4- Data Representation

As we know machine learning is about to create model from data. for this purpose, computer must understand the data first. here, we are going to discuss various ways to represent the data:

4-1- Data as Table

the best way to represent data is the form of tables.A table represents a 2-D grid of data where rows represent the individual elements of the database and columns represents the quantities related to those individual elements. as you see, each column of the data represents a quantitative information describing each sample.

```
 import seaborn as sns
dat=sns.load_dataset('iris')
dat.head()
```

output:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

4-2- Data as Feature Matrix

Feature matrix may be defined as the table layout where information can be thought of as a 2-D matrix. it is stored in a variable named **X** and assumed to be two dimensional with shape[n_samples,n_features]. Mostly, it is contained in a NumPy array or Pandas DataFrame. The samples always represent the individual objects described by the dataset and the features represent the distinct observations that describe each sample in a quantitative manner.

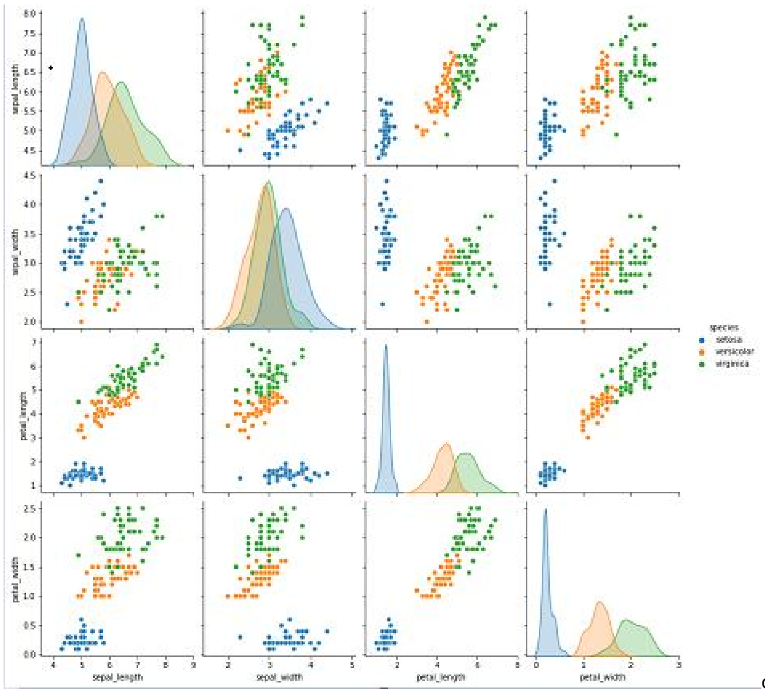
4-3- Data as Target array

It is also called **label**. it is denoted by **y**. The label or target array is usually one-dimensional having length n_samples. It is generally contained in NumPy ***array*** or Pandas ***Series***. Target array may have both the values, continuous numerical values and discrete values.

we can distinguish target array from feature columns by one point that the target array is usually the quantity we want to predict from the data. in statistical term it is the dependent variable.

in the example, from iris dataset predict the species of flower based on other measurement.

```
 import seaborn as sns
data=sns.load_dataset('iris')
sns.pairplot(data,hue='species',height=3)
```



output:

5- Estimator API

It is one of the main APIs implemented by Scikit-learn. It provides a consistent interface for a wide range of ML applications that's why all machine learning algorithms in Scikit-Learn are implemented via Estimator API. The object that learns from the data (fitting the data) is an estimator. It can be used with any of the algorithms like classification, regression, clustering or even with a transformer, that extracts useful features from row data.

For fitting the data, all estimator objects expose a fit method that takes a dataset shown as follows:

```
estimator.fit(data)
```

Next, all the parameters of an estimator can be set, as follows, when it is instantiated by the corresponding attribute:

```
estimator= Estimator (param1=1, param2=2)
```

Ones data is fitted with an estimator, parameters are estimated from the data at hand.

5-1- Use of Estimator API

Estimator object is used for estimation and decoding of a model. Furthermore, the model is estimated as a deterministic function of the following:

- The parameters which are provided in object construction.
- The global random state(numpy.random) if the estimator's random_state parameter is set to none.
- Any data passed to the most recent call to **fit, fit_transform, fit_predict**
- Any data passed in a sequence of calls to **partial_fit**

5-2- Steps in using Estimator API1. Choose a class of model 1. Choose model hyperparameters 1. Arranging the data 1. Model Fitting 1. Applying the model

6- Scikit Learn Conventions

Scikit-learn's object share a uniform basic API that consist of the following three complementary interfaces:

- Estimator interface: it is for building and fitting the models.

- Predictor interface: it is for making predictions.

- Transformer interface: it is for converting data.

Various Conventions 6-1- Type Casting

It states that the input should be cast to float64.

```
 import numpy as np
from sklearn import random_projection
rannage=np.random.RandomState(0)
X=rannage.rand(10,2000)
X=np.array(X,dtype='float32')
print(X.dtype)
Transformer_data=random_projection.GaussianRandomProjection()
X_new=Transformer_data.fit_transform(X)
print(X_new.dtype)
```

output:

float32

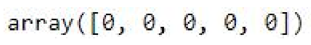
float64

6-2- Refitting and Updating Parameters

Hyper-parameters of an estimator can be updated and refitted after it has been constructed via the **set_params()**

```
 import numpy as np
from sklearn.datasets import load_iris
from sklearn.svm import SVC
X,y=load_iris(return_X_y=True)
clf=SVC()
clf.set_params(kernel='linear').fit(X,y)
clf.predict(X[:5])
```

output:

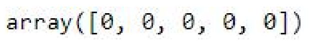


```
array([0, 0, 0, 0, 0])
```

Now we can change back the kernel to rbf to refit the estimator and to make a second prediction:

```
 import numpy as np
from sklearn.datasets import load_iris
from sklearn.svm import SVC
X,y=load_iris(return_X_y=True)
clf=SVC()
# clf.set_params(kernel='linear').fit(X,y)
# clf.predict(X[:5])
clf.set_params(kernel='rbf',gamma='scale').fit(X,y)
clf.predict(X[:5])
```

output:



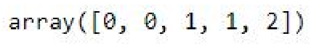
```
array([0, 0, 0, 0, 0])
```

6-3- Multiclass and Multilabel fitting

In case of multiclass fitting, both learning and the prediction takes are dependent on the format of the target data fit upon.The module used is **sklearn.multiclass**.

```
 from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import LabelBinarizer
X=[[1,2],[3,5],[4,8],[1,1],[5,2]]
y=[0,0,1,1,2]
classif=OneVsRestClassifier(estimator=SVC(gamma='scale',random_state=0))
classif.fit(X,y).predict(X)
```

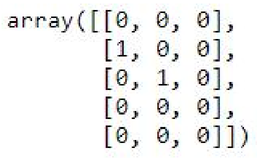
output:



```
array([0, 0, 1, 1, 2])
```

```
 from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import LabelBinarizer
X=[[1,2],[3,5],[8,4],[1,2],[5,3]]
y=LabelBinarizer().fit_transform(y)
classif.fit(X,y).predict(X)
```

output:



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