SPY_FORECASTING_FINAL_PROJECT

Brandon Shaw

April 18, 2019

Introduction

The S&P 500 is a US market index composed of the 500 largest US publicly traded companies. It is weighted relatively by market cap and is commonly regarded as one of the best indicators of the US corporate economy along with the DOW, NASDAQ, and RUSSELL 2000. The S&P 500 is often traded on the New York Stock Exchange using the SPDR S&P ETF (symbol: SPY). SPY was launched in 1993 and has returned a 9.43% annual total return since inception. The weighted average market cap of companies in the index is $249.7 billion. The ETF pays a quarterly dividend that must be accounted for in modelling total returns, as well as a 0.0945% expense ratio. The interest in modelling the index is to create an algorithmic system of investing that either increases returns or reduces drawdown, or volatility. By implementing a model into your investment strategies of stock indexes, somebody can reduce their losses during market swings. Part of the problem in modelling stock returns using time series analysis is that markets have been in a bull market for the last roughly 10 years. This can throw off forecasts by leading to positive bias when the most recent 10 years of training data are used. In this report, I will explore if any modelling techniques exist that can account for this bias in order to predict future stock returns. There are two goals to aim for ultimately, although the model could be considered a success if either are achieved. These goals are reducing drawdown (essentially volatility) or maximizing returns. The data used is pulled from Yahoo finance using the quantmod library, and represents monthly closing prices adjusted for dividends, splits, and dividends.

Decomposition Method

Pulling mothly adjusted close from Yahoo finance

```
## [1] "SPY"
```

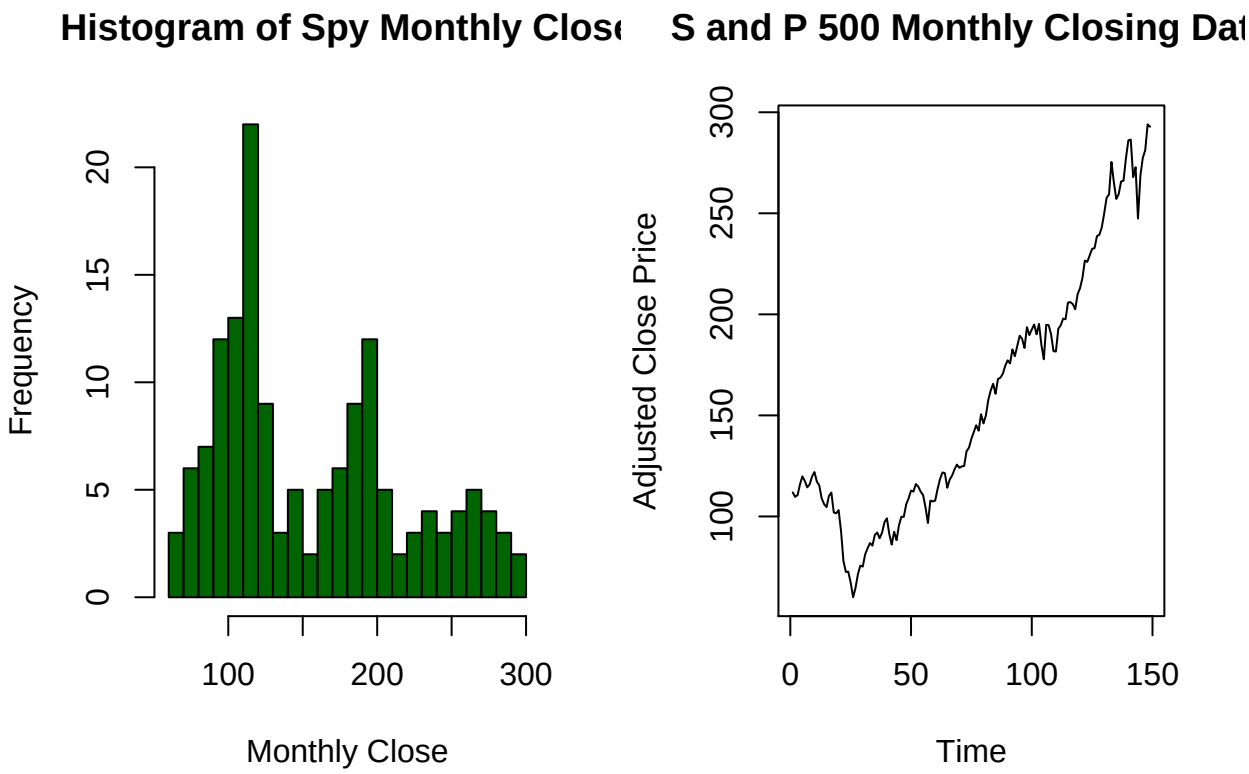
```
##            Adjusted_Close
## 2007-01-01       111.8884
## 2007-02-01       109.6934
## 2007-03-01       110.5263
## 2007-04-01       115.8800
## 2007-05-01       119.8106
## 2007-06-01       117.5522
```

Printing a summary of the time series, we see there are 149 monthly time steps and the data ranges from 60.02 to 294.03

```
##  Adjusted_Close        time
##  Min.   : 60.02   Min.   :  1
##  1st Qu.:107.83   1st Qu.: 38
##  Median :138.47   Median : 75
##  Mean   :157.30   Mean   : 75
##  3rd Qu.:195.28   3rd Qu.:112
##  Max.   :294.02   Max.   :149
```

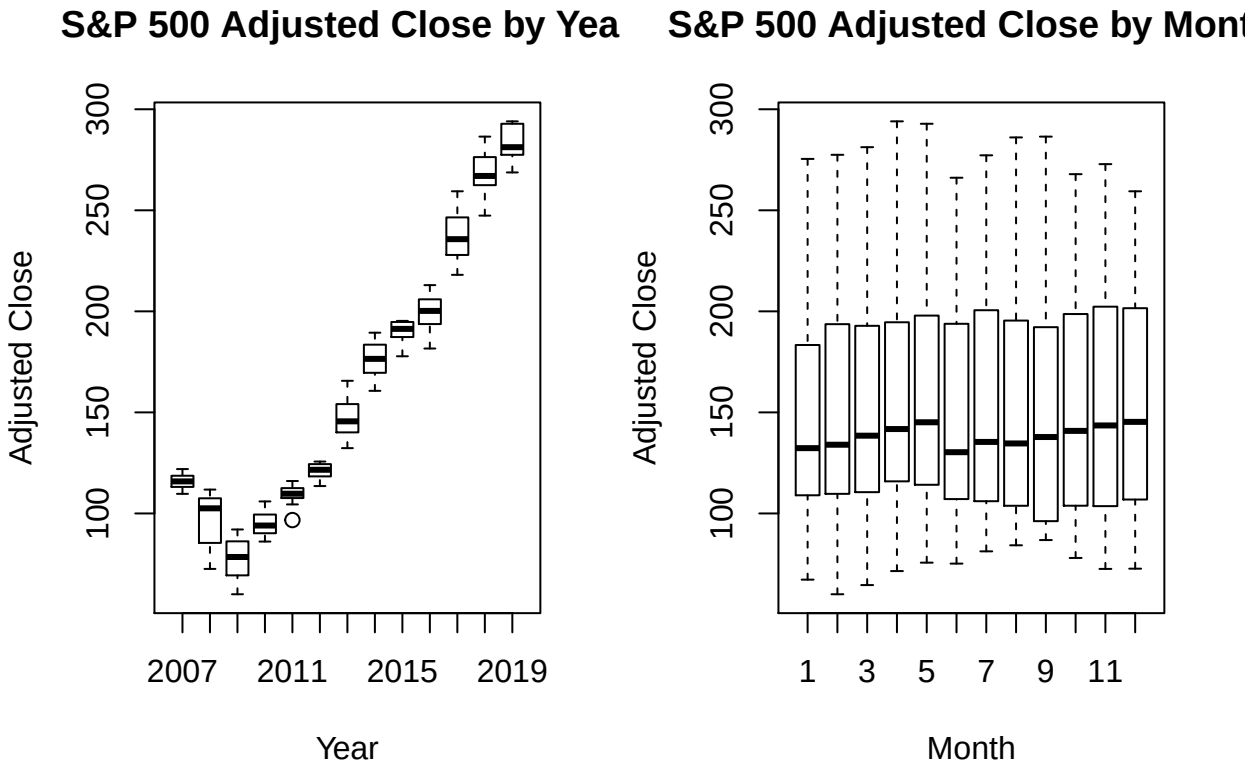
The data begins at January, 2007 and ends at the most recent month.

Plotting histogram of the monthly close and the history of adjusted close



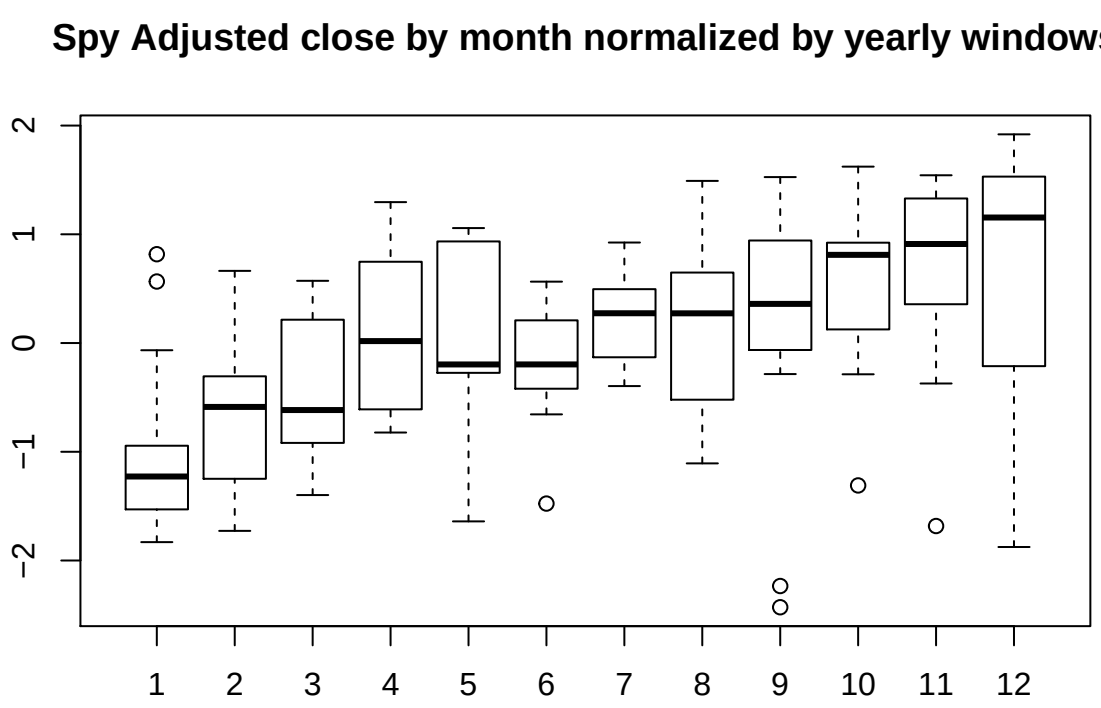
Looking at the plots, we can see that we have been in a strong bull market since the 2008 recession. Looking at the histogram we see there has been less dips and periods of stagnation as markets have steeply increased. Higher prices show less frequency as market growth has accelerated.

Plotting yearly and monthly boxplots for adjusted close



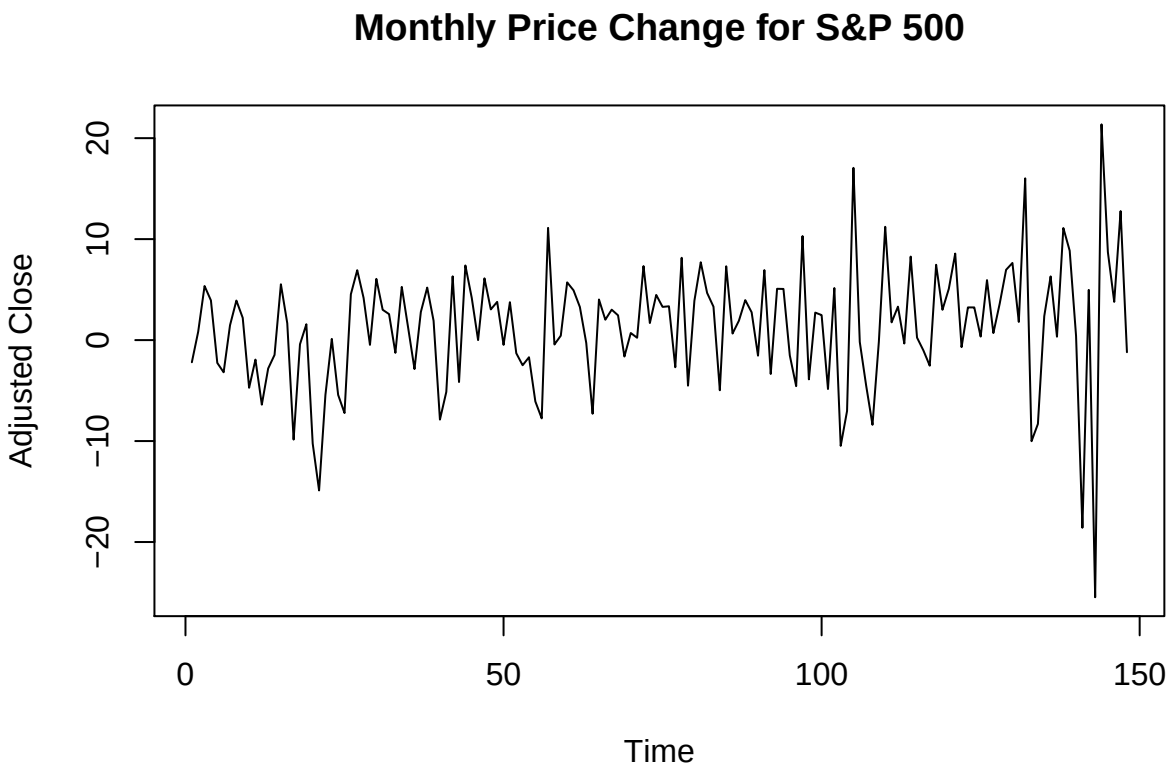
Looking at the boxplots above, we can still see the steep bull market from 2009 to 2019. Looking at the width of the boxplots, we see that the recession in 2008 and 2009 showed high volatility, and the past three years have shown a return in volatility, although this volatility represents steep increases rather than the previous volatile crash of the recession. The monthly boxplots show a mean growth between months 1 and 5, a steep dcrease in month 6, and another steep increase from months 6 to 7. The issue with this is that the range of years changes significantly, so we need to normalize data by yearly windows to understand monthly price movement within years.

