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 Pyramid ARIMA

08 ARIMA Model for Time Series Forecasting

09 SARIMA Model for Times Series Forecasting

10 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

```python
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()
```

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

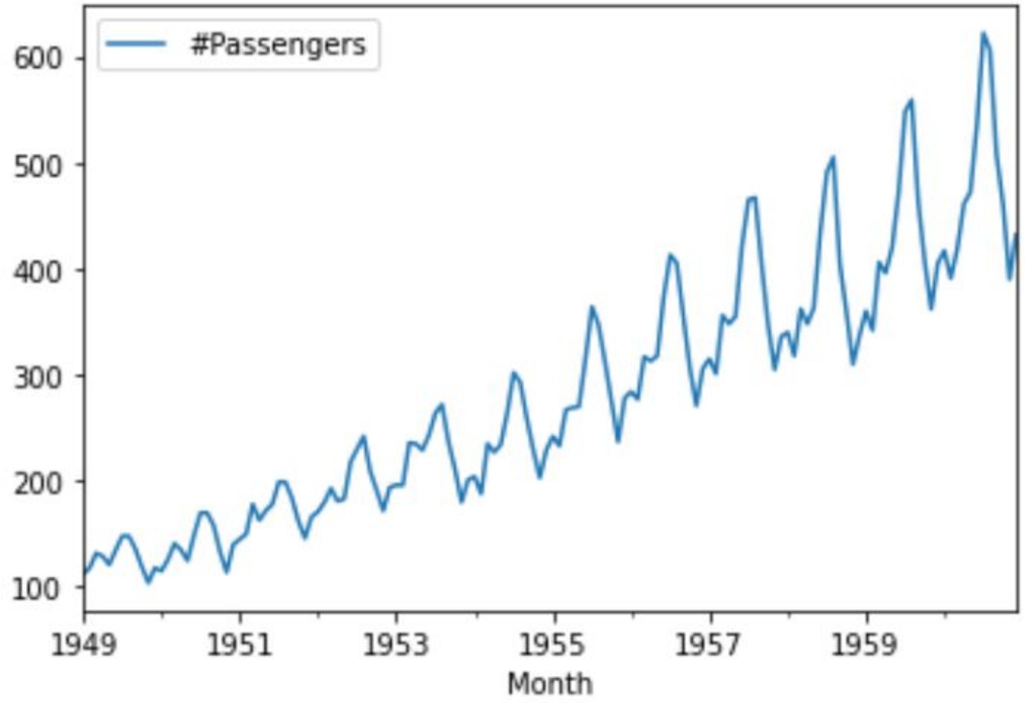
Time Series Visualization

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

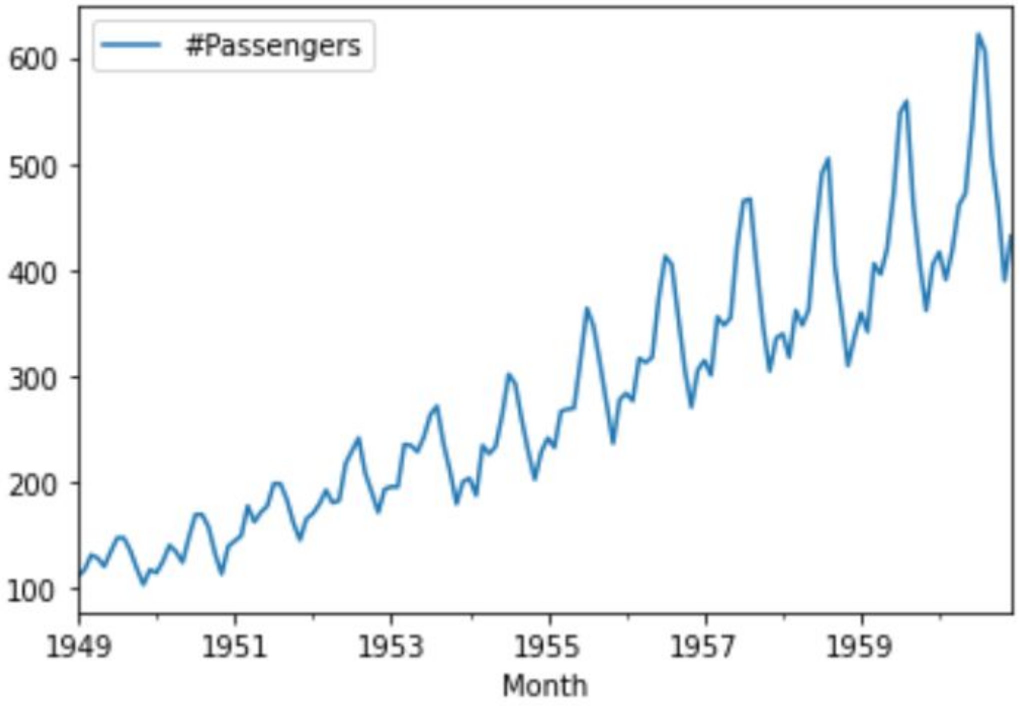
```
import matplotlib.pyplot as plt
import seaborn as sns

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 of decrease in the data.

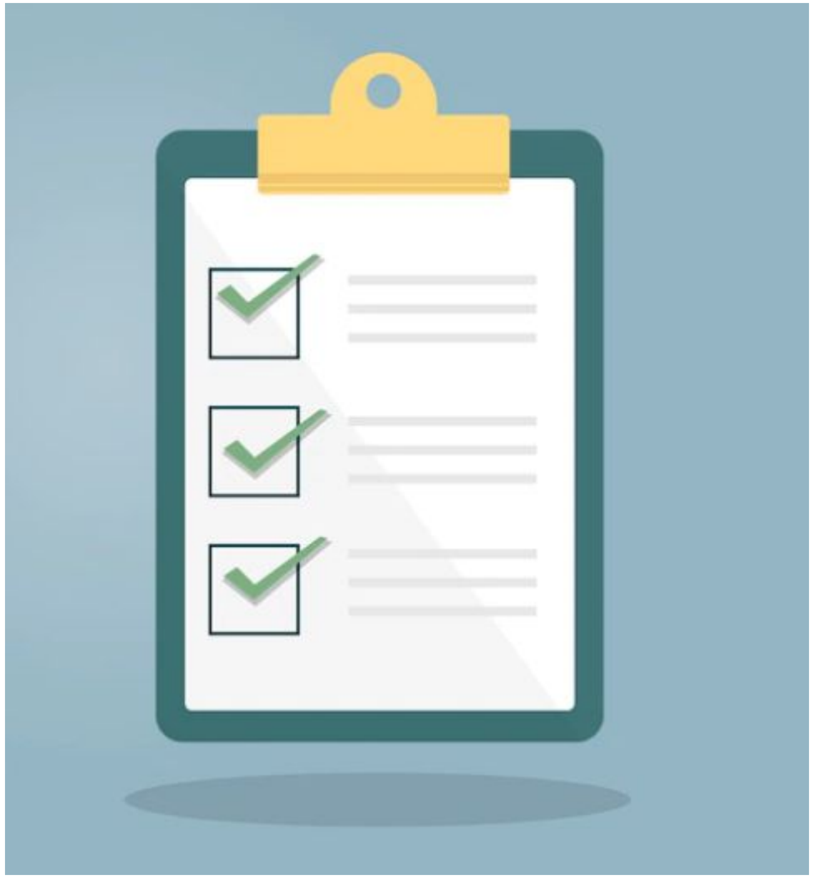
Seasonality pattern happens in equal intervals based on parameters like week, month or a year.

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.

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 not able to reject the null hypotheses after the computation, the time series is a 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 data
from statsmodels.tsa.stattools import adfuller

result = adfuller(data['#Passengers'])
print(result)
```

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 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 subsequent observation.

Other ways to remove Stationarity

There are several ways to remove stationarity from the time series you can choose from.

Differencing is the most common technique to remove stationarity.

You can use power transformation for removing the stationarity.

Log transformations of the time series is another technique to remove the stationarity.

