Performing optimal time series modelling using the ARIMA models requires various efforts and one of the major efforts is finding the value of its parameters. This model includes three-parameter p, d and q. We are going to discuss how we can choose optimal values for these parameters.

Table of content

About ARIMA model

We have already discussed that the ARIMA models combine two models and 1 method. Two models are Auto Regression(AR) and Moving Average(MA). One method is differencing(I). These three work together when the time series we use is non-stationary. In simple words, we can call a model an ARIMA model if we apply differencing (I) at least once to make the data stationary and combine autoregressive and moving averages to make some forecasting based on old time-series data. The equation of this model can be explained by the following expressions:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$

In words, we can explain this expression as,

Prediction = constant + linear combination lags of Y + linear combination of lagged forecast errors

Here we came to a point where we are required to understand what p, q, and d mean. Let’s take a look at the below section.

About p, d, q, values in ARIMA

For a better explanation of ARIMA we can also write it as (AR, I, MA) and by this, we can assume that in the ARIMA, p is AR, d is I and q is MA.

These parameters can be explained as follows

- p is the number of autoregressive terms,
- d is the number of nonseasonal differences,
- q is the number of lagged forecast errors in the prediction equation.

For an example, ARIMA(1, 1, 2) can also be called a damped-trend linear exponential smoothing where we are applying one time differencing on the time series if it is non-stationary and after that, we are performing autoregression on the series with one lag when the series is stationary by differencing and 2 average moving average order is applied.

How to Choose Values of p, d and q?

There are various ways to choose the values of parameters of the ARIMA model. Without being confused we can do this using the following steps:

1. Test for stationarity using the augmented dickey fuller test.
2. If the time series is stationary try to fit the ARMA model, and if the time series is non-stationary then seek the value of d.
3. If the data is getting stationary, then draw the autocorrelation and partial autocorrelation graph of the data.
4. Draw a partial autocorrelation graph(ACF) of the data. This will help us in finding the value of p because the cut-off point to the PACF is p.
5. Draw an autocorrelation graph(ACF) of the data. This will help us in finding the value of q because the cut-off point to the ACF is q.

Implementation of ARIMA

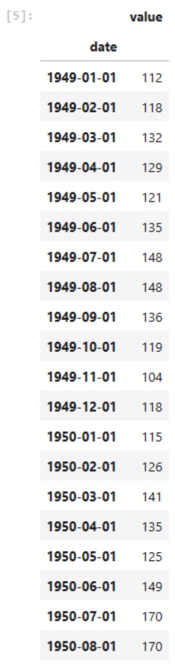
Let’s take a look at how we can perform these steps one by one.

import pandas as pd

data = pd.read_csv(“AirValue.csv”, index_col='date')

data.head(20)

Output:

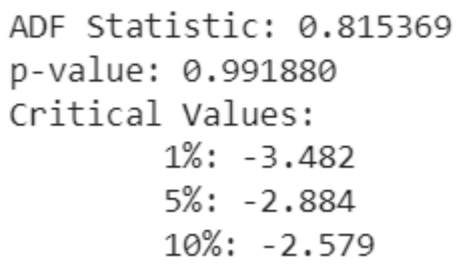


```
[5]:            value
      date
      1949-01-01  112
      1949-02-01  118
      1949-03-01  132
      1949-04-01  129
      1949-05-01  121
      1949-06-01  135
      1949-07-01  148
      1949-08-01  148
      1949-09-01  136
      1949-10-01  119
      1949-11-01  104
      1949-12-01  118
      1950-01-01  115
      1950-02-01  126
      1950-03-01  141
      1950-04-01  135
      1950-05-01  125
      1950-06-01  149
      1950-07-01  170
      1950-08-01  170
```

Augmented Dickey-Fuller test

```
from statsmodels.tsa.stattools import adfuller

result = adfuller(data['value'])

print('ADF Statistic: %f' % result[0])

print('p-value: %f' % result[1])

print('Critical Values:')

for key, value in result[4].items():

  print('\t%s: %.3f' % (key, value))
```

