Time Series MA-641 Final Project

Sharven Rane CWID: 2001 1112

Motivation

The motivation behind our time series forecasting project stems from the critical need for businesses to proactively manage and optimize their operations in an ever-changing environment. Time series forecasting provides a powerful tool to predict future trends based on historical data, enabling organizations to make informed decisions and allocate resources effectively.

Introduction

The Airbnb dataset encapsulates a comprehensive snapshot of the global short-term rental marketplace, providing a rich array of information on property listings, host details, pricing, availability, and guest reviews. This dataset serves as a valuable resource for researchers, data scientists, and enthusiasts to explore trends, patterns, and correlations within the dynamic ecosystem of Airbnb. Analyzing this data can yield insights into traveler preferences, the economic impact on local communities, pricing strategies, and the overall evolution of the sharing economy.

The seasonal monthly sales dataset serves as a crucial foundation for time series forecasting, offering a detailed chronicle of sales trends over specific periods. This dataset encompasses the cyclical variations and recurring patterns inherent in seasonal sales, providing a comprehensive record of monthly fluctuations. As businesses strive to optimize inventory management, marketing strategies, and resource allocation, the analysis of this time series data becomes paramount.

R Markdown

```
library(tseries)
```

```
## Warning: package 'tseries' was built under R version 4.1.3
```

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

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.1.3
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
```

```
## Warning: package 'tibble' was built under R version 4.1.3
```

```
## Warning: package 'tidyr' was built under R version 4.1.3
```

```
## Warning: package 'readr' was built under R version 4.1.3
```

```
## Warning: package 'purrr' was built under R version 4.1.3
```

```
## Warning: package 'dplyr' was built under R version 4.1.3
```

```
## Warning: package 'stringr' was built under R version 4.1.3
```

```
## Warning: package 'forcats' was built under R version 4.1.3
```

```
## Warning: package 'lubridate' was built under R version 4.1.3
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.1     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.1     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
```

```
## -- Conflicts --------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to be
come errors
```

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

```
## Warning: package 'forecast' was built under R version 4.1.3
```

```
library(TSA)
```

```
## Warning: package 'TSA' was built under R version 4.1.3
```

```
## Registered S3 methods overwritten by 'TSA':
##   method       from
##   fitted.Arima forecast
##   plot.Arima   forecast
##
## Attaching package: 'TSA'
##
## The following object is masked from 'package:readr':
##
##     spec
##
## The following objects are masked from 'package:stats':
##
##     acf, arima
##
## The following object is masked from 'package:utils':
##
##     tar
```

```
library(urca)
```

```
## Warning: package 'urca' was built under R version 4.1.3
```

```
library(FinTS)
```

```
## Warning: package 'FinTS' was built under R version 4.1.3
```

```
## Loading required package: zoo
```

```
## Warning: package 'zoo' was built under R version 4.1.3
```

```
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
##
## Attaching package: 'FinTS'
##
## The following object is masked from 'package:forecast':
##
##     Acf
```

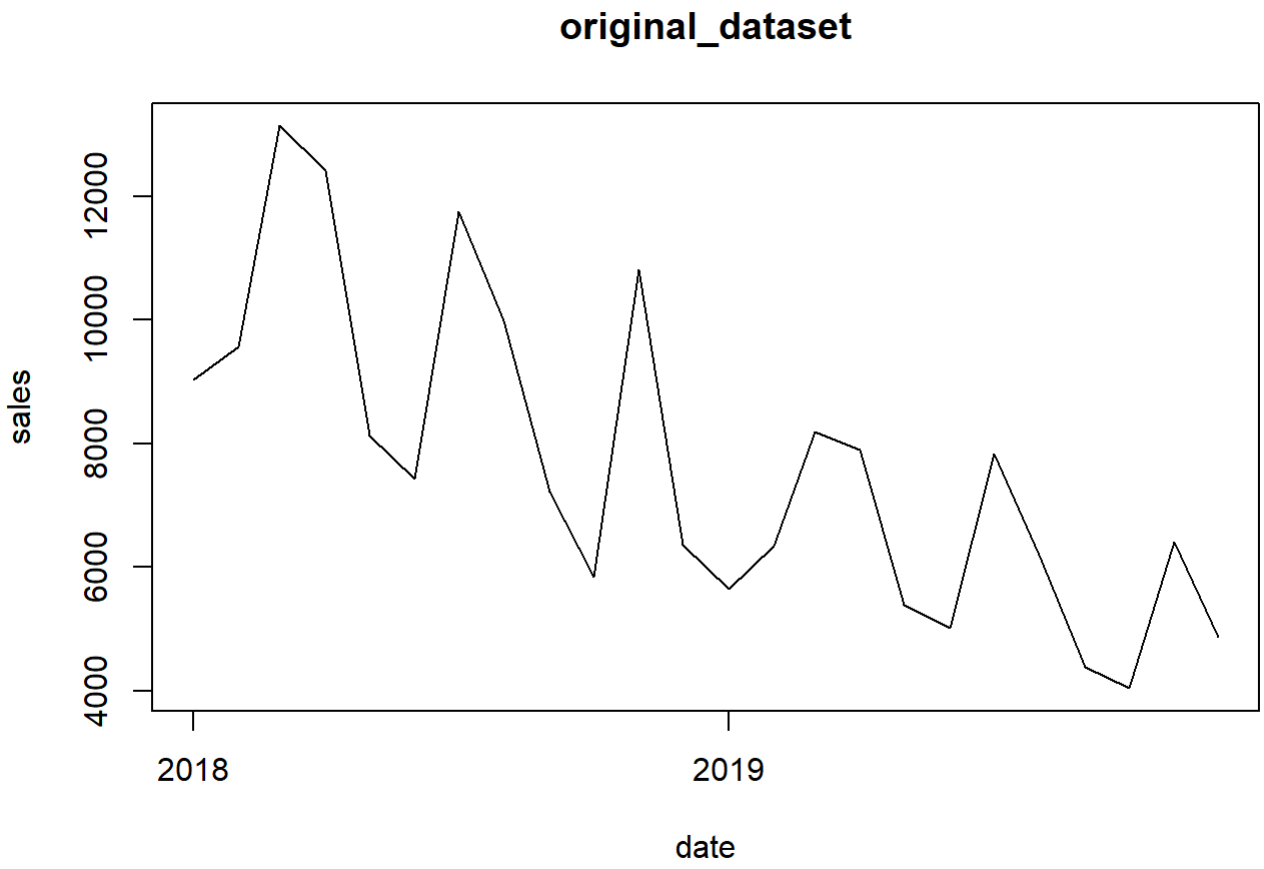
```
df_main = read_csv("C:/Users/sharv/Desktop/Time Series Project/Seasonal/monthlysales.csv",sho
w_col_types = FALSE)
str(df_main)
```

```
## spc_tbl_ [24 x 2] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ date : Date[1:24], format: "2018-01-01" "2018-02-01" ...
##  $ sales: num [1:24] 9035 9571 13143 12427 8133 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   date = col_date(format = ""),
##   ..   sales = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

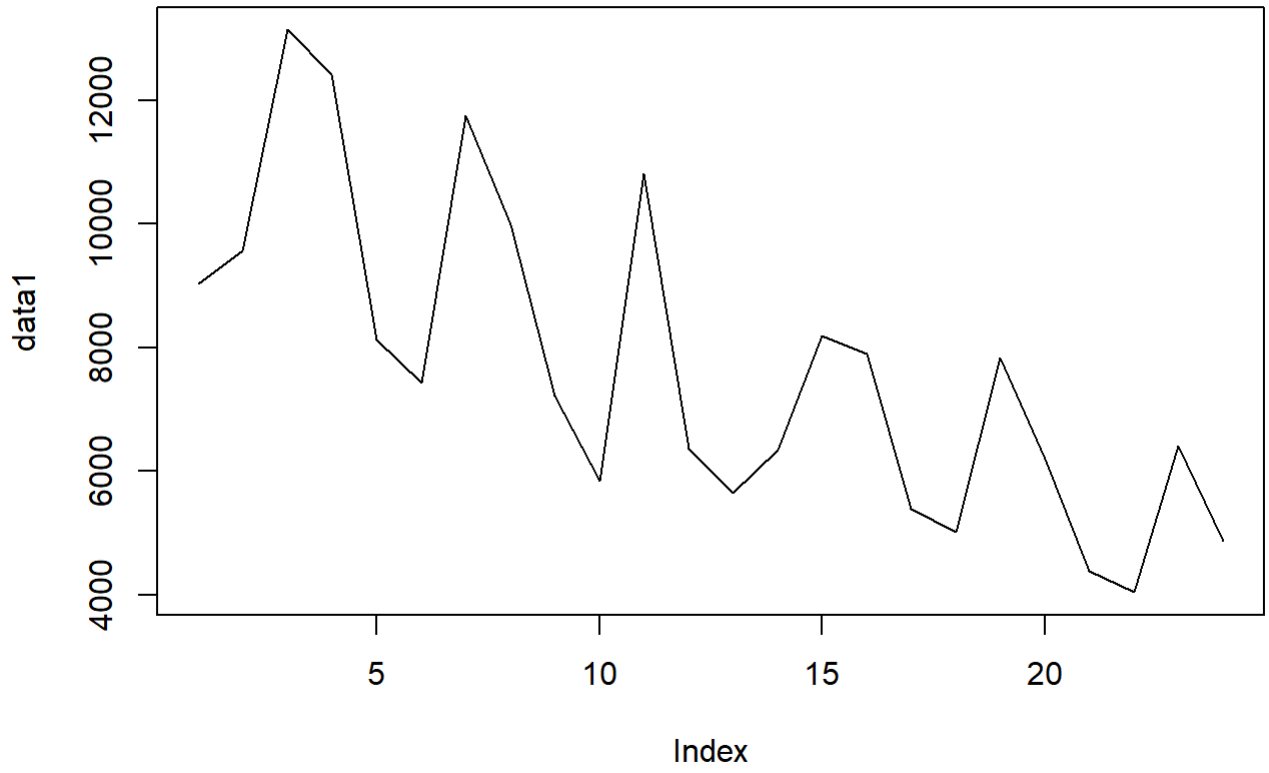
```
colnames(df_main) = c('date','sales')

df_main$date <- as.Date(df_main$date, "%d-%m-%Y")
df_main$date
```

```
##  [1] "2018-01-01" "2018-02-01" "2018-03-01" "2018-04-01" "2018-05-01"
##  [6] "2018-06-01" "2018-07-01" "2018-08-01" "2018-09-01" "2018-10-01"
## [11] "2018-11-01" "2018-12-01" "2019-01-01" "2019-02-01" "2019-03-01"
## [16] "2019-04-01" "2019-05-01" "2019-06-01" "2019-07-01" "2019-08-01"
## [21] "2019-09-01" "2019-10-01" "2019-11-01" "2019-12-01"
```