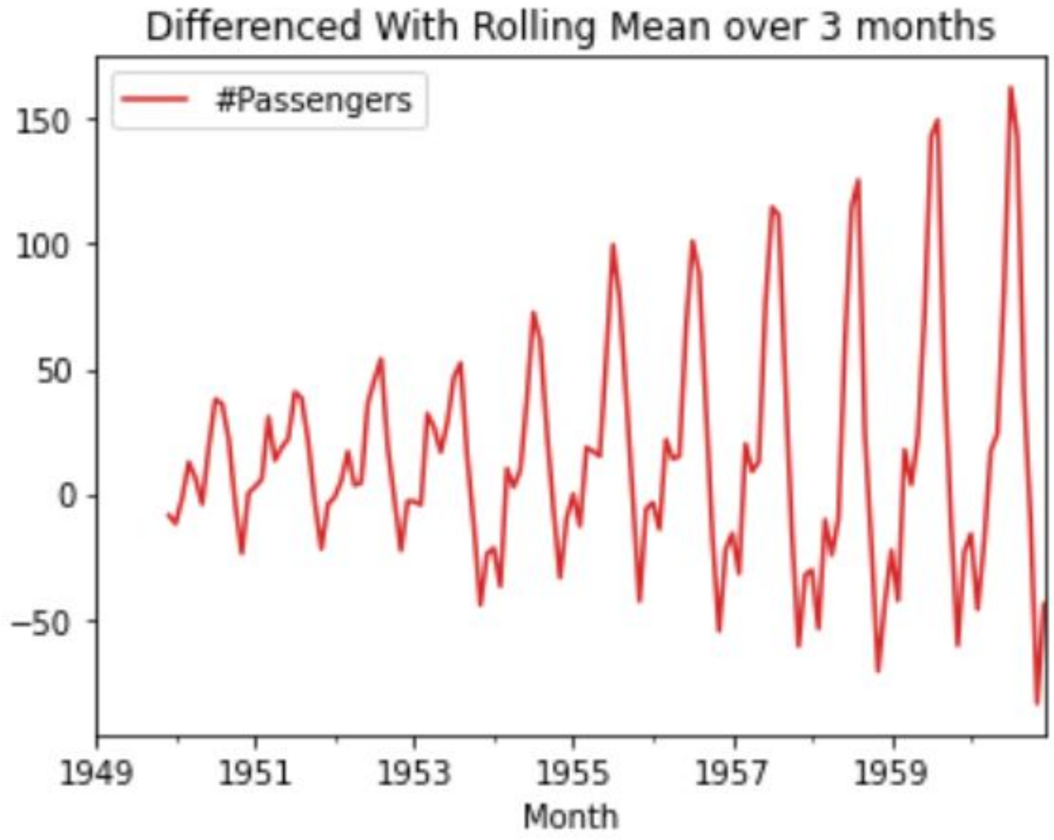
Removing Trends and Seasonality

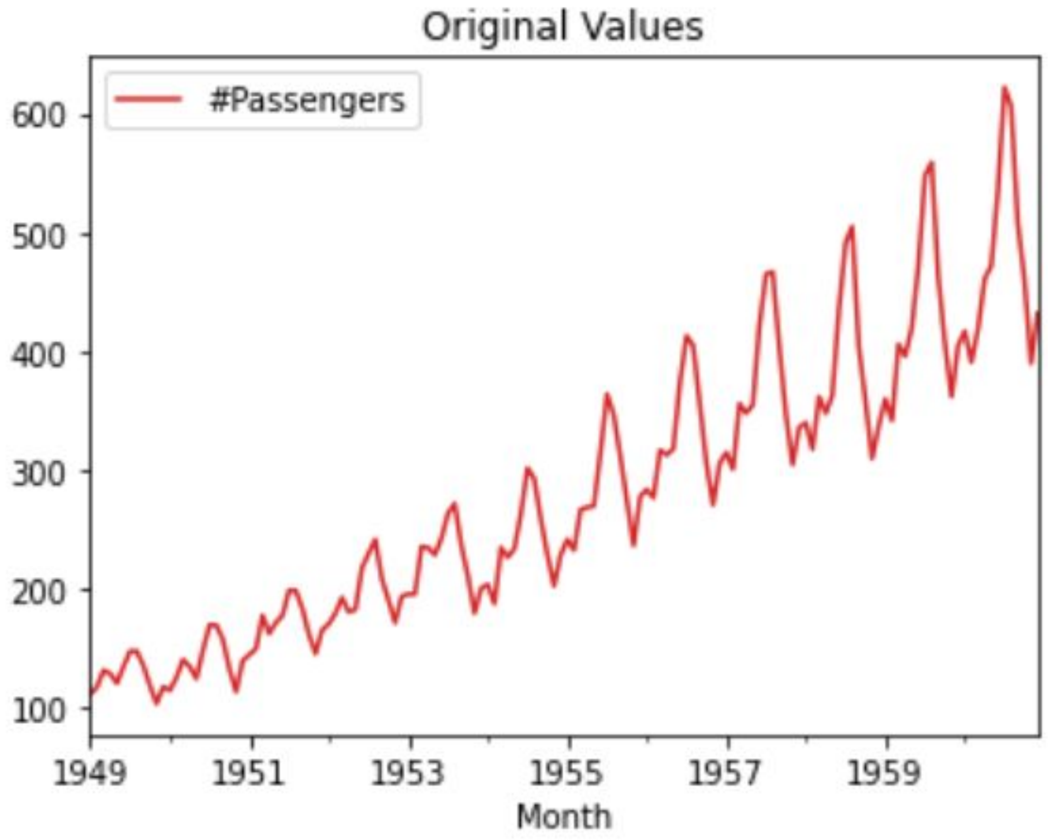
Removing Trends

we will remove the trend from the time series using the rolling mean differencing as shown in the example below.

```
#removing trend using the rolling mean differencing
rolling_mean = data.rolling(window=12).mean()
rolling_mean_detrended = data - rolling_mean

ax1 = plt.subplot(121)
rolling_mean_detrended.plot(figsize=(12,4),color="tab:red",
                            title="Differenced With Rolling Mean over 12 months",
                            ax=ax1)
ax2 = plt.subplot(122)
data.plot(figsize=(12,4),
          color="tab:red",
          title="Original Values",
          ax=ax2)
```

