MACHINE LEARNING
LAB RECORD

WEEK 1

Program 1

AIM: To create a class circle and initialize the radius value to print area and circumference.

DESCRIPTION:

In python, the class is considered as the blue print of the object. The constructor class is invoked as soon as an object is created.

In the program,

- Create a class circle, and define a constructor which takes the radius value.
- Define a function area, which takes radius as an argument and prints the area using the formula, Area=pi*r**2, pi=(3.14…)
- Define a function circumference, which takes radius as an argument and prints the circumference using the formula, circumference = 2*pi*r, pi=(3.14…)
- Create on object c and pass the radius value. Next using the object name call the area and circumference methods.

CODE:

```
import math


class circle:
    def __init__(self,r):
        self.r=r
    def area(self):
        print("Area of the circle:",round(math.pi*self.r**2,4))
    def circum(self):
        print("Circumference of the circle: ", round(2*math.pi*self.r,4))


r=float(input("Enter the radius of the circle:"))
```

c=circle(r)

c.area()

c.circum()

OUTPUT:

```
================ RESTART: C:/Users/lenov
Enter the radius of the circle:2
Area of the circle: 12.5664
Circumference of the circle:  12.5664

================ RESTART: C:/Users/lenov
Enter the radius of the circle:2.4
Area of the circle: 18.0956
Circumference of the circle:  15.0796
```

CONCLUSION: The program is error free and the area and the circumference of the circle is calculated.

Program 2

AIM: To create a temperature class and define methods to convert Celsius to Fahrenheit and vice versa.

DESCRIPTION:

Celsius and Fahrenheit are the two measures to calculate the temperature.

The formula to convert Celsius to Fahrenheit: fahrenheit = celsius * 1.8 + 32

The formula to convert Fahrenheit to Celsius: celsius = (fahrenheit - 32) / 1.8

CODE:

```
class temperature:
    def ConvertToF(self,c):
        self.c=c
        print("Temperature in Fahrenheit: ", round(self.c*1.8+32,3))
    def ConvertToC (self,f):
        self.f=f
        print("Temperature in Celsius: ",round((self.f-32)/1.8,3))


t=temperature()
c=float(input("Enter temperature in Celsius:"))
t.ConvertToF(c)
f=float(input("Enter temperature in Fahrenheit:"))
t.ConvertToC(f)
```

OUTPUT:

```
Enter temperature in Celsius:32
Temperature in Fahrenheit:  89.6
Enter temperature in Fahrenheit:89.6
Temperature in Celsius:  32.0
```

CONCLUSION: The code is error free and the temperature has been converted to Celsius and and Fahrenheit.

Program 3

AIM: To create a student class and initialize it with name and roll number. Create methods to display details, set age and set marks.

DESCRIPTION:

The student class takes the input name and roll number from the object that has been created.

The setAge method takes the age of the student as an input.

The display methods is used to print the details of the student using the instance variables.

CODE:

```
class Student:
  def __init__(self, Name, Rollno):
    self.Name = Name
   self.Rollno = Rollno
  def setAge(self, age):
    self.age = age
  def setmarks(self, marks):
    self.marks = marks
  def display(self):
    print("Student Details")
    print("---------------"); print("Name : ", self.Name)
    print("Roll Number :", self.Rollno); print("Age : ", self.age); print("Marks : ", self.marks)
s = Student("Ash", 123)
s.setAge(18)
s.setmarks(100)
s.display()
```

OUTPUT:

```
Student Details
---------------
Name :  Ash
Roll Number : 123
Age :  18
Marks :  100
```

CONCLUSION: The program is error free and the details of the student has been printed.

Program 4

AIM: To create a class time and initialize with hours and minutes. Create methods to add time, display time and time in minutes.

DESCRIPTION: We create two objects that takes time in hours and minutes. The third object is created by adding the time of both the objects. The AddTime method adds the time of both the objects.

CODE:

```
class time:
    def __init__(self,h,m):
        self.h=h
        self.m=m
    def AddTime(t1,t2):
        t3=time(0,0)
        t3.h=t2.h+t1.h
        t3.m=t2.m+t1.m
        if t3.m>=60:
            t3.h+=1
            t3.m-=60
        return t3
    def display(self):
        print("Time is ",self.h," hours and ",self.m," minutes")
    def display_m(self):
        print("Time in mins: ", self.h*60+self.m)
t1=time(1,20)
t2=time(1,42)
t=time.AddTime(t1,t2)
t.display()
t.display_m()
```

OUTPUT:

```
Time is  3  hours and  2  minutes
Time in mins:  182
```

CONCLUSION: The added time and the time in minutes has been displayed.

WEEK 2

Program 1

AIM: To Read the following data set. Apply preprocessing techniques for data cleaning. Apply min – max normalization , Z- score normalization and decimal normalization on salary column and print it.(Note: refer .xls file)

DESCRIPTION:

Normalization is one of the most frequently used data preparation techniques, which helps us to change the values of numeric columns in the dataset to use a common scale

Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale.

- For min max scaling: We convert the values of the attribute such that their scaled values lies between 0 and 1.
  The formula is given as **Xn = ((X - Xminimum) / (Xmaximum - Xminimum))* (new_max-new_min) + new_min**
- For z-score normalisation, the values of the attribute are changed in such a way that their mean is equal to 0 and the standard deviation is 1.
  New value = (x – μ) / σ, where μ: Mean of data, σ: Standard deviation of data
- For decimal scaling normalisation,
  Normalized value of attribute = (v^i / 10^j)
- **Fit_transform (data)** – used to fit the data

CODE:

Min Max Scaling:

```
from sklearn.preprocessing import MinMaxScaler

import pandas as pd

df=pd.read_csv("employees.csv")

data=df[['DEPARTMENT_ID']]

scaler=MinMaxScaler()

sc=scaler.fit_transform(data)

print("Unscaled data:",data)

print("Scaled data:",sc)
```

Z-Score normalisation:

```
from sklearn.preprocessing import StandardScaler
import pandas as pd

df=pd.read_csv("employees.csv")
data=df[['DEPARTMENT_ID']]
print("Unscaled data:\n",data)

scaler=StandardScaler()
sc=scaler.fit_transform(data)
print("Scaled data:\n",sc)
```

Decimal Scaling Normalisation:

```
import pandas as pd
import math
df=pd.read_csv(“employees.csv”)
data=df[['department_id']]

j=math.ceil(math.log(data.max(),10))
v= df[['department_id']]/(10**j)
print(“scaled data:\n”,v)
```

OUTPUT:

