R Notebook

Load Packages

```
library(ggplot2)
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 3.4.3
```

```
library(plotly)
```

```
## Warning: package 'plotly' was built under R version 3.4.3

##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':

##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(ggfortify)
```

```
## Warning: package 'ggfortify' was built under R version 3.4.4
```

```
library(tseries)
```

```
## Warning: package 'tseries' was built under R version 3.4.3
```

```
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 3.4.4
```

```
library(docstring)
```

```
## Warning: package 'docstring' was built under R version 3.4.4

##
## Attaching package: 'docstring'

## The following object is masked from 'package:utils':
##
##     ?
```

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 3.4.3
```

```
library(here)
```

```
## Warning: package 'here' was built under R version 3.4.4

## here() starts at C:/Users/samid/Desktop/course/FinTS
```

Get Data

Now we collect our data. We want to use reliable sources of complete and accurate data. We collected 10 years (1995-2017) of S&P 500 Stock Index data at a monthly frequency (a total of 267 observations) from Yahoo Finance.

```
data_master <- read.csv("C:/Users/samid/Desktop/course/FinTS/sp.csv")
dim(data_master)
```

```
## [1] 267   7
```

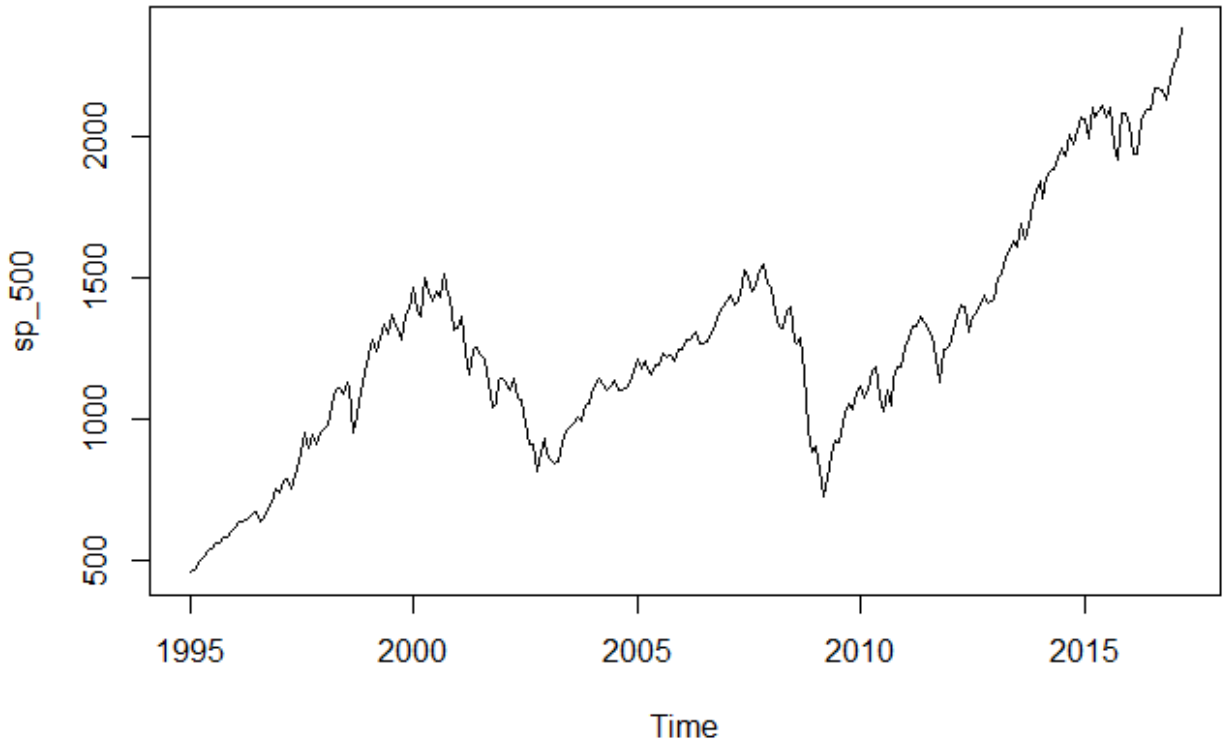
```
#Creating time-series data object
sp_500 <-ts(data_master$Open, start=c(1995, 1), freq=12)
class(sp_500)
```

```
## [1] "ts"
```

Exploratory Analysis

Now we want to get a feel for our data to get an intuition about the models that may be appropriate for our forecast. For this, we plot our data and diagnose for trend, seasonality, heteroskedasticity, and stationarity.

