Time Series Analysis in Python

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.

Table of Contents

1. Introduction to Time-Series Analysis

Time-series data consists of 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.

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 data is typically visualized using line charts. Therefore, time series analysis involves understanding the inherent aspects of time series data to create meaningful and accurate forecasts.

Applications of time series are found in statistics, finance, and various business applications. A common example of time series data is the daily closing value of stock indices like NASDAQ or Dow Jones. Other common applications of time series include sales and demand forecasting, weather forecasting, econometrics, signal processing, pattern recognition, and earthquake prediction.

Components of a Time-Series

Trend - This component demonstrates the general direction of the time series data over an extended period. A trend can be upward (increasing), downward (decreasing), or horizontal (stationary).

Seasonality - The seasonality component exhibits a recurring trend concerning timing, direction, and magnitude. For instance, an example would be increased water consumption during summer due to hot weather conditions.

Cyclical Component - These trends lack a specific repetition over a defined period. A cycle refers to the periods of ups and downs, booms and slumps within a time series, often observed in business cycles. These cycles do not display seasonal variation but typically occur over a 3 to 12-year span, depending on the nature of the time series.

Irregular Variation - These fluctuations in time series data become apparent when trend and cyclical variations are removed. They are unpredictable, erratic, and may or may not follow a pattern.

ETS Decomposition - ETS Decomposition is a method used to disentangle different components of a time series. The term "ETS" stands for Error, Trend, and Seasonality.

2. Types of data

Time series analysis involves statistically analyzing data recorded at different time periods or intervals. Time series data refers to observations of variable values recorded at various points in time. There are three primary types of data:

Time series data - This consists of observations of variable values recorded at different points in time.

Cross-sectional data - This data comprises one or more variables recorded at the same point in time.

Pooled data - Pooled data is a combination of both time series data and cross-sectional data.

3. Time Series terminology

Dependence - Refers to the relationship between two observations of the same variable at prior time periods.

Stationarity - Indicates that the mean value of the series remains constant over the entire time span. If the values tend to accumulate past effects and increase infinitely, stationarity is not achieved.

Differencing - Used to make a series stationary and manage auto-correlations. In some cases of time series analysis, differencing may not be necessary, as an over-differenced series can lead to inaccurate estimates.

Specification - Involves testing the linear or non-linear relationships of dependent variables using time series models such as ARIMA models.

Exponential Smoothing - Predicts the next period's value based on past and current values. It involves averaging data to cancel out non-systematic components in each individual observation. This method is used for short-term predictions in time series analysis.

Curve fitting - Utilized in time series analysis when data demonstrates a non-linear relationship, employing regression techniques to fit a curve to the data.

ARIMA - Short for Auto Regressive Integrated Moving Average, it's a popular method used in time series analysis for modeling and forecasting data.

4. Time Series Analysis

4.1 Basic set up

In [ ]:
```
import numpy as np
import pandas as pd


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

import os
for dirname, _, filenames in os.walk('/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

4.2 Import data

In [2]:
```
path = '/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.

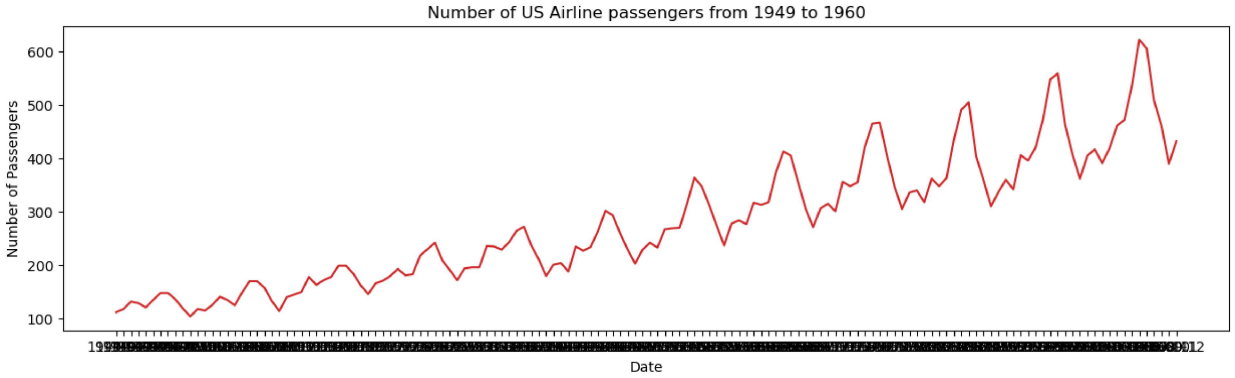
In [3]:
```
df.columns = ['Date','Number of Passengers']

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

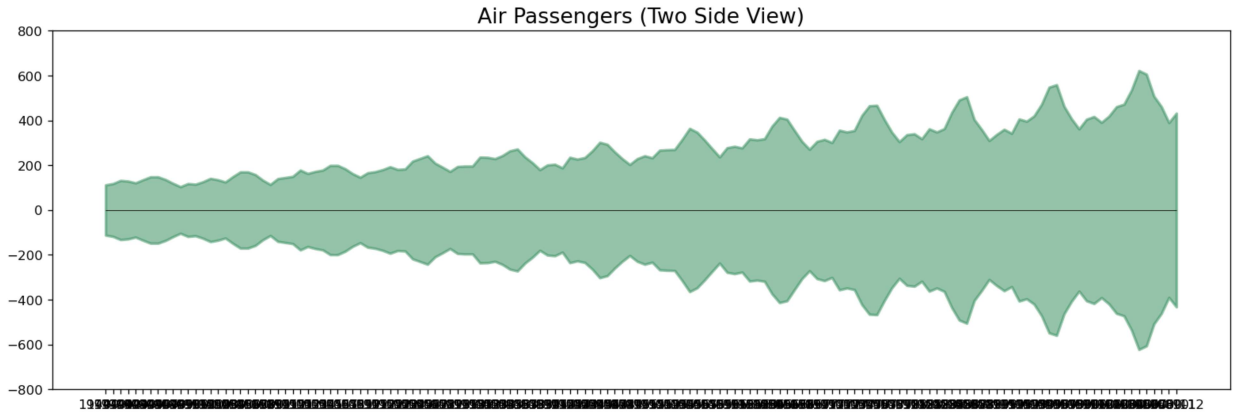
5. Visualize the Time Series

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 pa
```



- 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 it's a monthly time series and follows a certain repetitive pattern every year. Therefore, we can plot each year as a separate line in the same graph. This allows us to compare the year-wise patterns side-by-side.

