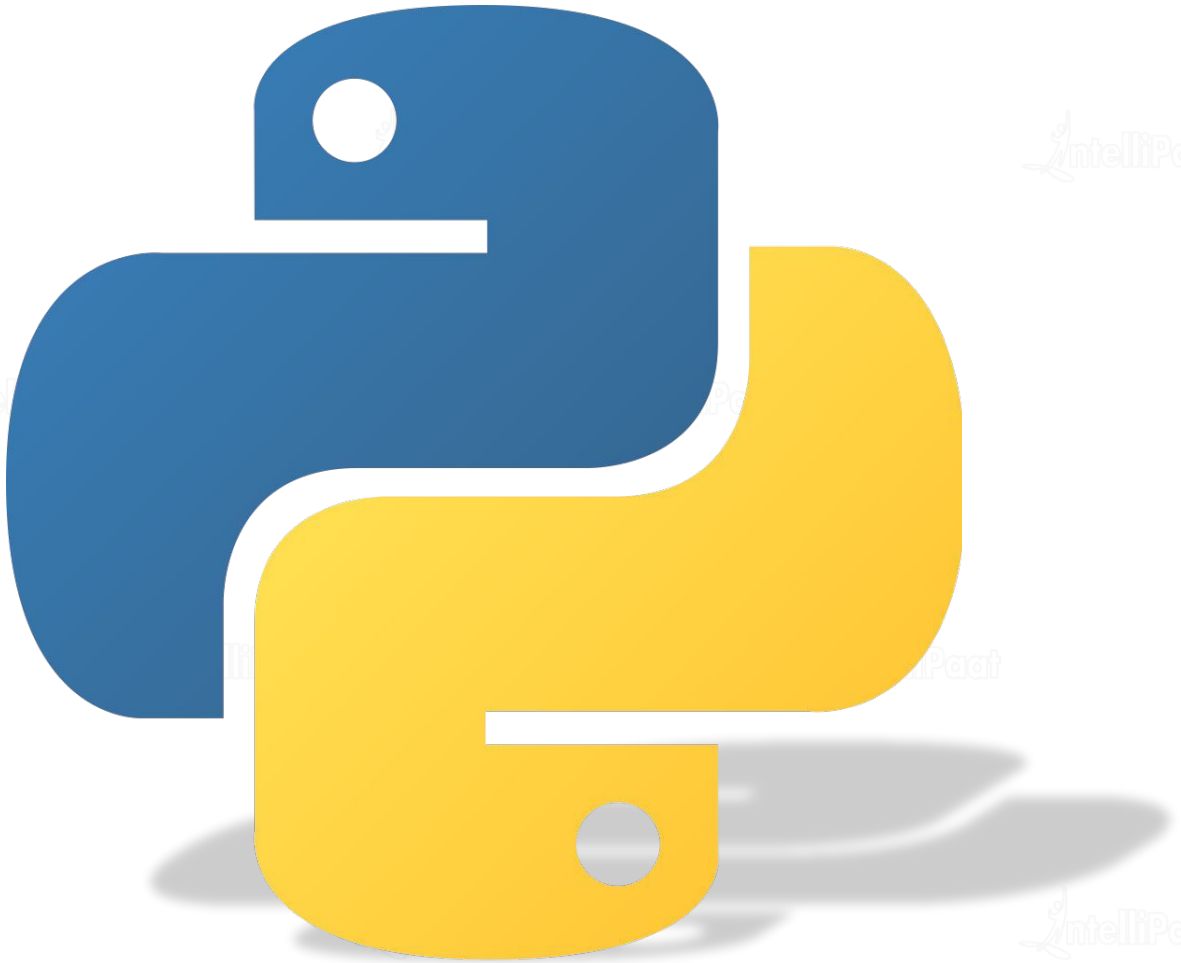
Time Series Forecasting



Agenda

01 Loading Time Series Data

02 Time Series Visualization

03 Time Series Analysis

04 Stationarity in Time Series

05 Time Series Differencing

06 Removing trends and Seasonality

07 ARIMA Model

08 Seasonal ARIMA Model

09 Inferences

Problem Statement

Problem Statement

The sudden influx of passengers can be an overwhelming task for the airport authorities across the globe. With time series data of the passengers over the years, one can forecast the estimated number of passengers that will fly for the specific months of the year. Use the Passenger Dataset to forecast the number of passengers for the upcoming years.

Dataset Information

The Dataset contains the monthly passenger data from **January 1949** to **December 1960**. A decade long monthly information of the data converted in a time series.

The data contains two columns – **Month and #Passengers**.

Loading Time Series Data

How to load the Time Series Data?

We will make use of the Pandas library in Python Programming to load our dataset. We will follow the following steps to load the time series data.

01 We will use the **read_csv()** method to import the dataset from the local machine.

02 After importing the dataset, we will convert the Month column to datetime, using the **to_datetime()** method from the pandas module.

03 Since we have two columns, we will convert the month column to index for easier computation.

Hands on

```
import pandas as pd

#importing the dataset from local machine
data = pd.read_csv("AirPassengers.csv")
#converting the data to datetime object type
data['Month'] = pd.to_datetime(data['Month'])
#changing the column to index
data.index = data['Month']
del data['Month']
#displaying the data
data.head()
```