Plotting monthly boxplots scaled by yearly windows



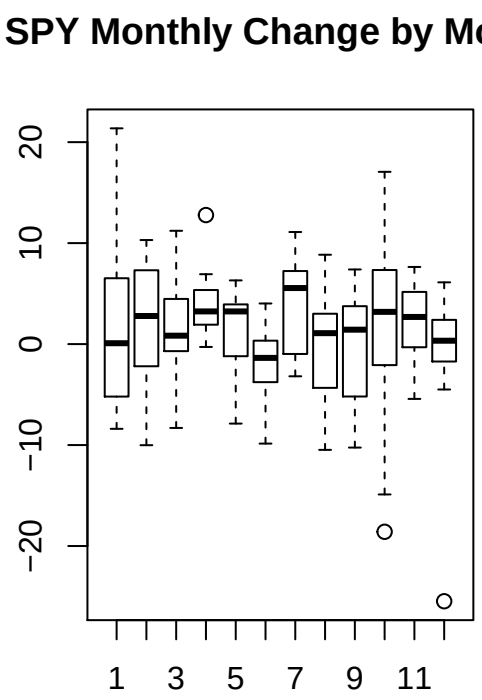
By scaling data into yearly windows we can see that months 1 to 4 show an increase within years, month 5 shows a slight decline, and months 6 to 12 show another increase. Months 5, 8, and 12 show the most volatility within years.

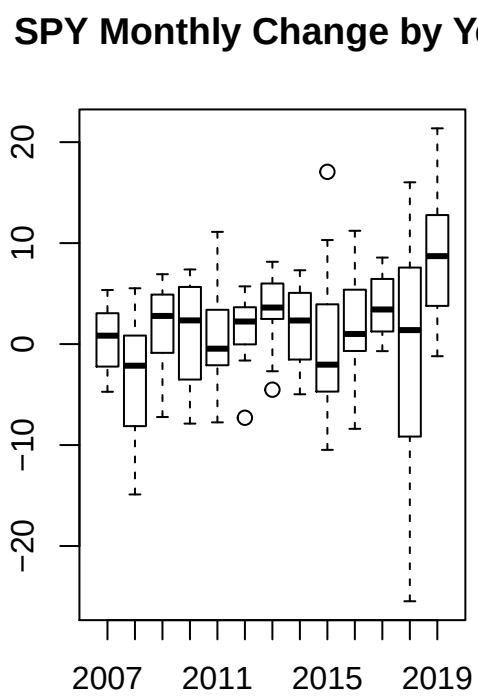
Calculating and plotting a one-time difference by month



Looking at the one time difference for monthly price, we can see that volatility has increased in recent years. This is likely becasue markets move in terms of percent rather than actual price value changes.

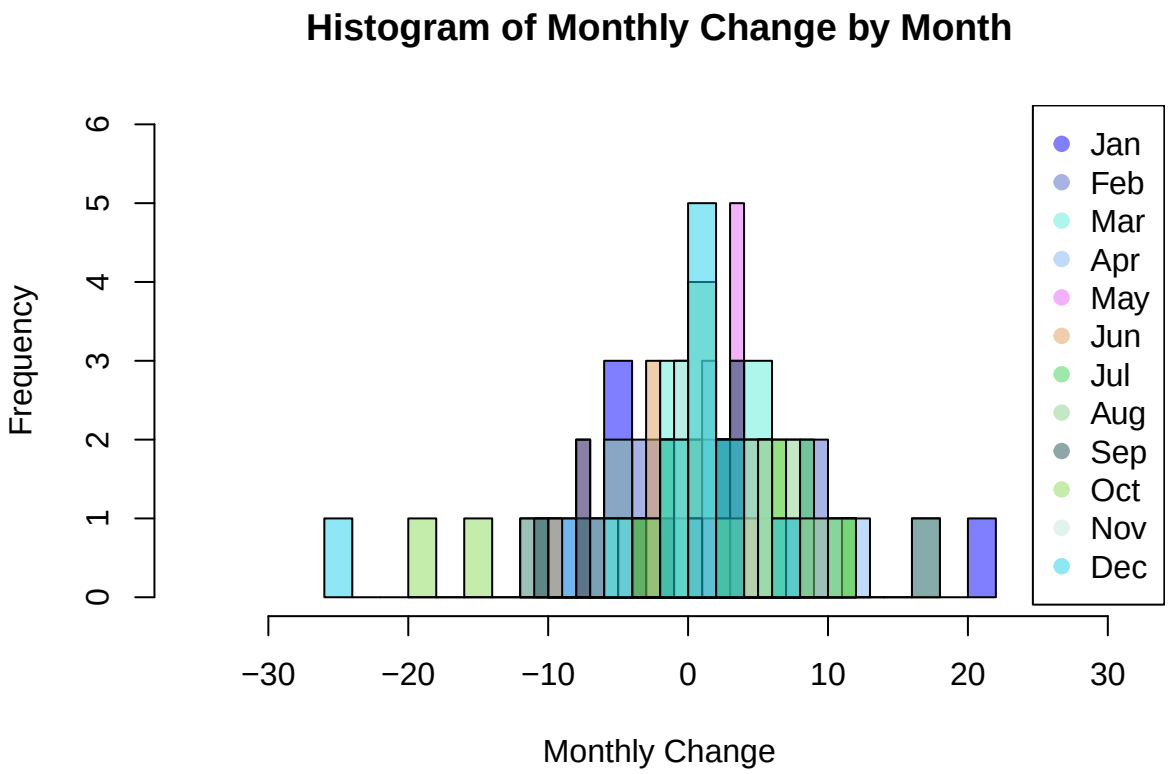
Plotting boxplot of one-time difference price change by month





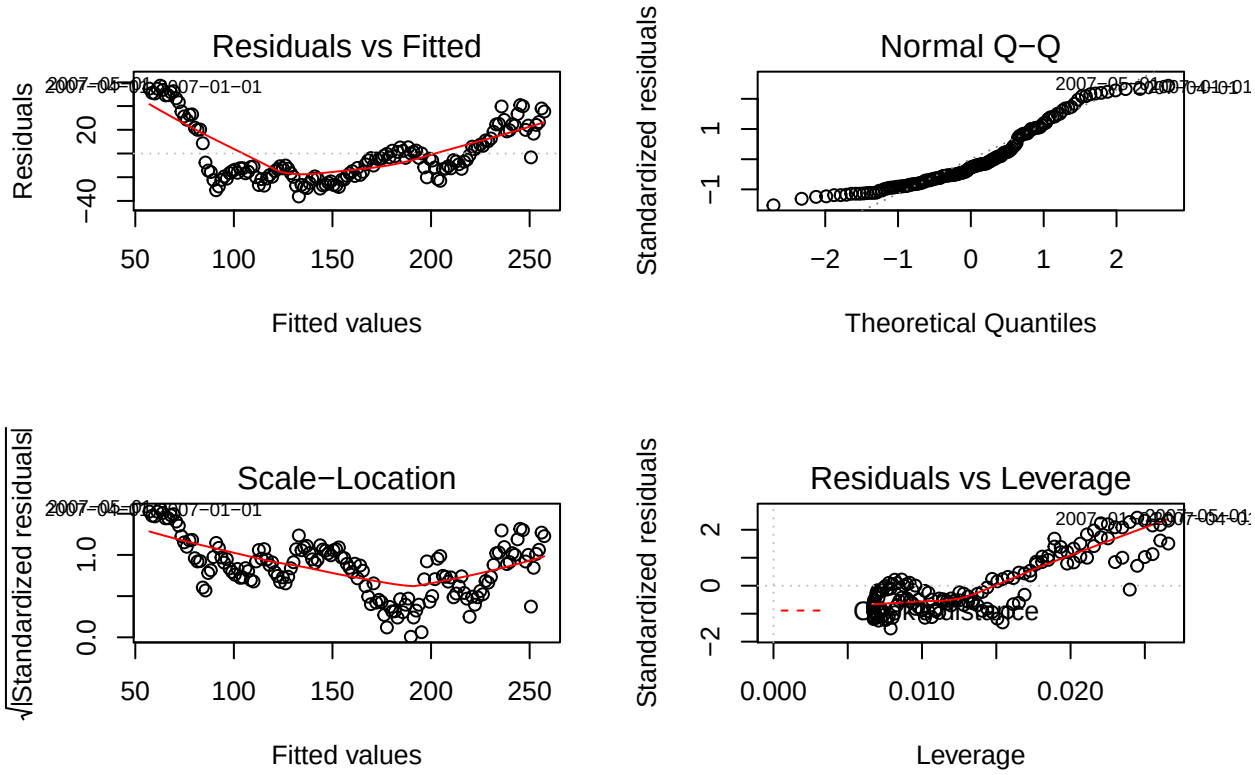
Looking at the boxplot above for one-time difference monthtly price changes, we see that January and October show the highest volatility in terms of price changes.

Histogram of monthly 1-time difference price changes



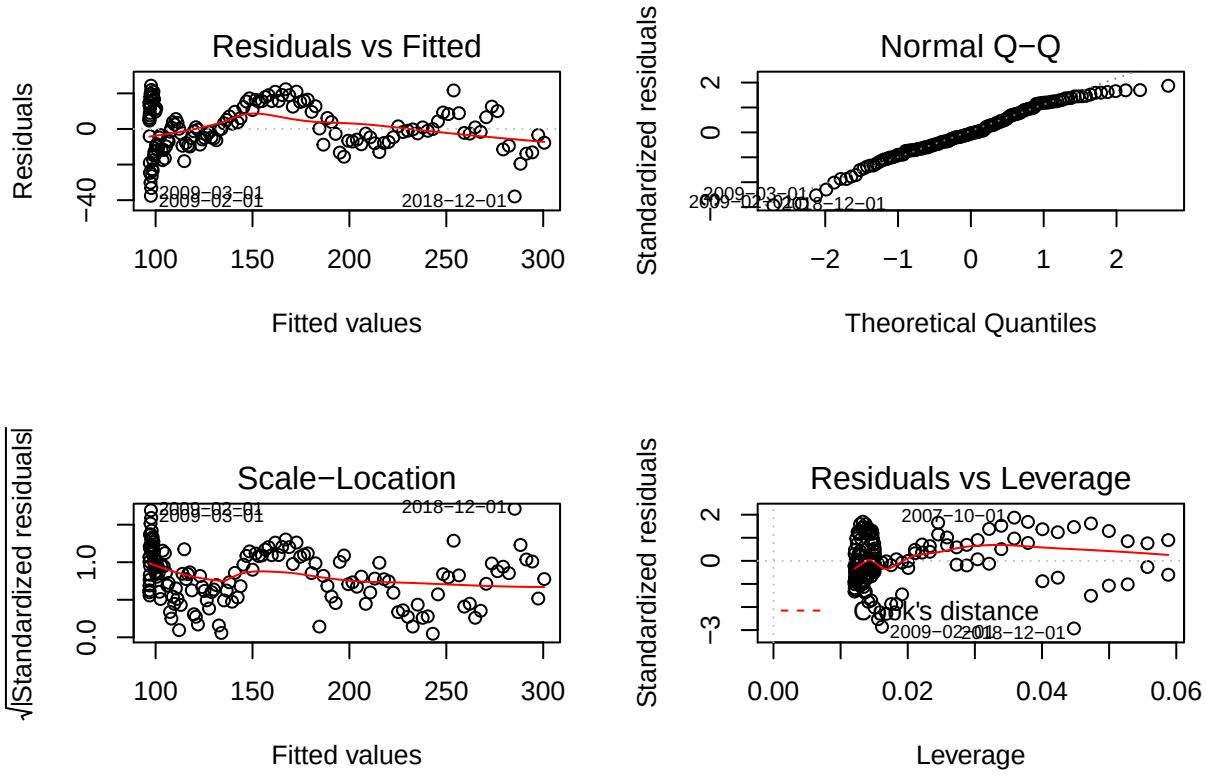
The above histogram of monthly price changes shows that there are no major trends in monthly price change. The data appears generally random and normally distributed across months, although it is interesting that January and February show the highest and lowest monthly price changes. This could be due to "tax selling", a phenomenon in which people buy or sell shares of stock in large quanitites in order to write their buys or sells off at the end of a tax year.