Output:

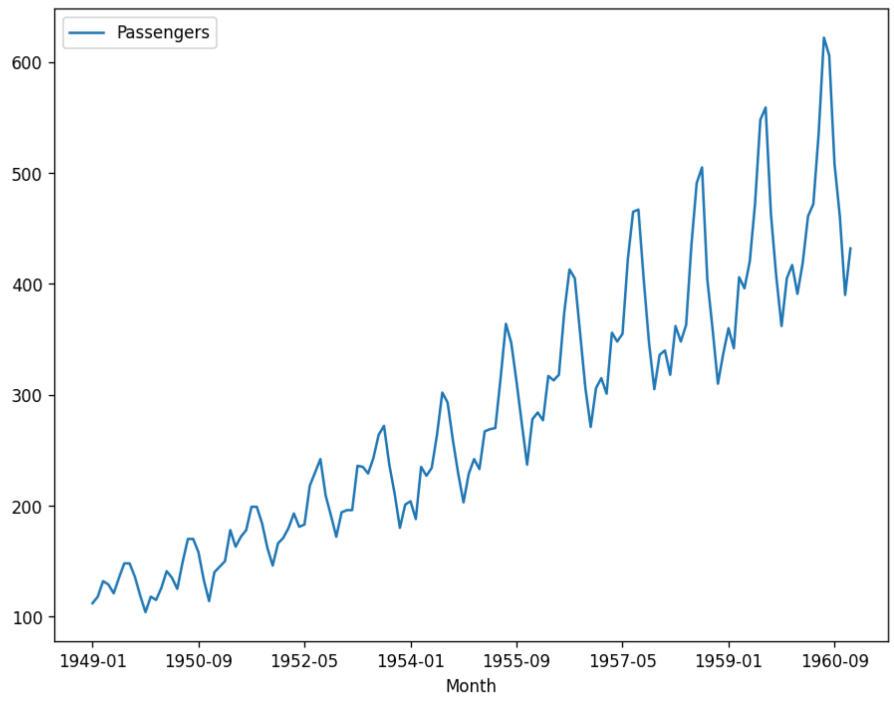


```
ADF Statistic: 0.815369
p-value: 0.991880
Critical Values:
        1%: -3.482
        5%: -2.884
        10%: -2.579
```

Here we can see that the p-value is more than 0.05 this means our null hypothesis will be rejected and we will take this series as non-stationary. Let's make a plot of this data

```
data.plot()
```

Output:



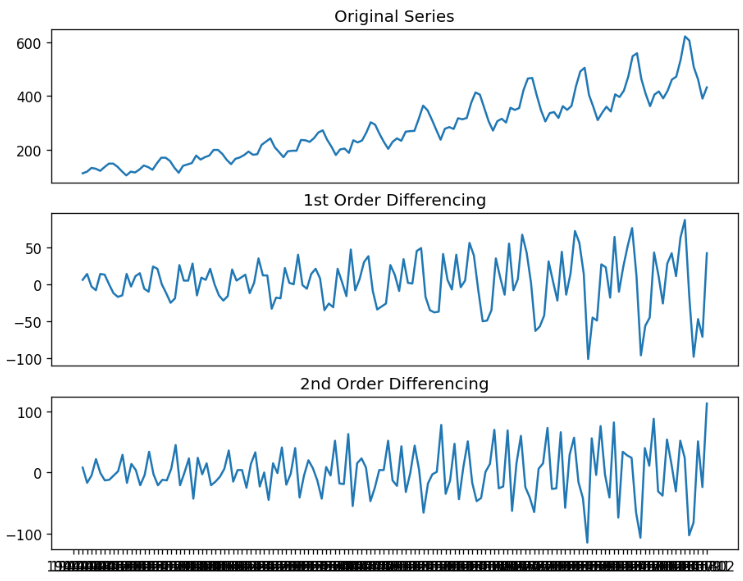
Here it is visible that the data is not stationary and requires differentiation.

