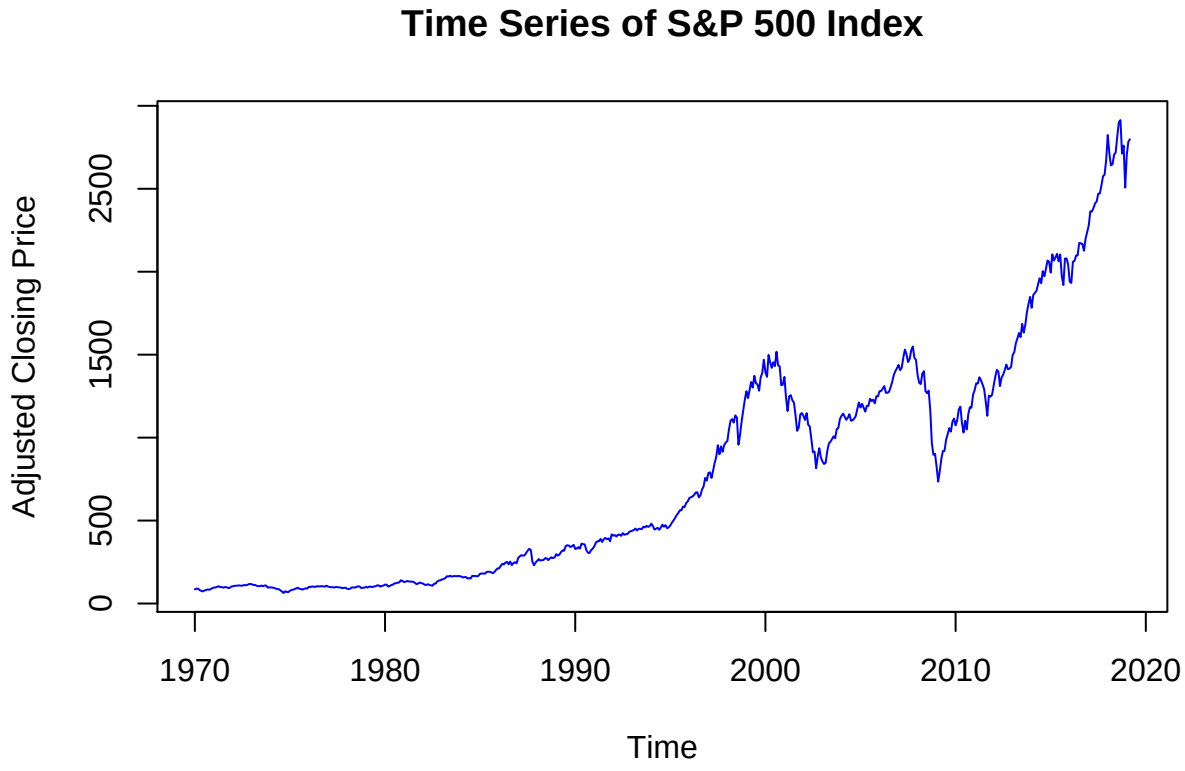
Time Series Analysis: S&P 500-monthly Price

April 4, 2019

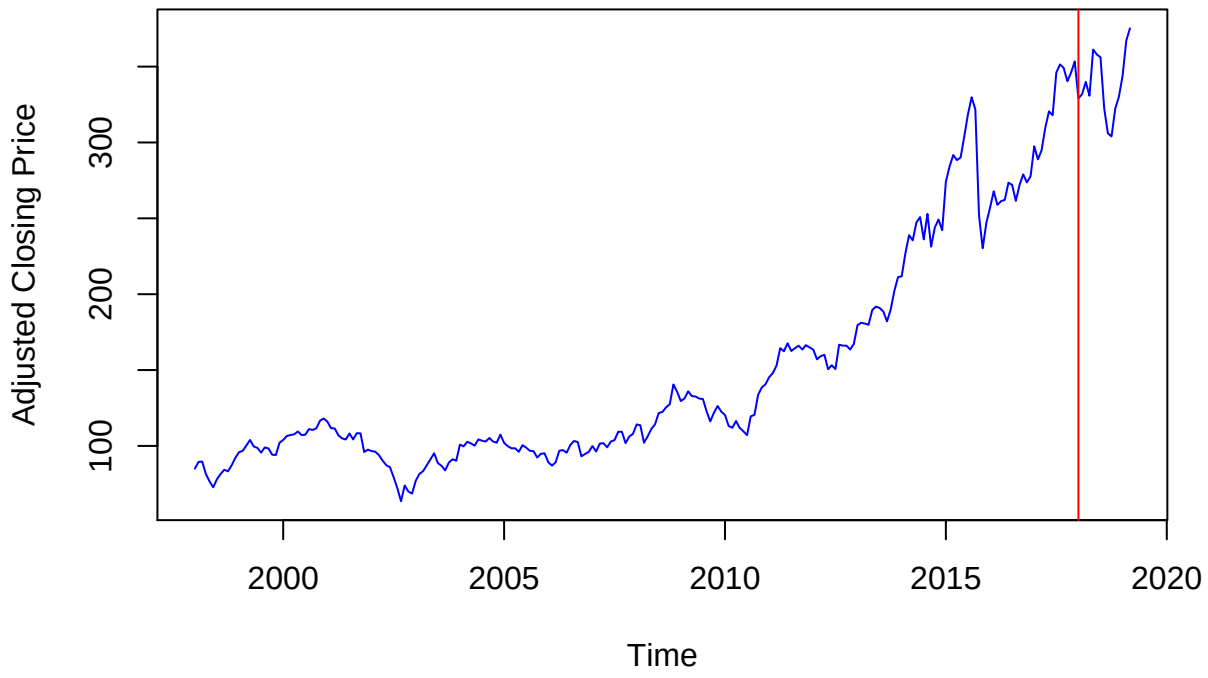
I - Introduction and Preliminaries

In the following, we consider time-series data for the S&P500. The data was collected from Yahoo finance, for the years 1970-2019, on a monthly basis. We use the adjusted closing value. We examine this data in attempts to fit a time-series mode, and forecast future values. We will do this "by hand", using manual fitting techniques, and also by the built-in R functions. Then, we will compare the results, and gain insight to this time-series process.

First, we consider the raw data. We see that in the approximate years of 1970-1980, the data does not appear to be meaningful. As such, we remove these points from our training set. Additionally, we reserve 2018 onwards as our testing set.



This our cleaned-up data set.



Now, we perform some preliminary tests:

```
summary(sp_500)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   63.54   98.94  114.24  159.33  206.72  375.22
```

We can easily see that our time series has instances of both positive and negative trend. Overall, it is very volatile.

We perform the Box-Ljung test to see determine if our data is IID, or not.

```
Box.test(SP500_training, lag = 50, type = 'Ljung-Box')
```

```
##
##  Box-Ljung test
##
## data:  SP500_training
## X-squared = 4954, df = 50, p-value < 2.2e-16
```

This indicates for us to reject the null hypothesis, that all of the autocorrelation functions up to 50 are zero. We reach the conclusion numercally that our data is not IID.

Additionally, the Q-Q plot (figure 1) shows our time series is not normally distributed.

```
ggtsdisplay(SP500_training,lag.max=100,plot.type = "scatter")
```

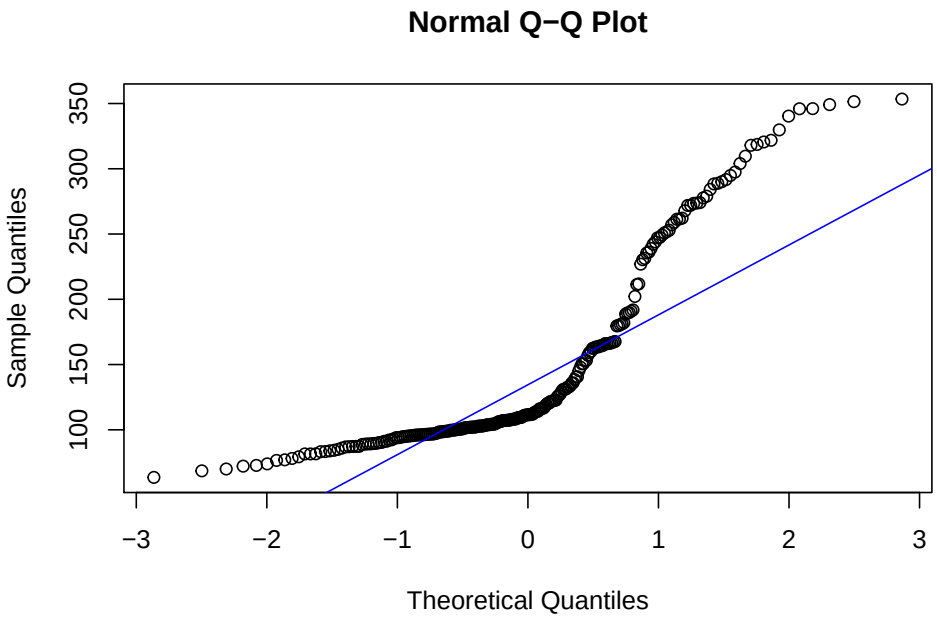
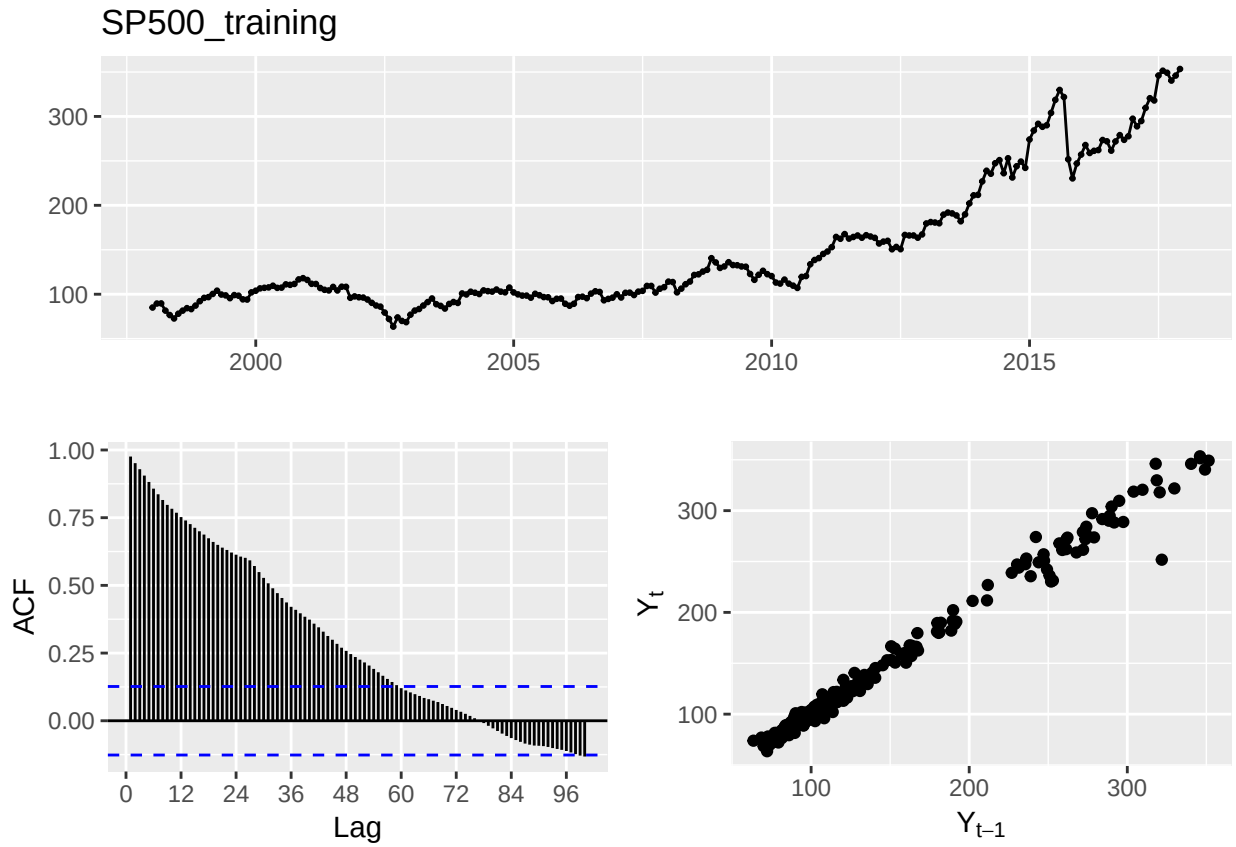


