**AIM:- Demonstrate the following data preprocessing tasks using python libraries.**

A)Loading the dataset.

**DESC:-**

Data loading is the process of copying and loading data or data sets from a source file, folder or application to a database or similar application. It is usually implemented by copying digital data from a source and pasting or loading the data to a data storage or processing utility.

**PROGRAM:-**

from sklearn import datasets

from sklearn.datasets import fetch\_california\_housing

df=fetch\_california\_housing()

x=df.data

print(x)

**OUTPUT:-**



**Result:-** Program Executed Successfully.

B)Identifying the dependent and independent variables.

**DESC:-**

Independent variables (also referred to as Features) are the input for a process that is being analyzes. Dependent variables are the output of the process.

For example, in the below data set, the independent variables are the input of the purchasing process being analyzed. The result (whether a user purchased or not) is the dependent variable

**PROGRAM:-**

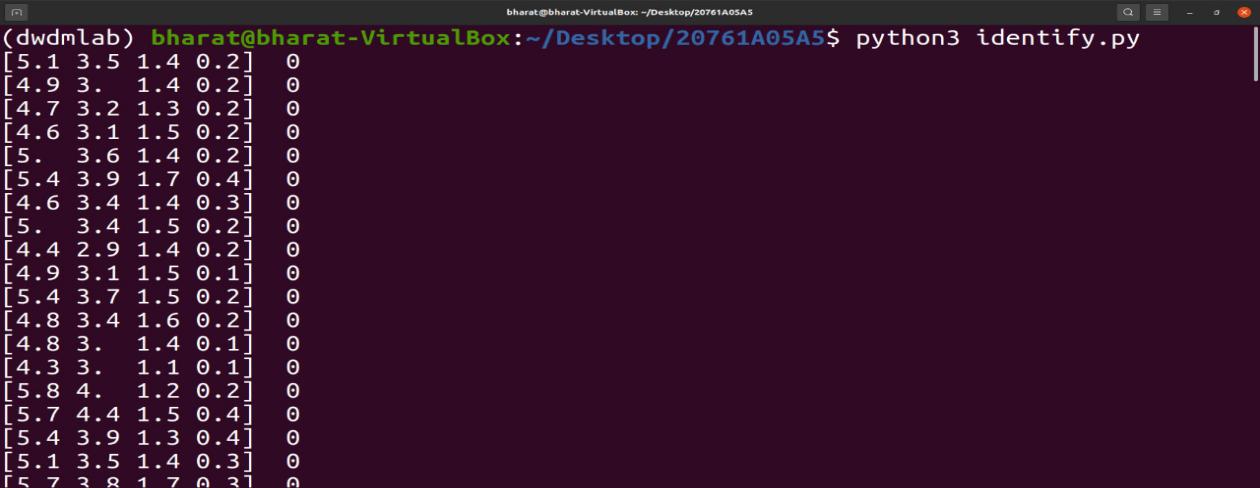
from sklearn.datasets import load\_iris

i=load\_iris()

X,Y=i.data,i.target

for i in range(0,len(X)):

print(X[i],"",Y[i])

**Output:-**

**Result:-** Program Executed Successfully.

C)Dealing with missing data

**DESC:-**

Missing data is defined as the values or data that is not stored (or not present) for some variable/s in the given dataset.

Below is a sample of the missing data from the Titanic dataset. You can see the columns ‘Age’ and ‘Cabin’ have some missing values.

**PROGRAM:-**

import numpy as np

import pandas as pd

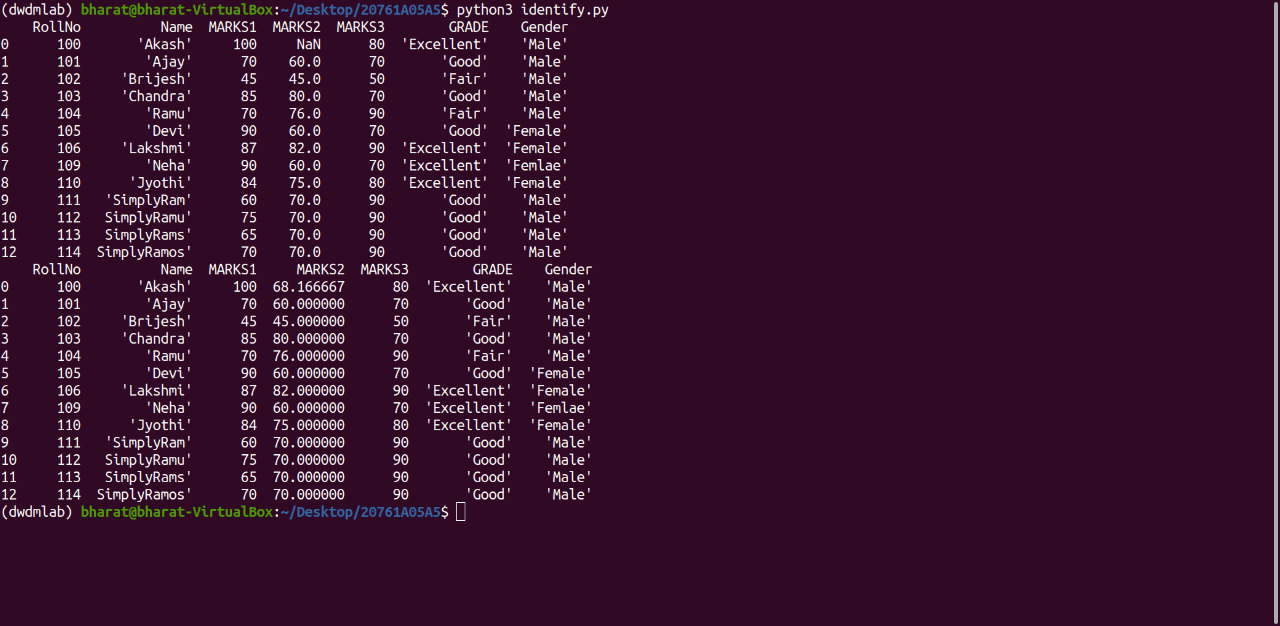
df=pd.read\_csv('stu.csv')

print(df)

df['MARKS2']=df['MARKS2'].fillna(df['MARKS2'].mean())

print(df)