6. Patterns in a Time Series

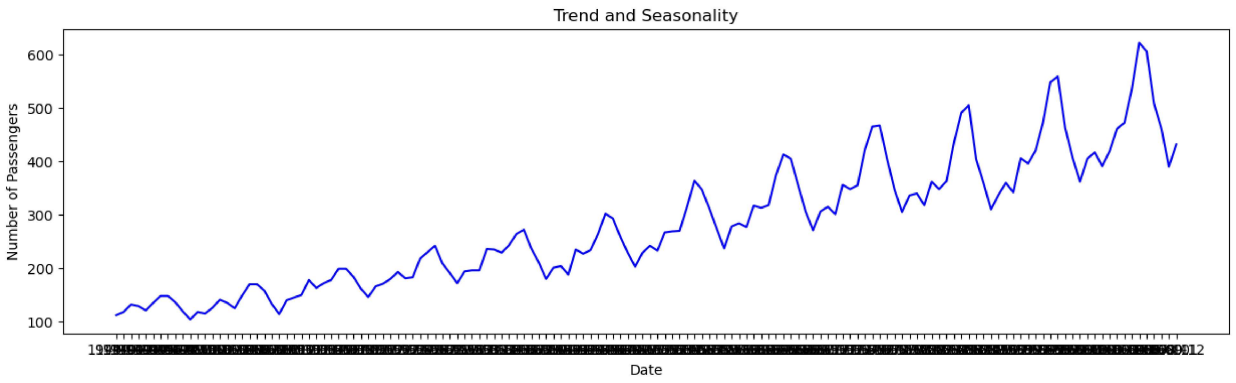
Any time series visualization may consist of the following components: Base Level, Trend, Seasonality, and Error.

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

Seasonality Seasonality is observed when a distinct repeated pattern occurs at regular intervals due to seasonal factors. This pattern could be influenced by the month of the year, the day of the month, weekdays, or even the time of day.

However, it's not mandatory for all time series to exhibit both trend and seasonality. A time series may lack a distinct trend but display 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 essential consideration is the concept of cyclic behavior. This occurs when the rise and fall pattern in the series does not adhere to fixed calendar-based intervals. It's important not to confuse 'cyclic' effects with 'seasonal' effects.

Cyclic behavior is identified when patterns do not follow fixed calendar frequencies. Unlike seasonality, cyclic effects are typically influenced by business and other socio-economic factors.

7. Additive and Multiplicative Time Series

We might encounter various combinations of trends and seasonality in time series data. Based on the nature of these trends and seasonality, a time series can be modeled as either an additive or multiplicative time series. Each observation in the series can be expressed as either a sum or a product of its individual components.

Additive time series: Value = Base Level + Trend + Seasonality + Error

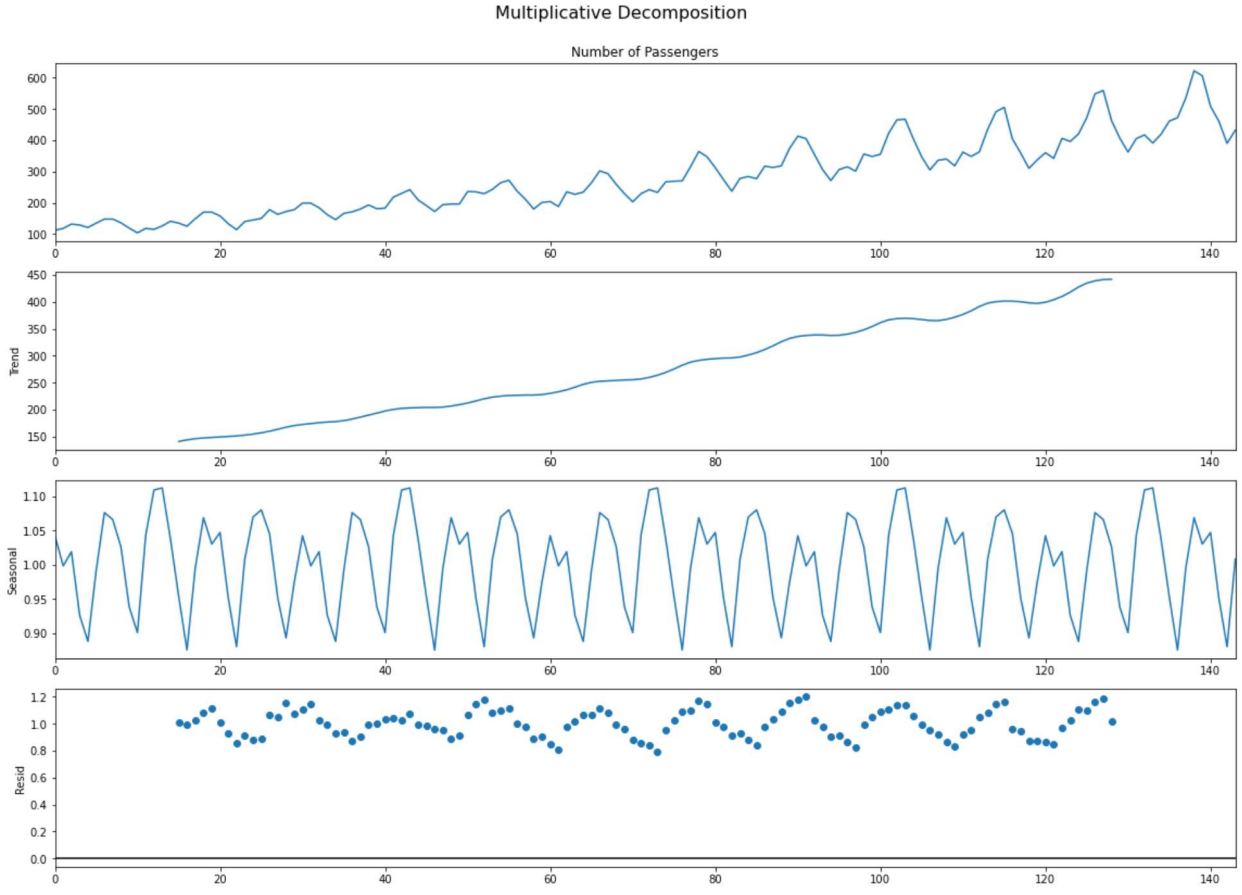
Multiplicative Time Series: Value = Base Level × Trend × Seasonality × Error

