Group Project Report

Group D8

3/16/2020

Introduction:

In this project, we seek to inspect a well-known time series, the S&P 500 Index, in order to identify trends and attempt to make a future forecast.

Abstract

Using one year of historical closing price values from the S&P 500 Index, we attempt to fit a variety of models and analyze the trends. From the portmanteu test on the original data, we could say that the data is not generated from a random noise process at the 5% level. To remove an upward trend, the data was differenced, we could no longer reject the null hypothesis that the series is a white noise process.

TODO: Explain about all the models

Using cross validation, we identified the top ARIMA models to model this sequence using RMSE and AIC as metrics. The best model by this criteria is ARIMA(1, 1, 0). Using this model, we attempt to predict the next 5 closing price values and demonstrate the comparison to the actual values.

Data

The S&P 500 Index is a collection of some of the largest US stocks from a variety of industries, weighted by their market capitalization. A closing price is the last price at which the stock was trading during the previous trading day. In the case of S&P 500, this is the last weighted calculation of the index for the trading day, calculated at 4PM ET/EDT. In this report, we use the S&P 500 index closing price for January 1, 2019 to Dec 31st 2019 for model training, and January 2 to January 8 2020 for model prediction.

The data source is Yahoo Finance, which we used because it is a free, trusted source for accurate stock data. We used the quantmod package (Quantitative Financial Modelling & Trading Framework for R) which has an API for importing the data from Yahoo.

The S&P 500 is superior to individual stocks for modeling, as it is an aggregate – this allows us to obtain a "big picture" in comparison to individual stocks, which are more volatile and subject to more extreme fluctuation. Our motivation for using this data was to obtain a prediction model based on historical price data. Such a model would not only be useful for profiting from market trades, but also as an indicator for US stock movement, and therefore GDP growth and economic sentiment. Of course, a simple model would be unlikely, so we used methods such as Neural Networks to model the data.

Analysis:

```r
library(doSNOW)
library(forecast)
library(ggfortify)
library(gtools)
library(parallel)
library(quantmod)
library(tcltk)
options('getSymbols.warning4.0' = F)
```

We aim to predict the daily closing values of The S&P 500 Index from 2020–01–02 to 2020–01–08. In this case, we will use one year of S&P 500 historical closing price values from 2019–01–01 to 2019–12–31, which can be accessed through yahoo finance. After downloading, the dataset looks like this:

Using data from Yahoo! Finance

```r
getSymbols(Symbols = '^GSPC',
           src     = 'yahoo',
           auto.assign = T,
           from    = '2019-01-01',
           to      = '2020-01-01')
```

```
## [1] "^GSPC"
```

```r
data <- GSPC[, 'GSPC.Close']
head(data)
```

```
##            GSPC.Close
## 2019-01-02    2510.03
## 2019-01-03    2447.89
## 2019-01-04    2531.94
## 2019-01-07    2549.69
## 2019-01-08    2574.41
## 2019-01-09    2584.96
```

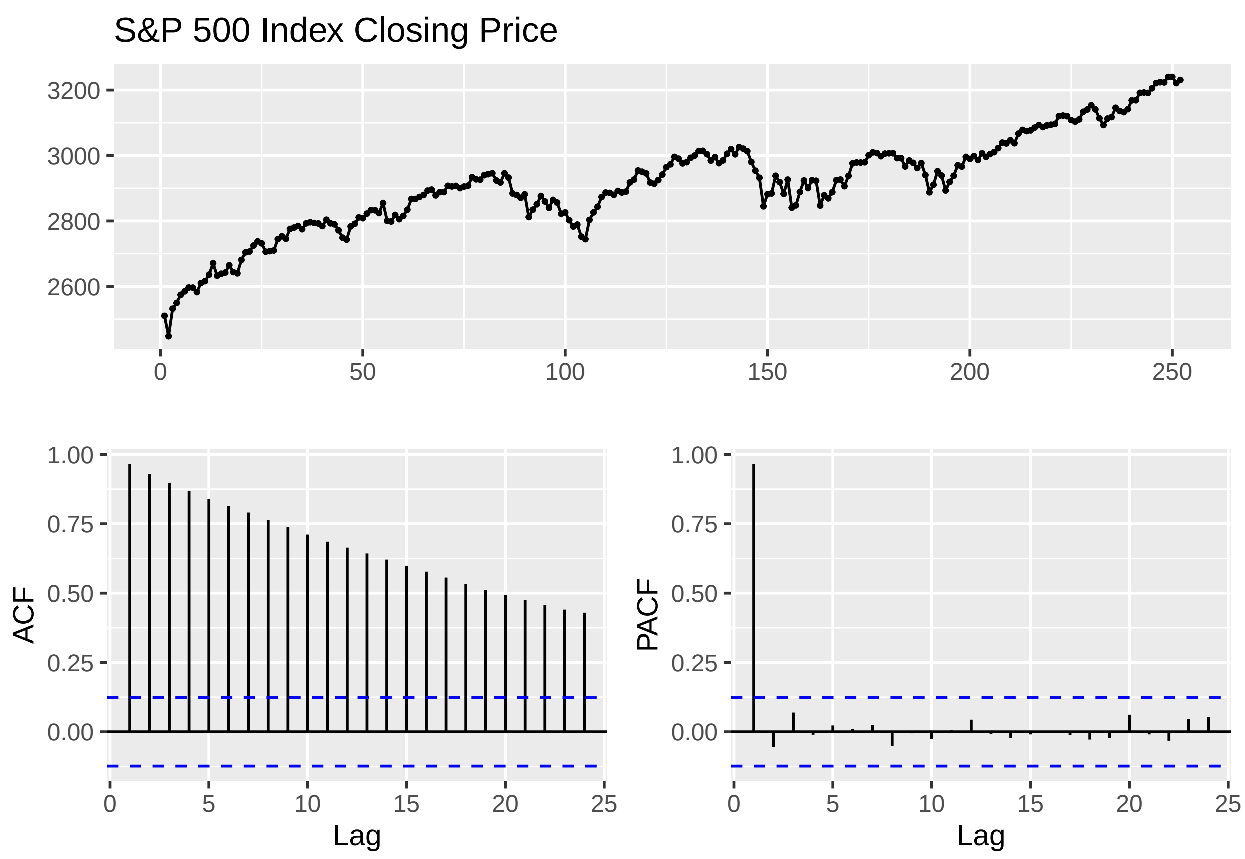
```r
tail(data)
```

```
##            GSPC.Close
## 2019-12-23    3224.01
## 2019-12-24    3223.38
## 2019-12-26    3239.91
## 2019-12-27    3240.02
## 2019-12-30    3221.29
## 2019-12-31    3230.78
```

```r
sp500 <- ts(data)
```

Time Series Plot

```r
ggtsdisplay(sp500, main = 'S&P 500 Index Closing Price')
```



The time series plot shows an increasing linear trend with no obvious seasonality.

ACF plot shows a very slow decay in time suggesting that the series might not be stationary.

PACF cuts off at lag 1

Portmanteau Test

```
Box.test(sp500, lag = 12, type = 'Box-Pierce')
```

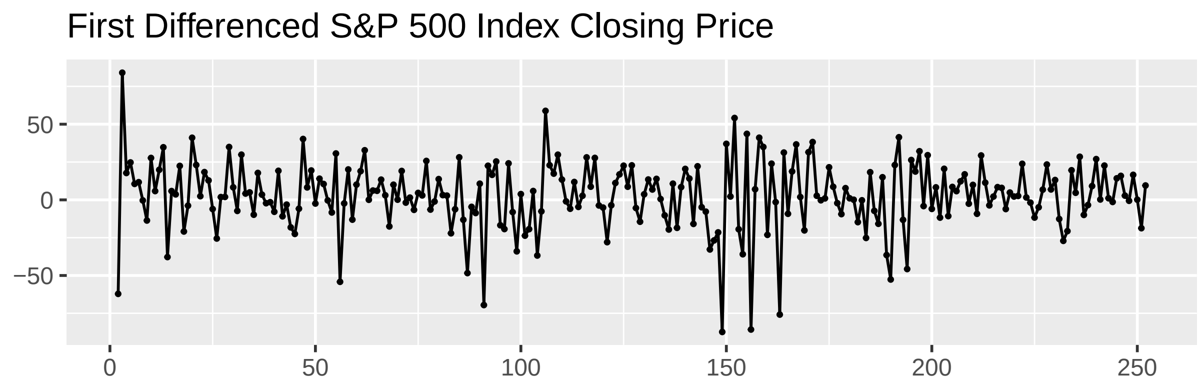
```
##
##  Box-Pierce test
##
## data:  sp500
## X-squared = 1990, df = 12, p-value < 2.2e-16
```