Removing Trends





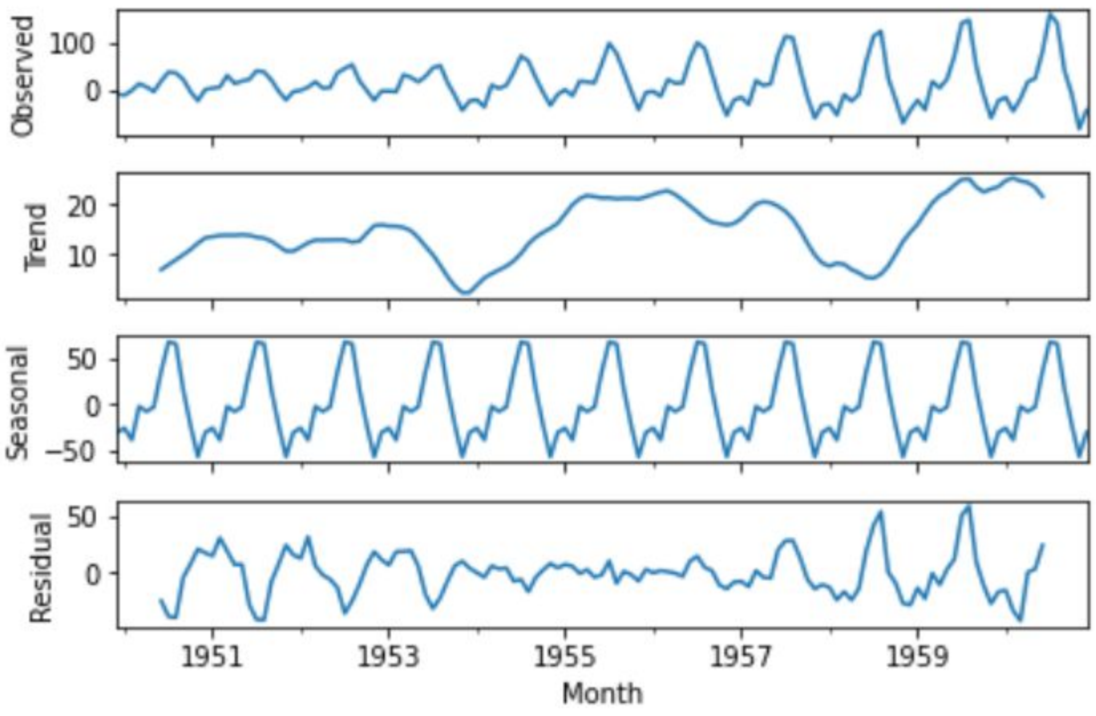
Time Series Decomposition

We can get the decomposition of the time series data using the seasonal_decompose() method from the statsmodels.tsa.seasonal.

```
from statsmodels.tsa.seasonal import seasonal_decompose
decompose_result = seasonal_decompose(rolling_mean_detrended.dropna())

decompose_result.plot()
```

Time Series Decomposition

We can check the trend and seasonality using the seasonal_decompose() method.