For Min Max Scaling

```
Unscaled data:      DEPARTMENT_ID
0                50
1                50
2                10
3                20
4                20
5                40
6                70
7               110
8               110
9                90
10               90
11               90
12               60
13               60
14               60
15               60
16               60
```

```
Scaled data: [[0.
 [0.4]
 [0. ]
 [0.1]
 [0.1]
 [0.3]
 [0.6]
 [1. ]
 [1. ]
 [0.8]
 [0.8]
 [0.8]
 [0.5]
 [0.5]
 [0.5]
 [0.5]
 [0.5]
 [0.9]
 [0.9]
 [0.9]
 [0.9]
 [0.9]
```

Z-Score normalisation:

```
Unscaled data:
     DEPARTMENT_ID
0                50
1                50
2                10
3                20
4                20
5                40
6                70
7               110
8               110
9                90
10               90
```

```
Scaled data:
 [[-0.30565749]
 [-0.30565749]
 [-1.91438112]
 [-1.51220021]
 [-1.51220021]
 [-0.7078384 ]
 [ 0.49870433]
 [ 2.10742796]
 [ 2.10742796]
 [ 1.30306614]
```

Decimal Scaling Normalisation:

```
Unscaled data:
     DEPARTMENT_ID
0                50
1                50
2                10
3                20
4                20
5                40
6                70
7               110
8               110
9                90
10               90
```

```
    DEPARTMENT_ID
0            0.05
1            0.05
2            0.01
3            0.02
4            0.02
5            0.04
6            0.07
7            0.11
8            0.11
9            0.09
10           0.09
```

CONCLUSION: The various normalization techniques have been applied to scale the data.

Program 2a

AIM: Download a data set from Kaggle which contains at least one feature as numeric or continuous data. Get the nrows, ncolumns, datatype, summary stats of each column of a dataframe.

DESCRIPTION:

- The number of rows in the data set can be extracted using len(df)
- The number of columns in the dataset can be extracted using len(df.columns)
- Datatype: The datatype of each column can be extracted using df.dtypes, we can use df.info() to get the number of rows, data type
- Summary: The describe() function gives a summary of each column. The function gives mean, median, std, IQR values.

CODE:

```
import pandas as pd


df=pd.read_csv("employees.csv")

print("Number of rows:", len(df))

print("\nNumber of columns:",len(df.columns))

print("\nThe datatypes:\n",df.dtypes)

print("\nSummary Stats:",df.describe())
```

OUTPUT:

```
Number of rows: 50

Number of columns: 11

The datatypes:
 EMPLOYEE_ID         int64
FIRST_NAME         object
LAST_NAME          object
EMAIL              object
PHONE_NUMBER       object
HIRE_DATE          object
JOB_ID             object
SALARY              int64
COMMISSION_PCT     object
MANAGER_ID        float64
DEPARTMENT_ID       int64
dtype: object

Summary Stats:        EMPLOYEE_ID        SALARY  MANAGER_ID
count    50.000000     50.000000   49.000000       50.00000
mean    134.760000   6182.320000  114.836735       57.60000
std      33.631594   4586.181772   20.591611       25.11687
min     100.000000   2100.000000  100.000000       10.00000
25%     112.250000   2725.000000  101.000000       50.00000
50%     124.500000   4600.000000  114.000000       50.00000
75%     136.750000   8150.000000  121.000000       60.00000
max     206.000000  24000.000000  205.000000      110.00000
```

CONCLUSION: The number of rows, columns, the datatypes of the dataset has been identified

Program 2b

AIM: To count number of missing values in each column of the dataset

DESCRIPTION: The number of missing values can be identified using df.isnull().sum()

CODE:

```
import pandas as pd
df=pd.read_csv("employees.csv")
missing=df.isnull().sum()
print("Number of missing values in each column:\n", missing)
```

OUTPUT:

```
Number of missing values in each column:
 EMPLOYEE_ID       0
FIRST_NAME         0
LAST_NAME          0
EMAIL              0
PHONE_NUMBER       0
HIRE_DATE          0
JOB_ID             0
SALARY             0
COMMISSION_PCT     0
MANAGER_ID         1
DEPARTMENT_ID      0
dtype: int64
```

CONCLUSION: The number of missing values per column has been identified.

Program 2c

AIM: Rename a specific column in a dataframe in a dataset

DESCRIPTION:

We use df.rename() to rename the names of specific columns. The input is given in the form of a dictionary, with keys being current names and values being new name. We set inplace=True.

CODE:

```
import pandas as pd

df=pd.read_csv("employees.csv")

print(df.columns)

#renaming SALARY to Salary

df.rename(columns={'SALARY':'Salary'},inplace=True)

print(df.columns)
```

OUTPUT:

```
Index(['EMPLOYEE_ID', 'FIRST_NAME', 'LAST_NAME', 'EMAIL', 'PHONE_NUMBER'
       'HIRE_DATE', 'JOB_ID', 'SALARY', 'COMMISSION_PCT', 'MANAGER_ID',
       'DEPARTMENT_ID'],
      dtype='object')
Index(['EMPLOYEE_ID', 'FIRST_NAME', 'LAST_NAME', 'EMAIL', 'PHONE_NUMBER'
       'HIRE_DATE', 'JOB_ID', 'Salary', 'COMMISSION_PCT', 'MANAGER_ID',
       'DEPARTMENT_ID'],
      dtype='object')
```

CONCLUSION: The program changes the column name of the dataset from ‘SALARY’ to ‘Salary’.

Program 2d

AIM: Replace missing values of multiple numeric columns with the mean

DESCRIPTION:

We can use mean(), median(), mode()[0] functions inorder to replace the missing values.

CODE:

```
import pandas as pd
df=pd.read_csv("employees.csv")
x=df['MANAGER_ID'].mean()
print(x)
df['MANAGER_ID'].fillna(x,inplace=True)
print(df['MANAGER_ID'])
```

OUTPUT:

```
114.83673469387755
0     124.000000
1     124.000000
2     101.000000
3     100.000000
4     201.000000
5     101.000000
6     101.000000
7     101.000000
8     205.000000
9     114.836735
```

CONCLUSION: The mean value of the column is identified and the missing value is replaced by the mean.

Program 2e

AIM: Change the order of columns of a data frame in a dataset.

DESCRIPTION:

In order to change the order of the columns in Pandas, we can use new_df=df.iloc[:,[0,2,1]]

Or we can use the loc method where we pass the column names.