Finding the value of the d parameter

There is no such method that can tell us how much value of d will be optimal. However, the value of differencing can be optimal till 2 so we will try our time series in both. Pandas provide this option of differencing. Let’s utilize this.

```
import numpy as np, pandas as pd

import matplotlib.pyplot as plt

plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})


# Original Series

fig, (ax1, ax2, ax3) = plt.subplots(3)

ax1.plot(data.Value); ax1.set_title('Original Series'); ax1.axes.xaxis.set_visible(False)

# 1st Differencing

ax2.plot(data.Value.diff()); ax2.set_title('1st Order Differencing');
ax2.axes.xaxis.set_visible(False)

# 2nd Differencing

ax3.plot(data.Value.diff().diff()); ax3.set_title('2nd Order Differencing')

plt.show()
```

Output:

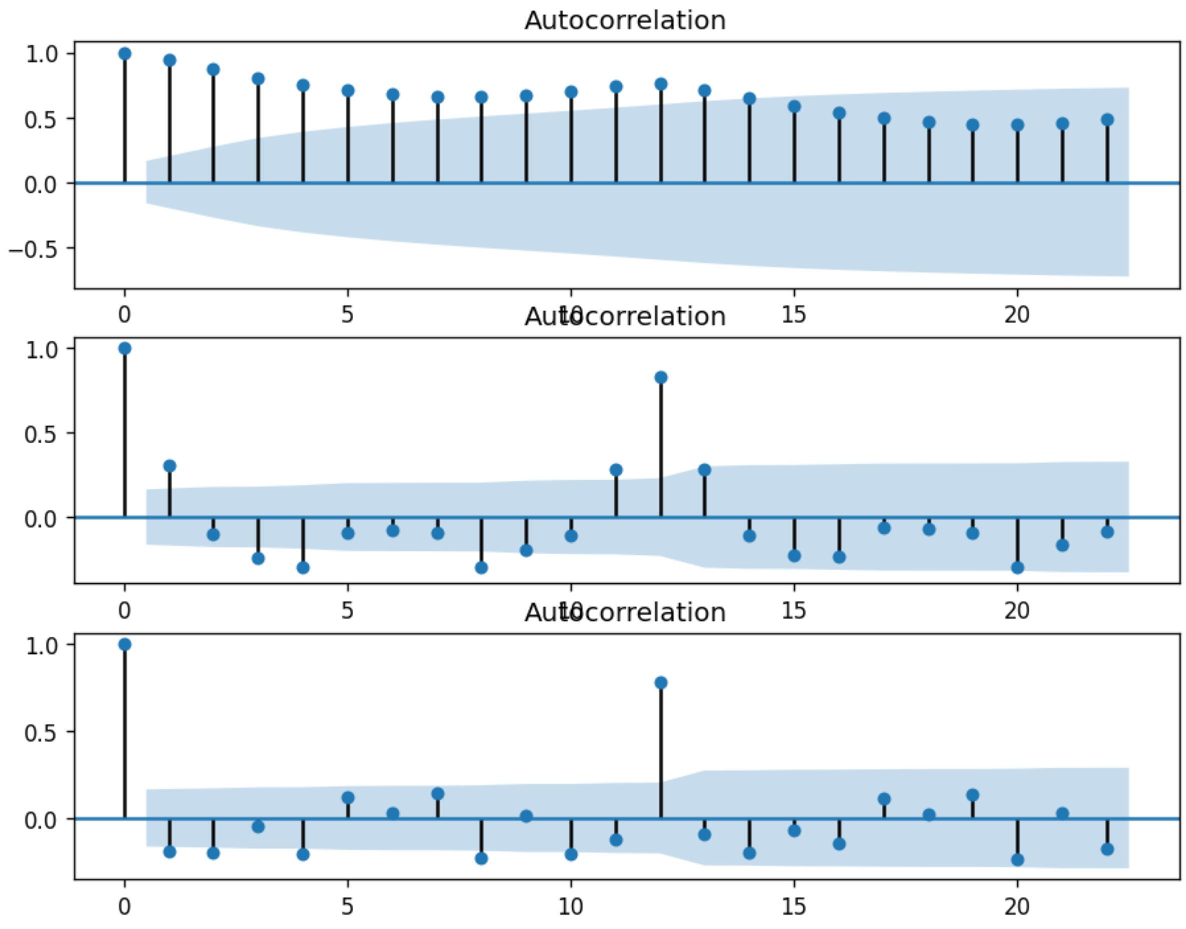


Here we can see how the time series has become stationary. One thing which is noticeable here is in first-order differencing we have fewer noises in the data while after 1st order there is an increase in the noise. So we can select 1st order differencing for our model.