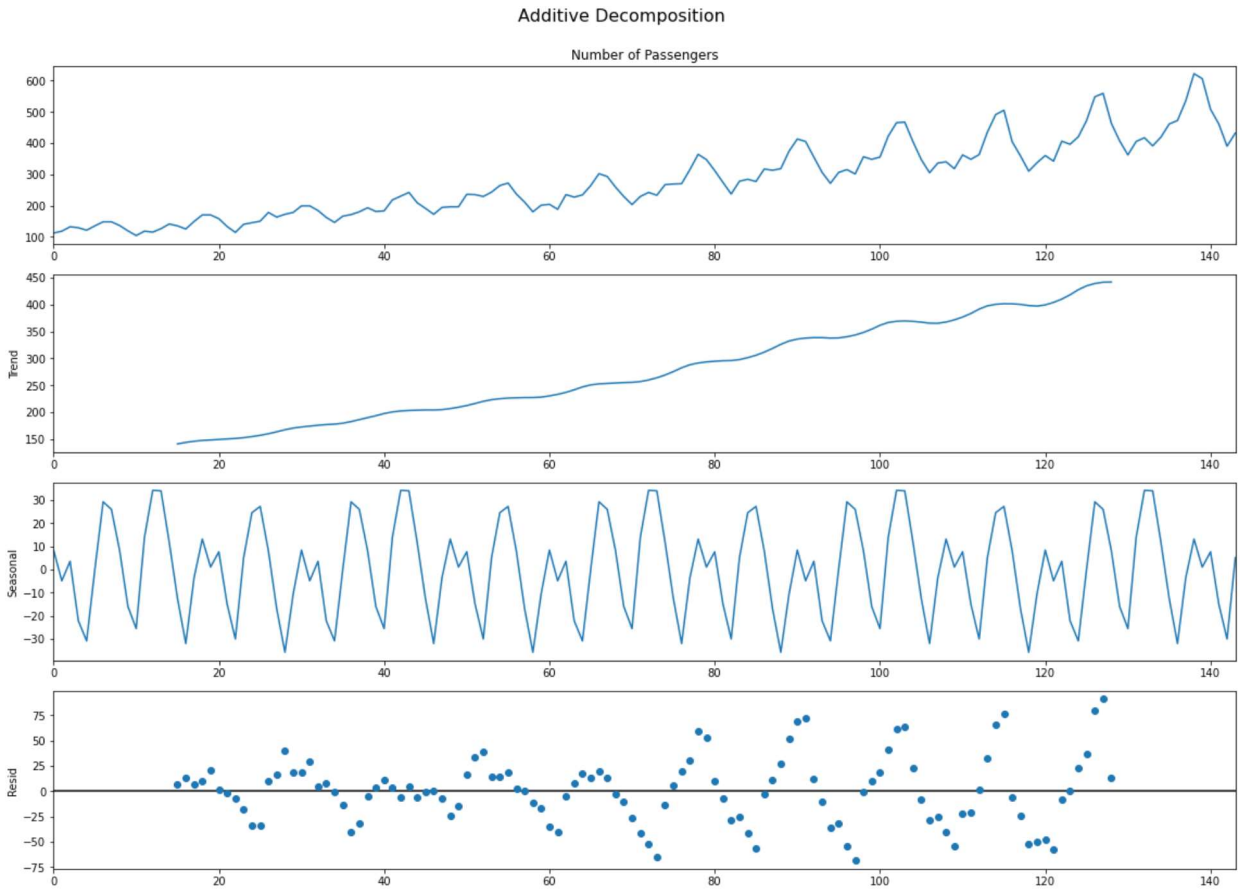
8. Decomposition of a Time Series

Time series decomposition involves breaking down the series into an additive or multiplicative combination of components such as the base level, trend, seasonal index, and the residual term.

The 'seasonal_decompose' function in statsmodels conveniently implements this process.