Linear trend model



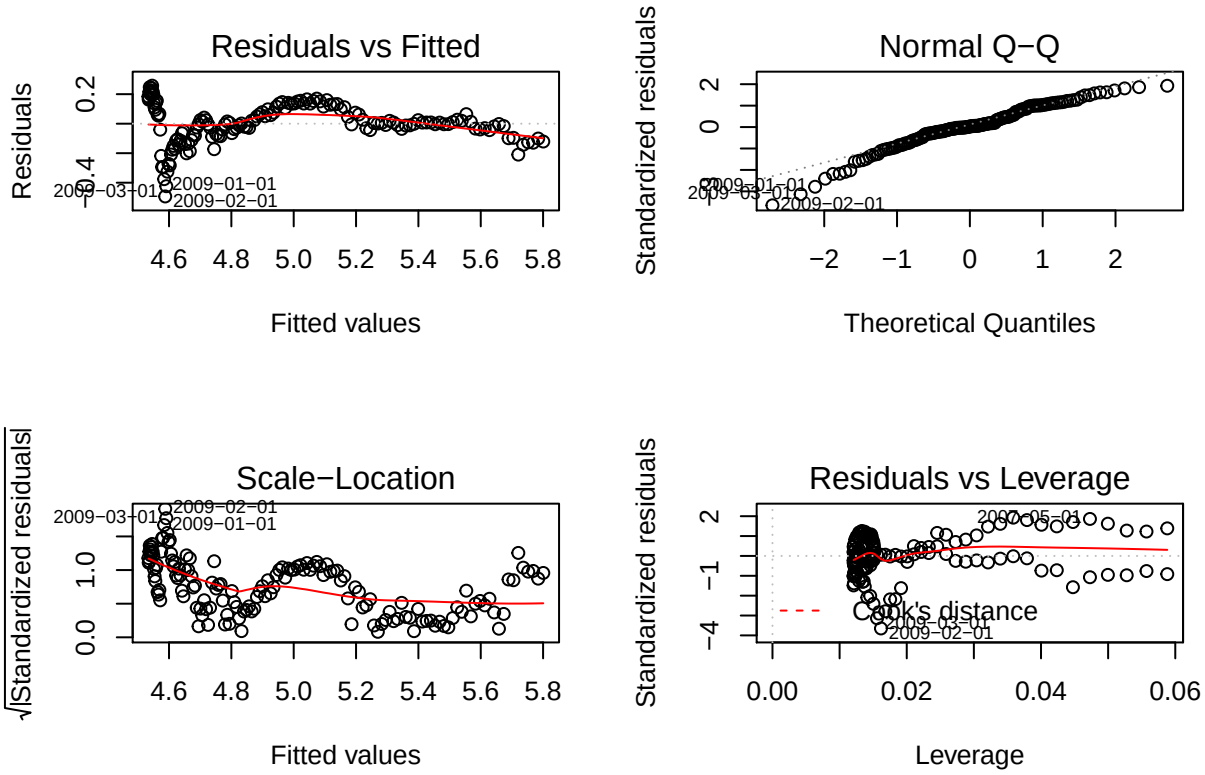
Plotting the residuals plots for a linear trend model, we see the residuals are far from normally distributed with a mean of 0 across time. This suggests that the data is not linear. Looking at the previous adjusted close plot, we expect the increase in prices to be exponential, which is confirmed in the above plots.

Calculating and plotting an exponential trend model



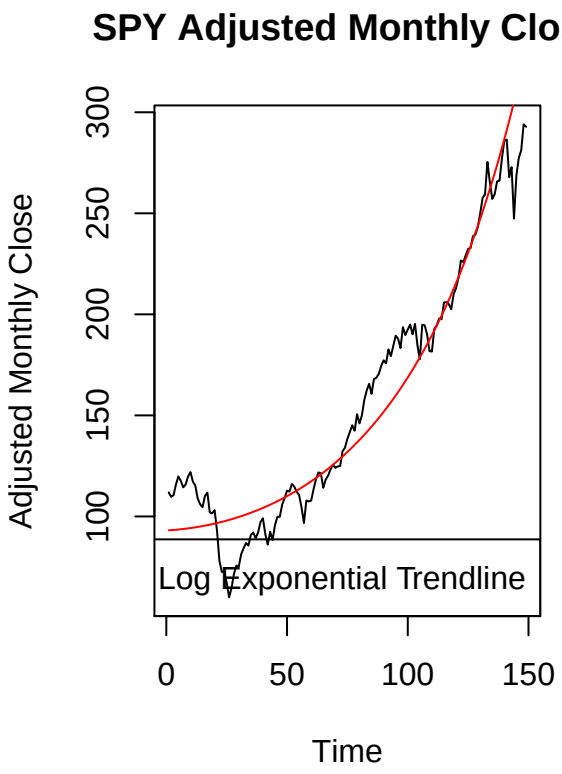
The exponential trend model shows much mnore robust residuals, with a mean of 0 and strong normality. Although the mean and normality has been corrected, there still appears to be cyclical patterns in the residuals vs fitted plot (top left). To help reduce these cyclical trends, we will perform a log transformation on the adjusted close value to help reduce cyclical trends in the residuals.

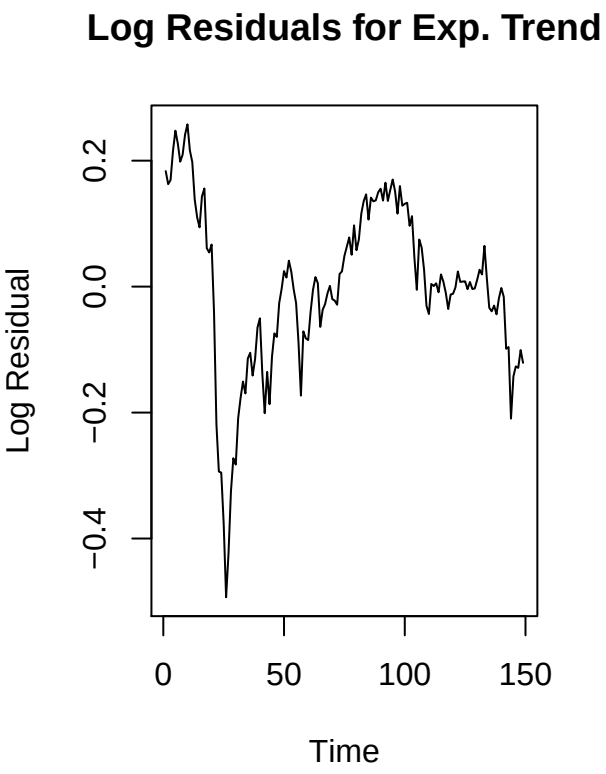
Log transformed Exponential Trend Model



After performing the log transformation, we see that the data is slightly more normal and that the mean of residuals is significantly close to 0 across time with reduced cyclical patterns. This confirms that a log transformed exponential trend model is the most appropriate trend model for the S&P 500 monthly adjusted close.

Plotting the log exponential trend model



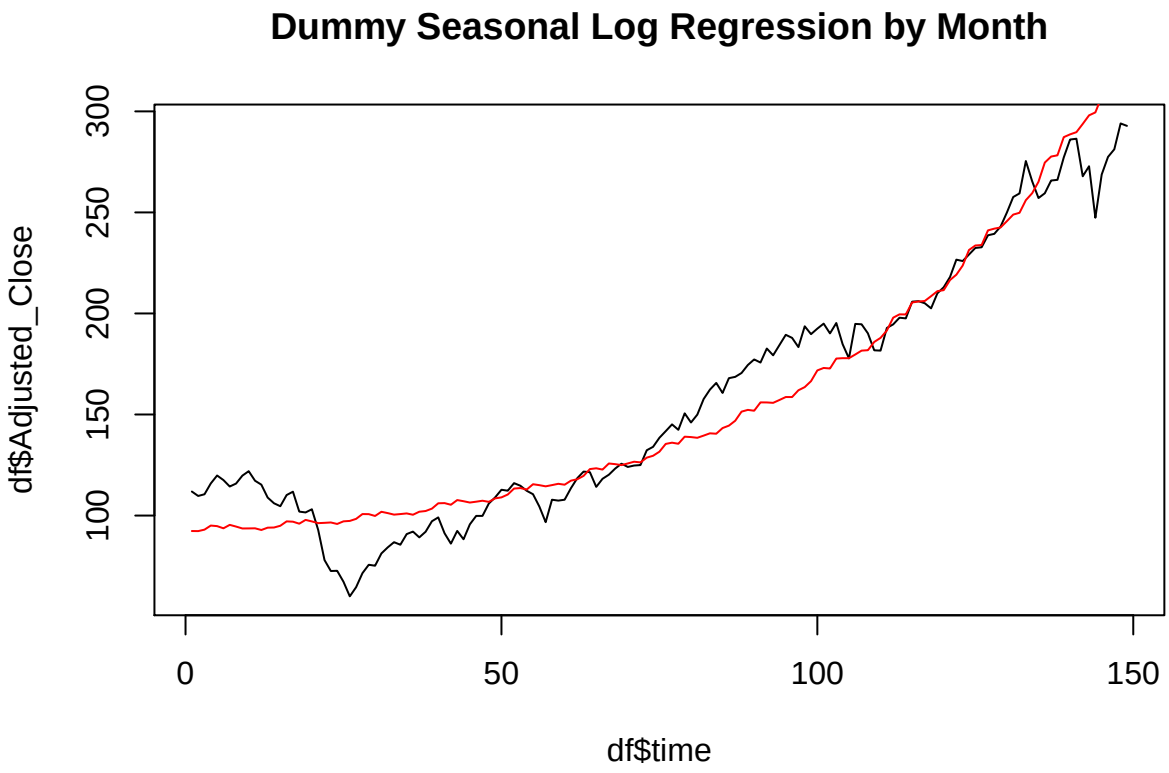


The log exponential trend model appears to closely follow the general shape of the S&P 500, although there is significant deviation from the mean of 0 in the residuals, as the log residuals show a steep decline during the recession and an inverted parabola from 2009 onward.

Dummy seasonal regression for monthly seasons

```
##
## Call:
## lm(formula = logClose ~ df$time + I(df$time^2) + month, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.48359 -0.06667  0.00183  0.09569  0.26425
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   4.525e+00  5.059e-02  89.432  < 2e-16 ***
## df$time       7.598e-04  1.087e-03   0.699    0.486
## I(df$time^2)  5.200e-05  7.019e-06   7.408 1.25e-11 ***
## month2       -1.316e-03  5.538e-02  -0.024    0.981
## month3        5.532e-03  5.538e-02   0.100    0.921
## month4        2.589e-02  5.538e-02   0.467    0.641
## month5        2.171e-02  5.539e-02   0.392    0.696
## month6        8.880e-03  5.656e-02   0.157    0.875
## month7        2.552e-02  5.656e-02   0.451    0.653
## month8        1.505e-02  5.656e-02   0.266    0.791
## month9        3.384e-03  5.656e-02   0.060    0.952
```

```
## month10       1.917e-03  5.657e-02   0.034    0.973
## month11       6.779e-04  5.657e-02   0.012    0.990
## month12      -1.015e-02  5.657e-02  -0.179    0.858
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1412 on 135 degrees of freedom
## Multiple R-squared:  0.8879, Adjusted R-squared:  0.8771
## F-statistic: 82.23 on 13 and 135 DF,  p-value: < 2.2e-16
```



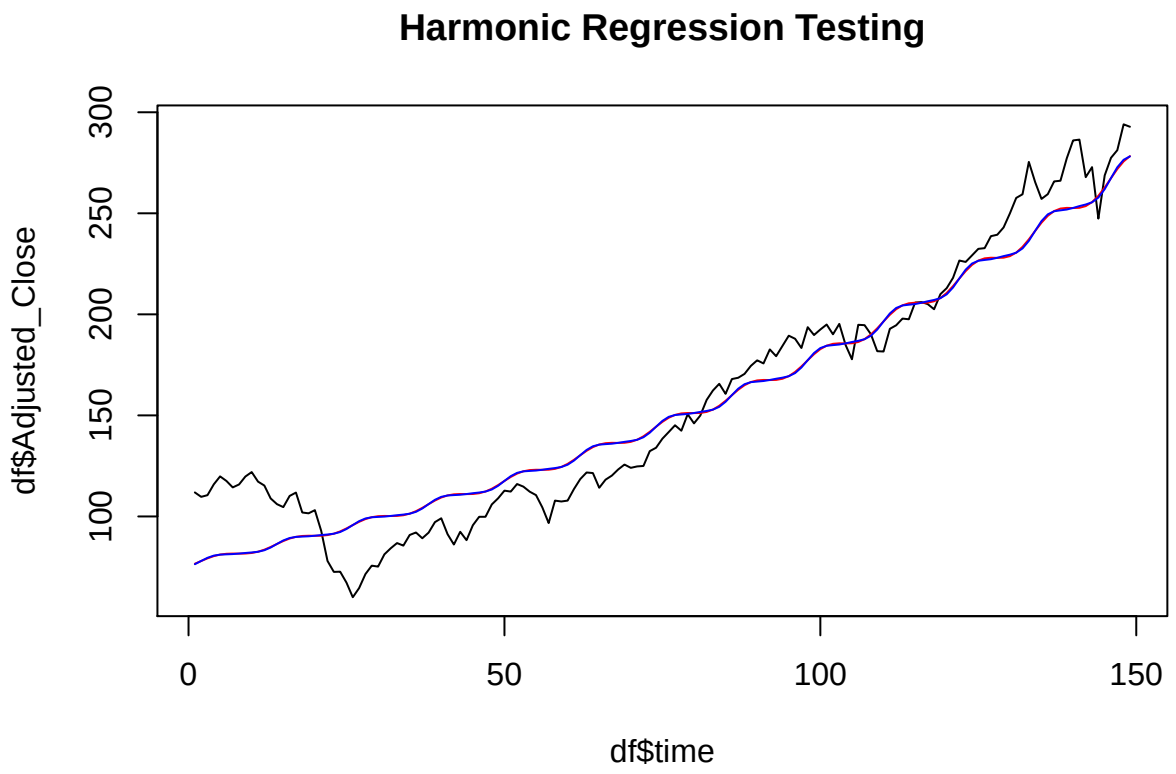
We first test dummy seasonal regresion using 12 seasonal variables representing monthly factors. We implement the seasonal dummy variables into the log exponential trend model. The plot shows that the monthly variables affect the trend model, but not very significantly. All of the monthly variables are shown to be insignificant.

