Choosing (p,d,q) for ARIMA Models

Kushal Kharel

10/25/2021

ARIMA Models for Stock Price Prediction

A time series is a sequence where we record a metric over regular intervals and in this case we will analyze stock prices. The forecasting refers to the future values that this sequence can take. In this discussion, we are going to talk about aaple prices but we are only going to use the adjusted daily closing price feature to predict the future value which is also called the uni-variate time series forecasting.

ARIMA model which is known as Auto Regressive Integrated Moving Average is a forecasting algorithm that takes into account previous values to predict future values because it considers the information that is found in those past values which can be indicative of future values. So in short, ARIMA model explains a time series based on its lags and its lags forecast errors. Consider a model defined below:

$$Y_t = c + \phi_1 Y_{t-1} + ... + \phi_p Y_{t-p} + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q} + \epsilon_t$$

where, the AR part or lags are $\phi_1 Y_{t-1} + ... + \phi_p Y_{t-p}$, the MA part or its lagged forecast errors are $\theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q} + \epsilon_t$, c is the intercept. The ARIMA model is characterized by three terms (p, d, q) which is very important to fit the model to the series. p refers to the order of the autoregressive(AR) term, d refers to the order of differencing that is the number of differencing required to make the time series stationary and q is the order of the MA term. We need stationary time series to fit ARIMA model. So, what is stationarity? It refers to the fact that the price series is mean reverting. As we know, most prices series are not mean reverting and if it was then it will be very easy to profit from stock market as we can buy low and sell high or short high and buy low. But, we know that returns are more likely to mean revert because they are distributed randomly around the zero mean. Let us go through the example to see how everything works in practice.

```
library(tidyquant)
```

```
## Warning: package 'tidyquant' was built under R version 4.0.5
```

```
## Loading required package: lubridate
```

```
## Warning: package 'lubridate' was built under R version 4.0.3
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
## Loading required package: PerformanceAnalytics

## Warning: package 'PerformanceAnalytics' was built under R version 4.0.5

## Loading required package: xts

## Warning: package 'xts' was built under R version 4.0.2

## Loading required package: zoo

## Warning: package 'zoo' was built under R version 4.0.2

##
## Attaching package: 'zoo'

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

##
## Attaching package: 'PerformanceAnalytics'

## The following object is masked from 'package:graphics':
##
##     legend

## Loading required package: quantmod

## Warning: package 'quantmod' was built under R version 4.0.3

## Loading required package: TTR

## Warning: package 'TTR' was built under R version 4.0.2

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

## == Need to Learn tidyquant? ============================================
## Business Science offers a 1-hour course - Learning Lab #9: Performance Analysis & Portfolio Optimiza
## </> Learn more at: https://university.business-science.io/p/learning-labs-pro </>
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
aapl <- tq_get('AAPL',
               from = "2020-01-01",
               to = "2021-01-01",
               get = "stock.prices")
aapl %>%
  ggplot(aes(x = date, y = adjusted)) +
  geom_line() +
  theme_classic() +
  labs(x = 'Date',
       y = "Adjusted Price",
       title = "Apple price chart") +
  scale_y_continuous(breaks = seq(0,300,10))
```


```
head(aapl)
```

```
## # A tibble: 6 x 8
##   symbol date        open  high   low close    volume adjusted
##   <chr>  <date>     <dbl> <dbl> <dbl> <dbl>     <dbl>    <dbl>
## 1 AAPL   2020-01-02  74.1  75.2  73.8  75.1 135480400     74.1
## 2 AAPL   2020-01-03  74.3  75.1  74.1  74.4 146322800     73.4
## 3 AAPL   2020-01-06  73.4  75.0  73.2  74.9 118387200     74.0
## 4 AAPL   2020-01-07  75.0  75.2  74.4  74.6 108872000     73.6
## 5 AAPL   2020-01-08  74.3  76.1  74.3  75.8 132079200     74.8
## 6 AAPL   2020-01-09  76.8  77.6  76.6  77.4 170108400     76.4
```

```
aaplPrice = aapl[8]
aaplPrice = ts(aaplPrice)
```

In order to find whether the time series is stationary or not, we can use augumented Dicky-Fuller test and what this test does is that it tries to prove the null hypothesis wrong and our null hypothesis is that the time series is non-stationary. So, if the p-value of the test is less than the significance value of 0.05 then we can reject the null hypothesis and infer that the time series might be stationary. If the p-value if higher than 0.05 in our case then we are going to need to find the order of differencing. Let us run the augumented Dickey-Fuller test.

```
library(aTSA)
```

```
## Warning: package 'aTSA' was built under R version 4.0.3
```

```
##
## Attaching package: 'aTSA'
```

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

```
adf.test(aaplPrice)
```