Figure 1: Q-Q plot for Training Set



Another test is Unit root test. This test is used to find out that first difference or regression which should be

used on the trending data to make it stationary. In Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test, small p-values suggest differencing is required.

```
kpss.test(SP500_training)
```

```
## Warning in kpss.test(SP500_training): p-value smaller than printed p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  SP500_training
## KPSS Level = 3.7915, Truncation lag parameter = 4, p-value = 0.01
```

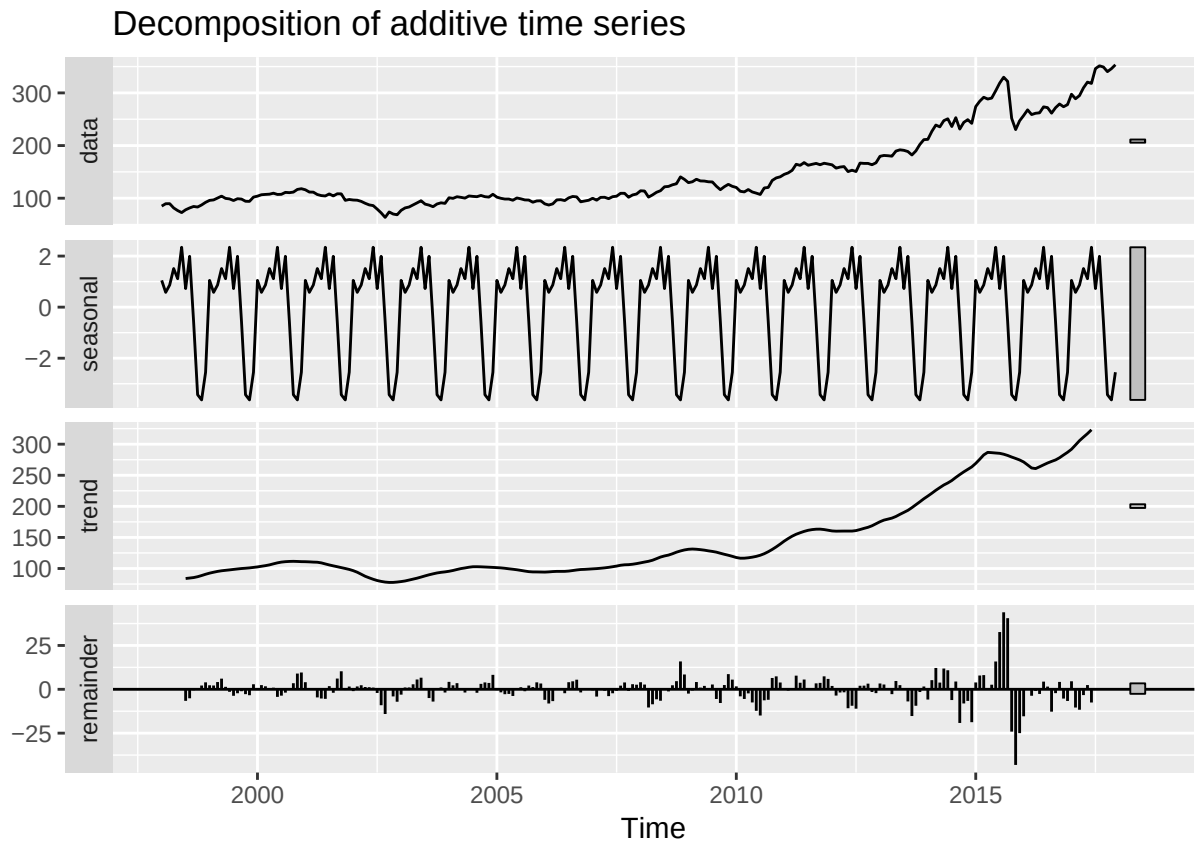
We reach the conclusion numerically that the series itself is not a white noise process, and so its innovations are not completely random and we need differencing to have an stationary time Series.

II - Fitting the Model using Manual Techniques

Now, we want to further analyze our data in order to decide which models may be appropriate for our forecast. To do this, we will plot our data and diagnose for trend, seasonality, and stationarity visually. Note that visualizing our time-series data enables us to make inferences about important components, such as trend, seasonality, and stationarity.

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

```
Series<- decompose(SP500_training,filter=NULL)
autoplot(Series)
```