Testing harmonic regression for two and four components

```
##
## Call:
## lm(formula = logClose ~ time + I(sin(2 * pi * time/L)) + I(cos(2 *
##     pi * time/L)), data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46685 -0.10060  0.00917  0.06357  0.39796
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)              4.3361233  0.0266452 162.735   <2e-16 ***
## time                     0.0085606  0.0003081  27.781   <2e-16 ***
## I(sin(2 * pi * time/L))  0.0108963  0.0186909   0.583    0.561
## I(cos(2 * pi * time/L)) -0.0133340  0.0188071  -0.709    0.479
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1618 on 145 degrees of freedom
## Multiple R-squared:  0.8419, Adjusted R-squared:  0.8386
## F-statistic: 257.4 on 3 and 145 DF,  p-value: < 2.2e-16
```

```
##
## Call:
## lm(formula = logClose ~ time + I(sin(2 * pi * time/L)) + I(cos(2 *
##     pi * time/L)) + I(sin(4 * pi * time/L)) + I(cos(4 * pi *
##     time/L)), data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46709 -0.10175  0.00940  0.06114  0.39473
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)              4.3361619  0.0268360 161.580   <2e-16 ***
## time                     0.0085598  0.0003104  27.578   <2e-16 ***
## I(sin(2 * pi * time/L))  0.0108361  0.0188216   0.576    0.566
## I(cos(2 * pi * time/L)) -0.0132802  0.0189454  -0.701    0.484
## I(sin(4 * pi * time/L)) -0.0016614  0.0188243  -0.088    0.930
## I(cos(4 * pi * time/L)) -0.0033643  0.0189374  -0.178    0.859
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1629 on 143 degrees of freedom
## Multiple R-squared:  0.842,  Adjusted R-squared:  0.8364
## F-statistic: 152.4 on 5 and 143 DF,  p-value: < 2.2e-16
```



The two and four component harmonic regression models appear similar, although all components are shown to be insignificant in both the two and four component models. The plot shows that the harmonic models appear to be nearly identical, and the shape of sin wave movement does not really follow the actual price movement.

Comparing AIC for trend, dummy, and harmonic regression models

```
##                     Models  BIC_list
## 1          Exp Trend Model -582.0676
## 2           Seasonal Model -528.0296
## 3  Two Comp Harmonic Model -526.8754
## 4 Four Comp Harmonic Model -516.9086
```

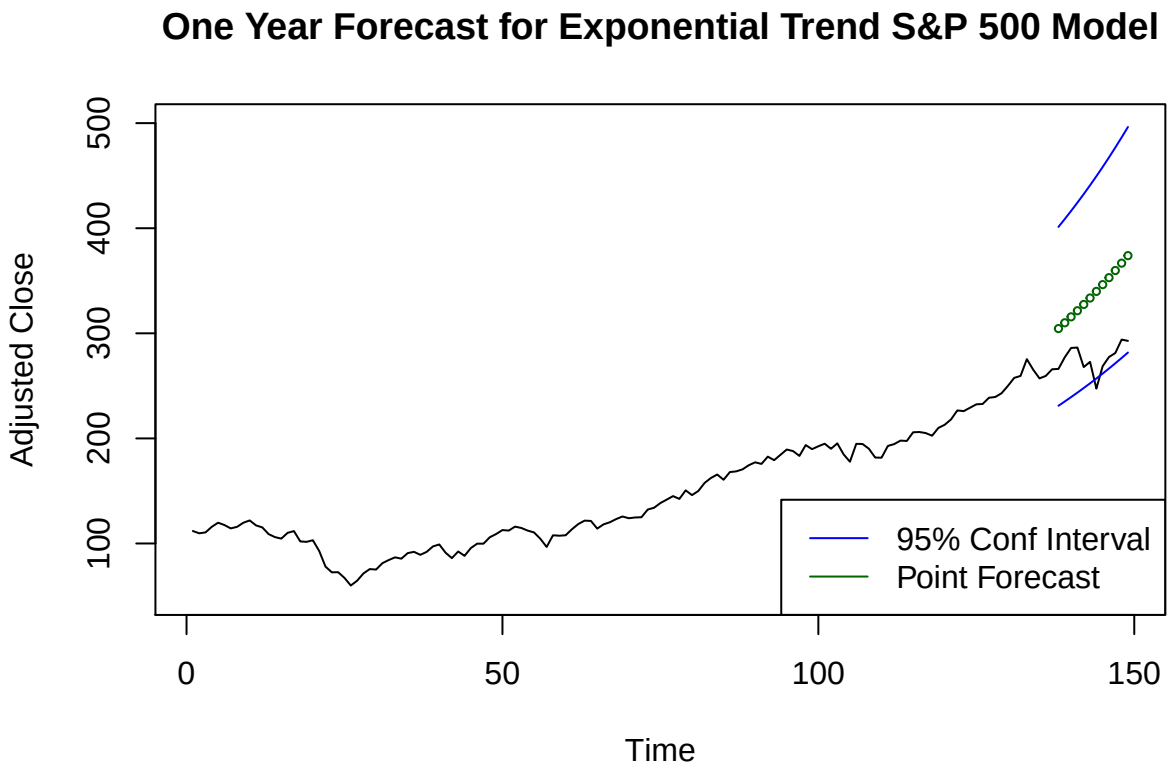
The above comparison of AIC for the four models confirms that both the dummy and harmonic component variables are insignificant for modelling the S&P 500. The exponential trend; however, is signficant, and this is confirmed by the log exponential trend model showing the lowest AIC (582.1216). This shows that a simple log exponential trend model is the most accurate at modelling past performance of the S&P 500.

Performing 12-step (one year) forecast using log exponential trend model

```
##     Point.Forecast    Lo.80    Hi.80    Lo.95    Hi.95
## 138       5.718616 5.538922 5.898310 5.442660 5.994571
## 139       5.736597 5.556564 5.916631 5.460121 6.013074
## 140       5.754718 5.574329 5.935107 5.477695 6.031741
## 141       5.772977 5.592215 5.953740 5.495381 6.050573
## 142       5.791376 5.610223 5.972528 5.513180 6.069571
## 143       5.809913 5.628352 5.991474 5.531091 6.088736
## 144       5.828590 5.646602 6.010577 5.549112 6.108067
## 145       5.847405 5.664972 6.029838 5.567244 6.127566
```

```
## 146       5.866359 5.683462 6.049256 5.585485 6.147234
## 147       5.885453 5.702072 6.068834 5.603835 6.167070
## 148       5.904685 5.720800 6.088570 5.622293 6.187077
## 149       5.924056 5.739647 6.108465 5.640859 6.207253
```

Plotting the one year log exponential forecast



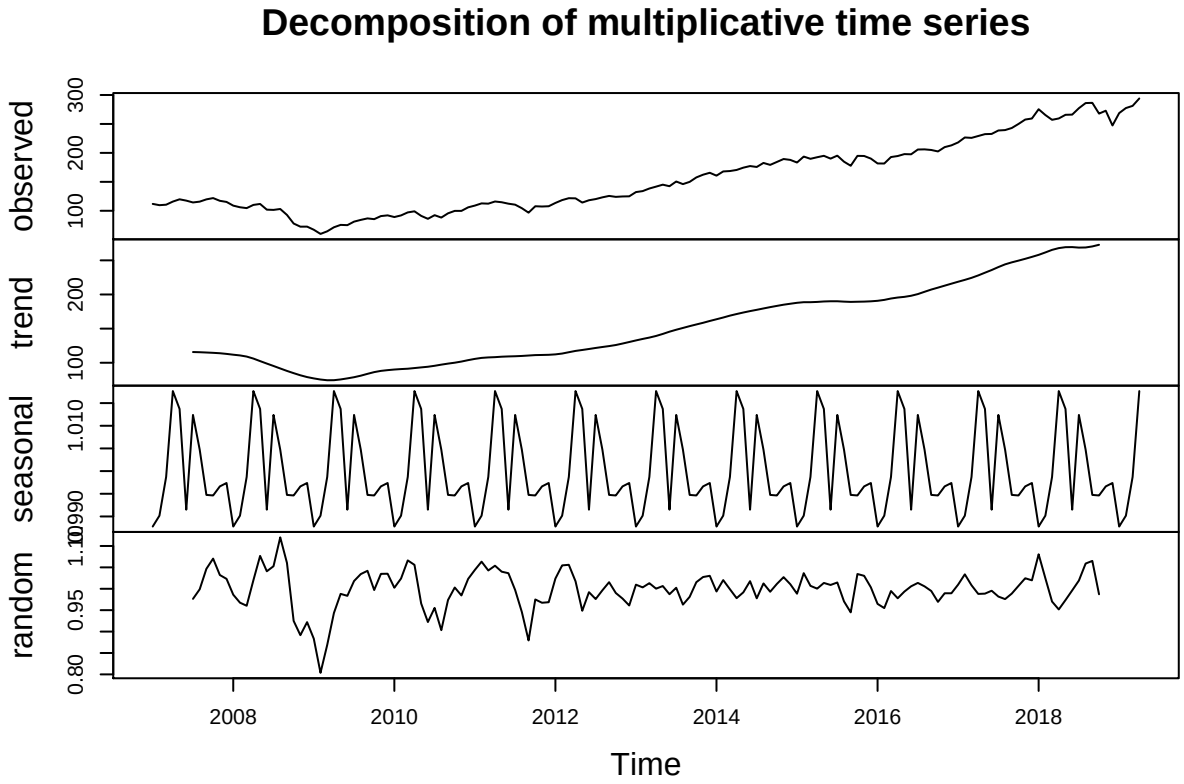
The one year forecast for the log exponential model, as expected, shows a steady exponential increase is expected for the S&P 500. While this model has performed fairly well, it is flawed because it is extremely bias due to the recent 10 year bull market. Markets will not go up forever, as forecasted by the model. The exponential model naturally goes to infinity as time increases, which is completely unrealistic. This model only performs well in a bull market, as it always predicts that you should buy the stock if the current cycle is a bull market. At some point, markets will eventually cycle into a bear or stagnant market due to natural economic cycles and the maximum limits of an economy. This model should not be trusted or used for investment, as it always predicts a buy if the previous trend has been positive. We expect an ARIMA forecast to perform much more robustly with far less trend bias.

Reloading the S&P 500 data

```
## [1] "SPY"
```

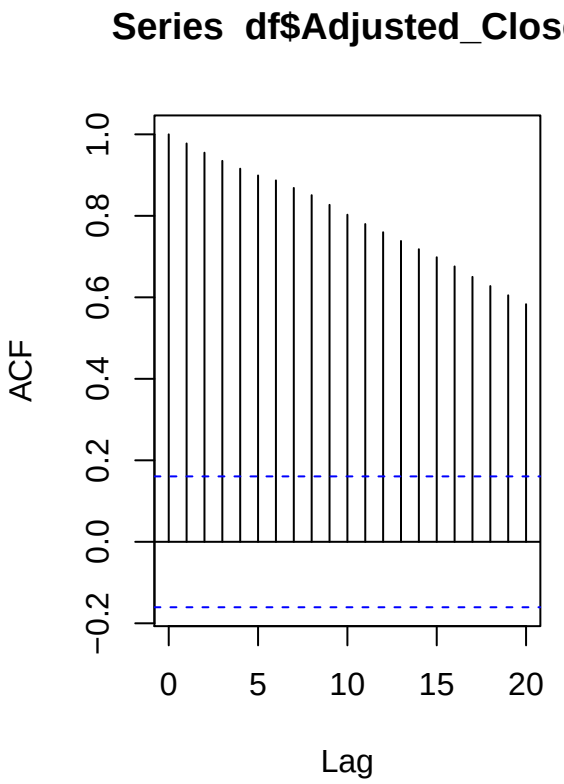
```
##   Adjusted_Close
## 1       111.8884
## 2       109.6934
## 3       110.5263
## 4       115.8800
## 5       119.8106
## 6       117.5522
```

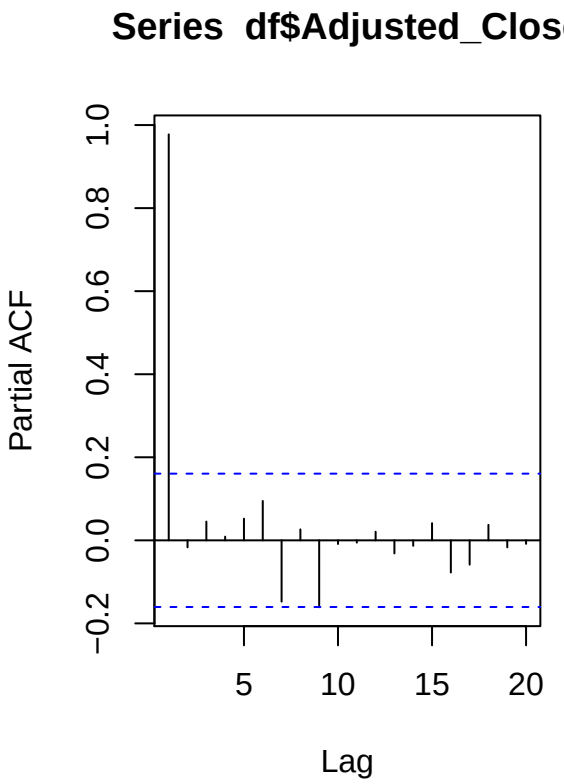
Plotting decomposed components of the time series



Looking at the above components, we see that there is an increasing trend, and slight monthly cycles, although the range of the seasonal coefficients are barely varied from the mean of around 1.0995 (range is 1.099 to 1.010).

