MACHINE LEARNING

Submitted by,

- Name:Jagannath v v
- Reg.No.-21122024
- class:1st MSc DataScience

LAB OVERVIEW

Objectives

PART 1 - EXPLORATION Explore the below subparts of the module sklearn

- a. train_test_split from sklearn.model_selection
- b. make_classification and load_iris from sklearn.datasets
- c. make_regression and load_boston from sklearn.datasets
- d. DummyClassifier and DummyRegressor from sklearn.dummy
- e. accuracy_score, classification_report, and confusion_matrix from sklearn.metrics

PART 2 - Questions

- What are the different parameters of the above functions/methods that are part of the above SKLearn modules?
- What is the effect, when you modify certain parameters that are present in the same?
- How to get different train and test datasets?
- Identify which are features and which are targets in Part 1b, and Part 1c (make_classification and make_regression would depend on your inputs on the function call)
- Identify what the things mentioned in Part 1d stands for.
- Making use of Part 1d, explore the various options available under Part 1e.

Problem Definition

Here we try to explore some subparts of module sklearn.

- 1). train_test_split
- 2). make_classification and load_iris
- 3). make_regression and load_boston
- 4). DummyClassifier and DummyRegressor
- 5). accuracy_score, classification_report, and confusion_matrix Then we did some examples on it**

Approach

Here i reffered internet to find some more about the libraries

PART 1 - EXPLORATION

LibrarY

SKLEARN Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

IMPORTING SKLEARN

In [60]:
```
import sklearn
```

A)train_test_split from sklearn.model_selection

- **train_test_split is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data. With this function, you don't need to divide the dataset manually.**
- **Sklearn train_test_split will make random partitions for the two subsets. However, you can also specify a random state for the operation.**

In [59]:
```
# split a dataset into train and test sets
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
# create dataset
X, y = make_blobs(n_samples=150)
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.55)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(67, 2) (83, 2) (67,) (83,)
```

B)make_classification and load_iris from sklearn.datasets

Generate a random n-class classification problem.This initially creates clusters of points normally distributed (std=1) about vertices of an n_informative -dimensional hypercube with sides of length 2*class_sep and assigns an equal number of clusters to each class.

In [16]:
```
from sklearn.datasets import load_iris
data = load_iris()
```

In [20]:
```
sklearn.datasets.make_classification(
    n_samples=100, # no.of samples
    n_features=20, #no.of features
    n_informative=2, #no.of inormative features
    n_redundant=2, #The number of redundant features.
    n_repeated=0, # no.of duplicated features
    n_classes=2, #no. of classes
    shuffle=True, #Shuffle the samples and the features
    random_state=None) #Determines random number generation for dataset creation.
```

Out[20]:
```
(array([[-1.7329251 , -1.19158129, -1.08139217, ...,  1.92329322,
          0.6905207 ,  0.35887058],
        [ 1.6452431 ,  1.43560321,  1.91337001, ..., -0.79313824,
          0.09984434, -0.71811141],
        [-0.69428886, -0.95608094,  2.2863974 , ...,  1.12644453,
          0.26772815, -0.71159645],
        ...,
        [-1.75357377, -1.57976   ,  1.28292229, ...,  0.61710777,
          1.74316841, -0.47107791],
        [-1.17018874,  0.3649127 , -1.11281125, ...,  1.09853114,
         -0.74092822, -0.90302422],
        [ 0.30518959,  0.56413645, -0.94441993, ...,  1.29298457,
          0.62760659, -0.79019038]]),
 array([0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
        0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
        1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0]))
```