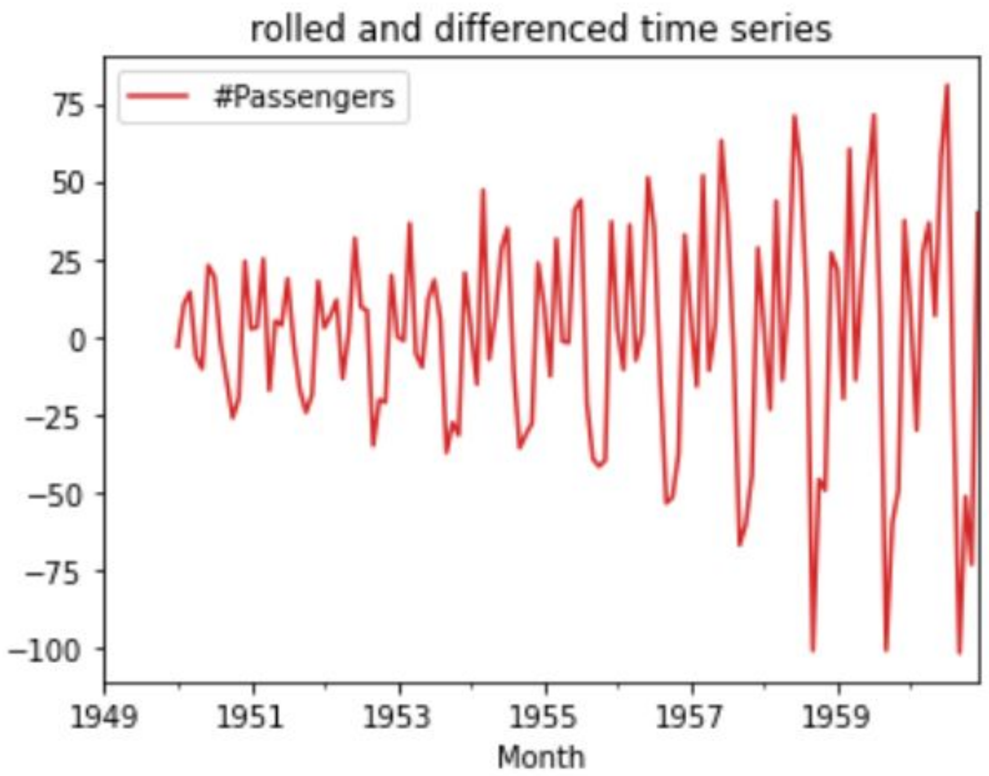


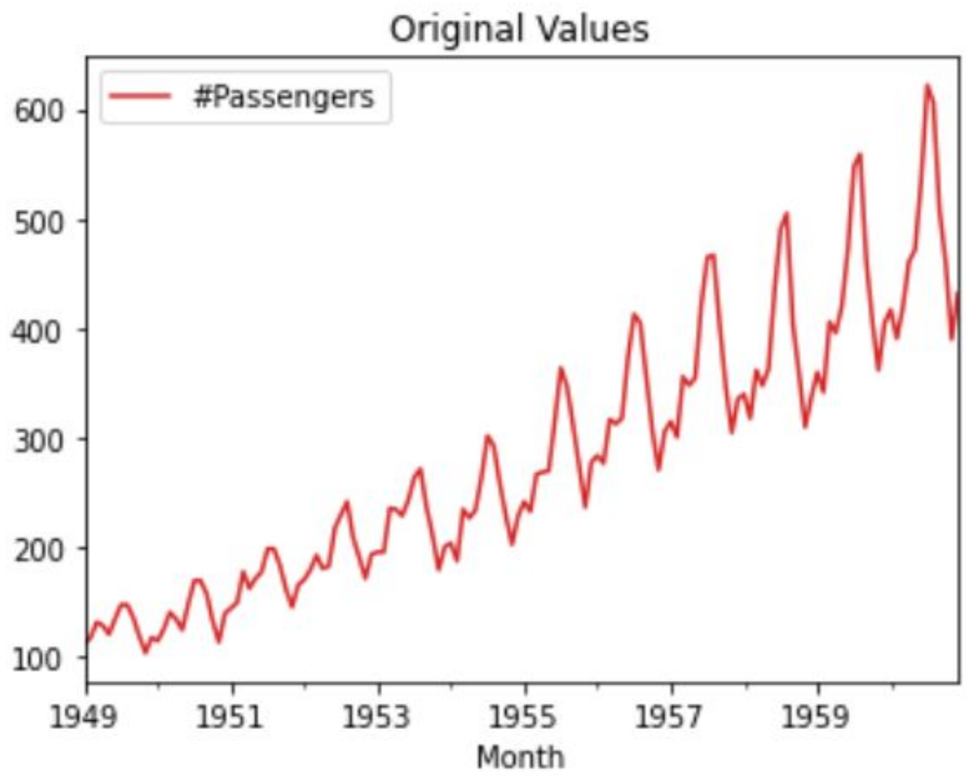
Removing Seasonality

we will remove the seasonality from the time series using the rolling mean differencing as shown in the example below.

```
#removing seasonality from the time series
rolling_mean_detrended_diff = rolling_mean_detrended - rolling_mean_detrended.shift()

ax1 = plt.subplot(121)
rolling_mean_detrended_diff.plot(figsize=(12,4),
                                 color="tab:red",