**OUTPUT:-**



**Result:-** Program Executed Successfully.

**AIM:- Demonstrate the following data preprocessing tasks using python libraries.**

A)Dealing with categorical data.

**DESC:-**

Categorical data is the statistical data comprising categorical variables of data that are converted into categories. One of the examples is a grouped data. More precisely, categorical data could be derived from qualitative data analysis that are countable, or from quantitative data analysis grouped within given intervals. These data are summarised in the form of a probability table. However, when we consider data analysis, it is referred to use the term “categorical data”, which is applied to data sets. Also, it is to be noted that, while containing some categorical variables, the data set may also contain non-categorical variables.

**PROGRAM:**-

#using LabelEncoder

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

df=pd.read\_csv('stu.csv')

print(df['Name'])

a=LabelEncoder()

df['Name']=a.fit\_transform(df['Name'])

print(df['Name'])

**OUTPUT:-**

**PROGRAM:-**

#Using LabelBinarizer

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelBinarizer

df=pd.read\_csv('stu.csv')

print(df['Name'])

a=LabelBinarizer()

df=a.fit\_transform(df['Name'])

print(df)

**OUTPUT:-**



**PROGRAM:-**

**#**Using asType,cat.Codes

import numpy as np

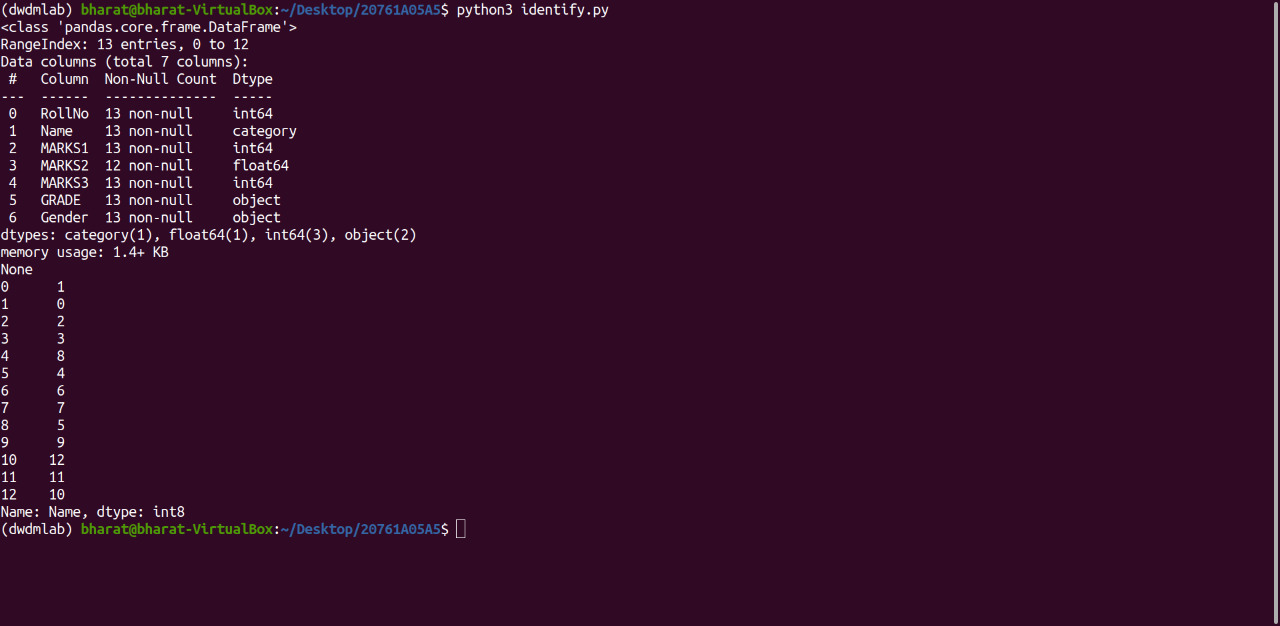
import pandas as pd

df=pd.read\_csv('stu.csv')

df['Name']=df['Name'].astype('category')

print(df.info())

df['Name']=df['Name'].cat.codes

print(df['Name'])

**OUTPUT:-**

**Result:-** Program Executed Successfully.

B)Scaling the features

**PROGRAM:-**

import pandas as pd

from sklearn.preprocessing import MinMaxScaler,StandardScaler

df = pd.read\_csv('iris.csv')

x = df.iloc[:, 1:3].values

min\_max = MinMaxScaler(feature\_range =(0, 1))

min\_max1=StandardScaler()

x\_after\_min\_max = min\_max.fit\_transform(x)

x\_after\_min\_max1=min\_max1.fit\_transform(x)

print("Min Max Scaler output is\n", x\_after\_min\_max)

print("Standard Scaler output is\n",x\_after\_min\_max1)

**OUTPUT:-**

**Result:-** Program Executed Successfully.

C)Splitting dataset into Training and Testing Sets

**DESC:-**

Training Data

The observations in the training set form the experience that the algorithm uses to learn. In supervised learning problems, each observation consists of an observed output variable and one or more observed input variables.

