Apple Stock Adata Analysis & Prediction Models

Rena Haswah Anna Kong Christopher Hong

10/20/2019

Data Processing

```
library(zoo)
library(xts)
library(quantmod)
```

```
# Import the apple stock from yahoo & save it as xts object
aapl_xts <- getSymbols(Symbols = "AAPL", scr = "yahoo", from = "2014-09-30", to = "2019-11-20", auto.as
```

```
# Save the xts object to rds file
#saveRDS(object = aapl_xts, file = "aapl_xts.rds")
# Read the aapl_xts data from the rds file
#aapl_xts <- readRDS("aapl_xts.rds")

# Export the xts object to a csv file
#write.zoo(aapl_xts, file = "aapl_xts.csv", sep = ",")

# Open the saved object
#aapl_zoo <- read.zoo("aapl_xts.csv", sep = ",", FUN = as.Date, header = TRUE, index.column = 1)

# Encode the new bojet back into xts
#aapl_xts <- as.xts(aapl_zoo)
```

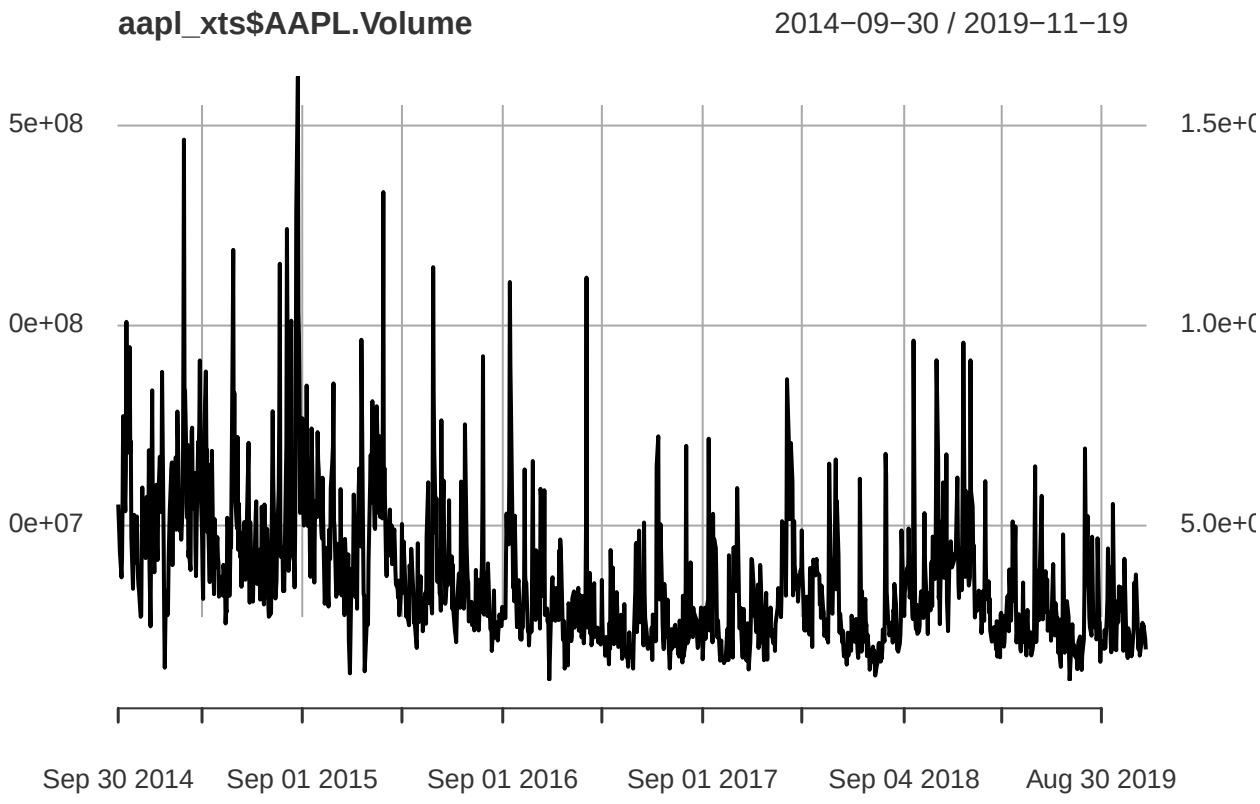
```
# View all columns
plot.xts(aapl_xts$AAPL.Open)
```


```
plot.xts(aapl_xts$AAPL.High)
```

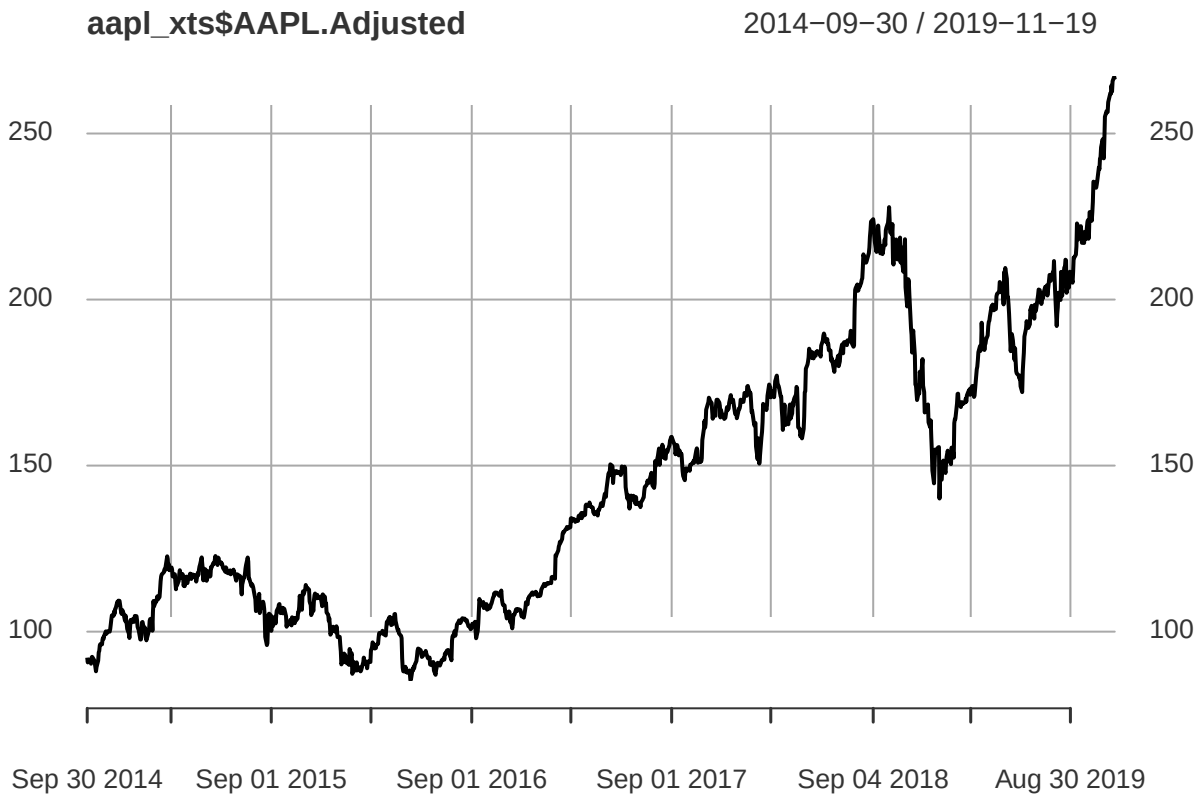

```
plot.xts(aapl_xts$AAPL.Low)
```


```
plot.xts(aapl_xts$AAPL.Close)
```


```
plot.xts(aapl_xts$AAPL.Volume)
```

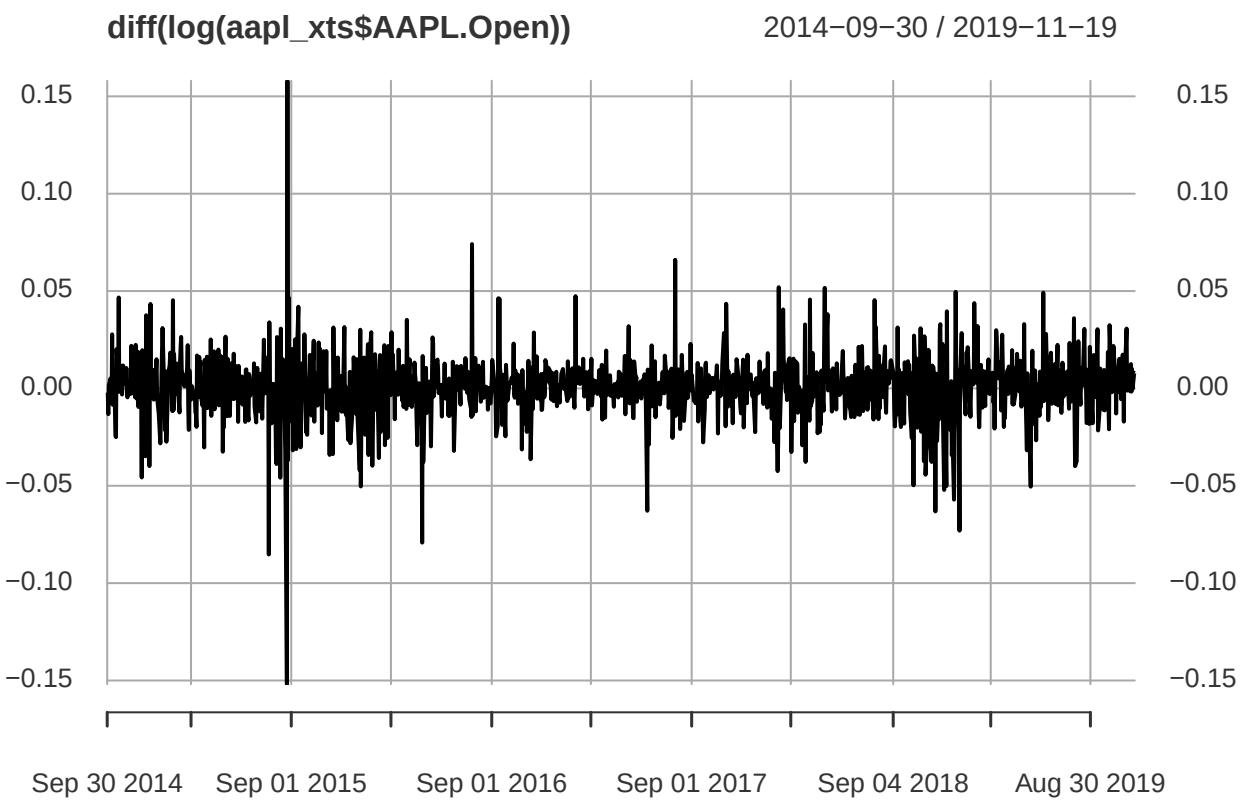



```
plot.xts(aapl_xts$AAPL.Adjusted)
```

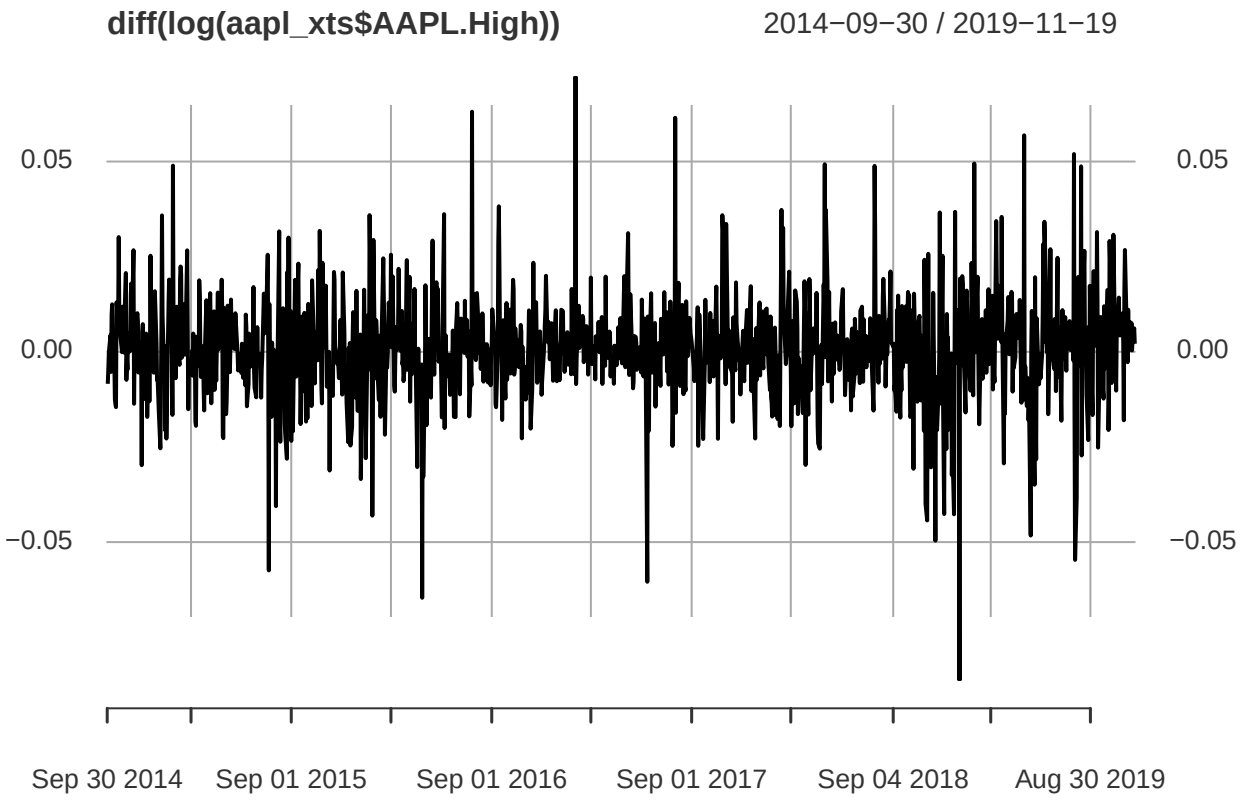


The time series data is an approximate trend and peroidicity.

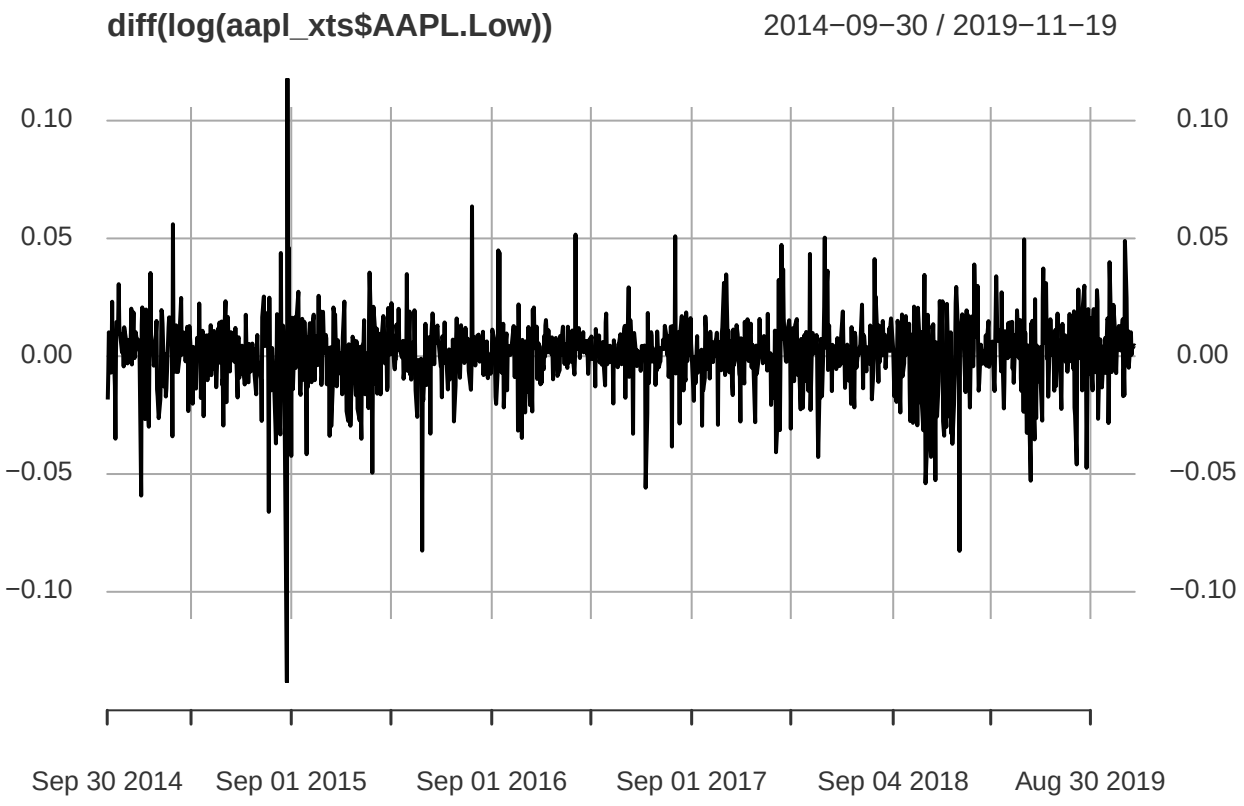
```
# Detranding all columns
plot.xts(diff(log(aapl_xts$AAPL.Open)))
```



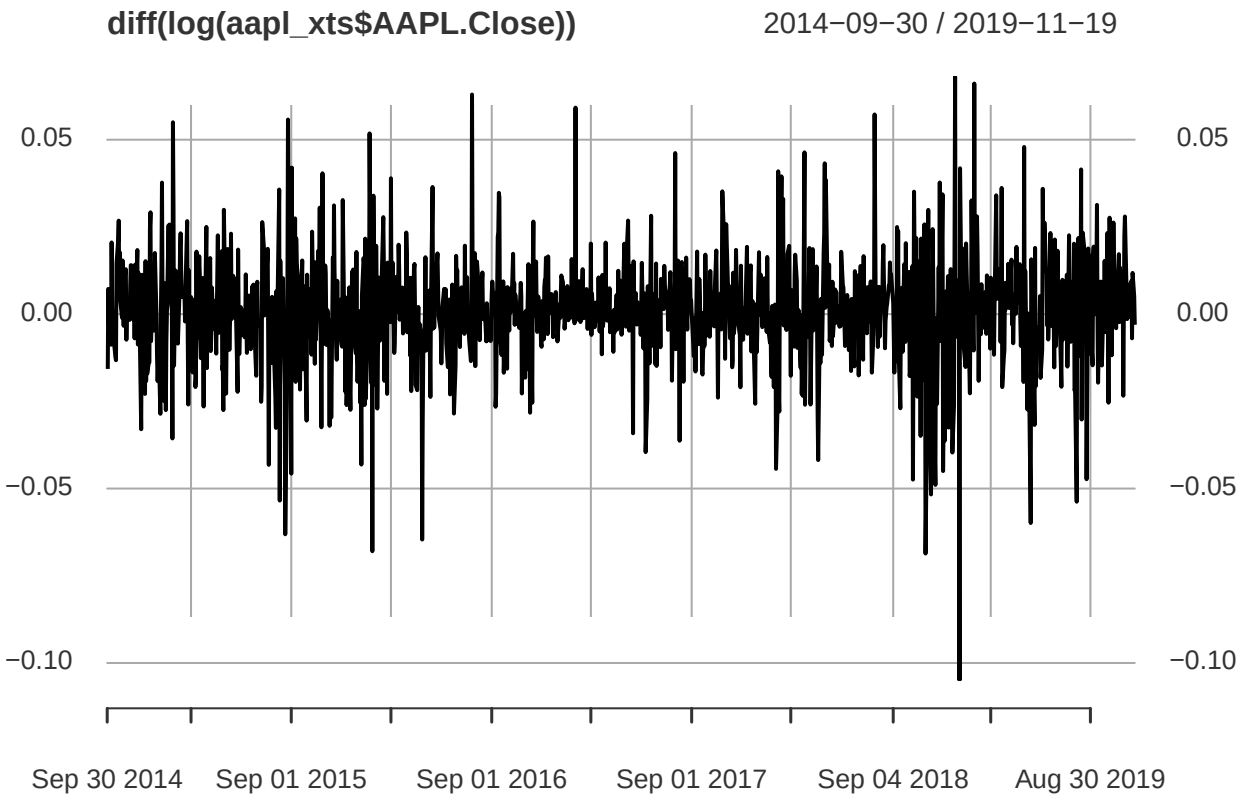

```
plot.xts(diff(log(aapl_xts$AAPL.High)))
```



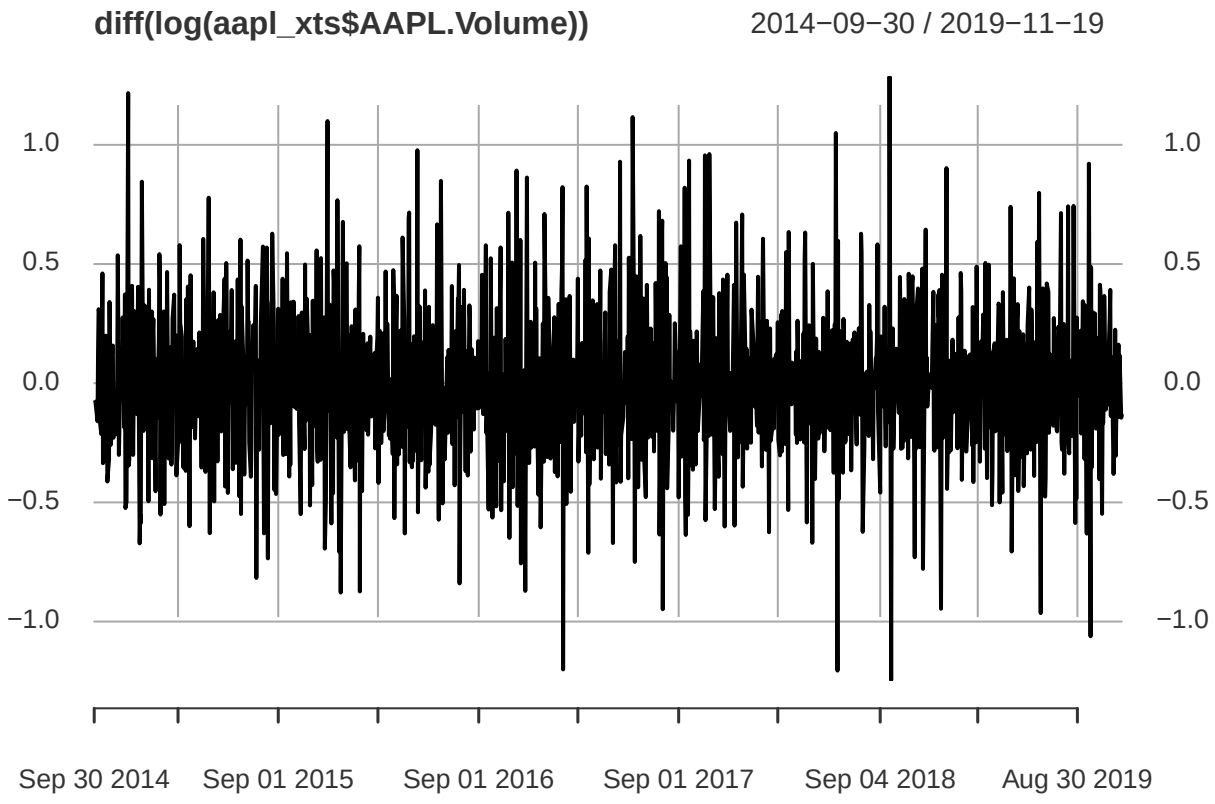

```
plot.xts(diff(log(aapl_xts$AAPL.Low)))
```