                                 title="rolled and differenced time series",
                                 ax=ax1)
ax2 = plt.subplot(122)
data.plot(figsize=(12,4),
          color="tab:red",
          title="Original Values",
          ax=ax2)
```

Removing Seasonality





Hands on

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

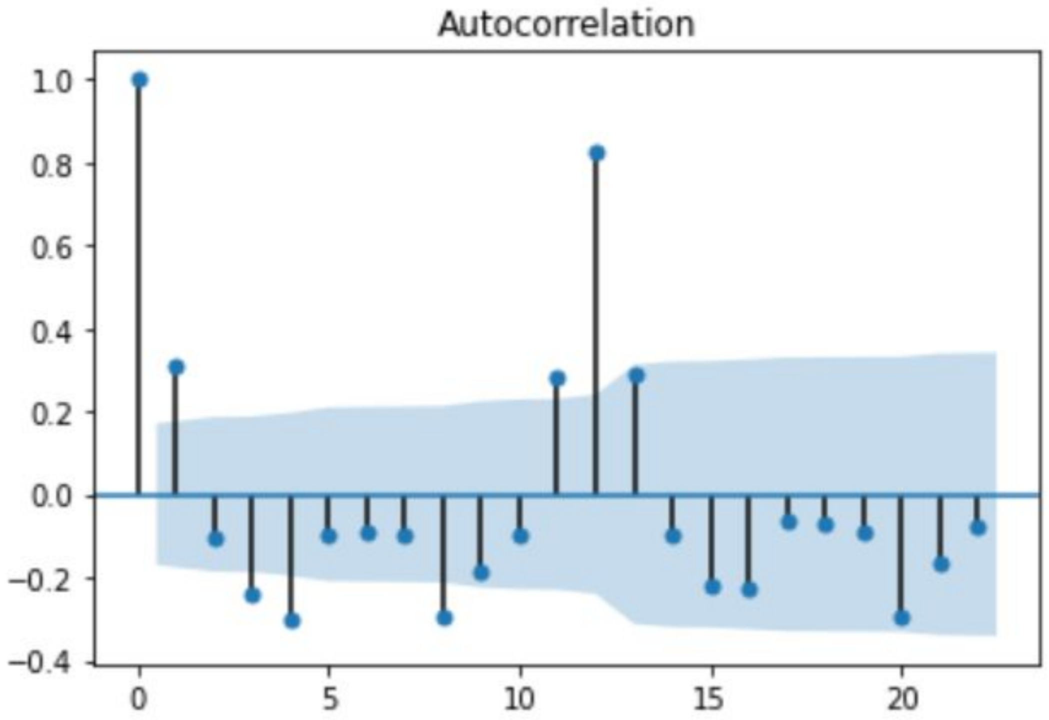
```
result = adfuller(rolling_mean_detrended_diff['#Passengers'].dropna())
print(result)
```