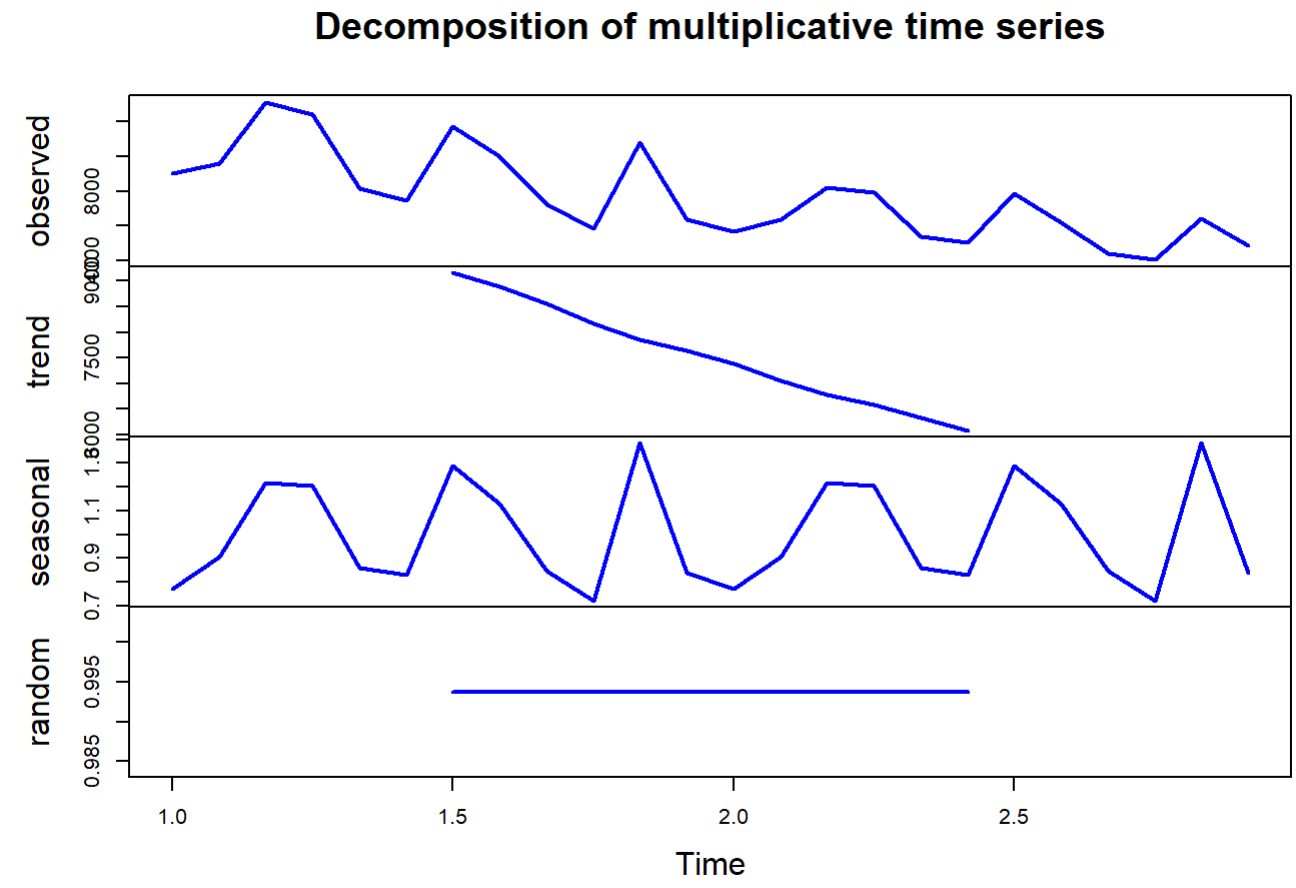
```
plot(df_main, main = 'original_dataset', type='l')
```




```
#################################################################################

start_date <- as.Date("2018-01-01")
end_date <- as.Date("2019-12-01")
last_date = as.Date("2019-12-01")
########################

#for prediction purpose: ## just next 36 month
##############################

## for model fitting purpose
df <- df_main[df_main$date >= start_date & df_main$date < end_date,]

df_test <- df_main[df_main$date >= end_date & df_main$date < as.Date("2020-03-01"),]

data1 = df_main$sales
data_365 = ts(data1, frequency = 12)
decompose_data = decompose(data_365, "multiplicative")
plot(data1, type='l')
```




```
plot(decompose_data, type='l',lwd=2, col = 'blue')
```



```
############################################################

############################################################

data1 = df_main$sales
data1
```

```
##  [1]  9035.00  9571.00 13143.00 12427.00  8133.00  7437.00 11758.00  9992.00
##  [9]  7216.00  5852.00 10813.00  6355.00  5644.35  6354.35  8188.55  7894.25
## [17]  5397.40  5012.80  7837.30  6207.00  4379.25  4043.45  6406.95  4867.25
```

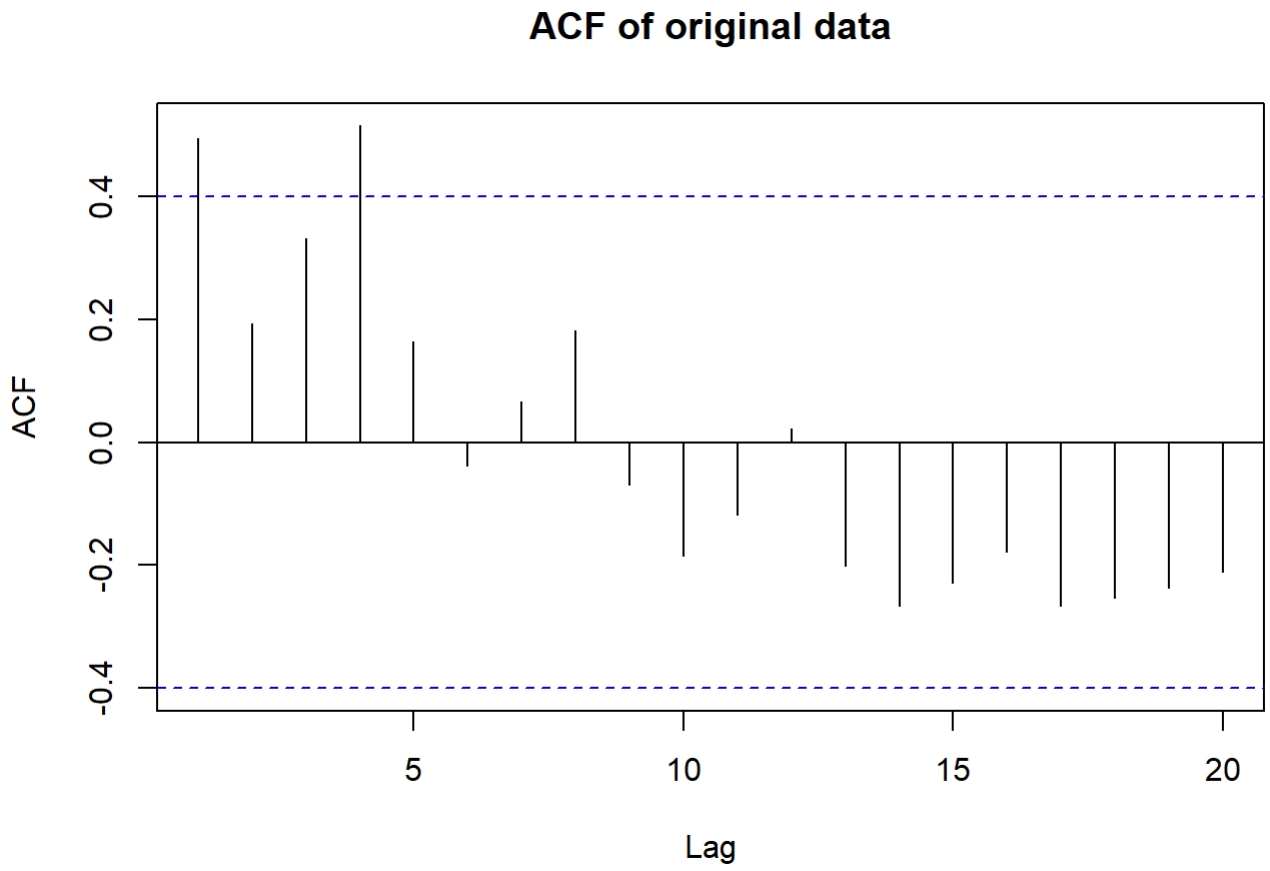
```
## spikes are there at start

## Performing dickey Fuller Test

adf_test = adf.test(data1)
```

```
## Warning in adf.test(data1): p-value smaller than printed p-value
```

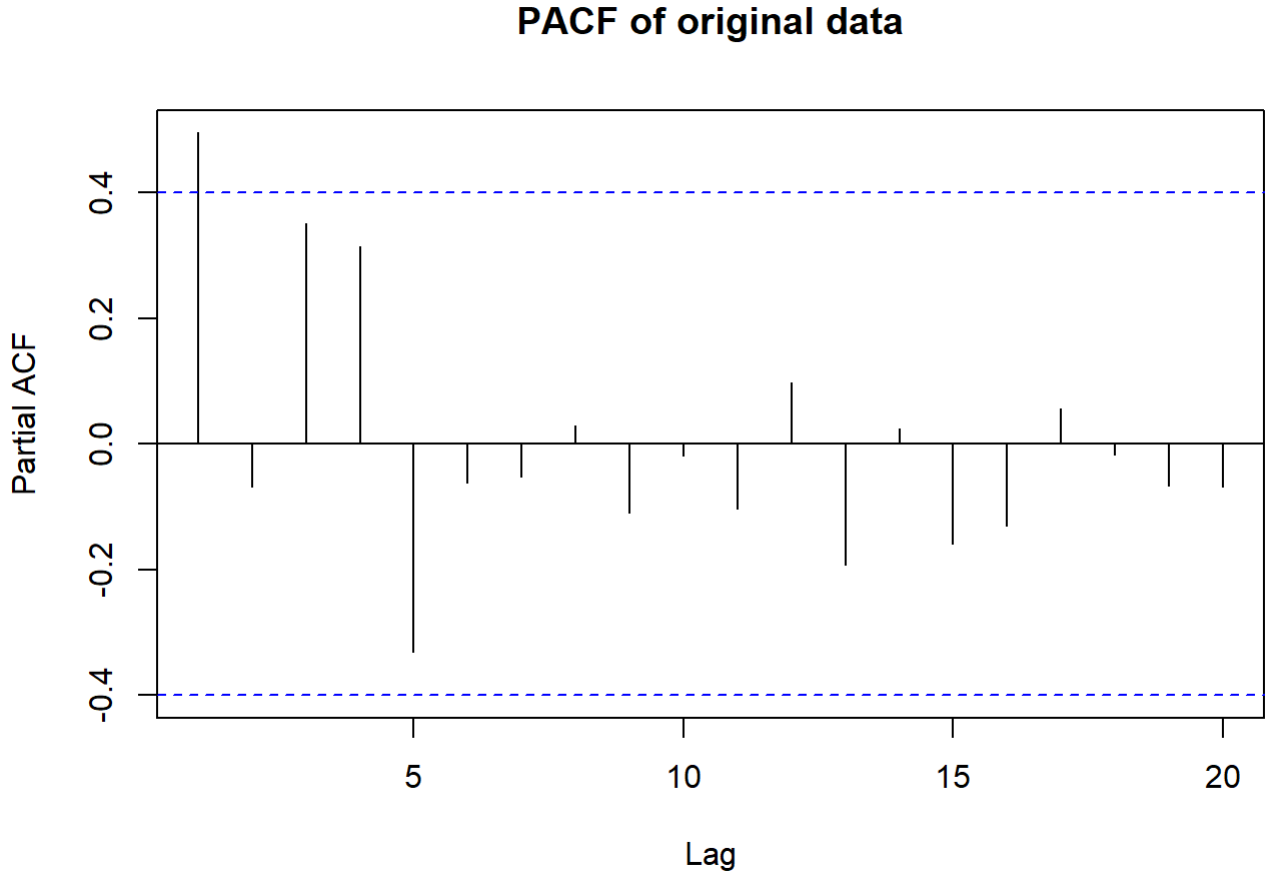
```
print(adf_test)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  data1
## Dickey-Fuller = -4.7502, Lag order = 2, p-value = 0.01
## alternative hypothesis: stationary
```

```
acf(data1, main = 'ACF of original data', lag.max = 20)
```




```
pacf(data1,main = 'PACF of original data', lag.max = 20 )
```