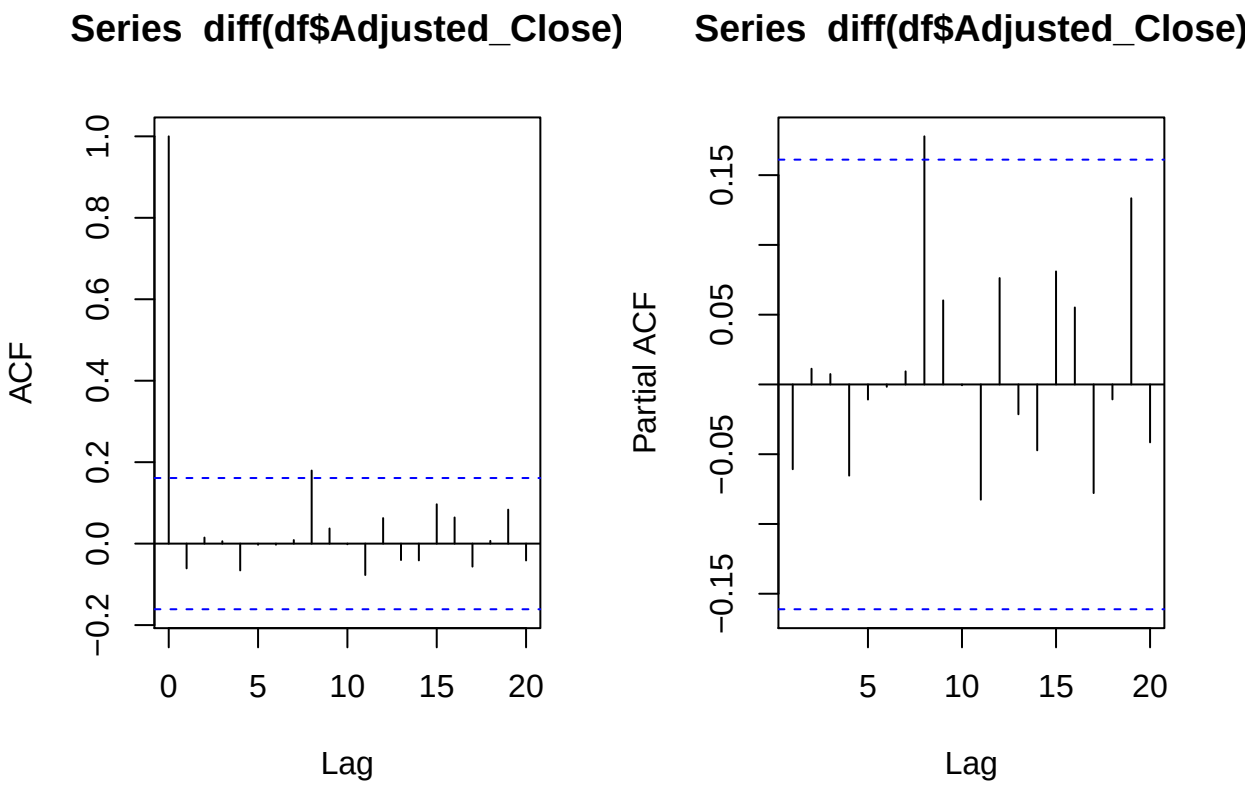
Plotting ACF and PACF of adjusted close





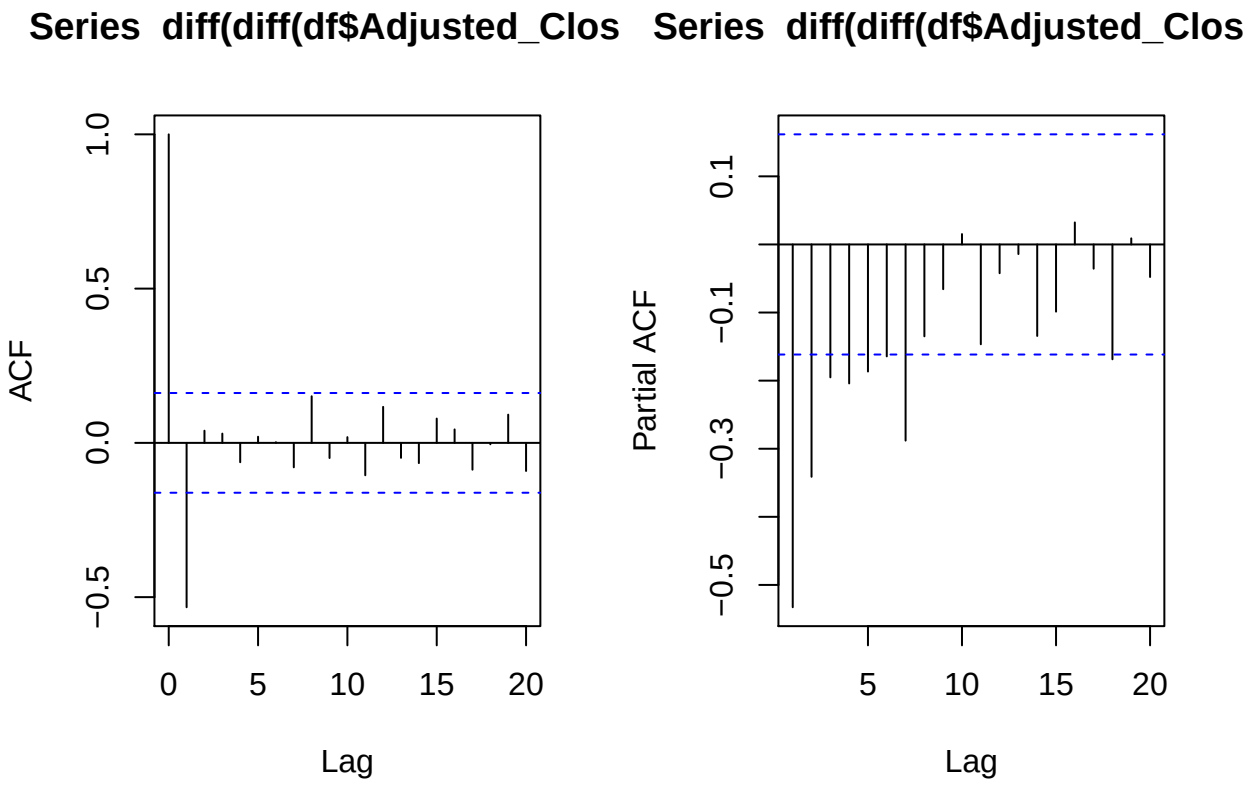
The ACF is significantly higher than the confidence interval, so we will perform a one-time difference transformation.

Plotting ACF and PACF of adjusted close with 1-time difference transformation



The PACF appears to follow a white noise process, and the ACF cuts off at time 0. The PACF rises above the confidence limit, so we will perform a two-time difference transformation next.

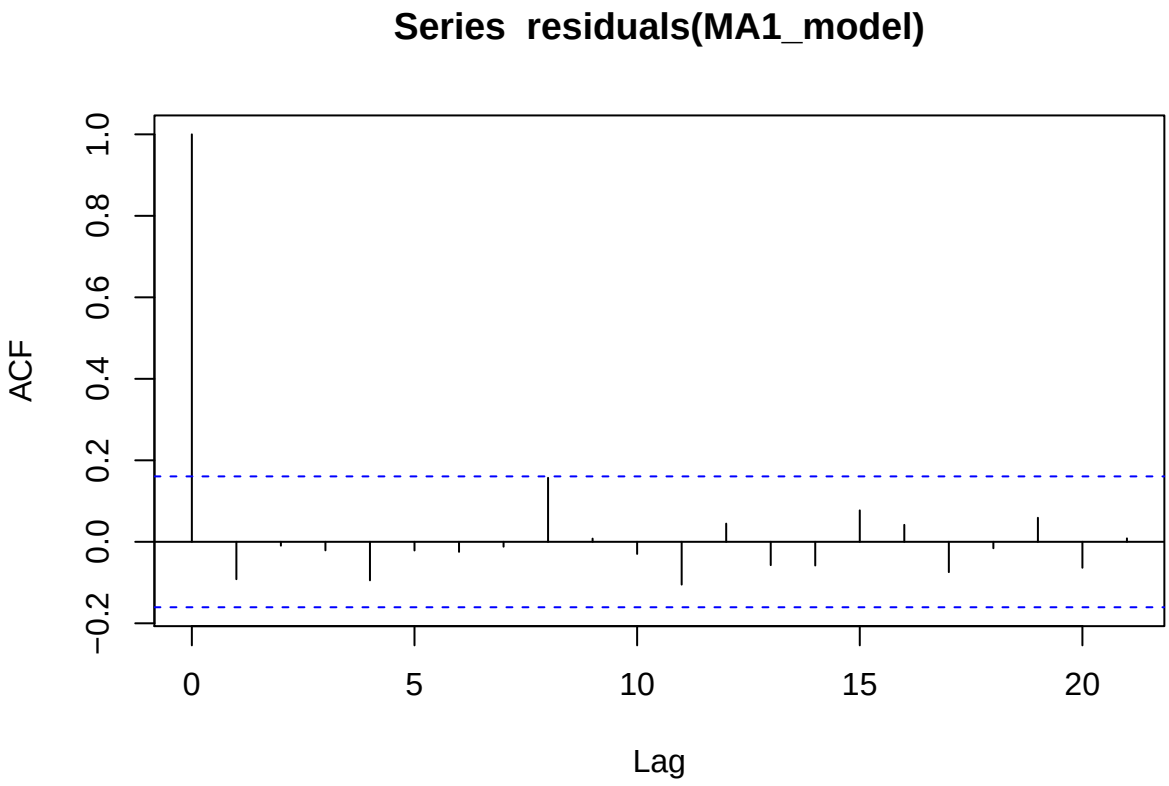
****Plotting ACF and PACF of adjusted close for 2-time difference transformation**

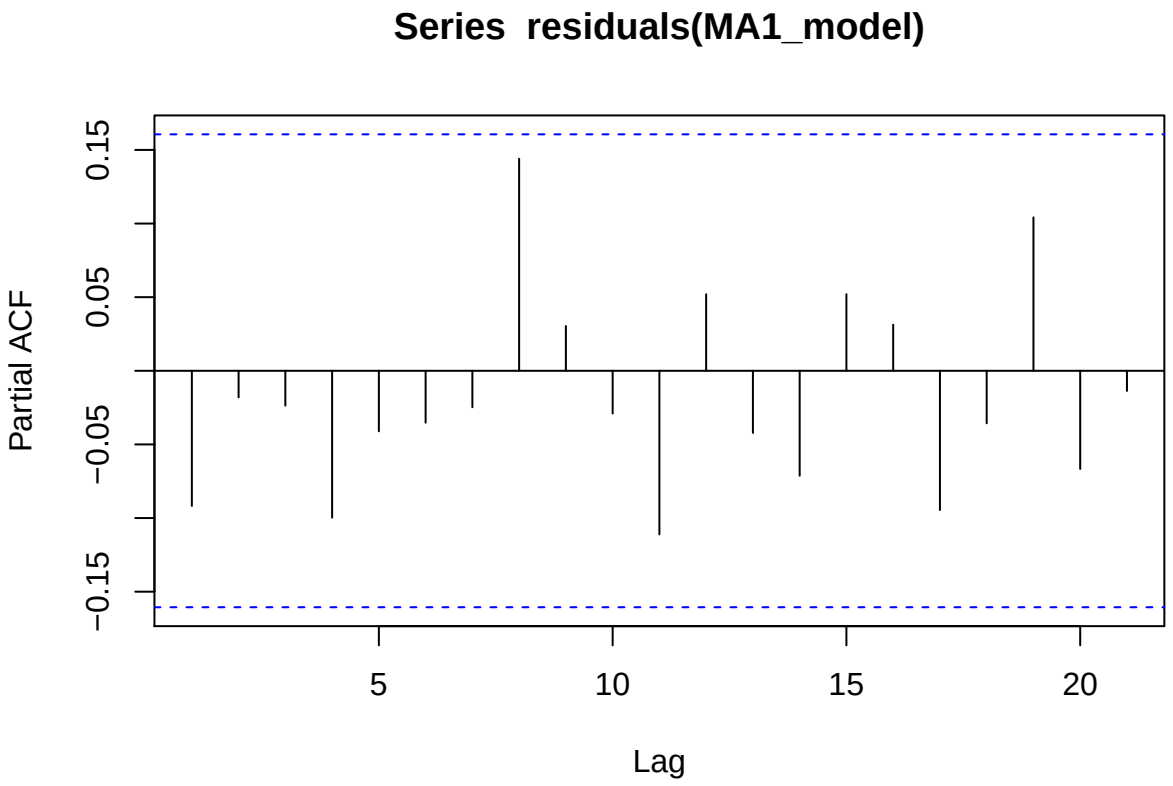


Looking at the two time difference transformed ACF and PACF, we see that the PACF decays while the ACF cuts off after lag 1 and remains within the confidence bounds. This suggests that an MA(1) model with two-time difference transformation is the best selection.