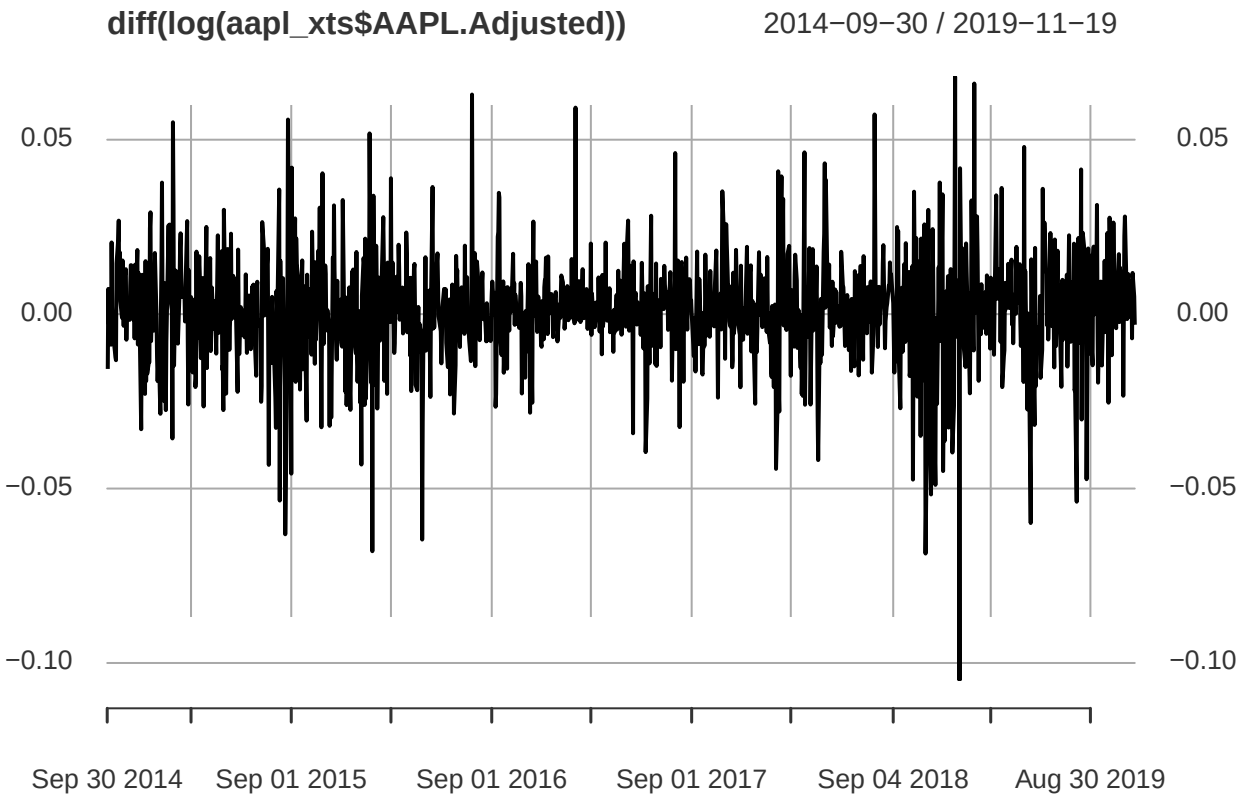

```
plot.xts(diff(log(aapl_xts$AAPL.Close)))
```




```
plot.xts(diff(log(aapl_xts$AAPL.Volume)))
```




```
plot.xts(diff(log(aapl_xts$AAPL.Adjusted)))
```



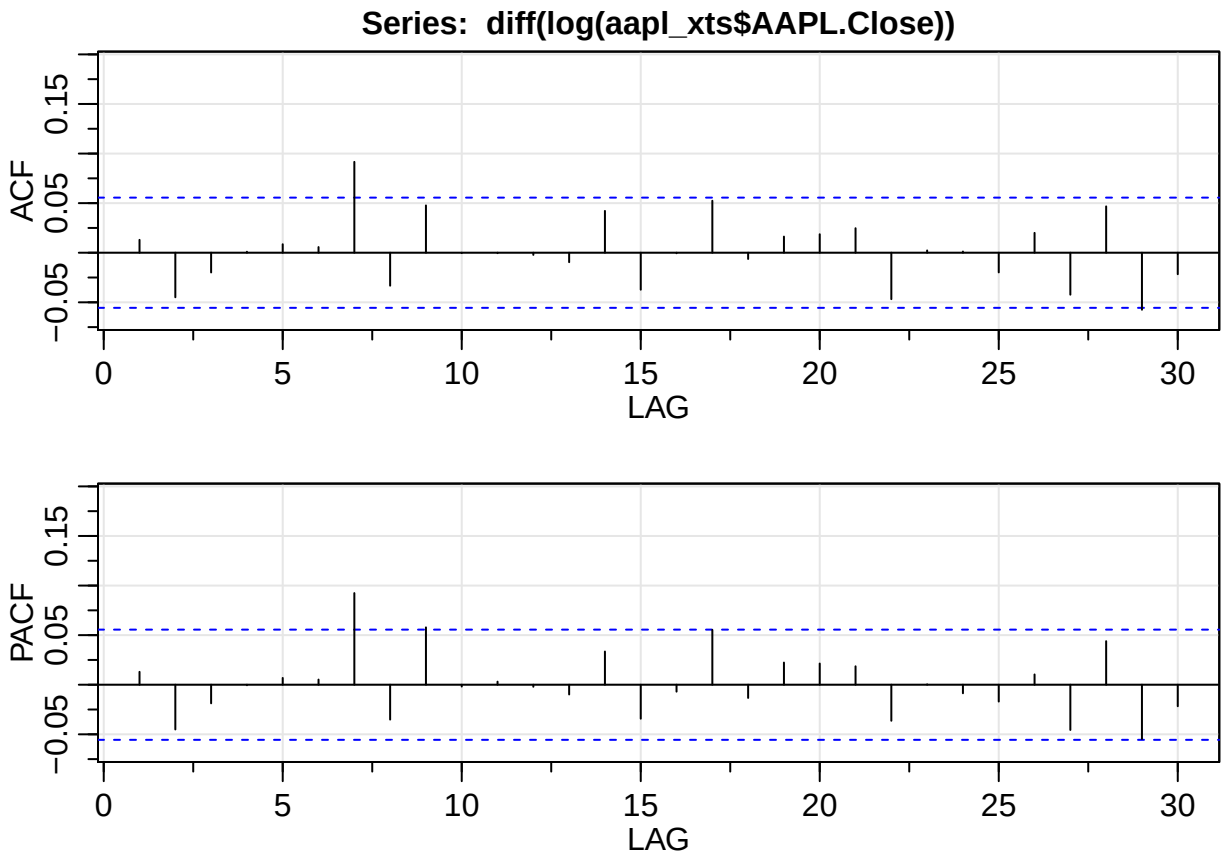
Regression

Feature engineering

```
# Create 10 days lagging
lags=Lag(dailyReturn(aapl_xts),k=1:10)

# Convert the xts object to data frame object and trim off adjusted and Volume
aapl_df <- na.omit(cbind.data.frame(aapl_xts, lags))[, -c(5, 6)]

# Create intraction terms
aapl_df$Open.High <- aapl_df$AAPL.Open * aapl_df$AAPL.High
aapl_df$Open.Low <- aapl_df$AAPL.Open * aapl_df$AAPL.Low
aapl_df$High.Low <- aapl_df$AAPL.Low * aapl_df$AAPL.High
aapl_df$Open.High.Low <- aapl_df$AAPL.Open * aapl_df$AAPL.High * aapl_df$AAPL.Low
```

We are building a model to predict AAPL's Close price, and are not able to obtain the volume and the adjusted close price before the stock is closed. Therefore, we have to excludes the AAPL.Volume and AAPL.Adj.Close from the model.

```
library(astsa)
# Splot the autocorrelation
acf2(diff(log(aapl_xts$AAPL.Close)), max.lag = 30)
```




```
##          ACF  PACF
##  [1,]  0.01  0.01
##  [2,] -0.04 -0.05
##  [3,] -0.02 -0.02
##  [4,]  0.00  0.00
##  [5,]  0.01  0.01
##  [6,]  0.01  0.01
##  [7,]  0.09  0.09
##  [8,] -0.03 -0.04
##  [9,]  0.05  0.06
## [10,]  0.00  0.00
## [11,]  0.00  0.00
## [12,]  0.00  0.00
## [13,] -0.01 -0.01
## [14,]  0.04  0.03
## [15,] -0.04 -0.03
## [16,]  0.00 -0.01
## [17,]  0.05  0.06
## [18,] -0.01 -0.01
## [19,]  0.02  0.02
## [20,]  0.02  0.02
## [21,]  0.02  0.02
## [22,] -0.05 -0.04
## [23,]  0.00  0.00
## [24,]  0.00 -0.01
## [25,] -0.02 -0.02
```

```
## [26,]  0.02  0.01
## [27,] -0.04 -0.05
## [28,]  0.05  0.04
## [29,] -0.06 -0.05
## [30,] -0.02 -0.02
```

It is a random walk model and predictors Open, High, Low and Close are highly correlated.

Feature Selection

Regression Test

```
library(caret)
set.seed(1)

# Create a 80/20 stratified train-test split
train <- createDataPartition(aapl_df$AAPL.Close, p = 0.8, list = F, times = 1)
test <- (-train)

# Fit a full linear regression model on the training data
model_lm_full <- lm(AAPL.Close ~ ., data = aapl_df, subset = train)
summary(model_lm_full)
```

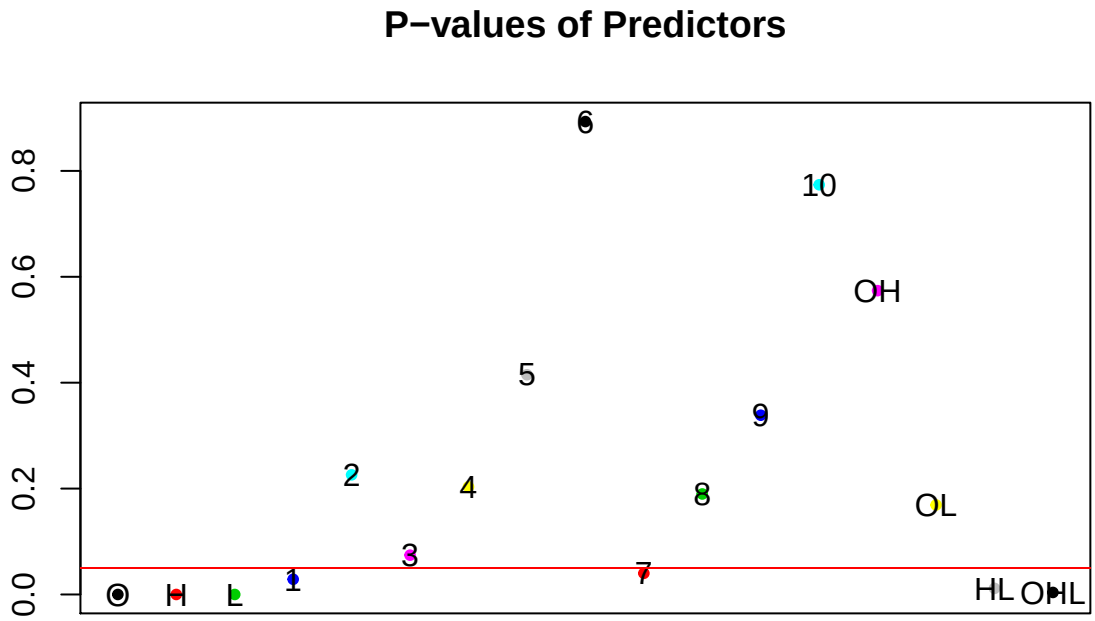
```
##
## Call:
## lm(formula = AAPL.Close ~ ., data = aapl_df, subset = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4789 -0.4132  0.0209  0.4676  6.9310
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -3.895e+00  1.522e+00  -2.559  0.01066 *
## AAPL.Open     -7.471e-01  1.113e-01  -6.713 3.17e-11 ***
## AAPL.High      9.873e-01  1.064e-01   9.280  < 2e-16 ***
## AAPL.Low       8.373e-01  8.807e-02   9.508  < 2e-16 ***
## Lag.1         -3.895e+00  1.778e+00  -2.191  0.02871 *
## Lag.2         -2.108e+00  1.740e+00  -1.212  0.22594
## Lag.3         -3.046e+00  1.704e+00  -1.787  0.07420 .
## Lag.4         -2.167e+00  1.702e+00  -1.273  0.20333
## Lag.5         -1.416e+00  1.736e+00  -0.815  0.41499
## Lag.6          2.303e-01  1.714e+00   0.134  0.89314
## Lag.7          3.517e+00  1.710e+00   2.057  0.03997 *
## Lag.8         -2.273e+00  1.732e+00  -1.312  0.18981
## Lag.9         -1.666e+00  1.740e+00  -0.958  0.33852
## Lag.10        -5.088e-01  1.770e+00  -0.287  0.77381
## Open.High      2.673e-04  4.750e-04   0.563  0.57378
## Open.Low       8.618e-04  6.262e-04   1.376  0.16907
## High.Low      -1.625e-03  6.420e-04  -2.531  0.01153 *
## Open.High.Low  1.017e-06  3.508e-07   2.900  0.00381 **
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8494 on 1011 degrees of freedom
## Multiple R-squared:  0.9996, Adjusted R-squared:  0.9996
## F-statistic: 1.353e+05 on 17 and 1011 DF,  p-value: < 2.2e-16
```

```
# Plot the p-values
x <- seq(1, 17, 1)
p <- c('O', 'H', 'L', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'OH', 'OL', 'HL', 'OHL')
coef <- summary(model_lm_full)$coefficients[2:18, 4]
df <- data.frame(x, p, coef)
plot(x, coef, pch = 20, col = 1:17, xaxt = "n", main = "P-values of Predictors", xlab = "", ylab = "")
text(x, coef, labels = p, data = df)
```

```
## Warning in text.default(x, coef, labels = p, data = df): "data" is not a
## graphical parameter
```

```
abline(h = 0.05, col = "red")
```



The F-test shows that there is at least one predictor is significant.

```
set.seed(1)
# Fit a single linear regression model to determine which variable is most sinificant to the reponse AA
predictors <- names(aapl_df)[-4]
```

```r
pvalues <- 0
for (i in 1:length(predictors)) {
    fmla <- as.formula(paste("AAPL.Close ~", predictors[i]))
    model <- lm(fmla, aapl_df, subset = train)
    pvalues[i] <- summary(model)$coefficients[, 4][2]
}
# Extract significant predictors whose p-value is less than or equal to 0.05
predictors_significant <- predictors[pvalues <= 0.05]

print("Predictors:")
```

```
## [1] "Predictors:"
```

```r
print(predictors)
```

```
##  [1] "AAPL.Open"     "AAPL.High"     "AAPL.Low"      "Lag.1"
##  [5] "Lag.2"         "Lag.3"         "Lag.4"         "Lag.5"
##  [9] "Lag.6"         "Lag.7"         "Lag.8"         "Lag.9"
## [13] "Lag.10"        "Open.High"     "Open.Low"      "High.Low"
## [17] "Open.High.Low"
```

```r
print("P-values of predicots:")
```

```
## [1] "P-values of predicots:"
```

```r
print(pvalues)
```

```
##  [1] 0.00000000 0.00000000 0.00000000 0.27175331 0.09367353 0.18954705
##  [7] 0.01679424 0.75360438 0.18862619 0.07246602 0.06268187 0.10739254
## [13] 0.05850310 0.00000000 0.00000000 0.00000000 0.00000000
```

```r
print("Sifinicant predictors:")
```

```
## [1] "Sifinicant predictors:"
```

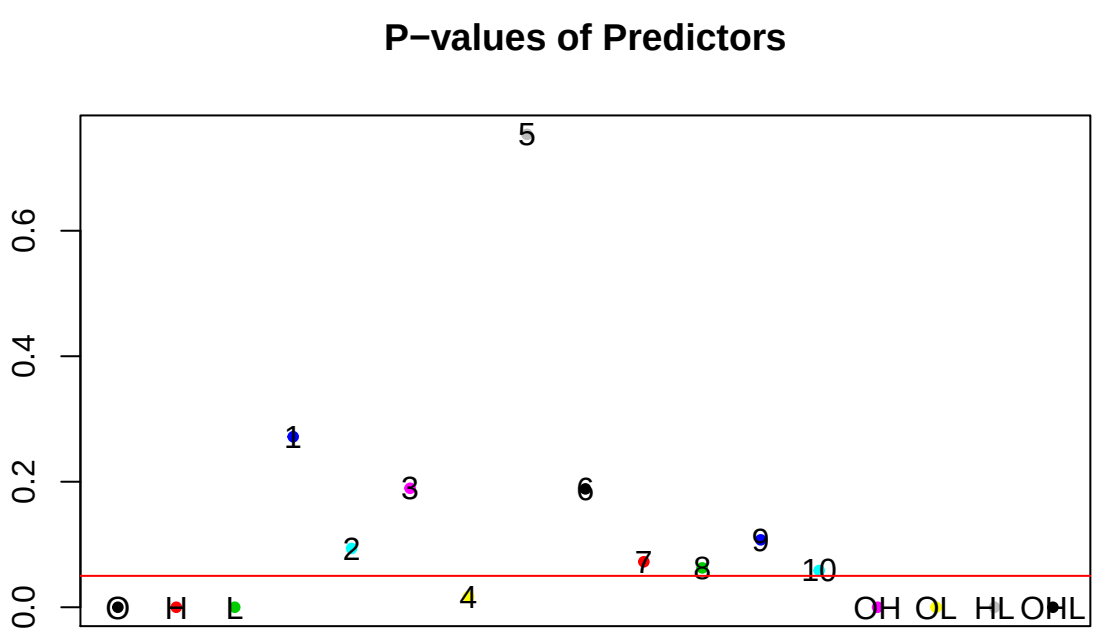
```r
print(predictors_significant)
```

```
## [1] "AAPL.Open"     "AAPL.High"     "AAPL.Low"      "Lag.4"
## [5] "Open.High"     "Open.Low"      "High.Low"      "Open.High.Low"
```

```r
print(paste("Predictor with minimum p-value:", predictors[which.min(pvalues)]))
```

```
## [1] "Predictor with minimum p-value: AAPL.Open"
```

```r
# Plot the p-values
x <- seq(1, 17, 1)
p <- c('O', 'H', 'L', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'OH', 'OL', 'HL', 'OHL')
df <- data.frame(x, p, pvalues)
plot(x, pvalues, pch = 20, col = 1:17, xaxt = "n", main = "P-values of Predictors", xlab = "", ylab = "
text(x, pvalues, labels = p, data = df)
abline(h = 0.05, col = "red")
```



From the p-values, it shows that "AAPL.Open", "AAPL.High", "AAPL.Low", "Lag.4", "Open.High", "Open.Low", "High.Low", "Open.High.Low" are relevant to AAPL.Close.

Best Subset Selection

```
# Perform best subset selection
library(leaps)
set.seed(1)
regfit <- regsubsets(AAPL.Close ~ ., aapl_df[train, ], nvmax = 17)

regfit_summary <- summary(regfit)
regfit_summary
```

```
## Subset selection object
## Call: regsubsets.formula(AAPL.Close ~ ., aapl_df[train, ], nvmax = 17)
## 17 Variables  (and intercept)
##               Forced in Forced out
## AAPL.Open         FALSE      FALSE
## AAPL.High         FALSE      FALSE
## AAPL.Low          FALSE      FALSE
## Lag.1             FALSE      FALSE
## Lag.2             FALSE      FALSE
## Lag.3             FALSE      FALSE
## Lag.4             FALSE      FALSE
```

```
## Lag.5             FALSE      FALSE
## Lag.6             FALSE      FALSE
## Lag.7             FALSE      FALSE
## Lag.8             FALSE      FALSE
## Lag.9             FALSE      FALSE
## Lag.10            FALSE      FALSE
## Open.High         FALSE      FALSE
## Open.Low          FALSE      FALSE
## High.Low          FALSE      FALSE
## Open.High.Low     FALSE      FALSE
## 1 subsets of each size up to 17
## Selection Algorithm: exhaustive
##           AAPL.Open AAPL.High AAPL.Low Lag.1 Lag.2 Lag.3 Lag.4 Lag.5 Lag.6
## 1  ( 1 )  " "       " "       "*"      " "   " "   " "   " "   " "   " "
## 2  ( 1 )  " "       "*"       "*"      " "   " "   " "   " "   " "   " "
## 3  ( 1 )  "*"       "*"       "*"      " "   " "   " "   " "   " "   " "
## 4  ( 1 )  "*"       "*"       "*"      " "   " "   " "   " "   " "   " "
## 5  ( 1 )  "*"       "*"       "*"      "*"   " "   " "   " "   " "   " "
## 6  ( 1 )  "*"       "*"       "*"      " "   " "   " "   " "   " "   " "
## 7  ( 1 )  "*"       "*"       "*"      "*"   " "   " "   " "   " "   " "
## 8  ( 1 )  "*"       "*"       "*"      "*"   " "   "*"   " "   " "   " "
## 9  ( 1 )  "*"       "*"       "*"      "*"   " "   "*"   " "   " "   " "
## 10  ( 1 ) "*"       "*"       "*"      "*"   " "   "*"   "*"   " "   " "
## 11  ( 1 ) "*"       "*"       "*"      "*"   " "   "*"   "*"   " "   " "
## 12  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   " "   " "
## 13  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   " "   " "
## 14  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   " "
## 15  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   " "
## 16  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   " "
## 17  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   "*"
##           Lag.7 Lag.8 Lag.9 Lag.10 Open.High Open.Low High.Low
## 1  ( 1 )  " "   " "   " "   " "    " "       " "      " "
## 2  ( 1 )  " "   " "   " "   " "    " "       " "      " "
## 3  ( 1 )  " "   " "   " "   " "    " "       " "      " "
## 4  ( 1 )  "*"   " "   " "   " "    " "       " "      " "
## 5  ( 1 )  "*"   " "   " "   " "    " "       " "      " "
## 6  ( 1 )  "*"   " "   " "   " "    " "       " "      "*"
## 7  ( 1 )  "*"   " "   " "   " "    " "       " "      "*"
## 8  ( 1 )  "*"   " "   " "   " "    " "       " "      "*"
## 9  ( 1 )  "*"   " "   " "   " "    " "       "*"      "*"
## 10  ( 1 ) "*"   " "   " "   " "    " "       "*"      "*"
## 11  ( 1 ) "*"   "*"   " "   " "    " "       "*"      "*"
## 12  ( 1 ) "*"   "*"   " "   " "    " "       "*"      "*"
## 13  ( 1 ) "*"   "*"   "*"   " "    " "       "*"      "*"
## 14  ( 1 ) "*"   "*"   "*"   " "    " "       "*"      "*"
## 15  ( 1 ) "*"   "*"   "*"   " "    "*"       "*"      "*"
## 16  ( 1 ) "*"   "*"   "*"   "*"    "*"       "*"      "*"
## 17  ( 1 ) "*"   "*"   "*"   "*"    "*"       "*"      "*"
##           Open.High.Low
## 1  ( 1 )  " "
## 2  ( 1 )  " "
## 3  ( 1 )  " "
## 4  ( 1 )  " "
## 5  ( 1 )  " "
```

```
## 6  ( 1 )  "*"
## 7  ( 1 )  "*"
## 8  ( 1 )  "*"
## 9  ( 1 )  "*"
## 10  ( 1 ) "*"
## 11  ( 1 ) "*"
## 12  ( 1 ) "*"
## 13  ( 1 ) "*"
## 14  ( 1 ) "*"
## 15  ( 1 ) "*"
## 16  ( 1 ) "*"
## 17  ( 1 ) "*"
```

```
# Examine all statistics to select the best overall model
print("RSS:")
```

```
## [1] "RSS:"
```

```
print(regfit_summary$rss)
```

```
##  [1] 1590.4383 1057.7815  752.9417  748.7428  745.4524  741.8189  738.3748
##  [8]  736.1612  733.9508  732.9062  731.8609  730.7935  730.1805  729.7295
## [15]  729.4880  729.4287  729.4157
```

```
print("Adjusted Rsq:")
```

```
## [1] "Adjusted Rsq:"
```

```
print(regfit_summary$adjr2)
```

```
##  [1] 0.9990411 0.9993617 0.9995452 0.9995473 0.9995488 0.9995506 0.9995522
##  [8] 0.9995531 0.9995540 0.9995542 0.9995544 0.9995546 0.9995546 0.9995544
## [15] 0.9995541 0.9995537 0.9995533
```

```
print("bic:")
```

```
## [1] "bic:"
```

```
print(regfit_summary$bic)
```

```
##  [1] -7138.436 -7551.163 -7894.026 -7892.845 -7890.440 -7888.532 -7886.384
##  [8] -7882.537 -7878.695 -7873.224 -7867.757 -7862.322 -7856.249 -7849.949
## [15] -7843.353 -7836.500 -7829.582
```