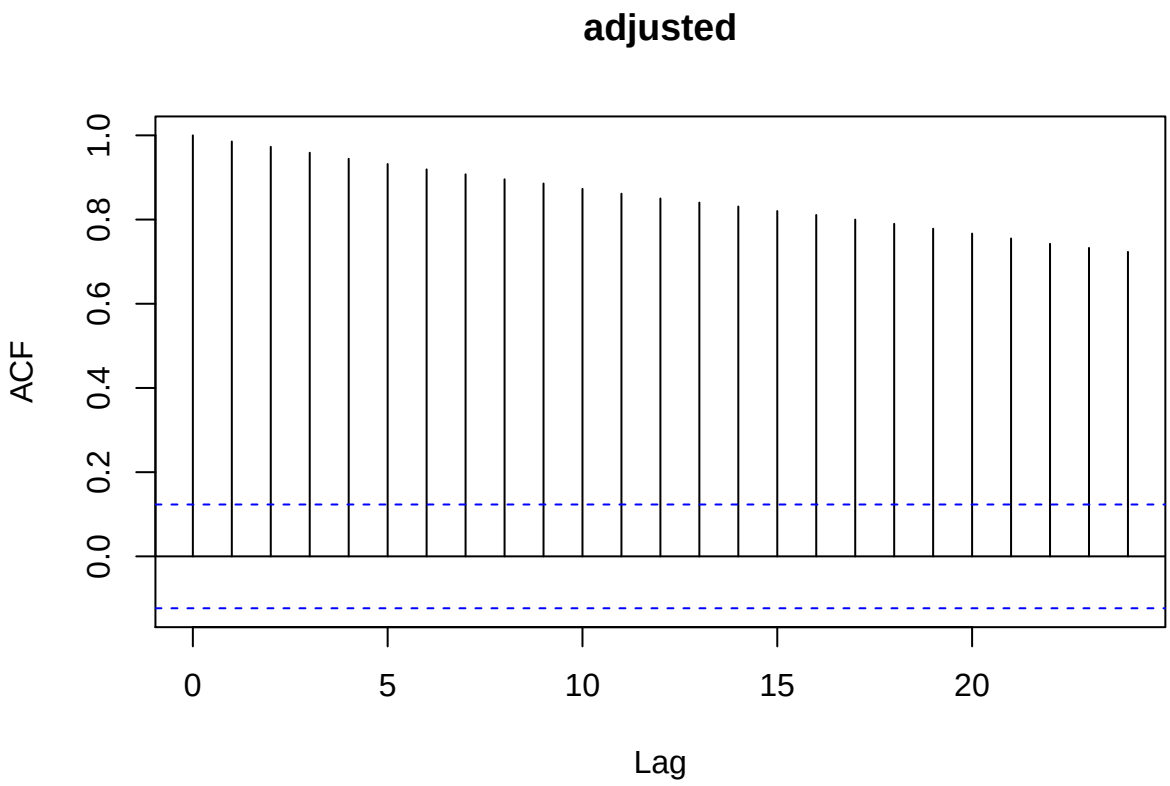
```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag  ADF p.value
## [1,]   0 1.29   0.951
## [2,]   1 1.63   0.975
## [3,]   2 1.53   0.968
## [4,]   3 1.53   0.968
## [5,]   4 1.58   0.971
## Type 2: with drift no trend
##      lag    ADF p.value
## [1,]   0 -0.385   0.905
## [2,]   1 -0.134   0.941
## [3,]   2 -0.191   0.933
## [4,]   3 -0.192   0.933
## [5,]   4 -0.110   0.944
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -2.41   0.405
## [2,]   1 -2.07   0.548
## [3,]   2 -2.15   0.512
## [4,]   3 -2.14   0.518
## [5,]   4 -2.12   0.527
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

We can see from above results that the p-values are quite high for different lags and we can be sure that this price series is not stationary. In order to ARIMA models to work, we need to difference the values of the

time series. As in this case, we need to transform into returns as it randomly distributes around the zero mean. We just simply need to subtract the previous value from the current value. If we just difference once, we might not get the stationary series so we might need to difference multiple times which is the d term in our ARIMA model.

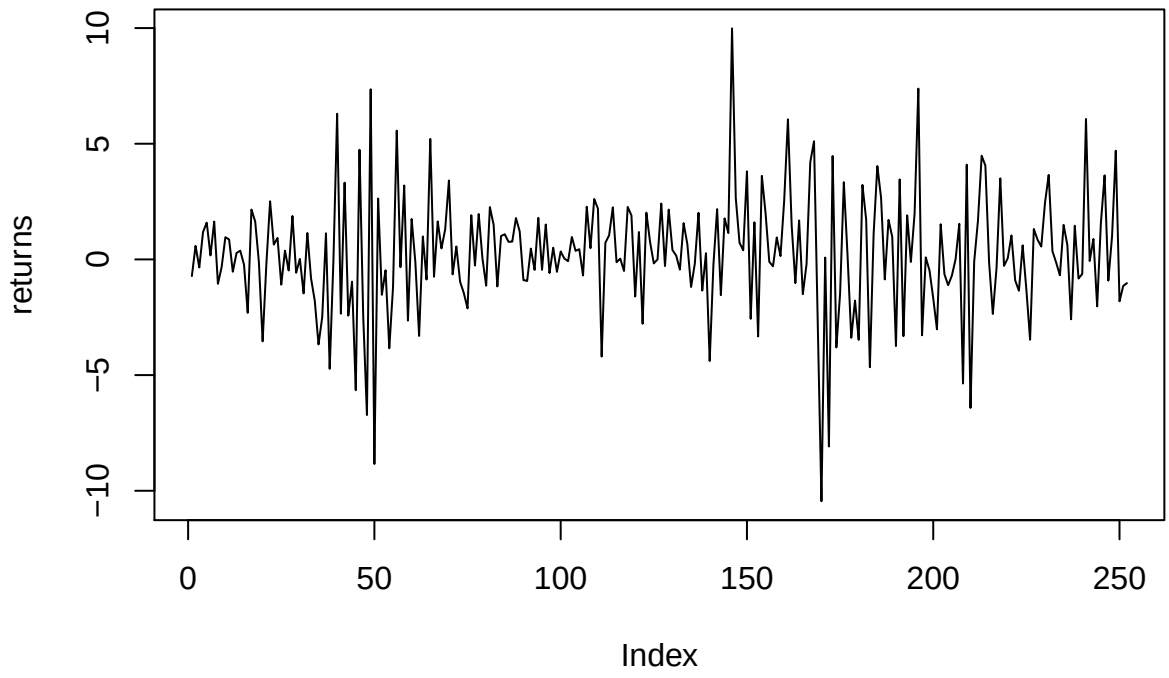
The minimum number of differencing operations that we need to make the time series stationary needs to be imputed in our model. So if the time series is already stationary then d is zero. In order to find the order of differencing manually, we are going to use the acf plot which tells us how many terms are required to remove any autocorrelation in the series.