**Performing MA(1) model with two-time difference transformation.

```
##
## Call:
## arima(x = df$Adjusted_Close, order = c(0, 2, 1))
##
## Coefficients:
##           ma1
##       -0.9735
## s.e.   0.0202
##
## sigma^2 estimated as 38.62:  log likelihood = -478.61,  aic = 961.21
```

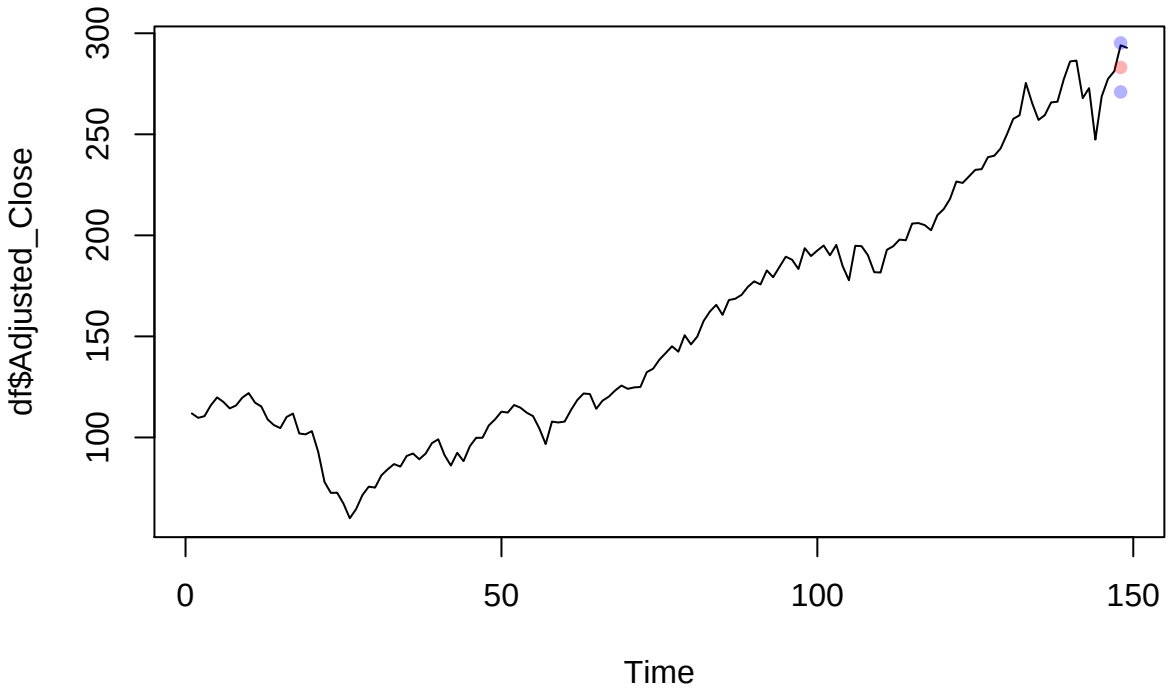

Series residuals(MA1_model)
ACF
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
Lag




```
##
##  Box-Ljung test
##
## data:  residuals(MA1_model)
## X-squared = 13.842, df = 20, p-value = 0.8384
```

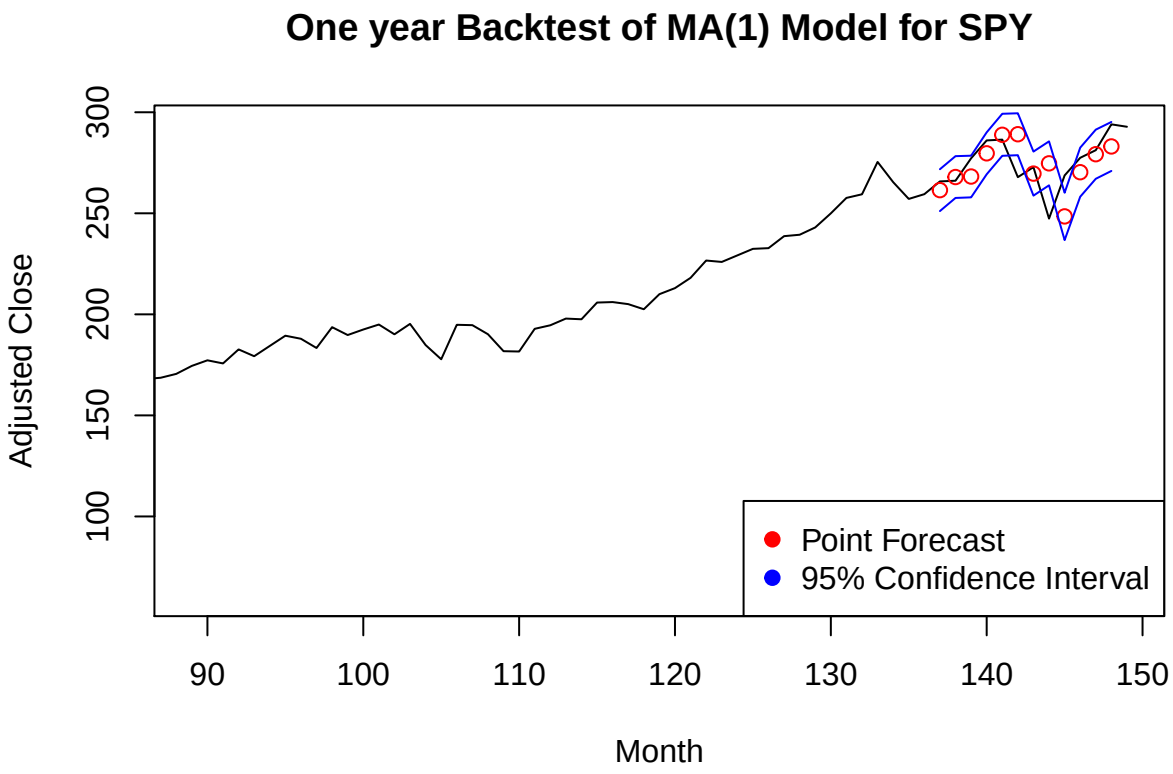
The MA(1) model shows an AIC of 961.04, and has residuals with a strong white noise process as confirmed by the ACF/PACF plots that cut off after lag 0, and remain within the confidence bounds across lag respectively. This suggests that MA(1) with a two time difference transformation is an appropriate model.

Performing a one step forecast



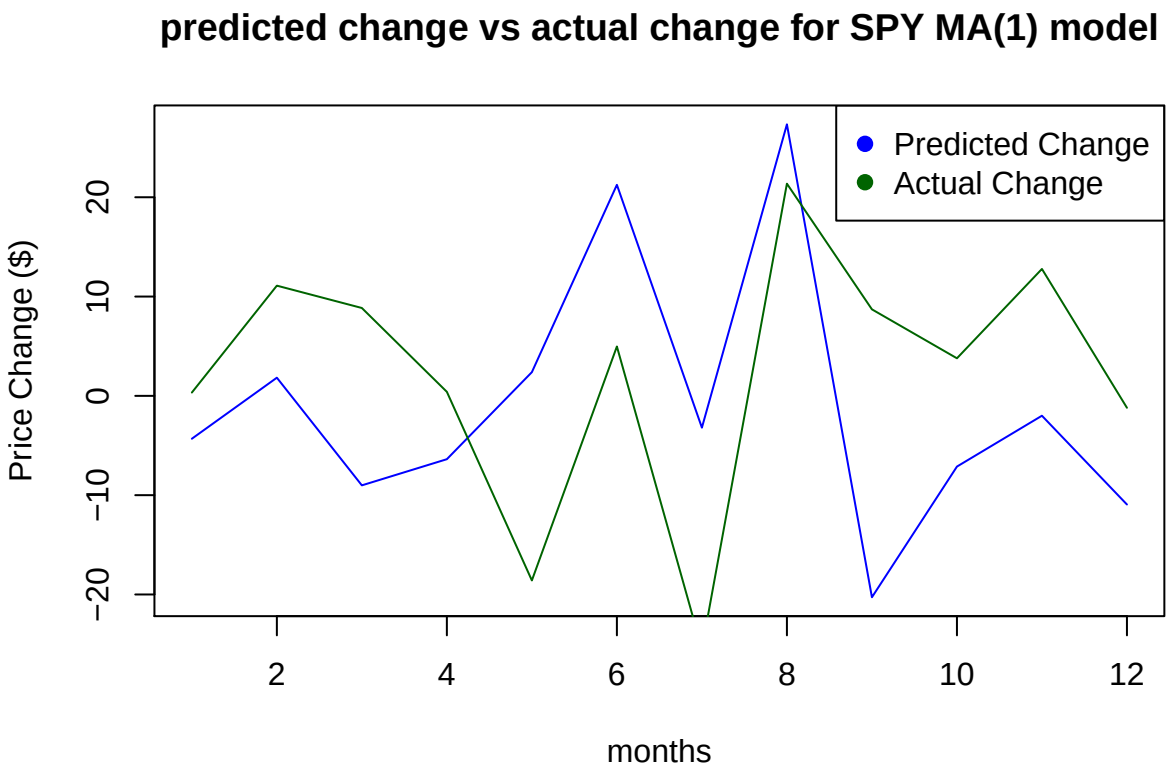
Only a one-step forecast can be performed because of the nature of an MA(1) model. the forecast appears to correctly have predicted the true actual adjusted close price with the point forecast.

Performing a one-year backtest

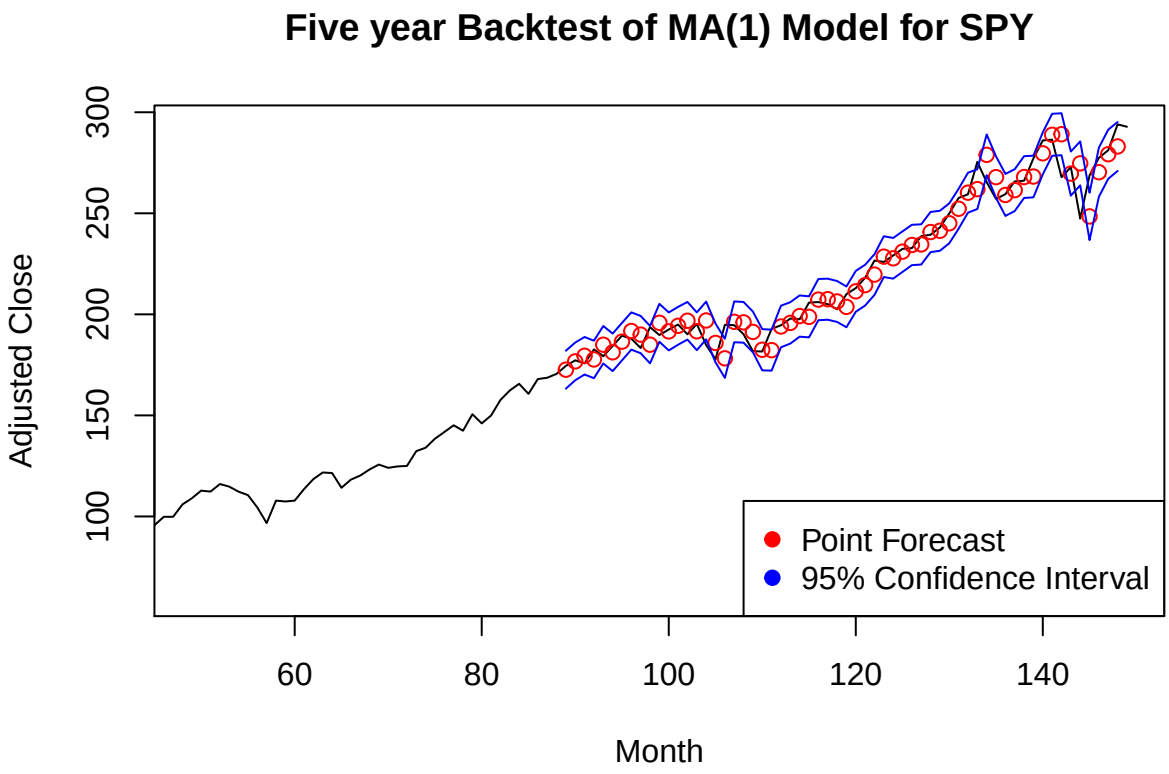


We can perform a one year backtest by iteratively performing one-step forecasts going back 12 time steps and plotting the results. We see that the point forecasts do correctly follow the shape of the monthly price movement, and only two of the price values fall outside of the 95% confidence interval for the fitted predictions. This model is performing generally well and appears to capture the shape of the price movement over a one-year backtest.

Comparing actual and predicted price changes for the MA(1) model

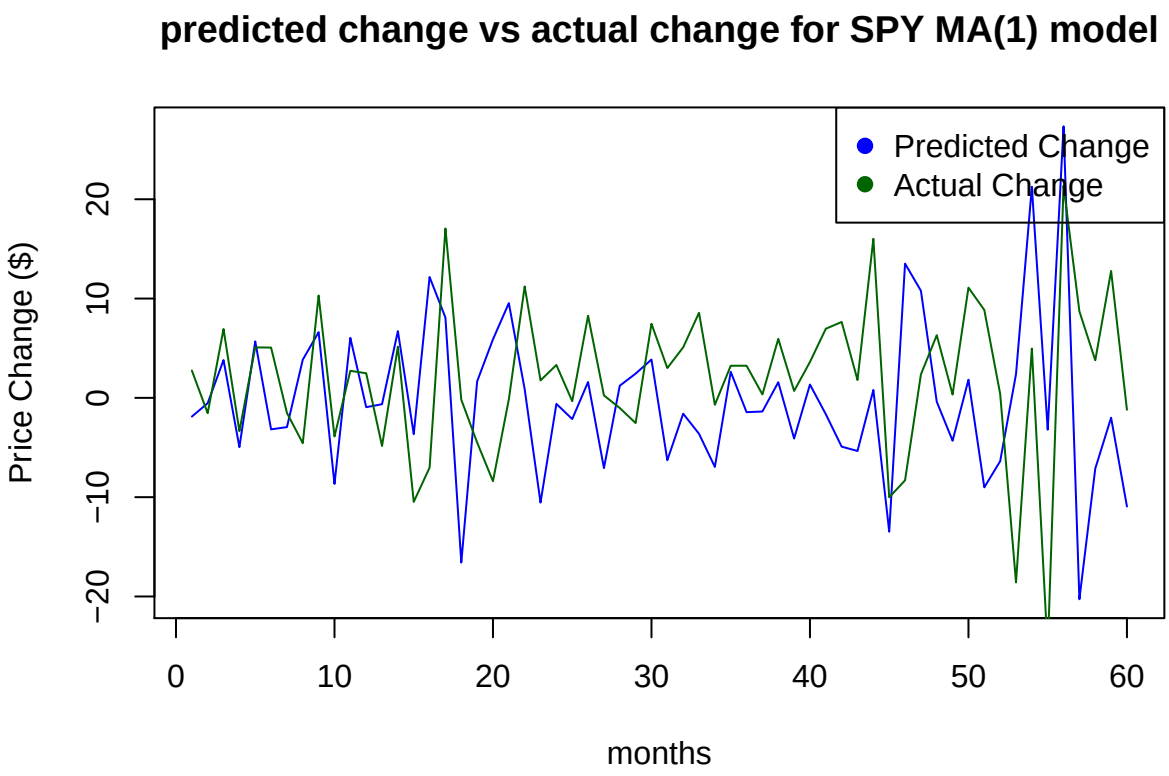


Comparing the predicted and actual price changes for the one-year backtest, we see that the model captures the shape of price movement very well. Althogh actual price changes aren't captured consistently, capturing the shape of the price movement is still very helpful, as this model will likely help an investor to avoid major losses.



