PRACTICAL : 1

# Use Appropriate Python & R Libraries to Manage and Visualize ML Dataset.

SETUP/INSTALL PYTHON IDE [Pycharm IDE] :

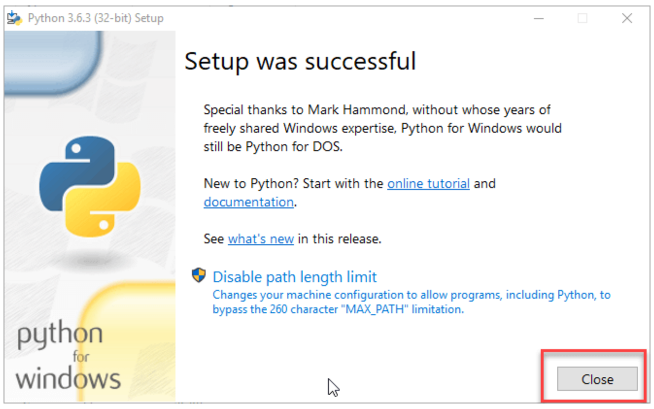
Below is a step by step process on how to download and install Python on Windows:

Step 1) To download and install Python, visit the official website of Python https://www.python.org/downloads/ and choose your version. We have chosen Python version 3.6.3

Step 2) Once the download is completed, run the .exe file to install Python. Now click on Install Now.

Step 3) You can see Python installing at this point.

Step 4) When it finishes, you can see a screen that says the Setup was successful. Now click on "Close".



Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

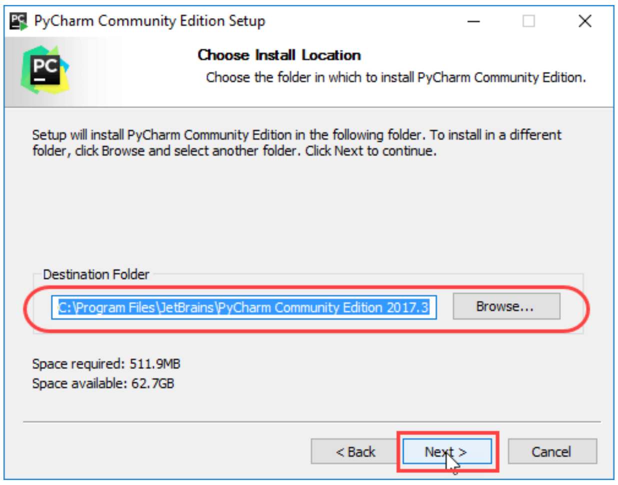
How to Install Pycharm :

Here is a step by step process on how to download and install Pycharm IDE on Windows:

Step 1) To download PyCharm visit the website https://www.jetbrains.com/pycharm/download/ and Click the "DOWNLOAD" link under the Community Section.

Step 2) Once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click "Next".

Step 3) On the next screen, Change the installation path if required. Click "Next".


PyCharm Community Edition Setup

Choose Install Location

Choose the folder in which to install PyCharm Community Edition.

Setup will install PyCharm Community Edition in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Program Files\JetBrains\PyCharm Community Edition 2017.3

Space required: 511.9MB
Space available: 62.7GB

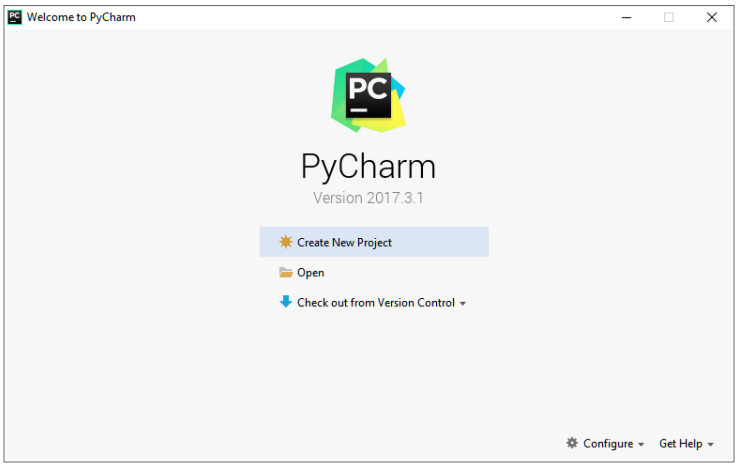
Step 4) On the next screen, you can create a desktop shortcut if you want and click on “Next”.

Step 5) Choose the start menu folder. Keep selected JetBrains and click on “Install”.

Step 6) Wait for the installation to finish.

Step 7) Once installation finished, you should receive a message screen that PyCharm is installed. If you want to go ahead and run it, click the “Run PyCharm Community Edition” box first and click “Finish”.

Step 8) After you click on “Finish,” the Following screen will appear.



IMPORT LIBRARIES IN PyCharm :

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

```
import numpy as nm
```

Here we have used nm, which is a short name for Numpy, and it will be used in the whole program.

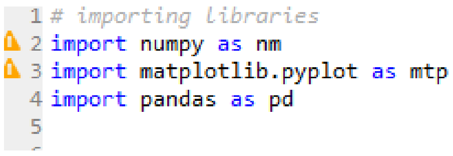
Matplotlib: The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:

```
import matplotlib.pyplot as mpt
```

Here we have used mpt as a short name for this library.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. It will be imported as below:

Here, we have used pd as a short name for this library. Consider the below image:



```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

Python read csv file :

CSV File : A csv stands for "comma separated values", which is defined as a simple file format that uses specific structuring to arrange tabular data. It stores tabular data such as spreadsheet or database in plain text and has a common format for data interchange. A csv file opens into the excel sheet, and the rows and columns data define the standard format.

Reding CSV File with Pandas

The Pandas is defined as an open-source library which is built on the top of the NumPy library. It provides fast analysis, data cleaning, and preparation of the data for the user.

Reading the csv file into a pandas **DataFrame** is quick and straight forward. We don't need to write enough lines of code to open, analyze, and read the csv file in pandas and it stores the data in **DataFrame**.

Here, we are taking a slightly more complicated file to read, called hrdata.csv, which contains data of company employees.