CODE:

```
import pandas as pd

df=pd.read_csv("employees.csv")

print(df.columns)


new_df=df.iloc[:,[0,1,3,2,4,6,7,5,8,9,10]]

print(new_df.columns)
```

OUTPUT:

```
Index(['EMPLOYEE_ID', 'FIRST_NAME', 'LAST_NAME', 'EMAIL', 'PHONE_NUMBER',
       'HIRE_DATE', 'JOB_ID', 'SALARY', 'COMMISSION_PCT', 'MANAGER_ID',
       'DEPARTMENT_ID'],
      dtype='object')
Index(['EMPLOYEE_ID', 'FIRST_NAME', 'EMAIL', 'LAST_NAME', 'PHONE_NUMBER',
       'JOB_ID', 'SALARY', 'HIRE_DATE', 'COMMISSION_PCT', 'MANAGER_ID',
       'DEPARTMENT_ID'],
      dtype='object')
```

CONCLUSION: The columns order have been changed using the iloc method.

WEEK 3

Program

AIM: Use employees.csv and perform operations to deal with missing values.

DESCRIPTION:

isnull(object) is used to detect missing values for an array like object and indicates where values are missing (NaN) in object arrays.

pd.isna(array) – gives Boolean values in the format of an array

pd.isna(index) –

pd.isna(df) – gives a table of Boolean values

We can extract rows / columns having missing values using isnull() or isna() that checks if the element has a missing value.

Df.isnull() – returns Boolean values for each cell in the table

Df[‘column_name’].isnull() – returns missing values in that particular column

Df[df[‘column_name’].isnull()] – returns tables with missing values

Df.iloc[2].isnull() – used to return columns with missing values in a specific row

Count null values in a column: df[‘col’].isna().sum()

Count null values in a dataframe: df.isna().sum().sum()

Count NaN values across a row: df.loc[[index value]].isna().sum().sum()

Idxmin() – returns columns with most number of null values

CODE:

1. Extract rows with missing values for a speci c column, use isnull() for that column.

CODE:

```
import pandas as pd


df=pd.read_csv("employees.csv")
print("Rows with missing values in Manager_id:")
print(df['MANAGER_ID'].isnull())
```

Output:

```
Rows with missing values in Manager_id:
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9      True
```

2. Extract columns that contain at least one missing value.

Code:

```
import pandas as pd
df=pd.read_csv("employees.csv")
print("Columns with missing values:")
print(df.loc[:,df.isnull().any()])
```

Output:

```
Columns with missing values:
    MANAGER_ID
0        124.0
1        124.0
2        101.0
3        100.0
4        201.0
5        101.0
6        101.0
7        101.0
8        205.0
```

3. Extract rows that contain at least one missing value, use any() method.

Code:

```
import pandas as pd
df= pd.read_csv("employees.csv")
print(df[df.isnull().any(axis=1)])
```

Output:

```
   EMPLOYEE_ID FIRST_NAME LAST_NAME  ... COMMISSION_PCT MANAGER_ID DEPARTMENT_ID
9          100     Steven      King  ...              -        NaN            90
```

4. Find a list of columns with missing data

Code:

```
import pandas as pd
df= pd.read_csv("employees.csv")
print(df.isnull().any())
```

Output:

```
EMPLOYEE_ID       False
FIRST_NAME        False
LAST_NAME         False
EMAIL             False
PHONE_NUMBER      False
HIRE_DATE         False
JOB_ID            False
SALARY            False
COMMISSION_PCT    False
MANAGER_ID         True
DEPARTMENT_ID     False
dtype: bool
```

5. Find the number of missing values/data per column

Code:

```
import pandas as pd
df= pd.read_csv("employees.csv")
print("Number of missing values per column:")
print(df.isnull().sum())
```

Output:

```
Number of missing values per column
EMPLOYEE_ID       0
FIRST_NAME        0
LAST_NAME         0
EMAIL             0
PHONE_NUMBER      0
HIRE_DATE         0
JOB_ID            0
SALARY            0
COMMISSION_PCT    0
MANAGER_ID        1
DEPARTMENT_ID     0
dtype: int64
```

6. Find the column with the maximum number of missing data

Code:

```
import pandas as pd
df= pd.read_csv("employees.csv")
print("Column with maximum number of missing data:")
print(df.isnull().sum().idxmax())
```

Output:

```
Column with maximum number of missing data:
MANAGER ID
```

7. Find the number total of missing values in the DataFrame

Code:

```
import pandas as pd
df= pd.read_csv("employees.csv")
print("Column with maximum number of missing data:")
print(df.isnull().sum().sum())
```

Output:

```
Column with maximum number of missing data:
1
```

8. Find rows with missing data

Code:

```
import pandas as pd
df= pd.read_csv("employees.csv")
print(df[df.isnull().any(axis=1)])
```

Output:

```
   EMPLOYEE_ID FIRST_NAME LAST_NAME  ... COMMISSION_PCT MANAGER_ID DEPARTMENT_II
9          100     Steven      King  ...              -        NaN            90

[1 rows x 11 columns]
```

9. Print a list of rows with missing data

Code:

```
import pandas as pd
df= pd.read_csv("employees.csv")
print(df[df.isnull().any(axis=1)])
```

Output:

```
   EMPLOYEE_ID FIRST_NAME LAST_NAME  ... COMMISSION_PCT MANAGER_ID DEPARTMENT_II
9          100     Steven      King  ...              -         NaN            90

[1 rows x 11 columns]
```

10. Print the number of missing data per row

Code:

```
import pandas as pd
df = pd.read_csv('data.csv')
for i in range (len(df.index)):
    print("NaN in row", i, ":", df.iloc[i].isnull().sum())
```

Output:

```
NaN in row 0 : 0
NaN in row 1 : 0
NaN in row 2 : 0
NaN in row 3 : 0
NaN in row 4 : 0
NaN in row 5 : 0
NaN in row 6 : 0
NaN in row 7 : 0
NaN in row 8 : 0
NaN in row 9 : 1
```

11. Find the row with the largest number of missing data

Code:

```
import pandas as pd
df = pd.read_csv('employees.csv')
print("Row with largest number of missing data:")
print(df.isnull().sum(axis=1).idxmax())
```

Output:

```
Row with largest number of missing data:
9
```

12. Remove rows with missing data

Code:

```
import pandas as pd

df = pd.read_csv('employees.csv')

df.dropna(inplace = True)

print(df.to_string())
```

Output:

```
    EMPLOYEE_ID   FIRST_NAME    LAST_NAME
0           198       Donald     OConnell
1           199      Douglas        Grant
2           200     Jennifer       Whalen
3           201      Michael    Hartstein
4           202          Pat          Fay
5           203        Susan       Mavris
6           204      Hermann         Baer
7           205      Shelley      Higgins
8           206      William        Gietz
10          101        Neena      Kochhar
11          102          Lex      De Haan
12          103    Alexander       Hunold
```

CONCLUSION:

The operations are performed on the data set containing missing values.

WEEK 4

Program 1

AIM:

Implement perceptron learning algorithm and find out final weight vector.

Input 1:

N1(0,0,0), P1(0,0,1),P2(0,1,0), P3(0,1,1), P4(1,0,0), P5(1,0,1), P6(1,1,0), P7(1,1,1).

Consider initial weights as W = (1,-1,0)

Input 2:

N1 [1,0,0,0] , P1 [1,0,0,1], P2 [1,0,1,0], P3 [1,0,1,1], P4 [1,1,0,0], P5 [1,1,0,1], P6 [1,1,1,0] , P7 [1,1,1,1]

w = [0,0,-1,2] initially

DESCRIPTION:

Perceptron is the smallest unit of the neural network.

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$\sum wi*xi = x1*w1 + x2*w2 + \ldots wn*xn$

Add a special term called **bias 'b'** to this weighted sum to improve the model's performance.

$\mathbf{\sum wi*xi + b}$

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

$\mathbf{Y = f(\sum wi*xi + b)}$

CODE:

```
N=int(input("Enter the number of negative values:"))
n=[[int(j) for j in input().split()] for i in range(N)]
P=int(input("Enter the number of positive values:"))
p=[[int(j) for j in input().split()] for j in range(P)]

print("Enter the weight vector:")
w=[int(j) for j in input().split()]

count=len(w)
prev=[]

while (prev!=w):
    prev=w.copy()
    for i in n:
        temp=0
        for j in range(count):
            temp+=i[j]*w[j]
        if temp>=0:
            for k in range(count):
                w[k]-=i[k]
    for i in p:
        temp=0
        for j in range(count):
            temp+=i[j]*w[j]
        if temp<0:
            for k in range(count):
                w[k]+=i[k]
print("Final weight:",w)
```

OUTPUT:

```
Enter the number of negative values:1
0 0 0
Enter the number of positive values:7
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
Enter the weight vector:
1 -1 0
Final weight: [1, 0, 0]
```

CONCLUSION: The program takes the negative and positive values and a weight vector and gives us the final weight vector.

Program 2

AIM: Implement AND, EX-OR truth table using a perceptron learning algorithm

DESCRIPTION:

In the field of Machine Learning, the Perceptron is a Supervised Learning Algorithm for binary classifiers. The Perceptron Model implements the following function:

$$\hat{y} = \Theta\left(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b\right)$$
$$= \Theta(\mathbf{w} \cdot \mathbf{x} + b)$$
$$\text{where } \Theta(v) = \begin{cases} 1 & \text{if } v \geqslant 0 \\ 0 & \text{otherwise} \end{cases}$$

CODE:

For And

```
import numpy as np
def unitStep(v):
    if v>=0:
        return 1
    else:
        return 0

def perceptronModel(x,w,b):
    v=np.dot(w,x)+b
    y=unitStep(v)
    return y

def AND(x):
    w=np.array([1,1])
    b=-1.5
    return perceptronModel(x,w,b)

test1=np.array([0,0])
```

```
test2=np.array([0,1])
test3=np.array([1,0])
test4=np.array([1,1])

print("AND ({}, {}) = {}".format(0,0,AND(test1)))
print("AND ({}, {}) = {}".format(0,1,AND(test2)))
print("AND ({}, {}) = {}".format(1,0,AND(test3)))
print("AND ({}, {}) = {}".format(1,1,AND(test4)))
```

For XOR

```
import numpy as np
def unitStep(v):
    if v>=0:
        return 1
    else:
        return 0

def perceptronModel(x,w,b):
    v=np.dot(w,x)+b
    y=unitStep(v)
    return y

def AND(x):
    w=np.array([1,1])
    b=-1.5
    return perceptronModel(x,w,b)

def NOT(x):
    w_not=-1
    b_not=0.5
```

```
    return perceptronModel(x,w_not,b_not)


def OR(x):
    w_or=np.array([1,1])
    b_or=-0.5
    return perceptronModel(x,w_or,b_or)


def XOR(x):
    y1=AND(x)
    y2=OR(x)
    y3=NOT(y1)
    final=np.array([y2,y3])
    output=AND(final)
    return output
test1=np.array([0,0])
test2=np.array([0,1])
test3=np.array([1,0])
test4=np.array([1,1])
print("XOR ({}, {}) = {}".format(0,0,XOR(test1)))
print("XOR ({}, {}) = {}".format(0,1,XOR(test2)))
print("XOR ({}, {}) = {}".format(1,0,XOR(test3)))
print("XOR ({}, {}) = {}".format(1,1,XOR(test4)))
```

OUTPUT:

```
AND (0, 0) = 0
AND (0, 1) = 0
AND (1, 0) = 0
AND (1, 1) = 1
```

```
XOR (0, 0) = 0
XOR (0, 1) = 1
XOR (1, 0) = 1
XOR (1, 1) = 0
```

CONCLUSION: Hence, it is verified that the perceptron algorithm for AND and XOR logic gates is correctly implemented.

WEEK 5

Program 1

AIM:

1. As you are aware missing values can be handled in three ways

a. Removing the whole line

b. Creating a sub model to predict those features

c. Using an automatic strategy to input them according to the other known values

However we can go with option a and option b. Applying option c, down a data set which contains numeric field, fill the data using Imputer class(Use the latest version of the class) and replace with mean, median and mode.

DESCRIPTION:

scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines.

Sklearn.impute.SimpleImputer is a library used for complete missing values with simple strategies.

The imputation strategy.

- If “mean”, then replace missing values using the mean along each column. Can only be used with numeric data.
- If “median”, then replace missing values using the median along each column. Can only be used with numeric data.
- If “most_frequent”, then replace missing using the most frequent value along each column. Can be used with strings or numeric data. If there is more than one such value, only the smallest is returned.

CODE:

Mean

```
from sklearn.impute import SimpleImputer

import numpy as np

import pandas as pd


df=pd.read_csv("employees.csv",usecols=['MANAGER_ID'])
```

```
imputer_mean=SimpleImputer(strategy="mean")
data_mean=imputer_mean.fit_transform(df)
print("Mean Imputation:\n",data_mean)
```

Median

```
from sklearn.impute import SimpleImputer
import pandas as pd
import numpy as np

df=pd.read_csv("employees.csv",usecols=['MANAGER_ID'])
imputer_median=SimpleImputer(strategy='median')
data_median=imputer_median.fit_transform(df)
print("Median Imputation:\n",data_median)
```

Mode

```
from sklearn.impute import SimpleImputer
import pandas as pd
import numpy as np

df=pd.read_csv("employees.csv",usecols=['MANAGER_ID'])
imputer_mode=SimpleImputer(strategy='most_frequent')
data_mode=imputer_mode.fit_transform(df)
print("Mode Imputation:\n",data_mode)
```

OUTPUT:

Mean

```
Mean Imputation:
 [[124.        ]
 [124.        ]
 [101.        ]
 [100.        ]
 [201.        ]
 [101.        ]
 [101.        ]
 [101.        ]
 [205.        ]
 [114.83673469]]
```

Median

```
Median Imputation:
 [[124.]
 [124.]
 [101.]
 [100.]
 [201.]
 [101.]
 [101.]
 [101.]
 [205.]
```

Mode

```
Mode Imputation:
 [[124.]
 [124.]
 [101.]
 [100.]
 [201.]
 [101.]
 [101.]
 [101.]
 [205.]
```

CONCLUSION:

The missing values are replaced by the mean, median, and mode based on the strategy.

Program 2

AIM: Download any data set which contains one categorical field, apply one hot encoding technique and print the new data set.

DESCRIPTION:

One hot encoding is one method of converting data to prepare it for an algorithm and get a better prediction. With one-hot, we convert each categorical value into a new categorical column and assign a binary value of 1 or 0 to those columns. Each integer value is represented as a binary vector.

The sklearn.preprocessing module is used to extract the function OneHotEncoder

CODE:

```
from sklearn.preprocessing import OneHotEncoder

import pandas as pd


df=pd.read_csv("employees.csv",usecols=['FIRST_NAME','LAST_NAME'])

one=OneHotEncoder(sparse_output=False)

data_one=one.fit_transform(df)

print("Encoded data:\n",data_one)
```

OUTPUT:

```
Encoded data:
 [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

CONCLUSION:

The data has been transformed into a binary vector format.

Program 3

AIM: Download any data set which contains at least one continuous data. Apply L1 norm, L2 Norm and Max norm on that column and replace data with new data.

DESCRIPTION:

The normalisation technique is used to scale the data that would be useful for the machine to process the data faster. The formulas for L1 Norm, L2 norm and Max norm

$$\begin{cases} Max\ norm: & \|X\|_{max} = \frac{X}{|max_i X|} \\ L1\ norm: & \|X\|_{L1} = \frac{X}{\sum_i |x_i|} \\ L2\ norm: & \|X\|_{L2} = \frac{X}{\sqrt{\sum_i |x_i|^2}} \end{cases}$$

CODE:

Max Norm

```
import pandas as pd


df=pd.read_csv('employees.csv')


max=df['SALARY'].max()
print(df)


df['new_salary']=df['SALARY']/max
print(new_df)
```

L1 Norm

```
import pandas as pd
df=pd.read_csv('employees.csv')
sum=df['SALARY'].sum()
data_sum=df['SALARY']/sum
print(data_sum)
```

L2 Norm

```
import pandas as pd
import numpy as np
df=pd.read_csv('employees.csv')


sum=(df['SALARY']**2).sum()
data_sum = df['SALARY']/np.sqrt(sum)
print(data_sum)
```

OUTPUT:

Max Norm

```
[50 rows x 11 columns]
0     0.420554
1     0.420554
2     0.711707
3     2.102770
4     0.970509
5     1.051385
6     1.617516
7     1.942313
8     1.342538
9     3.882038
10    2.749777
11    2.749777
```

L1 Norm

```
0     0.008411
1     0.008411
2     0.014234
3     0.042055
4     0.019410
5     0.021028
6     0.032350
7     0.038846
8     0.026851
9     0.077641
```

L2 Norm

```
0     0.047938
1     0.047938
2     0.081125
3     0.239688
4     0.110625
5     0.119844
6     0.184375
7     0.221398
8     0.153031
9     0.442501
10    0.313438
```

CONCLUSION: The normalisation technique has been applied.

WEEK 6

Program 1

AIM: Implement Id3 algorithm on 'weather.csv' dataseT.

DESCRIPTION:

D3 algorithm, stands for **Iterative Dichotomiser 3**, is a classification algorithm that follows a greedy approach of building a decision tree by selecting a best attribute that yields maximum Information Gain (IG) or minimum Entropy (H).

$$H(S) = \sum_{c \in C} -p(c) log_2 p(c)$$

Where,

- S - The current dataset for which entropy is being calculated(changes every iteration of the ID3 algorithm).
- C - Set of classes in S {example - C ={yes, no}}
- p(c) - The proportion of the number of elements in class c to the number of elements in set S.

$$IG(A,S) = H(S) - \sum_{t \in T} p(t) H(t)$$

Where,

- H(S) - Entropy of set S.
- T - The subsets created from splitting set S by attribute A such that
- p(t) - The proportion of the number of elements in t to the number of elements in set S.
- H(t) - Entropy of subset t

The **steps in ID3 algorithm** are as follows:

1. Calculate entropy for dataset.
2. For each attribute/feature.
   2.1. Calculate entropy for all its categorical values.
   2.2. Calculate information gain for the feature.
3. Find the feature with maximum information gain.
4. Repeat it until we get the desired tree.

CODE:

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("weather.csv")
features = [feat for feat in data]
features.remove("play")
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["play"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
def info_gain(examples, attr):
```

```
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain
def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
```

```
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["play"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)

    return root
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
def classify(root: Node, new):
    for child in root.children:
        if child.value == new[root.value]:
            if child.isLeaf:
                print ("Predicted Label for new example", new," is:", child.pred)
```

```
            exit
        else:
            classify (child.children[0], new)
root = ID3(data, features)
print("Decision Tree is:")
printTree(root)
print ("------------------")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"}
classify (root, new)
```

OUTPUT:

```
Decision Tree is:
outlook
        overcast ->  ['yes']
        rainy
                windy
                        False ->  ['yes']
                        True ->  ['no']
        sunny
                humidity
                        high ->  ['no']
                        normal ->  ['yes']

------------------
Predicted Label for new example {'outlook': 'sunny', 'temperature': 'hot', 'humidity': 'normal', 'wind': 'strong'}  is: ['yes']
```

CONCLUSION:

The decision tree has been identified for the given data set.

WEEK 7

Program 1

AIM: Implement naive bayes algorithm on 'weather.csv' dataset .You can use the python notebook uploaded here. Include screenshots of your results. prepare a report (code and results) as pdf .

Answer the following on buys a computer data set

1. X = (senior, High, No, Fair)
2. X = (middle-aged , Medium, No, Excellent)

Answer the following on the weather data set

1. X = (Sunny, Mild, High, True)
2. X = (Overcast, cool, High, False)

DESCRIPTION:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of weather conditions and corresponding target variable "Play". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

Convert the given dataset into frequency tables.

Generate Likelihood table by finding the probabilities of given features.

Now, use Bayes theorem to calculate the posterior probability.

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Steps to implement:

- Data Pre-processing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)

- Visualizing the test set result.

CODE:

```
from sklearn.naive_bayes import GaussianNB
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('weather.csv')
Numerics = LabelEncoder()
inputs = df.drop('play', axis='columns')
target = df['play']

inputs['outlook_n'] = Numerics.fit_transform(inputs['outlook'])
inputs['temp_n'] = Numerics.fit_transform(inputs['temperature'])
inputs['humidity_n'] = Numerics.fit_transform(inputs['humidity'])
inputs['windy_n'] = Numerics.fit_transform(inputs['windy'])
inputs_n = inputs.drop(['outlook', 'temperature', 'humidity', 'windy'], axis='columns')

print(inputs_n)

classifier = GaussianNB()
classifier.fit(inputs_n, target)

import warnings
warnings.simplefilter(action='ignore')
warnings.simplefilter(action='ignore', category=FutureWarning)

print(classifier.predict([[2, 2, 0, 1]]))
print(classifier.predict([[0, 0, 0, 0]]))
```

OUTPUT:

```
    outlook_n  temp_n  humidity_n  windy_n
0           0       1           0        0
1           0       0           1        1
2           0       2           0        1
3           0       1           1        0
4           1       2           0        0
5           1       0           1        0
6           1       0           1        1
7           1       2           1        0
8           1       2           0        1
9           2       1           0        0
10          2       1           0        1
11          2       2           0        0
12          2       0           1        0
13          2       2           1        1
When input X = ( Sunny, Mild, High, True)
['no']
When input X = (Overcast, cool, High, False)
['yes']
```

CODE:

```
from sklearn.naive_bayes import GaussianNB
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df=pd.read_csv('buys_computer.csv')
print(df)

n=LabelEncoder()
inputs=df.drop('Buys',axis='columns')
target=df['Buys']

inputs['age_n']=n.fit_transform(inputs['Age'])
inputs['income_n']=n.fit_transform(inputs['Income'])
inputs['student_n']=n.fit_transform(inputs['Student'])
inputs['Credit_rating_n']=n.fit_transform(inputs['Credit_rating'])
inputs_n = inputs.drop(['RID','Age', 'Income', 'Student', 'Credit_rating'], axis='columns')

print(inputs_n)
```

```
classifier = GaussianNB()
classifier.fit(inputs_n, target)

import warnings
warnings.simplefilter(action='ignore')
warnings.simplefilter(action='ignore', category=FutureWarning)

print("When input X = (senior, High, No, Fair)")
print(classifier.predict([[1, 0, 0, 1]]))

print("When input X = (middle-aged , Medium, No, Excellent)")
print(classifier.predict([[0, 1, 0, 0]]))
```

OUTPUT:

```
    RID          Age  Income Student Credit_rating Buys
0     1        Youth    High      No          Fair   No
1     2        Youth    High      No     Excellent   No
2     3  Middle aged    High      No          Fair  Yes
3     4       Senior  Medium      No          Fair  Yes
4     5       Senior     Low     Yes          Fair  Yes
5     6       Senior     Low     Yes     Excellent   No
6     7  Middle aged     Low     Yes     Excellent  Yes
7     8        Youth  Medium      No          Fair   No
8     9        Youth     Low     Yes          Fair  Yes
9    10       Senior  Medium     Yes          Fair  Yes
10   11        Youth  Medium     Yes     Excellent  Yes
11   12  Middle aged  Medium      No     Excellent  Yes
12   13  Middle aged    High     Yes          Fair  Yes
13   14       Senior  Medium      No     Excellent   No
    age_n  income_n  student_n  Credit_rating_n
0       2         0          0                1
1       2         0          0                0
2       0         0          0                1
3       1         2          0                1
4       1         1          1                1
5       1         1          1                0
6       0         1          1                0
7       2         2          0                1
8       2         1          1                1
9       1         2          1                1
10      2         2          1                0
11      0         2          0                0
12      0         0          1                1
13      1         2          0                0
When input X = (senior, High, No, Fair)
['No']
When input X = (middle-aged , Medium, No, Excellent)
['Yes']
```

CONCLUSION:

The output for the values has been predicted.

WEEK 8

Program 1

AIM: To apply KNN classifier on the above data set and predict the values for the given input

X = (5.2,2.8)
X = (5.6,2.7)
X = (4.9 , 2.4)

DESCRIPTION:

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for classification and regression tasks. In the context of classification, KNN works by finding the K closest neighbors of a given test data point in the feature space, based on a chosen distance metric (e.g., Euclidean distance). The class of the test data point is then determined by majority voting among the K neighbors, where each neighbor's vote is weighted by its proximity to the test point.

Here are the steps involved in implementing a KNN classifier:

1. Choose the number of neighbors (K) to consider.
2. Calculate the distance between the test data point and all the training data points.
3. Select the K-nearest data points based on the calculated distances.
4. Determine the class of the test data point based on the majority class among the K-nearest data points.
5. Repeat steps 2-4 for all test data points.

KNN is a simple and intuitive algorithm that can work well on small datasets or when the decision boundary is highly irregular. However, it can be computationally expensive for large datasets and may not perform well when the feature space is high-dimensional. Additionally, choosing the optimal value for K can be challenging and can impact the performance of the algorithm.

CODE:

```
import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

df=pd.read_csv('week.csv')

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

y=df.iloc[:,-1].values
```

```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(x,y)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
accuracy=knn.score(x_test,y_test)
print("Accuracy: ",accuracy)
new=[[5.2,2.8],[5.6,2.7],[4.9 , 2.4]]
predict=knn.predict(new)
print("Predictions when the input is: [5.2,2.8],[5.6,2.7],[4.9 , 2.4]")
print(predict)
```

OUTPUT:

```
Accuracy:  0.8
Predictions when the input is: [5.2,2.8],[5.6,2.7],[4.9 , 2.4]
['setosa' 'verscicolor' 'setosa']
```

CONCLUSION:

The accuracy and the predictions have been made using the knn classifier.

PROGRAM 2

AIM: To apply clustering algorithms – kmeans, agglomerative and DBSCAN to classify for some data sets.

DESCRIPTION:

K-means is an unsupervised machine learning algorithm used for clustering data points into K distinct groups or clusters based on their similarity. The goal of K-means is to partition the input data into K clusters, where each cluster represents a group of data points that are similar to each other and dissimilar to data points in other clusters.

Agglomerative clustering is a hierarchical clustering algorithm used in unsupervised machine learning to group similar data points into clusters based on a chosen distance metric. The algorithm starts by treating each data point as a separate cluster and iteratively merges the closest pairs of clusters until only a single cluster remains.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm used in unsupervised machine learning to group together closely packed data points into clusters, while also identifying and excluding outliers or noise points.

DBSCAN works by grouping together data points that are located in high-density regions and separating them from data points that are located in low-density regions. The algorithm defines two key parameters: the minimum number of points (minPts) required to form a dense region, and a maximum distance (ε or eps) within which points are considered to be neighbors.

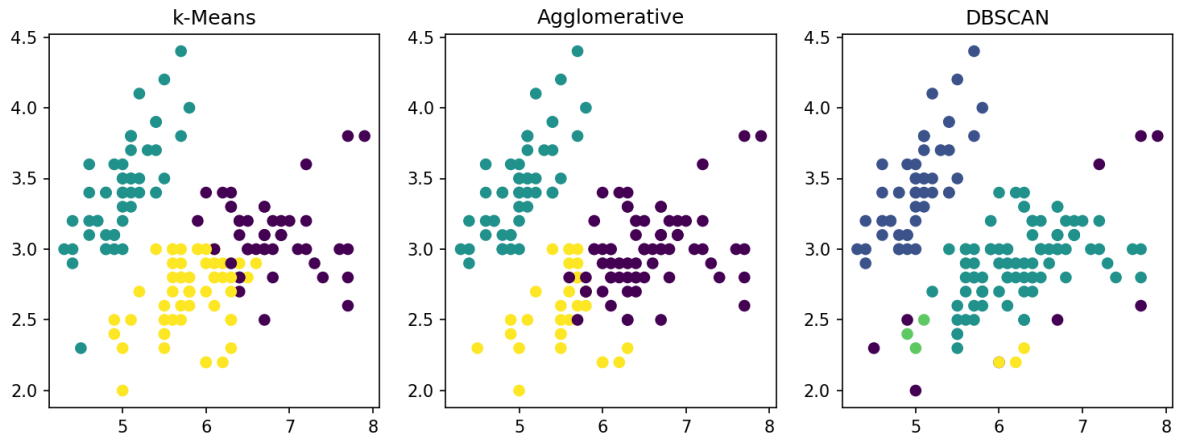
PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
iris = load_iris()
X = iris.data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)
kmeans_labels = kmeans.fit_predict(X_std)
agglo = AgglomerativeClustering(n_clusters=3)
agglo_labels = agglo.fit_predict(X_std)
dbscan = DBSCAN(eps=0.6, min_samples=3)
dbscan_labels = dbscan.fit_predict(X_std)
print("k-Means silhouette score:", silhouette_score(X_std, kmeans_labels))
print("Agglomerative silhouette score:", silhouette_score(X_std, agglo_labels))
```

```
print("DBSCAN silhouette score:", silhouette_score(X_std, dbscan_labels))
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 4))
ax1.scatter(X[:, 0], X[:, 1], c=kmeans_labels)
ax1.set_title("k-Means")
ax2.scatter(X[:, 0], X[:, 1], c=agglo_labels)
ax2.set_title("Agglomerative")
ax3.scatter(X[:, 0], X[:, 1], c=dbscan_labels)
ax3.set_title("DBSCAN")
plt.show()
```

OUTPUT:

```
k-Means silhouette score: 0.45994823920518635
Agglomerative silhouette score: 0.4466890410285909
DBSCAN silhouette score: 0.29516241784322833
```



WEEK 9

Program 1

AIM: To demonstrate naïve Bayesian classifier for a sample training dataset and calculate the accuracy, precision and recall for that dataset

DESCRIPTION:

Naive Bayes classifier is a probabilistic machine learning algorithm used for classification tasks. It is based on Bayes' theorem and the assumption of independence between the features. The algorithm calculates the probability of a given data point belonging to a particular class based on its feature values and the prior probability of each class.

Accuracy is the proportion of correct predictions made by the model among all the predictions made. It is calculated as:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

where TP (True Positive) is the number of correct positive predictions, TN (True Negative) is the number of correct negative predictions, FP (False Positive) is the number of incorrect positive predictions, and FN (False Negative) is the number of incorrect negative predictions.

Precision is the proportion of correct positive predictions made by the model among all the positive predictions made. It is calculated as:

$$\text{Precision} = TP / (TP + FP)$$

Recall is the proportion of correctly predicted positive instances among all the actual positive instances. It is calculated as:

$$\text{Recall} = TP / (TP + FN)$$

In summary, accuracy measures the overall correctness of the model's predictions, precision measures the model's ability to correctly predict positive instances, and recall measures the model's ability to correctly identify all positive instances.

PROGRAM:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
iris = load_iris()
iris
print(iris.data.shape)
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3, random_s
tate=42)
clf = GaussianNB()
```

```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(y_pred)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
new_data = [[5.1, 3.5, 1.4, 0.2], [6.2, 2.9, 4.3, 1.3], [7.6, 3.0, 6.6, 2.1]]
new_predictions = clf.predict(new_data)
print("New predictions:", new_predictions)
```

OUTPUT:

```
Accuracy: 0.9777777777777777
Precision: 0.9793650793650793
Recall: 0.9777777777777777
F1 score: 0.9777448559670783
New predictions: [0 1 2]
```

CONCLUSION:

The classification metrics have been calculated for the given dataset.

WEEK 10

AIM:

To apply the Logistic regression, ID3, Random forest, XG Boost, Naïve Bayes Algorithm models on the dataset and compare the accuracy and plot the appropriate graphs.

DESCRIPTION:

- **Logistic Regression**: Logistic regression is a statistical method used to analyze the relationship between a binary dependent variable and one or more independent variables. It is commonly used in predictive modeling and machine learning applications to predict the probability of a binary outcome based on a set of input variables.

- **The ID3 algorithm** uses a tree structure to represent the decision-making process. The root of the tree represents the initial dataset, and each internal node represents a test on an attribute. The branches that emanate from the node correspond to the possible values of the attribute, and each leaf node represents a class label.

- The **random forest** algorithm works by creating a large number of decision trees and aggregating their predictions. Each decision tree is constructed by randomly selecting a subset of the training data and a subset of the input features. The tree is then grown by recursively splitting the data based on the feature that provides the most information gain, using a greedy algorithm.

