Level 1

Name : Bhargav Adya L

Task 1

Linear Regression Boston Dataset

Linear Regression is a machine learning algorithm based on supervised learning. It is used for finding out the relationship between variables and forecasting.

A linear regression line has an equation of the form **Y = a + bX**, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when x = 0).

Linear Regression is one of the basic algorithms in machine learning. We start with importing basic modules such as numpy, pandas and matplotlib.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.datasets
from sklearn.model_selection import train_test_split
```

We import datasets from sklearn.datasets and load the Boston dataset to variable.

```
house_price = sklearn.datasets.load_boston()
```

```
print(house_price)
```

```
{'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. ,
```

We put the data into a data frame which behaves like an 2D array in C, using Pandas. With data frames we can manipulate the data mathematically by using inbuilt methods in pandas which is called data cleaning. pd.DataFrame() converts data to data frame , columns headings are set to the feature_names. Then we add the house prices to it.

```
house_df = pd.DataFrame(house_price.data, columns = house_price.feature_names)
```

```
house_df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

:
```
house_df['price']=house_price.target
```

:
```
house_df.shape
```

:
```
(506, 14)
```

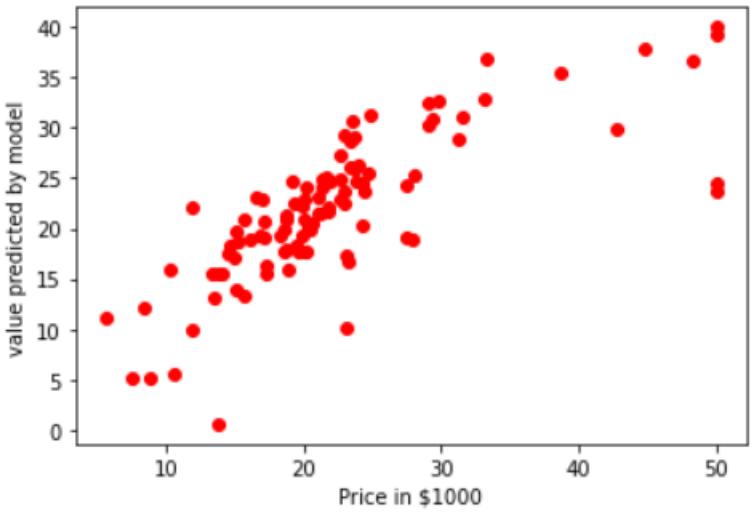
```
x = house_price.data
y = house_price.target
```

```
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size= 0.2,random_state=0)

print("xtrain , ",xtrain.shape)
print("xtets , ",xtest.shape)
print("ytrain , ",ytrain.shape)
print("ytest , ",ytest.shape)
```

```
xtrain ,  (404, 13)
xtets ,  (102, 13)
ytrain ,  (404,)
ytest ,  (102,)
```

```
from sklearn.linear_model import LinearRegression # importing linear regression
regressor = LinearRegression()                    # creating an instance
regressor.fit(xtrain, ytrain)                     #fitting the model
y_pred= regressor.predict(xtest)                  # pricting the test values in xtest
```

```
plt.scatter(ytest,y_pred, c= 'red')
plt.xlabel('Price in $1000')
plt.ylabel('value predicted by model')
plt.show()
```




```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(ytest,y_pred)  # calculating mean squared error
print(mse)
```

```
33.44897999767653
```

The model 66.55% accurate.

Logistic regression on Iris Dataset

Logistic regression is modelling the probability of a discrete outcome given an input variable. It says whether something is true or false. The prediction logistic regression is categorical.

We start with loading the required modules and iris data from sklearn.datasets.

```
import pandas as pd       #importing required modules
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.datasets import load_iris
iris = load_iris() #importing data
```

The different classes in iris dataset is present in iris.target and different features based on which they are divided are present in iris.feature_names.

```
print(iris.target_names)
```

```
['setosa' 'versicolor' 'virginica']
```

```
print(iris.feature_names)
```

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

The data is divided to train and test using train_test_split. The parameter test_size is used to say how much percent of the data is used for testing the model.

```
x_train,x_test,y_train,y_test= sklearn.model_selection.train_test_split(x,y,test_size=0.25,random_state = 2)
```

```
logr = LogisticRegression(random_state = 0) # creating an instance
logr.fit(x_train,y_train)          # fitting the model with data
```

The logistic regression instance is created and the model is fit with training data

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
cm
```

```
array([[16,  0,  0],
       [ 0, 10,  1],
       [ 0,  0, 11]], dtype=int64)
```

The results are tested with confusion matrix only once the prediction was wrong. When the data was versicolor it predicted virginica. The accuracy is found to be 97%

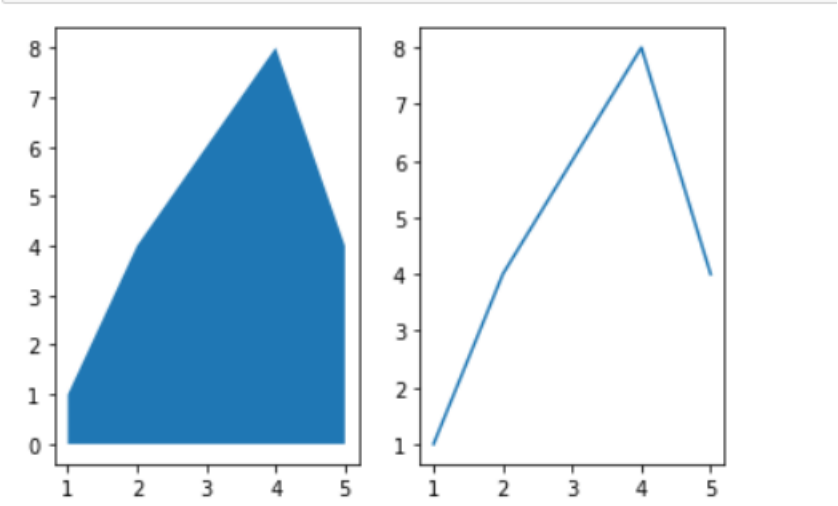
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_pred))
```

```
0.9736842105263158
```

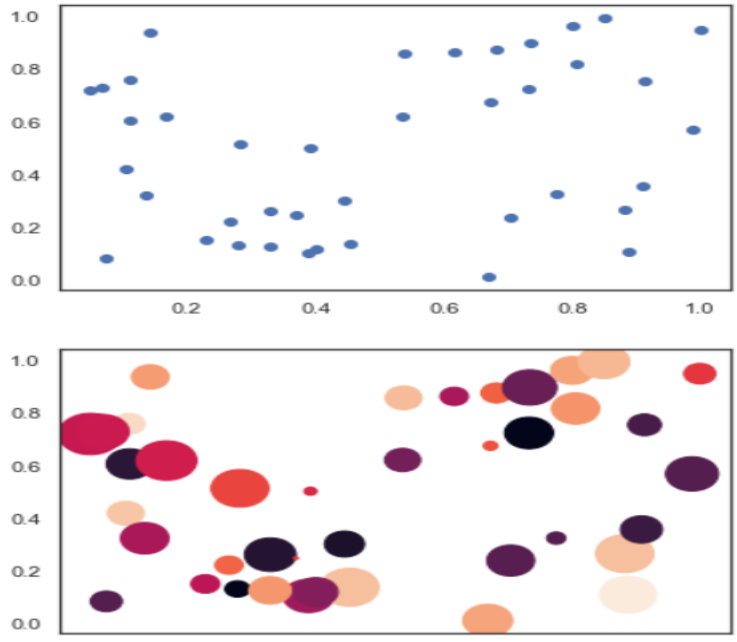
Task 2: Plotting graphs

Some of the plot are shown here, matplotlib.pyplot and seaborn are used to plot graphs.

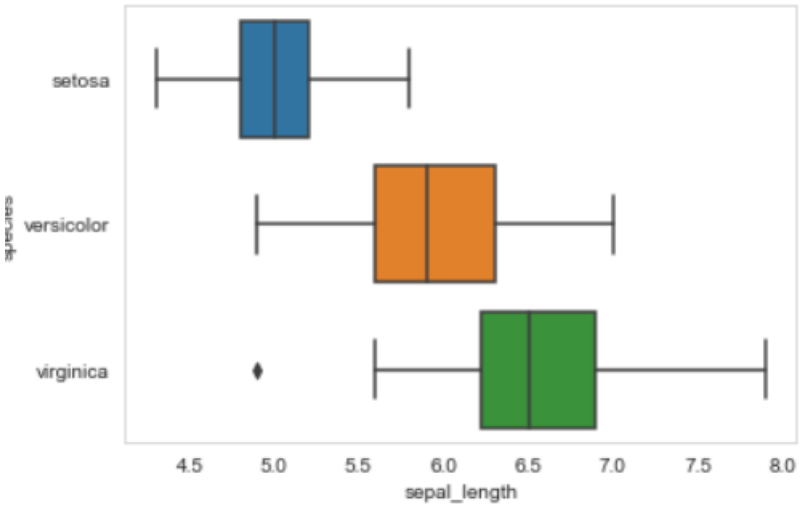
Area plot and line plot



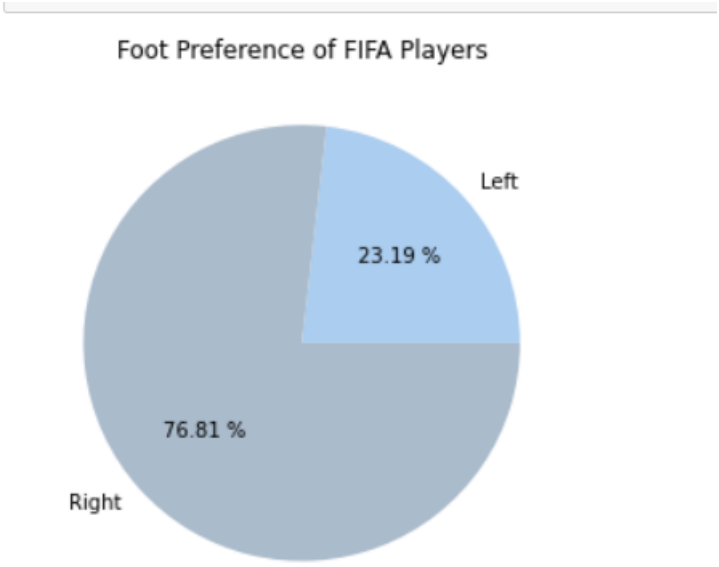
Scatter and Bubble Plot



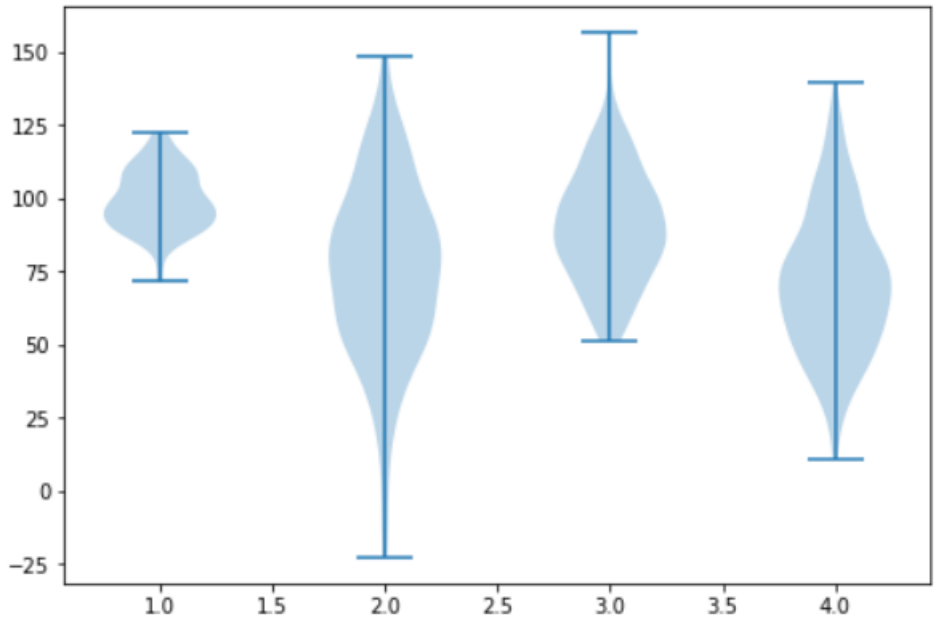
Box plot



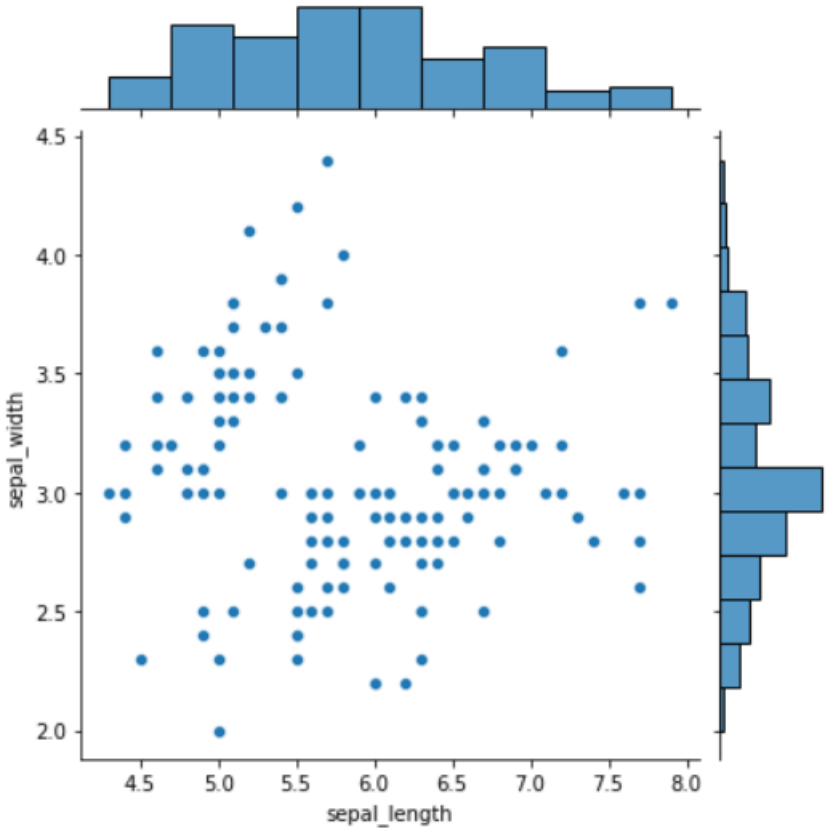
Pie plot



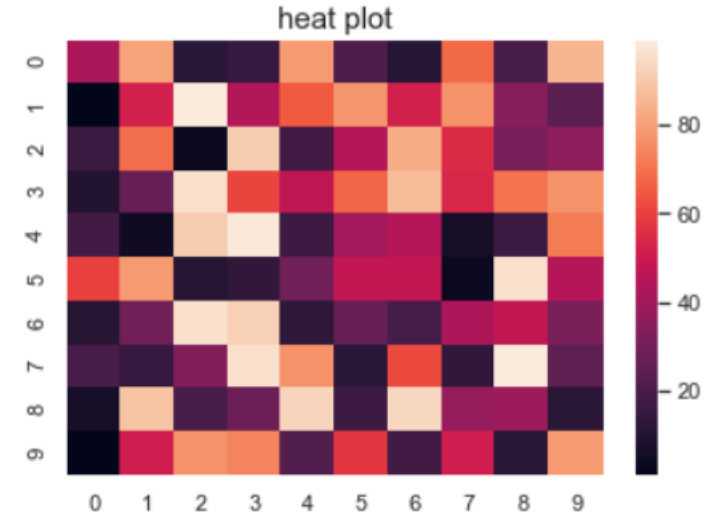
Violin plot

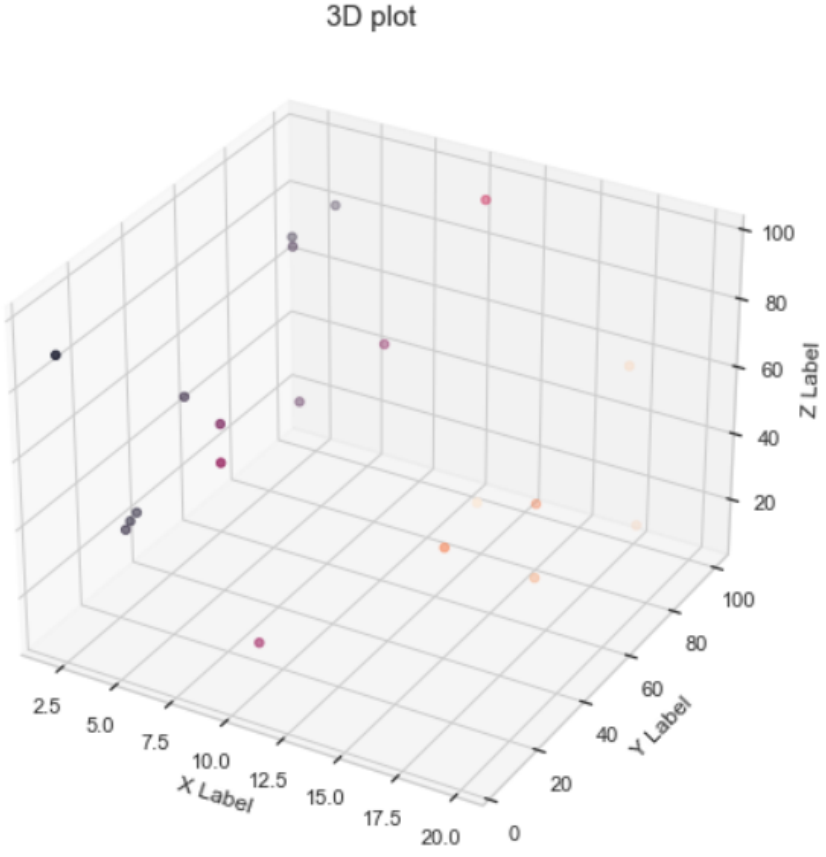


Marginal plot




Filled Contours Plot
y (cm)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
4.8
4.2
3.6
3.0
2.4
1.8
1.2
0.6
0.0


heat plot
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
80
60
40
20


3D plot
X Label
Y Label
Z Label
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
20
40
60
80
100
20
40
60
80
100

Task 3: Metrics and Performance Evaluation

The top_k_accuracy_score function is a generalization of accuracy_score. a prediction is considered correct as long as the true label is associated with one of the k highest predicted scores.

```
import numpy as np
from sklearn.metrics import top_k_accuracy_score
y_true = np.array([0, 1, 2, 2])
y_score = np.array([[0.5, 0.2, 0.2],
                    [0.3, 0.4, 0.2],
                    [0.2, 0.4, 0.3],
                    [0.7, 0.2, 0.1]])
top_k_accuracy_score(y_true, y_score, k=2)
```

```
0.75
```

The accuracy_score function computes either the fraction (default) or the count (normalize=False) of correct predictions.

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

acc = np.sum(predictions == y_test)/len(y_test)

The classification_report function builds a text report showing the main classification metrics.

```
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 0]
y_pred = [0, 0, 2, 1, 0]
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       0.67      1.00      0.80         2
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.50      0.67         2

    accuracy                           0.60         5
   macro avg       0.56      0.50      0.49         5
weighted avg       0.67      0.60      0.59         5
```

precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples

The F-measure is a weighted harmonic mean of the precision and recall. A Fβ measure reaches its best value at 1 and its worst score at 0

The hamming_loss computes the average Hamming loss or Hamming distance between two sets of samples

max_error function computes the maximum residual error , a metric that captures the worst case error between the predicted value and the true value.

```
from sklearn.metrics import max_error
y_true = [3, 2, 7, 1]
y_pred = [9, 2, 7, 1]
max_error(y_true, y_pred)
```

$$\text{Max Error}(y, \hat{y}) = max(|y_i - \hat{y}_i|)$$

Mean absolute error: function computes mean absolute error, a risk metric corresponding to the expected value of the absolute error loss or l1-norm loss.

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|.$$

R^2 score represents the proportion of variance (of y) that has been explained by the independent variables in the model. It provides an indication of goodness of fit and therefore a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance

```python
from sklearn.metrics import r2_score
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]
r2_score(y_true, y_pred)

y_true = [[0.5, 1], [-1, 1], [7, -6]]
y_pred = [[0, 2], [-1, 2], [8, -5]]
r2_score(y_true, y_pred, multioutput='variance_weighted')

y_true = [[0.5, 1], [-1, 1], [7, -6]]
y_pred = [[0, 2], [-1, 2], [8, -5]]
r2_score(y_true, y_pred, multioutput='uniform_average')

r2_score(y_true, y_pred, multioutput='raw_values')

r2_score(y_true, y_pred, multioutput=[0.3, 0.7])
```

```
0.9253456221198156
```

Task 4:

Implementing Linear regression from scratch

In linear Regression we try to predict the continuous values through the linear equation y = wx +b where w is the weight and b is called the bias

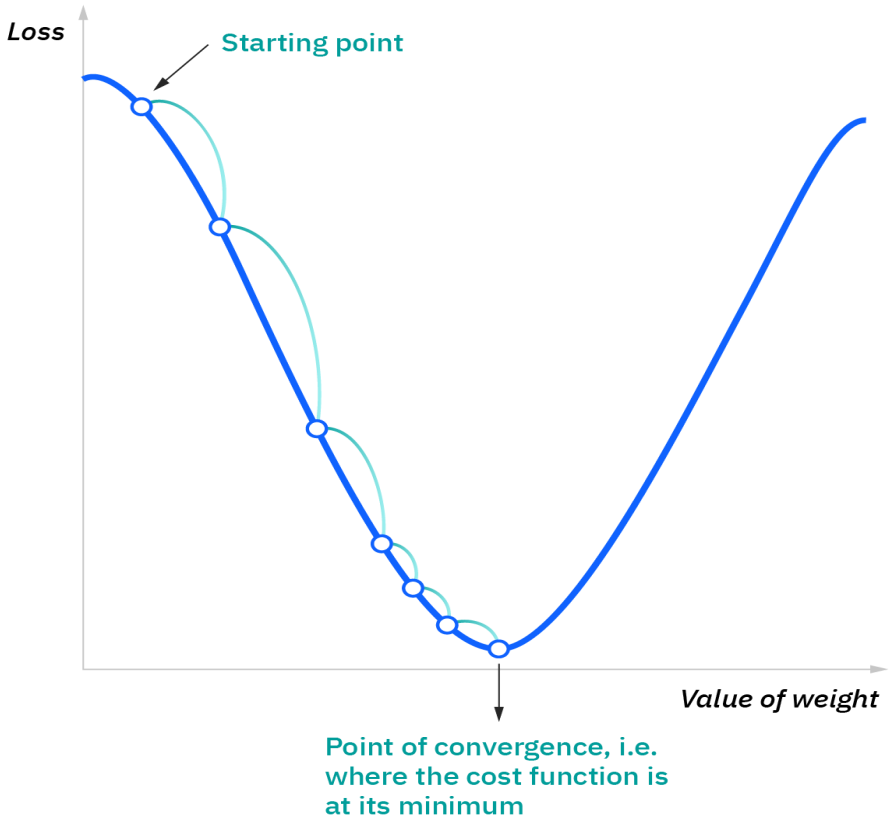
Cost Function

We initially take random value as bias then we calculate the error that we get which is called cost function.

$$MSE = J(w,b) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

$$J'(m,b) = \begin{bmatrix} \frac{df}{dw} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum -2x_i(y_i - (wx_i + b)) \\ \frac{1}{N}\sum -2(y_i - (wx_i + b)) \end{bmatrix}$$

Then plot a graph of error and for different weights, using derivatives we find the least loss which is called the gradient dissent



We rapidly or slowly move down the curve based on how far are we from the global minimum.

Adjusted weights and bias.

w = w- a.dw

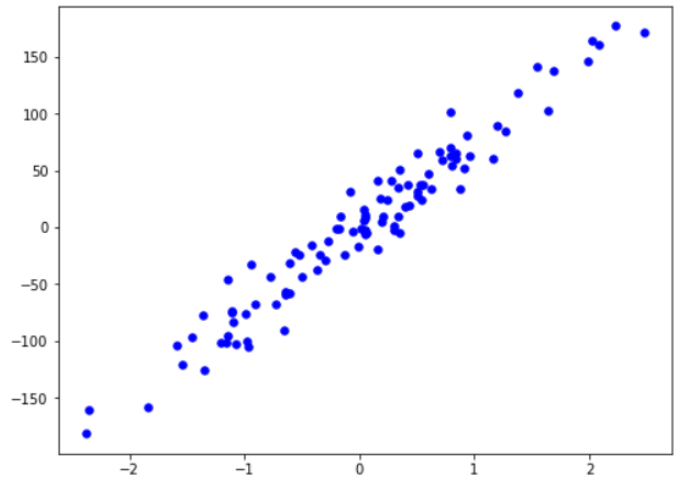
b = b – a.db

$$\frac{dJ}{dw} = dw = \frac{1}{N}\sum_{i=1}^{n} -2x_i(y_i - (wx_i + b)) = \frac{1}{N}\sum_{i=1}^{n} -2x_i(y_i - \hat{y}) = \frac{1}{N}\sum_{i=1}^{n} 2x_i(\hat{y} - y_i)$$

$$\frac{dJ}{db} = db = \frac{1}{N}\sum_{i=1}^{n} -2(y_i - (wx_i + b)) = \frac{1}{N}\sum_{i=1}^{n} -2(y_i - \hat{y}) = \frac{1}{N}\sum_{i=1}^{n} 2(\hat{y} - y_i)$$

```python
import numpy as np
class LinearRegression:
    def __init__(self,lr=0.001,n_iters=1000):
        self.lr = lr
        self.n_iters = n_iters
        self.weights =None #initially set to none
        self.bias =None
    def fit(self,X,y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iters):
            y_predicted = np.dot(X,self.weights) + self.bias

            dw = (1/n_samples) * np.dot(X.T, (y_predicted - y)) #.T gives the transpose of the matrix
            db = (1/n_samples) * np.sum(y_predicted-y)

            self.weights -= self.lr * dw # for updating the values fir each step
            self.bias -= self.lr * db

    def predict(self,X):
        y_predicted = np.dot(X,self.weights) + self.bias # final value is calculated through obtained weights and bias values
        return y_predicted
```

```
X,y = datasets.make_regression(n_samples =100,n_features=1,noise =20,random_state=4)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)
fig = plt.figure(figsize=(8,6))
plt.scatter(X[:,0],y,color = 'b', marker ='o', s = 30 )
plt.show()
```



The dataset is taken from dataset.make_regression which gives random data suitable for regression.

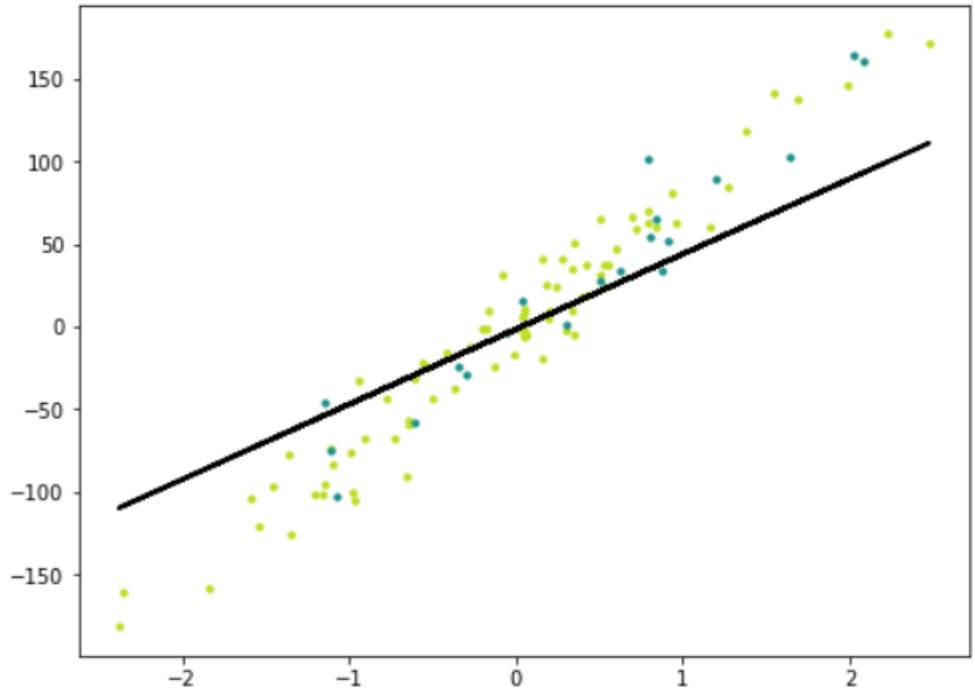
```
regressor = LinearRegression()
regressor.fit(X_train,y_train)
predicted = regressor.predict(X_test)
```

```
def mse(y_true,y_predicted):
    return np.mean((y_true- y_predicted)**2)
mse_value = mse (y_test,predicted)
print(mse_value)
```

```
1146.4968742011124
```

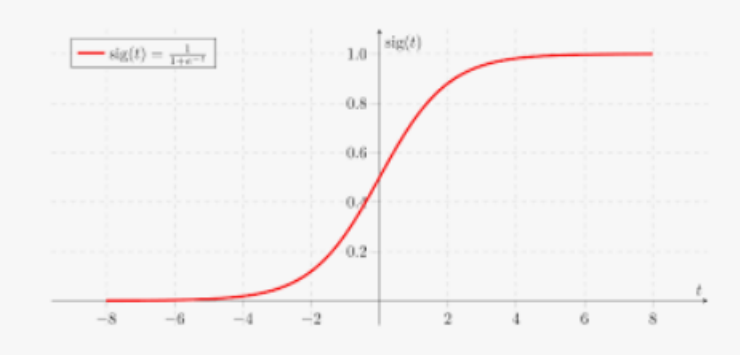
```
y_pred_line = regressor.predict(X)
cmap = plt.get_cmap('viridis')
fig = plt.figure(figsize =(8,6))
m1 = plt.scatter(X_train,y_train,color = cmap(0.9),s= 10)
m2 = plt.scatter(X_test, y_test, color=cmap(0.5), s= 10)
plt.plot(X,y_pred_line,color = 'black', linewidth =2,label ="Prediction")
```

```
[<matplotlib.lines.Line2D at 0x28f6ed3a7c0>]
```



Logistic Regression from scratch:

Logistic regression is used for binary classification so we use a sigmoid function s(x) = 1/(1 + (e)**-x)



If value is less than 0.5 then it is zero else it is classified one.

```python
import numpy as np

class LogisticRegression:
    def __init__(self,lr= 0.001, n_iters = 1000):
        self.lr =lr
        self.n_iters = n_iters
        self.weights = None
        self.bias =None

    def fit(self,X,y):
        n_samples,n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iters):
            linear_model = np.dot(X,self.weights) + self.bias
            y_predicted = self._sigmoid(linear_model)

            dw = 1/(n_samples) * np.dot(X.T,(y_predicted - y))
            db =  1/(n_samples) * np.sum(y_predicted - y)

            self.weights -= self.lr *dw
            self.bias -= self.lr*db

    def predict(self,X):
        linear_model = np.dot(X,self.weights) + self.bias
        y_predicted = self._sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else  0 for i in y_predicted]
        return y_predicted_cls

    def _sigmoid(self,x):
        return 1/(1 + np.exp(-x))
```

```python
from sklearn.model_selection import train_test_split
from sklearn import datasets
import matplotlib.pyplot as plt

bc = datasets.load_breast_cancer()
X,y = bc.data, bc.target
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2)
def accuracy(y_true,y_pred):
    accuracy = np.sum(y_true == y_pred)/len(y_true)
    return accuracy

regressor = LogisticRegression(lr = 0.0001, n_iters =1000)
regressor.fit(X_train,y_train)
predictions = regressor.predict(X_test)
print("LR classification accuracy",accuracy(y_test,predictions))
```

```
LR classification accuracy 0.9473684210526315
```

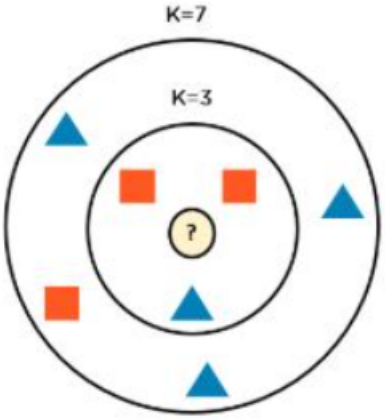
Task 5:

K – Nearest Neighbour (KNN) Algorithm :

KNN is a supervised classification algorithm we can use to assign a class to new data point. It can be used for regression as well when data is continuous.

Working:

Start with dataset with known categories. The new datapoints are classified based on the distance of the new data point from the nearest k neighbours. For computing the distance measures such as Euclidean distance, Hamming distance or Manhattan distance will be used.



In the case of Iris dataset which consists of 150 instances of three classes namely Setosa, Versicolour and Virginica. Based on four features sepal length, sepal width, petal length, petal width.

Importing the data from sklearn :

```
from sklearn import datasets
iris = datasets.load_iris() #importing iris dataset
```

Training the model:

The 80% of the available data is used for training and the remaining is used for the testing the model. We use module train_test_split available in sklern.model_selection to split the data.

```
from sklearn.model_selection import train_test_split
x = features # numpy array of four different features
y = labels  # their coressponding classes
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size =0.2,random_state =2)
```

```
len(x_train),len(x_test)
```

```
(120, 30)
```

```
from sklearn.neighbors import KNeighborsClassifier #importing knn classifier
```

```
classifier = KNeighborsClassifier(n_neighbors =2)
classifier.fit(x_train,y_train)
```

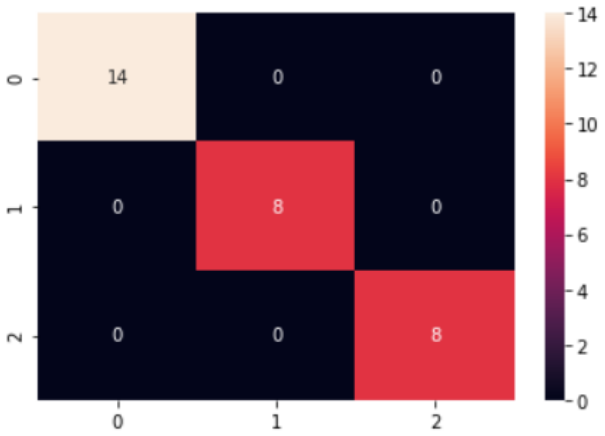
```
classifier.score(x_test,y_test)
```

```
1.0
```

```
from sklearn.metrics import confusion_matrix
y_pred = classifier.predict(x_test)

cm  =confusion_matrix(y_test,y_pred)
```

```
%matplotlib inline
import matplotlib as plt
import seaborn as sn
sn.heatmap(cm,annot = True)
```

```
<AxesSubplot:>
```