Test Data

The test set is a set of observations used to evaluate the performance of the model using some performance metric. It is important that no observations from the training set are included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it.

**PROGRAM:-**

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

i=load\_iris()

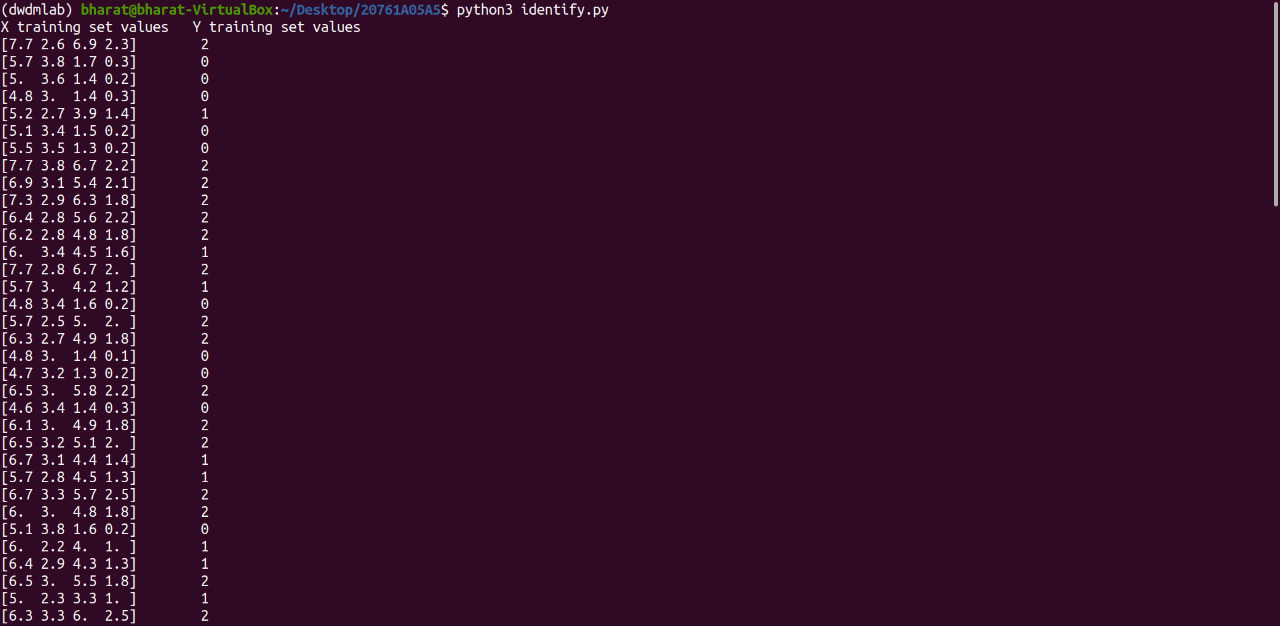
X,Y=i.data,i.target

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(X,Y,test\_size=0.3,random\_state=1)

print("X training set values\tY training set values")

for i in range(0,len(X\_train)):

print(X\_train[i],'\t',Y\_train[i])

**OUTPUT:-**

**Result:-** Program Executed Successfully.

AIM:-Demonstrate the following Similarity and Dissimilarity Measures using python

**A)Pearson’s Correlation**

**DESC:-**

The Pearson's Correlation Coefficient is also known as the Pearson Product-Moment Correlation Coefficient. It is a measure of the linear relationship between two random variables - ****X**** and ****Y****. Mathematically, if (****σXY****) is the covariance between ****X**** and ****Y****, and (****σX****) is the standard deviation of ****X****, then the Pearson's correlation coefficient ****ρ**** is given by:

ρX,Y=σXY/σXσY

**PROGRAM:-**

from scipy.stats import pearsonr

X=[-2,-1,0,1,2]

Y=[4,1,3,2,0]

corr=pearsonr(X,Y)

print("pearson correlation:",corr)

B)Cosine Similarity

**DESC:-**

Cosine distance measure for clustering determines the **cosine** of the angle between two vectors given by the following formula.

Rendered by QuickLaTeX.com  
Here (**theta**) gives the angle between two vectors and A, B are n-dimensional vectors.

**PROGRAM:-**

from sklearn.feature\_extraction.text import CountVectorizer

import pandas as pd

from sklearn.metrics.pairwise import cosine\_similarity

A="deep learning can be hard"

B="deep learning can be soft"

documents=[A,B]

ob=CountVectorizer()

X=ob.fit\_transform(documents)

Y=X.todense()

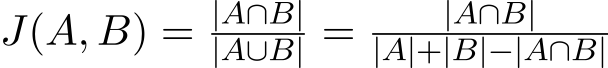
df=pd.DataFrame(Y,columns=ob.get\_feature\_names\_out(),index=['A','B'])

print(df)

print("similarity matrix:\n",cosine\_similarity(df,df))

C)Jaccard Similarity

**DESC:-**

The Jaccard distance measures the similarity of the two data set items as the **intersection** of those items divided by the **union** of the data items.

**PROGRAM:-**

import numpy as np

from scipy.spatial.distance import jaccard

a=np.array([1,0,1,0,0,1])

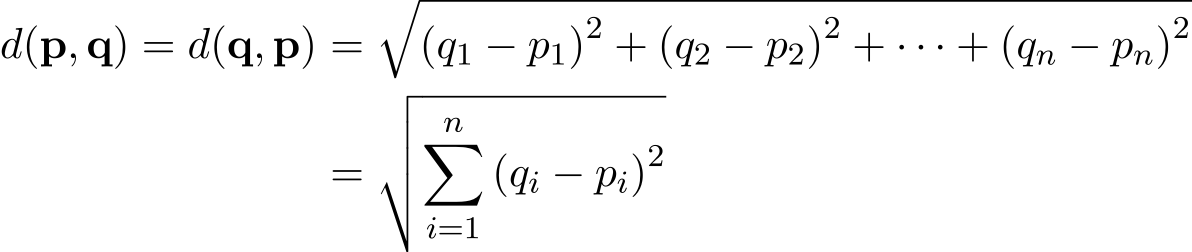
b=np.array([0,1,0,1,0,1])

print(" jaccard distance:",jaccard(a,b))

D)Euclidean Distance

**DESC:-**

Euclidean distance is considered the traditional metric for problems with geometry. It can be simply explained as the **ordinary distance** between two points. It is one of the most used algorithms in the cluster analysis. One of the algorithms that use this formula would be **K-mean**. Mathematically it computes the **root of squared differences** between the coordinates between two objects.



**PROGRAM:-**

import numpy as np

import pandas as pd

from sklearn.metrics.pairwise import manhattan\_distances

X=np.ones((1,2))

Y=np.full((2,2),2)

manhattan\_distances(X,Y,sum\_over\_features=False)

E)Manhattan Distance

**DESC:-**

This determines the absolute difference among the pair of the coordinates.

Suppose we have two points P and Q to determine the distance between these points we simply have to calculate the perpendicular distance of the points from X-Axis and Y-Axis.  
In a plane with P at coordinate (x1, y1) and Q at (x2, y2).

Manhattan distance between P and Q = |x1 – x2| + |y1 – y2|

**PROGRAM:-**

from sklearn.metrics.pairwise import euclidean\_distances

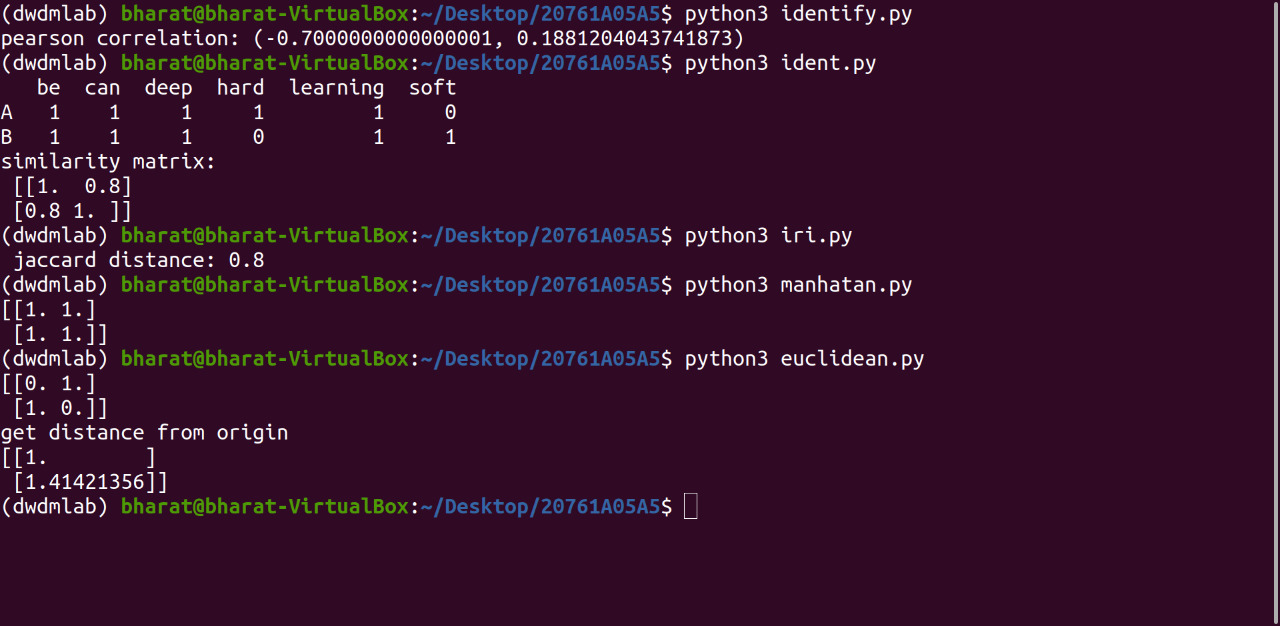
X=[[0,1],[1,1]]

print(euclidean\_distances(X,X))

print("get distance from origin")

print(euclidean\_distances(X,[[0,0]]))

**OUTPUT:-**

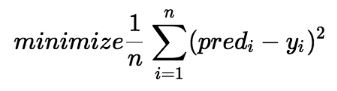
****

**Result:-** Program Executed Successfully.

AIM:-Build a model using linear regression algorithm on any dataset.