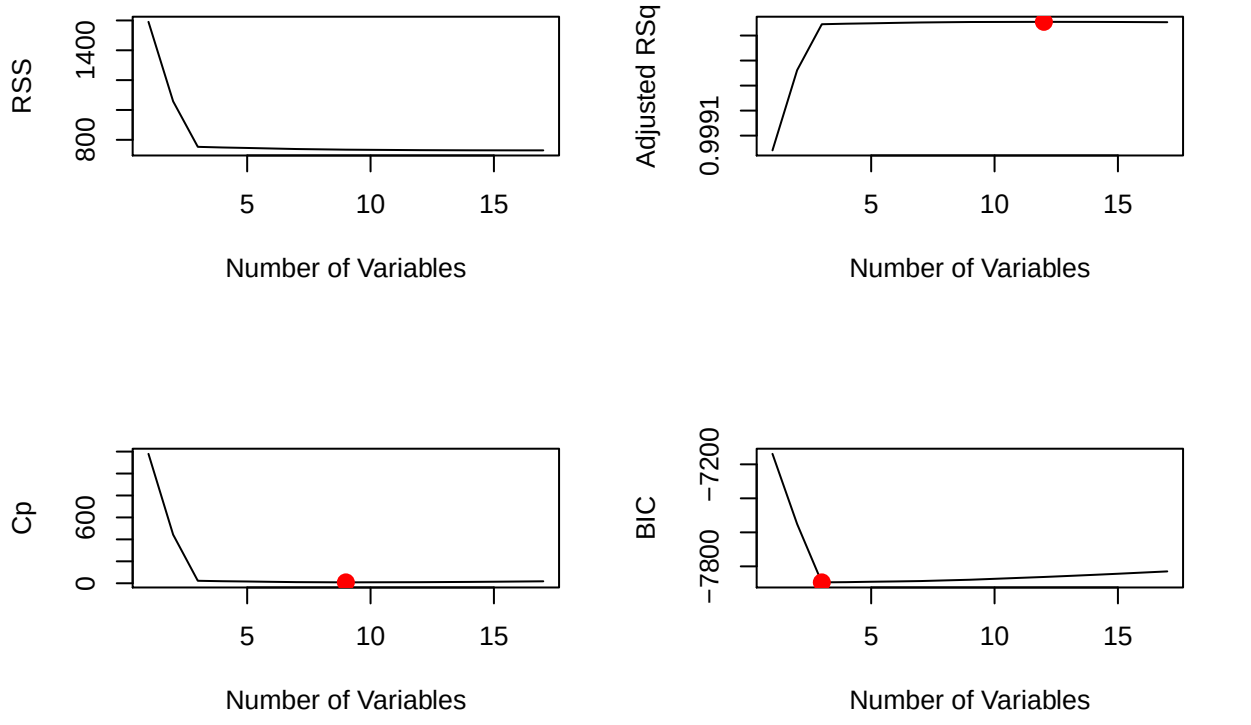
```
print("cp:")
```

```
## [1] "cp:"
```

```
print(regfit_summary$cp)
```

```
##  [1] 1179.412544  443.128392   22.607956   18.788088   16.227520
##  [6]   13.191329   10.417714    9.349473    8.285824    8.838020
## [11]    9.389098    9.909704   11.060017   12.434900   14.100194
## [16]   16.018054   18.000000
```

```
par(mfrow=c(2, 2))
plot(regfit_summary$rss, xlab = "Number of Variables", ylab = "RSS", type = "l")

plot(regfit_summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted RSq", type = "l")
max_adjr2 <- which.max(regfit_summary$adjr2)
points(max_adjr2, regfit_summary$adjr2[max_adjr2], col="red", cex = 2, pch = 20)

plot(regfit_summary$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")
min_cp <- which.min(regfit_summary$cp)
points(min_cp, regfit_summary$cp[min_cp], col = "red", cex = 2, pch = 20)

min_bic <- which.min(regfit_summary$bic)
plot(regfit_summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")
points(min_bic, regfit_summary$bic[min_bic], col = "red", cex = 2, pch = 20)
```



Based on the above statistics, AAPL.Open, AAPL.High, AAPL.Low should be included in the model even-though Lag.1 and Lag.7 help a little.

Stepwise Selection

```r
# Perform stepwise selection
regfit_fwd <- regsubsets(AAPL.Close ~ ., data = aapl_df[train, ], method = "forward", nvmax = 17)
summary(regfit_fwd)
```

```
## Subset selection object
## Call: regsubsets.formula(AAPL.Close ~ ., data = aapl_df[train, ], method = "forward",
##     nvmax = 17)
## 17 Variables  (and intercept)
##               Forced in Forced out
## AAPL.Open         FALSE      FALSE
## AAPL.High         FALSE      FALSE
## AAPL.Low          FALSE      FALSE
## Lag.1             FALSE      FALSE
## Lag.2             FALSE      FALSE
## Lag.3             FALSE      FALSE
## Lag.4             FALSE      FALSE
## Lag.5             FALSE      FALSE
## Lag.6             FALSE      FALSE
## Lag.7             FALSE      FALSE
## Lag.8             FALSE      FALSE
## Lag.9             FALSE      FALSE
## Lag.10            FALSE      FALSE
## Open.High         FALSE      FALSE
## Open.Low          FALSE      FALSE
## High.Low          FALSE      FALSE
## Open.High.Low     FALSE      FALSE
## 1 subsets of each size up to 17
## Selection Algorithm: forward
##           AAPL.Open AAPL.High AAPL.Low Lag.1 Lag.2 Lag.3 Lag.4 Lag.5 Lag.6
## 1  ( 1 )  " "       " "       "*"      " "   " "   " "   " "   " "   " "
## 2  ( 1 )  " "       "*"       "*"      " "   " "   " "   " "   " "   " "
## 3  ( 1 )  "*"       "*"       "*"      " "   " "   " "   " "   " "   " "
## 4  ( 1 )  "*"       "*"       "*"      " "   " "   " "   " "   " "   " "
## 5  ( 1 )  "*"       "*"       "*"      "*"   " "   " "   " "   " "   " "
## 6  ( 1 )  "*"       "*"       "*"      "*"   " "   "*"   " "   " "   " "
## 7  ( 1 )  "*"       "*"       "*"      "*"   " "   "*"   "*"   " "   " "
## 8  ( 1 )  "*"       "*"       "*"      "*"   " "   "*"   "*"   " "   " "
## 9  ( 1 )  "*"       "*"       "*"      "*"   " "   "*"   "*"   " "   " "
## 10  ( 1 ) "*"       "*"       "*"      "*"   " "   "*"   "*"   " "   " "
## 11  ( 1 ) "*"       "*"       "*"      "*"   " "   "*"   "*"   " "   " "
## 12  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   " "   " "
## 13  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   " "   " "
## 14  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   " "
## 15  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   " "
## 16  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   " "
## 17  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   "*"
##           Lag.7 Lag.8 Lag.9 Lag.10 Open.High Open.Low High.Low
## 1  ( 1 )  " "   " "   " "   " "    " "       " "      " "
## 2  ( 1 )  " "   " "   " "   " "    " "       " "      " "
## 3  ( 1 )  " "   " "   " "   " "    " "       " "      " "
## 4  ( 1 )  "*"   " "   " "   " "    " "       " "      " "
```

```
## 5  ( 1 )  "*"   " "   " "   " "    " "       " "      " "
## 6  ( 1 )  "*"   " "   " "   " "    " "       " "      " "
## 7  ( 1 )  "*"   " "   " "   " "    " "       " "      " "
## 8  ( 1 )  "*"   " "   " "   " "    " "       " "      " "
## 9  ( 1 )  "*"   " "   " "   " "    " "       " "      "*"
## 10  ( 1 ) "*"   " "   " "   " "    " "       "*"      "*"
## 11  ( 1 ) "*"   "*"   " "   " "    " "       "*"      "*"
## 12  ( 1 ) "*"   "*"   " "   " "    " "       "*"      "*"
## 13  ( 1 ) "*"   "*"   "*"   " "    " "       "*"      "*"
## 14  ( 1 ) "*"   "*"   "*"   " "    " "       "*"      "*"
## 15  ( 1 ) "*"   "*"   "*"   " "    "*"       "*"      "*"
## 16  ( 1 ) "*"   "*"   "*"   "*"    "*"       "*"      "*"
## 17  ( 1 ) "*"   "*"   "*"   "*"    "*"       "*"      "*"
##           Open.High.Low
## 1  ( 1 )  " "
## 2  ( 1 )  " "
## 3  ( 1 )  " "
## 4  ( 1 )  " "
## 5  ( 1 )  " "
## 6  ( 1 )  " "
## 7  ( 1 )  " "
## 8  ( 1 )  "*"
## 9  ( 1 )  "*"
## 10  ( 1 ) "*"
## 11  ( 1 ) "*"
## 12  ( 1 ) "*"
## 13  ( 1 ) "*"
## 14  ( 1 ) "*"
## 15  ( 1 ) "*"
## 16  ( 1 ) "*"
## 17  ( 1 ) "*"
```

```
print("RSS:")
```

```
## [1] "RSS:"
```

```
print(regfit_fwd$rss)
```

```
##  [1] 1660289.7595    1590.4383    1057.7815     752.9417     748.7428
##  [6]     745.4524     743.5441     742.5515     741.7110     735.0272
## [11]     732.9062     731.8609     730.7935     730.1805     729.7295
## [16]     729.4880     729.4287     729.4157
```

```
regfit_bwd <- regsubsets(AAPL.Close ~ ., data = aapl_df[train, ], method = "backward", nvmax = 17)
summary(regfit_bwd)
```

```
## Subset selection object
## Call: regsubsets.formula(AAPL.Close ~ ., data = aapl_df[train, ], method = "backward",
##     nvmax = 17)
## 17 Variables  (and intercept)
##               Forced in Forced out
## AAPL.Open         FALSE      FALSE
```

```
## AAPL.High         FALSE      FALSE
## AAPL.Low          FALSE      FALSE
## Lag.1             FALSE      FALSE
## Lag.2             FALSE      FALSE
## Lag.3             FALSE      FALSE
## Lag.4             FALSE      FALSE
## Lag.5             FALSE      FALSE
## Lag.6             FALSE      FALSE
## Lag.7             FALSE      FALSE
## Lag.8             FALSE      FALSE
## Lag.9             FALSE      FALSE
## Lag.10            FALSE      FALSE
## Open.High         FALSE      FALSE
## Open.Low          FALSE      FALSE
## High.Low          FALSE      FALSE
## Open.High.Low     FALSE      FALSE
## 1 subsets of each size up to 17
## Selection Algorithm: backward
##           AAPL.Open AAPL.High AAPL.Low Lag.1 Lag.2 Lag.3 Lag.4 Lag.5 Lag.6
## 1  ( 1 )  " "       " "       "*"      " "   " "   " "   " "   " "   " "
## 2  ( 1 )  " "       "*"       "*"      " "   " "   " "   " "   " "   " "
## 3  ( 1 )  "*"       "*"       "*"      " "   " "   " "   " "   " "   " "
## 4  ( 1 )  "*"       "*"       "*"      " "   " "   " "   " "   " "   " "
## 5  ( 1 )  "*"       "*"       "*"      " "   " "   " "   " "   " "   " "
## 6  ( 1 )  "*"       "*"       "*"      " "   " "   " "   " "   " "   " "
## 7  ( 1 )  "*"       "*"       "*"      "*"   " "   " "   " "   " "   " "
## 8  ( 1 )  "*"       "*"       "*"      "*"   " "   "*"   " "   " "   " "
## 9  ( 1 )  "*"       "*"       "*"      "*"   " "   "*"   " "   " "   " "
## 10  ( 1 ) "*"       "*"       "*"      "*"   " "   "*"   "*"   " "   " "
## 11  ( 1 ) "*"       "*"       "*"      "*"   " "   "*"   "*"   " "   " "
## 12  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   " "   " "
## 13  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   " "   " "
## 14  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   " "
## 15  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   " "
## 16  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   " "
## 17  ( 1 ) "*"       "*"       "*"      "*"   "*"   "*"   "*"   "*"   "*"
##           Lag.7 Lag.8 Lag.9 Lag.10 Open.High Open.Low High.Low
## 1  ( 1 )  " "   " "   " "   " "    " "       " "      " "
## 2  ( 1 )  " "   " "   " "   " "    " "       " "      " "
## 3  ( 1 )  " "   " "   " "   " "    " "       " "      " "
## 4  ( 1 )  " "   " "   " "   " "    " "       " "      " "
## 5  ( 1 )  " "   " "   " "   " "    " "       " "      "*"
## 6  ( 1 )  "*"   " "   " "   " "    " "       " "      "*"
## 7  ( 1 )  "*"   " "   " "   " "    " "       " "      "*"
## 8  ( 1 )  "*"   " "   " "   " "    " "       " "      "*"
## 9  ( 1 )  "*"   " "   " "   " "    " "       "*"      "*"
## 10  ( 1 ) "*"   " "   " "   " "    " "       "*"      "*"
## 11  ( 1 ) "*"   "*"   " "   " "    " "       "*"      "*"
## 12  ( 1 ) "*"   "*"   " "   " "    " "       "*"      "*"
## 13  ( 1 ) "*"   "*"   "*"   " "    " "       "*"      "*"
## 14  ( 1 ) "*"   "*"   "*"   " "    " "       "*"      "*"
## 15  ( 1 ) "*"   "*"   "*"   " "    "*"       "*"      "*"
## 16  ( 1 ) "*"   "*"   "*"   "*"    "*"       "*"      "*"
## 17  ( 1 ) "*"   "*"   "*"   "*"    "*"       "*"      "*"
```

```
##           Open.High.Low
## 1  ( 1 )  " "
## 2  ( 1 )  " "
## 3  ( 1 )  " "
## 4  ( 1 )  "*"
## 5  ( 1 )  "*"
## 6  ( 1 )  "*"
## 7  ( 1 )  "*"
## 8  ( 1 )  "*"
## 9  ( 1 )  "*"
## 10  ( 1 ) "*"
## 11  ( 1 ) "*"
## 12  ( 1 ) "*"
## 13  ( 1 ) "*"
## 14  ( 1 ) "*"
## 15  ( 1 ) "*"
## 16  ( 1 ) "*"
## 17  ( 1 ) "*"
```

```
print("RSS:")
```

```
## [1] "RSS:"
```

```
print(regfit_bwd$rss)
```

```
##  [1] 1660289.7595    1590.4383    1057.7815     752.9417     752.1129
##  [6]     745.6518     741.8189     738.3748     736.1612     733.9508
## [11]     732.9062     731.8609     730.7935     730.1805     729.7295
## [16]     729.4880     729.4287     729.4157
```

Forward selection outputs the same result as best subset selection. However, backward selection picks up High.Low and Open.High.Low as the 4th and the 5th important features.

Validation Set Approach

```
set.seed(1)
# Credit to James, etc.
# Choose the best features using valiation set approach
# Apply the regsubsets() to the training set in order to perform best subset selection
regfit_best <- regsubsets(AAPL.Close ~ ., aapl_df[train, ], nvmax = 17)

# Make a model matrix fro the test data
library(glmnet)
test_mat <- model.matrix(AAPL.Close ~ ., data = aapl_df[test, ])

# Extract the coefficients of the features, we choose 13 because AAPL.Close is excluded
val_errors <- rep(NA, 17)
for (i in 1:17) {
    # Extract the coefficients of model size i
    coefi <- coef(regfit_best, id = i)
```

```
    # Multiply them into the appropriate columns of the test model matrix to form the predictions
    p <- test_mat[, names(coefi)]%*%coefi
    # Compute the test MSE
    val_errors[i] = mean((aapl_df$AAPL.Close[test] - p)^2)
}
print("Validation errors: ")
```

```
## [1] "Validation errors: "
```

```
print(val_errors)
```

```
##  [1] 1.6779853 1.0936389 0.6951724 0.6934632 0.7082215 0.6920734 0.7064702
##  [8] 0.7212941 0.7389914 0.7375827 0.7396356 0.7372538 0.7357860 0.7330816
## [15] 0.7283755 0.7288671 0.7282559
```

```
print(paste("Best model size:", which.min(val_errors)))
```

```
## [1] "Best model size: 6"
```

```
print("Best model size coefficients:")
```

```
## [1] "Best model size coefficients:"
```

```
print(coef(regfit_best, which.min(val_errors)))
```

```
##   (Intercept)     AAPL.Open     AAPL.High      AAPL.Low         Lag.7
## -3.980494e+00 -5.619514e-01  8.672434e-01  7.745102e-01  3.913564e+00
##      High.Low Open.High.Low
## -5.120539e-04  1.035058e-06
```

```
plot(val_errors, main = "Validation Errors", xlab = "Size of Best Subset", ylab = "Quantities")
```

The validation approach have similiar results as the backward selection.

Cross Validation Approach

```
set.seed(1)
# Credit to James, etc.
# Choose the best features using cross validation approach
predict_regsubsets <- function(object, newdata, id, ...) {
    form <- as.formula(object$call[[2]])
    mat <- model.matrix(form, newdata)
    coefi <- coef(object, id = id)
    mat[, names(coefi)] %*% coefi
}

set.seed(1)
k <- 10
folds <- sample(1:k, nrow(aapl_df), replace = T)
cv_errors <- matrix(NA, k, 17, dimnames = list(NULL, paste(1:17)))

for (j in 1:k) {
    best_fit <- regsubsets(AAPL.Close ~ ., data = aapl_df[folds!=j, ], nvmax = 17)
    for (i in 1:17) {
        p <- predict_regsubsets(best_fit, newdata = aapl_df[folds==j, ], id = i)
        cv_errors[j, i] <- mean((aapl_df$AAPL.Close[folds==j] - p)^2)
    }
```

```
}

mean_cv_errors <- apply(cv_errors, 2, mean)
print("Mean CV errors:")
```

```
## [1] "Mean CV errors:"
```

```
print(mean_cv_errors)
```

```
##         1         2         3         4         5         6         7
## 1.5811688 1.0557818 0.7265530 0.7304999 0.7418827 0.7399883 0.7476844
##         8         9        10        11        12        13        14
## 0.7496637 0.7535663 0.7540093 0.7600540 0.7580468 0.7564969 0.7575780
##        15        16        17
## 0.7566535 0.7567827 0.7568599
```

```
print(paste("Best model size:", which.min(mean_cv_errors)))
```

```
## [1] "Best model size: 3"
```

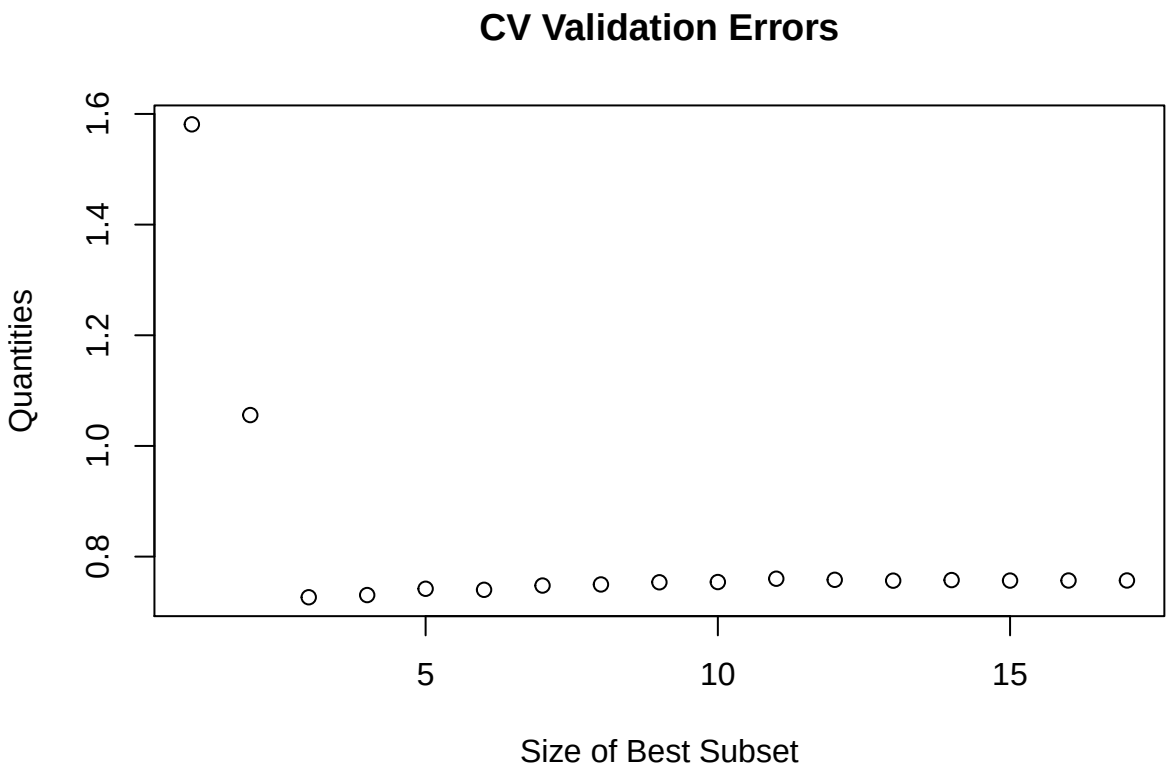
```
print("Best model size coefficients:")
```

```
## [1] "Best model size coefficients:"
```

```
print(coef(regfit_best, which.min(mean_cv_errors)))
```

```
## (Intercept)   AAPL.Open   AAPL.High    AAPL.Low
##  0.07442447 -0.52853624  0.79870357  0.72903220
```

```
plot(mean_cv_errors, main = "CV Validation Errors", xlab = "Size of Best Subset", ylab = "Quantities")
```



The cross validation approach confirms that AAPL.Open, AAPL.High and AAPL.Low should be included.