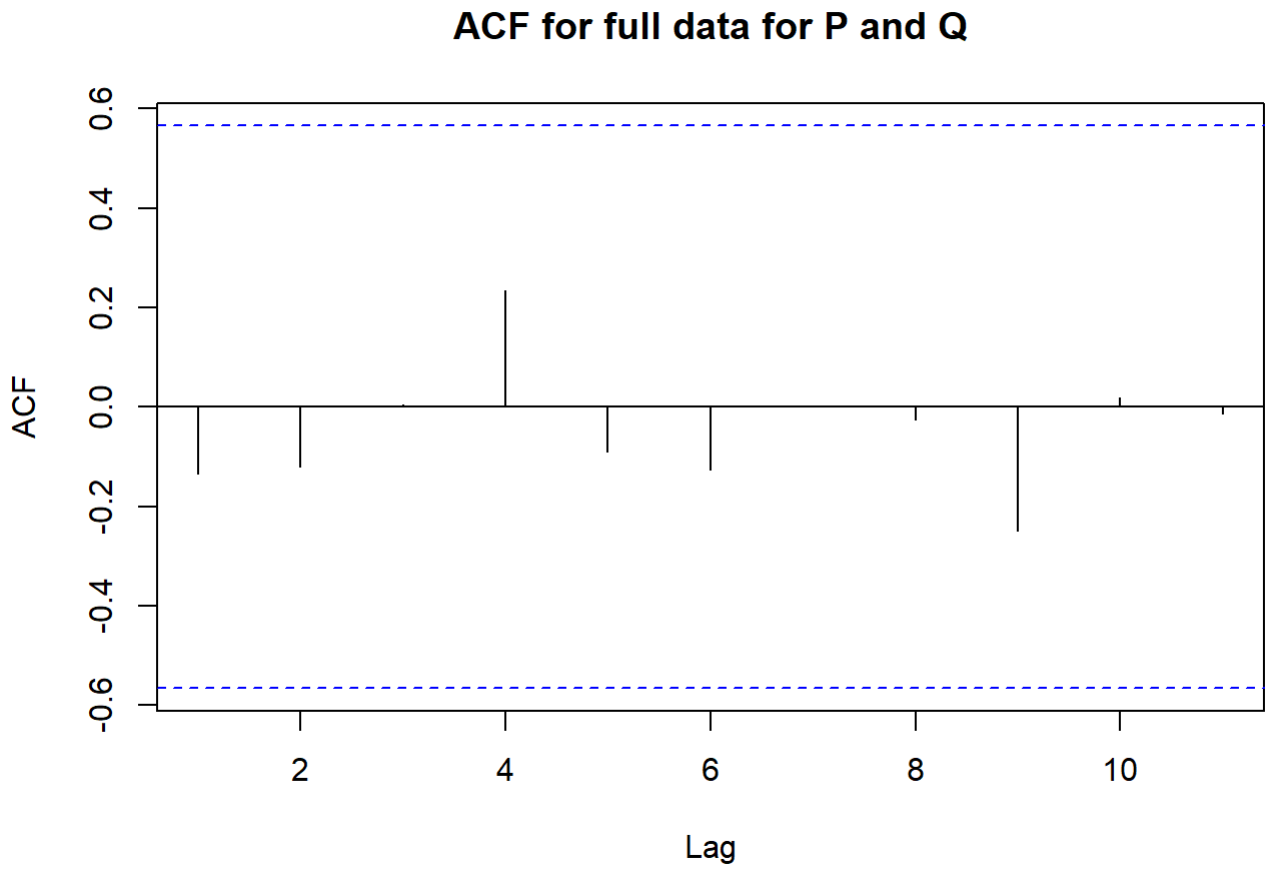
```
#########################################################################
#########################################################################

## full data with seasonal and first difference

close_sdiff = (diff(data1, lag = 12))
#plot(close_sdiff, type='l', main = 'seasonal log diff ')

adf_test = adf.test(close_sdiff)
```

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

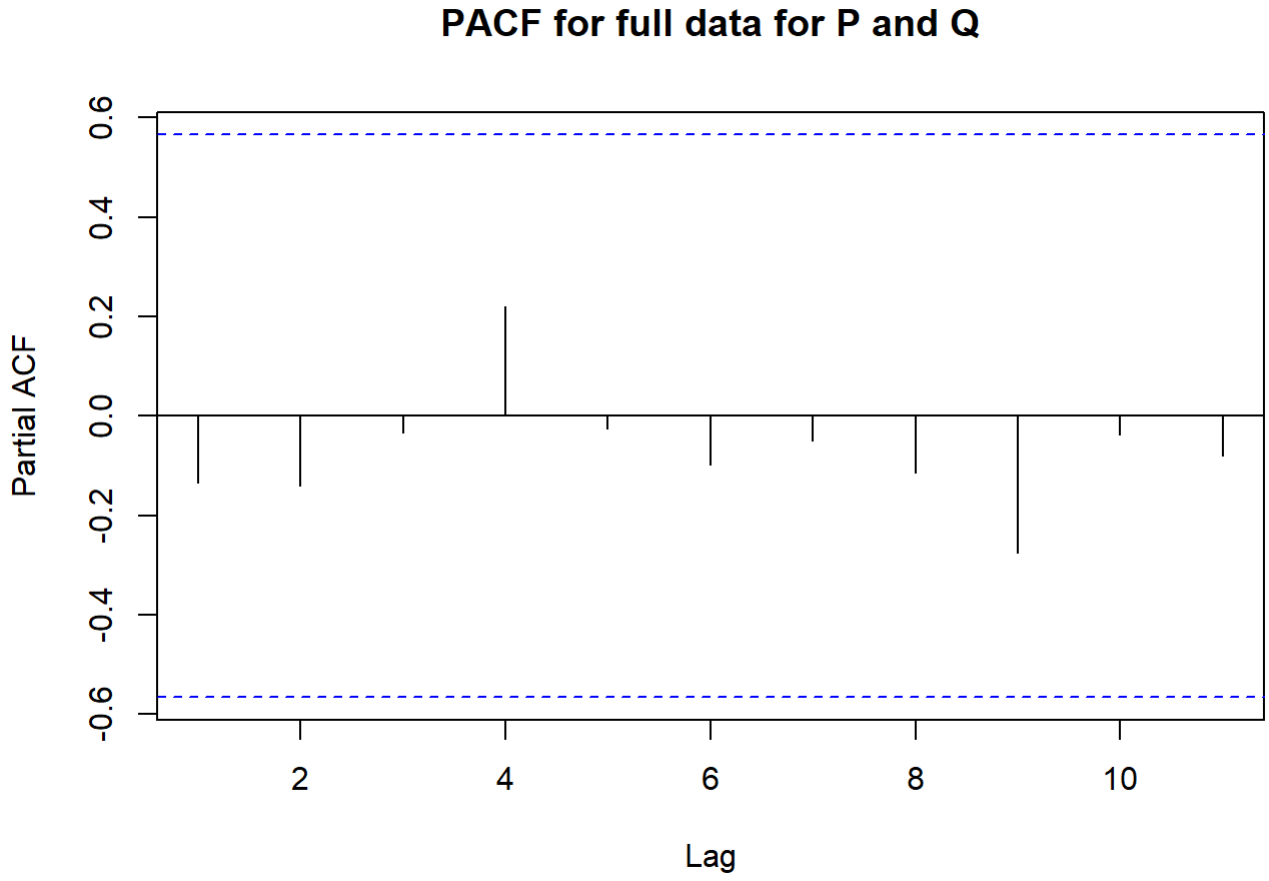
```
adf_test
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  close_sdiff
## Dickey-Fuller = -4.8455, Lag order = 2, p-value = 0.01
## alternative hypothesis: stationary
```

```
#################################################

## Model Determination:


## for the SARMA Model
acf(as.vector(close_sdiff), lag.max = 50, main = 'ACF for full data for P and Q')
```




```
pacf(as.vector(close_sdiff), lag.max = 50, main = 'PACF for full data for P and Q')
```



```
## from the ACF: its significant: so SMA(Q) = 0
## from the PACF: its significant: so SAR(P) = 0
# SARIMA (p,0,q)x(0,1,0) m = 12
```

```r
  {r}
# object to store ARIMA summary in
model_summary <- data.frame()

# loop to store
for(p in 0:4){
  for(q in 0:4){
      fit <- Arima(df_main$sales, order = c(p,0,q), seasonal = list(order = c(0,1,0), period = 12))

      s <- shapiro.test(rstandard(fit))

      # H0: The model does not show lack of fit
      # H1: not H0
      lb <- LB.test(fit, lag = 10)

      # gather everything into a single data frame
      # AIC, BIC, SHAPIRO OF RESIDUALS, LB TEST
      acc_ext <- data.frame(# arima order
        p,
        d=0,
        q,
        # goodness of fit
        LJUNG = lb$p.value,
        SHAPIRO = s$p.value,
        AIC = AIC(fit),
        BIC = BIC(fit)
      )

      # add ARIMA summary
      model_summary <- rbind(model_summary, acc_ext)
  }
}
```

6:1 (Top Level)

Source Visual Outline

```r
      )

      # add ARIMA summary
      model_summary <- rbind(model_summary, acc_ext)
  }
}

# show summary
filter(model_summary[order(model_summary$BIC, decreasing = FALSE),], LJUNG > 0.05)
```

Description: df [19 x 7]

p <int>	d <dbl>	q <int>	LJUNG <dbl>	SHAPIRO <dbl>	AIC <dbl>	BIC <dbl>
1	0	1	0.56441666	0.0144414550	212.3320	213.7868
1	0	0	0.28977794	0.0006515328	214.7967	215.7665
1	0	3	0.48748628	0.0037584308	213.4706	215.8952
2	0	1	0.54368878	0.0090166535	214.1572	216.0969
3	0	0	0.41025035	0.0025188468	214.4598	216.3994
2	0	0	0.38189733	0.0212868452	215.1305	216.5852
3	0	1	0.34907535	0.0016592925	215.0365	217.4610
1	0	4	0.39322547	0.0037260992	215.4026	218.3121
2	0	3	0.37643559	0.0040145857	215.4491	218.3586
2	0	2	0.39500417	0.0112277229	216.2670	218.6915

