Code Sample

Wayne Monical

3/3/2021

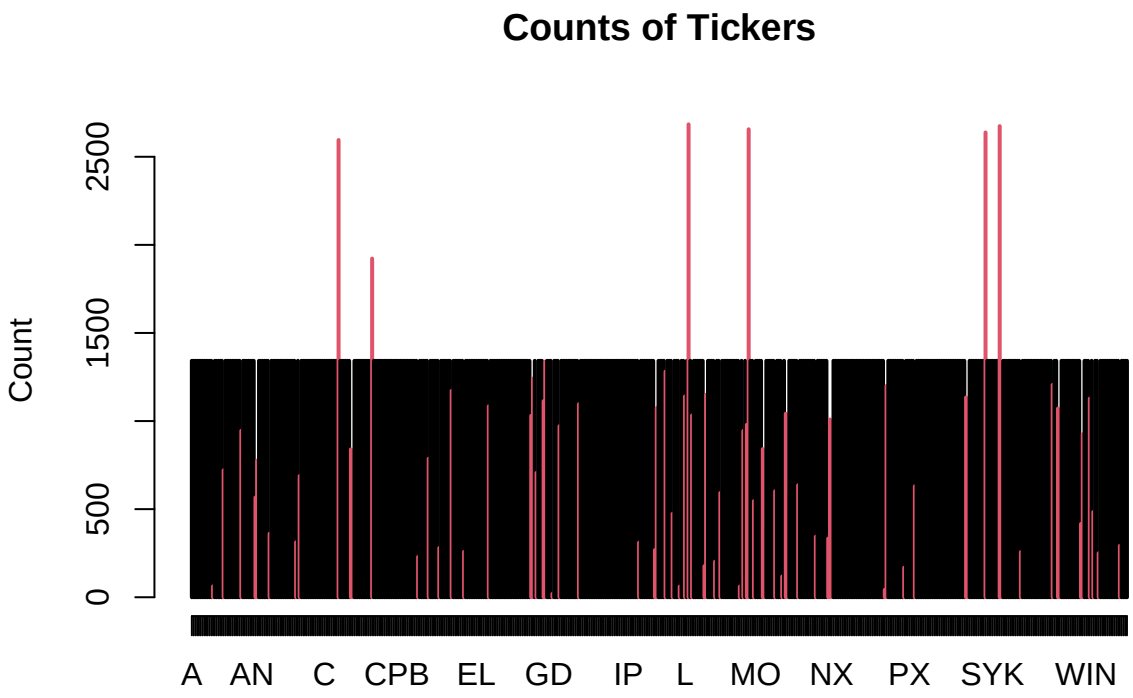
Part 1: Data Cleaning and Correcting

Loading and cleaning the data frame. Each row of this data frame contains the information of one stock's price on one day between 2012 and 2016. Highlights include correcting the date information for weekends, holidays and leap years, as well as correcting negative entries.

```
library(dplyr)

# data load
stock_rawdata <- read.csv("stock_data.csv", header = TRUE)
```

Displayed here are the number of times that each stock appears in the data frame. The vast majority of stocks appear exactly 1342 times. The stocks that do not appear this number of times are highlighted in red.