```
Name,Hire Date,Salary,Leaves Remaining
John Idle,08/15/14,50000.00,10
Smith Gilliam,04/07/15,65000.00,8
Parker Chapman,02/21/14,45000.00,10
Jones Palin,10/14/13,70000.00,3
Terry Gilliam,07/22/14,48000.00,7
Michael Palin,06/28/13,66000.00,8
```

Example :

```
import pandas
df = pandas.read_csv('hrdata.csv')
print(df)
```

In the above code, the three lines are enough to read the file, and only one of them is doing the actual work, i.e., pandas.read_csv()

```
         Name                Hire Date     Salary      Leaves Remaining
0     John Idle              03/15/14      50000.0       10
1     Smith Gilliam          06/01/15      65000.0       8
2     Parker Chapman         05/12/14      45000.0       10
3     Jones Palin            11/01/13      70000.0       3
4     Terry Gilliam          08/12/14      48000.0       7
5     Michael Palin          05/23/13      66000.0       8
```

Managing the Dataset :

- **Getting the dataset**
- **Importing datasets**
- **Finding Missing Data**
- **Encoding Categorical Data**
- **Splitting dataset into training and test set**

1) Get the Dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the **dataset**.

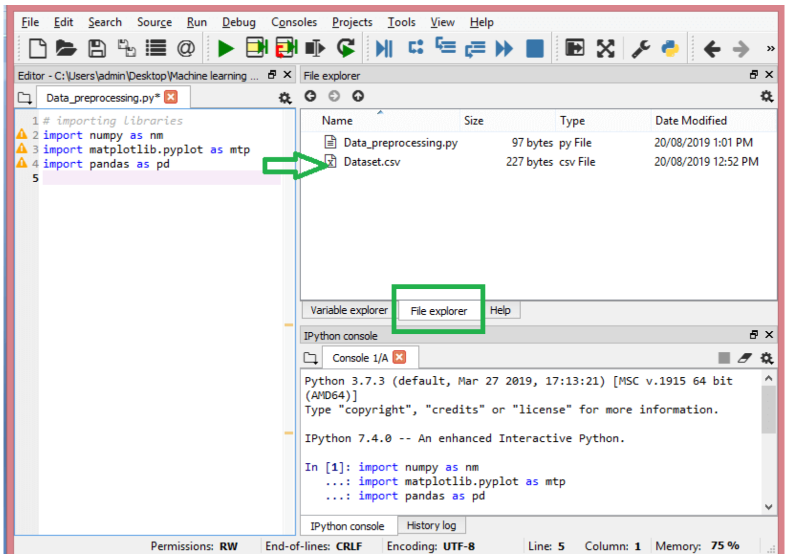
Dataset may be of different formats for different purposes, such as, if we want to create a machine learning model for business purpose, then dataset will be different with the dataset required for a liver patient. So each dataset is different from another dataset. To use the dataset in our code, we usually put it into a CSV **file**. However, sometimes, we may also need to use an HTML or xlsx file.

3) Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory. To set a working directory in Spyder IDE, we need to follow the below steps:

1. Save your Python file in the directory which contains dataset.
2. Go to File explorer option in Spyder IDE, and select the required directory.
3. Click on F5 button or run option to execute the file.

Here, in the below image, we can see the Python file along with required dataset. Now, the current folder is set as a working directory



```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

```
Name                     Size       Type      Date Modified
Data_preprocessing.py    97 bytes   py File   20/08/2019 1:01 PM
Dataset.csv              227 bytes  csv File  20/08/2019 12:52 PM
```

```
Python 3.7.3 (default, Mar 27 2019, 17:13:21) [MSC v.1915 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.4.0 -- An enhanced Interactive Python.

In [1]: import numpy as nm
   ...: import matplotlib.pyplot as mtp
   ...: import pandas as pd
```

Permissions: RW   End-of-lines: CRLF   Encoding: UTF-8   Line: 5   Column: 1   Memory: 75 %

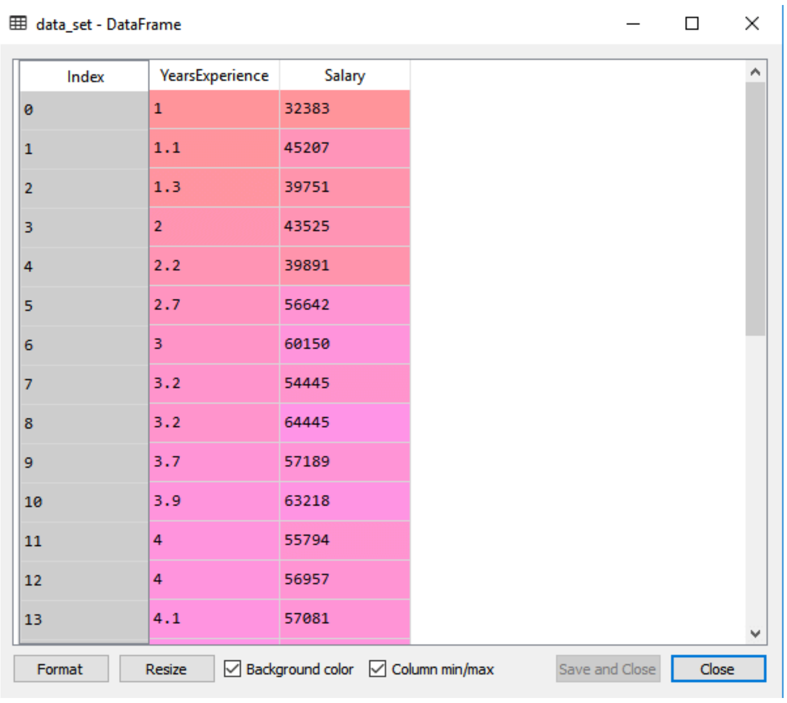
read_csv() function:

Now to import the dataset, we will use read_csv() function of pandas library, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL.

