Predicting Movements in the Quantity of Social Security Retirees

P. Christopher J. Daigle

7 December 2018

Table of Contents

1. Topic

"For the great majority of Americans, the most important form of household wealth is the anticipated social security retirement benefits." (Feldstein, 1974). Predicting movements in social security may be beneficial in determining expectations of the funding mechanism.

Through the collection of data related to the US Financial System, I evaluate changes in the number of people receiving social security retirement benefits. Through utilizing different econometric and machine learning methodologies, I am able to successfully predict at different levels of accuracy if the number of social security recipients will rise or fall.

2. Data

The data is of a time-series nature covering a period from 1985 to 2018 with observations at each month. The level is national; regional social security information is not evaluated nor acknowledged. Financial values are inflated to the latest available CPI information, September 2018.

Collection was completed in a few manners. First, I developed a web-scraper in Python with the library BeautifulSoup that parsed information from Yahoo Finance on the Dow Jones Industrial Average and the S&P500. This web-scraper was also applied to the Federal Reserve Bank of St. Louis website for CPI and the Federal Funds Rate. The web-scraper I built did not yield consistent outputs (e.g. two runs of the web-scraper yielded two different results or crashed altogether), so I chose to manually download these from the site.

Utilizing Python with the NumPy and Pandas library along with some SQL syntax, I imported, transformed, and combined the data sets to export a single dataset. Transformations were necessary as the joining key, date, was not standard from each source. At this point, the data set consisted of 405 observations and 22 variables ($T = 405$, $P = 22$).

From here, I cleaned the data in R and created a number of synthetic variables ranging from an inflation factor, real dollar values from nominal dollar values, difference values, percent change values, and indicators of positive changes in these values. Variables that were created as the result of differences in their values contained null values for the first time period. Because of this, the data lost the first time period to account for null values (i.e. I dropped the first time period because it was full of NAs). These transformations and variable creations resulted in 100 new variables and the loss of one time period (T = 404, P = 122).

The data began with 405 observations and reduced to 404 relevant observations because of the type of response, changes. The time dimension is still relatively long and so the loss of this one observation appears trivial.

As this data was of a time-series nature, it was necessary to ensure only stationary values were utilized for prediction. I observed different measures of seasonal decomposition and I applied the Augmented Dickey-Fuller Test which tests for stationarity. From that test, I removed all variables that did not appear stationary. This resulted in a shrinking of the data set by 64 variables (T = 404, P = 58).

3. Variables

The response I measured is a directional change in the quantity of social security recipients as increasing or decreasing. For this, I begin with a binary variable with the positive class (i.e. y = 1) indicating a positive difference in the quantity of recipients from one month to the next and the negative class (i.e. y = 0) indicating a negative movement in the quantity of recipients from one month to the next. From this, I used a series of stepwise sub-setting to select appropriate factors for prediction. I found forward selection to yield the best predictors. These predictors are as follows:

PosΔDJIopen	posΔRealSPadjClose
PosΔDJIclose	%ΔRealDJIadjClose
posΔDJIadjClose	%ΔRealSPadjClose
PosΔRealSPopen	posΔRealAverageFemaleSSRetiredPay
%ΔRealDJIhigh	posΔAverageFemalSSRetiredPay

This is a total of ten factors from the 58 I found by ensuring the data was stationary. It is worth noting that many of these variables are closely related so much so that a few are transformations of one another. While this is not a good method for establishing a causal model, my goal in this examination is to evaluate predictors and not perform causal inference.

4. Summary Statistics

Contained is a table of summary statistics of relevant variables:

	posΔDJIopen	**posΔDJIclose**	**posΔDJIadjClose**	**posΔRealSPopen**	**posΔaverageFemaleSSRetiredPay**	**posΔRealaverageFemaleSSRetiredPay**	**posΔRealSPadjClose**	**%ΔRealDJIhigh**	**%ΔRealDJIadjClose**	**%ΔRealSPadjClose**
Obs	404	404	404	404	404	404	404	404	404	404
Min	0.000	0.000	0.000	0.000	0.000	0.000	0.000	-0.241	-0.234	-0.220
Max	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.117	0.132	0.126
Med	1.000	0.000	1.000	0.000	1.000	1.000	0.000	0.006	0.008	0.009
Avg	0.639	0.490	0.606	0.418	0.606	0.611	0.468	0.006	0.006	0.006
Var	0.231	0.251	0.239	0.244	0.239	0.238	0.250	0.001	0.002	0.002
SD	0.481	0.501	0.489	0.494	0.489	0.488	0.500	0.032	0.042	0.042

5. Model / Methodology

The basic model of analysis is a logistic regression where the influencing factors are selected via forward stepwise sub-setting where the factors for the model yield the smallest sum of squared residuals (RSS) (ISLR, 2017).

The initial model is as such:

$$\textbf{(1)}\quad \boldsymbol{logit(Pos\Delta SSRetired) = pos\Delta DJIopen + pos\Delta DJIclose + pos\Delta DJIadjClose + pos\Delta AverageFemaleSSRetiredPay + pos\Delta RealSPopen + \%\Delta RealDJIhigh + pos\Delta RealSPadjClose + pos\Delta RealAverageFemaleSSRetiredPay + \%\Delta RealDJIadjClose + \%\Delta RealSPadjClose + \varepsilon}$$

This model, utilizing logistic regression, yields a prediction accuracy of about 93% when trained on about 70% of the data. From this model, I tested more models for prediction accuracy by removing factors at each step related to their statistical significance.

By that method, I was able to remove 7 factors. The restricted model then appeared as:

(2) $$\boldsymbol{logit(pos\Delta SSretired) = pos\Delta DJIadjClose + pos\Delta RealSPopen + pos\Delta RealSPadjClose + } \ \mathcal{E}$$

At the same split for the training and testing of the model as the unrestricted model, this model accurately predicts if the total number of social security recipients will rise at the 95% level and fall at the 91.5% level, with an overall model accuracy of 93%. More specifically, the original model of 10 predictors cut down to three predictors yield the same level of accuracy.

Notably, the logit model was selected for its ability in classification, but the probit model was also tested as the underlying distribution of the positive change in total social security recipients was about normal.

(3) $$\boldsymbol{probit(pos\Delta SSretired) = pos\Delta DJIadjClose + pos\Delta RealSPopen + pos\Delta RealSPadjClose +} \ \mathcal{E}$$

This model yielded the same prediction accuracy as the logit model on the same splits of the training and testing set.

For a more robust analysis of the model's fit than simply the prediction outcomes, I tested the restricted model (3 factors) and unrestricted model (10 factors) with a likelihood-ratio test.

(4) $$\begin{aligned} \boldsymbol{H_0: \theta = \theta_0} \\ \boldsymbol{H_1: \theta = \theta_1} \end{aligned}$$

(5) $$\boldsymbol{\Lambda(x) = \frac{\mathcal{L}(\theta_0|x)}{\mathcal{L}(\theta_1|x)} = \frac{\mathcal{L}(\theta_0|x)}{\sup\{\mathcal{L}(\theta_1|x): \theta \in \{\theta_o, \theta_1\}\}}}$$

(6) $$\boldsymbol{P - value = 0.133 > 0.05}$$

The findings are that the null hypothesis, i.e. the restricted model fits the data better than the unrestricted model, fails to be rejected. Thus, the restricted model of only three factors can be employed with at least as good of prediction power (i.e. the model with less factors is just as good as the one with more).

6. Concluding Remarks

Economics generally focuses on exogeneity. Discerning the causal effect of factors on the response variable is important for policy prescription. As data continues to be generated at a rapid rate, predicting the outcome of an event may become a simpler task. A tradeoff seems necessary between demanding causality and demanding accuracy of determining the likelihood of occurrence.

If the field of economics holds predictive models (forecasting) to the same standard as inferential models (causal relationships), the field as a whole may lose the ability to take advantage of quickly growing information and the field will limit its tools.

While directional changes in the quantity of social security retirees has been shown to be predictable from the data and models employed here, I make no claim to the causal relationship. With only a few and easily accessible pieces of data, it is shown that a signal can be assessed in determining the need to prepare for changes in social security by the policy maker. The power of this tool is not in its ability to explain why people retire, but to predict if people will.

As continued funding for social security is in question, determining whether the number of recipients will rise, or fall, can help to inform the transmission of funds from the system. Of value may also be to supplement this prediction with predicting the number of labor force participants as returns to labor are taxed to fund the social security system.

7. References

Feldstein, Martin (1974) Social Security, Induced Retirement, and Aggregate Capital Accumulation. *Journal of Political Economy, 82-5*.

FRED, Federal Reserve Bank of St. Louis (2018). *Consumer Price Index for All Urban Consumers: All Items,* https://fred.stlouisfed.org/series/CPIAUCSL/ accessed October 10, 2018.

FRED, Federal Reserve Bank of St. Louis (2018). *Effective Federal Funds Rate,* https://fred.stlouisfed.org/series/FEDFUNDS accessed October 10, 2018.

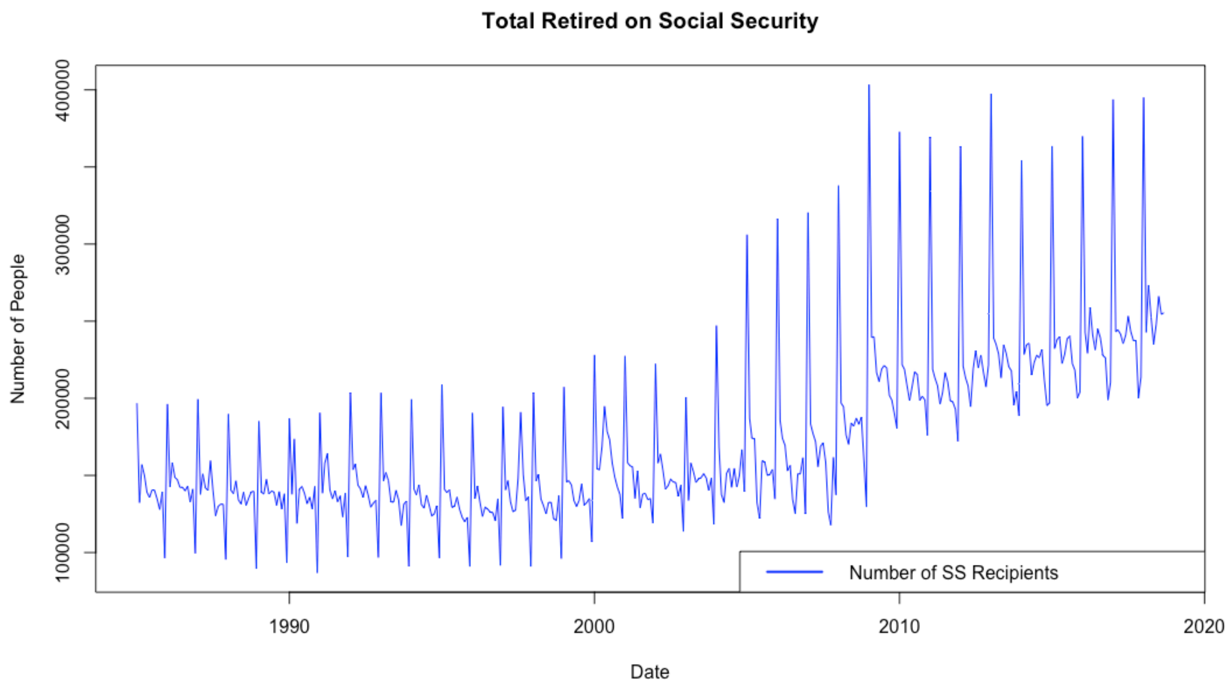
James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). An Introduction to Statistical Learning with Applications in R (ISLR). Springer New York.

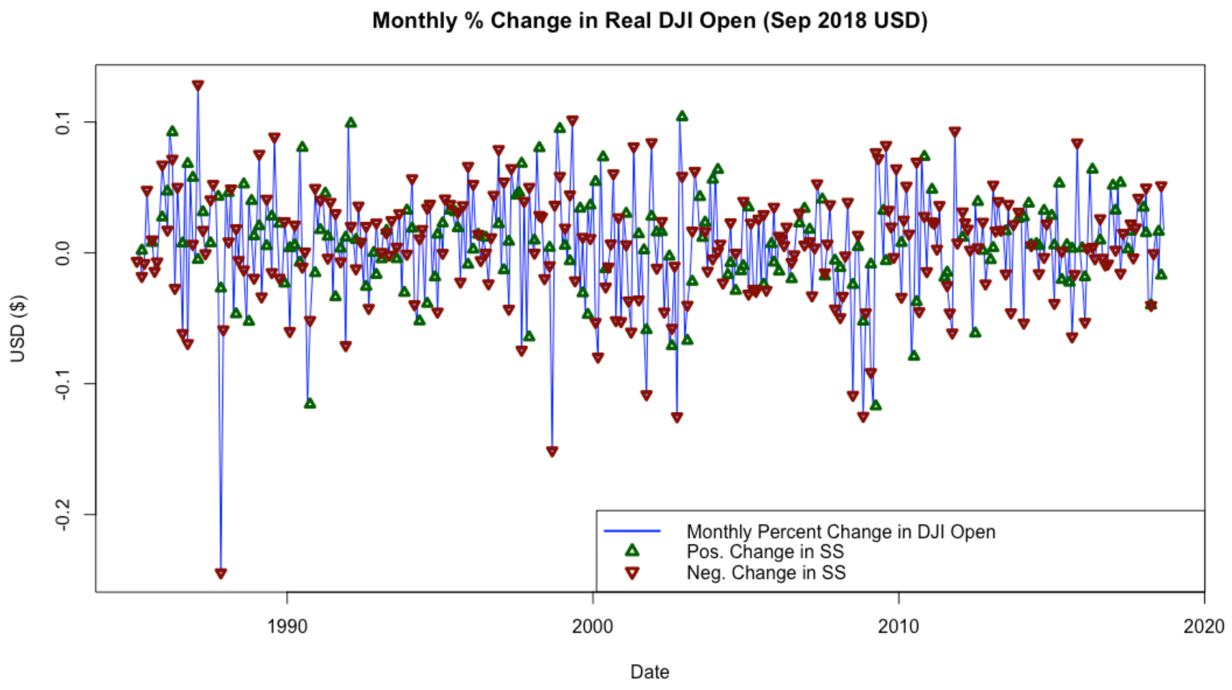
Social Security Administration (2018). *Benefits Awarded by Type of Beneficiary,* https://www.ssa.gov/OACT/ProgData/awards.html accessed October 10, 2018.

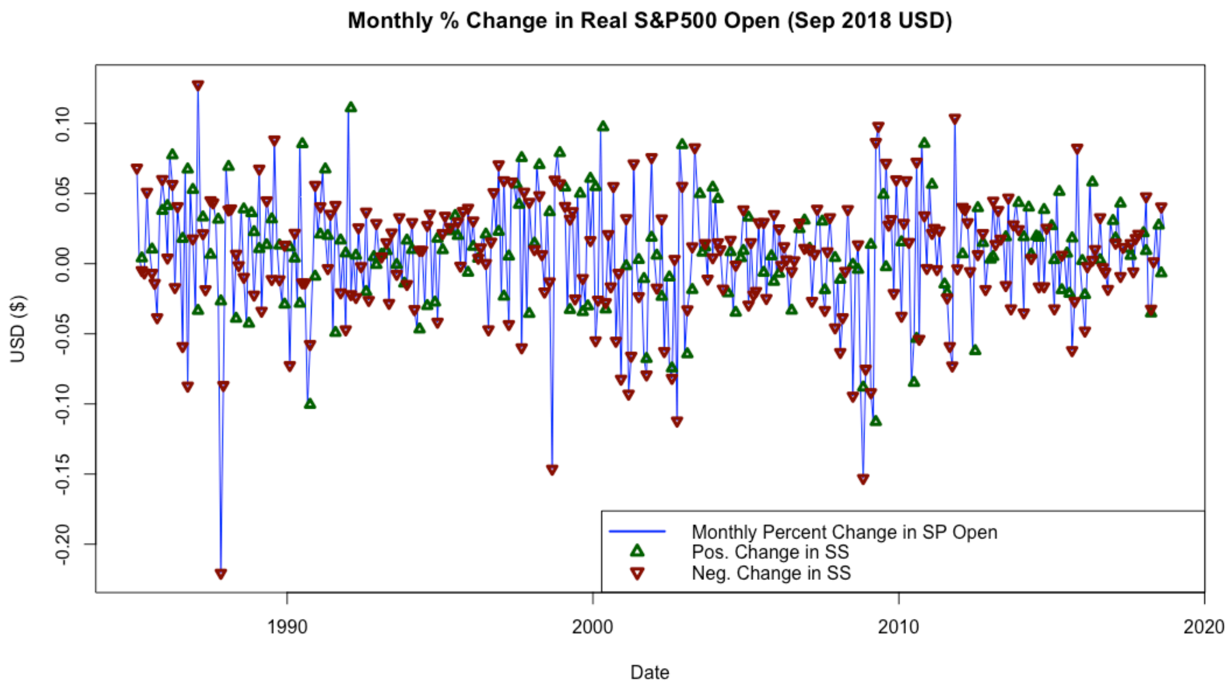
Yahoo Finance (2018). *^GSPC Historical Data,* https://finance.yahoo.com/quote/%5EGSPC/history?period1=-630957600&period2=1542520800&interval=1mo&filter=history&frequency=1mo accessed October 10, 2018.

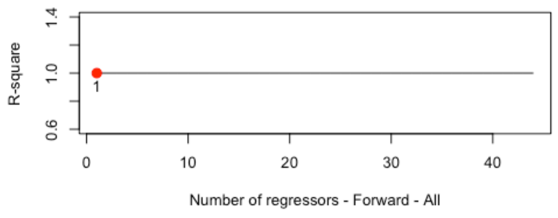
Yahoo Finance (2018). *^DJI Historical Data,* https://finance.yahoo.com/quote/%5EDJI/history?period1=475826400&period2=1542520800&interval=1mo&filter=history&frequency=1mo accessed October 10, 2018.

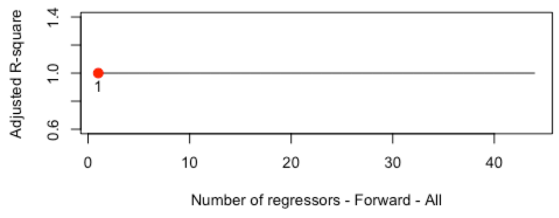
A. Appendix: Figures

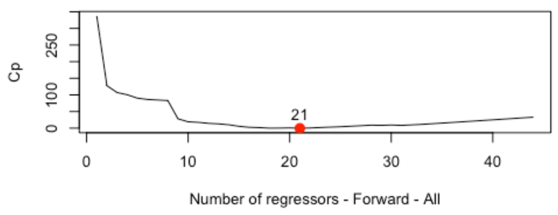


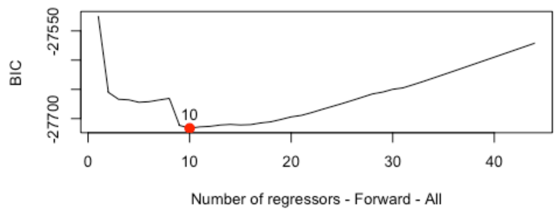



Monthly % Change in Real S&P500 Open (Sep 2018 USD)
USD ($)
Date
Monthly Percent Change in SP Open
Pos. Change in SS
Neg. Change in SS
1990
2000
2010
2020


R-square
Number of regressors - Forward - All


Adjusted R-square
Number of regressors - Forward - All


Cp
21
Number of regressors - Forward - All


BIC
10
Number of regressors - Forward - All

B. Python Code

```
#!/usr/bin/env python
# coding: utf-8
# # Data Assembly
# In[1]:
import pandas as pd
import numpy as np
# In[2]:
DJI = pd.read_csv(
'~/Git/MachineLearningAndBig
DataWithR/Data/^DJIMonthly.c
sv'
    , sep = ',')
SP500 = pd.read_csv(
'~/Git/MachineLearningAndBig
DataWithR/Data/^GSPCMonthly.
csv'
    , sep = ',')
FedFunds = pd.read_csv(
'~/Git/MachineLearningAndBig
DataWithR/Data/FEDFUNDS.csv'
    , sep = ',')
SS = pd.read_csv(
'~/Git/MachineLearningAndBig
DataWithR/Data/SSRetired.csv
'
    , sep = ',')
CPI = pd.read_csv(
'~/Git/MachineLearningAndBig
DataWithR/Data/CPIAUCSL.csv'
    , sep = ',')
# ### DJI
#
https://finance.yahoo.com/qu
ote/%5EDJI/history?period1=4
75826400&period2=1542520800&
interval=1mo&filter=history&
frequency=1mo
# In[3]:
#DJI.head()
DJI.columns = ['date',
'DJIopen', 'DJIhigh',
'DJIlow', 'DJIclose',
'DJIadjClose', 'DJIvolume']
#DJI.head()
# ### S&P 500
#
https://finance.yahoo.com/qu
ote/%5EGSPC/history?period1=
-
630957600&period2=1542520800
&interval=1mo&filter=history
&frequency=1mo
# In[4]:
```

```
#SP500.head()
SP500.columns = ['date',
'SPopen', 'SPhigh', 'SPlow',
'SPclose', 'SPadjClose',
'SPvolume']
#SP500.head()
df = pd.merge(DJI,SP500, how
= 'inner', on = 'date')
#df.head()
# ### Federal funds rate
#
https://fred.stlouisfed.org/
series/FEDFUNDS
# In[5]:
#FedFunds.head()
FedFunds.columns = ['date',
'fedFundRate']
#FedFunds.head()
df = pd.merge(df,FedFunds,
how = 'inner', on = 'date')
#df.head()
# ### Retired social
security filings
#
https://www.ssa.gov/OACT/Pro
gData/awards.html
# In[6]:
#SS.head()
SS.columns = ['date',
'totalSSRetired',
'averageSSRetiredPay',
'totalMaleSSRetired',
'averageMaleSSRetiredPay',
'totalFemaleSSRetired',
'averageFemaleSSRetiredPay',
'DROPME']
SS.drop('DROPME', axis = 1,
inplace=True)
#SS.head()
adjDate =
SS['date'].str.split("-", n
= 1, expand = True)
#adjDate.head()
adjDate['month'] =
np.where(adjDate[0] ==
'Jan', '-01-01'

, np.where(adjDate[0] ==
'Feb', '-02-01'
       , np.where(adjDate[0]
== 'Mar', '-03-01'
,np.where(adjDate[0] ==
'Apr', '-04-01'
```

```
,np.where(adjDate[0] ==
'May', '-05-01'
,np.where(adjDate[0] ==
'Jun', '-06-01'
,np.where(adjDate[0] ==
'Jul', '-07-01'
,np.where(adjDate[0] ==
'Aug', '-08-01'
,np.where(adjDate[0] ==
'Sep', '-09-01'
,np.where(adjDate[0] ==
'Oct', '-10-01'
,np.where(adjDate[0] ==
'Nov', '-11-01'
,np.where(adjDate[0] ==
'Dec', '-12-01'
, 0))))))))))))
adjDate['year'] =
np.where(adjDate[1].astype(i
nt) <= 18, '20'
,np.where(adjDate[1].astype(
int) > 18, '19'

, 0))
adjDate['combined'] =
adjDate['year'] + adjDate[1]
+ adjDate['month']
#adjDate.head()
SS['date'] =
adjDate['combined']
#SS.head()
df = pd.merge(df,SS, how =
'inner', on = 'date')
#df.head()
# ### CPI - Consumer Price
Index for All Urban
Consumers: All Items
#
https://fred.stlouisfed.org/
series/CPIAUCSL/
# In[7]:
#CPI.head()
CPI.columns = ['date',
'cpi']
df = pd.merge(df,CPI, how =
'inner', on = 'date')
#df.head()
# ## Write to file
# In[8]:
df.to_csv(path_or_buf =
'~/Git/MachineLearningAndBig
DataWithR/Data/assembled.csv
', sep=',
```

C. R Markdown

DaiglePredictionofSocialSecurity.R

daiglechris

Sun Dec 9 13:45:23 2018

Chris Daigle Prediction of Social Security Awards

```
# Prepare workspace ####
rm(list = ls())
library(tseries)
library(quantmod)
```

```
## Loading required package: xts

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: TTR

## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:xts':
##
##     first, last
```

```
library(leaps)
library(plm)
```

```
## Loading required package: Formula

##
## Attaching package: 'plm'

## The following object is masked from 'package:data.table':
##
##     between
```

```
library(class)
library(lmtest)
library(caret)
```

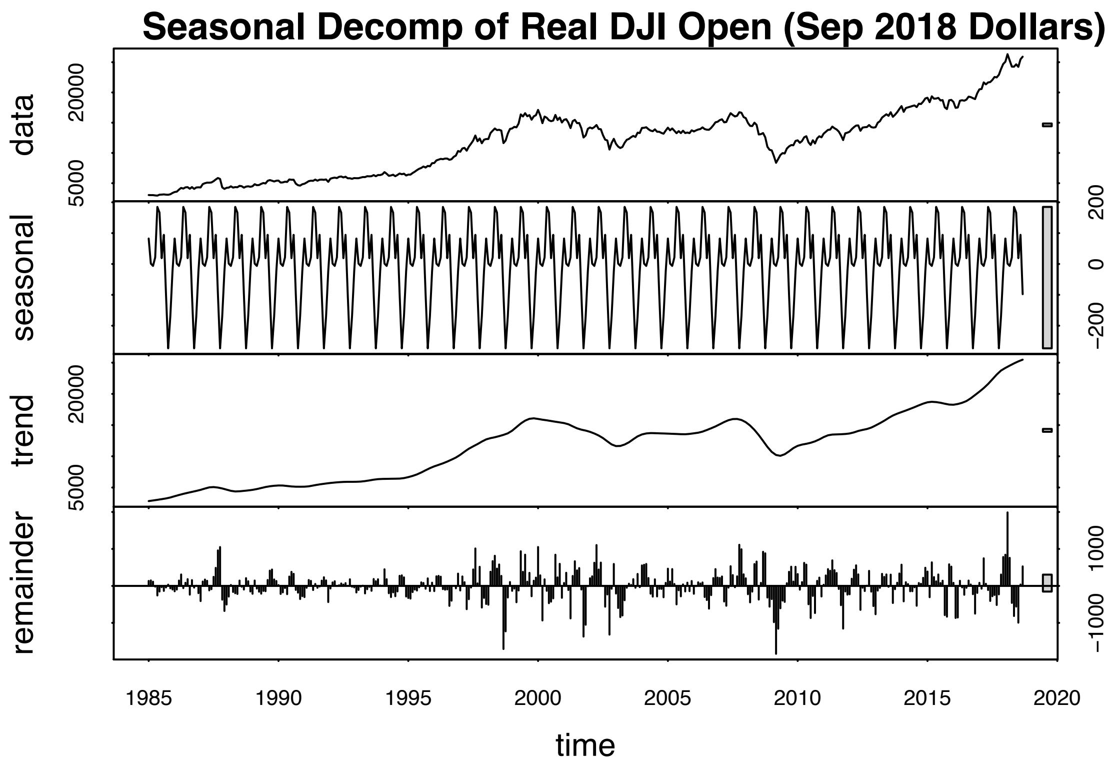
```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
library(knitr)
library(pastecs)
```