KNN for wine dataset

```
wine = datasets.load_wine()
print(wine.DESCR)
```

```
x1_train,x1_test,y1_train,y1_test = train_test_split(x1,y1,test_size =0.2,random_state =2)
```

```
knn = KNeighborsClassifier(n_neighbors =20)
```

```
knn.fit(x1_train,y1_train)
```

```
KNeighborsClassifier(n_neighbors=20)
```

```
knn.score(x1_test,y1_test)
```

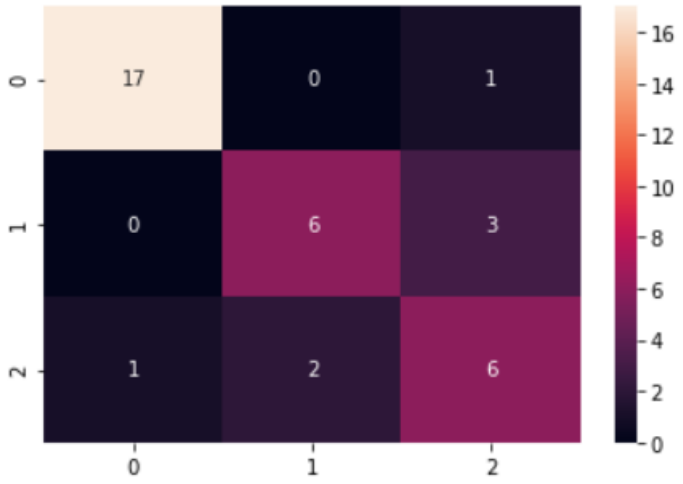
```
0.8055555555555556
```

The model score is 80%

```
y1_pred = knn.predict(x1_test)
cm1 = confusion_matrix(y1_test,y1_pred)
```

```
sn.heatmap(cm1,annot=True)
```

```
<AxesSubplot:>
```

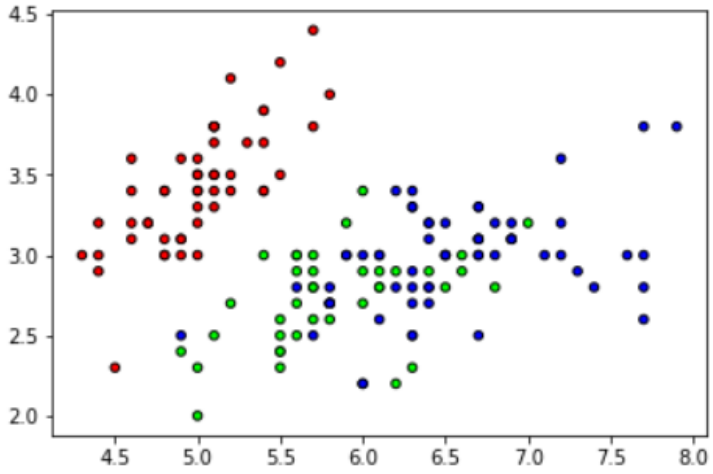


Implementing KNN from scratch

```
import numpy as np
from collections import Counter
def euclidean_dist(x1,x2):
    return np.sqrt(np.sum((x1-x2)**2))

class KNN:
    def __init__(self,k=3):
        self.k=k

    def fit(self,X,y):
        self.X_train = X
        self.y_train = y

    def predict(self,X):
        predicted_labels = [self._predict(x) for x in X]
        return np.array(predicted_labels)

    def _predict(self,x):
        distances = [euclidean_dist(x,x_train) for x_train in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        most_common =  Counter(k_nearest_labels).most_common()
        return most_common[0][0]
```

```
from matplotlib.colors import ListedColormap
```

```
cmap = ListedColormap(['#FF0000','#00FF00','#0000FF'])
import matplotlib.pyplot as plt
plt.figure()
plt.scatter(X[:,0],X[:,1],c=y,cmap =cmap,edgecolor ='k', s= 20)

iris_data = datasets.load_iris()
X,y = iris_data.data, iris_data.target
```




```
clf = KNN(k=3)
clf.fit(X_train,y_train)
predictions = clf.predict(X_test)
```

```
acc = np.sum(predictions == y_test)/len(y_test)
print(acc)
```

```
0.9
```

After the model is trained with 80% of data we got 90% accuracy.

Reference:

[1] https://hub.uvcemarvel.in/

[2] www.youtube.com

[3] https://scikit-learn.org/

[4] https://towardsdatascience.com/

[5] www.medium.com

[6] www.educative.io/