We can use read_csv function as below:

data_set= pd.read_csv(' Salary_Data.csv')

Here, **data_set** is a name of the variable to store our dataset, and inside the function, we have passed the name of our dataset. Once we execute the above line of code, it will successfully import the dataset in our code. We can also check the imported dataset by clicking on the section **variable explorer**, and then double click on **data_set**. Consider the below image:



Index	YearsExperience	Salary
0	1	32383
1	1.1	45207
2	1.3	39751
3	2	43525
4	2.2	39891
5	2.7	56642
6	3	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4	55794
12	4	56957
13	4.1	57081

As in the above image, indexing is started from 0, which is the default indexing in Python. We can also change the format of our dataset by clicking on the format option.

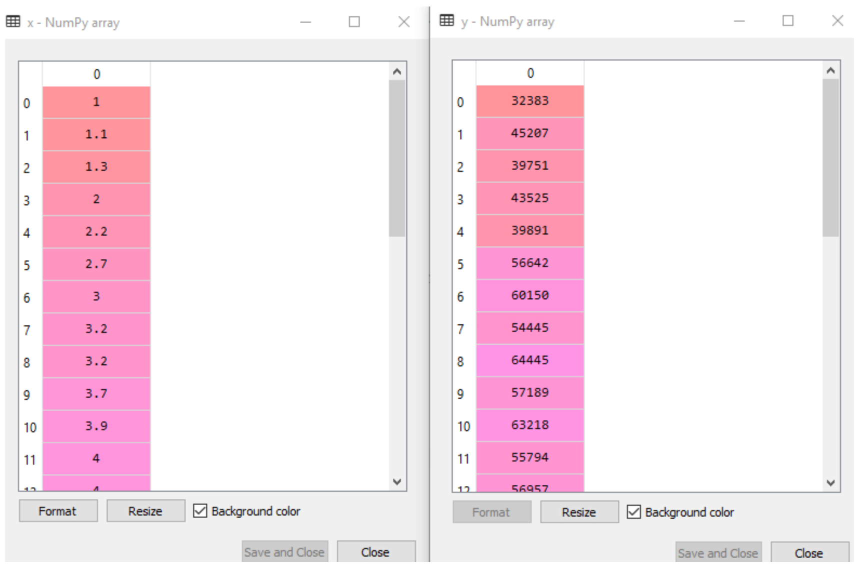
Extracting dependent and independent variables:

- After that, we need to extract the dependent and independent variables from the given dataset. The independent variable is years of experience, and the dependent variable is salary. Below is code for it:

```
x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values
```

In the above lines of code, for x variable, we have taken -1 value since we want to remove the last column from the dataset. For y variable, we have taken 1 value as a parameter, since we want to extract the second column and indexing starts from the zero.

By executing the above line of code, we will get the output for X and Y variable as:



```
     0
0    1
1    1.1
2    1.3
3    2
4    2.2
5    2.7
6    3
7    3.2
8    3.2
9    3.7
10   3.9
11   4
12   4
```

```
     0
0    32383
1    45207
2    39751
3    43525
4    39891
5    56642
6    60150
7    54445
8    64445
9    57189
10   63218
11   55794
12   56957
```

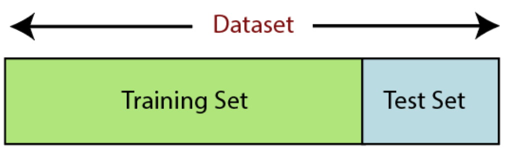
In the above output image, we can see the X (independent) variable and Y (dependent) variable has been extracted from the given dataset.

Splitting the Dataset into the Training set and Test set

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.

Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:



Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

- Next, we will split both variables into the test set and training set. We have 30 observations, so we will take 20 observations for the training set and 10 observations for the test set. We are splitting our dataset so that we can train our model using a training dataset and then test the model using a test dataset. The code for this is given below:

```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 1/3, random_state=0)
```

Explanation:

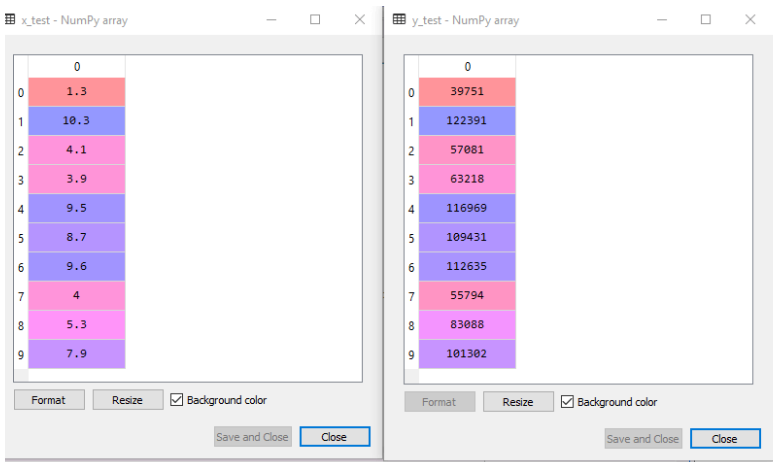
- In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
- In the second line, we have used four variables for our output that are
  - **x_train:** features for the training data
  - **x_test:** features for testing data
  - **y_train:** Dependent variables for training data
  - **y_test:** Independent variable for testing data

- In **train_test_split() function**, we have passed four parameters in which first two are for arrays of data, and **test_size** is for specifying the size of the test set. The test_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.
- The last parameter **random_state** is used to set a seed for a random generator so that you always get the same result, and the most used value for this is 42.