```
acf(aaplPrice)
```



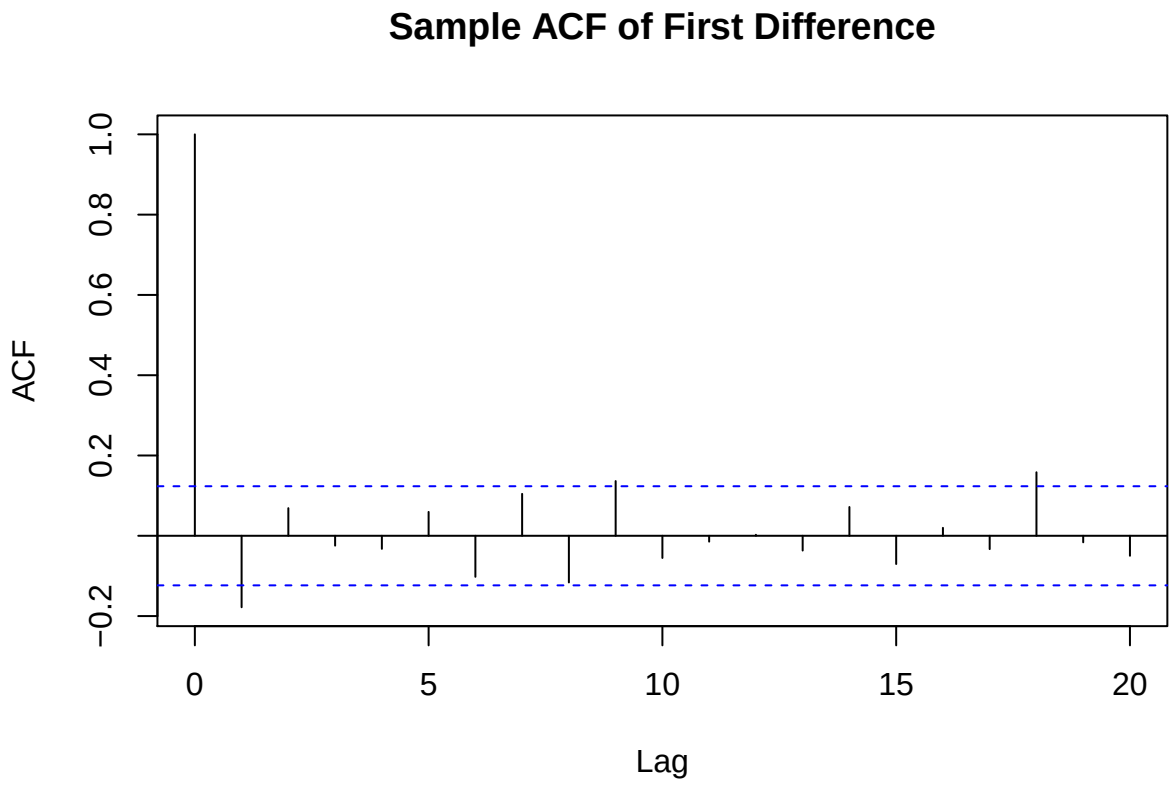
When we difference the series once, we get the return of the adjusted close prices.

```
returns = as.vector(diff(aaplPrice))
plot(returns, type = 'l')
```



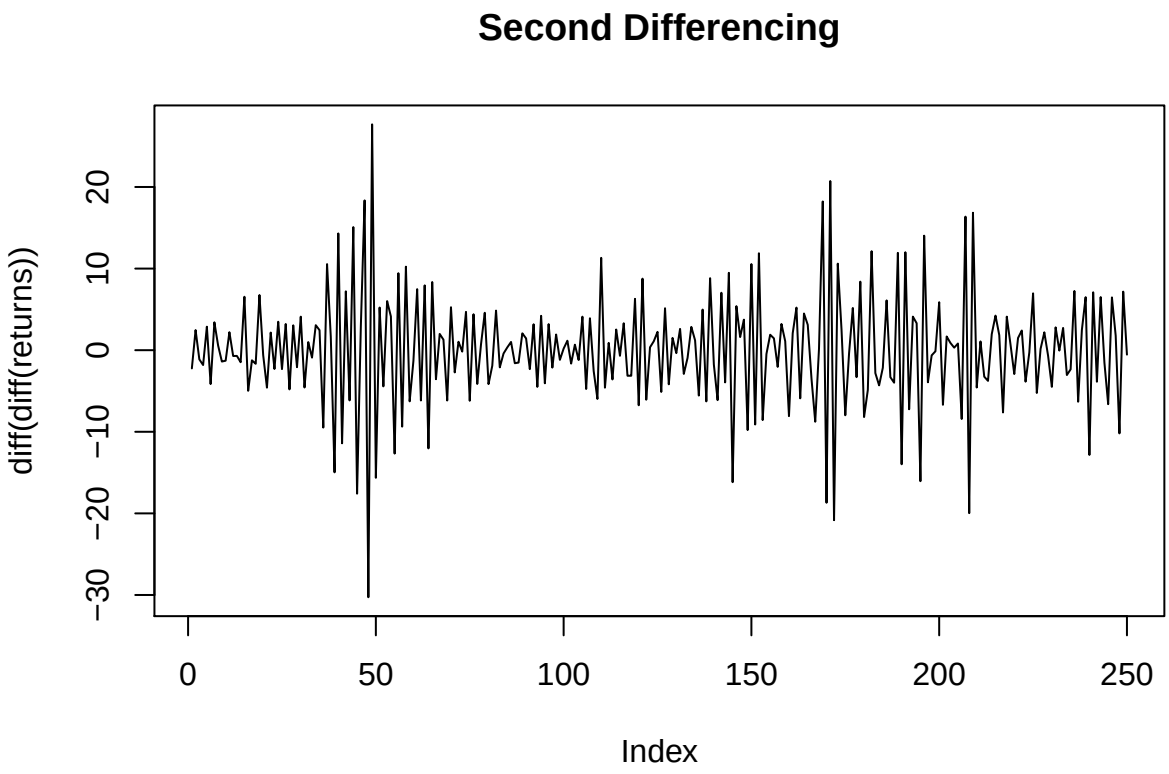
We can see from the plot above that returns are distributed randomly around zero mean. Now, let us do the see the acf plot of the returns.