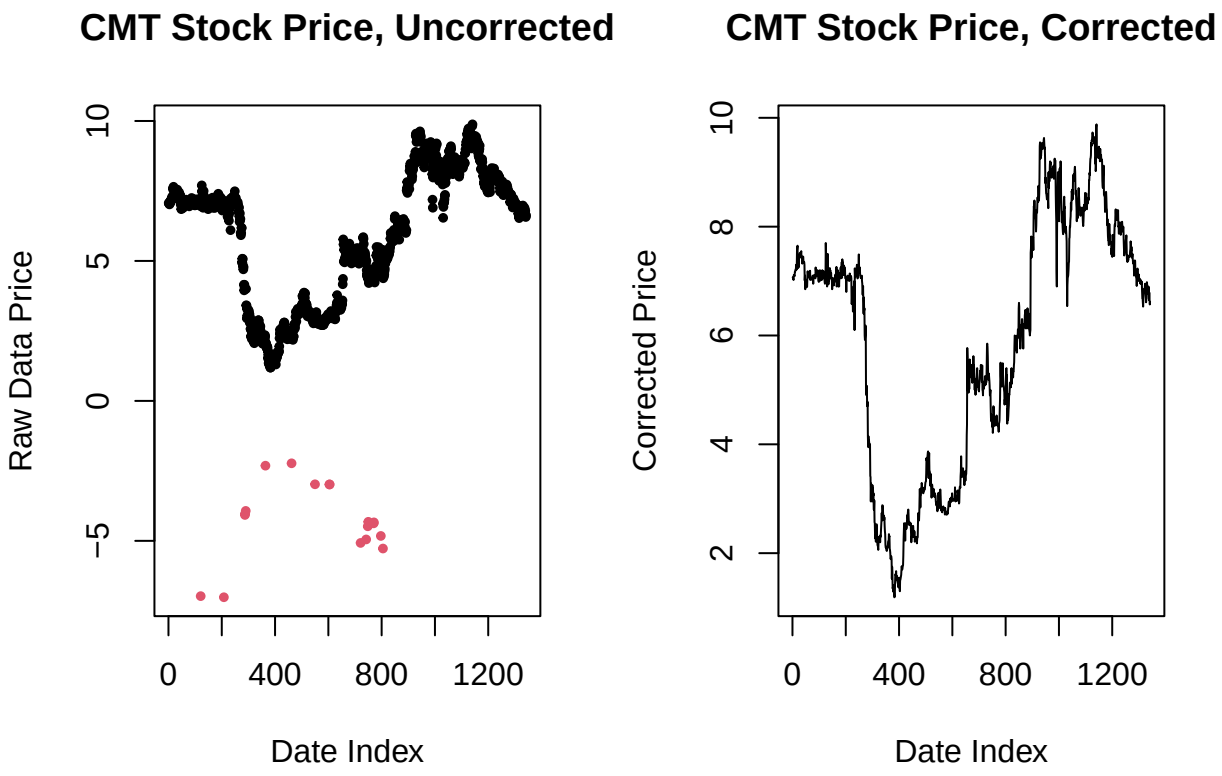
```
plot(table(stock$TICKER),
     main = "Counts of Tickers",
     ylab = "Count",
     col = factor(!stock_boolean))
```




```
## [1] 1342
```

```
# turning the numerical date vector to a more friendly format.
# note: we have to specify the format that the date vector is in, i.e. '%Y%m%d'
dateform <- as.Date(as.character(stock$date), format = '%Y%m%d')
```

```
# plotting corrected and uncorrected side by side
par(mfrow = c(1, 2))
plot(cmt$PRC, pch = 20, main = "CMT Stock Price, Uncorrected",
     xlab = "Date Index", ylab = "Raw Data Price",
     col = factor(cmt$PRC < 0), cex = 0.8)
plot(abs(cmt$PRC), type = 'l', main = "CMT Stock Price, Corrected",
     xlab = "Date Index" , ylab = "Corrected Price")
```




```
# Fixing the data
stock_new <- mutate(stock_new, PRC = abs(stock_new$PRC))
```

Part 2: kNN Prediction Function

Here, I construct a simple function to predict an unknown variable from a data set using a simple averaging of its K nearest neighbors. It takes K, the number of nearest numbers, the training set, and the value to predict at as arguments. When K equal to N, where N is the number of data points in the training set, the function will return a simple average. `knn.predict` relies on the auxiliary function `closest.k`, which indicates which data points are to be used in prediction.

```
# predicts the response at value, from data
knn.predict <- function(value, response, data, k = 1){
  mat_closest <- closest.k(k = k, vec = data, value = value)

  temp_index <- mat_closest[1:k,1]


  closest_response <- response[temp_index]

  return(sum(closest_response) / k)
}
```

The `knn.apply` function is a simple wrapper for the `knn.predict` function so that it may be called on a sequence of values. This function is helpful when applying `knn.predict` on multiple training sets in parallel.