Month	#Passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121

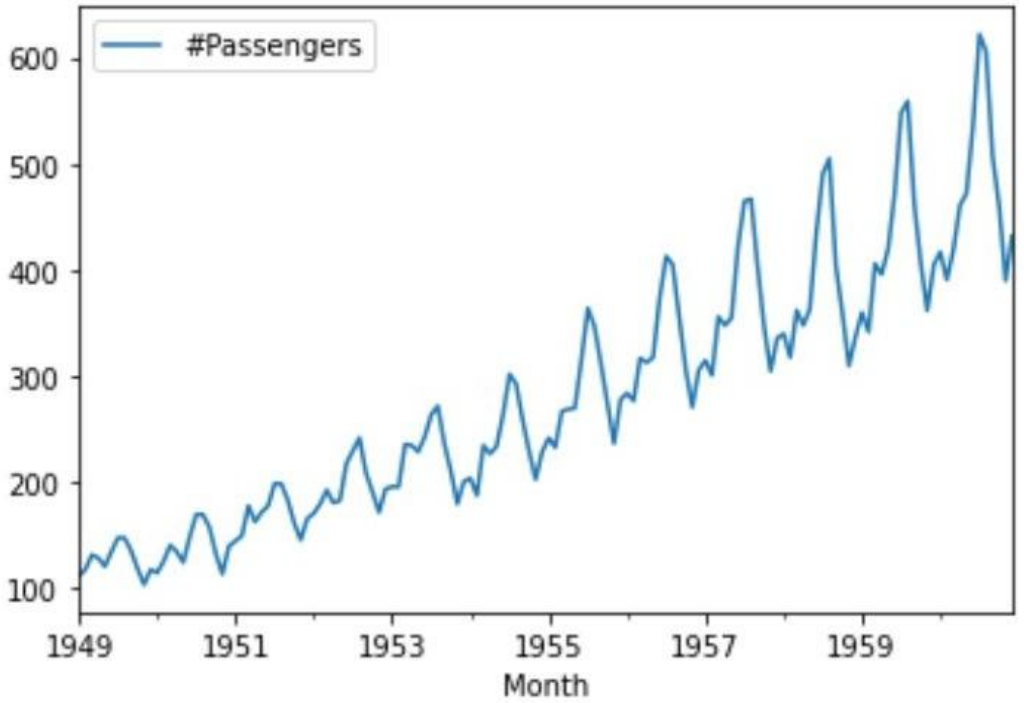
Time Series Visualization

Plotting the Time Series

Plotting the Time series can help in analyzing the visible trends, seasonality in the data. We can use matplotlib or seaborn library to create to create simple line plots to study the data.

Hands on

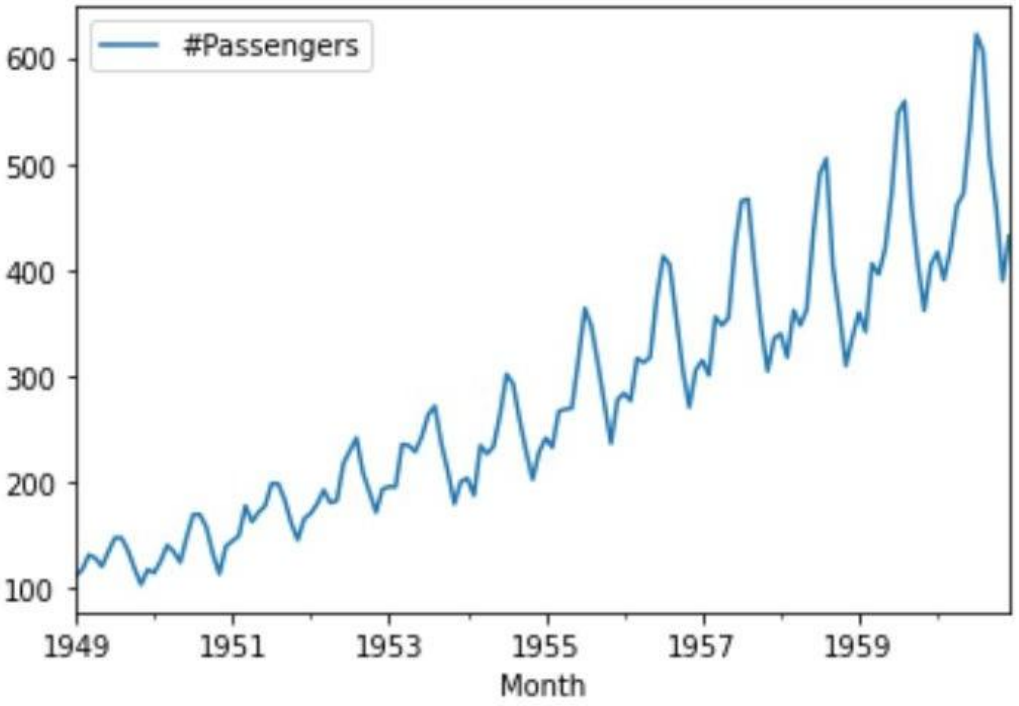
```
import matplotlib.pyplot as plt
import seaborn as sns

data.plot()
```



Time Series Analysis

Analyze the Time Series

From the plot drawn in the previous section, we can clearly see a few patterns. There is a upward trend in the data with seasonality. To understand this, let's take a look at various patterns to identify from time series plots.

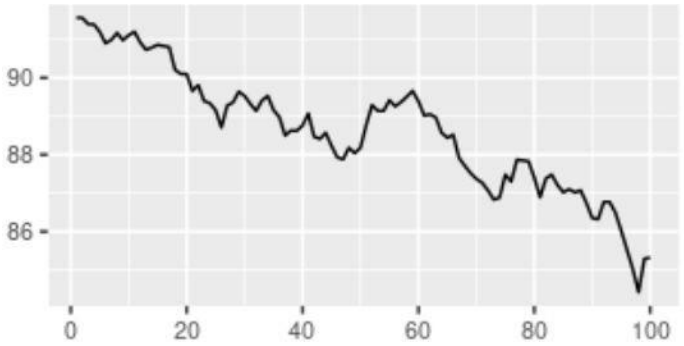


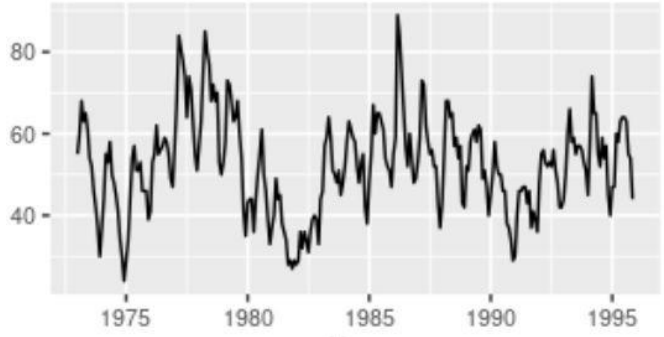
Hands on

A trend is a pattern when there is a long time increase or decrease in the data.