In [17]:
```
data
```

Out[17]:
```
{'data': array([[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],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
        [4.8, 3.1, 1.6, 0.2],
        [5.4, 3.4, 1.5, 0.4],
        [5.2, 4.1, 1.5, 0.1],
        [5.5, 4.2, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.2],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.6, 1.4, 0.1],
        [4.4, 3. , 1.3, 0.2],
        [5.1, 3.4, 1.5, 0.2],
        [5. , 3.5, 1.3, 0.3],
        [4.5, 2.3, 1.3, 0.3],
        [4.4, 3.2, 1.3, 0.2],
        [5. , 3.5, 1.6, 0.6],
        [5.1, 3.8, 1.9, 0.4],
```

```
[4.8, 3. , 1.4, 0.3],
[5.1, 3.8, 1.6, 0.2],
[4.6, 3.2, 1.4, 0.2],
[5.3, 3.7, 1.5, 0.2],
[5. , 3.3, 1.4, 0.2],
[7. , 3.2, 4.7, 1.4],
[6.4, 3.2, 4.5, 1.5],
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
```

```
        [5.8, 2.8, 5.1, 2.4],
        [6.4, 3.2, 5.3, 2.3],
        [6.5, 3. , 5.5, 1.8],
        [7.7, 3.8, 6.7, 2.2],
        [7.7, 2.6, 6.9, 2.3],
        [6. , 2.2, 5. , 1.5],
        [6.9, 3.2, 5.7, 2.3],
        [5.6, 2.8, 4.9, 2. ],
        [7.7, 2.8, 6.7, 2. ],
        [6.3, 2.7, 4.9, 1.8],
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data
Set Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classe
s)\n    :Number of Attributes: 4 numeric, predictive attributes and the class\n    :
Attribute Information:\n        - sepal length in cm\n        - sepal width in cm\n
- petal length in cm\n        - petal width in cm\n        - class:\n
- Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virginica
\n                \n    :Summary Statistics:\n\n    ============== ==== ==== =======
===== ====================\n                    Min  Max   Mean    SD   Class Correl
ation\n    ============== ==== ==== ======= ===== ====================\n    sepal le
ngth:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43
-0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal wi
dth:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==== ====
=== ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Di
stribution: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Mic
hael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Ir
is database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s pap
er. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Reposi
tory, which has two wrong data points.\n\nThis is perhaps the best known database to
be found in the\npattern recognition literature.  Fisher\'s paper is a classic in th
e field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)
The\ndata set contains 3 classes of 50 instances each, where each class refers to a
\ntype of iris plant.  One class is linearly separable from the other 2; the\nlatter
are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher,
R.A. "The use of multiple measurements in taxonomic problems"\n     Annual Eugenics,
```

```
7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics"
(John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification
and Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See pa
ge 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n
Structure and Classification Rule for Recognition in Partially Exposed\n     Environ
ments".  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol.
PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".
IEEE Transactions\n     on Information Theory, May 1972, 431-433.\n   - See also: 19
88 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clusteri
ng system finds 3 classes in the data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'C:\\Users\\jagan\\anaconda3\\lib\\site-packages\\sklearn\\datasets\\da
ta\\iris.csv'}
```

c)make_regression and load_boston from sklearn.datasets

- It is a function in sklearn_datasets which is used to generate our dataset for the regression problem
- It is described in the code in python below.

IMPORTING MAKE_REGRESSION

In [3]:
```
from sklearn.datasets import make_regression
```

In [5]:
```
from sklearn.datasets import load_boston
import pandas as pd
X, y = load_boston(return_X_y=True)
```

In [10]:
```
X.shape
```

Out[10]: (506, 13)

d. DummyClassifier and DummyRegressor from sklearn.dummy

- DummyClassifier makes predictions that ignore the input features.
- This classifier serves as a simple baseline to compare against other more complex classifiers.

In [37]:
```
from sklearn.dummy import DummyClassifier
```

- A DummyClassifier is a classifier in the sklearn library that makes predictions using simple rules and does not generate any valuable insights about the data.
- dummy classifiers are used as a baseline and can be compared to real classifiers and thus we must not use it for actual problems.

- All the other (real) classifiers are expected to perform better on any dataset when compared to the dummy classifier. The classifier does not take into account the training data and instead uses one of the strategies to predict the class label. Stratified, most frequent, constant, and uniform are a few of the strategies used by dummy classifiers.
- We will implement all these strategies in our code below and check out the results.

In [36]:
```
from sklearn.dummy import DummyRegressor
```

- The Dummy Regressor is a kind of Regressor that gives prediction based on simple strategies without paying any attention to the input Data.