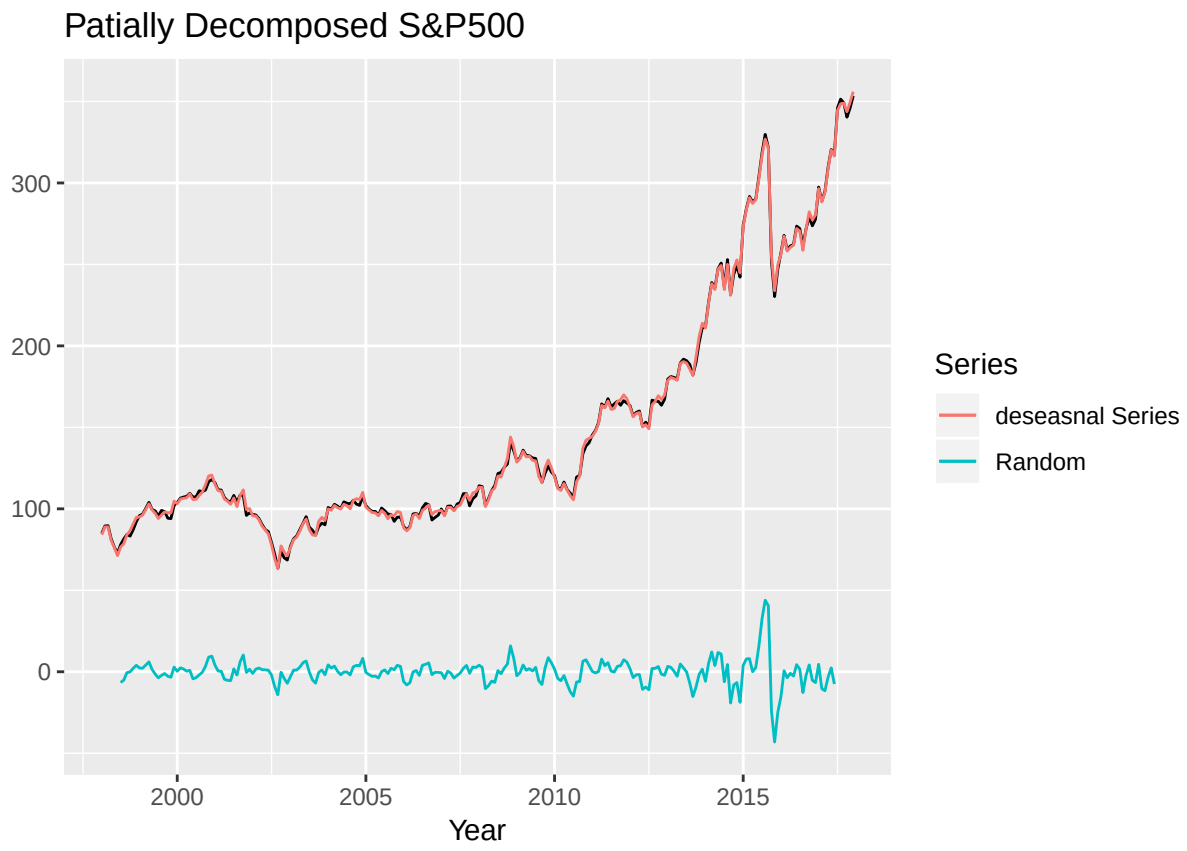


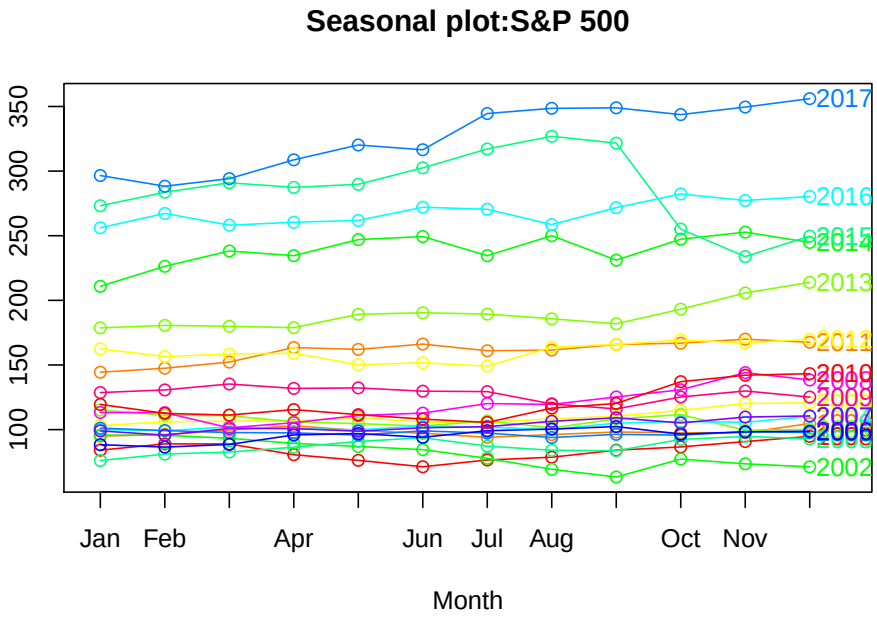
The trend line shows us what we already know from the vanilla plot of our time series. From the seasonal plot, we can see there may be some seasonality in our time-series as well.

Note that de-seasonalizing lends insight into the seasonal pattern in the time series and helps to model the data without the seasonal effects. We do this in two steps. First, we decompose the time series using *stl*(), and then use *seasadj*() from 'forecast' package.

Now we consider the impact of de-seasonalizing. From the 'Seasonal plot', there is no obvious seasonality that we detect by eye. This is further supported by the 'Partially Decomposed plot', in which the de-seasonalized plot and unaltered plot are very close. Overall, this show us that the S&P500 has no significant seasonality.

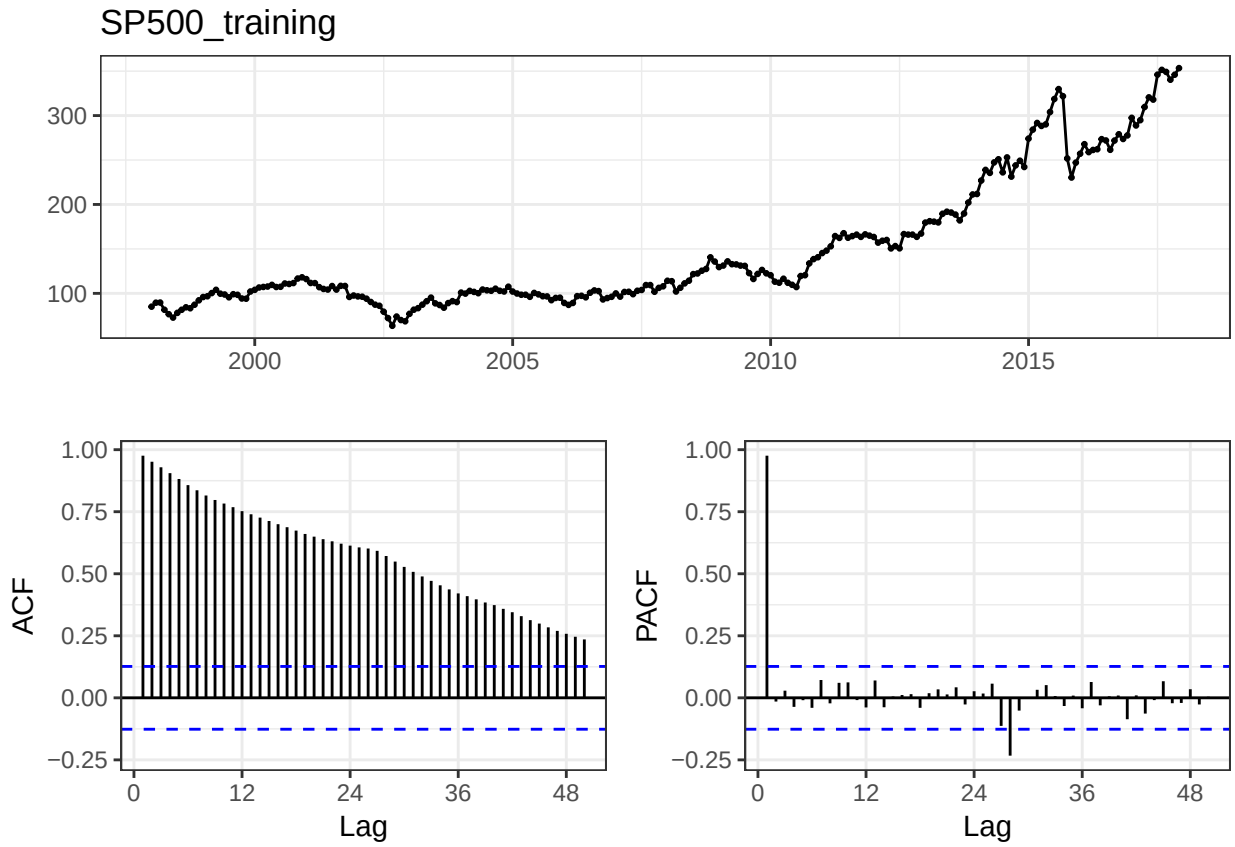
```
decomp<- stl(SP500_training, s.window="periodic") # decompose the Data
deseasonal_cnt <- seasadj(decomp)# de-seasonalize
```





Now that we have concluded that our time series is not seasonal, we move on to determining if it is stationary or not. We will utilize a few statistical methods to test for stationarity. note that we must be wary of our model having a unit root; this will lead to non-stationary processes.

In general, the Autocorrelation and Partial Autocorrelation functionsare used to determine if the time series is stationary or not. A stationary time series will have the autocorrelation fall to zero fairly quickly, but for a non-stationary series it drops gradually.



On the ACF (left figure), we have that time series in non-stationary, and we cannot see any periodic behaviour. The ACF also indicates that the time series may not be an MA model, and it is more likely to be either an ARMA or an AR model. The cut of after lag 1 in PACF also suggests we may have AR(1) model.

From the above results, it is reasonable to assume we have an AR model. So, we can us the (Adjusted Dickey-Fuller) ADF method to test for stationary time series. A smaller p value indicates stationarity (We will be using 0.05 as our α value.).

```
adf.test(SP500_training,alternative = "stationary")
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  SP500_training
## Dickey-Fuller = -0.61936, Lag order = 6, p-value = 0.9759
## alternative hypothesis: stationary
```

It is clear that our p value for the ADF test is relatively high, and that our time series contains a unit root.

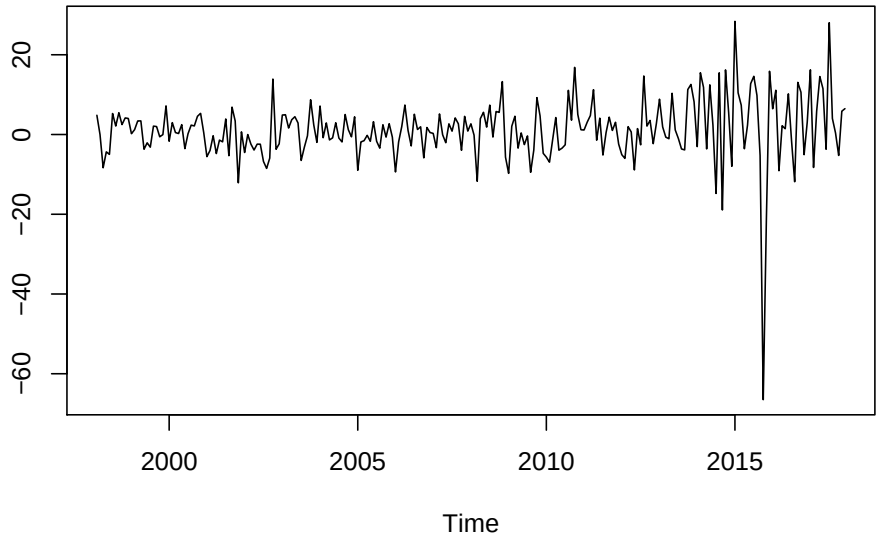
Based on our visual inspection of the time-series object and the statistical tests used for exploratory analysis(e.g KPSS test), it is appropriate to differentiate our time-series object to account for the non-stationarity.

One way to make a time series stationary is to find the difference across its consecutive values. This helps stabilize the mean, thereby making the time-series object stationary. For this we use the $diff()$ method.

We can start with the order of d = 1 and re-evaluate whether further differencing is required. For most time

series patterns, 1 or 2 differentiations is necessary to make it a stationary series.

```
tsDiff <- diff(deseasonal_cnt,1)
plot.ts(tsDiff,ylab = "")
```



This plot suggests that our working data is stationary (Except sometimes around 2015-2016). You will want to confirm this by running the same tests and looking at the ACF and PACF diagnostics over the differenced data to find out if you can proceed to estimating a model. Let's apply the same tests to our differenced time-series object:

Now let's use the ADF test:

```
adf.test(tsDiff, alternative = "stationary")
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  tsDiff
## Dickey-Fuller = -5.7548, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

The p values suggest we fail to reject the null hypothesis. Our once differentiated time series is likely stationary.

In the following code segment (using "nsdiffs, ndiffs"), we allow R to determine the appropriate amount of differentiation. We get the same results as before, with d = 1.