Seasonality pattern happens in equal intervals based on parameters like week, month or a year.

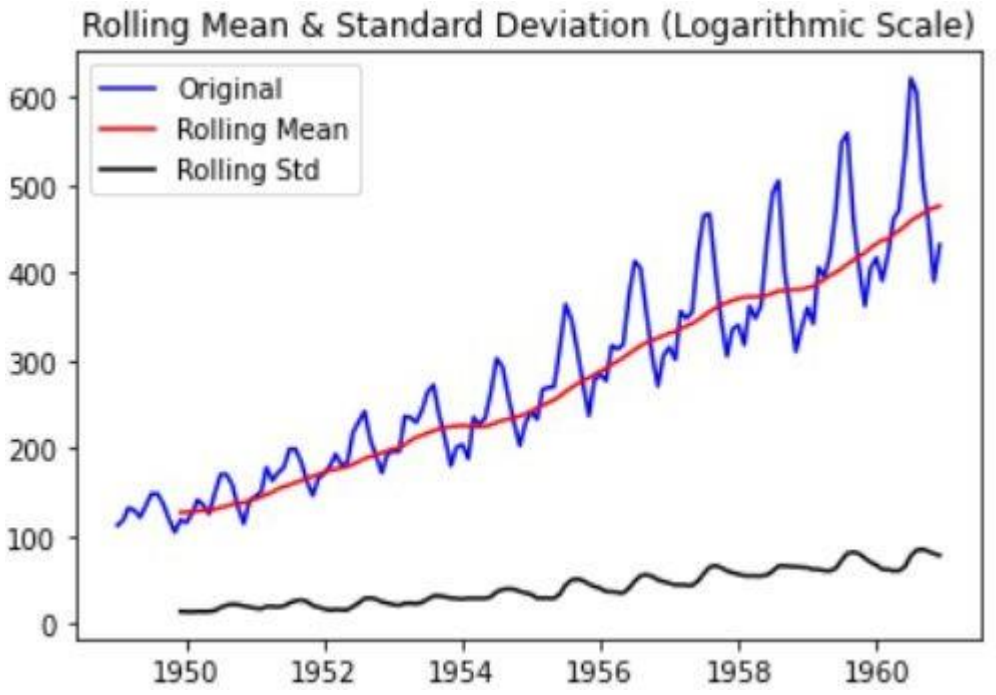
Cyclic pattern showcases the rise and fall in the data that are not fixed with respect to a specific frequency like in a seasonal pattern.





Analyze the Time Series

```
mean_log = data.rolling(window=12).mean()
std_log = data.rolling(window=12).std()

plt.plot(data, color='blue', label='Original')
plt.plot(mean_log, color='red', label='Rolling Mean')
plt.plot(std_log, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation (Logarithmic Scale)')
```

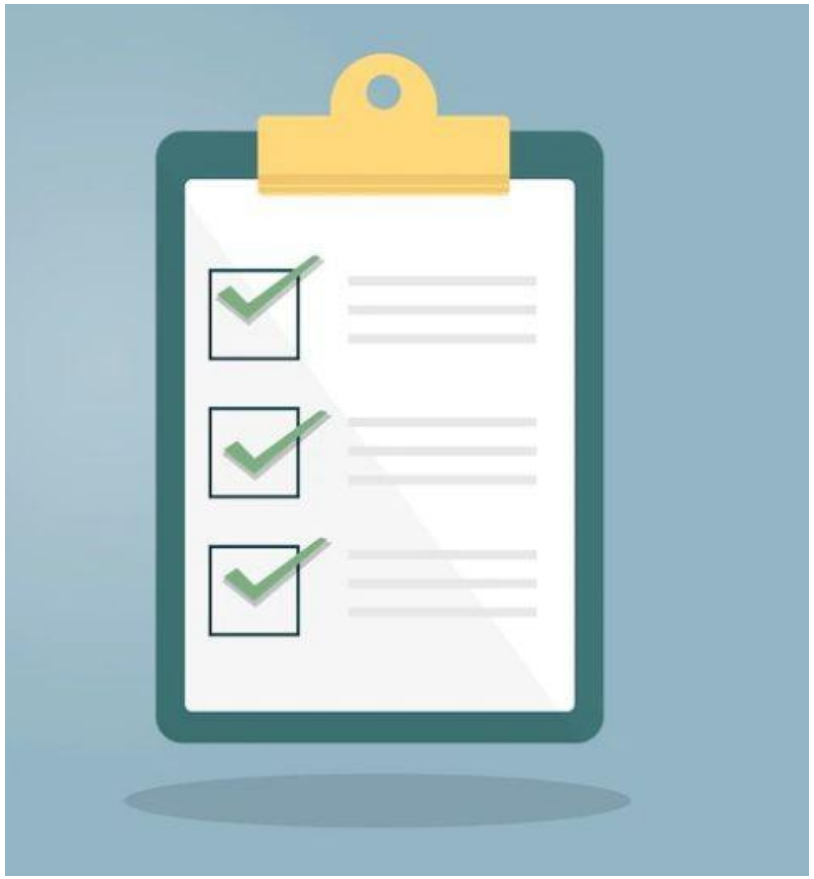


The rolling mean and standard deviation on the original time series data shows a constant increase in the mean.

Stationarity in a Time Series

What is Stationarity?

Before we can begin modeling with the ARIMA model for forecasting, we have to make sure the time series data is stationary. In simple terms, if a data consists of trends and seasonality, the data is not going to be a stationary data. In our case, we have both, so let's take a look at how we can fix this.



How to Find the Stationarity of a Time Series?

To find the stationarity in a data, we can use the statistical test such as Augmented Dickey Fuller test.

Here, we will take two hypotheses, a null hypotheses and an alternate hypotheses. If we are able to reject the null hypotheses after the computation, the time series is stationary series.

To reject the null hypotheses, the following must be true:

1. If the p-value after the adfuller test is greater than 0.05, we fail to reject the hypotheses.
2. If the p-value is less than 0.05, we can reject the null hypotheses and assume that the time series is stationary.