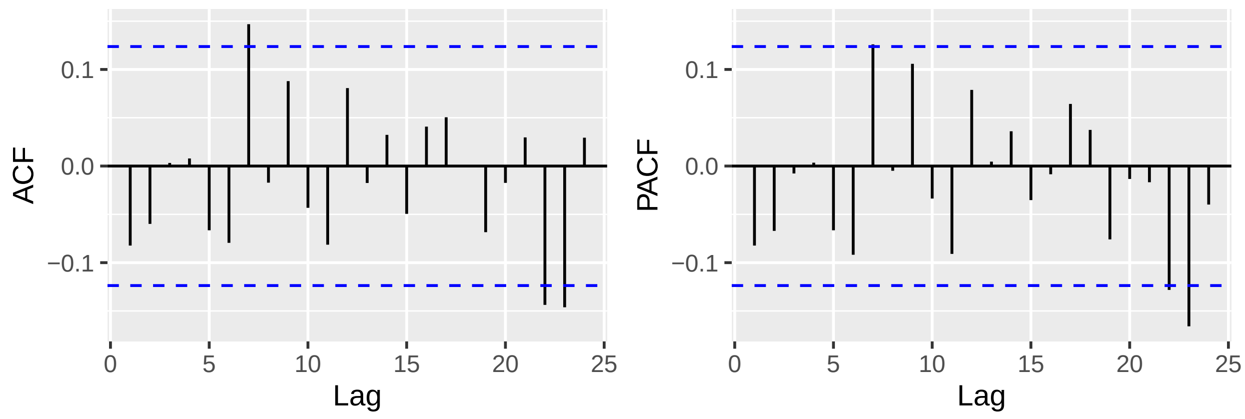
The p-value of the portmanteau test for residuals is less than 0.05. Therefore, we have sufficient evidence to reject the null hypothesis that the series is white noise.

Then, to make the series stationary, we compute the differences between consecutive observations.

First Differenced Time Series Plot

```
sp500 %>%
  diff() %>%
  ggtsdisplay(main = 'First Differenced S&P 500 Index Closing Price')
```





The differenced series plot shows variation without apparend trend.

Although both ACF and pacf plots show significant values at lag 7, 22,and 23, they drop to zero quickly and most of values are not significant. Based on that, we could probably conclude that the series is a white noise process.

Portmanteau Test

```
sp500 %>%
  diff() %>%
  Box.test(lag = 12, type = 'Box-Pierce')
```

```
##
##  Box-Pierce test
##
## data:  .
## X-squared = 16.506, df = 12, p-value = 0.1692
```

The p-value of the portmanteau test for residuals is greater than 0.05. Therefore, we do not have sufficient evidence to reject the null hypothesis at 5% significant level, so we cannot conclude that the differenced series is not a white noise process.

Modeling

In the next part, we will use four different forecasting techniques: Regression, Exponential Smoothing, Box-Jenkins Methodology, and Feedforward Neural Network.

Regression (Linear model and Cubic Spline)

Since our time series data has a linear trend, we will try to model it using a trend variable.

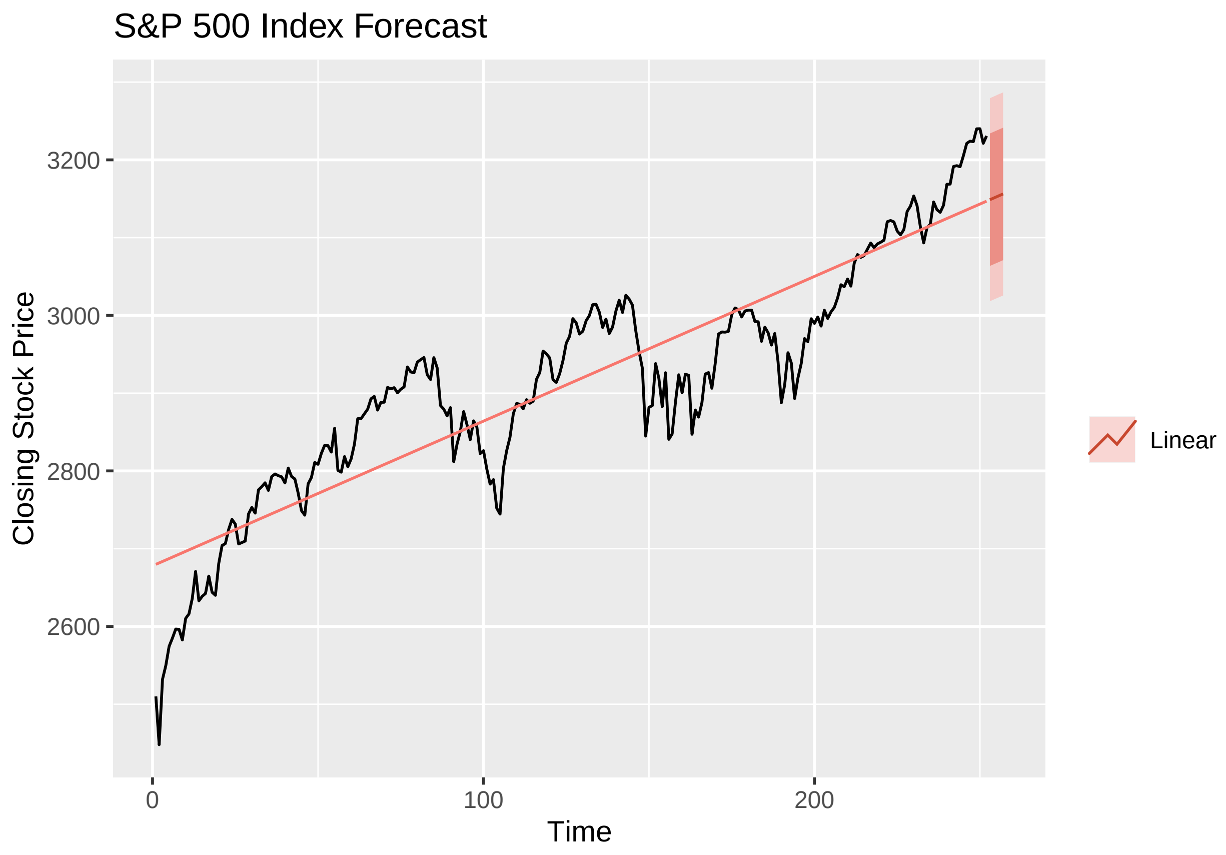
```r
tslm_linear <- tslm(sp500 ~ trend)
summary(tslm_linear)
```

```
##
## Call:
## tslm(formula = sp500 ~ trend)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -233.804  -49.283    5.163   48.633  115.277
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.678e+03  8.308e+00  322.34   <2e-16 ***
## trend       1.861e+00  5.693e-02   32.68   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 65.75 on 250 degrees of freedom
## Multiple R-squared:  0.8103, Adjusted R-squared:  0.8096
## F-statistic:  1068 on 1 and 250 DF,  p-value: < 2.2e-16
```

```r
fcast_linear <- forecast(tslm_linear, h = 5)
```

Forecast Plot

```r
autoplot(sp500) +
  autolayer(fitted(tslm_linear), series = 'Linear') +
  autolayer(fcast_linear, series = 'Linear') +
  xlab('Time') +
  ylab('Closing Stock Price') +
  ggtitle('S&P 500 Index Forecast') +
  guides(colour = guide_legend(title = ' '))
```



As we can see the series is not exactly linear so it might be better to use a non-linear regression like cubic spline.

```
t <- time(sp500)
t1 <- ts(pmax(0, t - 37))
t2 <- ts(pmax(0, t - 205))
tslm_spline <- tslm(sp500 ~ t + I(t^2) + I(t^3) + I(t1^3) + I(t2^3))
summary(tslm_spline)
```