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='m

# Additive Decomposition
additive_decomposition = seasonal_decompose(df['Number of Passengers'], model='additiv

# Plot
plt.rcParams.update({'figure.figsize': (16,12)})
multiplicative_decomposition.plot().suptitle('Multiplicative Decomposition', fontsize=
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()
```





When closely examining the residuals of the additive decomposition, some discernible patterns remain.

Conversely, in the multiplicative decomposition, the residuals appear more random, which is preferable. Therefore, ideally, the multiplicative decomposition should be favored for this particular series.

9. Stationary and Non-Stationary Time Series

we'll discuss Stationary and Non-Stationary Time Series. Stationarity is a defining property of a time series. A stationary series is one where the values of the series are not influenced by time; in other words, they are time-independent.

As a result, statistical properties of the series, such as mean, variance, and autocorrelation, remain constant over time. Autocorrelation, in this context, refers to the series' correlation with its previous values.

A stationary time series is not impacted by seasonal effects either.

We will now plot examples of both stationary and non-stationary time series to provide clarity.

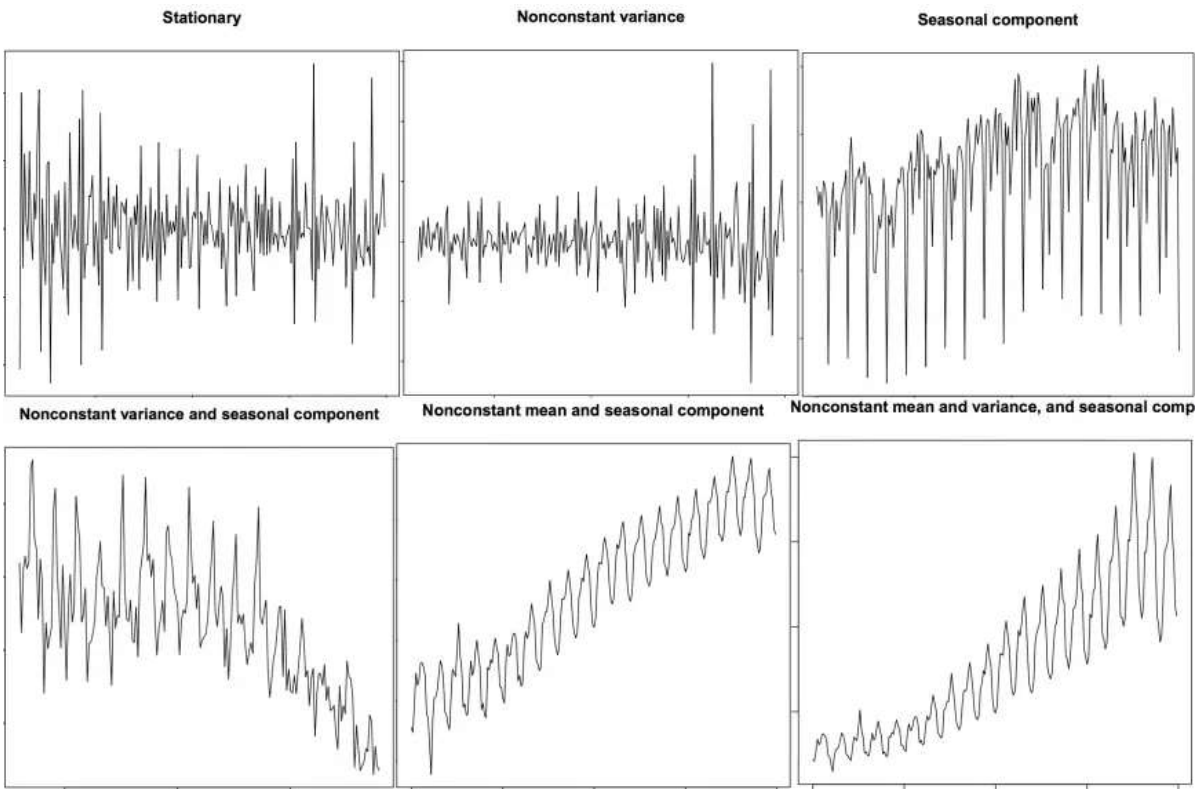


image source : https://www.machinelearningplus.com/wp-content/uploads/2019/02/stationary-and-non-stationary-time-series-865x569.png?ezimgfmt=ng:webp/ngcb1

We can convert any non-stationary time series into a stationary one by applying a suitable transformation. Most statistical forecasting methods are designed to operate effectively on stationary time series. Typically, the initial step in the forecasting process involves transforming a non-stationary series into a stationary one.

10. How to make a time series stationary?

We can employ various transformations to render a time series stationary, including:

Differencing the series (once or multiple times) Taking the logarithm of the series Taking the nth root of the series Using a combination of the above methods Among these, the most commonly used and convenient method to achieve stationarity is by differencing the series, at least once, and repeating this process until the series becomes approximately stationary.

10.1 Introduction to Differencing

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, let's consider the following series: [1, 5, 2, 12, 20]

First differencing yields: [5-1, 2-5, 12-2, 20-12] = [4, -3, 10, 8]

Second differencing should be applied to the first difference, resulting in: [-3-4, 10-(-3), 8-10] = [-7, 13, -2]

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

There are several reasons why converting a non-stationary series into a stationary one is important:

Forecasting a stationary series is notably simpler, and the forecasts tend to be more reliable.

Autoregressive forecasting models essentially function as linear regression models, utilizing the lags of the series as predictors.

For linear regression models to perform optimally, it's crucial that predictors (X variables) are not correlated with each other. Stationarizing the series resolves this issue by eliminating persistent autocorrelation, thereby ensuring that the predictors (lags of the series) in forecasting models are nearly independent.

11. How to test for stationarity?

The stationarity of a series can be visually examined by observing the plot of the series.

Another approach involves splitting the series into 2 or more contiguous parts and calculating summary statistics such as the mean, variance, and autocorrelation. If these statistics significantly differ between segments, the series is likely not stationary.

Quantitative methods for determining the stationarity of a series include statistical tests called Unit Root Tests, which assess if a time series is non-stationary and possesses a unit root.

Several implementations of Unit Root tests are available, including:

Augmented Dickey Fuller test (ADF Test) Kwiatkowski-Phillips-Schmidt-Shin – KPSS test (trend stationary) Philips Perron test (PP Test)

11.1 Augmented Dickey Fuller test (ADF Test)

The Augmented Dickey Fuller test (ADF Test) is widely used to detect stationarity. In this test, the null hypothesis assumes that the time series possesses a unit root and is non-stationary. Evidence is gathered to either support or reject this null hypothesis. If the p-value in the ADF test is less than the chosen significance level (often 0.05), we reject the null hypothesis.

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

The KPSS test, unlike the ADF test, is utilized to examine trend stationarity. In this test, the null hypothesis and the interpretation of the p-value are opposite to that of the ADF test.

11.3 Philips Perron test (PP Test)

The Philips Perron (PP) test is a unit root test used in time series analysis to examine the null hypothesis that a time series is integrated of order 1. This test is based on the principles of the ADF test discussed previously.

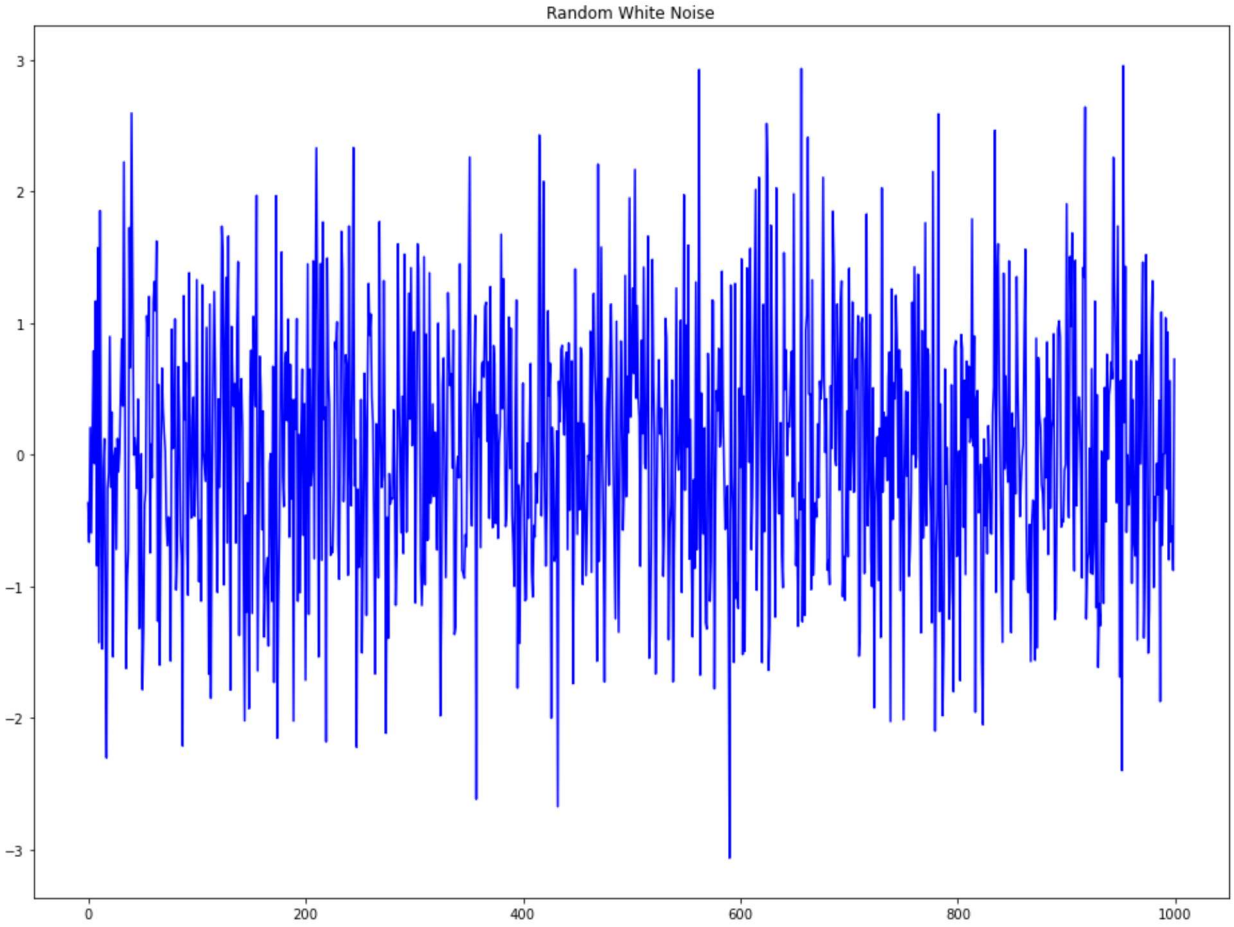
12. Difference between white noise and a stationary series

Similar to a stationary series, white noise is also not dependent on time, meaning its mean and variance remain constant over time. However, the key distinction is that white noise is entirely random, characterized by a mean of 0. In white noise, there is no discernible pattern.

Mathematically, a sequence of completely random numbers with a mean of zero constitutes white noise in time series analysis.

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 0x7f90bdf72790>
```