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.

Autocorrelation & Partial Autocorrelation

Autocorrelation

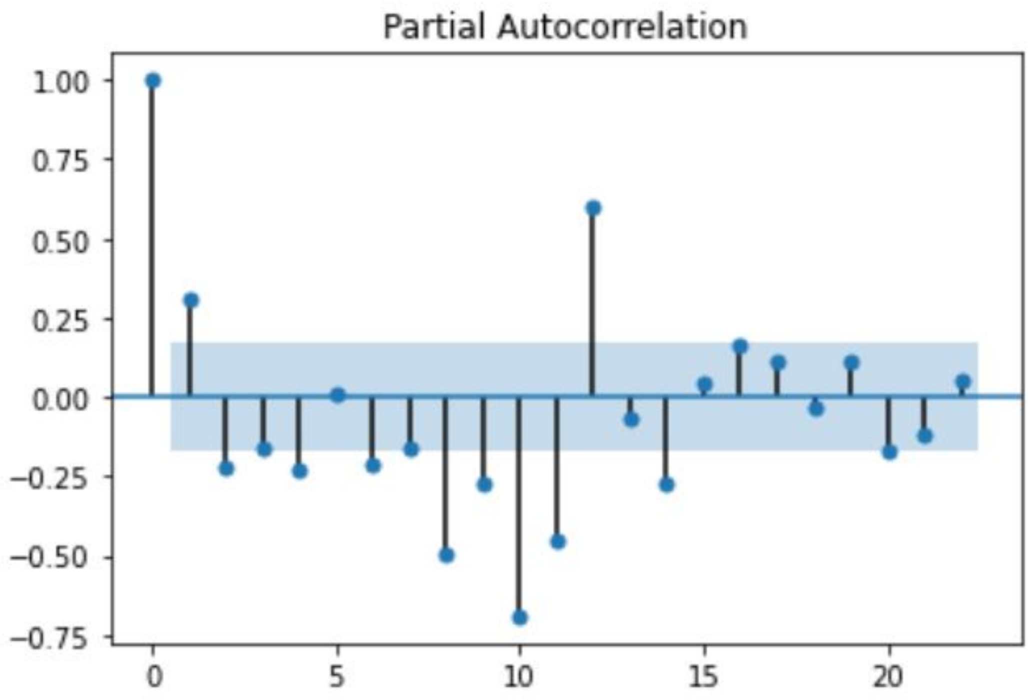
```
from pandas.plotting import autocorrelation_plot
from statsmodels.graphics.tsaplots import plot_acf

plot_acf(rolling_mean_detrended_diff['#Passengers'])
```



Autocorrelation and partial autocorrelation plots can be used to determine the ARIMA orders for the AR and MA models.

Partial Autocorrelation

```
from statsmodels.graphics.tsaplots import plot_pacf

plot_pacf(rolling_mean_detrended_diff['#Passengers'])
```



Pyramid ARIMA

What is Pyramid ARIMA?

The pyramid ARIMA or pmdarima package from python programming can be used to compute the order of the ARIMA model using the auto_arima module.

```
!pip install pmdarima
from pmdarima import auto_arima

order = auto_arima(rolling_mean_detrended_diff['#Passengers'], trace=True)
order.summary()
```

The auto_arima model searches the best order for the model that has the lowest AIC value.

Auto ARIMA in Pyramid ARIMA

```
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=inf, Time=0.17 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=inf, Time=0.18 sec
 ARIMA(0,0,2)(0,0,0)[0]             : AIC=1295.960, Time=0.05 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=1296.291, Time=0.04 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=1242.347, Time=0.16 sec
 ARIMA(3,0,2)(0,0,0)[0]             : AIC=inf, Time=0.37 sec
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=inf, Time=0.37 sec
 ARIMA(1,0,3)(0,0,0)[0]             : AIC=inf, Time=0.25 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=inf, Time=0.14 sec
 ARIMA(3,0,3)(0,0,0)[0]             : AIC=1235.785, Time=0.47 sec
 ARIMA(4,0,3)(0,0,0)[0]             : AIC=inf, Time=0.40 sec
 ARIMA(3,0,4)(0,0,0)[0]             : AIC=inf, Time=0.53 sec
 ARIMA(2,0,4)(0,0,0)[0]             : AIC=inf, Time=0.95 sec
 ARIMA(4,0,2)(0,0,0)[0]             : AIC=inf, Time=0.55 sec
 ARIMA(4,0,4)(0,0,0)[0]             : AIC=inf, Time=1.28 sec
 ARIMA(3,0,3)(0,0,0)[0] intercept   : AIC=1236.433, Time=2.16 sec

Best model:  ARIMA(3,0,3)(0,0,0)[0]
Total fit time: 9.725 seconds
```

From the output, the best order for the ARIMA model is (3,0,3), for (p,d,q) respectively.