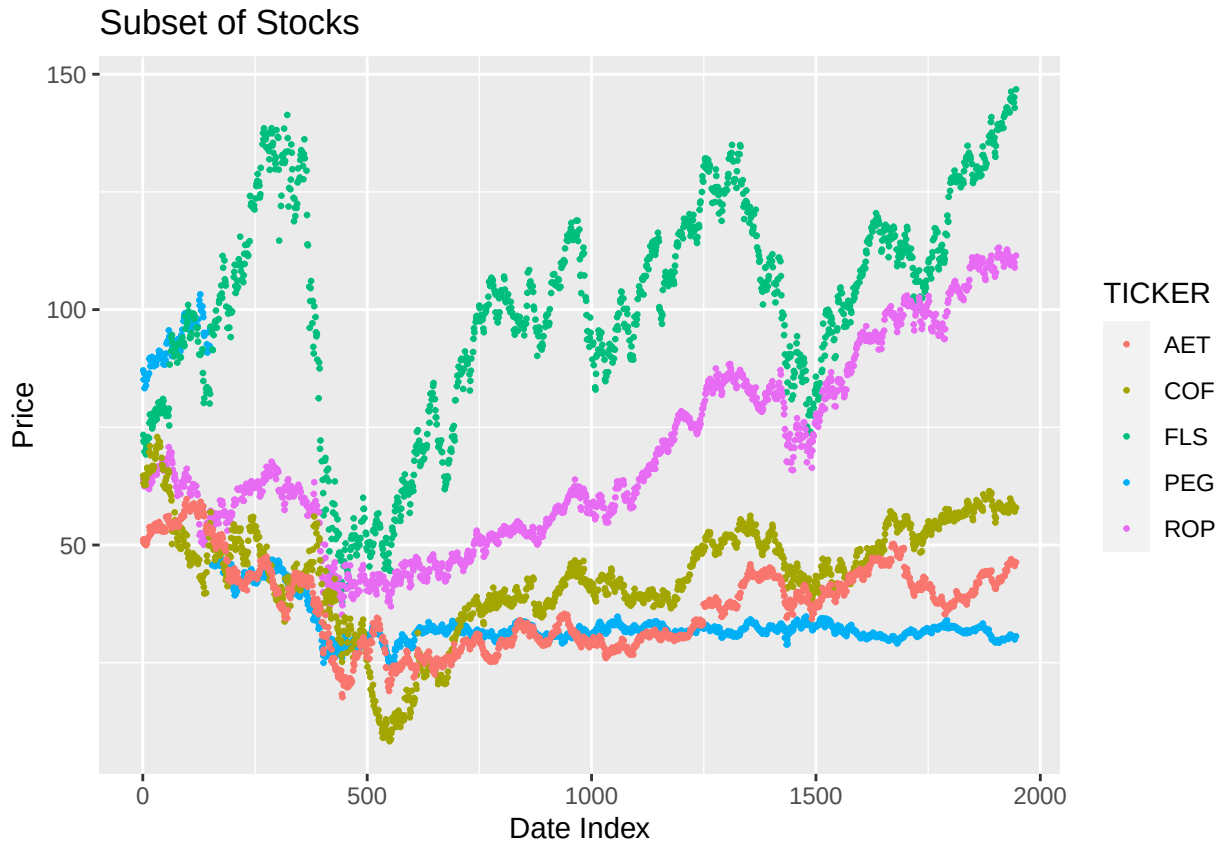
```
# apply wrapper
knn.apply <- function(sequence, response, data, k){
```

```
  answer <- sapply(X = sequence, FUN = function(x){knn.predict(value = x, response = response, data = da
  return(answer)
}
```