13. Detrend a Time Series

Detrending a time series involves removing the trend component from the series. There are several approaches to achieve this:

Subtracting the line of best fit from the time series, obtained from a linear regression model with time steps as predictors. For more complex trends, quadratic terms (x^2) may be used in the model.

Subtracting the trend component obtained from time series decomposition.

Subtracting the mean.

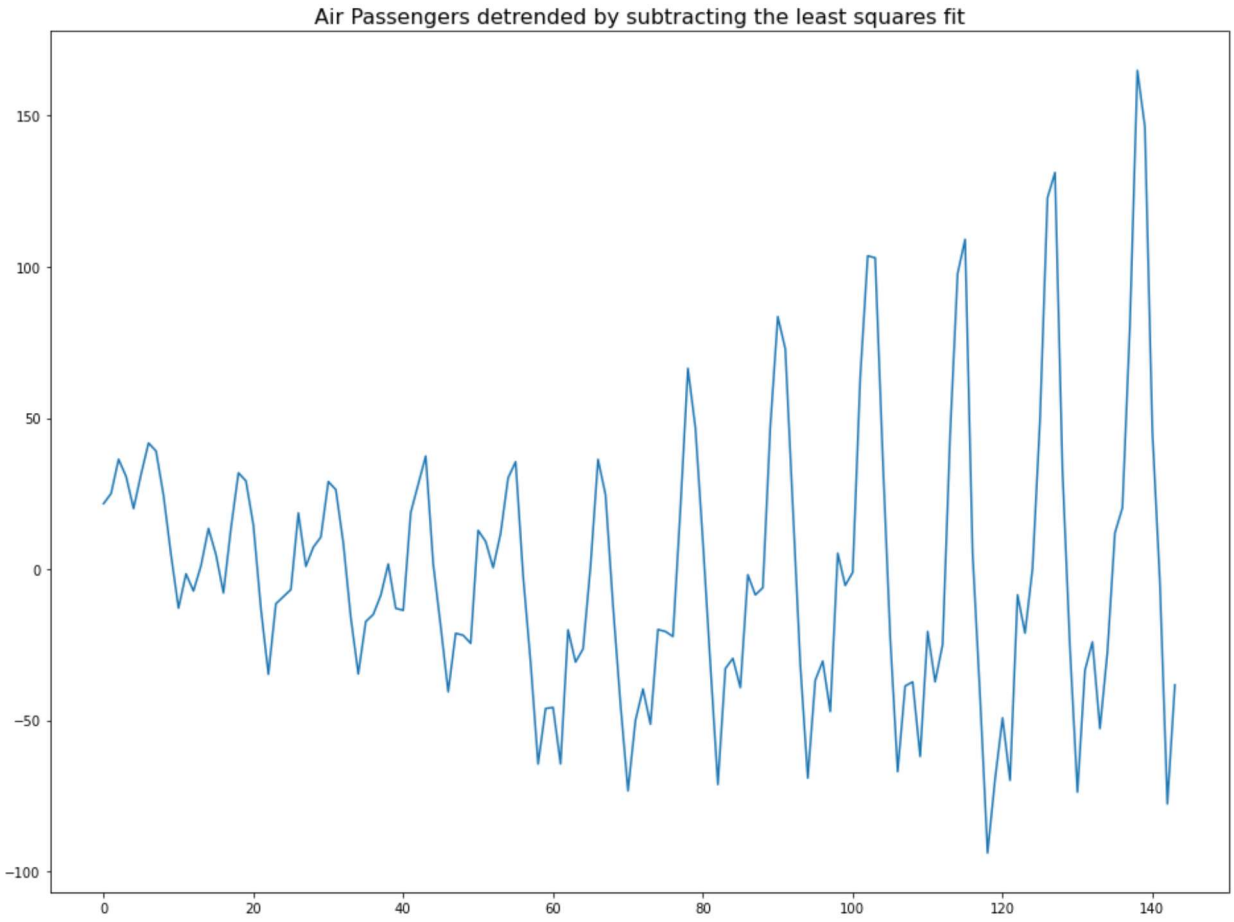
Applying a filter, such as the Baxter-King filter (statsmodels.tsa.filters.bkfilter) or the Hodrick-Prescott Filter (statsmodels.tsa.filters.hpfilter), to eliminate moving average trend lines or cyclical components.