In [ ]:
```

```

e. accuracy_score, classification_report, and confusion_matrix from sklearn.metrics

Accuracy_score:

- this functions computes the accuracy

syntax

```
accuracy_score(true_label, predicted)
```

classificaton_report:

- used to measure the of prediction from classification algorithm

syntax

```
classification_report(true_label, predicted)
```

Confusion matrix

- used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

syntax

```
confusionmatrix(y_pred,y_true)
```

In [ ]:
```

```

PART 2 - Questions

(Q)What are the different parameters of the above functions/methods that are part of the above SKLearn modules?

1)train_test_split

Parameters

- Arrays:sequence of indexables with same length / shape[0] Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.
- test_size:float or int, default=None If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If train_size is also None, it will be set to 0.25.
- train_size:float or int, default=None If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.
- random_state:int, RandomState instance or None, default=None Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.
- shuffle:bool, default=True Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.
- stratify:array-like, default=None

2)make classification

Parameters

- n_samplesint The number of samples.
- n_featuresint The total number of features. These comprise n_informative informative features, n_redundant redundant features, n_repeated duplicated features and n_features-n_informative-n_redundant-n_repeated useless features drawn at random.
- n_informativeint The number of informative features. Each class is composed of a number of gaussian clusters each located around the vertices of a hypercube in a subspace of dimension n_informative. For each cluster, informative features are drawn independently from N(0, 1) and then randomly linearly combined within each cluster in order to add covariance. The clusters are then placed on the vertices of the hypercube.
- n_redundantint The number of redundant features. These features are generated as random linear combinations of the informative features.
- n_repeatedint The number of duplicated features, drawn randomly from the informative and the redundant features.
- n_classesint The number of classes (or labels) of the classification problem.
- n_clusters_per_classint The number of clusters per class.
- weightsarray-like of shape (n_classes,) or (n_classes - 1,) The proportions of samples assigned to each class. If None, then classes are balanced. Note that if len(weights) == n_classes - 1, then the last class weight is automatically inferred. More than n_samples

samples may be returned if the sum of weights exceeds 1. Note that the actual class proportions will not exactly match weights when flip_y isn't 0.

- flip_yfloat The fraction of samples whose class is assigned randomly. Larger values introduce noise in the labels and make the classification task harder. Note that the default setting flip_y > 0 might lead to less than n_classes in y in some cases.
- class_sepfloat The factor multiplying the hypercube size. Larger values spread out the clusters/classes and make the classification task easier.
- hypercubebool If True, the clusters are put on the vertices of a hypercube. If False, the clusters are put on the vertices of a random polytope.
- shiftfloat, ndarray of shape (n_features,) or None Shift features by the specified value. If None, then features are shifted by a random value drawn in [-class_sep, class_sep].
- scalefloat, ndarray of shape (n_features,) or None Multiply features by the specified value. If None, then features are scaled by a random value drawn in [1, 100]. Note that scaling happens after shifting.
- shufflebool Shuffle the samples and the features.
- random_stateint, RandomState instance or None Determines random number generation for dataset creation. Pass an int for reproducible output across multiple function calls. See Glossary.

3) make_regression

Parameters

- n_samplesint: The number of samples.
- n_featuresint The number of features.
- n_informativeint The number of informative features, i.e., the number of features used to build the linear model used to generate the output.
- n_targetsint The number of regression targets, i.e., the dimension of the y output vector associated with a sample. By default, the output is a scalar.
- biasfloat The bias term in the underlying linear model.
- effective_rankint if not None: The approximate number of singular vectors required to explain most of the input data by linear combinations. Using this kind of singular spectrum in the input allows the generator to reproduce the correlations often observed in practice.

  if None: The input set is well conditioned, centered and gaussian with unit variance.
- tail_strengthfloat The relative importance of the fat noisy tail of the singular values profile if effective_rank is not None. When a float, it should be between 0 and 1.
- noisefloat The standard deviation of the gaussian noise applied to the output.
- shufflebool Shuffle the samples and the features.

- coefbool If True, the coefficients of the underlying linear model are returned.
- random_stateint, RandomState instance or None Determines random number generation for dataset creation. Pass an int for reproducible output across multiple function calls. See Glossary.