We perform a five-year backtest in order to better access the performance of the model. A one year backtest is not nearly enough to make conclusions about a stock trading algorithm, and ideally we would perform at least a 10 year backtest as well, although there is not enough data to properly do so. We see in the 5 year backtest that the model captures the price within the confidence interval very well over the 5 year history of the backtest, and the point estimates follow the shape of price movement very well.

Plotting predicte vs. actual changes for the 5 year backtest



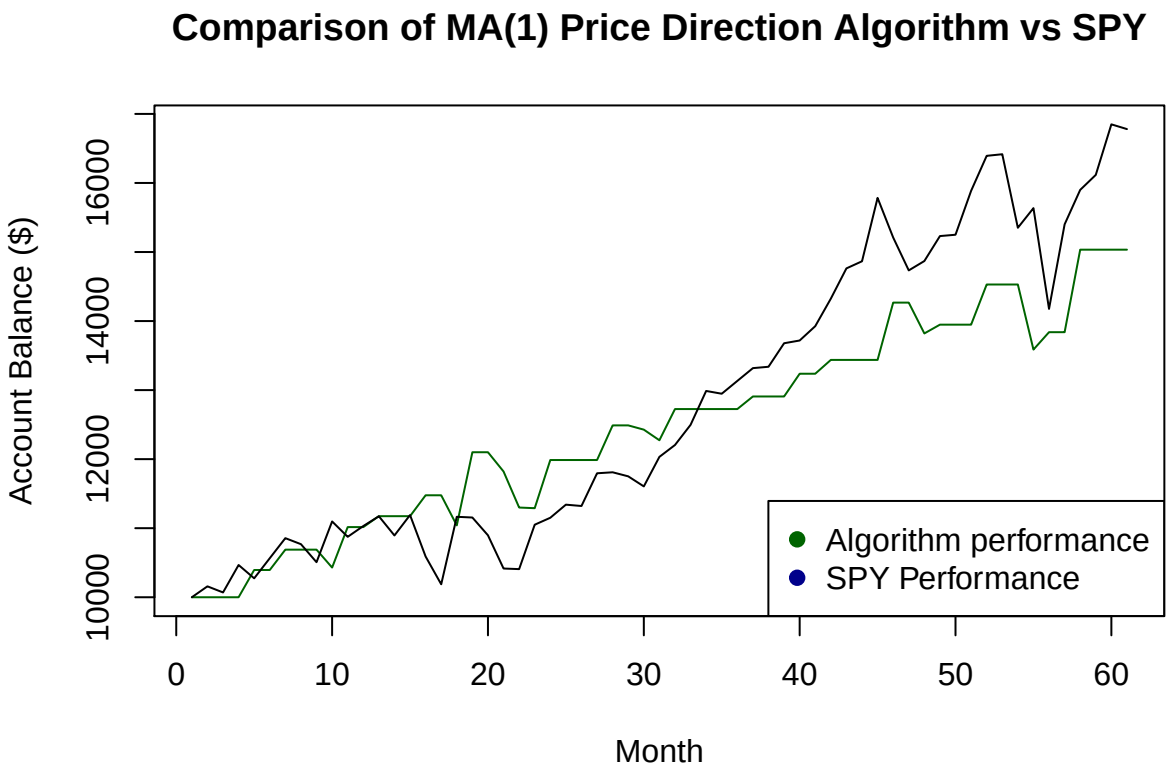
In comparing the predicted and actual changes for the 5 year backtest, we see that the model again does a very good job of capturing the shape of price movement in predicted values, although point estimates aren't consistently accurate. This is a very good sign that the model can reduce volatility in investing, as periods of high volatility of price movement were correctly predicted to be high volatility. The model is expected to help reduce volatility (drawdown) in investing in the index.

Calculating win rate of the model

```
## [1] "Win Ratio over 60 month Backtest = 48.33%"
```

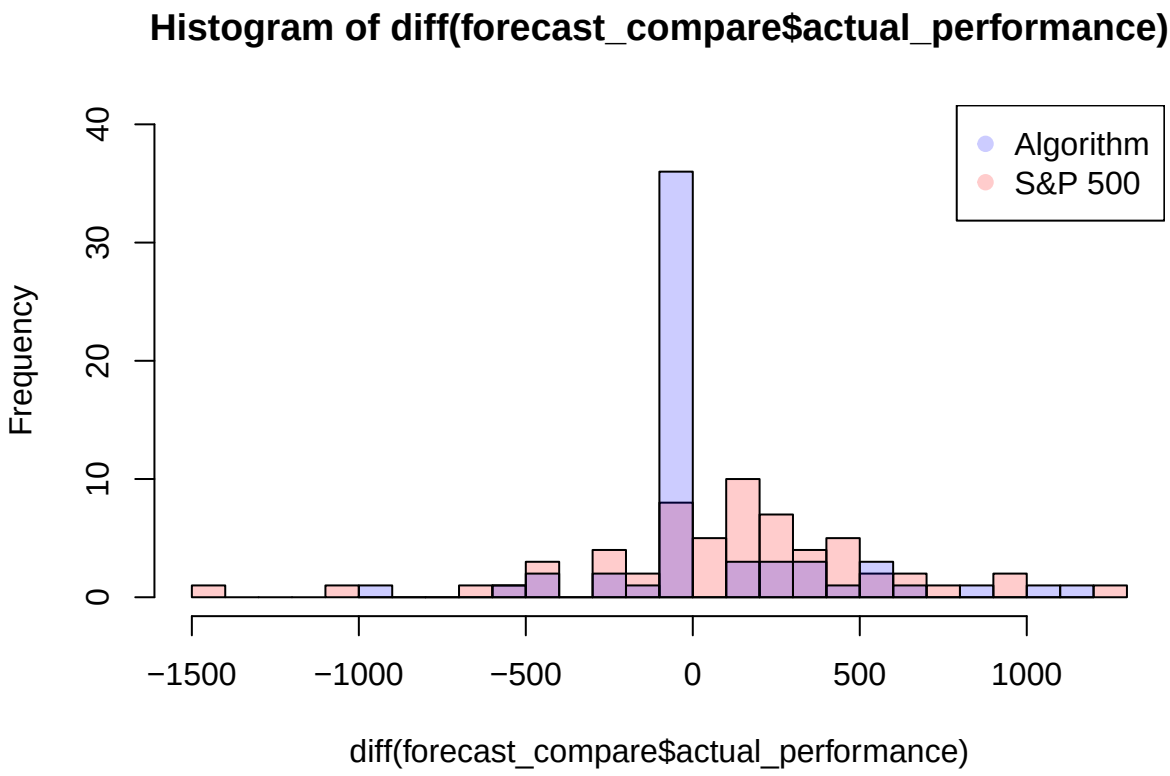
While the win rate is below 50% (the expected win rate of random investing) over the 5-year backtest, this is not necessarily a bad sign. while this strategy will certainly not help maximize returns in a bull market, the model may very well be avoiding overly aggressive buying bias in our current bull market. While the model is not expected to "beat the market" in a strong bull market, it could very well beat the market by avoiding losses during a change to a bear market. We will now simulate returns by making a simple buy and sell algorithm using expected 1-time step predictions over the full 5-year history of the backtest.

Simulating an investment of $10,000 for the algorithm and S&P 500



Looking at the simulated backtest assuming an investment of $10,000, we can see that my algorithm performs much more consistently that the actual S&P 500 with nearly the same returns. This plot is a very good sign that the algorithm is a "safer" investment than the S&P 500 index itself, as there is signficantly less volatility. The algorithm appears to "beat the market" during market dips and stagnation. We see the model beats the market during the stagnant period of months 15 to 30, and reduced losses during this period. We can see with the recent market dip in months 53 to 57, the model avoided losses far better than the index itself. As an investor, I would much rather invest my money into the algorithm rather than the S&P due to the loss protection it provides and still significant returns. During a recession, it appears that the model will produce far lower losses, or even possibly returns, due to the evidence of performance during market stagnation shown in the backtest. I would consider this model a success because it significantly reduced drawdown and volatility of the index, and I plan to explore testing for investing actual money using the model using automated trading through an electronic broker.

