INTRODUCTION MACHINE LEARNING LIBRARIES PART 2 - SCIKIT-LEARN

Machine Learning lab 1B

Name : Kavitha.S
Register number : 21122033
Class : 2MSDS

LAB OVERVIEW:

OBJECTIVES:

To explore the various modules of the Scikit-learn.
To evaluate how Scikit_learn Functions behave under various parameter values.
To load and explore various datasets using certain functions.
To visually interpret any of my findings. To learn about the features and targets of the methods.

PROBLEM DEFINITION:

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.
Do the documentation properly and find out answers to the above-mentioned questions.
Justify your findings by making use of Pandas and MatplotLibrary.

APPROACH

Load Toy datsets from sklearn.module to perform various functions on it.
Import all the functions necessary to perform the basic commands given in Part 1.
Exploring the basic commands given in Part 1 and trying to answer the questions provided in Part 2

SECTIONS:

The sections for this lab are:

1.Lab Overview

2.Executing commands given in Part 1.

3.Answering the questions given in Part 2 with the help of part 1

4.Conclusion

5.References

Part 1 - a

Train_test_split:

The train_test_split function is for splitting a single dataset for two different purposes: training and testing. The testing subset is for building your model. The testing subset is for using the model on unknown data to evaluate the performance of the model.

sklearn.model_selection.train_test_split(*arrays, **options)

In [1]:
```
from sklearn import datasets, linear_model
```

In [2]:
```
wine = datasets.load_wine()
```

In [3]:
```
# input(features)
x = wine.data
```

In [4]:
```
# ouput
y = wine.target
```

In [5]:
```
print(x[:5])
print(x.shape)
print(y[:5])
print(y.shape)
```

```
[[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
  2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
 [1.320e+01 1.780e+00 2.140e+00 1.120e+01 1.000e+02 2.650e+00 2.760e+00
  2.600e-01 1.280e+00 4.380e+00 1.050e+00 3.400e+00 1.050e+03]
 [1.316e+01 2.360e+00 2.670e+00 1.860e+01 1.010e+02 2.800e+00 3.240e+00
  3.000e-01 2.810e+00 5.680e+00 1.030e+00 3.170e+00 1.185e+03]
 [1.437e+01 1.950e+00 2.500e+00 1.680e+01 1.130e+02 3.850e+00 3.490e+00
  2.400e-01 2.180e+00 7.800e+00 8.600e-01 3.450e+00 1.480e+03]
 [1.324e+01 2.590e+00 2.870e+00 2.100e+01 1.180e+02 2.800e+00 2.690e+00
  3.900e-01 1.820e+00 4.320e+00 1.040e+00 2.930e+00 7.350e+02]]
(178, 13)
[0 0 0 0 0]
(178,)
```

In [6]:

```
from sklearn.model_selection import train_test_split
```

In [7]:

```
# performing the split
x_train, x_test, y_train, y_test = train_test_split(x,y)
```

In [8]:

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(133, 13)
(45, 13)
(133,)
(45,)
```

75% - Training data

25% - Test data

Part 2 - Question 3:

There are two approaches - Either by setting the test size or setting random state as 0

We can also specify the split size manually by specifying the parameter of "test_size"

We are setting the test data size to 0.1, that is

10% - Test data

90% - Training data

In [9]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.1)
```

When we run the split function again and again, a new set of split data is created each time.

In [10]:

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(160, 13)
(18, 13)
(160,)
(18,)
```

Finding the mean of the train and test dataset

In [11]:

```
import numpy as np
```

In [12]:

```
print(np.mean(x_train))
print(np.mean(x_test))
print(np.mean(y_train))
print(np.mean(y_test))
```

```
68.18606538413461
77.55675213675214
```

```
0.95625
0.7777777777777778
```

To get the same set of split everytime, we have to use specify the random state instance with the use of the parameter random_state.

In [13]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.1, random_state=1)
```

In [14]:
```
X, y = np.arange(10).reshape((5, 2)), range(5)
X
```

Out[14]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

In [15]:
```
list(y)
```

Out[15]:
```
[0, 1, 2, 3, 4]
```

Part 2- Question 2:

We are changing the random_state value here. So as the parameter is changed, the split size is getting changed.

In [101…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=
```

In [17]:
```
X_train
```

Out[17]:
```
array([[4, 5],
       [0, 1],
       [6, 7]])
```

In [18]:
```
y_train
```

Out[18]:
```
[2, 0, 3]
```

In [19]:
```
X_test
```

Out[19]:
```
array([[2, 3],
       [8, 9]])
```

In [20]:
```
y_test
```

Out[20]:
```
[1, 4]
```

In [21]:
```
train_test_split(y,shuffle=False)
```

Out[21]:
```
[[0, 1, 2], [3, 4]]
```

Part 2 - Q1

Parameter (train_test_split):

*arrays : sequence of arrays or scipy.sparse matrices with same shape[0]
test_size : float, int, or None (default is None)
train_size : float, int, or None (default is None)
random_state : int or RandomState
shuffle
stratify

It is one of the most popular Machine learning library. Using it we can write any Machine learning code in a couple of lines.

Part 1 - b

Make_classification:

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 [22]:
```
from sklearn.datasets import make_classification
import pandas as pd
```

In [23]:
```
X,y = make_classification(n_samples = 500, n_features = 6)
```

Part 1 - Question 4

Feature

In [24]:
```
X
```

Out[24]:
```
array([[ 1.2532371 , -0.93720378,  0.5628373 ,  0.54399037, -1.24874463,
        -0.02270613],
       [ 0.8095222 , -1.29465283, -0.2546249 ,  1.40886022, -0.6511929 ,
         0.49277839],
       [-1.24627567,  1.14966662, -1.65933466, -0.87491392,  1.19272484,
        -1.94602468],
       ...,
       [-1.16718993, -0.04848762, -0.21174868,  0.90687536,  1.37076458,
        -1.77757278],
       [ 0.04745106, -0.54543081, -0.46091015,  0.80295057,  0.06770946,
        -0.34401129],
       [-1.35796154,  0.90191144, -0.35055692, -0.4151516 ,  1.37871176,
        -0.45718706]])
```

Target

In [25]:
```
y
```

Out[25]:
```
array([0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1])
```

In [26]:
```
def make_classification_df(n_samples: int = 1024,
                           n_num_features: int = 20,
                           n_cat_features: int = 0,
                           class_sep: float = 1.0,
                           n_classes: int = 2,
                           feature_name: str = 'col_{}',
                           target_name: str = 'target',
                           random_state: int = 0,
                           id_column: str = None):
    np.random.seed(random_state)
    X, y = make_classification(n_samples=n_samples, n_features=n_num_features, class_se
                               random_state=random_state, n_classes=n_classes, n_inform

    X = pd.DataFrame(X, columns=[feature_name.format(i) for i in range(n_num_features)]
    y = pd.Series(y, name=target_name)

    if id_column is not None:
        X[id_column] = range(n_samples)

    for i in range(n_cat_features):
        X['cat_{}'.format(i)] = \
            pd.Series(np.random.choice(['A', 'B', None], size=n_samples)).astype('categ

    return X, y
make_classification_df()
```

Out[26]:
```
(         col_0     col_1     col_2     col_3     col_4     col_5     col_6  \
 0     1.914264  0.640563  0.233148  0.160922  0.715963  0.900157  1.305518
 1    -0.910303  0.468672 -0.404183 -0.127438  0.121353  0.365365  0.936349
 2    -0.034210  2.258895  0.619382 -1.203291  2.717515  2.208683 -0.743103
 3    -0.683089  3.308392 -0.122563  0.632212 -1.173905  1.979877  0.938855
 4    -0.588780  0.165048 -1.105255 -1.603076 -1.128883  0.917366 -0.048229
 ...        ...       ...       ...       ...       ...       ...       ...
 1019  2.270059 -0.708671 -1.035967  0.230788  0.886659 -1.710860 -0.643594
 1020 -0.923201 -1.296979 -1.456516  0.622597 -0.929357  0.821772 -1.486823
```

```
1021  0.170209 -1.167598 -0.386775 -0.990763  1.289432 -0.407231  0.939885
1022 -0.213236  1.028228 -0.255860  0.306318 -0.153791  1.000186  2.499514
1023  0.615972 -1.928527 -0.216477 -1.685129  1.382400  0.877991 -1.024049

         col_7     col_8     col_9    col_10    col_11    col_12    col_13  \
0    -1.016623 -0.902633 -0.751206 -1.261875  0.906170  0.279409 -0.506621
1    -1.830244 -0.743203 -0.273587 -1.262602 -0.394977  0.137470  1.127205
2     0.521020 -1.666383 -0.094875 -0.500900 -0.639387  0.764807  1.490501
3    -0.417213 -0.376233  1.784998 -0.754830 -2.270149  0.833568  1.181070
4     2.306931  0.388762 -1.461770 -0.025625 -0.217583  0.228511  0.101411
...        ...       ...       ...       ...       ...       ...       ...
1019  0.049040  0.003909 -1.182276 -0.594156 -1.461771 -0.472386 -0.664644
1020 -1.150932  1.411416  0.945274 -0.159247 -1.304802  0.038108  1.186234
1021 -0.189618 -0.858206  2.229974  1.409773 -1.300798 -0.227617  0.952467
1022  1.329745 -0.426604  0.146850 -1.394385  0.805601  0.346949 -0.814259
1023 -0.597175  0.906340 -0.774971  1.358871  1.743978 -0.021873 -1.513432

        col_14    col_15    col_16    col_17    col_18    col_19
0    -1.514168  0.341040 -0.322267 -0.358241  0.356703 -0.459777
1    -0.587455 -1.440550 -0.128974  1.704433 -0.331964 -0.348640
2    -0.992565  1.380337 -0.784706 -0.100985 -0.570849  1.013357
3    -2.319832 -0.673204 -0.692155 -1.236327  0.581651 -0.951222
4     0.188426  0.712606 -0.332707  0.760493 -0.764560  0.282604
...        ...       ...       ...       ...       ...       ...
1019  0.943393 -0.280315  0.616971  0.186374  0.042394 -1.309774
1020  1.605774 -0.503980 -0.310712  1.288707  0.560312 -0.649512
1021 -0.899628 -1.238775  0.138093  0.532162 -1.578459  0.256726
1022  1.014654  0.042998 -0.355302  0.089994 -0.204119  0.221410
1023 -1.335559 -0.293818 -0.336730  0.439979 -0.156315 -1.660219

[1024 rows x 20 columns],
0       1
1       1
2       1
3       1
4       1
       ..
1019    0
1020    1
1021    0
1022    1
1023    1
Name: target, Length: 1024, dtype: int32)
```

Part 2 - Question 1

Parameter (make_classification):

n_samples : int, optional (default=100)

n_features : int, optional (default=20)

n_informative : int, optional (default=2)

n_redundant : int, optional (default=2)

n_repeated : int, optional (default=0)

n_classes : int, optional (default=2)

n_clusters_per_class : int, optional (default=2)

weights : list of floats or None (default=None)

flip_y : float, optional (default=0.01)

class_sep : float, optional (default=1.0)

hypercube : boolean, optional (default=True) shift : float, array of shape [n_features] or None, optional (default=0.0)

scale : float, array of shape [n_features] or None, optional (default=1.0)

shuffle : boolean, optional (default=True)

random_state : int, RandomState instance or None, optional (default=None)

Part 1 - c

Make Regression

It is a function in sklearn_datasets which is used to generate our dataset for the regression problem.

Part 2 - Question 1

Parameter (make_regression):

n_samples : int, optional (default=100)

n_features : int, optional (default=100)

n_informative : int, optional (default=10)

n_targets : int, optional (default=1)

bias : float, optional (default=0.0)

effective_rank : int or None, optional (default=None)

tail_strength : float between 0.0 and 1.0, optional (default=0.5)

noise : float, optional (default=0.0)

shuffle : boolean, optional (default=True)

coef : boolean, optional (default=False)

random_state : int, RandomState instance or None (default)

In [27]:
```
from sklearn.datasets import make_regression
```

In [29]:
```
from matplotlib import pyplot as plt
```

Part 2 - Question 4:

Features :

In [121…
```
x = make_regression(n_samples = 150, n_features = 2,noise = 0.2)
x[0]
```

Out[121…
```
array([-111.88472235,  -13.69939682,  -77.72803018,   63.71019135,
        -50.64128668,   10.60056532,   -2.96466943,   75.9851209 ,
          8.47459549,   29.49061755,    6.16795349,  -18.35775287,
         35.85980496,   63.61304542,  -55.67909168,   30.80090128,
         46.99825115,   -0.64614159,    1.86403661,   27.05951254,
         22.72496622,   55.73228202,   62.25761832,   -6.02734221,
        -17.38075734,   35.01394593,   40.55034569,  -34.61847684,
        -50.46917492,   -0.78081729,  -18.46402062,   14.46972691,
        110.3516799 ,    5.20452804,    2.99419959,   36.38120559,
         -5.96562651,  -37.26936149,   16.83886992,   42.76953853,
         17.77871869,  -71.66052833,  -31.57441851,  103.37799605,
```

```
        -10.90575495,  -29.89843446,   21.92379347,  -52.42763692,
        -49.4002047 ,  -75.799739  ,  -20.23604975,  -35.27765354,
         94.77831617,   11.53377449,  -37.30549253,    9.14992549,
        -13.72432056,  -20.14064229,   13.56271162,   -5.28548112,
        109.36173286,   36.86755237,   24.16395148,  -27.3517445 ,
        -35.58038677,  -14.12479494,  -33.53459016,   56.25319936,
        -34.19179021,    1.25106664,  -16.11434567,   27.68239756,
         40.55928042, -107.24669707,   -4.82633606,    2.65495039,
         28.45670883,  -17.33285231,   49.41495068,   50.56988028,
        -50.37507512,   26.60617654,   -7.33822709,   12.77218777,
        -34.31011612,    4.69144407,  -14.84980435,   19.81700979,
        -52.2157119 ,   35.27987692,   54.85996966,  -43.84115876,
        -35.76833503,  -19.51726977,   57.66147355,   48.48920688,
        -39.1436013 ,  -43.91587996,  -35.76518648, -101.24553161,
        -14.73671152,  -23.22989159,   72.47354761,   -6.94746254,
        -25.62891236,    7.35650889,  -45.56256226,  -11.03502266,
         42.85596724,   -2.15269515,   87.08132223,  -60.77684706,
        -46.36083759,  119.49145753,  -77.06545453,  -60.80587005,
        -83.50436507,   23.96738696,   27.72912753,   32.76739056,
         44.41587512,  -27.56457498,  -10.72052554, -105.60692297,
          8.71613123,  -14.77000144,    2.17289866,  -12.6847859 ,
        -81.09335779,    4.60418658,   23.42971151, -108.79035205,
        -31.85485159,    8.53127529,  -20.70771697,   87.61092899,
         74.71006746,   -7.64842481,  -61.45152424,   73.42102344,
         62.30500456,   21.3819465 ,   53.28582482,   64.28667212,
         63.87490285,  -30.56998668,  -33.00639898,   46.92795523,
        -24.60995596,  -12.19012364])
```

Target:

In [123…

```
x = make_regression(n_samples = 150, n_features = 2,noise = 0.2)
x[1]
```

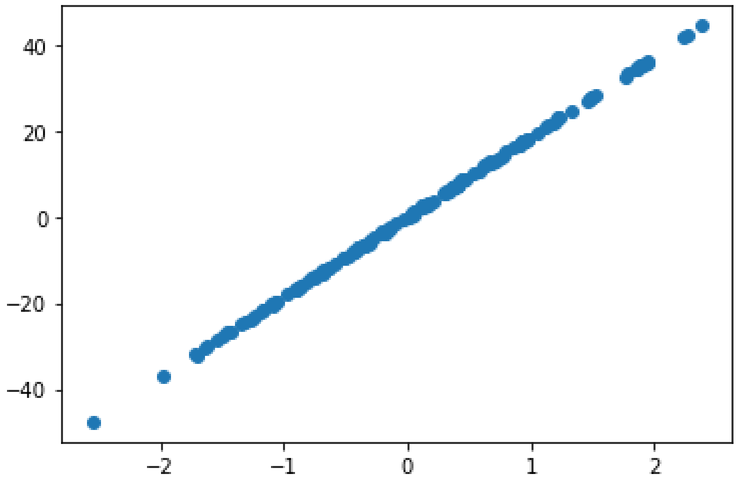
Out[123…

```
array([ -71.57716935, -128.76365262,  230.67229949, -104.79316044,
        -16.10543613,  -77.73921804,  189.92630093, -291.27057151,
        -12.9343821 ,  181.06475303,  -85.27463166,   62.51875215,
        -54.90910447,  -28.7027691 ,  -41.32441087,  102.63411741,
         -4.58504736,  221.35594796,  100.14430198,  -25.66359543,
        -53.1181356 , -160.34696597,  -93.0201453 , -105.65536026,
       -150.43044235,  140.47816334, -105.8266687 , -165.03127531,
        210.99209072,   89.41904709,  -32.22599918,  -49.34226858,
         -9.77500052,  235.4554689 , -120.41263826,  -99.56774833,
        -64.41155391,   18.10420839, -156.95576433,  169.56395071,
        -33.93878637,  -97.51580009,  161.6443916 , -213.31163647,
        -19.69551568,   41.57551154,   10.30614472,  -92.0724496 ,
        303.70676017,   49.37059851,    8.70901456,   92.00213129,
         -8.31327445,  -95.38627176, -374.29187944,  -20.83463224,
        -96.58984186,  125.66329247,  -42.54868638,  -65.3159553 ,
       -266.72036548,  -41.87846009,  -53.74015059, -105.90583618,
         -2.08398288,  -17.68866267,   18.7496694 ,   -7.82131626,
        -22.12522539, -121.61182105,  217.28562659,    2.56940645,
        -75.12130344, -247.7814983 ,  -72.35989261,  132.63774236,
         56.53502682,  -31.82218903,  -97.73592456,  -34.30507308,
         19.38319902,   78.7157953 , -355.94858189,   57.6538585 ,
        137.94770206,   68.19871691,  111.01640805,   17.11412225,
        117.62862401,  239.07290176,   11.43518451,   52.11483106,
        119.6722551 ,  195.16247636,  164.73826857,   14.62139381,
         97.85712837,    6.42052438,  109.78775089,   14.86495082,
        121.26184544, -238.80261058,  -84.63741184,   89.74293592,
       -185.8628243 , -114.39456672,   23.26514279, -237.39443551,
       -216.51181808,   61.8782885 ,  164.854787  , -103.19885869,
        174.50713018, -180.28751098,   69.80984054,  -21.51942196,
        -40.55697929, -255.208447  ,   71.77496961,   86.89391266,
```

```
      -116.23082563, -146.27115236,  -96.29700693,  -28.28203981,
        21.0093062 ,    7.98452179,  -22.83498412,   14.05286038,
       214.93059763,  262.73581343,   11.49713921,   93.50603583,
       158.26516355,    7.91107628, -124.05211841,  -13.97713788,
      -110.46687924,   57.72281466,  172.2851798 ,  -57.2687673 ,
       168.81105118,  170.40015335,    8.51285632, -185.26486972,
       -96.25742429,   49.89181493,   93.30828313, -215.92148484,
       -87.75114299,  -51.37247733])
```

In [30]:

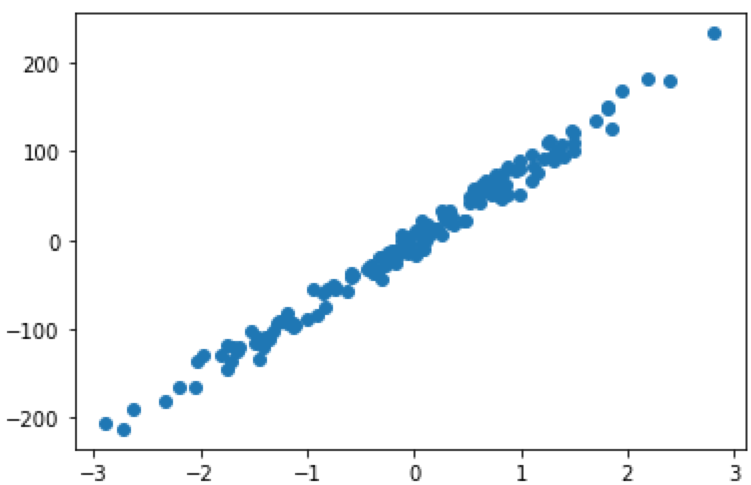
```
X_test, y_test = make_regression(n_samples = 150, n_features = 1,noise = 0.2)
plt.scatter(X_test, y_test)
plt.show()
```



Initially, we imported the required library. After that, we made a dataset using make_regression(). Then using matplotlib we plotted our plot as you can see in the above figure. Further, one thing to notice here is that we have made a dataset of 150 entries and with noise 0. However, we can change these values as I have demonstrated below.

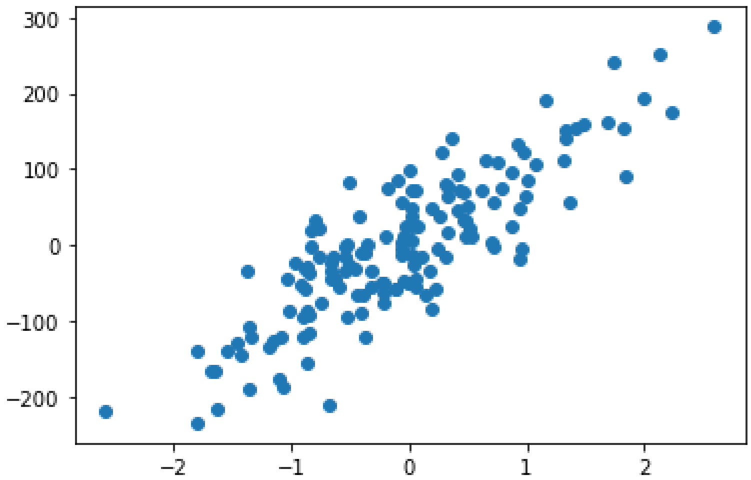
In [31]:

```
X_test, y_test = make_regression(n_samples = 150, n_features = 1,noise = 10)
plt.scatter(X_test, y_test)
plt.show()
```



In [32]:

```
X_test, y_test = make_regression(n_samples = 150, n_features = 1,noise = 50)
plt.scatter(X_test, y_test)
plt.show()
```



Part 2 - Question 2:

In the above two figures, you can observe that as the noise increased from 10 to 50 our dataset become more scattered. Noise is nothing but the parameter of make_regression. So when we change a parameter in any function/method, the output get changed accordingly

In [33]:

```
def make_regression_with_outliers(n_samples=50, n_features=20):
    rng = np.random.RandomState(0)
# Generate data with outliers by replacing 10% of the samples with noise.
    X, y = make_regression(
        n_samples=n_samples, n_features=n_features,
        random_state=0, noise=0.05)

# Replace 10% of the sample with noise.
    num_noise = int(0.1 * n_samples)
    random_samples = rng.randint(0, n_samples, num_noise)
    X[random_samples, :] = 2.0 * rng.normal(0, 1, (num_noise, X.shape[1]))
    return X, y
make_regression_with_outliers()
```

Out[33]:

```
(array([[ 1.22991463e+00,  2.85932153e+00, -4.23904513e-01,
         -1.60674512e-01,  8.10795568e-01,  2.37213170e-01,
          2.50882814e+00,  2.83820408e+00, -1.48771217e+00,
         -5.03487421e+00, -3.01419205e+00,  2.29815225e+00,
         -2.38715650e+00,  2.28208489e+00,  3.01889016e+00,
          2.13555026e+00, -1.37317895e+00,  2.97466321e-02,
         -7.51331792e-01, -7.64472749e-02],
        [-4.24663302e-01, -7.57870860e-01,  1.48935596e+00,
          1.31247037e+00, -5.00840943e-02, -1.34149673e+00,
         -1.35978073e+00,  4.22628622e-01,  6.11927193e-01,
         -8.97400927e-01, -1.07709907e+00, -8.58972388e-01,
          7.45864065e-02, -4.14008116e-02,  5.21303748e-01,
         -8.98942156e-01,  5.29045238e-01,  1.48449581e-01,
          4.76898369e-01, -8.29964598e-01],
        [ 8.20321797e-01, -6.87299037e-01,  6.76460732e-01,
         -3.70242441e-01, -1.17474546e-01, -3.02249730e-01,
         -5.61330204e-02,  1.67094303e+00, -2.24258934e-01,
          4.66166426e-01,  2.52496627e-01, -4.53804041e-01,
         -9.18004770e-01, -1.38504274e-03, -3.82008956e-01,
          4.03264540e-01,  1.83339199e-01, -1.22619619e+00,
         -3.75147117e-01,  1.35994854e+00],
        [-2.10151591e+00,  2.02415811e+00,  3.08743286e+00,
```

```
        -8.04229778e-01,  1.72949820e+00, -4.53738454e-01,
         1.62232054e+00,  5.75589226e-01, -1.07693632e+00,
         3.81692420e+00, -9.51552002e-01,  2.88146224e+00,
        -2.29815136e+00,  1.61142910e+00,  3.51221406e+00,
         1.94595983e+00, -3.10223822e+00,  3.82286956e-01,
         3.20955214e+00, -2.84005186e+00],
       [-2.03068447e+00,  3.99046346e-01, -1.18885926e+00,
         3.90093323e-01, -2.77259276e+00, -5.25672963e-02,
        -3.10886172e-01,  8.84220870e-02, -5.96314038e-01,
         1.95591231e+00, -1.16103939e-01, -6.52408582e-01,
         4.93741777e-01,  9.74001663e-02, -5.06816354e-01,
        -3.90953375e-01,  5.23891024e-01,  1.88778597e-01,
        -1.93627981e+00,  2.06449286e+00],
       [ 1.97967290e-01,  3.30576756e-01, -5.91402668e-01,
        -1.77766695e+00,  9.49246474e-01,  8.67407411e-01,
        -3.57680719e-02, -1.61087840e+00,  7.55395696e-01,
        -1.50239657e+00, -7.94636321e-01, -5.32702792e-01,
        -3.46249448e-01,  2.38074535e+00,  1.12441918e+00,
         1.09074973e+00,  2.11679102e+00, -2.83455451e+00,
        -6.56463675e-01,  1.08193522e+00],
       [ 1.02179059e+00, -7.04921353e-01, -1.44494020e+00,
        -2.90397101e-01,  6.79974844e-01,  1.09463837e+00,
         1.26507784e+00, -3.50951769e-02, -7.88669255e-01,
        -6.96326654e-01, -4.64337691e-01,  1.32778270e+00,
        -8.03141387e-01,  2.11497013e-01, -1.21054299e+00,
        -1.01281486e-01,  9.36445726e-01,  2.13215341e+00,
         2.34821526e-01, -5.52540673e-01],
       [-6.16264021e-02, -1.33701560e-01, -1.30652685e+00,
        -7.30677753e-01,  1.07774381e+00, -6.80178204e-01,
         6.93773153e-01, -1.34671751e+00, -1.18164045e-01,
        -1.12682581e+00, -2.86887192e-01, -3.84879809e-01,
        -4.21714513e-02, -1.59573438e-01,  1.65813068e+00,
         9.43515893e-02, -1.33425847e+00, -4.60719787e-01,
         6.66383082e-01, -1.07305276e-01],
       [-5.35270165e-01, -2.58279663e+00,  2.39582760e-01,
        -1.35338886e+00, -1.15395036e+00,  2.13386825e+00,
        -7.49690345e-01, -5.39132637e-01,  9.72535789e-01,
        -3.47961856e-01, -4.06071796e-01, -1.03264310e+00,
        -1.64296529e+00,  3.28087476e-02, -3.69801166e-01,
        -4.36748337e-01,  7.55740289e-01, -1.93176702e-01,
         4.06415494e-01,  2.54052084e-02],
       [ 4.70433145e-01,  3.24869616e-01, -1.11456118e-02,
        -6.96415784e-02,  9.97117981e-01, -5.91183104e-01,
         1.37098901e+00,  2.22594433e+00, -8.37678042e-01,
         3.06018243e-02, -3.25669469e-01,  5.15749428e-02,
        -8.48320523e-01, -5.09843242e-01,  1.14988999e-02,
         8.67276629e-01,  3.30035115e-01,  3.26962595e-01,
        -6.67720286e-01,  3.11447072e-01],
       [-3.70704003e-01,  2.86904488e-01, -9.03820073e-02,
         5.20040615e-01, -2.32059428e+00, -9.96212640e-01,
        -6.00657558e-01, -7.22870076e-02,  1.03440989e+00,
         3.17160626e-01, -1.31839587e+00,  2.25608654e-01,
        -6.72756089e-02,  1.55224318e+00,  1.36759724e+00,
         4.49712100e-01,  1.02893549e+00, -3.04963638e-01,
        -1.21793851e+00, -9.45615796e-01],
       [ 1.46657872e+00,  5.67290278e-01, -1.69810582e+00,
        -1.61647419e+00, -2.22675101e-01, -1.02250684e+00,
         1.64813493e+00, -1.47183501e+00, -2.25556423e+00,
        -3.53431749e-01,  1.14110187e+00, -2.91837363e-01,
         8.57923924e-01,  1.64227755e-01,  3.87280475e-01,
        -7.61492212e-01, -9.85510738e-01, -1.65671510e+00,
         3.86305518e-02,  8.52551939e-01],
       [-1.14761094e+00,  4.41032907e-01,  3.59504400e-01,
         2.40787510e-01,  1.78792866e-01,  1.06458514e+00,
         1.38526155e+00,  7.24368505e-01, -3.61599281e-01,
```

```
 -7.99422400e-01,  9.41923003e-02,  2.89120505e-01,
 -1.98398897e-01, -3.03098253e-01, -1.44566817e-01,
  4.12870820e-01, -4.05941727e-01,  4.33107953e-01,
 -9.37880231e-01, -3.58114075e-01],
[-3.87326817e-01,  1.54947426e-01, -2.55298982e+00,
 -1.98079647e+00,  3.78162520e-01, -7.42165020e-01,
  1.53277921e+00, -1.87183850e-01,  8.64436199e-01,
 -8.87785748e-01,  1.20237985e+00, -3.47912149e-01,
  1.23029068e+00,  1.46935877e+00,  6.53618595e-01,
  1.56348969e-01,  4.57585173e-02, -1.45436567e+00,
  2.26975462e+00, -3.02302751e-01],
[-1.43579512e-01, -6.57926093e-01, -3.65551090e-01,
  1.38914532e+00,  9.68882639e-01,  8.29986159e-01,
  6.90429024e-01,  1.56925961e+00,  2.96733172e-01,
  2.25581664e-01, -8.64044991e-01,  2.01406015e+00,
 -4.06303130e-01,  7.96672108e-01,  9.38092541e-01,
 -3.06765776e-01,  1.22319836e-02, -7.48049827e-02,
 -4.96102334e-01, -3.82025449e-01],
[ 3.97667346e-02,  7.72694837e-01, -1.17762896e+00,
  1.69618157e+00, -1.66159829e+00, -1.32988422e-01,
 -1.82425666e+00,  7.20033759e-01,  1.75498615e+00,
  4.48195284e-01, -7.07505698e-01, -1.48577034e-02,
  6.70570450e-01,  3.03603904e-01, -1.14019630e+00,
  8.21405937e-01,  1.03493146e-02,  5.55786964e-01,
 -7.65702194e-01, -1.56699471e+00],
[-8.52585847e-01, -5.32489919e-01,  1.41117206e+00,
 -6.57951045e-01,  6.45055273e-01, -3.91217052e-01,
 -1.68823003e+00, -2.61922373e-02, -5.74695185e-02,
  1.01184243e+00,  1.68192174e+00,  4.68385234e-01,
 -6.67712721e-01, -1.12465983e-01,  7.85803827e-01,
  1.73587900e+00,  4.98052405e-01,  4.05204080e-01,
  9.40917615e-01,  2.29597556e-02],
[-4.79655814e-01, -7.64143924e-01,  1.06850940e+00,
 -6.89449185e-01, -1.43779147e+00, -1.21407740e+00,
  5.78521498e-01,  1.56703855e-01, -6.87837611e-01,
  1.36453185e+00, -4.77974004e-01, -6.52293600e-01,
 -1.84306955e+00,  3.49654457e-01, -4.53385804e-01,
 -5.21189312e-01, -3.64693544e-01, -2.80355495e-01,
 -4.40922632e-01,  6.20358298e-01],
[ 1.28598401e+00, -1.74235620e+00, -1.55042935e+00,
  8.95555986e-01, -1.30324275e+00,  2.38103148e-01,
  1.15147873e-01, -1.66069981e+00, -9.44368491e-01,
  6.05120084e-01,  3.29622982e-01, -1.31908640e-01,
  2.23843563e-01, -3.79147563e-01,  4.17318821e-01,
  4.04761812e-01, -1.10489405e-01, -5.90057646e-01,
 -1.40596292e+00, -1.50699840e+00],
[-6.34322094e-01, -8.95466561e-01, -1.04855297e+00,
 -1.18063218e+00,  3.86902498e-01,  1.95077540e+00,
 -1.61389785e+00,  7.77490356e-01, -1.70627019e+00,
 -5.10805138e-01,  3.02471898e-01, -2.81822283e-02,
  6.65172224e-02, -2.12740280e-01, -1.42001794e+00,
  4.28331871e-01, -1.25279536e+00, -4.38074302e-01,
 -5.09652182e-01, -3.62741166e-01],
[-1.58293840e+00,  5.21064876e-01, -6.37437026e-01,
 -3.19328417e-01, -5.75787970e-01, -2.97790879e-01,
 -8.13364259e-01,  1.07961859e+00, -1.32880578e-01,
  1.41953163e-01, -1.38336396e+00,  6.91538751e-01,
 -7.25597378e-01, -1.46642433e+00, -3.97271814e-01,
  6.94749144e-01,  1.15233156e+00, -1.67600381e+00,
 -3.09012969e-01,  6.10379379e-01],
[ 1.99300197e-01, -1.05462846e+00, -3.95228983e-01,
  2.79095764e-01,  8.20247837e-01,  1.94292938e-01,
 -7.82629156e-01, -9.64612014e-01, -8.59307670e-02,
  4.63130329e-01, -2.20144129e+00,  3.38904125e-01,
 -4.68864188e-01, -1.10389299e-01, -1.15942052e+00,
```

```
 2.02104356e+00,  4.57415606e-01, -1.15107468e-01,
 8.75832762e-01, -5.06035410e-02],
[ 4.26258731e-01, -1.42406091e+00, -9.55945000e-01,
 4.16050046e-01, -4.93319883e-01,  4.81481474e-01,
-5.97316069e-01,  2.32181036e-01, -4.63595975e-01,
-5.42861476e-01, -2.06998503e+00, -1.15618243e+00,
 1.49448454e+00, -2.37921730e-01, -3.45981776e-01,
 7.81198102e-01,  1.56506538e-01,  6.32619942e-02,
-1.54079701e+00,  6.76908035e-01],
[-6.84010898e-01,  7.47188334e-01, -7.19604389e-01,
-1.18388064e+00, -1.18894496e+00, -8.90915083e-01,
-7.04700276e-01,  2.25672350e+00,  2.74516358e-01,
 7.73252977e-01,  4.50934462e-01, -2.65917224e+00,
-1.75589058e+00,  9.43260725e-01, -8.12992989e-01,
 6.06319524e-01, -1.57667016e-01, -3.12292251e-01,
-1.15735526e+00,  1.65955080e+00],
[-1.14746865e+00, -6.82416053e-02, -1.49125759e+00,
-8.26438539e-01,  1.71334272e+00,  6.35031437e-01,
-1.31590741e+00,  1.11701629e+00,  1.66673495e-01,
-7.44754822e-01, -1.07993151e+00, -9.84525244e-02,
 1.12663592e+00, -4.61584605e-01,  4.39391701e-01,
-6.63478286e-01, -9.12822225e-01,  9.44479487e-01,
 2.38314477e+00, -4.37820045e-01],
[ 6.14079370e-01,  1.86755896e+00,  1.88315070e+00,
 1.91006495e+00,  9.06044658e-01,  9.69396708e-01,
 1.92294203e+00, -7.47454811e-01, -1.27048500e+00,
-8.61225685e-01, -1.55010093e-01, -2.68003371e-01,
 9.47251968e-01,  1.48051479e+00, -1.34775906e+00,
 8.02456396e-01, -4.13618981e-01,  1.94362119e+00,
-1.17312341e+00,  9.22206672e-01],
[ 2.59442459e+00, -6.47181432e-01,  1.32646164e+00,
-1.75316402e-01,  4.72247150e-01, -2.12523045e-01,
 2.71170185e-01, -5.25640593e-01,  5.98946831e-02,
 9.30408496e-01, -1.54158740e+00, -1.42191987e+00,
-8.56549308e-01, -8.01496885e-01, -9.64606424e-01,
 1.99795608e+00,  9.36398544e-01, -8.87780137e-01,
-7.62114512e-01, -4.04032294e-01],
[-1.63242330e+00, -4.99016638e-01, -4.51303037e-01,
 1.92753849e-01,  2.13512238e-02,  2.46121252e-02,
 1.84959125e+00,  1.92793845e-02,  7.23100494e-01,
-9.19113445e-01, -3.17543094e-01, -3.65055217e-01,
-5.85865511e-02, -2.14166656e-01,  2.65687975e-01,
-1.79132755e+00, -1.01697275e-01, -1.10290621e+00,
 7.19983730e-01, -6.71341546e-02],
[-3.99449029e-01, -6.28087560e-01, -1.10540657e-01,
-1.06001582e+00, -4.81027118e-01,  1.53637705e+00,
 6.89818165e-01,  1.21114529e+00, -6.92049848e-01,
 2.30391670e+00,  5.82953680e-01, -1.35949701e-01,
 9.77249677e-02,  1.30184623e+00,  1.02017271e+00,
 1.13689136e+00, -1.04525337e+00,  6.08843834e-01,
 2.86343689e-01,  3.70055888e-01],
[ 3.13067702e-01,  1.44043571e-01,  1.76405235e+00,
 1.21675016e-01,  1.45427351e+00,  2.24089320e+00,
-1.03218852e-01, -1.51357208e-01,  9.78737984e-01,
 7.61037725e-01, -2.05158264e-01,  4.43863233e-01,
 1.49407907e+00,  4.10598502e-01,  4.00157208e-01,
 3.33674327e-01,  9.50088418e-01, -9.77277880e-01,
 1.86755799e+00, -8.54095739e-01],
[ 2.84279671e-01,  8.95260273e-01,  3.82732430e-01,
-1.96862469e+00,  1.37496407e+00, -2.34215801e-01,
-1.17915793e+00, -1.56776772e+00,  1.09634685e+00,
-1.33221165e+00,  1.04797216e+00, -6.60056320e-01,
 4.98690275e-01,  1.30142807e+00, -3.42422805e-02,
 1.75818953e-01, -1.63263453e+00, -5.81268477e-01,
-3.47450652e-01,  1.74266878e+00],
```

```
[-2.02896841e-01, -2.23960406e+00, -2.22605681e-01,
  6.48561063e-02,  4.01499055e-01, -8.88971358e-01,
 -2.36958691e+00,  1.41232771e+00, -1.68121822e+00,
  1.22487056e+00, -1.82244784e-01, -1.27968917e+00,
 -2.61645446e-01,  8.64052300e-01, -9.13079218e-01,
 -5.85431204e-01,  9.36742464e-01, -8.88720257e-01,
  2.42117961e-01, -1.09882779e-01],
[-1.34792542e+00,  1.63928572e-01,  2.13480049e-01,
 -2.67594746e-01,  9.63213559e-02,  1.51826117e+00,
  1.18137860e+00, -2.55918467e+00, -2.42019830e-01,
  9.42468119e-01,  2.03341817e-02, -6.78025782e-01,
 -2.36417382e+00, -6.31903758e-01, -1.20857365e+00,
  1.29784579e+00,  1.07819730e+00, -4.43836093e-01,
 -3.84645423e-01, -7.61573388e-01],
[-1.04343491e-01, -5.05358317e-01, -3.86870847e-01,
 -1.05188010e+00, -8.15791542e-01, -3.85489760e-01,
  8.12674042e-01,  1.24331938e+00,  1.83925494e-01,
 -5.07517602e-01, -1.28455230e+00,  2.49720039e+00,
  5.64008535e-01,  5.87259379e-01, -5.10292740e-01,
 -2.24532165e+00, -9.32789042e-01, -8.87180942e-01,
 -1.60183605e+00, -9.88001942e-01],
[ 1.51999486e+00, -1.44653470e+00, -5.98653937e-01,
 -2.33466662e-01,  8.00297949e-01,  3.56292817e-01,
 -1.85053671e-01,  5.89255892e-02,  7.66663182e-01,
 -3.09114445e-01,  1.42061805e-01,  1.73272119e+00,
  3.70825001e-01, -8.07648488e-01, -1.11589699e+00,
  6.84501107e-01,  8.14519822e-01,  3.55481793e-01,
 -1.76853845e+00,  1.71958931e+00],
[ 3.52816606e-01, -8.17493098e-01, -1.46173269e+00,
 -2.04732361e+00, -1.40134729e+00,  1.90311558e-01,
  9.60693398e-01, -1.18468659e+00,  3.67544896e-01,
  1.03043827e+00, -2.63937349e-01, -1.22662166e+00,
 -5.53525480e-02,  1.32906285e+00, -6.83439767e-01,
  9.67446150e-01, -5.21579678e-01,  1.82272360e+00,
 -8.51729197e-01, -1.52774424e-01],
[ 8.00564803e-01, -4.66845546e-01,  9.29505111e-01,
  2.76871906e-01, -1.41690611e+00,  1.23721914e-01,
 -5.69312053e-01, -2.73967717e+00, -2.09460307e+00,
  8.68963487e-01,  5.29264630e-03, -9.71104570e-01,
  8.21585712e-01,  2.69904355e-01,  5.82224591e-01,
  3.14817205e-01,  9.43046087e-01,  9.39532294e-02,
 -1.30106954e-01,  7.82601752e-02],
[-2.06903676e-01,  2.80441705e-01,  6.98457149e-01,
 -2.49458580e-01, -9.93123611e-01,  3.39964984e-01,
 -2.67733537e-01, -3.94849514e-01,  9.31848374e-01,
  8.41631264e-01, -1.57062341e+00,  4.94949817e-02,
  6.43314465e-01, -1.12801133e+00,  3.77088909e-03,
  4.93836776e-01, -1.90653494e-01,  1.60928168e-01,
 -1.56821116e-02,  8.80178912e-01],
[-3.11552532e-01,  7.29090562e-01, -6.72460448e-01,
 -1.23482582e+00,  1.28982911e-01, -1.72628260e+00,
 -9.07298364e-01,  4.62782256e-01, -8.13146282e-01,
  1.13940068e+00, -5.78849665e-01,  4.02341641e-01,
 -8.70797149e-01,  5.19453958e-02, -3.59553162e-01,
 -6.84810091e-01, -1.63019835e+00, -4.01780936e-01,
  1.77426142e-01,  5.61653422e-02],
[ 5.09542770e-02,  1.08137603e+00,  1.15418403e+00,
 -8.78190343e-01, -6.31375988e-01,  9.94544570e-02,
 -1.37075998e+00,  3.08751242e-01,  2.10620213e-02,
 -2.41337791e-01, -8.58919908e-01,  6.99380484e-01,
 -2.22477010e-01,  8.65652923e-01,  1.72504416e-01,
 -1.06122229e+00, -1.14775325e-01, -1.01673865e+00,
  2.27392775e-01, -1.79422927e+00],
[-1.31054012e-01,  4.53781913e-01,  2.01125668e+00,
  7.67902408e-01, -1.82974041e+00, -1.78156286e+00,
```

```
         8.62789892e-03,  6.16886554e-01,  1.95069697e-01,
         3.70057219e-02, -1.11831192e+00,  5.89879821e-01,
        -8.05626508e-01,  5.27004208e-01, -4.45954265e-02,
        -3.63858810e-01,  3.54757693e-01,  1.96557401e-01,
        -7.29044659e-01,  1.13307988e+00],
       [-9.30156503e-01,  5.43311891e-01, -1.95180410e+00,
        -1.08403662e+00,  4.39042958e-01,  7.84957521e-01,
        -3.92388998e-01,  4.45393251e-01, -1.13980246e+00,
        -2.19541028e-01, -2.16731471e-01,  3.51780111e-01,
        -4.70032883e-01, -3.04614305e+00, -6.59891730e-01,
         3.79235534e-01, -2.16949570e-01, -4.70637658e-01,
        -5.54309627e-01, -1.78589092e-01],
       [ 2.25930895e+00,  9.10178908e-01, -3.69181838e-01,
        -4.66419097e-01,  3.17218215e-01,  6.55263731e-01,
         2.79924599e-01, -7.38030909e-01,  1.09965960e+00,
         7.86327962e-01,  3.79151736e-01, -9.44446256e-01,
        -1.70204139e-02, -9.81503896e-02, -2.39379178e-01,
        -4.10049693e-01, -2.43261244e-02, -1.61695604e+00,
         6.40131526e-01, -4.22571517e-02],
       [-7.92286662e-01,  1.59277075e+00, -1.29868672e+00,
        -1.37808347e+00, -2.58572632e-01,  2.05332564e-01,
         3.64481249e-01, -2.59576982e-01,  1.32501405e+00,
         3.08331246e-01,  1.68157672e+00, -3.11976108e-01,
        -1.00683175e+00,  1.47132196e+00,  1.27607535e+00,
        -8.40290395e-01, -2.76432845e-01,  2.33962481e+00,
         4.51340154e-02, -5.31605908e-01],
       [ 7.40510770e-01,  2.08106150e+00, -3.03396547e+00,
        -1.73255242e+00, -1.10070246e-01, -2.14620904e-01,
         2.73093436e+00, -1.95391443e-01, -4.85190914e+00,
        -9.06111604e-01, -9.41542009e-01,  1.94603199e+00,
        -2.55629824e+00,  2.87474136e+00, -1.55409133e-01,
         2.17926033e+00,  1.93085335e-01,  2.83733422e+00,
         2.33654627e+00,  1.89437189e+00],
       [ 3.96006713e-01, -7.69916074e-01,  3.76425531e-01,
         3.18305583e-02,  5.39249191e-01,  1.32638590e+00,
         6.72294757e-01,  1.84926373e+00,  2.98238174e-01,
        -6.74332661e-01, -2.08298756e-01, -6.35846078e-01,
         5.76590817e-01,  4.07461836e-01, -1.09940079e+00,
         6.76433295e-01, -4.35153552e-01, -1.49634540e-01,
        -6.94567860e-01, -1.09306151e+00],
       [-1.01804188e+00, -1.02993528e+00,  7.71405949e-01,
         1.29802197e+00, -3.49943365e-01, -4.24317621e-01,
        -4.57039607e-02,  5.53132064e-01, -9.08763246e-01,
         1.10028434e+00, -5.14233966e-01,  2.69622405e+00,
        -6.58552967e-01,  2.20507656e-01,  1.02943883e+00,
        -7.39246663e-02,  1.51332808e+00, -2.65561909e+00,
         8.62596011e-01, -7.78547559e-02],
       [ 2.17097407e+00,  4.76444890e+00, -8.12047473e-01,
         5.32890682e-01, -2.71142745e+00, -2.28205064e-01,
        -1.68846173e+00,  1.41128162e+00, -7.97572347e-01,
        -1.65439307e+00, -8.31489403e-01, -1.04902439e+00,
         1.62620254e+00, -4.58501258e-01,  4.32343475e+00,
        -1.91386286e+00,  1.34621664e-01,  4.12997674e-01,
        -9.13762652e-01, -2.11995153e+00],
       [ 1.12859406e+00, -2.01640663e+00, -5.17519043e-01,
        -7.09727966e-01, -5.39454633e-01,  1.81338429e-01,
        -2.28862004e+00, -7.93117363e-01, -4.39189522e-01,
        -2.75670535e-01, -8.82418819e-01,  1.73887268e+00,
         1.31913688e+00,  2.51484415e-01, -9.78829859e-01,
         9.94394391e-01, -9.60504382e-01,  2.41245368e+00,
        -5.02816701e-01,  4.96000946e-01],
       [ 2.16323595e+00, -1.29285691e+00, -3.53993911e-01,
        -1.16809350e+00,  2.67050869e-01, -2.22340315e+00,
        -7.39562996e-01,  5.21650793e-02, -6.43618403e-01,
        -3.92828182e-02,  8.23504154e-01,  5.23276661e-01,
```

```
          7.71790551e-01,  1.54301460e+00, -1.37495129e+00,
         -1.71546331e-01, -1.10438334e+00, -1.60205766e+00,
          6.25231451e-01,  1.33652795e+00]]),
 array([ 1.20866536e+02,  1.65803518e+02, -1.53756503e+02,  4.69095117e+02,
        -2.86860094e+02,  5.35979427e+01,  1.29164776e+02, -1.53076680e+02,
         3.42737911e+02, -1.37362797e+02,  7.40341003e+01, -6.30316704e+02,
         3.01786231e+01, -7.36119863e+01,  1.31035960e+02, -2.74922835e+01,
         2.65667308e+02, -1.93082191e+02, -3.51896400e+02, -3.14499314e+02,
        -2.87778330e+02, -2.65077974e+01, -2.08842388e+02, -2.68681688e+02,
         1.93296790e+02,  1.69420194e+02, -9.25306276e+01, -1.69610825e+01,
         2.18255739e+02,  5.95331662e+02, -4.48494943e+01, -2.97274829e+02,
         2.11881438e+01, -3.32267944e+02, -1.04423680e+02,  8.39884740e+00,
        -3.81137659e-01,  1.35426279e+02, -4.43431425e+02,  5.66710656e+01,
         1.23973757e+01, -4.37652098e+02, -5.07097220e+00,  3.88478984e+02,
        -2.02361875e+02,  3.47374988e+01, -8.03250608e+01,  2.70115543e+02,
         1.18312506e+01, -5.23890666e+02]))
```

Part 1 - c

Load Boston

In [34]:
```
from sklearn.datasets import load_boston
boston = load_boston()
```

Sklearn returns Dictionary-like object, the interesting attributes are: ‘data’, the data to learn, ‘target’, the regression targets, ‘DESCR’, the full description of the dataset, and ‘filename’, the physical location of boston csv dataset. This we can from the following Operations.

In [35]:
```
print(type(boston))
print('\n')
print(boston.keys())
print('\n')
print(boston.data.shape)
print('\n')
```

```
<class 'sklearn.utils.Bunch'>


dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])


(506, 13)
```

The data has 506 rows and 13 feature variable. Notice that this doesn’t include target variable. Also the names of the columns are also extracted.

The details about the features and more information about the dataset can be seen by using boston.DESCR`

In [36]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------
```

```
**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 1
4) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon U
niversity.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address r
egression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and So
urces of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceeding
s on the Tenth International Conference of Machine Learning, 236-243, University of Mass
achusetts, Amherst. Morgan Kaufmann.
```

Answer to Part 2 Question 4:

In [37]:

```python
print(boston.feature_names)
print(boston.target)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
[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]
```

We can convert this to a pandas dataframe, which we can do by calling the dataframe on boston.data. We also adds the target variable to the dataframe from boston.target

In [38]:

```
bos = pd.DataFrame(boston.data, columns = boston.feature_names)
bos['PRICE'] = boston.target

print(bos.head())
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0   
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0   
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0   
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0   
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0   

   PTRATIO       B  LSTAT  PRICE  
0     15.3  396.90   4.98   24.0  
1     17.8  396.90   9.14   21.6  
2     17.8  392.83   4.03   34.7  
3     18.7  394.63   2.94   33.4  
4     18.7  396.90   5.33   36.2  
```

In [39]:

```
bos.isnull().sum()
print(bos.describe())
```

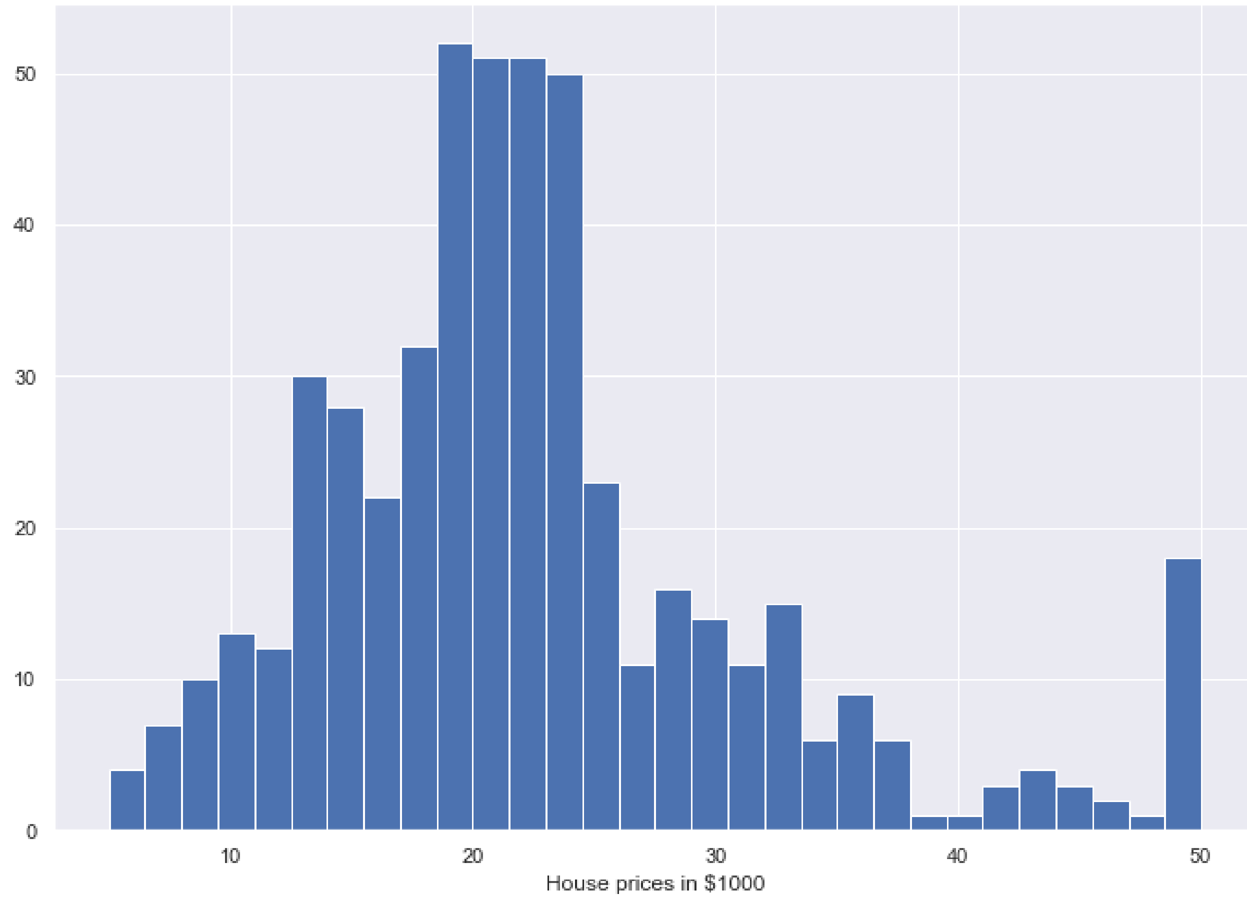
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
```

```
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT       PRICE
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

Let’s first plot the distribution of the target variable. We will use the histogram plot function from the matplotlib library.

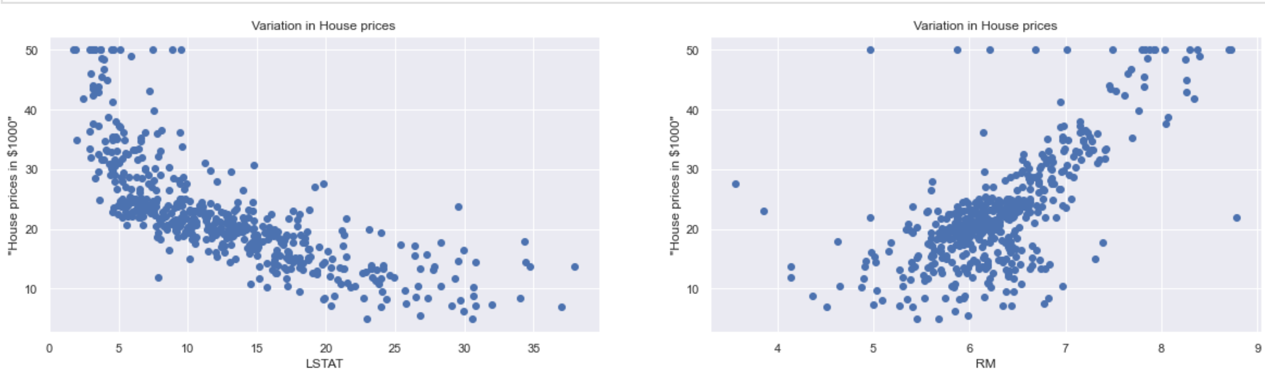
In [40]:

```
import seaborn as sns
sns.set(rc={'figure.figsize':(11.7,8.27)})
plt.hist(bos['PRICE'], bins=30)
plt.xlabel("House prices in $1000")
plt.show()
```



In [41]:

```
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = bos['PRICE']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = bos[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title("Variation in House prices")
    plt.xlabel(col)
    plt.ylabel('"House prices in $1000"')
```



In [42]:

```
boston
```

Out[42]:

```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]]),
 'target': array([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]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RA
D',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n------------------------
--\n\n**Data Set Characteristics:**  \n\n    :Number of Instances: 506 \n\n    :Number o
f Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually
```

```
the target.\n\n    :Attribute Information (in order):\n        - CRIM     per capita cri
me rate by town\n        - ZN       proportion of residential land zoned for lots over 2
5,000 sq.ft.\n        - INDUS    proportion of non-retail business acres per town\n
- CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)\n
- NOX      nitric oxides concentration (parts per 10 million)\n        - RM       averag
e number of rooms per dwelling\n        - AGE      proportion of owner-occupied units bu
ilt prior to 1940\n        - DIS      weighted distances to five Boston employment centr
es\n        - RAD      index of accessibility to radial highways\n        - TAX      ful
l-value property-tax rate per $10,000\n        - PTRATIO  pupil-teacher ratio by town\n
- B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town\n        - LST
AT    % lower status of the population\n        - MEDV     Median value of owner-occupie
d homes in $1000's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D.
and Rubinfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttps://archive.ics.uc
i.edu/ml/machine-learning-databases/housing/\n\n\nThis dataset was taken from the StatLi
b library which is maintained at Carnegie Mellon University.\n\nThe Boston house-price d
ata of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air',
J. Environ. Economics & Management,\nvol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsc
h, 'Regression diagnostics\n...', Wiley, 1980.   N.B. Various transformations are used i
n the table on\npages 244-261 of the latter.\n\nThe Boston house-price data has been use
d in many machine learning papers that address regression\nproblems.   \n     \n.. topi
c:: References\n\n   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influ
ential Data and Sources of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1993).
Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth Internati
onal Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morg
an Kaufmann.\n",
 'filename': 'C:\\Users\\SRIDHAR\\anaconda3\\lib\\site-packages\\sklearn\\datasets\\data
\\boston_house_prices.csv'}
```


Part 1 - d

Dummy Classifier

class sklearn.dummy.DummyClassifier(*, strategy='prior', random_state=None, constant=None)

Part 2 - Question 5

A dummy classifier is a type of classifier which does not generate any valuable 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. AS the name suggests, 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. It is based on the philosophy that any analytic approach for a classification problem should be better than a random guessing approach.

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

Most Frequent: The classifier always predicts the most frequent class label in the training data.
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.
Uniform: It generates predictions uniformly at random.
Constant: The classifier always predicts a constant label and is primarily used when classifying non-majority class labels.

Part 2 - Question 1

Parameters:

strategy{“most_frequent”, “prior”, “stratified”, “uniform”, “constant”}, default=”prior”
random_stateint, RandomState instance or None, default=None
constantint or str or array-like of shape (n_outputs,), default=None

In [66]:
```
from sklearn.dummy import DummyClassifier
```

In [108…
```
df = pd.read_csv('weatherAUS.csv')
df.columns
```

Out[108…
```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')
```

In [109…
```
features = ['MinTemp','MaxTemp','Rainfall','Pressure9am','Pressure3pm']
check_rows = features[:]
check_rows.append('RainTomorrow')
df = df.dropna(subset = check_rows)
X = df[features]
y = df['RainTomorrow']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

We are picking out some features and training our classifier based on it, to see how it works

In [110…
```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
```

Out[110…
```
KNeighborsClassifier(n_neighbors=3)
```

In [58]:
```
len(X_train) / len(X)
```

Out[58]:
```
0.75
```

In [111…
```
y_pred = clf.predict(X_test)
y_pred
```

Out[111…
```
array(['No', 'No', 'No', ..., 'Yes', 'Yes', 'No'], dtype=object)
```

In [65]:
```
clf.score(X_test, y_test)
```

Out[65]:
```
0.7898578199052133
```

In [67]:
```
frequent_clf = DummyClassifier(strategy = 'most_frequent').fit(X_train, y_train)
```

In [68]:
```
y_freq_pred = frequent_clf.predict(X_test)
```

In [69]:
```
uniform_clf = DummyClassifier(strategy = 'uniform').fit(X_train, y_train)
```

In [70]:
```
y_uniform_pred = uniform_clf.predict(X_test)
```

In [71]:
```
frequent_clf.score(X_test, y_test)
```

Out[71]:
```
0.7777567140600316
```

In [72]:
```
uniform_clf.score(X_test, y_test)
```

Out[72]:
```
0.4996208530805687
```

Part 2 Question 6

Confusion Matrix for the weather dataset

In [73]:
```
from sklearn.metrics import confusion_matrix
```

In [74]:
```
confusion_matrix(y_test, y_pred)
```

Out[74]:
```
array([[22080,  2536],
       [ 4115,  2919]], dtype=int64)
```

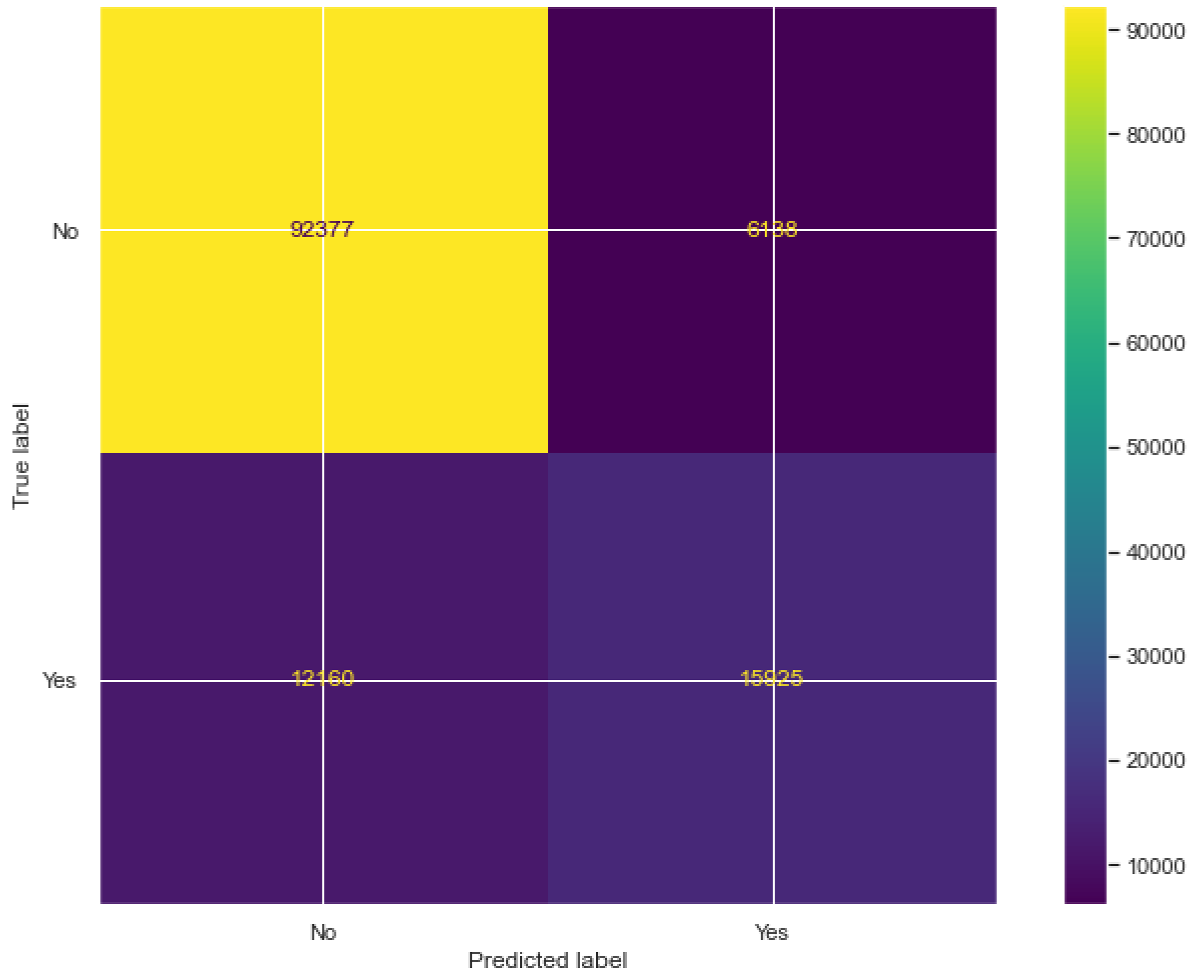
In [75]:
```
confusion_matrix(y_test, y_uniform_pred)
```

Out[75]:
```
array([[12268, 12348],
       [ 3572,  3462]], dtype=int64)
```

In [76]:
```
confusion_matrix(y_test, y_freq_pred)
```

Out[76]:
```
array([[24616,     0],
       [ 7034,     0]], dtype=int64)
```

In [100…
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clf, X, y)
plt.show()
```



In [79]:
```
dummy_clf.predict(X)
```
Out[79]: array([1, 0, 0, 1])

In [80]:
```
dummy_clf.score(X, y)
```
Out[80]: 0.75

Part 1- c

Loading Iris dataset

In [103…
```
from sklearn.datasets import load_iris
iris = load_iris()
```

In [104…
```
# Feature names of iris dataset
iris.feature_names
```
Out[104…
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [105…

```
# Target names of iris dataset
iris.target_names
```

Out[105…

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

In [106…

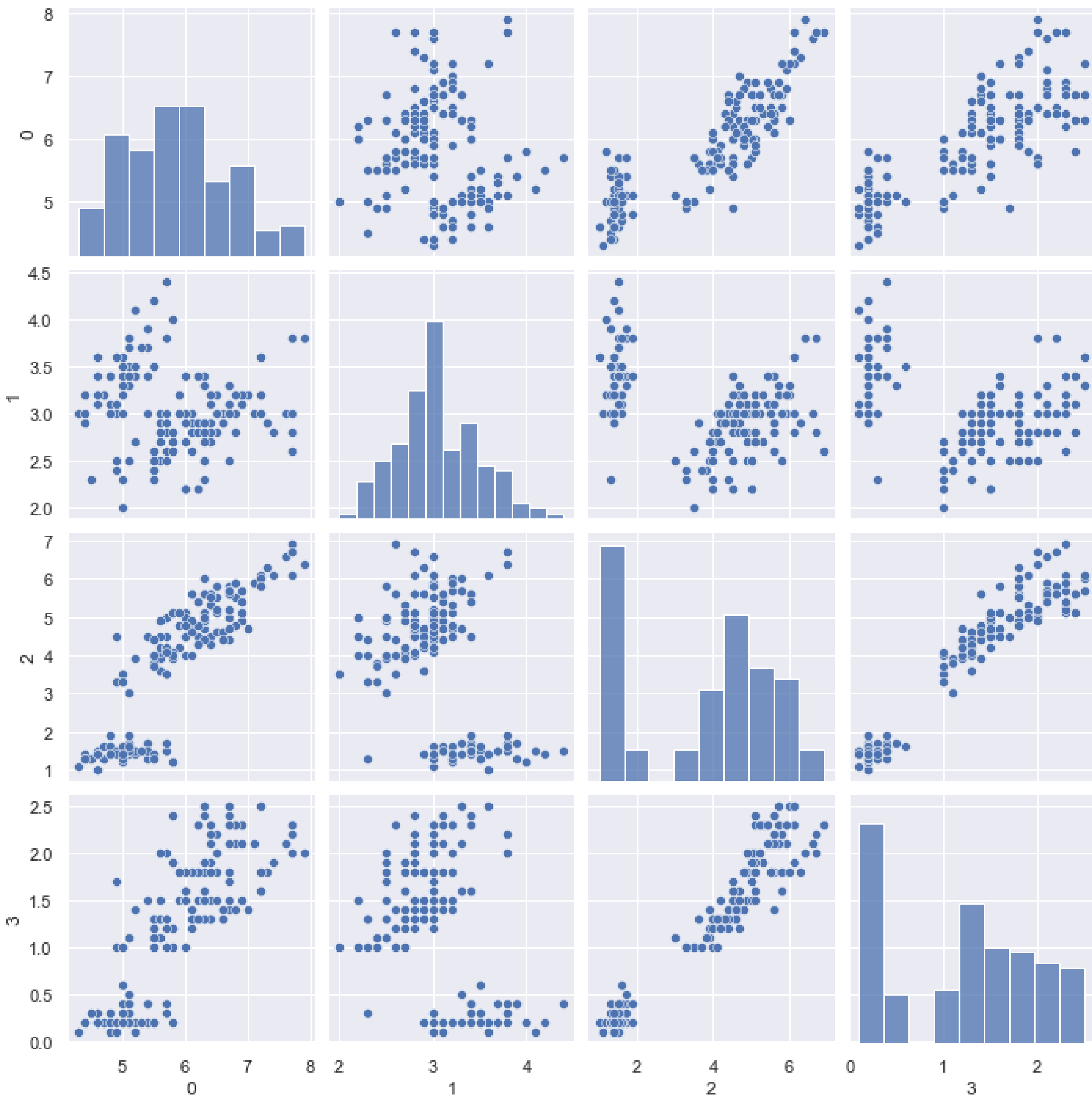
```
data1=pd.DataFrame(data=iris.data)
```

In [107…

```
# plotting multiple data pairwise bivariate distributions in iris dataset
import seaborn as sns
sns.pairplot(data1)
```

Out[107…

```
<seaborn.axisgrid.PairGrid at 0x1ddcc0e7f70>
```



Part 2 - QUESTION 6:

In [141…

```
def evaluateModel(X,y,percentage):
    X_train, X_test,y_train,y_test =train_test_split(X,y,train_size=percentage)

    model = DummyClassifier()
    model.fit(X_test, y_test)
    y_predicted = model.predict(X_test)

    acc=accuracy_score(y_test, y_predicted)
    print("Accuracy is: {}". format(acc))
    return acc
```

Part 1 - d

Part 2 - Question 5:

DummyRegressor

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.

Part 2- Question 1

Parameters:

strategy{“mean”, “median”, “quantile”, “constant”}, default=”mean”
constantint or float or array-like of shape (n_outputs,), default=None
quantilefloat in [0.0, 1.0], default=None

In [49]:

```
from sklearn.dummy import DummyRegressor
X = np.array([1.0, 2.0, 3.0, 4.0])
y = np.array([2.0, 3.0, 5.0, 10.0])
dummy_regr = DummyRegressor(strategy="mean")
dummy_regr.fit(X, y)
```

Out[49]: DummyRegressor()

In [50]:

```
dummy_regr.predict(X)
```

Out[50]: array([5., 5., 5., 5.])