```
#Seasonal Differencing
sdif<-nsdiffs(SP500_training)  # number for seasonal differencing needed
# seasonal differencing
if (sdif==0){print("There is no seasonality")
  SP_seasdiff=SP500_training
}
```

```
## [1] "There is no seasonality"
```

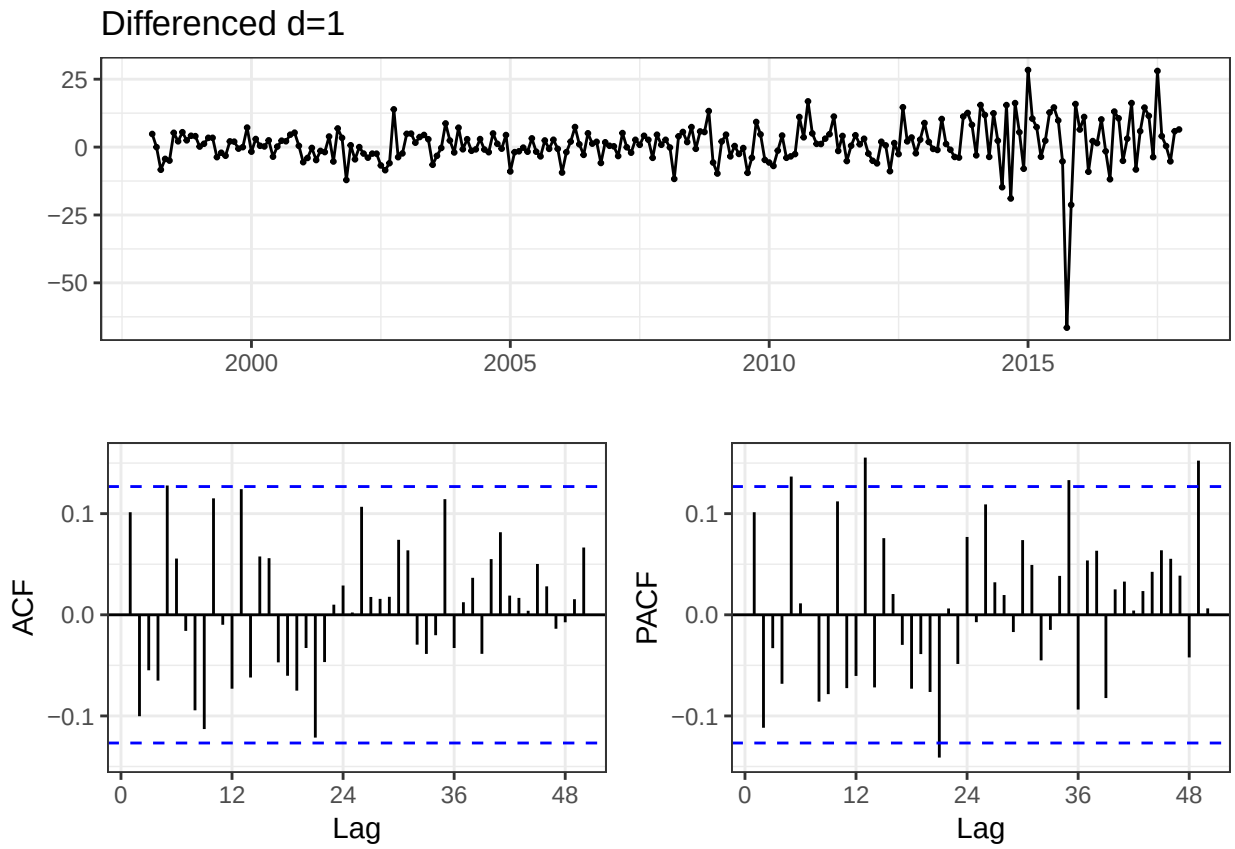


Figure 2:

```
#plot(SP_seasdiff, type="l", main="Seasonally Differenced")
if (sdif!=0){
SP_seasdiff <- diff(SP500_training, lag=frequency(SP500_training), differences=sdif)
#plot(SP_seasdiff, type="l", main="Seasonally Differenced")
}
```

```
# still not stationary!
# Make it stationary
tdiff<- ndiffs(SP_seasdiff)  # number of differences need to make it stationary
tdiff
```

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

```
# appears to be stationary
```

Figure 2 helps us confirm that we have stationarity and helps us decide which model we will use. It is important to keep in mind that we have a difference parameter equal to one (i.e., d = 1) because of the previous transformation we carried out.

From the above plots, we deduce that an MA(1) model maybe is best fits our data because the ACF cuts off at one significant lag and the PACF shows decay.

The augmented Dickey-Fuller test on differentiated data rejects the null hypotheses of non-stationarity. Plotting the differentiated series, we see an oscillating pattern around 0 with no visible strong trend except some trend around 2015. This suggests that differentiating of order 1 is sufficient, and should be included in

the model.

Since we are examining the differentiated time series, we have to use the combined model ARIMA. Thus, our model so far is ARIMA(0,1,1).

III - Fitting the Model using Built-in Functions

Our findings in part II suggest that the model ARIMA(0, 1, 1) might be the best fit. Fortunately, there is a function in R that we can use to test our findings.

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

We note that by using *Auto.arima*(), we do not need to remove trend and/or seasonality before fitting an ARIMA model. These models can handle certain types of trends and certain types of seasonality by themselves, or by including external regressors (the xreg argument, where you can include more complicated related effects such as moving holidays, or non-polynomial trends, breaks in the trend, etc).

```
fit<- auto.arima(SP500_training,trace=TRUE)
```

```
##
##  Fitting models using approximations to speed things up...
##
##  ARIMA(2,1,2)(1,0,1)[12] with drift         : 1697.342
##  ARIMA(0,1,0)            with drift         : 1681.303
##  ARIMA(1,1,0)(1,0,0)[12] with drift         : 1692.489
##  ARIMA(0,1,1)(0,0,1)[12] with drift         : 1681.497
##  ARIMA(0,1,0)                               : 1683.757
##  ARIMA(0,1,0)(1,0,0)[12] with drift         : 1692.094
##  ARIMA(0,1,0)(0,0,1)[12] with drift         : 1683.137
##  ARIMA(0,1,0)(1,0,1)[12] with drift         : 1694.152
##  ARIMA(1,1,0)            with drift         : 1681.49
##  ARIMA(0,1,1)            with drift         : 1679.948
##  ARIMA(0,1,1)(1,0,0)[12] with drift         : 1690.703
##  ARIMA(0,1,1)(1,0,1)[12] with drift         : 1692.716
##  ARIMA(1,1,1)            with drift         : 1680.354
##  ARIMA(0,1,2)            with drift         : 1679.279
##  ARIMA(0,1,2)(1,0,0)[12] with drift         : 1690.068
##  ARIMA(0,1,2)(0,0,1)[12] with drift         : 1680.973
##  ARIMA(0,1,2)(1,0,1)[12] with drift         : 1692.005
##  ARIMA(1,1,2)            with drift         : 1682.108
##  ARIMA(0,1,3)            with drift         : 1681.271
##  ARIMA(1,1,3)            with drift         : 1684.196
##  ARIMA(0,1,2)                               : 1681.672
##
##  Now re-fitting the best model(s) without approximations...
##
##  ARIMA(0,1,2)            with drift         : 1683.539
##
##  Best model: ARIMA(0,1,2)            with drift
```

Using *auto.ARIMA*(), we get that the best fit is an ARIMA(0,1,2) model.

IV - Diagnostic Checking

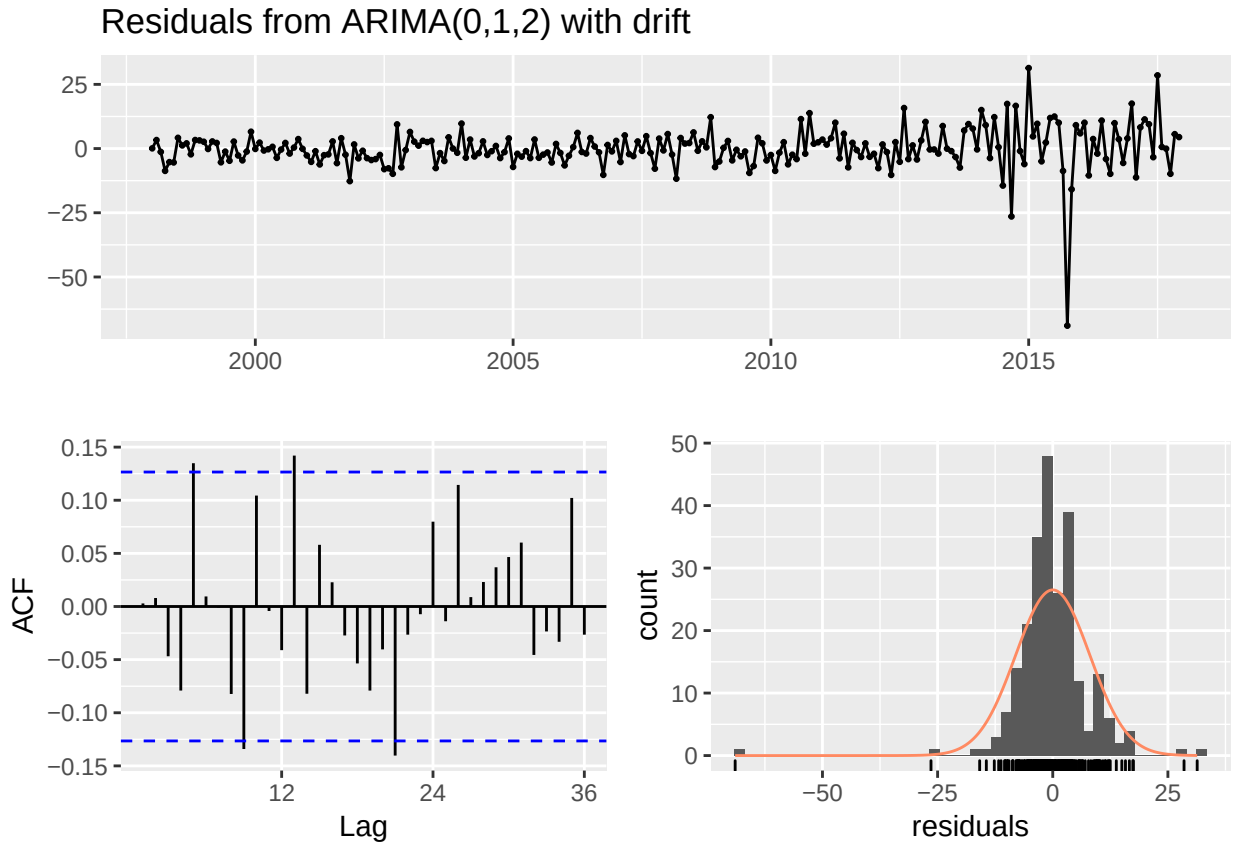
The next step is to run residual diagnostics to ensure our residuals are white noise. First we try to find out the pattern in the residuals of the chosen model by plotting the ACF of the residuals. We need to try modified models if the plot doesn't look like white noise. Once the residuals look like white noise, calculate forecasts. Here's the summary of our models ARIMA(0,1,1) and ARIMA(0,1,2) (using the *summary*()).

In order to do residual test we use *checkresiduals*() which will produce a time plot, ACF plot and histogram of the residuals (with an overlaid normal distribution for comparison), and do a Ljung-Box test with the correct degrees of freedom.