**DESC:-**

**Linear Regression** is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables gettingused.

**Hypothesis function for Linear Regression :**  


**PROGRAM:-**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

x=np.array([5,10,15,20,25,30]).reshape((-1,1))

y=np.array([5,20,14,32,22,38])

print(x,y)

model=LinearRegression()

model.fit(x,y)

result=model.score(x,y)

print("Score:",result)

print("Intercret:",model.intercept\_)

print("slope:",model.coef\_)

y\_pred=model.predict(x)

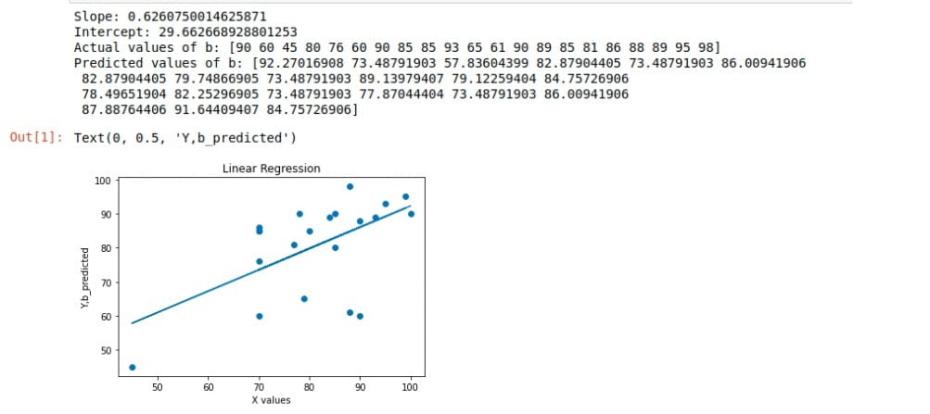
print('Actual values of y:',y)

print('predicted values of y:',y\_pred)

plt.scatter(x,y,color='black')

plt.plot(x,y\_pred,color='blue',linewidth=2,marker='o',markerfacecolor='green')

**OUTPUT:-**



**(without using builtin functions)**

**PROGRAM:-**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

n=pd.read\_csv('datastudent.csv')

a=np.array(n['MARKS1'])

b=np.array(n['MARKS2'])

n=np.size(a)

m\_a=np.mean(a)

m\_b=np.mean(b)

SS\_xx=np.sum((m\_a-a)\*(m\_a-a))

SS\_xy=np.sum((m\_a-a)\*(m\_b-b))

slope=SS\_xy/SS\_xx

print("Slope:",slope)

intercept=m\_b-(slope\*m\_a)print("Intercept:",intercept)

b\_pred=slope\*a+intercept

print("Actual values of b:",b)

print("Predicted values of b:",b\_pred)

plt.scatter(a,b)

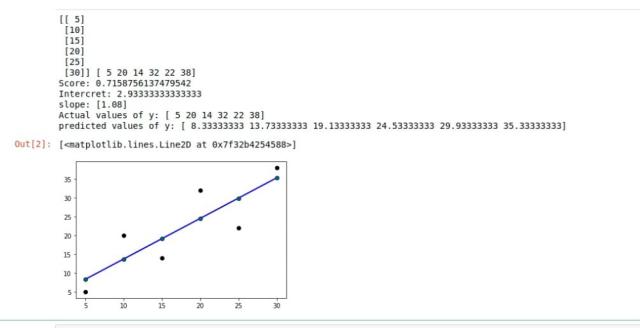
plt.plot(a,b\_pred)

plt.title("Linear Regression")

plt.xlabel("X values")

plt.ylabel("Y,b\_predicted")

**OUTPUT:-**



**Result:-** Program Executed Successfully.

AIM:-Build a classification model using Decision Tree algorithm on iris dataset

**DESC:-**

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

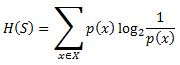
1. ****Classification trees**** (Yes/No types)

What we’ve seen above is an example of classification tree, where the outcome was a variable like ‘fit’ or ‘unfit’. Here the decision variable is ****Categorical****.

1. ****Regression trees**** (Continuous data types)

Here the decision or the outcome variable is ****Continuous****, e.g. a number like 123. ****Working**** Now that we know what a Decision Tree is, we’ll see how it works internally. There are many algorithms out there which construct Decision Trees, but one of the best is called as ****ID3 Algorithm****. ID3 Stands for ****Iterative Dichotomiser 3****. Before discussing the ID3 algorithm, we’ll go through few definitions.

* ****Entropy:****

Entropy, also called as Shannon Entropy is denoted by H(S) for a finite set S, is the measure of the amount of uncertainty or randomness in data. [](https://www.xoriant.com/sites/default/files/uploads/2017/08/Decision-Trees-modified-2.jpg) Intuitively, it tells us about the predictability of a certain event. Example, consider a coin toss whose probability of heads is 0.5 and probability of tails is 0.5. Here the entropy is the highest possible, since there’s no way of determining what the outcome might be. Alternatively, consider a coin which has heads on both the sides, the entropy of such an event can be predicted perfectly since we know beforehand that it’ll always be heads. In other words, this event has ****no randomness**** hence it’s entropy is zero. In particular, lower values imply less uncertainty while higher values imply high uncertainty.

* ****Information Gain:****

nformation gain is also called as Kullback-Leibler divergence denoted by IG(S,A) for a set S is the effective change in entropy after deciding on a particular attribute A. It measures the relative change in entropy with respect to the independent variables. [IMG_257](https://www.xoriant.com/sites/default/files/uploads/2017/08/Decision-Trees-modified-3.jpg) Alternatively, [IMG_258](https://www.xoriant.com/sites/default/files/uploads/2017/08/Decision-Trees-modified-4.jpg) where IG(S, A) is the information gain by applying feature A. H(S) is the Entropy of the entire set, while the second term calculates the Entropy after applying the feature A, where P(x) is the probability of event x.

**PROGRAM:-**

from sklearn.datasets import load\_iris

from sklearn import tree

import graphviz

iris=load\_iris()

X,y=iris.data,iris.target

clf=tree.DecisionTreeClassifier()

clf=clf.fit(X,y)

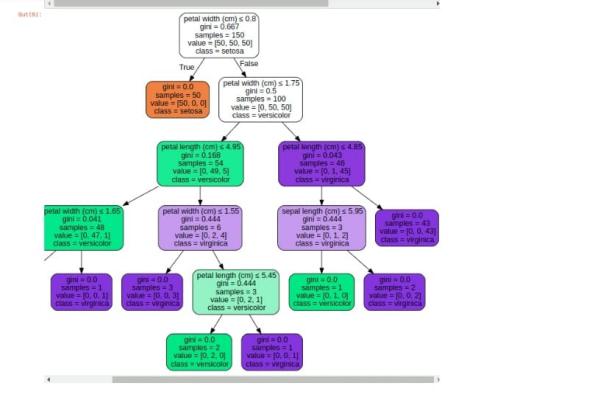
dot\_data=tree.export\_graphviz(clf,out\_file=None,feature\_names=iris.feature\_names,class\_names=iris.target\_n ames,filled=True,rounded=True,special\_characters=True)

graph=graphviz.Source(dot\_data)

graph.render("iris")

Graph

**OUTPUT:-**



**Result:-** Program Executed Successfully.

AIM:-Apply Naïve Bayes Classification algorithm on any dataset

**DESC:-**

* Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
* IMG_256****It is a probabilistic classifier, which means it predicts on the basis of the probability of an object****.

****Where,****

****P(A|B) is Posterior probability****: Probability of hypothesis A on the observed event B.

****P(B|A) is Likelihood probability****: Probability of the evidence given that the probability of a hypothesis is true.

****P(A) is Prior Probability****: Probability of hypothesis before observing the evidence.

****P(B) is Marginal Probability****: Probability of Evidence.

**PROGRAM:-**

# importing required libraries

import pandas as pd

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

# read the train and test dataset

train\_data = pd.read\_csv('train-data.csv')

test\_data = pd.read\_csv('test-data.csv')

# shape of the dataset

print('Shape of training data :',train\_data.shape)

print('Shape of testing data :',test\_data.shape)

# Now, we need to predict the missing target variable in the test data

# target variable - Survived

# seperate the independent and target variable on training data

train\_x = train\_data.drop(columns=['Survived'],axis=1)

train\_y = train\_data['Survived']

# seperate the independent and target variable on testing data

test\_x = test\_data.drop(columns=['Survived'],axis=1)

test\_y = test\_data['Survived']

'''

Create the object of the Naive Bayes model

You can also add other parameters and test your code here

Some parameters are : var\_smoothing

Documentation of sklearn GaussianNB:

https://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.GaussianNB.html

'''

model = GaussianNB()

# fit the model with the training data

model.fit(train\_x,train\_y)

# predict the target on the train dataset

predict\_train = model.predict(train\_x)

print('Target on train data',predict\_train)

# Accuray Score on train dataset

accuracy\_train = accuracy\_score(train\_y,predict\_train)

print('accuracy\_score on train dataset : ', accuracy\_train)

# predict the target on the test dataset

predict\_test = model.predict(test\_x)

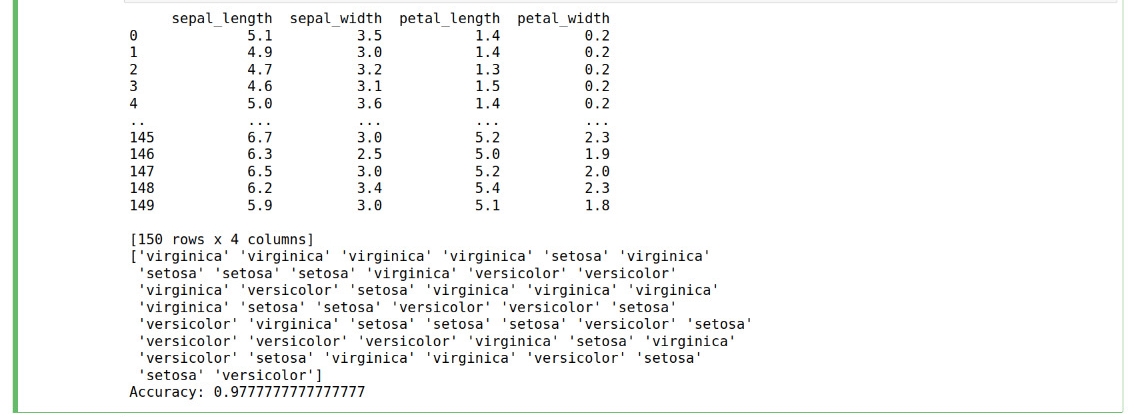
print('Target on test data',predict\_test)

# Accuracy Score on test dataset

accuracy\_test = accuracy\_score(test\_y,predict\_test)

print('accuracy\_score on test dataset : ', accuracy\_test)

**OUTPUT:-**



**Result:-** Program Executed Successfully.

AIM:-Generate frequent itemsets using Apriori Algorithm in python and also generate **association rules for any market basket data.**

**DESC:-**

The key concept in the Apriori algorithm is that it assumes all subsets of a frequent itemset to be frequent. Similarly, for any infrequent itemset, all its supersets must also be infrequent.****Grab high-paying analytics jobs with the help of these Top**[Data Science Interview Questions](https://intellipaat.com/blog/interview-question/data-science-interview-questions/" \t "https://intellipaat.com/blog/data-science-apriori-algorithm/_blank)**!****Let us try and understand the working of an Apriori algorithm with the help of a very famous business scenario, market basket analysis.

In order to select the interesting rules out of multiple possible rules from this small business scenario, we will be using the following measures:

* IMG_256****Support****

IMG_256

* ****Confidence****
* ****List****
* IMG_256****Conviction****

conv(x => y) =

**PROGRAM:-**

import pandas as pd

from apyori import apriori

st\_df=pd.read\_csv("Market\_Basket\_Optimisation.csv",header=None)

st\_df

#converting data frames into list of lists

l=[]

for i in range(1,7501):

l.append([str(st\_df.values[i,j]) for j in range(0,20)])

#Applying apriori algorithm

association\_rules = apriori(l,min\_support=0.0045, min\_confidence=0.2 ,min\_lift=3 ,min\_length=2)

association\_results = list(association\_rules)

for i in range(0, len(association\_results)):

print(association\_results[i][0])

for item in association\_results:

#first index of the inner list

#contains base item and add item

pair= item[0]

items= [x for x in pair]

print("Rule: " +items[0] + "->"+ items[1] )

#second index of the inner list

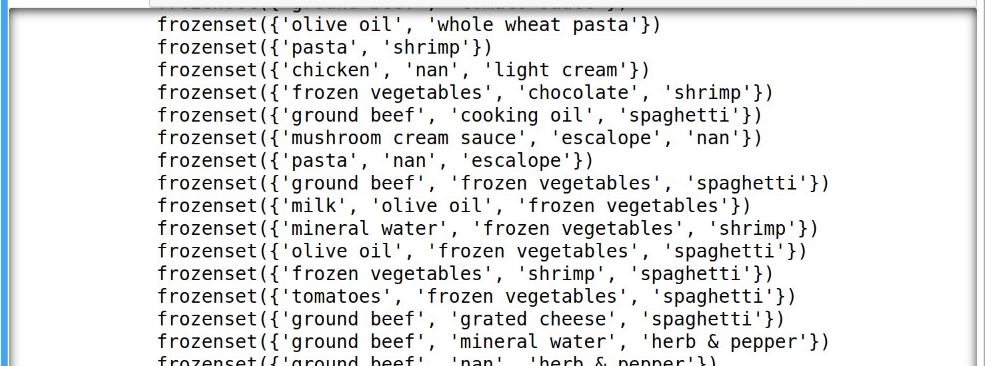
print("Support: "+ str(item[1]))

#third index of the list located at 0th position of the third index of the inner list

print("Confidence: "+str(item[2][0][2]))

print("Lift: " +str(item[2][0][3]))

print("--------------------------------------------------------------")



**OUTPUT:-**



**Result:-** Program Executed Successfully.

AIM:-Apply K- Means clustering algorithm on any dataset.

**DESC:-**

The working of the K-Means algorithm is explained in the below steps:

****Step-1:**** Select the number K to decide the number of clusters.

****Step-2:**** Select random K points or centroids. (It can be other from the input dataset).

****Step-3:**** Assign each data point to their closest centroid, which will form the predefined K clusters.

****Step-4:**** Calculate the variance and place a new centroid of each cluster.

****Step-5:**** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

****Step-6:**** If any reassignment occurs, then go to step-4 else go to FINISH.

****Step-7****: The model is ready.

**WCSS= ∑Pi in Cluster1 distance(Pi C1)2 +∑Pi in Cluster2distance(Pi C2)2+∑Pi in CLuster3 distance(Pi C3)2**

**Program:-**

import matplotlib.pyplot as plt

import numpy as np

from sklearn.cluster import KMeans

X=np.array([[1,1],[1.5,2],[3,4],[5,7],[3.5,5],[4.5,5],[3.5,4.5]])

print(X)

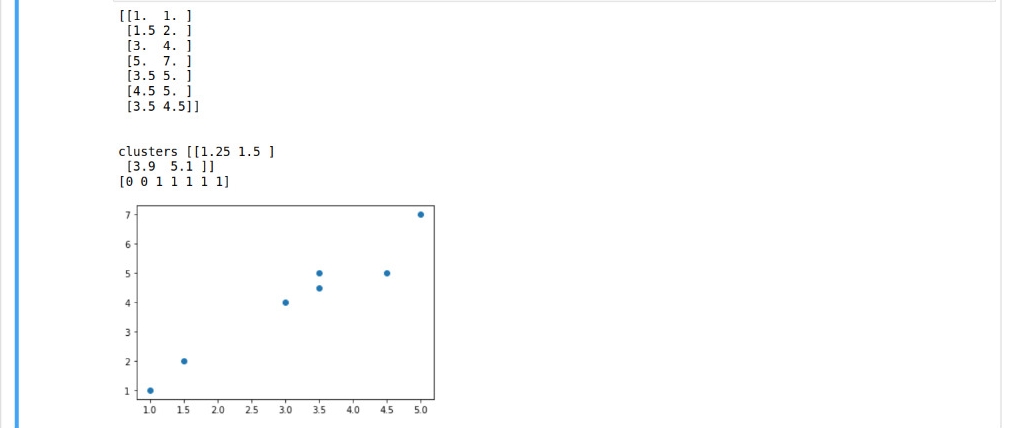
KMeans=KMeans(n\_clusters=2)

KMeans.fit(X)

plt.scatter(X[:,0],X[:,1])

print('\n\nclusters',KMeans.cluster\_centers\_)

print(KMeans.labels\_)

**Output:-**

**Result:-** Program Executed Successfully.

**AIM:-Apply Hierarchical Clustering algorithm on any dataset.**

**DESC:-**

**Following are the steps involved in agglomerative clustering:**

1. At the start, treat each data point as one cluster. Therefore, the number of clusters at the start will be K, while K is an integer representing the number of data points.
2. Form a cluster by joining the two closest data points resulting in K-1 clusters.
3. Form more clusters by joining the two closest clusters resulting in K-2 clusters.
4. Repeat the above three steps until one big cluster is formed.
5. Once single cluster is formed, [dendrograms](https://en.wikipedia.org/wiki/Dendrogram" \t "https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/_blank) are used to divide into multiple clusters depending upon the problem. We will study the concept of dendrogram in detail in an upcoming section.