Model Selection

Multivariate Linear Regression

```
# Build a mutiple linear regression model
fmla_best <- as.formula("AAPL.Close ~ AAPL.Open + AAPL.High + AAPL.Low")
model_lm_best <- lm(fmla_best, data = aapl_df, subset = train)
summary(model_lm_best)
```

```
##
## Call:
## lm(formula = fmla_best, data = aapl_df, subset = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6226 -0.4111  0.0266  0.4568  7.3777
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.07442    0.10324   0.721    0.471
## AAPL.Open   -0.52854    0.02595 -20.371   <2e-16 ***
## AAPL.High    0.79870    0.02378  33.588   <2e-16 ***
```

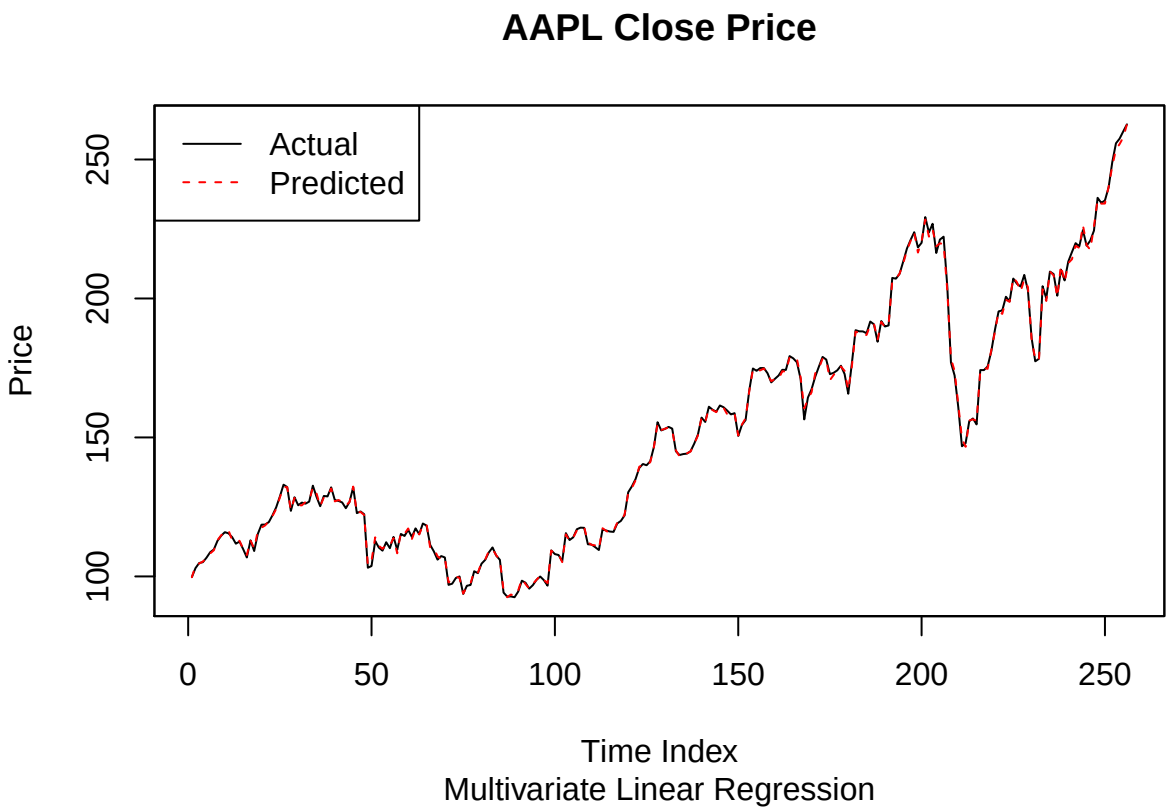
```
## AAPL.Low     0.72903    0.02017  36.141   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8571 on 1025 degrees of freedom
## Multiple R-squared:  0.9995, Adjusted R-squared:  0.9995
## F-statistic: 7.531e+05 on 3 and 1025 DF,  p-value: < 2.2e-16
```

```
calculate_test_RMSE <- function(model, model_name) {
    # Predict the close price
    p <- predict(model, newdata = aapl_df[test, ])

    # Compute errors
    error <- p - aapl_df[test, ][["AAPL.Close"]]

    # Calculate RMSE
    RMSE <- sqrt(mean(error^2))
    cat(model_name, "test RMSE:", RMSE, "\n")
    return (RMSE)
}

calculate_test_RMSE(model_lm_best, "Linear model")
```

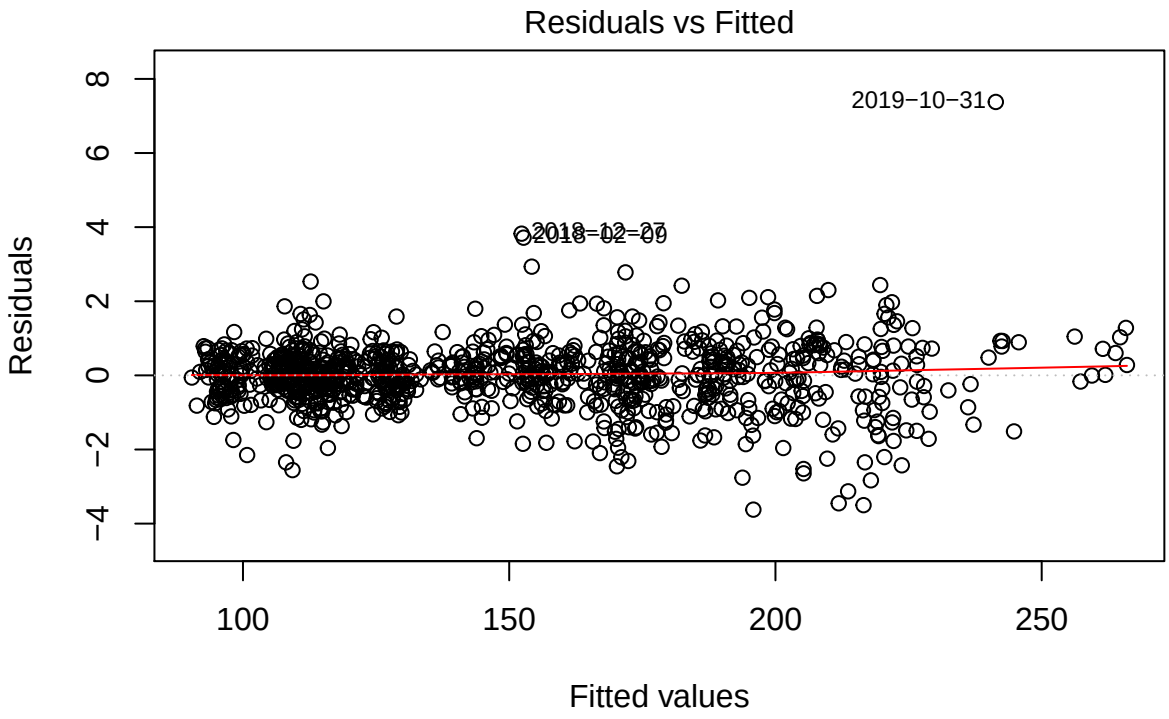
```
## Linear model test RMSE: 0.83377
```

```
## [1] 0.83377
```

```
plot_model <- function(model, model_name) {
    plot(aapl_df[test, 4], type = "l", col = "black", main = "AAPL Close Price", sub = model_name, lty =
    par(new = T)
    plot(predict(model, aapl_df[test, ]), type = "l", col = 'red', lty = 2, xlab = "", ylab = "", xaxt =
    legend("topleft", c("Actual", "Predicted"), col = c("black", "red"), lty = 1:2)
}

plot_model(model_lm_best, "Multivariate Linear Regression")
```

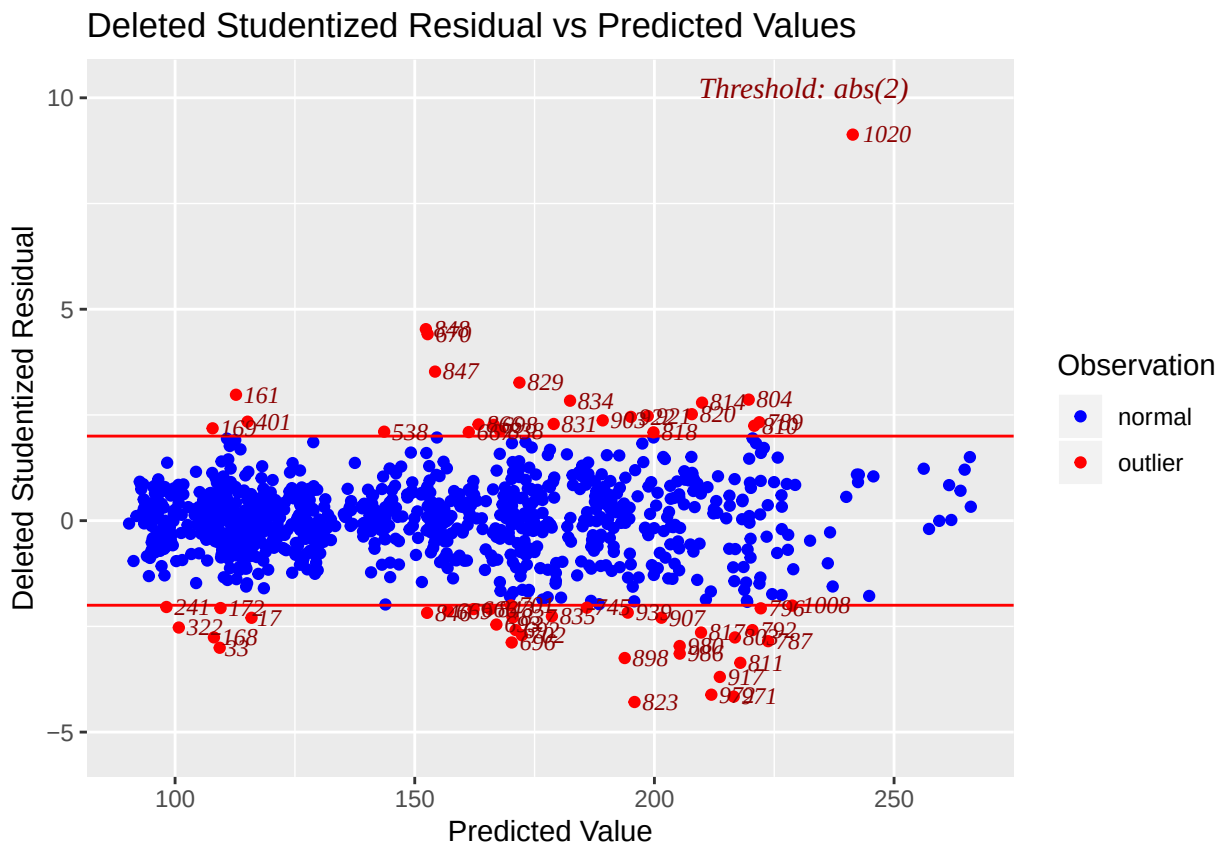



```
# Check if there is non-linearity by plotting fitted values vs. residuals
plot(model_lm_best, which = 1, sub = "")
```

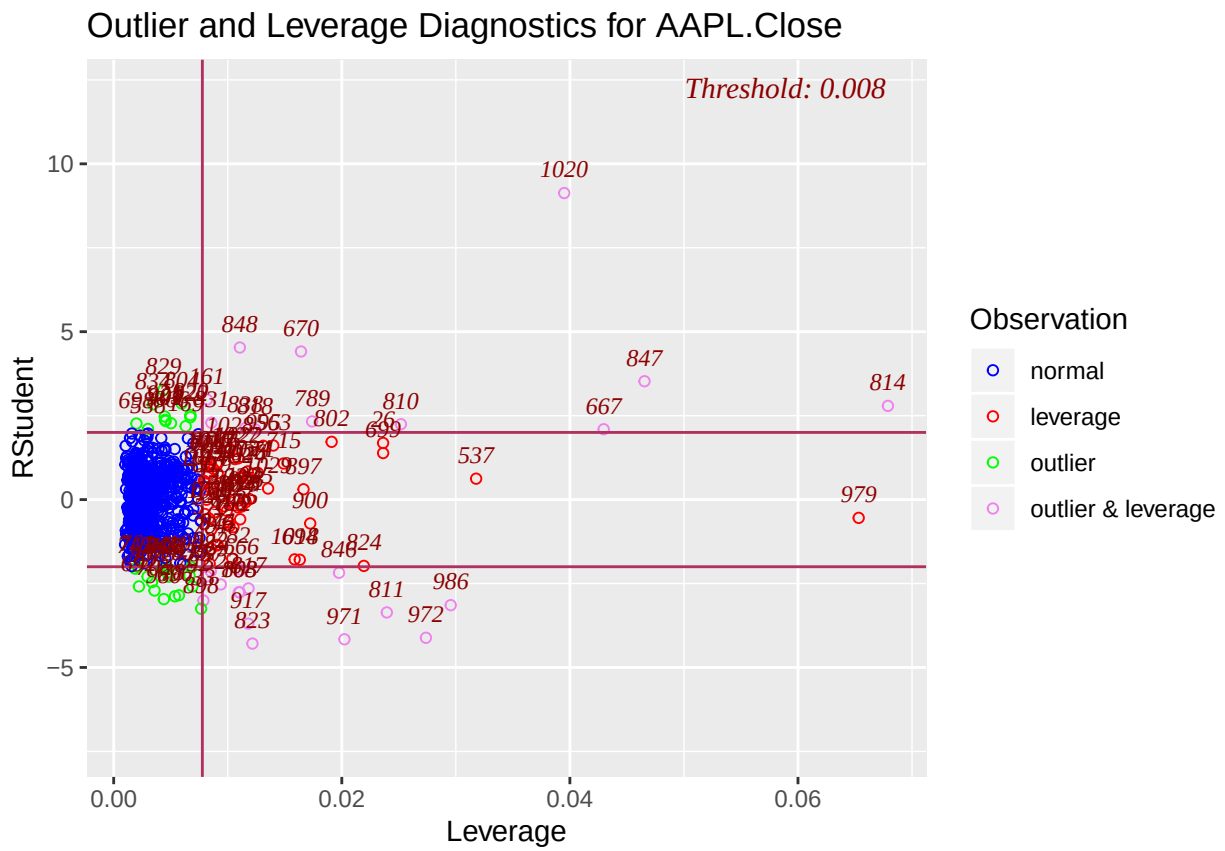


No disernable trend is spotted. It suggests a linear relationship.

```
# Check for outliers and leverage points
library(olsrr)
ols_plot_resid_stud_fit(model_lm_best)
```




```
ols_plot_resid_lev(model_lm_best)
```



There are outliers and some of them are leverage points as well. We need to figure out what affected the close price on these dates instead of deleting them. Meanwhile, it implies that the financial market is higly unstable.

```
# Check for muticollinarity
library(car)
vif(model_lm_best)
```

```
## AAPL.Open AAPL.High  AAPL.Low
## 1517.2387 1299.9905  902.1389
```

It shows that there is multicollinearity among these predictors. However, removing one of them leads to the decrease of the Adjusted R^2. Therefore, all variables are kept.

CV Multivariate Linear Regression

```
# Fit lm model using different 10 * k-fold CV
set.seed(1)
k = 2
RMSE <- rep(100, 10)
repeat {
    model <- train(
    fmla_best,
    aapl_df[train, ],
```

```
    method = "lm",
    trControl = trainControl(
        method = "repeatedcv",
        number = k,
        repeats = 10,
        verboseIter = F
        )
    )
    RMSE[k] <- mean(model$resample$RMSE)

    if (k >= 10) {
        break;
    }
    k = k + 1
}
RMSE
```

```
##  [1] 100.0000000   0.8634279   0.8666983   0.8624726   0.8611497
##  [6]   0.8611761   0.8607509   0.8584124   0.8582630   0.8576270
```

```
which.min(RMSE)
```

```
## [1] 10
```

The optimal number of folds is 9 which results in the lowest RMSE.

```
set.seed(1)
# Fit a 10 * 9-fold CV linear model
model_lm_best_cv <- train(
    fmla_best,
    aapl_df[train, ],
    method = "lm",
    trControl = trainControl(
        method = "repeatedcv",
        number = 9,
        repeats = 10,
        verboseIter = F
        )
    )

calculate_test_RMSE(model_lm_best_cv, "CV linear model")
```

```
## CV linear model test RMSE: 0.83377
```

```
## [1] 0.83377
```

The CV model does not improve the performance of the original multiple linear regression model.

CV Multivariate Linear Regression w PCA preprocessing

```r
set.seed(1)
# Fit a 10 * 9-fold CV linear model with PCA preprocess
model_lm_pca <- train(
    fmla_best,
    aapl_df[train, ],
    method = "lm",
    preProcess = c("center", "scale", "pca"),
    trControl = trainControl(
        method = "repeatedcv",
        number = 9,
        repeats = 10,
        verboseIter = F
        )
)

calculate_test_RMSE(model_lm_pca, "Best subset PCA preprocess linear model")
```

```
## Best subset PCA preprocess linear model test RMSE: 1.258917
```

```
## [1] 1.258917
```

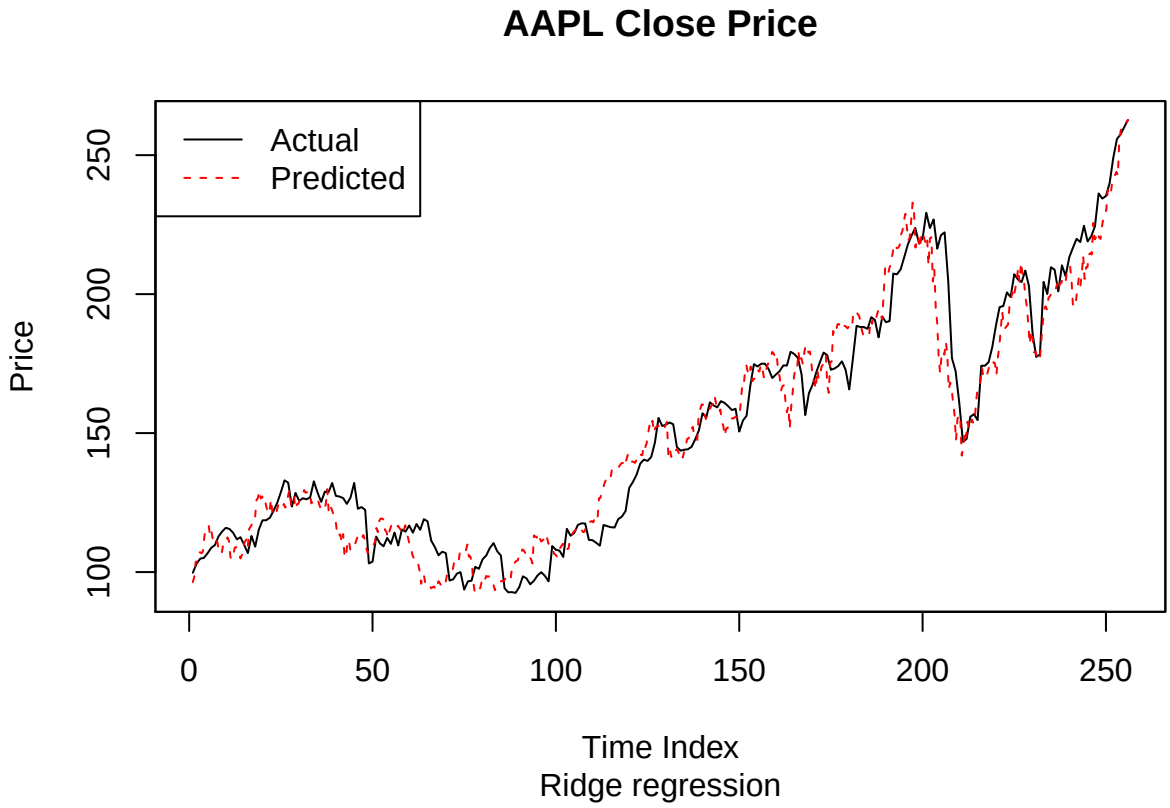
PCA preprocess does not help improve the performance of the ogirinal linear model.

Ridge Regression

```r
set.seed(1)
library(glmnet)
trn <- sample(c(TRUE, FALSE), nrow(aapl_df), rep = TRUE)
tst <- (!trn)
# Trim off mpg column
X <- model.matrix(AAPL.Close ~ ., aapl_df)
y <- aapl_df$AAPL.Close

# Specify a vector of 100 values of lambda for tuning
grid <- 10^seq(10, -2, length = 100)

# Fit ridge regression model
model_ridge <- glmnet(X[trn, ], y[trn], alpha = 0, lambda = grid, standardize=FALSE)

# Fit cross-validation ridge regression model
model_ridge_cv <- cv.glmnet(X[trn, ], y[trn], alpha = 0, parallel = T)

# Obtain the best value of lambda
best_lambda <- model_ridge_cv$lambda.min

# Access the performance of the ridge model on test data
p_test <- predict(model_ridge, s = best_lambda, newx =X[tst, ])
MSE <- sqrt(mean((y[tst] - p_test)^2))
cat("Ridge model test RMSE:", MSE, "\n\n")
```

```
## Ridge model test RMSE: 1.199273
```

```
plot(aapl_df[test, 4], type = "l", col = "black", main = "AAPL Close Price", sub = "Ridge regression", l
par(new = T)
plot(p_test, type = "l", col = 'red', lty = 2, xlab = "", ylab = "", xaxt = "n", yaxt = "n")
legend("topleft", c("Actual", "Predicted"), col = c("black", "red"), lty = 1:2)
```




```
# Print out the best coefficients
predict(model_ridge, type = "coefficients", s = best_lambda, newx =X[tst, ])
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept)    4.240078e+01
## (Intercept)    .
## AAPL.Open     -1.167801e-01
## AAPL.High      2.496978e-01
## AAPL.Low       4.930493e-02
## Lag.1          1.497672e-02
## Lag.2         -2.221193e-03
## Lag.3         -1.170855e-04
## Lag.4          5.866054e-03
## Lag.5          9.580030e-03
## Lag.6          3.884985e-03
## Lag.7          6.938748e-03
## Lag.8          1.338014e-03
```

```
## Lag.9         -3.504066e-04
## Lag.10        -2.337754e-03
## Open.High     -6.095482e-04
## Open.Low       2.009370e-03
## High.Low       3.602157e-03
## Open.High.Low -9.685770e-06
```

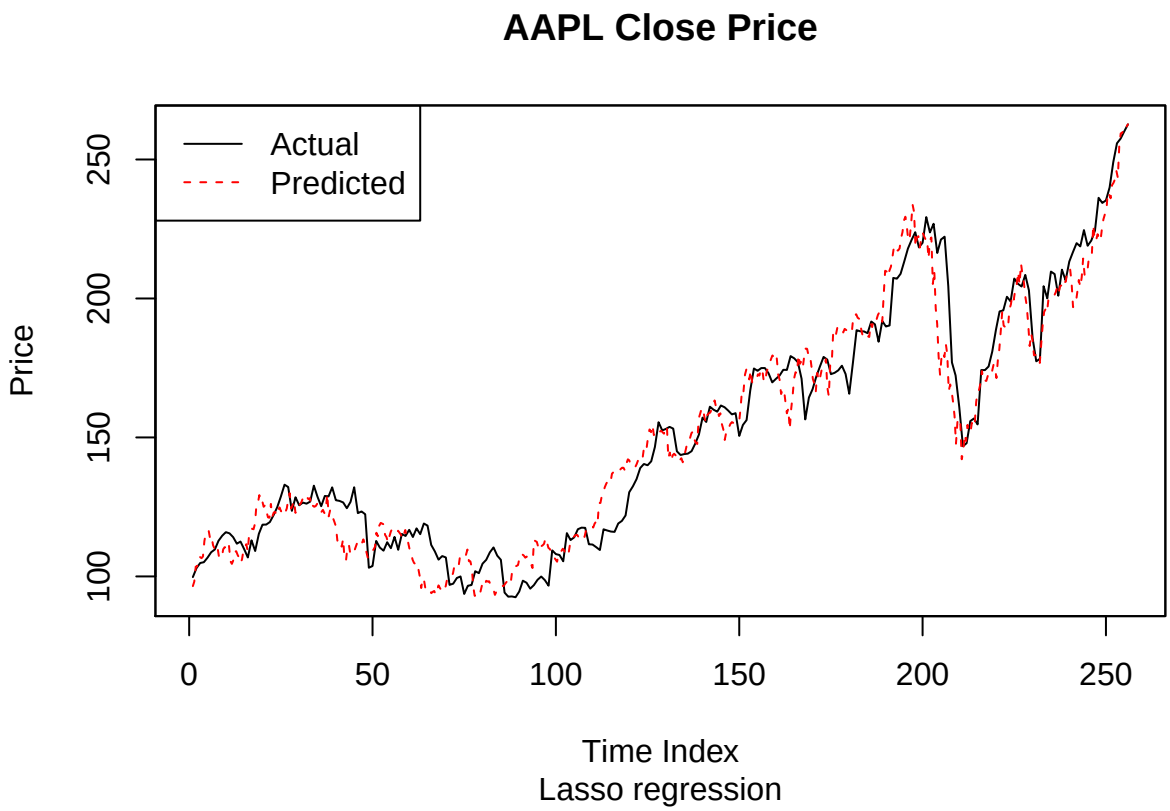
The riddge model perfors worse than the ogiginal linear model.

