Scikit-learn is the most useful library for machine learning in Python. It provides a selection of efficient tools for ***machine learning*** and ***statistical modeling*** including classification, regression, clustering and diversionary reduction . Also scikit-learn focused on modeling the data. This library is built upon NumPy, Scipy and Matplotlib.This tutorial will explore statistical learning with scikit-learn.

1- Installation

```
pip install scikit-learn
```

or in Anaconda:

```
conda install scikit-learn
```

2- Features:

Some of models provided by sklearn are as follows:

- Supervised Learning algorithm
- Unsupervised Learning algorithm
- Clustering
- Cross Validation
- Dimensionality Reduction
- Ensemble methods
- Feature Extraction
- Feature Selection
- Open Source

3- Modelling Process

3-1- Dataset Loading

Dataset have two components:

- **Features**: variable of data are called its features. They are also known as predictors, inputs or attributes:

- Featue matrix: collection of features
- Feature names: list of all names of the features

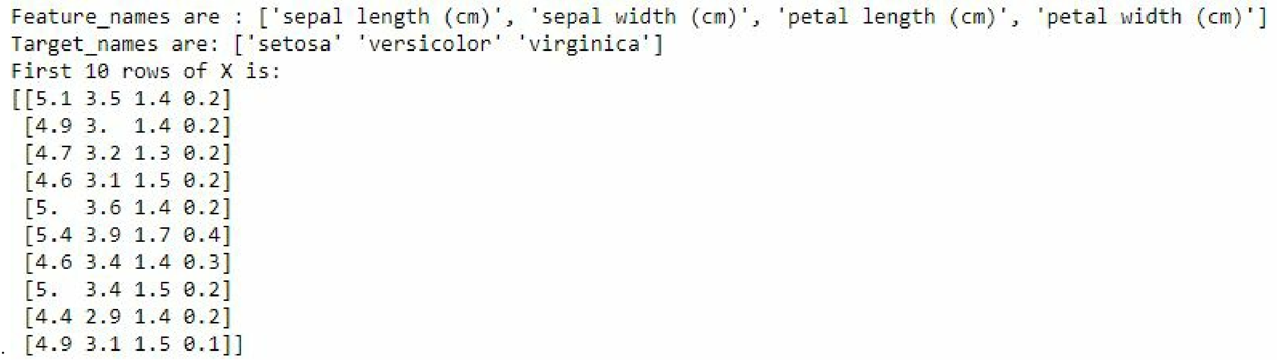
- **Response**: output variable that basically depends upon the feature variables. They are also known as target, label or output:

- Respone Vector: it is used to represent response column
- Target Names: it represent the possible values taken by a response vector

Scikit-learn have few example datasets like ***iris*** and ***digits*** for classification and the ***Boston house price*** for regression.

```
 from sklearn.datasets import load_iris
iris=load_iris()
X=iris.data
y=iris.target
feature_names=iris.feature_names
target_names=iris.target_names
print(f'Feature_names are : {feature_names}')
print(f'Target_names are: {target_names}')
print(f'First 10 rows of X is:\n{X[:10]}')
```

output:



```
Feature_names are : ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target_names are: ['setosa' 'versicolor' 'virginica']
First 10 rows of X is:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

3-2- Spliting the dataset

To check the accuracy of model, we can split the dataset into two pieces- ***a training set*** and ***a test set***. then use the training set to train the model nd testing set to test the model.

train_test_split(X,y,test_size,random_size):

-X: is feature matrix

-y: is response vector

-test_size: this represent the ratio of test data to the total given data.

-random_size: it is used to guarantee that the split will always be the same.

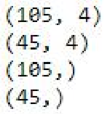
```
 from sklearn.datasets import load_iris
iris=load_iris()

X=iris.data
y=iris.target

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

output:



```
(105, 4)
(45, 4)
(105,)
(45,)
```

3-3- Train the model

Now, we can use our dataset to train some prediction-model. here for example we use KNN(have seprate chapter for that)

```
 from sklearn.datasets import load_iris
iris=load_iris()
X=iris.data
y=iris.target
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
y_pred=knn.predict(X_test)
print(f'Accuracy is: {metrics.accuracy_score(y_test,y_pred)}')
```

output:

Accuracy is: 0.9777777777777777

3-4- Preprocessing the Data

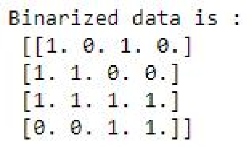
Before inputting data to machine learning algorithms, we need to convert it into meaningful data. This process is called **Preprocessing the Data**. Scikit-learn has package named ***preprocessing***.