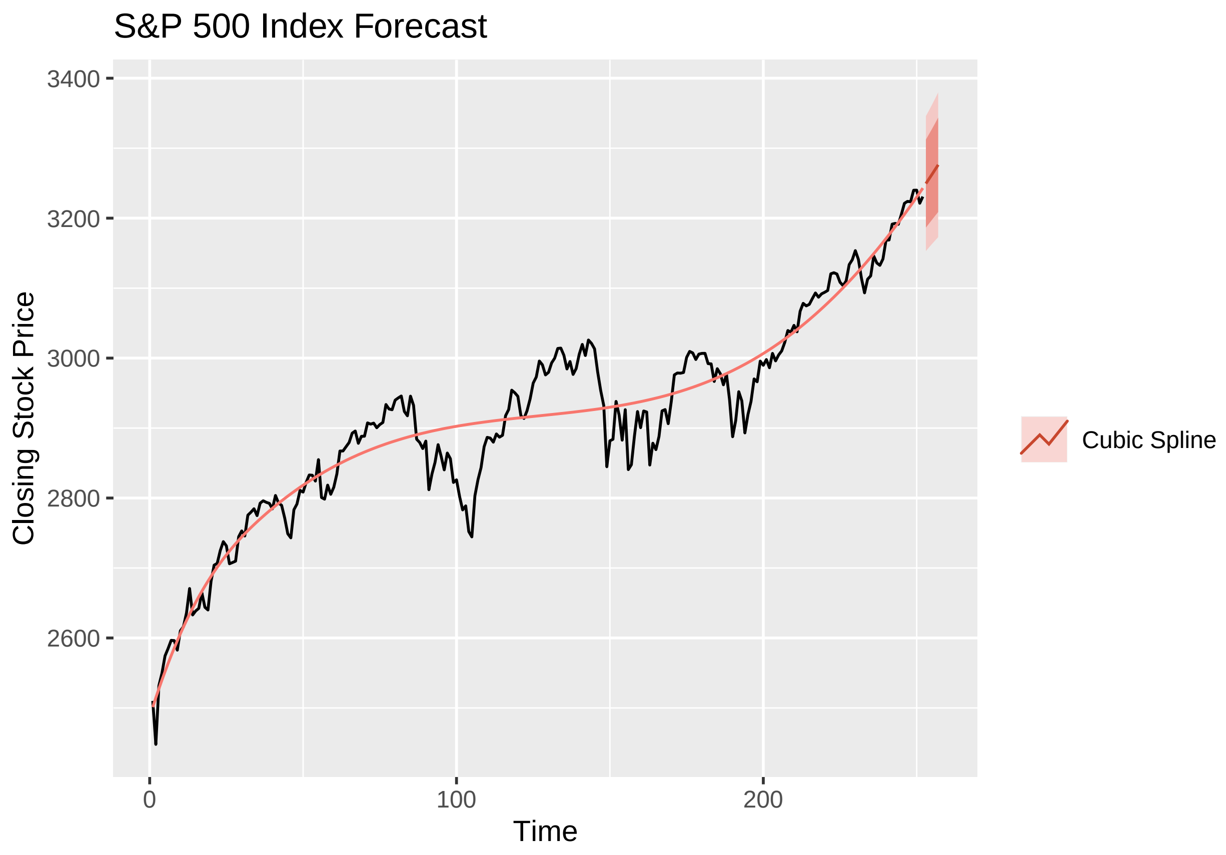
```
##
## Call:
## tslm(formula = sp500 ~ t + I(t^2) + I(t^3) + I(t1^3) + I(t2^3))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -161.629  -22.244    4.592   25.623  100.431
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.487e+03  2.263e+01 109.939  < 2e-16 ***
## t            1.406e+01  2.534e+00   5.547  7.5e-08 ***
## I(t^2)      -2.367e-01  7.772e-02  -3.045  0.00258 **
## I(t^3)       1.788e-03  7.287e-04   2.454  0.01483 *
## I(t1^3)     -1.646e-03  7.397e-04  -2.225  0.02696 *
## I(t2^3)     -1.004e-04  3.533e-04  -0.284  0.77644
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 44.65 on 246 degrees of freedom
## Multiple R-squared:  0.9139, Adjusted R-squared:  0.9122
## F-statistic: 522.5 on 5 and 246 DF,  p-value: < 2.2e-16
```

```
fcast_spline <- forecast(tslm_spline,
                         newdata = data.frame(t = t[length(t)] + seq(5),
                                              t1 = t1[length(t1)] + seq(5),
                                              t2 = t2[length(t2)] + seq(5)))
```

Forecast Plot

```
autoplot(sp500) +
  autolayer(fitted(tslm_spline), series = 'Cubic Spline') +
  autolayer(fcast_spline, series = 'Cubic Spline') +
  xlab('Time') +
  ylab('Closing Stock Price') +
  ggtitle('S&P 500 Index Forecast') +
  guides(colour = guide_legend(title = ' '))
```



Five-step Time Series Cross-Validation

We will not be doing CV for cubic spline because we would have to determine the location and number of knots for each fold.

```
ff_linear <- function(x, h) {
  forecast(tslm(x ~ trend), h = h)
}
e_linear <- tsCV(sp500, ff_linear, h = 5)
rmse_linear <- sqrt(mean(e_linear^2, na.rm = T))
```

Exponential Smoothing

Simple Exponential Smoothing

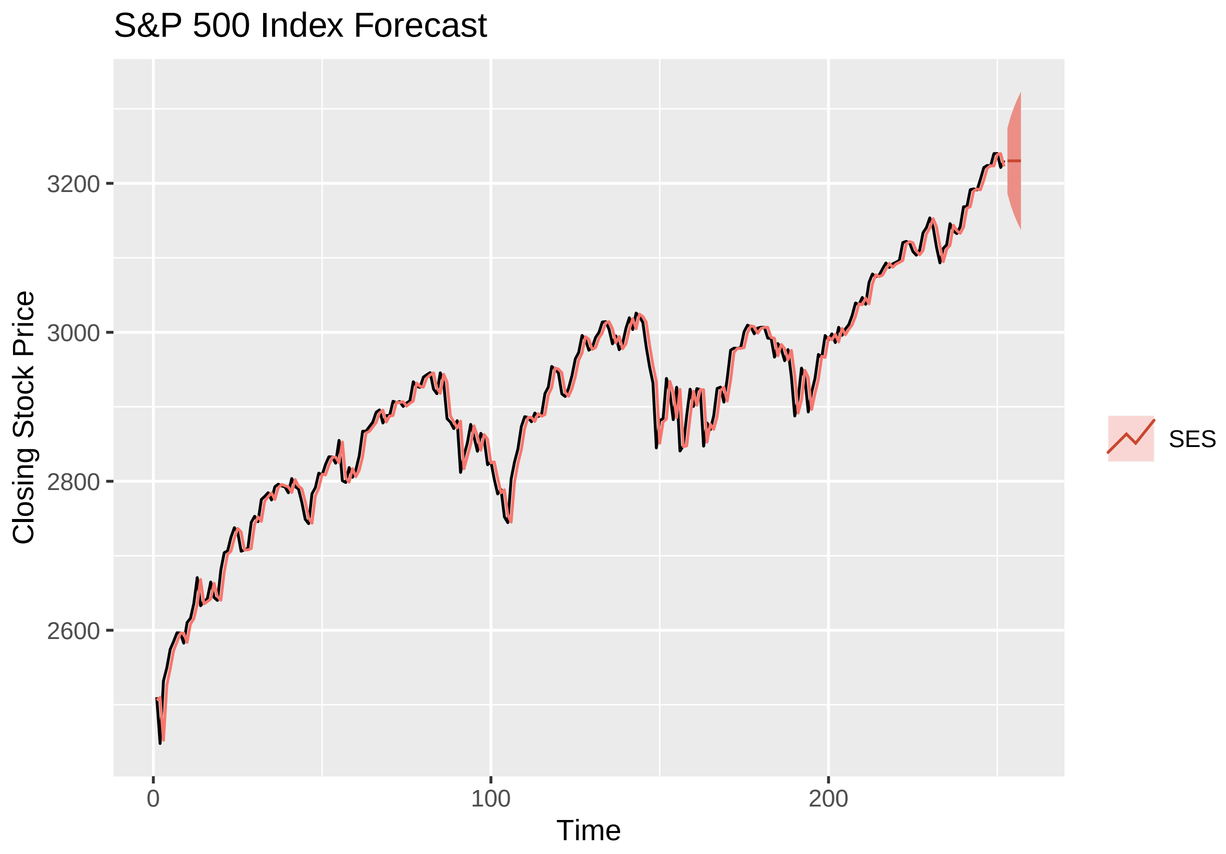
We will use this method because it is suitable for forecasting data with no clear seasonal pattern.

```
fcast_ses <- ses(sp500, h = 5, level = 95)
summary(fcast_ses)
```

```
##
## Forecast method: Simple exponential smoothing
##
## Model Information:
## Simple exponential smoothing
##
## Call:
##  ses(y = sp500, h = 5, level = 95)
##
##   Smoothing parameters:
##     alpha = 0.929
##
##   Initial states:
##     l = 2506.0199
##
##   sigma:  22.4816
##
##      AIC     AICc      BIC
## 2966.207 2966.304 2976.796
##
## Error measures:
##                    ME    RMSE      MAE       MPE      MAPE      MASE
## Training set 3.093363 22.3922 16.32749 0.1050945 0.5681546 0.9989009
##                     ACF1
## Training set -0.01663231
##
## Forecasts:
##     Point Forecast    Lo 95    Hi 95
## 253         3230.2 3186.137 3274.263
## 254         3230.2 3170.057 3290.343
## 255         3230.2 3157.448 3302.952
## 256         3230.2 3146.723 3313.678
## 257         3230.2 3137.226 3323.174
```