Lasso Regression

```
set.seed(1)
# Fit ridge regression model
model_lasso <- glmnet(X[trn, ], y[trn], alpha = 1, lambda = grid, standardize=FALSE)

# Fit cross-validation ridge regression model
model_lasso_cv <- cv.glmnet(X[trn, ], y[trn], alpha = 1, parallel = T)

# Obtain the best value of lambda
best_lambda <- model_lasso_cv$lambda.min

# Access the performance of the ridge model on test data
p_test <- predict(model_lasso, s = best_lambda, newx =X[tst, ])
MSE <- sqrt(mean((y[tst] - p_test)^2))
cat("Lasso model test RMSE:", MSE, "\n\n")
```

```
## Lasso model test RMSE: 1.313807
```

```
plot(aapl_df[test, 4], type = "l", col = "black", main = "AAPL Close Price", sub = "Lasso regression", l
par(new = T)
plot(p_test, type = "l", col = 'red', lty = 2, xlab = "", ylab = "", xaxt = "n", yaxt = "n")
legend("topleft", c("Actual", "Predicted"), col = c("black", "red"), lty = 1:2)
```

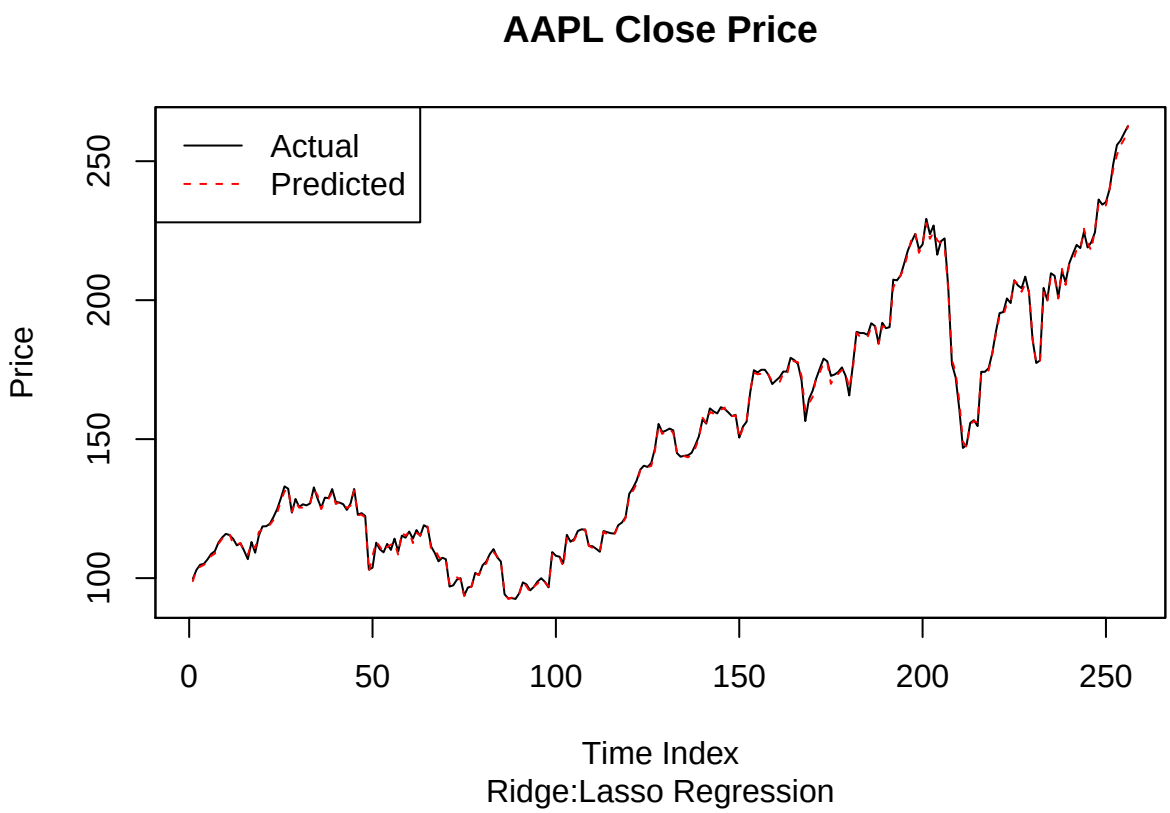



```
# Print out the best coefficients
predict(model_lasso, type = "coefficients", s = best_lambda, newx =X[tst, ])
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept)     4.773127e+01
## (Intercept)     .
## AAPL.Open       .
## AAPL.High       .
## AAPL.Low        8.183082e-02
## Lag.1           .
## Lag.2           .
## Lag.3           .
## Lag.4           .
## Lag.5           .
## Lag.6           .
## Lag.7           .
## Lag.8           .
## Lag.9           .
## Lag.10          .
## Open.High       1.281371e-03
## Open.Low       -8.597724e-05
## High.Low        4.403576e-03
## Open.High.Low -1.092844e-05
```

Ridge:Lasso Regression

```
# Choose the best alpha between ridge and lasso
model_glmnet <- train(
    AAPL.Close ~ .,
    aapl_df[train,],
    method = "glmnet",
    tuneGrid = expand.grid(
        alpha = 0:1,
        lambda = 10^seq(10, -2, length = 100)
    ),
    trControl = trainControl(
        method = "cv",
        number = 9,
        verboseIter = F
    )
)

plot_model(model_glmnet, "Ridge:Lasso Regression")
```




```
calculate_test_RMSE(model_glmnet, "Ridge:Lasso model")
```

```
## Ridge:Lasso model test RMSE: 1.337053

## [1] 1.337053
```

The lasso model has an even worse performance than the ridge one.

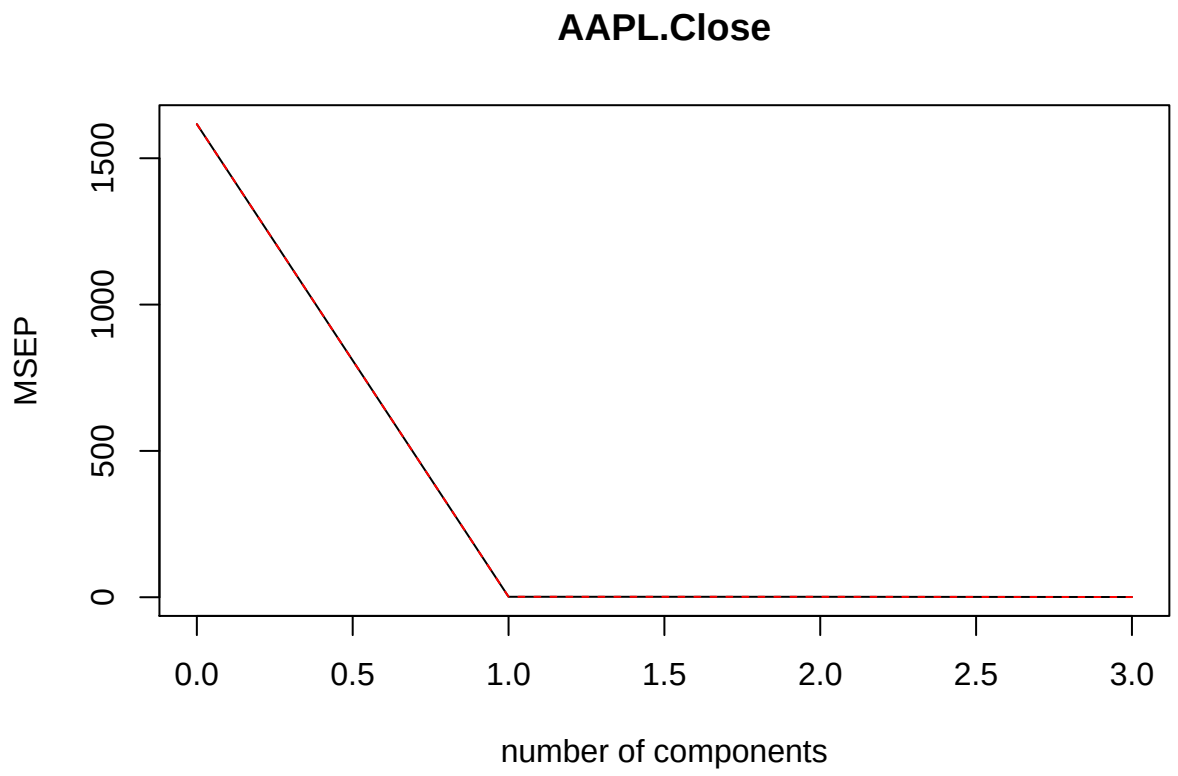
Principal Component Regression

```
# Function calculate the train and test RMSE for PCA model
calculate_PCA_test_RMSE <- function(model, model_name, ncomp) {

    # Predict the close price
    p <- predict(model, newdata = aapl_df[test, ], ncomp = ncomp)

    # Compute errors
    error <- p - aapl_df[test, ][["AAPL.Close"]]

    # Calculate RMSE
    RMSE <- sqrt(mean(error^2))
    cat(model_name, "test RMSE:", RMSE, "\n")
}
```

```
library(pls)
set.seed(1)
model_pcr <- pcr(fmla_best,
                 data = aapl_df,
                 subset = train,
                 scale = T,
                 validation = "CV")
summary(model_pcr)
```

```
## Data:    X dimension: 1029 3
##  Y dimension: 1029 1
## Fit method: svdpc
## Number of components considered: 3
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps
## CV           40.21     1.24    1.204   0.8648
## adjCV        40.21     1.24    1.203   0.8643
##
## TRAINING: % variance explained
##             1 comps  2 comps  3 comps
## X             99.96    99.99   100.00
## AAPL.Close    99.91    99.91    99.95
```

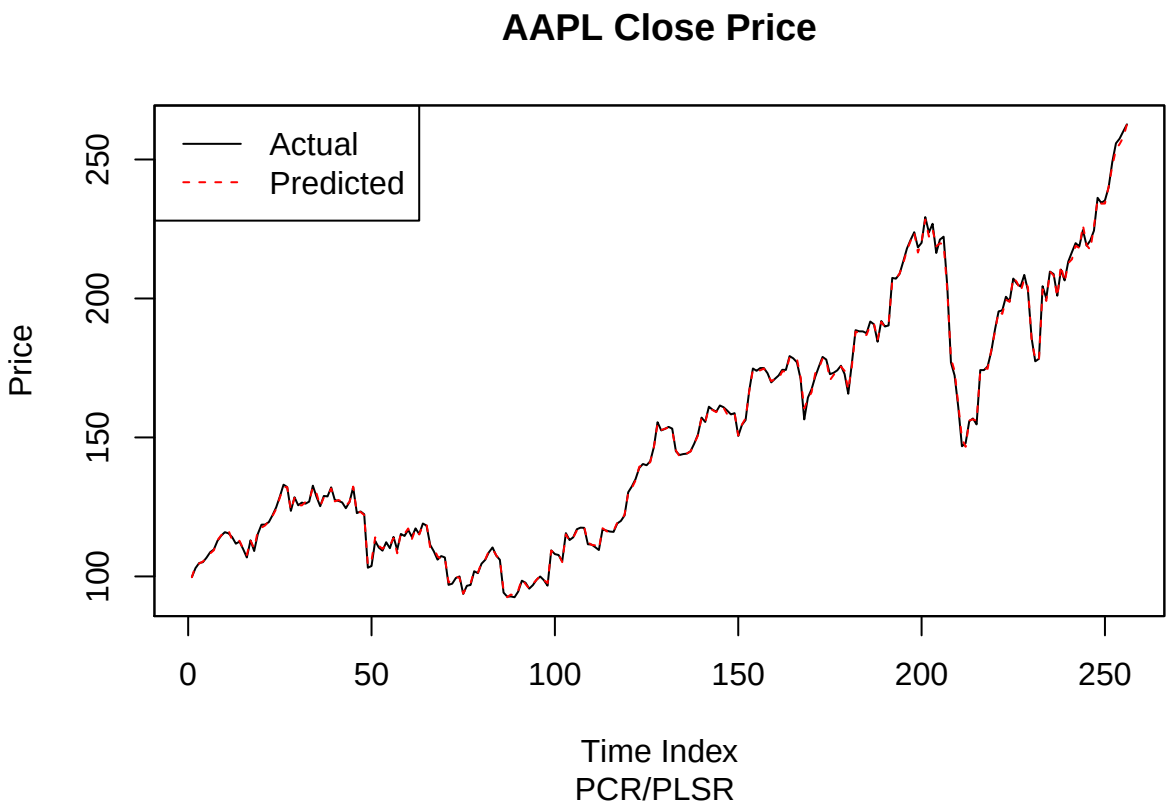
```
validationplot(model_pcr, val.type = "MSEP")
```




```
calculate_PCA_test_RMSE(model_pcr, "Best subset PCA model", 3)
```

```
## Best subset PCA model test RMSE: 0.83377
```

```
    plot(aapl_df[test, 4], type = "l", col = "black", main = "AAPL Close Price", sub = "PCR/PLSR ", lty
    par(new = T)
    plot(predict(model_pcr, aapl_df[test, ], 3), type = "l", col = 'red', lty = 2, xlab = "", ylab = ""
    legend("topleft", c("Actual", "Predicted"), col = c("black", "red"), lty = 1:2)
```



Principal component regression model has the same performance as the linear regressio model.

Partial Least Squares Regression

```
set.seed(1)
model_plsr <- plsr(fmla_best,
                   data = aapl_df,
                   subset = train,
                   scale = T,
                   validation = "CV")
summary(model_plsr)
```

```
## Data:    X dimension: 1029 3
##  Y dimension: 1029 1
## Fit method: kernelpls
## Number of components considered: 3
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps
## CV           40.21     1.24   0.9002   0.8648
## adjCV        40.21     1.24   0.8988   0.8643
##
## TRAINING: % variance explained
```

```
##             1 comps  2 comps  3 comps
## X             99.96    99.98   100.00
## AAPL.Close    99.91    99.95    99.95
```

```
calculate_PCA_test_RMSE(model_plsr, "Full  PLR model", 3)
```

```
## Full  PLR model test RMSE: 0.83377
```

Partial least squares model has the same performance as the linear model.

Polynomial Regression

```
set.seed(1)
# Perform polynomial regressions
poly_1 <- lm(AAPL.Close ~ poly(AAPL.Low, 1) + poly(AAPL.High, 1) + poly(AAPL.Open, 1), data = aapl_df[t
poly_2 <- lm(AAPL.Close ~ poly(AAPL.Low, 2) + poly(AAPL.High, 2) + poly(AAPL.Open, 2), data = aapl_df[t
poly_3 <- lm(AAPL.Close ~ poly(AAPL.Low, 3) + poly(AAPL.High, 3) + poly(AAPL.Open, 3), data = aapl_df[t
poly_4 <- lm(AAPL.Close ~ poly(AAPL.Low, 4) + poly(AAPL.High, 4) + poly(AAPL.Open, 4), data = aapl_df[t
poly_5 <- lm(AAPL.Close ~ poly(AAPL.Low, 5) + poly(AAPL.High, 5) + poly(AAPL.Open, 5), data = aapl_df[t
poly_6 <- lm(AAPL.Close ~ poly(AAPL.Low, 6) + poly(AAPL.High, 6) + poly(AAPL.Open, 6), data = aapl_df[t

# Peform hypothesis tests
anova(poly_1, poly_2, poly_3, poly_4, poly_5, poly_6)
```

```
## Analysis of Variance Table
##
## Model 1: AAPL.Close ~ poly(AAPL.Low, 1) + poly(AAPL.High, 1) + poly(AAPL.Open,
##     1)
## Model 2: AAPL.Close ~ poly(AAPL.Low, 2) + poly(AAPL.High, 2) + poly(AAPL.Open,
##     2)
## Model 3: AAPL.Close ~ poly(AAPL.Low, 3) + poly(AAPL.High, 3) + poly(AAPL.Open,
##     3)
## Model 4: AAPL.Close ~ poly(AAPL.Low, 4) + poly(AAPL.High, 4) + poly(AAPL.Open,
##     4)
## Model 5: AAPL.Close ~ poly(AAPL.Low, 5) + poly(AAPL.High, 5) + poly(AAPL.Open,
##     5)
## Model 6: AAPL.Close ~ poly(AAPL.Low, 6) + poly(AAPL.High, 6) + poly(AAPL.Open,
##     6)
##   Res.Df    RSS Df Sum of Sq       F    Pr(>F)
## 1   1025 752.94
## 2   1022 748.52  3    4.4219  2.1248 0.0954602 .
## 3   1019 735.57  3   12.9516  6.2233 0.0003454 ***
## 4   1016 705.92  3   29.6454 14.2448 4.264e-09 ***
## 5   1013 702.98  3    2.9397  1.4125 0.2375896
## 6   1010 700.65  3    2.3310  1.1201 0.3398823
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
coef(summary(poly_6))
```

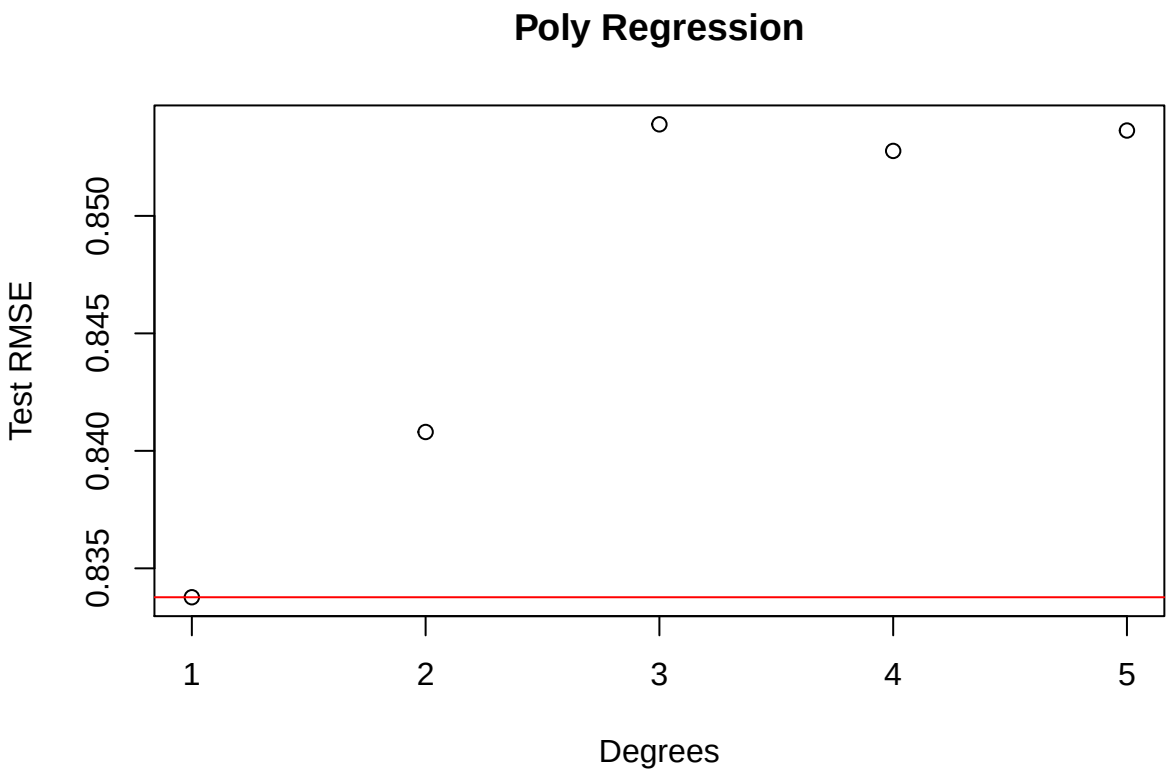
```
##                          Estimate  Std. Error       t value      Pr(>|t|)
## (Intercept)          149.7270554  0.02596467 5766.56851058  0.000000e+00
## poly(AAPL.Low, 6)1   914.4660529 28.61317555   31.95961425 1.960243e-155
## poly(AAPL.Low, 6)2    -2.7054864 14.48475077   -0.18678171  8.518693e-01
## poly(AAPL.Low, 6)3   -30.9421023 10.35282812   -2.98875842  2.868925e-03
## poly(AAPL.Low, 6)4   -45.0613472  8.94325771   -5.03858311  5.554163e-07
## poly(AAPL.Low, 6)5    -8.9467798  7.15785490   -1.24992472  2.116166e-01
## poly(AAPL.Low, 6)6     8.5540963  5.76641426    1.48343424  1.382710e-01
## poly(AAPL.High, 6)1 1106.8677175 34.14771760   32.41410541 1.422683e-158
## poly(AAPL.High, 6)2  -26.3915388 17.69661487   -1.49133261  1.361864e-01
## poly(AAPL.High, 6)3    4.7009184 12.96248874    0.36265554  7.169381e-01
## poly(AAPL.High, 6)4   52.4903030 11.78070994    4.45561458  9.299374e-06
## poly(AAPL.High, 6)5   12.3355523  7.21211941    1.71039213  8.750049e-02
## poly(AAPL.High, 6)6   -7.3059358  5.94692332   -1.22852363  2.195366e-01
## poly(AAPL.Open, 6)1 -733.3270480 36.99089773  -19.82452692  3.901382e-74
## poly(AAPL.Open, 6)2   28.5367363 17.96564989    1.58840545  1.125077e-01
## poly(AAPL.Open, 6)3   27.5372789 12.88559067    2.13705989  3.283281e-02
## poly(AAPL.Open, 6)4   -5.3995258 12.14647341   -0.44453444  6.567515e-01
## poly(AAPL.Open, 6)5   -3.0896672  7.22472093   -0.42765211  6.689956e-01
## poly(AAPL.Open, 6)6   -0.4053587  5.80010123   -0.06988821  9.442965e-01
```

A 4^{th} polynomial appear to provide a reasonale fit to the data. However, from the p-value of the coefficients of the 6th degree polynomial regression model, it shows that the linear are the best.

```
set.seed(1)
poly <- rep(0, 5)
for (i in 1:5) {
# Fit a polynomial regression model up to degrees of 5
model_poly_best <- lm(AAPL.Close ~ poly(AAPL.Low, i)
                      + poly(AAPL.High, i)
                      + poly(AAPL.Open, i),
                      data = aapl_df[train, ])

cat("Degrees:", i, "\n")
poly[i] <- calculate_test_RMSE(model_poly_best, "Poly model")
}
```

```
## Degrees: 1
## Poly model test RMSE: 0.83377
## Degrees: 2
## Poly model test RMSE: 0.8408028
## Degrees: 3
## Poly model test RMSE: 0.8538956
## Degrees: 4
## Poly model test RMSE: 0.8527649
## Degrees: 5
## Poly model test RMSE: 0.8536326
```

```
# Plot the test RMSE
plot_rmse <- function(data, main, xlab, ylab) {
plot(data, main = main, xlab = xlab, ylab = ylab)
abline(h = 0.83377, col = 'red')
}
plot_rmse(poly, "Poly Regression", "Degrees", "Test RMSE")
```




```
# Build the best poly model
model_poly <- lm(AAPL.Close ~ poly(AAPL.Low, 1)
                        + poly(AAPL.High, 1)
                        + poly(AAPL.Open, 1),
                        data = aapl_df[train, ])
```

As the degree goes up, the training RMSE goes down, while it is the opposite case for the test RMSE. All polynomial regression model with degrees of more than 3 are overfitted model. The quadratic polynomial regression model has better performance than the linear model.