Part 3: Graphics and Analysis

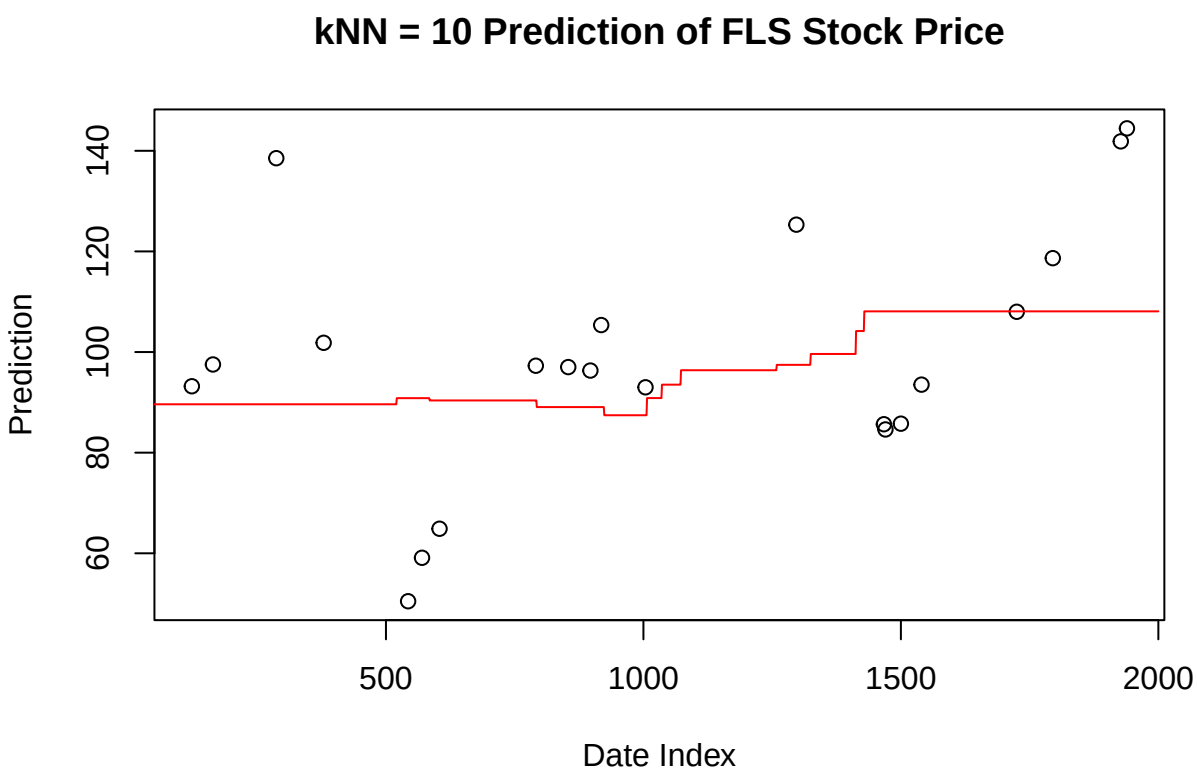
Here, I take an initial look at a small subset of stocks and their prices over the time period. For a simple analysis, I will split the data into a training and test set, then apply the K nearest neighbors prediction function to each stock's training set in parallel. I will plot my predictions with the training set, and finally aggregate a table displaying each model's mean squared-error on its training and test set.

```
library(ggplot2)

# initial plot
ggplot(data = stock_subset, aes(x = date_num,  y = PRC)) + geom_point(aes(col = TICKER), size = 0.5) +
  ggtitle("Subset of Stocks") +
  labs(x = "Date Index", y = "Price")
```



The line of code below predicts the values of the FLS stock with the K nearest neighbors method. here, K is equal to 10