Forecast Plot

```
autoplot(sp500) +
  autolayer(fitted(fcast_ses), series = 'SES') +
  autolayer(fcast_ses, series = 'SES') +
  xlab('Time') +
  ylab('Closing Stock Price') +
  ggtitle('S&P 500 Index Forecast') +
  guides(colour = guide_legend(title = ' '))
```



Holt's linear trend method

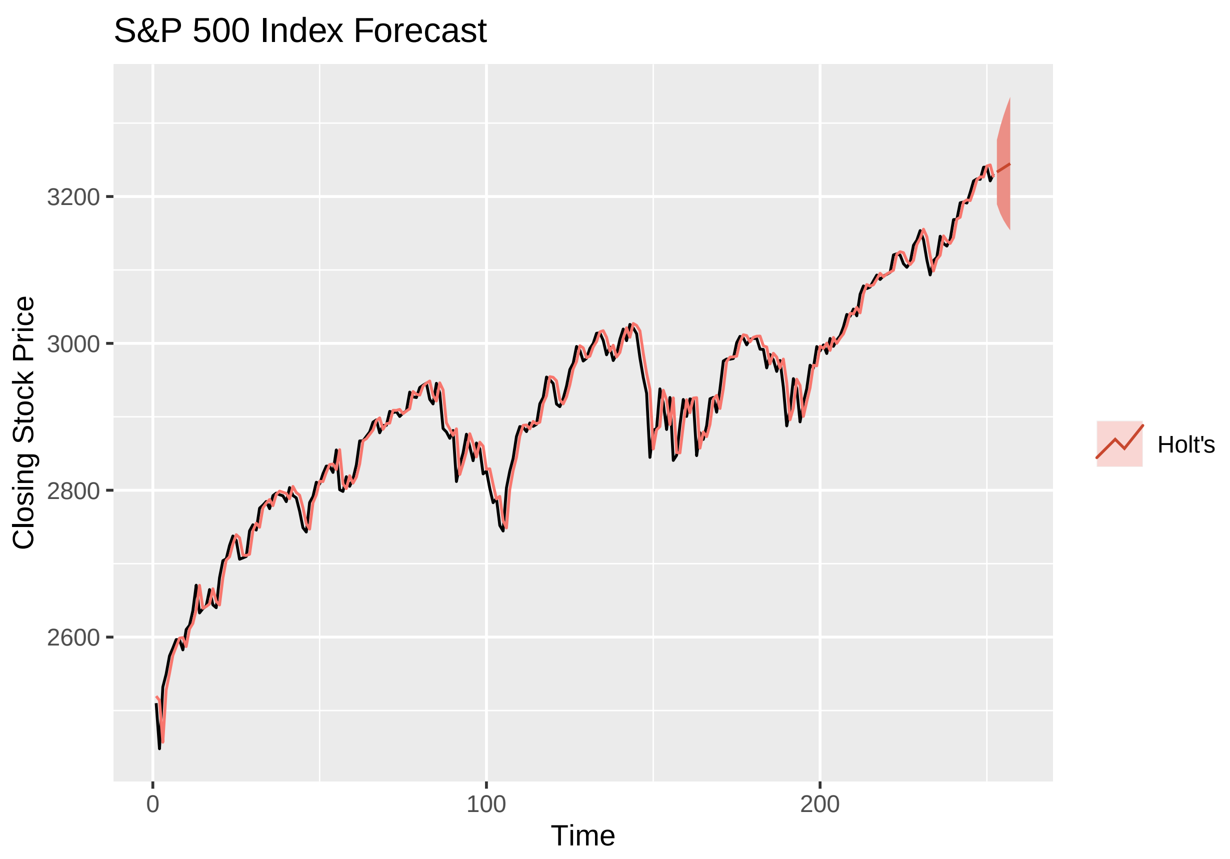
```
holt_fcast <- holt(sp500, h = 5, level = 95)
summary(holt_fcast)
```

```
##
## Forecast method: Holt's method
##
## Model Information:
## Holt's method
##
## Call:
##  holt(y = sp500, h = 5, level = 95)
##
##   Smoothing parameters:
##     alpha = 0.9079
##     beta  = 1e-04
##
##   Initial states:
##     l = 2516.7752
##     b = 2.9005
##
##   sigma:  22.3699
```

```
##
##      AIC     AICc      BIC
## 2965.674 2965.917 2983.321
##
## Error measures:
##                       ME     RMSE      MAE        MPE      MAPE      MASE
## Training set -0.07930167 22.19167 15.98918 -0.0043808 0.5569463 0.9782034
##                     ACF1
## Training set 0.008240877
##
## Forecasts:
##     Point Forecast    Lo 95    Hi 95
## 253       3233.256 3189.412 3277.100
## 254       3236.154 3176.934 3295.375
## 255       3239.053 3167.694 3310.412
## 256       3241.952 3160.235 3323.668
## 257       3244.850 3153.947 3335.753
```

Forecast Plot

```r
autoplot(sp500) +
  autolayer(fitted(holt_fcast), series = "Holt's") +
  autolayer(holt_fcast, series = "Holt's") +
  xlab('Time') +
  ylab('Closing Stock Price') +
  ggtitle('S&P 500 Index Forecast') +
  guides(colour = guide_legend(title = ' '))
```



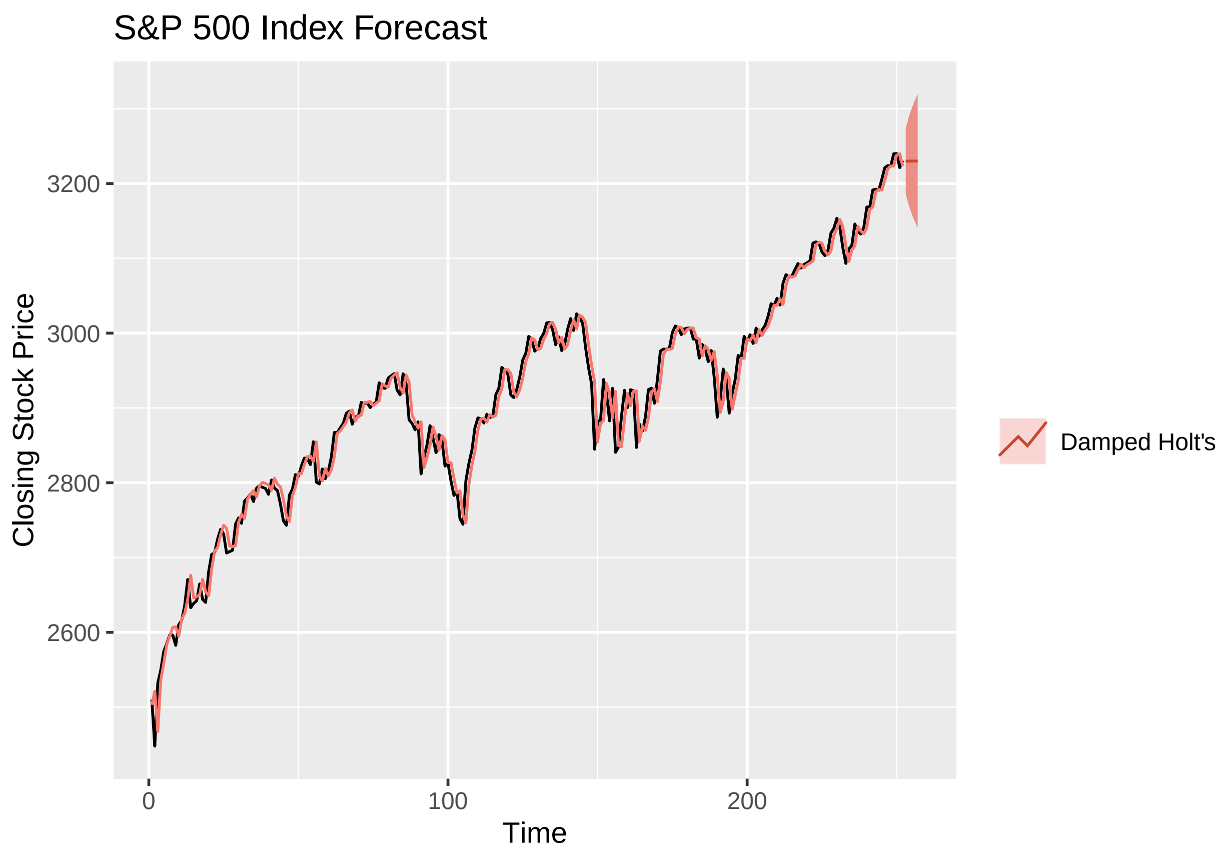
```r
holt_fcast_damped <- holt(sp500, h = 5, level = 95, damped = T)
summary(holt_fcast_damped)
```

```
##
## Forecast method: Damped Holt's method
##
## Model Information:
## Damped Holt's method
##
## Call:
##  holt(y = sp500, h = 5, damped = T, level = 95)
##
##   Smoothing parameters:
##     alpha = 0.8907
##     beta  = 1e-04
##     phi   = 0.9729
##
##   Initial states:
##     l = 2490.7652
##     b = 12.594
##
##   sigma:  22.3338
##
##      AIC     AICc      BIC
## 2965.841 2966.184 2987.017
##
```

```
## Error measures:
##                    ME     RMSE      MAE        MPE      MAPE      MASE
## Training set 1.276967 22.11111 16.06489 0.03803907 0.5584818 0.9828349
##                     ACF1
## Training set 0.003062467
##
## Forecasts:
##     Point Forecast    Lo 95    Hi 95
## 253       3229.995 3186.221 3273.768
## 254       3230.021 3171.398 3288.644
## 255       3230.047 3159.636 3300.457
## 256       3230.071 3149.580 3310.563
## 257       3230.096 3140.651 3319.540
```