Output:

By executing the above code, we will get x-test, x-train and y-test, y-train dataset. Consider the below images

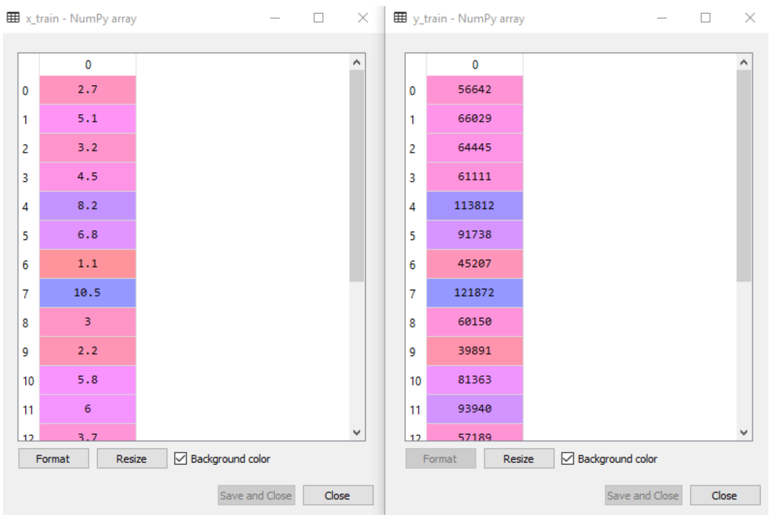
Test-dataset:



	0
0	1.3
1	10.3
2	4.1
3	3.9
4	9.5
5	8.7
6	9.6
7	4
8	5.3
9	7.9

	0
0	39751
1	122391
2	57081
3	63218
4	116969
5	109431
6	112635
7	55794
8	83088
9	101302

Training Dataset:



	0
0	2.7
1	5.1
2	3.2
3	4.5
4	8.2
5	6.8
6	1.1
7	10.5
8	3
9	2.2
10	5.8
11	6
12	3.7

	0
0	56642
1	66029
2	64445
3	61111
4	113812
5	91738
6	45207
7	121872
8	60150
9	39891
10	81363
11	93940
12	57189

- For simple linear Regression, we will not use Feature Scaling. Because Python libraries take care of it for some cases, so we don't need to perform it here. Now, our dataset is well prepared to work on it and we are going to start building a Simple Linear Regression model for the given problem.

Step-2: Fitting the Simple Linear Regression to the Training Set:

Now the second step is to fit our model to the training dataset. To do so, we will import the **LinearRegression** class of the **linear_model** library from the **scikit learn**. After importing the class, we are going to create an object of the class named as a **regressor**. The code for this is given below:

```
#Fitting the Simple Linear Regression model to the training dataset
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

In the above code, we have used a **fit()** method to fit our Simple Linear Regression object to the training set. In the fit() function, we have passed the x_train and y_train, which is our training dataset for the dependent and an independent variable. We have fitted our regressor object to the training set so that the model can easily learn the correlations between the predictor and target variables. After executing the above lines of code, we will get the below output.

Output:

```
Out[7]:    LinearRegression(copy_X=True,    fit_intercept=True,    n_jobs=None,
normalize=False)
```

Step: 3. visualizing the Training set results:

Now in this step, we will visualize the training set result. To do so, we will use the scatter() function of the pyplot library, which we have already imported in the pre-processing step. The **scatter () function** will create a scatter plot of observations. In the x-axis, we will plot the Years of Experience of employees and on the y-axis, salary of employees. In the function, we will pass the real values of training set, which means a year of experience x_train, training set of Salaries y_train, and color of the observations. Here we are taking a green color for the observation, but it can be any color as per the choice.

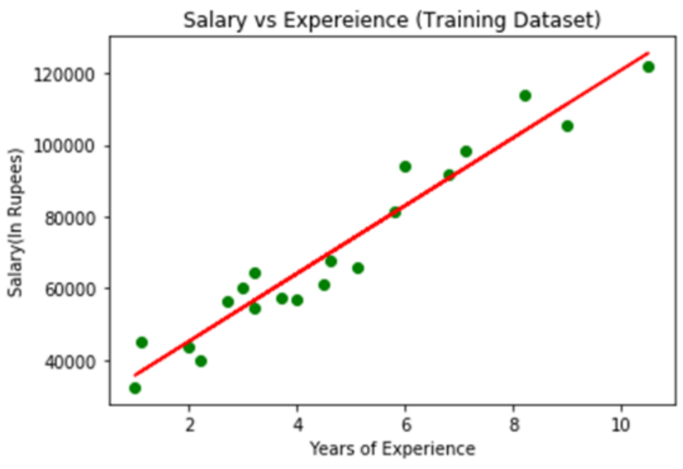
Now, we need to plot the regression line, so for this, we will use the **plot() function** of the pyplot library. In this function, we will pass the years of experience for training set, predicted salary for training set x_pred, and color of the line.

Next, we will give the title for the plot. So here, we will use the **title()** function of the **pyplot** library and pass the name ("Salary vs Experience (Training Dataset)". After that, we will assign labels for x-axis and y-axis using **xlabel() and ylabel() function**.

Finally, we will represent all above things in a graph using show(). The code is given below:

```
mtp.scatter(x_train, y_train, color="green")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Salary vs Experience (Training Dataset)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```

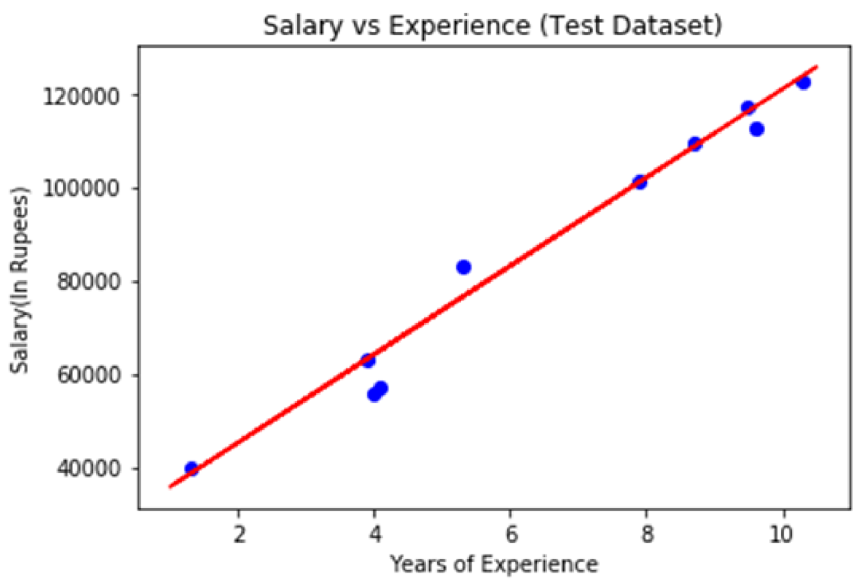
Output: By executing the above lines of code, we will get the below graph plot as an output.



Step: 4. visualizing the Test set results:

```
#visualizing the Test set results
mtp.scatter(x_test, y_test, color="blue")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Salary vs Experience (Test Dataset)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```

Output: By executing the above lines of code, we will get the below graph plot as an output.



PRACTICAL : 2

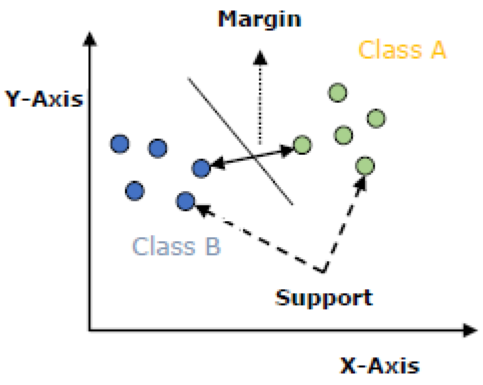
# Implement Support Vector Machine Classifier.

❖ Introduction to SVM

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

Working of SVM

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).



The followings are important concepts in SVM −

- **Support Vectors** − Datapoints that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.
- **Hyperplane** − As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.
- **Margin** − It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Python Implementation of Support Vector Machine

Now we will implement the SVM algorithm using Python. Here we will use the same dataset **user_data**, which we have used in Logistic regression and KNN classification.

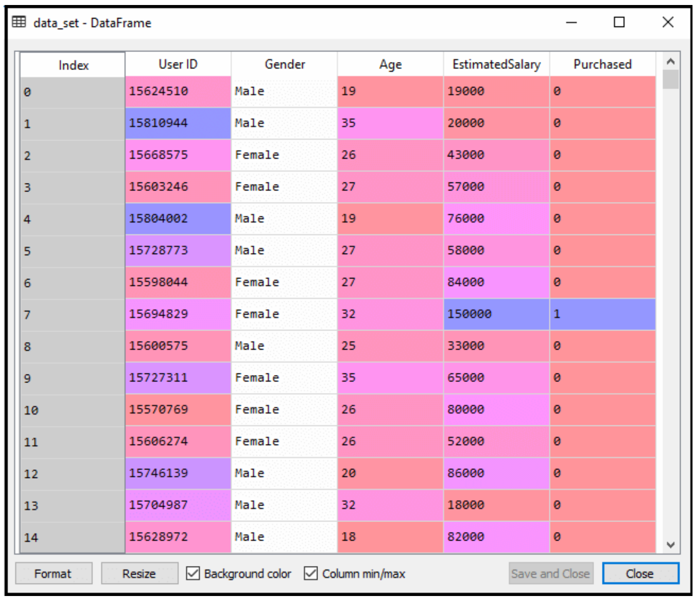
- **Data Pre-processing step**

Till the Data pre-processing step, the code will remain the same. Below is the code:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
#importing datasets
data_set= pd.read_csv('user_data.csv')
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

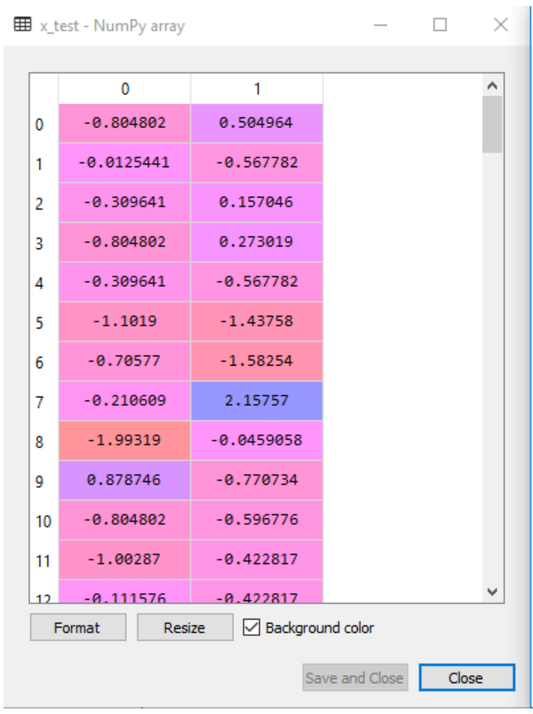
After executing the above code, we will pre-process the data. The code will give the dataset as:



data_set - DataFrame

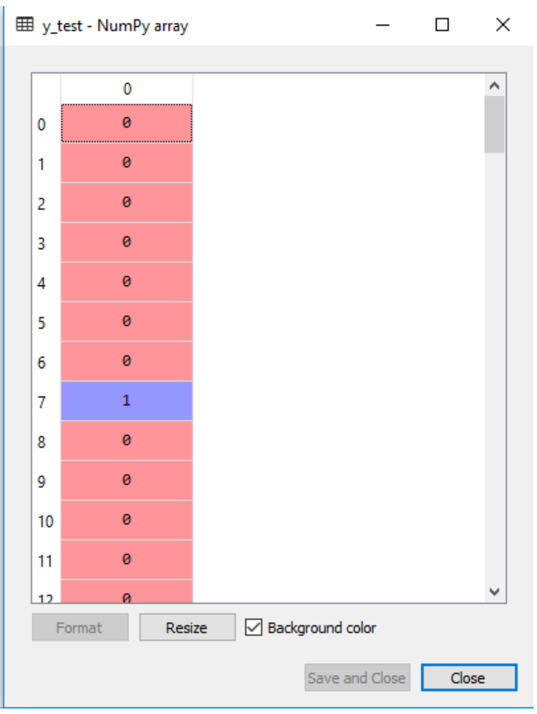
Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0

The scaled output for the test set will be:



x_test - NumPy array

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817



y_test - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

Fitting the SVM classifier to the training set:

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import **SVC** class from **Sklearn.svm** library. Below is the code for it:

```
m sklearn.svm import SVC # "Support vector classifier"
ssifier = SVC(kernel='linear', random_state=0)
ssifier.fit(x_train, y_train)
```

In the above code, we have used **kernel='linear'**, as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(x_train, y_train)

Output:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,

decision_function_shape='ovr', degree=3, gamma='auto_deprecated',

    kernel='linear', max_iter=-1, probability=False, random_state=0,

    shrinking=True, tol=0.001, verbose=False)
```

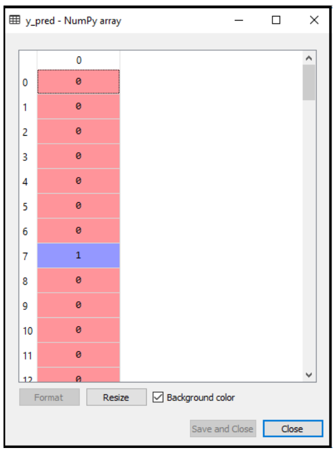
Predicting the test set result

Now, we will predict the output for test set. For this, we will create a new vector y_pred. Below is the code for it:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

After getting the y_pred vector, we can compare the result of **y_pred** and **y_test** to check the difference between the actual value and predicted value.

Output: Below is the output for the prediction of the test set:

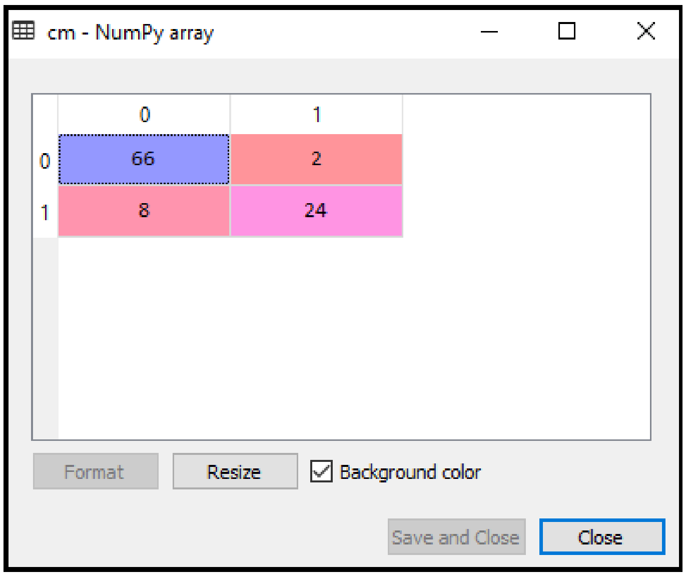


	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0

- **Creating the confusion matrix:**
  Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier. To create the confusion matrix, we need to import the **confusion_matrix** function of the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y_true**(the actual values) and **y_pred** (the targeted value return by the classifier). Below is the code for it:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

Output:


cm - NumPy array

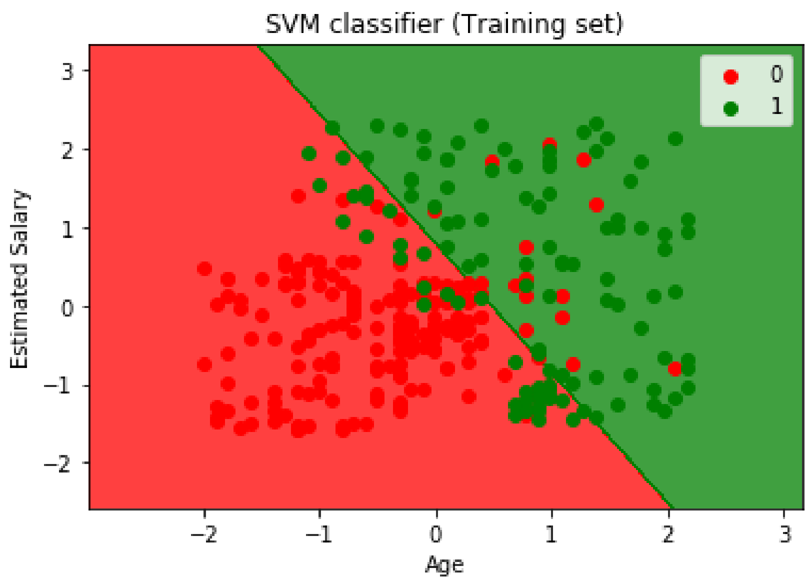
	0	1
0	66	2
1	8	24

As we can see in the above output image, there are 66+24= 90 correct predictions and 8+2= 10 correct predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