Hands on

In Python, we can make use of the **statsmodels.tsa.stattools** package that provides the adfuller module to conduct the augmented dickey-fuller test on the time series.

```
#checking the stationarity of the series
from statsmodels.tsa.stattools import adfuller
result = adfuller(data['#Passengers'])
print(result)
```

```
(0.8153688792060472, 0.991880243437641, 13, 130,
```

The p-value from the result is on the index – 1, and we can check if the data is stationary or not. In our case, we have the p-value = 0.9918 more than
0.05 and thus, we cannot reject the null hypotheses and assume the data to be non-stationary.

Time Series Differencing

Differencing

Differencing in time series is the process of reducing the non-stationary time series to a stationary time series with a series of subtraction operations i.e. subtracting the observations from one another.

Diff(t) = x(t) – x(t-1),
Where Diff(t) is the differenced series, x(t) is the observation at given time t, and the x(t-1) is the previous observation.

Other ways to Make Time Series Stationarity

There are several ways to make the time series stationary you can choose from.

Differencing is the most common technique to make time series stationary.

You can use power transformation.

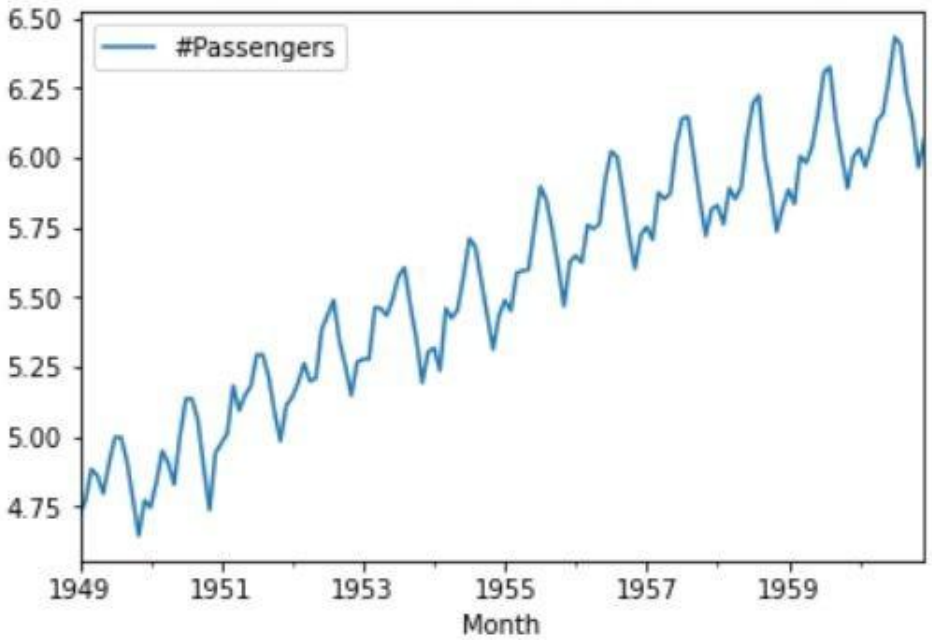
Log transformations of the time series is another technique to make the time series stationary.

Hands-on

```
#logrithmic comutation to make the time series stationary
first_log = np.log(data)
first_log = first_log.dropna()
first_log.plot()
```

Using the log transformation, we will try to make the time series stationary.

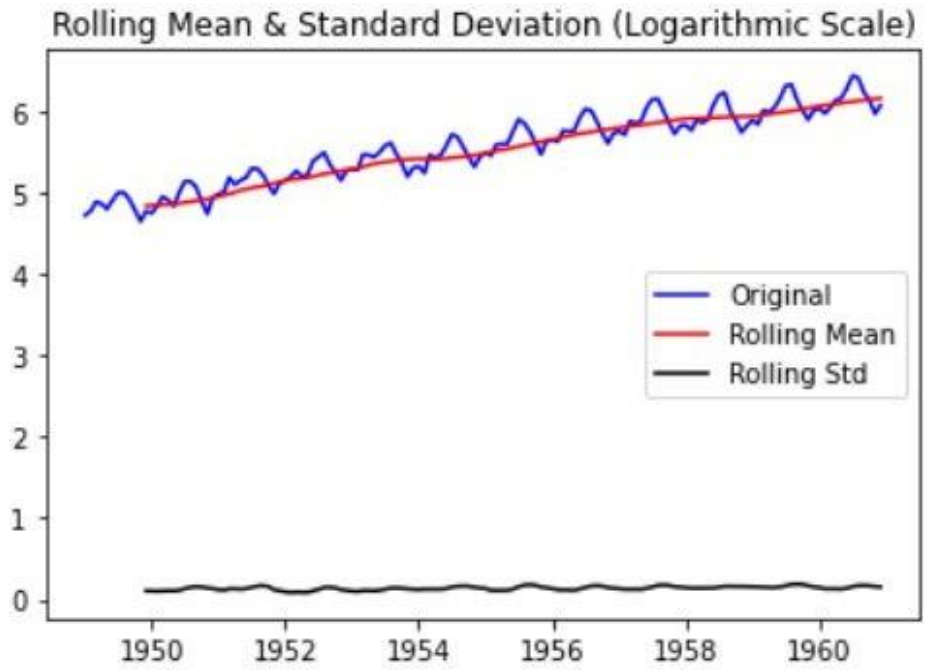
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbe6be16a50>
```



Hands-on

```
mean_log = first_log.rolling(window=12).mean()
std_log = first_log.rolling(window=12).std()

plt.plot(first_log, color='blue', label='Original')
plt.plot(mean_log, color='red', label='Rolling Mean')
plt.plot(std_log, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation (Logarithmic Scale)')
```

We have plotted the rolling standard deviation, and rolling mean on the log transformed data.



From the plot, we can see that we have improved the time series a little in terms of mean and standard deviation.

Removing Trends and Seasonality

Removing Trends and Seasonality

