# Study of Python Libraries for ML application such as Pandas and Matplotlib

* 1. **Pandas**

Pandas is an open-source Python library that provides high-performance, easy-to-use data structure, and data analysis tools for the Python [programming language](https://www.simplilearn.com/pgp-full-stack-web-development-certification-training-course?source=GhPreviewCoursePages).

Python with pandas is used in a wide range of fields, including academics, retail, finance, economics, statistics, analytics, and many others.

Python pandas is well suited for different kinds of data, such as:

* Ordered and unordered time series data
* Unlabeled data
* Any other form of observational or statistical data sets

**Pandas** is a powerful and versatile library that simplifies tasks of data manipulation in [Python](https://www.geeksforgeeks.org/python-programming-language/) . Pandas is built on top of the NumPy library and is particularly well-suited for working with tabular data, such as spreadsheets or SQL tables. Its versatility and ease of use+ make it an essential tool for data analysts, scientists, and engineers working with structured data in Python.

#### Installing Pandas

The first step of working in pandas is to ensure whether it is installed in the system or not. If not then we need to install it in our system using the **pip command**. Type the cmd command in the search box and locate the folder using the cd command

where **python-pip file** has been installed. After locating it, type the command:

# pip install pandas

For more reference take a look at this article on [installing pandas](https://www.geeksforgeeks.org/how-to-install-python-pandas-on-windows-and-linux/) follows.

#### Importing Pandas

After the pandas have been installed into the system, you need to import the library. This module is generally imported as follows:

# import pandas as pd

#### What can you do using Pandas?

Pandas are generally used for data science but have you wondered why? This is because pandas are used in conjunction with other libraries that are used for data science. It is built on the top of the **NumPy** library which means that a lot of structures of NumPy are used or replicated in Pandas. The data produced by Pandas are often used as input for plotting functions of **Matplotlib**, statistical analysis in **SciPy**, and machine learning algorithms in **Scikit-learn**. Here is a list of things that we can do using Pandas.

* + Data set cleaning, merging, and joining.
  + Easy handling of missing data (represented as NaN) in floating point as well as non- floating point data.
  + Columns can be inserted and deleted from DataFrame and higher dimensional objects.
  + Powerful group by functionality for performing split-apply-combine operations on data sets.
  + Data Visulaization

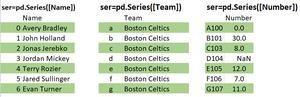
#### Pandas Data Structures

Pandas generally provide two data structures for manipulating data, They are:

* + **Series**
  + **DataFrame**

**Pandas Series**

A [Pandas Series](https://www.geeksforgeeks.org/python-pandas-series/) is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called indexes. Pandas Series is nothing but a column in an Excel sheet. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.



#### Creating a Series

In the real world, a Pandas Series will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, or an Excel file. Pandas Series can be created from lists, dictionaries, and from scalar values, etc.

#### Example:

import pandas as pd

import numpy as np

# Creating empty series

ser = pd.Series()

print("Pandas Series: ", ser)

# simple array

data = np.array(['g', 'e', 'e', 'k', 's'])

ser = pd.Series(data)

print("Pandas Series:\n", ser)

#### Output

Pandas Series: Series([], dtype: float64) Pandas Series:

0 g

1 e

2 e

3 k

4 s

dtype: object

Note: For more information, refer to Creating a Pandas Series

#### DataFrame

[Pandas DataFrame](https://www.geeksforgeeks.org/python-pandas-dataframe/) is a two-dimensional data structure with labeled axes (rows and columns).

**Note:** For more information, refer to [Python | Pandas DataFrame](https://www.geeksforgeeks.org/python-pandas-dataframe/) Creating Data Frame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, or an Excel file. Pandas DataFrame can be created from lists, dictionaries, and from a list of dictionaries, etc.