```
plot.ts(sp_500)
```



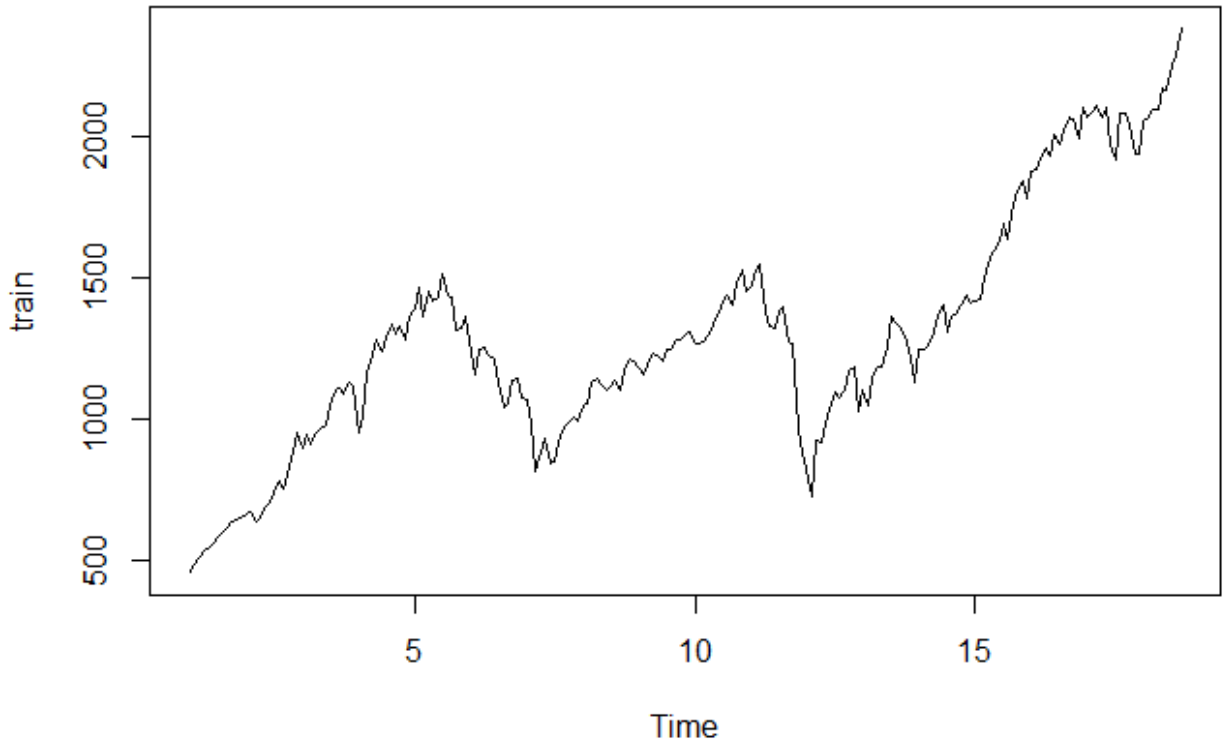
Before we begin any analysis, we will be splitting the data to in 80:20 ratio to use as our test set.

```
dt = sort(sample(nrow(data_master), nrow(data_master)*.8))
train<-data_master[dt,]
test<-data_master[-dt,]
train<-ts(train$Open, freq=12)
test<- ts(test$Open, freq=12)
```

Plotting our Time Series

Plotting the data is arguably the most critical step in the exploratory analysis phase

```
plot.ts(train)
```



We can quickly see that our time-series has instances of both positive and negative trend. Overall, it is very volatile, which tells us that we will have transform the data in order for the Box-Jenkins Methodology to predict with better accuracy.

Testing for Stationarity

We will utilize a few statistical tests to test for stationarity. We must be weary of our model having a unit root, this will lead to non-stationary processes.

```
Box.test(sp_500, lag = 20, type = 'Ljung-Box')
```

```
##
##  Box-Ljung test
##
## data:  sp_500
## X-squared = 3047.5, df = 20, p-value < 2.2e-16
```

Now we will utilize the Augmented Dickey-Fuller Test for stationarity. The null hypothesis states that large p-values indicate non-stationarity and smaller p values indicate stationarity (We will be using 0.05 as our alpha value).

```
adf.test(sp_500)
```

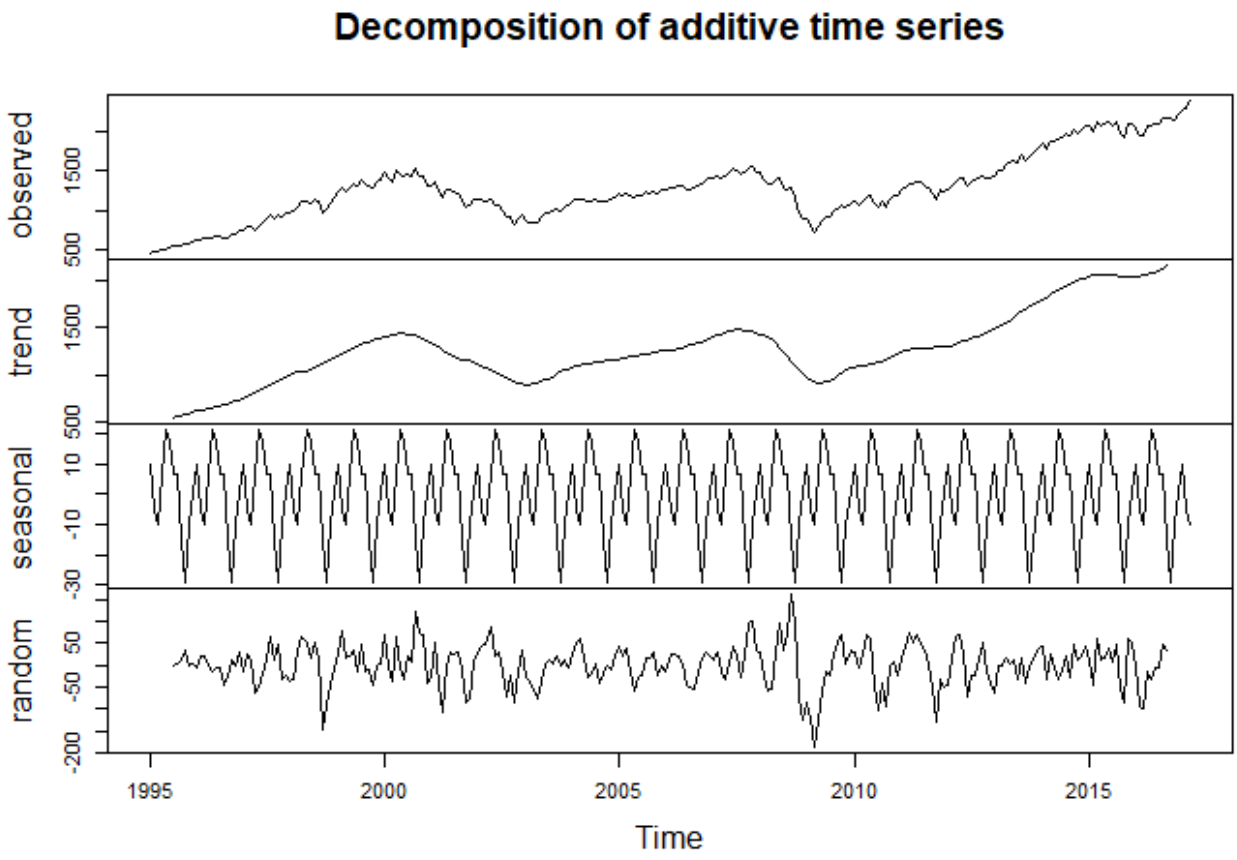
```
##
##  Augmented Dickey-Fuller Test
##
## data:  sp_500
## Dickey-Fuller = -1.3517, Lag order = 6, p-value = 0.849
## alternative hypothesis: stationary
```

We can see our p-value for the ADF test is relatively high, so we'll do some further visual inspection. But we know we will most likely have to difference our time series for stationarity.

Decomposing our time-series

Beyond understanding the trend of our time-series, we want to further understand the anatomy of our data. For this reason we break-down our time-series into its seasonal component, trend, and residuals.

```
decom<-decompose(sp_500)
plot(decom)
```

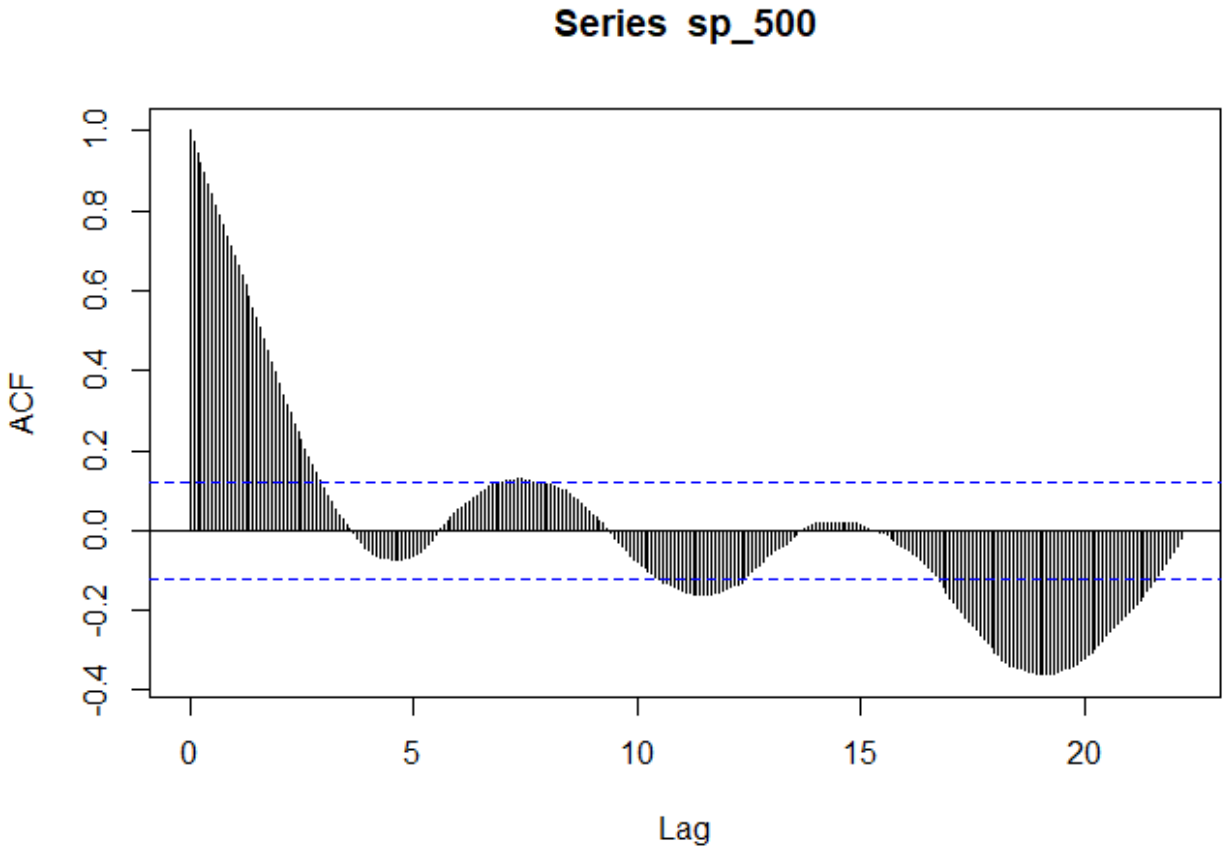


The trend line already shows us what we know and we can see that there might be some seasonality in our time series object.

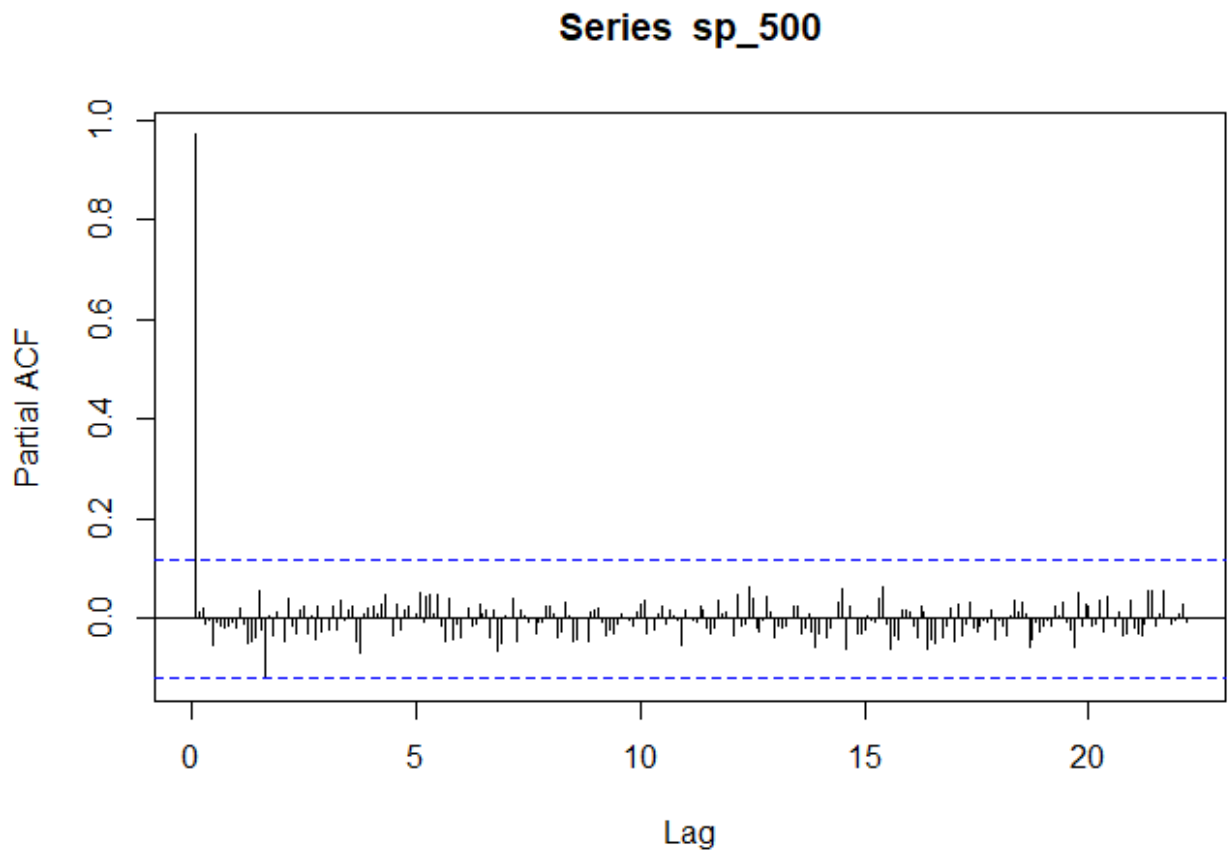
Model Estimation

Diagnosing the ACF and PACF Plots of our Time-Series Object

```
acf(sp_500,'S&P 500')
```



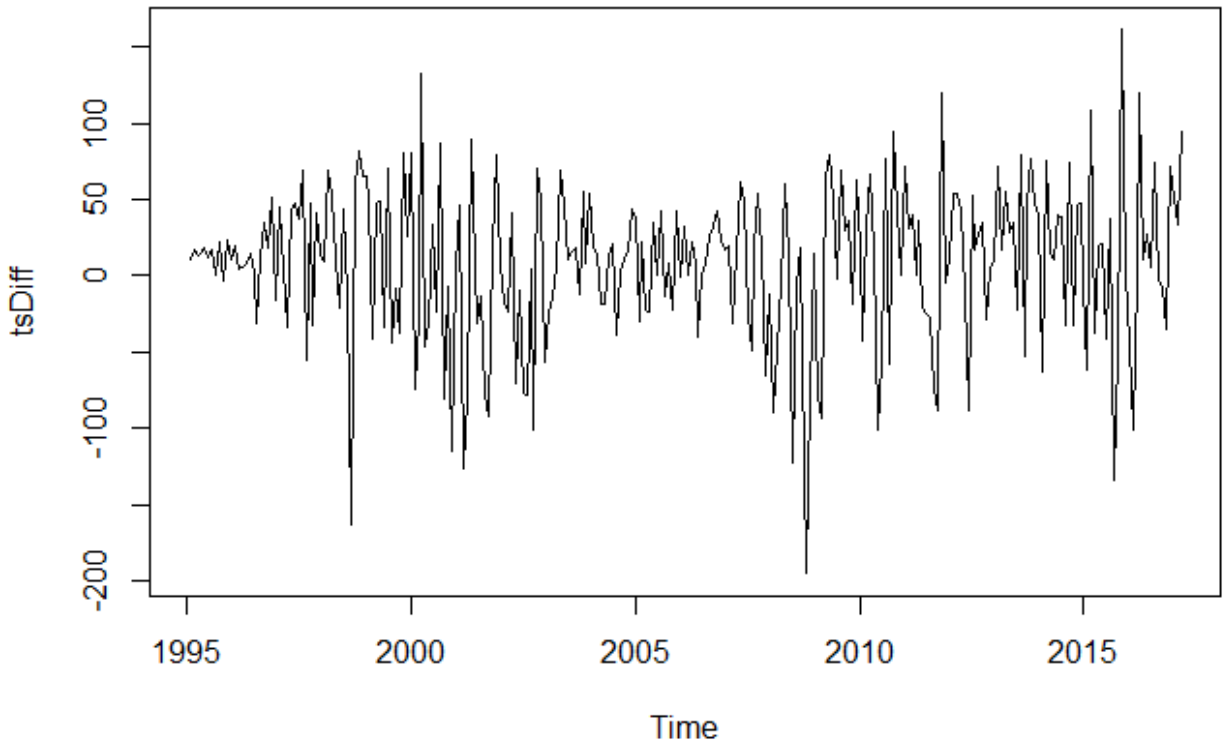

```
pacf(sp_500,'S&P 500')
```



When there is large autocorrelation within our lagged values, we see geometric decay in our plots, which is a huge indicator that we will have to take the difference of our time series object.

Transforming our data to adjust for non-stationary

```
tsDiff <- diff(sp_500)
plot.ts(tsDiff)
```



This plot suggests that our working data is stationary. We want to confirm this running an ACF and PACF diagnostics over this data to find our if we can proceed to estimating a model.

Testing for Stationarity

```
Box.test(tsDiff, lag = 20, type = 'Ljung-Box')
```

```
##
##  Box-Ljung test
##
## data:  tsDiff
## X-squared = 18.953, df = 20, p-value = 0.5249
```

```
adf.test(tsDiff)
```

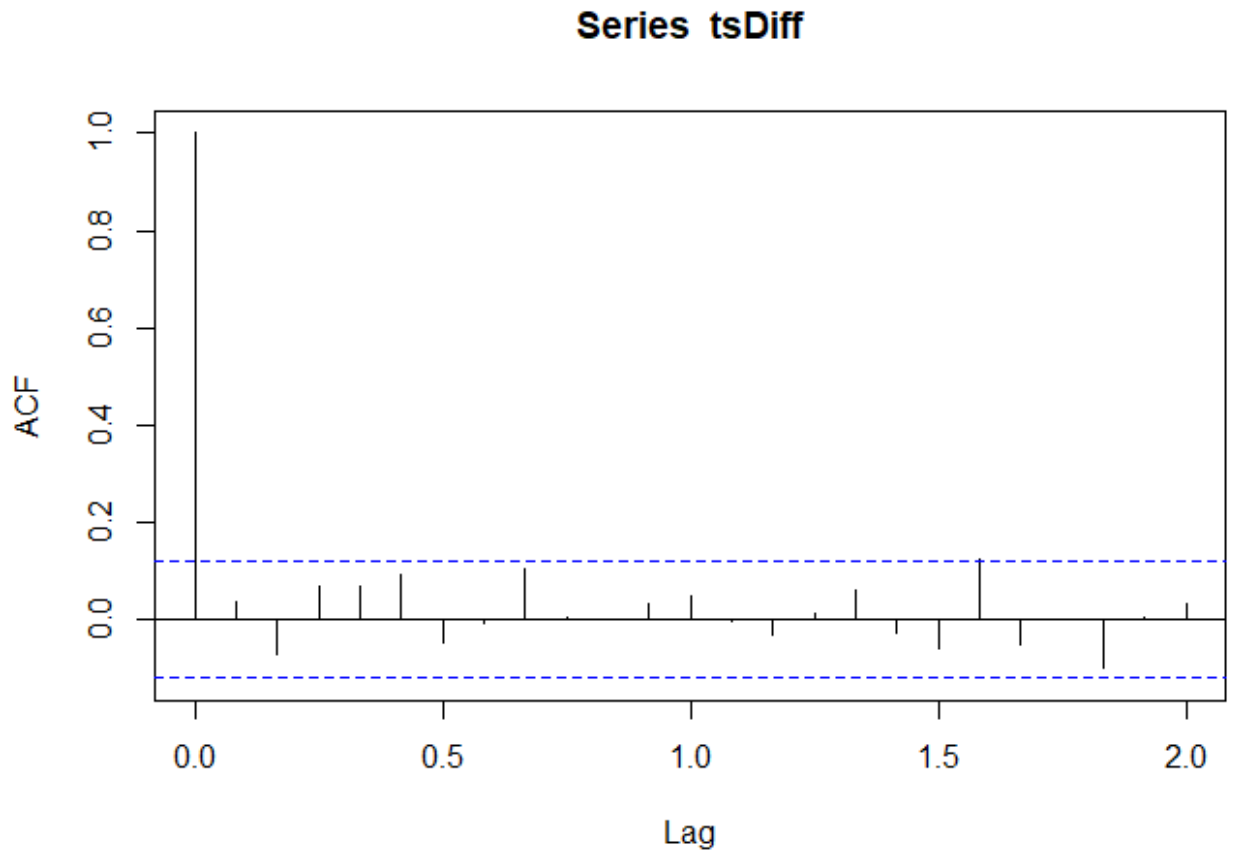
```
## Warning in adf.test(tsDiff): p-value smaller than printed p-value

##
```

```
##  Augmented Dickey-Fuller Test
##
## data:  tsDiff
## Dickey-Fuller = -5.3946, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

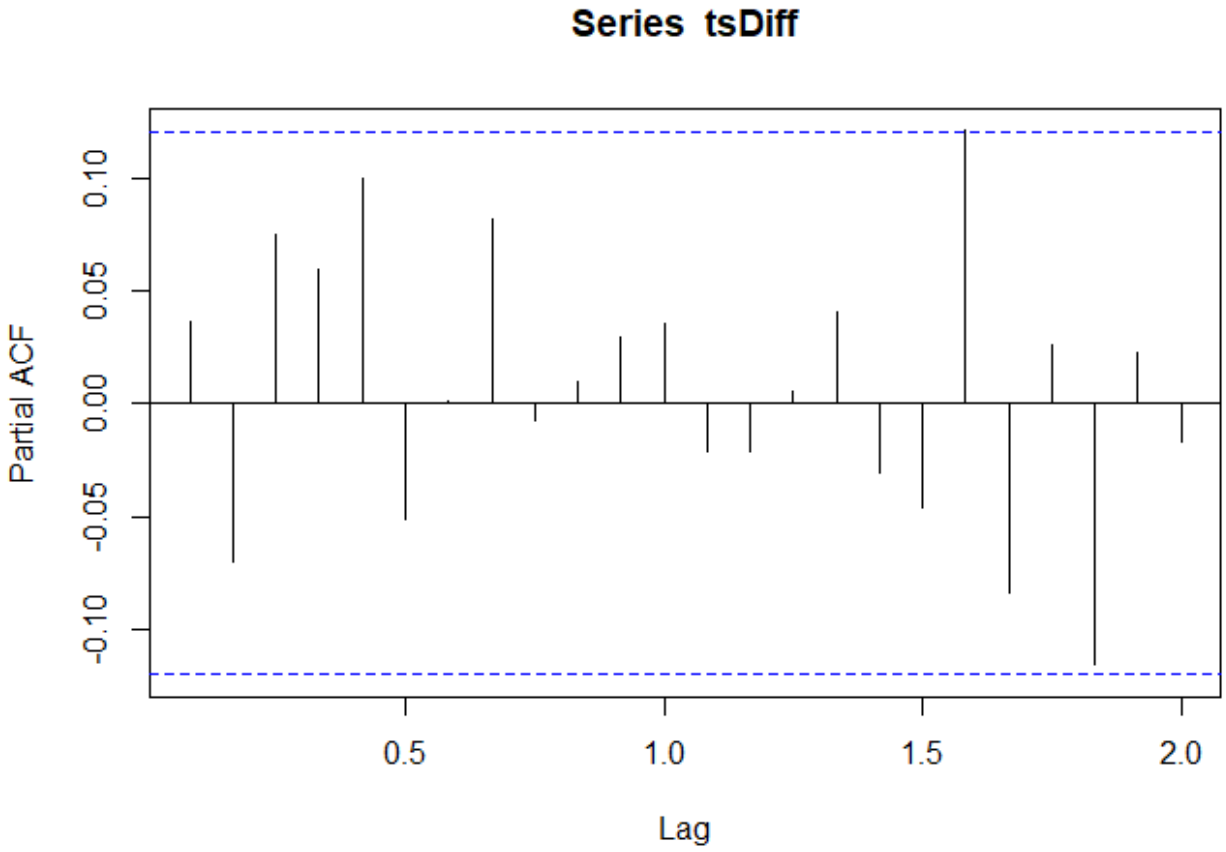
We can see that the result yields a small p-value which makes us reject the null suggestion stationarity.

Diagnosing the acf and pacf of our transformed time-series object

```
acf(tsDiff)
```




```
pacf(tsDiff)
```



Build Model

The auto.arima() method, found within the forecast package, yields the best model for a time-series based on Akaike-Information-Criterion (AIC). The AIC is a measurement of quality used across various models to find the best fit.

```
fit<- auto.arima(sp_500)
fit
```

```
## Series: sp_500
## ARIMA(0,1,0) with drift
##
## Coefficients:
##        drift
##       7.2215
## s.e.  3.1864
##
## sigma^2 estimated as 2711:  log likelihood=-1428.31
## AIC=2860.62   AICc=2860.67   BIC=2867.79
```

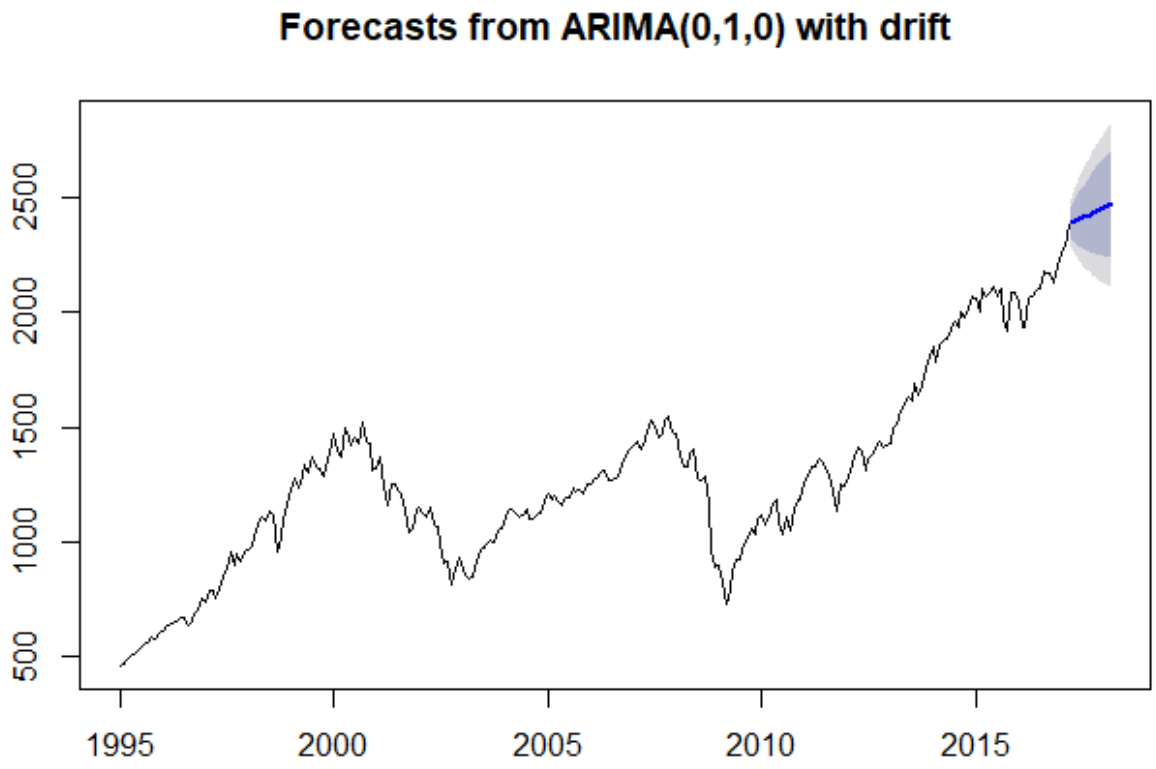
```
fit1 <- Arima(sp_500, order = c(0,1,0),
    include.drift = TRUE)
summary(sp_500)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   459.2  1025.3  1239.9  1274.8  1433.7  2380.1
```

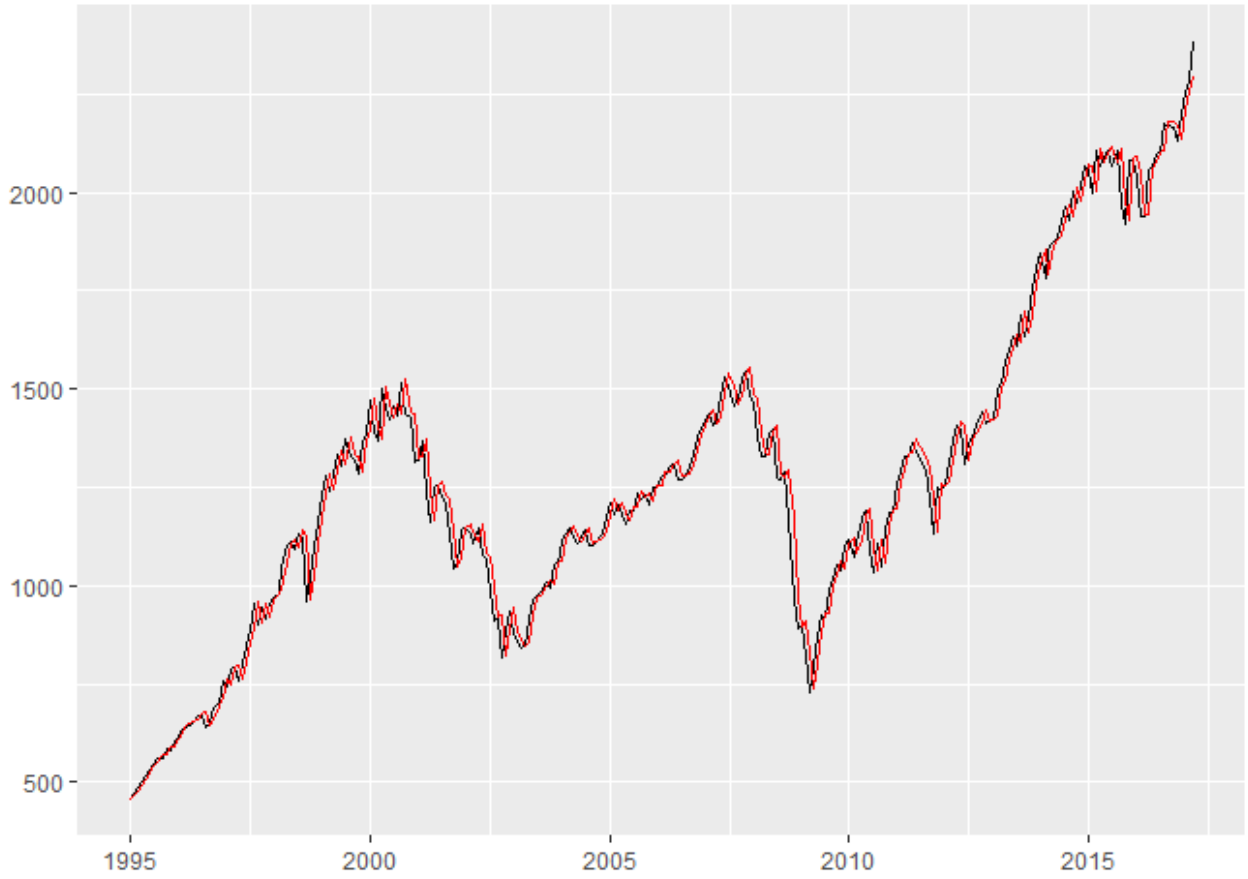
Forecasting

We proceed to forecasting now that we believe we found the appropriate model!

```
for_sp500_all <- forecast(fit1, h = 12)
plot(for_sp500_all)
```




```
autoplot(fit1,
    holdout = sp_500,
    forc_name = 'ARIMA',
    ts_object_name = 'S&P 500')
```



Other Forecasting Methods

Box-Cox Forecast

Box-Cox transformations are generally used to transform non-normally distributed data to become approximately normal! Although we do not think this an appropriate transformation for our data set, it is still included in our analysis because it's a useful transformation to do especially since most real time data is not approximately normally distributed.

```
lambda <- BoxCox.lambda(sp_500)
fit_sp500_BC <- ar(BoxCox(sp_500,lambda))
fit_BC <- forecast(fit_sp500_BC,h=12,lambda=lambda)
autoplot(fit_BC,
    holdout = sp_500,
    forc_name = 'Box-Cox Transformation',
    ts_object_name = 'S&P 500')
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.3
```

Exponential Smoothing Forecast

The following forecasting method is far more complex than the previous methods. This forecasting method relies on weighted averages of past observations where the most recent observations hold higher weight!

```
fit_ets <- forecast(ets(sp_500), h = 36)
autoplot(fit_ets,
    holdout=sp_500,
    forc_name = 'Exponential Smoothing',
    ts_object_name = 'S&P 500')
```

###Naive Forecast The naive forecasting method returns an ARIMA(0, 1, 0) with random walk model that is applied to our time series object.

```
fit_naive <- naive(sp_500, h = 12)
autoplot(fit_naive,
    holdout = sp_500,
    forc_name = 'Naive Forecast',
    ts_object_name = 'S&P 500')
```

###Conclusions

The forecasting method we use to find the best model is recieving the lowest MAE and MAPE. We run the accuracy function on all the forecast methods and we check which performed best!

```
#round(accuracy(fit1, sp_500), 3)
#round(accuracy(fit_BC, sp_500), 3)
#round(accuracy(fit_ets, sp_500), 3)
#round(accuracy(fit_naive, sp_500), 3)
```