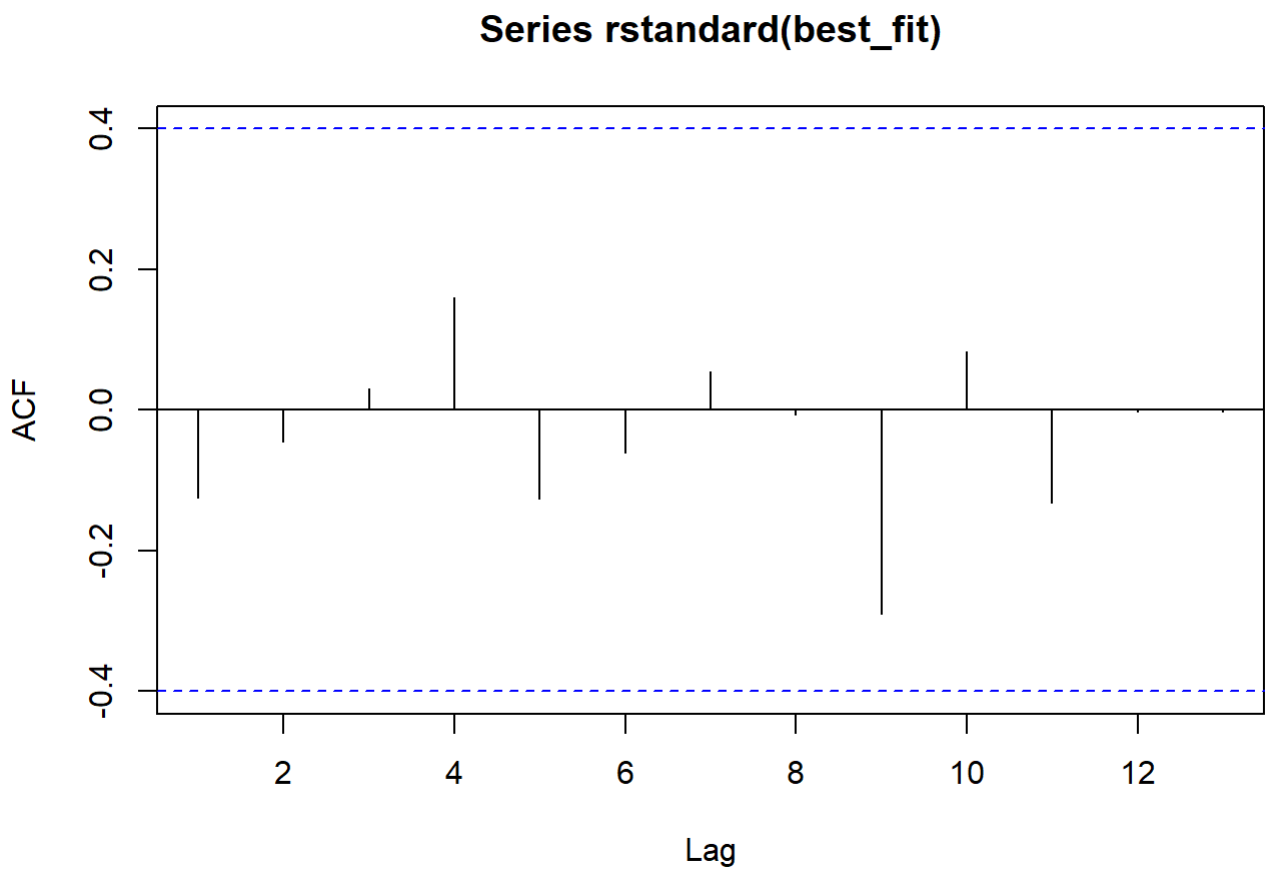
1-10 of 19 rows Previous 1 2 Next

6:1 (Top Level) R Markdown

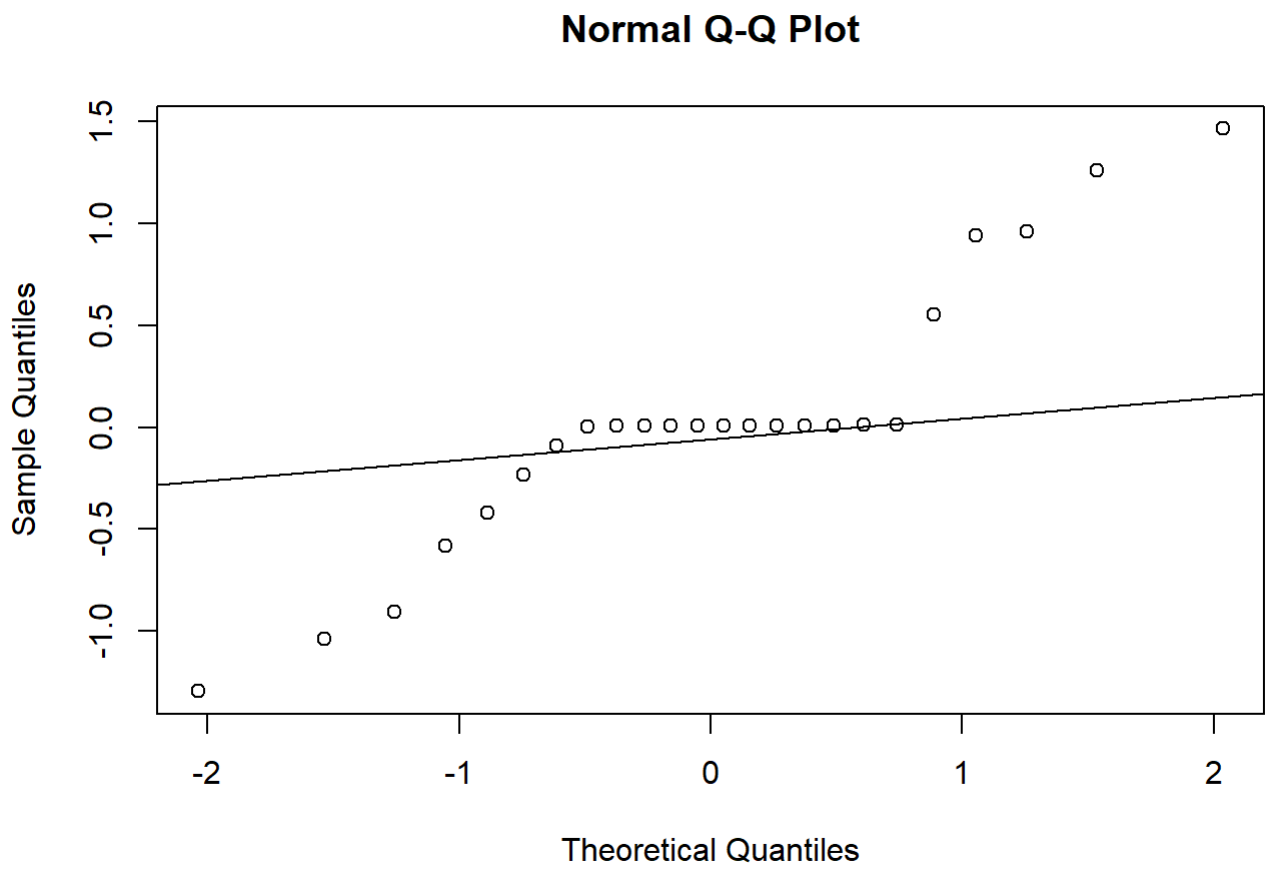
Console Terminal Render Jobs

```
## from the ACF: its significant: so SMA(Q) = 0
## from the PACF: its significant: so SAR(P) = 0
# SARIMA (p,0,q)x(0,1,0) m = 12
```

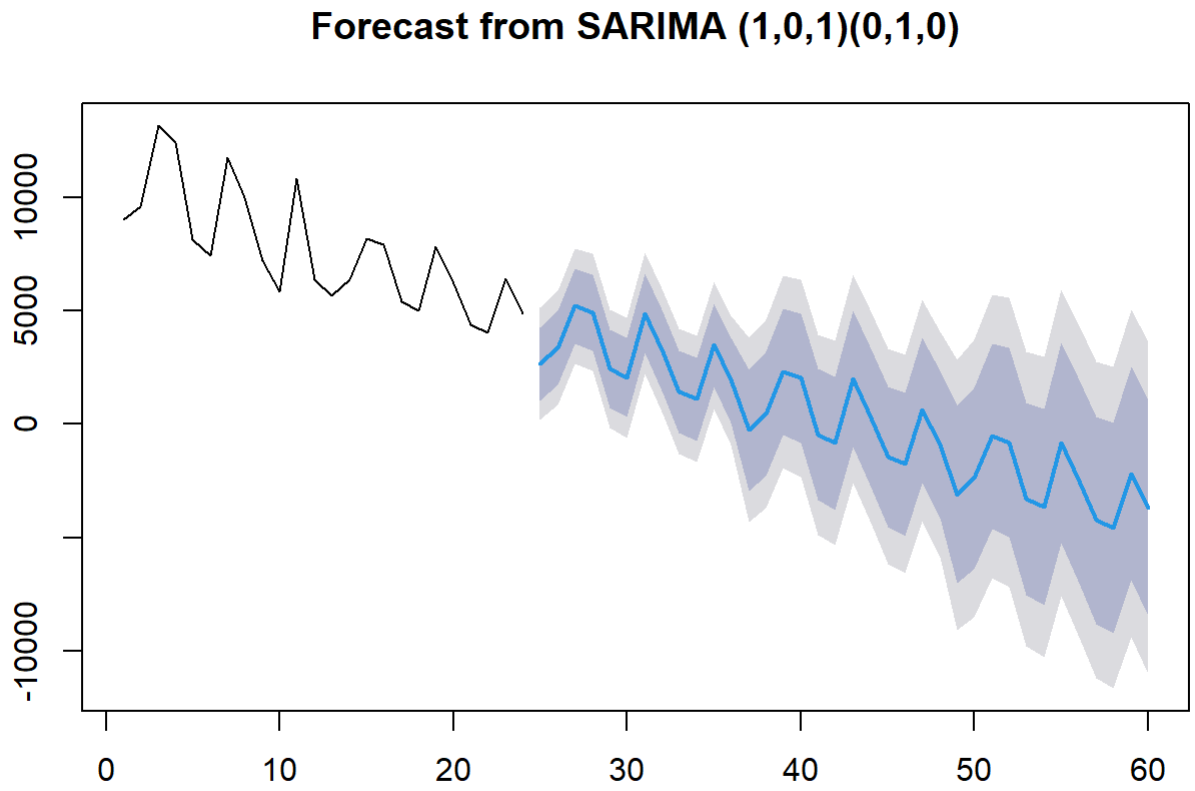
```
best_fit <- Arima(df_main$sales, order =  c(1,0,1), seasonal = list(order = c(0,1,0), period
= 12))

acf(rstandard(best_fit))
```




```
qqnorm(rstandard(best_fit))
qqline(rstandard(best_fit))
```




```
plot(forecast(best_fit, h = 36), main = 'Forecast from SARIMA (1,0,1)(0,1,0)')
```



sharven_NonSeasonal

2023-12-18

```
library(tseries)
```

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

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.1     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.3     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.1
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflic
ts to become errors
```