```
fitA<- auto.arima(SP500_training )
summary(fitA)
```

```
## Series: SP500_training
## ARIMA(0,1,2) with drift
##
## Coefficients:
##         ma1     ma2   drift
##       0.113  -0.121  1.1202
## s.e.  0.065   0.072  0.5170
##
## sigma^2 estimated as 65.66:  log likelihood=-837.68
## AIC=1683.37   AICc=1683.54   BIC=1697.27
##
## Training set error measures:
##                        ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 0.0001710109 8.035194 5.119552 -0.4206221 3.538067 0.2303483
##                      ACF1
## Training set 0.002896088
```

```
checkresiduals(fitA)
```

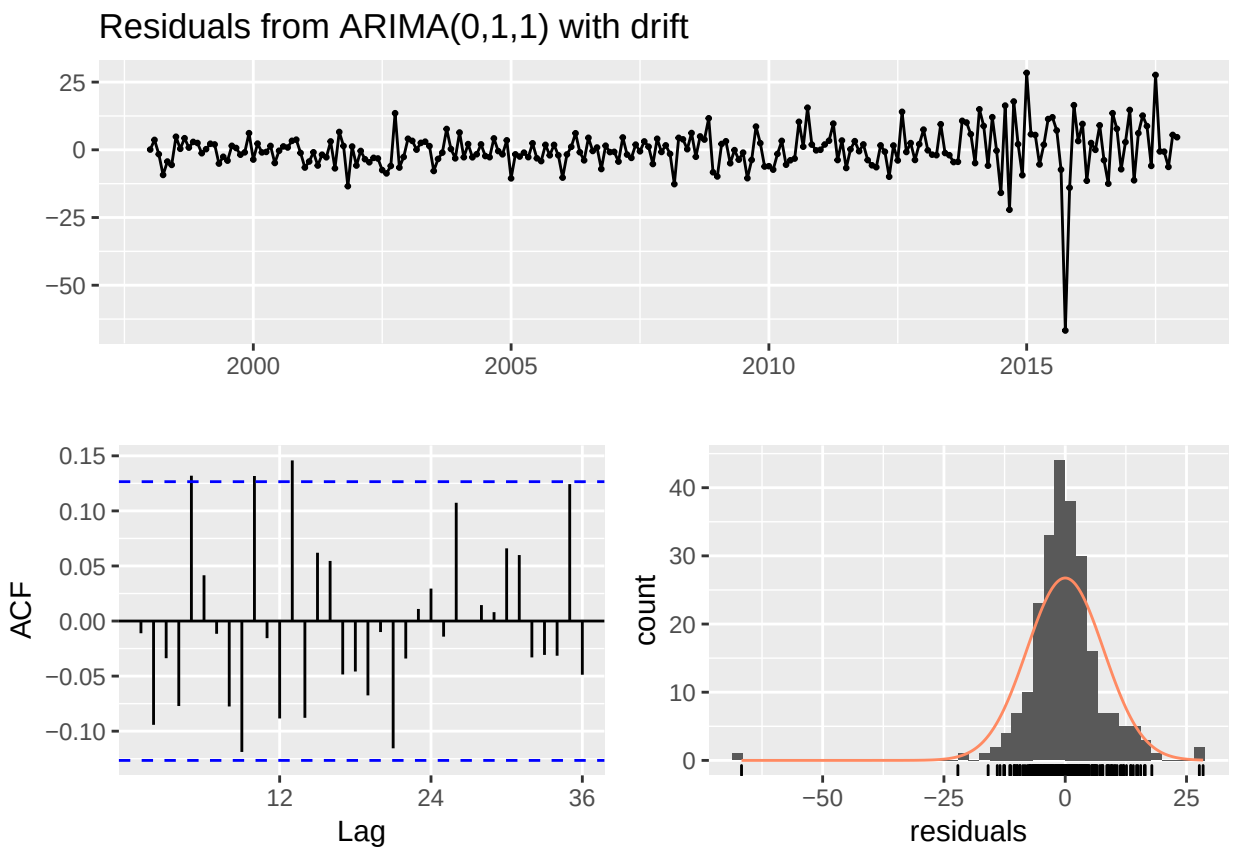



```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,2) with drift
## Q* = 34.065, df = 21, p-value = 0.03567
##
## Model df: 3.   Total lags used: 24
```

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

```
## Series: deseasonal_cnt
## ARIMA(0,1,1) with drift
##
## Coefficients:
##          ma1   drift
##       0.1261  1.1420
## s.e.  0.0707  0.5765
##
## sigma^2 estimated as 63.22:  log likelihood=-833.65
## AIC=1673.3   AICc=1673.4   BIC=1683.73
##
## Training set error measures:
##                       ME     RMSE      MAE        MPE    MAPE      MASE
## Training set -0.001354994 7.901239 5.083687 -0.3789138 3.50852 0.2287346
##                      ACF1
## Training set -0.01109008
```

```
checkresiduals(fitB)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1) with drift
## Q* = 35.892, df = 22, p-value = 0.03119
##
## Model df: 2.   Total lags used: 24
```

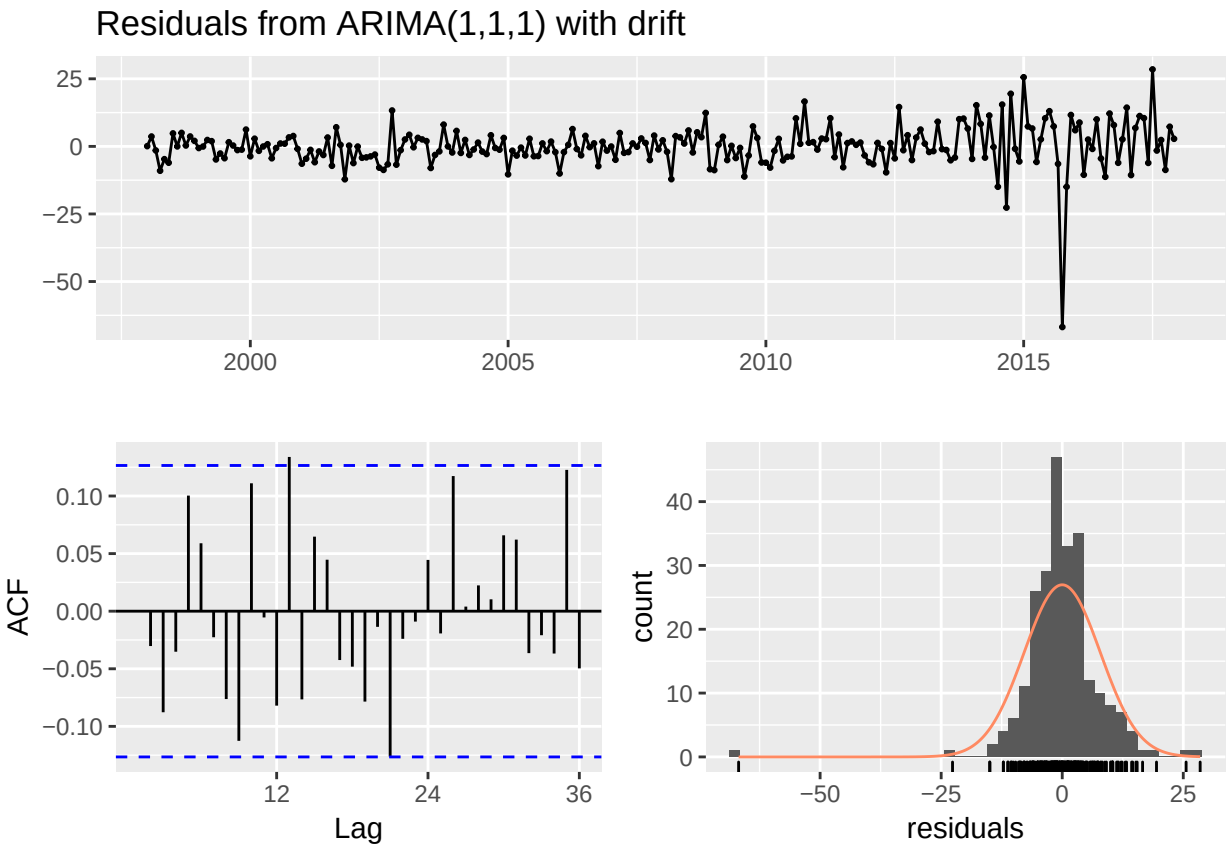
In addition we tested ARIMA(1,1,1) which is combination of our first findinf AR(1) and MA(1) with d=1 .