We can also verify this using an autocorrelation plot.

```
from statsmodels.graphics.tsaplots import plot_acf

fig, (ax1, ax2, ax3) = plt.subplots(3)

plot_acf(data.Value, ax=ax1)

plot_acf(data.Value.diff().dropna(), ax=ax2)

plot_acf(data.Value.diff().diff().dropna(), ax=ax3)
```

output:



Here we can see that in second-order differencing the immediate lag has gone on the negative side, representing that in the second-order the series has become too differenced and that the first order is preferred.

Finding the value of the p parameter

In the above section, we have identified the optimal value of d. Now in this section, we are going to find the optimal value of p which is our number of autoregressive terms. We can find this value by inspecting the PACF plot.

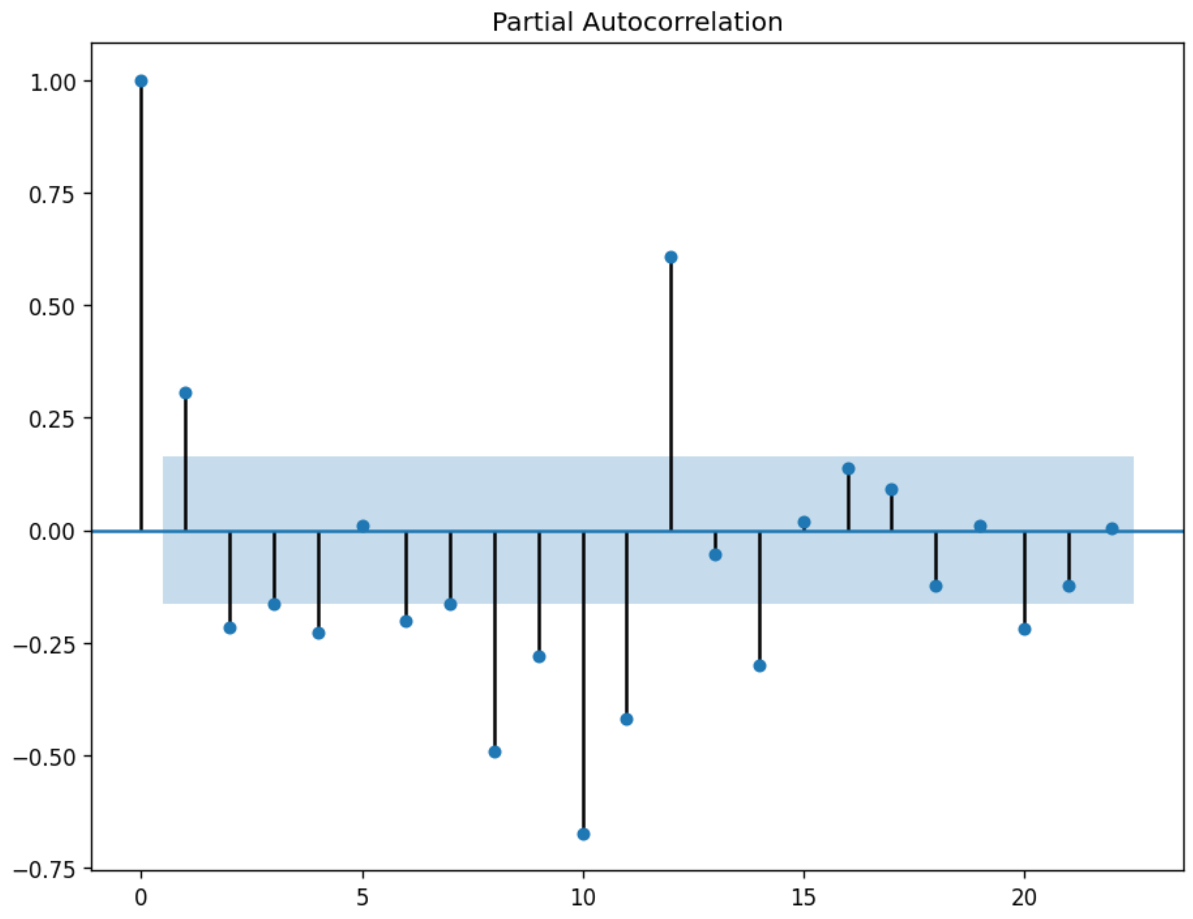
The partial autocorrelation function plot can be used to draw a correlation between the time series and its lag while the contribution from intermediate lags can be ignored. This plotting will let us know about the lags that are not required in the autoregression part.

Significant correlation in a stationary time series can be represented by adding auto regression terms. Using the PACF plot we can take the order of AR terms to be equal to the lags that can cross a significance limit.

```
from statsmodels.graphics.tsaplots import plot_pacf
```

```
plot_pacf(data.value.diff().dropna())
```

Output:



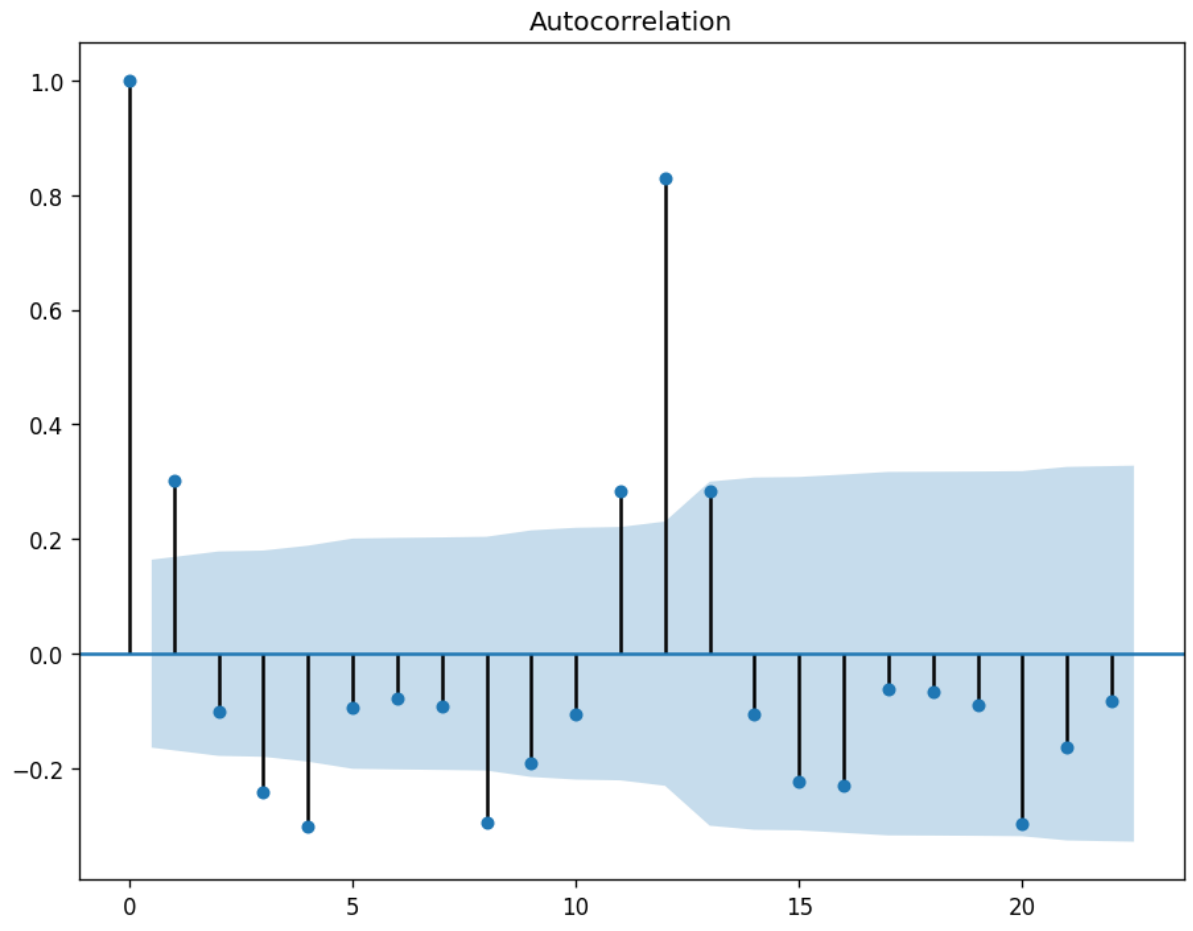
Here we can see that the first lag is significantly out of the limit and the second one is also out of the significant limit but it is not that far so we can select the order of the p as 1.

Finding the value of the q parameter

To find out the value of q we can use the ACF plot. Which will tell us how much moving average is required to remove the autocorrelation from the stationary time series.

```
plot_acf(data.value.diff().dropna())
```

Output:



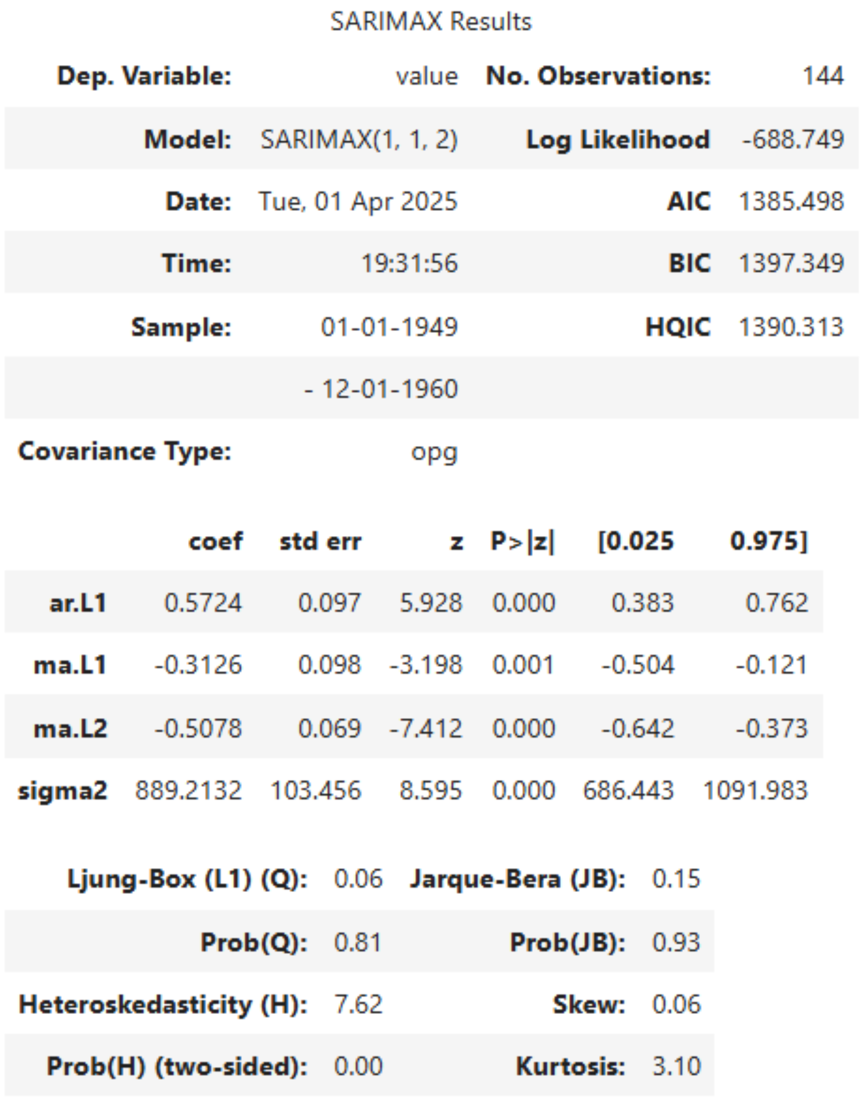
Here we can see that 2 of the lags are out of the significance limit so we can say that the optimal value of our q (MA) is 2.