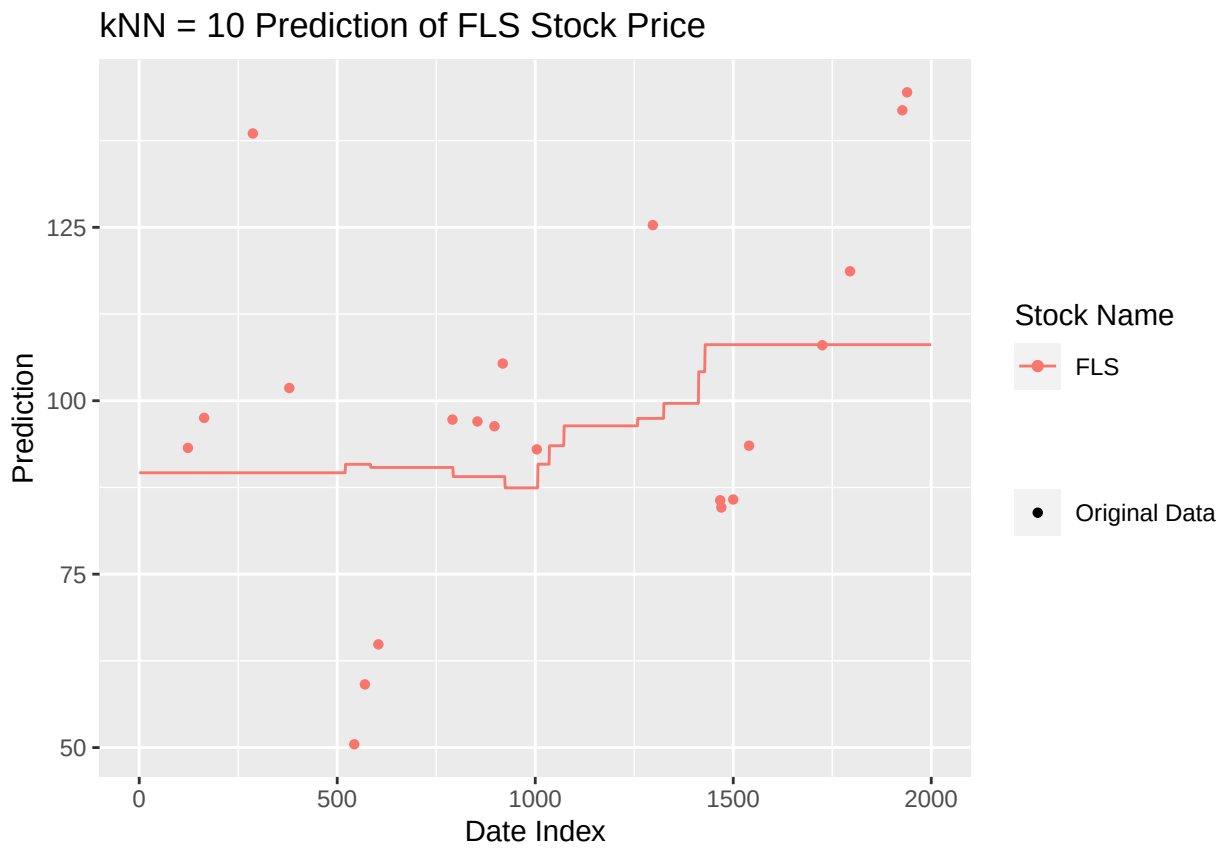
```
# applying kNN to FLS on date index
predict_FLS <- knn.apply(sequence = date_sequence, response = train_FLS$PRC, data = train_FLS$date_num,
```




```
## integer(0)
```

Here,

```
ggplot(data = train_FLS, ) +
         geom_point(data = subset(train_FLS, data_type == "Original Data"),
                    aes(x = date_num, y = PRC, color = TICKER, shape = data_type)) +
         geom_line(data = subset(train_FLS, data_type == "Prediction"),
                   aes(x = date_num, y = PRC, color = TICKER)) +
         ggtitle(label = "kNN = 10 Prediction of FLS Stock Price") +
         labs(x = "Date Index", y = "Prediction", col = "Stock Name", shape = "")
```