Methods

* **pd.DataFrame()**: Creates a DataFrame from a dictionary, list, or other data structure.
* **pd.Series()**: Creates a one-dimensional labeled array (Series).
* **pd.read\_csv()**: Reads data from a CSV file into a DataFrame.
* **pd.read\_excel()**: Reads data from an Excel file into a DataFrame.
* **pd.read\_sql()**: Reads data from a SQL query or database into a DataFrame.
* **head()**: Displays the first 5 rows of the DataFrame (customizable using n).
* **tail()**: Displays the last 5 rows of the DataFrame.
* **info()**: Provides a summary of the DataFrame (data types, non-null values, etc.).
* **describe()**: Summarizes statistical properties of numeric columns.
* **shape**: Returns the dimensions of the DataFrame as (rows, columns).
* **dtypes**: Returns the data types of each column.
* **dropna()**: Removes rows/columns with missing values.
* **fillna()**: Fills missing values with a specified value or method (e.g., forward fill).
* **duplicated()**: Identifies duplicate rows

import pandas as pd

# Calling DataFrame constructor df = pd.DataFrame()

print(df)

# list of strings

lst = ['CMR', 'CMRTC', 'SET', 'CMREC', 'ENGG', 'CMR1', 'CMR2']

# Calling DataFrame constructor on list df = pd.DataFrame(lst)

print(df)

Output:

Empty DataFrame

Columns: [] Index: []

#### Example:

1. CMR
2. CMRTC
3. SET
4. CMREC
5. ENGG
6. CMR1
7. CMR2

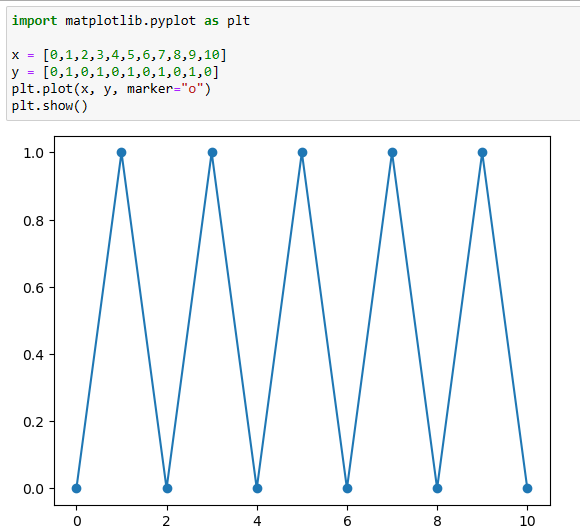
# Matplotlib

Neuroscientist John D. Hunter began developing matplotlib in 2003, mainly inspired by the emulation of Mathworks MATLAB software teams. Matplotlib is today a whole product of the community: it is developed and supported by many people. John talked about the evolution of matplotlib at the SciPy conference in 2012. Learning matplotlib at times can be a difficult process. The problem is not the lack of documentation (which is very extensive, by the way). Difficulties may arise with the following:

* + - The size of the library is huge in itself, about 70,000 lines of code
    - Matplotlib contains several different interfaces (ways to build a figure) and can interact with a large number of backends. (The backends are responsible for how in fact the diagrams will be displayed, not only for the internal structure)
    - Despite the vastness, some of the matplotlib's own documentation is seriously outdated. The library is still evolving, and many old examples on the web may include 70% less code than in their current version.

Understanding that matplotlib roots grow from MATLAB helps explain the existence of pylab. pylab is a module inside the matplotlib library that has been built in to emulate the overall MATLAB style. It exists only for introducing a number of class functions from NumPy and matplotlib into the namespace, which simplifies the transition of MATLAB users who did not encounter the need for import statements. Former MATLAB users love its functionality, because with from pylab import \* they can simply call plot() or array() directly, just like they did in MATLAB.

#### Python code example for plotting

The Python code itself is quite simple and straightforward. Here's an example of a simple plot:

Types of graphs and charts

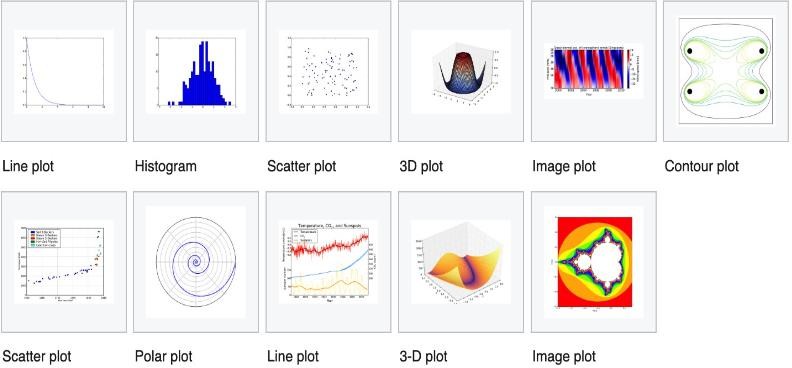
The package supports many types of graphs and charts:

1. Charts (line plot)
2. Scatter plot
3. Bar charts and histograms
4. Pie Chart
5. Chart trunk (stem plot)
6. Contour plots
7. Gradient Fields (quiver)
8. Spectrograms

The user can specify the coordinate axis, grid, add labels and explanations, use a logarithmic scale or polar coordinates

Simple 3D graphics can be built using the mplot3d toolkit. There are other toolkits: for mapping, for working with Excel, utilities for GTK and others. With Matplotlib, you can make animated images.