4)DummyClassifier

Parameters

- "most_frequent": the predict method always returns the most frequent class label in the observed y argument passed to fit. The predict_proba method returns the matching one-hot encoded vector.
- "prior": the predict method always returns the most frequent class label in the observed y argument passed to fit (like "most_frequent"). predict_proba always returns the empirical class distribution of y also known as the empirical class prior distribution.
- "stratified": the predict_proba method randomly samples one-hot vectors from a multinomial distribution parametrized by the empirical class prior probabilities. The predict method returns the class label which got probability one in the one-hot vector of predict_proba. Each sampled row of both methods is therefore independent and identically distributed.
- "uniform": generates predictions uniformly at random from the list of unique classes observed in y, i.e. each class has equal probability.
- "constant": always predicts a constant label that is provided by the user. This is useful for metrics that evaluate a non-majority class.
- random_stateint, RandomState instance or None, default=None Controls the randomness to generate the predictions when strategy='stratified' or strategy='uniform'. Pass an int for reproducible output across multiple function calls. See Glossary.
- constantint or str or array-like of shape (n_outputs,), default=None The explicit constant as predicted by the "constant" strategy. This parameter is useful only for the "constant" strategy.

5)Make classification

Parameters

- strategy{"mean", "median", "quantile", "constant"} Strategy to use to generate predictions.

"mean": always predicts the mean of the training set

"median": always predicts the median of the training set

"quantile": always predicts a specified quantile of the training set, provided with the quantile parameter.

"constant": always predicts a constant value that is provided by the user.

- constantint or float or array-like of shape (n_outputs,): The explicit constant as predicted by the “constant” strategy. This parameter is useful only for the “constant” strategy.
- quantilefloat in [0.0, 1.0] The quantile to predict using the “quantile” strategy. A quantile of 0.5 corresponds to the median, while 0.0 to the minimum and 1.0 to the maximum.
- Attributes constant_ndarray of shape (1, n_outputs) Mean or median or quantile of the training targets or constant value given by the user.
- n_features_in_None DEPRECATED: n_features*in* is deprecated in 1.0 and will be removed in 1.2.
- n_outputs_int Number of outputs.

6)accuracy_score

Parameters

- y_true1d array-like, or label indicator array / sparse matrix Ground truth (correct) labels.
- y_pred1d array-like, or label indicator array / sparse matrix Predicted labels, as returned by a classifier.
- normalizebool If False, return the number of correctly classified samples. Otherwise, return the fraction of correctly classified samples.
- sample_weightarray-like of shape (n_samples,) Sample weights.

7)Make classification

Parameters

- y_true1d array-like, or label indicator array / sparse matrix Ground truth (correct) target values.
- y_pred1d array-like, or label indicator array / sparse matrix Estimated targets as returned by a classifier.
- labelsarray-like of shape (n_labels,), default=None Optional list of label indices to include in the report.
- target_nameslist of str of shape (n_labels,), default=None Optional display names matching the labels (same order).
- sample_weightarray-like of shape (n_samples,), default=None Sample weights.
- digitsint, default=2 Number of digits for formatting output floating point values. When output_dict is True, this will be ignored and the returned values will not be rounded.
- output_dictbool, default=False If True, return output as dict.
- zero_division“warn”, 0 or 1, default=”warn” Sets the value to return when there is a zero division. If set to “warn”, this acts as 0, but warnings are also raised.

confusion_matrix parameters

Parameters

- y_truearray-like of shape (n_samples,) Ground truth (correct) target values.
- y_predarray-like of shape (n_samples,) Estimated targets as returned by a classifier.
- labelsarray-like of shape (n_classes), default=None List of labels to index the matrix. This may be used to reorder or select a subset of labels. If None is given, those that appear at least once in y_true or y_pred are used in sorted order.
- sample_weightarray-like of shape (n_samples,), default=None Sample weights.
- normalize{'true', 'pred', 'all'}, default=None Normalizes confusion matrix over the true (rows), predicted (columns) conditions or all the population. If None, confusion matrix will not be normalized.

(Q)What is the effect, when you modify certain parameters that are present in the same?