```
fitC <- Arima(deseasonal_cnt, order = c(1,1,1),include.drift = TRUE)
summary(fitC)
```

```
## Series: deseasonal_cnt
## ARIMA(1,1,1) with drift
##
## Coefficients:
##           ar1     ma1   drift
##       -0.6571  0.7763  1.1348
## s.e.   0.2292  0.1942  0.5457
##
## sigma^2 estimated as 62.74:  log likelihood=-832.25
## AIC=1672.5   AICc=1672.67   BIC=1686.4
##
## Training set error measures:
```

```
##                         ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -3.873311e-05 7.854286 5.078677 -0.3942068 3.501005 0.2285092
##                      ACF1
## Training set 0.0008759913
```

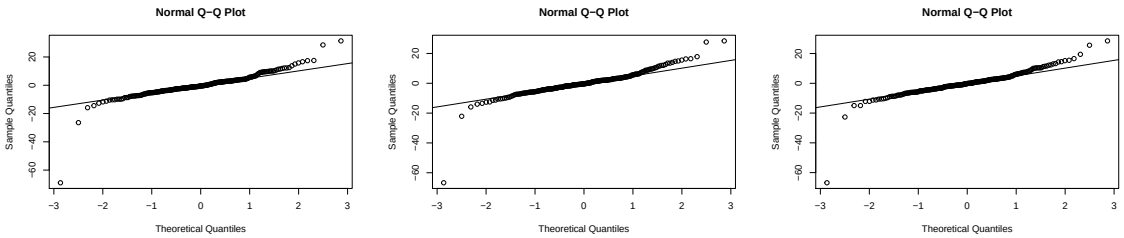
```
checkresiduals(fitC)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,1) with drift
## Q* = 30.739, df = 21, p-value = 0.07811
##
## Model df: 3.   Total lags used: 24
```

Based on our diagnostic plot, the residuals to seem to display a normal distribution for all models. This is important because a good forecasting model will have zero correlation between its residuals. It naturally follows that if you can forecast the error terms, then a better model must exist.

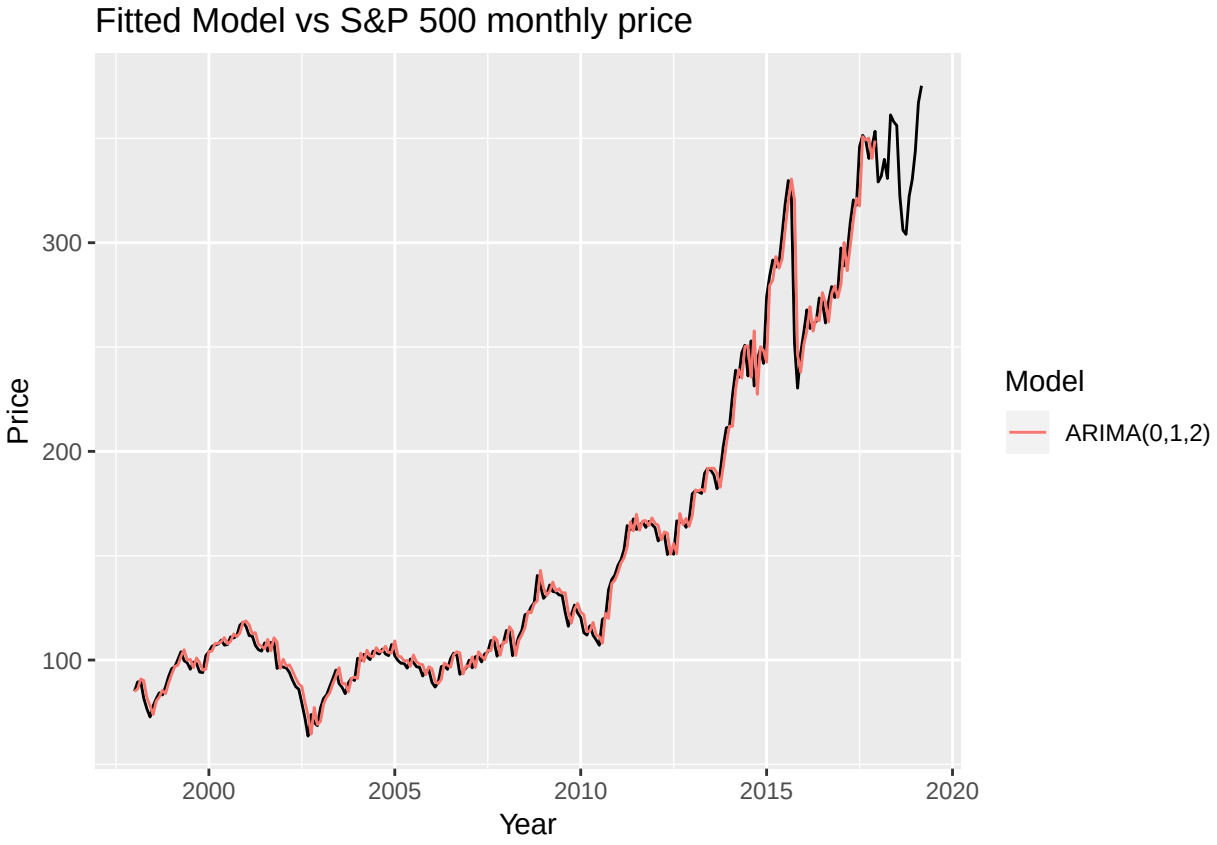
Recall Box-Ljung test is a test of independence at all lags up to the one specified. Instead of testing randomness at each distinct lag, it tests the "overall" randomness based on a number of lags. It is applied to the residuals of a fitted ARIMA model, not the original series, and in such applications the hypothesis actually being tested is that the residuals from the ARIMA model have no autocorrelation. Note that the p-values for the Ljung-Box Q test for ARIMA(1,1,1) well above 0.05, indicating "non-significance."

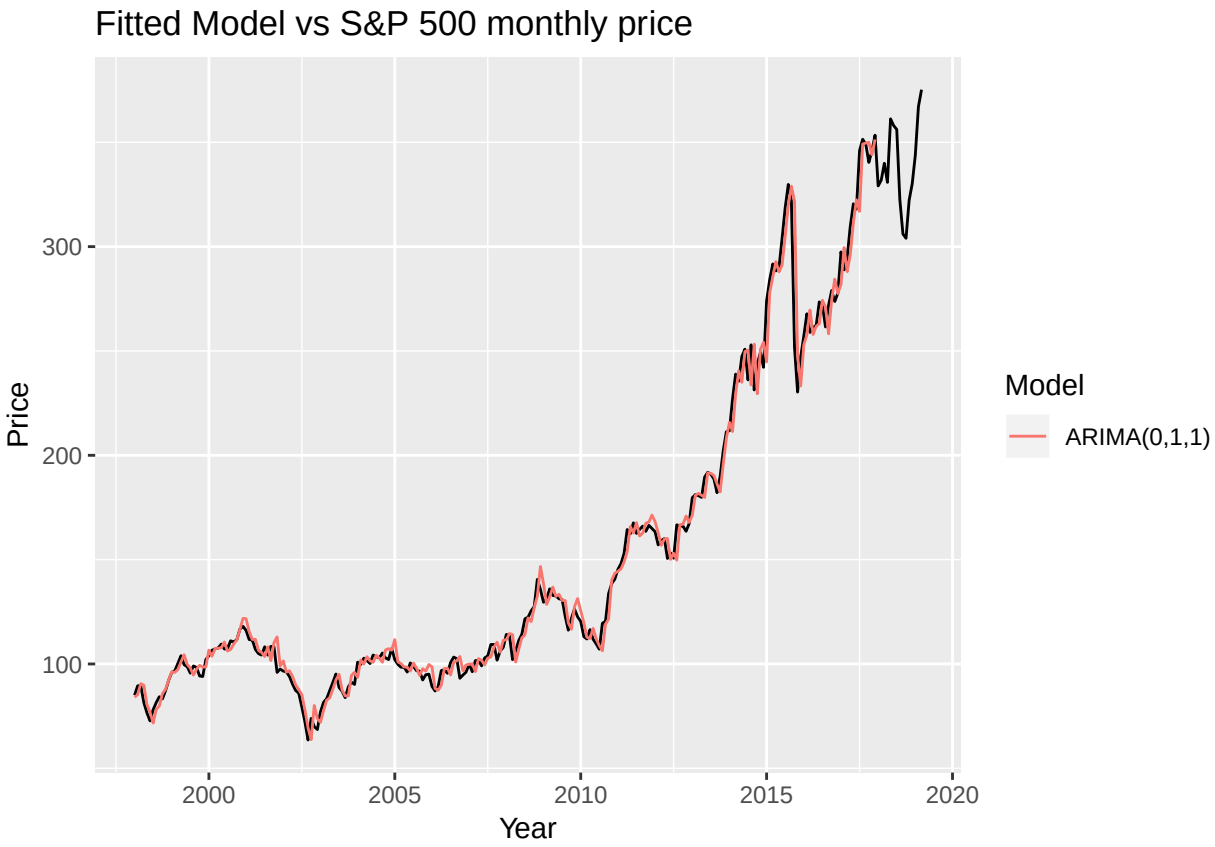


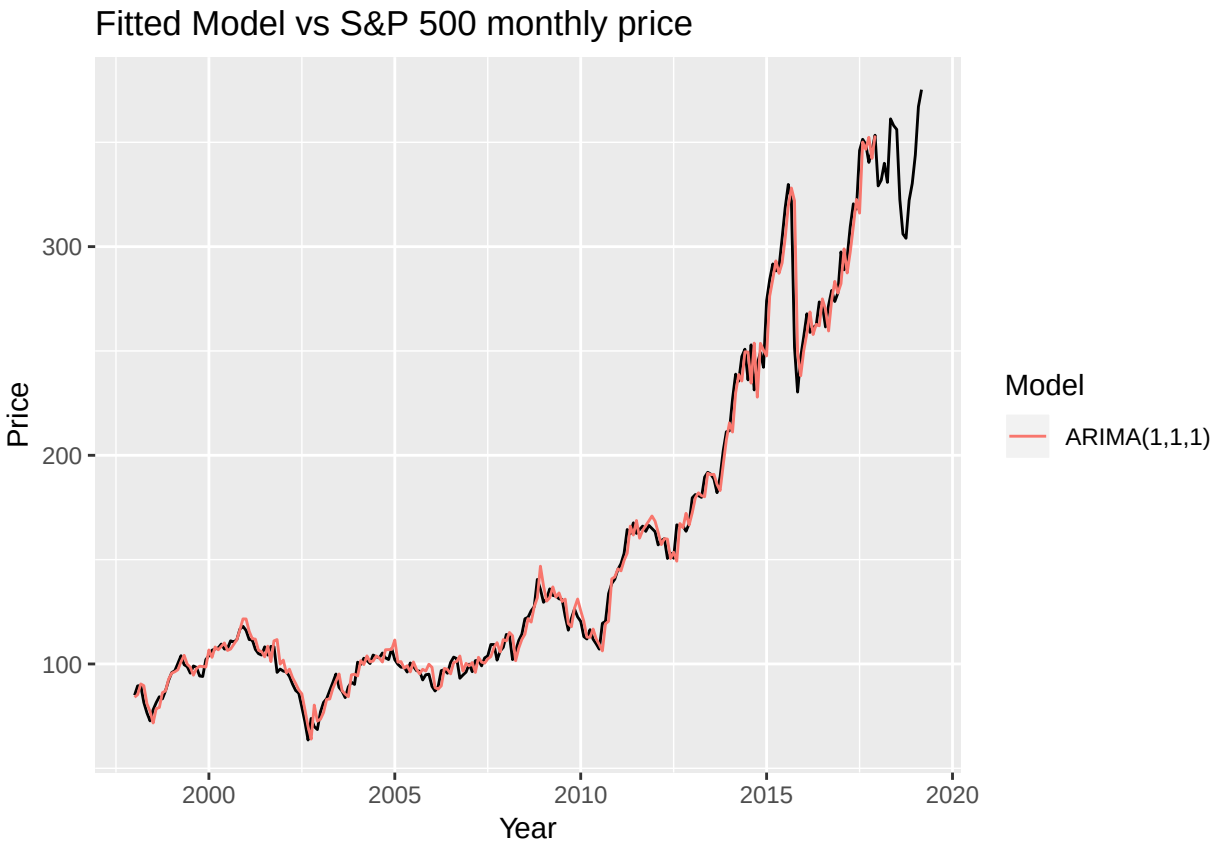