ARIMA Model For Time Series Forecasting

What is ARIMA?

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

AR	I	MA
p	d	q

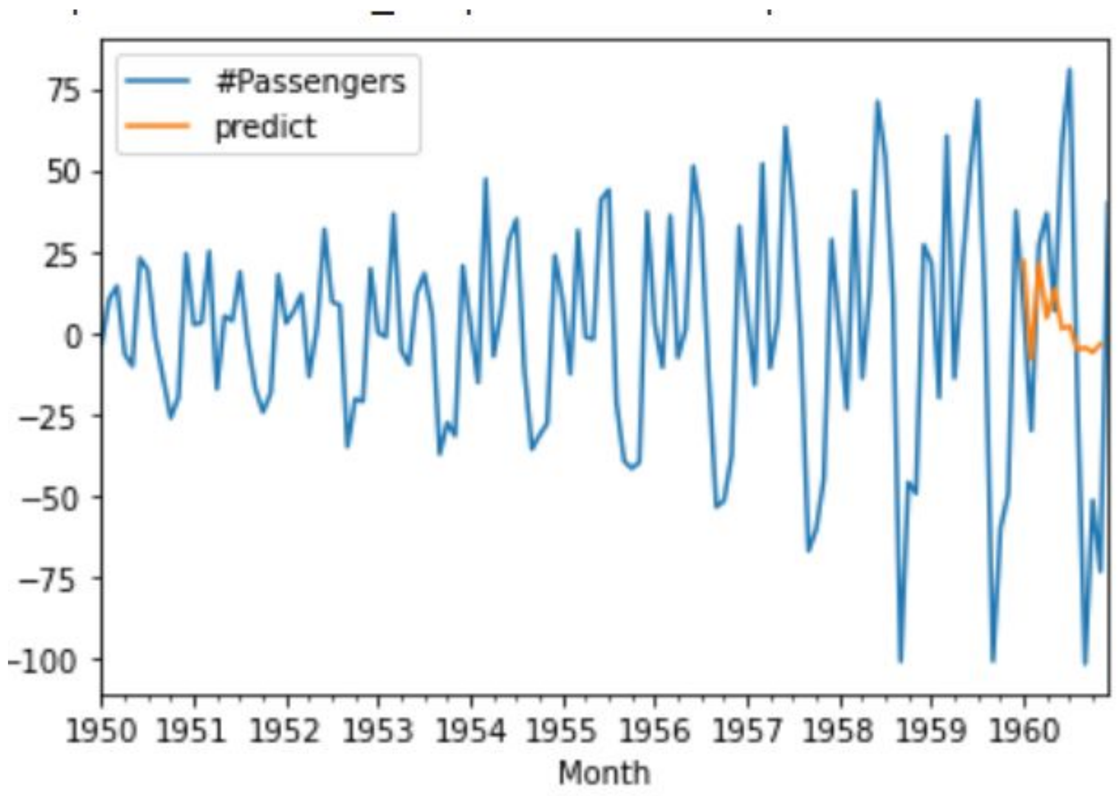
Here, p, d and q are the order of AR, order of differencing and order of MA respectively. We have already calculated these values using the auto_arima, or we can deduce these values using the ACF and PCF.

Hands on

```
from statsmodels.tsa.arima_model import ARIMA

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

model = ARIMA(train, order=(3,0,3))
model_fit = model.fit()
model_fit.summary()
```