Setting the right parameter values is very important because it directly impacts the performance of the model that will result from them being used during model training.So the result will change if the parameter changes

In [56]:
```
# for example
import sklearn
sklearn.datasets.make_classification(
    n_samples=100, # no.of samples
    n_features=20, #no.of features
)
```

Out[56]:
```
(array([[ 0.41337271, -1.85941819, -0.97719729, ..., -0.5497276 ,
         -0.05129686,  1.30828882],
        [-2.0179403 ,  0.90522797,  1.04712012, ..., -1.24466694,
          0.28611898,  0.75653211],
        [-0.48415669, -0.28027442, -2.02825739, ..., -1.43446392,
          0.55572686, -0.33717738],
        ...,
        [ 0.07595304,  2.26836168, -0.94771406, ...,  1.81504292,
         -0.14850981,  1.75021079],
        [ 0.04486688, -2.00209007, -0.29141225, ..., -0.13262084,
         -0.76331302,  0.08066363],
        [-0.02368497, -0.73655863,  1.71986747, ..., -0.37200301,
         -1.50652592,  1.49891768]]),
 array([0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,
        1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0,
        0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0,
        1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1,
        0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1]))
```

In [57]:
```
sklearn.datasets.make_classification(
    n_samples=100, # no.of samples
    n_features=70, #no.of features
)
```

Out[57]:
```
(array([[-0.55979283, -0.07814939, -1.60217433, ...,  0.00518184,
         -0.14664888,  2.14007251],
        [ 1.31119706, -0.40110704,  1.09064056, ...,  0.37753368,
          1.68400061, -0.37111349],
```

```
        [ 1.80661723, -1.06817963, -0.18373386, ...,  0.52109668,
          0.259209  ,  0.39229005],
        ...,
        [-2.57160569, -0.62233944, -1.24486755, ...,  0.9101838 ,
         -1.10139871,  2.07076567],
        [-0.16128214, -0.83869438, -0.04134701, ...,  0.21090354,
         -0.90060104, -0.32925131],
        [-0.38674603,  0.02161866, -1.88113563, ...,  0.41631828,
          0.06426276,  0.09869218]]),
 array([1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
        0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
        1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1,
        0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
        1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1]))
```

so here we can the values were changed

(Q)How to get different train and test datasets?

Importance of Training and Testing Sets

The most common procedure when training a (basic) model in Machine Learning follows the same rough outline:

Acquiring and processing data which we'll feed into a model. Scikit-learn has various datasets to be loaded and used for training the model (iris, diabetes, digits...), mainly for benchmarking/learning.

Splitting sets into training and test sets Building a model and defining the architecture Compiling the model Training the model Verifying the results

In [7]:
```
from sklearn.model_selection import train_test_split# importing
```

The main two arguments are train_size and test_size, where their values range between 0 and 1 and their sum has to be 1. Their values denote the percentage proportion of the dataset, so even if you define just one, such as train_size, the test_size is equal to 1 - train_size, and vice versa.

In [2]:
```
# Setting the train_size and test_size Arguments
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y)
```

In [3]:
```
print(X_train.shape)
```

```
(112, 4)
```

In [4]:
```
print(X_test.shape)
```

```
(38, 4)
```

In [5]:
```
print(y_train.shape)
```

```
(112,)
```

In [6]:
```
print(y_test.shape)
```

```
(38,)
```

(Q)Identify which are features and which are targets in Part 1b, and Part 1c (make_classification and make_regression would depend on your inputs on the function call)