In [51]:

```
dummy_regr.score(X, y)
```

Out[51]: 0.0

In [112…

```
from sklearn.linear_model import LinearRegression
boston=datasets.load_boston()
X=boston.data[:, None, 6]
y= boston.target
```

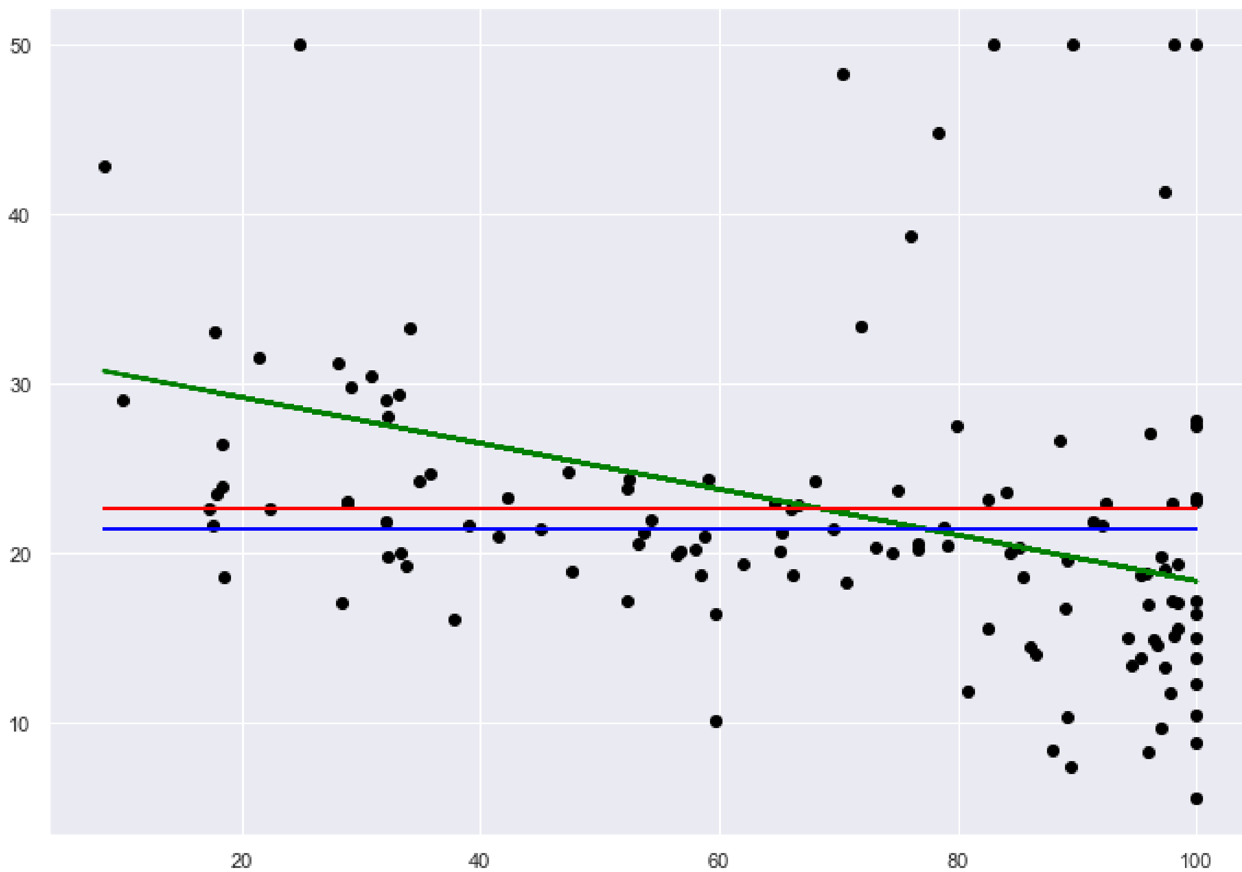
In [114...

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lm = LinearRegression().fit(X_train, y_train)
lm_dummy_mean = DummyRegressor(strategy = 'mean').fit(X_train, y_train)

lm_dummy_median = DummyRegressor(strategy = 'median').fit(X_train, y_train)
y_predict = lm.predict(X_test)
y_predict_dummy_mean = lm_dummy_mean.predict(X_test)
y_predict_dummy_median = lm_dummy_median.predict(X_test)
```

In [115...

```
plt.scatter(X_test, y_test,  color='black')
plt.plot(X_test, y_predict, color='green', linewidth=2)
plt.plot(X_test, y_predict_dummy_median, color='blue', linestyle = 'dashed',
         linewidth=2, label = 'dummy')
plt.plot(X_test, y_predict_dummy_mean, color='red', linestyle = 'dashed',
         linewidth=2, label = 'dummy')
```

Out[115... [<matplotlib.lines.Line2D at 0x1ddcbe75fd0>]



Part 1 - e

ACCURACY SCORE

sklearn.metrics.accuracy_score(y_true,y_pred,normalize=False,sample_weight=None)

The accuracy_score method is used to calculate the accuracy of either the faction or count of correct prediction in Python Scikit learn. Mathematically it represents the ratio of the sum of true positives and true negatives out of all the predictions.

Accuracy Score = (TP+TN)/ (TP+FN+TN+FP)

Here we can also calculate accuracy with the help of the accuracy_score method from sklearn.

accuracy_score(y_true, y_pred, normalize=False)

Parameter:

y_true : 1d array-like, or label indicator array / sparse matrix

y_pred : 1d array-like, or label indicator array / sparse matrix

Normalize : bool, optional (default=True)

sample_weight : array-like of shape = [n_samples], optional

In [82]:
```
from sklearn.metrics import accuracy_score
y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2, 3]
accuracy_score(y_true, y_pred)
```

Out[82]:
```
0.5
```

If normalize == False, it returns the number of correctly confidentialsamples(int)

In [83]:
```
accuracy_score(y_true, y_pred, normalize=False)
```

Out[83]:
```
2
```

In [84]:
```
import numpy as np
accuracy_score(np.array([[0, 1], [1, 1]]), np.ones((2, 2)))
```

Out[84]:
```
0.5
```

The scikit learn accuracy_score works with multilabel classification in which the accuracy_score function calculates subset accuracy.
The accuracy of the model is calculated as the ratio between the number of correct predictions to the total number of predictions.

In [85]:
```
import sklearn.metrics

y_true = ["positive", "negative", "negative", "positive", "positive", "positive", "nega
y_pred = ["positive", "negative", "positive", "positive", "negative", "positive", "posi

r = sklearn.metrics.confusion_matrix(y_true, y_pred)

r = np.flip(r)

accuracy = (r[0][0] + r[-1][-1]) / np.sum(r)
print(accuracy)
```

```
0.5714285714285714
```

Classification Report

In [88]:
```
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3

    accuracy                           0.60         5
   macro avg       0.50      0.56      0.49         5
weighted avg       0.70      0.60      0.61         5
```

In [89]:
```
y_pred = [1, 1, 0]
y_true = [1, 1, 1]
print(classification_report(y_true, y_pred, labels=[1, 2, 3]))
```

```
              precision    recall  f1-score   support

           1       1.00      0.67      0.80         3
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0

   micro avg       1.00      0.67      0.80         3
   macro avg       0.33      0.22      0.27         3
weighted avg       1.00      0.67      0.80         3

C:\Users\SRIDHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labe
ls with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\SRIDHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels
with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\SRIDHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labe
ls with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\SRIDHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels
with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\SRIDHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labe
ls with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\SRIDHAR\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Un
definedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels
with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Here we have used datasets to load the inbuilt wine dataset to train the model. Create Classification Report.

In [91]:
```
from sklearn.tree import DecisionTreeClassifier
wine = datasets.load_wine()
X = wine.data
y = wine.target
class_names = wine.target_names
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
classifier_tree = DecisionTreeClassifier()
y_predict = classifier_tree.fit(X_train, y_train).predict(X_test)
print(classification_report(y_test, y_predict, target_names=class_names))
```

```
              precision    recall  f1-score   support

     class_0       0.83      0.95      0.88        20
     class_1       0.94      0.80      0.86        20
     class_2       1.00      1.00      1.00        14

    accuracy                           0.91        54
   macro avg       0.92      0.92      0.92        54
weighted avg       0.91      0.91      0.91        54
```

Confusion Matrix for wine dataset

In [93]:
```
print(confusion_matrix(y_test, y_predict))
```

```
[[19  1  0]
 [ 4 16  0]
 [ 0  0 14]]
```

sklearn.metrics.confusion_matrix(y_true, y_pred, labels=None, sample_weight=None)

Compute confusion matrix to evaluate the accuracy of a classification

By definition a confusion matrix (C) is such that (C_{i, j}) is equal to the number of observations known to be in group (i) but predicted to be in group (j).

Thus in binary classification, the count of true negatives is (C*{0,0}), false negatives is (C*{1,0}), true positives is (C*{1,1}) and false positives is (C*{0,1}).

PARAMETER:

y_true : array, shape = [n_samples]

y_pred : array, shape = [n_samples]

labels : array, shape = [n_classes], optional

sample_weight : array-like of shape = [n_samples], optional

In [95]:
```
y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
confusion_matrix(y_true, y_pred, labels=["ant", "bird", "cat"])
```

Out[95]:
```
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]], dtype=int64)
```

In the binary case, we can extract true positives, etc as follows:

In [96]:
```
tn, fp, fn, tp = confusion_matrix([0, 1, 0, 1], [1, 1, 1, 0]).ravel()
(tn, fp, fn, tp)
```

Out[96]:
```
(0, 2, 1, 1)
```

CONCLUSION:

We have imported and explored the various methods of Scikit-learn library. We have learnt the various functions it does, the parameters of the functions. How to alter the parameters to get the desired output. We also learnt to explore one function using another function of the Scikit-learn. We represented some of our findings using graphs and plots. We used some toy datasets to perform certain functions.

REFERENCES:

https://www.youtube.com/watch?v=BUkqYGPnLZ8 https://scikit-learn.org/0.19/modules/generated/sklearn.model_selection.train_test_split.html
https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html
https://www.kite.com/python/docs/sklearn.datasets.make_classification
https://www.codespeedy.com/make_regression-function-in-sklearn-with-python/
https://www.geeksforgeeks.org/splitting-data-for-machine-learning-models/
https://www.programcreek.com/python/example/85943/sklearn.datasets.make_classification
https://stackoverflow.com/questions/18259372/y-from-sklearn-datasets-make-classification
https://www.youtube.com/watch?v=XZKSLVy6MnA
https://docs.w3cub.com/scikit_learn/modules/generated/sklearn.datasets.make_regression
https://ogrisel.github.io/scikit-learn.org/sklearn-tutorial/modules/generated/sklearn.datasets.load_boston.html
https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html#sklearn.dummy.DummyClas
https://www.geeksforgeeks.org/ml-dummy-classifiers-using-sklearn/
https://www.geeksforgeeks.org/dummy-regressor/
https://www.codespeedy.com/dummy-classifiers-using-sklearn-library-in-python/
https://www.kite.com/python/docs/sklearn.dummy.DummyClassifier
https://www.programcreek.com/python/example/95237/sklearn.dummy.DummyClassifier
https://stackoverflow.com/questions/29441943/what-is-the-theorical-foundation-for-scikit-learn-dummy-classifier
https://www.kaggle.com/sarose13/using-dummy-classifiers-as-performance-baseline https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
https://pythonguides.com/scikit-learn-accuracy-score/
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
https://www.projectpro.io/recipes/generate-classification-report-and-confusion-matrix-in-python
https://docs.w3cub.com/scikit_learn/modules/generated/sklearn.metrics.classification_report
https://www.youtube.com/results?search_query=accurary_score+and+dummyclassifier