Matplotlib can be technically and syntactically complex. To create a ready-made diagram, it can take half an hour to google search alone and combine all this hash to fine-tune the graph. However, understanding how matplotlib interfaces interact with each other is an investment that can pay off.



4. Write a Python program to implement Simple Linear Regression

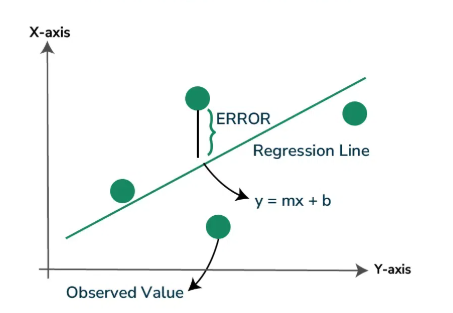
Aim: Program to implement Simple Linear Regression

Description:

**Simple Linear Regression:** Simple Linear Regression: Linear regression is a linear model, that assumes a linear relationship between the input variable(x) and the single variable (y).

**Least Square Method**

Least Square Method formula is used to find the best-fitting line through a set of data points. For a simple linear regression, which is a line of the form *y*=m*x*+*c*, where *y* is the dependent variable, *x* is the independent variable, *a* is the slope of the line, and *b* is the y-intercept, the formulas to calculate the slope (*m*) and intercept (*c*) of the line are derived from the following equations:

1. **Slope (*m*) Formula:***m*= *n*(∑*xy*)−(∑*x*)(∑*y*) / *n*(∑*x2*)−(∑*x*)2​
2. **Intercept (*c*) Formula:** *c*= (∑*y*)−*a*(∑*x*) / n​ 

**Program:**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn import linear\_model

from sklearn.model\_selection import train\_test\_split

#Read Datasert

df=pd.read\_csv(r"C:\Users\Tabeen Fatima\Downloads\Salary\_Data[1].csv")

#check for missing value

print(df.isnull().sum())

#drop null values

df2=df.dropna()

#Extract x feature and y target for dataset

x=df2[['Years of Experience']].values

print(x)

y=df2['Salary'].values

print(y)

# splitting the dataset

X\_train,X\_test,Y\_train,Y\_test= train\_test\_split(x,y,test\_size=1/3,random\_state=0)

# fitting the regression model

model=linear\_model.LinearRegression()

model.fit(X\_train,Y\_train)

predicted\_value=model.predict([[22.5]])

print("predicted value for 22.5 years of experience:",predicted\_value)

# visualizing the results

plt.title("Years of Experience vs Salary (Training Data)")

plt.xlabel("Years of Experience")

plt.ylabel("Salary")

plt.scatter(X\_train,Y\_train,color='g',marker='\*')

plt.plot(X\_train,model.predict(X\_train),color='r',linewidth=2,label='Regeression Line')

plt.legend()

plt.show()

plt.title("Years of Experience vs Salary (Test Data)")

plt.xlabel("Years of Experience")

plt.ylabel("Salary")

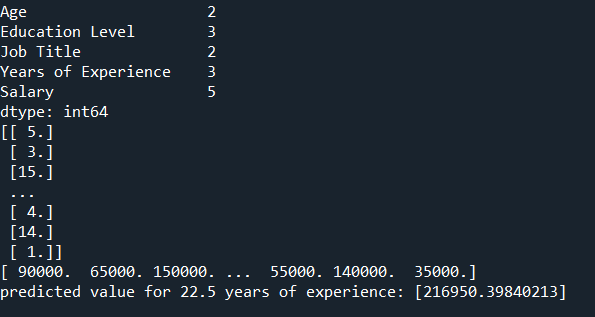
plt.scatter(X\_test,Y\_test,color='g',marker='\*')

plt.plot(X\_test,model.predict(X\_test),color='r',linewidth=2,label='Regeression Line')

plt.legend()

plt.show()

**Output:**

****

**Plots:**