In [11]:
```
from sklearn.datasets import load_iris
data = load_iris()
```

In [12]:
```
data
```

Out[12]:
```
{'data': array([[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],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
```

```
[6.9, 3.1, 4.9, 1.5],
[5.5, 2.3, 4. , 1.3],
[6.5, 2.8, 4.6, 1.5],
[5.7, 2.8, 4.5, 1.3],
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
[6.8, 3. , 5.5, 2.1],
[5.7, 2.5, 5. , 2. ],
[5.8, 2.8, 5.1, 2.4],
[6.4, 3.2, 5.3, 2.3],
[6.5, 3. , 5.5, 1.8],
[7.7, 3.8, 6.7, 2.2],
[7.7, 2.6, 6.9, 2.3],
[6. , 2.2, 5. , 1.5],
[6.9, 3.2, 5.7, 2.3],
```

```
        [5.6, 2.8, 4.9, 2. ],
        [7.7, 2.8, 6.7, 2. ],
        [6.3, 2.7, 4.9, 1.8],
        [6.7, 3.3, 5.7, 2.1],
        [7.2, 3.2, 6. , 1.8],
        [6.2, 2.8, 4.8, 1.8],
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data
Set Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classe
s)\n    :Number of Attributes: 4 numeric, predictive attributes and the class\n    :
Attribute Information:\n        - sepal length in cm\n        - sepal width in cm\n
- petal length in cm\n        - petal width in cm\n        - class:\n
- Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virginica
\n                \n    :Summary Statistics:\n\n    ============== ==== ==== =======
===== ====================\n                    Min  Max   Mean    SD   Class Correl
ation\n    ============== ==== ==== ======= ===== ====================\n    sepal le
ngth:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4   3.05   0.43
-0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal wi
dth:    0.1  2.5   1.20   0.76    0.9565  (high!)\n    ============== ==== ==== ====
=== ===== ====================\n\n    :Missing Attribute Values: None\n    :Class Di
stribution: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Mic
hael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Ir
is database, first used by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s pap
er. Note that it\'s the same as in R, but not as in the UCI\nMachine Learning Reposi
tory, which has two wrong data points.\n\nThis is perhaps the best known database to
be found in the\npattern recognition literature.  Fisher\'s paper is a classic in th
e field and\nis referenced frequently to this day.  (See Duda & Hart, for example.)
The\ndata set contains 3 classes of 50 instances each, where each class refers to a
\ntype of iris plant.  One class is linearly separable from the other 2; the\nlatter
are NOT linearly separable from each other.\n\n.. topic:: References\n\n   - Fisher,
R.A. "The use of multiple measurements in taxonomic problems"\n     Annual Eugenics,
7, Part II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistics"
(John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification
and Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See pa
ge 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n
Structure and Classification Rule for Recognition in Partially Exposed\n     Environ
ments".  IEEE Transactions on Pattern Analysis and Machine\n     Intelligence, Vol.
PAMI-2, No. 1, 67-71.\n   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".
```

```
IEEE Transactions\n     on Information Theory, May 1972, 431-433.\n   - See also: 19
88 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clusteri
ng system finds 3 classes in the data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'C:\\Users\\jagan\\anaconda3\\lib\\site-packages\\sklearn\\datasets\\da
ta\\iris.csv'}
```

In [17]:
```
data.target
```

Out[17]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [16]:
```
data.target_names
```

Out[16]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [18]:
```
data.feature_names
```

Out[18]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

Targets are :

- 'setosa'
- 'versicolor'
- 'virginica'

Features are:

- 'sepal length (cm)',
- 'sepal width (cm)',
- 'petal length (cm)',
- 'petal width (cm)'

(Q)Identify what the things mentioned in Part 1d stands for.

Dummy regressor

The Dummy Regressor is a kind of Regressor that gives prediction based on simple strategies without paying any attention to the input Data. As similar to Dummy Classifier the sklearn library also provides Dummy Regressor which is used to set up a baseline for comparing other existing Regressor namely Poisson Regressor, Linear Regression, Ridge Regression and many more. However, in this article, the main focus will be to draw a comparison between Dummy Regression and Linear regression.

STEP 1- Importing the necessary modules. The dummy module of sklearn provides an in-built DummyRegressor model which will be used in this case `from sklearn.dummy`

`import DummyRegressor` **STEP 2- Loading the Dataset. Here the Boston Dataset has been used for the purpose which is available in the sklearn

STEP 2- Loading the Dataset. Here the Boston Dataset has been used for the purpose which is available in the sklearn

STEP 3- Training and testing the dummy and the linear model.

Training the dummy model is similar to training any regular regression model, except for the strategies. The main role of strategy is to predict target values without any influence of the training data. There are namely four types of strategies that are used by the Dummy Regressor:-

- Mean: This is the default strategy used by the Dummy Regressor. It always predicts the mean of the training target values.
- Median: This is used to predict the median of the training target values.
- Quantile: It is used to predict a particular quantile of training target values provided the quantile parameter is used along with it.
- Constant: This is generally used to predict a specific custom value that is provided and the constant parameter must be mentioned.