- **Visualizing the training set result:**
  Now we will visualize the training set result, below is the code for it:

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step
=0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
      c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output: By executing the above code, we will get the output as:



As we can see, the above output is appearing similar to the Logistic regression output. In the output, we got the straight line as hyperplane because we have **used a linear kernel in the classifier**. And we have also discussed above that for the 2d space, the hyperplane in SVM is a straight line.

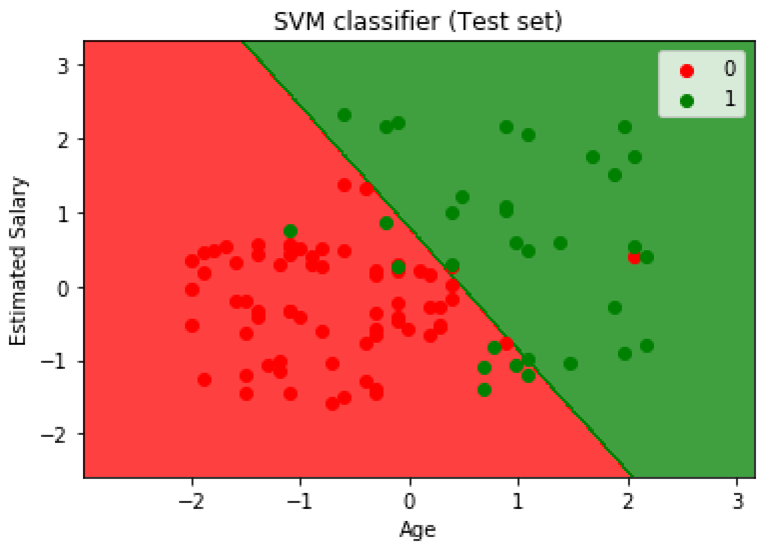
- **Visualizing the test set result:**

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, st
ep  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

By executing the above code, we will get the output as:



❖ SVM as Classification

Support Vector Machines (SVM) are a popular classification algorithm in machine learning. They are effective for both linear and nonlinear classification tasks. SVMs aim to find an optimal hyperplane that separates different classes in the feature space by maximizing the margin between the closest data points of each class. SVMs have several advantages, including their ability to handle high-dimensional data and their robustness against overfitting. They can also handle both binary and multiclass classification problems by using various techniques such as one-vs-one or one-vs-all. SVMs utilize a kernel function to map the input data into a higher-dimensional feature space, enabling them to handle nonlinear decision boundaries.

During training, SVMs aim to minimize the classification error and maximize the margin simultaneously, using optimization techniques. The resulting model can then be used to predict the class labels of new, unseen instances. SVMs are particularly useful when dealing with small to medium-sized datasets.

However, SVMs have some limitations. They can be computationally expensive, especially for large datasets. Additionally, SVMs may struggle with datasets that have overlapping classes or a large number of noisy or irrelevant features. In such cases, careful preprocessing and feature selection techniques may be required to improve performance. Overall, SVMs are a versatile and powerful classification algorithm in machine learning, suitable for various applications, including text categorization, image recognition, and bioinformatics.

❖ SVM as Regression

Support Vector Machines (SVM) can also be used for regression tasks in machine learning. This variant is known as Support Vector Regression (SVR). Unlike classification, where the goal is to find a decision boundary, SVR aims to find a function that predicts continuous numeric values.

In SVR, the algorithm seeks to find a hyperplane that best fits the training data while maintaining a certain margin called the ε-insensitive tube. The training instances that fall within this tube are considered correctly predicted, while those outside the tube are penalized based on their distance from the tube. The objective is to minimize the sum of these penalties while still fitting as many instances as possible within the tube.

Similar to classification, SVR can handle both linear and nonlinear regression problems using kernel functions. The kernel function allows the algorithm to map the input data into a higher-dimensional feature space, enabling it to capture nonlinear relationships.

During the training phase, SVR solves an optimization problem to determine the optimal hyperplane and support vectors, which are the data points that influence the position of the hyperplane. The resulting model can then be used to make predictions for new, unseen instances.

It's worth noting that SVR is sensitive to its hyperparameters, such as the regularization parameter (C) and the kernel function. Careful tuning of these parameters is essential to achieve good performance. Additionally, like SVM for classification, SVR may also be computationally expensive for large datasets.

Overall, SVR is a powerful regression algorithm in machine learning, suitable for tasks where the relationships between input variables and continuous target variables are complex or nonlinear.

PRACTICAL : 3

# Compute Model's Performance with Various Performance Metrics.

❖ Measuring Accuracy Using Cross-Validation

Cross-validation is a widely used technique for measuring the accuracy of machine learning models. It provides a more reliable estimate of a model's performance by mitigating the potential bias introduced by using a single train-test split.

The basic idea behind cross-validation is to divide the available dataset into multiple subsets or "folds." The model is then trained and evaluated multiple times, with each fold serving as the test set once while the remaining folds are used for training. The results from these iterations are averaged to obtain an overall performance estimate.

Here's a common cross-validation approach called k-fold cross-validation:

1. Split the dataset into k equally sized folds.
2. Iterate through each fold, treating it as the test set, and the remaining folds as the training set.
3. Train the model on the training set and evaluate its performance on the test set.
4. Repeat steps 2 and 3 for each fold, using a different fold as the test set each time.
5. Calculate the average performance metric across all iterations to obtain the overall model accuracy.

The choice of k (the number of folds) can vary depending on the dataset size and computational constraints. Common values for k include 5 and 10.

Cross-validation provides a more robust assessment of a model's performance by leveraging all available data for training and testing. It helps to identify potential issues like overfitting or underfitting and allows for better comparisons between different models or hyperparameter settings.

Some popular performance metrics used in cross-validation include accuracy, precision, recall, F1-score, and mean squared error (MSE), depending on the nature of the problem (classification or regression) and the evaluation criteria.

By using cross-validation, you can obtain a more accurate estimate of your model's performance and make informed decisions regarding its effectiveness.

Confusion Matrix

A **confusion matrix** is a matrix that summarizes the performance of a machine learning model on a set of test data. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance. The matrix displays the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) produced by the model on the test data.
For binary classification, the matrix will be of a 2X2 table, For multi-class classification, the matrix shape will be equal to the number of classes i.e for n classes it will be nXn.

A 2X2 Confusion matrix is shown below for the image recognization having a Dog image or Not Dog image.

		Actual	
		Dog	**Not Dog**
Predicted	**Dog**	True Positive (TP)	False Positive (FP)
	Not Dog	False Negative (FN)	True Negative (TN)

- **True Positive (TP):** It is the total counts having both predicted and actual values are Dog.
- **True Negative (TN):** It is the total counts having both predicted and actual values are Not Dog.
- **False Positive (FP):** It is the total counts having prediction is Dog while actually Not Dog.
- **False Negative (FN):** It is the total counts having prediction is Not Dog while actually, it is Dog.

Example

Index	1	2	3	4	5	6	7	8	9	10
Actual	Dog	Dog	Dog	Not Dog	Dog	Not Dog	Dog	Dog	Not Dog	Not Dog
Predicted	Dog	Not Dog	Dog	Not Dog	Dog	Dog	Dog	Dog	Not Dog	Not Dog
Result	TP	FN	TP	TN	TP	FP	TP	TP	TN	TN

- Actual Dog Counts = 6
- Actual Not Dog Counts = 4
- True Positive Counts = 5
- False Positive Counts = 1
- True Negative Counts = 3
- False Negative Counts = 1

		Actual	
		Dog	**Not Dog**
Predicted	**Dog**	True Positive (TP =5)	False Positive (FP=1)
	Not Dog	False Negative (FN =1)	True Negative (TN=3)

Confusion Matrix

Implementations of Confusion Matrix in Python

Steps:

- Import the necessary libraries like Numpy, confusion_matrix from sklearn.metrics, seaborn, and matplotlib.
- Create the NumPy array for actual and predicted labels.
- compute the confusion matrix.
- Plot the confusion matrix with the help of the seaborn heatmap.

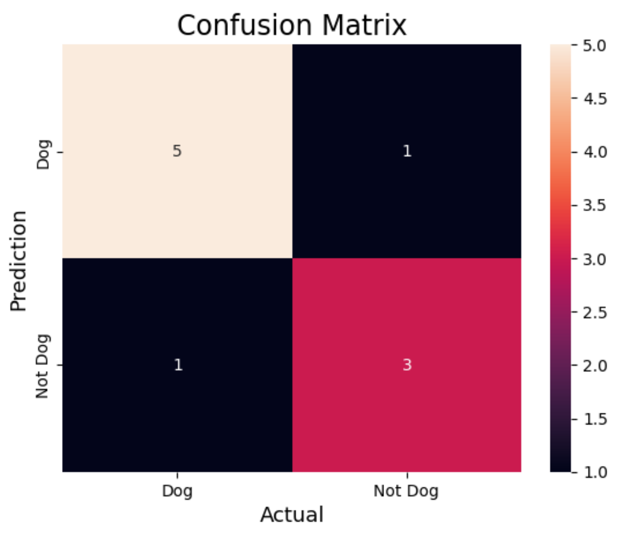
```
#Import the necessary libraries
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

#Create the NumPy array for actual and predicted labels.
actual = np.array(
['Dog','Dog','Dog','Not Dog','Dog','Not Dog','Dog','Dog','Not Dog','Not Dog'])
predicted = np.array(
['Dog','Not Dog','Dog','Not Dog','Dog','Dog','Dog','Dog','Not Dog','Not Dog'])

#compute the confusion matrix.
cm = confusion_matrix(actual,predicted)

#Plot the confusion matrix.
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['Dog','Not Dog'],
            yticklabels=['Dog','Not Dog'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```

Output:



Difference between Precision and Recall in Machine Learning

Precision	**Recall**
It helps us to measure the ability to classify positive samples in the model.	It helps us to measure how many positive samples were correctly classified by the ML model.
While calculating the Precision of a model, we should consider both Positive as well as Negative samples that are classified.	While calculating the Recall of a model, we only need all positive samples while all negative samples will be neglected.
When a model classifies most of the positive samples correctly as well as many false-positive samples, then the model is said to be a high recall and low precision model.	When a model classifies a sample as Positive, but it can only classify a few positive samples, then the model is said to be high accuracy, high precision, and low recall model.
The precision of a machine learning model is dependent on both the negative and positive samples.	Recall of a machine learning model is dependent on positive samples and independent of negative samples.
In Precision, we should consider all positive samples that are classified as positive either correctly or incorrectly.	The recall cares about correctly classifying all positive samples. It does not consider if any negative sample is classified as positive.