3-4-1- Binarisation

for example we need to convert numerical values into Boolean values. here , use threshold value=0.5 , then all values above 0.5 would be converted to 1, and other value converted to 0.

```
 import numpy as np
from sklearn import preprocessing
data=np.array([[2.1,-5,6.3,-0.1],
                    [3.9,9.8,-6.3,-7],
                   [2,3.8,6,5],
                    [-8,0,2.1,6]])
print(f'Binarized data is :\n {preprocessing.Binarizer(threshold=0.5).transform(data)}')
```

output:

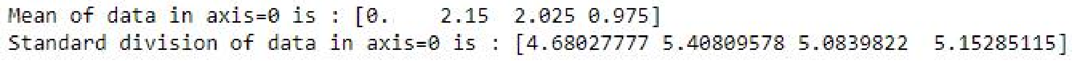


```
Binarized data is :
 [[1. 0. 1. 0.]
 [1. 1. 0. 0.]
 [1. 1. 1. 1.]
 [0. 0. 1. 1.]]
```

3-4-2- Mean and Standard division

```
 import numpy as np
from sklearn import preprocessing
data=np.array([[2.1,-5,6.3,-0.1],
                    [3.9,9.8,-6.3,-7],
                   [2,3.8,6,5],
                    [-8,0,2.1,6]])
print(f'Mean of data in axis=0 is : {data.mean(axis=0)}')
print(f'Standard division of data in axis=0 is : {data.std(axis=0)}')
```

output:



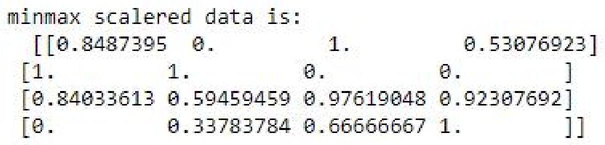
```
Mean of data in axis=0 is : [0.     2.15   2.025 0.975]
Standard division of data in axis=0 is : [4.68027777 5.40809578 5.0839822  5.15285115]
```

3-4-3- Scaling

We use scaling technique, because the features should not be synthetically large or small.

```
 import numpy as np
from sklearn import preprocessing
data=np.array([[2.1,-5,6.3,-0.1],
                    [3.9,9.8,-6.3,-7],
                   [2,3.8,6,5],
                    [-8,0,2.1,6]])
scaler=preprocessing.MinMaxScaler(feature_range=(0,1))
data_scalered=scaler.fit_transform(data)
print(f'minmax scalered data in is \n : {data_scalered}')
```

output:



```
minmax scalered data is:
  [[0.8487395  0.         1.         0.53076923]
 [1.         1.         0.         0.        ]
 [0.84033613 0.59459459 0.97619048 0.92307692]
 [0.         0.33783784 0.66666667 1.        ]]
```

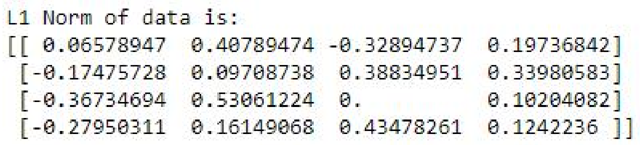
3-4-4- Normalization

We use Normalization technique for modify the feature vectors. Normalization is necessary so that the feature vectors can be measured t common scale. there are two type of normalization: L1 Normalization and L2 Normalization

L1 Normalization also called Least Absolute Deviations. It modifies the value in such a manner that the sum of the absolute values remains always up to 1 in each row.

```
 import numpy as np
from sklearn import preprocessing
data=np.array([[1,6.2,-5,3],
              [-3.6,2,8,7],
              [-3.6,5.2,0,1],
              [-9,5.2,14,4]])
normalize_data=preprocessing.normalize(data,norm='l1')
print(f'L1 Norm of data is:\n{normalize_data}')
```

output:

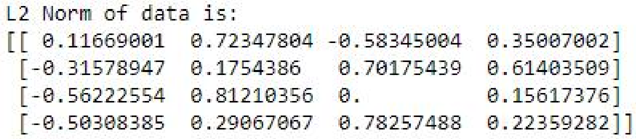


```
L1 Norm of data is:
[[ 0.06578947  0.40789474 -0.32894737  0.19736842]
 [-0.17475728  0.09708738  0.38834951  0.33980583]
 [-0.36734694  0.53061224  0.          0.10204082]
 [-0.27950311  0.16149068  0.43478261  0.1242236 ]]
```

L2 Normalization also called Least Squares. It modifies the value in such a manner that the sum of the squares remains always up to 1 in each row.

```
 import numpy as np
from sklearn import preprocessing
data=np.array([[1,6.2,-5,3],
              [-3.6,2,8,7],
              [-3.6,5.2,0,1],
              [-9,5.2,14,4]])
normalize_data=preprocessing.normalize(data,norm='l2')
print(f'L2 Norm of data is:\n{normalize_data}')
```

output:



```
L2 Norm of data is:
[[ 0.11669001  0.72347804 -0.58345004  0.35007002]
 [-0.31578947  0.1754386   0.70175439  0.61403509]
 [-0.56222554  0.81210356  0.          0.15617376]
 [-0.50308385  0.29067067  0.78257488  0.22359282]]
```