**Complete Guide on Time Series Analysis in Python**

Hello friends,

As the name implies, this notebook is all about **Time Series Analysis**. A time series is a series of data points recorded at different time-intervals. The time series analysis means analyzing the time series data using various statistical tools and techniques.

So, let's get started.

**I hope you find this notebook useful and your UPVOTES keep me motivated.**

**Table of Contents**

1. [Introduction to Time Series Analysis](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#1)
2. [Types of data](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#2)
3. [Time Series terminology](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#3)
4. [Time Series Analysis](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#4)
5. [Visualize the Time Series](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#5)
6. [Patterns in a Time Series](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#6)
7. [Additive and Multiplicative Time Series](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#7)
8. [Decomposition of a Time Series](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#8)
9. [Stationary and Non-Stationary Time Series](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#9)
10. [How to make a time series stationary](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#10)
11. [How to test for stationarity](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#11)
    * 11.1 [Augmented Dickey Fuller test (ADF Test)](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#11.1)
    * 11.2 [Kwiatkowski-Phillips-Schmidt-Shin – KPSS test (trend stationary)](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#11.2)
    * 11.3 [Philips Perron test (PP Test)](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#11.3)
12. [Difference between white noise and a stationary series](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#12)
13. [Detrend a Time Series](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#13)
14. [Deseasonalize a Time Series](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#14)
15. [How to test for seasonality of a time series](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#15)
16. [Autocorrelation and Partial Autocorrelation Functions](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#16)
17. [Computation of Partial Autocorrelation Function](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#17)
18. [Lag Plots](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#18)
19. [Granger Causality Test](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#19)
20. [Smoothening a Time Series](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#20)
21. [References](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#21)

**1. Introduction to Time-Series Analysis**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* A **time-series** data is a series of data points or observations recorded at different or regular time intervals. In general, a time series is a sequence of data points taken at equally spaced time intervals. The frequency of recorded data points may be hourly, daily, weekly, monthly, quarterly or annually.
* **Time-Series Forecasting** is the process of using a statistical model to predict future values of a time-series based on past results.
* A time series analysis encompasses statistical methods for analyzing time series data. These methods enable us to extract meaningful statistics, patterns and other characteristics of the data. Time series are visualized with the help of line charts. So, time series analysis involves understanding inherent aspects of the time series data so that we can create meaningful and accurate forecasts.
* Applications of time series are used in statistics, finance or business applications. A very common example of time series data is the daily closing value of the stock index like NASDAQ or Dow Jones. Other common applications of time series are sales and demand forecasting, weather forecasting, econometrics, signal processing, pattern recognition and earthquake prediction.

**Components of a Time-Series**

* **Trend** - The trend shows a general direction of the time series data over a long period of time. A trend can be increasing(upward), decreasing(downward), or horizontal(stationary).
* **Seasonality** - The seasonality component exhibits a trend that repeats with respect to timing, direction, and magnitude. Some examples include an increase in water consumption in summer due to hot weather conditions.
* **Cyclical Component** - These are the trends with no set repetition over a particular period of time. A cycle refers to the period of ups and downs, booms and slums of a time series, mostly observed in business cycles. These cycles do not exhibit a seasonal variation but generally occur over a time period of 3 to 12 years depending on the nature of the time series.
* **Irregular Variation** - These are the fluctuations in the time series data which become evident when trend and cyclical variations are removed. These variations are unpredictable, erratic, and may or may not be random.
* **ETS Decomposition** - ETS Decomposition is used to separate different components of a time series. The term ETS stands for Error, Trend and Seasonality.
* In this notebook, I conduct time series analysis of video game sales over time.

**2. Types of data**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

As stated above, the time series analysis is the statistical analysis of the time series data. A time series data means that data is recorded at different time periods or intervals. The time series data may be of three types:-

1 **Time series data** - The observations of the values of a variable recorded at different points in time is called time series data.

2 **Cross sectional data** - It is the data of one or more variables recorded at the same point in time.

3 **Pooled data**- It is the combination of time series data and cross sectional data.

**3. Time Series terminology**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

There are various terms and concepts in time series that we should know. These are as follows:-

1 **Dependence**- It refers to the association of two observations of the same variable at prior time periods.

2 **Stationarity**- It shows the mean value of the series that remains constant over the time period. If past effects accumulate and the values increase towards infinity then stationarity is not met.

3 **Differencing**- Differencing is used to make the series stationary and to control the auto-correlations. There may be some cases in time series analyses where we do not require differencing and over-differenced series can produce wrong estimates.

4 **Specification** - It may involve the testing of the linear or non-linear relationships of dependent variables by using time series models such as ARIMA models.

5 **Exponential Smoothing** - Exponential smoothing in time series analysis predicts the one next period value based on the past and current value. It involves averaging of data such that the non-systematic components of each individual case or observation cancel out each other. The exponential smoothing method is used to predict the short term prediction.

6 **Curve fitting** - Curve fitting regression in time series analysis is used when data is in a non-linear relationship.

7 **ARIMA** - ARIMA stands for Auto Regressive Integrated Moving Average.

**4. Time Series Analysis**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

**4.1 Basic set up**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

In [1]:

*# This Python 3 environment comes with many helpful analytics libraries installed*

*# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python*

*# For example, here's several helpful packages to load*

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

import matplotlib as mpl

import matplotlib.pyplot as plt *# data visualization*

import seaborn as sns *# statistical data visualization*

*# Input data files are available in the read-only "../input/" directory*

*# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory*

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

*# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"*

*# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session*

/kaggle/input/air-passengers/AirPassengers.csv

/kaggle/input/dataset/dataset.txt

**4.2 Import data**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

In [2]:

path = '/kaggle/input/air-passengers/AirPassengers.csv'

df = pd.read\_csv(path)

df.head()

Out[2]:

|  | Month | #Passengers |
| --- | --- | --- |
| 0 | 1949-01 | 112 |
| 1 | 1949-02 | 118 |
| 2 | 1949-03 | 132 |
| 3 | 1949-04 | 129 |
| 4 | 1949-05 | 121 |

* We should rename the column names.

In [3]:

df.columns = ['Date','Number of Passengers']

df.head()

Out[3]:

|  | Date | Number of Passengers |
| --- | --- | --- |
| 0 | 1949-01 | 112 |
| 1 | 1949-02 | 118 |
| 2 | 1949-03 | 132 |
| 3 | 1949-04 | 129 |
| 4 | 1949-05 | 121 |

**5. Visualize the Time Series**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

In [4]:

def plot\_df(df, x, y, title="", xlabel='Date', ylabel='Number of Passengers', dpi=100):

plt.figure(figsize=(15,4), dpi=dpi)

plt.plot(x, y, color='tab:red')

plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)

plt.show()

plot\_df(df, x=df['Date'], y=df['Number of Passengers'], title='Number of US Airline passengers from 1949 to 1960')

* Since all the values are positive, we can show this on both sides of the Y axis to emphasize the growth.

In [5]:

x = df['Date'].values

y1 = df['Number of Passengers'].values

*# Plot*

fig, ax = plt.subplots(1, 1, figsize=(16,5), dpi= 120)

plt.fill\_between(x, y1=y1, y2=-y1, alpha=0.5, linewidth=2, color='seagreen')

plt.ylim(-800, 800)

plt.title('Air Passengers (Two Side View)', fontsize=16)

plt.hlines(y=0, xmin=np.min(df['Date']), xmax=np.max(df['Date']), linewidth=.5)

plt.show()

* It can be seen that its a monthly time series and follows a certain repetitive pattern every year. So, we can plot each year as a separate line in the same plot. This let us compare the year wise patterns side-by-side.

**6. Patterns in a Time Series**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* Any time series visualization may consist of the following components: **Base Level + Trend + Seasonality + Error**.

**Trend**

* A **trend** is observed when there is an increasing or decreasing slope observed in the time series.

**Seasonality**

* A **seasonality** is observed when there is a distinct repeated pattern observed between regular intervals due to seasonal factors. It could be because of the month of the year, the day of the month, weekdays or even time of the day.

However, It is not mandatory that all time series must have a trend and/or seasonality. A time series may not have a distinct trend but have a seasonality and vice-versa.

In [6]:

def plot\_df(df, x, y, title="", xlabel='Date', ylabel='Number of Passengers', dpi=100):

plt.figure(figsize=(15,4), dpi=dpi)

plt.plot(x, y, color='blue')

plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)

plt.show()

plot\_df(df, x=df['Date'], y=df['Number of Passengers'], title='Trend and Seasonality')

**Cyclic behaviour**

* Another important thing to consider is the **cyclic behaviour**. It happens when the rise and fall pattern in the series does not happen in fixed calendar-based intervals. We should not confuse 'cyclic' effect with 'seasonal' effect.
* If the patterns are not of fixed calendar based frequencies, then it is cyclic. Because, unlike the seasonality, cyclic effects are typically influenced by the business and other socio-economic factors.

**7. Additive and Multiplicative Time Series**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* We may have different combinations of trends and seasonality. Depending on the nature of the trends and seasonality, a time series can be modeled as an additive or multiplicative time series. Each observation in the series can be expressed as either a sum or a product of the components.

**Additive time series:**

Value = Base Level + Trend + Seasonality + Error

**Multiplicative Time Series:**

Value = Base Level x Trend x Seasonality x Error

**8. Decomposition of a Time Series**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* Decomposition of a time series can be performed by considering the series as an additive or multiplicative combination of the base level, trend, seasonal index and the residual term.
* The seasonal\_decompose in statsmodels implements this conveniently.

In [7]:

from statsmodels.tsa.seasonal import seasonal\_decompose

from dateutil.parser import parse

*# Multiplicative Decomposition*

multiplicative\_decomposition = seasonal\_decompose(df['Number of Passengers'], model='multiplicative', period=30)

*# Additive Decomposition*

additive\_decomposition = seasonal\_decompose(df['Number of Passengers'], model='additive', period=30)

*# Plot*

plt.rcParams.update({'figure.figsize': (16,12)})

multiplicative\_decomposition.plot().suptitle('Multiplicative Decomposition', fontsize=16)

plt.tight\_layout(rect=[0, 0.03, 1, 0.95])

additive\_decomposition.plot().suptitle('Additive Decomposition', fontsize=16)

plt.tight\_layout(rect=[0, 0.03, 1, 0.95])

plt.show()

* If we look at the residuals of the additive decomposition closely, it has some pattern left over.
* The multiplicative decomposition, looks quite random which is good. So ideally, multiplicative decomposition should be preferred for this particular series.

**9. Stationary and Non-Stationary Time Series**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* Now, we wil discuss **Stationary and Non-Stationary Time Series**. **Stationarity** is a property of a time series. A stationary series is one where the values of the series is not a function of time. So, the values are independent of time.
* Hence the statistical properties of the series like mean, variance and autocorrelation are constant over time. Autocorrelation of the series is nothing but the correlation of the series with its previous values.
* A stationary time series is independent of seasonal effects as well.
* Now, we will plot some examples of stationary and non-stationary time series for clarity.

image source : <https://www.machinelearningplus.com/wp-content/uploads/2019/02/stationary-and-non-stationary-time-series-865x569.png?ezimgfmt=ng:webp/ngcb1>

* We can covert any non-stationary time series into a stationary one by applying a suitable transformation. Mostly statistical forecasting methods are designed to work on a stationary time series. The first step in the forecasting process is typically to do some transformation to convert a non-stationary series to stationary.

**10. How to make a time series stationary?**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* We can apply some sort of transformation to make the time-series stationary. These transformation may include:

1. Differencing the Series (once or more)
2. Take the log of the series
3. Take the nth root of the series
4. Combination of the above

* The most commonly used and convenient method to stationarize the series is by differencing the series at least once until it becomes approximately stationary.

**10.1 Introduction to Differencing**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* If Y\_t is the value at time t, then the first difference of Y = Yt – Yt-1. In simpler terms, differencing the series is nothing but subtracting the next value by the current value.
* If the first difference doesn’t make a series stationary, we can go for the second differencing and so on.
* For example, consider the following series: [1, 5, 2, 12, 20]
* First differencing gives: [5-1, 2-5, 12-2, 20-12] = [4, -3, 10, 8]
* Second differencing gives: [-3-4, -10-3, 8-10] = [-7, -13, -2]

**10.2 Reasons to convert a non-stationary series into stationary one before forecasting**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

There are reasons why we want to convert a non-stationary series into a stationary one. These are given below:

* Forecasting a stationary series is relatively easy and the forecasts are more reliable.
* An important reason is, autoregressive forecasting models are essentially linear regression models that utilize the lag(s) of the series itself as predictors.
* We know that linear regression works best if the predictors (X variables) are not correlated against each other. So, stationarizing the series solves this problem since it removes any persistent autocorrelation, thereby making the predictors(lags of the series) in the forecasting models nearly independent.

**11. How to test for stationarity?**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* The stationarity of a series can be checked by looking at the plot of the series.
* Another method is to split the series into 2 or more contiguous parts and computing the summary statistics like the mean, variance and the autocorrelation. If the stats are quite different, then the series is not likely to be stationary.
* There are several quantitative methods we can use to determine if a given series is stationary or not. This can be done using statistical tests called [Unit Root Tests](https://en.wikipedia.org/wiki/Unit_root). This test checks if a time series is non-stationary and possess a unit root.
* There are multiple implementations of Unit Root tests like:

**1. Augmented Dickey Fuller test (ADF Test)**

**2. Kwiatkowski-Phillips-Schmidt-Shin – KPSS test (trend stationary)**

**3. Philips Perron test (PP Test)**

**11.1 Augmented Dickey Fuller test (ADF Test)**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* **Augmented Dickey Fuller test or (ADF Test)** is the most commonly used test to detect stationarity. Here, we assume that the null hypothesis is the time series possesses a unit root and is non-stationary. Then, we collect evidence to support or reject the null hypothesis. So, if we find that the p-value in ADF test is less than the significance level (0.05), we reject the null hypothesis.
* Feel free to check the following links to learn more about the ADF Test.

<https://en.wikipedia.org/wiki/Augmented_Dickey%E2%80%93Fuller_test>

<https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/>

<https://machinelearningmastery.com/time-series-data-stationary-python/>

<http://www.insightsbot.com/augmented-dickey-fuller-test-in-python/>

<https://nwfsc-timeseries.github.io/atsa-labs/sec-boxjenkins-aug-dickey-fuller.html>

<https://www.statisticshowto.com/adf-augmented-dickey-fuller-test/>

**11.2 Kwiatkowski-Phillips-Schmidt-Shin – KPSS test (trend stationary)**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* The KPSS test, on the other hand, is used to test for trend stationarity. The null hypothesis and the P-Value interpretation is just the opposite of ADH test.
* Interested readers can learn more about the KPSS test from the below links:

<https://en.wikipedia.org/wiki/KPSS_test>

<https://www.machinelearningplus.com/time-series/kpss-test-for-stationarity/>

<https://www.statisticshowto.com/kpss-test/>

<https://nwfsc-timeseries.github.io/atsa-labs/sec-boxjenkins-kpss.html>

**11.3 Philips Perron test (PP Test)**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* The Philips Perron or PP test is a [unit root test](https://en.wikipedia.org/wiki/Unit_root). It is used in the [time series analysis](https://en.wikipedia.org/wiki/Time_series) to test the [null hypothesis](https://en.wikipedia.org/wiki/Null_hypothesis) that a time series is integrated of order 1. It is built on the ADF test discussed above.
* For more information on PP test, please visit the following links:

<https://en.wikipedia.org/wiki/Phillips%E2%80%93Perron_test>

<https://www.mathworks.com/help/econ/pptest.html>

<https://people.bath.ac.uk/hssjrh/Phillips%20Perron.pdf>

<https://www.stata.com/manuals13/tspperron.pdf>

**12. Difference between white noise and a stationary series**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* Like a stationary series, the white noise is also not a function of time. So, its mean and variance does not change over time. But the difference is that, the white noise is completely random with a mean of 0. In white noise there is no pattern.
* Mathematically, a sequence of completely random numbers with mean zero is a white noise.

In [8]:

rand\_numbers = np.random.randn(1000)

pd.Series(rand\_numbers).plot(title='Random White Noise', color='b')

Out[8]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fc20b725a90>

**13. Detrend a Time Series**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* Detrending a time series means to remove the trend component from the time series. There are multiple approaches of doing this as listed below:

1. Subtract the line of best fit from the time series. The line of best fit may be obtained from a linear regression model with the time steps as the predictor. For more complex trends, we may want to use quadratic terms (x^2) in the model.
2. We subtract the trend component obtained from time series decomposition.
3. Subtract the mean.
4. Apply a filter like Baxter-King filter(statsmodels.tsa.filters.bkfilter) or the Hodrick-Prescott Filter (statsmodels.tsa.filters.hpfilter) to remove the moving average trend lines or the cyclical components.

Now, we will implement the first two methods to detrend a time series.

In [9]:

*# Using scipy: Subtract the line of best fit*

from scipy import signal

detrended = signal.detrend(df['Number of Passengers'].values)

plt.plot(detrended)

plt.title('Air Passengers detrended by subtracting the least squares fit', fontsize=16)

Out[9]:

Text(0.5, 1.0, 'Air Passengers detrended by subtracting the least squares fit')

In [10]:

*# Using statmodels: Subtracting the Trend Component*

from statsmodels.tsa.seasonal import seasonal\_decompose

result\_mul = seasonal\_decompose(df['Number of Passengers'], model='multiplicative', period=30)

detrended = df['Number of Passengers'].values - result\_mul.trend

plt.plot(detrended)

plt.title('Air Passengers detrended by subtracting the trend component', fontsize=16)

Out[10]:

Text(0.5, 1.0, 'Air Passengers detrended by subtracting the trend component')

**14. Deseasonalize a Time Series**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

There are multiple approaches to deseasonalize a time series. These approaches are listed below:

* 1. Take a moving average with length as the seasonal window. This will smoothen in series in the process.
  2. Seasonal difference the series (subtract the value of previous season from the current value).
  3. Divide the series by the seasonal index obtained from STL decomposition.

If dividing by the seasonal index does not work well, we will take a log of the series and then do the deseasonalizing. We will later restore to the original scale by taking an exponential.

In [11]:

*# Subtracting the Trend Component*

*# Time Series Decomposition*

result\_mul = seasonal\_decompose(df['Number of Passengers'], model='multiplicative', period=30)

*# Deseasonalize*

deseasonalized = df['Number of Passengers'].values / result\_mul.seasonal

*# Plot*

plt.plot(deseasonalized)

plt.title('Air Passengers Deseasonalized', fontsize=16)

plt.plot()

Out[11]:

[]

**15. How to test for seasonality of a time series?**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

The common way to test for seasonality of a time series is to plot the series and check for repeatable patterns in fixed time intervals. So, the types of seasonality is determined by the clock or the calendar.

1. Hour of day
2. Day of month
3. Weekly
4. Monthly
5. Yearly

However, if we want a more definitive inspection of the seasonality, use the **Autocorrelation Function (ACF) plot**. There is a strong seasonal pattern, the ACF plot usually reveals definitive repeated spikes at the multiples of the seasonal window.

In [12]:

*# Test for seasonality*

from pandas.plotting import autocorrelation\_plot

*# Draw Plot*

plt.rcParams.update({'figure.figsize':(10,6), 'figure.dpi':120})

autocorrelation\_plot(df['Number of Passengers'].tolist())

Out[12]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fc208cdd5d0>

Alternately, if we want a statistical test, the [CHTest](https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.CHTest.html" \l "pmdarima.arima.CHTest) can determine if seasonal differencing is required to stationarize the series.

**16. Autocorrelation and Partial Autocorrelation Functions**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* **Autocorrelation** is simply the correlation of a series with its own lags. If a series is significantly autocorrelated, that means, the previous values of the series (lags) may be helpful in predicting the current value.
* **Partial Autocorrelation** also conveys similar information but it conveys the pure correlation of a series and its lag, excluding the correlation contributions from the intermediate lags.

In [13]:

from statsmodels.tsa.stattools import acf, pacf

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

*# Draw Plot*

fig, axes = plt.subplots(1,2,figsize=(16,3), dpi= 100)

plot\_acf(df['Number of Passengers'].tolist(), lags=50, ax=axes[0])

plot\_pacf(df['Number of Passengers'].tolist(), lags=50, ax=axes[1])

/opt/conda/lib/python3.7/site-packages/statsmodels/regression/linear\_model.py:1406: RuntimeWarning: invalid value encountered in sqrt

return rho, np.sqrt(sigmasq)

Out[13]:

**17. Computation of Partial Autocorrelation Function**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* The partial autocorrelation function of lag (k) of a series is the coefficient of that lag in the autoregression equation of Y. The autoregressive equation of Y is nothing but the linear regression of Y with its own lags as predictors.
* For example, if **Y\_t** is the current series and **Y\_t-1** is the lag 1 of Y, then the partial autocorrelation of **lag 3 (Y\_t-3)** is the coefficient α3𝛼3 of Y\_t-3 in the following equation:

image source : <https://www.machinelearningplus.com/wp-content/uploads/2019/02/12_5_Autoregression_Equation-min.png?ezimgfmt=ng:webp/ngcb1>

**18. Lag Plots**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* A **Lag plot** is a scatter plot of a time series against a lag of itself. It is normally used to check for autocorrelation. If there is any pattern existing in the series, the series is autocorrelated. If there is no such pattern, the series is likely to be random white noise.

In [14]:

from pandas.plotting import lag\_plot

plt.rcParams.update({'ytick.left' : False, 'axes.titlepad':10})

*# Plot*

fig, axes = plt.subplots(1, 4, figsize=(10,3), sharex=True, sharey=True, dpi=100)

for i, ax **in** enumerate(axes.flatten()[:4]):

lag\_plot(df['Number of Passengers'], lag=i+1, ax=ax, c='firebrick')

ax.set\_title('Lag ' + str(i+1))

fig.suptitle('Lag Plots of Air Passengers', y=1.05)

plt.show()

**19. Granger Causality Test**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

* **Granger causality test** is used to determine if one time series will be useful to forecast another. It is based on the idea that if X causes Y, then the forecast of Y based on previous values of Y AND the previous values of X should outperform the forecast of Y based on previous values of Y alone.
* So, **Granger causality test** should not be used to test if a lag of Y causes Y. Instead, it is generally used on exogenous (not Y lag) variables only. It is implemented in the statsmodel package.
* It accepts a 2D array with 2 columns as the main argument. The values are in the first column and the predictor (X) is in the second column. The Null hypothesis is that the series in the second column, does not Granger cause the series in the first. If the P-Values are less than a significance level (0.05) then we reject the null hypothesis and conclude that the said lag of X is indeed useful. The second argument maxlag says till how many lags of Y should be included in the test.

In [15]:

from statsmodels.tsa.stattools import grangercausalitytests

data = pd.read\_csv('/kaggle/input/dataset/dataset.txt')

data['date'] = pd.to\_datetime(data['date'])

data['month'] = data.date.dt.month

grangercausalitytests(data[['value', 'month']], maxlag=2)

Granger Causality

number of lags (no zero) 1

ssr based F test: F=54.7797 , p=0.0000 , df\_denom=200, df\_num=1

ssr based chi2 test: chi2=55.6014 , p=0.0000 , df=1

likelihood ratio test: chi2=49.1426 , p=0.0000 , df=1

parameter F test: F=54.7797 , p=0.0000 , df\_denom=200, df\_num=1

Granger Causality

number of lags (no zero) 2

ssr based F test: F=162.6989, p=0.0000 , df\_denom=197, df\_num=2

ssr based chi2 test: chi2=333.6567, p=0.0000 , df=2

likelihood ratio test: chi2=196.9956, p=0.0000 , df=2

parameter F test: F=162.6989, p=0.0000 , df\_denom=197, df\_num=2

Out[15]:

{1: ({'ssr\_ftest': (54.7796748355736, 3.661425871353102e-12, 200.0, 1),

'ssr\_chi2test': (55.6013699581072, 8.876175235021508e-14, 1),

'lrtest': (49.14260233004984, 2.38014300604565e-12, 1),

'params\_ftest': (54.77967483557362, 3.6614258713530505e-12, 200.0, 1.0)},

[<statsmodels.regression.linear\_model.RegressionResultsWrapper at 0x7fc20bdc81d0>,

<statsmodels.regression.linear\_model.RegressionResultsWrapper at 0x7fc20bdc8090>,

array([[0., 1., 0.]])]),

2: ({'ssr\_ftest': (162.6989179987325, 1.913323508685587e-42, 197.0, 2),

'ssr\_chi2test': (333.6566643222738, 3.52676008812746e-73, 2),

'lrtest': (196.9955927718221, 1.670900349911483e-43, 2),

'params\_ftest': (162.69891799873264, 1.913323508685476e-42, 197.0, 2.0)},

[<statsmodels.regression.linear\_model.RegressionResultsWrapper at 0x7fc20bdc8390>,

<statsmodels.regression.linear\_model.RegressionResultsWrapper at 0x7fc20bdc8810>,

array([[0., 0., 1., 0., 0.],

[0., 0., 0., 1., 0.]])])}

* In the above case, the p-values are zero for all tests. So the ‘month’ indeed can be used to forecast the values.

**20. Smoothening a Time Series**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

Smoothening of a time series may be useful in the following circumstances:

* Reducing the effect of noise in a signal get a fair approximation of the noise-filtered series.
* The smoothed version of series can be used as a feature to explain the original series itself.
* Visualize the underlying trend better.

We can smoothen a time series using the following methods:

* Take a moving average
* Do a LOESS smoothing (Localized Regression)
* Do a LOWESS smoothing (Locally Weighted Regression)

**Moving Average**

* **Moving average** is the average of a rolling window of defined width. We must choose the window-width wisely, because, large window-size will over-smooth the series. For example, a window-size equal to the seasonal duration (ex: 12 for a month-wise series), will effectively nullify the seasonal effect.

linkcode

**Localized Regression**

* LOESS, short for ‘Localized Regression’ fits multiple regressions in the local neighborhood of each point. It is implemented in the statsmodels package, where you can control the degree of smoothing using frac argument which specifies the percentage of data points nearby that should be considered to fit a regression model.

**21. References**

[Table of Contents](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0.1)

The concepts and code in this notebook is taken from the following websites:-

1. <https://www.machinelearningplus.com/time-series/time-series-analysis-python/>
2. <https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b>
3. <https://towardsdatascience.com/time-series-analysis-in-python-an-introduction-70d5a5b1d52a>
4. <https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-visualization-with-python-3>

So, now we will come to the end of this notebook.

I hope you find this notebook useful and enjoyable.

Your comments and feedback are most welcome.

Thank you

[Go to Top](https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python#0)