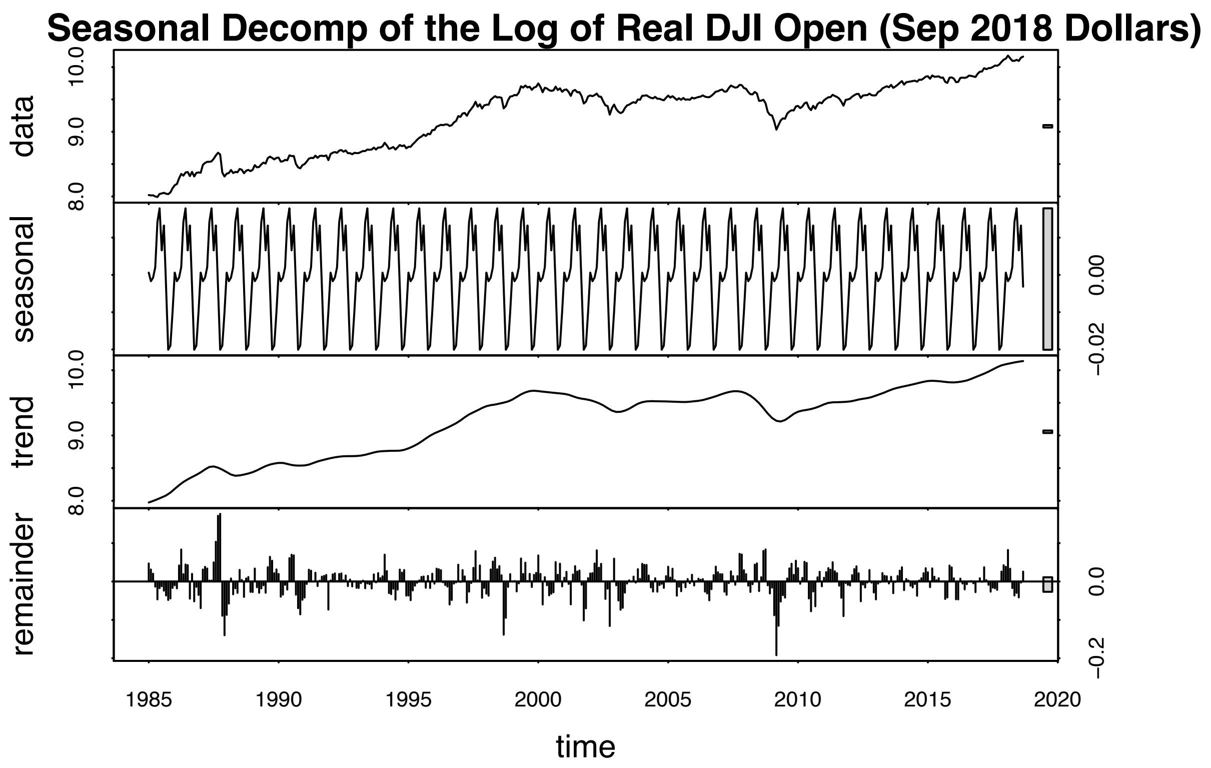
```
##
## Attaching package: 'pastecs'

## The following objects are masked from 'package:data.table':
##
##     first, last

## The following objects are masked from 'package:xts':
##
##     first, last
```

```r
setwd('~/Git/MachineLearningAndBigDataWithR/Data')
dataName <- 'assembled.csv'
df <- read.csv(dataName, stringsAsFactors = FALSE)
# Summarize and clean data ####
# head(df)
df <- df[-1]
# head(df)
# str(df)

# Variable Manipulation ####
# Set dates
df$date <- as.Date(df$date, "%Y-%m-%d")
# Functions to clean data #
spaceless <- function(x) {
  x <- gsub(" ", ".", x)
  x
}

commaless <- function(x) {
  x <- gsub(",", "", x)
  x
}

dollarless <- function(x) {
  x <- gsub("\\$", "", x)
  x
}

# Loops to apply functions #
for (i in 15:20) {
  df[, i] <- commaless(df[, i])
}
for (i in 15:20) {
  df[, i] <- dollarless(df[, i])
}

# Loop to transform variable types #
for (i in 15:20) {
  df[, i] <- as.numeric(df[, i])
}
# Names with Index ####
# 1 date                        : Date
# 2 DJIopen                     : num
# 3 DJIhigh                     : num
# 4 DJIlow                    : num
# 5 DJIclose                  : num
# 6 DJIadjClose               : num
# 7 DJIvolume                 : num
# 8 SPopen                    : num
# 9 SPhigh                    : num
# 10 SPlow                     : num
# 11 SPclose                   : num
# 12 SPadjClose                : num
# 13 SPvolume                  : num
# 14 fedFundRate               : num
# 15 totalSSRetired            : num
# 16 averageSSRetiredPay       : num
# 17 totalMaleSSRetired        : num
# 18 averageMaleSSRetiredPay   : num
# 19 totalFemaleSSRetired      : num
# 20 averageFemaleSSRetiredPay: num
# 21 cpi                       : num
#
# Order Change #
df <- df[, c(1, 15, 17, 19, 21, 14, 7, 13, 2:6, 8:12, 16, 18, 20)]
# 1 date                      : Date
# 2 totalSSRetired            : num
# 3 totalMaleSSRetired        : num
# 4 totalFemaleSSRetired      : num
# 5 cpi                       : num
# 6 fedFundRate               : num
# 7 DJIvolume                 : num
# 8 SPvolume                  : num
# 9 DJIopen                   : num
# 10 DJIhigh                   : num
# 11 DJIlow                    : num
# 12 DJIclose                  : num
# 13 DJIadjClose               : num
# 14 SPopen                    : num
# 15 SPhigh                    : num
# 16 SPlow                     : num
# 17 SPclose                   : num
# 18 SPadjClose                : num
# 19 averageSSRetiredPay       : num
# 20 averageMaleSSRetiredPay   : num
# 21 averageFemaleSSRetiredPay: num
#
# Variable Creation ####
# CPI Inflator
latestDate <- tail(df$date, n = 1)

baseCpi <- df$cpi[df$date == latestDate]
df$inflator <- baseCpi / df$cpi

df <- df[, c(1:6, 22, 7:21)]

realNames <-
```

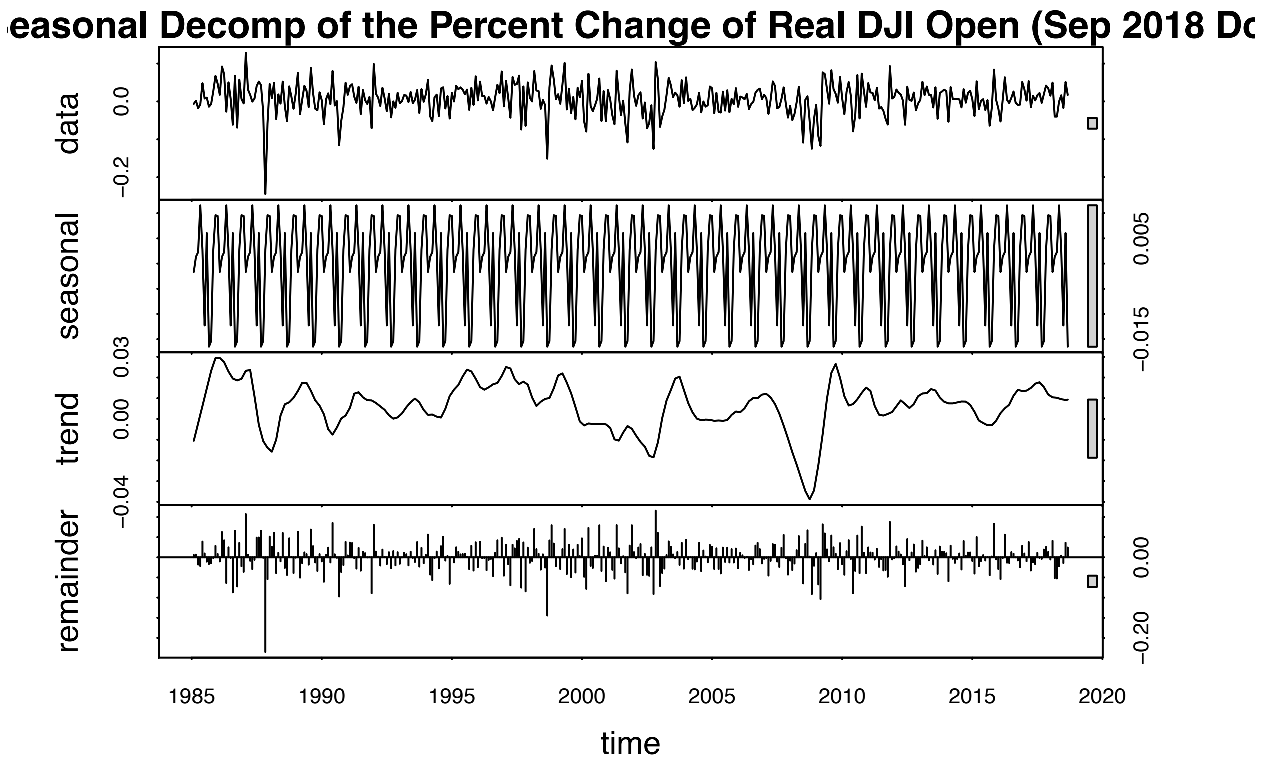
```
  paste('real',
        colnames(df[, 10:22]),
        sep = "")

df[, realNames] <- df$inflator * df[10:22]

# Differences #
diffNames <-
  paste('diff',
        c(colnames(df[10:22]),
          paste('Real',
                colnames(df[10:22]),
                sep = "")),
        sep = "")
df[, diffNames] <- rep(NA, nrow(df))
for (i in 36:61) {
  df[, i][2:nrow(df)] <- diff(df[, i - 26], lag = 1)
}
diffTargetNames <-
  paste('diff',
        c(colnames(df[2:4])),
        sep = "")
df[, diffTargetNames] <- rep(NA, nrow(df))
for (i in 62:64) {
  df[, i][2:nrow(df)] <- diff(df[, i - 60], lag = 1)
}

# Positive Indicator #
posNames <-
  paste('pos',
        c(colnames(df[10:22]),
          paste('Real',
                colnames(df[10:22]),
                sep = "")),
        sep = "")
df[, posNames] <- rep(0, nrow(df))
for (i in 65:90) {
  df[, i][df[, i - 20] > 0] <- 1
}

posTargetNames <-
  paste('pos',
        c(colnames(df[2:4])),
        sep = "")
df[, posTargetNames] <- rep(0, nrow(df))
for (i in 91:93) {
  df[, i][df[, i - 29] > 0] <- 1
}

# Percent Changes #
percChangeNames <-
  paste('percChange',
        c(colnames(df[10:22]),
          paste('Real', colnames(df[10:22]), sep = "")),
        sep = "")
df[, percChangeNames] <- rep(NA, nrow(df))

for (i in 94:119) {
  df[, i] <- Delt(df[, i - 84])
}
for (i in 94:119) {
  df[, i] <- as.numeric(df[, i])
}
percChangeTargetNames <-
  paste('percChange',
        c(colnames(df[2:4])),
        sep = "")
df[, percChangeTargetNames] <- rep(NA, nrow(df))
for (i in 120:122) {
  df[, i] <- Delt(df[, i - 118])
}
for (i in 120:122) {
  df[, i] <- as.numeric(df[, i])
}

# Place all target variables - totalRetired* - together
df <- df[, c(1:4, 62:64, 91:93, 120:122, 5:61, 65:90, 94:119)]
df1 <- df[complete.cases(df), ]

# Timeseries Evaluation ####
realDJIOpen <-
  ts(
    df$realDJIopen,
    start = c(1985, 1),
    end = c(2018, 9),
    frequency = 12
  )
percRealDJIOpen <-
  ts(
    df1$percChangeRealDJIopen,
    start = c(1985, 2),
    end = c(2018, 9),
    frequency = 12
  )
realSPOpen <-
  ts(
    df$realSPopen,
    start = c(1985, 1),
    end = c(2018, 9),
    frequency = 12
  )
percRealSPOpen <-
  ts(
    df1$percChangeRealSPopen,
    start = c(1985, 2),
    end = c(2018, 9),
    frequency = 12
  )
fedFund <-
  ts(
    df$fedFundRate,
    start = c(1985, 1),
    end = c(2018, 9),
    frequency = 12
  )

totalRetired <-
  ts(
    df$totalSSRetired,
    start = c(1985, 1),
    end = c(2018, 9),
    frequency = 12
  )


plot(stl(realDJIOpen, s.window = "period"), lwd = 1)
title(main = 'Seasonal Decomp of Real DJI Open (Sep 2018 Dollars)')
```



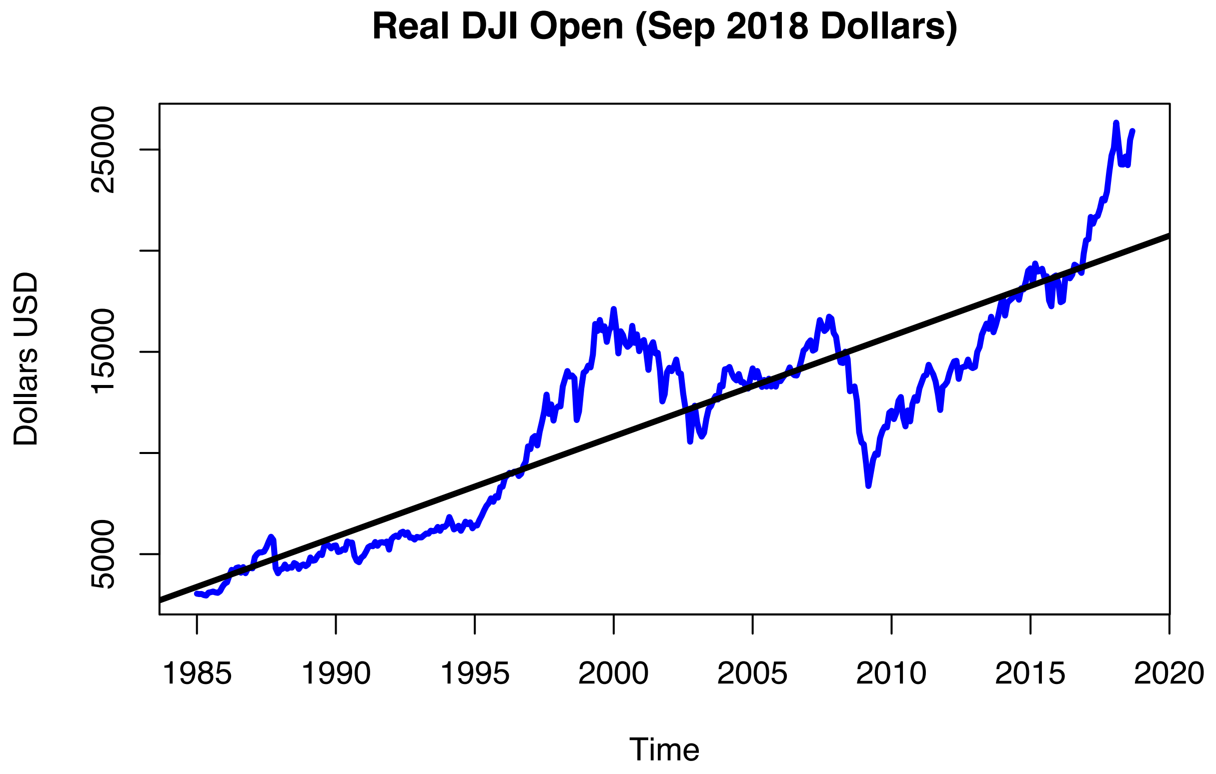
```
plot(stl(log(realDJIOpen), s.window = "period"), lwd = 1)
title(main = 'Seasonal Decomp of the Log of Real DJI Open (Sep 2018 Dollars)')
```



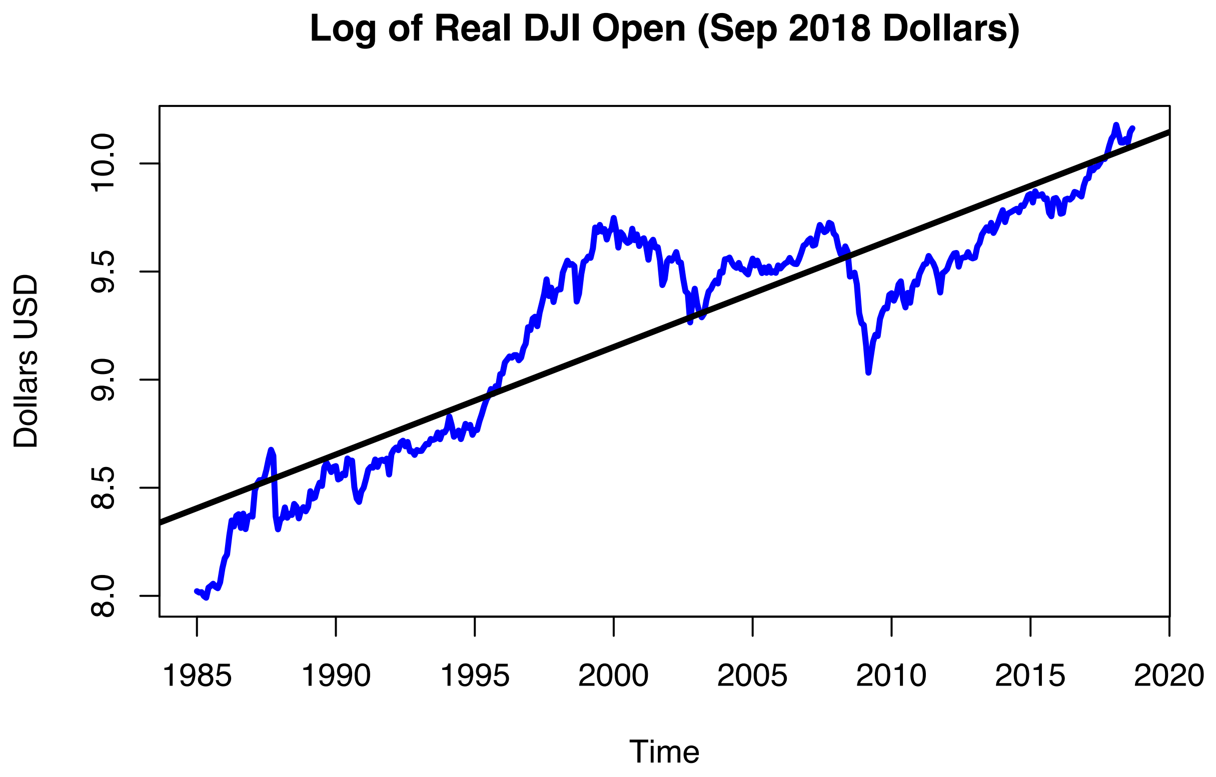
```r
plot(stl(percRealDJIOpen, s.window = "period"), lwd = 1)
title(main = 'Seasonal Decomp of the Percent Change of Real DJI Open (Sep 2018 Dollars)')
```



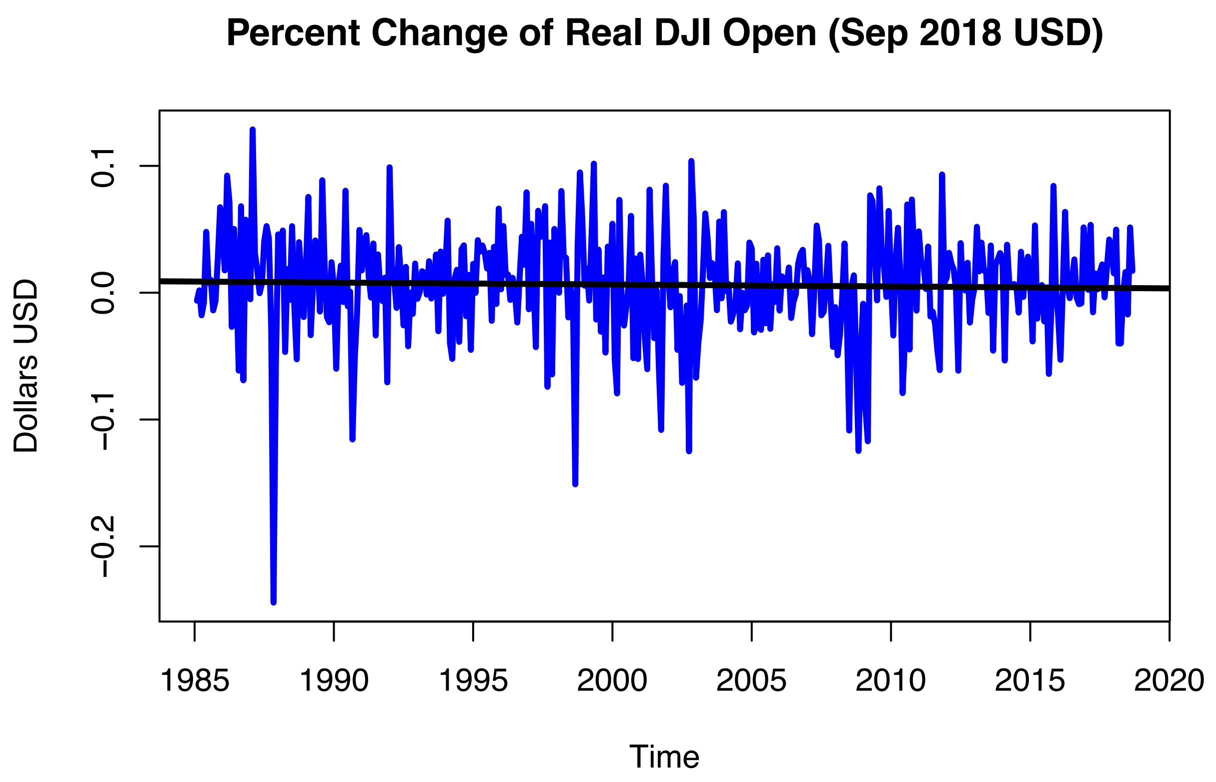
```r
plot(realDJIOpen,
     col = 'blue',
     lwd = 3,
     ylab = 'Dollars USD')
abline(reg = lm(realDJIOpen ~ time(realDJIOpen)), lwd = 3)
title(main = 'Real DJI Open (Sep 2018 Dollars)')
```



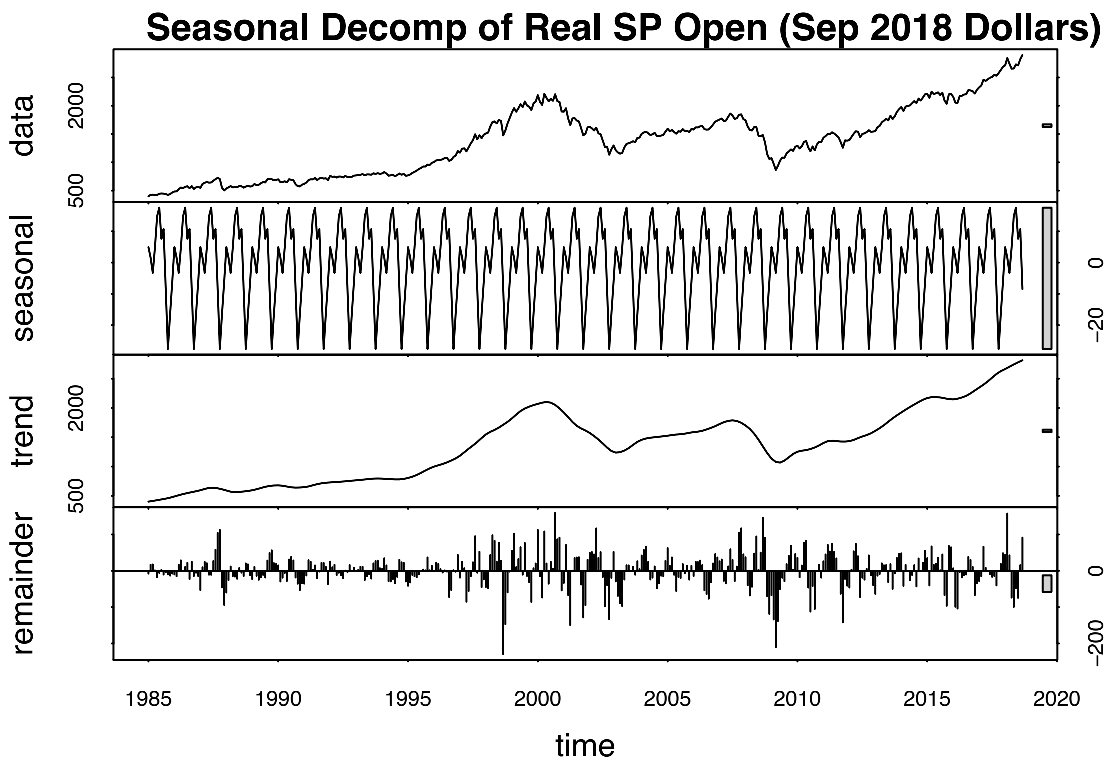
```r
plot(log(realDJIOpen),
     col = 'blue',
     lwd = 3,
     ylab = 'Dollars USD')
abline(reg = lm(log(realDJIOpen) ~ time(log(realDJIOpen))), lwd = 3)
title(main = 'Log of Real DJI Open (Sep 2018 Dollars)')
```



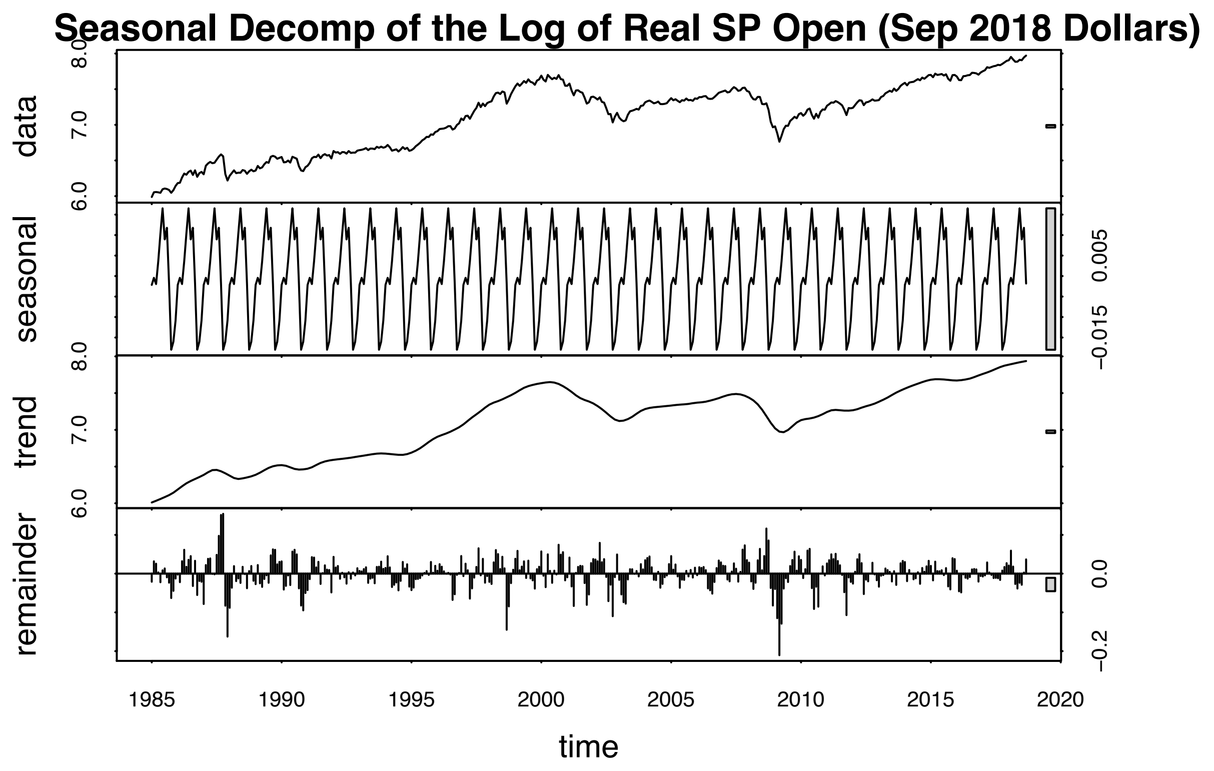
```r
plot(percRealDJIOpen,
     col = 'blue',
     lwd = 3,
     ylab = 'Dollars USD')
abline(reg = lm(percRealDJIOpen ~ time(percRealDJIOpen)), lwd = 3)
title(main = 'Percent Change of Real DJI Open (Sep 2018 USD)')
```



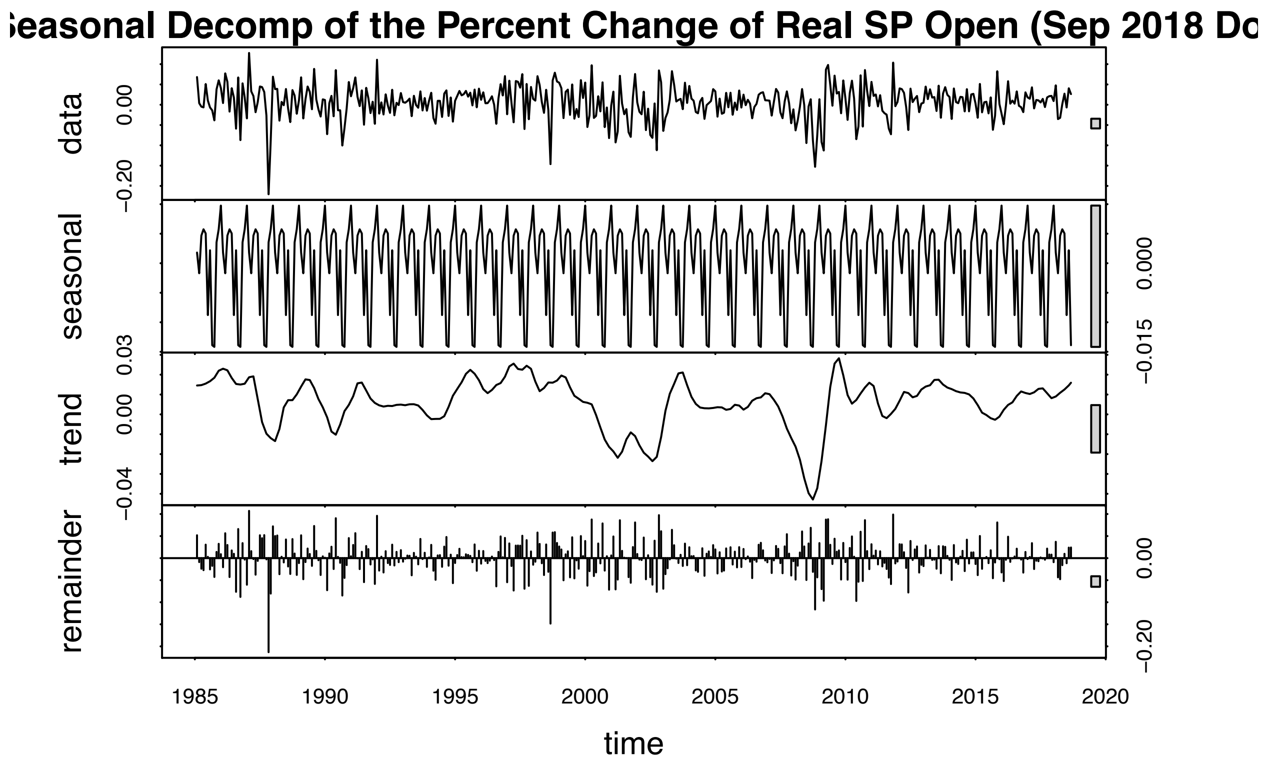
```
plot(stl(realSPOpen, s.window = "period"), lwd = 1)
title(main = 'Seasonal Decomp of Real SP Open (Sep 2018 Dollars)')
```