There are different ways to find distance between the clusters. The distance itself can be Euclidean or Manhattan distance. Following are some of the options to measure distance between two clusters:

1. Measure the distance between the closes points of two clusters.
2. Measure the distance between the farthest points of two clusters.
3. Measure the distance between the centroids of two clusters.
4. Measure the distance between all possible combination of points between the two clusters and take the mean.

**PROGRAM:-**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import scipy.cluster.hierarchy as shc

from scipy.spatial.distance import squareform,pdist

a=[0.07,0.85,0.66,0.49,0.80]

b=[0.83,0.14,0.89,0.64,0.46]

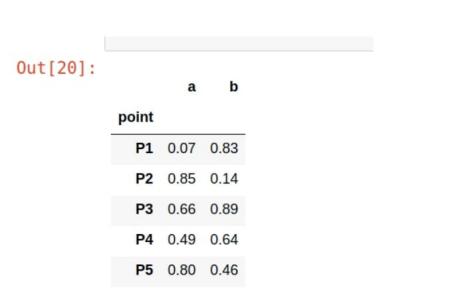
point=['P1','P2','P3','P4','P5']

data=pd.DataFrame({'point':point,'a':np.round(a,2),'b':np.round(b,2)})

data=data.set\_index('point')

data

**OUTPUT:-**

/

**Result:-** Program Executed Successfully.

**AIM:-Apply DBSCAN clustering algorithm on any dataset.**

**DESC:-**

The following are the DBSCAN clustering algorithmic steps:

* ****Step 1:**** Initially, the algorithms start by selecting a point (x) randomly from the data set and finding all the neighbor points within *Eps* from it. If the number of *Eps-neighbours* is greater than or equal to ****MinPoints****, we consider x a core point. Then, with its *Eps-neighbours*, x forms the first cluster.After creating the first cluster, we examine all its member points and find their respective *Eps -neighbors*. If a member has at least *MinPoints* *Eps-neighbours*, we expand the initial cluster by adding those *Eps-neighbours* to the cluster. This continues until there are no more points to add to this cluster.

****Step 2:**** For any other core point not assigned to cluster, create a new cluster.

****Step 3:**** To the core point cluster, find and assign all points that are recursively connected to it.

****Step 4:**** Iterate through all unattended points in the dataset and assign them to the nearest cluster at *Eps* distance from themselves. If a point does not fit any available clusters, locate it as a noise point.

**PROGRAM:-**

import numpy as np

from sklearn.datasets import make\_blobs#generating a sequence of numbers

from sklearn.preprocessing import StandardScaler#preprocessing technique used to scale the graph

import matplotlib.pyplot as plt

from sklearn.cluster import DBSCAN

centers=[[0.5,2],[-1,-1],[1.5,-1]]

#scaling can be done by StandardScaler

#create dataset

X,y=make\_blobs(n\_samples=100, centers=centers, cluster\_std=0.5,random\_state=0)

#print(X,y)

X=StandardScaler().fit\_transform(X)

print(X,y)

db=DBSCAN(eps=0.4,min\_samples=5)

db.fit(X)

labels=db.labels\_

n\_clusters\_=len(set(labels))-(1 if -1 in labels else 0)

print("Estimated number of clusters:%d" %n\_clusters\_)

y\_pred=db.fit\_predict(X)

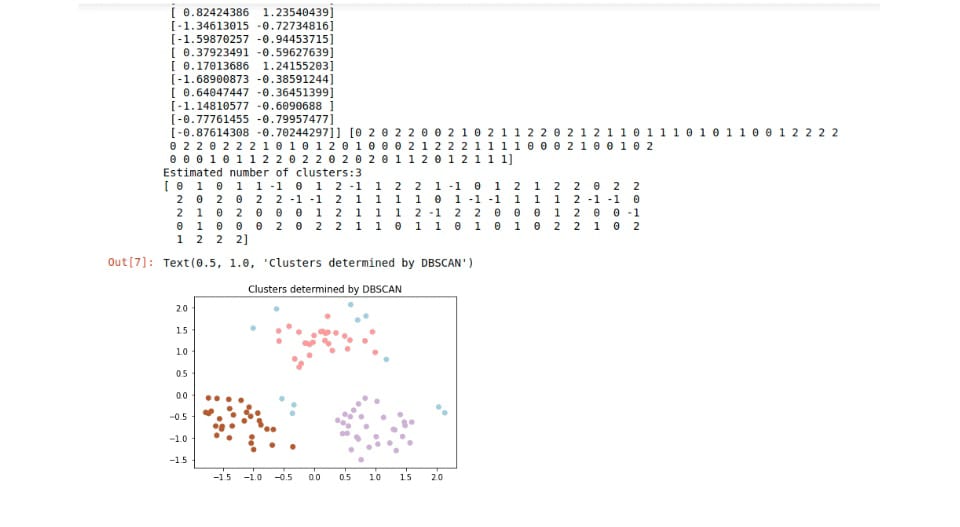
print(db.labels\_)

plt.figure(figsize=(6,4))

plt.scatter(X[:,0],X[:,1],c=y\_pred ,cmap='Paired')

plt.title("Clusters determined by DBSCAN")

**OUTPUT:-**



**Result:-** Program Executed Successfully.