```
acf(returns,lag.max = 20, main="Sample ACF of First Difference")
```

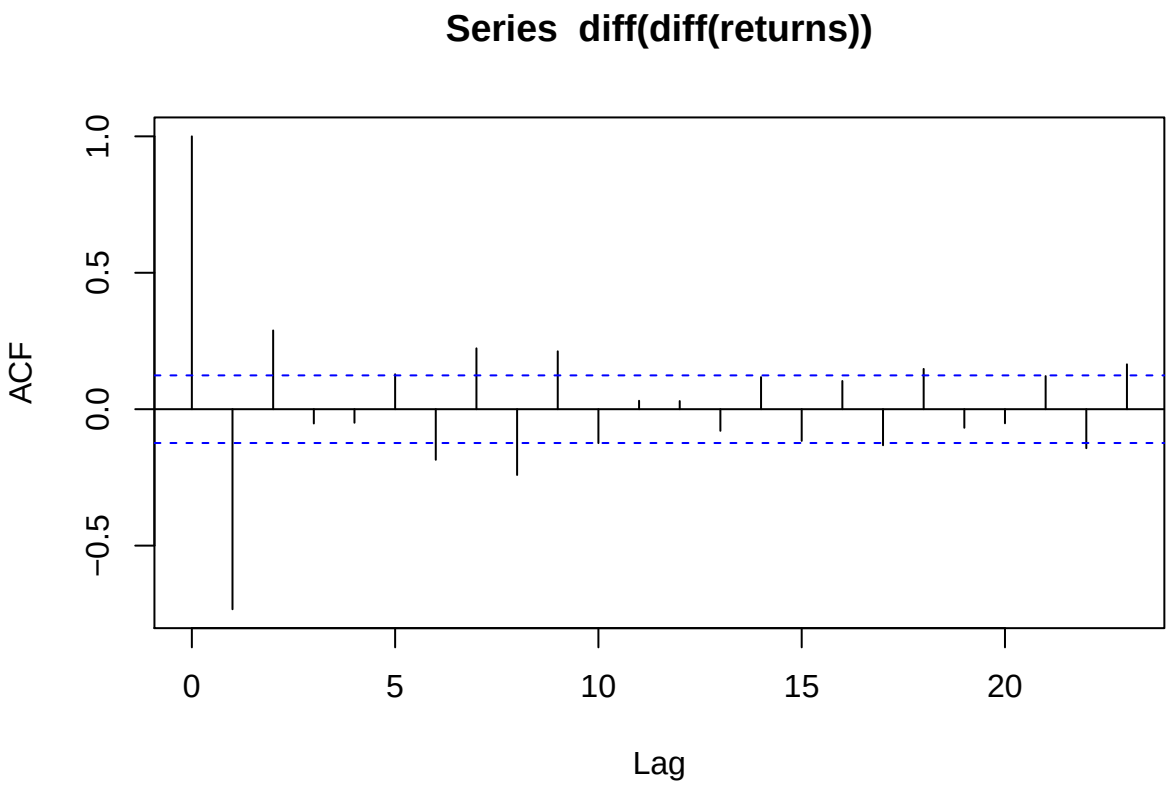


Let us try to difference twice and see the acf plot.

```
plot(diff(diff(returns)), type = 'l', main = "Second Differencing")
```




```
acf(diff(diff(returns)))
```



The acf plot for first and second difference looks very similar. There is not much difference between the two except that in the second differencing the lag goes into the far negative which indicates that we have overdifferenced our series. Therefore, we are just going to choose our order of differencing as 1.

Or, we can just use the ndiffs function from forecast package to get the number of differencing required.

```
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 4.0.5
```

```
##
## Attaching package: 'forecast'
```

```
## The following object is masked from 'package:aTSA':
##
##     forecast
```

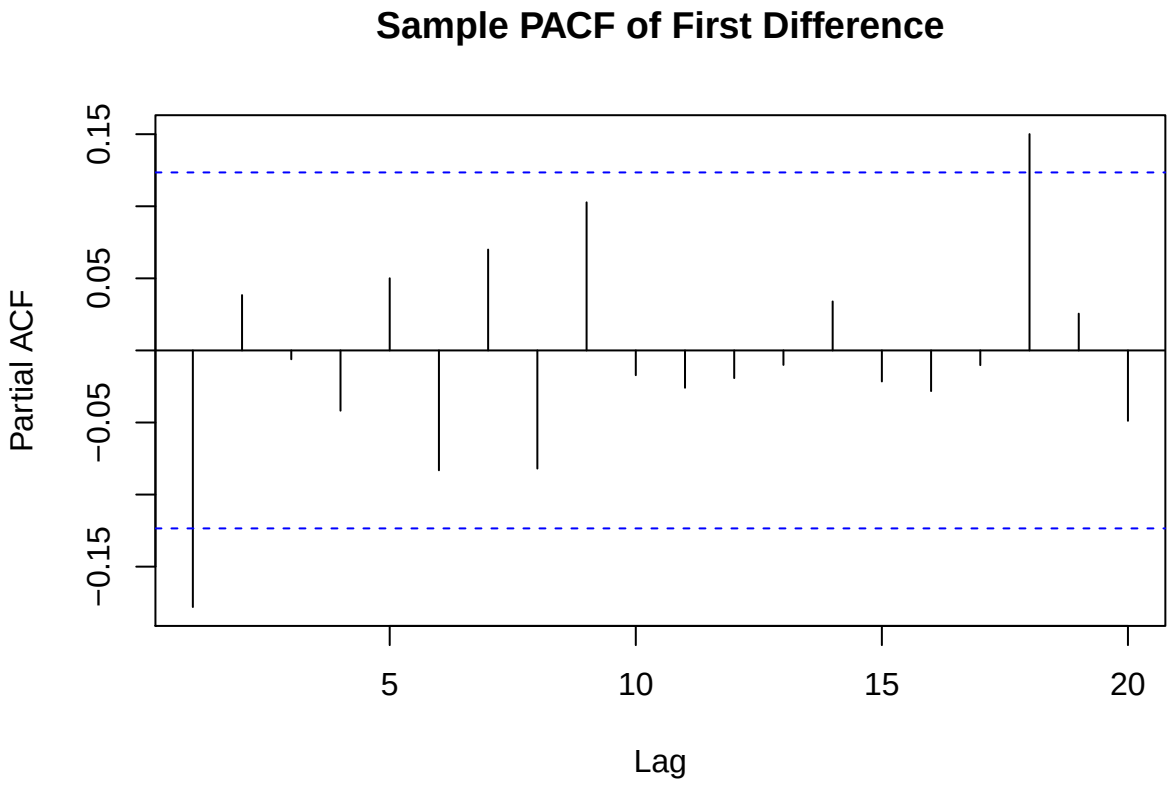
```
ndiffs(aaplPrice,alpha= 0.05,test="adf")
```

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

We got the same results from both the package and by analyzing acf plot.

Now let us discuss the p term which is the order of the autoregressive term. It refers to the number of lags to be used as predictors. We can find out the required number of AR terms by inspecting PACF plot. The partial autocorrelation represents the correlation between the series and its lags.

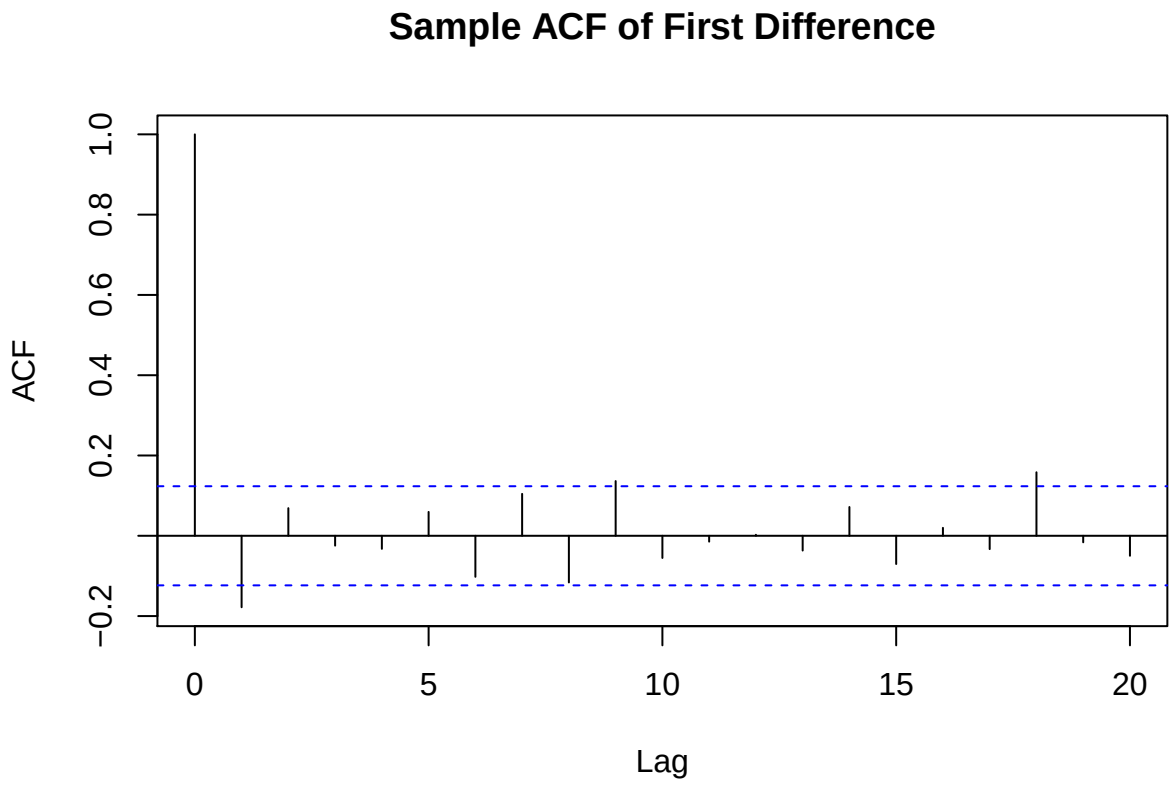
```
pacf(returns,lag.max = 20, main="Sample PACF of First Difference")
```



We can observe from the plot above that lag 1 and lag 18 stands out well above the significance level. We are going to use lag 1 for this case.

Now, the q term is the order of the Moving Average(MA). It refers to the number of lagged forecast erros that should go into the ARIMA model. We can look at the ACF plot for the number of MA terms.

```
acf(returns,lag.max = 20, main="Sample ACF of First Difference")
```



We can see that the lag 1,9 and 18 stands out well above the significane line. So, lag 1 maybe suitable for this model. Let us choose 9 as our lag for q term.

Now, we have p, d, q terms which are set as $ARIMA(1, 1, 1)$ for this purpose.

Finally, let us fit the ARIMA model.

```
model = arima(returns,order=c(1,1,1))
model
```

```
##
## Call:
## arima(x = returns, order = c(1, 1, 1))
##
## Coefficients:
##           ar1      ma1
##       -0.1743  -1.0000
## s.e.   0.0621   0.0255
##
## sigma^2 estimated as 6.46:  log likelihood = -593.22,  aic = 1192.45
```

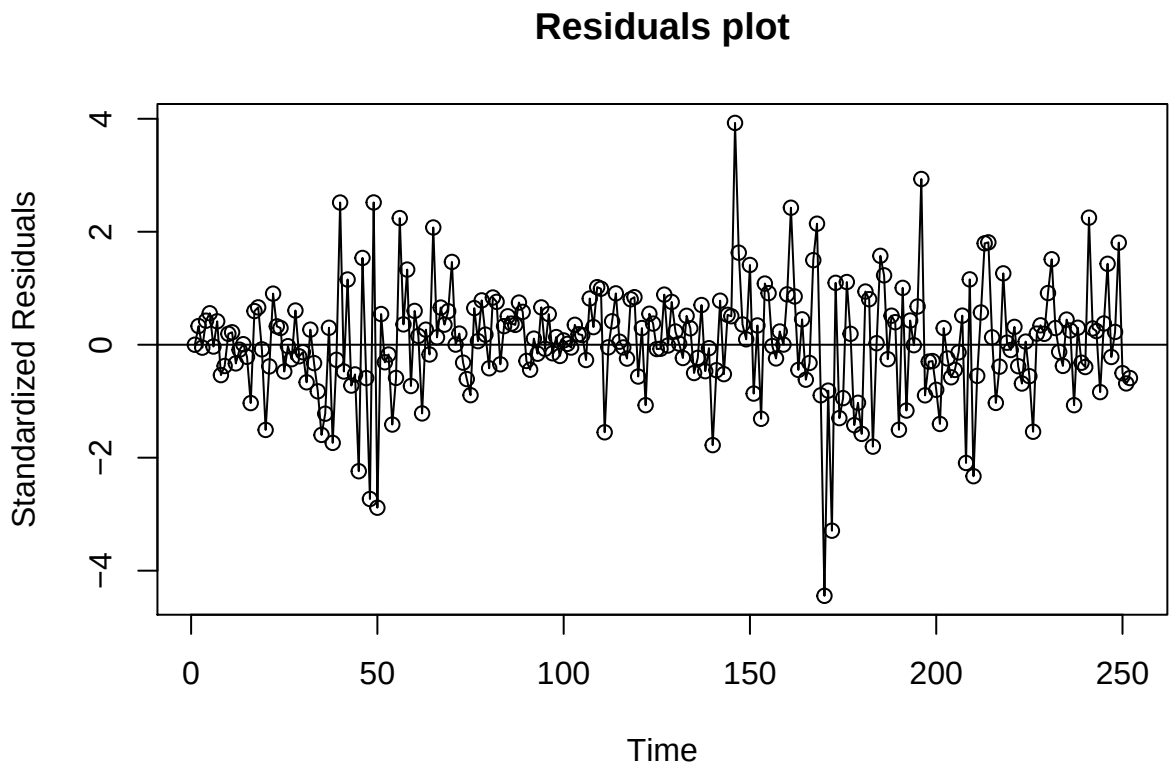
Now, let us plot the residuals to see how they look like.

```
library(TSA)
```

```
## Warning: package 'TSA' was built under R version 4.0.5
```

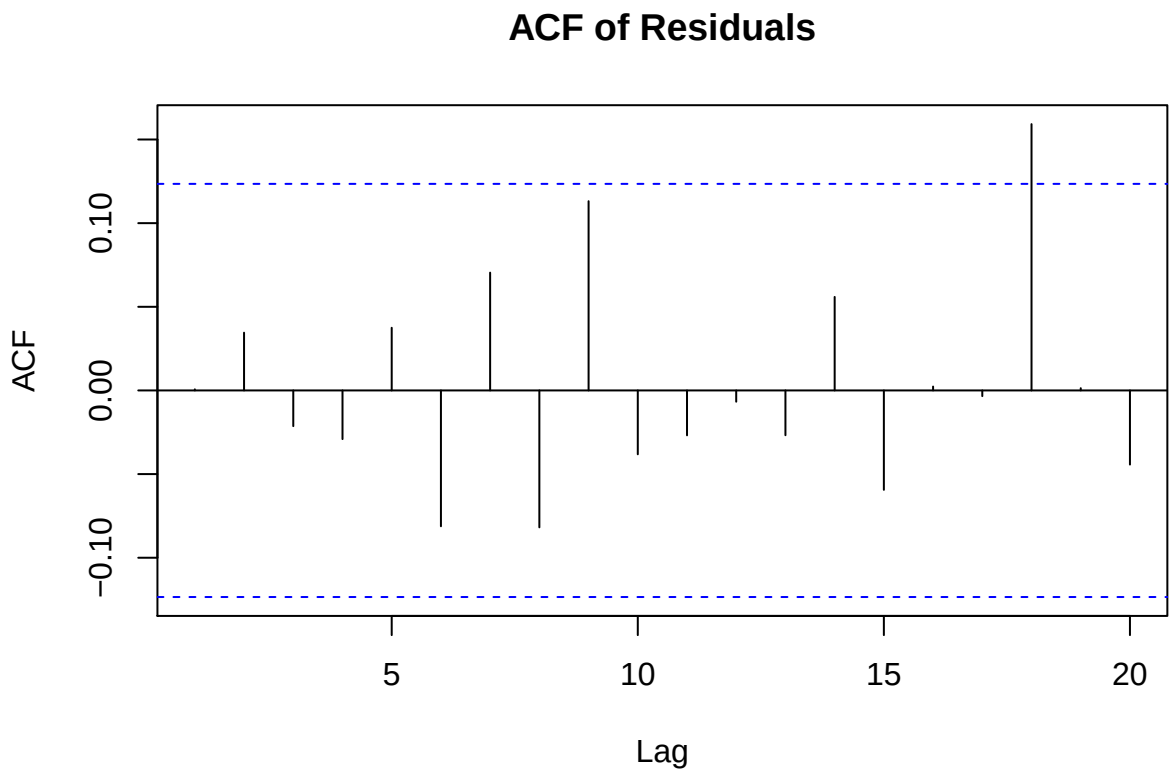
```
## Registered S3 methods overwritten by 'TSA':
##   method       from
##   fitted.Arima forecast
##   plot.Arima   forecast

##
## Attaching package: 'TSA'

## The following objects are masked from 'package:PerformanceAnalytics':
##
##     kurtosis, skewness

## The following objects are masked from 'package:stats':
##
##     acf, arima

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

```
plot(window(rstandard(model)),ylab="Standardized Residuals", type = 'o', main = "Residuals plot");ablin
```



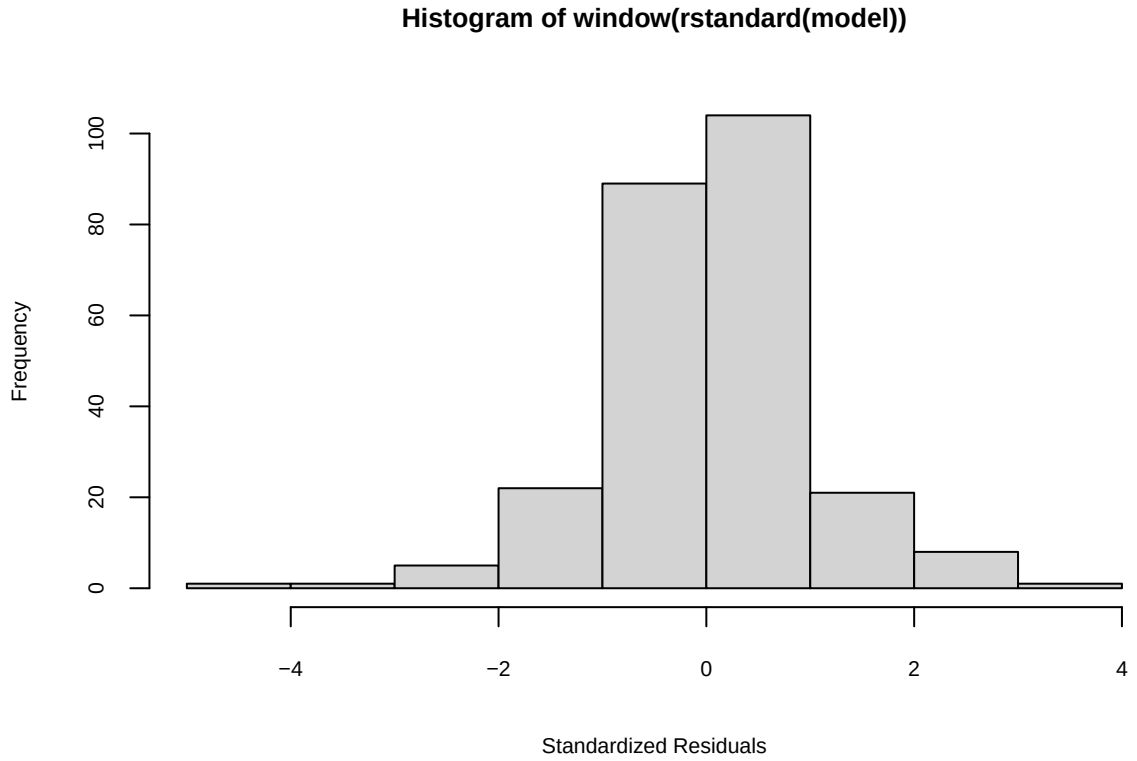
Let us visualize the acf of residuals.

```
acf(as.vector(window(rstandard(model))),lag.max=20, main= "ACF of Residuals")
```

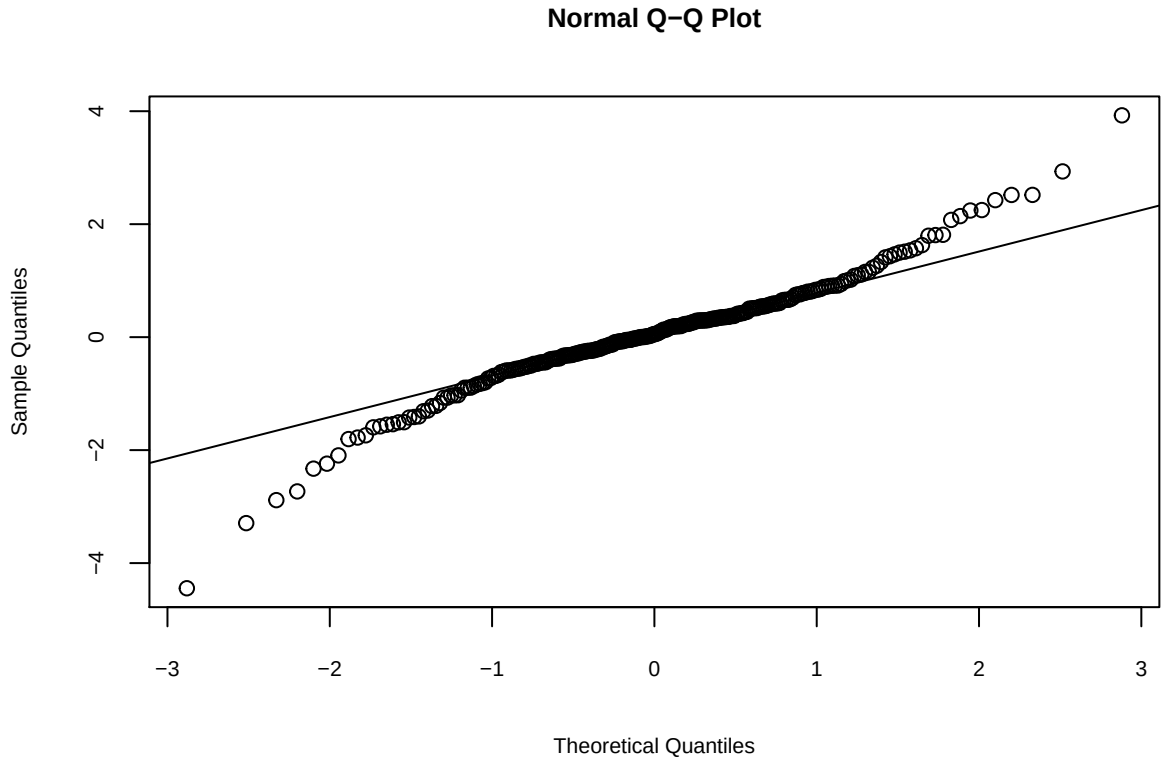


From the plot above, we can see that the only statistically significant correlation is at lag 18 and this correlation. Except for significance at lag 18, the model seems to have captured the essence of the dependence in the series.