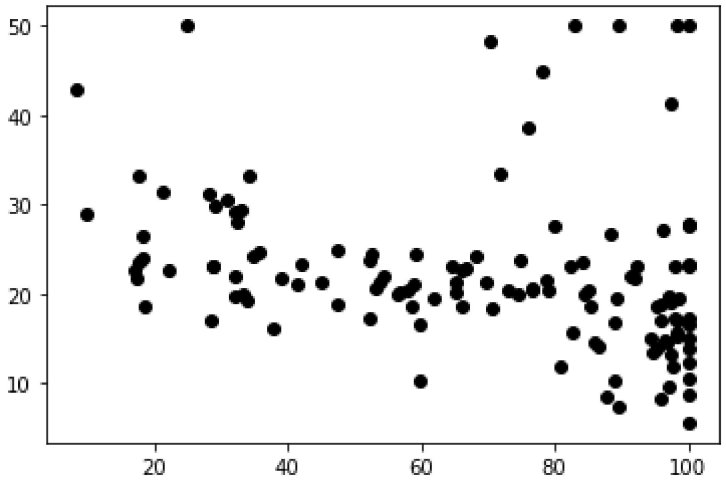
In [53]:
```
from sklearn.dummy import DummyRegressor
from sklearn.model_selection import train_test_split
from sklearn import datasets
import matplotlib.pyplot as plt
boston=datasets.load_boston()
X=boston.data[:, None, 6]
y= boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
print(x)
print(y)
```

```
[1 1 1 1 5]
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.5
 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.8
 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.1
 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.1
 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.3
 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.2
  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.
 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.4
```

```
16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.3
11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.7
19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3
16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.
 8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9
22.  11.9]
```

In [19]:
```
plt.scatter(X_test, y_test,  color='black')
```

Out[19]: <matplotlib.collections.PathCollection at 0x248d5761340>



dummy classifier

A dummy classifier is a type of classifier which does not generate any insight about the data and classifies the given data using only simple rules. The classifier's behavior is completely independent of the training data as the trends in the training data are completely ignored and instead uses one of the strategies to predict the class label. It is used only as a simple baseline for the other classifiers i.e. any other classifier is expected to perform better on the given dataset. It is especially useful for datasets where are sure of a class imbalance

Below are a few strategies used by the dummy classifier to predict a class label –

- 1)Most Frequent: The classifier always predicts the most frequent class label in the training data.
- 2)Stratified: It generates predictions by respecting the class distribution of the training data. It is different from the "most frequent" strategy as it instead associates a probability with each data point of being the most frequent class label.
- 4)Uniform: It generates predictions uniformly at random.
- 5)Constant: The classifier always predicts a constant label and is primarily used when classifying non-majority class labels.

In [23]:
```
from sklearn.dummy import DummyClassifier # importing
```

(Q)Making use of Part 1d, explore the various options available under Part 1e.

In [38]:
```
import numpy as np
x=np.array([1,1,1,1,5])
y=np.array([-2,1,1,5,5])
```

In [39]:
```
r=DummyClassifier(strategy="most_frequent")
r.fit(x,y)
```

Out[39]:
```
DummyClassifier(strategy='most_frequent')
```

In [40]:
```
from sklearn.metrics import accuracy_score
accuracy_score(x,y)
```

Out[40]:
```
0.6
```

In [44]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(x,y)
```

Out[44]:
```
array([[0, 0, 0],
       [1, 2, 1],
       [0, 0, 1]], dtype=int64)
```

In [47]:
```
from sklearn.metrics import classification_report
print(classification_report(x,y))
```

```
              precision    recall  f1-score   support

          -2       0.00      0.00      0.00         0
           1       1.00      0.50      0.67         4
           5       0.50      1.00      0.67         1

    accuracy                           0.60         5
   macro avg       0.50      0.50      0.44         5
weighted avg       0.90      0.60      0.67         5

C:\Users\jagan\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in l
abels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in l
abels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\jagan\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in l
abels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Conclusion

Here i understand some basic Sklearn commands. and i tried to explore more about some of the modules and all from part A and Comming to the part B tried to answer about the modules. And i got some more ideas about sklearn.So basically sklearn 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 dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

References

- https://scikit-learn.org/stable/

- https://www.geeksforgeeks.org/machine-learning/
- https://www.tutorialspoint.com/machine_learning/index.html
- https://www.kite.com/python/docs/sklearn

In [ ]: