Seasonal and Nonseasonal GARCH Time Series Analysis

Jaykumar Patel

Department of Mathematical Sciences, Stevens Institute of Technology, Hoboken, NJ

Project Supervisor: Dr. Hadi Safari Katesari

PART A: SEASONAL DATASET: perrin freres monthly champagne sales data

Introduction and Motivation

The motivation for analyzing this dataset is to gain insights into the trends, seasonality, and other patterns in champagne sales over time. This information can be useful for Perrin Freres to forecast future sales, make informed business decisions, and identify areas for improvement in their sales strategies.

In addition, analyzing this dataset can provide valuable insights for other businesses in the food and beverage industry that rely on seasonal sales patterns. By understanding the trends and seasonality in sales, businesses can make better decisions regarding inventory management, pricing strategies, and marketing campaigns.

Overall, the Perrin Freres monthly champagne sales dataset presents an interesting and challenging time series analysis project that can provide valuable insights for businesses and researchers alike.

Data Description

Date Range: From Jan 1964 to sept 1972

Datasource Description: The data is from Kaggle website and can be accessed using the link: https://www.kaggle.com/datasets/anupamshah/perrin-freres-monthly-champagne-sales (https://www.kaggle.com/datasets/anupamshah/perrin-freres-monthly-champagne-sales).

Dataset Description: The dataset contains 105 entries, 2 total columns. One is date and another is sales.

```
library(TSA)
```

```
## Warning: package 'TSA' was built under R version 4.2.2
```

```
##
## Attaching package: 'TSA'
```

```
## The following objects are masked from 'package:stats':
##
##     acf, arima
## The following object is masked from 'package:utils':
##
##     tar
```

```
data <- read.csv("perrin-freres-monthly-champagne.csv")
```

```
summary(data)
```

```
##     Month               sales
##  Length:107         Min.   : 1413
##  Class :character   1st Qu.: 3113
##  Mode  :character   Median : 4217
##                     Mean   : 4761
##                     3rd Qu.: 5221
##                     Max.   :13916
##                     NA's   :2
```

Checking seasonality

```
library(seastests)
```

```
## Warning: package 'seastests' was built under R version 4.2.3
```

```
isSeasonal(data$sales, test = "combined", freq = 12)
```

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

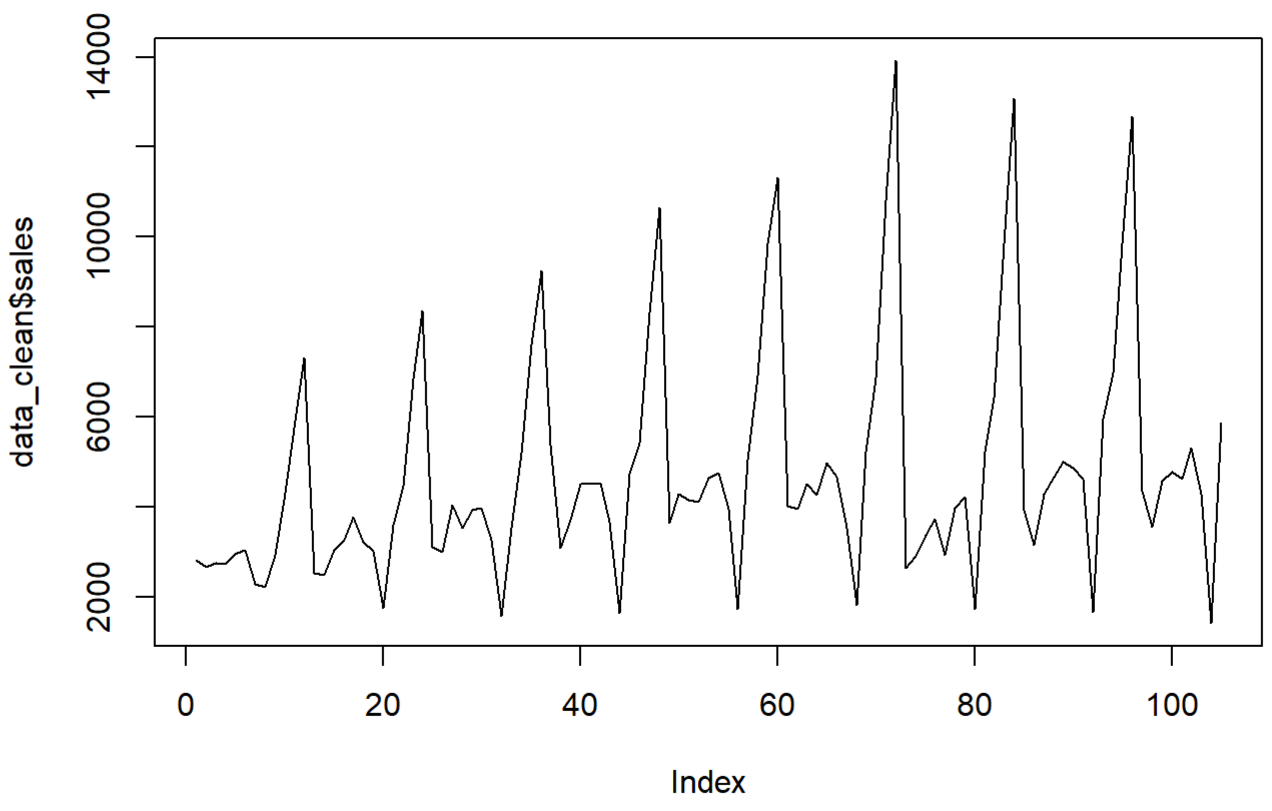
```
## Registered S3 methods overwritten by 'forecast':
##   method       from
##   fitted.Arima TSA
##   plot.Arima   TSA
```

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

Data Pre-processing

Before any further preprocessing we want to remove the null values.

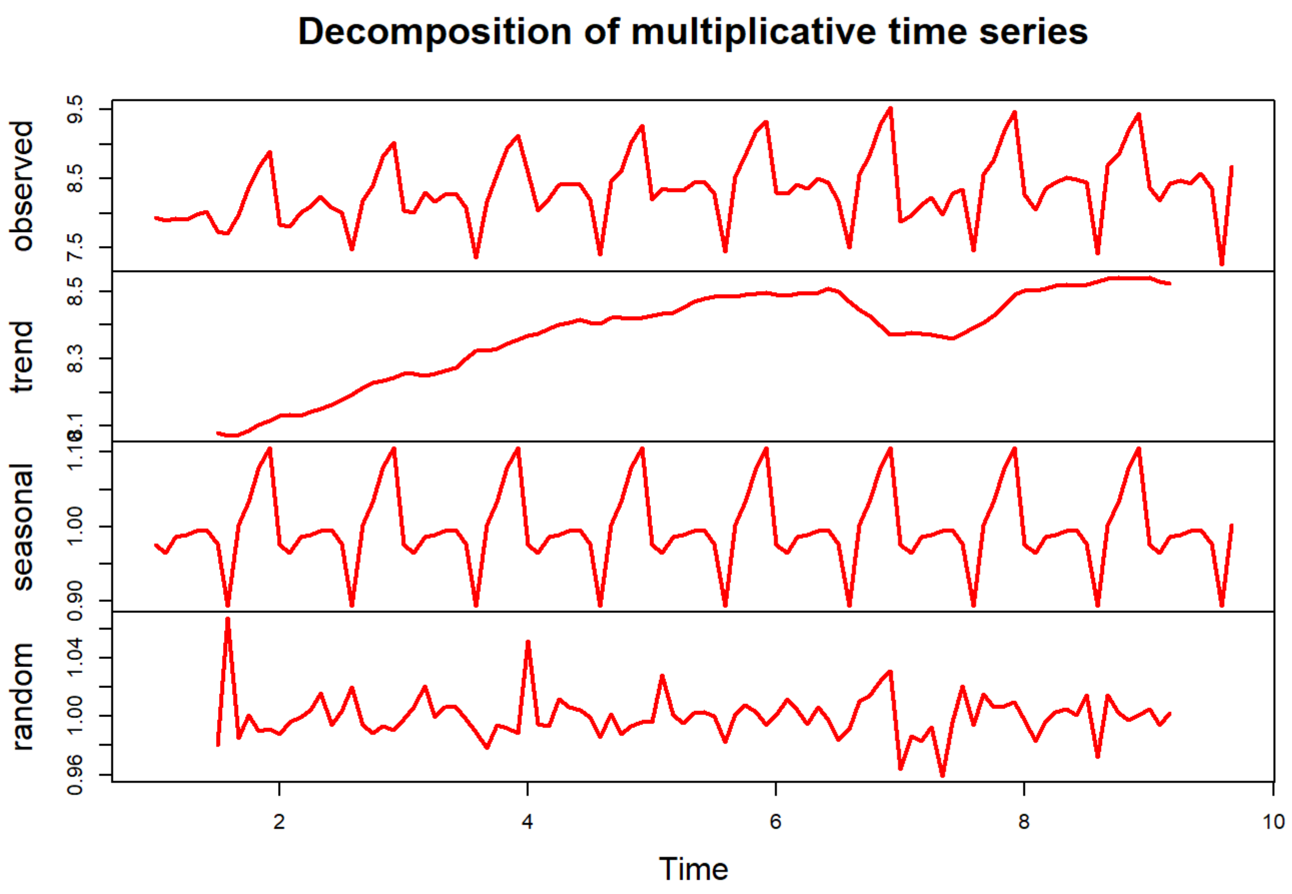
```
data_clean <- na.omit(data)
plot(data_clean$sales, type='l')
```



Decomposing the time series

Decomposing the time series to have a look at the seasonal components, trend components, and residuals in it.

```
ts_sales = ts(log(data_clean$sales), frequency = 12)
decompose_sales = decompose(ts_sales, "multiplicative")
plot(decompose_sales, type='l',lwd=2, col = 'red')
```



The series exhibits multiplicative decomposition. As the amplitude of both the seasonal and irregular variations increase as the level of the trend rises. In the multiplicative model, the original time series is expressed as the product of trend, seasonal and irregular components.

Stationarity check

Let's start by checking if the time series is stationary or not. To do so we are going to use the Dickey Fuller and/or augmented Dickey fuller test

```
library(aTSA)
```

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

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

```
adf.test(data_clean$sales)
```

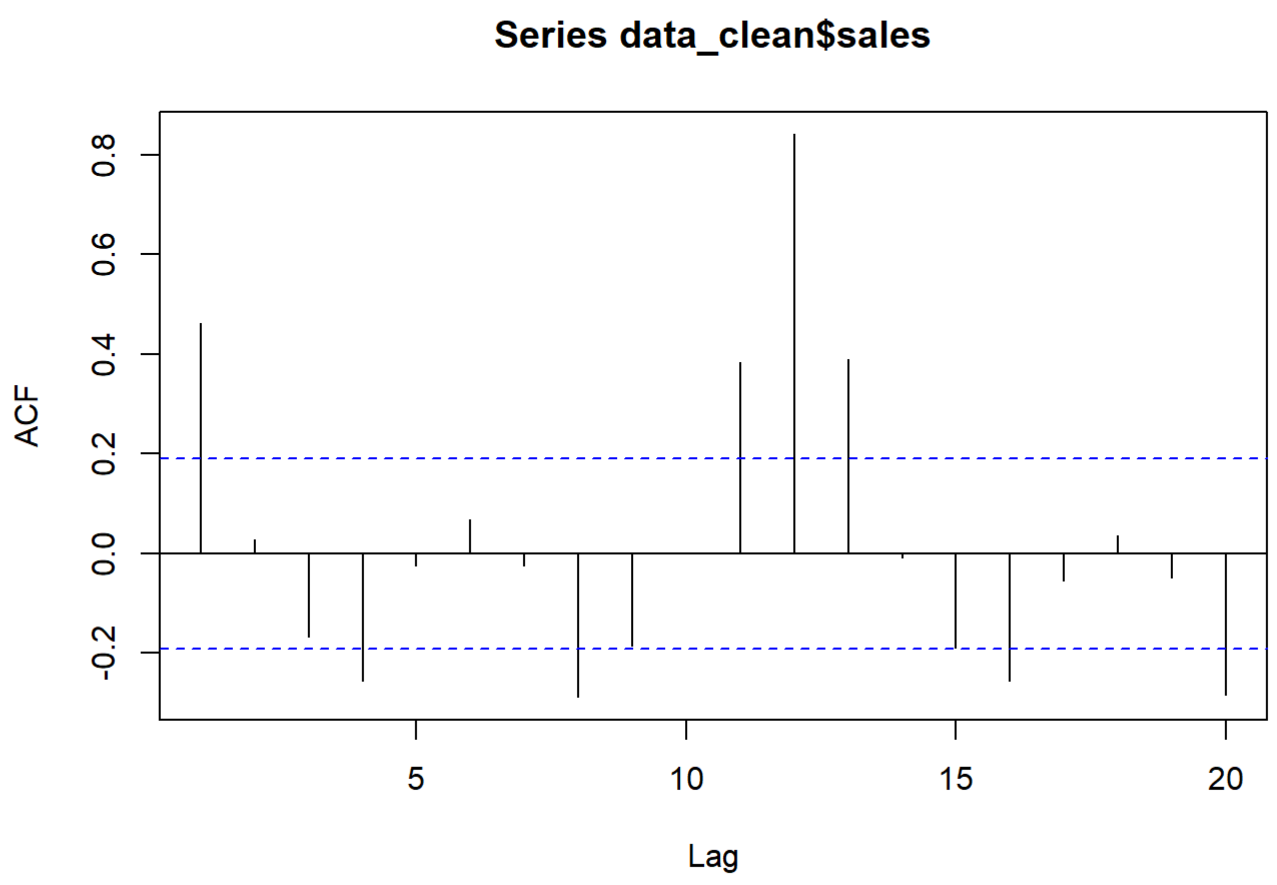
```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag    ADF p.value
## [1,]   0 -2.461  0.0158
## [2,]   1 -2.280  0.0233
## [3,]   2 -1.769  0.0767
## [4,]   3 -1.490  0.1435
## [5,]   4 -0.863  0.3691
## Type 2: with drift no trend
##      lag   ADF p.value
## [1,]   0 -6.13    0.01
## [2,]   1 -6.69    0.01
## [3,]   2 -6.09    0.01
## [4,]   3 -6.09    0.01
## [5,]   4 -4.25    0.01
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -6.39    0.01
## [2,]   1 -7.15    0.01
## [3,]   2 -6.69    0.01
## [4,]   3 -6.96    0.01
## [5,]   4 -4.89    0.01
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

The augmented Dickey Fuller test demonstrates that the P values are not less than.05. Therefore, the series is not stationary, and the null hypothesis must be rejected. In other words, the variance is not constant over time and the time series has some sort of time dependent structure.

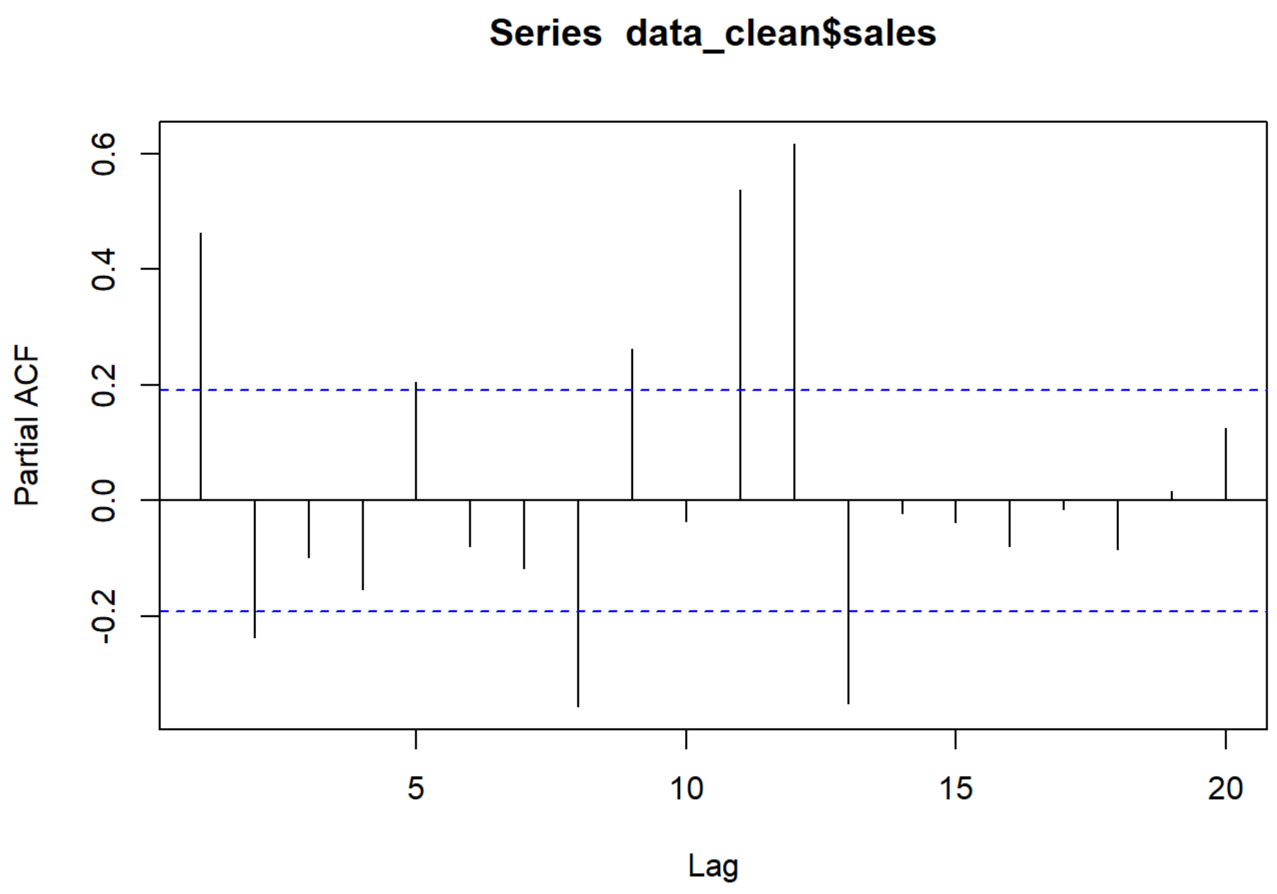
Before fitting a model to the series, it is crucial to make it stationary because we only ever see one instance of a stochastic process, as opposed to many instances. So, in order for watching a lengthy run of a stochastic process to be comparable to observing numerous independent runs of a stochastic process, stationarity and ergodicity are required.

Plotting the ACF and PCF of the series.

```
acf(data_clean$sales)
```



```
pacf(data_clean$sales)
```



The data’s autocorrelation plot shows relatively little deterioration. This supports the finding that the time series is not stationary from the Dickey-Fuller test. A strong autocorrelation is apparent from the trend in the Data, which is visible.

MAKING THE TIME SERIES STATIONARY

Using the Box Cox transformation:

A parameter lambda is used to index the Box-Cox transformation family of power transformations. Anytime we have a non-stationary time series (with non-constant variance), we can utilize this transformation. When Box-Cox is used with a specific lambda value, the process could become stationary.

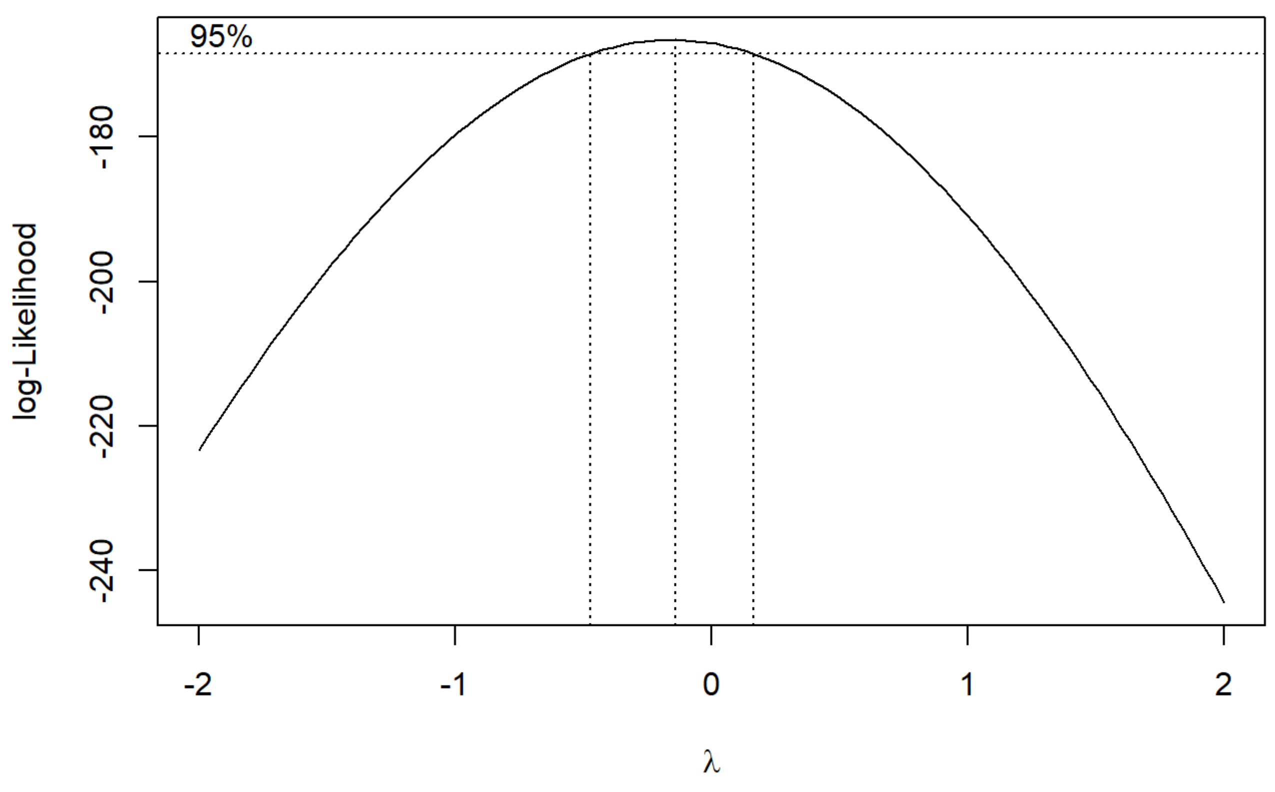
```
library(MASS)
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 4.2.3
```

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

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

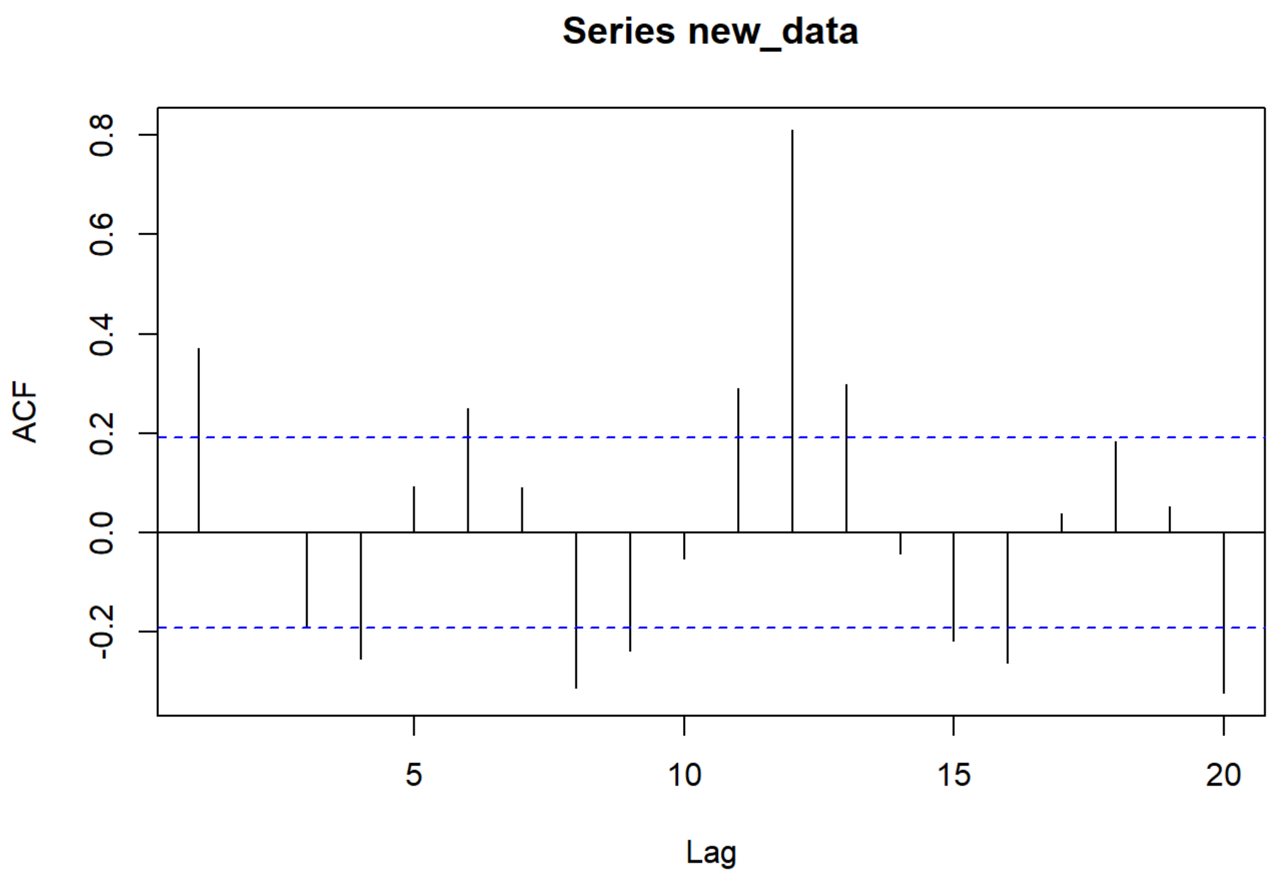
```
b <- boxcox(lm(data_clean$sales ~ 1))
```



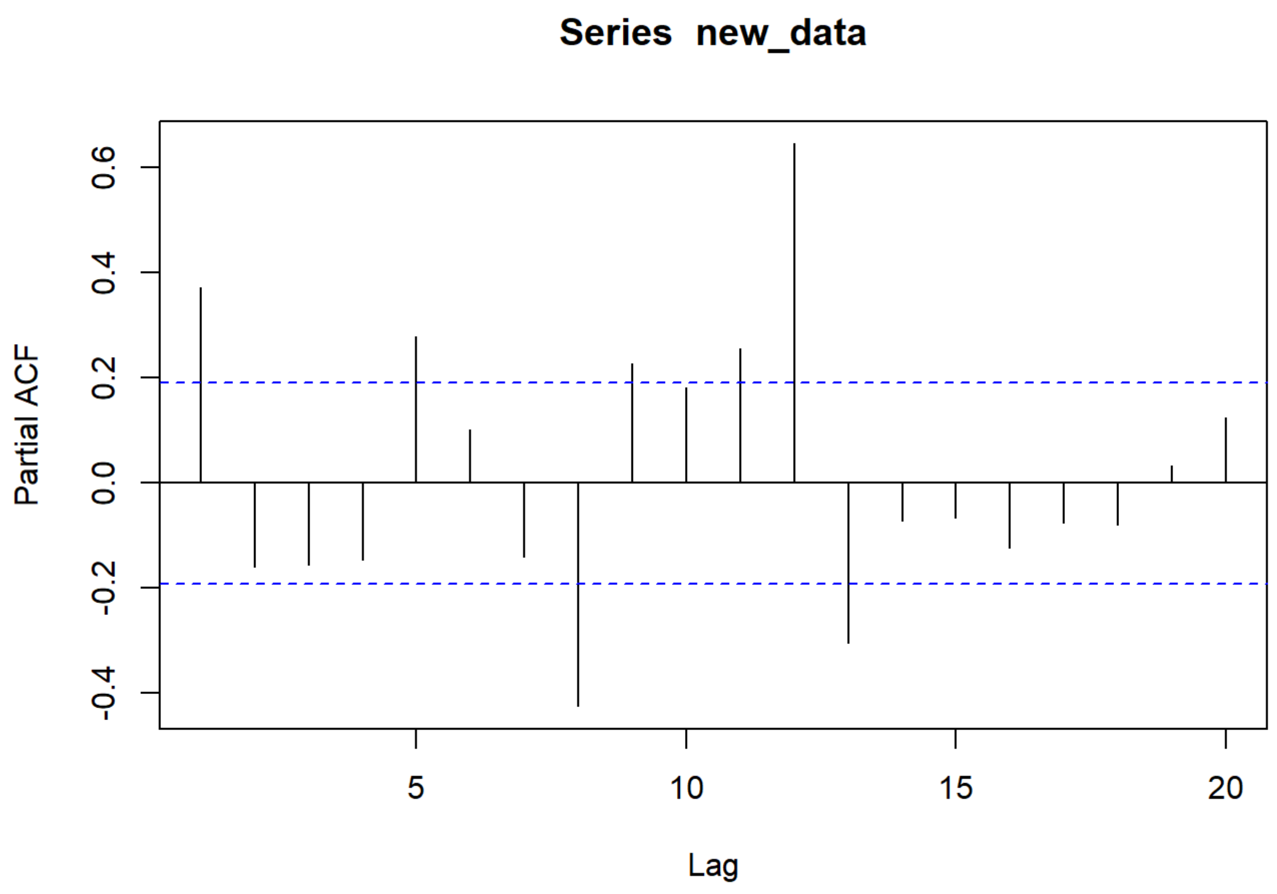
```
lambda <- b$x[which.max(b$y)]
lambda
```

```
## [1] -0.1414141
```

```
new_data <- (data_clean$sales ^ lambda - 1) / lambda
acf(new_data)
```



```
pacf(new_data)
```



```
adf.test(new_data)
```

```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag     ADF p.value
## [1,]   0 -0.0353   0.633
## [2,]   1 -0.0208   0.637
## [3,]   2  0.0171   0.648
## [4,]   3  0.0652   0.662
## [5,]   4  0.1845   0.696
## Type 2: with drift no trend
##      lag   ADF p.value
## [1,]   0 -6.83    0.01
## [2,]   1 -6.72    0.01
## [3,]   2 -6.49    0.01
## [4,]   3 -6.34    0.01
## [5,]   4 -4.04    0.01
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -7.14    0.01
## [2,]   1 -7.21    0.01
## [3,]   2 -7.20    0.01
## [4,]   3 -7.39    0.01
## [5,]   4 -4.66    0.01
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

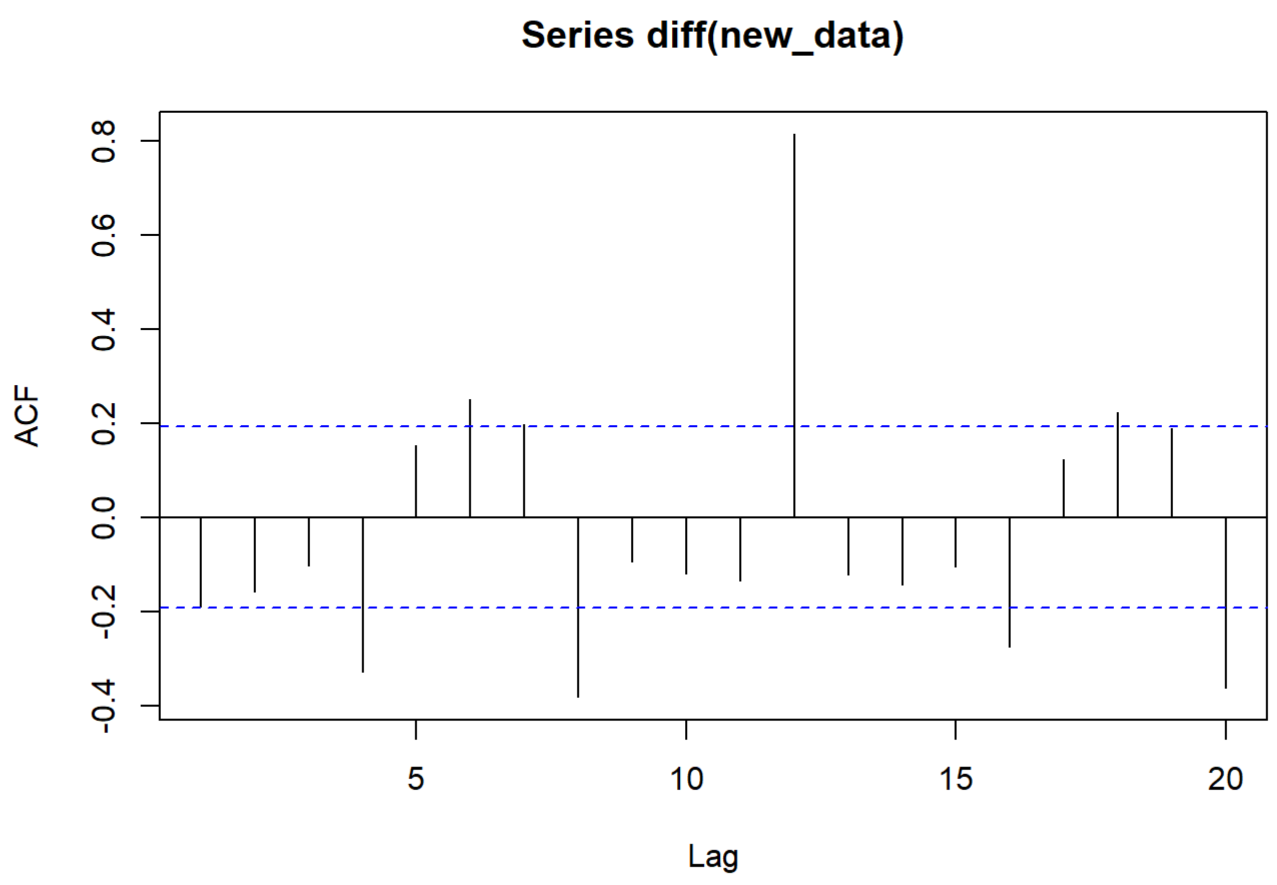
We can observe that the series is still moving even after the Box Cox change. The series is not stationary since the P value is bigger than 0.5. Additionally, the acf and pacf plots demonstrate that the time series still exhibits autocorrelation. Let’s do a seasonal differentiation of the time series to eliminate this association and make the series stationary.

SEASONAL DIFFERENCING

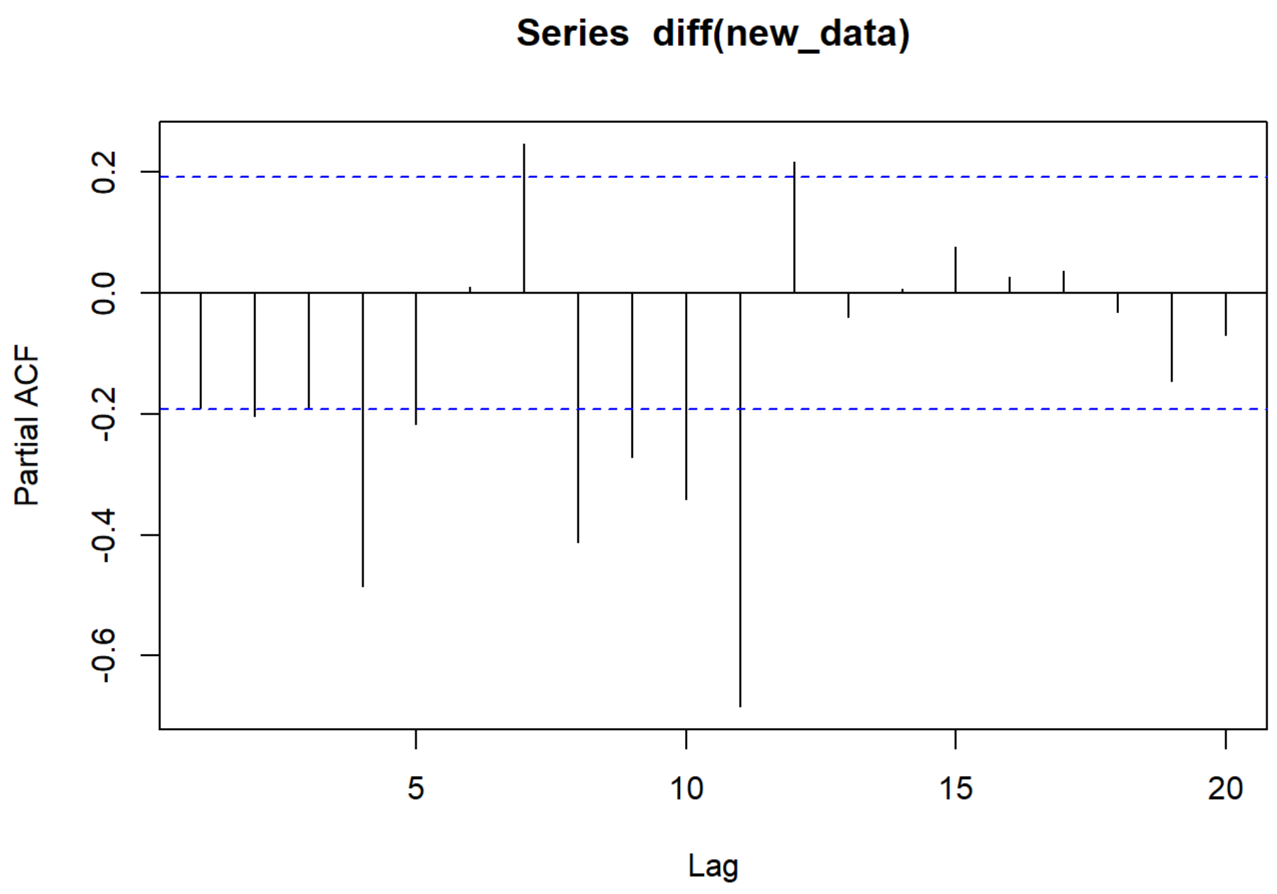
```
diff_ser <- diff(new_data)
adf.test(diff(new_data,lag = 12))
```

```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag   ADF p.value
## [1,]   0 -6.98    0.01
## [2,]   1 -4.50    0.01
## [3,]   2 -3.24    0.01
## [4,]   3 -2.71    0.01
## Type 2: with drift no trend
##      lag   ADF p.value
## [1,]   0 -7.52  0.0100
## [2,]   1 -4.97  0.0100
## [3,]   2 -3.58  0.0100
## [4,]   3 -2.96  0.0448
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -7.77  0.0100
## [2,]   1 -5.26  0.0100
## [3,]   2 -3.86  0.0195
## [4,]   3 -3.19  0.0943
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

```
acf(diff(new_data))
```



```
pacf(diff(new_data))
```



```
eacf(diff(new_data))
```

```
## AR/MA
##   0 1 2 3 4 5 6 7 8 9 10 11 12 13
## 0 o o o x o x o x o o o  x  o  o
## 1 x o o x o o o x o o o  x  x  o
## 2 x o o x o o x x o o o  x  x  o
## 3 x x o x o o o x o o o  x  o  x
## 4 x o x x x o o x o o o  x  x  o
## 5 o o o x x o o x o o o  x  o  o
## 6 o x o x o o o x o o o  x  o  x
## 7 x x o x x x x x o o o  x  x  o
```

After comparing the series, we can observe that the P value from the enhanced Dickey-Fuller test is less than 0.05, which indicates that it is statistically significant. As a result, we can say that the series is stationary at this point.Additionally, we can see that the correlations have greatly decreased in the ACF and PCF lots.

Now that we have a stationary series, we'll have a look at the EACF plots to finalize the order of our AR, MA models and then fit the model.

DETERMINING THE ORDER OF THE MODEL

The best model to fit the data can have p, q, P, and Q in the range of 0 to 3, according to the ACF, PACF, and EACF. Based on the lowest AIC for P, Q, p, and q values, we will choose the best model.

I've created a nested for loop that goes through each conceivable combination of P, Q, p, and q values in the range of 0 to 3 and fits a SARIMA model for each of them. The AICs for each of these models are then listed along with the matching P, Q, p, and q values. The top value in the list is then popped once the list has been sorted in ascending order. The P, Q, p, and q values that correspond to the lowest AIC model are represented by this value.

NESTED FOR LOOP FOR DERTMING THE VALUES OF P, Q, p and q

```r
# Load the forecast package
library(forecast)

# Define the range of values for p, q, P, and Q
p_values <- c(0, 1, 3)
q_values <- c(0, 1, 3)
P_values <- c(0, 1, 3)
Q_values <- c(0, 1, 3)

# Initialize variables for storing the best model and its performance
best_model <- NULL
best_aic <- Inf

# Nested for loops to iterate over different parameter values
for (p in p_values) {
  for (q in q_values) {
    for (P in P_values) {
      for (Q in Q_values) {

        # Fit a seasonal ARIMA model with the current parameter values
        fit <- arima(data_clean$sales, order=c(p,1,q), seasonal=c(P,1,Q), method="ML")

        # Evaluate the model performance using AIC
        current_aic <- AIC(fit)

        # Update the best model and its performance if the current model is better
        if (current_aic < best_aic) {
          best_model <- fit
          best_aic <- current_aic
          best_params <- c(p, q, P, Q)
        }
      }
    }
  }
}
```

```
## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced
```

```
# Print the best SARIMA model parameters and AIC
print(paste("Best SARIMA model parameters:", paste(best_params, collapse=",")))
```

```
## [1] "Best SARIMA model parameters: 3,3,1,3"
```

```
print(paste("Best AIC:", best_aic))
```

```
## [1] "Best AIC: 1869.95463051615"
```

Parameter Estimation using best model

```
(fit <- arima(data_clean$sales, order = c(3,1,3)))
```

```
##
## Call:
## arima(x = data_clean$sales, order = c(3, 1, 3))
##
## Coefficients:
##          ar1     ar2      ar3      ma1      ma2     ma3
##       0.5630  0.3081  -0.3571  -1.0642  -0.7230  0.8300
## s.e.  0.1231  0.1512   0.1190   0.1015   0.1234  0.0828
##
## sigma^2 estimated as 3791574:  log likelihood = -938.05,  aic = 1888.09
```

```
(fit2 <- arima(data_clean$sales, order = c(3,1,5)))
```

```
##
## Call:
## arima(x = data_clean$sales, order = c(3, 1, 5))
##
## Coefficients:
##          ar1      ar2     ar3      ma1      ma2      ma3      ma4     ma5
##       0.2645  -0.3000  0.3042  -0.7903  -0.1345  -0.4724  -0.2901  0.7649
## s.e.  0.1465   0.1879  0.1207   0.1223   0.1956   0.1150   0.1171  0.0930
##
## sigma^2 estimated as 3341545:  log likelihood = -933.38,  aic = 1882.77
```

As per the aic I got the best model (3,1,3) but due to showing dependancies in Ljung-box test I decided to go with Arima(3,1,5) which is suggested by eacf and it works better.

```
# Load the "forecast" package
library(forecast)

# Fit an ARIMA model to the "sales" dataset
arima_model <- Arima(data_clean$sales, order = c(3,1,5))

# Make a seasonal ARIMA (SARIMA) model from the ARIMA model
sarima_model <- Arima(data_clean$sales, order = c(3,1,5), seasonal = list(order = c(1,1,3), p
eriod = 12))
# Print the model summaries
summary(arima_model)
```

```
## Series: data_clean$sales
## ARIMA(3,1,5)
##
## Coefficients:
##          ar1      ar2     ar3      ma1      ma2      ma3      ma4     ma5
##       0.2645  -0.3000  0.3042  -0.7903  -0.1345  -0.4724  -0.2901  0.7649
## s.e.  0.1465   0.1879  0.1207   0.1223   0.1956   0.1150   0.1171  0.0930
##
## sigma^2 = 3620008:  log likelihood = -933.38
## AIC=1884.77   AICc=1886.68   BIC=1908.57
##
## Training set error measures:
##                    ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 255.4961 1819.264 1399.104 -9.843906 37.67181 0.8200434
##                      ACF1
## Training set -0.008447569
```

```
summary(sarima_model)
```

```
## Series: data_clean$sales
## ARIMA(3,1,5)(1,1,3)[12]
##
## Coefficients:
##          ar1      ar2      ar3      ma1     ma2     ma3      ma4      ma5
##       0.0026  -0.4307  -0.4428  -0.7208  0.3613  0.1956  -0.5210  -0.2153
## s.e.  2.0069   1.2372   1.5789   1.9995  2.6889  2.3434   0.9432   0.5913
##          sar1     sma1    sma2     sma3
##       -0.2411  -0.0472  0.0431  -0.1117
## s.e.   1.2932   1.3002  0.4442   0.2185
##
## sigma^2 = 507648:  log likelihood = -732.35
## AIC=1490.69   AICc=1495.36   BIC=1523.47
##
## Training set error measures:
##                     ME    RMSE      MAE       MPE     MAPE      MASE
## Training set -20.90154 621.916 429.2668 -3.348272 11.02379 0.2516019
##                     ACF1
## Training set -0.01147262
```

These summaries provide information about the model coefficients, standard errors, and other statistics. By examining these summaries, we can assess the goodness of fit of the models and evaluate their forecasting performance.

Residual Analysis

```
# Load the "forecast" package
library(forecast)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

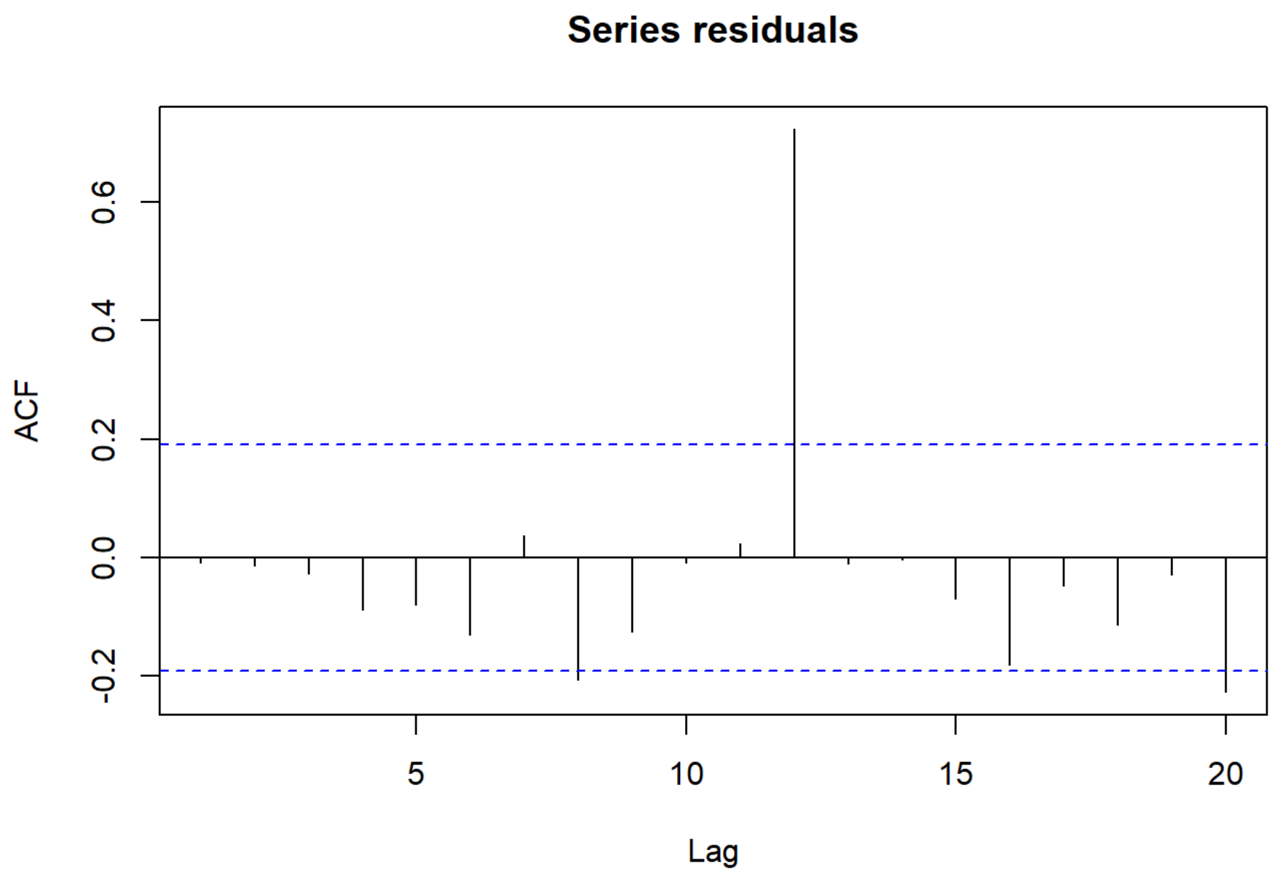
```
library(car)
```

```
## Warning: package 'car' was built under R version 4.2.3
```

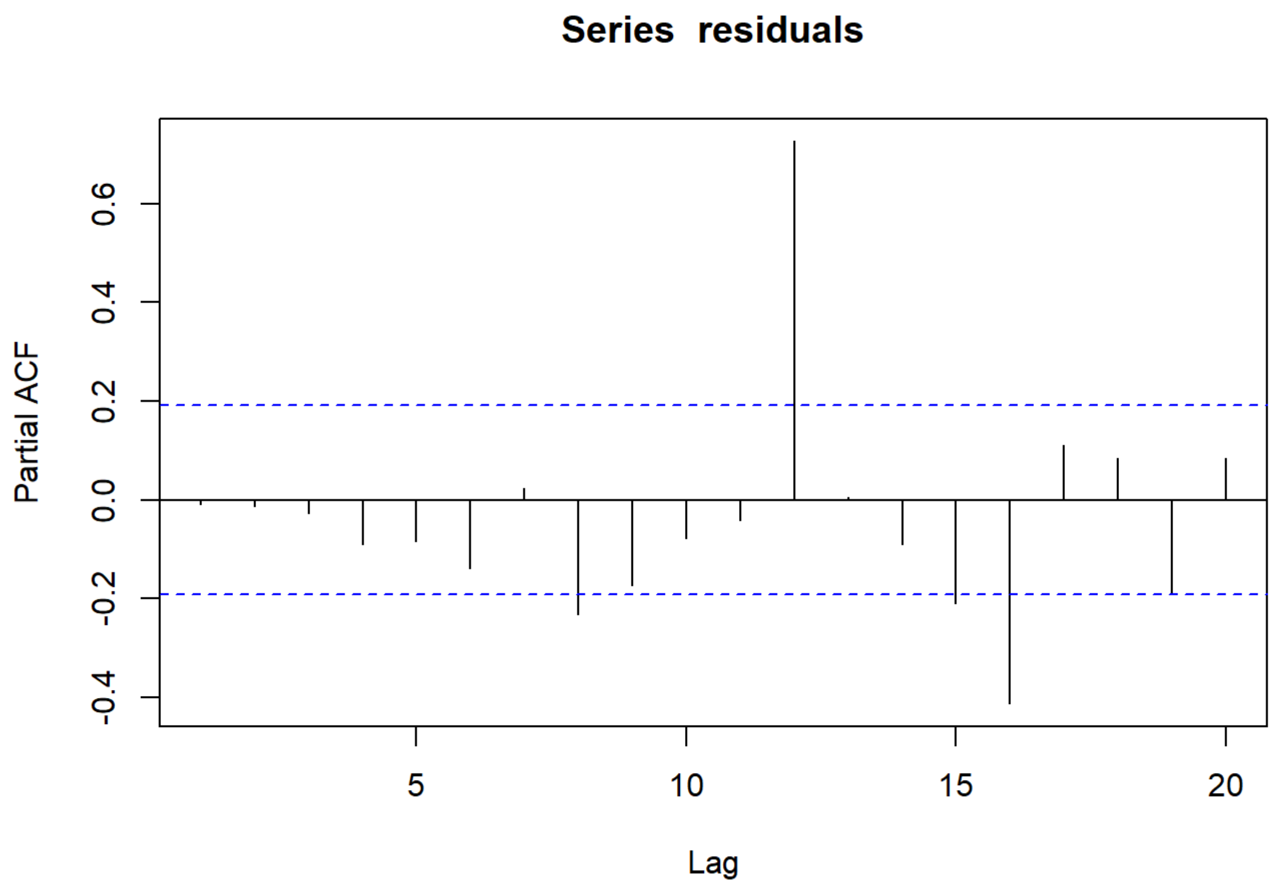
```
## Loading required package: carData
```

```
## Warning: package 'carData' was built under R version 4.2.3
```

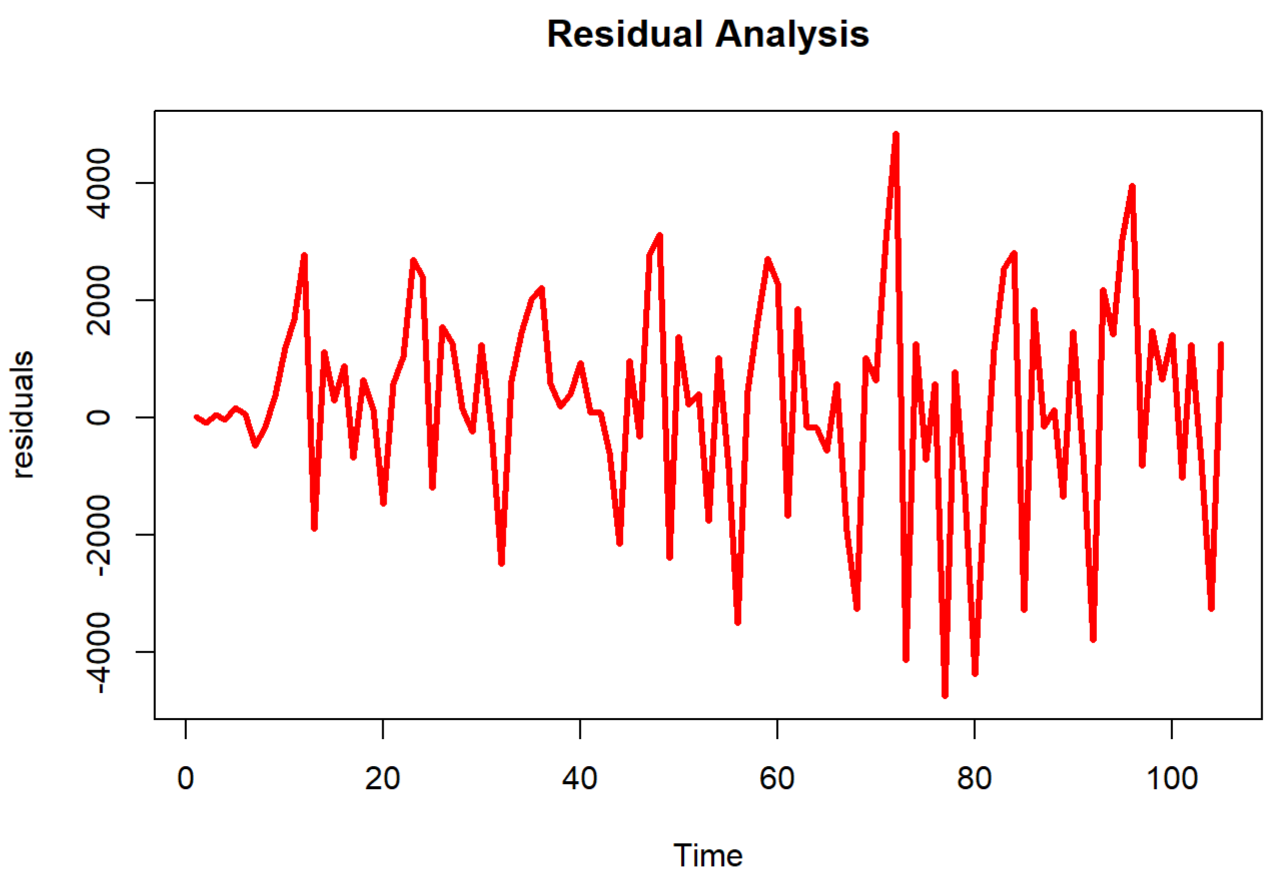
```
# Fit an ARIMA(3,1,3) model to the time series
arima_model <- Arima(data_clean$sales, order = c(3,1,5))

# Extract the residuals from the ARIMA model
residuals <- residuals(arima_model)

# Plot the ACF and PACF of the residuals
acf(residuals)
```



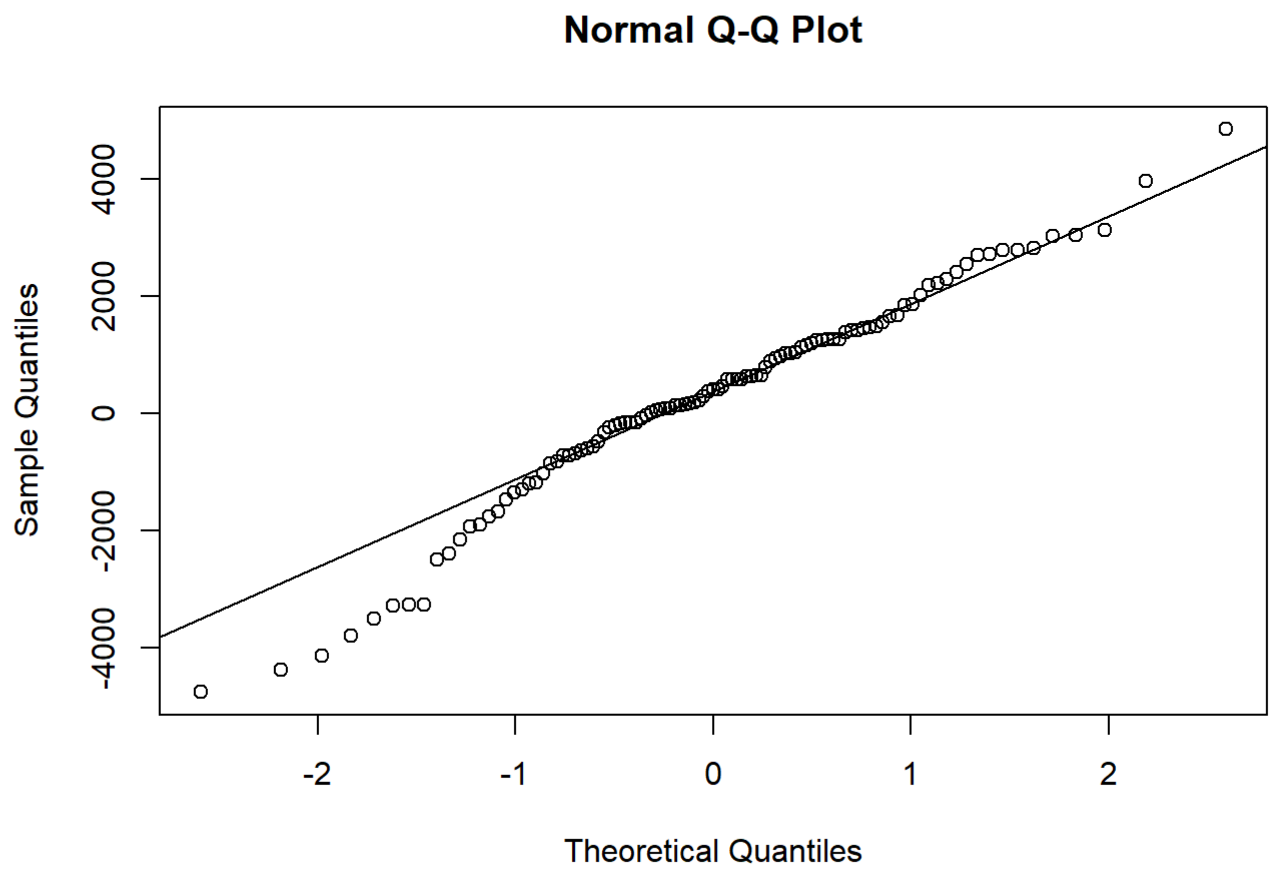
```
pacf(residuals)
```



```
#plot time series data
ts.plot(residuals,lwd=3,col="red",main='Residual Analysis')
```

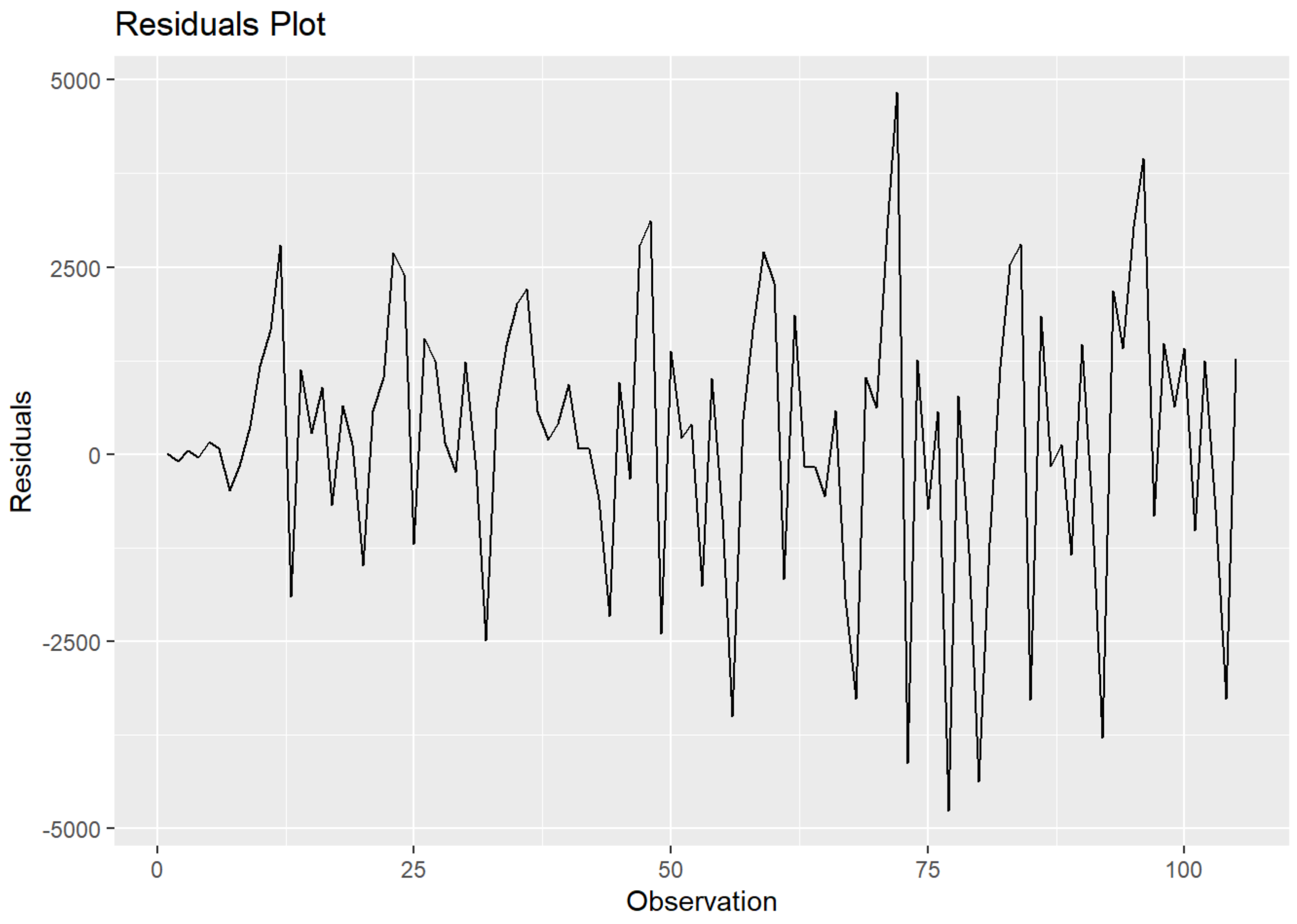


```
qqnorm(residuals)
qqline(residuals)
```

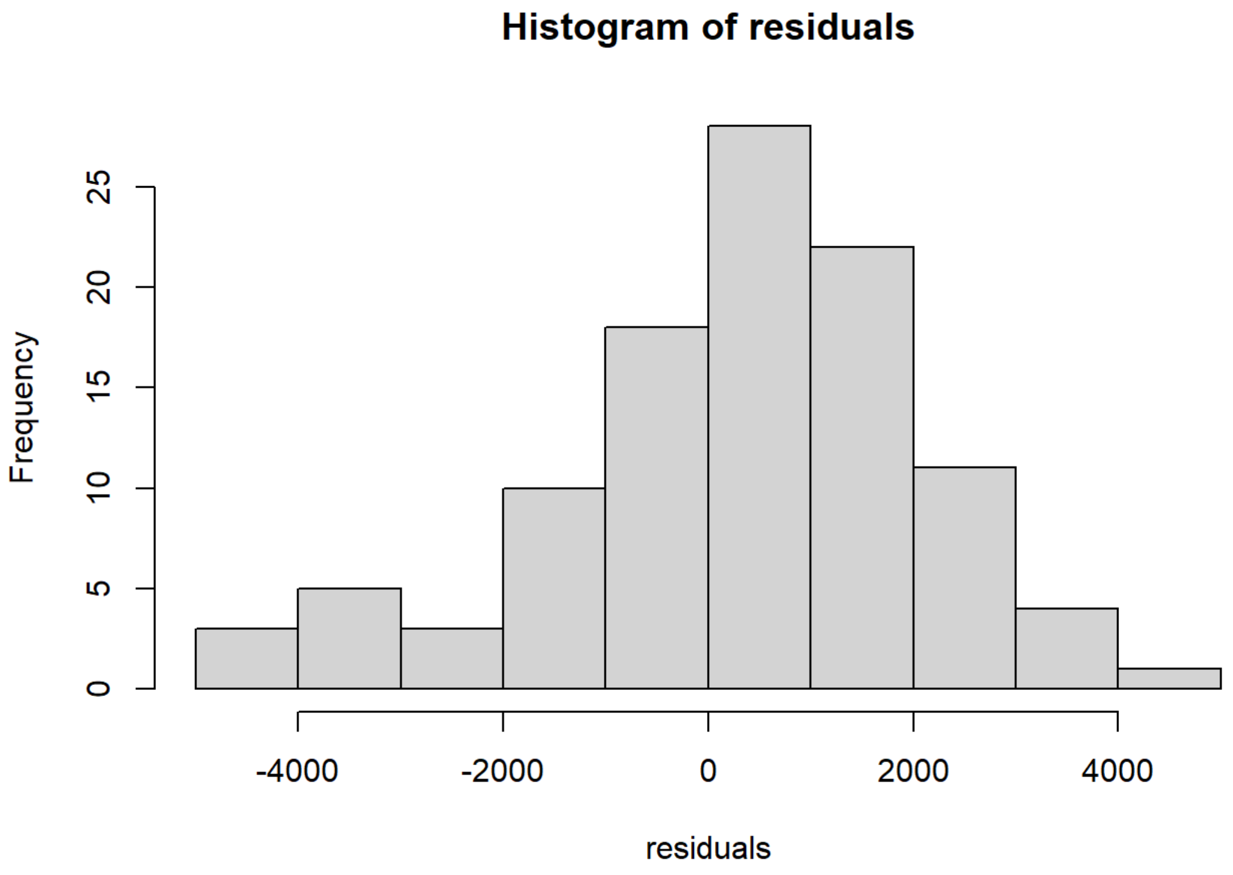


```
ggplot(data.frame(residuals = residuals), aes(x = 1:length(residuals), y = residuals)) +
  geom_line() +
  labs(x = "Observation", y = "Residuals", title = "Residuals Plot")
```

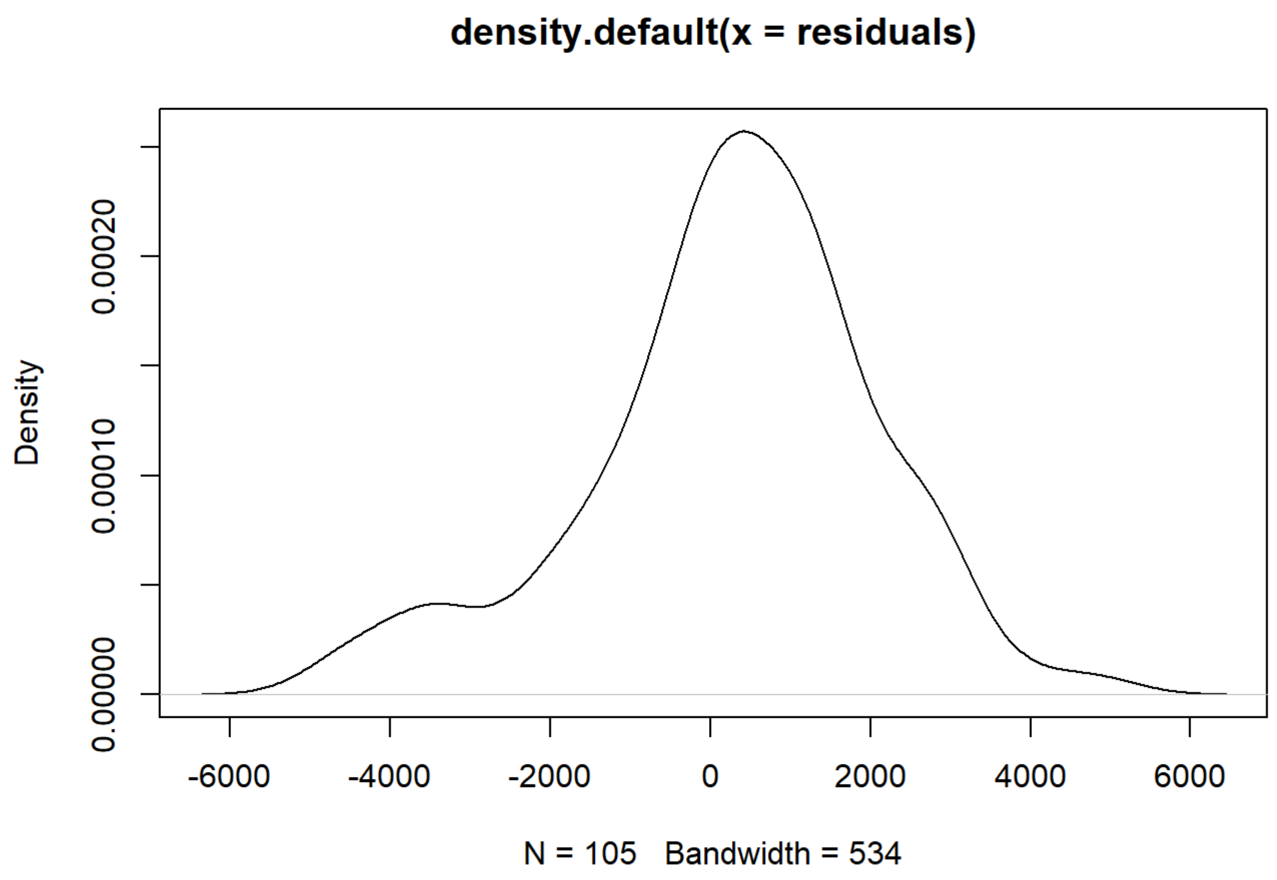
```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```



```
# Plot the histogram and density of the residuals
hist(residuals)
```



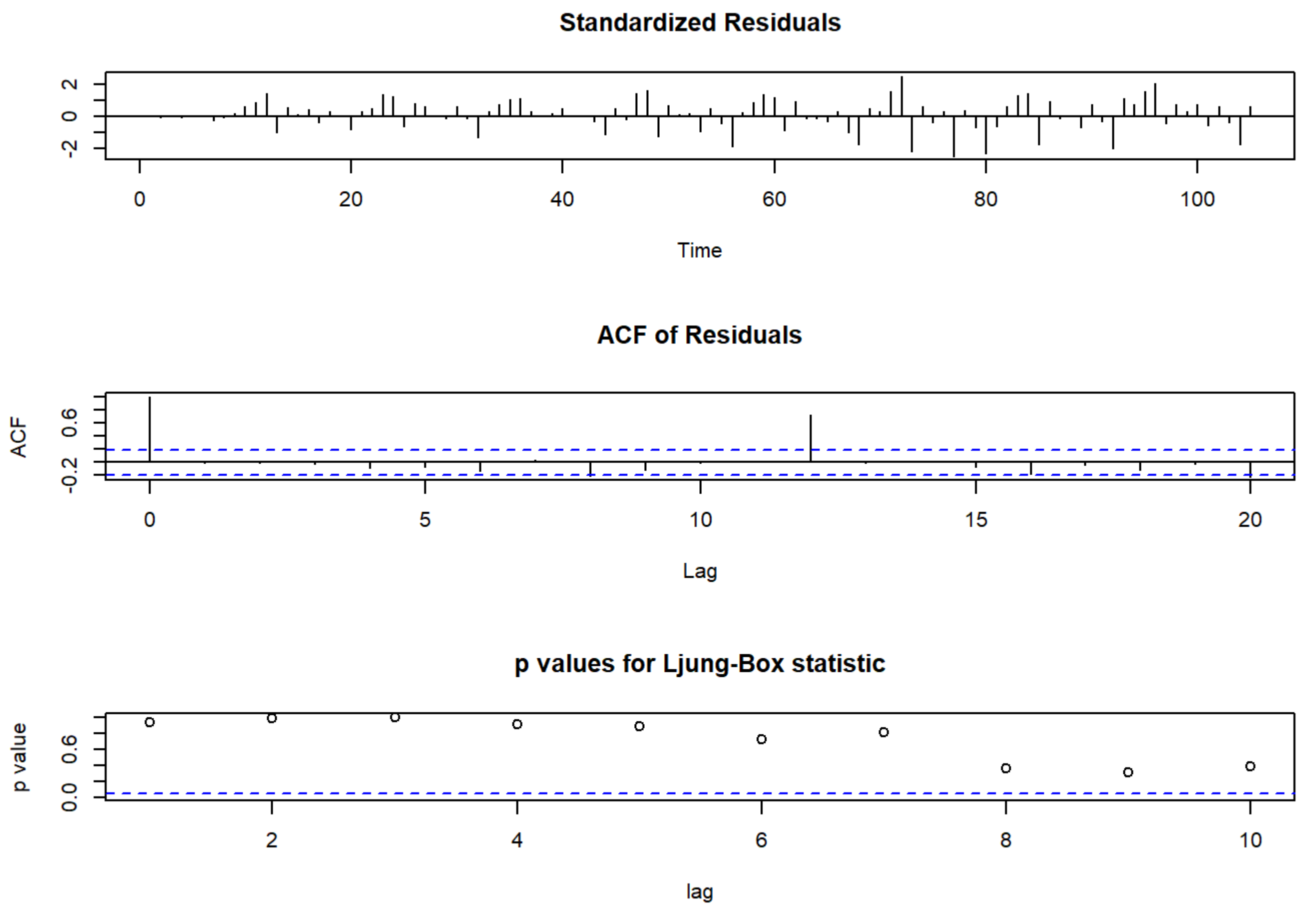
```
plot(density(residuals))
```



```
# Perform a Ljung-Box test on the residuals
Box.test(residuals, lag = 10, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals
## X-squared = 10.583, df = 10, p-value = 0.3909
```

```
#LBQPlot(residuals, lag.max = 20, SquaredQ = FALSE)
tsdiag(arima_model)
```

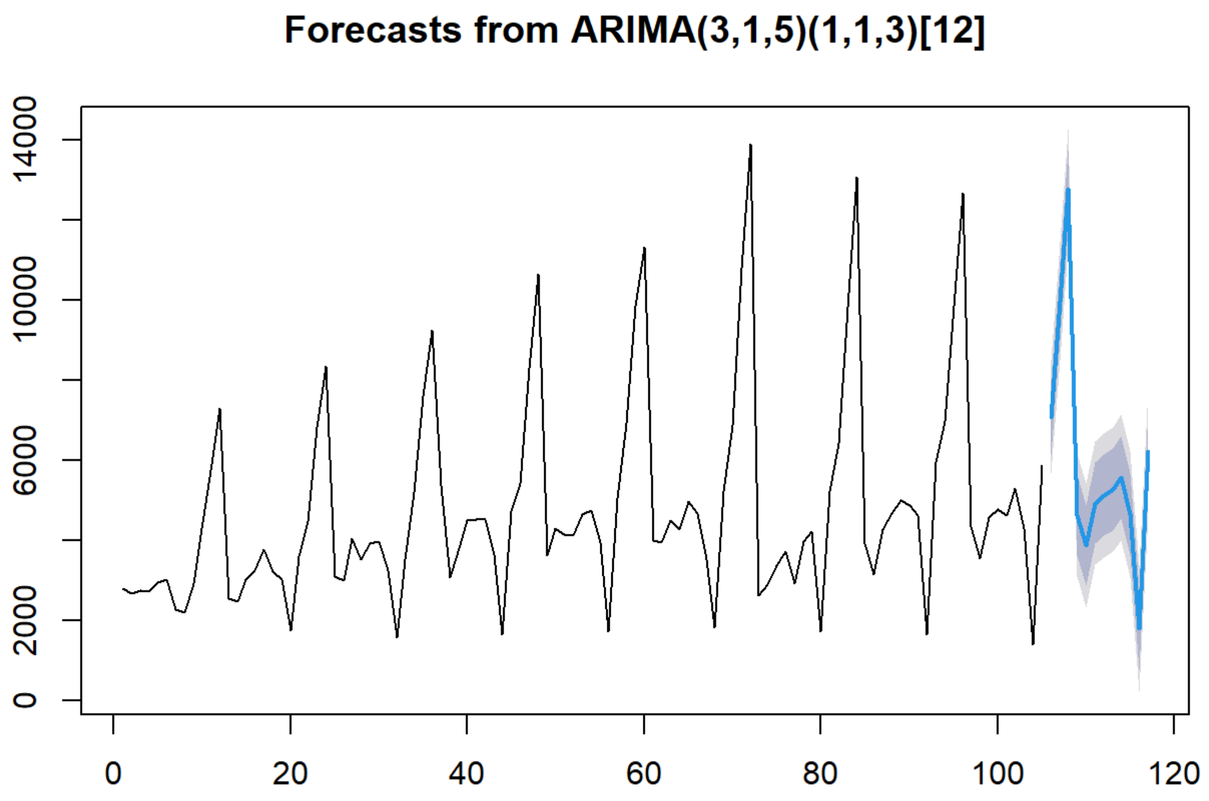


the code performs a Ljung-Box test on the residuals using the “Box.test” function to formally test for residual autocorrelation. The test statistic is compared to a chi-squared distribution with degrees of freedom equal to the number of lags specified in the test. A significant p-value (i.e., less than 0.05) indicates evidence of residual autocorrelation, which suggests that the model may be misspecified and may require further modification.

In this case, I got a p-value more than 0.05 in the Ljung-Box test. It suggests that this can be a good model.

Forecast Using the best model

```
# Load the "forecast" package
library(forecast)

# Fit a SARIMA(3,1,3)(1,1,3) model to the time series
sarima_model <- Arima(data_clean$sales, order = c(3,1,5), seasonal = list(order = c(1,1,3), p
eriod = 12))

# Generate a 12-month forecast from the SARIMA model
forecast_data <- forecast(sarima_model, h =12)

# Plot the forecasted values
plot(forecast_data)
```



Forecast looks preety much good and now I am moving on non-seasonal dataset.

PART B: NON-SEASONAL DATASET: Robinhood stocks data

Data Description

Date Range: From july 2021 to August 2022

Datasource Description: The data is from Kaggle website and can be accessed using the link: https://www.kaggle.com/datasets/evangower/robinhood-stock-data (https://www.kaggle.com/datasets/evangower/robinhood-stock-data).

Dataset Description : This contains the historical stock price of Robinhood (ticker symbol HOOD) an American financial services company headquartered in Menlo Park, California, that facilitates commission-free trades of stocks, exchange-traded funds and cryptocurrencies via a mobile app introduced in March 2015.

The company went public on the Nasdaq on July 29, 2021, under the stock ticker HOOD. The opening price was $38, but dropped shortly afterwards to a low of $33.35 before starting to recover, reaching an all time high at $85. Subsequently, it fell sharply again after facing growing regulatory uncertainty, plunging on Q3 earnings and disclosing that a security breach affected 7 million customers.

```
library(TSA)
```

```
## Warning: package 'TSA' was built under R version 4.2.2
```

```
##
## Attaching package: 'TSA'
```

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

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

```
data <- read.csv("HOOD.csv")
summary(data)
```

```
##      date             close_last        volume             open      
##  Length:268         Min.   : 6.89   Min.   :  2579553   Min.   : 6.84  
##  Class :character   1st Qu.:10.30   1st Qu.:  9255350   1st Qu.:10.33  
##  Mode  :character   Median :13.52   Median : 14689590   Median :13.53  
##                     Mean   :21.56   Mean   : 18290238   Mean   :21.62  
##                     3rd Qu.:35.45   3rd Qu.: 21859865   3rd Qu.:35.92  
##                     Max.   :70.39   Max.   :175790500   Max.   :62.90  
##       high            low        
##  Min.   : 7.14   Min.   : 6.810  
##  1st Qu.:10.63   1st Qu.: 9.925  
##  Median :14.12   Median :13.225  
##  Mean   :22.42   Mean   :20.867  
##  3rd Qu.:37.03   3rd Qu.:34.612  
##  Max.   :85.00   Max.   :56.000
```

Data Pre-processing

```
data_clean <- na.omit(data)
```

Checking seasonality

```
library(seastests)
```

```
## Warning: package 'seastests' was built under R version 4.2.3
```

```
isSeasonal(data_clean$close_last, test = "combined", freq = 12)
```

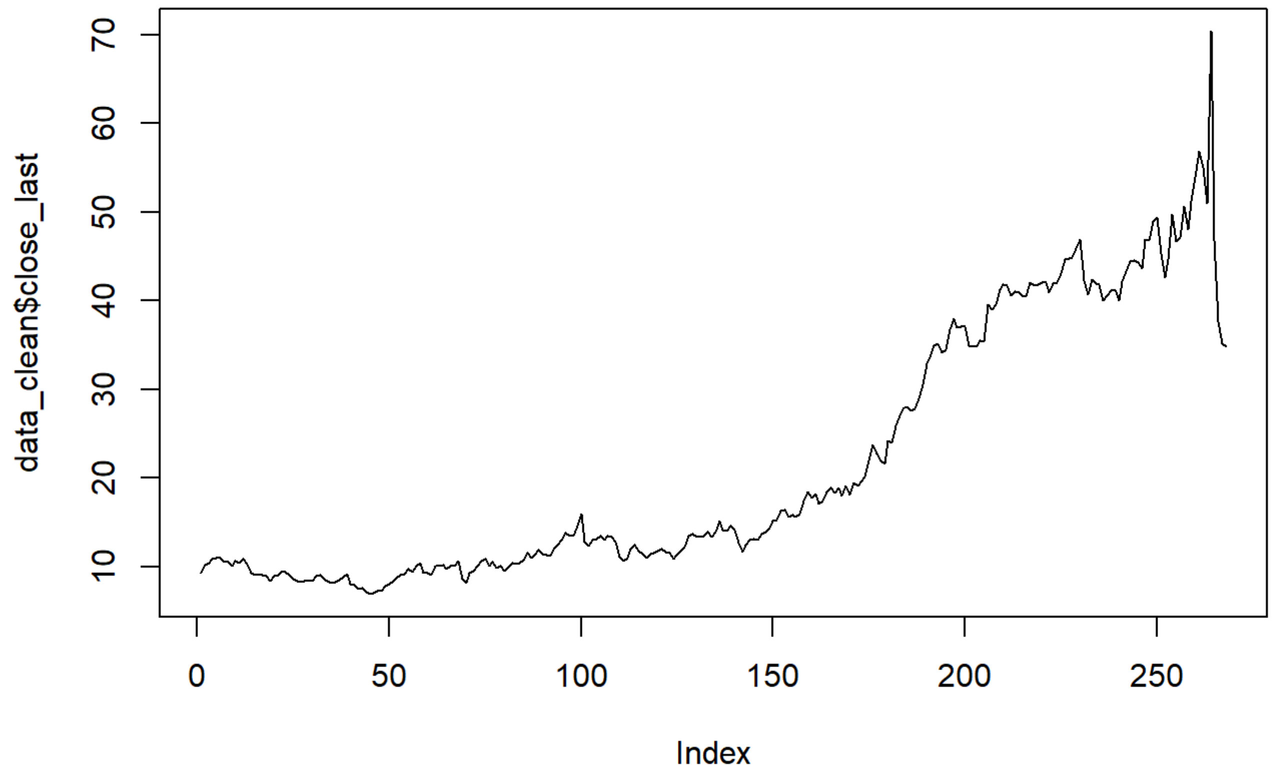
```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

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

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

Result of this test shows that the dataset is non-seasonal so we can go ahead with different tests.

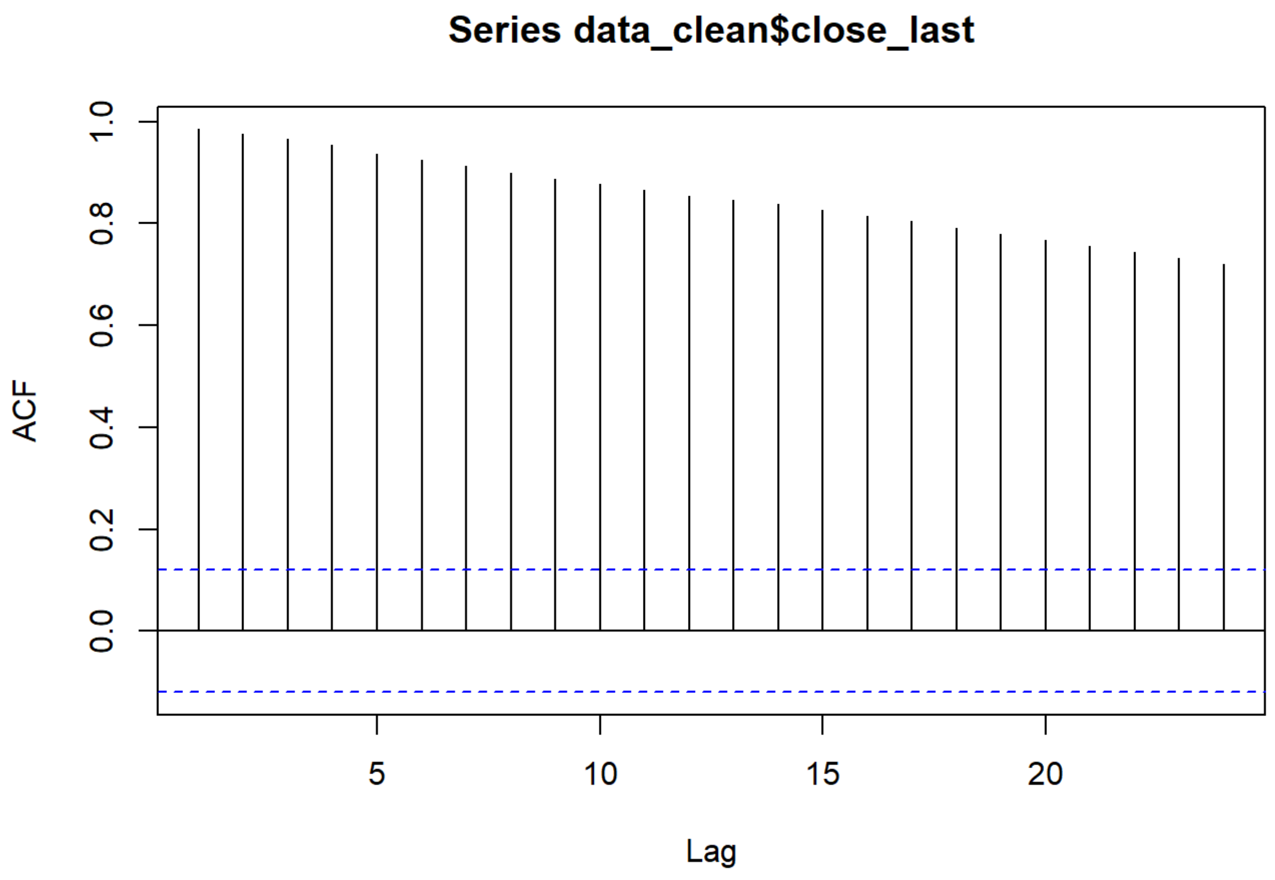
```
plot(data_clean$close_last, type='l')
```



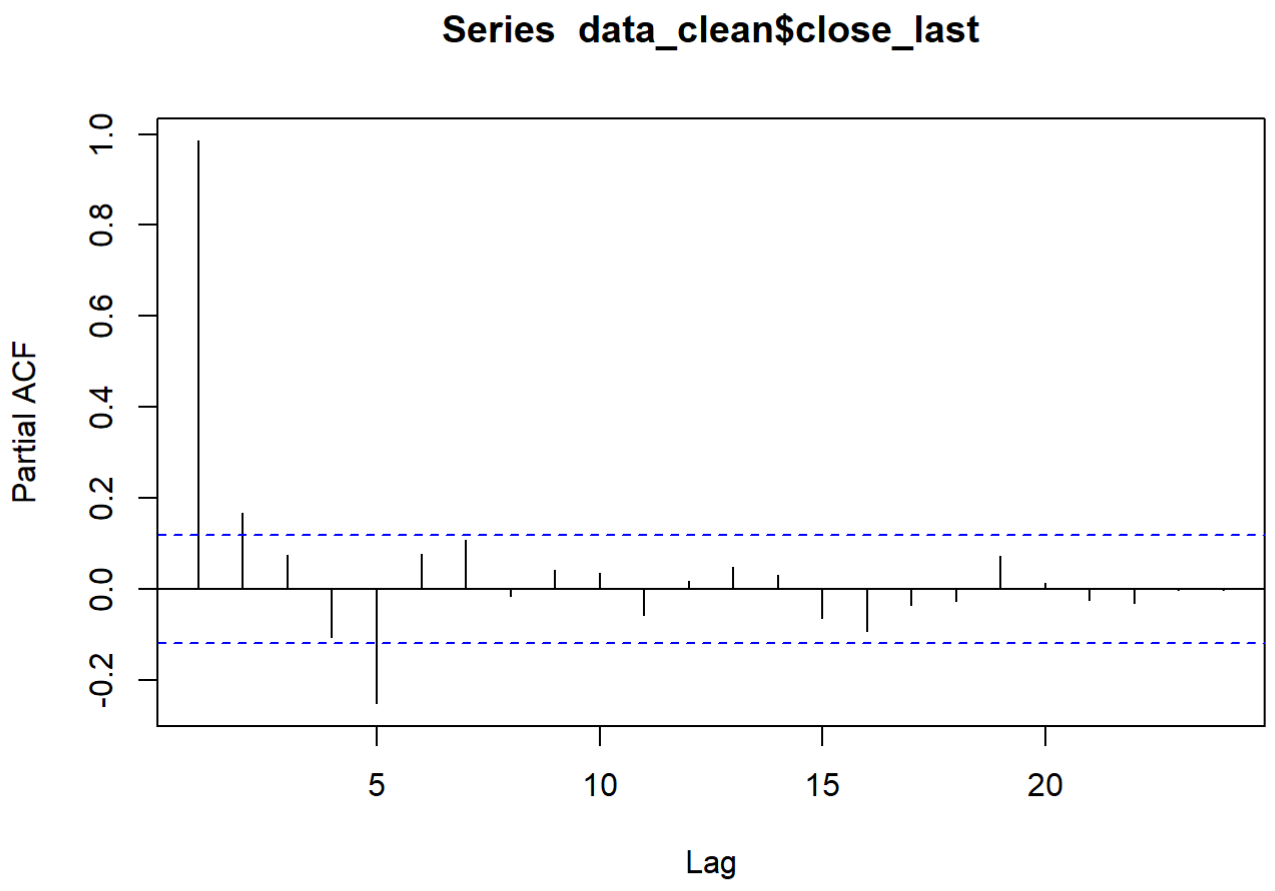
Plotting the ACF and PCF of the series.

Stationarity check.

```
acf(data_clean$close_last)
```



```
pacf(data_clean$close_last)
```



```
library(aTSA)
```

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

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

```
adf.test(data_clean$close_last)
```

```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag    ADF p.value
## [1,]   0 -0.119   0.610
## [2,]   1  0.149   0.687
## [3,]   2  0.334   0.740
## [4,]   3  0.300   0.730
## [5,]   4  0.111   0.676
## Type 2: with drift no trend
##      lag    ADF p.value
## [1,]   0 -1.246   0.611
## [2,]   1 -0.974   0.706
## [3,]   2 -0.819   0.760
## [4,]   3 -0.822   0.760
## [5,]   4 -0.933   0.720
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -2.98   0.163
## [2,]   1 -2.49   0.368
## [3,]   2 -2.25   0.470
## [4,]   3 -2.31   0.443
## [5,]   4 -2.37   0.419
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

The augmented Dickey Fuller test demonstrates that the P values are not less than.05. Therefore, the series is not stationary, and the null hypothesis must be rejected. In other words, the variance is not constant over time and the time series has some sort of time dependent structure.

The data’s autocorrelation plot shows relatively little deterioration. This supports the finding that the time series is not stationary from the Dickey-Fuller test. A strong autocorrelation is apparent from the trend in the Data, which is visible.

MAKING THE TIME SERIES STATIONARY

Using the Box Cox transformation

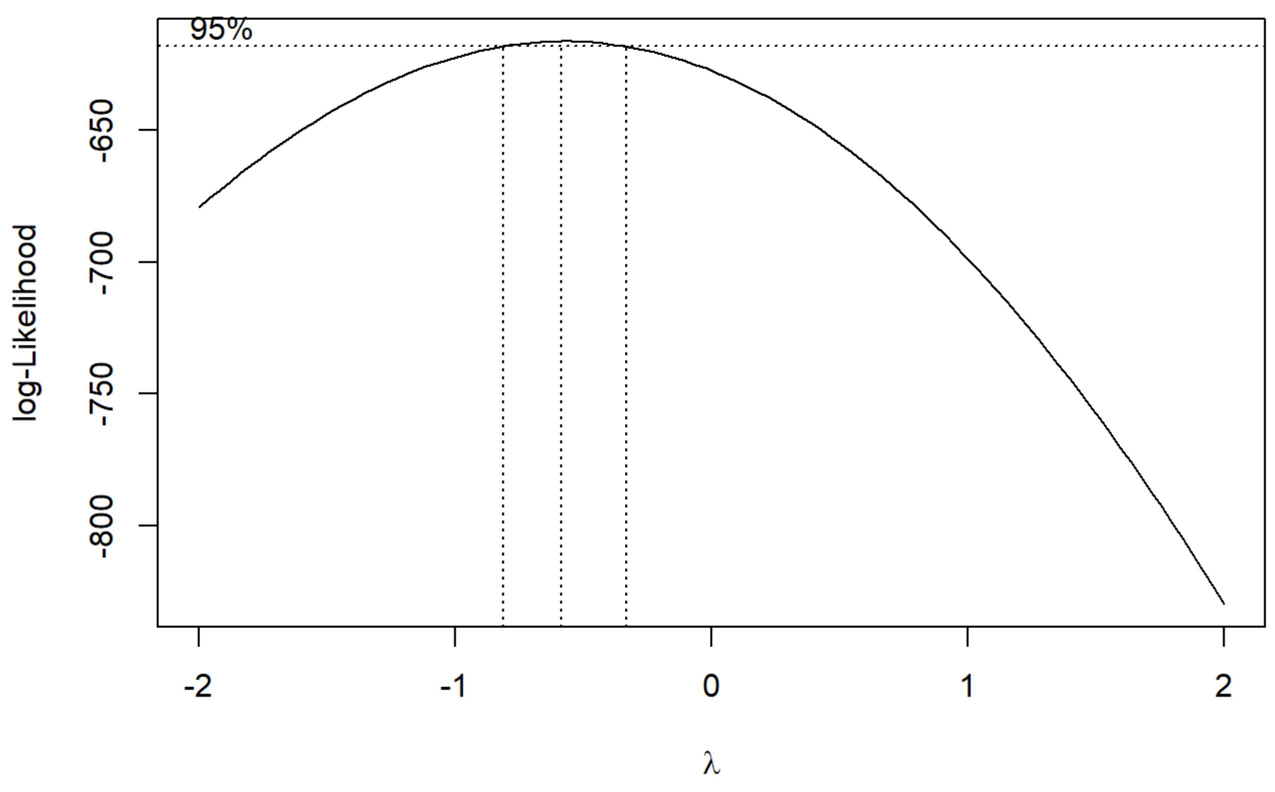
```
library(MASS)
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 4.2.3
```

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

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

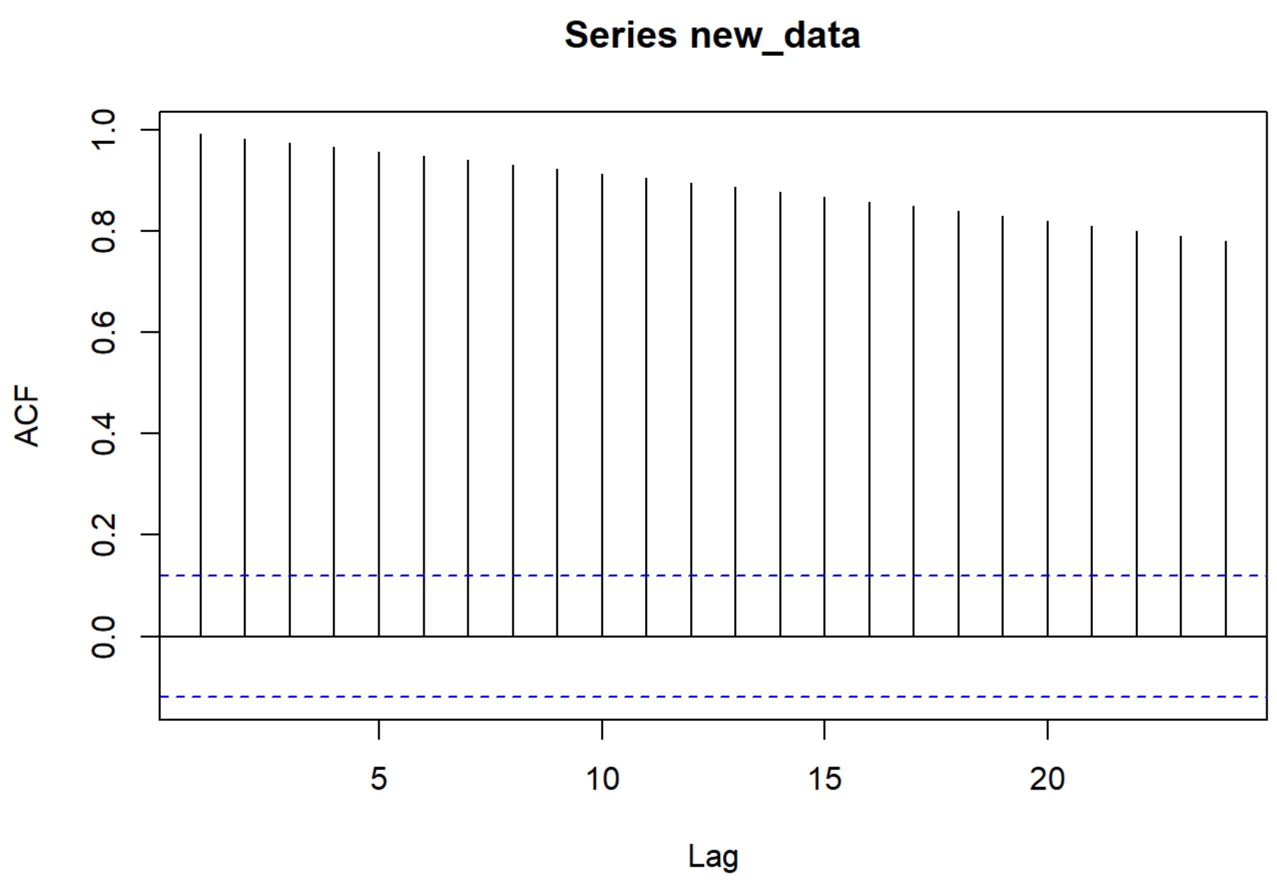
```
b <- boxcox(lm(data_clean$close_last ~ 1))
```



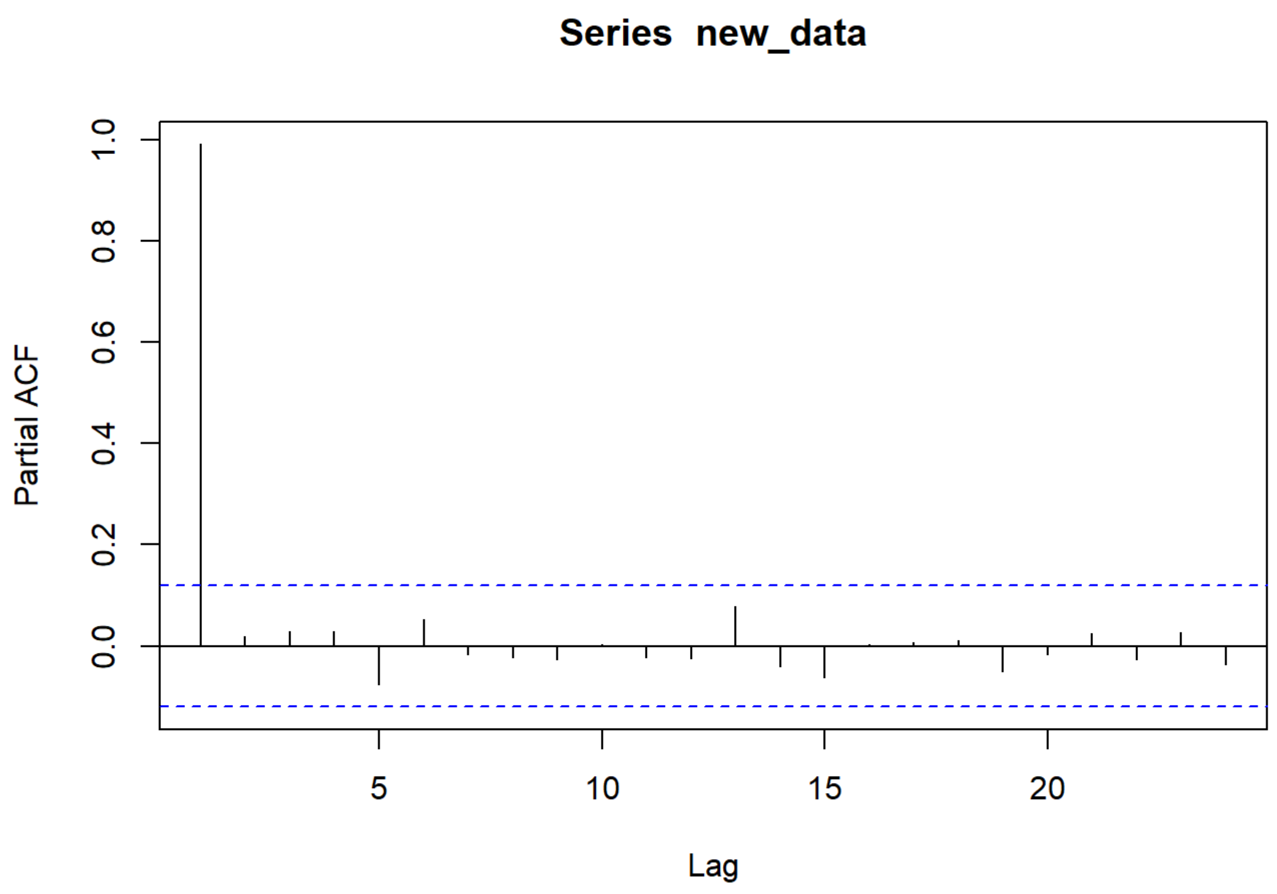
```
lambda <- b$x[which.max(b$y)]
lambda
```

```
## [1] -0.5858586
```

```
new_data <- (data_clean$close_last ^ lambda - 1) / lambda
acf(new_data)
```



```
pacf(new_data)
```



```
adf.test(new_data)
```

```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag  ADF p.value
## [1,]   0 1.27   0.948
## [2,]   1 1.11   0.928
## [3,]   2 1.19   0.939
## [4,]   3 1.34   0.954
## [5,]   4 1.31   0.952
## Type 2: with drift no trend
##      lag    ADF p.value
## [1,]   0 -0.789   0.771
## [2,]   1 -0.684   0.808
## [3,]   2 -0.587   0.842
## [4,]   3 -0.407   0.902
## [5,]   4 -0.377   0.906
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -2.71  0.2787
## [2,]   1 -3.11  0.1070
## [3,]   2 -3.03  0.1421
## [4,]   3 -3.02  0.1463
## [5,]   4 -3.14  0.0976
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

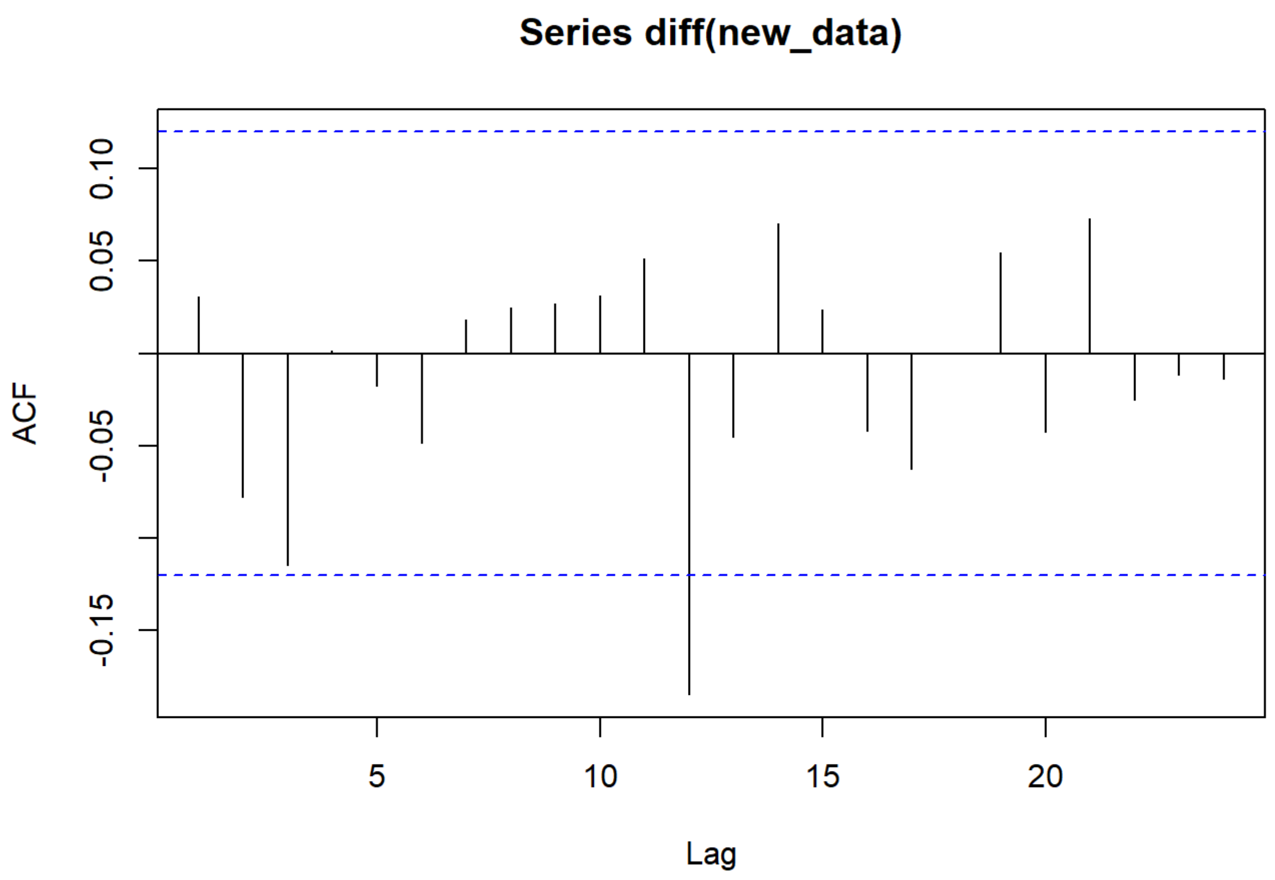
We can observe that the series is still moving even after the Box Cox change. The series is not stationary since the P value is bigger than 0.5. Additionally, the acf and pacf plots demonstrate that the time series still exhibits autocorrelation. Let’s do a seasonal differentiation of the time series to eliminate this association and make the series stationary.

SEASONAL DIFFERENCING

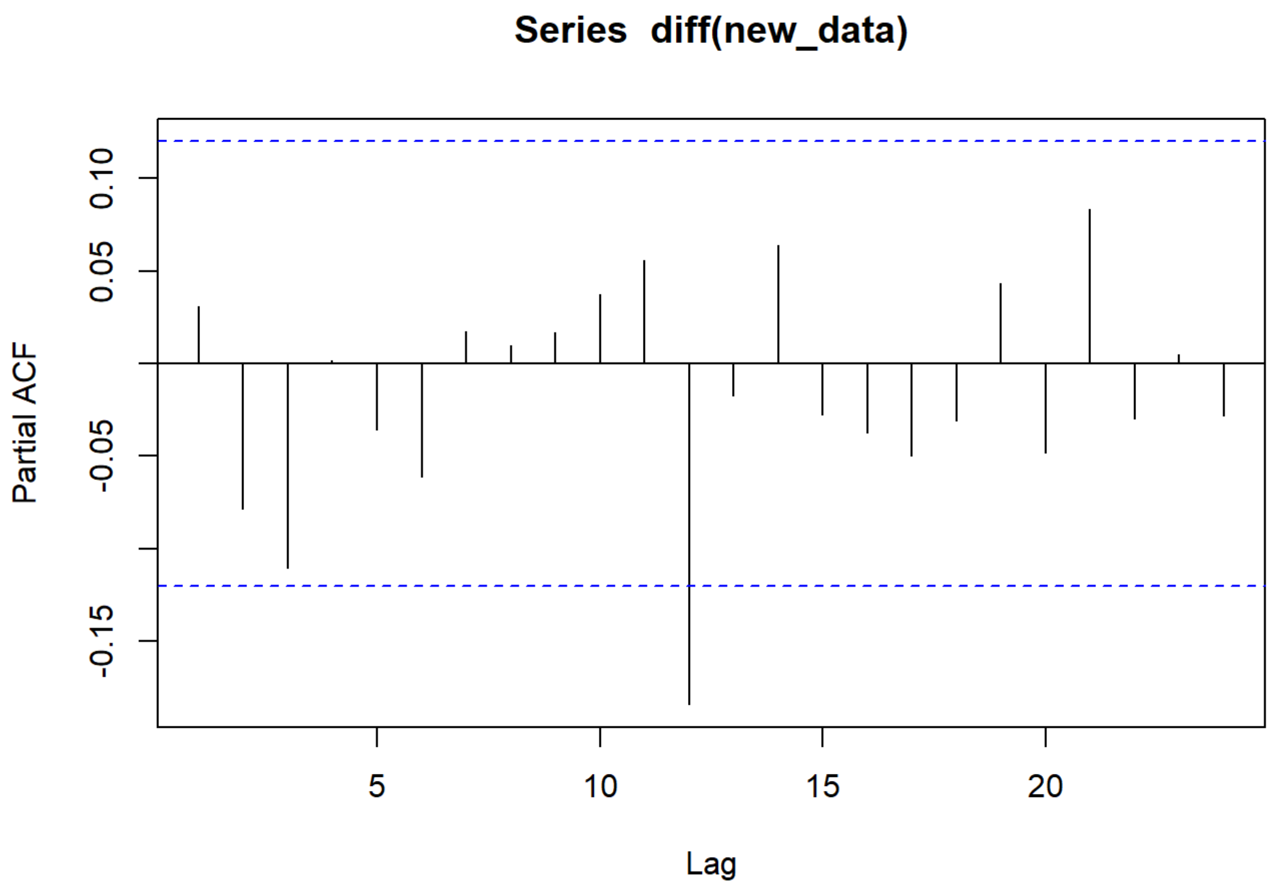
```
diff_ser <- diff(new_data)
adf.test(diff(new_data,lag = 12))
```

```
## Augmented Dickey-Fuller Test
## alternative: stationary
##
## Type 1: no drift no trend
##      lag   ADF p.value
## [1,]   0 -4.08    0.01
## [2,]   1 -4.22    0.01
## [3,]   2 -3.82    0.01
## [4,]   3 -3.43    0.01
## [5,]   4 -3.51    0.01
## Type 2: with drift no trend
##      lag   ADF p.value
## [1,]   0 -4.27    0.01
## [2,]   1 -4.52    0.01
## [3,]   2 -4.13    0.01
## [4,]   3 -3.74    0.01
## [5,]   4 -3.87    0.01
## Type 3: with drift and trend
##      lag   ADF p.value
## [1,]   0 -4.31  0.0100
## [2,]   1 -4.50  0.0100
## [3,]   2 -4.08  0.0100
## [4,]   3 -3.66  0.0280
## [5,]   4 -3.77  0.0209
## ----
## Note: in fact, p.value = 0.01 means p.value <= 0.01
```

```
acf(diff(new_data))
```



```
pacf(diff(new_data))
```



```
eacf(diff(new_data))
```

```
## AR/MA
##   0 1 2 3 4 5 6 7 8 9 10 11 12 13
## 0 o o o o o o o o o o o  x  o  o
## 1 x o o o o o o o o o o  x  o  o
## 2 x x o o o o o o o o o  x  o  o
## 3 o x x o o o o o o o o  x  o  o
## 4 x o x x o o o o o o o  x  o  o
## 5 x o x x x o o o o o o  x  o  o
## 6 x x x x x o o o o o o  x  o  o
## 7 x x o o o o o o o o o  x  o  o
```

After comparing the series, we can observe that the P value from the enhanced Dickey-Fuller test is less than 0.05, which indicates that it is statistically significant. As a result, we can say that the series is stationary at this point.Additionally, we can see that the correlations have greatly decreased in the ACF and PCF lots.

Now that we have a stationary series, we'll have a look at the EACF plots to finalize the order of our AR, MA models and then fit the model.

DETERMINING THE ORDER OF THE MODEL

The best model to fit the data can have p, q, P, and Q in the range of 0 to 2, according to the ACF, PACF, and EACF. Based on the lowest AIC for P, Q, p, and q values, we will choose the best model.

I’ve created a nested for loop that goes through each conceivable combination of P, Q, p, and q values in the range of 0 to 2 and fits a SARIMA model for each of them. The AICs for each of these models are then listed along with the matching P, Q, p, and q values. The top value in the list is then popped once the list has been sorted in ascending order. The P, Q, p, and q values that correspond to the lowest AIC model are represented by this value.

NESTED FOR LOOP FOR DERTMING THE VALUES OF P, Q, p and q

```
# Load the forecast package
library(forecast)

# Define the range of values for p, q, P, and Q
p_values <- c(0, 1, 2)
q_values <- c(0, 1, 2)
P_values <- c(0, 1, 2)
Q_values <- c(0, 1, 2)

# Initialize variables for storing the best model and its performance
best_model <- NULL
best_aic <- Inf

# Nested for loops to iterate over different parameter values
for (p in p_values) {
  for (q in q_values) {
    for (P in P_values) {
      for (Q in Q_values) {

        # Fit a seasonal ARIMA model with the current parameter values
        fit <- arima(data_clean$close_last, order=c(p,1,q), seasonal=c(P,1,Q), method="ML")

        # Evaluate the model performance using AIC
        current_aic <- AIC(fit)

        # Update the best model and its performance if the current model is better
        if (current_aic < best_aic) {
          best_model <- fit
          best_aic <- current_aic
          best_params <- c(p, q, P, Q)
        }
      }
    }
  }
}
```

```
## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced

## Warning in log(s2): NaNs produced
```

```
# Print the best SARIMA model parameters and AIC
print(paste("Best SARIMA model parameters:", paste(best_params, collapse=",")))
```

```
## [1] "Best SARIMA model parameters: 0,2,1,1"
```

```
print(paste("Best AIC:", best_aic))
```

```
## [1] "Best AIC: 1180.17374812786"
```

Parameter Estimation using best model

```
# Load the "forecast" package
library(forecast)

# Fit an ARIMA model to the "Hood stocks" dataset
arima_model <- Arima(data_clean$close_last, order = c(0,1,2))

# Make a seasonal ARIMA (SARIMA) model from the ARIMA model
sarima_model <- Arima(data_clean$close_last, order = c(0,1,2), seasonal = list(order = c(1,1,
1), period = 12))
# Print the model summaries
summary(arima_model)
```

```
## Series: data_clean$close_last
## ARIMA(0,1,2)
##
## Coefficients:
##           ma1      ma2
##       -0.2336  -0.0714
## s.e.   0.0622   0.0693
##
## sigma^2 = 4.818:  log likelihood = -587.79
## AIC=1181.59   AICc=1181.68   BIC=1192.35
##
## Training set error measures:
##                     ME     RMSE       MAE       MPE    MAPE     MASE
## Training set 0.1441677 2.182628 0.9792603 0.4685118 4.37784 1.048787
##                      ACF1
## Training set -0.008256217
```

```
summary(sarima_model)
```

```
## Series: data_clean$close_last
## ARIMA(0,1,2)(1,1,1)[12]
##
## Coefficients:
##           ma1      ma2     sar1     sma1
##       -0.2207  -0.0672  -0.2480  -0.8391
## s.e.   0.0642   0.0832   0.1454   0.0877
##
## sigma^2 = 5.179:  log likelihood = -579.48
## AIC=1168.96   AICc=1169.2   BIC=1186.67
##
## Training set error measures:
##                      ME     RMSE       MAE       MPE     MAPE     MASE
## Training set 0.03374839 2.202333 0.9563707 0.1763515 4.274156 1.024272
##                      ACF1
## Training set -0.003943351
```

These summaries provide information about the model coefficients, standard errors, and other statistics. By examining these summaries, we can assess the goodness of fit of the models and evaluate their forecasting performance.

Residual Analysis

```
# Load the "forecast" package
library(forecast)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

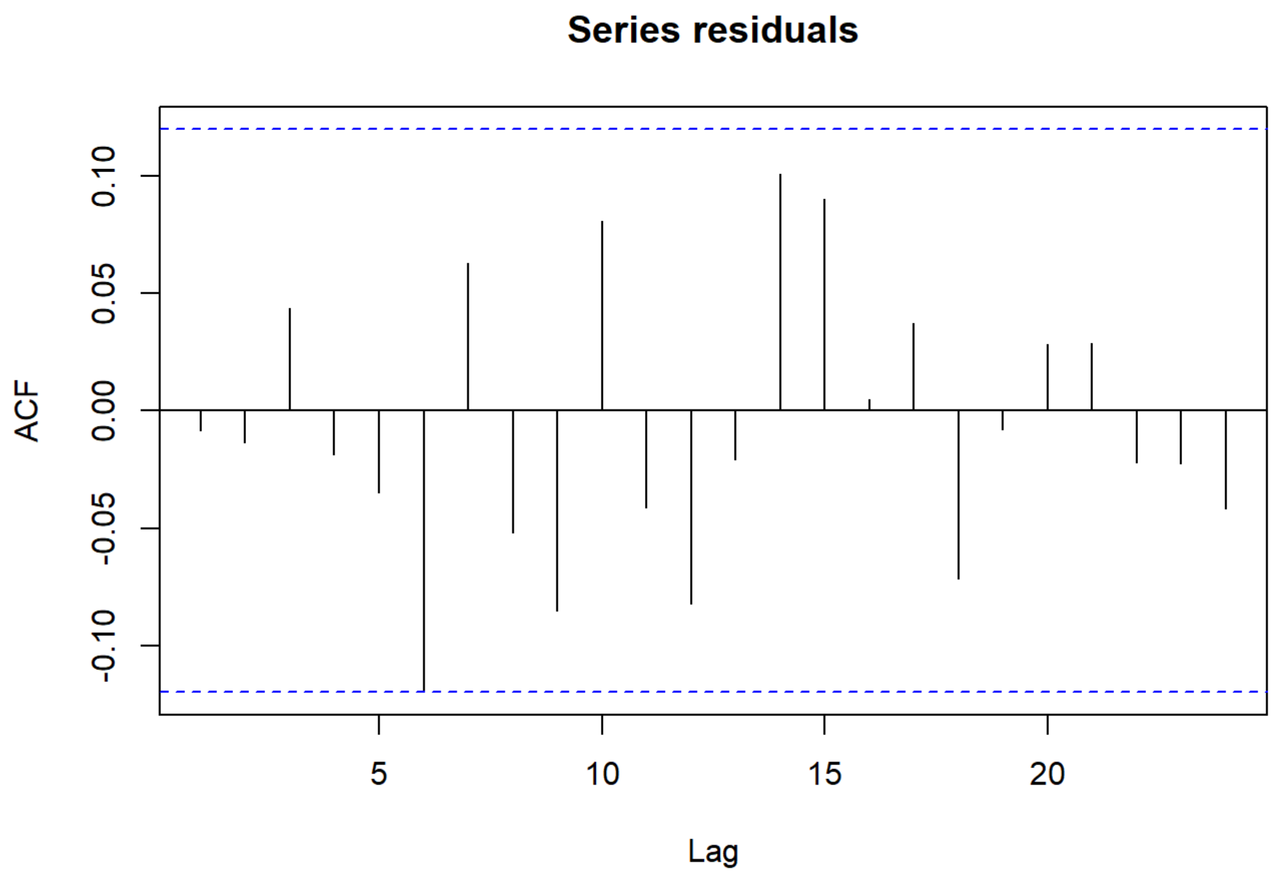
```
library(car)
```

```
## Warning: package 'car' was built under R version 4.2.3
```

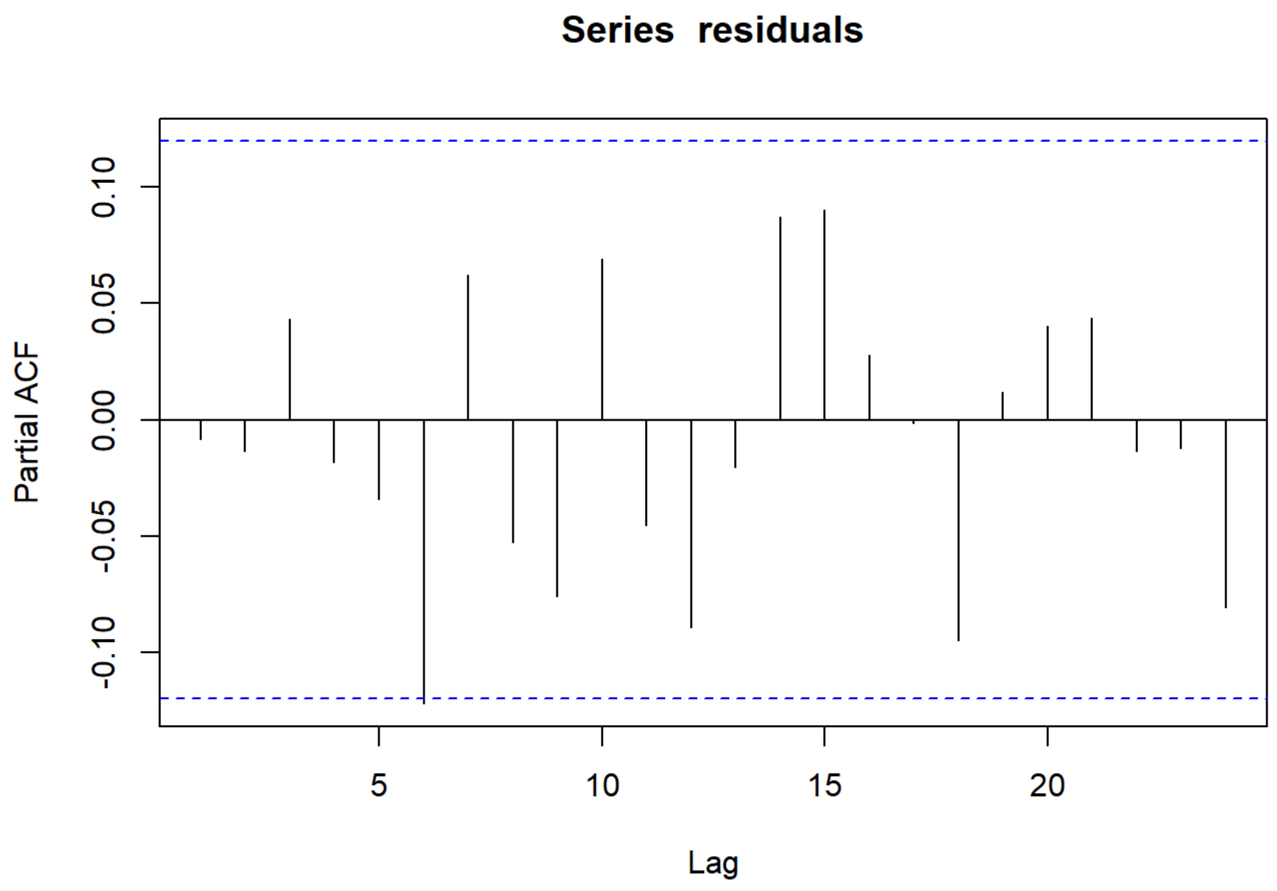
```
## Loading required package: carData
```

```
## Warning: package 'carData' was built under R version 4.2.3
```

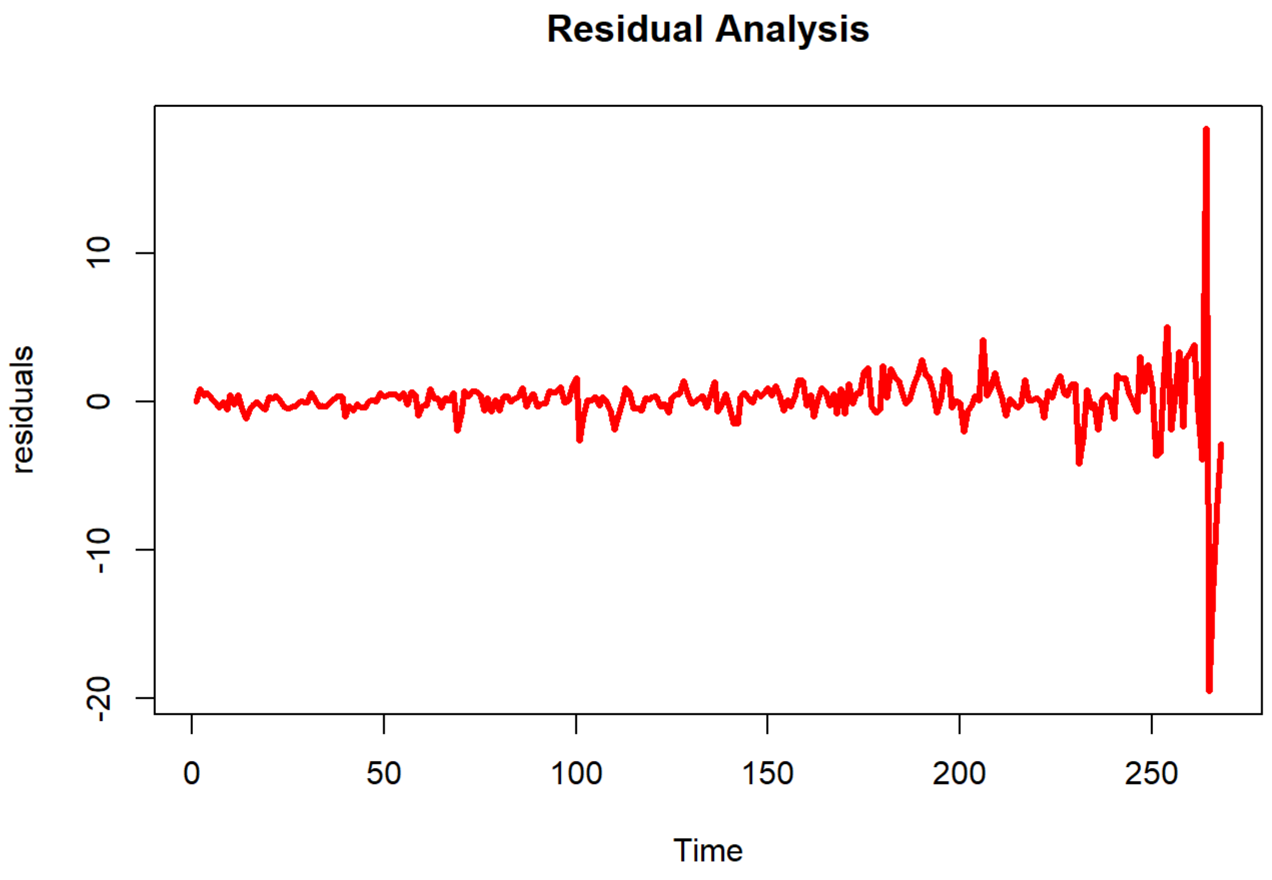
```
#library(Fit)

# Fit an ARIMA(0,1,2) model to the time series
arima_model <- Arima(data_clean$close_last, order = c(0,1,2))

# Extract the residuals from the ARIMA model
residuals <- residuals(arima_model)

# Plot the ACF and PACF of the residuals
acf(residuals)
```



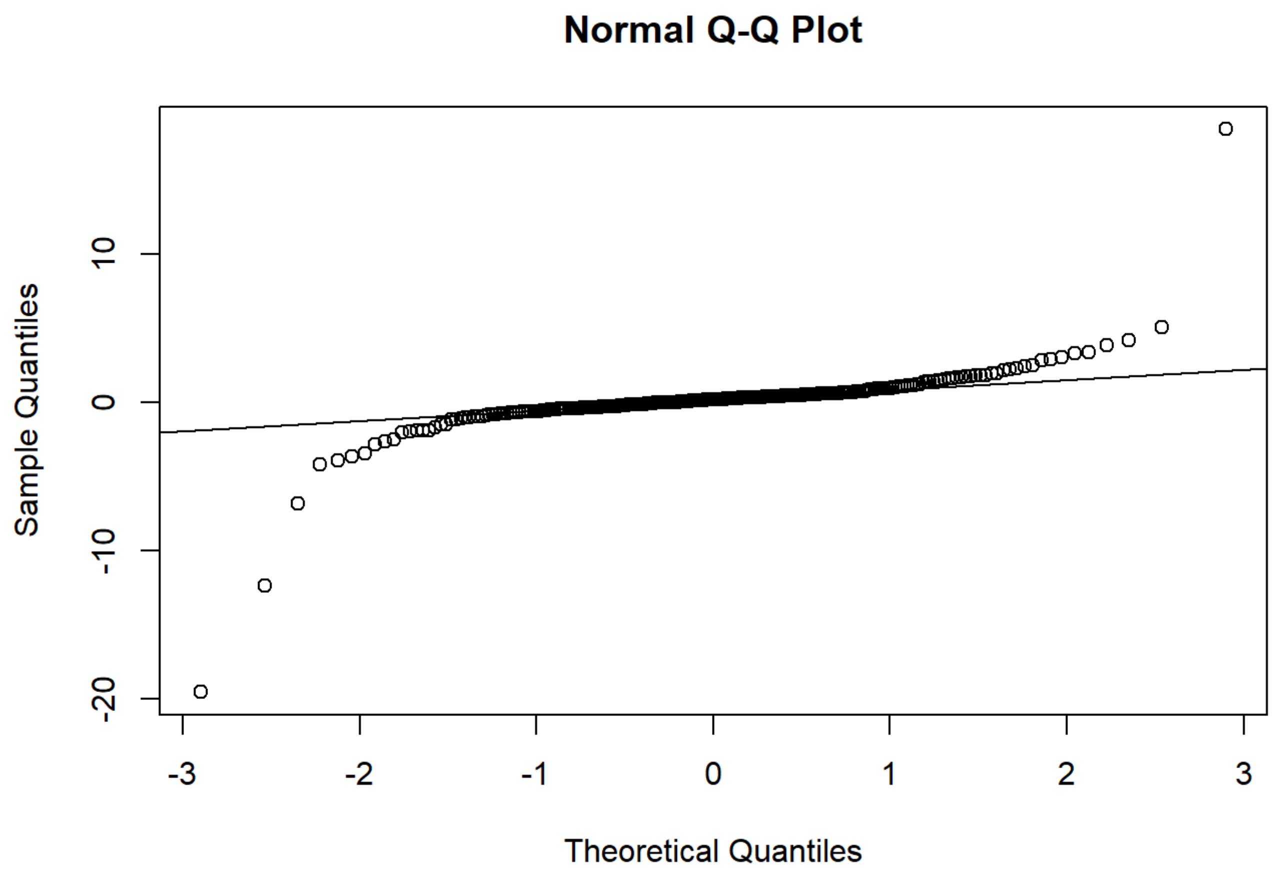
```
pacf(residuals)
```



```
#plot time series data
ts.plot(residuals,lwd=3,col="red",main='Residual Analysis')
```

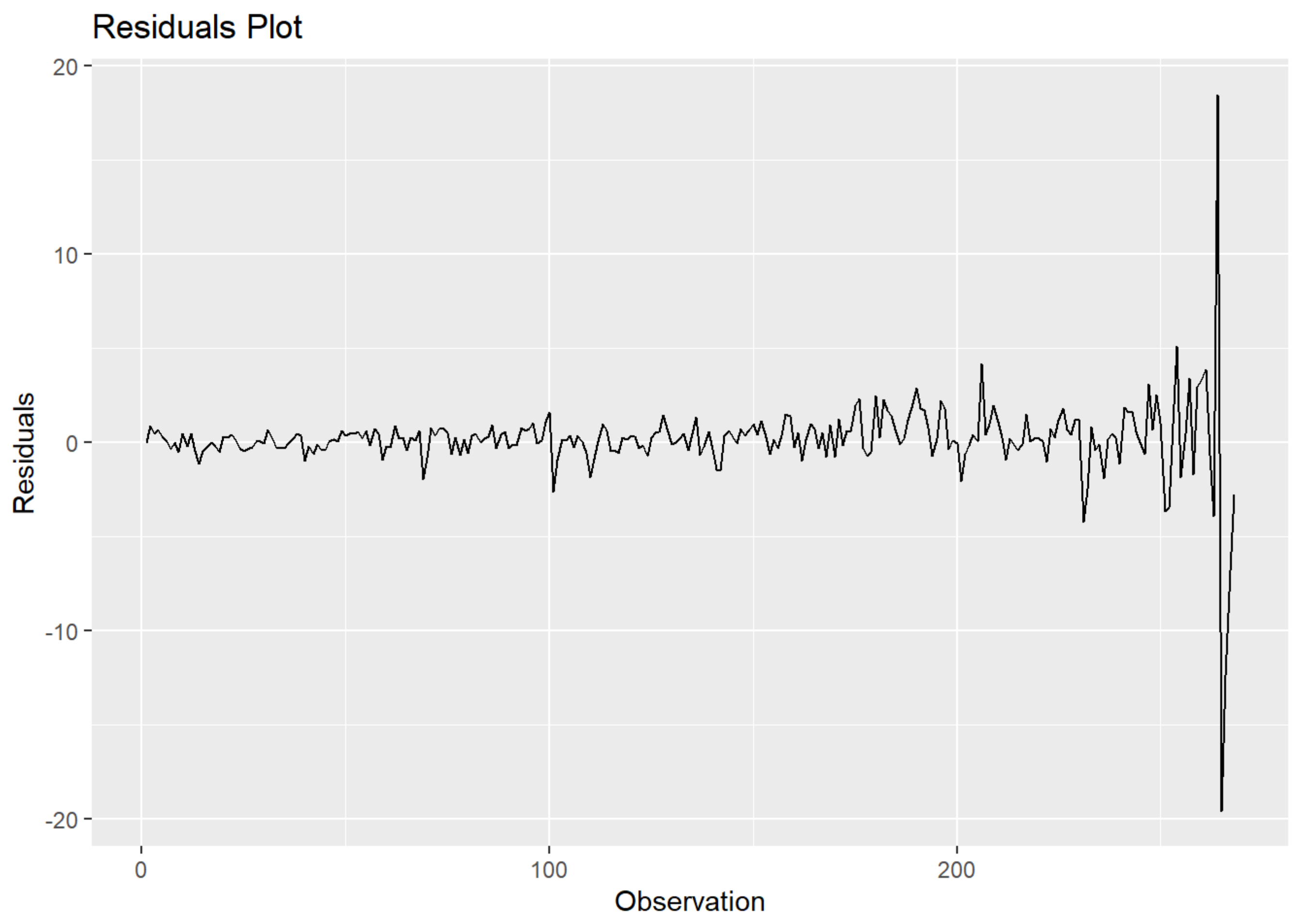


```
qqnorm(residuals)
qqline(residuals)
```

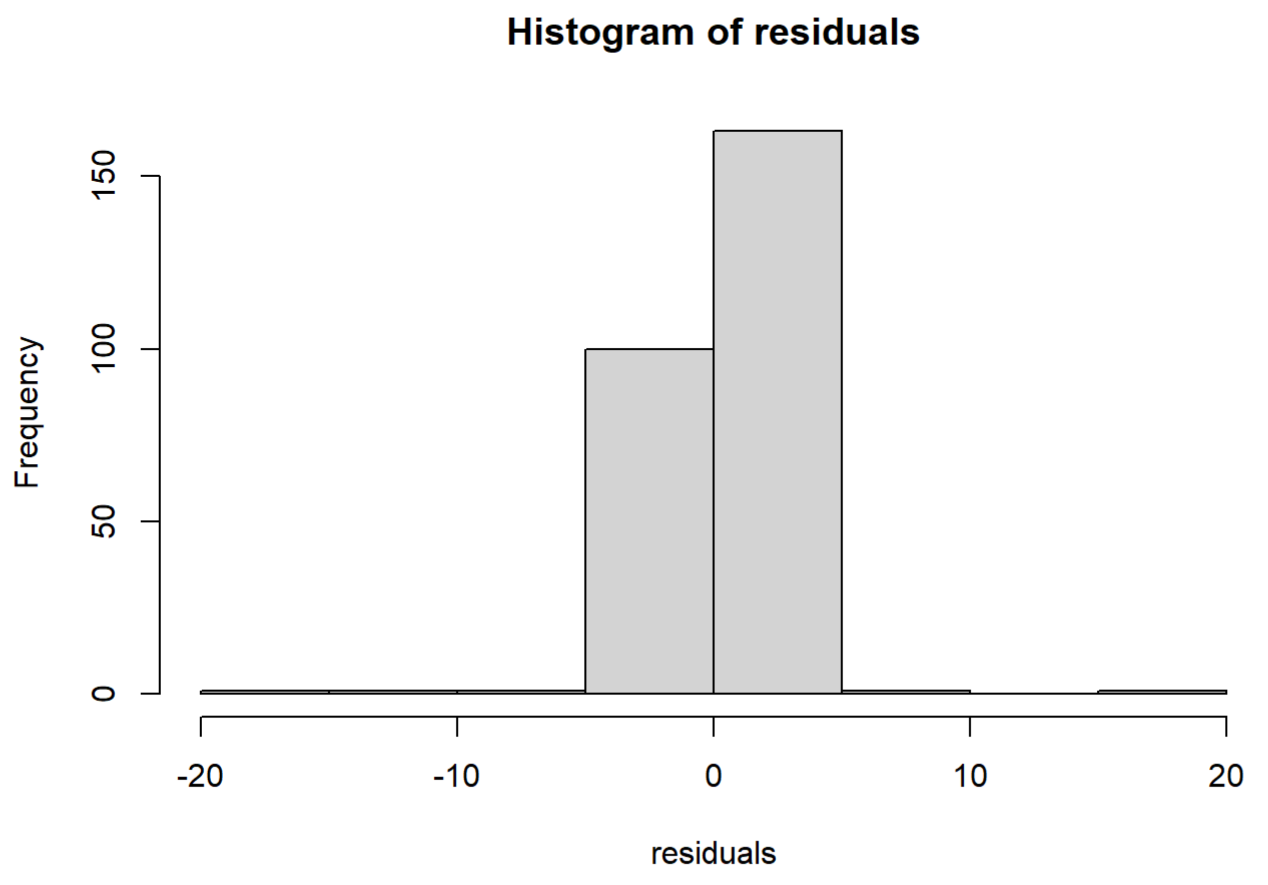


```
ggplot(data.frame(residuals = residuals), aes(x = 1:length(residuals), y = residuals)) +
  geom_line() +
  labs(x = "Observation", y = "Residuals", title = "Residuals Plot")
```

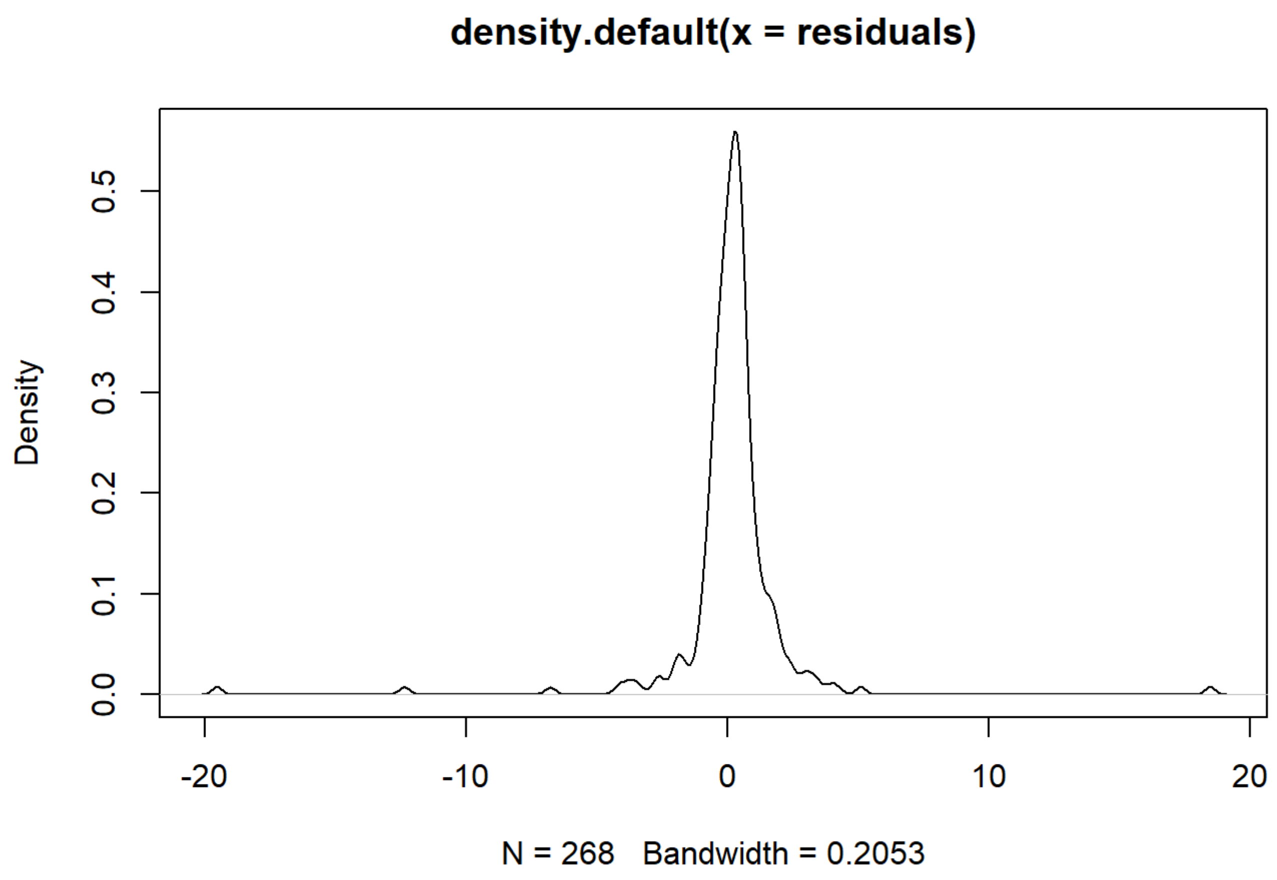
```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```



```
# Plot the histogram and density of the residuals
hist(residuals)
```



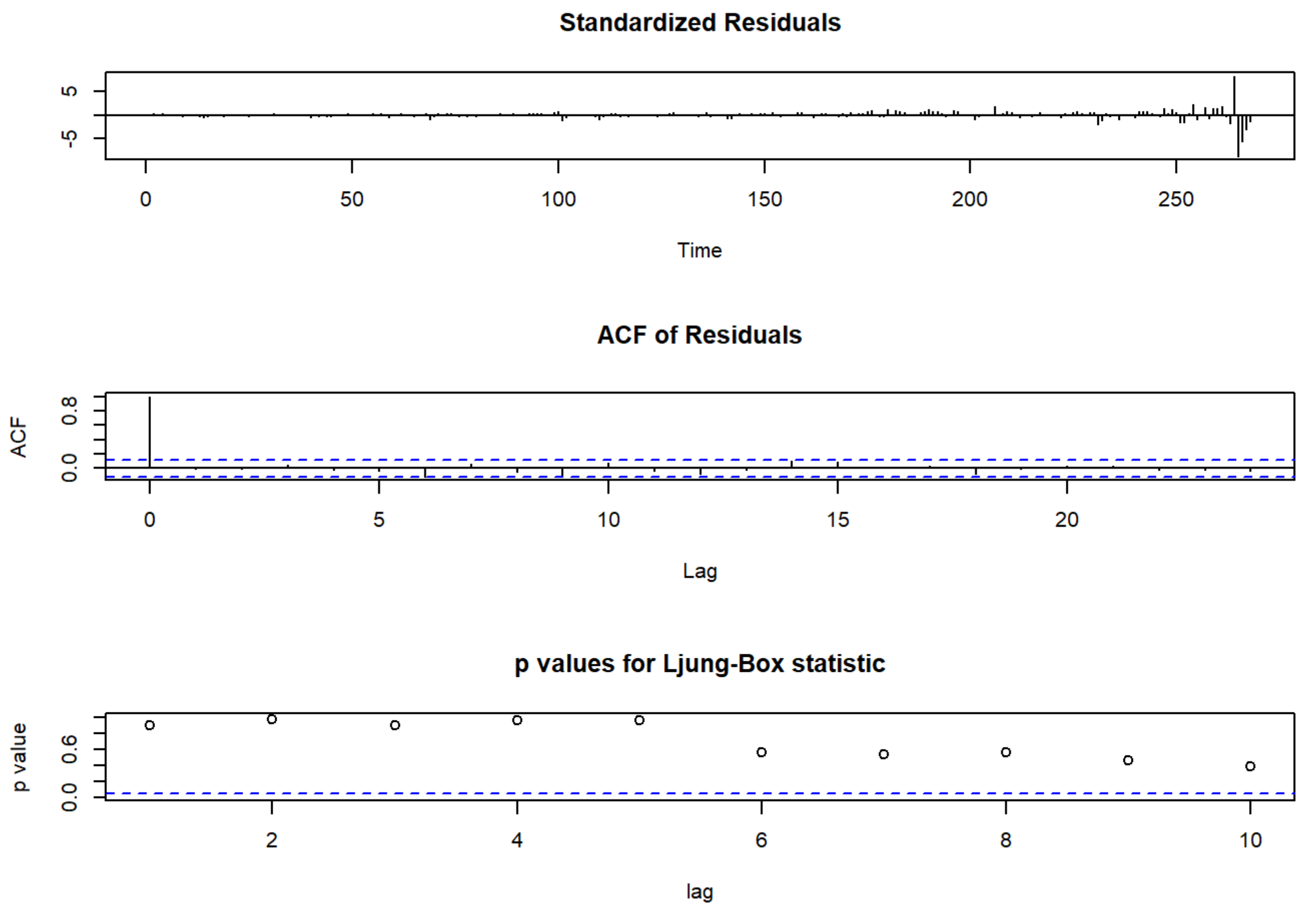
```
plot(density(residuals))
```



```
# Perform a Ljung-Box test on the residuals
Box.test(residuals, lag = 20, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals
## X-squared = 20.469, df = 20, p-value = 0.4289
```

```
#LBQPlot(residuals, lag.max = 20, SquaredQ = FALSE)
tsdiag(arima_model)
```

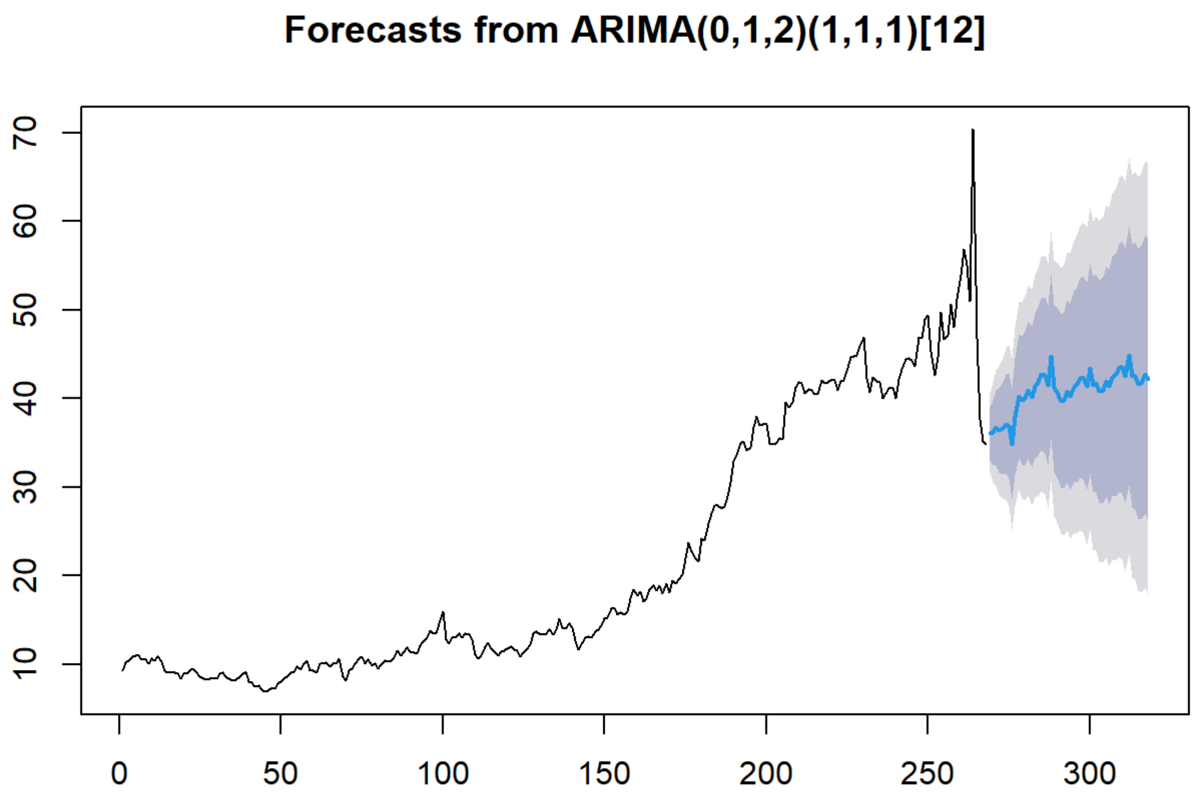


the code performs a Ljung-Box test on the residuals using the “Box.test” function to formally test for residual autocorrelation. The test statistic is compared to a chi-squared distribution with degrees of freedom equal to the number of lags specified in the test. A significant p-value (i.e., less than 0.05) indicates evidence of residual autocorrelation, which suggests that the model may be misspecified and may require further modification.

In this case, I got a p-value more than 0.05 in the Ljung-Box test. It suggests that this can be a good model.

Forecast Using the best model

```
# Load the "forecast" package
library(forecast)

# Load the dataset and convert it to a time series object

# Fit a SARIMA(0,1,2)(1,1,1) model to the time series
sarima_model <- Arima(data_clean$close_last, order = c(0,1,2), seasonal = list(order = c(1,1,
1), period = 12))

# Generate a 50 days forecast from the SARIMA model
forecast_data <- forecast(sarima_model, h = 50)

# Plot the forecasted values
plot(forecast_data)
```



Forecast looks good.

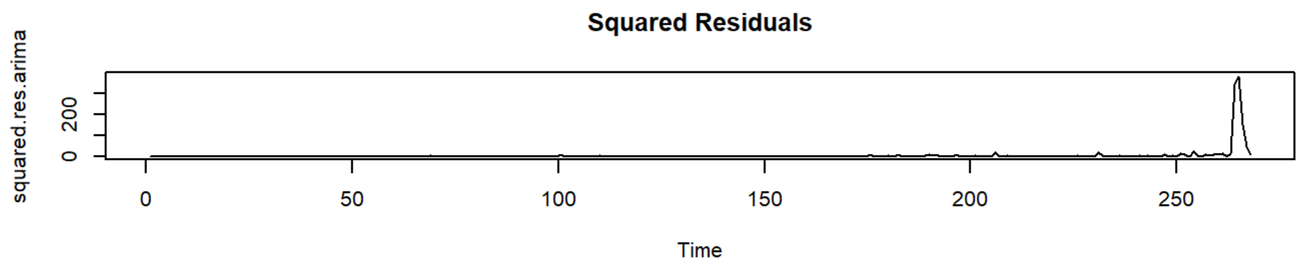
Fitting ARMA-GARCH model

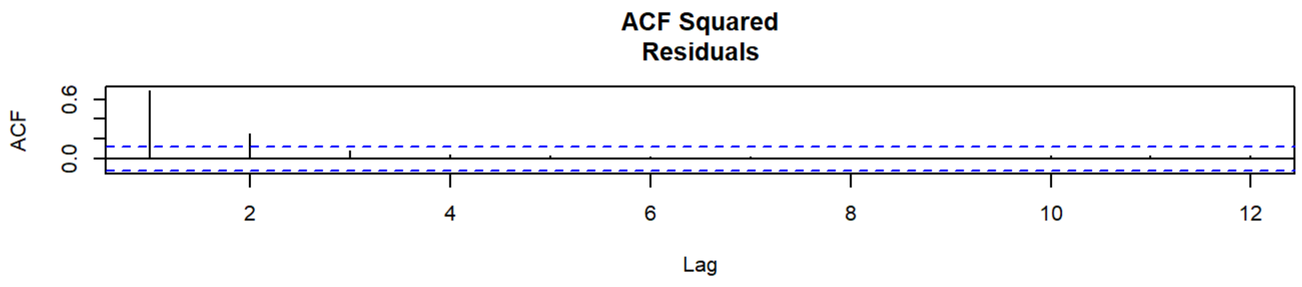
1. Fit ARIMA model and find out the orders of p and q as before

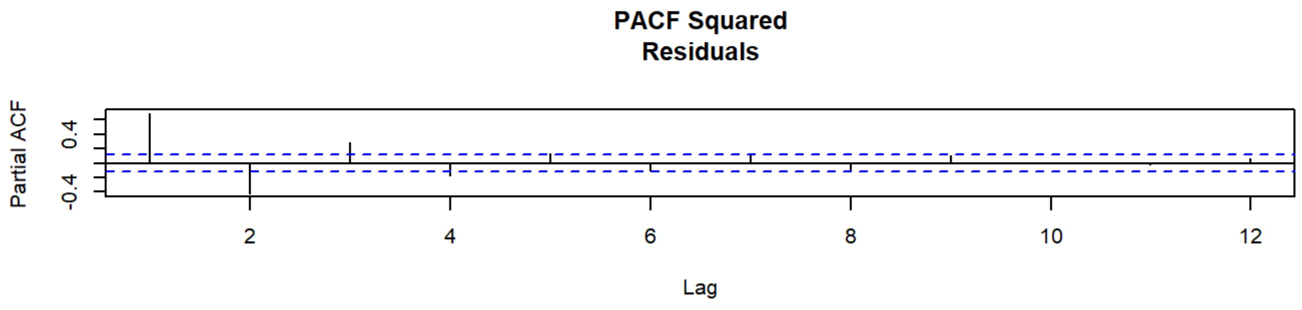
we found the p value 0 and q value 2 as per the lowest aic.

2. square the residuals of model in step 1, find out the orders of p and q (using ACF/PACF/EACF of squared residuals) similar to ARMA models. These new p and q of squared residuals are orders p and q of GARCH model.

```
res.arima=arima_model$residuals
squared.res.arima=res.arima^2
par(mfcol=c(3,1))
plot(squared.res.arima,main='Squared Residuals')
acf.squared=acf(squared.res.arima,main='ACF Squared
Residuals',lag.max=12)
pacf.squared=pacf(squared.res.arima,main='PACF Squared
Residuals',lag.max=12)
```







```
eacf.squared=eacf(squared.res.arima)
```

```
## AR/MA
##   0 1 2 3 4 5 6 7 8 9 10 11 12 13
## 0 x x o o o o o o o o o  o  o  o
## 1 x o x o o o o o o o o  o  o  o
## 2 x o o o o o o o o o o  o  o  o
## 3 x x o o o o o o o o o  o  o  o
## 4 x x x o o o o o o o o  o  o  o
## 5 x x x o o o o o o o o  o  o  o
## 6 x x x o o o o o o o o  o  o  o
## 7 x o o o o o o o o o o  o  o  o
```

By looking at the model we can identify that there is one significant lag in both ACF and PACf and even EACF suggest that (1,1) model fits better. So GARCH(1,1) can be a option.

3. Now you have ARMA (p,q) and GARCH (p,q) model. Use package of “rugarch” in R to fit the ARMA-GARCH model together.

```
library(rugarch)
```

```
## Warning: package 'rugarch' was built under R version 4.2.3
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'rugarch'
```

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

```
ts_data <- ts(data_clean$close_last, frequency = 12)

# Define the ARMA(0,2) model
spec_arma <- ugarchspec(mean.model = list(arimaOrder = c(0,1,2)),
                        variance.model = list(garchOrder = c(2,2)))
```

```
## Warning: unidentified option(s) in mean.model:
##  arimaOrder
```

```
# Fit the ARMA(0,2) model
fit_arma <- ugarchfit(spec_arma, data = ts_data)

# Define the GARCH(2,2) model
spec_garch <- ugarchspec(mean.model = list(arimaOrder = c(0,1,2)),
                         variance.model = list(garchOrder = c(2,2)))
```

```
## Warning: unidentified option(s) in mean.model:
##  arimaOrder
```

```
# Fit the GARCH(2,2) model
fit_garch <- ugarchfit(spec_garch, data = ts_data)

# Combine the ARMA and GARCH models
spec_armagarch <- ugarchspec(mean.model = list(arimaOrder = c(0,1,2)),
                             variance.model = list(garchOrder = c(2,2)),
                             distribution.model = "std")
```

```
## Warning: unidentified option(s) in mean.model:
##  arimaOrder
```

```
# Fit the ARMA-GARCH model
fit_armagarch <- ugarchfit(spec_armagarch, data = ts_data)
print(fit_armagarch)
```

```
## 
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
## 
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(2,2)
## Mean Model   : ARFIMA(1,0,1)
## Distribution : std
## 
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error    t value Pr(>|t|)
## mu      9.186610    1.208890   7.599213 0.000000
## ar1     1.000000    0.005206 192.084711 0.000000
## ma1     0.069477    0.071080   0.977447 0.328348
## omega   0.114054    0.096559   1.181191 0.237527
## alpha1  0.238650    0.100679   2.370410 0.017768
## alpha2  0.209478    0.095747   2.187840 0.028681
## beta1   0.000012    0.321031   0.000037 0.999971
## beta2   0.550859    0.163467   3.369859 0.000752
## shape   3.622339    0.689806   5.251243 0.000000
## 
## Robust Standard Errors:
##         Estimate  Std. Error    t value Pr(>|t|)
## mu      9.186610    0.460005  19.970674 0.000000
## ar1     1.000000    0.006780 147.484432 0.000000
## ma1     0.069477    0.084372   0.823462 0.410245
## omega   0.114054    0.172108   0.662689 0.507530
## alpha1  0.238650    0.123928   1.925717 0.054140
## alpha2  0.209478    0.125232   1.672720 0.094382
## beta1   0.000012    0.243495   0.000049 0.999961
## beta2   0.550859    0.393922   1.398396 0.161994
## shape   3.622339    0.757230   4.783674 0.000002
## 
## LogLikelihood : -363.0224
## 
## Information Criteria
## ------------------------------------
##                     
## Akaike       2.7763
## Bayes        2.8969
## Shibata      2.7741
## Hannan-Quinn 2.8247
## 
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic   p-value
## Lag[1]                      3.656 0.0558671
## Lag[2*(p+q)+(p+q)-1][5]     5.719 0.0002379
## Lag[4*(p+q)+(p+q)-1][9]     7.411 0.0933255
## d.o.f=2
## H0 : No serial correlation
## 
```

```
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                      1.150  0.2835
## Lag[2*(p+q)+(p+q)-1][11]     2.435  0.9274
## Lag[4*(p+q)+(p+q)-1][19]     4.333  0.9681
## d.o.f=4
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[5]    0.2788 0.500 2.000  0.5975
## ARCH Lag[7]    0.2910 1.473 1.746  0.9504
## ARCH Lag[9]    0.3807 2.402 1.619  0.9912
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  2.4682
## Individual Statistics:
## mu     0.002664
## ar1    0.600092
## ma1    0.037718
## omega  1.046013
## alpha1 0.318003
## alpha2 0.308814
## beta1  0.885562
## beta2  0.910382
## shape  0.877985
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          2.1 2.32 2.82
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value   prob sig
## Sign Bias           0.8809 0.3792
## Negative Sign Bias  0.8376 0.4030
## Positive Sign Bias  1.4792 0.1403
## Joint Effect        3.4776 0.3237
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     22.30       0.2697
## 2    30     32.90       0.2820
## 3    40     45.73       0.2128
## 4    50     55.88       0.2322
##
##
## Elapsed time : 0.2242541
```

as per that p values of ljung-box test and p values of goodness of fit we can conclude that this is a good model.

Forecast using ARMA-GARCH model

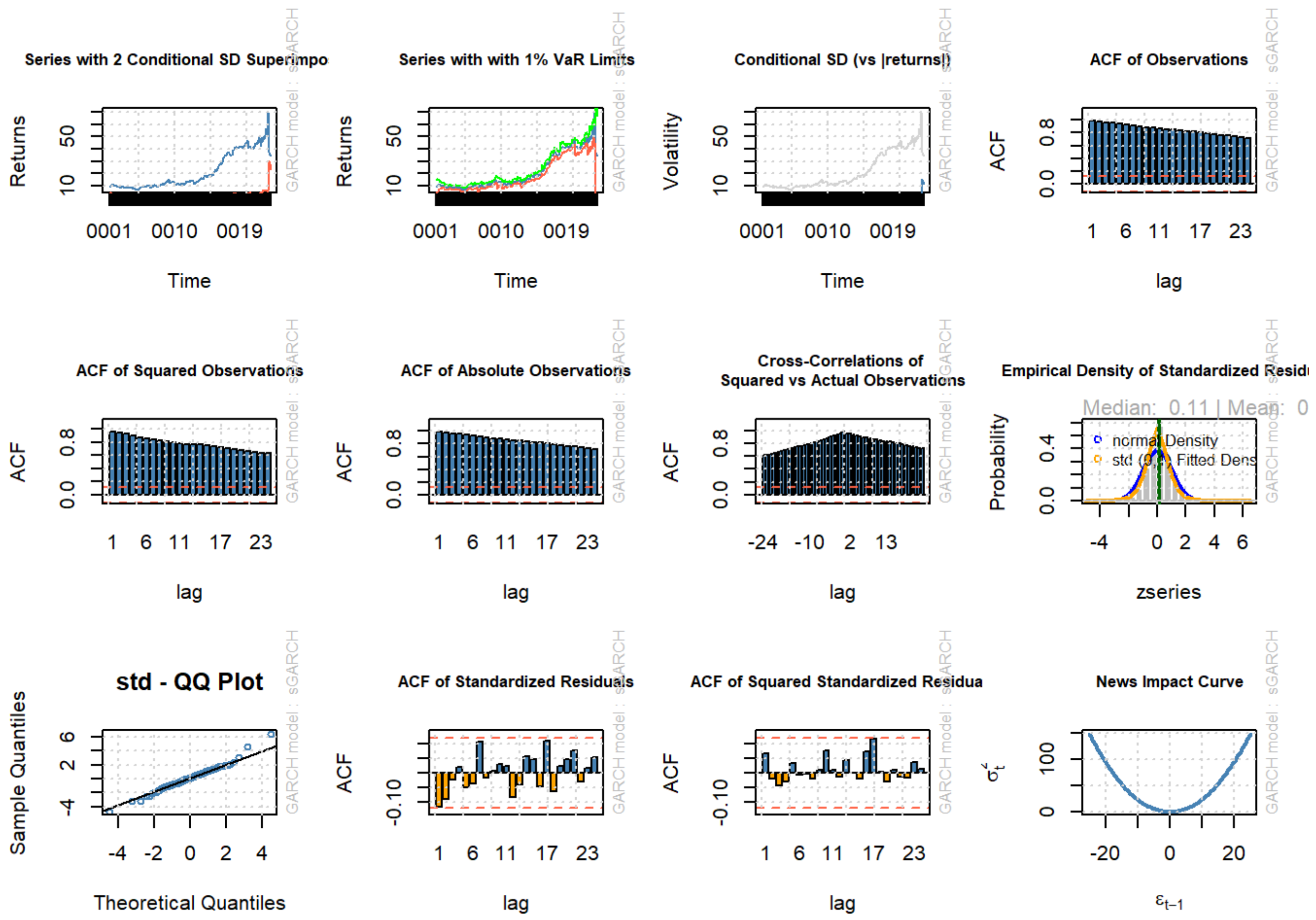
```
library(TSA)
library(rugarch)
# Make a forecast for the next 12 months
forecast_armagarch <- ugarchforecast(fit_armagarch, n.ahead = 12)
print(forecast_armagarch)
```

```
##
## *------------------------------------*
## *       GARCH Model Forecast         *
## *------------------------------------*
## Model: sGARCH
## Horizon: 12
## Roll Steps: 0
## Out of Sample: 0
##
## 0-roll forecast [T0=Apr 0023]:
##      Series Sigma
## T+1   34.81 10.51
## T+2   34.81 10.23
## T+3   34.81 10.44
## T+4   34.81 10.28
## T+5   34.81 10.41
## T+6   34.81 10.31
## T+7   34.81 10.38
## T+8   34.81 10.33
## T+9   34.81 10.37
## T+10  34.81 10.34
## T+11  34.81 10.36
## T+12  34.81 10.35
```

This is the sigma values of forecasting using the ARMA-GARCH model.

```
forecast <- plot(fit_armagarch,which='all')
```

```
##
## please wait...calculating quantiles...
```



Simulation and Forecasting

```
library(zoo)
```

```
## Warning: package 'zoo' was built under R version 4.2.3
```

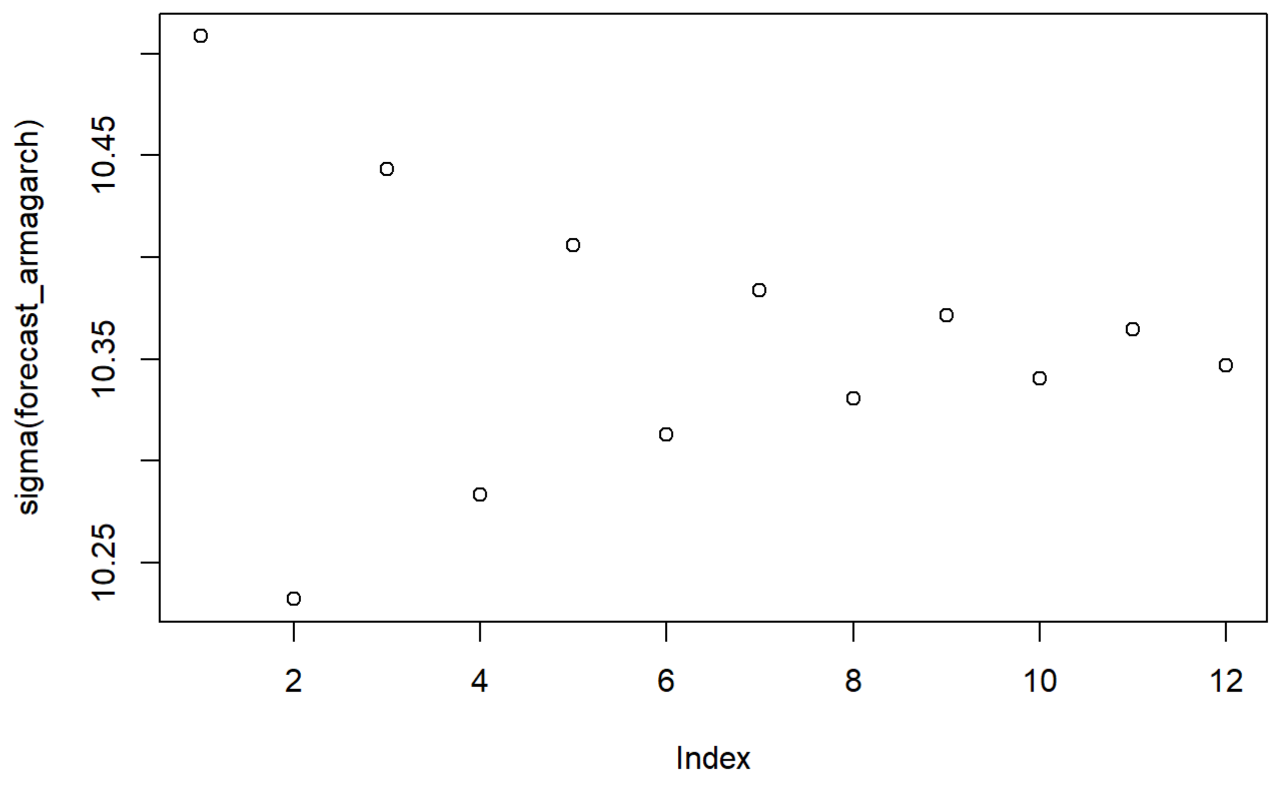
```
##
## Attaching package: 'zoo'
```

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

```
setfixed(spec_armagarch) <- as.list(coef(fit_armagarch))
coef(fit_armagarch)
```

```
##           mu          ar1          ma1        omega       alpha1       alpha2
## 9.186610e+00 1.000000e+00 6.947700e-02 1.140543e-01 2.386500e-01 2.094783e-01
##        beta1        beta2        shape
## 1.183603e-05 5.508593e-01 3.622339e+00
```

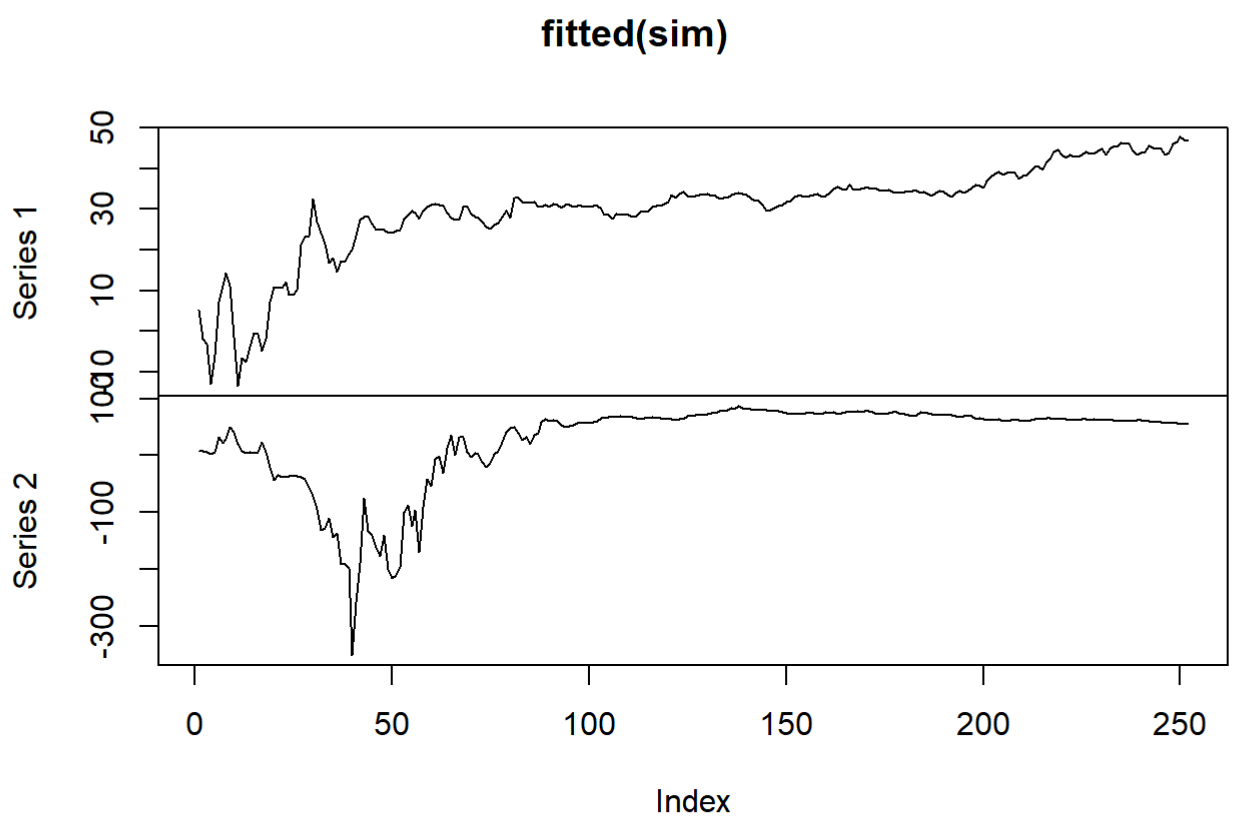
```
plot(sigma(forecast_armagarch))
```



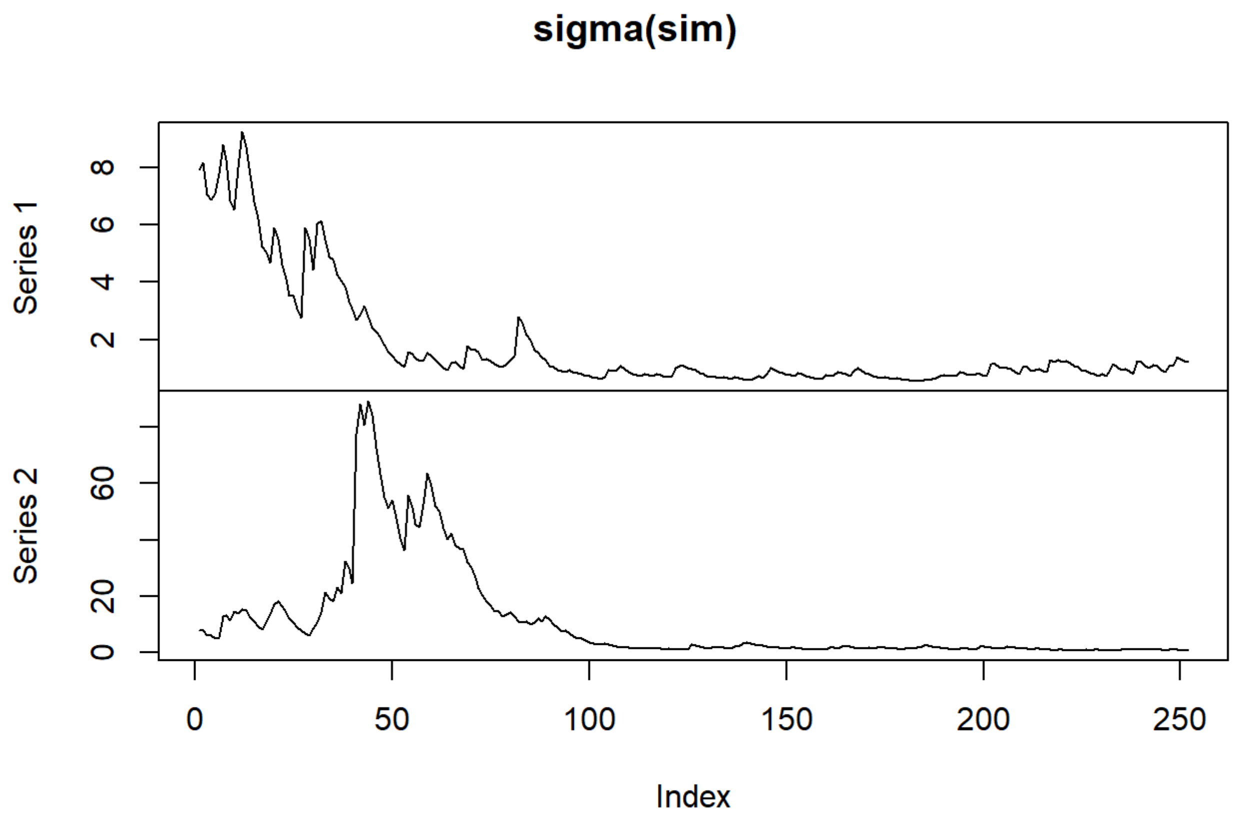
```
sim <- ugarchpath(spec = spec_armagarch, m.sim = 2, n.sim = 1*252, rseed = 123)
```

Here I plots the volatility forecast produced by the forecast_armagarch object, which was obtained by calling the ugarchforecast() function on the fit_armagarch object and simulates 2 paths of length 252 (i.e., 1 year) from the ARMA-GARCH model specified in spec_armagarch, using a random seed of 123.

```
plot.zoo(fitted(sim))
```

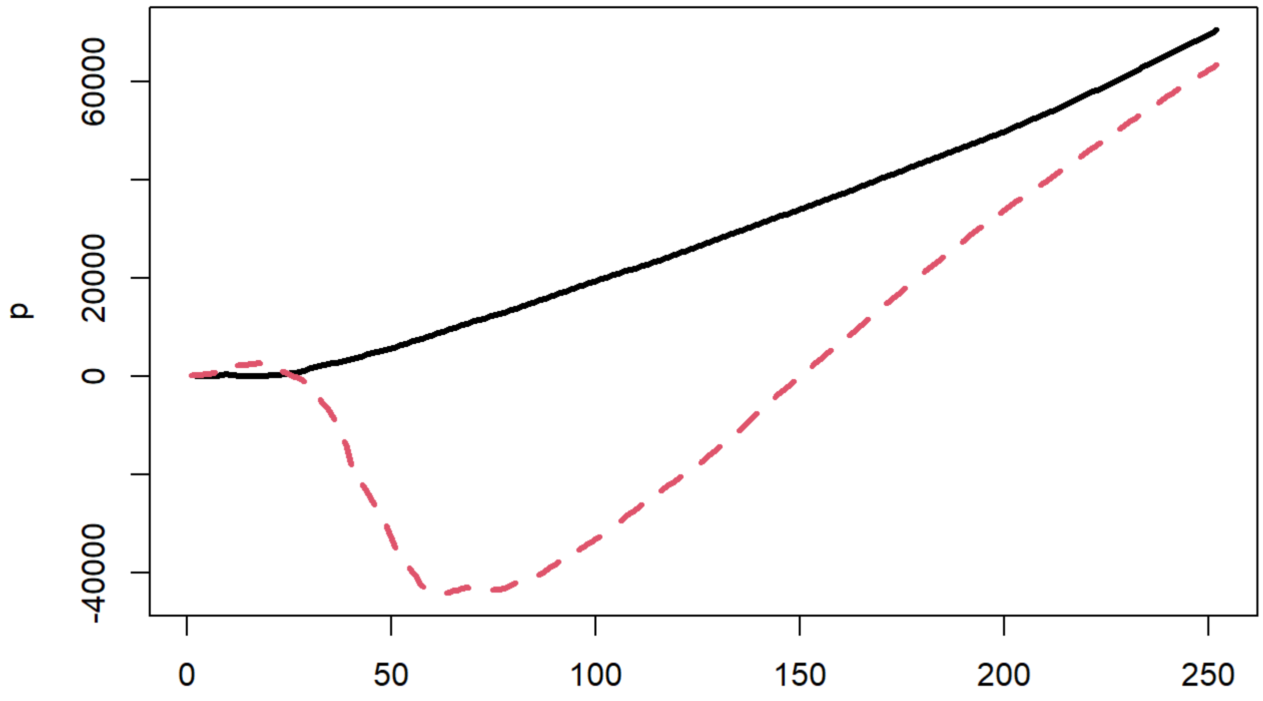


```
plot.zoo(sigma(sim))
```



Then I plotted the fitted values of the simulated data, which are obtained by applying the model to the simulated data. After plotting fitted values I plot the sigma values of that fitted simulated data.

```
p <- 9.26*apply(fitted(sim), 2, 'cumsum') + 9.26
matplot(p, type = "l", lwd = 3)
```



It calculates the cumulative returns from the simulated data using the formula provided. Here, the constant 9.26 represents the initial value of the time series. plots the cumulative returns over time.

Overall, this code fits an ARMA-GARCH model to time series data, simulates from the model, and plots the results. The resulting plots show the volatility forecast, simulated data, simulated volatility, and cumulative returns. In the last plot it is a forecast of two simulated series and the mean of two forecast can be fit to our original forecast.

Thank you.