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

Hands on



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

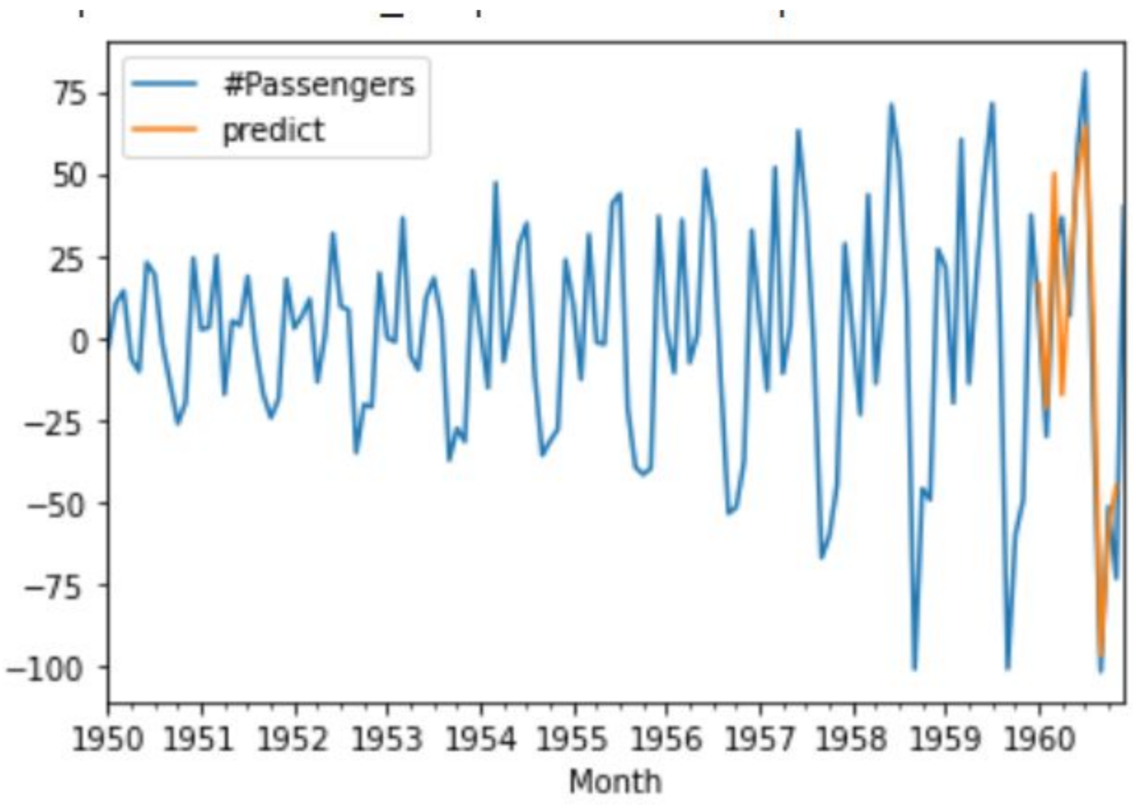
What is Seasonal ARIMA?

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

model = SARIMAX(train, order=(3,0,3), seasonal_order=(3,0,3,12))
model = model.fit()
```

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

Hands on

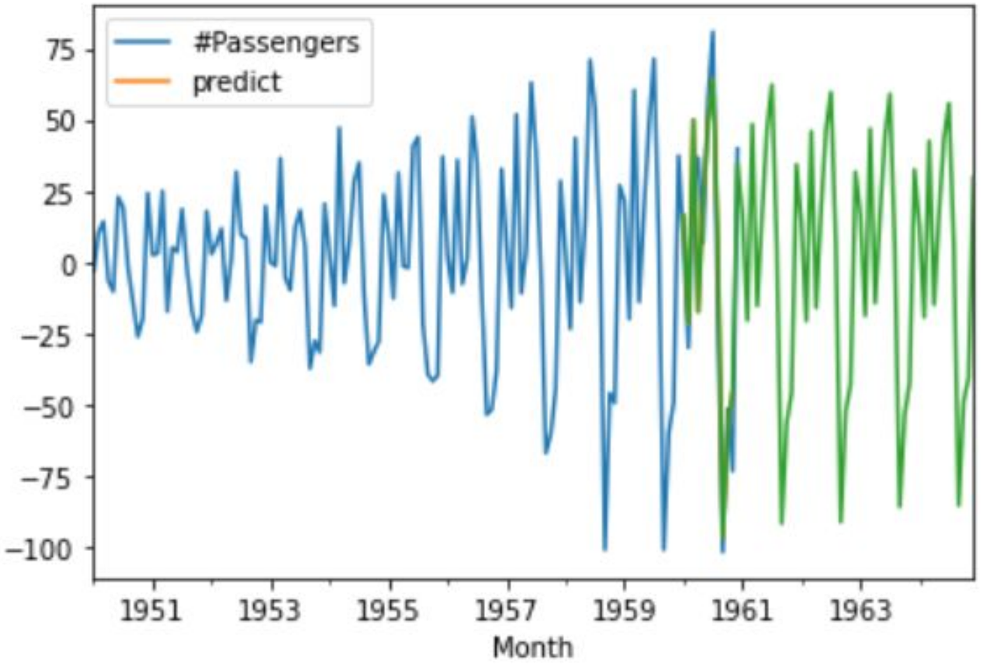


Here, we can see the predicted values on the test set are more accurate than the ARIMA model. Therefore we have successfully created a Time series forecast model. Now we will use this model to forecast the time series.

Inferences

Inferences

```
#predicting the projections for the next 5 years

forecast = model.forecast(steps=60)
rolling_mean_detrended_diff.plot()
forecast.plot()
```



We had trained the model on the rolling_mean_detrended_diff values, 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.

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

support@intellipaat.com

24/7 Chat with Our Course Advisor