```
library(ggplot2)
library(stats)
library(forecast)
library(TSA)
```

```
## Registered S3 methods overwritten by 'TSA':
##   method       from
##   fitted.Arima forecast
##   plot.Arima   forecast
##
## Attaching package: 'TSA'
##
## The following object is masked from 'package:readr':
##
##     spec
##
## The following objects are masked from 'package:stats':
##
##     acf, arima
##
## The following object is masked from 'package:utils':
##
##     tar
```

```
library(urca)
library(FinTS)
```

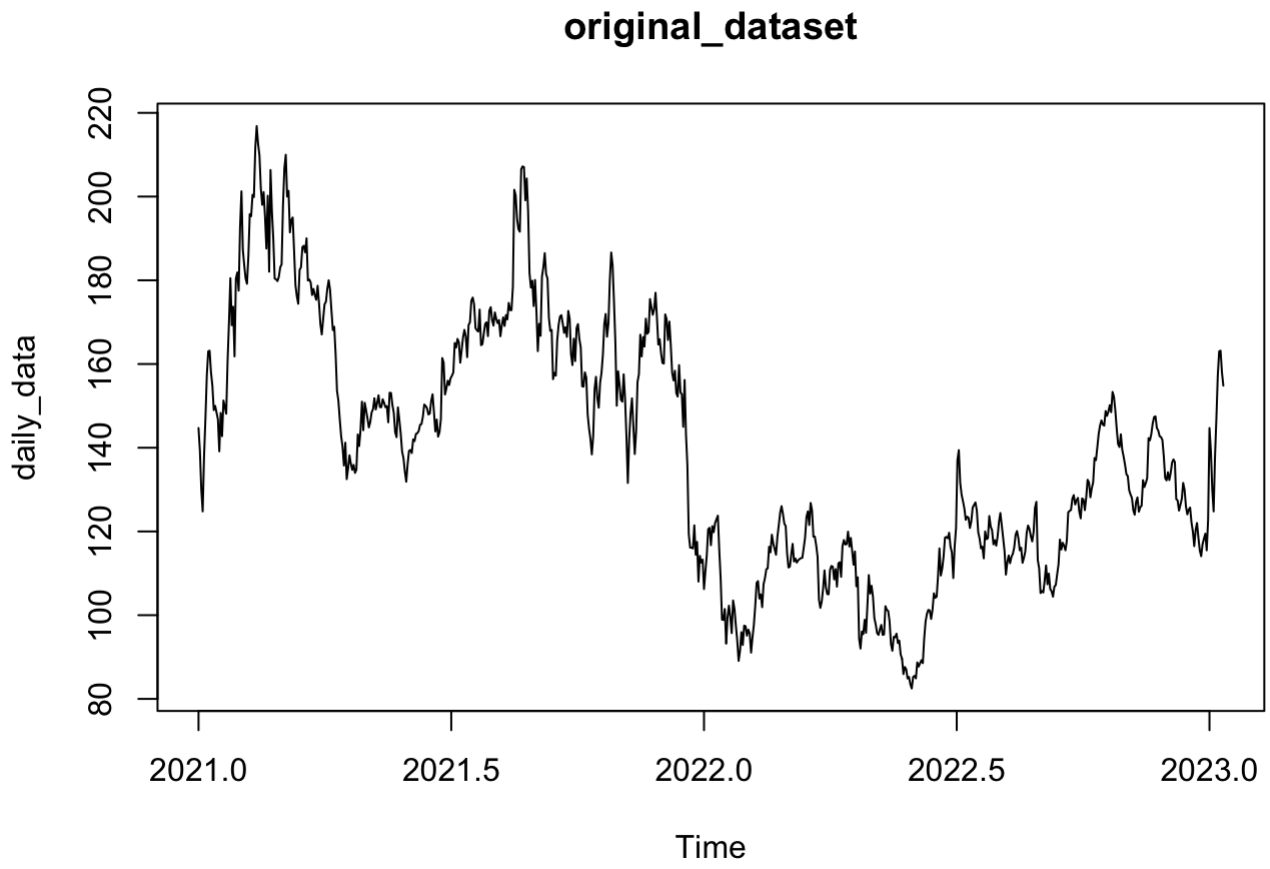
```
## Warning: package 'FinTS' was built under R version 4.2.3
```

```
## Loading required package: zoo
##
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
##
## Attaching package: 'FinTS'
##
## The following object is masked from 'package:forecast':
##
##     Acf
```

```
library(rugarch)
```

```
## Loading required package: parallel
##
## Attaching package: 'rugarch'
##
## The following object is masked from 'package:purrr':
##
##     reduce
##
## The following object is masked from 'package:stats':
##
##     sigma
```

```
df = read_csv("ABNB.csv",show_col_types = FALSE)
df = select(df, Date, Close)
df$Date <- as.Date(df$Date, "%Y-%m-%d")

# Create a time series with daily data from 2020-12-10 to 2023-11-03
daily_data <- ts(df$Close, start = c(2021,1), end = c(2023,11), frequency = 365)

plot(daily_data, main = 'original_dataset')
```




```
## Performing dickey Fuller Test

adf_test = adf.test(daily_data)
print(adf_test)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  daily_data
## Dickey-Fuller = -2.4251, Lag order = 9, p-value = 0.3984
## alternative hypothesis: stationary
```

```
## differencing

close_diff = diff(daily_data)

adf_test = adf.test(close_diff)
```

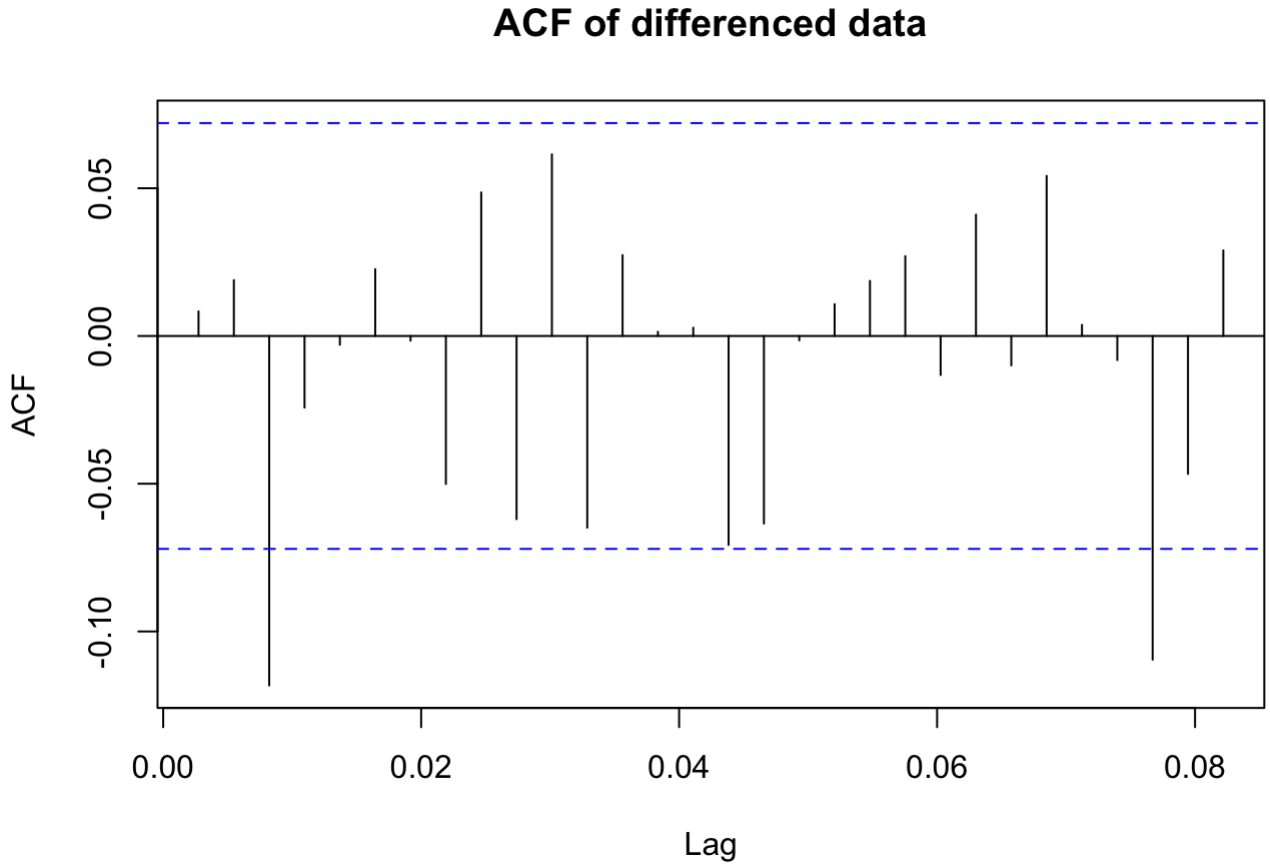
```
## Warning in adf.test(close_diff): p-value smaller than printed p-value
```

```
print(adf_test)
```

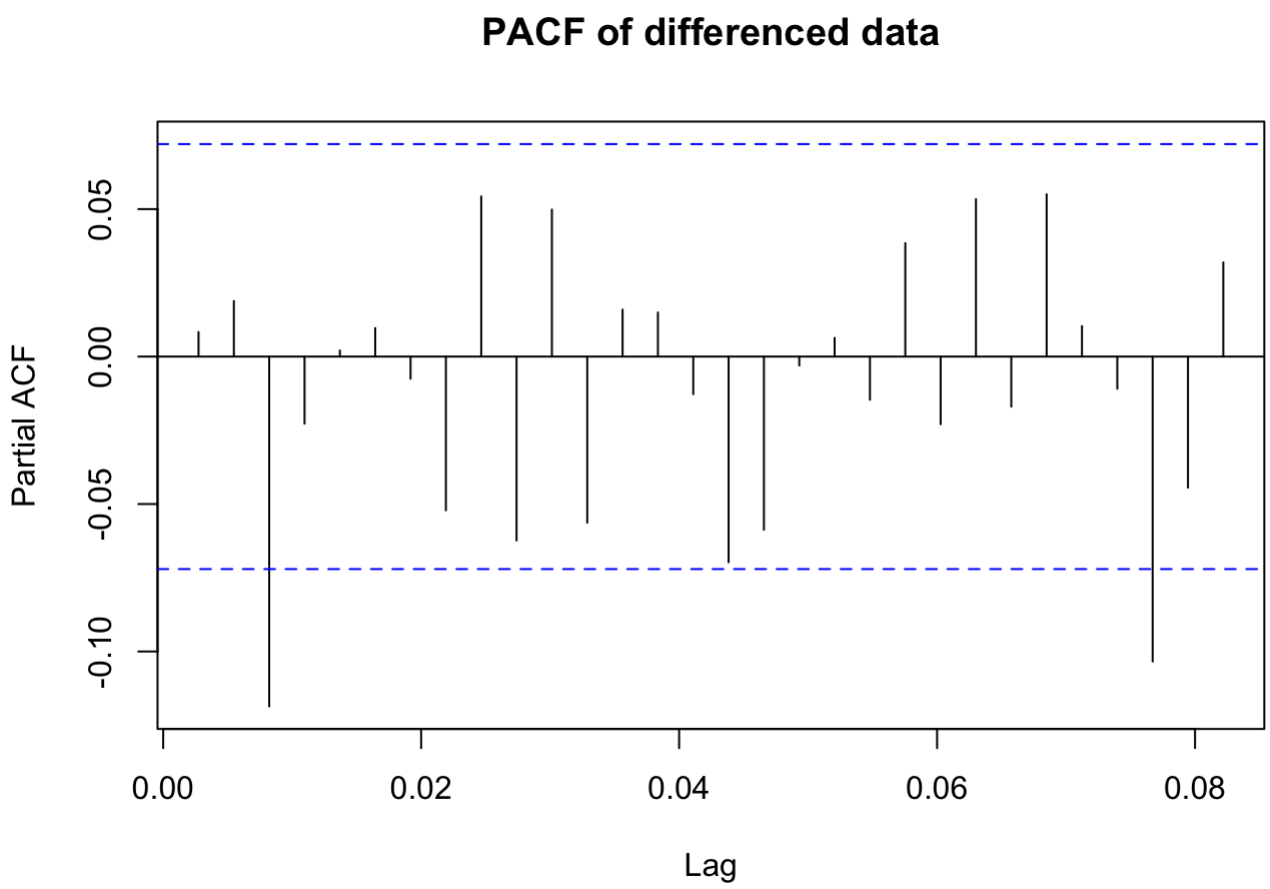
```
##
##  Augmented Dickey-Fuller Test
##
## data:  close_diff
## Dickey-Fuller = -9.2074, Lag order = 9, p-value = 0.01
## alternative hypothesis: stationary
```

```
## Means the data has become stationary
```

```
## acf and pacf plots

acf(close_diff, lag.max = 30, main = 'ACF of differenced data')
```




```
pacf(close_diff, lag.max = 30, main = 'PACF of differenced data')
```




```
eacf(close_diff)
```

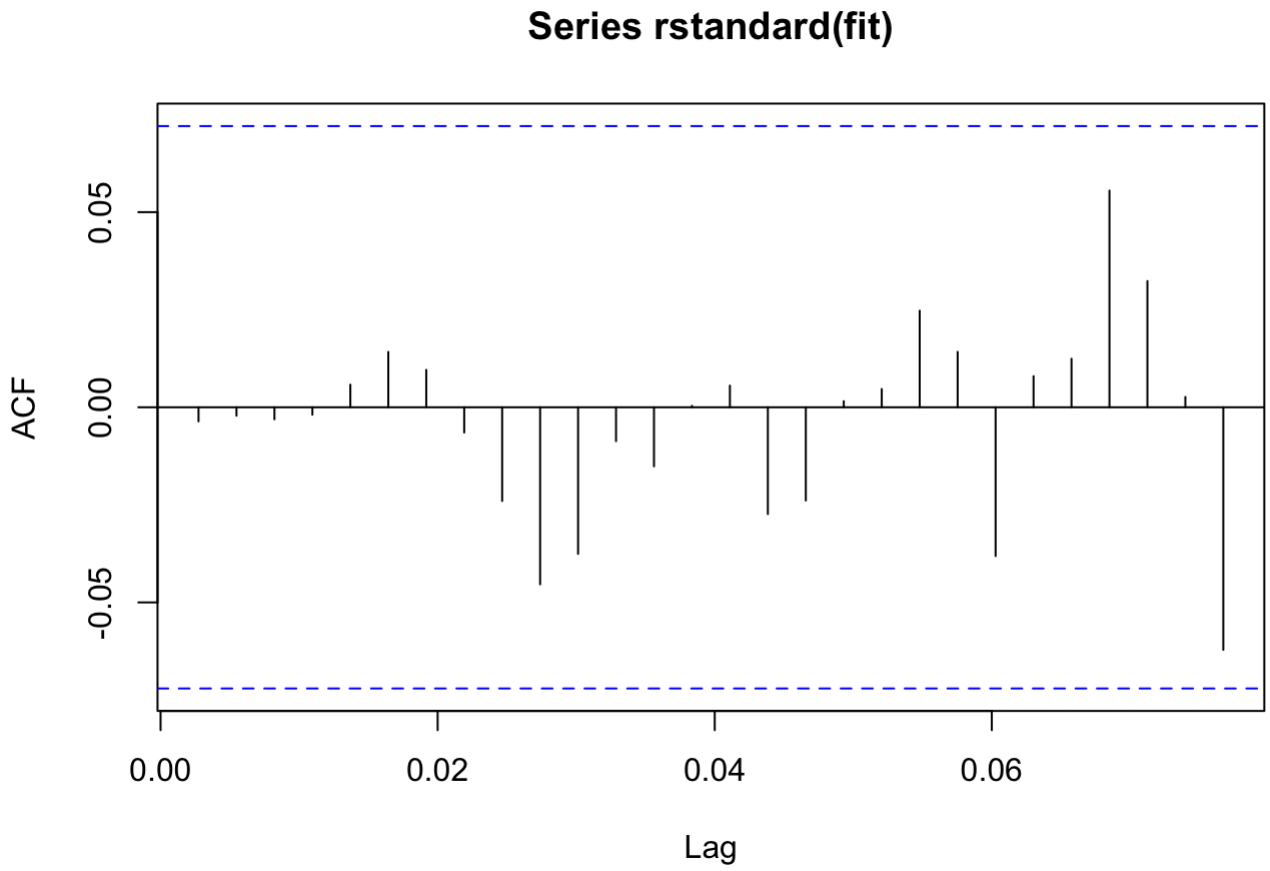
```
## AR/MA
##   0 1 2 3 4 5 6 7 8 9 10 11 12 13
## 0 o o x o o o o o o o o  o  o  o
## 1 x o x o o o o o o o o  o  o  o
## 2 x o x o o o o o o o o  o  o  o
## 3 x o x o o o o o o o o  o  o  o
## 4 x x x o o o o o o o o  o  o  o
## 5 x x o o x o o o o o o  o  o  o
## 6 x x x o x x o o o o o  o  o  o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

```
# object to store ARIMA summary in
model_summary <- data.frame()

# loop to store
for(p in 0:10){
  for(q in 0:10){
    for (d in 1:1) {
      fit <- Arima(close_diff, order = c(p,d,q))

      s <- shapiro.test(rstandard(fit))

      # H0: The model does not show lack of fit
      # H1: not H0
      lb <- LB.test(fit, lag = 50)

      # gather everything into a single data frame
      # AIC, BIC, SHAPIRO OF RESIDUALS, LB TEST
      acc_ext <- data.frame(# arima order
        p,
        d,
        q,
        # goodness of fit
        LJUNG = lb$p.value,
        SHAPIRO = s$p.value,
        AIC = AIC(fit),
        BIC = BIC(fit)
      )

      # add ARIMA summary
      model_summary <- rbind(model_summary, acc_ext)
    }
  }
}

# show summary
filter(model_summary[order(model_summary$BIC, decreasing = FALSE),], LJUNG > 0.05)
```

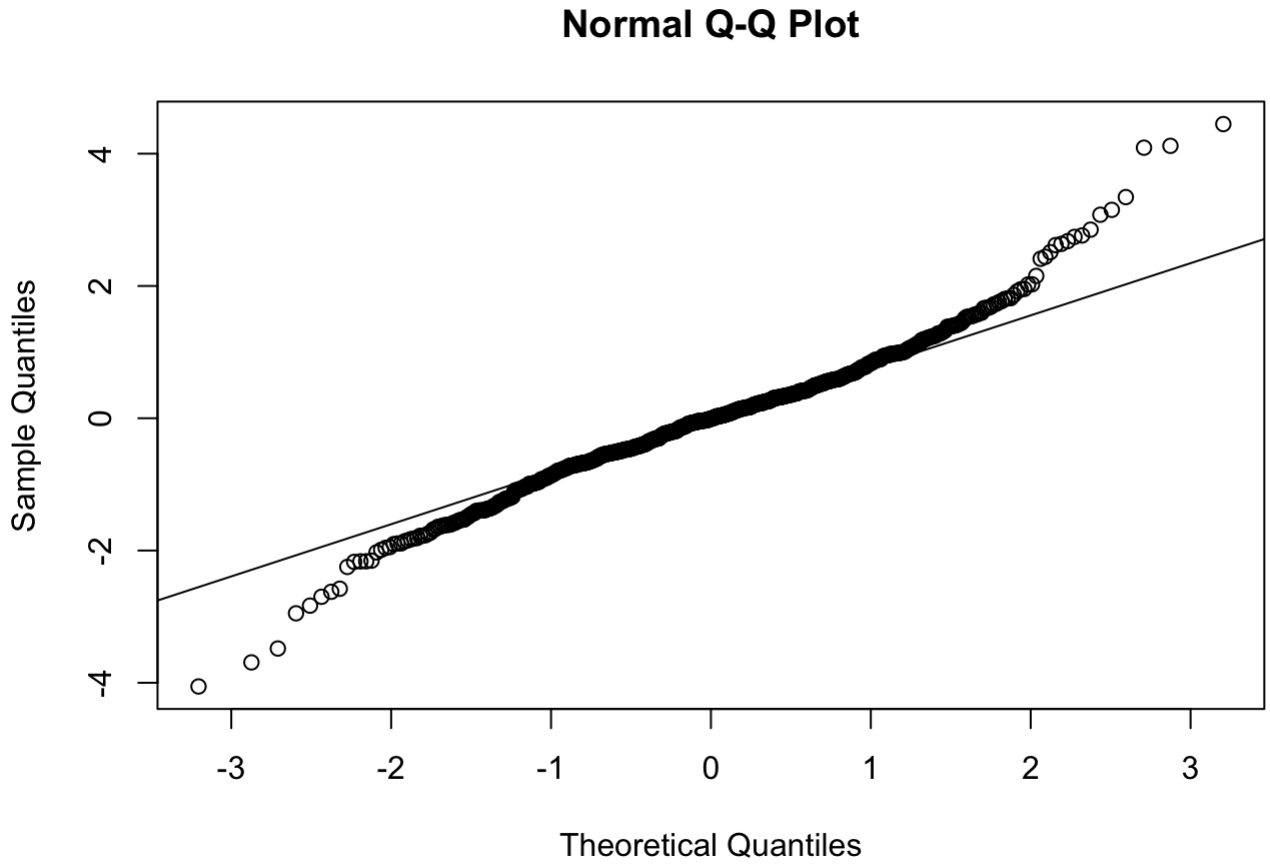
```
##     p d  q      LJUNG      SHAPIRO      AIC      BIC
## 1   3 1  1 0.09285176 4.245802e-11 4483.048 4506.074
## 2   0 1  4 0.07478170 4.795130e-11 4483.611 4506.638
## 3   4 1  1 0.08962169 3.543882e-11 4484.688 4512.320
## 4   3 1  2 0.08922201 3.540031e-11 4484.728 4512.360
## 5   0 1  5 0.07676896 3.785268e-11 4485.014 4512.646
## 6   1 1  4 0.07811686 3.791272e-11 4485.032 4512.664
## 7   1 1  5 0.08147314 5.053558e-11 4483.844 4516.081
## 8   3 1  3 0.07221553 3.785343e-11 4486.589 4518.826
## 9   5 1  1 0.07309234 3.703949e-11 4486.676 4518.913
## 10  2 1  4 0.06168141 3.449487e-11 4486.804 4519.041
## 11  4 1  2 0.07154565 3.726936e-11 4486.845 4519.082
## 12  0 1  6 0.06783736 3.468040e-11 4486.953 4519.190
## 13  5 1  2 0.10409483 6.903206e-11 4483.659 4520.502
## 14  1 1  6 0.07176411 4.740399e-11 4485.799 4522.642
## 15  0 1  7 0.07938398 2.923375e-11 4487.916 4524.758
## 16  3 1  5 0.09307343 5.714049e-11 4483.595 4525.043
## 17  3 1  4 0.06007479 3.642028e-11 4488.556 4525.398
## 18  6 1  1 0.06207116 3.291401e-11 4488.557 4525.400
## 19  4 1  3 0.06000520 3.524337e-11 4488.620 4525.462
## 20  2 1  5 0.05205966 3.484667e-11 4488.761 4525.603
## 21  2 1  6 0.06880340 1.157574e-10 4484.470 4525.918
## 22  5 1  3 0.09790765 5.899007e-11 4485.702 4527.149
## 23  1 1  7 0.07409230 3.563709e-11 4487.142 4528.589
## 24  0 1  8 0.06625195 3.690047e-11 4489.774 4531.222
## 25  1 1  8 0.12953423 2.111187e-11 4485.572 4531.625
## 26  4 1  6 0.24315303 5.610546e-11 4481.207 4531.865
## 27  3 1  6 0.10805570 4.343087e-11 4486.228 4532.281
## 28  2 1  7 0.06109112 1.203120e-10 4486.431 4532.484
## 29  5 1  4 0.07721104 7.501725e-11 4486.621 4532.674
## 30  8 1  2 0.14447996 5.045331e-11 4483.909 4534.568
## 31  7 1  3 0.12808250 3.858064e-11 4484.753 4535.412
## 32  1 1  9 0.14251778 3.744409e-11 4484.758 4535.417
## 33  3 1  7 0.07151408 1.174486e-10 4485.124 4535.782
## 34  8 1  1 0.06615585 3.479322e-11 4490.673 4536.726
## 35  0 1  9 0.06183305 3.172578e-11 4490.784 4536.837
## 36  5 1  5 0.07846080 5.646590e-11 4486.987 4537.645
## 37  5 1  7 0.22209675 5.545504e-11 4478.970 4538.839
## 38  9 1  1 0.06085732 2.828985e-11 4490.341 4541.000
## 39  9 1  2 0.11796089 4.945853e-11 4485.750 4541.014
## 40  8 1  3 0.11913329 5.020562e-11 4485.761 4541.024
## 41  1 1 10 0.12562097 3.689833e-11 4486.112 4541.375
## 42  6 1  4 0.05450466 2.395161e-11 4491.218 4541.877
## 43  3 1  8 0.05413515 1.279467e-10 4487.032 4542.296
## 44  6 1  7 0.11373080 9.024662e-11 4480.321 4544.795
## 45 10 1  1 0.08286922 4.451654e-11 4489.547 4544.811
## 46  3 1  9 0.14192975 6.430753e-11 4485.893 4545.762
## 47  2 1 10 0.13315459 1.511014e-10 4486.537 4546.406
## 48  8 1  4 0.11551887 5.250367e-11 4487.323 4547.192
## 49  7 1  5 0.07909107 1.459998e-10 4487.386 4547.255
## 50  4 1  8 0.15539489 7.003329e-11 4487.560 4547.429
## 51  9 1  3 0.10545546 5.014943e-11 4487.650 4547.518
## 52 10 1  2 0.09922591 5.012357e-11 4487.734 4547.603
## 53  9 1  4 0.08748625 6.345799e-11 4483.636 4548.110
## 54  4 1  9 0.13476990 1.010476e-10 4487.108 4551.582
```

```
## 55  8 1  5 0.12189753 6.009125e-11 4487.640 4552.114
## 56  3 1 10 0.09414121 6.601000e-11 4487.877 4552.352
## 57  8 1  7 0.07628510 4.963196e-10 4479.544 4553.229
## 58  9 1  5 0.07407188 8.575068e-11 4484.371 4553.450
## 59 10 1  3 0.08204493 5.065010e-11 4489.719 4554.193
## 60  8 1  6 0.11035846 1.061958e-10 4485.661 4554.741
## 61  5 1  9 0.13984776 1.111378e-10 4485.906 4554.986
## 62  5 1  8 0.07853078 1.032066e-10 4491.084 4555.558
## 63  7 1 10 0.12547430 2.556878e-09 4472.763 4555.658
## 64  4 1 10 0.15476013 7.699488e-11 4488.800 4557.879
## 65  7 1  9 0.07377151 7.553209e-10 4479.925 4558.216
## 66  9 1  6 0.19479064 1.293435e-10 4484.879 4558.564
## 67 10 1  4 0.07132290 4.378940e-11 4489.801 4558.881
## 68  6 1  9 0.14421812 1.643574e-10 4485.209 4558.893
## 69  5 1 10 0.17447332 1.422230e-10 4485.241 4558.926
## 70  7 1  8 0.06927986 2.318904e-10 4486.374 4560.059
## 71 10 1  5 0.06880973 1.007860e-10 4489.813 4563.497
## 72  9 1  7 0.06604445 1.177317e-10 4485.739 4564.030
## 73  8 1  8 0.13478534 1.568139e-10 4486.950 4565.240
## 74 10 1  6 0.17840685 1.228180e-10 4487.147 4565.437
## 75  6 1 10 0.13783909 1.248676e-10 4487.530 4565.820
## 76  8 1  9 0.08469126 2.666842e-10 4483.088 4565.983
## 77 10 1  7 0.16540259 1.926140e-10 4484.888 4567.783
## 78 10 1  8 0.16015884 2.842848e-10 4484.231 4571.732
## 79  9 1  8 0.09167674 1.388047e-10 4490.274 4573.169
## 80  9 1  9 0.13940325 7.426714e-10 4486.415 4573.916
## 81 10 1  9 0.06225180 4.167657e-10 4483.037 4575.143
## 82  9 1 10 0.05217681 1.288580e-10 4488.542 4580.647
## 83 10 1 10 0.08118358 6.097353e-10 4485.375 4582.086
```

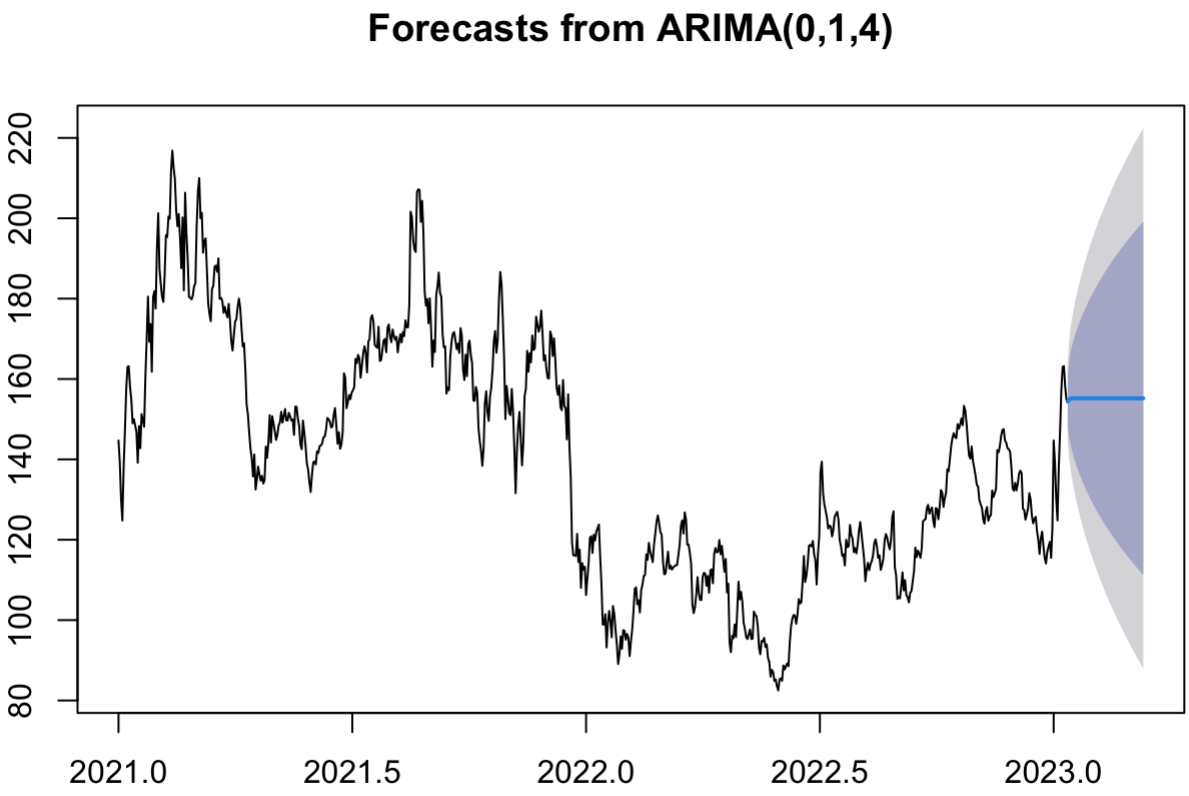
```
## WE choose (0,1,4) Model
best_fit <- Arima(daily_data, order = c(0,1,4))

acf(rstandard(fit))
```




```
qqnorm(rstandard(fit))
qqline(rstandard(fit))
```



```
plot(forecast(best_fit, h = 60))
```




```
#GARCH MODELLING

# we now need to fit either abs(return) or sqaure(return)


#GARCH - square(return)
print(daily_data)
```

```
## Time Series:
## Start = c(2021, 1)
## End = c(2023, 11)
## Frequency = 365
##   [1] 144.710 139.250 130.000 124.800 137.990 147.050 157.300 163.020 163.190
##  [10] 158.010 154.840 149.000 150.000 148.430 146.800 139.150 148.300 142.770
##  [19] 151.270 149.770 148.130 160.800 169.990 180.500 169.270 173.690 161.830
##  [28] 180.400 181.870 177.530 192.740 201.250 187.370 183.630 180.440 179.170
##  [37] 185.720 195.800 195.310 200.430 199.880 211.660 216.840 212.680 209.860
##  [46] 201.960 198.040 201.070 195.340 187.590 200.200 182.060 206.350 196.420
##  [55] 189.900 180.400 180.230 179.810 180.810 183.110 183.790 197.870 206.740
##  [64] 209.990 200.010 201.360 191.450 194.390 195.000 187.140 178.850 176.160
##  [73] 174.400 182.500 183.100 187.940 188.240 186.690 190.030 179.940 180.180
##  [82] 179.500 176.490 177.950 176.430 175.350 178.690 174.580 169.570 167.070
##  [91] 170.710 174.250 174.880 177.940 180.000 177.680 172.710 168.110 168.900
## [100] 162.330 153.640 151.210 146.730 142.730 140.250 135.750 141.200 132.500
## [109] 135.020 138.190 136.200 134.710 135.910 133.990 134.750 143.170 140.400
## [118] 144.310 151.000 144.190 150.730 148.970 147.010 144.850 146.120 148.440
## [127] 149.210 151.780 149.150 150.700 152.520 149.700 149.680 151.580 150.730
## [136] 149.670 149.940 146.080 153.140 153.080 150.230 148.370 143.720 142.530
## [145] 149.640 146.690 143.410 139.090 137.500 134.310 131.880 136.090 139.250
## [154] 139.470 138.730 142.000 141.580 143.300 143.470 144.010 145.490 145.650
## [163] 147.400 150.320 149.990 149.440 147.950 148.160 151.150 152.760 148.570
## [172] 143.900 146.740 142.650 143.700 146.790 161.420 160.350 152.730 154.180
## [181] 156.020 154.990 156.590 157.200 158.000 165.000 163.930 166.000 165.200
## [190] 160.320 163.300 166.370 168.150 166.590 161.640 169.290 169.960 175.130
## [199] 175.880 174.260 168.580 168.070 167.750 173.010 164.500 164.740 167.250
## [208] 169.600 169.970 166.670 172.750 173.580 170.500 169.180 172.320 170.740
## [217] 169.760 170.500 166.640 169.240 171.140 169.100 171.700 170.660 174.600
## [226] 172.870 172.870 178.450 201.620 200.320 194.680 192.220 191.610 206.540
## [235] 207.210 207.040 199.110 204.330 196.420 181.730 178.270 179.890 173.860
## [244] 180.080 172.540 163.080 169.600 166.750 180.850 183.140 186.490 181.460
## [253] 180.420 171.040 167.960 168.140 156.380 157.910 157.230 165.660 169.290
## [262] 171.310 171.680 169.710 167.440 168.780 166.490 172.680 170.800 162.250
## [271] 159.750 166.050 160.710 168.610 169.540 166.000 163.990 154.690 154.580
## [280] 158.000 156.730 147.900 144.560 142.140 138.410 142.770 153.970 156.940
## [289] 151.760 149.570 155.560 157.910 162.260 169.530 171.950 166.530 169.660
## [298] 180.070 186.640 183.600 174.900 165.240 150.040 158.260 155.090 151.490
## [307] 151.010 157.530 151.690 142.700 131.590 142.130 148.310 151.800 145.140
## [316] 138.500 143.950 155.750 157.550 167.000 161.800 166.440 164.160 170.830
## [325] 167.220 167.650 175.520 173.630 171.760 173.070 177.020 171.210 164.660
## [334] 165.910 162.560 160.250 160.110 171.850 170.700 165.740 170.120 164.550
## [343] 157.910 156.090 158.390 153.040 152.230 159.740 153.210 152.780 145.000
## [352] 156.180 143.090 135.840 119.370 116.130 116.150 115.940 121.450 114.440
## [361] 117.500 108.030 114.170 112.550 113.280 106.240 110.400 114.300 120.500
## [370] 120.870 116.720 121.260 119.830 122.020 122.900 123.770 115.720 108.910
## [379]  98.930  98.870 101.470  93.260  99.490 102.270  99.530  95.720 103.510
## [388] 101.500  97.530  93.930  89.080  91.410  95.940  92.880  97.500  97.350
## [397]  95.100  96.550  95.640  91.050  94.660  97.670 102.200 107.730 108.140
## [406] 103.970 104.950 101.910 107.360 108.840 110.980 111.200 116.340 115.020
## [415] 119.220 117.110 115.820 114.440 118.730 121.500 124.510 126.040 124.180
## [424] 121.870 121.270 114.760 111.390 111.600 114.510 117.030 112.820 113.540
## [433] 112.560 113.120 113.400 113.640 113.700 116.070 118.580 123.480 124.790
## [442] 121.540 126.800 125.040 118.750 118.780 116.710 114.010 103.720 101.750
## [451] 103.230 106.370 110.690 106.660 105.040 105.000 110.810 111.760 111.580
```

```
## [460] 108.540 110.990 106.820 112.350 112.650 109.160 116.500 117.940 116.870
## [469] 117.000 119.950 116.390 118.450 115.310 112.060 115.210 106.910 109.050
## [478]  94.410  92.020  96.090  95.460  98.900  95.790 102.400 109.570 105.160
## [487] 107.010 104.430  99.300  97.770  95.710  95.280  96.630  97.670  95.300
## [496]  95.380 102.140 101.270 101.000  98.510  93.120  91.500  94.830  94.700
## [505]  95.580  93.340  93.930  90.610  89.570  85.930  87.620  87.070  84.870
## [514]  85.250  83.490  82.490  85.230  85.500  84.900  88.720  87.710  88.520
## [523]  89.240  88.540  94.440  98.490 100.370 101.270 101.110  99.100 101.280
## [532] 105.220 104.130 104.440 109.420 115.940 109.480 111.110 113.990 118.520
## [541] 118.700 118.350 119.690 116.360 114.940 108.870 116.420 120.870 137.010
## [550] 139.420 131.600 128.780 127.210 125.520 122.780 123.530 123.280 120.830
## [559] 122.380 125.730 126.330 126.940 125.000 119.840 118.300 115.960 116.360
## [568] 113.580 119.990 118.150 118.400 123.690 121.170 120.300 116.920 117.910
## [577] 116.610 118.800 122.280 124.400 121.460 118.460 115.340 109.690 112.600
## [586] 114.250 112.420 113.950 114.650 116.170 119.190 120.100 118.400 115.500
## [595] 116.120 112.505 113.680 115.250 119.670 121.420 120.580 118.860 117.630
## [604] 119.900 125.650 127.070 113.190 111.200 105.275 105.780 105.410 108.330
## [613] 111.870 107.380 109.930 106.250 105.710 104.420 106.810 107.190 109.770
## [622] 112.160 118.060 115.690 117.300 116.550 115.500 117.860 124.590 124.900
## [631] 125.140 127.850 128.680 126.450 127.460 127.960 124.730 123.130 127.890
## [640] 127.630 125.100 128.160 132.350 131.690 128.160 130.320 131.710 137.540
## [649] 137.020 140.090 143.340 145.360 146.530 145.535 145.280 148.770 147.620
## [658] 148.650 150.170 148.500 153.330 152.190 148.910 144.560 140.880 140.170
## [667] 143.200 139.550 137.870 135.970 133.680 133.240 129.850 128.830 127.990
## [676] 125.060 124.000 127.080 128.130 124.720 125.790 126.155 132.250 130.610
## [685] 131.550 132.690 142.290 141.770 143.270 145.820 147.330 147.500 144.715
## [694] 144.120 142.750 142.550 141.850 138.010 132.750 132.200 134.140 132.280
## [703] 134.030 136.470 137.210 136.560 127.730 127.410 124.990 126.360 127.770
## [712] 131.590 130.000 125.970 124.080 125.190 125.710 122.170 119.970 116.490
## [721] 120.390 122.010 118.500 115.470 114.090 116.950 118.290 119.470 115.500
## [730] 122.640 144.710 139.250 130.000 124.800 137.990 147.050 157.300 163.020
## [739] 163.190 158.010 154.840
```

```
dataset_nd <- daily_data
square_return <- dataset_nd^2



eacf(square_return)
```

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

```
adf.test(square_return)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  square_return
## Dickey-Fuller = -2.618, Lag order = 9, p-value = 0.3167
## alternative hypothesis: stationary
```

```
square_return <- diff(square_return, differences = 1)

adf.test(square_return)
```

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

```
##
##  Augmented Dickey-Fuller Test
##
## data:  square_return
## Dickey-Fuller = -9.5366, Lag order = 9, p-value = 0.01
## alternative hypothesis: stationary
```

```
##Now Finding best ARIMA model for square(return)

# object to store ARIMA summary in
model_summary <- data.frame()

# loop to store
for(p in 0:7){
  for(q in 0:10){
    for (d in 1:1) {
      fit <- Arima(square_return, order=c(p,d,q), method="ML")

      s <- shapiro.test(rstandard(fit))

      # H0: The model does not show lack of fit
      # H1: not H0
      lb <- LB.test(fit, lag = 50)

      # gather everything into a single data frame
      # AIC, BIC, SHAPIRO OF RESIDUALS, LB TEST
      acc_ext <- data.frame(# arima order
        p,
        d,
        q,
        # goodness of fit
        LJUNG = lb$p.value,
        SHAPIRO = s$p.value,
        AIC = AIC(fit),
        BIC = BIC(fit)
      )

      # add ARIMA summary
      model_summary <- rbind(model_summary, acc_ext)
    }
  }
}

# show summary
filter(model_summary[order(model_summary$BIC, decreasing = FALSE),], LJUNG > 0.05)
```

```
## [1] p       d       q       LJUNG   SHAPIRO AIC     BIC
## <0 rows> (or 0-length row.names)
```

```
## Best forecast is 3,1,2
##FORECASTING

garch_model <- ugarchspec(variance.model = list(model = "sGARCH"), mean.model = list
(armaOrder = c(3, 2)), distribution.model = "norm")
garch_fit <- ugarchfit(garch_model, data = daily_data)
y_pred <- ugarchforecast(garch_fit, n.ahead = 365)

sfinal <- garch_model
setfixed(sfinal) <- as.list(coef(garch_fit))
sim <- ugarchpath(spec = sfinal,
                  m.sim = 1,
                  n.sim = 1*60,
                  rseed = 16)

last_value = tail(dataset_nd)[1]
print(last_value)
```

```
## [1] 147.05
```

```
p <- ts(fitted(sim), frequency = 365, start = c(2023, 11, 4))
print(p)
```

```
## Time Series:
## Start = c(2023, 11)
## End = c(2023, 70)
## Frequency = 365
##           [,1]
##  [1,] 138.4462
##  [2,] 137.8891
##  [3,] 142.2499
##  [4,] 136.0794
##  [5,] 141.3680
##  [6,] 138.9441
##  [7,] 134.2221
##  [8,] 134.4450
##  [9,] 138.8180
## [10,] 141.4795
## [11,] 149.1294
## [12,] 149.6883
## [13,] 145.9715
## [14,] 152.7109
## [15,] 155.9389
## [16,] 147.1164
## [17,] 149.8967
## [18,] 152.1492
## [19,] 149.4070
## [20,] 154.5963
## [21,] 146.8899
## [22,] 145.1614
## [23,] 144.3161
## [24,] 151.0399
## [25,] 146.7204
## [26,] 154.2144
## [27,] 160.0064
## [28,] 165.4967
## [29,] 169.6574
## [30,] 169.9604
## [31,] 165.6055
## [32,] 165.2403
## [33,] 164.2600
## [34,] 160.3096
## [35,] 156.2874
## [36,] 157.6903
## [37,] 160.1942
## [38,] 161.0866
## [39,] 162.2063
## [40,] 161.1090
## [41,] 156.5354
## [42,] 163.1633
## [43,] 163.7091
## [44,] 161.6093
## [45,] 158.5729
## [46,] 159.2707
## [47,] 159.5546
## [48,] 161.4463
## [49,] 163.3715
## [50,] 163.0712
```

```
## [51,] 163.6053
## [52,] 165.5181
## [53,] 167.7133
## [54,] 169.8372
## [55,] 163.3863
## [56,] 165.5458
## [57,] 169.7277
## [58,] 164.3307
## [59,] 178.7734
## [60,] 173.2697
```

```
p_1 = last_value - p[1]
```

```
#par(mfrow=c(2,1))
plot(dataset_nd, type = "l")
lines(p+p_1+p_1, col='red')
```