Forecast Plot

```r
autoplot(sp500) +
  autolayer(fitted(holt_fcast_damped), series = "Damped Holt's") +
  autolayer(holt_fcast_damped, series = "Damped Holt's") +
  xlab('Time') +
  ylab('Closing Stock Price') +
  ggtitle('S&P 500 Index Forecast') +
  guides(colour = guide_legend(title = ' '))
```



Five-step Time Series Cross-Validation

```r
tsCV_ses <- tsCV(sp500, ses, h = 5)
rmse_ses <- sqrt(mean(tsCV_ses^2, na.rm = T))

tsCV_holt <- tsCV(sp500, holt, h = 5)
rmse_holts <- sqrt(mean(tsCV_holt^2, na.rm = T))

tsCV_holt_damped <- tsCV(sp500, holt, damped = T, h = 5)
rmse_damped_holts <- sqrt(mean(tsCV_holt_damped^2, na.rm = T))
```

Box-Jenkins Methodology

Five-step Time Series Cross-Validation for ARIMA Model Selection

We will use a five-step time series cv to measure the goodness of prediction for all possible models (ARIMA(p, 1, q) where 0 <= p <= 4 and 0 <= q <= 4).

For every i = 1, . . . , 251: a) Train the model on every point before i b) Compute the test error on the held out point i b) Average the test errors

To evaluate the effectiveness of our methods, we will use the root mean square error (RMSE) and Akaike information criterion (AIC) metrics. For both metrics, the lower the value, the better the prediction.

```r
cl <- makeSOCKcluster(detectCores() - 1)
registerDoSNOW(cl)

order <- permutations(5, 2, 0:4, repeats = T)
order <- lapply(1:nrow(order), function(i) order[i, ]) # Convert matrix to list
pb <- tkProgressBar(max = length(order))
opts <- list(progress = function(n) setTkProgressBar(pb, n))

score <- foreach(pq = order, .options.snow = opts, .packages = c('forecast')) %dopar% {
  p <- pq[1]
  q <- pq[2]
  aic <- Arima(sp500, order = c(p, 1, q))$aic
  farima <- function(x, h) {
    forecast(Arima(x, order = c(p, 1, q)), h = h)
  }
  e <- tsCV(sp500, farima, h = 5)
  rmspe <- sqrt(mean(e^2, na.rm = T))
  return(c(p, q, aic, rmspe))
}

close(pb)
stopCluster(cl)

result <- data.frame(Reduce(rbind, score), row.names = NULL)
colnames(result) <- c('p', 'q', 'AIC', 'RMSE')
print(result)
```

```
##    p q      AIC     RMSE
## 1  0 0 2277.018 36.55985
## 2  0 1 2277.884 37.08191
## 3  0 2 2279.511 37.57035
## 4  0 3 2281.405 38.46575
## 5  0 4 2283.396 39.51089
## 6  1 0 2277.979 35.82020
## 7  1 1 2279.671 36.12562
## 8  1 2 2280.976 36.98181
## 9  1 3 2282.969 38.10640
## 10 1 4 2284.969 38.70294
## 11 2 0 2279.444 36.54921
## 12 2 1 2281.417 37.85181
## 13 2 2 2274.345 38.94127
## 14 2 3 2282.634 38.26201
## 15 2 4 2284.428 39.58919
## 16 3 0 2281.381 36.63444
## 17 3 1 2280.847 38.12661
## 18 3 2 2282.782 38.67445
## 19 3 3 2284.544 38.85263
## 20 3 4 2282.708 39.23841
## 21 4 0 2283.235 37.48825
## 22 4 1 2282.827 37.95757
## 23 4 2 2284.728 38.74572
## 24 4 3 2281.377 39.71132
## 25 4 4 2284.972 39.43857
```

Best ARIMA models based on RMSPE

```
head(result[order(result$RMSE),])
```

```
##    p q      AIC     RMSE
## 6  1 0 2277.979 35.82020
## 7  1 1 2279.671 36.12562
## 11 2 0 2279.444 36.54921
## 1  0 0 2277.018 36.55985
## 16 3 0 2281.381 36.63444
## 8  1 2 2280.976 36.98181
```

Best ARIMA models based on AIC

```
head(result[order(result$AIC),])
```

```
##    p q      AIC     RMSE
## 13 2 2 2274.345 38.94127
## 1  0 0 2277.018 36.55985
## 2  0 1 2277.884 37.08191
## 6  1 0 2277.979 35.82020
## 11 2 0 2279.444 36.54921
## 3  0 2 2279.511 37.57035
```

Best model: ARIMA(1, 1, 0), because it has the lowest RMSPE, and an AIC value that is very close to the second lowest.

Fit best ARIMA model on the full training set

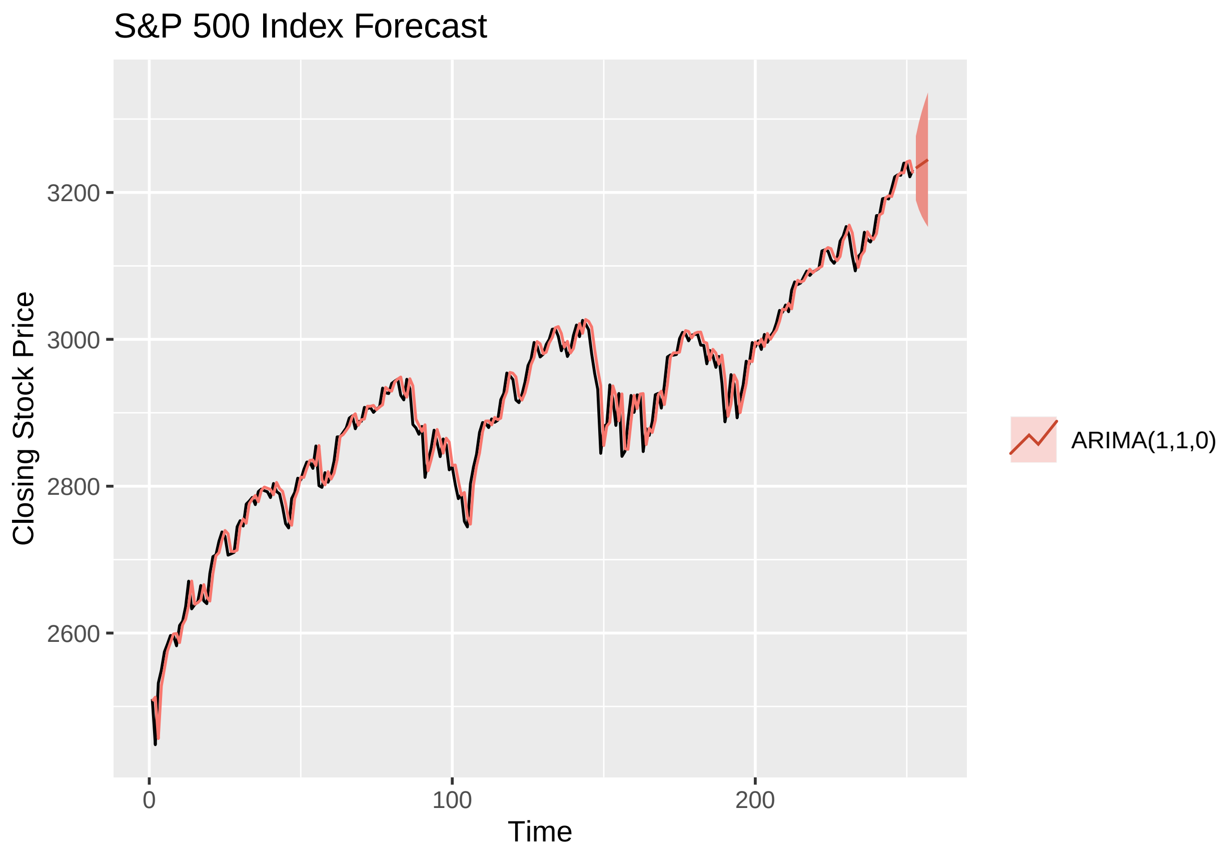
```
arima_model <- Arima(sp500, order = c(1, 1, 0), include.constant = T)
summary(arima_model)
```

```
## Series: sp500
## ARIMA(1,1,0) with drift
##
## Coefficients:
##           ar1   drift
##       -0.0849  2.8897
## s.e.   0.0639  1.2936
##
## sigma^2 estimated as 498:  log likelihood=-1134.57
## AIC=2275.13   AICc=2275.23   BIC=2285.71
##
## Training set error measures:
##                      ME    RMSE      MAE          MPE      MAPE      MASE
## Training set -0.0110232 22.1817 15.96988 -0.001719674 0.5560667 0.9770227
##                      ACF1
## Training set -0.004317636
```

```
rmse_arima <- min(result$RMSE)
```

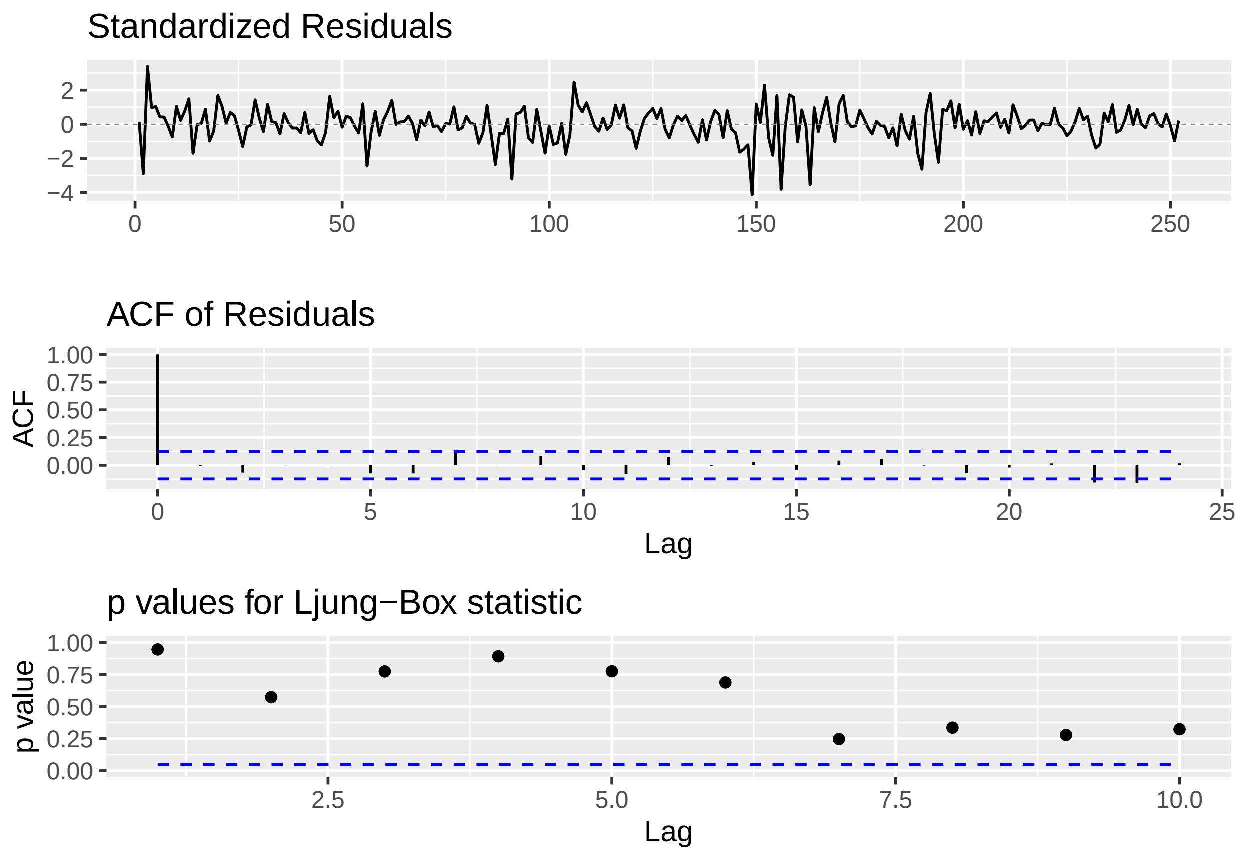
Forecast Plot

```
fcast_arima <- forecast(arima_model, h = 5, level = 95)
autoplot(sp500) +
  autolayer(fitted(fcast_arima), series = 'ARIMA(1,1,0)') +
  autolayer(fcast_arima, series = 'ARIMA(1,1,0)') +
  xlab('Time') +
  ylab('Closing Stock Price') +
  ggtitle('S&P 500 Index Forecast') +
  guides(colour = guide_legend(title = ' '))
```

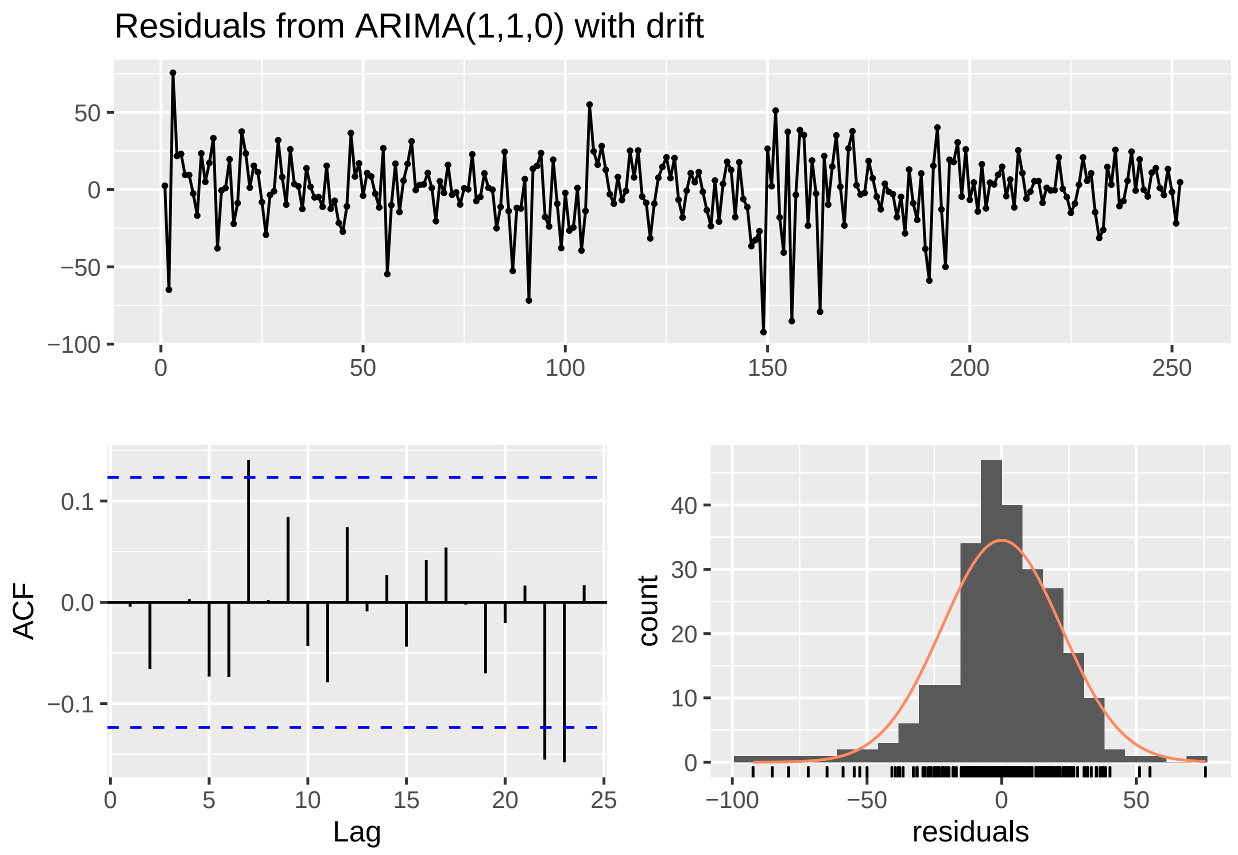


Residual Analysis

```
ggtsdiag(arima_model)
```



```r
checkresiduals(arima_model, lag = 12)
```



```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,0) with drift
## Q* = 14.571, df = 10, p-value = 0.1485
##
## Model df: 2.   Total lags used: 12
```

We can see that there is no pattern apparent in the residuals analysis plot. The acf values are not significant for lags other than 0. THe p-values for Ljung-Box test are also large suggesting nothing untoward about the fit of the model. Hence, we will use the ARIMA(1,1,0) to forcast the next 5 closing price of S&P 500, which are the closing values from 2020–01–02 to 2020–01–08.

Feedforward Neural Network

We will use a five-step time series cv to measure the goodness of prediction for all possible models (NNAR(p,1) where 0 <= p <= 4).

For every i = 1, . . . , 251: a) Train the model on every point before i b) Compute the test error on the held out point i b) Average the test errors

To evaluate the effectiveness of our methods, we will use the root mean square error (RMSE) metric. For RMSE, the lower the value, the better the prediction.

```r
cl <- makeSOCKcluster(detectCores() - 1)
registerDoSNOW(cl)

order <- 1:4
```

```
pb <- tkProgressBar(max = length(order))
opts <- list(progress = function(n) setTkProgressBar(pb, n))

score <- foreach(p = order, .options.snow = opts, .packages = c('forecast')) %dopar% {
  fnnetar <- function(x, h) {
    set.seed(2020)
    forecast(nnetar(y = x, p = p, size = 1), h = h)
  }
  e <- tsCV(sp500, fnnetar, h = 5)
  rmspe <- sqrt(mean(e^2, na.rm = T))
  return(c(p, rmspe))
}

close(pb)
stopCluster(cl)

result <- data.frame(Reduce(rbind, score), row.names = NULL)
colnames(result) <- c('p', 'RMSE')
print(result)
```

```
##   p     RMSE
## 1 1 39.75401
## 2 2 47.29434
## 3 3 45.22063
## 4 4 53.00826
```

Best model: NNAR(1,1), because it has the lowest RMSE.

Fit the best NNAR model

```
set.seed(2020)
nnar_model <- nnetar(y = sp500, p = 1, size = 1, repeats = 200)
print(nnar_model)
```

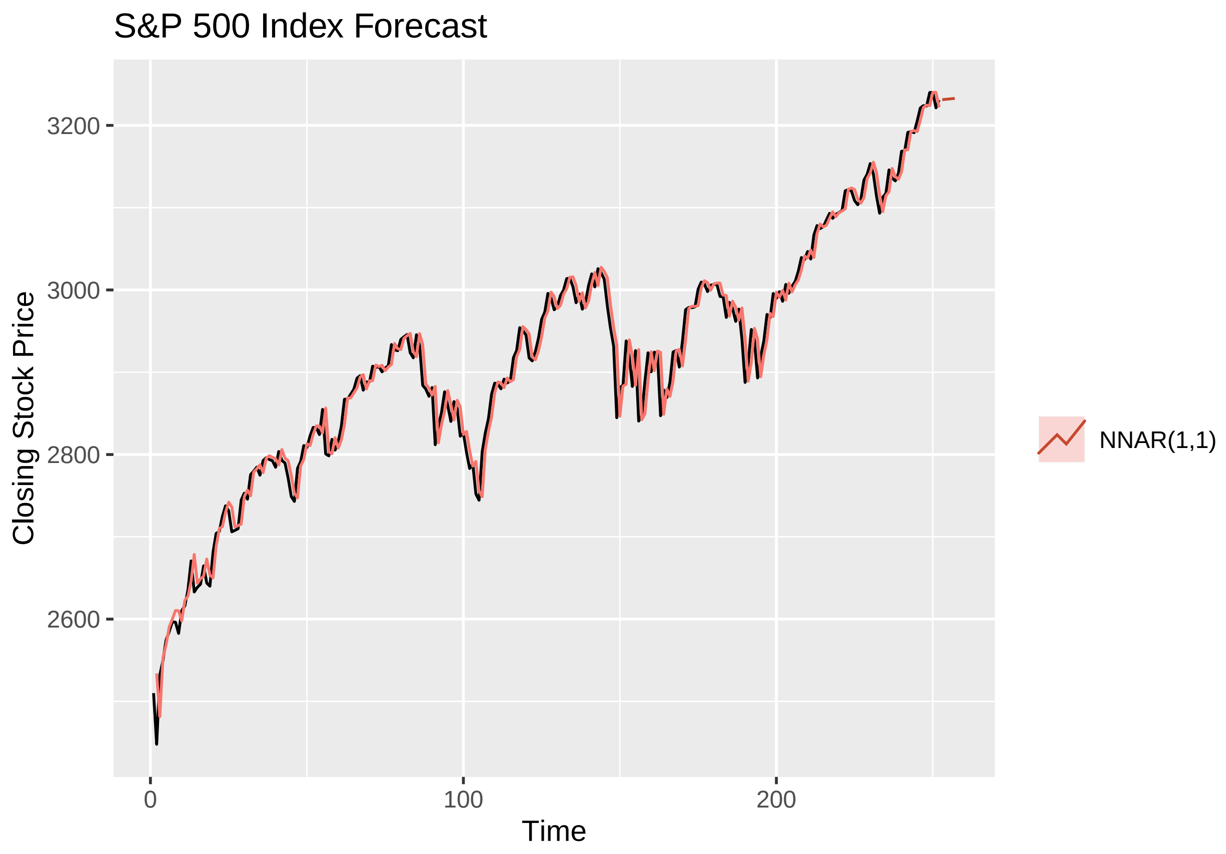
```
## Series: sp500
## Model:  NNAR(1,1)
## Call:   nnetar(y = sp500, p = 1, size = 1, repeats = 200)
##
## Average of 200 networks, each of which is
## a 1-1-1 network with 4 weights
## options were - linear output units
##
## sigma^2 estimated as 486.5
```

```
rmse_nnar <- min(result$RMSE)
```

Forecast Plot

```
fcast_nnar <- forecast(nnar_model, h = 5)
autoplot(sp500) +
  autolayer(fitted(fcast_nnar), series = 'NNAR(1,1)') +
  autolayer(fcast_nnar, series = 'NNAR(1,1)') +
  xlab('Time') +
  ylab('Closing Stock Price') +
  ggtitle('S&P 500 Index Forecast') +
  guides(colour = guide_legend(title = ' '))
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```



Forecasting

CV RMSE Comparison

```
rmse_cv <- setNames(data.frame(rmse_linear, rmse_ses, rmse_holts, rmse_damped_holts, rmse_arima, rmse_nn
                    c('Linear Regression', 'SES', "Holt's", "Damped Holt's", "ARIMA", 'NNAR'))
print(rmse_cv)
```

```
##      Linear Regression      SES   Holt's Damped Holt's   ARIMA     NNAR
## RMSE          74.69097 36.96683 42.11971      40.60588 35.8202 39.75401
```

We can see that ARIMA has the lowest RMSE obtained from cross validation. We will now compare the top 3 model using the test set. Since we could not calculate the CV RMSE for cubic spline, we will evaluate it with the test set too.

Retrieve the next 5 closing prices (First 5 in 2020)

```
getSymbols(Symbols = '^GSPC',
           src     = 'yahoo',
           auto.assign = T,
           from    = '2020-01-01',
           to      = '2020-03-31')
```

```
## [1] "^GSPC"
```

```
data <- GSPC[1:5, 'GSPC.Close']
head(data)
```

```
##            GSPC.Close
## 2020-01-02    3257.85
## 2020-01-03    3234.85
## 2020-01-06    3246.28
## 2020-01-07    3237.18
## 2020-01-08    3253.05
```

```
test <- as.vector(data)
```

Evaluate MSE for Top 3 Models

```
rmse_test <- data.frame(rbind(accuracy(fcast_ses$mean, test),
                              accuracy(fcast_arima$mean, test),
                              accuracy(fcast_nnar$mean, test),
                              accuracy(fcast_spline$mean, test)),
                        row.names = c('SES', 'ARIMA', 'NNAR', 'Cubic Spline'))
print(rmse_test[order(rmse_test$RMSE), ])
```

```
##                      ME     RMSE       MAE        MPE      MAPE
## ARIMA          6.917322 12.31737  9.253049  0.2123299 0.2844937
## NNAR          13.845228 16.48340 13.845228  0.4258055 0.4258055
## SES           15.641849 17.97517 15.641849  0.4811633 0.4811633
## Cubic Spline -16.904491 21.74303 20.279535 -0.5216402 0.6252375
```

We can see that ARIMA has the lowest test RMSE score. Cubic Spline seems to have the worst performance here, this is not surprising as it can still be further improved by using higher of knots or we can even use smoothing splines instead so we would not need to optimize the number of knots.

```
cbind(data.frame(fcast_arima), Actual = test)
```

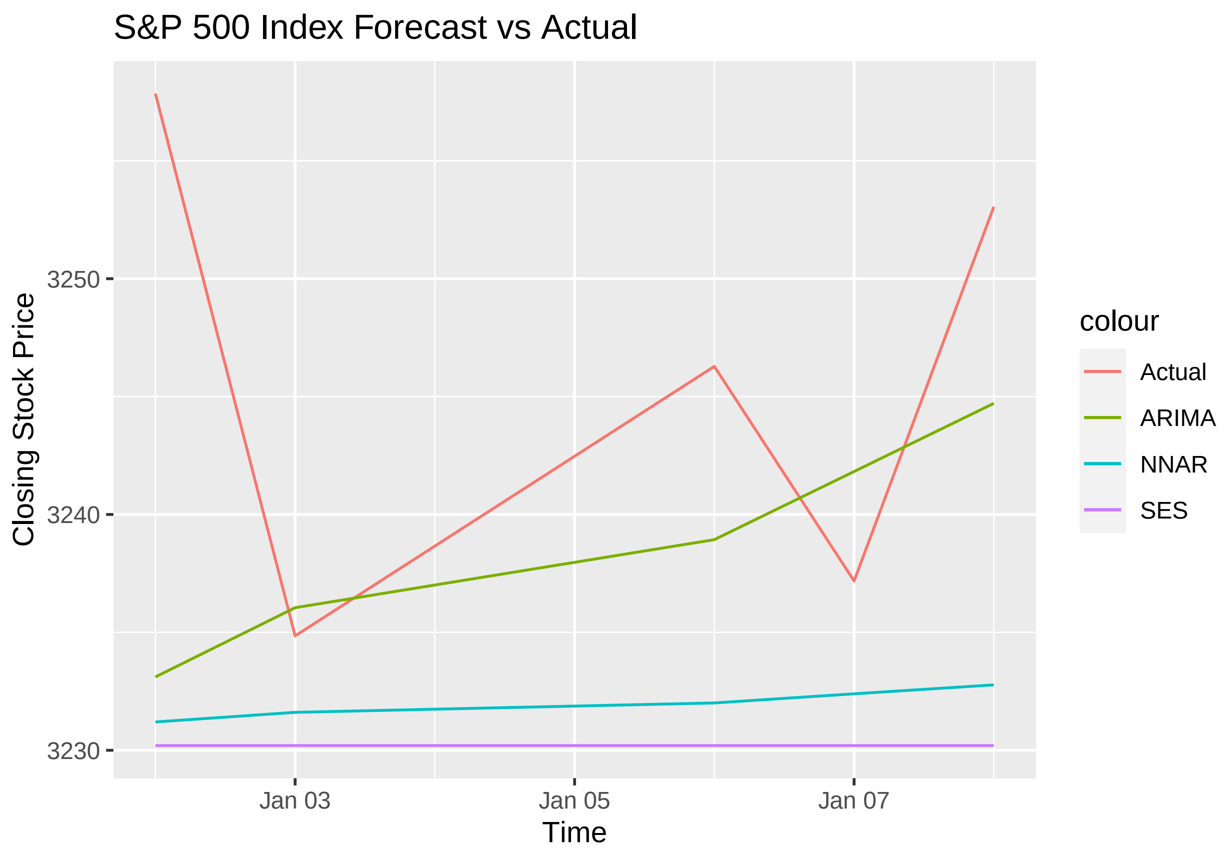
```
##     Point.Forecast    Lo.95    Hi.95  Actual
## 253       3233.110 3189.373 3276.846 3257.85
## 254       3236.047 3176.761 3295.333 3234.85
## 255       3238.932 3167.224 3310.641 3246.28
## 256       3241.823 3159.560 3324.085 3237.18
## 257       3244.712 3153.102 3336.322 3253.05
```

Our best model ARIMA has a Root Mean Squared Error of 12.31737, and we also obseved all actual values are lying between the 95% prediction confidence interval. Thus, the ARIMA(1,1,0) seems to perform well in the prediction.

Forecast vs Actual

```
ggplot(data.frame(data), aes(x = index(data))) +
  geom_line(aes(y = GSPC.Close, color = 'Actual')) +
  geom_line(aes(y = fcast_arima$mean, color = 'ARIMA')) +
  geom_line(aes(y = fcast_nnar$mean, color = 'NNAR')) +
  geom_line(aes(y = fcast_ses$mean, color = 'SES')) +
  xlab('Time') +
```

```r
  ylab('Closing Stock Price') +
  ggtitle('S&P 500 Index Forecast vs Actual')
```



Conclusions:

TODO: Include other models (Linear, Cubic Spline, SES, Holt's, Damped Holt's, Neural Net)

We set out to determine if there was a meaningful model that could be found, which would be able to predict future stock price of the S&P500 index. In this project, we collected 252 observations from Yahoo Finance API. The data represents the daily stock price from 2019–01–01 to 2020–01–01 (no including 104 days for the weekends when the exchange is closed). We were also able to obtain the actual stock prices for the period of 2020–01–02 to 2020–01–08 (5 values) which we will attempt to forecast using the data from 2019. The actual values compared to our forecasted values will be used as a measure of the quality of our model.

Preliminary data exploration showed that the daily stock price of the S&P500 index was not a stationary series, the data plot showed an obvious upward trend, without seasonality or periodicity. Moreover, the acf showed significant correlation at all lag points. Ljung-Box p-value was less than 0.05, therefore we concluded that the series was not a realization of a white noise (stationary) process. Before further downstream analysis, we had to transform the original, non-stationary, data series into a stationary series.

TODO: mean is not zero, box test

The series was transformed using diff = 1, generate a new series. When plotted, the new series data values centered around mean zero, with no obvious trend. The acf and pacf plots show significant values at lag 7, 22, and 23. After doing the Ljung-Box test, we concluded that the new series is a realization from a stationary process, and that the significant values in the acf and pacf were likely to be outliers.

TODO: AIC

Next we wanted to find the optimal model for forecasting. To do this, we enumerated over possible values of p and q ($d = 1$), constrainted by $p >= 0$ and $p <= 4$, and $q >= 0$, and $q <= 4$. We used a leave-one-out cross-validation approach quantify the accuracy of each model. The chosen model used ARIMA(1, 1, 0), because it has the lowest RMSPE (22.62201), and the second lowest AIC (1159.596). Fitting the AR(1) model to the full dataset returned an alpha coefficient of -0.0849. Residual analysis on the fitted model shows that the residuals follow a realization of a white noise process, as expected from a good fitting model.

We forecasted the first five stock prices from the year 2020, from the period of 2020–01–02 to 2020–01–08 (less 1 weekend). The actual values are shown to fall within the 95% confidence interval of the forecasted values. As the time increases into the future, the confidence interval widens, as expected. Overall, we conclude that an ARIMA(1,1,0) model is a good approximatation for the daily stock price of the S&P500 index for the year of 2019. The ARIMA(1,1,0) model is also accuracte for forecasting purposes, up until the first week of 2020.

During post-analysis, we looked at various years through out the S&P500 data and realized that we had been extremely fortunate selecting 2019 as our training dataset. The trends in other years were not as clear and there is no guarantee that using other years as training would yield an ARIMA(1,1,0) as the best fitted model. Furthermore, the stock market experiences various periods of extreme growth, stagnation, and crashes, however there was no such novel event during the year of 2019. Taking into account these extenuating circumstances, it is extremely unlikely that an ARIMA(1,1,0) would be the best fitted model for the S&P500 in general.

Moving forward, we would like to try to use the entire S&P500 dataset from inception in 1926 until 2019. This may yield a more accurate model of the S&P500 stock index. However, even that model is unlikely to be able to predict market crashes like what we are currently experiencing due to the COVID-19 global pandemic. More sophisticated models which can account for anomaly detection would be necessary in order to generate a model which can accurately model the S&P500 index.