```
new_data = first_log - mean_log
new_data = new_data.dropna()
new_data.head()
```

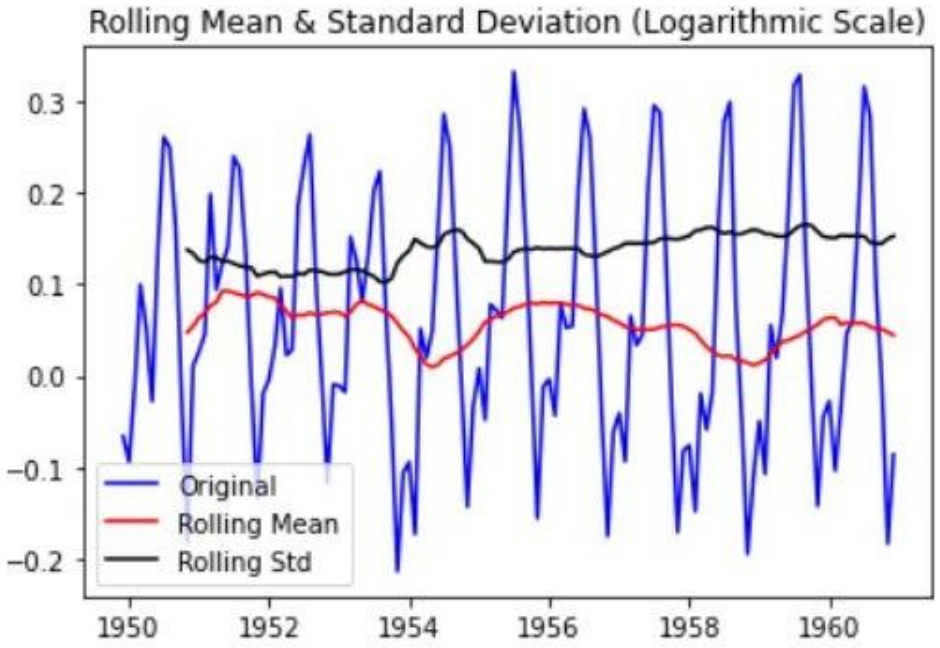
Month	#Passengers
1949-12-01	-0.065494
1950-01-01	-0.093449
1950-02-01	-0.007566
1950-03-01	0.099416
1950-04-01	0.052142

We create a new time series, by subtracting the rolling mean with the log transformed time series.

Removing Trends and Seasonality

```
mean_log = new_data.rolling(window=12).mean()
std_log = new_data.rolling(window=12).std()

plt.plot(new_data, color='blue', label='Original')
plt.plot(mean_log, color='red', label='Rolling Mean')
plt.plot(std_log, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation (Logarithmic Scale)')
```

Log transformations of the time series is another technique to make the time series stationary.



When we plot the rolling mean and standard deviation, we can clearly see the mean and standard deviation is slightly bettered.

Hands on

Adfuller test on the new detrended time series to check the stationarity of the time series.

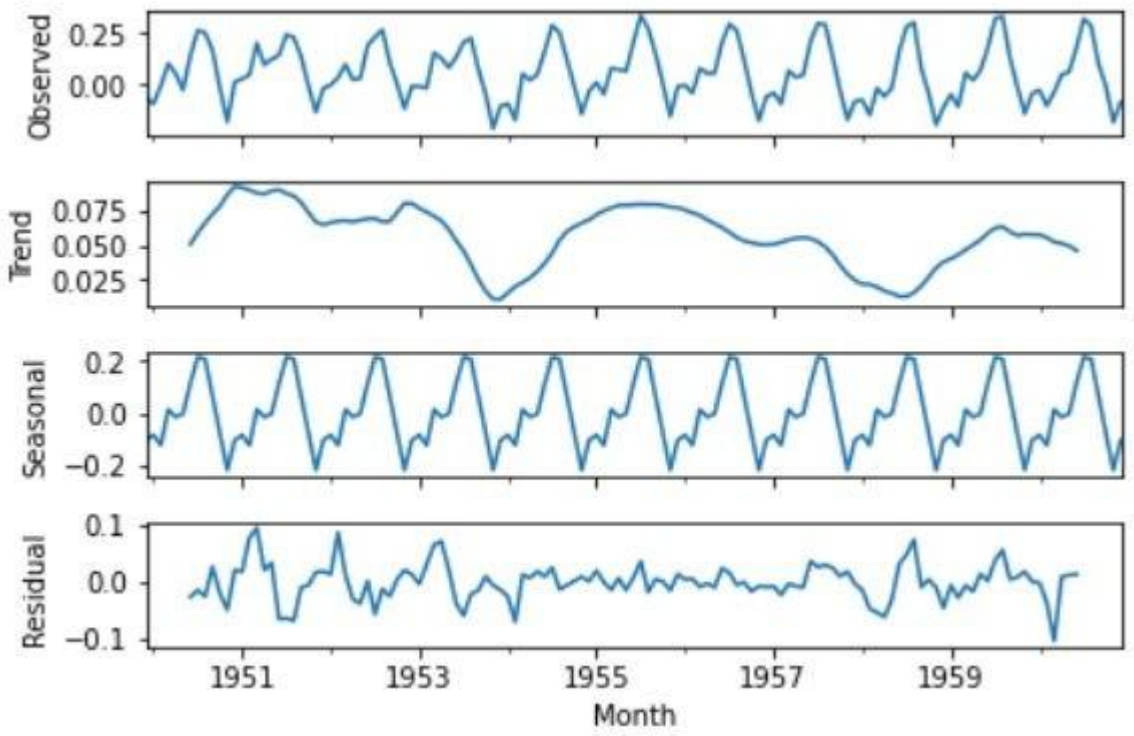
```
#adfuller test for stationarity
result = adfuller(new_data['#Passengers'])
print(result)
```

```
(-3.162907991300889, 0.02223463000124189, 13, 119, {
```

The p-value from the result is less than 0.05, therefore we can reject the null hypotheses and consider the time series to be stationary.

Hands on