All models, ARIMA(0,1,2) and ARIMA(0,1,1),ARIMA(1,1,1) (from left to right), seem normal based on Q-Q plots. The values are normal as they rest on a line and are not all over the place.

For ARIMA(0,1,2), the auto-correlation plot show that very few of the sample autocorrelations for lags 1-36 exceed the significance bounds, the result for ARIMA(0,1,1) are comparable, and overall ARIMA(1,1,1) seems better models since the residuals shows no significant autocorrelations.

The following figures show the fitted model vs actual data for all models.



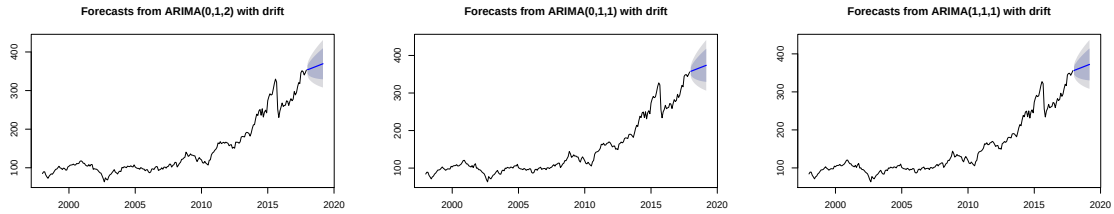

Fitted Model vs S&P 500 monthly price
Price
300
200
100
2000
2005
2010
2015
2020
Year
Model
ARIMA(0,1,1)



With all these tests complete, we can conclude that all tested models, ARIMA(0,1,2) and ARIMA(0,1,1) and ARIMA(1,1,1), are plausible models for our data. As all the graphs are in support of the assumption that there is no pattern in the residuals, we can go ahead and calculate the forecast.

V - Forcasting

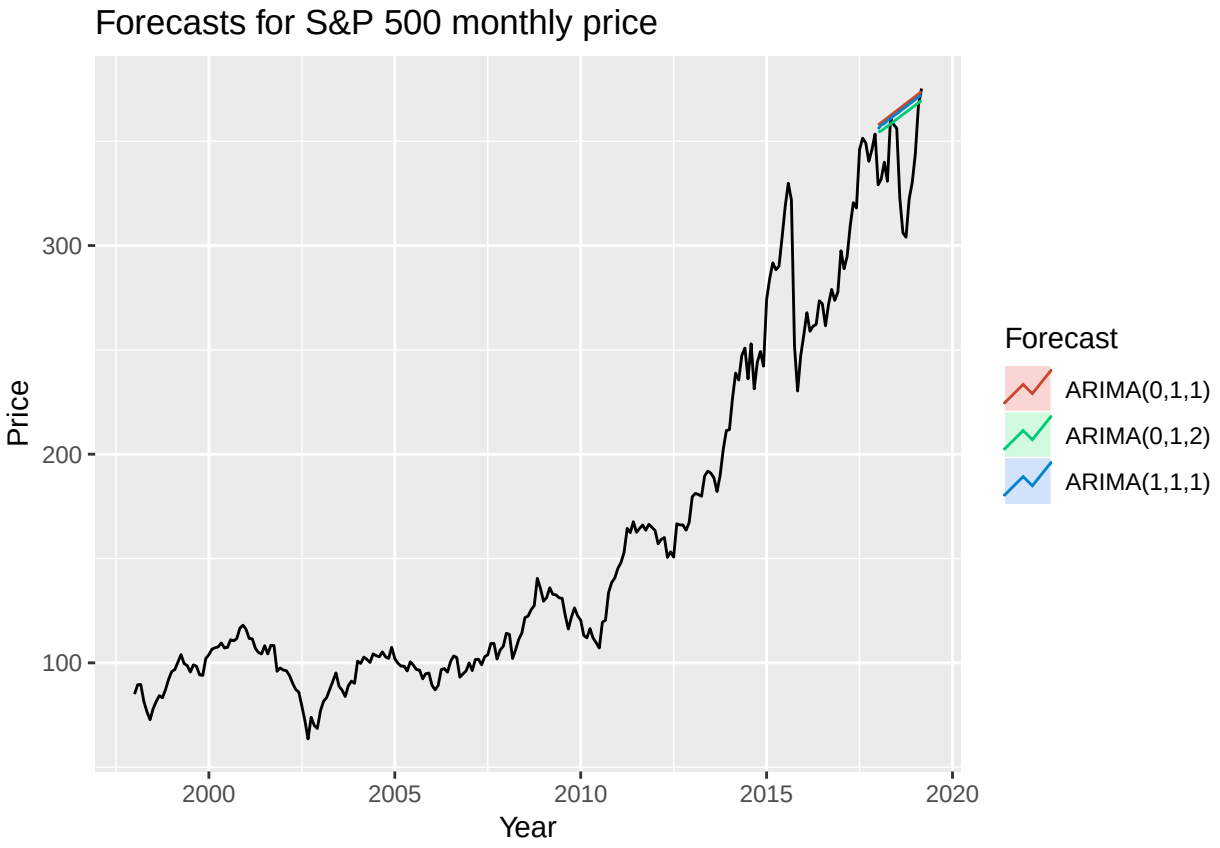
Once the residuals look like white noise, then we can do the prediction. We use a built-in function $forcast()$ to predict S&P500 prices for 2018-2019, about 15 months. We do this all for models ARIMA(0,1,2), ARIMA(0,1,1) and ARIMA(1,1,1). The results for both models are very comparable. The prediction also seems to be accurate, our reserved test data falls neatly withing our confidence bounds, and share a trend.

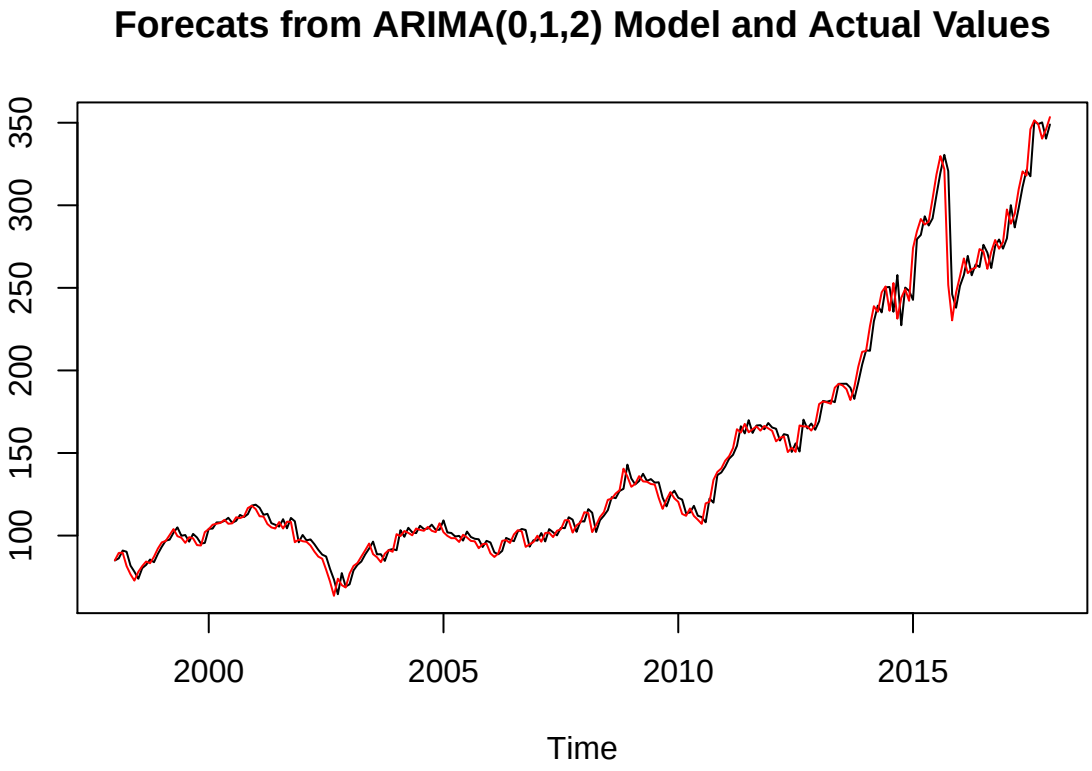


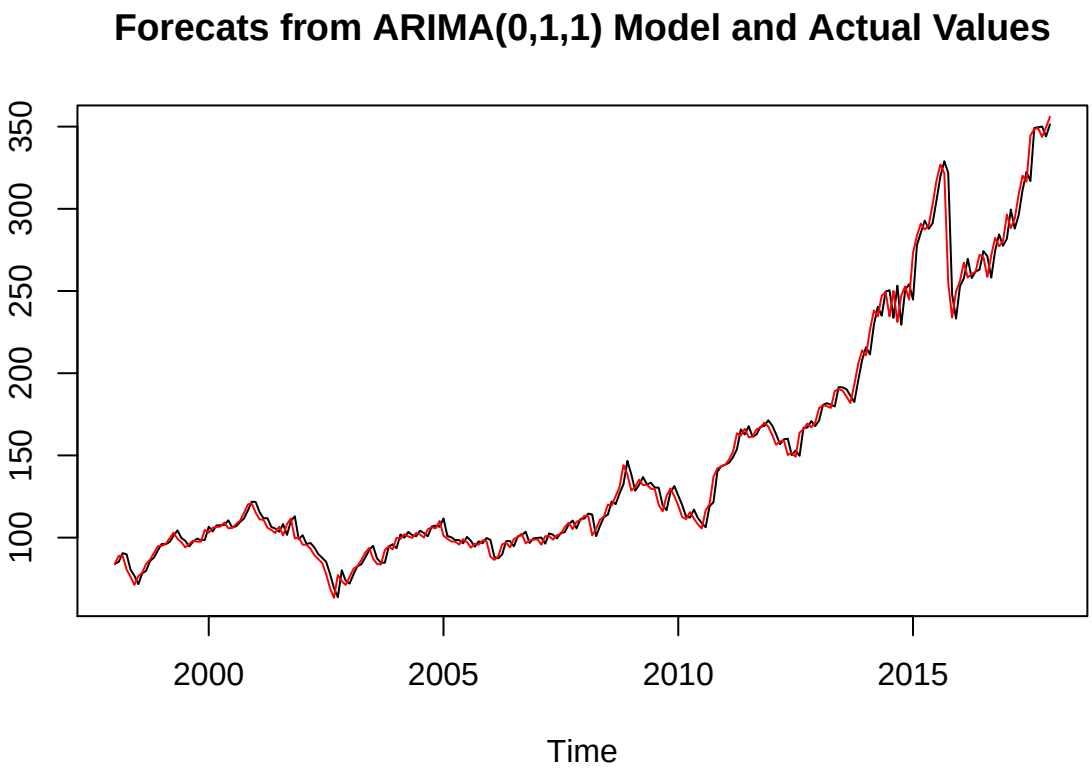
In the following we plot the real data vs. 1-step prediction. The prediction is based on lag 1. This gives us a well-fitted prediction, though it seems to be delayed by a short period. This is obvious since we use past information to evaluate the prediction.

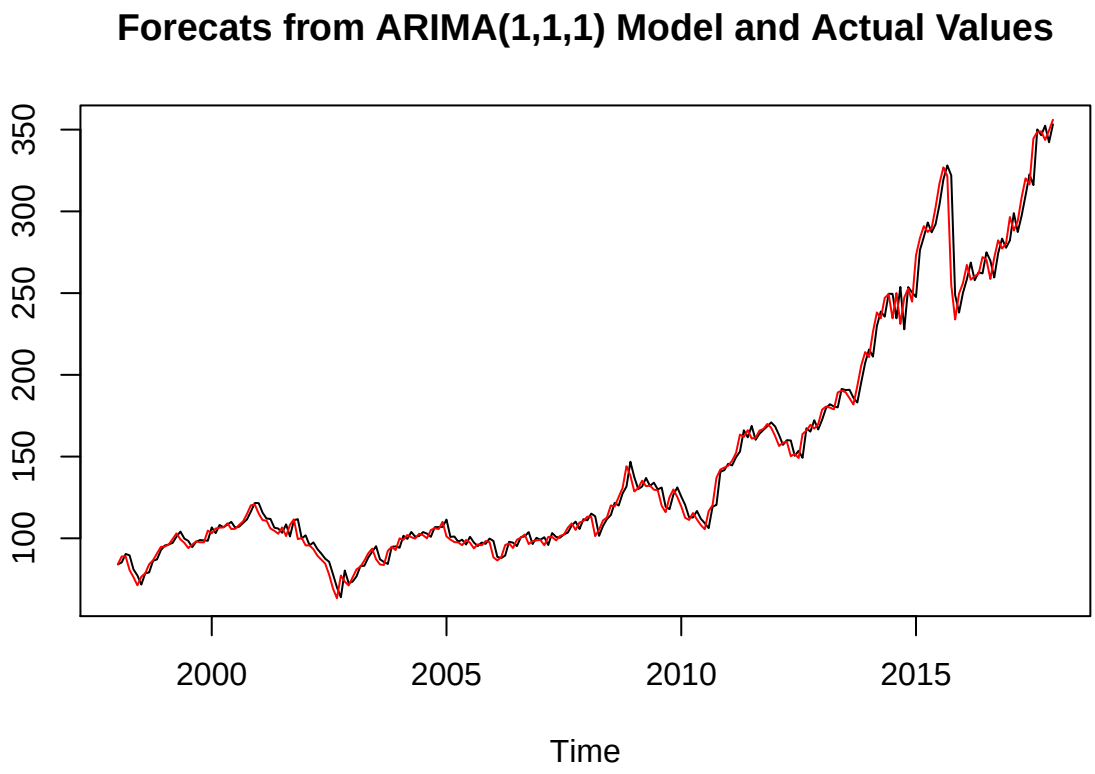
```
autoplot(window(sp_500, start=1998)) +
  autolayer(fcA, series="ARIMA(0,1,2)", PI=FALSE) +
```