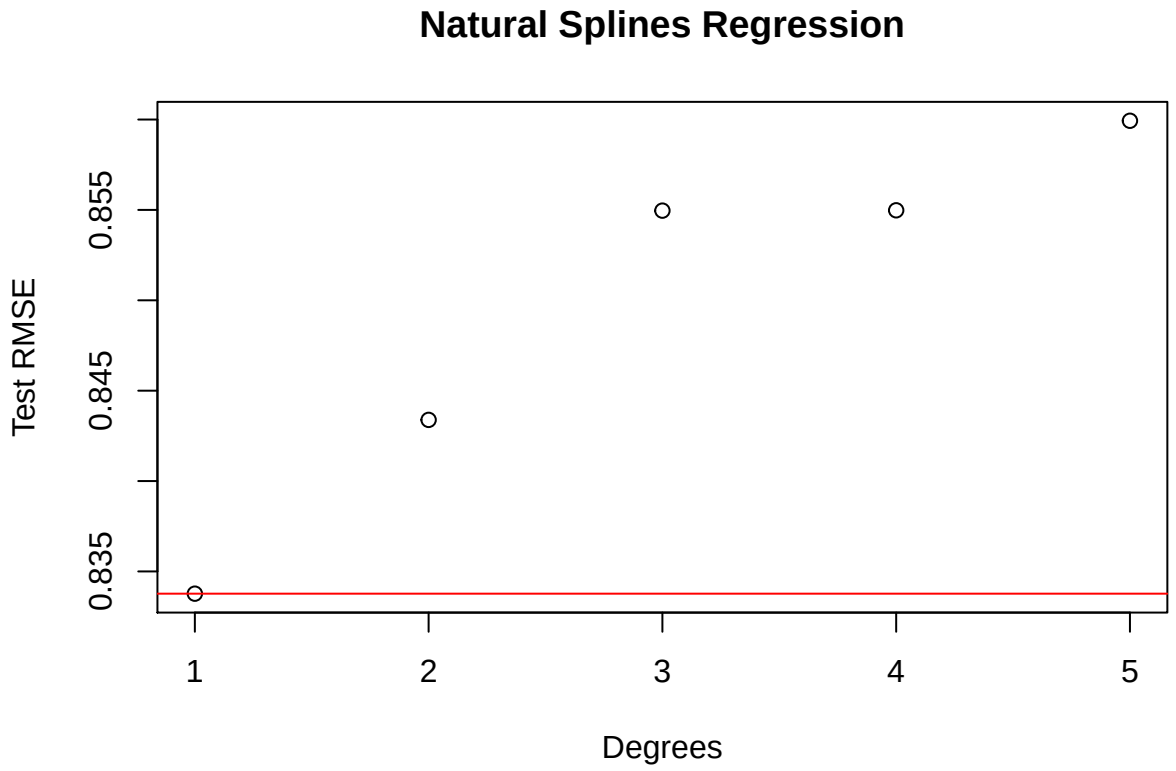
Natural Splines Regression

```
library(gam)

# Fit GAMs using natural splines
ns <- rep(0, 5)
for (i in 1:5) {
model_ns <- lm(AAPL.Close ~ ns(AAPL.Open, df = i)
          + ns(AAPL.High, df = i)
          + ns(AAPL.Low, df = i),
          data = aapl_df[train, ])

cat("Degrees:", i, "\n")
```

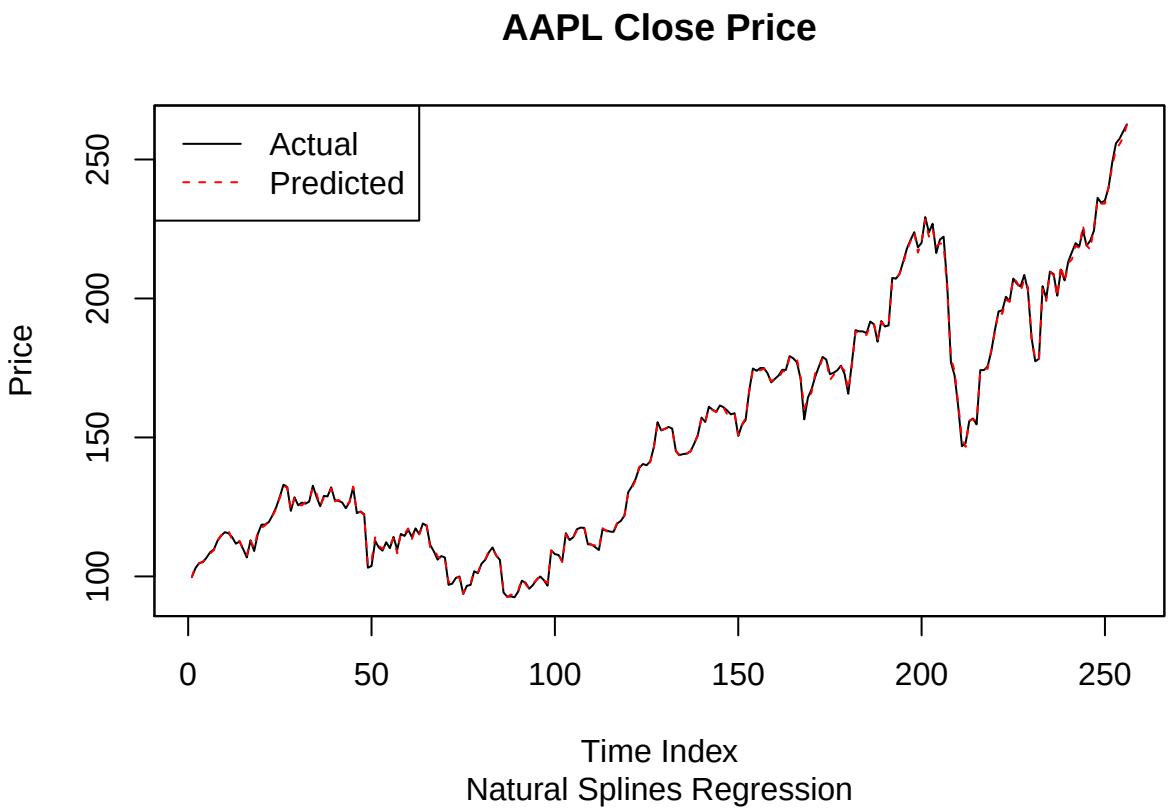
```
ns[i] <- calculate_test_RMSE(model_ns, "GAM natrual splines model")
}
```

```
## Degrees: 1
## GAM natrual splines model test RMSE: 0.83377
## Degrees: 2
## GAM natrual splines model test RMSE: 0.8433837
## Degrees: 3
## GAM natrual splines model test RMSE: 0.8549608
## Degrees: 4
## GAM natrual splines model test RMSE: 0.8549786
## Degrees: 5
## GAM natrual splines model test RMSE: 0.8599322
```

```
# Plot the test RMSE
plot_rmse(ns, main = "Natural Splines Regression", xlab = "Degrees", ylab = "Test RMSE")
```




```
model_ns <- lm(AAPL.Close ~ ns(AAPL.Open, df = 1)
          + ns(AAPL.High, df = 1)
          + ns(AAPL.Low, df = 1),
          data = aapl_df[train, ])
plot_model(model_ns, "Natural Splines Regression")
```



Natural Splines Regression

The natural splines model with 3 degress of freedom has slightly better performance than the quadratic and linear regression.

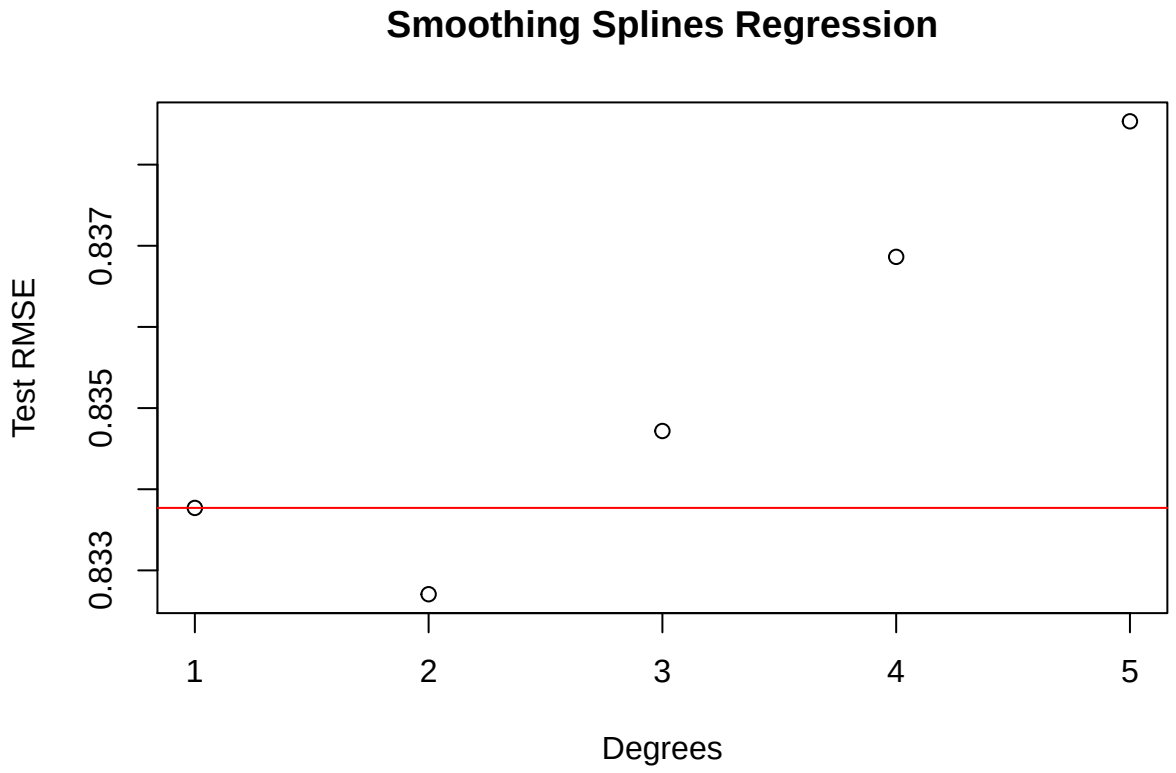
Smoothing Splines Regression

```
set.seed(1)
# Fit GAMs using somoothing splines
s <- rep(0, 5)
for (i in 1:5) {
model_s <- gam(AAPL.Close ~ s(AAPL.Open, df = i)
           + s(AAPL.High, df = i)
           + s(AAPL.Low, df = i),
           data = aapl_df[train, ])

cat("Degrees:", i, "\n")
s[i] <- calculate_test_RMSE(model_s, "GAM smoothing splines model")
}
```

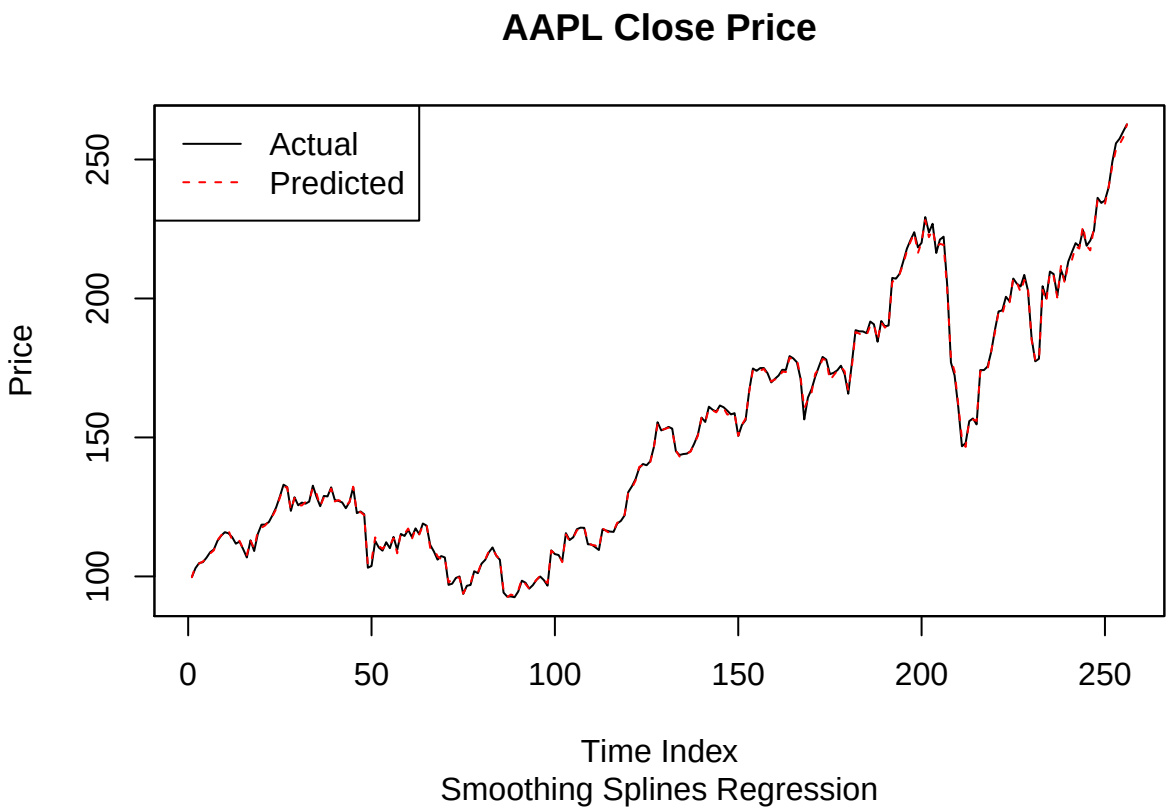
```
## Degrees: 1
## GAM smoothing splines model test RMSE: 0.8337696
## Degrees: 2
## GAM smoothing splines model test RMSE: 0.8327063
## Degrees: 3
## GAM smoothing splines model test RMSE: 0.8347175
## Degrees: 4
```

```
## GAM smoothing splines model test RMSE: 0.8368632
## Degrees: 5
## GAM smoothing splines model test RMSE: 0.8385335
```

```
plot_rmse(s, main = "Smoothing Splines Regression", xlab = "Degrees", ylab = "Test RMSE")
```




```
model_s <- gam(AAPL.Close ~ s(AAPL.Open, df = 2)
           + s(AAPL.High, df = 2)
           + s(AAPL.Low, df = 2),
           data = aapl_df[train, ])

    plot(aapl_df[test, 4], type = "l", col = "black", main = "AAPL Close Price", sub = "Smoothing Splin
    par(new = T)
    plot(predict(model_s, aapl_df[test, ]), type = "l", col = 'red', lty = 2, xlab = "", ylab = "", xax
    legend("topleft", c("Actual", "Predicted"), col = c("black", "red"), lty = 1:2)
```



The smoothing splines model performs worse than the previous two models.

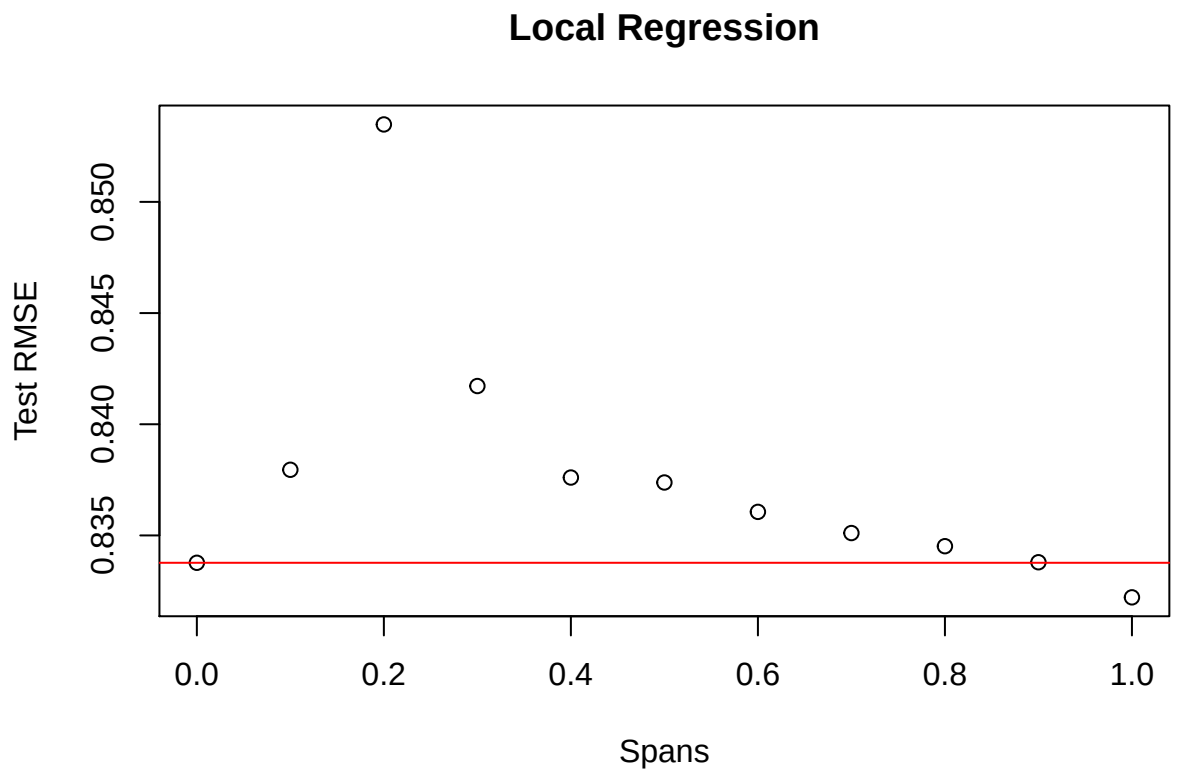
Local Regression

```
# Fit GAMs local somoothing splines
lo <- rep(0, 11)
j <- 1
for (i in seq(0, 1, length.out = 11)) {
gam <- gam(AAPL.Close ~ lo(AAPL.Open, span = i)
           + lo(AAPL.High, span = i)
           + lo(AAPL.Low, span = i),
           data = aapl_df[train, ])

cat("Span =", i, "\n")
lo[j] <- calculate_test_RMSE(gam, "GAM local regression model")
j = j + 1
}
```

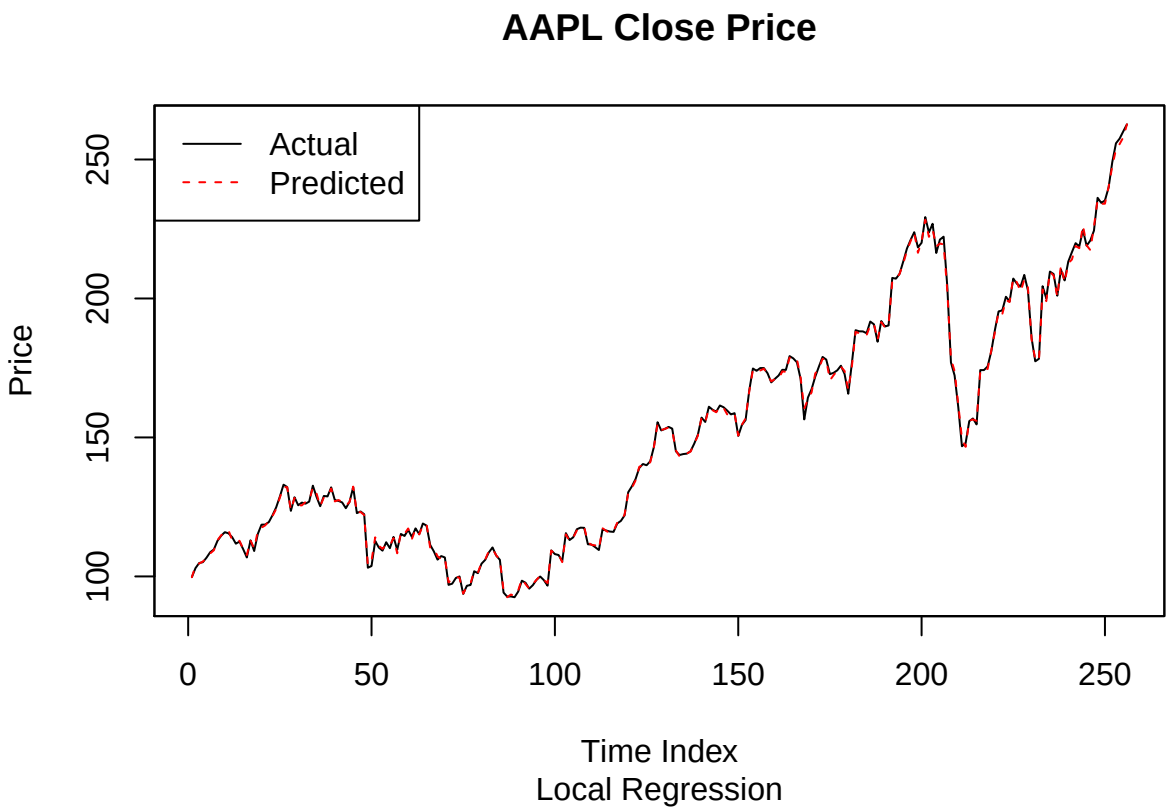
```
## Span = 0
## GAM local regression model test RMSE: 0.83377
## Span = 0.1
## GAM local regression model test RMSE: 0.8379531
## Span = 0.2
## GAM local regression model test RMSE: 0.8534827
```

```
## Span = 0.3
## GAM local regression model test RMSE: 0.8417186
## Span = 0.4
## GAM local regression model test RMSE: 0.8376032
## Span = 0.5
## GAM local regression model test RMSE: 0.8373807
## Span = 0.6
## GAM local regression model test RMSE: 0.8360588
## Span = 0.7
## GAM local regression model test RMSE: 0.8351069
## Span = 0.8
## GAM local regression model test RMSE: 0.8345154
## Span = 0.9
## GAM local regression model test RMSE: 0.8337931
## Span = 1
## GAM local regression model test RMSE: 0.8322157
```

```r
plot(x = seq(0, 1, length.out=11), y = lo, main = "Local Regression", xlab = "Spans", ylab = "Test RMSE
abline(h = 0.83377, col = 'red')
```




```r
model_lo <- gam(AAPL.Close ~ lo(AAPL.Open, span = 1)
           + lo(AAPL.High, span = 1)
           + lo(AAPL.Low, span = 1),
           data = aapl_df[train, ])
plot_model(model_lo, "Local Regression")
```



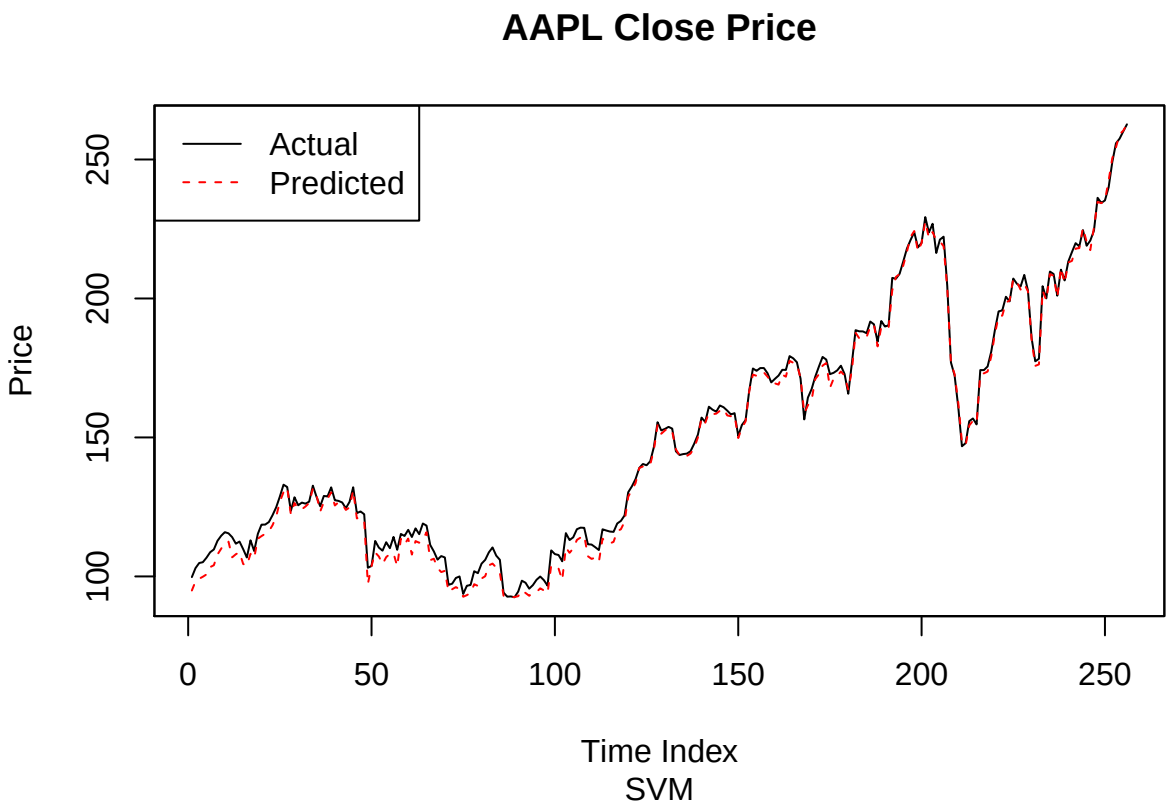
Support Vector Machine

```
library(e1071)
model_svm <- svm(fmla_best, data = aapl_df[train, ])
calculate_test_RMSE(model_svm, "SVM")
```

```
## SVM test RMSE: 1.556552
```

```
## [1] 1.556552
```

```
plot_model(model_svm, "SVM")
```



The local regression has the worst performance amond those non-linear regression models.

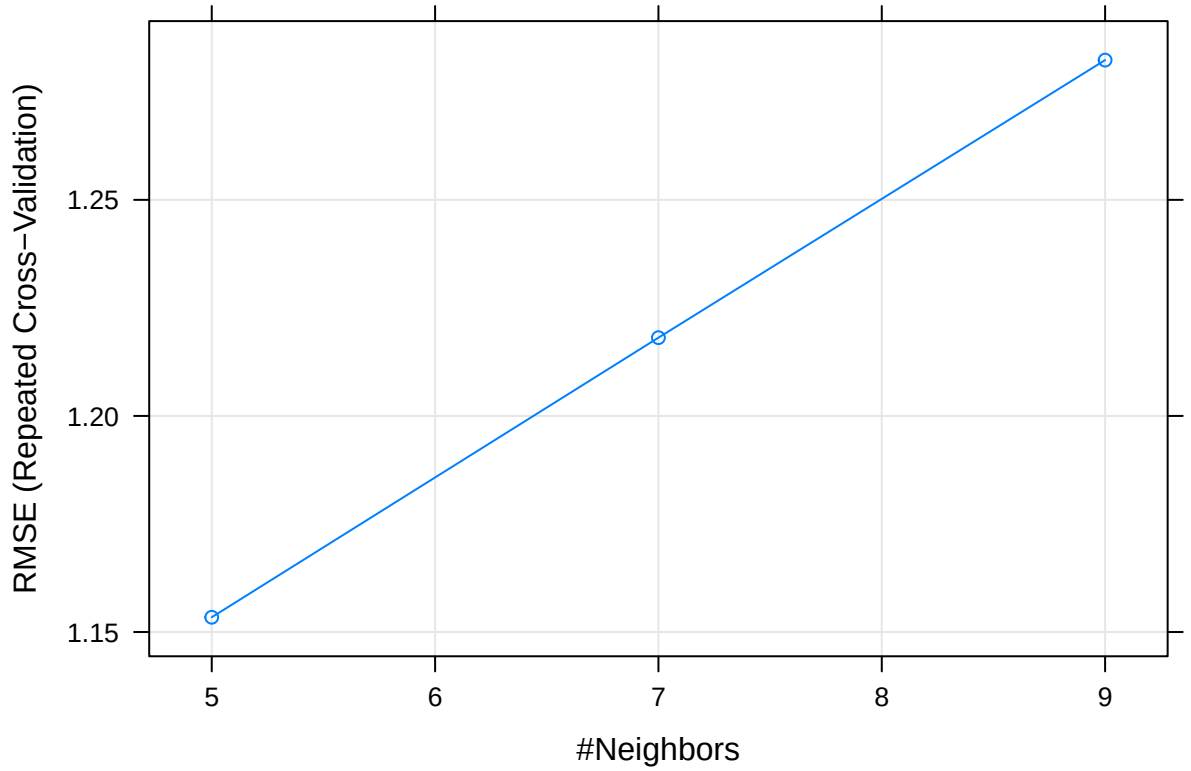
K-Nearest Neighbors

```
# Fit a 10*10-fold CV KNN model
set.seed(1)
model_knn <- train(
    fmla_best,
    aapl_df[train, ],
    method = "knn",
    trControl = trainControl(
        method = "repeatedcv",
        number = 10,
        repeats = 10,
        verboseIter = F
    )
)
model_knn
```

```
## k-Nearest Neighbors
##
## 1029 samples
##    3 predictor
##
```

```
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 928, 925, 925, 925, 925, 928, ...
## Resampling results across tuning parameters:
##
##   k  RMSE      Rsquared   MAE
##   5  1.153410  0.9991860  0.8329453
##   7  1.218109  0.9991013  0.8562370
##   9  1.282341  0.9990113  0.8785837
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 5.
```

```r
plot(model_knn)
```



The optimal K is 5 which leads to the lowest RMSE.

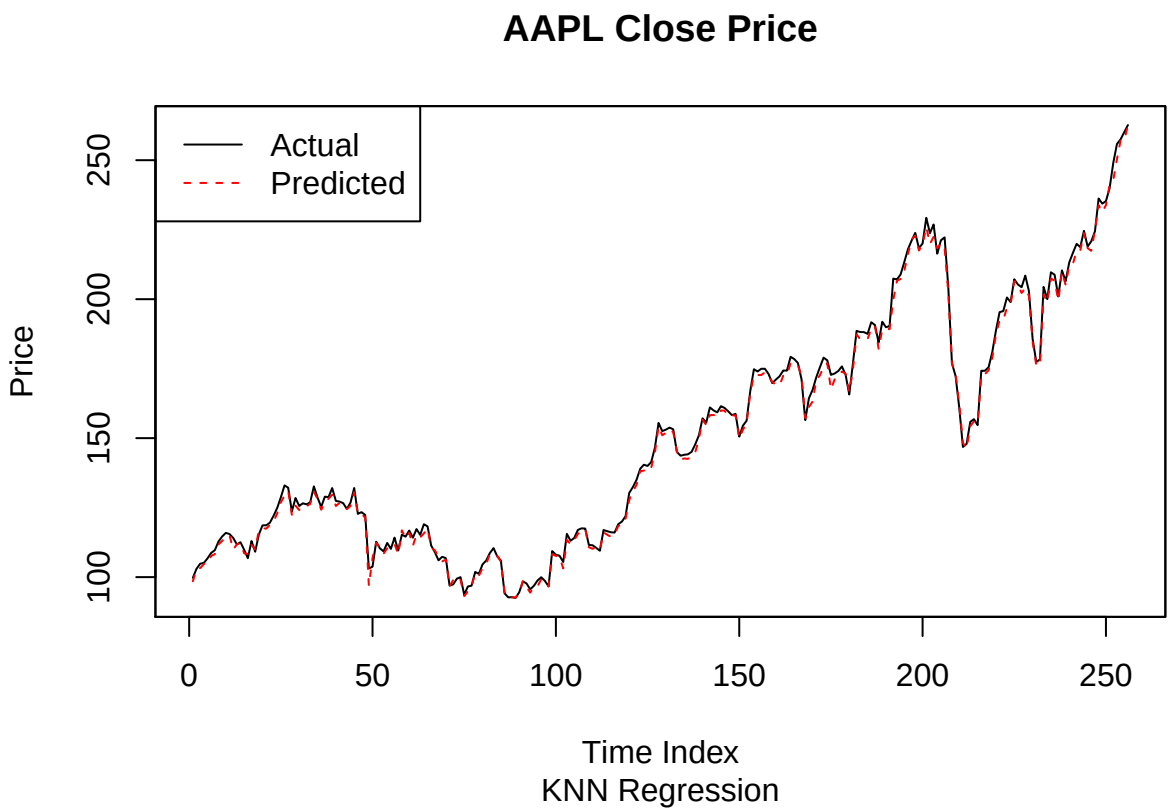
```r
# Fit a K=5 KNN model using 10*10 CV
set.seed(1)
model_knn <- train(
    fmla_best,
    aapl_df[train, ],
    method = "knn",
    preProcess = c("center", "scale", "pca"),
    tuneLenghth = 5,
    trControl = trainControl(
```

```
        method = "repeatedcv",
        number = 10,
        repeats = 10,
        verboseIter = F
    )
)

calculate_test_RMSE(model_knn, "KNN model")
```

```
## KNN model test RMSE: 1.32794
```

```
## [1] 1.32794
```

```
plot_model(model_knn, "KNN Regression")
```



The linear regression model has better performance than the KNN model.

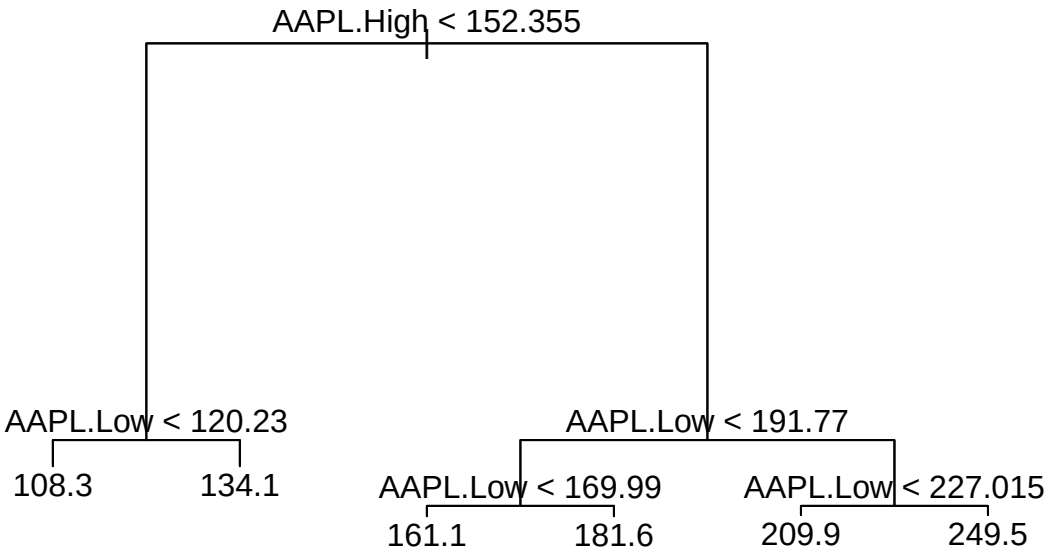
Decision Tree Regression

```
library(tree)

# Fit a regression tree model
set.seed(1)
```

```
model_tree <- tree(fmla_best,
                   data = aapl_df,
                   subset = train)
summary(model_tree)
```

```
##
## Regression tree:
## tree(formula = fmla_best, data = aapl_df, subset = train)
## Variables actually used in tree construction:
## [1] "AAPL.High" "AAPL.Low"
## Number of terminal nodes:  6
## Residual mean deviance:  67.26 = 68810 / 1023
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -19.4600  -6.9450   0.0865   0.0000   6.9820  18.4600
```

```
plot(model_tree)
text(model_tree, pretty = 0)
```

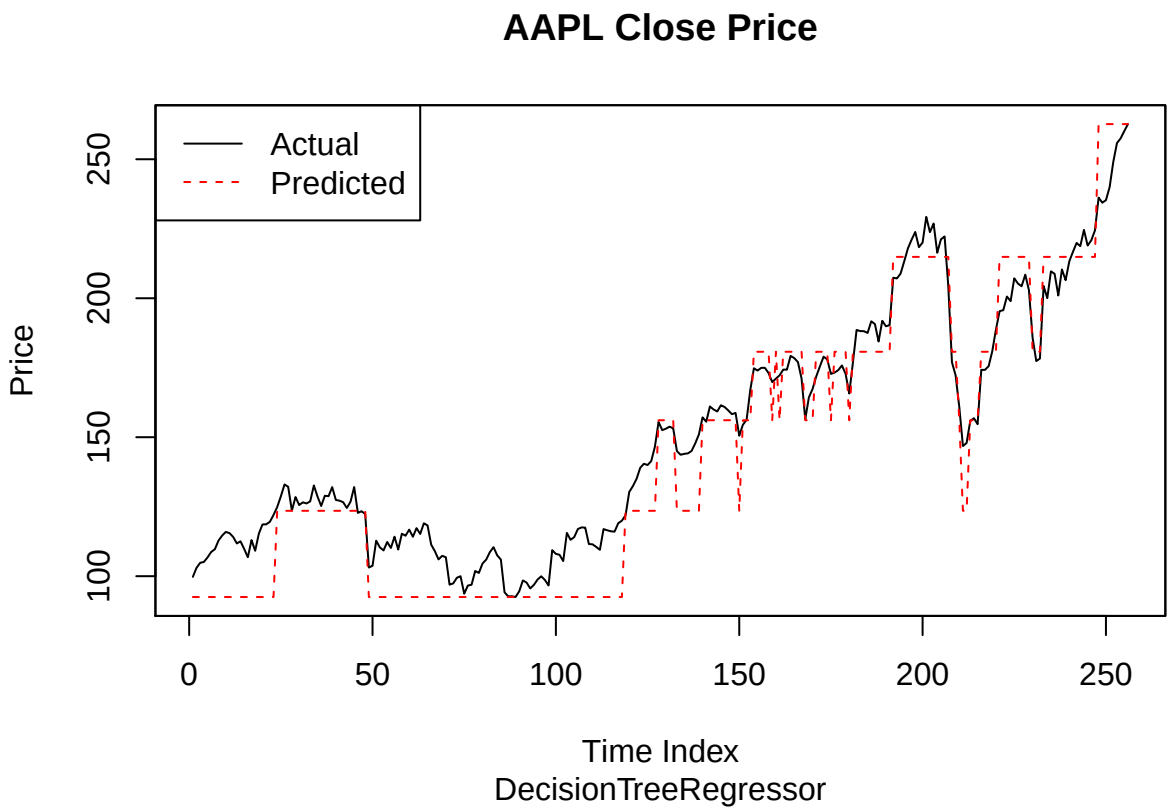



```
calculate_test_RMSE(model_tree, "Regression tree model")
```

```
## Regression tree model test RMSE: 8.071919
```

```
## [1] 8.071919
```

```
plot_model(model_tree, "DecisionTreeRegressor")
```




```
# Prune the tree
for (i in 2:6) {
model_tree_prune <- prune.tree(model_tree, best = i)
print(paste("Leaf nodes =", i))
calculate_test_RMSE(model_tree_prune, "Regression tree model")
}
```

```
## [1] "Leaf nodes = 2"
## Regression tree model test RMSE: 21.52818
## [1] "Leaf nodes = 3"
## Regression tree model test RMSE: 14.22221
## [1] "Leaf nodes = 4"
## Regression tree model test RMSE: 11.28961
## [1] "Leaf nodes = 5"
## Regression tree model test RMSE: 10.16642
## [1] "Leaf nodes = 6"
## Regression tree model test RMSE: 8.071919
```

Pruning the tree leads to even worse performance.

Bagging

```
library(randomForest)
set.seed(1)

# Use bagging to improve the performance of the regression tree
model_tree_bag <- randomForest(fmla_best,
                               data = aapl_df,
                               subset = train,
                               mtry = 3,
                               importance = T,
                               ntree = 2000)
model_tree_bag
```

```
##
## Call:
##  randomForest(formula = fmla_best, data = aapl_df, mtry = 3, importance = T,      ntree = 2000, subs
##                Type of random forest: regression
##                      Number of trees: 2000
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 1.357499
##                     % Var explained: 99.92
```

```
calculate_test_RMSE(model_tree_bag, "Bagging regression tree model")
```

```
## Bagging regression tree model test RMSE: 1.187881
```

```
## [1] 1.187881
```

```
set.seed(1)
# Use the caret package to perform bagging
model_bagging <- train(
  fmla_best,
  tuneLength = 3,
  data = aapl_df[train, ],
  method = "ranger",
  trControl = trainControl(
    method = "cv",
    number = 10,
    verboseIter = F
  )
)
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

```
model_bagging
```

```
## Random Forest
##
```

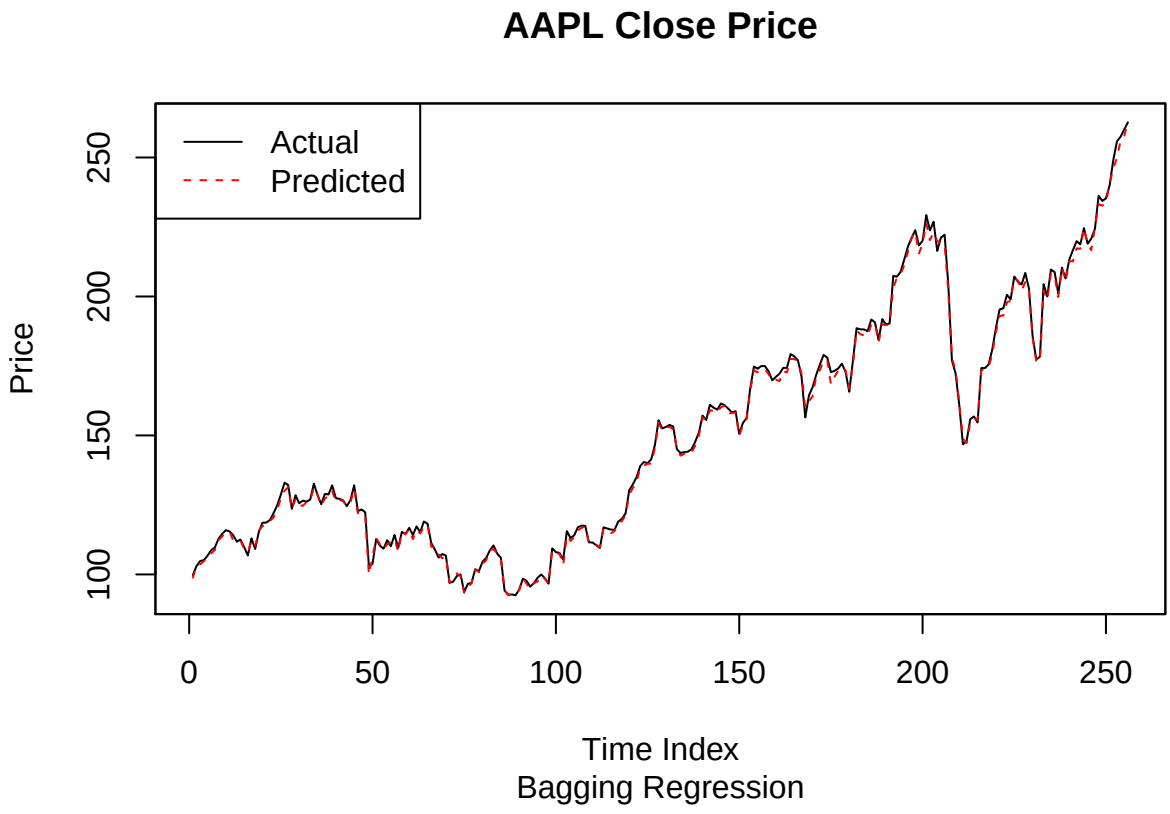
```
## 1029 samples
##    3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 928, 925, 925, 925, 925, 928, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   RMSE      Rsquared   MAE
##   2     variance    1.158987  0.9991817  0.8204066
##   2     extratrees  1.126973  0.9992266  0.7982295
##   3     variance    1.168780  0.9991699  0.8195747
##   3     extratrees  1.121985  0.9992325  0.7904626
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 3, splitrule =
##  extratrees and min.node.size = 5.
```

```
calculate_test_RMSE(model_bagging, "Bagging regression tree model")
```

```
## Bagging regression tree model test RMSE: 1.110949
```

```
## [1] 1.110949
```

```
plot_model(model_bagging, "Bagging Regression")
```



Bagging substancially improve the performance of the regression tree model. However, it still performs worse than the linear model.

Random Forest

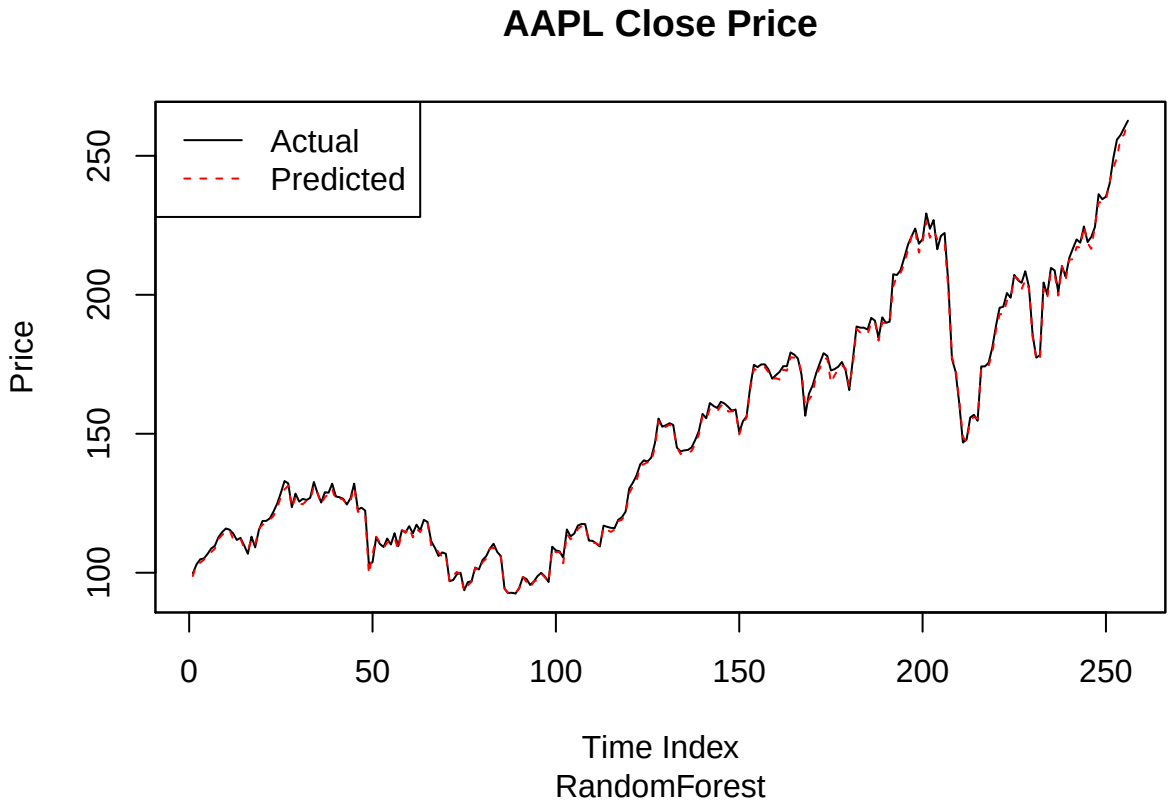
```
# Use random forest to improve the performance of the regression tree
set.seed(1)
for (i in 1:3) {
model_tree_rf <- randomForest(AAPL.Close ~ .,
                              data = aapl_df[train,],
                              mtry = i,
                              importance = T)
cat("Number of variables used:", i, "\n")
calculate_test_RMSE(model_tree_bag, "Random forest regression tree model")
}
```

```
## Number of variables used: 1
## Random forest regression tree model test RMSE: 1.187881
## Number of variables used: 2
## Random forest regression tree model test RMSE: 1.187881
## Number of variables used: 3
## Random forest regression tree model test RMSE: 1.187881
```

```
set.seed(1)
# Use the caret package to fit random forest
for (i in 1:3) {
model_rf <- train(
  fmla_best,
  tuneLength = i,
  data = aapl_df[train, ],
  method = "ranger",
  trControl = trainControl(
    method = "cv",
    number = 10,
    verboseIter = F
  )
)
cat("Number of variables used:", i, "\n")
calculate_test_RMSE(model_rf, "RandomForest model")
}
```

```
## Number of variables used: 1
## RandomForest model test RMSE: 1.156179
## Number of variables used: 2
## RandomForest model test RMSE: 1.117912
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
##
## Number of variables used: 3
## RandomForest model test RMSE: 1.128437
```

```
model_rf <- train(
  fmla_best,
  tuneLength = 2,
  data = aapl_df[train, ],
  method = "ranger",
  trControl = trainControl(
    method = "cv",
    number = 10,
    verboseIter = F
  )
)
plot_model(model_rf, "RandomForest")
```



Random forest has the same perfornce as the bagging method.

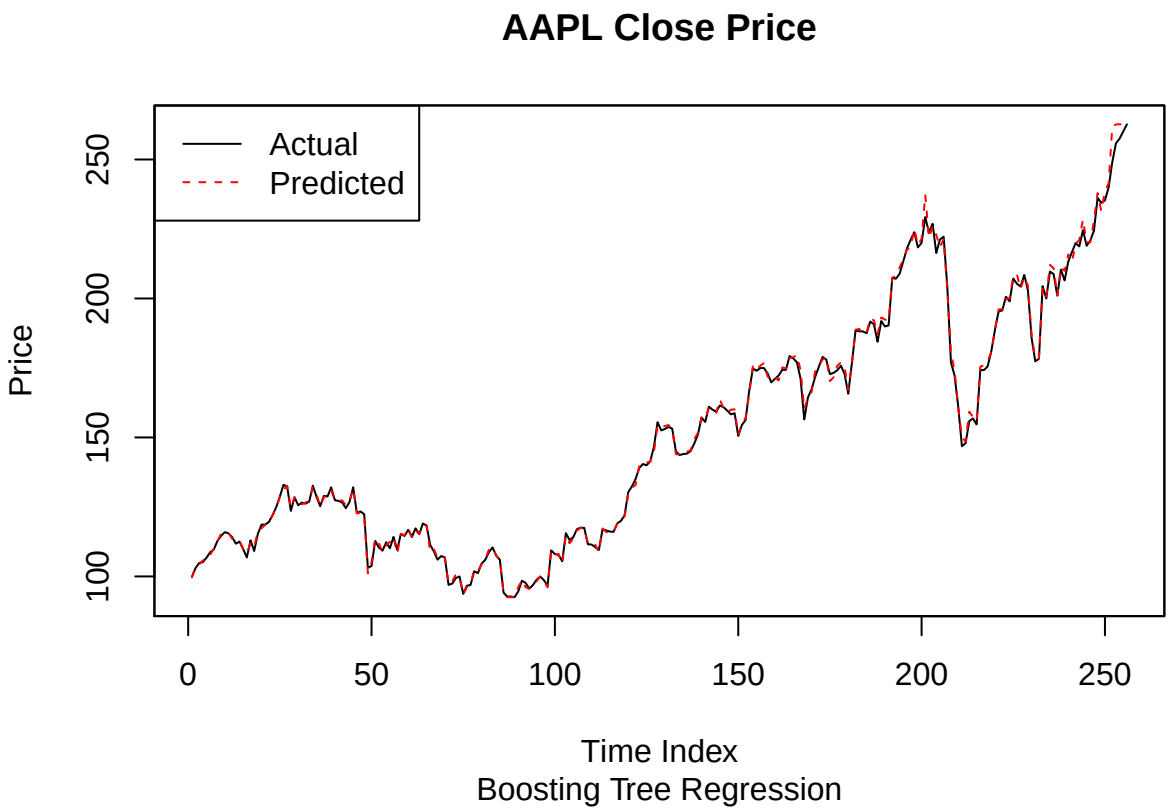
Boosting

```
library(gbm)
set.seed(1)
b <- rep(0, 10)
j = 1
for (i in seq(5000, 50000, by = 5000)) {
# Use boosting to improve the performance of the regression tree performance
model_tree_boost <- gbm(fmla_best,
```

```
                        data = aapl_df[train, ],
                        distribution = "gaussian",
                        n.trees = i,
                        interaction.depth = 4,
                        shrinkage = 0.2,
                        verbose = F)
b[j] <- sqrt(mean((predict(model_tree_boost, newdata=aapl_df[test, ], n.trees = i) - aapl_df[test, 4])^
j = j + 1
}

model_tree_boost <- gbm(fmla_best,
                        data = aapl_df[train, ],
                        distribution = "gaussian",
                        n.trees = 5000,
                        interaction.depth = 4,
                        shrinkage = 0.2,
                        verbose = F)
preds <- predict(model_tree_boost, newdata=aapl_df[test, ], n.trees=5000)
error <- preds - aapl_df[test, 4]
rmse <- sqrt(mean(error^2))
cat("Boosting Tree model test RMSE:", rmse, "\n")
```

```
## Boosting Tree model test RMSE: 1.543132
```

```
    plot(aapl_df[test, 4], type = "l", col = "black", main = "AAPL Close Price", sub = "Boosting Tree R
    par(new = T)
    plot(preds, type = "l", col = 'red', lty = 2, xlab = "", ylab = "", xaxt = "n", yaxt = "n")
    legend("topleft", c("Actual", "Predicted"), col = c("black", "red"), lty = 1:2)
```



AAPL.Low, AAPL.High and AAPL.OPEN are the most important variables, which confirms to the result of cross validation approach. It has the identical performance as bagging and random forest.

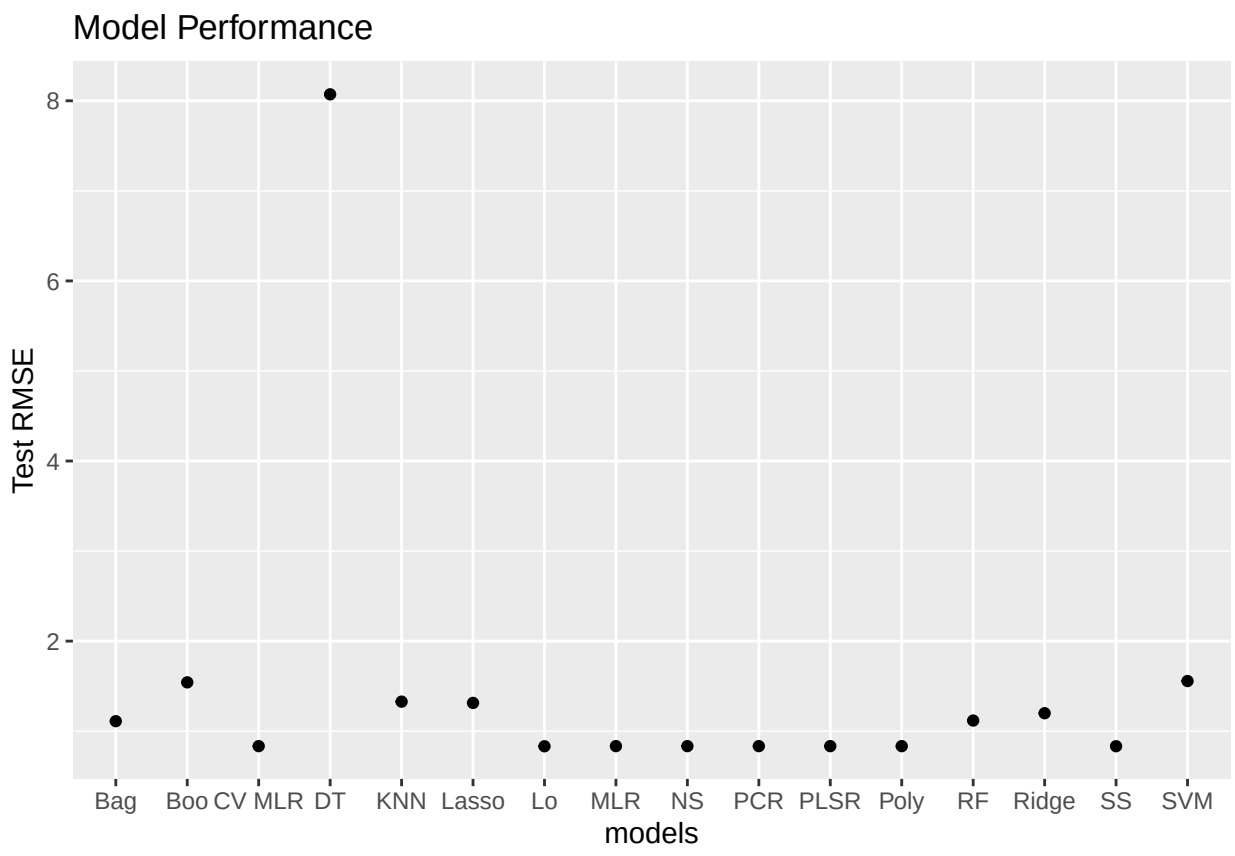
Regression Conclusion

```
models <- c("MLR", "CV MLR", "PCR", "PLSR", "Ridge", "Lasso", "Poly", "NS", "SS", "Lo", "SVM", "KNN", "
model_rmse <- c(0.83377, 0.83377, 0.83377, 0.83377, 1.199273, 1.313807, 0.83377, 0.83377, 0.8327063, 0.8
model_df <- data.frame(models, model_rmse, stringsAsFactors=F)
model_df <- model_df[order(model_rmse), ]
model_df
```

```
##    models model_rmse
## 10     Lo  0.8322157
## 9      SS  0.8327063
## 1     MLR  0.8337700
## 2  CV MLR  0.8337700
## 3     PCR  0.8337700
## 4    PLSR  0.8337700
## 7    Poly  0.8337700
## 8      NS  0.8337700
## 14    Bag  1.1109490
## 15     RF  1.1179120
## 5   Ridge  1.1992730
## 6   Lasso  1.3138070
## 12    KNN  1.3279400
```

```
## 16    Boo  1.5418710
## 11    SVM  1.5565520
## 13     DT  8.0719190
```

```
qplot(x = models, y = model_rmse, main = "Model Performance", ylab = "Test RMSE")
```



Out-of-Sample Test

```
aapl_today <- getSymbols(Symbols = "AAPL", scr = "yahoo", from = "2019-11-20", to = "2019-11-21", auto.a
aapl_today <- data.frame(aapl_today)[, -c(5, 6)]

test_rmse <- function(model, model_name, data) {
    pred <- predict(model, data)
    error <- pred - data$AAPL.Close
    rmse <- sqrt(mean(error^2))
    cat(model_name, "predicted close price:", pred, "\n")
    cat("Error:", abs(aapl_today$AAPL.Close - pred), "\n")
}
cat("Today's close price: ", aapl_today$AAPL.Close, "\n")
```

```
## Today's close price:  263.19
```

```r
test_rmse(model_lo, "Local Regression", aapl_today)
```

```
## Local Regression predicted close price: 262.2029
## Error: 0.9871324
```

```r
test_rmse(model_s, "Smoothing Splines", aapl_today)
```

```
## Smoothing Splines predicted close price: 262.3672
## Error: 0.8228347
```

```r
test_rmse(model_ns, "Natural Smoothing", aapl_today)
```

```
## Natural Smoothing predicted close price: 262.0859
## Error: 1.104081
```

```r
test_rmse(model_lm_best, "MLR", aapl_today)
```

```
## MLR predicted close price: 262.0859
## Error: 1.104081
```

Classification

```r
# Import libraries
library(zoo)
library(xts)
library(quantmod)
library(caret)
library(MASS)
library(class)
library(pROC)
library(e1071)
library(klaR)
```

Features

```r
# Import the apple stock from yahoo & save it as xts object
aapl_xts <- getSymbols(Symbols = "AAPL", scr = "yahoo", from = "2014-09-30", to = "2019-10-02", auto.as
lags=Lag(dailyReturn(aapl_xts),k=1:10)
direction = ifelse((dailyReturn(aapl_xts) > 0),'UP','DOWN')
volume = Vo(aapl_xts)
aapl_df <- cbind.data.frame(volume, lags, direction)
colnames(aapl_df)[12] = "Direction";
aapl_df = na.omit(aapl_df);
```

Test/Train split

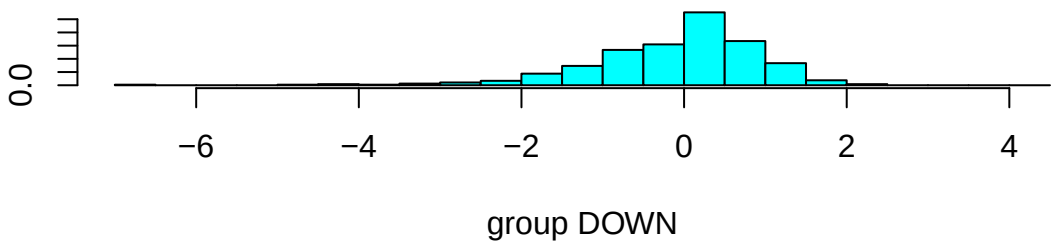
```r
set.seed(1)
train_idx <- createDataPartition(aapl_df$Direction, p = 0.8, list = F, times = 1)
aapl_df_train <- aapl_df[train_idx, ]
aapl_df_test <- aapl_df[-train_idx, ]
```

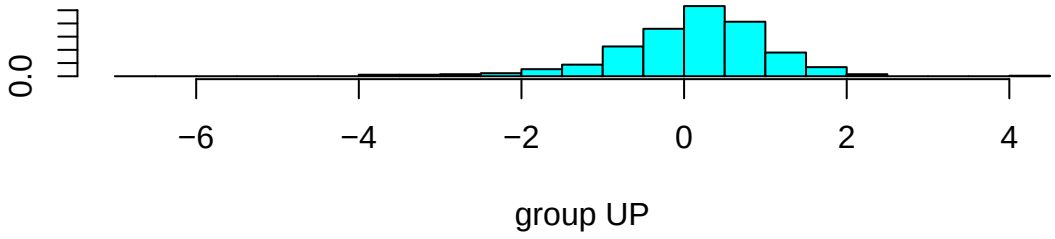
Linear Discriminant Analysis

```r
lda.fit = lda(Direction ~ ., aapl_df_train)
lda.fit
```

```
## Call:
## lda(Direction ~ ., data = aapl_df_train)
##
## Prior probabilities of groups:
##      DOWN        UP
## 0.4775225 0.5224775
##
## Group means:
##      AAPL.Volume        Lag.1        Lag.2        Lag.3        Lag.4
## DOWN    38453445 0.0005259238 0.0005873104 0.0004507633 0.0003053013
## UP      35689893 0.0004779619 0.0006591746 0.0010682960 0.0010595195
##             Lag.5        Lag.6        Lag.7        Lag.8        Lag.9
## DOWN 0.0003417945 0.0009606555 0.0005155127 0.0013671320 0.0005528054
## UP   0.0007861114 0.0006571136 0.0015104157 0.0003021882 0.0013740524
##            Lag.10
## DOWN 0.0005073500
## UP   0.0003053356
##
## Coefficients of linear discriminants:
##                        LD1
## AAPL.Volume -4.205561e-08
## Lag.1       -3.517299e+00
## Lag.2       -4.077665e+00
## Lag.3        1.207520e+01
## Lag.4        9.851526e+00
## Lag.5        7.874757e+00
## Lag.6       -8.712065e+00
## Lag.7        1.840436e+01
## Lag.8       -2.543457e+01
## Lag.9        1.501267e+01
## Lag.10      -7.914956e+00
```

```r
plot(lda.fit)
```






```
#Because they are both centered at 0 and spread is similar, suggests that there is not much to distingu
```

```
lda.pred = predict(lda.fit, aapl_df_test)
lda.class=lda.pred$class
lda.table=table(lda.class,aapl_df_test$Direction,  dnn = c("Predicted", "Actual"))
lda.table
```

```
##           Actual
## Predicted DOWN  UP
##      DOWN   39  29
##      UP     80 101
```

The series are trend and peroidicity.

```
lda.acc=mean(lda.class==aapl_df_test$Direction)
lda.acc
```

```
## [1] 0.562249
```

Quadratic Discriminant Analysis

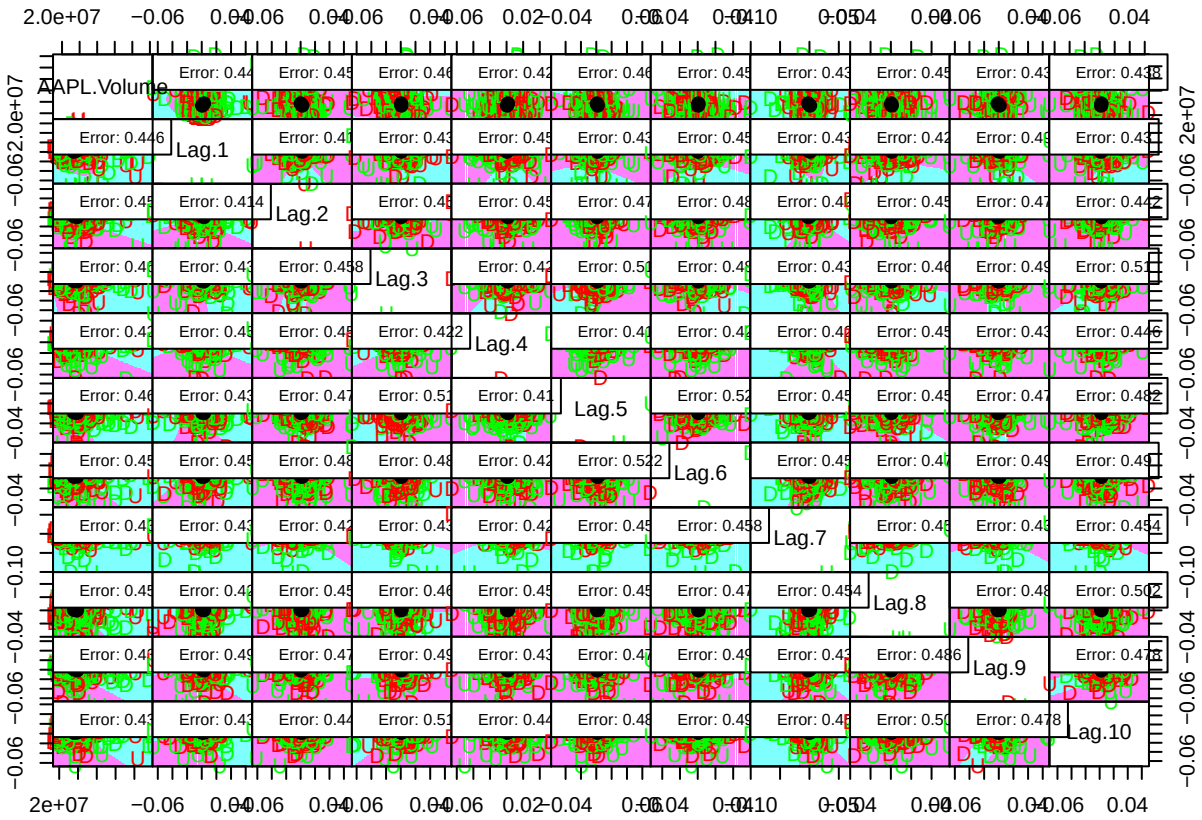
```
qda.fit = qda(Direction ~ ., aapl_df_train)
qda.pred = predict(qda.fit, aapl_df_test)
qda.class=qda.pred$class
qda.table=table(qda.class,aapl_df_test$Direction,  dnn = c("Predicted", "Actual"))
qda.table
```

```
##           Actual
## Predicted DOWN UP
##      DOWN   56 53
##      UP     63 77
```

```
qda.acc=mean(qda.class==aapl_df_test$Direction)
qda.acc
```

```
## [1] 0.5341365
```

```
partimat(Direction~ ., data = aapl_df_test, method = "qda", plot.matrix = TRUE, col.correct='green', col
```



KNN

```
getknnerr <-function(n, traindata, testdata, trainresp, testresp) {

  ncount=0
  err=0
```

```
  errcount=0
  tablen=0
  knnsamp <- knn(traindata, testdata, trainresp, k=1)
  tablesamp <- table(knnsamp, testresp)
  print(tablesamp)
  err=(tablesamp[1,1]+tablesamp[2,2])/(nrow(testdata))
  cat("Base error at k=1 :",err)
  for(i in 2:n){
    knnsamp2 <- knn(traindata, testdata, trainresp, k=i)
    tablesamp2 <- table(knnsamp2, testresp)
    errcheck=(tablesamp2[1,1]+tablesamp2[2,2])/(nrow(testdata))
    if(errcheck>err){
      ncount=i
      err=errcheck
      tablen=tablesamp2
      }
  }
  cat("\nfinal accuracy score:",err)
  cat("\nachieved at k=",ncount)
  tablen
  return(ncount)
}


X_train = aapl_df_train
X_test = aapl_df_test
Y_test = aapl_df_test$Direction
Y_train = aapl_df_train$Direction
X_train$Direction = NULL
X_test$Direction = NULL

set.seed(321)

knnreturn <- knn(X_train, X_test, Y_train, k=6)
tablesamp <- table(knnreturn, Y_test)
tablesamp
```

```
##          Y_test
## knnreturn DOWN UP
##      DOWN   55 56
##      UP     64 74
```

```
knnreturn <- sapply(knnreturn, as.numeric)
err=(tablesamp[1,1]+tablesamp[2,2])/(nrow(X_test))
err
```

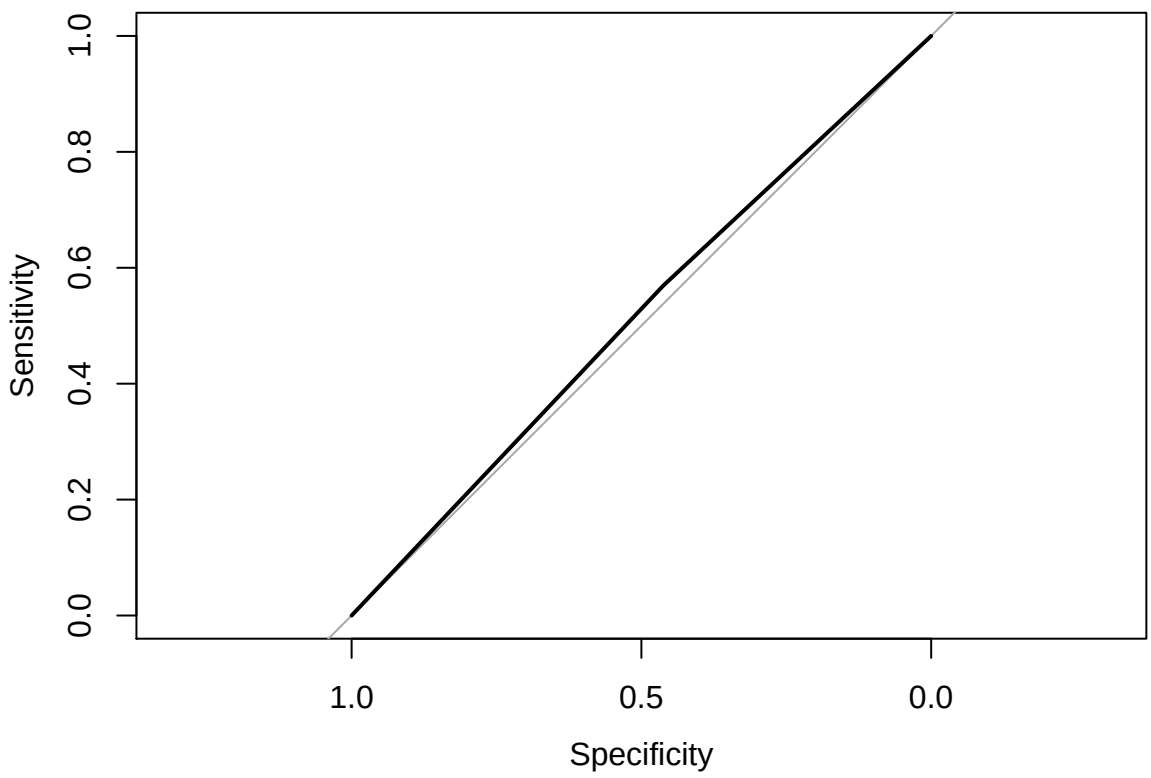
```
## [1] 0.5180723
```

```
#Plot roc curve
rocg <- roc(Y_test,knnreturn)
```

```
## Setting levels: control = DOWN, case = UP
```

```
## Setting direction: controls < cases
```

```
plot(rocg)
```




```
aucsc <- auc(Y_test,knnreturn)
```

```
## Setting levels: control = DOWN, case = UP
## Setting direction: controls < cases
```

```
aucsc
```

```
## Area under the curve: 0.5157
```

SVM

```
tune.out=tune(svm, Direction~., data=aapl_df_train, kernel="linear",ranges=list(cost=c(0.01, 0.1,.5, 1,
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
```

```
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.4795545
##
## - Detailed performance results:
##    cost     error dispersion
## 1  0.01 0.4795545 0.02805786
## 2  0.10 0.4835743 0.03577105
## 3  0.50 0.4855743 0.03731718
## 4  1.00 0.4845743 0.03610883
## 5 10.00 0.4845743 0.03610883
```

```
svmfit1=svm(Direction~., data=aapl_df_train, kernel = 'linear', cost=.01)

postResample(predict(svmfit1, newdata = X_test), Y_test) #best performance
```

```
##   Accuracy      Kappa
## 0.54216867 0.04520686
```