We will now demonstrate the implementation of the first two methods for detrending 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)
```

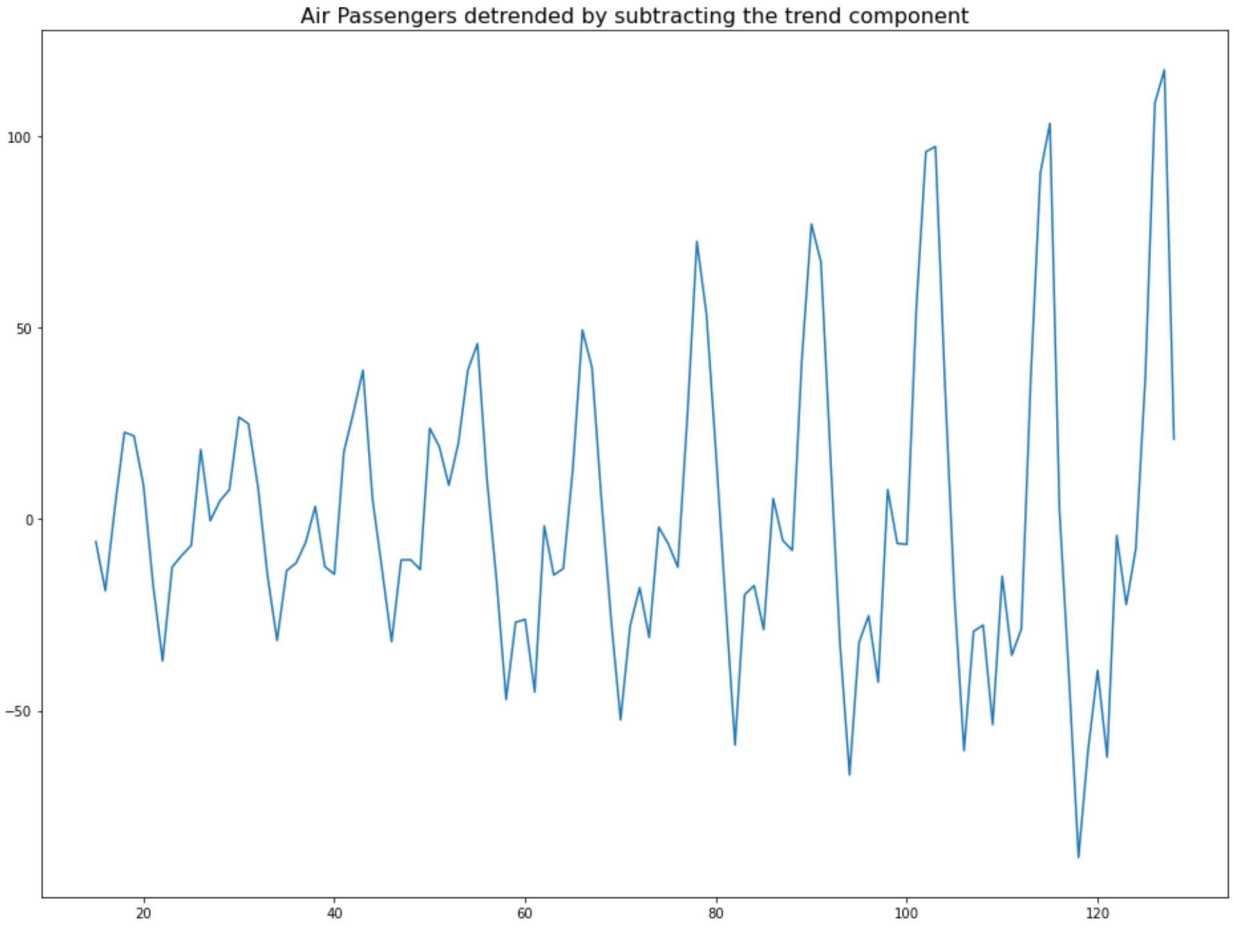
```
plt.plot(detrended)
plt.title('Air Passengers detrended by subtracting the least squares fit', fontsize=16
```

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', pe
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

There are several approaches to deseasonalize a time series:

Taking a moving average with a length equivalent to the seasonal window, which smoothens the series.

Seasonal differencing of the series by subtracting the value of the previous season from the current value.

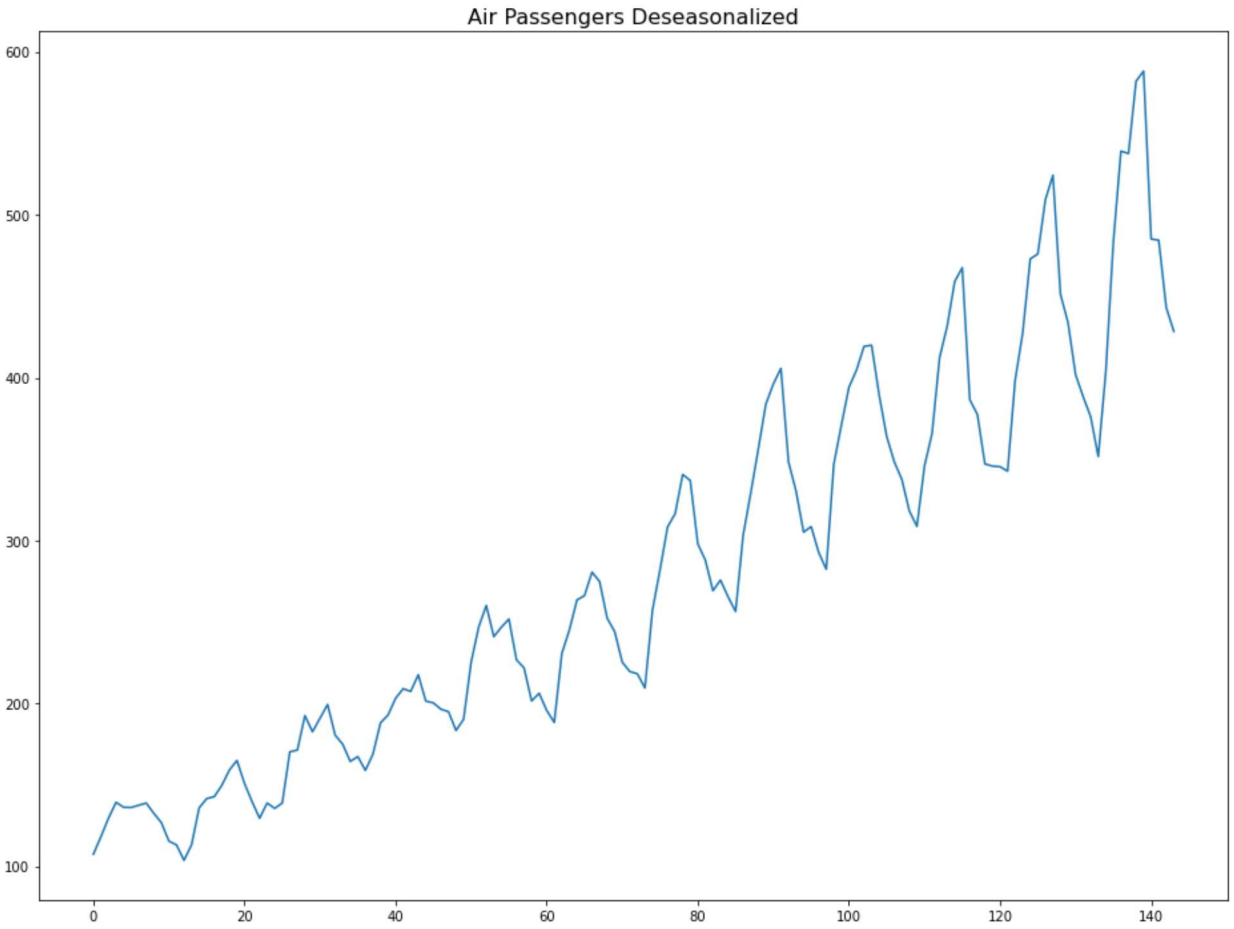
Dividing the series by the seasonal index obtained from Seasonal and Trend decomposition using Loess (STL decomposition).

In cases where dividing by the seasonal index doesn't produce satisfactory results, we will apply a logarithm to the series for deseasonalizing. Later, to return to the original scale, we will revert 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', pe

# 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?

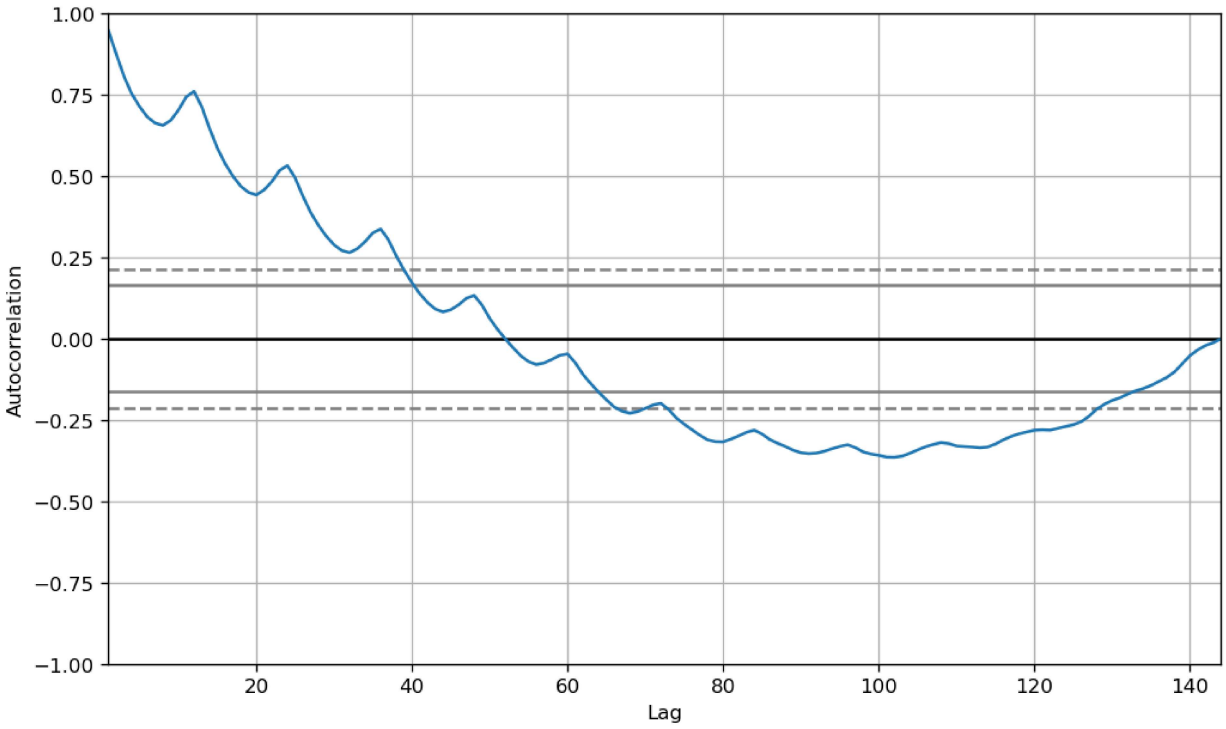
The types of seasonality in a time series are commonly identified by observing repeatable patterns at fixed time intervals, often associated with the clock or calendar, such as:

Hour of the day Day of the month Weekly patterns Monthly cycles Yearly fluctuations

For a more precise analysis of seasonality, the Autocorrelation Function (ACF) plot is recommended. A strong seasonal pattern is often indicated by definitive repeated spikes at multiples of the seasonal window in the ACF plot.

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 0x7f90be3bd350>



Alternatively, if we seek a statistical test, the CHTest can determine whether seasonal differencing is necessary to stationarize the series.

16. Autocorrelation and Partial Autocorrelation Functions

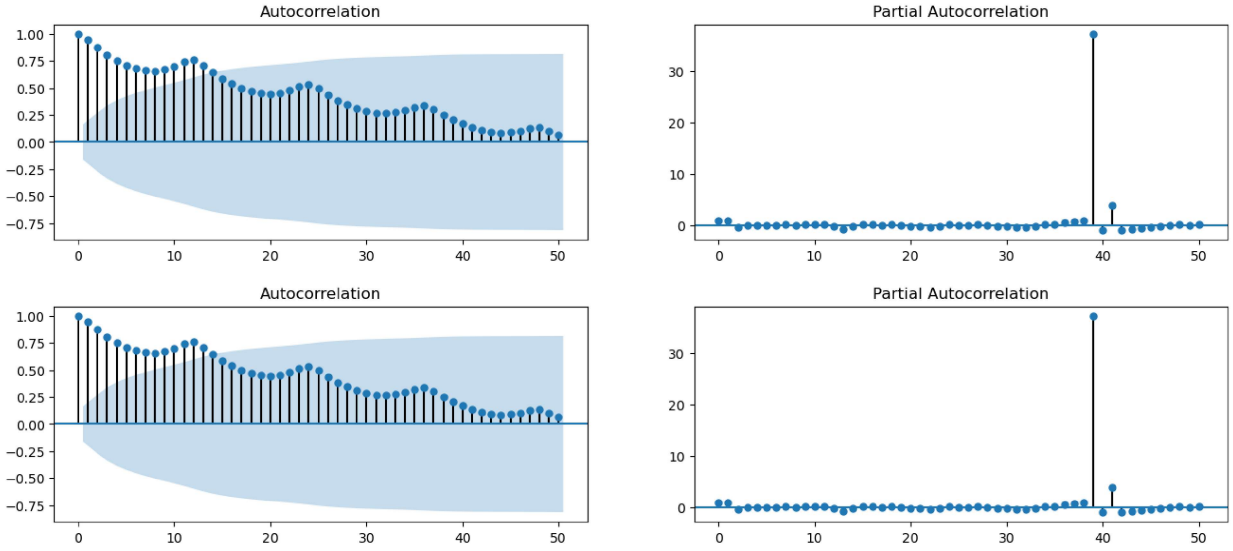
Autocorrelation is essentially the correlation of a series with its own lags. A significant autocorrelation implies that previous values of the series (lags) might aid in predicting the current value.

