethereum$open[9:753]
eth_t <- ethereum$open[10:754]


ethereum_model_more_dates <- lm(eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4 + eth_t5 + eth_t6
                                + eth_t7 + eth_t8 + eth_t9)
ethereum_model_4 <- lm(eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4)
summary(ethereum_model_more_dates)
```

```
##
## Call:
## lm(formula = eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4 + eth_t5 +
##     eth_t6 + eth_t7 + eth_t8 + eth_t9)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -183.210   -6.259   -1.727    6.995  172.332
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.17746    1.61627   1.347  0.17833
## eth_t1       1.00113    0.03666  27.305  < 2e-16 ***
## eth_t2       0.05911    0.05190   1.139  0.25512
## eth_t3       0.06200    0.05171   1.199  0.23096
## eth_t4      -0.13852    0.05159  -2.685  0.00741 **
## eth_t5      -0.01279    0.05184  -0.247  0.80522
## eth_t6      -0.11590    0.05159  -2.247  0.02497 *
## eth_t7       0.13305    0.05172   2.573  0.01029 *
## eth_t8      -0.10404    0.05191  -2.004  0.04541 *
## eth_t9       0.10927    0.03665   2.981  0.00296 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27.56 on 735 degrees of freedom
## Multiple R-squared:  0.9872, Adjusted R-squared:  0.9871
## F-statistic:  6305 on 9 and 735 DF,  p-value: < 2.2e-16
```

```
anova(ethereum_model_4, ethereum_model_more_dates)
```

```
## Analysis of Variance Table
##
## Model 1: eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4
## Model 2: eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4 + eth_t5 + eth_t6 +
##     eth_t7 + eth_t8 + eth_t9
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    740 578549
## 2    735 558229  5     20320 5.3509 7.691e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
eth_t4 <- ethereum$open[1:750]
eth_t3 <- ethereum$open[2:751]
```

```r
eth_t2 <- ethereum$open[3:752]
eth_t1 <- ethereum$open[4:753]
eth_t <- ethereum$open[5:754]

google_t4 <- google$open[1:750]
google_t3 <- google$open[2:751]
google_t2 <- google$open[3:752]
google_t1 <- google$open[4:753]

ethereum_and_google_model <- lm(eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4 +
                                google_t1 + google_t2 + google_t3 + google_t4)
summary(ethereum_and_google_model)
```

```
##
## Call:
## lm(formula = eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4 + google_t1 +
##     google_t2 + google_t3 + google_t4)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -213.771   -6.539   -1.392    6.653  197.810
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 11.996942   8.284848   1.448  0.14802
## eth_t1       1.000573   0.036705  27.260  < 2e-16 ***
## eth_t2       0.063113   0.051890   1.216  0.22427
## eth_t3       0.037527   0.051907   0.723  0.46992
## eth_t4      -0.109213   0.036616  -2.983  0.00295 **
## google_t1    0.045904   0.063445   0.724  0.46958
## google_t2    0.069404   0.085579   0.811  0.41763
## google_t3   -0.119813   0.085641  -1.399  0.16223
## google_t4   -0.004451   0.063655  -0.070  0.94427
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27.84 on 741 degrees of freedom
## Multiple R-squared:  0.987,  Adjusted R-squared:  0.9868
## F-statistic:  7013 on 8 and 741 DF,  p-value: < 2.2e-16
```

```
anova(ethereum_model, ethereum_and_google_model)
```

```
## Analysis of Variance Table
##
## Model 1: eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4
## Model 2: eth_t ~ eth_t1 + eth_t2 + eth_t3 + eth_t4 + google_t1 + google_t2 +
##     google_t3 + google_t4
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    745 578590
## 2    741 574530  4    4060.2 1.3092 0.2649
```

```
eth_xgb1 <- data.frame(eth_t, eth_t1, eth_t2, eth_t3, eth_t4)


eth_xgb1 <- as.matrix(eth_xgb1)


library(xgboost)
```

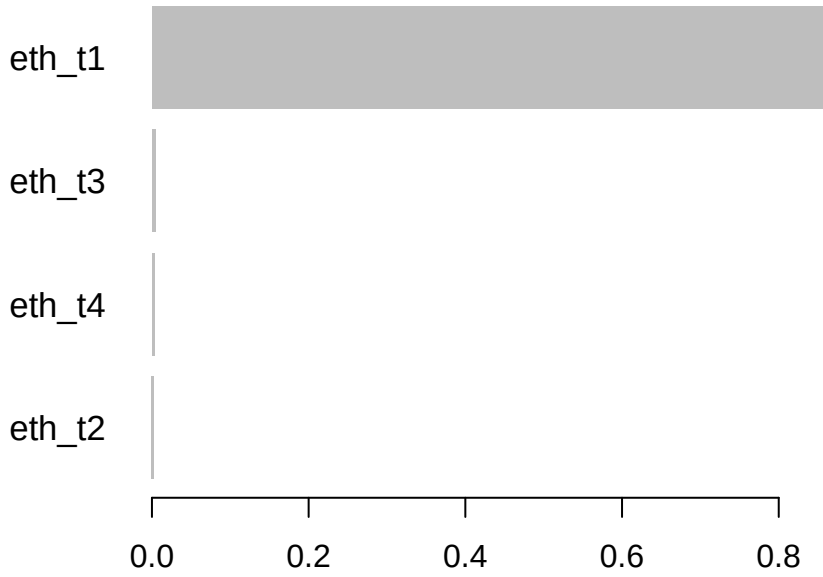
```
## Warning: package 'xgboost' was built under R version 3.6.2
##
## Attaching package: 'xgboost'
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
train = eth_xgb1[1:650,]
test = eth_xgb1[651:750,]



xtrain = train[,-1]
ytrain = train[,1]
```

```
xtest = test[,-1]
ytest = test[,1]


# create xgb model
xgb = xgboost(xtrain, label = ytrain, eta = 0.3, nrounds = 750, verbose = 0, prediction = TRUE)
```

```
## [18:14:24] WARNING: amalgamation/../src/learner.cc:573:
## Parameters: { "prediction" } might not be used.
##
##   This may not be accurate due to some parameters are only used in language bindings but
##   passed down to XGBoost core.  Or some parameters are not used but slip through this
##   verification. Please open an issue if you find above cases.
```

```
xgb.plot.importance(xgb.importance(model = xgb))
```




```
print(xgb.importance(model = xgb))
```

```
##    Feature        Gain     Cover Frequency
## 1:  eth_t1 0.989959475 0.2553408 0.3735433
## 2:  eth_t3 0.004403694 0.2637369 0.2115262
## 3:  eth_t4 0.002991160 0.2154501 0.1834875
## 4:  eth_t2 0.002645671 0.2654722 0.2314429
```

```
#RSME
sqrt(mean((ytest - predict(xgb, xtest))^2))
```

```
## [1] 14.42571
```

```
df <- data.frame("index" = 1:100, "predict" = predict(xgb, xtest), "actual" = ytest)

ggplot(data = df) + geom_point(aes(x = index, y = predict, color = 'prediction')) +
  geom_point(aes(x = index, y = actual, color = 'actual')) +
  geom_line(aes(x = index, y = actual, color = 'actual')) +
  geom_line(aes(x = index, y = predict, color = 'prediction')) +
  xlab("Daily Index") +
  ylab("Price")
```