```
autolayer(fcB, series="ARIMA(0,1,1)", PI=FALSE) +
autolayer(fcC, series="ARIMA(1,1,1)", PI=FALSE) +
xlab("Year") + ylab("Price") +
ggtitle("Forecasts for S&P 500 monthly price") +
guides(colour=guide_legend(title="Forecast"))
```

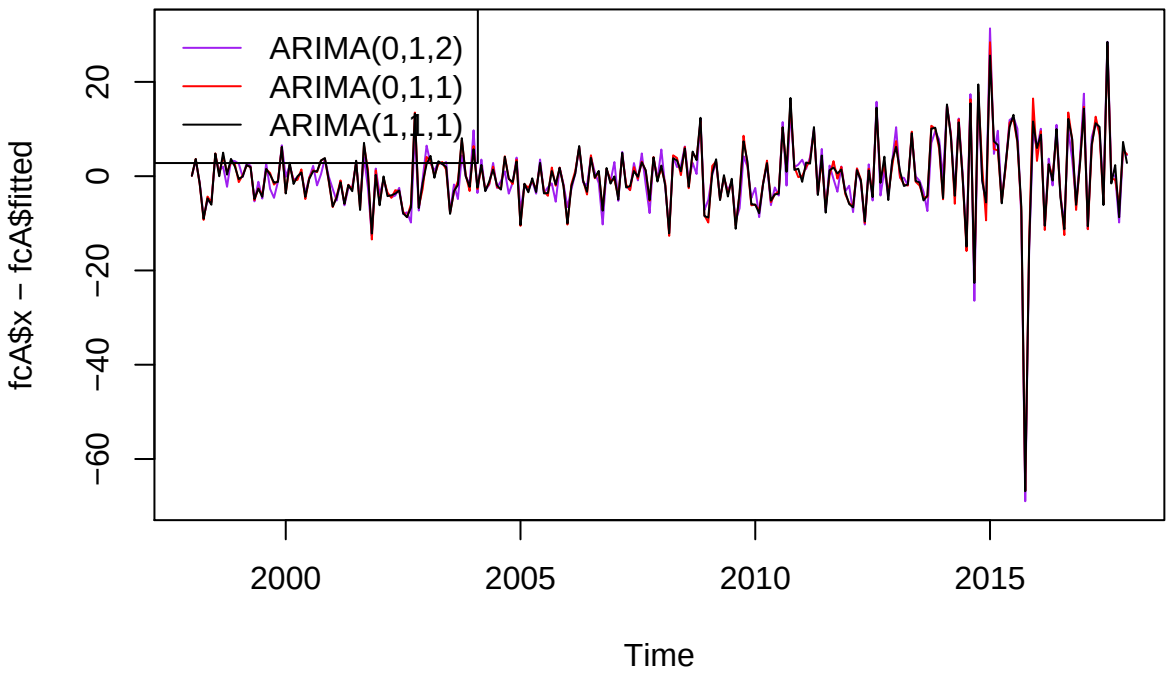



Forecats from ARIMA(0,1,2) Model and Actual Values
100
150
200
250
300
350
2000
2005
2010
2015
Time


Forecats from ARIMA(0,1,1) Model and Actual Values
100
150
200
250
300
350
2000
2005
2010
2015
Time



In the following, we shows the difference between the actual and fitted models for all models.



VI - Conclusion

All in all, we managed to find significant insight into the S&P500 using varied techniques. Manual and automatic model selection methods yield similar results. We conclude that the S&P500 data is reasonably well fitted to a ARIMA(0,1,1), ARIMA(0,1,2) and ARIMA(1,1,1) processess. Amonge all models, ARIMA(1,1,1) by far the best model choice to fit our time series. According to the following table, surprisingly ARIMA(1,1,1) has lowest values in all criteria as well as smallest errors.

	Information Criteria				Errors With Training Set		
Models	σ^2	AIC	AICc	BIC	ME	RMSE	MAE
ARIMA(0,1,2)	65.66	1683.37	1697.27	1697.27	0.00017	8.03	5.1195
ARIMA(0,1,1)	63.22	1673.3	1673.4	1683.73	-0.00135	7.90	5.0836
ARIMA(1,1,1)	62.74	1672.5	1672.67	1686.4	0.00000	7.85	5.0784

Additionally, the forecasts from these the processes are relatively similar (It seems ARIMA(1,1,1) has the better bounds). In general, we can reconfirm that manual and automatic model selection methods are all serviceable. Eventhough we found that using the $auto.ARIMA()$ function was much simpler to implement, but among all models has worth results.

In order to extend this analysis, our recommendation would be use ARCH or GARCH models on the data. Then, to do a larger comparison, including forecasts from those models.