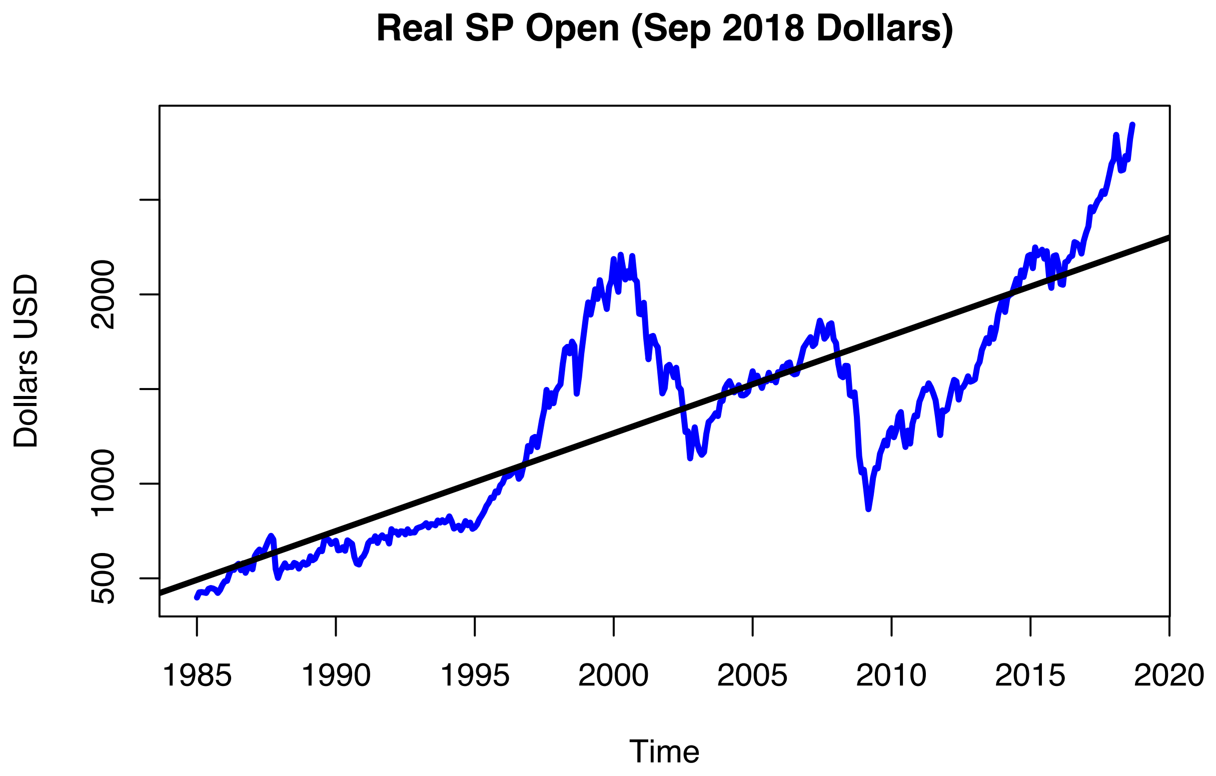
```
plot(stl(log(realSPOpen), s.window = "period"), lwd = 1)
title(main = 'Seasonal Decomp of the Log of Real SP Open (Sep 2018 Dollars)')
```



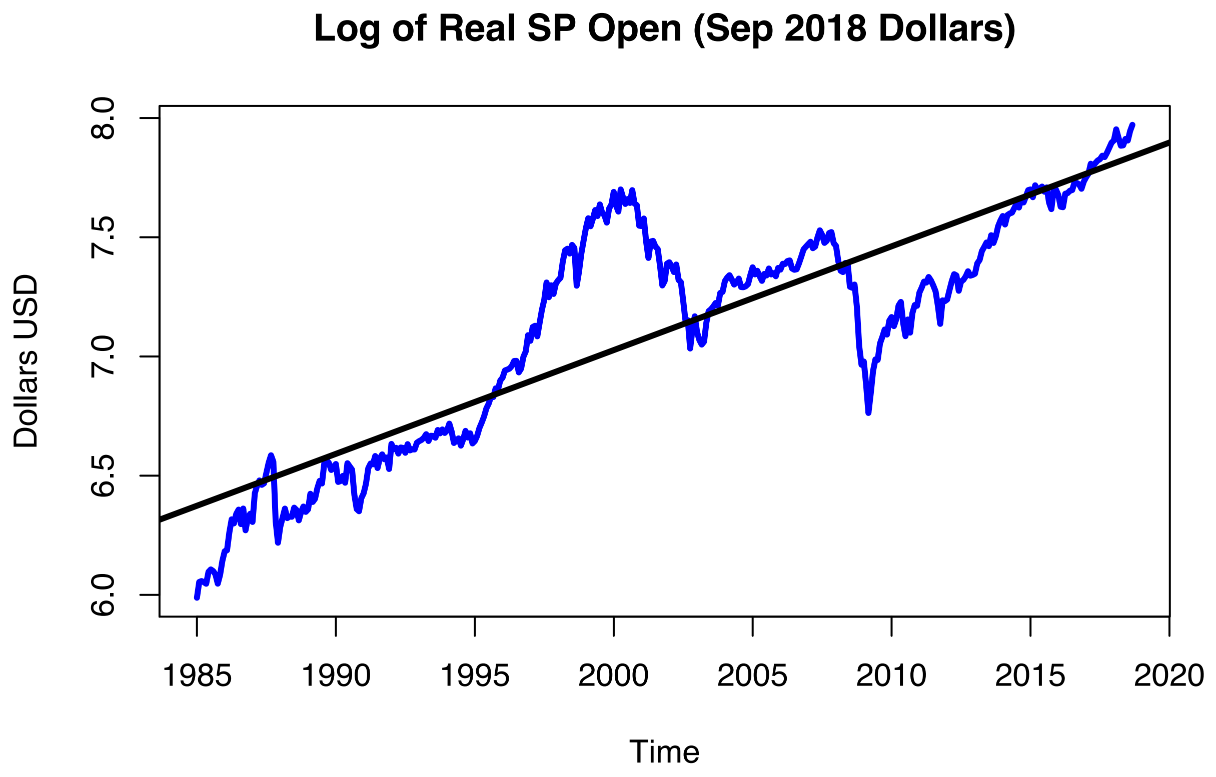
```r
plot(stl(percRealSPOpen, s.window = "period"), lwd = 1)
title(main = 'Seasonal Decomp of the Percent Change of Real SP Open (Sep 2018 Dollars)')
```



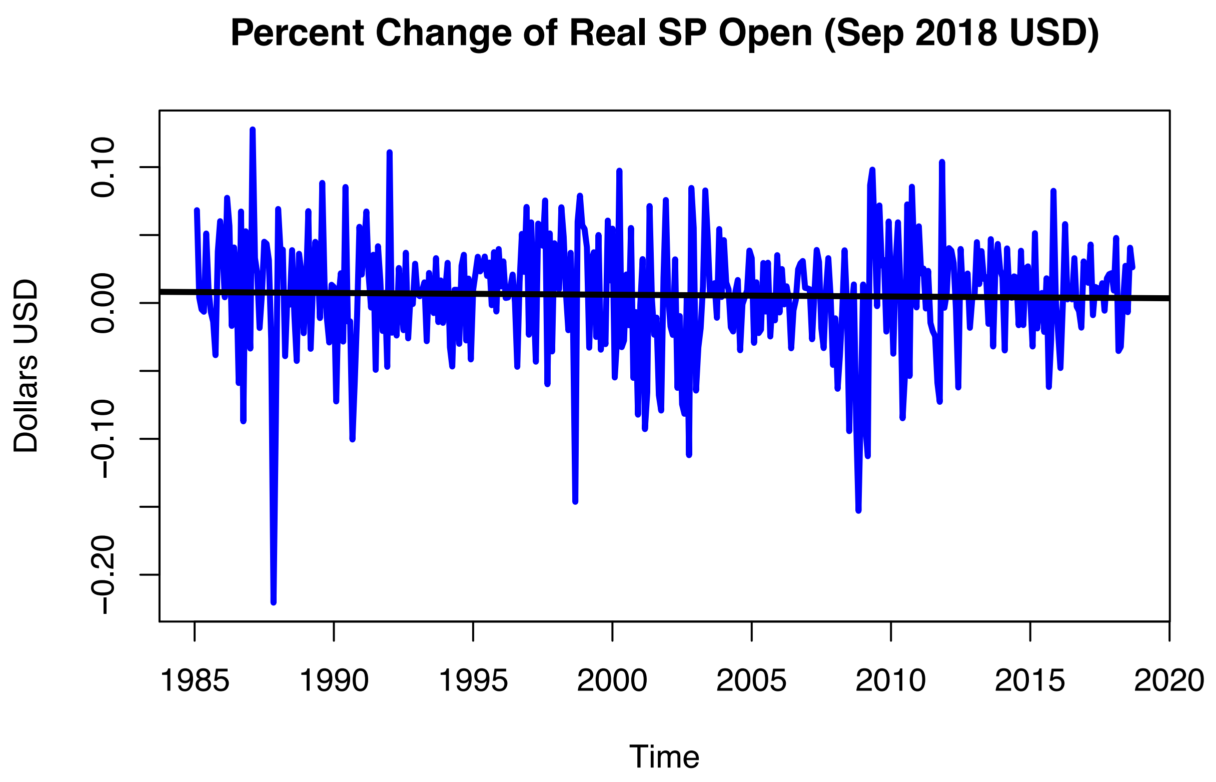
```r
plot(realSPOpen,
     col = 'blue',
     lwd = 3,
     ylab = 'Dollars USD')
abline(reg = lm(realSPOpen ~ time(realSPOpen)), lwd = 3)
title(main = 'Real SP Open (Sep 2018 Dollars)')
```



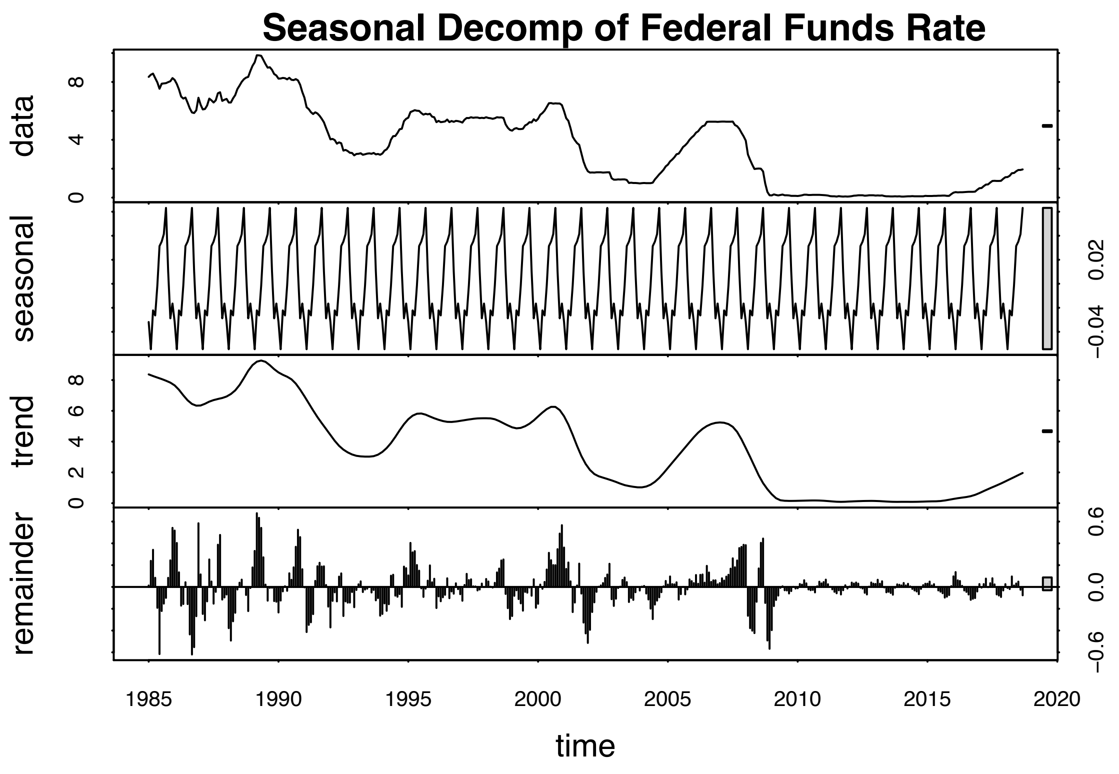
```
plot(log(realSPOpen),
     col = 'blue',
     lwd = 3,
     ylab = 'Dollars USD')
abline(reg = lm(log(realSPOpen) ~ time(log(realSPOpen))), lwd = 3)
title(main = 'Log of Real SP Open (Sep 2018 Dollars)')
```



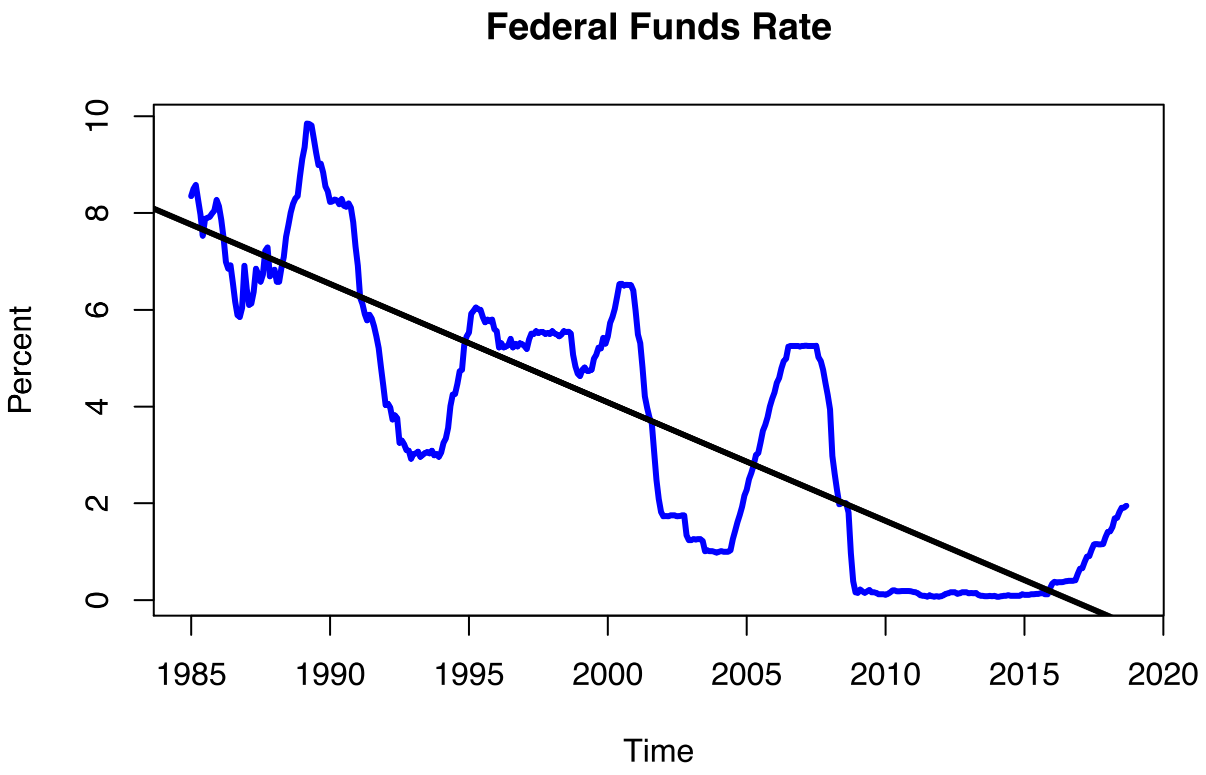
```r
plot(percRealSPOpen,
     col = 'blue',
     lwd = 3,
     ylab = 'Dollars USD')
abline(reg = lm(percRealSPOpen ~ time(percRealSPOpen)), lwd = 3)
title(main = 'Percent Change of Real SP Open (Sep 2018 USD)')
```



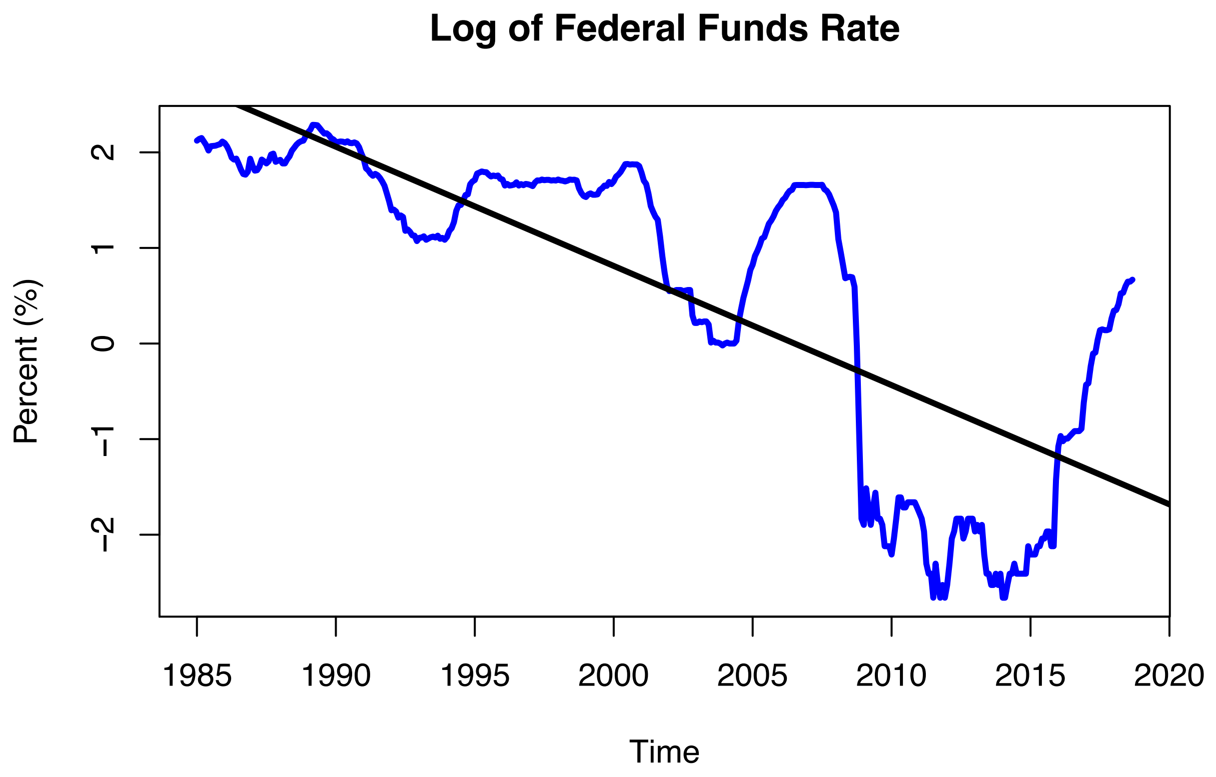
```r
plot(stl(fedFund, s.window = "period"), lwd = 1)
title(main = 'Seasonal Decomp of Federal Funds Rate')
```



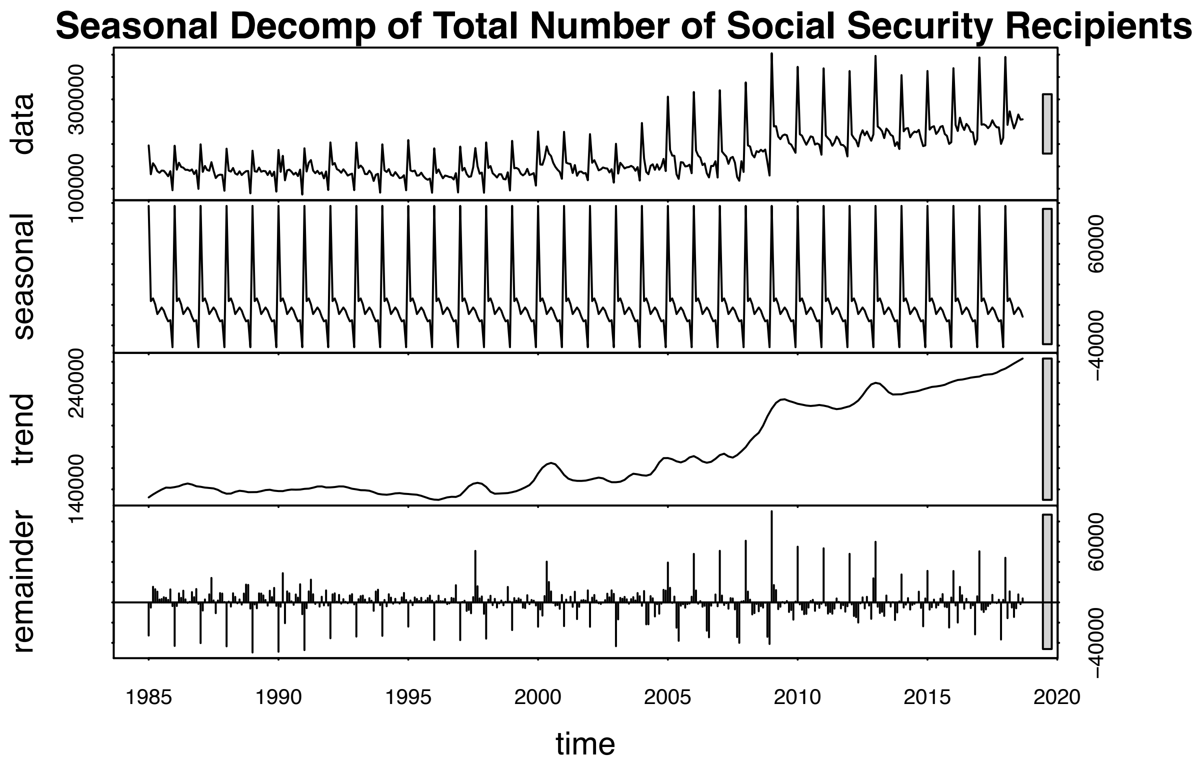
```r
plot(fedFund,
     col = 'blue',
     lwd = 3,
     ylab = 'Percent')
abline(reg = lm(fedFund ~ time(fedFund)), lwd = 3)
title(main = 'Federal Funds Rate')
```



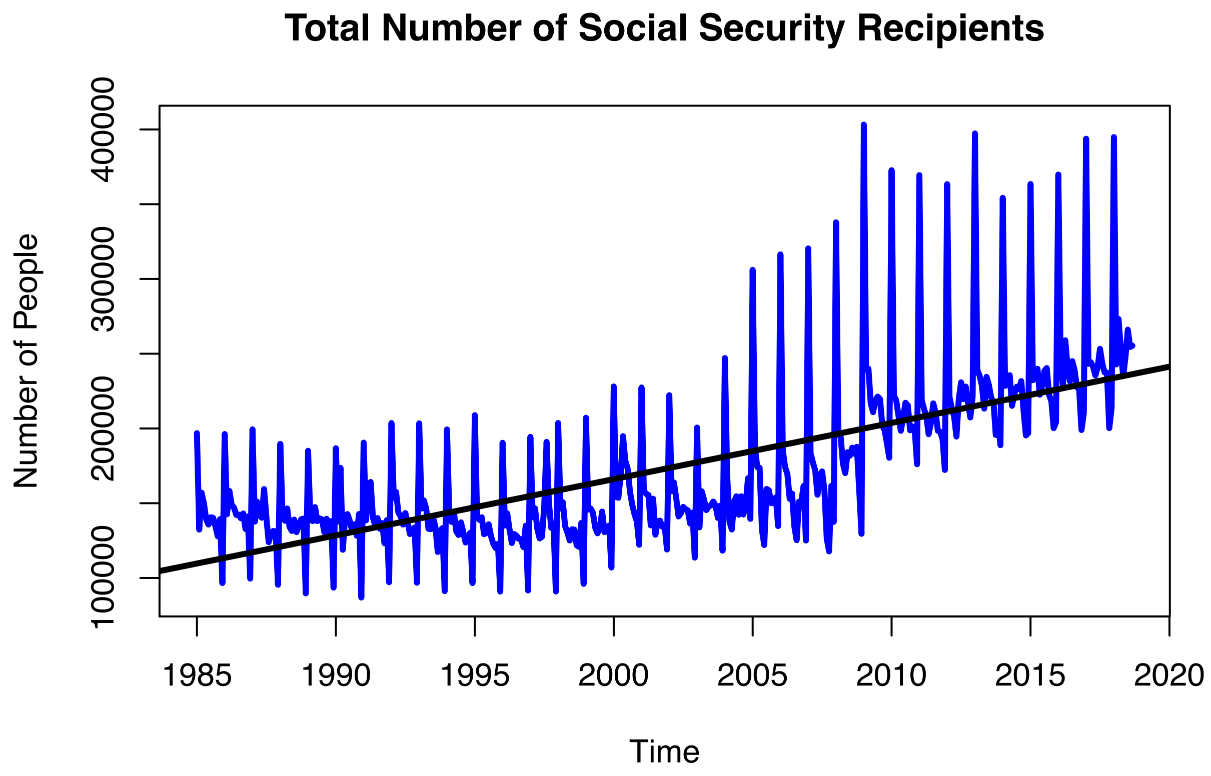
```
plot(log(fedFund),
     col = 'blue',
     lwd = 3,
     ylab = 'Percent (%)')
abline(reg = lm(log(fedFund) ~ time(log(fedFund))), lwd = 3)
title(main = 'Log of Federal Funds Rate')
```



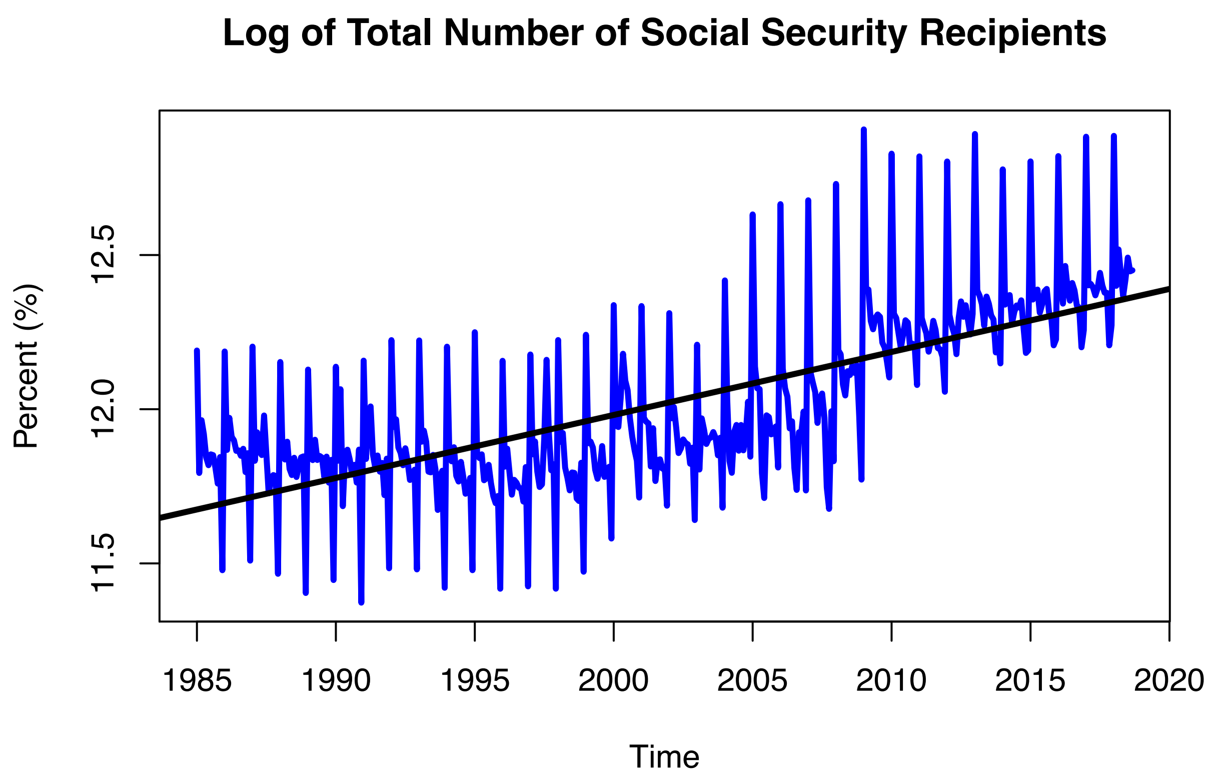
```r
plot(stl(totalRetired, s.window = "period"), lwd = 1)
title(main = 'Seasonal Decomp of Total Number of Social Security Recipients')
```



```r
plot(totalRetired,
     col = 'blue',
     lwd = 3,
     ylab = 'Number of People')
abline(reg = lm(totalRetired ~ time(totalRetired)), lwd = 3)
title(main = 'Total Number of Social Security Recipients')
```



```r
plot(log(totalRetired),
     col = 'blue',
     lwd = 3,
     ylab = 'Percent (%)')
abline(reg = lm(log(totalRetired) ~ time(log(totalRetired))), lwd = 3)
title(main = 'Log of Total Number of Social Security Recipients')
```