On the other hand, partial autocorrelation provides similar information but specifically indicates the pure correlation between a series and its lag, excluding the correlation contributions from 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: R
untimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```

Out[13]:



17. Computation of Partial Autocorrelation Function

The partial autocorrelation function of lag (k) for a series is the coefficient of that lag in the autoregression equation of Y. This autoregressive equation of Y is essentially a 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 represented by the coefficient α_3 in the following equation:

$$Y_t = \alpha_0 + \alpha_1 Y_{t-1} + \alpha_2 Y_{t-2} + \alpha_3 Y_{t-3}$$

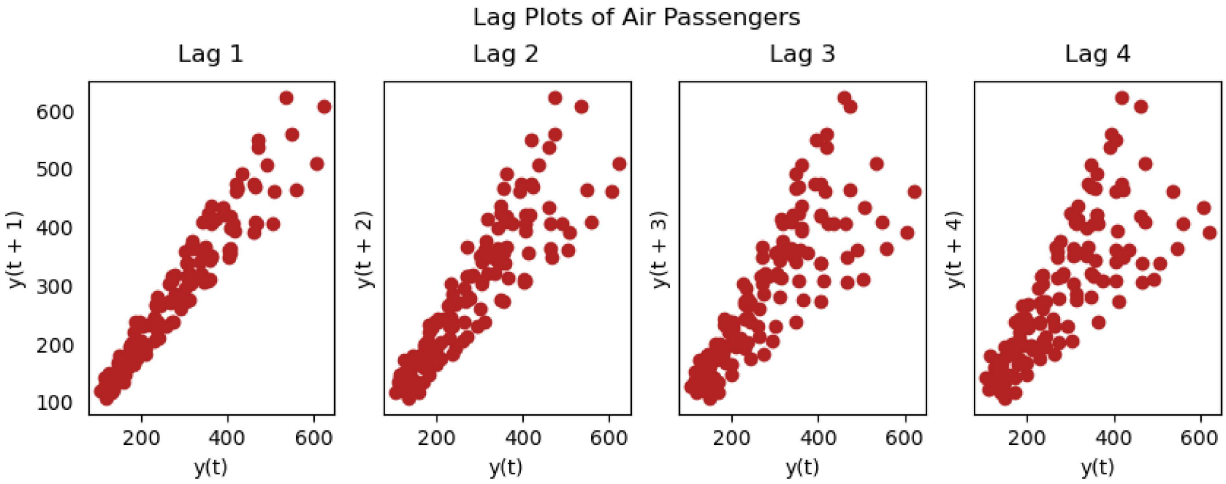
image source : https://www.machinelearningplus.com/wp-content/uploads/2019/02/12_5_Autoregression_Equation-min.png?ezimgfmt=ng:webp/ngcb1

18. Lag Plots

A Lag plot is a scatter plot of a time series against a lag of itself. It's commonly utilized to identify autocorrelation within the series. If any discernible pattern exists in the plot, it suggests that the series is autocorrelated. Conversely, if no such pattern is evident, the series is more likely to resemble 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

The Granger causality test determines whether one time series is useful in forecasting another. It operates on the principle that if X causes Y, then forecasting Y based on previous values of Y AND previous values of X should outperform forecasting Y based on previous values of Y alone.

It's important to note that the Granger causality test isn't suitable for determining if a lag of Y causes Y itself. Typically, it's used on exogenous variables (not lagged Y values) and is commonly implemented in the statsmodels package.

When conducting this test, it requires a 2D array with 2 columns as the main argument. The values of the time series are placed in the first column, while the predictor (X) is in the second column. The Null hypothesis posits that the series in the second column does not Granger cause the series in the first. If the P-values are less than a chosen significance level (often 0.05), then we reject the null hypothesis, indicating that the lag of X is indeed useful. The second argument, maxlag, specifies the number of lags of Y to include 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': (54.77967483557367, 3.661425871352945e-12, 200.0, 1.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x7f90b6ce3450>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x7f90b6cf7150>,
   array([[0., 1., 0.]])]),
 2: ({'ssr_ftest': (162.69891799873247, 1.9133235086856426e-42, 197.0, 2),
   'ssr_chi2test': (333.6566643222737, 3.526760088127662e-73, 2),
   'lrtest': (196.99559277182198, 1.6709003499115789e-43, 2),
   'params_ftest': (162.69891799873236, 1.9133235086857257e-42, 197.0, 2.0)},
  [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x7f90b6cf7e90>,
   <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x7f90b6cf7850>,
   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

Smoothing a time series can be advantageous in several scenarios:

Reducing the influence of noise in a signal to obtain a more accurate, noise-filtered series. Using the smoothed version of the series as a feature to explain the original series itself. Enhancing the visualization of the underlying trend. Various methods can be employed to smoothen a time series, including:

Applying a moving average. Implementing LOESS smoothing (Localized Regression). Using LOWESS smoothing (Locally Weighted Regression).

Moving Average

A moving average represents the average of a rolling window with a defined width. It's crucial to choose the window width wisely because a large window size can over-smooth the series. For instance, a window size equal to the seasonal duration (e.g., 12 for a monthly series) would effectively eliminate the seasonal effect.

Localized Regression

LOESS, an acronym for 'Localized Regression,' fits multiple regressions within the local neighborhood of each data point. It's available in the statsmodels package, where you can adjust the degree of smoothing using the 'frac' argument. This argument specifies the percentage of nearby data points considered for fitting a regression model.