Now, let us investigate the question of normality of the error terms through the residuals. Let us see the histogram of the residuals.

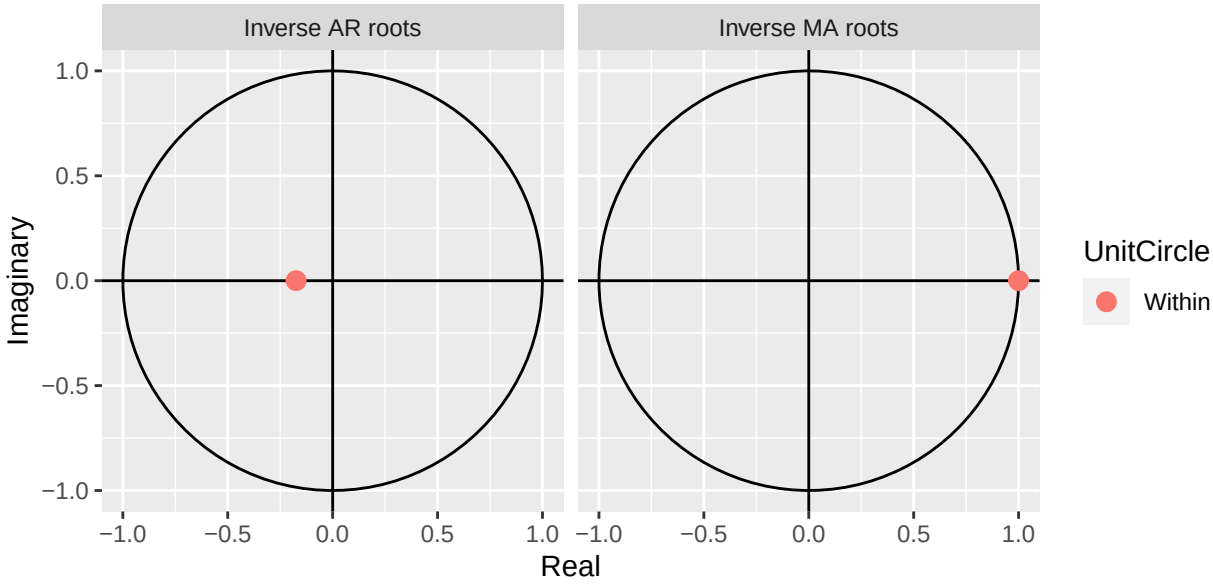


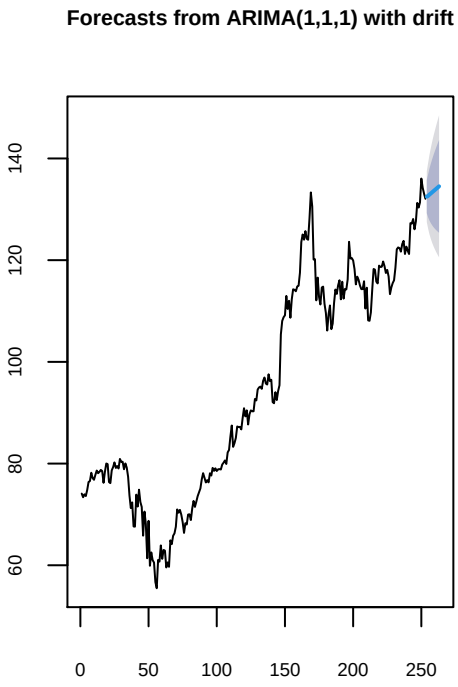
We can see from the plot above that the shape is somewhat bell-shaped but not ideal. Let us visualize the quantile-quantile plot which gives us much better insight than above histogram.

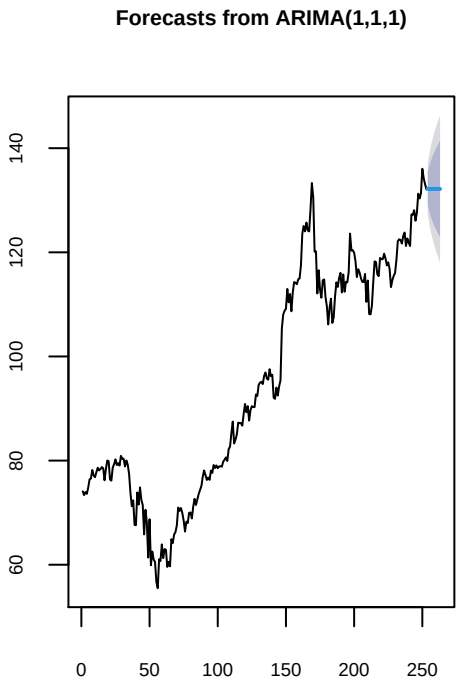


We can see that standardized sample residuals on both tails are larger than they would be for normally distributed data which could be related to the non-constant variance.

Let us forecast and have forecast limits for the aapl returns model.


Inverse AR roots
Inverse MA roots
Imaginary
Real
1.0
0.5
0.0
−0.5
−1.0
UnitCircle
Within


Forecasts from ARIMA(1,1,1) with drift
60
80
100
120
140
0
50
100
150
200
250


Forecasts from ARIMA(1,1,1)
60
80
100
120
140
0
50
100
150
200
250