AD18413 APPLIED MACHINE LEARNING LABORATORY

EXP NO: 1 **PACKAGES**
DATE: 10/02/2023

a) NUMPY:

i) Fibonacci Series using Binet Formula

AIM: To print Fibonacci Series using Binet Formula.

FORMULA:

$$f(n) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

ALGORITHM:

1. Start
2. Get the number of terms ‘n’ from the user
3. Generate an array from numbers 1 to n
4. Using the Binet’s formula calculate the Fibonacci series for every number in the array
5. Stop

PROGRAM:

```
import numpy as np
n =int(input(("Enter the no.of terms:")))
a =np.arange(1,n+1)
sq5=np.sqrt(5)
f=(((1 + sq5)/2)**a-((1-sq5)/2)**a)/sq5
print("Fibonacci series of first",n,"terms are:",f)
```

OUTPUT:

```
Enter the no.of terms:5
Fibonacci series of first 5 terms are: [1. 1. 2. 3. 5.]
```

RESULT:
The python program to print first n terms of Fibonacci Series using Binet Formula was executed successfully.

ii) Array rotation by reversing the elements

AIM: To rotate an array by reversing the elements using numpy package.

ALGORITHM:

1. Start
2. Get the number of elements from the user
3. Get the elements from the user
4. Change the list into an nd array
5. Get the number of rotations ‘r’ from the user
6. Using indexing, reverse the elements from index 0 to r
7. Using indexing, reverse the elements from index r to -1
8. Reverse the list using indexing
9. Stop

PROGRAM:

```
import numpy as np
n =int(input(("Enter the no.of terms:")))
a =[int(input("enter the elements:"))for i in range(n)]
arr=np.array(a)
r=int(input("Enter the no.of rotation:"))
arr[:r]=arr[:r][::-1]
arr[:r]=arr[:r][::-1]
arr=arr[::-1]
print(arr)
```

OUTPUT:

```
Enter the no.of terms:5
enter the elements:10
enter the elements:20
enter the elements:30
enter the elements:40
enter the elements:50
Enter the no.of rotation:3
[50 40 30 20 10]
```

RESULT:
The python program to rotate & reverse array element using numpy package was executed successfully.

b) PANDAS:

i) Consider the following data and execute the following.

	School	Class	Name	Age
S1	S001	V	X	12
S2	S002	V	Y	12
S3	S003	VI	Z	13
S4	S001	VI	A	13
S5	S002	V	B	14
S6	S004	VI	C	12

a) Write a program to create a dataframe for given dataset.
b) Split the dataframe into two groups based on school code and display its size.
c) Split the dataframe into two groups based on school code and class and display its size.

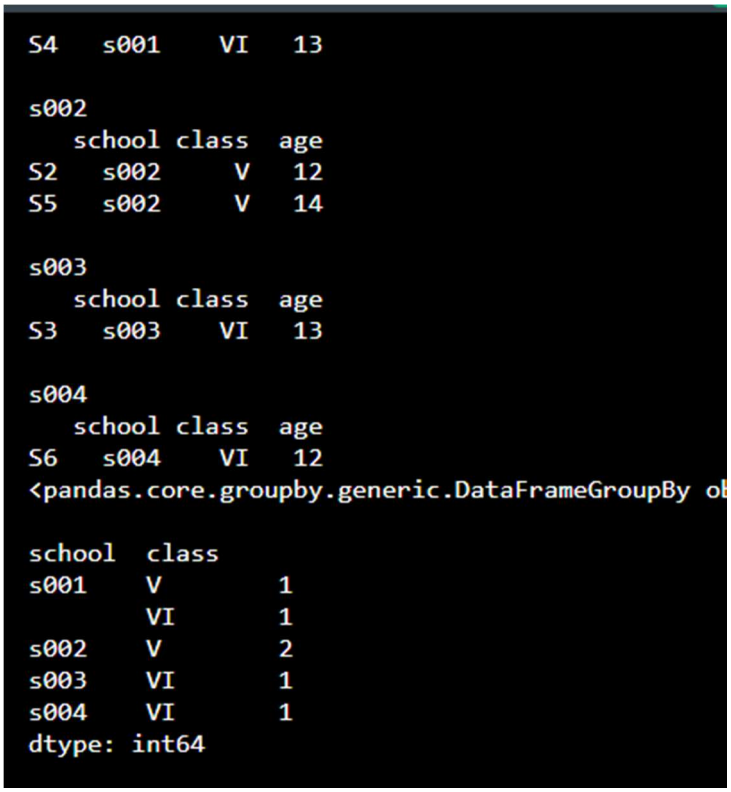
PROGRAM:

```
import pandas as pd
pd.set_option('display.max_rows', None)
data = pd.DataFrame({
   'school': ['s001','s002','s003','s001','s002','s004'],
   'class': ['V', 'V', 'VI', 'VI', 'V', 'VI'],
   'age': [12, 12, 13, 13, 14, 12],},
   index=['S1', 'S2', 'S3', 'S4', 'S5', 'S6'])

print("Original DataFrame:")
print(data)
print('\nSplit  data :')
result = data.groupby(['school'])
for name,group in result:
   print()
   print(name)
   print(group)
print(result)
print()
print(result.size())

a=data.groupby(['school','class'])
for name,group in result:
   print()
   print(name)
   print(group)
print(a)
print()
print(a.size())
```

OUTPUT:

```
Original DataFrame:
   school class  age
S1   s001     V   12
S2   s002     V   12
S3   s003    VI   13
S4   s001    VI   13
S5   s002     V   14
S6   s004    VI   12

Split  data :

s001
   school class  age
S1   s001     V   12
S4   s001    VI   13

s002
   school class  age
S2   s002     V   12
S5   s002     V   14
```

```
s003
   school class  age
S3   s003    VI   13

s004
   school class  age
S6   s004    VI   12
<pandas.core.groupby.generic.Data

school
s001    2
s002    2
s003    1
s004    1
dtype: int64

s001
   school class  age
S1   s001     V   12
S4   s001    VI   13
```



RESULT:

The python program to create and group data frame using pandas package was executed successfully.

iii) Consider the following data and execute the following.

Order_no	Amount	Cust_id
1	150	5
9	270	1
2	65	2
4	110	9
7	948	5
5	2400	7
8	5760	2
10	1983	4
3	2080	9
12	250	8
11	75	3
13	3045	2

a) Create a dataframe for given dataset.
b) Find the mean, max, min of purchase amount group by cust_id.
c) Split the dataframe into groups based on cust_id and display the count of each.

PROGRAM:

```
import pandas as pd
data = pd.DataFrame({'order':[1,9,2,4,7,5,8,10,3,12,11,13],
    'amount': [150,270,65,110,948,2400,5760,1983,2480,250,75,3045],
    'cust_id': [5,1,2,9,5,7,2,4,9,8,3,2]})
a = data.groupby('cust_id').agg({'amount':['mean','min','max']})
print(a)
print( )
b = data.groupby('cust_id').nunique()
print(b)
```

OUTPUT:

```
              amount
                mean   min   max
cust_id
1          270.000000   270   270
2         2956.666667    65  5760
3           75.000000    75    75
4         1983.000000  1983  1983
5          549.000000   150   948
7         2400.000000  2400  2400
8          250.000000   250   250
9         1295.000000   110  2480

          order  amount  cust_id
cust_id
1             1       1        1
2             3       3        1
3             1       1        1
4             1       1        1
5             2       2        1
7             1       1        1
8             1       1        1
9             2       2        1
```

RESULT:

The python program to create and group data frame using pandas package was executed successfully.

c) Matplotlib

Read the CSV file.

PROGRAM:

```
import pandas as pd
cars = pd.read_csv("/home/user/Downloads/Toyota.csv",index_col=0,
na_values=(["??","????","###"]))
cars.dropna(axis=0,inplace=True)
print(cars)
```

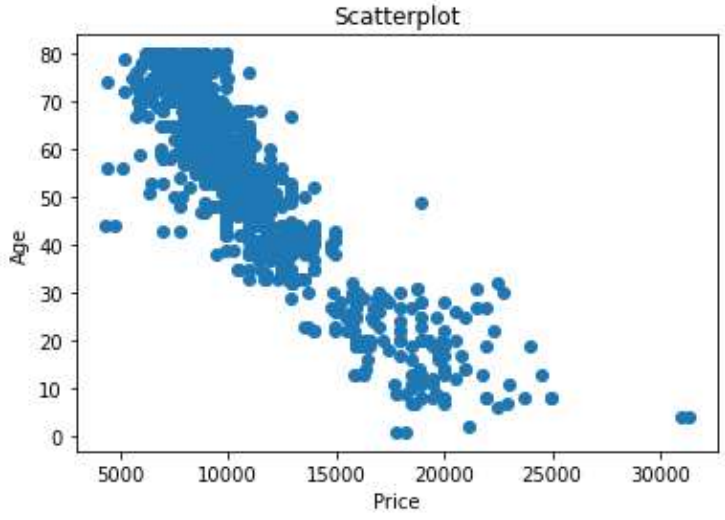
OUTPUT:

```
      Price   Age      KM FuelType  ... Automatic    CC  Doors  Weight
0     13500  23.0  46986.0   Diesel  ...         0  2000  three    1165
1     13750  23.0  72937.0   Diesel  ...         0  2000      3    1165
3     14950  26.0  48000.0   Diesel  ...         0  2000      3    1165
4     13750  30.0  38500.0   Diesel  ...         0  2000      3    1170
5     12950  32.0  61000.0   Diesel  ...         0  2000      3    1170
...     ...   ...      ...      ...  ...       ...   ...    ...     ...
1423   7950  80.0  35821.0   Petrol  ...         1  1300      3    1015
1424   7750  73.0  34717.0   Petrol  ...         0  1300      3    1015
1429   8950  78.0  24000.0   Petrol  ...         1  1300      5    1065
1430   8450  80.0  23000.0   Petrol  ...         0  1300      3    1015
1435   6950  76.0      1.0   Petrol  ...         0  1600      5    1114

[1096 rows x 10 columns]
```

Construct a scatterplot between age and price.

```
import matplotlib.pyplot as plt
plt.scatter(cars['Price'], cars['Age'])
plt.title('Scatterplot')
plt.xlabel('Price')
plt.ylabel('Age')
plt.show()
```

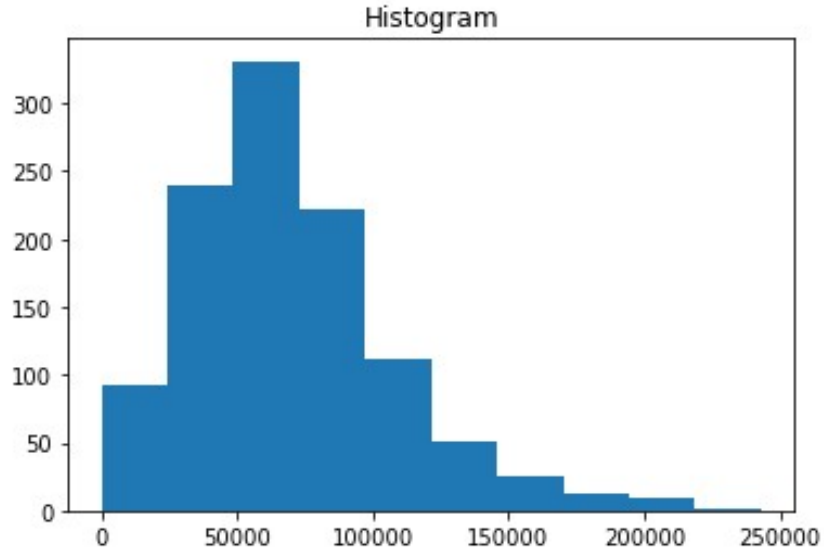
OUTPUT:



Construct a histogram for kilometer.

```
plt.hist(cars['KM'])
plt.title("Histogram")
plt.show()
```

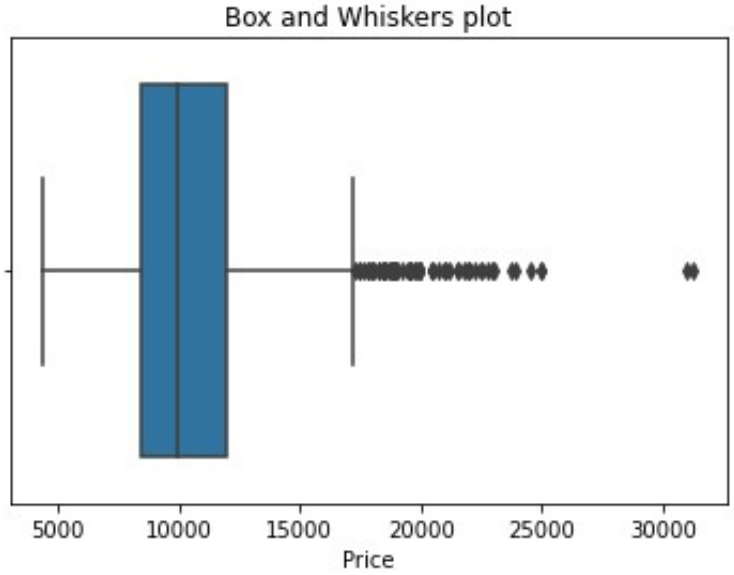
OUTPUT:



Construct a box and whiskers plot for the parameter price.

```
import seaborn as sns
sns.boxplot(cars['Price'])
plt.title("Box and Whiskers plot")
sns.show()
```

OUTPUT:

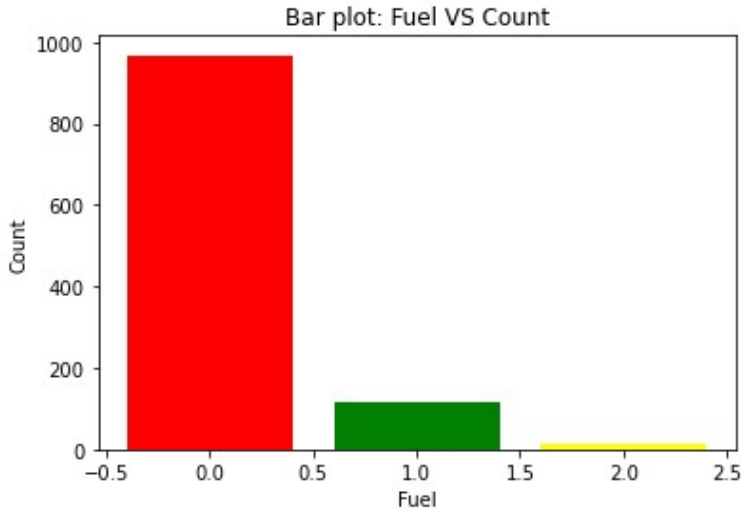


Construct a bar plot for the attribute fuel.

```
import numpy as np
c = cars['FuelType'].value_counts()
count = [968,116,12]
Fuel = ['Petrol','Diesel','CNG']
index = np.arange(len(Fuel))
plt.bar(index,count,color=['red','green','yellow'])
```

```
plt.title("Bar plot: Fuel VS Count")
plt.xlabel('Fuel')
plt.ylabel('Count')
plt.show()
```

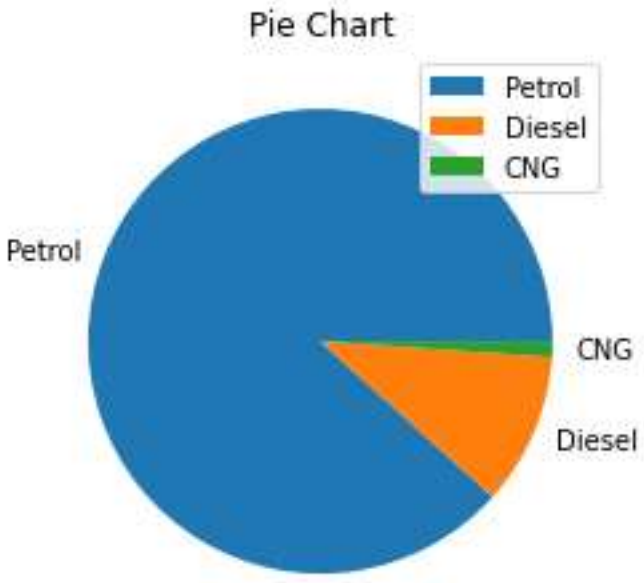
OUTPUT:



Construct a pie chart for the attribute fuel.

```
plt.pie(count,labels=Fuel)
plt.title("Pie chart")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

The python program to read the given csv file and to visualize the relation between the following attributes using Matplotlib was executed successfully

EXP NO: 2 **DATA EXPLORATION AND PROCESSING**
DATE: 17/02/2023

AIM: To read the given dataset and to explore & pre-process the data.

QUERIES:
Read the Dataset:

```
import pandas as pd
cars = pd.read_csv("Toyota.csv",index_col=0)
```

i) Get Index of the dataset:

```
print(cars.index)
Int64Index([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
 ...
 1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435],
 dtype='int64', length=1436)
```

ii) Get the column labels of the DataFrame:

```
print(cars.columns)
Index(['Price', 'Age', 'KM', 'FuelType', 'HP', 'MetColor', 'Automatic', 'CC',
 'Doors', 'Weight'],
 dtype='object')
```

iii) Get the total No. Of. Elements from the DataFrame:

```
print(cars.size)
14360
```

iv) Get the Dimensionality of the DataFrame:

```
print(cars.shape)
(1436, 10)
```

v) Return the first ‘n’ rows:

```
print(cars.head(5))
 Price Age KM FuelType HP MetColor Automatic CC Doors Weight
0 13500 23.0 46986 Diesel 90 1.0 0 2000 three 1165
1 13750 23.0 72937 Diesel 90 1.0 0 2000 3 1165
2 13950 24.0 41711 Diesel 90 NaN 0 2000 3 1165
3 14950 26.0 48000 Diesel 90 0.0 0 2000 3 1165
4 13750 30.0 38500 Diesel 90 0.0 0 2000 3 1170
```

v) Return the last ‘n’ rows:

```
print(cars.tail(5))
 Price Age KM FuelType HP MetColor Automatic CC Doors Weight
1431 7500 NaN 20544 Petrol 86 1.0 0 1300 3 1025
1432 10845 72.0 ?? Petrol 86 0.0 0 1300 3 1015
1433 8500 NaN 17016 Petrol 86 0.0 0 1300 3 1015
1434 7250 70.0 ?? NaN 86 1.0 0 1300 3 1015
1435 6950 76.0 1 Petrol 110 0.0 0 1600 5 1114
```

vii) Access the group of rows & columns by label:

```
print(cars.at[4,'FuelType'])
Diesel

print(cars.iat[4,3])
Diesel

print(cars.loc[:,"FuelType"])
0 Diesel
```

1 Diesel
2 Diesel
3 Diesel
4 Diesel
… … … …
1431 Petrol
1432 Petrol
1433 Petrol
1434 NaN
1435 Petrol
Name: FuelType, Length: 1436, dtype: object
print(cars.loc[0:6,'FuelType'])
0 Diesel
1 Diesel
2 Diesel
3 Diesel
4 Diesel
5 Diesel
6 Diesel
Name: FuelType, dtype: object

viii) Return a series with datatypes of each column.
print(cars.dtypes)
Price int64
Age float64
KM object
FuelType object
HP object
MetColor float64
Automatic int64
CC int64
Doors object
Weight int64
dtype: object

ix) Return the count of unique datatypes in DataFrame:
print(cars.dtypes.value_counts())
int64 4
object 4
float64 2
dtype: int64

x) Return the subset of columns from DataFrame based on the column datatype:
print(cars.select_dtypes(include='int64'))
Price Automatic CC Weight
0 13500 0 2000 1165
1 13750 0 2000 1165
2 13950 0 2000 1165
3 14950 0 2000 1165
4 13750 0 2000 1170
... ...
1431 7500 0 1300 1025
1432 10845 0 1300 1015
1433 8500 0 1300 1015
1434 7250 0 1300 1015
1435 6950 0 1600 1114
[1436 rows x 4 columns]
print(cars.select_dtypes(exclude='object'))
Price Age MetColor Automatic CC Weight

0 13500 23.0 1.0 0 2000 1165
1 13750 23.0 1.0 0 2000 1165
2 13950 24.0 NaN 0 2000 1165
3 14950 26.0 0.0 0 2000 1165
4 13750 30.0 0.0 0 2000 1170
... ...
1431 7500 NaN 1.0 0 1300 1025
1432 10845 72.0 0.0 0 1300 1015
1433 8500 NaN 0.0 0 1300 1015
1434 7250 70.0 1.0 0 1300 1015
1435 6950 76.0 0.0 0 1600 1114
[1436 rows x 6 columns]
print(cars.select_dtypes(include='int64',exclude='object'))
Price Automatic CC Weight
0 13500 0 2000 1165
1 13750 0 2000 1165
2 13950 0 2000 1165
3 14950 0 2000 1165
4 13750 0 2000 1170
... ...
1431 7500 0 1300 1025
1432 10845 0 1300 1015
1433 8500 0 1300 1015
1434 7250 0 1300 1015
1435 6950 0 1600 1114
[1436 rows x 4 columns]

xi) Return the summary of the DataFrame:
print(cars.info)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 Price 1436 non-null int64
1 Age 1336 non-null float64
2 KM 1436 non-null object
3 FuelType 1336 non-null object
4 HP 1436 non-null object
5 MetColor 1286 non-null float64
6 Automatic 1436 non-null int64
7 CC 1436 non-null int64
8 Doors 1436 non-null object
9 Weight 1436 non-null int64
dtypes: float64(2), int64(4), object(4)

xii) Find the unique elements of the column KM, HP, MetColor, Automatic:
print(np.unique(cars['KM']))
['1' '10000' '100123' ... '99865' '99971' '??']
print(np.unique(cars['HP']))
['107' '110' '116' '192' '69' '71' '72' '73' '86' '90' '97' '98' '????']
print(np.unique(cars['MetColor']))
[0. 1. nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan

```
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan nan nan nan nan nan]
print(np.unique(cars['Automatic']))
[0 1]
```

xiii) Explicitly convert Datatypes from one to another (MetColor and Automatic as Object) :

```
print(cars2.info())
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 Price 1436 non-null int64
1 Age 1336 non-null float64
2 KM 1421 non-null float64
3 FuelType 1336 non-null object
4 HP 1430 non-null float64
5 MetColor 1286 non-null float64
6 Automatic 1436 non-null int64
7 CC 1436 non-null int64
8 Doors 1436 non-null object
9 Weight 1436 non-null int64
dtypes: float64(4), int64(4), object(2)
None
cars2['MetColor'] = cars2['MetColor'].astype('object')
cars2['Automatic'] = cars2['Automatic'].astype('object')
print(cars2.info())
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 Price 1436 non-null int64
1 Age 1336 non-null float64
2 KM 1421 non-null float64
3 FuelType 1336 non-null object
4 HP 1430 non-null float64
5 MetColor 1286 non-null object
6 Automatic 1436 non-null object
7 CC 1436 non-null int64
8 Doors 1436 non-null object
9 Weight 1436 non-null int64
dtypes: float64(3), int64(3), object(4)
memory usage: 123.4+ KB
```

xiv) Replace a value with a desired value for the attribute Doors:

```
print(np.unique(cars2['Doors']))
['2' '3' '4' '5' 'five' 'four' 'three']
cars2['Doors'].replace('five',5,inplace=True)
cars2['Doors'].replace('four',4,inplace=True)
cars2['Doors'].replace('three',3,inplace=True)
```

xv) convert Door to integer DataType:

```
cars2['Doors'] = cars2['Doors'].astype('int')
print(np.unique(cars2['Doors']))
[2 3 4 5]
```

xvi) Find the count of missing value passed in each column:

```
print(cars1.isnull().sum())
```

```
Price          0
Age          100
KM            15
FuelType     100
HP             6
MetColor     150
Automatic      0
CC             0
Doors          0
Weight         0
```

xvii) Copying a dataframe to another dataframe:

```
cars2 = cars1.copy()
```

xviii) Fill the missing values in integer datatype:

```
cars2['Age'].fillna(cars2['Age'].mean(),inplace=True)
cars2['KM'].fillna(cars2['KM'].mean(),inplace=True)
cars2['HP'].fillna(cars2['HP'].mean(),inplace=True)
print(cars2.isnull().sum())
```

```
Price          0
Age            0
KM             0
FuelType     100
HP             0
MetColor     150
Automatic      0
CC             0
Doors          0
Weight         0
```

xix) Fill the missing value in categorial datatype:

```
cars2['FuelType'].fillna(cars2['FuelType'].value_counts().index[0],inplace=True)
cars2['MetColor'].fillna(cars2['MetColor'].value_counts().index[0],inplace=True)
print(cars2.isnull().sum())
```

```
Price          0
Age            0
KM             0
FuelType       0
HP             0
MetColor       0
Automatic      0
CC             0
Doors          0
Weight         0
```

RESULT: The python program to read the given csv file and to explore & pre-process the data was executed successfully

EXP NO: 3a **IMPLEMENTATION OF NAIVE BAYES CLASSIFICATION**

DATE: 17/03/2022

AIM: To implement Naive Bayes classification using package.

ALGORITHM:
Read the Dataset
Do required preprocessing in the Dataset
Import required packages for the Dataset classification
Train and test the Dataset and implement Naive bayes classification
Get the output as Confusion Matrix and Accuracy Score.

PROGRAM:

```
import numpy as np
import pandas as pd
data = pd.read_csv("/home/user/Downloads/iris.csv",index_col=0,na_values=['??','???','###'])
print(data.info())

print(np.unique(data['SepalLengthCm']))
print(np.unique(data['SepalWidthCm']))
print(np.unique(data['PetalLengthCm']))
print(np.unique(data['PetalWidthCm']))
print(np.unique(data['Species']))
print(data.isnull().sum())

data['SepalLengthCm'].fillna(data['SepalLengthCm'].mean(),inplace=True)
data['SepalWidthCm'].fillna(data['SepalWidthCm'].mean(),inplace=True)
data['PetalLengthCm'].fillna(data['PetalLengthCm'].mean(),inplace=True)
print(data.isnull().sum())

column_list= list(data.columns)
feature_list=list(set(data.columns)-set(['Species']))
x = data[feature_list].values
y = data['Species'].values

from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(x,y,test_size=0.25,random_state=0)
print(train_x)
print(test_x)
print(train_y)
print(test_y)

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(train_x,train_y)
ydash = classifier.predict(test_x)
print(ydash)

from sklearn.metrics import accuracy_score,confusion_matrix
cm = confusion_matrix(test_y,ydash)
asc = accuracy_score(test_y,ydash)
print("Confusion Matrix: \n",cm)
print("Accuracy_score: ",asc*100)
print("M is classified samples = ",format((test_y!=ydash).sum()))
```

OUTPUT:

```
Confusion Matrix:  [[13  0  0]
                    [ 0 16  0]
                    [ 0  0  9]]
Accuracy_score:  100.0
M is classified samples =  0
```

RESULT: The Dataset has been classified using Naive Bayes classification successfully.

EXP NO: 3b **IMPLEMENTATION OF NAIVE BAYES CLASSIFICATION**

DATE: 24/03/2022

AIM: To implement Navie Bayes Classification without using packages

ALGORITHM:
Read the Dataset
Do required preprocessing in the Dataset
Create a function to define formula for Naive Bayes
Split the dataset into two arrays
Find the mean and variance of each value in the arrays and store it in a different arrays
Call the function in which formula is defined and print the results
To verify values get values from user and store it in list
Call the function and print the values

PROGRAM:

```
import numpy as np
import pandas as pd

def calculate_probability(x, mean, variance):
        exponent = np.exp(-((x-mean)**2 / (2 * variance )))
        return (1 / (np.sqrt(2 * np.pi) * variance)) * exponent

data=pd.read_csv("iris.csv", index_col=0,na_values=["??","????","###"])
print(data)
data.info()

print(np.unique(data['SepalLengthCm']))
print(np.unique(data['SepalWidthCm']))
print(np.unique(data['PetalLengthCm']))
print(np.unique(data['PetalWidthCm']))
print(np.unique(data['Species']))

data['Species'].replace('Iris-setosa',1,inplace=True)
data['Species'].replace('Iris-versicolor',2,inplace=True)
data['Species'].replace('Iris-virginica',3,inplace=True)
data['Species'] = data['Species'].astype('int64')
data.info()

print(data.isnull().sum())
print(data.describe())

data['SepalLengthCm'].fillna(data['SepalLengthCm'].mean(),inplace=True)
data['SepalWidthCm'].fillna(data['SepalWidthCm'].mean(),inplace=True)
data['PetalLengthCm'].fillna(data['PetalLengthCm'].mean(),inplace=True)
data['PetalWidthCm'].fillna(data['PetalWidthCm'].mean(),inplace=True)

print(data.isnull().sum())

class_label = np.array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'])
features = np.array(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'])

AM = np.zeros((len(features), len(class_label)))
AV = np.zeros((len(features), len(class_label)))

for i in range(len(class_label)):
   clas = pd.DataFrame(data[data["Species"].isin([i+1])])
```

```
    for j in range(len(features)):
        AM[j][i] = np.mean(clas[features[j]])
        AV[j][i] = np.var(clas[features[j]])

print("Enter the inputs:")
sl = float(input("Enter the Sepal Length :: "))
sw = float(input("Enter the Sepal Width :: "))
pl = float(input("Enter the Petal Length :: "))
pw = float(input("Enter the Petal Width ::"))

sample = [sl, sw, pl, pw]

A = np.zeros((len(features),len(class_label)))
for i in range(len(features)):
    A[i] = calculate_probability(sample[i], AM[i], AV[i])

lar = 0.00
PC = np.array([0.33,0.33,0.33])

for i in range(len(class_label)):
    for j in range(len(features)):
        PC[i] = PC[i] * A[j][i]
    if lar < PC[i]:
        lar = PC[i]
        k = i

print('Probabilities of each class is {}' . format(PC))
print("The sample {} belongs to Class :: {}" . format(sample, class_label[k]))
```

OUTPUT:

```
Enter the inputs:
Enter the Sepal Length :: 3.2
Enter the Sepal Width :: 3.5
Enter the Petal Length :: 3.9
Enter the Petal Width ::3.3
Probabilities of each class is [4.06572782e-189 9.30015752e-029 2.18515187e-013]
The sample [3.2, 3.5, 3.9, 3.3] belongs to Class :: Iris-virginica
```

RESULT: The Dataset has been classified using Naive Bayes classification without using package successfully

EXP NO: 4a **IMPLEMENTATION OF LINEAR REGRESSION**
DATE: 24/03/2022

AIM: To implement Linear Regression using package

ALGORITHM:
Read the Dataset
Check if preprocessing is required and complete the process
Assign each to column in the Dataset to a variable
Import required packages for linear regression
Train and test the data and implement the Linear regression
Using metrics of Linear regression output is printed

PROGRAM:

```
import numpy as np
import pandas as pd
data = pd.read_csv("Height-Weight.csv")
print(data.isnull().sum())
print(data.info())

x = data.Height.values[:,np.newaxis]
y = data.Weight.values

from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(x,y,test_size=0.25,random_state=0)

from sklearn.linear_model import LinearRegression
classifier = LinearRegression()
classifier.fit(train_x,train_y)
ydash = classifier.predict(test_x)

from sklearn.metrics import r2_score
r2 = r2_score(test_y,ydash)
print("Accuracy = %f" %(r2*100))
```

OUTPUT:
Accuracy = 96.447997

RESULT: The Dataset has been processed using Linear Regression using package successfully

EXP NO: 4b **IMPLEMENTATION OF LINEAR REGRESSION**
DATE: 24/03/2022

AIM: To implement Linear Regression without using package

ALGORITHM:
Read the Dataset
Check for preprocessing and complete the process
Assign each column in the Dataset to a variable
Using formula y = mx+c,
Linear regression is calculated and output is printed

PROGRAM:

```
import numpy as np
import pandas as pd
data = pd.read_csv("/home/user/Downloads/Height-Weight.csv")

x = data.Height.values
y = data.Weight.values

n = len(x)
xsq = x*x
xy = x*y
sxsq =np.sum(xsq)
sxy = np.sum(xy)
sx = np.sum(x)
sy = np.sum(y)

m = (n*sxy)-(sx*sy)//(n*sxsq)-(sx)**2
c = sy - (m*sx)//n

X = float(input("Enter a value: "))
y = (m*X)+c
print("Result = ",y)
```

OUTPUT:

```
Enter a value: 1.9
Result =  6560.051609999995
```

RESULT: The Dataset has been processed using Linear Regression without using package successfully

EXP NO: 5a **K NEAREST NEIGHBOURS (USING PACKAGES)**
DATE: 24/03/2023

AIM:
To implement K Nearest Neighbours using packages.
ALGORITHIM:
1. Read the Dataset
2. Do required preprocessing in the Dataset
3. Import required packages for the Dataset classification
4. Train and test the Dataset and implement K nearest neighbour's classification using the packages
5. Get the output as Confusion Matrix and Accuracy Score.

PROGRAM:

```
import numpy as np
import pandas as pd

data=pd.read_csv("/home/user/Downloads/iris.csv", index_col=0, na_values=(["??", "????", "###"]))

data["Species"].replace('Iris-setosa', 1, inplace=True)
data["Species"].replace('Iris-versicolor', 2, inplace=True)
data["Species"].replace('Iris-virginica', 3, inplace=True)

data["Species"]=data["Species"].astype("int64")

class_label=np.array(['Iris-sentosa', 'Iris-versicolor', 'Iris-virginica'])

data['SepalLengthCm'].fillna(data['SepalLengthCm'].mean(), inplace=True)
data['SepalWidthCm'].fillna(data['SepalWidthCm'].mean(), inplace=True)
data['PetalLengthCm'].fillna(data['PetalLengthCm'].mean(), inplace=True)
data['PetalWidthCm'].fillna(data['PetalWidthCm'].mean(), inplace=True)

column_list=list(data.columns)
feature_list=list(set(data.columns)-set(data['Species']))
X=data[feature_list].values
Y=data["Species"].values

from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y=train_test_split(X, Y, test_size=0.25, random_state=None)

from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier()
classifier.fit(train_x, train_y)
y_dash=classifier.predict(test_x)
from sklearn.metrics import accuracy_score, confusion_matrix
cm5=confusion_matrix(test_y, y_dash)
acc5=accuracy_score(test_y, y_dash)
print("The confusion matrix is: \n", cm5)
print("The accuracy score is: ", acc5*100)
print("Misclassified samples = {}".format((test_y != y_dash).sum()))
```

OUTPUT:
The confusion matrix is:
[[12 1 0]
[0 13 0]
[0 0 12]]
The accuracy score is: 97.36842105263158
Misclassified samples = 1

RESULT:
The dataset is processed and the K nearest neighbours model is applied and the confusion matrix, accuracy score and the number of misclassified is displayed successfully.

EXP NO: 5b
DATE: 24/03/2023 **K NEAREST NEIGHBOURS (WITHOUT USING PACKAGES)**

AIM:
To implement K Nearest Neighbours without using packages.

ALGORITHIM:
1. Read the Dataset.
2. Do required preprocessing in the Dataset.
3. Import required packages for the Dataset classification.
4. Train and test the Dataset and implement K nearest neighbour's classification without using the packages.
5. Get the input from the user and predict the corresponding output.

PROGRAM:

```
import numpy as np
import pandas as pd

data=pd.read_csv("/home/user/Downloads/iris.csv", index_col=0, na_values=(["??", "????", "###"]))

data["Species"].replace('Iris-setosa', 1, inplace=True)
data["Species"].replace('Iris-versicolor', 2, inplace=True)
data["Species"].replace('Iris-virginica', 3, inplace=True)

data["Species"]=data["Species"].astype("int64")

class_label=np.array(['Iris-sentosa', 'Iris-versicolor', 'Iris-virginica'])

data['SepalLengthCm'].fillna(data['SepalLengthCm'].mean(), inplace=True)
data['SepalWidthCm'].fillna(data['SepalWidthCm'].mean(), inplace=True)
data['PetalLengthCm'].fillna(data['PetalLengthCm'].mean(), inplace=True)
data['PetalWidthCm'].fillna(data['PetalWidthCm'].mean(), inplace=True)

features=np.array(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'])

data['Distance']=0

print("Enter the input: ")
SL=float(input("Enter the sepal length: "))
SW=float(input("Enter the sepal width: "))
PL=float(input("Enter the petal length: "))
PW=float(input("Enter the petal width: "))

k=75

sample=[SL, SW, PL, PW]
for i in range (len(data)):
    for j in range (len(features)):
        data.iloc[i, 5]+=((sample[j]-data.iloc[i, j])**2)

sorted_data=data.sort_values(by='Distance')
output=sorted_data.iloc[0:k, 4]
```

```
Y=output.value_counts().index[0]
print("The sample {} belongs to the class {}".format(sample, class_label[Y-1]))
```

OUTPUT:

```
Enter the input:
Enter the sepal length: 5.1
Enter the sepal width: 3.5
Enter the petal length: 1.4
Enter the peatal width: 0.2
The sample [5.1, 3.5, 1.4, 0.2] belongs to the class Iris-sentosa
```

RESULT:

The dataset is processed and the K nearest neighbours' model is applied without using packages. The input is successfully entered and the output is predicted correctly.

EXP NO: 6a K MEANS CLUSTERING (USINGPACKAGES)

DATE: 21/04/2023

AIM:

To implement K means clustering using packages

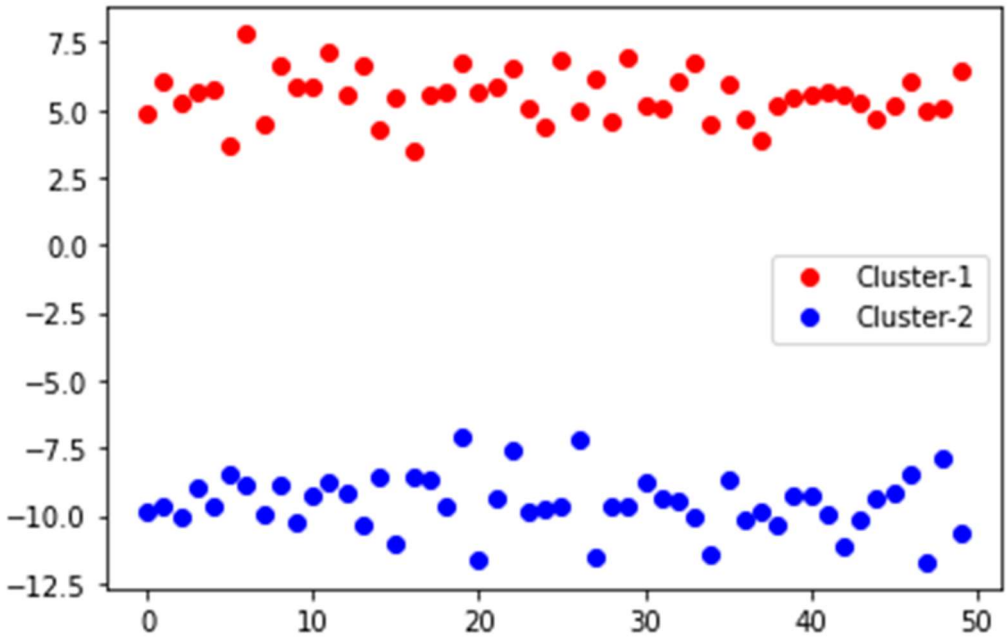
ALGORITHIM:

1. Create a dataset using make_blobs function where the inputs are X and the output is Y.
2. Import KMeans and fit and predict using the model.
3. Now create a dictionary to store distance between the centroids and the points which is predicted as output by the model.
4. Based on the distance, assign which cluster the points belongs to.
5. Plot a scatter plot for all the points and their corresponding clusters.

PROGRAM:

```
from sklearn.datasets import make_blobs
import numpy as np
X, Y = make_blobs(n_samples = 100, centers = 2, n_features = 1, random_state = 10)
from sklearn.cluster import KMeans
model = KMeans(init = 'random', n_clusters = 2, max_iter = 100 , random_state = 10)
model.fit(X)
iden_cluster = model.fit_predict(X)
c = iden_cluster + 1
y = {} # dict containing the distance between the respective centroids and the points
for j in range(2):
    y[j+1] = np.array([]).reshape(X.shape[1], 0)
for j in range(100):
    y[c[j]] = np.c_[y[c[j]], X[j]]
for j in range(2):
    y[j + 1] = y[j + 1].T
output = y
k=2
import matplotlib.pyplot as plt
color = ['red', 'blue']
labels = ['Cluster-1', 'Cluster-2']
for j in range(k):
    plt.plot(output[j + 1][:, 0], 'o', c = color[j], label = labels[j])
plt.ylabel("")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

The dataset is created and the K Means clustering model is applied successfully and the graph is plotted correctly using packages.

EXP NO: 6b **K MEANS CLUSTERING (WITHOUT USING PACKAGES)**
DATE: 21/04/2023

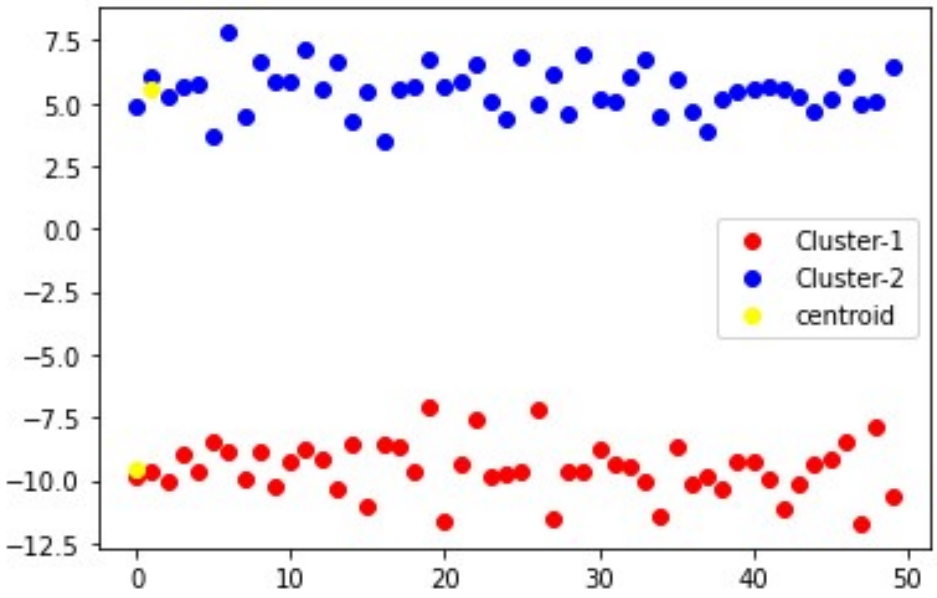
AIM:
To implement K means clustering without using packages.

ALGORITHIM:
1. Create a dataset using make_blobs function where the inputs are X and the output is Y.
2. Now choose any 2 random values as the centroids.
3. Find the distance between the data points and the centroids using Euclidean distance = √((x_n-x)^2)
4. Create a dictionary to assign the cluster to the data points based on their distance.
5. Plot a scatter plot for all the points and their corresponding clusters and highlight the centroids.

PROGRAM:

```
from sklearn.datasets import make_blobs
import numpy as np
X, Y = make_blobs(n_samples = 100, centers = 2, n_features = 1, random_state = 10)
m = X.shape[0]
n = X.shape[1]
k = 2
n_iter = 100
centroids = np.array([]).reshape(n, 0)
import random as rd
for i in range (k):
    rand = rd.randint(0, m-1)
    centroids = np.c_[centroids, X[rand]] # np.c_[arr1, arr2] - concatenates the second array to the first array
output = {}
for i in range(n_iter):
    ED = np.array([]).reshape(m, 0)
    for j in range(k):
        dist = np.sum(((X-centroids[:,j]) ** 2), axis = 1)
        ED=np.c_[ED, dist]
    c = np.argmin(ED, axis = 1) + 1
    y = {} # dict containing the distance between the respective centroids and the points
    for j in range(k):
        y[j+1] = np.array([]).reshape(n, 0)
    for j in range(m):
        y[c[j]] = np.c_[y[c[j]], X[j]]
    for j in range(k):
        y[j + 1] = y[j + 1].T
    for j in range(k):
        centroids[:, j] = np.mean(y[j+1], axis=0)
    output = y

import matplotlib.pyplot as plt
color = ['red', 'blue']
labels = ['Cluster-1', 'Cluster-2']
for j in range(k):
    plt.plot(output[j + 1][:, 0], 'o', c = color[j], label = labels[j])
plt.plot(centroids[0, :], 'o', c = 'yellow', label='centroid')
plt.ylabel("")
plt.legend()
plt.show()
```

OUTPUT:



RESULT:

The dataset is created and the K Means clustering model is applied successfully and the graph is plotted correctly without using packages

EXP NO: 7a **PERCEPTRON (USING PACKAGES)**
DATE: 28/04/2023

AIM:
To implement perceptron using packages.

ALGORITHIM:
1. The dataset is read and the input and output are obtained.
2. The data is split for training and testing.
3. The perceptron function is imported and the data is fitted and the output is predicted.
4. The confusion matrix, accuracy score and misclassified samples are displaced by comparing predicted output with the actual output.

PROGRAM:
```
import pandas as pd
data = pd.read_csv("D:\SVCE\SEM-4\Applied ML lab\AND.csv")
target_column = ['Output']
features = list(set(list(data.columns))-set(target_column))

x = data[features].values
y = data[target_column].values
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.25,random_state=1)

from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(learning_rate_init=0.1,random_state=1)
classifier.fit(x_train,y_train.ravel())
ydash = classifier.predict(x_test)

from sklearn.metrics import confusion_matrix,accuracy_score
CM = confusion_matrix(y_test, ydash)
ASC = accuracy_score(y_test,ydash)

print("Confusion Matrix:: ",CM)
print("Accuracy Score:: ",ASC*100)
print("Misclassified :: ",format((y_test!=ydash).sum()))
```

OUTPUT:
```
Confusion Matrix::
[[2 0]
[0 3]]
Accuracy Score:: 100.0
Misclassified :: 12
```

RESULT:
The perceptron is fitted successfully and the output is predicted successfully using packages.

EXP NO: 7b **PERCEPTRON (WITHOUT USING PACKAGES)**
DATE: 28/04/2023

AIM:
To implement perceptron without using packages.

ALGORITHIM:
1. The dataset is read and the input and output are obtained.
2. The data is split for training and testing.
3. The learning rate and epoch values are initialised.
4. The weights are initialised using random values and the change in weights are initialised as 0.
5. The output is predicted using, output = $w_0 + w_1x_1 + w_2x_2$.
6. The predicted output is obtained using sign function.
7. The error is obtained and then the change in weights is updated.
8. Now the weights are updated.
9. This is repeated for number of epochs.
10. The final output is displayed and the accuracy is calculated.

PROGRAM:
```
import pandas as pd
import numpy as np

data = pd.read_csv("D:\SVCE\SEM-4\Applied ML lab\AND.csv")
target_col = ['Output']
features = list(set(list(data.columns))-set(target_col))
x = data[features].values
y = data[target_col].values

test_size = int(round(0.25*len(data),0))
train_size = len(data)-test_size
x_train = x[0:train_size]
x_test = x[train_size:len(data)]
y_train = y[0:train_size]
y_test = y[train_size:len(data)]

lr = 0.1
epochs = 100

w0 = np.random.randn()
w1 = np.random.randn()
w2 = np.random.randn()
d_w0 = 0
d_w1 = 0
d_w2 = 0

for i in range(epochs):
        j = 0
        for x in x_train:
                out = w0 + w1*x[0] + w2*x[1]
                if out>0: res = 1
                else: res = 0
                error = y_train[j][0] - res
```

```
            d_w0 = lr*error
            d_w1 = lr*error*x[0]
            d_w2 = lr*error*x[1]
            w0 = w0 + d_w0
            w1 = w1 + d_w1
            w2 = w2 + d_w2
            j+=1

result = []
count = 0

for i in range(len(y_test)):
            out = out = w0 + w1*x_test[i][0] + w2*x_test[i][1]
            if out>=0:res=1
            else:res=0
            da ={"X1":x_test[i][0],"X2":x_test[i][1],"Predicted":res,"Actual":y_test[i][0]}
            if y_test[i][0]==res:
            count+=1
            result.append(da)

print(pd.DataFrame(result))
print("\nAccuracy: ",count/len(y_test)*100)
```

OUTPUT:

```
          X1     X2    Predicted    Actual
0       0.11    1.020     0            0
1       0.98    0.870     1            1
2       0.20    1.300     1            0
3       0.20    0.003     0            0
Accuracy: 75.0
```

RESULT:

The perceptron is fitted successfully and the output is predicted successfully without using packages.

EXP NO: 8a **LOGISTIC REGRESSION (USING PACKAGES)**
DATE: 28/04/2023

AIM:
To implement logistic regression model using packages.

ALGORITHIM:
1. First we create a dataset having 100 samples and 4 features using make_classifiication function.
2. Now we split the dataset into training and testing using train_test_split function.
3. Now we import the logistic regression model and fit for our training data
4. We then predict the outputs for the testing data
5. We compare the predict output and the actual output using the confusion matrix and accuracy score.

PROGRAM:
```
from sklearn.datasets import make_classification
x,y = make_classification(n_samples=100, n_features=4)

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.25,random_state=1)

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=1)

classifier.fit(x_train,y_train.ravel())
ydash = classifier.predict(x_test)

from sklearn.metrics import confusion_matrix,accuracy_score
CM = confusion_matrix(y_test, ydash)
ASC = accuracy_score(y_test,ydash)

print("Confusion Matrix::\n ",CM)
print("Accuracy Score::",ASC*100,"%")
print("Misclassified Samples ::",format((y_test!=ydash).sum()))
```

OUTPUT:
```
Confusion Matrix::
        [[ 9 0]
         [ 4 12]]
Accuracy Score:: 84.0 %
Misclassified Samples :: 4
```

RESULT:
The logistic regression model is fitted successfully and the output is predicted successfully using packages.

EXP NO: 8b **LOGISTIC REGRESSION (WITHOUT USING PACKAGES)**
DATE: 28/04/2023

AIM:
To implement logistic regression model without using packages.

ALGORITHIM:
1. First we create a dataset having 100 samples and 4 features using make_classifiication function.
2. Now we split the dataset into training and testing using train_test_split function.
3. Now we create a function to return the sigmoid of the passed value sigmoid(x) = 1/1+ex [-x]
4. Now we have to initialise the weights with random values and the change in weights with 0.
5. Now the for each enumeration of the inputs we calculate the output as op = w[i]*x[i-1]
6. We find the sigmoid of the output and classify the return value to the class.
7. Now find the error which is actual – predicted
8. Now update the weights.
9. We repeat the above steps for the number of epochs.
10. We then predict the output for the testing data.
11. Accuracy is calculated and printed.

PROGRAM:
```
from sklearn.datasets import make_classification
x,y = make_classification(n_samples=100, n_features=4)

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.25,random_state=1)

def sigmoid(out):
    return 1/(1+np.exp(-out))

epochs = 100
import numpy as np
import pandas as pd
w=[]
d_w=[]

for i in range(5):
    w.append(np.random.randn())
    d_w.append(0)

for i in range(epochs):
    for k,x in enumerate(x_train):
        out = w[0]
        for j in range(1,5):
            out+= w[j]*x[j-1]
        z = sigmoid(out)
        c=1 if z>0.5 else 0
        error = y_train[k] - c
        d_w[0] = error
        w[0] = w[0] + d_w[0]
        for j in range(1,5):
            d_w[j] = error*x[j-1]
```

```
            w[j] = w[j] + d_w[j]

result =[]
count = 0

for i in range(len(y_test)):
    out = w[0]
    for j in range(1,5):
        out+=w[j]*x_test[i][j-1]
        z = sigmoid(out)
        c=1 if z>0.5 else 0
        da = {"Predicted":c,"Actual":y_test[i]}
        if y_test[i]==c:
            count+=1
        result.append(da)

print(pd.DataFrame(result))
print("\nAccuracy: ",count/len(y_test)*100)
```

OUTPUT

	Predicted	Actual
0	0	0
1	0	0
2	1	0
3	1	1
4	1	1
5	0	0
6	1	1
7	1	1
8	0	0
9	1	1
10	1	0
11	0	0
12	0	0
13	1	1
14	0	0
15	0	0
16	0	0
17	0	1
18	1	0
19	1	1
20	0	0
21	1	1
22	1	0
23	1	1
24	1	1

Accuracy: 80.0

RESULT:

The logistic regression model is fitted successfully and the output is predicted successfully without using packages**.**

EXP NO: 9
DATE: 08/05/2023

SUPPORT VECTOR MACHINE

Aim:

To write a python program to implement support vector machine using packages.

Algorithm:

1. Start
2. Create dataset using make-classification function.
3. Split the dataset into training data and testing data.
4. Import sum model and fit for testing data .
5. Obtain the predicted output and actual output using confusion matrix and accuracy score and display it.
6. Stop.

Program:

```
from sklearn.datasets import make_classification
x,y=make_classification(n_samples=100,n_features=4)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=40)
from sklearn.svm import SVC
clf=SVC(random_state=1)
clf.fit(x_train,y_train)
ydash=clf.predict(x_test)
from sklearn.metrics import accuracy_score,confusion_matrix
cnf=confusion_matrix(y_test,ydash)
print(cnf)
acc=accuracy_score(y_test,ydash)
print("Accuracy score:",acc*100)
print("Misclassified samples:",(y_test!=ydash).sum())
```

Output:

```
[[14  1]
 [ 2  8]]
Accuracy score: 88.0
Misclassified samples: 3
```

Result:

Therefore, support vector machine has been implemented successfully.

EXP NO: 10
DATE: 15/05/2023

DECISION TREE

Aim:

To write a python program to implement decision making tree using packages.

Algorithm:

1. Start
2. Import required libraries
3. Import dataset
4. Split the data into training and testing data
5. Import decision tree model and fit the data
6. Obtain the predicted output
7. Compare the predicted output and actual output using confusion and accuracy matrix and display the decision tree
8. Stop

Program:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
col=['Class Name','Left Weight','Left Distance','Right Weight','Right Distance']
data=pd.read_csv("/home/user/Downloads/balance-scale.data",names=col,sep=',')
from sklearn.model_selection import train_test_split
x=data.drop('Class Name',axis=1)
y=data[['Class Name']]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=42)
from sklearn.tree import DecisionTreeClassifier
clf_model = DecisionTreeClassifier(criterion="gini", random_state=42,max_depth=3, min_samples_leaf=5)
clf_model.fit(x_train,y_train)
ydash = clf_model.predict(x_test)
from sklearn.metrics import accuracy_score,confusion_matrix
confusion_mat = confusion_matrix(y_test,ydash)
accuracy_score = accuracy_score(y_test,ydash)
print('Confusion Matrix')
print(confusion_mat)
print("Accuracy = %f" %(accuracy_score*100))
target = list(data['Class Name'].unique())
feature_names = list(x.columns)
from sklearn import tree
import graphviz
dot_data = tree.export_graphviz(clf_model,  out_file=None,
                feature_names=feature_names,
                class_names=target,
                filled=True, rounded=True,
                special_characters=True)
graph = graphviz.Source(dot_data)
```

Output:

Confusion Matrix

[[0 6 8]

 [0 50 16]

 [0 19 58]]

Accuracy = 68.789809

Result:

Therefore, the decision making problem has been implemented successfully