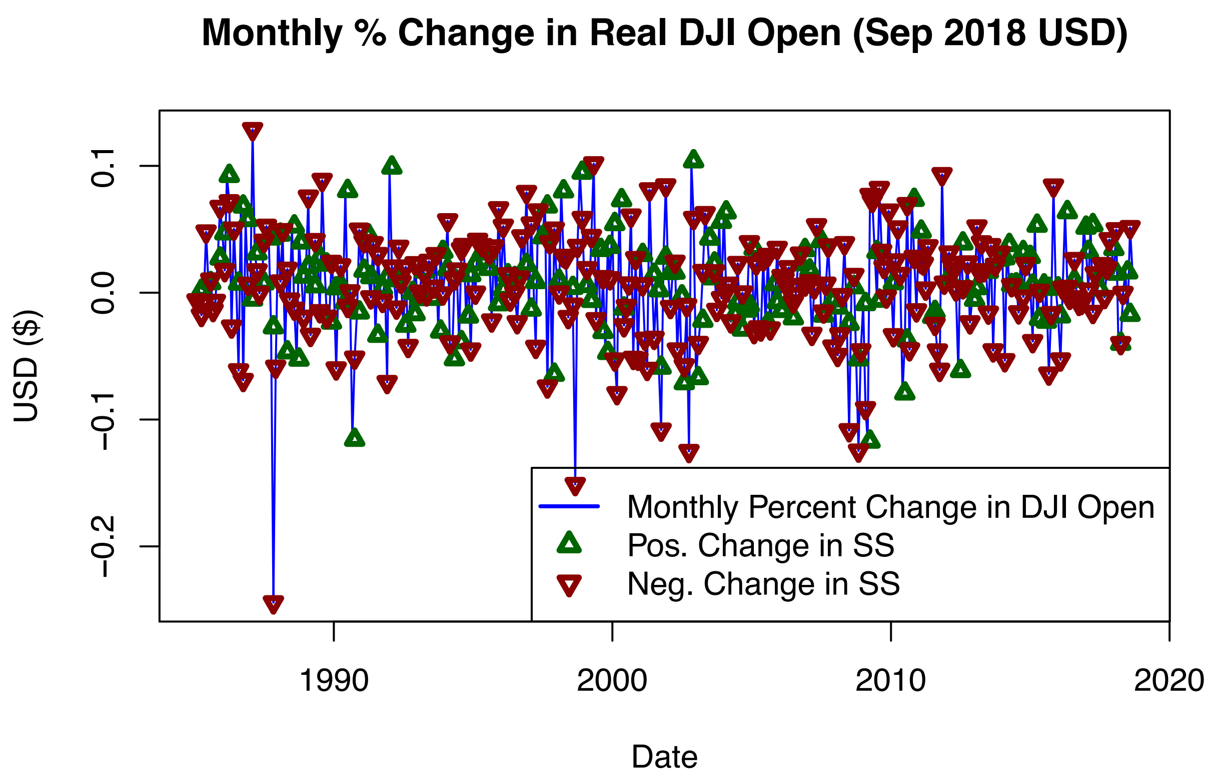


```
# Remove nominal values aside indicators of positive change
df2 <- df1[, c(1, 8:10, 11:18, 32:44, 58:96, 110:122)]
# remove components of the total SS Retirees (male + female = total) and percent increases and decrease
df3 <- df2[, c(1:2, 8:9, 13:77)]

# Hypothesis Tests ####
# Stationarity Loop Testing
statVars <- matrix(data = NA, nrow = 68, ncol = 2)
df3TS <- ts(
  df3,
  start = c(1985, 12),
  end = c(2018, 9),
  frequency = 12
)
for (i in c(1:68)) {
  statVars[i,1] <- i+1
  statVars[i,2] <- adf.test(df3TS[,i+1], alternative = 'stationary')[[4]]
}
```

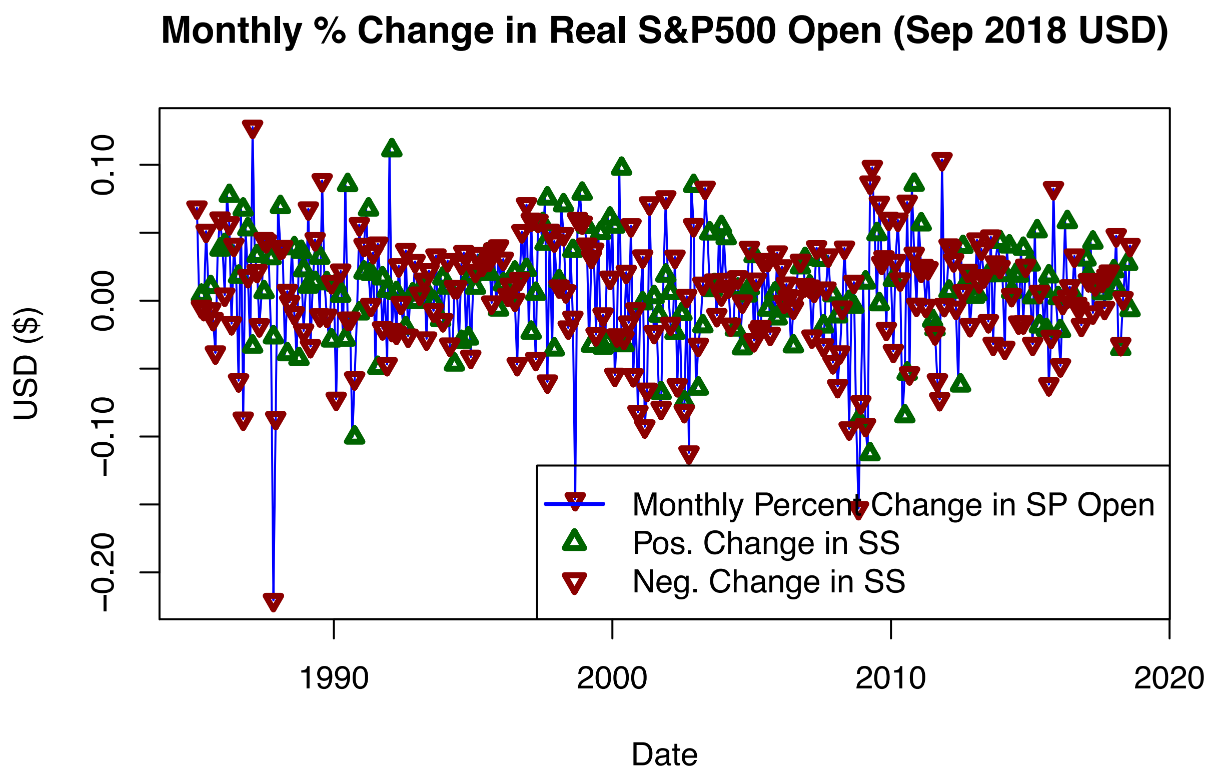
```
## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value
## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value

## Warning in adf.test(df3TS[, i + 1], alternative = "stationary"): p-value
## smaller than printed p-value
```

```r
# Reject the null when p < 0.05. So, the variables associated with this are
# likely stationary and useful for prediction of time series
dfStatSelect<- statVars[,1][statVars[,2] < 0.05]
dfStationary<- df3[,c(1,dfStatSelect)]
```

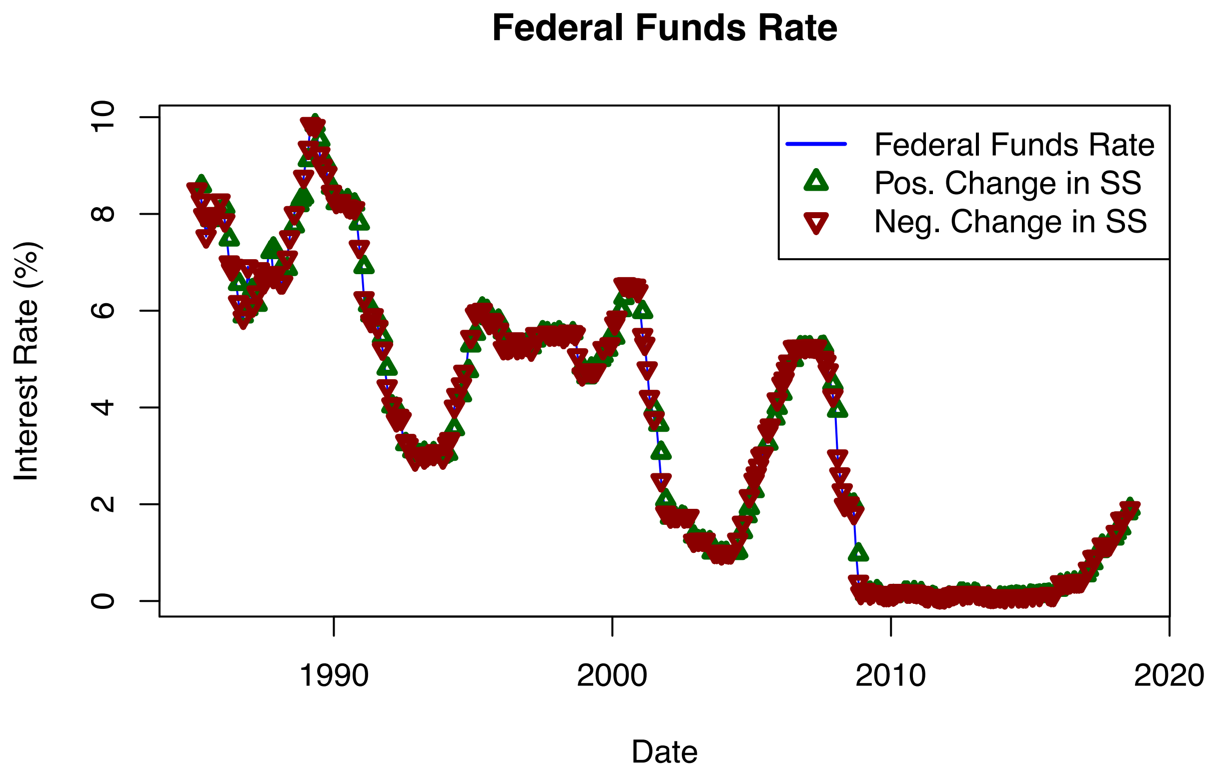
```r
# Visualizations ####
plot(
  x = dfStationary$date,
  y = dfStationary$percChangeRealDJIopen,
  col = 'blue',
  lwd = 1,
  type = 'l',
  ylab = 'USD ($)',
  xlab = 'Date'
)
points(
  x = dfStationary$date[df$postotalSSRetired == 1],
  y = dfStationary$percChangeRealDJIopen[dfStationary$postotalSSRetired == 1],
  pch = 24,
  col = 'darkgreen',
  cex = 0.8,
  lwd = 3
)
points(
  x = dfStationary$date[dfStationary$postotalSSRetired == 0],
  y = dfStationary$percChangeRealDJIopen[dfStationary$postotalSSRetired == 0],
  pch = 25,
  col = 'darkred',
  cex = 0.8,
  lwd = 3
)
legend(
  'bottomright',
  legend = c(
    'Monthly Percent Change in DJI Open',
    c('Pos. Change in SS', 'Neg. Change in SS')
  ),
  lty = c(1, c(NA, NA)),
  pch = c(NA, c(24, 25)),
  col = c('blue', c('darkgreen', 'darkred')),
  bg = c(NA, c('darkgreen', 'darkred')),
  lwd = c(2, c(3, 3))
)
title(main = 'Monthly % Change in Real DJI Open (Sep 2018 USD)')
```



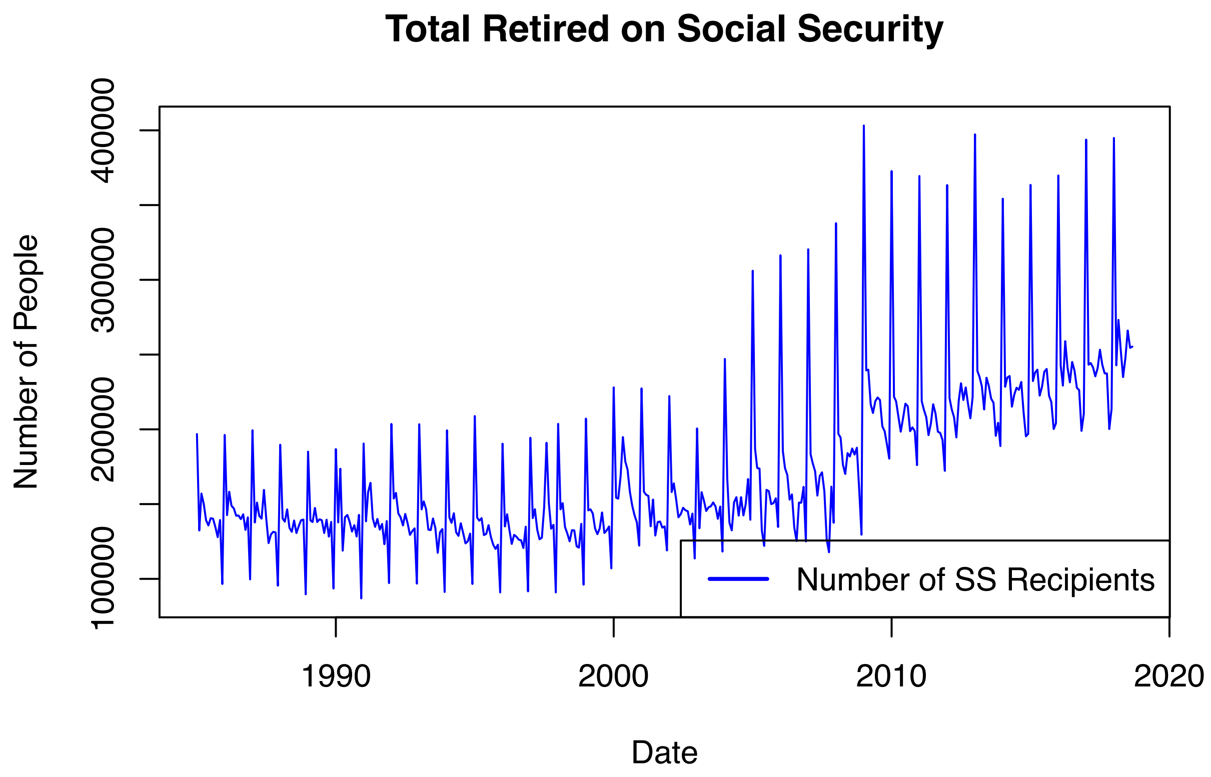
```r
plot(
  x = dfStationary$date,
  y = dfStationary$percChangeRealSPopen,
  col = 'blue',
  lwd = 1,
  type = 'l',
  ylab = 'USD ($)',
  xlab = 'Date'
)
points(
  x = dfStationary$date[df$postotalSSRetired == 1],
  y = dfStationary$percChangeRealSPopen[dfStationary$postotalSSRetired == 1],
  pch = 24,
  col = 'darkgreen',
  cex = 0.8,
  lwd = 3
)
points(
  x = dfStationary$date[dfStationary$postotalSSRetired == 0],
  y = dfStationary$percChangeRealSPopen[dfStationary$postotalSSRetired == 0],
  pch = 25,
  col = 'darkred',
  cex = 0.8,
  lwd = 3
)
legend(
  'bottomright',
  legend = c(
    'Monthly Percent Change in SP Open',
    c('Pos. Change in SS', 'Neg. Change in SS')
  ),
  lty = c(1, c(NA, NA)),
  pch = c(NA, c(24, 25)),
  col = c('blue', c('darkgreen', 'darkred')),
  bg = c(NA, c('darkgreen', 'darkred')),
  lwd = c(2, c(3, 3))
)
title(main = 'Monthly % Change in Real S&P500 Open (Sep 2018 USD)')
```



```
plot(
  x = dfStationary$date,
  y = dfStationary$fedFundRate,
  col = 'blue',
  lwd = 1,
  type = 'l',
  ylab = 'Interest Rate (%)',
  xlab = 'Date'
)
points(
  x = dfStationary$date[df$postotalSSRetired == 1],
  y = dfStationary$fedFundRate[dfStationary$postotalSSRetired == 1],
  pch = 24,
  col = 'darkgreen',
  cex = 0.8,
  lwd = 3
)
points(
  x = dfStationary$date[dfStationary$postotalSSRetired == 0],
  y = dfStationary$fedFundRate[dfStationary$postotalSSRetired == 0],
  pch = 25,
  col = 'darkred',
  cex = 0.8,
  lwd = 3
)
legend(
  'topright',
  legend = c('Federal Funds Rate',
             c('Pos. Change in SS', 'Neg. Change in SS')),
  lty = c(1, c(NA, NA)),
  pch = c(NA, c(24, 25)),
  col = c('blue', c('darkgreen', 'darkred')),
  bg = c(NA, c('darkgreen', 'darkred')),
  lwd = c(2, c(3, 3))
)
title(main = 'Federal Funds Rate')
```



```
plot(
  x = df$date,
  y = df$totalSSRetired,
  col = 'blue',
  lwd = 1,
  type = 'l',
  ylab = 'Number of People',
  xlab = 'Date'
)
legend(
  'bottomright',
  legend = c('Number of SS Recipients'),
  lty = c(1),
  col = c('blue'),
  lwd = c(2)
)
title(main = 'Total Retired on Social Security')
```



```
# Selection ####
# Set a few dataframes for different variables
dfDiff <- dfStationary[,c(2:5,7:19)]
dfPosChange <- dfStationary[,c(2:5, 20:45)]
dfPerc <- dfStationary[,c(2:5, 46:58)]
# Run the selections
# Differences ####
# SeqRep
regFitSelect <- regsubsets(
  postotalSSRetired~.,
  data=dfDiff,
  method= 'seqrep',
  nvmax=17)
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 2 linear dependencies found
```

```
## Reordering variables and trying again:
```

```
regSummary <- summary(regFitSelect)
names(regSummary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```
regSummary$rsq
```

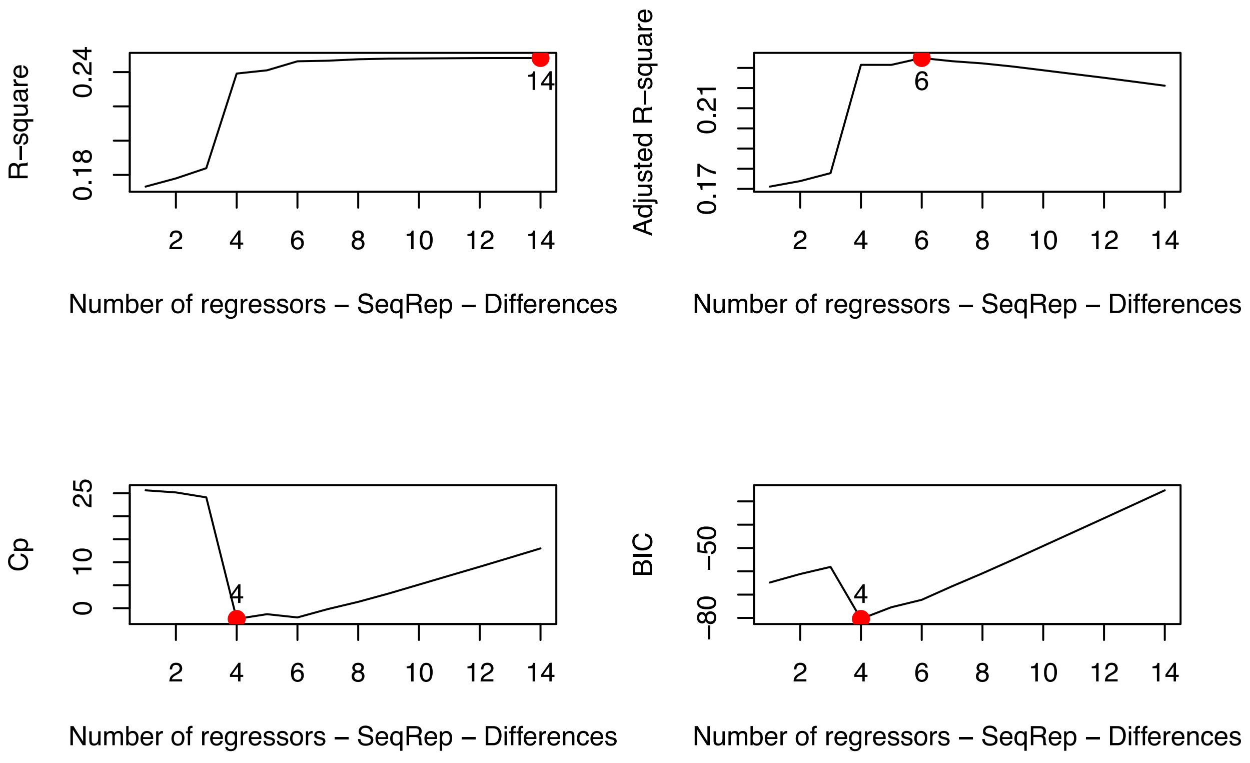
```
##  [1] 0.1731698 0.1779456 0.1839180 0.2391592 0.2410517 0.2463118 0.2466627
##  [8] 0.2474956 0.2478743 0.2479814 0.2480829 0.2481942 0.2482266 0.2482272
```

```
regSummary$adjr2
```

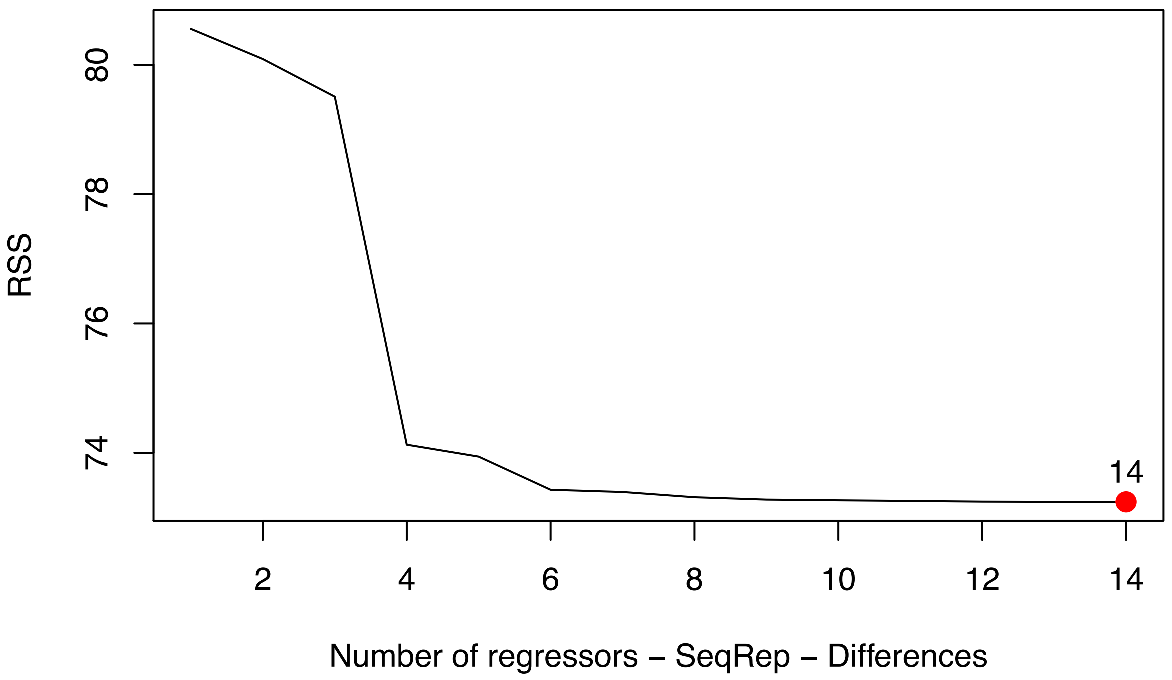
```
##  [1] 0.1711130 0.1738456 0.1777974 0.2315317 0.2315172 0.2349210 0.2333461
##  [8] 0.2322550 0.2306938 0.2288461 0.2269832 0.2251209 0.2231675 0.2211711
```

```
par(mfrow=c(2,2))
aRSQ <- which.max(regSummary$rsq)
aARSQ <- which.max(regSummary$adjr2)
aCP <- which.min(regSummary$cp)
aBIC <- which.min(regSummary$bic)
aRSS <- which.min(regSummary$rss)

par(mfrow = c(2, 2))

plot(
  regSummary$rsq,
  xlab = "Number of regressors - SeqRep - Differences",
  ylab = "R-square",
  type = "l"
)
points(
  aRSQ,
  regSummary$rsq[aRSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSQ,
     regSummary$rsq[aRSQ],
     labels = aRSQ,
     pos = 1)

plot(
  regSummary$adjr2,
  xlab = "Number of regressors - SeqRep - Differences",
  ylab = "Adjusted R-square",
  type = "l"
)
points(
  aARSQ,
  regSummary$adjr2[aARSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aARSQ,
     regSummary$adjr2[aARSQ],
     labels = aARSQ,
     pos = 1)

plot(regSummary$cp,
     xlab = "Number of regressors - SeqRep - Differences",
     ylab = "Cp",
```