Additional Analysis



In the above histogram of 1-time difference changes for the simulated account balances of the algorithm and S&P 500, we can really begin to see the advantage of the algorithm over the index itself. The algorithm shows significantly more stable days and has a much lower variance than that of the index. This suggests much more stable price action as shown in the previous time series plot.

```
library(knitr)
knitr::opts_chunk$set(fig.path = 'figures/', fig.pos = 'htb!', echo = FALSE, message=FALSE, warning=FALS
knit_hooks$set(plot = function(x, options)  {
  hook_plot_tex(x, options)
})
#Loading Quantmod and Lubridate libraries
library(quantmod)
library(lubridate)

#Initializing a new environment called 'Data'
Data <- new.env()
#Using yahoo as a source for Monthly S&P 500 data and saving to the Data environment (from 01-01-07 to
getSymbols('SPY', src = 'yahoo', periodicity = 'monthly', env=Data)
#Converting the closing values to adjusted close to adjust for dividends and splits
df <- data.frame(Ad(Data$SPY))
colnames(df) <- "Adjusted_Close"
#Outputting the head and summary of the time series
head(df)

df$time <- 1:length(df$Adjusted_Close)
#Printing a summary of the SPY time series dataset
summary(df)
```

```r
par(mfrow=c(1,2))
#Plotting histogram of Adjusted close over full time series
hist(df$Adjusted_Close, breaks = 25, col = 'dark green', main = "Histogram of Spy Monthly Close", xlab =
#Plotting the time series of Adjusted Monthly Close vs. Time
ts.plot(df$Adjusted_Close, main = "S and P 500 Monthly Closing Data", ylab = "Adjusted Close Price")
#Saving year as column
df$year <- year(as.Date(rownames(df)))
#saving Month as column
df$month <- month(as.Date(rownames(df)))

#Plotting yearly and monthly boxplots for Adjusted close
par(mfrow=c(1,2))
boxplot(Adjusted_Close~year, data = df, main = "S&P 500 Adjusted Close by Year", xlab = "Year", ylab =
boxplot(Adjusted_Close~month, data = df, main = "S&P 500 Adjusted Close by Month", xlab = "Month", ylab
#Loading DPLYR library and suppressing masking messages from output
invisible(library(dplyr))

#Using DPLYR to scale Adjusted Close in yearly windows in order to compare between months relative to e
df_window_scaled <- df %>% group_by(year) %>% mutate(scaled_adjusted_close = scale(Adjusted_Close))

#Boxplot for Adjusted close by Month scaled by yearly range
boxplot(scaled_adjusted_close~month, data=df_window_scaled, main = "Spy Adjusted close by month normali
#Calculating a 1 difference monthly change value by month
monthly_change <- diff(df$Adjusted_Close)
#Eliminating the first row of the df because no change can be calculated
change_df <- df[2:length(df$Adjusted_Close),]
#Adding monthly change to the dataset
change_df$monthly_change <- monthly_change
#Plotting the Monthly Change
ts.plot(change_df$monthly_change, main = "Monthly Price Change for S&P 500", ylab = "Adjusted Close", xl
par(mfrow=c(1,2))
#Boxplot of Monthly Changes by Month
boxplot(change_df$monthly_change~change_df$month, main = "SPY Monthly Change by Month")
#Boxplot of Monthly Changes by Year
boxplot(change_df$monthly_change~change_df$year, main = "SPY Monthly Change by Year")
#Saving the number of breaks for histogram plotting as a variable - using the number of values for each
break_value <- length(df$month[df$month==1])
#Initializing color list for legend
color_list <- c()

#Initializing histogram for january
hist(change_df$monthly_change[change_df$month==1], breaks = break_value, xlim = c(min(change_df$monthly_

#Adding color for January to color list
color_list <- c(color_list, rgb(0,0,1, alpha = .5))

for(i in 2:12){
  #Generating 3 random values between 0 and 1 for RGB coloring
  rgb_values <- runif(3,0.1,1)
  #Adding histograms for the other months to the plot
  hist(change_df$monthly_change[change_df$month==i], breaks = break_value, xlim = c(min(change_df$monthl
  #Adding color of Histogram to color list
  color_list <- c(color_list, rgb(rgb_values[1],rgb_values[2],rgb_values[3], alpha = .5))
```

```
}

#Adding plot legend with Month abbreviations
legend("topright", legend = month.abb, col = color_list, pch=19)


#Testing a linear trend
linear.trendMod <- lm(Adjusted_Close~df$time, data=df)
par(mfrow=c(2,2))
plot(linear.trendMod)
#Testing a exponential trend
exp.trendMod <- lm(Adjusted_Close~df$time + I(df$time^2), data=df)
par(mfrow=c(2,2))
plot(exp.trendMod)
df$logClose <- log(df$Adjusted_Close)

#Testing a linear trend
LogExp.trendMod <- lm(logClose~df$time + I(df$time^2), data=df)
par(mfrow=c(2,2))
plot(LogExp.trendMod)
#Plotting Exponential Fit and Residuals
par(mfrow=c(1,2))
#Plotting Adjusted Close vs Exponential fit
ts.plot(df$Adjusted_Close, ylab = "Adjusted Monthly Close", main = "SPY Adjusted Monthly Close")
lines(exp(LogExp.trendMod$fitted.values)~df$time, col = 'red')
legend('bottomright', legend="Log Exponential Trendline", col='red', lty=1)
#Plotting Log Residuals for Exponential Trend
ts.plot(LogExp.trendMod$residuals, main = "Log Residuals for Exp. Trend", ylab = "Log Residual")
#Performing Dummy Seasonal Regression for Month
df$month <- as.factor(df$month)

log_seasonal=lm(logClose~df$time + I(df$time^2) + month, data=df)
summary(log_seasonal)

plot(df$Adjusted_Close~df$time, type='l', main = "Dummy Seasonal Log Regression by Month")
lines(exp(log_seasonal$fitted.values)~df$time, col = 'red')
plot(df$Adjusted_Close~df$time, type='l', main = "Harmonic Regression Testing")

# saving 12 as L to represent 12 seasons
L=12

# Two components of sin  and cosin function model
two_comp_mod = lm(logClose~time+I(sin(2*pi*time/L))+I(cos(2*pi*time/L)), data = df)
summary(two_comp_mod)
lines(exp(two_comp_mod$fitted.values)~df$time, col="red")

# Four components of sin  and cosin function model
four_comp_mod = lm(logClose~time+I(sin(2*pi*time/L))+I(cos(2*pi*time/L))+I(sin(4*pi*time/L))+I(cos(4*pi
summary(four_comp_mod)
lines(exp(four_comp_mod$fitted.values)~df$time, col="blue")
#Specifying n for extracting BIC values
n=length(LogExp.trendMod$residuals)
#Making a list of BIC values for each model test
```

```
BIC_list <- c()
BIC_list[1] <- extractAIC(LogExp.trendMod, k = log(n))[2]
BIC_list[2] <- extractAIC(log_seasonal, k = log(n))[2]
BIC_list[3] <- extractAIC(two_comp_mod, k = log(n))[2]
BIC_list[4] <- extractAIC(four_comp_mod, k = log(n))[2]
#Making a list of model names for comparison
Models <- c("Exp Trend Model", "Seasonal Model", "Two Comp Harmonic Model","Four Comp Harmonic Model")
model_comparison <- data.frame(Models, BIC_list)
model_comparison
library(forecast)
#Splitting df into training and 12-step forecasting data
old_df <- df[0:(nrow(df)-12),]
newdf <- df[(nrow(df)-11):nrow(df),]
#Trend model using training data
LogExp.trendMod <- lm(logClose~time + I(time^2), data=old_df)
#Performing 12-step forecast
decomp_forecast <- forecast(LogExp.trendMod, newdata = newdf)
decomp_forecast <- data.frame(decomp_forecast)
rownames(decomp_forecast) <- newdf$time

decomp_forecast
#Plotting the Monthly SPY df
ts.plot(df$Adjusted_Close, ylim = c(50,500), main = "One Year Forecast for Exponential Trend S&P 500 Mo
#Plotting 95% forecast confidence interval
lines(exp(decomp_forecast$Lo.95)~newdf$time, col = 'blue')
lines(exp(decomp_forecast$Hi.95)~newdf$time, col = 'blue')
#Plotting Point Estimates
points(exp(decomp_forecast$Point.Forecast)~newdf$time, col = 'dark green', cex=.5)
#Adding legend
legend('bottomright', legend=c("95% Conf Interval", "Point Forecast"), col=c("blue", "dark green"), lty=
#Loading Quantmod and Lubridate libraries
library(quantmod)
library(lubridate)

#Initializing a new environment called 'Data'
Data <- new.env()
#Using yahoo as a source for Monthly S&P 500 data and saving to the Data environment (from 01-01-07 to
getSymbols('SPY', src = 'yahoo', periodicity = 'monthly', env=Data)
#Converting the closing values to adjusted close to adjust for dividends and splits
df <- data.frame(Ad(Data$SPY))
colnames(df) <- "Adjusted_Close"
rownames(df) <- 1:length(df$Adjusted_Close)
#Outputting the head and summary of the time series
head(df)
#Constructing the adjusted close time series
ts_Adjusted <- ts(df$Adjusted_Close, start = c(2007, 1), end = c(2019, 4), frequency = 12)
#Decomposing the data
model=decompose(ts_Adjusted, type = "multiplicative", filter = NULL)
plot(model)
#plotting ACF and PACF for original data
par(mfrow=c(1,2))
acf(df$Adjusted_Close,lag.max=20)
pacf(df$Adjusted_Close,lag.max=20)
```

```r
#plotting ACF and PACF for 1-time difference data
par(mfrow=c(1,2))
acf(diff(df$Adjusted_Close),lag.max=20)
pacf(diff(df$Adjusted_Close),lag.max=20)
#plotting ACF and PACF for 2-time difference data
par(mfrow=c(1,2))
acf(diff(diff(df$Adjusted_Close)),lag.max=20)
pacf(diff(diff(df$Adjusted_Close)),lag.max=20)
library(forecast)

#Creating a two difference MA1 arima model
MA1_model=arima(df$Adjusted_Close, order = c(0,2, 1))
print(MA1_model)

#Plotting residuals ACF/PACF for white noise
acf(residuals(MA1_model))
pacf(residuals(MA1_model))

#Performing box test for white noise
Box.test(residuals(MA1_model),type="Ljung",lag=20,fitdf=0)
MA1_model=arima(ts_Adjusted[1:(length(ts_Adjusted)-1)], order = c(0,2, 1))
predictions <- forecast(MA1_model, h=1)
predictions <- as.data.frame(predictions)

ts.plot(df$Adjusted_Close, ylim=c(min(predictions$`Point Forecast`, ts_Adjusted),max(predictions$`Point
points(predictions$`Point Forecast`~rownames(predictions), col = rgb(1,0,0,alpha=.3), pch=19, cex=.8)
points(predictions$`Lo 95`~rownames(predictions), col = rgb(0,0,1,alpha=.3), pch=19, cex=.8)
points(predictions$`Hi 95`~rownames(predictions), col = rgb(0,0,1,alpha=.3), pch=19, cex=.8)

#MA(1) can only predict 1 step ahead, so performing a 12 step backtest to test for 1-year
for(i in 1:12){
  if(i==1){
    MA1_model=arima(ts_Adjusted[1:(length(ts_Adjusted)-i)], order = c(0,2, 1))
    predictions <- forecast(MA1_model, h=1)
    predict_df <- as.data.frame(predictions)
  }

  else{
    MA1_model=arima(ts_Adjusted[1:(length(ts_Adjusted)-i)], order = c(0,2, 1))
    predictions <- forecast(MA1_model, h=1)
    new_df <- as.data.frame(predictions)
    predict_df <- rbind(predict_df, new_df)
  }
}

#Plotting predictions with a 95% confidence interval vs actual price
ts.plot(df$Adjusted_Close, ylim=c(min(predict_df$`Point Forecast`, ts_Adjusted),max(predict_df$`Point F
points(predict_df$`Point Forecast`~rownames(predict_df), col = 'red')
lines(predict_df$`Lo 95`~rownames(predict_df), col = 'blue')
lines(predict_df$`Hi 95`~rownames(predict_df), col = 'blue')
legend('bottomright', legend=c('Point Forecast', '95% Confidence Interval'), col=c("red", "blue"), pch=
#Comparing actual prices to predicted prices
actual_values <- df[(length(df$Adjusted_Close)-12):length(df$Adjusted_Close),]
```

```r
forecasts <- predict_df$`Point Forecast`
forecasts <- c(0,rev(forecasts))
forecast_compare <- cbind(actual_values, forecasts)
forecast_compare <- as.data.frame(forecast_compare)

#Comparing predicted change to actual change
predicted_change <- c()
actual_change <- c()
for(i in 2:13){
  predicted_change <- c(predicted_change, forecast_compare[i,2]- forecast_compare[(i-1),1])
  actual_change <- c(actual_change, forecast_compare[i,1]- forecast_compare[(i-1),1])
  }

#Adding a dummy value because previous value is unknown
predicted_change <- c(0,predicted_change)
actual_change <- c(0,actual_change)
forecast_compare$predicted_change <- predicted_change
forecast_compare$actual_change <- actual_change

#Plotting predicted vs actual change
ts.plot(forecast_compare$predicted_change[2:length(forecast_compare$forecasts)], col='blue', main = 'pre
lines(forecast_compare$actual_change[2:length(forecast_compare$forecasts)], col='dark green')
legend('topright', legend=c('Predicted Change', 'Actual Change'), col=c("blue", "dark green"), pch=19)

#Performing 5 year backtest to properly test win ratio and prediction accuracy - 12 sample size is not

#MA(1) can only predict 1 step ahead, so performing a 12 step backtest to test for 1-year
for(i in 1:(12*5)){
  if(i==1){
    MA1_model=arima(ts_Adjusted[1:(length(ts_Adjusted)-i)], order = c(0,2, 1))
    predictions <- forecast(MA1_model, h=1)
    predict_df <- as.data.frame(predictions)
  }

  else{
    MA1_model=arima(ts_Adjusted[1:(length(ts_Adjusted)-i)], order = c(0,2, 1))
    predictions <- forecast(MA1_model, h=1)
    new_df <- as.data.frame(predictions)
    predict_df <- rbind(predict_df, new_df)
  }
}

#Plotting predictions with a 95% confidence interval vs actual price
ts.plot(df$Adjusted_Close, ylim=c(min(predict_df$`Point Forecast`, ts_Adjusted),max(predict_df$`Point Fo
points(predict_df$`Point Forecast`~rownames(predict_df), col = 'red')
lines(predict_df$`Lo 95`~rownames(predict_df), col = 'blue')
lines(predict_df$`Hi 95`~rownames(predict_df), col = 'blue')
legend('bottomright', legend=c('Point Forecast', '95% Confidence Interval'), col=c("red", "blue"), pch=
#Comparing actual prices to predicted prices
actual_values <- df[(length(df$Adjusted_Close)-60):length(df$Adjusted_Close),]
forecasts <- predict_df$`Point Forecast`
forecasts <- c(0,rev(forecasts))
forecast_compare <- cbind(actual_values, forecasts)
```

```r
forecast_compare <- as.data.frame(forecast_compare)

#Comparing predicted change to actual change
predicted_change <- c()
actual_change <- c()
for(i in 2:61){
  predicted_change <- c(predicted_change, forecast_compare[i,2]- forecast_compare[(i-1),1])
  actual_change <- c(actual_change, forecast_compare[i,1]- forecast_compare[(i-1),1])
  }

#Adding a dummy value because previous value is unknown
predicted_change <- c(0,predicted_change)
actual_change <- c(0,actual_change)
forecast_compare$predicted_change <- predicted_change
forecast_compare$actual_change <- actual_change

#Plotting predicted vs actual change
ts.plot(forecast_compare$predicted_change[2:length(forecast_compare$forecasts)], col='blue', main = 'pr
lines(forecast_compare$actual_change[2:length(forecast_compare$forecasts)], col='dark green')
legend('topright', legend=c('Predicted Change', 'Actual Change'), col=c("blue", "dark green"), pch=19)
#Testing accuracy of price direcion prediction over a 5 year backtest using 1 step prediction of MA(1) m
year5_compare <- forecast_compare[2:length(forecast_compare$actual_values),]
year5_compare$Predicted_Direction <- sign(year5_compare$predicted_change)
year5_compare$Actual_Direction <- sign(year5_compare$actual_change)

#Calculating win loss by flagging each correctly or incorrectly predicted price movement
WinLoss <- c()
for(i in 1:length(year5_compare$actual_values)){
  if(year5_compare$Predicted_Direction[i]==year5_compare$Actual_Direction[i]){
    WinLoss <- c(WinLoss, 1)
  }
  else{
    WinLoss <- c(WinLoss, 0)
  }
}

#Calculating Win Ratio
year5_compare$WinLoss <- WinLoss
winratio <- round(sum(WinLoss)/length(WinLoss),4)*100
print(paste("Win Ratio over 60 month Backtest = ",winratio,"%",sep=''))
#Backtesting Algorithm and commparing to actual SPY performance assuming an investment of 10000
#Assuming investment of $10000
account_balance <- c(10000)
#Starting with 0 shares and cash
shares <- c(0)

#Iterating over the full dataframe of predictions and actual price values
for (i in 1:(length(forecast_compare$actual_values)-1)){
  #If the next predicted change is positive...
  if(sign(predicted_change[(i+1)])==1){
    #If account is all cash then purchase shares using full balance
    if(shares[i]==0){
      shares <- c(shares, account_balance[i]/(forecast_compare$actual_values[i]))
```

```
      account_balance <- c(account_balance, account_balance[i])
    }
    #Else hold shares and report values
    else{
      shares <- c(shares, shares[i])
      account_balance <- c(account_balance, shares[i]*forecast_compare$actual_values[i])
    }
  }
  #Else if the next predicted price change is negative
  else{
    #If shares are unowned then hold account in cash
    if(shares[i]==0){
      shares <- c(shares, 0)
      account_balance <- c(account_balance, account_balance[i])
    }
    #If shares are owned then sell shares
    else{
      shares <- c(shares, 0)
      account_balance <- c(account_balance, shares[i]*forecast_compare$actual_values[i])
    }
  }
}
#Add backtest account and shares to forecast compare df
forecast_compare$algo_balance <- account_balance
forecast_compare$shares <- shares

#Adding actual performance assuming an investment of 10000
investment <- 10000
#Saving first price to cacluate performance
starting_value <- forecast_compare$actual_values[1]
actual_performance <- (forecast_compare$actual_values/starting_value)*investment
forecast_compare$actual_performance <- actual_performance

#Saving min and max across both lists for plotting
minylim <- min(forecast_compare$actual_performance, forecast_compare$algo_balance)
maxylim <- max(forecast_compare$actual_performance, forecast_compare$algo_balance)

#Plotting actual vs algorithmic based SPY performance
ts.plot(forecast_compare$algo_balance, xlab = "Month", ylab="Account Balance ($)", main = "Comparison o
lines(forecast_compare$actual_performance)
legend('bottomright', legend=c('Algorithm performance', 'SPY Performance'), col=c("dark green", "dark b

hist(diff(forecast_compare$actual_performance), breaks=20, col=rgb(1,0,0,alpha=.2), ylim=c(0,40))
hist(diff(forecast_compare$algo_balance), breaks=20, col=rgb(0,0,1,alpha=.2), add=TRUE)
legend('topright', legend=c("Algorithm", "S&P 500"), col=c(rgb(0,0,1,alpha=.2), rgb(1,0,0,alpha=.2)), p
```