```
#seasonal Decompose
from statsmodels.tsa.seasonal import seasonal_decompose
decompose_result = seasonal_decompose(new_data['#Passengers'].dropna())

decompose_result.plot()
```

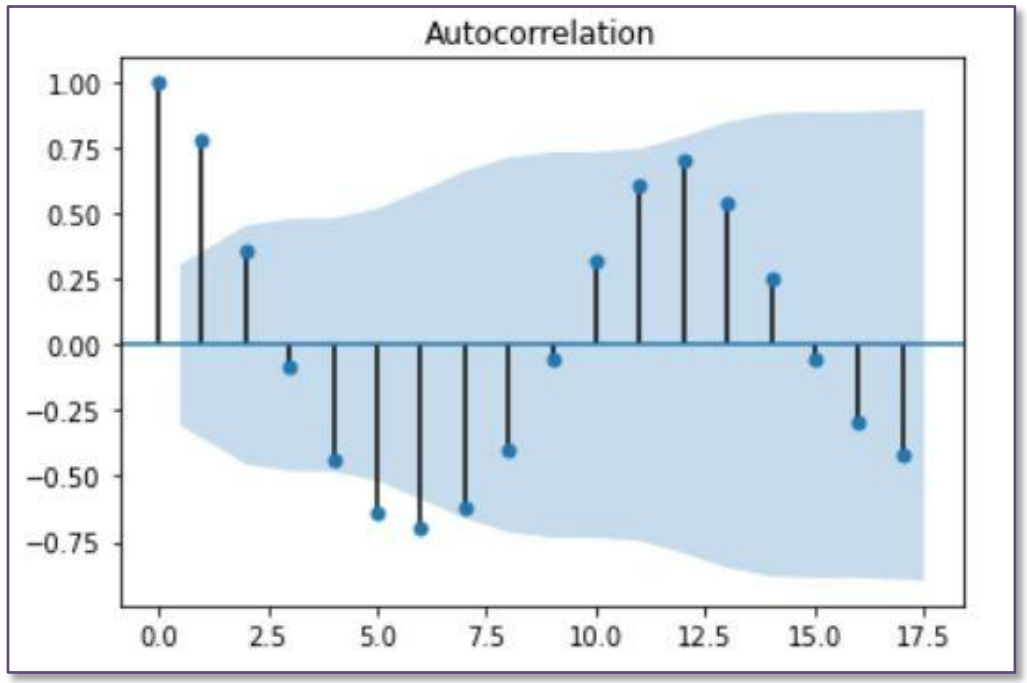


The seasonal decomposition shows the trend to be removed, but the seasonality might still be present in the time series.

Autocorrelation and Partial Autocorrelation

Autocorrelation

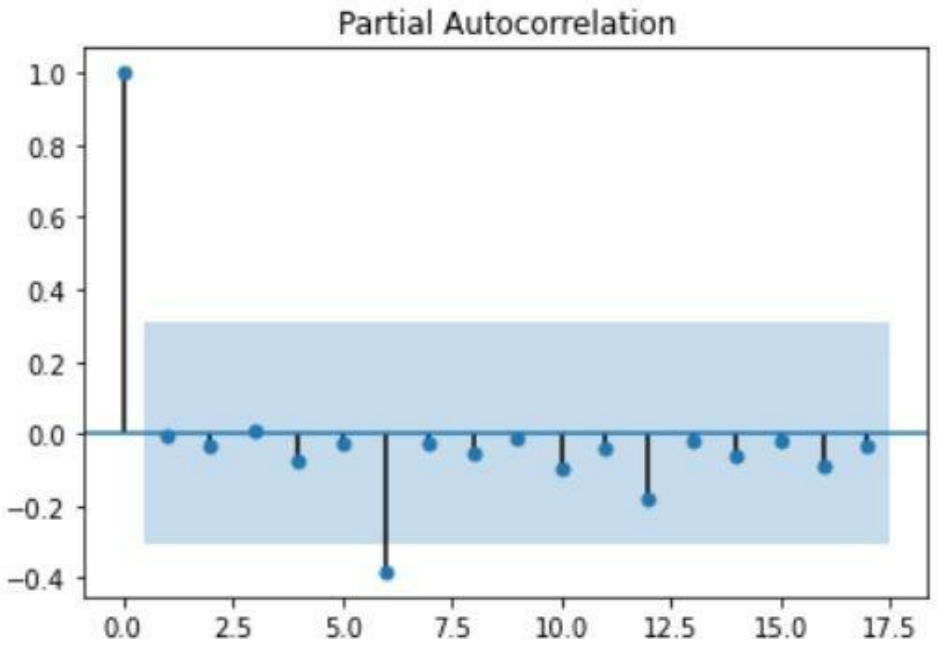
```
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
acf_plot=acf(new_data)
pacf_plot=pacf(new_data)
plot_acf(acf_plot)
plot_pacf(pacf_plot)
```



We use the autocorrelation plot to determine the q value for the ARIMA order.

Partial Autocorrelation

```
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
acf_plot=acf(new_data)
pacf_plot=pacf(new_data)
plot_acf(acf_plot)
plot_pacf(pacf_plot)
```

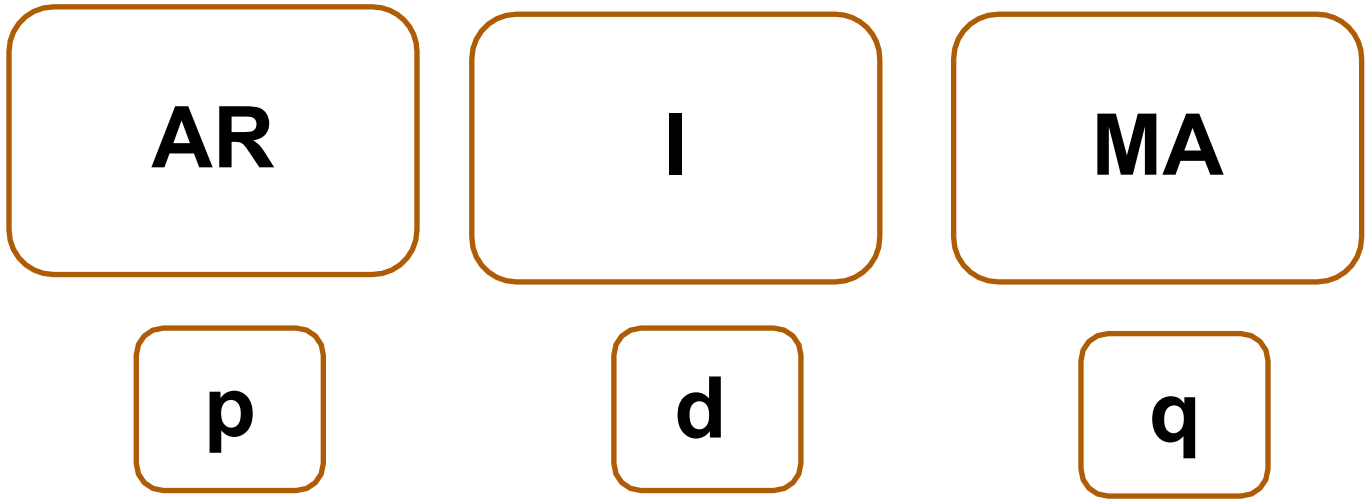


We use the Partial autocorrelation plot to determine the p value for the ARIMA order.

ARIMA Model For Time Series Forecasting

What is ARIMA?

ARIMA model is the combination of Autoregressive(AR), Integrated (I), and Moving Average(MA) models.



Here, p, d and q are the order of AR, order of differencing and order of MA respectively. We have already calculated these values using the autocorrelation and partial autocorrelation plots, or we can deduce these values using the ACF and PCF.

Hands on

```
from statsmodels.tsa.arima_model import ARIMA

train = new_data.iloc[:120]['#Passengers']
test = new_data.iloc[121:]['#Passengers']

model = ARIMA(train, order=(1,0,2))
model_fit = model.fit()
model_fit.summary()
```

The ARIMA model is fit on the training data using the order - (1,0,2). We have taken the order of differencing as 0 since we are using the new transformed time series for the model.

ARMA Model Results

Dep. Variable:	#Passengers	**No. Observations:**	120
Model:	ARMA(1, 2)	**Log Likelihood**	117.901
Method:	css-mle	**S.D. of innovations**	0.090
Date:	Sat, 07 May 2022	**AIC**	-225.803
Time:	08:28:32	**BIC**	-211.865
Sample:	12-01-1949 - 11-01-1959	**HQIC**	-220.143

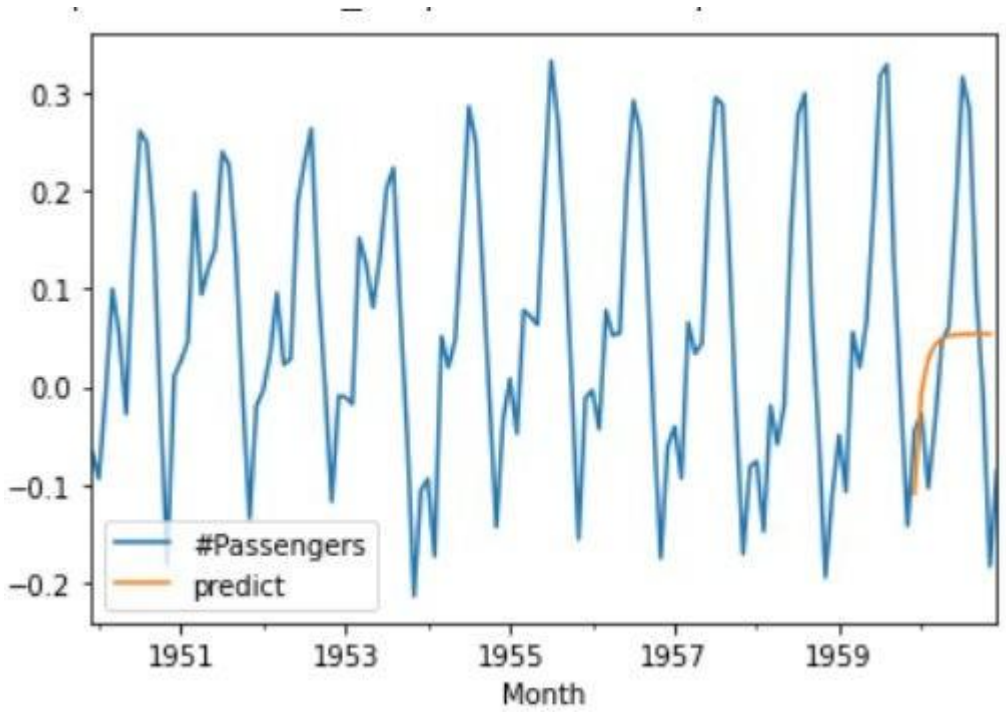
	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0538	0.022	2.470	0.015	0.011	0.096
ar.L1.#Passengers	0.4323	0.159	2.723	0.007	0.121	0.743
ma.L1.#Passengers	0.5917	0.163	3.636	0.000	0.273	0.911
ma.L2.#Passengers	-0.0778	0.169	-0.460	0.646	-0.409	0.254

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	2.3134	+0.0000j	2.3134	0.0000
MA.1	-1.4235	+0.0000j	1.4235	0.5000
MA.2	9.0268	+0.0000j	9.0268	0.0000

Hands on