Building ARIMA model

In the above sections, we have seen how we can find the value of p, d, and q. After finding them we are ready to use them in the ARIMA model. Here we can use the statsmodel library where under the tsa package we have a function for the ARIMA model.

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

ARIMAmodel= SARIMAX(data.value, order = (1,1,2))

ARIMAmodel = ARIMAmodel.fit(disp=0)

ARIMAmodel.summary()
```

Output:



SARIMAX Results

Dep. Variable:	value	**No. Observations:**	144
Model:	SARIMAX(1, 1, 2)	**Log Likelihood**	-688.749
Date:	Tue, 01 Apr 2025	**AIC**	1385.498
Time:	19:31:56	**BIC**	1397.349
Sample:	01-01-1949	**HQIC**	1390.313
	- 12-01-1960		
Covariance Type:	opg		

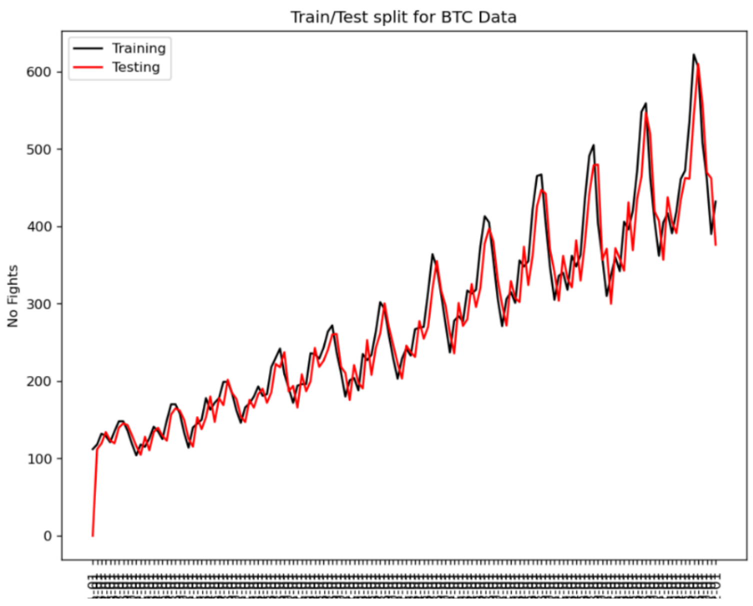
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	0.5724	0.097	5.928	0.000	0.383	0.762
ma.L1	-0.3126	0.098	-3.198	0.001	-0.504	-0.121
ma.L2	-0.5078	0.069	-7.412	0.000	-0.642	-0.373
sigma2	889.2132	103.456	8.595	0.000	686.443	1091.983

Ljung-Box (L1) (Q):	0.06	**Jarque-Bera (JB):**	0.15
Prob(Q):	0.81	**Prob(JB):**	0.93
Heteroskedasticity (H):	7.62	**Skew:**	0.06
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.10

Here we can see the summary of the model. Let's predict from the model.

```
y_pred = ARIMAmodel.get_forecast(len(data.index))

#y_pred_df = y_pred.conf_int(alpha = 0.05)

y_pred_df["Predictions"] = ARIMAmodel.predict(start = y_pred_df.index[0],

                                end = y_pred_df.index[-1])

y_pred_df.index = data.index

y_pred_outARIMA = y_pred_df["Predictions"]
```

Output:



Here we can see that the values are pretty close to the real values.