- **XGBoost** (Extreme Gradient Boosting) is a machine learning algorithm that is used for supervised learning problems, such as classification and regression. It is an ensemble learning method that combines the predictions of multiple decision trees to make a final prediction. XGBoost works by iteratively building decision trees in a greedy fashion, where each new tree attempts to correct the mistakes of the previous trees.

- **Naive Bayes** is a machine learning algorithm used for classification tasks, such as text classification and spam filtering. It is a probabilistic algorithm that uses Bayes' theorem to make predictions.Naive Bayes works by calculating the probability of each class based on the input features and selecting the class with the highest probability as the predicted class. The algorithm assumes that the input features are conditionally independent of each other, given the class. This assumption simplifies the computation and makes the algorithm very fast and scalable.

PROGRAM:

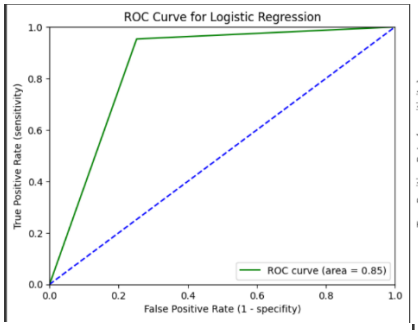
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import metrics

from sklearn.preprocessing import LabelEncoder
```

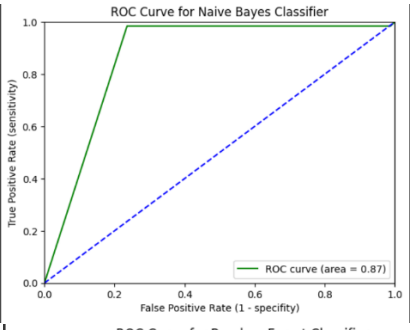
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_curve, confusion_matrix,
classification_report, auc
from xgboost.sklearn import XGBClassifier

data = pd.read_csv('seattle-weather.csv')
data = data.drop('date',axis=1)
le = LabelEncoder()
x = data.drop('weather',axis=1)
y = data['weather']
y = le.fit_transform(y)
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3, random_state = 42)
model_dict = {}
model_dict['Logistic Regression'] = LogisticRegression(solver='liblinear', random_state=42)
model_dict['Naive Bayes Classifier'] = GaussianNB()
model_dict['Decision Tree Classifier'] = DecisionTreeClassifier(random_state=42)
model_dict['Random Forest Classifier'] = RandomForestClassifier(random_state=42)
model_dict['XGB Classifier'] = XGBClassifier(random_state=42)
def model_test(x_train,y_train,x_test,y_test,model,model_name):
    model.fit(x_train,y_train)
    y_pred = model.predict(x_test)
    accuracy = accuracy_score(y_test,y_pred)
    print("========{}========".format(model_name))
    print("Score is: {}".format(accuracy))
    print()

for model_name, model in model_dict.items():
```

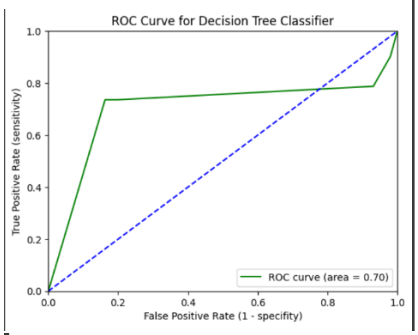
```
    model_test(x_train,y_train,x_test,y_test,model,model_name)


def Rocplot(x_train,y_train,x_test,y_test,model,model_name):
  model.fit(x_train,y_train)
  pred_res = model.predict(x_test)
  fpr_res,tpr_res,thresholds_res = roc_curve(y_test,pred_res,pos_label=4)
  roc_auc_res = metrics.auc(fpr_res, tpr_res)
  plt.plot(fpr_res, tpr_res,color='green', label='ROC curve (area = %0.2f)' % roc_auc_res)
  plt.plot([0,1],[0,1],color='blue',linestyle='--')
  plt.xlim([0.0,1.0])
  plt.ylim([0.0,1.0])
  plt.title('ROC Curve for '+model_name)
  plt.xlabel('False Positive Rate (1 - specifity)')
  plt.ylabel('True Positive Rate (sensitivity)')
  plt.legend(loc="lower right")
  plt.show()
  roc_curve(y_test,pred_res,pos_label=4)


for model_name, model in model_dict.items():
  Rocplot(x_train,y_train,x_test,y_test,model,model_name)
```

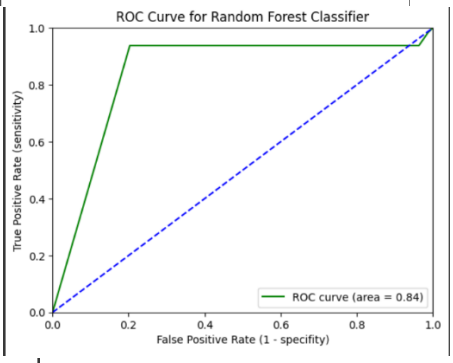
OUTPUT:

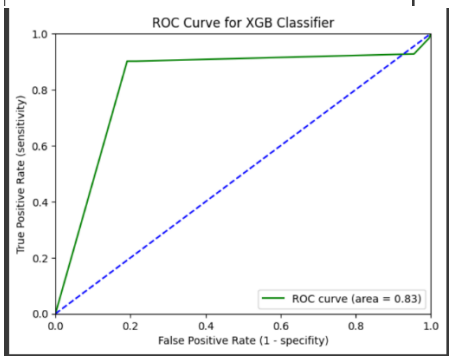
```
========Logistic Regression========
Score is: 0.8109339407744874

========Naive Bayes Classifier========
Score is: 0.8405466970387244

========Decision Tree Classifier========
Score is: 0.7425968109339408

========Random Forest Classifier========
Score is: 0.8314350797266514

========XGB Classifier========
Score is: 0.8223234624145785
```


ROC Curve for Logistic Regression
True Positive Rate (sensitivity)
False Positive Rate (1 - specifity)
ROC curve (area = 0.85)


ROC Curve for Naive Bayes Classifier
True Positive Rate (sensitivity)
False Positive Rate (1 - specifity)
ROC curve (area = 0.87)


ROC Curve for Decision Tree Classifier
True Positive Rate (sensitivity)
False Positive Rate (1 - specifity)
ROC curve (area = 0.70)


ROC Curve for Random Forest Classifier
True Positive Rate (sensitivity)
False Positive Rate (1 - specifity)
ROC curve (area = 0.84)


ROC Curve for XGB Classifier
True Positive Rate (sensitivity)
False Positive Rate (1 - specifity)
ROC curve (area = 0.83)