```
new_data['predict'] = model_fit.predict(start= len(train),
                                        end=len(train)+len(test)- 1,
                                        dynamic=True)
new_data[['#Passengers','predict']].plot()
```



As you can see, the predictions are way off the actual values from the test set. Therefore, we can move to the seasonal ARIMA model for our forecasting.

SARIMA Model For Time Series Forecasting

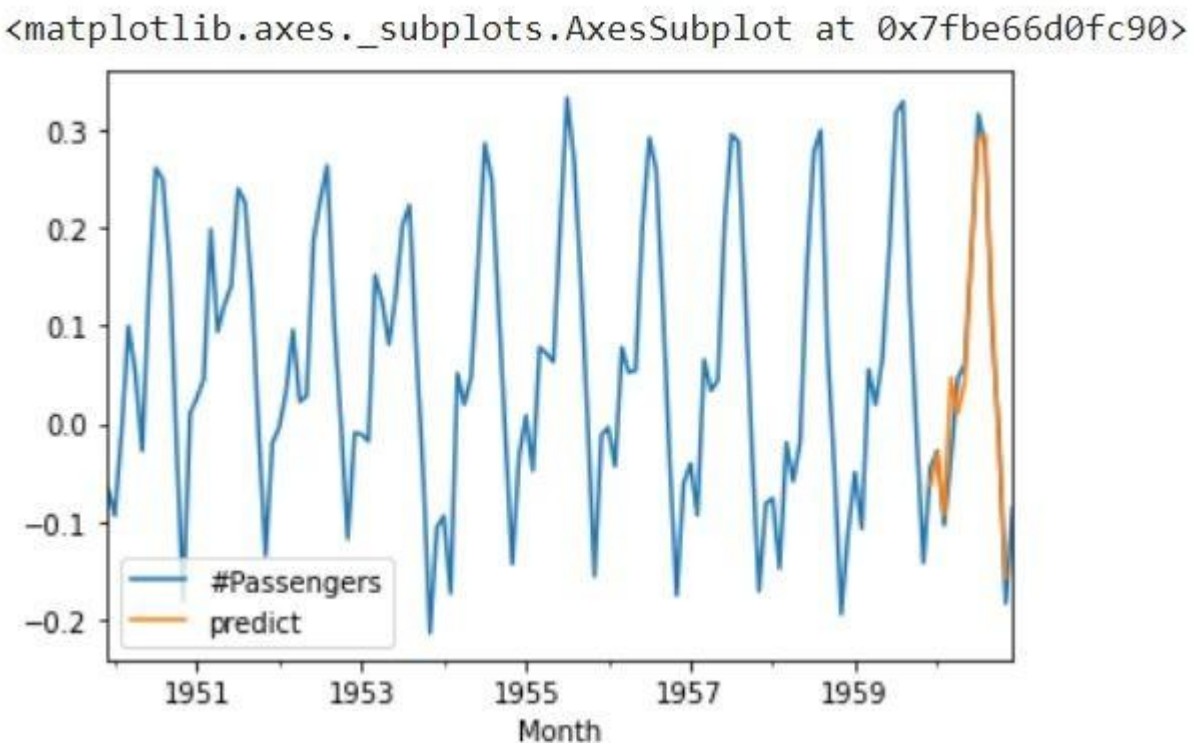
What is Seasonal ARIMA?

In the seasonal ARIMA model, we have to specify the seasonal order as well. The seasonal order remains the same as the ARIMA order, and we can add the periodic order in the seasonal order according to the periodicity.

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

model = SARIMAX(train, order=(1,0,2), seasonal_order=(1,0,2,12))
model = model.fit()
```

```
new_data['predict'] = model.predict(start= len(train) ,
                                    end=len(train)+len(test)- 1,
                                    dynamic=True)

new_data[['#Passengers','predict']].plot()
```

Hands on



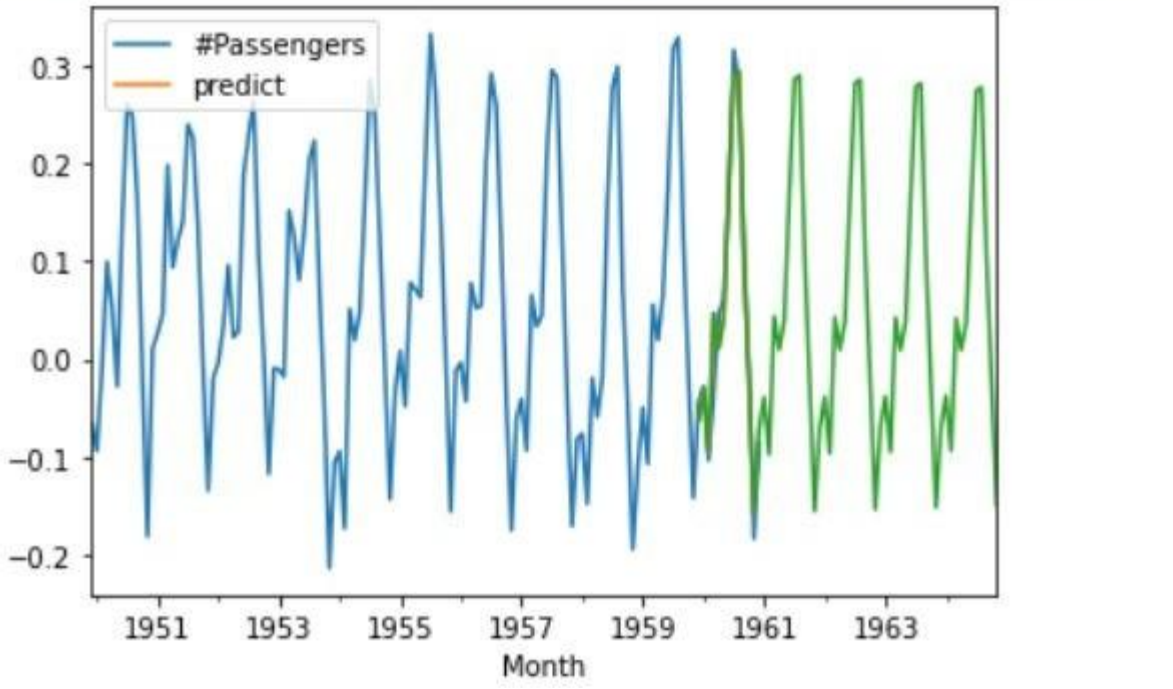
Here, we can see the predicted values on the test set are more accurate than the ARIMA model. Therefore we have successfully created a Time series forecast model. Now we will use this model to forecast the time series.

Inferences

Inferences

```
#predicting the projections for the next 5 years
forecast = model.forecast(steps=60)
new_data.plot()
forecast.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbe6483d250>
```



We had trained the model on the transformed time series, therefore the predictions are aligned to the same. We can train the model with the original dataset, and add the order of differencing manually and get the predictions on the actual values. Or reverse transform the predictions and plot with the original series.