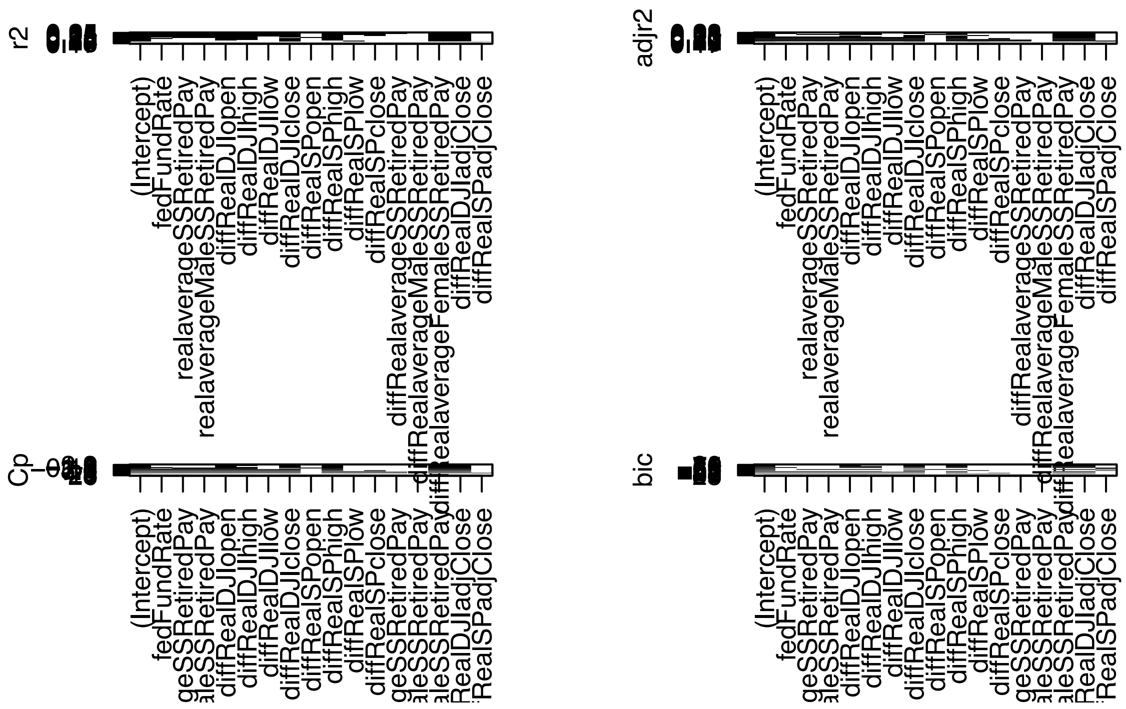
```r
     type = "l")
points(
  aCP,
  regSummary$cp[aCP],
  col = "red",
  cex = 2,
  pch = 20
)
text(aCP,
     regSummary$cp[aCP],
     labels = aCP,
     pos = 3)

plot(
  regSummary$bic,
  xlab = "Number of regressors - SeqRep - Differences",
  ylab = "BIC",
  type = "l"
)
points(
  aBIC,
  regSummary$bic[aBIC],
  col = "red",
  cex = 2,
  pch = 20
)
text(aBIC,
     regSummary$bic[aBIC],
     labels = aBIC,
     pos = 3)
```



```r
par(mfrow = c(1, 1))
plot(
  regSummary$rss,
  xlab = "Number of regressors - SeqRep - Differences",
  ylab = "RSS",
  type = "l"
)
points(
  aRSS,
  regSummary$rss[aRSS],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSS,
     regSummary$rss[aRSS],
     labels = aRSS,
     pos = 3)
```



```r
par(mfrow = c(2, 2))
plot(regFitSelect, scale = "r2")
plot(regFitSelect, scale = "adjr2")
plot(regFitSelect, scale = "Cp")
plot(regFitSelect, scale = "bic")
```



```r
valuesSeqRep <- c(names(coef(regFitSelect, id = 4))[-1])
# Forward
regFitSelect <- regsubsets(
  postotalSSRetired~.,
  data=dfDiff,
  method= 'forward',
  nvmax=17)
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 2 linear dependencies found

## Reordering variables and trying again:

## Warning in rval$lopt[] <- rval$vorder[rval$lopt]: number of items to
## replace is not a multiple of replacement length
```

```r
regSummary <- summary(regFitSelect)
names(regSummary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

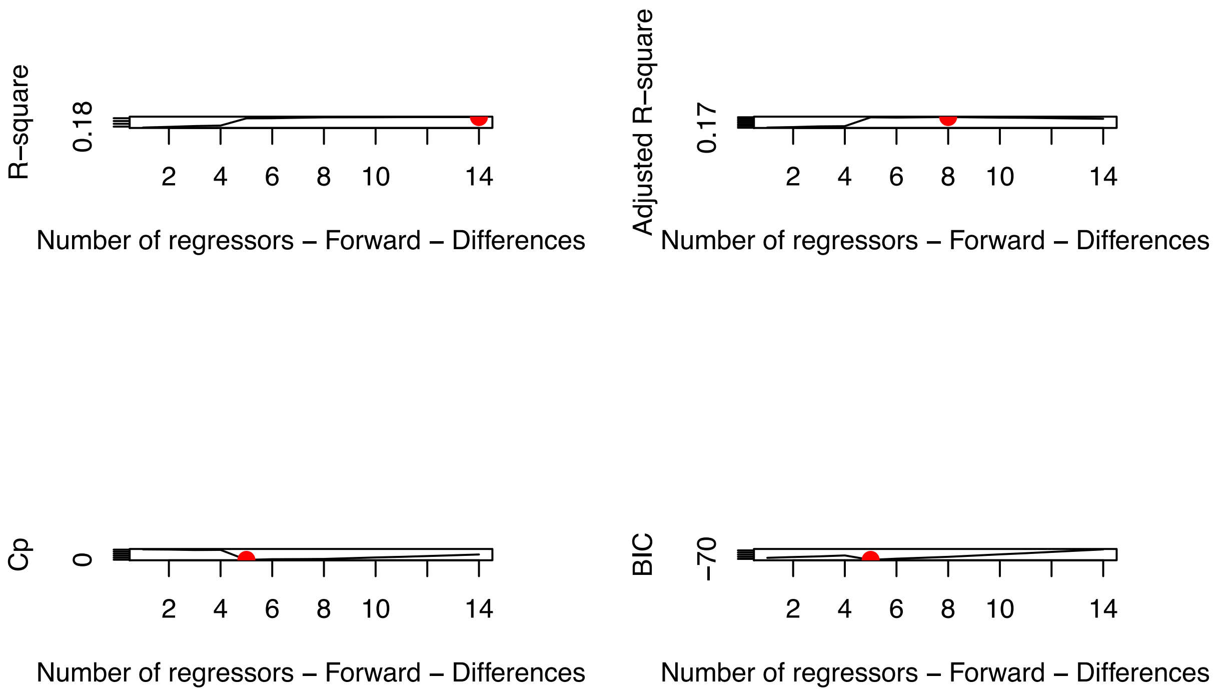
```r
regSummary$rsq
```

```
##  [1] 0.1731698 0.1779456 0.1839180 0.1871817 0.2391704 0.2399022 0.2433283
##  [8] 0.2463473 0.2466978 0.2469470 0.2479080 0.2480453 0.2481380 0.2482272
```

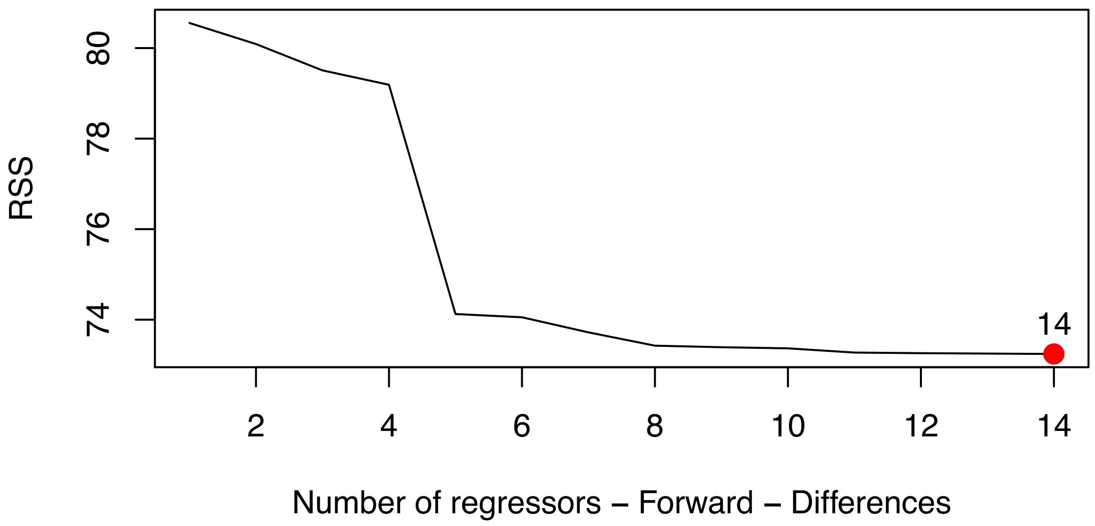
```r
regSummary$adjr2
```

```
##  [1] 0.1711130 0.1738456 0.1777974 0.1790331 0.2296123 0.2284146 0.2299528
##  [8] 0.2310835 0.2294904 0.2277853 0.2268034 0.2249674 0.2230759 0.2211711
```

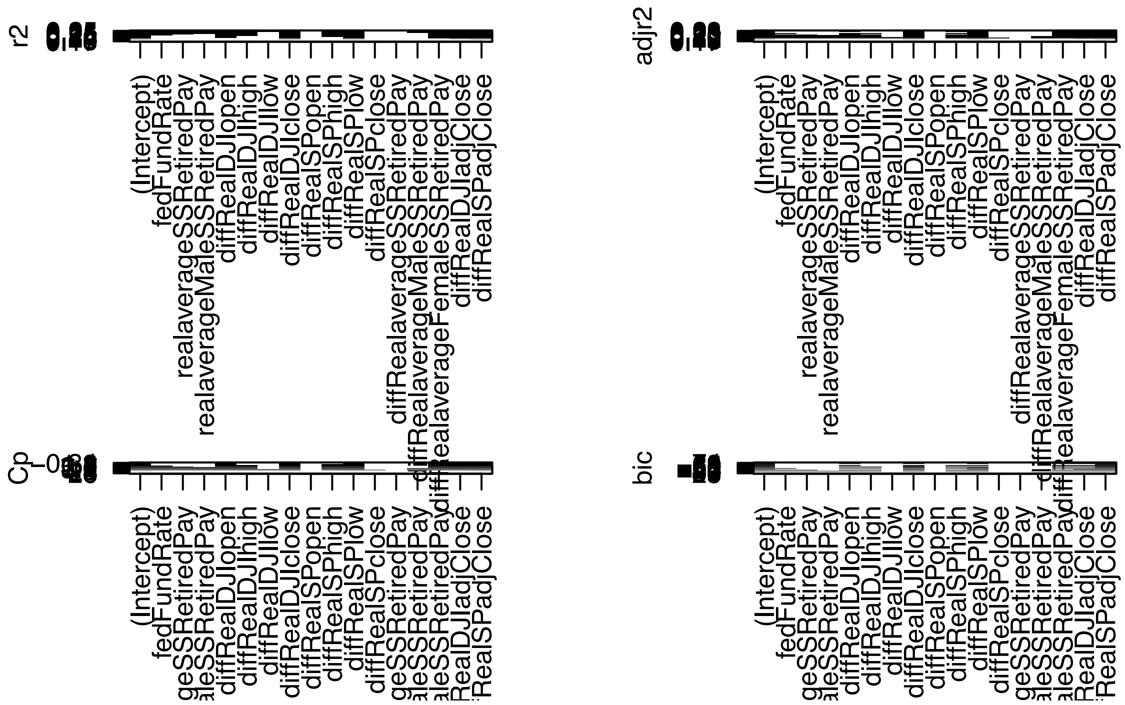
```r
par(mfrow=c(2,2))
aRSQ <- which.max(regSummary$rsq)
aARSQ <- which.max(regSummary$adjr2)
aCP <- which.min(regSummary$cp)
aBIC <- which.min(regSummary$bic)
aRSS <- which.min(regSummary$rss)

par(mfrow = c(2, 2))

plot(
  regSummary$rsq,
  xlab = "Number of regressors - Forward - Differences",
  ylab = "R-square",
  type = "l"
)
points(
  aRSQ,
  regSummary$rsq[aRSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSQ,
     regSummary$rsq[aRSQ],
     labels = aRSQ,
     pos = 1)

plot(
  regSummary$adjr2,
  xlab = "Number of regressors - Forward - Differences",
  ylab = "Adjusted R-square",
  type = "l"
)
points(
  aARSQ,
  regSummary$adjr2[aARSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aARSQ,
     regSummary$adjr2[aARSQ],
     labels = aARSQ,
     pos = 1)

plot(regSummary$cp,
     xlab = "Number of regressors - Forward - Differences",
     ylab = "Cp",
     type = "l")
points(
  aCP,
  regSummary$cp[aCP],
  col = "red",
  cex = 2,
  pch = 20
)
text(aCP,
     regSummary$cp[aCP],
     labels = aCP,
     pos = 3)

plot(
  regSummary$bic,
  xlab = "Number of regressors - Forward - Differences",
  ylab = "BIC",
  type = "l"
)
points(
  aBIC,
  regSummary$bic[aBIC],
  col = "red",
  cex = 2,
  pch = 20
)
text(aBIC,
     regSummary$bic[aBIC],
     labels = aBIC,
     pos = 3)
```



```r
par(mfrow = c(1, 1))
plot(
  regSummary$rss,
  xlab = "Number of regressors - Forward - Differences",
  ylab = "RSS",
  type = "l"
)
points(
  aRSS,
  regSummary$rss[aRSS],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSS,
     regSummary$rss[aRSS],
     labels = aRSS,
     pos = 3)
```



```r
par(mfrow = c(2, 2))
plot(regFitSelect, scale = "r2")
plot(regFitSelect, scale = "adjr2")
plot(regFitSelect, scale = "Cp")
plot(regFitSelect, scale = "bic")
```



```r
valuesForward <- c(names(coef(regFitSelect, id = 5))[-1])
# Backward
regFitSelect <- regsubsets(
  postotalSSRetired~.,
  data=dfDiff,
  method= 'backward',
  nvmax=17)
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 2 linear dependencies found

## Reordering variables and trying again:

## Warning in rval$lopt[] <- rval$vorder[rval$lopt]: number of items to
## replace is not a multiple of replacement length
```

```
regSummary <- summary(regFitSelect)
names(regSummary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```
regSummary$rsq
```

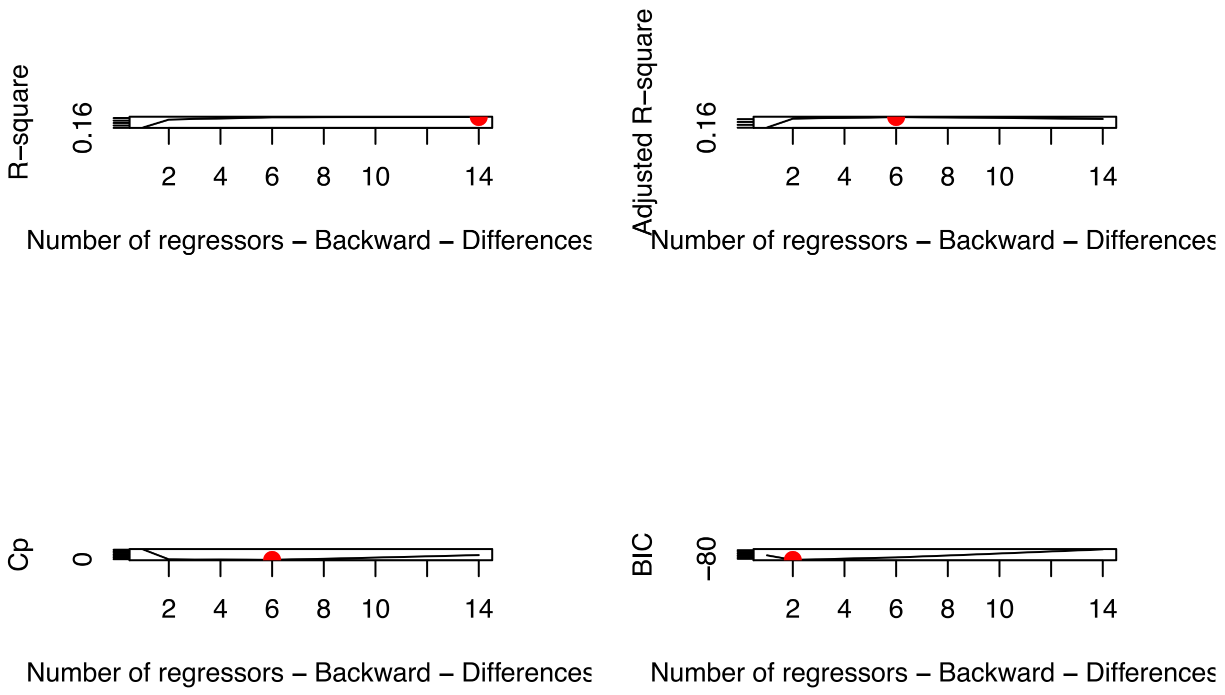
```
##  [1] 0.1630296 0.2273926 0.2330672 0.2364933 0.2410517 0.2463118 0.2464133
##  [8] 0.2474956 0.2478743 0.2479814 0.2480829 0.2481942 0.2482266 0.2482272
```

```
regSummary$adjr2
```

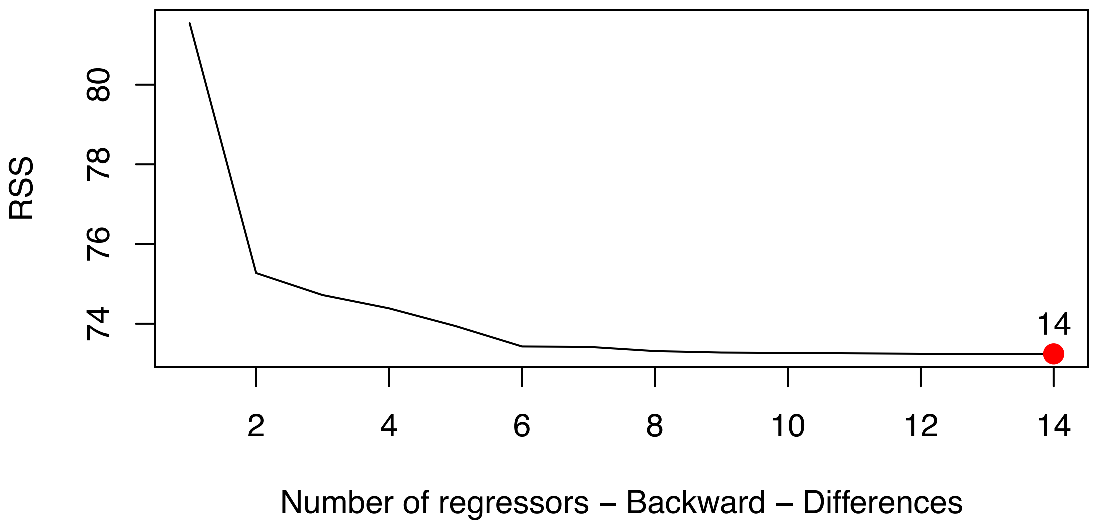
```
##  [1] 0.1609476 0.2235392 0.2273153 0.2288391 0.2315172 0.2349210 0.2330923
##  [8] 0.2322550 0.2306938 0.2288461 0.2269832 0.2251209 0.2231675 0.2211711
```

```
par(mfrow=c(2,2))
aRSQ <- which.max(regSummary$rsq)
aARSQ <- which.max(regSummary$adjr2)
aCP <- which.min(regSummary$cp)
aBIC <- which.min(regSummary$bic)
aRSS <- which.min(regSummary$rss)

par(mfrow = c(2, 2))

plot(
  regSummary$rsq,
  xlab = "Number of regressors - Backward - Differences",
  ylab = "R-square",
  type = "l"
)
points(
  aRSQ,
  regSummary$rsq[aRSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSQ,
     regSummary$rsq[aRSQ],
     labels = aRSQ,
     pos = 1)

plot(
  regSummary$adjr2,
  xlab = "Number of regressors - Backward - Differences",
  ylab = "Adjusted R-square",
  type = "l"
)
points(
  aARSQ,
  regSummary$adjr2[aARSQ],
```

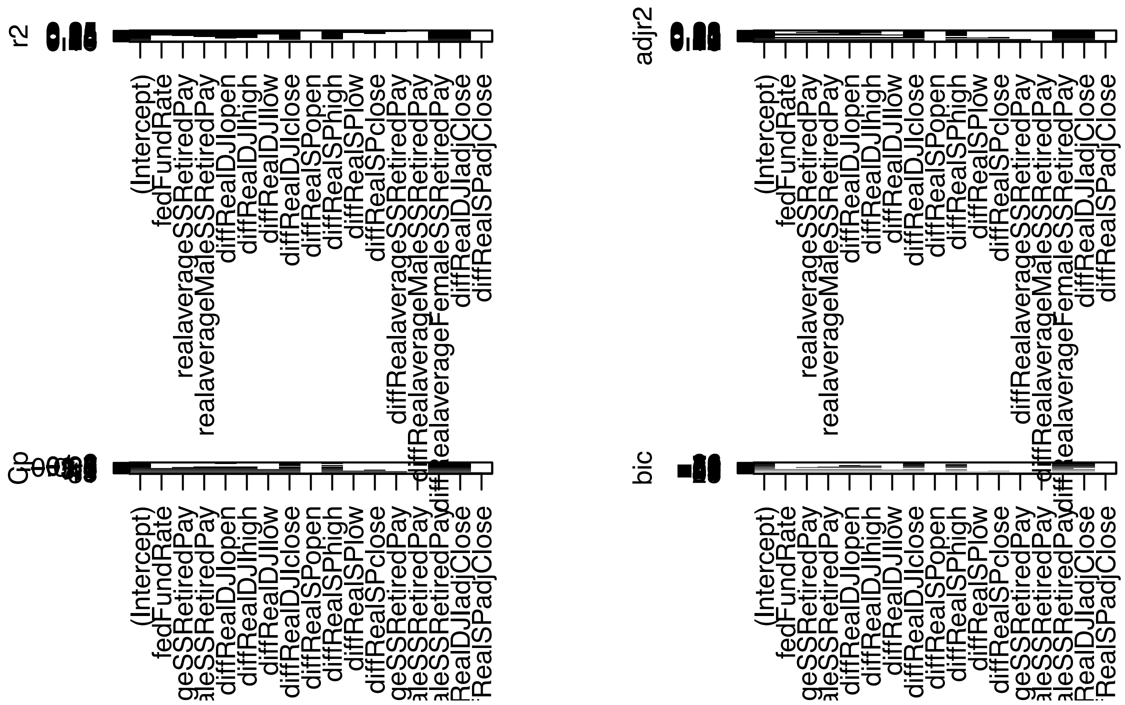
```r
  col = "red",
  cex = 2,
  pch = 20
)
text(aARSQ,
     regSummary$adjr2[aARSQ],
     labels = aARSQ,
     pos = 1)

plot(regSummary$cp,
     xlab = "Number of regressors - Backward - Differences",
     ylab = "Cp",
     type = "l")
points(
  aCP,
  regSummary$cp[aCP],
  col = "red",
  cex = 2,
  pch = 20
)
text(aCP,
     regSummary$cp[aCP],
     labels = aCP,
     pos = 3)

plot(
  regSummary$bic,
  xlab = "Number of regressors - Backward - Differences",
  ylab = "BIC",
  type = "l"
)
points(
  aBIC,
  regSummary$bic[aBIC],
  col = "red",
  cex = 2,
  pch = 20
)
text(aBIC,
     regSummary$bic[aBIC],
     labels = aBIC,
     pos = 3)
```



```
par(mfrow = c(1, 1))
plot(
  regSummary$rss,
  xlab = "Number of regressors - Backward - Differences",
  ylab = "RSS",
  type = "l"
)
points(
  aRSS,
  regSummary$rss[aRSS],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSS,
     regSummary$rss[aRSS],
     labels = aRSS,
     pos = 3)
```



```
par(mfrow = c(2, 2))
plot(regFitSelect, scale = "r2")
plot(regFitSelect, scale = "adjr2")
plot(regFitSelect, scale = "Cp")
plot(regFitSelect, scale = "bic")
```



```
valuesBackward <- c(names(coef(regFitSelect, id = 6))[-1])
# Exhaustive
regFitSelect <- regsubsets(
  postotalSSRetired~.,
  data=dfDiff,
  method= 'exhaustive',
  nvmax=17)
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 2 linear dependencies found
```

```
## Reordering variables and trying again:
```

```r
regSummary <- summary(regFitSelect)
names(regSummary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

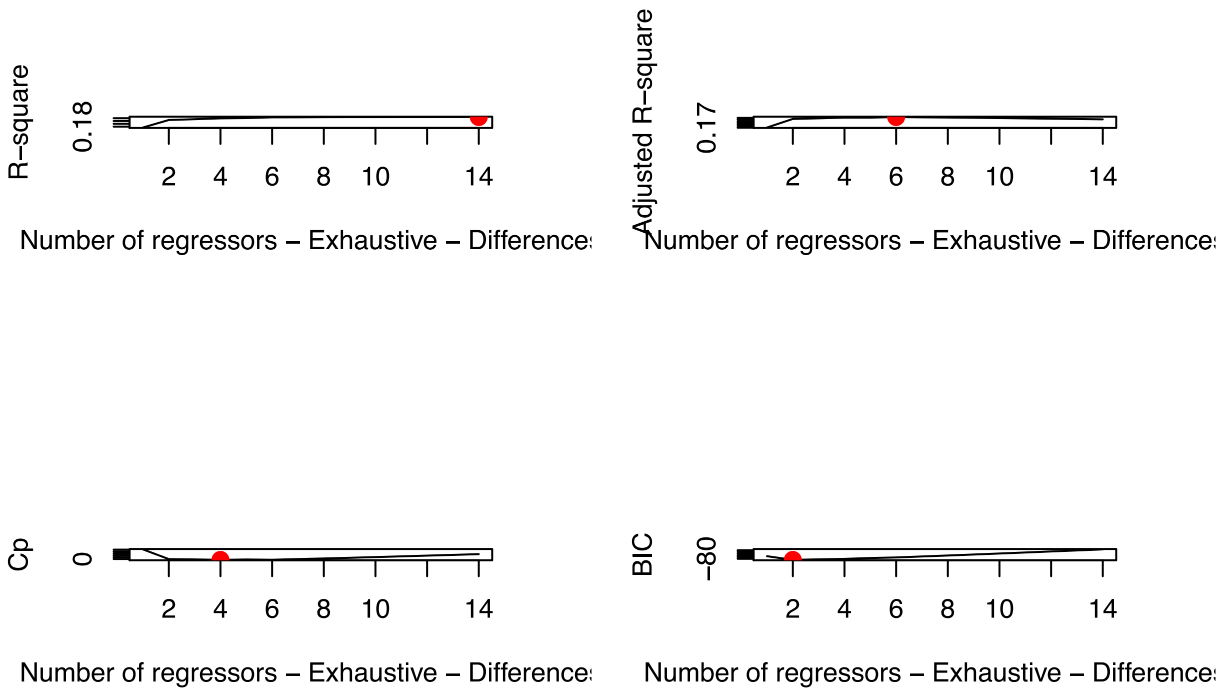
```r
regSummary$rsq
```

```
##  [1] 0.1731698 0.2273926 0.2330672 0.2391592 0.2410517 0.2463118 0.2466627
##  [8] 0.2474956 0.2478743 0.2479814 0.2480829 0.2481942 0.2482266 0.2482272
```

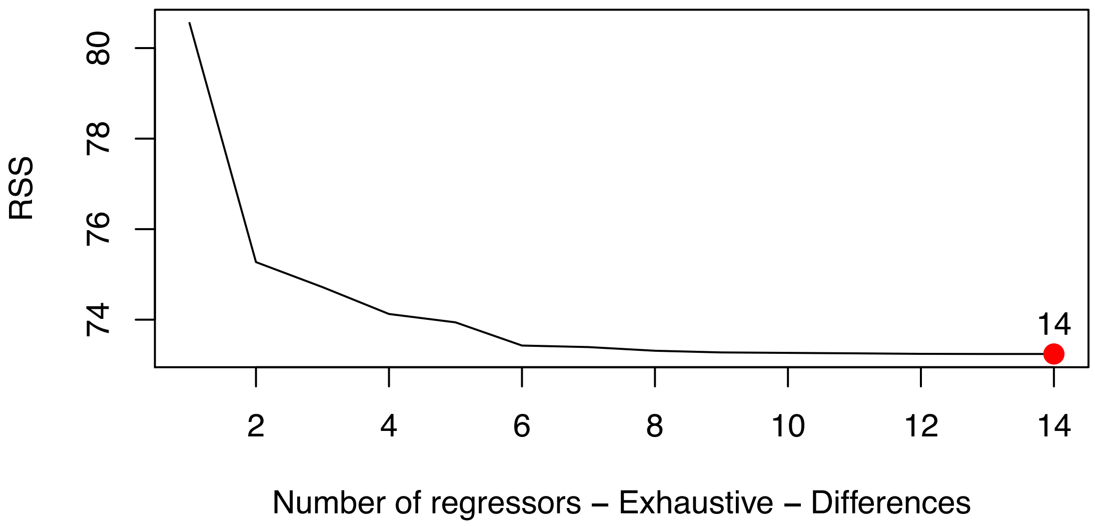
```r
regSummary$adjr2
```

```
##  [1] 0.1711130 0.2235392 0.2273153 0.2315317 0.2315172 0.2349210 0.2333461
##  [8] 0.2322550 0.2306938 0.2288461 0.2269832 0.2251209 0.2231675 0.2211711
```

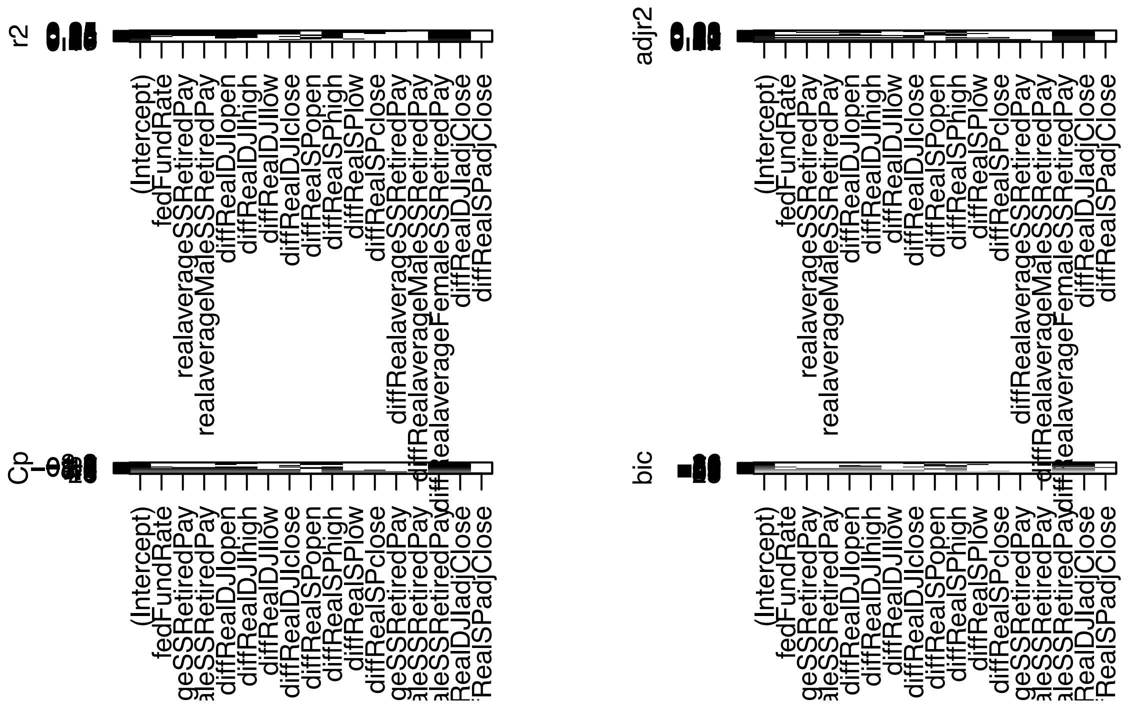
```r
par(mfrow=c(2,2))
aRSQ <- which.max(regSummary$rsq)
aARSQ <- which.max(regSummary$adjr2)
aCP <- which.min(regSummary$cp)
aBIC <- which.min(regSummary$bic)
aRSS <- which.min(regSummary$rss)

par(mfrow = c(2, 2))

plot(
  regSummary$rsq,
  xlab = "Number of regressors - Exhaustive - Differences",
  ylab = "R-square",
  type = "l"
)
points(
  aRSQ,
  regSummary$rsq[aRSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSQ,
     regSummary$rsq[aRSQ],
     labels = aRSQ,
     pos = 1)

plot(
  regSummary$adjr2,
  xlab = "Number of regressors - Exhaustive - Differences",
  ylab = "Adjusted R-square",
  type = "l"
)
points(
  aARSQ,
  regSummary$adjr2[aARSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aARSQ,
     regSummary$adjr2[aARSQ],
     labels = aARSQ,
     pos = 1)

plot(regSummary$cp,
     xlab = "Number of regressors - Exhaustive - Differences",
     ylab = "Cp",
     type = "l")
points(
  aCP,
  regSummary$cp[aCP],
  col = "red",
  cex = 2,
  pch = 20
)
text(aCP,
     regSummary$cp[aCP],
     labels = aCP,
     pos = 3)

plot(
  regSummary$bic,
  xlab = "Number of regressors - Exhaustive - Differences",
  ylab = "BIC",
  type = "l"
)
points(
  aBIC,
  regSummary$bic[aBIC],
  col = "red",
  cex = 2,
  pch = 20
)
text(aBIC,
     regSummary$bic[aBIC],
     labels = aBIC,
     pos = 3)
```



```
par(mfrow = c(1, 1))
plot(
  regSummary$rss,
  xlab = "Number of regressors - Exhaustive - Differences",
  ylab = "RSS",
  type = "l"
)
points(
  aRSS,
  regSummary$rss[aRSS],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSS,
     regSummary$rss[aRSS],
     labels = aRSS,
     pos = 3)
```



```
par(mfrow = c(2, 2))
plot(regFitSelect, scale = "r2")
plot(regFitSelect, scale = "adjr2")
plot(regFitSelect, scale = "Cp")
plot(regFitSelect, scale = "bic")
```



```
valuesExhaustive <- c(names(coef(regFitSelect, id = 4))[-1])

# Percentages - Fairly low value, not going to use ####
regFitSelect <- regsubsets(
  postotalSSRetired~.,
  data=dfPerc,
  nvmax=17)
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 2 linear dependencies found

## Reordering variables and trying again:
```

```
regSummary <- summary(regFitSelect)
names(regSummary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

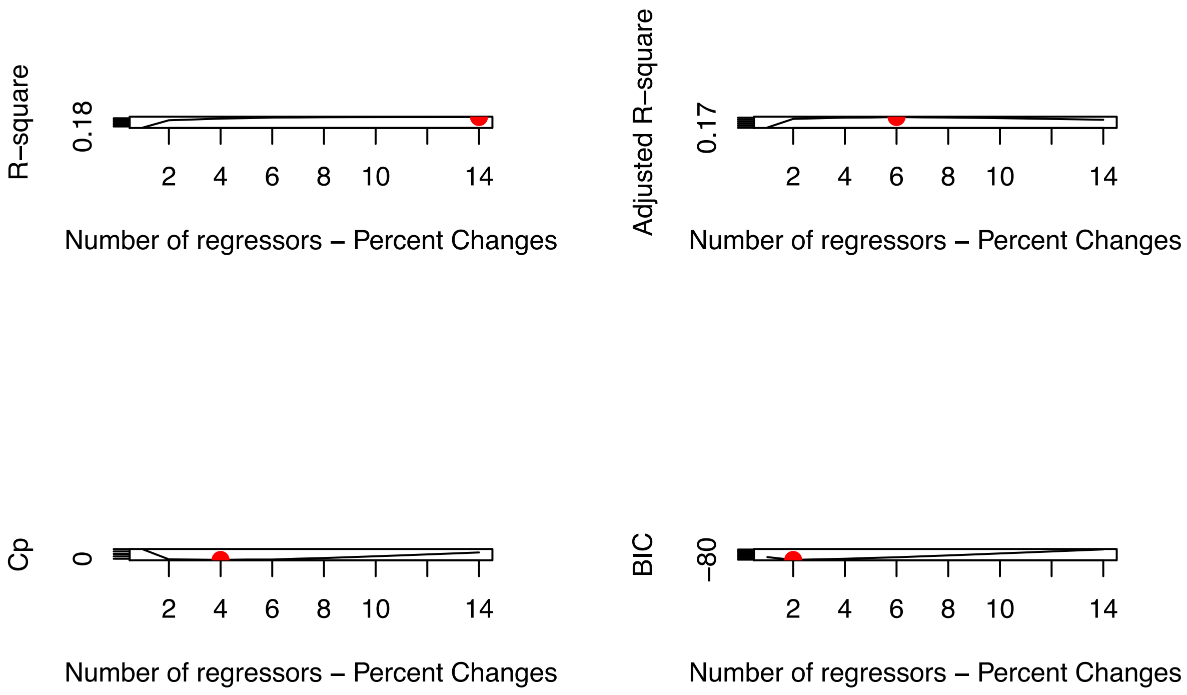
```
regSummary$rsq
```

```
##  [1] 0.1734833 0.2178484 0.2228415 0.2279842 0.2305041 0.2344034 0.2348847
##  [8] 0.2361503 0.2366112 0.2368900 0.2370863 0.2371085 0.2371529 0.2371530
```

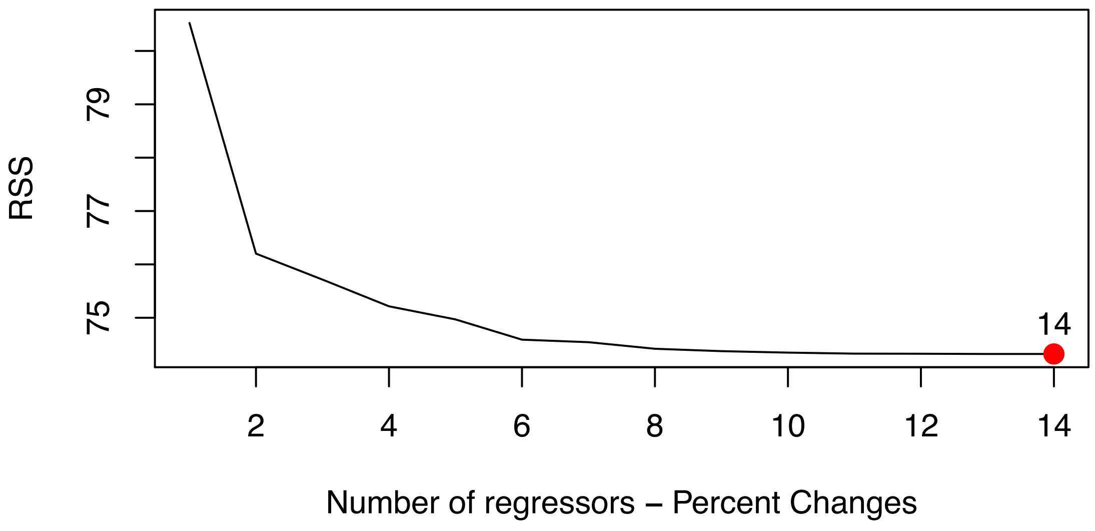
```
regSummary$adjr2
```

```
##  [1] 0.1714273 0.2139474 0.2170129 0.2202447 0.2208371 0.2228327 0.2213599
##  [8] 0.2206799 0.2191734 0.2174725 0.2156780 0.2136949 0.2117246 0.2096984
```

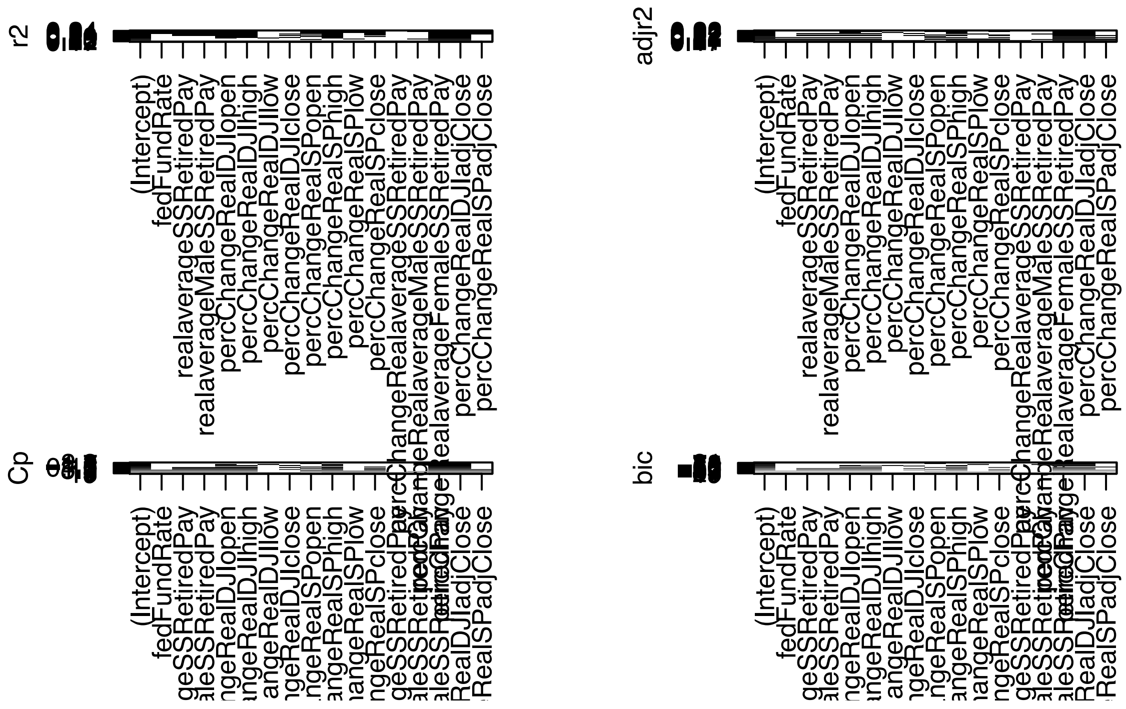
```
par(mfrow=c(2,2))
aRSQ <- which.max(regSummary$rsq)
aARSQ <- which.max(regSummary$adjr2)
aCP <- which.min(regSummary$cp)
aBIC <- which.min(regSummary$bic)
aRSS <- which.min(regSummary$rss)

par(mfrow = c(2, 2))

plot(
  regSummary$rsq,
  xlab = "Number of regressors - Percent Changes",
  ylab = "R-square",
  type = "l"
)
points(
  aRSQ,
  regSummary$rsq[aRSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSQ,
     regSummary$rsq[aRSQ],
     labels = aRSQ,
     pos = 1)

plot(
  regSummary$adjr2,
  xlab = "Number of regressors - Percent Changes",
  ylab = "Adjusted R-square",
  type = "l"
)
points(
  aARSQ,
  regSummary$adjr2[aARSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aARSQ,
     regSummary$adjr2[aARSQ],
     labels = aARSQ,
     pos = 1)

plot(regSummary$cp,
     xlab = "Number of regressors - Percent Changes",
     ylab = "Cp",
     type = "l")
points(
  aCP,
  regSummary$cp[aCP],
  col = "red",
  cex = 2,
  pch = 20
)
text(aCP,
     regSummary$cp[aCP],
     labels = aCP,
     pos = 3)

plot(
  regSummary$bic,
  xlab = "Number of regressors - Percent Changes",
  ylab = "BIC",
  type = "l"
)
points(
  aBIC,
  regSummary$bic[aBIC],
  col = "red",
  cex = 2,
  pch = 20
)
text(aBIC,
     regSummary$bic[aBIC],
     labels = aBIC,
     pos = 3)
```



```r
par(mfrow = c(1, 1))
plot(
  regSummary$rss,
  xlab = "Number of regressors - Percent Changes",
  ylab = "RSS",
  type = "l"
)
points(
  aRSS,
  regSummary$rss[aRSS],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSS,
     regSummary$rss[aRSS],
     labels = aRSS,
     pos = 3)
```



```r
par(mfrow = c(2, 2))
plot(regFitSelect, scale = "r2")
plot(regFitSelect, scale = "adjr2")
plot(regFitSelect, scale = "Cp")
plot(regFitSelect, scale = "bic")
```



```r
# All Data Selection ####
# Exhaustive - All Data
regFitSelect <- regsubsets(
  postotalSSRetired~.,
  data=dfStationary[-1],
  method= 'exhaustive',
  really.big = TRUE,
  nvmax=56)
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 12 linear dependencies found

## Reordering variables and trying again:
```

```
regSummary <- summary(regFitSelect)
```

```
## Warning in log(vr): NaNs produced
```

```
names(regSummary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

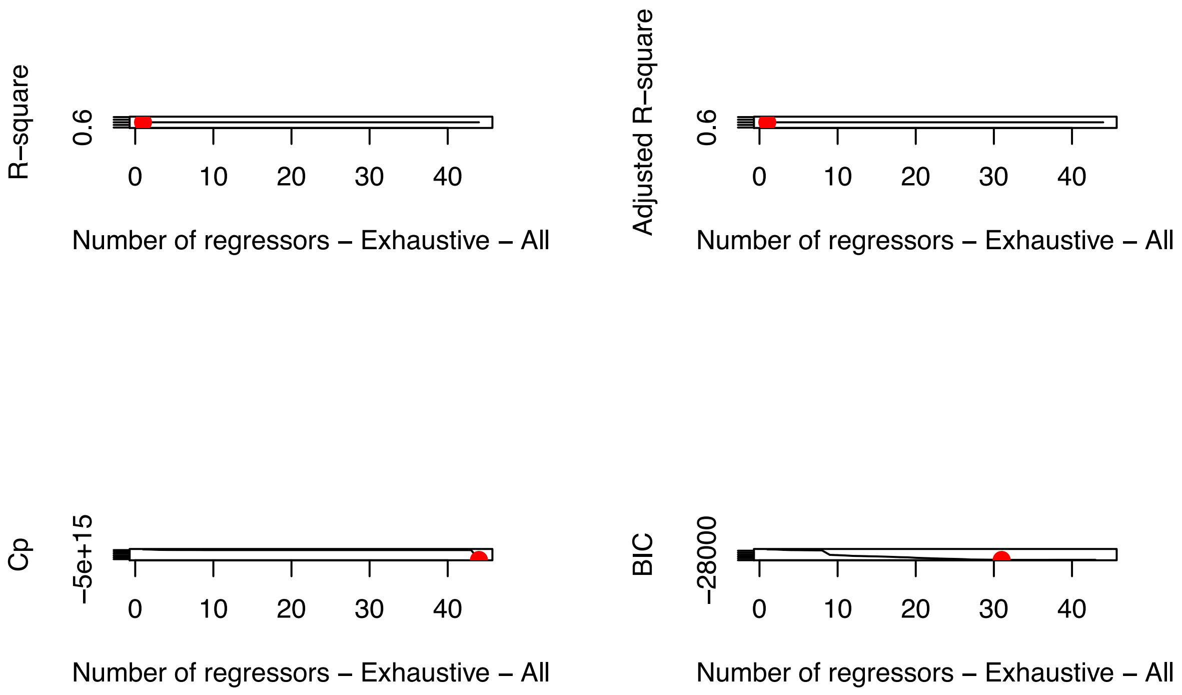
```
regSummary$rsq
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1
```

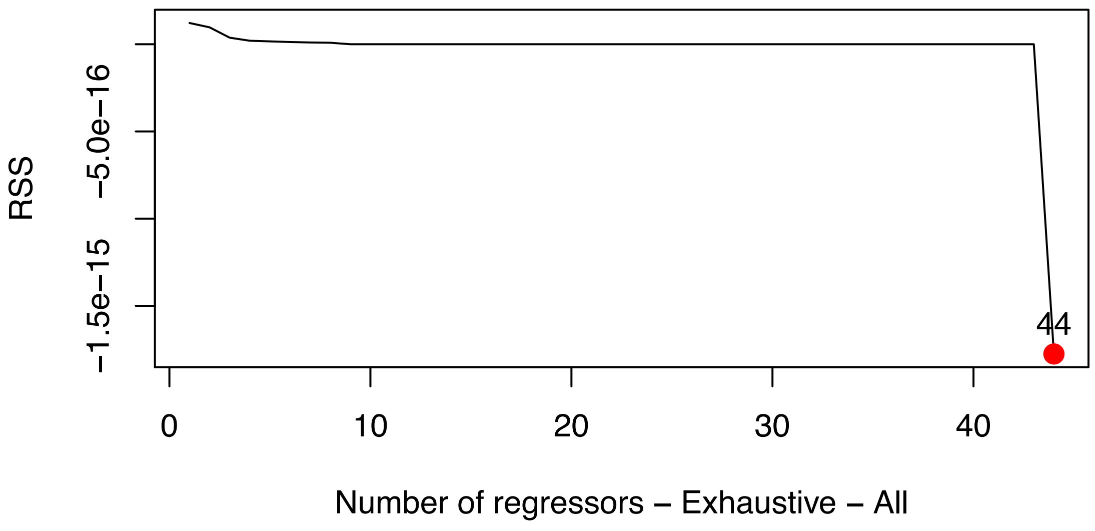
```
regSummary$adjr2
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1
```

```
par(mfrow=c(2,2))
aRSQ <- which.max(regSummary$rsq)
aARSQ <- which.max(regSummary$adjr2)
aCP <- which.min(regSummary$cp)
aBIC <- which.min(regSummary$bic)
aRSS <- which.min(regSummary$rss)

par(mfrow = c(2, 2))

plot(
  regSummary$rsq,
  xlab = "Number of regressors - Exhaustive - All",
  ylab = "R-square",
  type = "l"
)
points(
  aRSQ,
  regSummary$rsq[aRSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSQ,
     regSummary$rsq[aRSQ],
     labels = aRSQ,
     pos = 1)

plot(
  regSummary$adjr2,
  xlab = "Number of regressors - Exhaustive - All",
  ylab = "Adjusted R-square",
  type = "l"
)
points(
  aARSQ,
  regSummary$adjr2[aARSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aARSQ,
     regSummary$adjr2[aARSQ],
     labels = aARSQ,
     pos = 1)

plot(regSummary$cp,
     xlab = "Number of regressors - Exhaustive - All",
     ylab = "Cp",
     type = "l")
points(
  aCP,
  regSummary$cp[aCP],
  col = "red",
  cex = 2,
  pch = 20
)
text(aCP,
     regSummary$cp[aCP],
     labels = aCP,
     pos = 3)

plot(
  regSummary$bic,
  xlab = "Number of regressors - Exhaustive - All",
  ylab = "BIC",
  type = "l"
)
points(
  aBIC,
  regSummary$bic[aBIC],
  col = "red",
  cex = 2,
  pch = 20
)
text(aBIC,
     regSummary$bic[aBIC],
     labels = aBIC,
     pos = 3)
```



```r
par(mfrow = c(1, 1))
plot(
  regSummary$rss,
  xlab = "Number of regressors - Exhaustive - All",
  ylab = "RSS",
  type = "l"
)
points(
  aRSS,
  regSummary$rss[aRSS],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSS,
     regSummary$rss[aRSS],
     labels = aRSS,
     pos = 3)
```



```
par(mfrow = c(2, 2))
plot(regFitSelect, scale = "r2")
```

```
## Warning in log(vr): NaNs produced
```

```
plot(regFitSelect, scale = "adjr2")
```

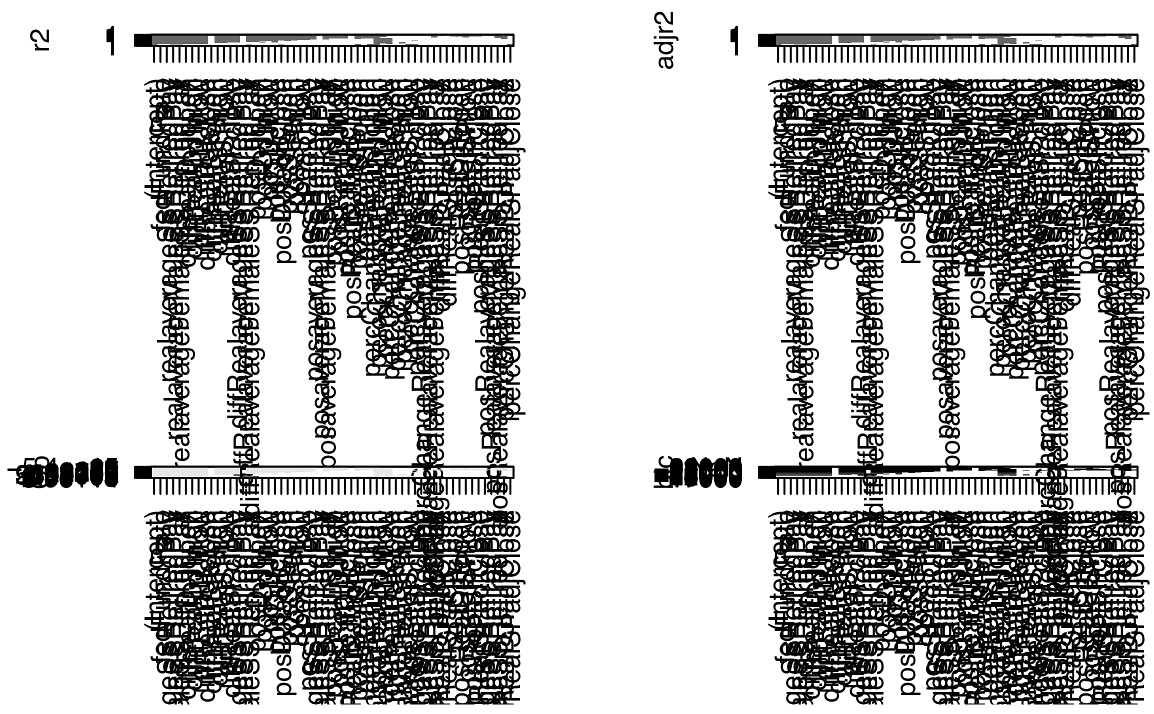
```
## Warning in log(vr): NaNs produced
```

```
plot(regFitSelect, scale = "Cp")
```

```
## Warning in log(vr): NaNs produced
```

```
plot(regFitSelect, scale = "bic")
```

```
## Warning in log(vr): NaNs produced
```



```r
valuesStatExhaustive <- c(names(coef(regFitSelect, id = 31))[-1])
```

```
## Warning in log(vr): NaNs produced
```

```r
# Backward
regFitSelect <- regsubsets(
  postotalSSRetired~.,
  data=dfStationary[-1],
  method= 'backward',
  really.big = TRUE,
  nvmax=55)
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 12 linear dependencies found
```

```
## Reordering variables and trying again:
```

```
## Warning in rval$lopt[] <- rval$vorder[rval$lopt]: number of items to
## replace is not a multiple of replacement length
```

```r
regSummary <- summary(regFitSelect)
names(regSummary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

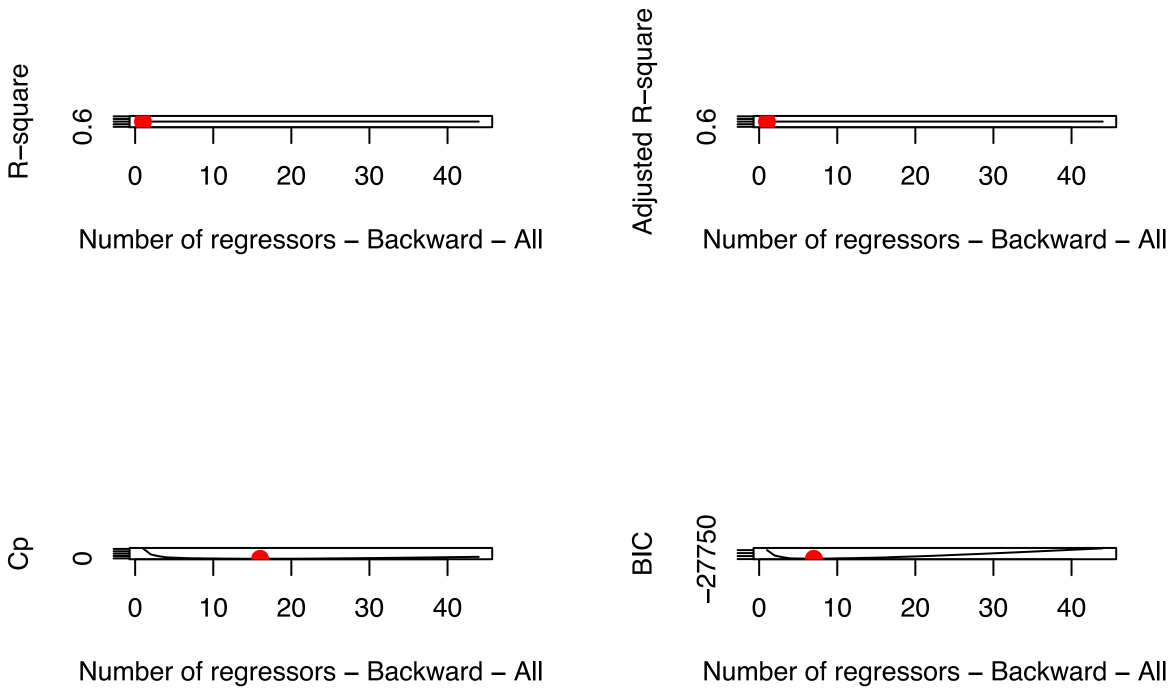
```r
regSummary$rsq
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1
```

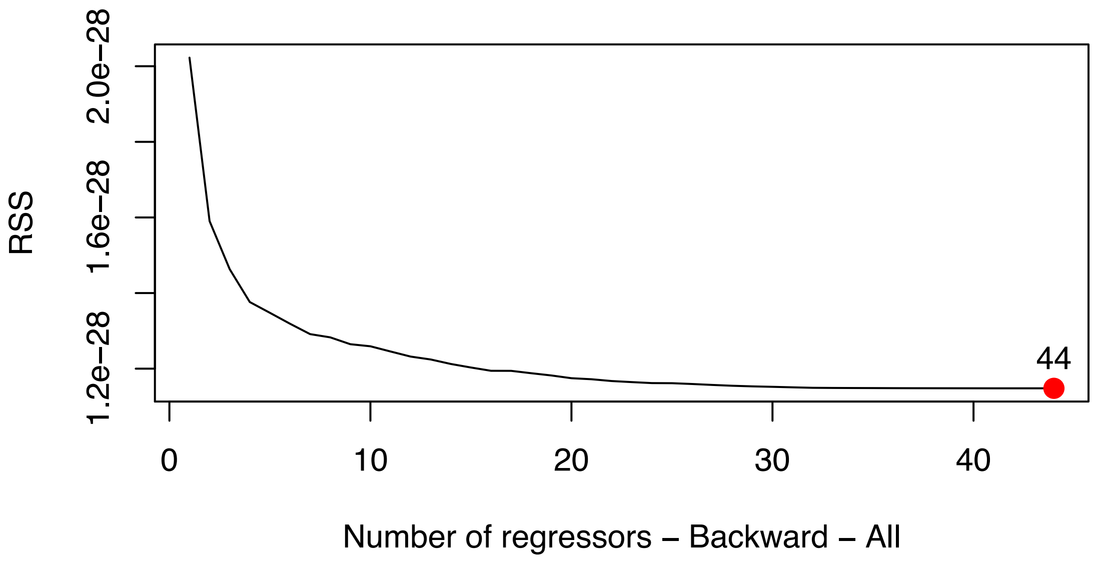
```r
regSummary$adjr2
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1
```

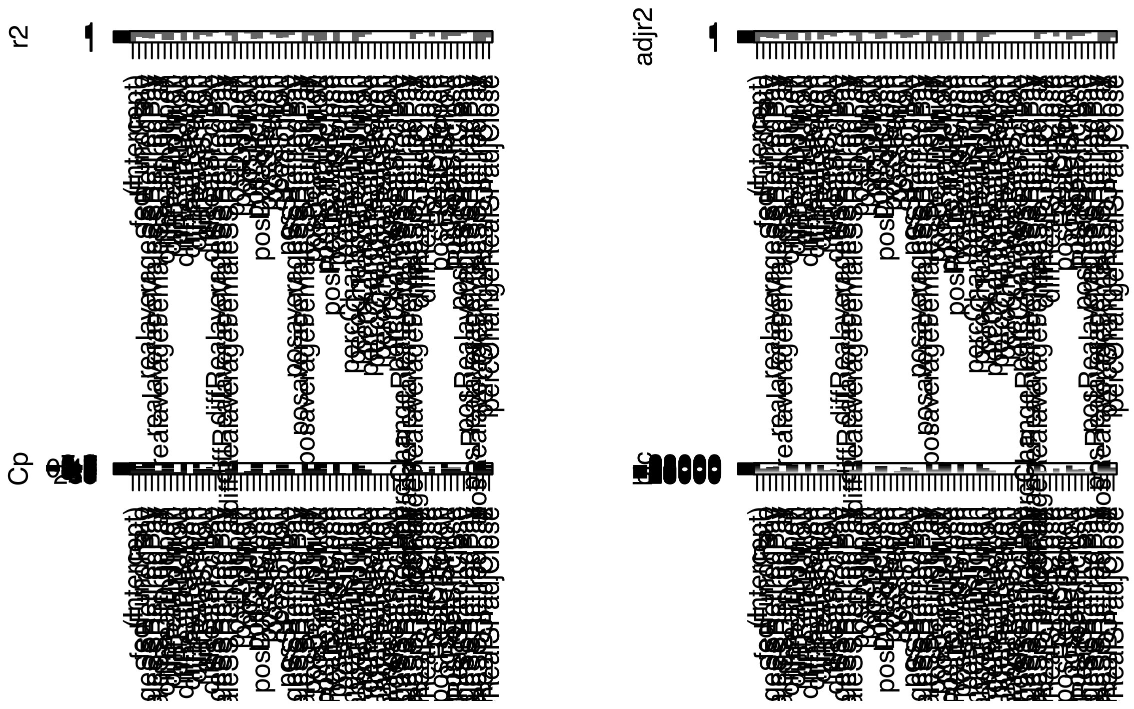
```r
par(mfrow=c(2,2))
aRSQ <- which.max(regSummary$rsq)
aARSQ <- which.max(regSummary$adjr2)
aCP <- which.min(regSummary$cp)
aBIC <- which.min(regSummary$bic)
aRSS <- which.min(regSummary$rss)

par(mfrow = c(2, 2))

plot(
  regSummary$rsq,
  xlab = "Number of regressors - Backward - All",
  ylab = "R-square",
  type = "l"
)
points(
  aRSQ,
  regSummary$rsq[aRSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSQ,
     regSummary$rsq[aRSQ],
     labels = aRSQ,
     pos = 1)

plot(
  regSummary$adjr2,
  xlab = "Number of regressors - Backward - All",
  ylab = "Adjusted R-square",
  type = "l"
)
points(
  aARSQ,
  regSummary$adjr2[aARSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aARSQ,
     regSummary$adjr2[aARSQ],
     labels = aARSQ,
     pos = 1)

plot(regSummary$cp,
     xlab = "Number of regressors - Backward - All",
     ylab = "Cp",
     type = "l")
points(
  aCP,
  regSummary$cp[aCP],
  col = "red",
  cex = 2,
  pch = 20
)
text(aCP,
     regSummary$cp[aCP],
     labels = aCP,
     pos = 3)

plot(
  regSummary$bic,
  xlab = "Number of regressors - Backward - All",
  ylab = "BIC",
  type = "l"
)
points(
  aBIC,
  regSummary$bic[aBIC],
  col = "red",
  cex = 2,
  pch = 20
)
text(aBIC,
     regSummary$bic[aBIC],
     labels = aBIC,
     pos = 3)
```



```
par(mfrow = c(1, 1))
plot(
  regSummary$rss,
  xlab = "Number of regressors - Backward - All",
  ylab = "RSS",
  type = "l"
)
points(
  aRSS,
  regSummary$rss[aRSS],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSS,
     regSummary$rss[aRSS],
     labels = aRSS,
     pos = 3)
```



```r
par(mfrow = c(2, 2))
plot(regFitSelect, scale = "r2")
plot(regFitSelect, scale = "adjr2")
plot(regFitSelect, scale = "Cp")
plot(regFitSelect, scale = "bic")
```



```r
valuesStatBackward <- c(names(coef(regFitSelect, id = 7))[-1])

# Forward
regFitSelect <- regsubsets(
  postotalSSRetired~.,
  data=dfStationary[-1],
  method= 'forward',
  really.big = TRUE,
  nvmax=55)
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 12 linear dependencies found

## Reordering variables and trying again:

## Warning in rval$lopt[] <- rval$vorder[rval$lopt]: number of items to
## replace is not a multiple of replacement length
```

```r
regSummary <- summary(regFitSelect)
names(regSummary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

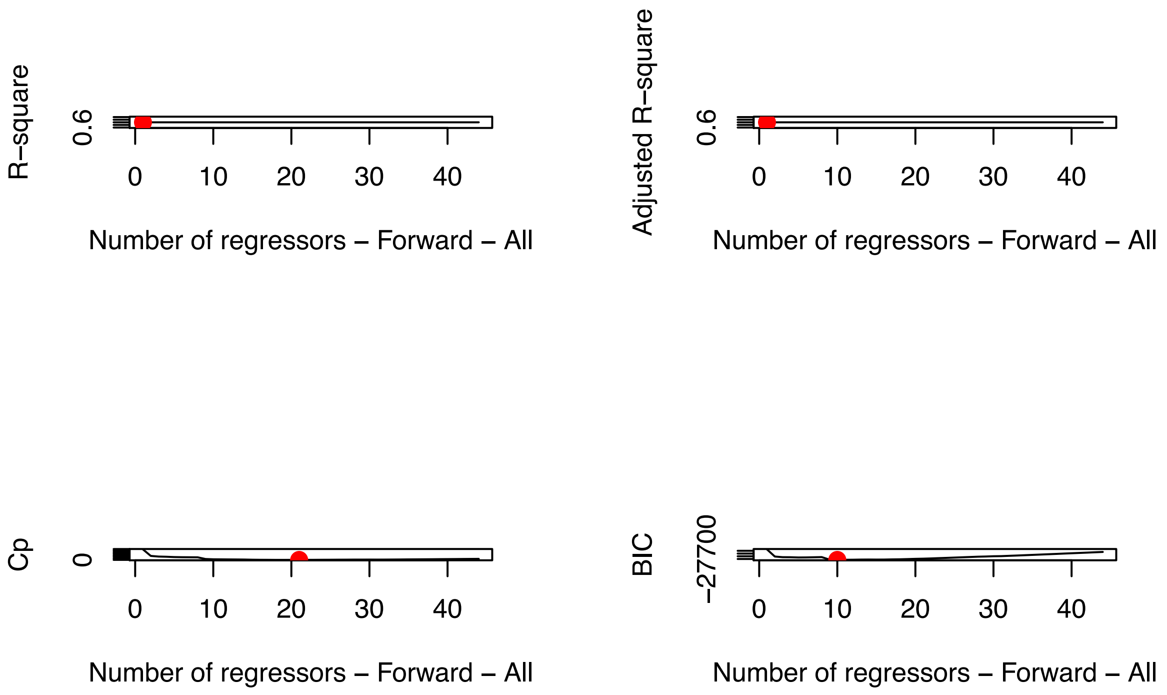
```r
regSummary$rsq
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1
```

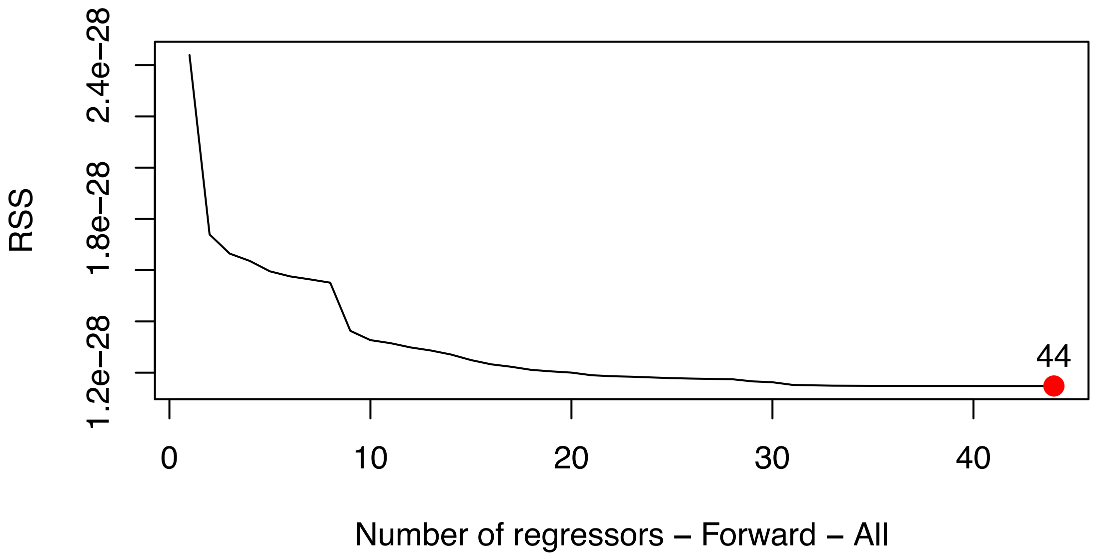
```r
regSummary$adjr2
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [36] 1 1 1 1 1 1 1 1 1
```

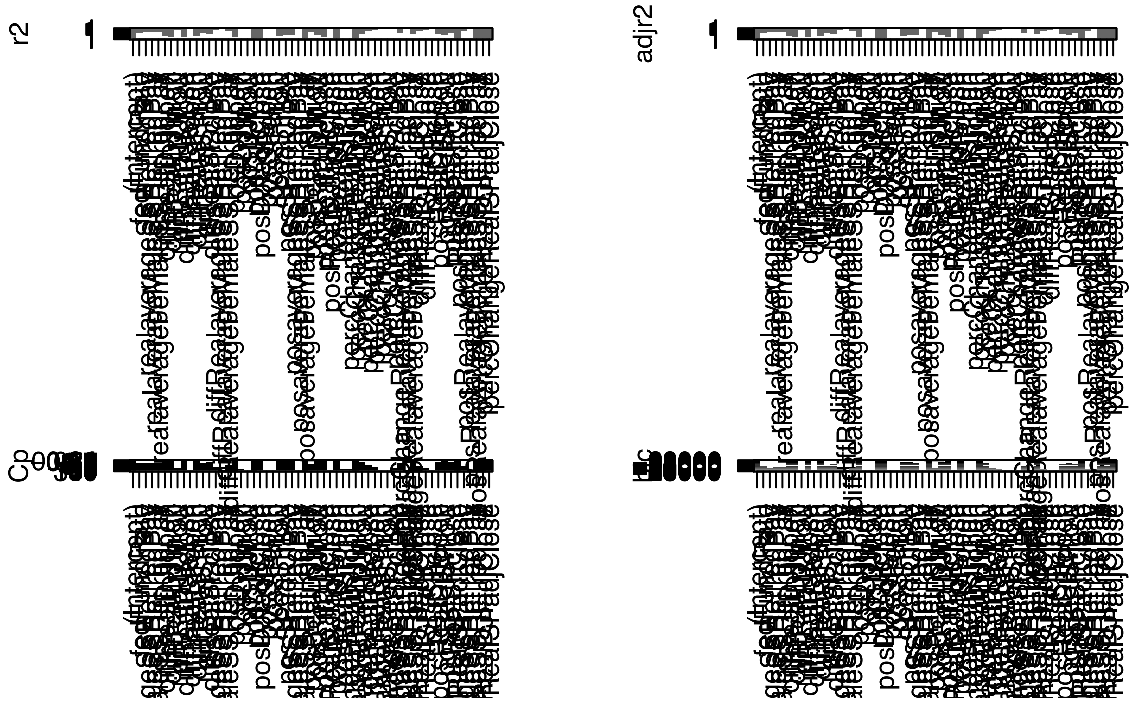
```r
par(mfrow=c(2,2))
aRSQ <- which.max(regSummary$rsq)
aARSQ <- which.max(regSummary$adjr2)
aCP <- which.min(regSummary$cp)
aBIC <- which.min(regSummary$bic)
aRSS <- which.min(regSummary$rss)

par(mfrow = c(2, 2))

plot(
  regSummary$rsq,
  xlab = "Number of regressors - Forward - All",
  ylab = "R-square",
  type = "l"
)
points(
  aRSQ,
  regSummary$rsq[aRSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSQ,
     regSummary$rsq[aRSQ],
     labels = aRSQ,
     pos = 1)

plot(
  regSummary$adjr2,
  xlab = "Number of regressors - Forward - All",
  ylab = "Adjusted R-square",
  type = "l"
)
points(
  aARSQ,
  regSummary$adjr2[aARSQ],
  col = "red",
  cex = 2,
  pch = 20
)
text(aARSQ,
     regSummary$adjr2[aARSQ],
     labels = aARSQ,
     pos = 1)

plot(regSummary$cp,
     xlab = "Number of regressors - Forward - All",
     ylab = "Cp",
     type = "l")
points(
  aCP,
  regSummary$cp[aCP],
  col = "red",
  cex = 2,
  pch = 20
)
text(aCP,
     regSummary$cp[aCP],
     labels = aCP,
     pos = 3)

plot(
  regSummary$bic,
  xlab = "Number of regressors - Forward - All",
  ylab = "BIC",
  type = "l"
)
points(
  aBIC,
  regSummary$bic[aBIC],
  col = "red",
  cex = 2,
  pch = 20
)
text(aBIC,
     regSummary$bic[aBIC],
     labels = aBIC,
     pos = 3)
```



```r
par(mfrow = c(1, 1))
plot(
  regSummary$rss,
  xlab = "Number of regressors - Forward - All",
  ylab = "RSS",
  type = "l"
)
points(
  aRSS,
  regSummary$rss[aRSS],
  col = "red",
  cex = 2,
  pch = 20
)
text(aRSS,
     regSummary$rss[aRSS],
     labels = aRSS,
     pos = 3)
```



```r
par(mfrow = c(2, 2))
plot(regFitSelect, scale = "r2")
plot(regFitSelect, scale = "adjr2")
plot(regFitSelect, scale = "Cp")
plot(regFitSelect, scale = "bic")
```



```r
valuesStatForward <- c(names(coef(regFitSelect, id = 10))[-1])

fmlaForward <- as.formula(paste("postotalSSRetired ~ ", paste(valuesStatForward, collapse= "+")))
fmlaBackward <- as.formula(paste("postotalSSRetired ~ ", paste(valuesStatBackward, collapse= "+")))
fmlaExhaust <- as.formula(paste("postotalSSRetired ~ ", paste(valuesStatExhaustive, collapse= "+")))

# Model ####
# Setting train/test split
```

```
set.seed(1)
trainSample <- sample(1:nrow(dfStationary), round(nrow(dfStationary)/2), replace = F)
trainData <- dfStationary[trainSample,]
testData <- dfStationary[-trainSample,]

trainX <- trainData[,c(1, 3:58)]
trainY <- trainData[,c(1:2)]
testX <- testData[,c(1, 3:58)]
trainY <- testData[,c(1:2)]

# Logistic ####

# Exhaustive Selected model
logFit <- glm(fmlaExhaust,
              family = binomial,
              data = dfStationary)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(logFit, diagnostics=TRUE)
```

```
##
## Call:
## glm(formula = fmlaExhaust, family = binomial, data = dfStationary)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -6.65e-05  -2.10e-08  -2.10e-08   2.10e-08   7.67e-05
##
## Coefficients:
##                                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      -3.740e+02  2.429e+06   0.000    1.000
## fedFundRate                      -1.826e+00  5.610e+04   0.000    1.000
## realaverageSSRetiredPay           4.457e-01  3.568e+03   0.000    1.000
## realaverageMaleSSRetiredPay       3.177e-01  1.549e+03   0.000    1.000
## realaverageFemaleSSRetiredPay    -7.487e-01  1.798e+03   0.000    1.000
## diffRealDJIopen                   8.708e-02  2.692e+02   0.000    1.000
## diffRealDJIhigh                  -9.536e-02  3.648e+02   0.000    1.000
## diffRealDJIlow                   -9.996e-02  2.308e+02   0.000    1.000
## diffRealDJIclose                  1.026e-01  2.825e+02   0.000    1.000
## diffRealSPhigh                   -5.578e-02  9.441e+02   0.000    1.000
## diffRealSPlow                     9.387e-01  2.054e+03   0.000    1.000
## diffRealSPclose                  -8.024e-01  1.307e+03  -0.001    1.000
## diffRealaverageSSRetiredPay      -2.077e+00  1.769e+03  -0.001    0.999
## diffRealaverageFemaleSSRetiredPay  2.315e+00  2.784e+03   0.001    0.999
## posDJIopen                       -8.073e+00  3.913e+05   0.000    1.000
## posDJIhigh                        8.907e+01  1.074e+05   0.001    0.999
## posDJIlow                        -7.844e+00  1.396e+05   0.000    1.000
## posDJIclose                      -8.014e+01  1.063e+05  -0.001    0.999
## posDJIadjClose                    4.842e+01  8.540e+04   0.001    1.000
## posSPopen                        -9.421e+00  1.519e+05   0.000    1.000
## posSPhigh                        -5.175e+00  7.073e+04   0.000    1.000
## posSPlow                          1.310e+01  3.304e+05   0.000    1.000
## posSPadjClose                      -5.501e+01  8.350e+04  -0.001    0.999
## posaverageSSRetiredPay              1.374e+01  1.007e+05   0.000    1.000
## posaverageFemaleSSRetiredPay      -6.536e+01  2.678e+05   0.000    1.000
## posRealDJIhigh                     -1.473e+00  8.302e+04   0.000    1.000
## posRealDJIlow                      -1.582e+01  1.677e+05   0.000    1.000
## posRealDJIclose                     4.006e+01  6.266e+04   0.001    0.999
## posRealSPopen                       1.578e+02  5.881e+04   0.003    0.998
## posRealSPhigh                       9.575e+01  3.884e+04   0.002    0.998
## percChangeRealDJIopen              -1.179e+03  4.426e+06   0.000    1.000
## percChangeRealDJIhigh               1.768e+03  5.401e+06   0.000    1.000
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5.4568e+02  on 403  degrees of freedom
## Residual deviance: 4.0544e-08  on 372  degrees of freedom
## AIC: 64
##
## Number of Fisher Scoring iterations: 25
```

```r
confint(logFit)
```

```
## Waiting for profiling to be done...

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

##                                         2.5 %       97.5 %
## (Intercept)                                NA 252002.49723
## fedFundRate                     -3.983147e+03   4703.37124
## realaverageSSRetiredPay         -2.546648e+02    268.77575
## realaverageMaleSSRetiredPay     -8.577694e+01     77.87345
## realaverageFemaleSSRetiredPay   -9.572672e+01     76.17133
## diffRealDJIopen                 -9.242949e+00      8.84281
## diffRealDJIhigh                 -1.829923e+01     16.61109
## diffRealDJIlow                  -1.504885e+01     16.18882
## diffRealDJIclose                -1.549278e+01     15.26941
## diffRealSPhigh                  -8.653673e+01     94.17654
## diffRealSPlow                   -9.404891e+01     84.85045
## diffRealSPclose                 -5.913237e+01     52.72950
## diffRealaverageSSRetiredPay     -6.627266e+01     70.74603
## diffRealaverageFemaleSSRetiredPay -1.311546e+02    162.12368
## posDJIopen                      -2.341307e+04  26188.87448
## posDJIhigh                      -3.817369e+03   4672.39169
## posDJIlow                       -6.241243e+03   5712.80741
## posDJIclose                     -8.751598e+03   8591.31205
## posDJIadjClose                  -4.072687e+03   4000.15889
## posSPopen                       -6.995852e+03   8452.16400
## posSPhigh                       -3.335126e+03   3324.77570
## posSPlow                        -1.239607e+04  15042.40911
## posSPadjClose                   -3.586792e+03   3898.06575
## posaverageSSRetiredPay          -3.065634e+03   3148.67282
## posaverageFemaleSSRetiredPay    -1.515226e+04  16011.10232
## posRealDJIhigh                  -6.244430e+03   5933.91934
## posRealDJIlow                   -8.292697e+03   6536.94787
## posRealDJIclose                 -5.699962e+03   6294.56015
## posRealSPopen                   -1.457712e+03   1784.94750
## posRealSPhigh                   -1.023044e+03   1260.88953
## percChangeRealDJIopen           -1.509047e+05 150183.97188
## percChangeRealDJIhigh           -2.276793e+05 227418.56595
```

```
logProbs <- predict(logFit, type = 'response')
logProbs[1:10]
```

```
##            2            3            4            5            6
## 2.220446e-16 1.000000e+00 2.220446e-16 2.220446e-16 2.220446e-16
##            7            8            9           10           11
## 1.000000e+00 4.891675e-10 2.220446e-16 2.220446e-16 1.000000e+00
```

```
logPred <- rep(NA, dim(dfStationary)[2])
logPred[logProbs > 0.5] <- 1
logPred[logProbs < 0.5] = 0

table(logPred)
```

```
## logPred
##   0   1
## 240 164
```

```r
table(logPred, dfStationary[,2])
```

```
##
## logPred   0   1
##       0 240   0
##       1   0 164
```

```r
mean(logPred == dfStationary[,2])
```

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

```r
# Testing Prediction
train <- subset(dfStationary, dfStationary$date < as.Date('2010-04-08'))
test3rdQuart <- subset(dfStationary, dfStationary$date >= as.Date('2010-04-08'))

logFit1 <- glm(fmlaExhaust,
               family = binomial,
               data = train)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
logProbs1 <- predict(logFit1, test3rdQuart, type = 'response') # setting prediction for the testing set

logPred1 <- rep(NA, dim(train)[2])
logPred1[logProbs1 >= 0.5] = 1
logPred1[logProbs1 < 0.5] = 0

table(logPred1, test3rdQuart$postotalSSRetired)
```

```
##
## logPred1  0  1
##        0 58  5
##        1  1 37
```

```r
mean(logPred1 == test3rdQuart$postotalSSRetired)
```

```
## [1] 0.9405941
```

Predicition accuracy is approximately 94% with a train/test split at the 3rd quartile mark of the dates. Positive class prediction: 37/5; 88% Negative Class prediction: 58/1; 98.3%

```r
# Backard Selected model
logFit <- glm(fmlaBackward,
              family = binomial,
              data = dfStationary)
summary(logFit, diagnostics=TRUE)
```

```
##
## Call:
## glm(formula = fmlaBackward, family = binomial, data = dfStationary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.87722  -0.16582  -0.06502   0.14678   2.93502
##
## Coefficients:
```

```
##                                  Estimate Std. Error z value Pr(>|z|)    
## (Intercept)                       -3.9626     0.8825  -4.490 7.12e-06 ***
## posDJIopen                         1.0262     1.2013   0.854 0.392953    
## posaverageFemaleSSRetiredPay      -2.3635     1.2734  -1.856 0.063449 .  
## posRealDJIlow                      2.6609     0.7858   3.386 0.000708 ***
## posRealSPopen                      7.6607     0.8839   8.667  < 2e-16 ***
## percChangeRealDJIhigh             31.9746    16.1912   1.975 0.048290 *  
## posRealaverageFemaleSSRetiredPay  -1.3853     0.8903  -1.556 0.119728    
## percChangeRealDJIadjClose         -1.6774    11.0572  -0.152 0.879418    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 545.68  on 403  degrees of freedom
## Residual deviance: 109.01  on 396  degrees of freedom
## AIC: 125.01
## 
## Number of Fisher Scoring iterations: 7
```

```r
confint(logFit)
```

```
## Waiting for profiling to be done...
```

```
##                                       2.5 %     97.5 %
## (Intercept)                       -6.017531 -2.4528686
## posDJIopen                        -1.605622  3.2647653
## posaverageFemaleSSRetiredPay      -4.803546  0.3041155
## posRealDJIlow                      1.316928  4.5526627
## posRealSPopen                      6.175544  9.7417619
## percChangeRealDJIhigh              1.004724 64.9354675
## posRealaverageFemaleSSRetiredPay  -3.220072  0.3118396
## percChangeRealDJIadjClose        -23.719774 19.5259955
```

```r
logProbs <- predict(logFit, type = 'response')
logProbs[1:10]
```

```
##           2           3           4           5           6           7 
## 0.004460318 0.974363975 0.011132525 0.002320106 0.001827138 0.996462622 
##           8           9          10          11 
## 0.014180239 0.012967488 0.002486971 0.955606920
```

```r
logPred <- rep(0, dim(dfStationary)[2])
logPred[logProbs >= 0.5] <- 1
logPred[logProbs < 0.5] <- 0

table(logPred)
```

```
## logPred
##   0   1 
## 235 169
```

```r
table(logPred, dfStationary[,2])
```

```
##        
## logPred   0   1
##       0 229   6
##       1  11 158
```

```
mean(logPred == dfStationary[,2])
```

```
## [1] 0.9579208
```

```
# Testing Prediction
train <- subset(dfStationary, dfStationary$date < as.Date('2010-04-08'))
test3rdQuart <- subset(dfStationary, dfStationary$date >= as.Date('2010-04-08'))

logFit1 <- glm(fmlaBackward,
               family = binomial,
               data = train)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
logProbs1 <- predict(logFit1, test3rdQuart, type = 'response') # setting prediction for the testing set

logPred1 <- rep(NA, dim(train)[2])
logPred1[logProbs1 >= 0.5] = 1
logPred1[logProbs1 < 0.5] = 0

table(logPred1, test3rdQuart$postotalSSRetired)
```

```
##
## logPred1  0  1
##        0 53  3
##        1  6 39
```

```
mean(logPred1 == test3rdQuart$postotalSSRetired)
```

```
## [1] 0.9108911
```

Predicition accuracy is approximately 91% with a train/test split at the 3rd quartile mark of the dates. Positive class prediction: 39/3; 92.8% Negative Class prediction: 53/6; 89.8%

```
# Forward Selected model - Best Model
logFitForwardAll <- glm(fmlaForward,
              family = binomial,
              data = dfStationary)
summary(logFit, diagnostics=TRUE)
```

```
##
## Call:
## glm(formula = fmlaBackward, family = binomial, data = dfStationary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.87722  -0.16582  -0.06502   0.14678   2.93502
##
## Coefficients:
##                                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)                      -3.9626     0.8825  -4.490 7.12e-06 ***
## posDJIopen                        1.0262     1.2013   0.854 0.392953
## posaverageFemaleSSRetiredPay     -2.3635     1.2734  -1.856 0.063449 .
## posRealDJIlow                     2.6609     0.7858   3.386 0.000708 ***
## posRealSPopen                     7.6607     0.8839   8.667  < 2e-16 ***
## percChangeRealDJIhigh            31.9746    16.1912   1.975 0.048290 *
## posRealaverageFemaleSSRetiredPay  -1.3853     0.8903  -1.556 0.119728
## percChangeRealDJIadjClose         -1.6774    11.0572  -0.152 0.879418
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 545.68  on 403  degrees of freedom
## Residual deviance: 109.01  on 396  degrees of freedom
## AIC: 125.01
##
## Number of Fisher Scoring iterations: 7
```

```
confint(logFit)
```

```
## Waiting for profiling to be done...

##                                       2.5 %     97.5 %
## (Intercept)                       -6.017531 -2.4528686
## posDJIopen                        -1.605622  3.2647653
## posaverageFemaleSSRetiredPay      -4.803546  0.3041155
## posRealDJIlow                      1.316928  4.5526627
## posRealSPopen                      6.175544  9.7417619
## percChangeRealDJIhigh              1.004724 64.9354675
## posRealaverageFemaleSSRetiredPay  -3.220072  0.3118396
## percChangeRealDJIadjClose        -23.719774 19.5259955
```

```
logProbs <- predict(logFit, type = 'response')
logProbs[1:10]
```

```
##           2           3           4           5           6           7
## 0.004460318 0.974363975 0.011132525 0.002320106 0.001827138 0.996462622
##           8           9          10          11
## 0.014180239 0.012967488 0.002486971 0.955606920
```

```
logPred <- rep(NA, dim(dfStationary)[2])
logPred[logProbs > 0.5] <- 1
logPred[logProbs < 0.5] = 0


table(logPred)
```

```
## logPred
##   0   1
## 235 169
```

```
table(logPred, dfStationary[,2])
```

```
##
## logPred   0   1
##       0 229   6
##       1  11 158
```

```
mean(logPred == dfStationary[,2])
```

```
## [1] 0.9579208
```

```
# Testing Prediction
train <- subset(dfStationary, dfStationary$date < as.Date('2010-04-08'))
test3rdQuart <- subset(dfStationary, dfStationary$date >= as.Date('2010-04-08'))

logFit1 <- glm(fmlaForward,
               family = binomial,
               data = train)
logProbs1 <- predict(logFit1, test3rdQuart, type = 'response') # setting prediction for the testing set

logPred1 <- rep(NA, dim(train)[2])
logPred1[logProbs1 >= 0.5] = 1
logPred1[logProbs1 < 0.5] = 0

table(logPred1, test3rdQuart$postotalSSRetired)
```

```
##
## logPred1  0  1
##        0 54  2
##        1  5 40
```

```
mean(logPred1 == test3rdQuart$postotalSSRetired)
```

```
## [1] 0.9306931
```

Predicition accuracy is approximately 93% with a train/test split at the 3rd quartile mark of the dates. Positive class prediction: 40/2; 95% Negative Class prediction: 54/5; 91.5% This is the best model.

```
# Check only those with good statistical significance
logFit <- glm(postotalSSRetired ~ posDJIclose + posDJIadjClose + posRealSPopen  + posRealSPadjClose + po
              family = binomial,
              data = dfStationary)
summary(logFit, diagnostics=TRUE)
```

```
##
## Call:
## glm(formula = postotalSSRetired ~ posDJIclose + posDJIadjClose +
##     posRealSPopen + posRealSPadjClose + posRealaverageFemaleSSRetiredPay,
##     family = binomial, data = dfStationary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -3.06535  -0.13360  -0.05683   0.13530   2.48559
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                       -6.4278     1.0182  -6.313 2.74e-10 ***
## posDJIclose                        1.5484     0.9552   1.621  0.10501
## posDJIadjClose                     1.7133     0.6531   2.623  0.00871 **
## posRealSPopen                      7.7315     0.9048   8.545  < 2e-16 ***
## posRealSPadjClose                  1.8369     0.9750   1.884  0.05956 .
## posRealaverageFemaleSSRetiredPay  -1.0678     0.6117  -1.746  0.08090 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 545.68  on 403  degrees of freedom
## Residual deviance: 104.59  on 398  degrees of freedom
## AIC: 116.59
##
## Number of Fisher Scoring iterations: 7
```

```
confint(logFit)
```

```
## Waiting for profiling to be done...

##                                    2.5 %     97.5 %
## (Intercept)                   -8.7614151 -4.6808299
## posDJIclose                   -0.1244147  3.6921985
## posDJIadjClose                 0.4920897  3.0838235
## posRealSPopen                  6.2167780  9.8785185
## posRealSPadjClose              0.1224186  4.0102672
## posRealaverageFemaleSSRetiredPay -2.3042923  0.1182437
```

```
logProbs <- predict(logFit, type = 'response')
logProbs[1:10]
```

```
##            2            3            4            5            6
## 0.0016134028 0.9897014801 0.0016134028 0.0005552172 0.0834023010
##            7            8            9           10           11
## 0.9952008178 0.0088844333 0.0016134028 0.0034749324 0.9952008178
```

```
logPred <- rep(0, dim(dfStationary)[2])
logPred[logProbs > 0.5] <- 1
table(logPred)
```

```
## logPred
##   0   1
##  34 169
```

```
table(logPred, dfStationary[,2])
```

```
##
## logPred   0   1
##       0  33   1
##       1  11 158
```

```
mean(logPred == dfStationary[,2])
```

```
## [1] NA
```

```
# Testing Prediction
train <- subset(dfStationary, dfStationary$date < as.Date('2010-04-08'))
test3rdQuart <- subset(dfStationary, dfStationary$date >= as.Date('2010-04-08'))

logFit1 <- glm(postotalSSRetired ~ posDJIclose + posDJIadjClose + posRealSPopen  + posRealSPadjClose + p
               family = binomial,
               data = train)
logProbs1 <- predict(logFit1, test3rdQuart, type = 'response') # setting prediction for the testing set

logPred1 <- rep(NA, dim(train)[2])
logPred1[logProbs1 >= 0.5] = 1
logPred1[logProbs1 < 0.5] = 0

table(logPred1, test3rdQuart$postotalSSRetired)
```

```
##
## logPred1  0  1
##        0 54  2
##        1  5 40
```

```
mean(logPred1 == test3rdQuart$postotalSSRetired)
```

```
## [1] 0.9306931
```

Predicition accuracy is approximately 93% with a train/test split at the 3rd quartile mark of the dates. Positive class prediction: 40/2; 95% Negative Class prediction: 54/5; 91.5% There is no change, but we have reduced the number of regressors by half, from 10 to 5.

```
# Subset this further by selecting only those with statistical significance from this model
logFit <- glm(postotalSSRetired ~  posDJIadjClose + posRealSPopen  + posRealSPadjClose + posRealaverage
              family = binomial,
              data = dfStationary)
summary(logFit, diagnostics=TRUE)
```

```
##
## Call:
## glm(formula = postotalSSRetired ~ posDJIadjClose + posRealSPopen +
##     posRealSPadjClose + posRealaverageFemaleSSRetiredPay, family = binomial,
##     data = dfStationary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.95006  -0.14982  -0.06188   0.16107   2.60136
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                       -6.2573     0.9852  -6.351 2.14e-10 ***
## posDJIadjClose                     1.9282     0.6514   2.960 0.003077 **
## posRealSPopen                      7.6875     0.8767   8.768  < 2e-16 ***
## posRealSPadjClose                  2.9082     0.7852   3.704 0.000212 ***
## posRealaverageFemaleSSRetiredPay  -1.1351     0.6097  -1.862 0.062626 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 545.68  on 403  degrees of freedom
## Residual deviance: 107.82  on 399  degrees of freedom
## AIC: 117.82
##
## Number of Fisher Scoring iterations: 7
```

```
confint(logFit)
```

```
## Waiting for profiling to be done...
```

```
##                                       2.5 %      97.5 %
## (Intercept)                      -8.4911635 -4.55434311
## posDJIadjClose                    0.7088555  3.29449737
## posRealSPopen                     6.2092825  9.74742204
## posRealSPadjClose                 1.5701719  4.80067290
## posRealaverageFemaleSSRetiredPay -2.3670651  0.04756246
```

```
logProbs <- predict(logFit, type = 'response')
logProbs[1:10]
```

```
##            2            3            4            5            6
## 0.0019128306 0.9663798429 0.0019128306 0.0006155424 0.0720274306
##            7            8            9           10           11
## 0.9941271498 0.0130086210 0.0019128306 0.0111603749 0.9941271498
```

```
logPred <- rep(0, dim(dfStationary)[2])
logPred[logProbs > 0.5] <- 1
table(logPred)
```

```
## logPred
##   0   1
##  34 169
```

```
table(logPred, dfStationary[,2])
```

```
##
## logPred   0   1
##       0  33   1
##       1  11 158
```

```
mean(logPred == dfStationary[,2])
```

```
## [1] NA
```

```
# Testing Prediction
train <- subset(dfStationary, dfStationary$date < as.Date('2010-04-08'))
test3rdQuart <- subset(dfStationary, dfStationary$date >= as.Date('2010-04-08'))

logFit1 <- glm(postotalSSRetired ~ posDJIadjClose + posRealSPopen  + posRealSPadjClose + posRealaverageF
               family = binomial,
               data = train)
logProbs1 <- predict(logFit1, test3rdQuart, type = 'response') # setting prediction for the testing set

logPred1 <- rep(NA, dim(train)[2])
logPred1[logProbs1 >= 0.5] = 1
logPred1[logProbs1 < 0.5] = 0

table(logPred1, test3rdQuart$postotalSSRetired)
```

```
##
## logPred1  0  1
##        0 54  2
##        1  5 40
```

```
mean(logPred1 == test3rdQuart$postotalSSRetired)
```

```
## [1] 0.9306931
```

Predicition accuracy is approximately 93% with a train/test split at the 3rd quartile mark of the dates. Positive class prediction: 40/2; 95% Negative Class prediction: 54/5; 91.5% There is no change, but we have reduced the number of regressors from 5 to 4.

```
# Subset once more based on those with statistical significance better than 0.05
logFitForwardMinimal <- glm(postotalSSRetired ~  posDJIadjClose + posRealSPopen  + posRealSPadjClose,
```

```r
               family = binomial,
               data = dfStationary)
summary(logFit, diagnostics=TRUE)
```

```
##
## Call:
## glm(formula = postotalSSRetired ~ posDJIadjClose + posRealSPopen +
##     posRealSPadjClose + posRealaverageFemaleSSRetiredPay, family = binomial,
##     data = dfStationary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.95006  -0.14982  -0.06188   0.16107   2.60136
##
## Coefficients:
##                                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)                       -6.2573     0.9852  -6.351 2.14e-10 ***
## posDJIadjClose                     1.9282     0.6514   2.960 0.003077 **
## posRealSPopen                      7.6875     0.8767   8.768  < 2e-16 ***
## posRealSPadjClose                  2.9082     0.7852   3.704 0.000212 ***
## posRealaverageFemaleSSRetiredPay  -1.1351     0.6097  -1.862 0.062626 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 545.68  on 403  degrees of freedom
## Residual deviance: 107.82  on 399  degrees of freedom
## AIC: 117.82
##
## Number of Fisher Scoring iterations: 7
```

```r
confint(logFit)
```

```
## Waiting for profiling to be done...
```

```
##                                       2.5 %      97.5 %
## (Intercept)                      -8.4911635 -4.55434311
## posDJIadjClose                    0.7088555  3.29449737
## posRealSPopen                     6.2092825  9.74742204
## posRealSPadjClose                 1.5701719  4.80067290
## posRealaverageFemaleSSRetiredPay -2.3670651  0.04756246
```

```r
logProbs <- predict(logFit, type = 'response')
logProbs[1:10]
```

```
##            2            3            4            5            6
## 0.0019128306 0.9663798429 0.0019128306 0.0006155424 0.0720274306
##            7            8            9           10           11
## 0.9941271498 0.0130086210 0.0019128306 0.0111603749 0.9941271498
```

```r
logPred <- rep(0, dim(dfStationary)[2])
logPred[logProbs > 0.5] <- 1
table(logPred)
```

```
## logPred
##   0   1
##  34 169
```

```
table(logPred, dfStationary[,2])
```

```
##
## logPred   0   1
##       0  33   1
##       1  11 158
```

```
mean(logPred == dfStationary[,2])
```

```
## [1] NA
```

```
# Testing Prediction
train <- subset(dfStationary, dfStationary$date < as.Date('2010-04-08'))
test3rdQuart <- subset(dfStationary, dfStationary$date >= as.Date('2010-04-08'))

logFit1 <- glm(postotalSSRetired ~ posDJIadjClose + posRealSPopen  + posRealSPadjClose,
               family = binomial,
               data = train)
logProbs1 <- predict(logFit1, test3rdQuart, type = 'response') # setting prediction for the testing set

logPred1 <- rep(NA, dim(train)[2])
logPred1[logProbs1 >= 0.5] = 1
logPred1[logProbs1 < 0.5] = 0

table(logPred1, test3rdQuart$postotalSSRetired)
```

```
##
## logPred1  0  1
##        0 54  2
##        1  5 40
```

```
mean(logPred1 == test3rdQuart$postotalSSRetired)
```

```
## [1] 0.9306931
```

Predicition accuracy is approximately 93% with a train/test split at the 3rd quartile mark of the dates. Positive class prediction: 40/2; 95% Negative Class prediction: 54/5; 91.5% There is no change, but we have reduced the number of regressors from 4 to 3. All factors are statisticall siginificant.

```
predForm <- as.formula(postotalSSRetired ~ posDJIadjClose + posRealSPopen  + posRealSPadjClose)
```

Switch up the train/test split to account for about 80% of the data

```
train80 <- subset(dfStationary, dfStationary$date < as.Date(dfStationary$date[round(nrow(dfStationary) 
test20 <- subset(dfStationary, dfStationary$date >= as.Date(dfStationary$date[round(nrow(dfStationary) 

logFit <- glm(predForm,
              family = binomial,
              data = dfStationary)
summary(logFit, diagnostics=TRUE)
```

```
##
## Call:
## glm(formula = predForm, family = binomial, data = dfStationary)
##
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -2.7339  -0.1139  -0.0556   0.2196   2.7439
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -6.4717     0.9659  -6.700 2.08e-11 ***
## posDJIadjClose      1.4368     0.5820   2.469 0.013560 *
## posRealSPopen       7.4540     0.8341   8.936  < 2e-16 ***
## posRealSPadjClose   2.7306     0.7696   3.548 0.000388 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 545.68  on 403  degrees of freedom
## Residual deviance: 111.36  on 400  degrees of freedom
## AIC: 119.36
##
## Number of Fisher Scoring iterations: 7
```

```r
confint(logFit)
```

```
## Waiting for profiling to be done...
```

```
##                        2.5 %    97.5 %
## (Intercept)       -8.6700560 -4.804306
## posDJIadjClose     0.3477433  2.669307
## posRealSPopen      6.0508612  9.436552
## posRealSPadjClose  1.4244555  4.600410
```

```r
logProbs <- predict(logFit, type = 'response')
logProbs[1:10]
```

```
##           2           3           4           5           6           7
## 0.001544224 0.918271072 0.001544224 0.001544224 0.090771223 0.994232497
##           8           9          10          11
## 0.006464772 0.001544224 0.023179357 0.994232497
```

```r
logPred <- rep(0, dim(dfStationary)[2])
logPred[logProbs > 0.5] <- 1
table(logPred)
```

```
## logPred
##   0   1
##  34 169
```

```r
table(logPred, dfStationary[,2])
```

```
##
## logPred   0   1
##       0  33   1
##       1  11 158
```

```r
mean(logPred == dfStationary[,2])
```

```
## [1] NA
```

```
# Testing Prediction ####
logFit1 <- glm(predForm,
               family = binomial,
               data = train80)
logProbs1 <- predict(logFit1, test20, type = 'response') # setting prediction for the testing set FROM

logPred1 <- rep(NA, dim(train80)[2])
logPred1[logProbs1 >= 0.5] = 1
logPred1[logProbs1 < 0.5] = 0

table(logPred1, test20$postotalSSRetired)
```

```
##
## logPred1  0  1
##        0 43  1
##        1  3 35
```

```
mean(logPred1 == test20$postotalSSRetired)
```

```
## [1] 0.9512195
```

Predicition accuracy is approximately 95% with a train/test split of 80/20. Positive class prediction: 43/3; 93.4% Negative Class prediction: 35/36; 97.2%.

```
# Test the models with all features and reduced (minimal) features
lrtest(logFitForwardAll, logFitForwardMinimal)
```

```
## Likelihood ratio test
##
## Model 1: postotalSSRetired ~ posDJIopen + posDJIclose + posDJIadjClose +
##     posaverageFemaleSSRetiredPay + posRealSPopen + percChangeRealDJIhigh +
##     posRealSPadjClose + posRealaverageFemaleSSRetiredPay + percChangeRealDJIadjClose +
##     percChangeRealSPadjClose
## Model 2: postotalSSRetired ~ posDJIadjClose + posRealSPopen + posRealSPadjClose
##   #Df  LogLik Df  Chisq Pr(>Chisq)
## 1  11 -50.117
## 2   4 -55.679 -7 11.124     0.1333
```

```
lrtest(logFitForwardMinimal, logFitForwardAll)
```

```
## Likelihood ratio test
##
## Model 1: postotalSSRetired ~ posDJIadjClose + posRealSPopen + posRealSPadjClose
## Model 2: postotalSSRetired ~ posDJIopen + posDJIclose + posDJIadjClose +
##     posaverageFemaleSSRetiredPay + posRealSPopen + percChangeRealDJIhigh +
##     posRealSPadjClose + posRealaverageFemaleSSRetiredPay + percChangeRealDJIadjClose +
##     percChangeRealSPadjClose
##   #Df  LogLik Df  Chisq Pr(>Chisq)
## 1   4 -55.679
## 2  11 -50.117  7 11.124     0.1333
```

Testing the null hypothesis that the restricted model (3 predictors) fits the data BETTER than the unrestricted model (10 predictors) results in a p-value greater than 0.05 (p-value = 0.1333), and thus we fail to reject the null hypothesis. The restricted model is at least as good as the unrestricted.

```
# Given that HO holds that the reduced model is true, a p-value for the overall model fit statistic tha
anova(logFitForwardAll, logFitForwardMinimal, test ="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: postotalSSRetired ~ posDJIopen + posDJIclose + posDJIadjClose +
##     posaverageFemaleSSRetiredPay + posRealSPopen + percChangeRealDJIhigh +
##     posRealSPadjClose + posRealaverageFemaleSSRetiredPay + percChangeRealDJIadjClose +
##     percChangeRealSPadjClose
## Model 2: postotalSSRetired ~ posDJIadjClose + posRealSPopen + posRealSPadjClose
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       393     100.23
## 2       400     111.36 -7  -11.124   0.1333
```

```
anova(logFitForwardMinimal, logFitForwardAll, test ="Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: postotalSSRetired ~ posDJIadjClose + posRealSPopen + posRealSPadjClose
## Model 2: postotalSSRetired ~ posDJIopen + posDJIclose + posDJIadjClose +
##     posaverageFemaleSSRetiredPay + posRealSPopen + percChangeRealDJIhigh +
##     posRealSPadjClose + posRealaverageFemaleSSRetiredPay + percChangeRealDJIadjClose +
##     percChangeRealSPadjClose
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1       400     111.36
## 2       393     100.23  7   11.124   0.1333
```

```
varImp(logFitForwardAll)
```

```
##                                    Overall
## posDJIopen                       0.7975565
## posDJIclose                      1.7542048
## posDJIadjClose                   2.1449539
## posaverageFemaleSSRetiredPay     1.5956928
## posRealSPopen                    8.0411833
## percChangeRealDJIhigh            0.9670385
## posRealSPadjClose                1.8205001
## posRealaverageFemaleSSRetiredPay 1.9270498
## percChangeRealDJIadjClose        0.5928742
## percChangeRealSPadjClose         0.3598276
```

```
varImp(logFitForwardMinimal)
```

```
##                    Overall
## posDJIadjClose    2.468723
## posRealSPopen     8.936028
## posRealSPadjClose 3.547970
```

Let's check how probit fits the data

```
# Probit ####
```

As the variable of interest is generated from the differences in Total SS Recipients, which appears about normally distributed, a probit model may be more appropriate.

```
probFit <- glm(predForm,
               family = binomial(link = "probit"),
               data = dfStationary)
summary(probFit, diagnostics=TRUE)
```

```
##
```

```
## Call:
## glm(formula = predForm, family = binomial(link = "probit"), data = dfStationary)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.67493  -0.09598  -0.02853   0.23807   2.79822
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -3.3480     0.4453  -7.519 5.50e-14 ***
## posDJIadjClose      0.7431     0.2886   2.575     0.01 *
## posRealSPopen       3.9669     0.3551  11.170  < 2e-16 ***
## posRealSPadjClose   1.2930     0.3300   3.918 8.91e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 545.68  on 403  degrees of freedom
## Residual deviance: 110.30  on 400  degrees of freedom
## AIC: 118.3
##
## Number of Fisher Scoring iterations: 8
```

```r
confint(probFit)
```

```
## Waiting for profiling to be done...

##                        2.5 %    97.5 %
## (Intercept)       -4.3500091 -2.558896
## posDJIadjClose     0.1993879  1.337637
## posRealSPopen      3.3429631  4.773703
## posRealSPadjClose  0.7017883  2.036072
```

```r
probProbs <- predict(probFit, type = 'response')
probProbs[1:10]
```

```
##            2            3            4            5            6
## 0.0004069635 0.9134087643 0.0004069635 0.0004069635 0.0947859691
##            7            8            9           10           11
## 0.9960355244 0.0045953487 0.0004069635 0.0199403179 0.9960355244
```

```r
probPred <- rep(NA, dim(dfStationary)[2])
probPred[probProbs >= 0.5] <- 1
probPred[probProbs < 0.5] <- 0

table(probPred)
```

```
## probPred
##   0   1
## 235 169
```

```r
table(probPred, dfStationary$postotalSSRetired)
```

```
##
## probPred   0   1
##        0 229   6
##        1  11 158
```

```
mean(probPred == dfStationary$postotalSSRetired)
```

```
## [1] 0.9579208
```

```
# Testing Prediction
probFit1 <- glm(predForm,
                family = binomial(link = "probit"),
                data = train)
probProbs1 <- predict(probFit1, test3rdQuart, type = 'response') # setting prediction for the testing s

probPred1 <- rep(0, dim(train)[2])
probPred1[probProbs1 >= 0.5] <- 1
probPred1[probProbs1 < 0.5] <- 0

table(probPred1, test3rdQuart$postotalSSRetired)
```

```
##
## probPred1  0  1
##         0 54  2
##         1  5 40
```

```
mean(probPred1 == test3rdQuart$postotalSSRetired)
```

```
## [1] 0.9306931
```

Predicition accuracy is approximately 93% with a train/test split at the 3rd quartile mark of the dates. Positive class prediction: 40/2; 95.2% Negative Class prediction: 54/5; 91.5%.

```
# Test it at the 80/20 split
train80 <- subset(dfStationary, dfStationary$date < as.Date(dfStationary$date[round(nrow(dfStationary) 
test20 <- subset(dfStationary, dfStationary$date >= as.Date(dfStationary$date[round(nrow(dfStationary) 

probFit1 <- glm(predForm,
                family = binomial(link = "probit"),
                data = train)
probProbs1 <- predict(probFit1, test20, type = 'response') # setting prediction for the testing set FRO

probPred1 <- rep(0, dim(train80)[2])
probPred1[probProbs1 >= 0.5] <- 1
probPred1[probProbs1 < 0.5] <- 0

table(probPred1, test20$postotalSSRetired)
```

```
##
## probPred1  0  1
##         0 43  1
##         1  3 35
```

```
mean(probPred1 == test20$postotalSSRetired)
```

```
## [1] 0.9512195
```

Predicition accuracy is approximately 95% with a train/test split of 80/20. Positive class prediction: 43/3; 93.4% Negative Class prediction: 35/1; 97.2% This prediction accuracy is the same as the logit model

```
# Descriptive Statistics ####
stat.desc(dfStationary[, valuesStatForward])
```

```
##                posDJIopen  posDJIclose posDJIadjClose
## nbr.val      404.00000000 404.00000000   404.00000000
## nbr.null     146.00000000 206.00000000   159.00000000
## nbr.na         0.00000000   0.00000000     0.00000000
## min            0.00000000   0.00000000     0.00000000
## max            1.00000000   1.00000000     1.00000000
## range          1.00000000   1.00000000     1.00000000
## sum          258.00000000 198.00000000   245.00000000
## median         1.00000000   0.00000000     1.00000000
## mean           0.63861386   0.49009901     0.60643564
## SE.mean        0.02393053   0.02490189     0.02433592
## CI.mean.0.95   0.04704427   0.04895383     0.04784120
## var            0.23135887   0.25052207     0.23926369
## std.dev        0.48099778   0.50052180     0.48914588
## coef.var       0.75319033   1.02126671     0.80659157
##              posaverageFemaleSSRetiredPay posRealSPopen
## nbr.val                      404.00000000  404.00000000
## nbr.null                     159.00000000  235.00000000
## nbr.na                         0.00000000    0.00000000
## min                            0.00000000    0.00000000
## max                            1.00000000    1.00000000
## range                          1.00000000    1.00000000
## sum                          245.00000000  169.00000000
## median                         1.00000000    0.00000000
## mean                           0.60643564    0.41831683
## SE.mean                        0.02433592    0.02457216
## CI.mean.0.95                   0.04784120    0.04830563
## var                            0.23926369    0.24393165
## std.dev                        0.48914588    0.49389437
## coef.var                       0.80659157    1.18067057
##              percChangeRealDJIhigh posRealSPadjClose
## nbr.val               404.000000000      404.00000000
## nbr.null                0.000000000      215.00000000
## nbr.na                  0.000000000        0.00000000
## min                    -0.241092661        0.00000000
## max                     0.117105913        1.00000000
## range                   0.358198574        1.00000000
## sum                     2.369177169      189.00000000
## median                  0.006099073        0.00000000
## mean                    0.005864300        0.46782178
## SE.mean                 0.001588591        0.02485514
## CI.mean.0.95            0.003122961        0.04886193
## var                     0.001019544        0.24958234
## std.dev                 0.031930293        0.49958217
## coef.var                5.444860179        1.06788992
##              posRealaverageFemaleSSRetiredPay percChangeRealDJIadjClose
## nbr.val                          404.00000000             404.000000000
## nbr.null                         157.00000000               0.000000000
## nbr.na                             0.00000000               0.000000000
## min                                0.00000000              -0.234162150
## max                                1.00000000               0.132106278
## range                              1.00000000               0.366268428
## sum                              247.00000000               2.529224083
## median                             1.00000000               0.008433098
## mean                                 0.61138614          0.006260456
## SE.mean                              0.02428088          0.002105521
## CI.mean.0.95                         0.04773300          0.004139177
## var                                  0.23818269          0.001791021
## std.dev                              0.48803964          0.042320456
## coef.var                             0.79825107          6.759964196
##              percChangeRealSPadjClose
## nbr.val                 404.000000000
## nbr.null                  0.000000000
## nbr.na                    0.000000000
## min                      -0.219671380
## max                       0.125671204
## range                     0.345342584
## sum                       2.290283887
## median                    0.008526646
## mean                      0.005669020
## SE.mean                   0.002104440
## CI.mean.0.95              0.004137051
## var                       0.001789182
## std.dev                   0.042298718
## coef.var                  7.461381574
```

```
kable(stat.desc(dfStationary[, valuesStatForward[c(1, 2, 3, 5)]], norm=TRUE, p=0.95), digits=3, align='
        "Summary Statistics of Relevant Variables 1")
```

Table 1: Summary Statistics of Relevant Variables 1

	posDJIopen	posDJIclose	posDJIadjClose	posRealSPopen
nbr.val	404.000	404.000	404.000	404.000
nbr.null	146.000	206.000	159.000	235.000
nbr.na	0.000	0.000	0.000	0.000
min	0.000	0.000	0.000	0.000
max	1.000	1.000	1.000	1.000
range	1.000	1.000	1.000	1.000
sum	258.000	198.000	245.000	169.000
median	1.000	0.000	1.000	0.000
mean	0.639	0.490	0.606	0.418
SE.mean	0.024	0.025	0.024	0.025
CI.mean.0.95	0.047	0.049	0.048	0.048
var	0.231	0.251	0.239	0.244
std.dev	0.481	0.501	0.489	0.494
coef.var	0.753	1.021	0.807	1.181
skewness	-0.575	0.039	-0.434	0.330
skew.2SE	-2.368	0.163	-1.788	1.359
kurtosis	-1.674	-2.003	-1.816	-1.896
kurt.2SE	-3.454	-4.135	-3.748	-3.913
normtest.W	0.608	0.636	0.620	0.627
normtest.p	0.000	0.000	0.000	0.000

```
kable(stat.desc(dfStationary[, valuesStatForward[c(4,7,8)]], norm=TRUE, p=0.95), digits=3, align='c',ca
        "Summary Statistics of Relevant Variables 2")
```

Table 2: Summary Statistics of Relevant Variables 2

	posaverageFemaleSSRetiredPay	posRealSPadjClose	posRealaverageFemaleSSRetiredPay
nbr.val	404.000	404.000	404.000
nbr.null	159.000	215.000	157.000
nbr.na	0.000	0.000	0.000
min	0.000	0.000	0.000
max	1.000	1.000	1.000
range	1.000	1.000	1.000
sum	245.000	189.000	247.000
median	1.000	0.000	1.000
mean	0.606	0.468	0.611
SE.mean	0.024	0.025	0.024
CI.mean.0.95	0.048	0.049	0.048
var	0.239	0.250	0.238
std.dev	0.489	0.500	0.488
coef.var	0.807	1.068	0.798
skewness	-0.434	0.129	-0.455
skew.2SE	-1.788	0.529	-1.875
kurtosis	-1.816	-1.988	-1.797
kurt.2SE	-3.748	-4.104	-3.709
normtest.W	0.620	0.635	0.618
normtest.p	0.000	0.000	0.000

```r
kable(stat.desc(dfStationary[, valuesStatForward[c(6, 9, 10)]], norm=TRUE, p=0.95), digits=3, align='c'
        "Summary Statistics of Relevant Variables 3")
```

Table 3: Summary Statistics of Relevant Variables 3

	percChangeRealDJIhigh	percChangeRealDJIadjClose	percChangeRealSPadjClose
nbr.val	404.000	404.000	404.000
nbr.null	0.000	0.000	0.000
nbr.na	0.000	0.000	0.000
min	-0.241	-0.234	-0.220
max	0.117	0.132	0.126
range	0.358	0.366	0.345
sum	2.369	2.529	2.290
median	0.006	0.008	0.009
mean	0.006	0.006	0.006
SE.mean	0.002	0.002	0.002
CI.mean.0.95	0.003	0.004	0.004
var	0.001	0.002	0.002
std.dev	0.032	0.042	0.042
coef.var	5.445	6.760	7.461
skewness	-1.295	-0.813	-0.781
skew.2SE	-5.331	-3.347	-3.217
kurtosis	8.998	3.002	2.509
kurt.2SE	18.571	6.197	5.178
normtest.W	0.924	0.963	0.966
normtest.p	0.000	0.000	0.000