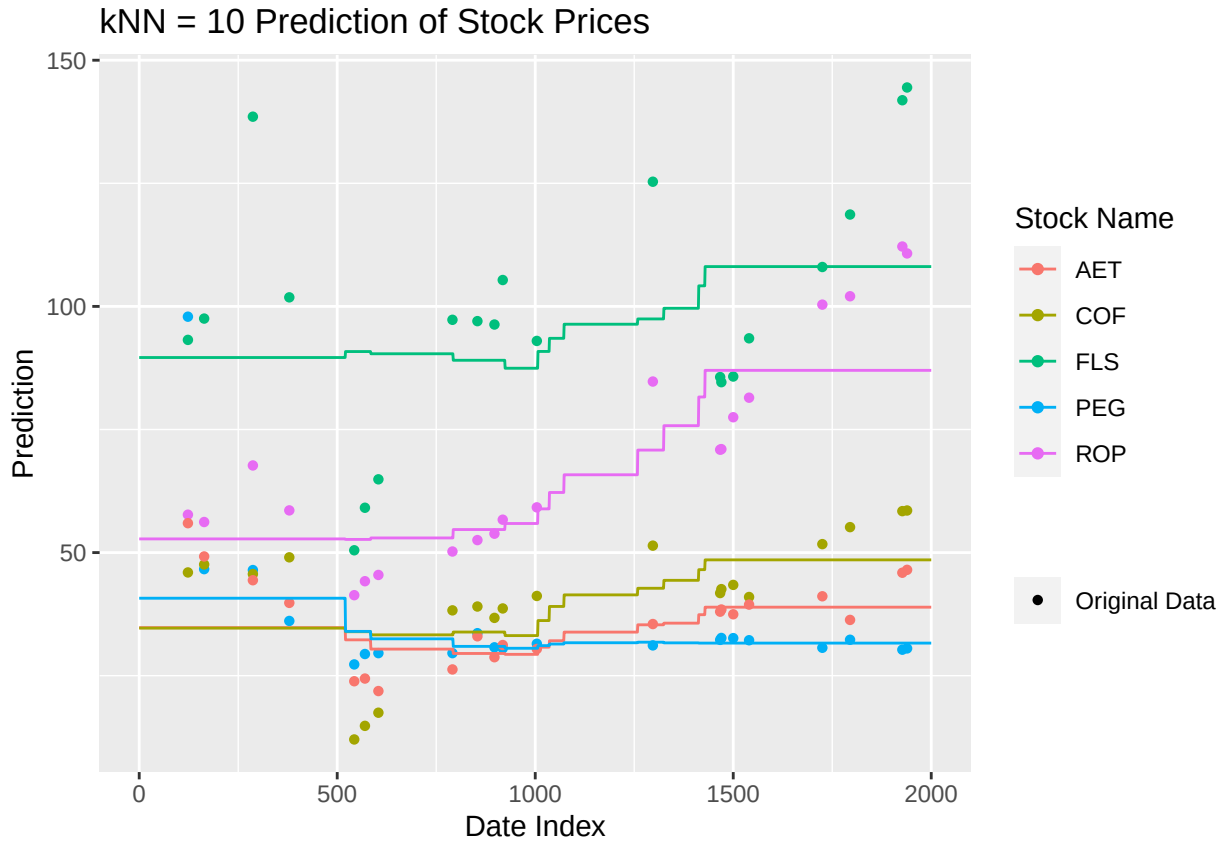
With a for-loop, we can apply the exact same process to all stocks in the subset. Here, the extra steps of filtering and binding to construct the ggplot data frames are not omitted. Finally, all ggplot data frames are row-binded so that they may be plotted together.

```
# predicing the other stock prices from kNN
for(g in ticker_subset){

  df_g <- filter(stock_train, TICKER == g)
  predict_g <- knn.apply(sequence = date_sequence, response = df_g$PRC,
                         data = df_g$date_num, k = k_star)

  df_predict_g <- data.frame(TICKER = g, PRC = predict_g,
                             date_num = date_sequence, data_type = "Prediction")


  stock_train <- rbind(stock_train, df_predict_g)
}
```

On this graph, the points represent training set data. The lines represent the K nearest neighbor method's predictions. Training sets are predictions are matched by stock, indicated with color, as specified by the legend. These graphic elements are added with additional functions `geom_point`, `geom_line`, and `ggtitle`. One subtle but imperative point in this graphic's programming is the liberal use of the `subset` function. This function differentiates between the observed and predicted data, i.e. which data must be represented by points and which must be represented by lines. Both are simply encoded as rows in data frame.

```
  #### Initialize Plot

        ggplot(data = stock_train) +
```

```
#### Data Elements

       geom_point(data = subset(stock_train, data_type == "Original Data"),
                  aes(x = date_num, y = PRC, color = TICKER, shape = data_type)) +

       geom_line(data = subset(stock_train, data_type == "Prediction"),
                 aes(x = date_num, y = PRC, color = TICKER)) +


#### Label Elements

       ggtitle(label = paste0("kNN = ", k_star," Prediction of Stock Prices")) +

       labs(x = "Date Index", y = "Prediction", col = "Stock Name", shape = "")
```



Here I use simple aggregation method to compare the mean squared-error of the models on the training and test set. Comparison of these errors are essential to validating models and avoiding over-fitting. The error rates on the test set are of course higher.

```
empirical_error[1,] <- aggregate(stock_merge_train["Error"], list(stock_merge_train$TICKER), mean)$x
empirical_error[2,] <- aggregate(stock_merge_test["Error"], list(stock_merge_test$TICKER), mean)$x


empirical_error
```

```
##                COF      FLS      AET      ROP      PEG
## test     5.201048 9.333381 19.31757 4.891143 9.800190
## training 5.786827 8.468502 17.94117 6.164363 8.670027
```