WEEK 1

1.AIM: Create a class Circle and initialize it with radius value. Make two methods get Area and get Circumference inside this class. And calculate area and Circumference then print.

DESCRIPTION: The functions of area and circumference of the circle are defined and the arguments related to it are passed into it. The obtained solution is printed.

CODE:

```
class Circle:
  def __init__(self, radius):
    self.r = radius
  def Area(self):
    print("Area = ",(3.14)*(self.r)**2)
  def Circumference(self):
    print("Circumference = ",2*(3.14)*(self.r))
n = int(input("Enter radius: "))
c = Circle(n)
c.Area()
c.Circumference()
```

OUTPUT:

```
Enter radius: 5
Area =  78.5
Circumference =  31.400000000000002
```

2.AIM: Create a Temperature class. Create two methods:
a. convertFahrenheit - It will take celsius and will print it into Fahrenheit.
b. convertCelsius - It will take Fahrenheit and will convert it into Celsius.

DESCRIPTION: The functions of convertFahrenheit and convertCelsius are defined and the arguments related to it are passed into it. The obtained solution is printed.

CODE:

```
class Temperature:
  def ConverttoF(self, Ctemp):
    print("Temp in Fahrenheit - ",Ctemp*9/5+32)
  def ConverttoC(self, Ftemp):
    print("Temp in Celsius - ",(Ftemp-32)*5/9)
```

```
T = Temperature()

n1 = float(input("Enter Temp in Celsius: "))

T.ConverttoF(n1)

n2 = float(input("Enter Temp in Fahrenheit: "))

T.ConverttoC(n2)
```

OUTPUT:

```
Enter Temp in Celsius: 98.2
Temp in Fahrenheit -  208.76000000000002
Enter Temp in Fahrenheit: 300
Temp in Celsius -  148.88888888888889
```

3.AIM: Create a student class and initialize it with name and roll number. Create methods:
a. Display - It should display all information of the student.
b. setAge - It should assign age to student
c. setMarks - It should assign marks to the student.

DESCRIPTION: The functions of Display, setAge and setMarks are defined and the arguments related to it are passed into it. The obtained solution is printed.

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("ML Lab", 1007)
s.setAge(20)
s.setmarks(99)
s.display()
```

OUTPUT:

```
C:\Users\91630\Deskt
Student Details
---------------
Name =  ML lab
Roll Number 1007
Age =  20
Marks =  99
```

4.AIM: Create a Time class and initialize it with hours and minutes.
a. Make a method addTime which should take two time object and add them. E.g.- (2 hour and 50 min)+(1 hr and 20 min) is (4 hr and 10 min)
b. Make a method displayTime which should print the time.
c. Make a method DisplayMinute which should display the total minutes in the Time. E.g.- (1 hr 2 min) should display 62 minute.

DESCRIPTION: The functions of addTime, displayTime and Display Minute are defined and the arguments related to it are passed into it. The obtained solution is printed.

CODE:

```
class Time:
  def __init__(self, hour, min):
    self.hour = hour
    self.min = min
  def addTime(t1, t2):
    t3 = Time(0, 0)
    t3.hour = t1.hour + t2.hour
    t3.min = t1.min + t2.min
    while t3.min >= 60:
      t3.hour += 1
      t3.min -= 60
    return t3
  def displayTime(self):
```

```
    print("Time is %d hours and %d minutes" %(self.hour, self.min))
  def displayMin(self):
      print((self.hour * 60) + self.min, "minutes")
a = Time(1, 23)
b = Time(1, 52)
c = Time.addTime(a,b)
c.displayTime()
c.displayMin()
```

OUTPUT:

```
C:\Users\91630\Desktop\ML lab>pytho
Time is 3 hours and 15 minutes
195 minutes
```

WEEK 2

1.AIM: 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.

DESCRIPTION:

Min-max normalization (usually called feature scaling) performs a linear transformation on the original data. This technique gets all the scaled data in the range (0, 1). The formula to achieve this is the following:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Min-max normalization preserves the relationships among the original data values. The cost of having this bounded range is that we will end up with smaller standard deviations, which can suppress the effect of outliers.

Z-score normalization in data mining is useful for those kinds of data analysis wherein there is a need to compare a value with respect to a mean(average) value, such as results from tests or surveys. Thus, Z-score normalization is also popularly known as Standardization. The following formula is used in the case of z-score normalization on every single value of the dataset: New value = $(x - \mu) / \sigma$

Here: x: Original value, μ: Mean of data, σ: Standard deviation of data

Decimal scaling is a data normalization technique like Z score, Min-Max, and normalization with standard deviation. Decimal scaling is a data normalization technique. In this technique, we move the decimal point of values of the attribute. This movement of decimal points totally depends on the maximum value among all values in the attribute. A value v of attribute A is can be normalized by the following formula: Normalized value of attribute = $(v^i / 10^j)$

CODE:

```
from scipy.stats import zscore

import math

import pandas as pd

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

column = df["SALARY"]

meanSalary = df["SALARY"].mean()

df.fillna(meanSalary, inplace=True)


# min-max normalization

new_min, new_max = [int(x) for x in input("Enter the new min and new max value for minmax nomralization: ").split()]

min_max_normalized_values = ((column - column.min()) / (column.max() - column.min())) * (new_max - new_min) + new_min
```

```
# Z-score normalization
z_score_normalized_values = zscore(list(df["SALARY"]))


# decimal normalization
n = math.ceil(math.log(column.max(), 10))
decimal_normalized_values = column / 10**n
print("\nMin-Max Normalized Values:\n", min_max_normalized_values)
print("\nZ-Score Normalized Values:\n", z_score_normalized_values)
print("\nDecimal Normalized Values:\n", decimal_normalized_values)
# print(column.min(), column.max())
```

OUTPUT:

```
Enter the new min and new max value for minmax nomralization: 0 1

Min-Max Normalized Values:
 0     0.022831
1     0.022831
2     0.105023
3     0.497717
4     0.178082
5     0.200913
6     0.360731
7     0.452420
8     0.283105
9     1.000000
10    0.680365
11    0.680365
12    0.315068
13    0.178082
14    0.123288
15    0.123288
16    0.095890
17    0.452420
18    0.315068
19    0.278539
20    0.255708
21    0.260274
22    0.219178
23    0.406393
24    0.045662
25    0.036530
26    0.031963
27    0.022831
28    0.018265
29    0.269406
30    0.278539
31    0.264840
32    0.200913
33    0.168950
34    0.050228
35    0.027397
36    0.013699
37    0.004566
```

```
Z-Score Normalized Values:
 [-0.81105699 -0.81105699 -0.41190052  1.49518042 -0.05709476  0.05378204
  0.82991962  1.27520085  0.45293851  3.93446998  2.3821948   2.3821948
  0.60816603 -0.05709476 -0.32319908 -0.32319908 -0.45625124  1.27520085
  0.60816603  0.43076315  0.31988635  0.34206171  0.14248347  1.05167322
 -0.70018019 -0.74453091 -0.76670627 -0.81105699 -0.83323235  0.38641243
  0.43076315  0.36423707  0.05378204 -0.10144548 -0.67800483 -0.78888163
 -0.85540771 -0.89975843 -0.65582947 -0.76670627 -0.83323235 -0.92193379
 -0.65582947 -0.74453091 -0.85540771 -0.89975843 -0.5893034  -0.67800483
 -0.78888163  0.        ]

Decimal Normalized Values:
 0     0.026000
1     0.026000
2     0.044000
3     0.130000
4     0.060000
5     0.065000
6     0.100000
7     0.120080
39    0.028000
40    0.025000
41    0.021000
42    0.033000
43    0.029000
44    0.024000
45    0.022000
46    0.036000
47    0.032000
48    0.027000
49    0.062575
Name: SALARY, dtype: float64
```

2. Down a data set from Kaggle which contains at least one feature as numeric or continuous data.

a.AIM: Get the nrows, ncolumns, datatype, summary stats of each column of a dataframe?

DESCRIPTION: To get the number of rows and columns, data types, and summary statistics of each column in a data frame, you can use the info() and describe() methods of the data frame. The info() method provides a concise summary of the data frame, including the number of non-null values and data types of each column. The describe() method provides summary statistics of the numerical columns in the data frame, including count, mean, standard deviation, minimum, maximum, and quartile values. Note that the describe() method only includes numerical columns by default.

CODE:

```
# Get the number of rows and columns

nrows, ncolumns = df.shape

print("Number of rows: ", nrows)

print("Number of columns: ", ncolumns)


# Get the data type of each column

print("\nData type of each column: \n", df.dtypes)


# Get the summary statistics of each column

print("\nSummary statistics of each column: \n", df.describe())
```

OUTPUT:

```
Number of rows: 50
Number of columns: 11
Data types:
Summary statistics:
       EMPLOYEE_ID        SALARY  MANAGER_ID  DEPARTMENT_ID
count    50.000000     49.000000   49.000000      48.000000
mean    134.760000   6257.469388  114.836735      57.500000
std      33.631594   4602.499082   20.591611      25.640726
min     100.000000   2100.000000  100.000000      10.000000
25%     112.250000   2800.000000  101.000000      50.000000
50%     124.500000   4800.000000  114.000000      50.000000
75%     136.750000   8200.000000  121.000000      62.500000
max     206.000000  24000.000000  205.000000     110.000000
```

b.AIM: Count the number of missing values in each column?

DESCRIPTION: To count the number of missing values in each column of a dataframe, you can use the isnull() method to create a boolean mask indicating which values are missing, and then use the sum() method to count the number of True values in each column.

CODE:

```
# Count the number of missing values in each column

missing_values_count = df.isnull().sum()

print("Number of missing values in each column:\n", missing_values_count)
```

OUTPUT:

```
FIRST_NAME        0
LAST_NAME         0
EMAIL             0
PHONE_NUMBER      0
HIRE_DATE         0
JOB_ID            0
SALARY            1
COMMISSION_PCT    0
MANAGER_ID        1
DEPARTMENT_ID     2
dtype: int64
```

c.AIM: Rename a specific column in a dataframe?

DESCRIPTION: To rename a specific column in a dataframe, you can use the rename() method of the dataframe. To rename a specific column:

df.rename(columns={"old_column_name": "new_column_name"}, inplace=True)

CODE:

```
# Rename a specific column

df = df.rename(columns={'number_courses' : 'Number_of_Courses', 'time_study' : 'Time_Allocated'})


# Verify that the column has been renamed

print("Dataframe after renaming a specific column: \n", df.head())
```

OUTPUT:

```
C:\Users\91630\Desktop\ML lab>python -u "c:\Users\91630\Desktop\ML lab\xyz.py"
Dataframe after renaming a specific column:
   EMPLOYEE_ID FIRST_NAME  LAST_NAME     EMAIL  PHONE_NUMBER  HIRE_DATE    JOB_ID   SALARY COMMISSION_PCT  MANAGER_ID  DEPARTMENT_ID
0          198     Donald   OConnell  DOCONNEL  650.507.9833  21-Jun-07  SH_CLERK   2600.0              -       124.0           50.0
1          199    Douglas      Grant    DGRANT  650.507.9844  13-Jan-08  SH_CLERK   2600.0              -       124.0           50.0
2          200   Jennifer     Whalen   JWHALEN  515.123.4444  17-Sep-03   AD_ASST   4400.0              -       101.0           10.0
3          201    Michael  Hartstein  MHARTSTE  515.123.5555  17-Feb-04    MK_MAN  13000.0              -       100.0           20.0
4          202        Pat        Fay      PFAY  603.123.6666  17-Aug-05    MK_REP   6000.0              -       201.0           20.0
```

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

DESCRIPTION: To replace missing values of multiple numeric columns with the mean of each column, you can use the fillna() method of the dataframe along with the mean() method.

CODE:

```
# Replace missing values of multiple numeric columns with their respective mean

numeric_columns = df.select_dtypes(include=[np.number]).columns

df[numeric_columns] = df[numeric_columns].fillna(df[numeric_columns].mean())
```

Verify that the missing values have been replaced

print("Dataframe after replacing missing values with mean: \n", df.head())

OUTPUT:

```
Dataframe after replacing missing values with mean:
   EMPLOYEE_ID FIRST_NAME  LAST_NAME     EMAIL  PHONE_NUMBER  HIRE_DATE    JOB_ID   SALARY COMMISSION_PCT  MANAGER_ID  DEPARTMENT_ID
0          198     Donald   OConnell  DOCONNEL  650.507.9833  21-Jun-07  SH_CLERK   2600.0              -       124.0           50.0
1          199    Douglas      Grant    DGRANT  650.507.9844  13-Jan-08  SH_CLERK   2600.0              -       124.0           50.0
2          200   Jennifer     Whalen   JWHALEN  515.123.4444  17-Sep-03   AD_ASST   4400.0              -       101.0           10.0
3          201    Michael  Hartstein  MHARTSTE  515.123.5555  17-Feb-04    MK_MAN  13000.0              -       100.0           20.0
4          202        Pat        Fay      PFAY  603.123.6666  17-Aug-05    MK_REP   6000.0              -       201.0           20.0

C:\Users\91630\Desktop\ML lab>
```

e.AIM: Change the order of columns of a dataframe?

DESCRIPTION: To change the order of columns in a dataframe, you can use the reindex() method of the dataframe.

CODE:

Get the current column order

current_column_order = df.columns

Define the desired column order

new_column_order = ['Marks', 'Number_of_Courses', 'Time_Allocated']

Change the order of columns

df = df[new_column_order]

Verify that the order of columns has been changed

print("Dataframe with new column order: \n", df.head())

OUTPUT:

```
C:\Users\91630\Desktop\ML lab>python -u  c:\Users\91630\Desktop\ML lab\xyz.py
   JOB_ID     EMAIL EMPLOYEE_ID FIRST_NAME      LAST_NAME PHONE_NUMBER HIRE_DATE   SALARY MANAGER_ID  DEPARTMENT_ID  COMMISSION_PCT
0     198    Donald    OConnell   DOCONNEL  650.507.9833     21-Jun-07  SH_CLERK   2600.0          -          124.0            50.0
1     199   Douglas       Grant     DGRANT  650.507.9844     13-Jan-08  SH_CLERK   2600.0          -          124.0            50.0
2     200  Jennifer      Whalen    JWHALEN  515.123.4444     17-Sep-03   AD_ASST   4400.0          -          101.0            10.0
3     201   Michael   Hartstein   MHARTSTE  515.123.5555     17-Feb-04    MK_MAN  13000.0          -          100.0            20.0
4     202       Pat         Fay       PFAY  603.123.6666     17-Aug-05    MK_REP   6000.0          -          201.0            20.0
```

WEEK 3

1.AIM: Extract rows with missing values for a specific column, use isnull() for that column.

DESCRIPTION: The above points describe the steps to extract rows with missing values for a specific column in a Pandas DataFrame: Load the data into a Pandas DataFrame. Use the isnull() method on the column you want to check for missing values to create a boolean mask. Use this boolean mask to filter the DataFrame and extract only the rows where the value is missing for the specific column.

CODE:

```
import pandas as pd

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

print(df[df['DEPARTMENT_ID'].isnull()])
```

OUTPUT:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
15	106	Valli	Pataballa	VPATABAL	590.423.4560	05-Feb-06	IT_PROG	4800.0	-	103.0	NaN
16	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-Feb-07	IT_PROG	4200.0	-	103.0	NaN

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

DESCRIPTION: To extract columns that contain at least one missing value using pandas, you need to:

1. Import the pandas library
2. Read in the data into a pandas DataFrame
3. Use the isnull() method to create a boolean DataFrame where True indicates a missing value
4. Use the any() method with axis=0 to determine which columns contain at least one missing value
5. Subset the original DataFrame using boolean indexing to extract only the columns with missing values, if desired.
6. In summary, pandas provides a straightforward way to identify columns with missing values in a DataFrame, which is useful for data cleaning and exploration.

CODE:

```
df2 = df.dropna(how='all').dropna(how='all', axis=1)

print(df2.loc[:, df2.isnull().any()])
```

OUTPUT:

	SALARY	MANAGER_ID	DEPARTMENT_ID
0	2600.0	124.0	50.0
1	2600.0	124.0	50.0
2	4400.0	101.0	10.0
3	13000.0	100.0	20.0
4	6000.0	201.0	20.0
5	6500.0	101.0	40.0
6	10000.0	101.0	70.0
7	12008.0	101.0	110.0
8	8300.0	205.0	110.0
9	24000.0	NaN	90.0
10	17000.0	100.0	90.0
11	17000.0	100.0	90.0
41	2100.0	121.0	50.0
42	3300.0	122.0	50.0
43	2900.0	122.0	50.0
44	2400.0	122.0	50.0
45	2200.0	122.0	50.0
46	3600.0	123.0	50.0
47	3200.0	123.0	50.0
48	2700.0	123.0	50.0
49	NaN	123.0	50.0

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

DESCRIPTION: The first step is to import the pandas library and read the dataset as a pandas DataFrame. The second step involves using the any() method to create a boolean mask that identifies the rows with at least one missing value. The any() method returns a boolean value indicating whether any element along a given axis is missing or not. The isnull() method is used to check for missing values in the DataFrame, and the any() method is applied along the rows by setting axis=1.The third step is to use the boolean mask to filter the original DataFrame and extract the rows with missing values. This is achieved by indexing the original DataFrame with the boolean mask created in step two. Finally, a new DataFrame with only the rows that contain at least one missing value is created, which can be further used for analysis or data cleaning purposes.

CODE:

```
print(df2[df2.isnull().any(axis=1)])
```

OUTPUT:

```
    EMPLOYEE_ID FIRST_NAME  LAST_NAME     EMAIL  PHONE_NUMBER  HIRE_DATE    JOB_ID   SALARY COMMISSION_PCT  MANAGER_ID  DEPARTMENT_ID
9           100     Steven       King     SKING  515.123.4567  17-Jun-03   AD_PRES  24000.0              -         NaN           90.0
15          106      Valli  Pataballa  VPATABAL  590.423.4560  05-Feb-06   IT_PROG   4800.0              -       103.0            NaN
16          107      Diana    Lorentz  DLORENTZ  590.423.5567  07-Feb-07   IT_PROG   4200.0              -       103.0            NaN
49          140     Joshua      Patel    JPATEL  650.121.1834  06-Apr-06  ST_CLERK      NaN              -       123.0           50.0
```

4.AIM: Find a list of columns with missing data.

DESCRIPTION: The steps to find a list of columns with missing data in a Pandas DataFrame using the any() method. The steps include importing the Pandas library, loading the data into a DataFrame, using the isna() method to create a DataFrame with boolean values indicating missing values, using the any() method to create a boolean Series indicating whether each column has missing data, filtering the column names with missing data using the boolean Series, and finally printing the list of columns with missing data to the console.

CODE:

```
print(df.isna().any())
```

OUTPUT:

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

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

DESCRIPTION:

1. Import the pandas library using the command import pandas as pd.
2. Load the dataset into a pandas dataframe using the read_csv function or any other appropriate method.
3. Use the isnal() function to create a boolean dataframe that indicates whether each value in the dataframe is missing or not.
4. Use the sum() function to count the number of missing values per column.
5. Print the resulting pandas series object to see the number of missing values per column.

CODE:

```
print(df.isna().sum())
```

OUTPUT:

```
C:\Users\91630\Desktop\ML I
EMPLOYEE_ID       0
FIRST_NAME        0
LAST_NAME         0
EMAIL             0
PHONE_NUMBER      0
HIRE_DATE         0
JOB_ID            0
SALARY            1
COMMISSION_PCT    0
MANAGER_ID        1
DEPARTMENT_ID     2
dtype: int64
```

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

1. **DESCRIPTION:** Import the Pandas library and read in your dataset using the relevant function.
2. Create a boolean DataFrame that identifies missing values in your dataset using the isnull method.
3. Count the number of missing values in each column of the boolean DataFrame using the sum method.
4. Identify the column with the maximum number of missing values using the idxmax method, which returns the index of the first occurrence of the maximum value in the DataFrame.

CODE:

```
print(df.isna().sum().idxmax())
```

OUTPUT:

```
DEPARTMENT_ID
```

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

DESCRIPTION: To find the total number of missing values in a pandas DataFrame, we can use the isnull() method to create a boolean mask indicating where each value in the DataFrame is missing. We can then use the sum() method to count the number of True values in the boolean mask, which will give us the total number of missing values in the DataFrame. Finally, we can print or store the result as a single integer value.

CODE:

```
print(df.isna().sum().sum())
```

OUTPUT:

```
C:\User
4
```

8.AIM: Find rows with missing data.

DESCRIPTION: To find rows with missing data in a Pandas DataFrame using the isnull() or isna() function. These functions return a boolean mask that identifies where the missing values are located in the DataFrame. By using the any() function with the axis=1 argument, we can check if there are any missing values in each row and select the rows with missing data from the original DataFrame.

CODE:

```
df.isnull().sum(axis=1)
```

OUTPUT:

```
C:\Users\91630\Desktop\ML lab>pyth
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     1
10    0
11    0
12    0
32    0
33    0
34    0
35    0
36    0
37    0
38    0
39    0
40    0
41    0
42    0
43    0
44    0
45    0
46    0
47    0
48    0
49    1
dtype: int64
```

9.AIM: Print a list of rows with missing data.

DESCRIPTION: To use pandas to print a list of rows with missing data in a dataset. First, the pandas library is imported and the dataset is loaded into a pandas dataframe. Then, missing values in the dataset are identified by checking if any row contains at least one missing value. The resulting boolean series is used to filter the original dataframe and create a new dataframe that contains only the rows with missing values. Finally, the resulting dataframe is printed to show the list of rows with missing data.

CODE:

print(df2[df2.isnull().any(axis=1)])

OUTPUT:

```
    EMPLOYEE_ID FIRST_NAME  LAST_NAME     EMAIL  ...   SALARY COMMISSION_PCT MANAGER_ID  DEPARTMENT_ID
9           100     Steven       King     SKING  ...  24000.0              -        NaN           90.0
15          106      Valli  Pataballa  VPATABAL  ...   4800.0              -      103.0            NaN
16          107      Diana    Lorentz  DLORENTZ  ...   4200.0              -      103.0            NaN
49          140     Joshua      Patel    JPATEL  ...      NaN              -      123.0           50.0

[4 rows x 11 columns]
```

10.AIM: Print the number of missing data per row.

DESCRIPTION:

1. Import pandas library
2. Create a DataFrame object containing your data
3. Call the isnull() method on the DataFrame to create a Boolean DataFrame where missing values are True and non-missing values are False
4. Call the sum() method on the Boolean DataFrame to count the number of True values per row
5. Print the resulting missing_per_row object to see the number of missing values per row.

CODE:

data.apply(lambda x:x isnull().sum(),axis=1)

OUTPUT:

```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     1
10    0
11    0
12    0
21    0
22    0
23    0
```

```
24    0
25    0
26    0
27    0
28    0
29    0
30    0
31    0
32    0
33    0
34    0
35    0
36    0
37    0
38    0
39    0
40    0
41    0
42    0
43    0
44    0
45    0
46    0
```

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

DESCRIPTION: To find the row with the largest number of missing data using pandas, you can first import the pandas library and load your data into a DataFrame. Then you can count the number of missing values in each row using the isna() method and sum them using the sum() method with axis=1. You can then use the idxmax() method to find the index of the row with the largest number of missing values, and use the iloc[] method with that index to retrieve the row data. This will give you the row with the largest number of missing values.

CODE:

```
# Count the number of missing values in each row
missing_count = df.isna().sum(axis=1)


# Find the index of the row with the largest number of missing data
max_missing_row_index = missing_count.idxmax()


# Print the row with the largest number of missing data
print(df.loc[max_missing_row_index])
```

OUTPUT:

```
PHONE_NUMBER      515.123.4567
HIRE_DATE            17-Jun-03
JOB_ID                 AD_PRES
SALARY                 24000.0
COMMISSION_PCT               -
MANAGER_ID                 NaN
DEPARTMENT_ID             90.0
Name: 9, dtype: object
```

12.AIM: Remove rows with missing data.

DESCRIPTION: To remove rows with missing data from a dataframe, you can use the dropna() method.

CODE:

```
data.dropna(axis=0, inplace=True)
```

OUTPUT:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
0	198	Donald	OConnell	DOCONNEL	650.507.9833	21-Jun-07	SH_CLERK	2600.0	-	124.0	50.0
1	199	Douglas	Grant	DGRANT	650.507.9844	13-Jan-08	SH_CLERK	2600.0	-	124.0	50.0
2	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-Sep-03	AD_ASST	4400.0	-	101.0	10.0
3	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-Feb-04	MK_MAN	13000.0	-	100.0	20.0
4	202	Pat	Fay	PFAY	603.123.6666	17-Aug-05	MK_REP	6000.0	-	201.0	20.0
5	203	Susan	Mavris	SMAVRIS	515.123.7777	07-Jun-02	HR_REP	6500.0	-	101.0	40.0
6	204	Hermann	Baer	HBAER	515.123.8888	07-Jun-02	PR_REP	10000.0	-	101.0	70.0
7	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-Jun-02	AC_MGR	12008.0	-	101.0	110.0
8	206	William	Gietz	WGIETZ	515.123.8181	07-Jun-02	AC_ACCOUNT	8300.0	-	205.0	110.0
10	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-Sep-05	AD_VP	17000.0	-	100.0	90.0
11	102	Lex	De Haan	LDEHAAN	515.123.4569	13-Jan-01	AD_VP	17000.0	-	100.0	90.0
12	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-Jan-06	IT_PROG	9000.0	-	102.0	60.0
13	104	Bruce	Ernst	BERNST	590.423.4568	21-May-07	IT_PROG	6000.0	-	103.0	60.0
14	105	David	Austin	DAUSTIN	590.423.4569	25-Jun-05	IT_PROG	4800.0	-	103.0	60.0
17	108	Nancy	Greenberg	NGREENBE	515.124.4569	17-Aug-02	FI_MGR	12008.0	-	101.0	100.0
18	109	Daniel	Faviet	DFAVIET	515.124.4169	16-Aug-02	FI_ACCOUNT	9000.0	-	108.0	100.0
19	110	John	Chen	JCHEN	515.124.4269	28-Sep-05	FI_ACCOUNT	8200.0	-	108.0	100.0
20	111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-Sep-05	FI_ACCOUNT	7700.0	-	108.0	100.0
21	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-Mar-06	FI_ACCOUNT	7800.0	-	108.0	100.0
22	113	Luis	Popp	LPOPP	515.124.4567	07-Dec-07	FI_ACCOUNT	6900.0	-	108.0	100.0
23	114	Den	Raphaely	DRAPHEAL	515.127.4561	07-Dec-02	PU_MAN	11000.0	-	100.0	30.0
24	115	Alexander	Khoo	AKHOO	515.127.4562	18-May-03	PU_CLERK	3100.0	-	114.0	30.0
25	116	Shelli	Baida	SBAIDA	515.127.4563	24-Dec-05	PU_CLERK	2900.0	-	114.0	30.0
40	131	James	Marlow	JAMRLOW	650.124.7234	16-Feb-05	ST_CLERK	2500.0	-	121.0	50.0
41	132	TJ	Olson	TJOLSON	650.124.8234	10-Apr-07	ST_CLERK	2100.0	-	121.0	50.0
42	133	Jason	Mallin	JMALLIN	650.127.1934	14-Jun-04	ST_CLERK	3300.0	-	122.0	50.0
43	134	Michael	Rogers	MROGERS	650.127.1834	26-Aug-06	ST_CLERK	2900.0	-	122.0	50.0
44	135	Ki	Gee	KGEE	650.127.1734	12-Dec-07	ST_CLERK	2400.0	-	122.0	50.0
45	136	Hazel	Philtanker	HPHILTAN	650.127.1634	06-Feb-08	ST_CLERK	2200.0	-	122.0	50.0
46	137	Renske	Ladwig	RLADWIG	650.121.1234	14-Jul-03	ST_CLERK	3600.0	-	123.0	50.0
47	138	Stephen	Stiles	SSTILES	650.121.2034	26-Oct-05	ST_CLERK	3200.0	-	123.0	50.0
48	139	John	Seo	JSEO	650.121.2019	12-Feb-06	ST_CLERK	2700.0	-	123.0	50.0

WEEK 4

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)

DESCRIPTION: Perceptron is an algorithm for Supervised Learning of single layer binary linear classifiers. Optimal weight coefficients are automatically learned. Weights are multiplied with the input features and decision is made if the neuron is fired or not. Perceptron is the most commonly used term for all folks. It is the primary step to learn Machine Learning and Deep Learning technologies, which consists of a set of weights, input values or scores, and a threshold.

Steps to perform a perceptron learning algorithm

1. Feed the features of the model that is required to be trained as input in the first layer.
2. All weights and inputs will be multiplied – the multiplied result of each weight and input will be added up
3. The Bias value will be added to shift the output function
4. This value will be presented to the activation function (the type of activation function will depend on the need)
5. The value received after the last step is the output value.

CODE:

```
print("Enter the initial weights: ")
w = list(map(int, input().split()))
N = [[0, 0, 0]]
P = [[0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]
# 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]
while True:
    temp = w[:]
    for x in N:
        sum = 0
        for y in range(len(x)):
            sum += temp[y]*x[y]
        if sum >= 0:
            for y in range(len(temp)):
                temp[y] -= x[y]
    for x in P:
        sum = 0
        for y in range(len(x)):
            sum += temp[y]*x[y]
        if sum < 0:
            for y in range(len(temp)):
```

```
            temp[y] += x[y]
    if temp == w:
        break
    w = temp[:]
print(w)
```

OUTPUT:

```
Enter the Initial Weights:
1 0 0 1
[-1, 1, 1, 1]
```

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

DESCRIPTION: The perceptron learning algorithm is a supervised learning algorithm used to train artificial neural networks. To implement the AND and EX-OR truth tables using the Perceptron Learning Algorithm, we defne the input and output data, initialize weights and biases, provide input data to the network, adjust weights and biases until output matches expected output, and repeat the process for all input data until the network can correctly predict the output for all inputs.

CODE:

```
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
# NOT Logic Function
# wNOT = -1, bNOT = 0.5
def NOT_logicFunction(x):
    wNOT = -1
    bNOT = 0.5
    return perceptronModel(x, wNOT, bNOT)
# AND Logic Function
# here w1 = wAND1 = 1,
# w2 = wAND2 = 1, bAND = -1.5
def AND_logicFunction(x):
    w = np.array([1, 1])
    bAND = -1.5
    return perceptronModel(x, w, bAND)
# OR Logic Function
# w1 = 1, w2 = 1, bOR = -0.5
def OR_logicFunction(x):
    w = np.array([1, 1])
    bOR = -0.5
```

```
    return perceptronModel(x, w, bOR)
# XOR Logic Function
# with AND, OR and NOT
# function calls in sequence
def XOR_logicFunction(x):
    y1 = AND_logicFunction(x)
    y2 = OR_logicFunction(x)
    y3 = NOT_logicFunction(y1)
    final_x = np.array([y2, y3])
    finalOutput = AND_logicFunction(final_x)
    return finalOutput
# testing the Perceptron Model
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_logicFunction(test1)))
print("XOR({}, {}) = {}".format(0, 1, XOR_logicFunction(test2)))
print("XOR({}, {}) = {}".format(1, 0, XOR_logicFunction(test3)))
print("XOR({}, {}) = {}".format(1, 1, XOR_logicFunction(test4)))
print("AND({}, {}) = {}".format(0, 0, AND_logicFunction(test1)))
print("AND({}, {}) = {}".format(0, 1, AND_logicFunction(test2)))
print("AND({}, {}) = {}".format(1, 0, AND_logicFunction(test3)))
print("AND({}, {}) = {}".format(1, 1, AND_logicFunction(test4)))
```

OUTPUT:

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

WEEK 5

1.AIM: As sometimes missing values are present in the data set. It can be handled in three ways.

i. Removing the whole line
ii. Creating a sub model to predict those features
iii. Using a automatic strategy to input them according to the other know values.

Now apply option (iii) to a data set, which contains numeric field, fill the data using Imputer class and replace it with mean, median and mode strategy.

DESCRIPTION: The imputer is an estimator used to fill the missing values in datasets. For numerical values, it uses mean, median, and constant. For categorical values, it uses the most frequently used and constant value. You can also train your model to predict the missing labels. SimpleImputer is a class in the sklearn. impute module that can be used to replace missing values in a dataset, using a variety of input strategies. SimpleImputer is designed to work with numerical data, but can also handle categorical data represented as strings.

CODE:

a. Using Mean

```
from sklearn.impute import SimpleImputer
import numpy as np
imputer = SimpleImputer(strategy= ‘mean’, missing_values=np.nan)
imputer = imputer.fit(data[['SALARY']])
data['SALARY']=imputer.transform(data[['SALARY']])
print(data)
```

OUTPUT:

```
    EMPLOYEE_ID  FIRST_NAME   LAST_NAME     EMAIL  ...        SALARY COMMISSION_PCT MANAGER_ID  DEPARTMENT_ID
0           198      Donald    OConnell  DOCONNEL  ...   2600.000000              -      124.0           50.0
1           199     Douglas       Grant    DGRANT  ...   2600.000000              -      124.0           50.0
2           200    Jennifer      Whalen   JWHALEN  ...   4400.000000              -      101.0           10.0
3           201     Michael   Hartstein  MHARTSTE  ...  13000.000000              -      100.0           20.0
4           202         Pat         Fay      PFAY  ...   6000.000000              -      201.0           20.0
5           203       Susan      Mavris   SMAVRIS  ...   6500.000000              -      101.0           40.0
6           204     Hermann        Baer     HBAER  ...  10000.000000              -      101.0           70.0
7           205     Shelley     Higgins  SHIGGINS  ...  12008.000000              -      101.0          110.0
8           206     William       Gietz    WGIETZ  ...   8300.000000              -      205.0          110.0
9           100      Steven        King     SKING  ...  24000.000000              -        NaN           90.0
10          101       Neena     Kochhar  NKOCHHAR  ...  17000.000000              -      100.0           90.0
11          102         Lex     De Haan   LDEHAAN  ...  17000.000000              -      100.0           90.0
12          103   Alexander      Hunold   AHUNOLD  ...   9000.000000              -      102.0           60.0
13          104       Bruce       Ernst    BERNST  ...   6000.000000              -      103.0           60.0
14          105       David      Austin   DAUSTIN  ...   4800.000000              -      103.0           60.0
15          106       Valli   Pataballa  VPATABAL  ...   4800.000000              -      103.0            NaN
16          107       Diana     Lorentz  DLORENTZ  ...   4200.000000              -      103.0            NaN
17          108       Nancy   Greenberg  NGREENBE  ...  12008.000000              -      101.0          100.0
18          109      Daniel      Faviet   DFAVIET  ...   9000.000000              -      108.0          100.0
19          110        John        Chen     JCHEN  ...   8200.000000              -      108.0          100.0
20          111      Ismael     Sciarra  ISCIARRA  ...   7700.000000              -      108.0          100.0
21          112 Jose Manuel       Urman   JMURMAN  ...   7800.000000              -      108.0          100.0
22          113        Luis        Popp     LPOPP  ...   6900.000000              -      108.0          100.0
23          114         Den    Raphaely  DRAPHEAL  ...  11000.000000              -      100.0           30.0
24          115   Alexander        Khoo     AKHOO  ...   3100.000000              -      114.0           30.0
25          116      Shelli       Baida    SBAIDA  ...   2900.000000              -      114.0           30.0
26          117       Sigal      Tobias   STOBIAS  ...   2800.000000              -      114.0           30.0
47          138     Stephen      Stiles   SSTILES  ...   3200.000000              -      123.0           50.0
48          139        John         Seo      JSEO  ...   2700.000000              -      123.0           50.0
49          140      Joshua       Patel    JPATEL  ...   6257.469388              -      123.0           50.0

[50 rows x 11 columns]
```

b. Using Median

```
from sklearn.impute import SimpleImputer
import numpy as np
imputer = SimpleImputer(strategy='median', missing_values=np.nan)
imputer = imputer.fit(data[['SALARY']])
data['SALARY']=imputer.transform(data[['SALARY']])
```

print(data)

OUTPUT:

```
    EMPLOYEE_ID  FIRST_NAME  LAST_NAME     EMAIL  ...   SALARY COMMISSION_PCT MANAGER_ID  DEPARTMENT_ID
0           198      Donald   OConnell  DOCONNEL  ...   2600.0              -      124.0           50.0
1           199     Douglas      Grant    DGRANT  ...   2600.0              -      124.0           50.0
2           200    Jennifer     Whalen   JWHALEN  ...   4400.0              -      101.0           10.0
3           201     Michael  Hartstein  MHARTSTE  ...  13000.0              -      100.0           20.0
4           202         Pat        Fay      PFAY  ...   6000.0              -      201.0           20.0
5           203       Susan     Mavris   SMAVRIS  ...   6500.0              -      101.0           40.0
6           204     Hermann       Baer     HBAER  ...  10000.0              -      101.0           70.0
7           205     Shelley    Higgins  SHIGGINS  ...  12008.0              -      101.0          110.0
8           206     William      Gietz    WGIETZ  ...   8300.0              -      205.0          110.0
9           100      Steven       King     SKING  ...  24000.0              -        NaN           90.0
10          101       Neena    Kochhar  NKOCHHAR  ...  17000.0              -      100.0           90.0
11          102         Lex    De Haan   LDEHAAN  ...  17000.0              -      100.0           90.0
12          103   Alexander     Hunold   AHUNOLD  ...   9000.0              -      102.0           60.0
13          104       Bruce      Ernst    BERNST  ...   6000.0              -      103.0           60.0
14          105       David     Austin   DAUSTIN  ...   4800.0              -      103.0           60.0
15          106       Valli  Pataballa  VPATABAL  ...   4800.0              -      103.0            NaN
16          107       Diana    Lorentz  DLORENTZ  ...   4200.0              -      103.0            NaN
17          108       Nancy  Greenberg  NGREENBE  ...  12008.0              -      101.0          100.0
18          109      Daniel     Faviet   DFAVIET  ...   9000.0              -      108.0          100.0
19          110        John       Chen     JCHEN  ...   8200.0              -      108.0          100.0
20          111      Ismael    Sciarra  ISCIARRA  ...   7700.0              -      108.0          100.0
21          112 Jose Manuel      Urman   JMURMAN  ...   7800.0              -      108.0          100.0
22          113        Luis       Popp     LPOPP  ...   6900.0              -      108.0          100.0
23          114         Den   Raphaely  DRAPHEAL  ...  11000.0              -      100.0           30.0
47          138     Stephen     Stiles   SSTILES  ...   3200.0              -      123.0           50.0
48          139        John        Seo      JSEO  ...   2700.0              -      123.0           50.0
49          140      Joshua      Patel    JPATEL  ...   4800.0              -      123.0           50.0

[50 rows x 11 columns]
```

c. Using Mode

```
from sklearn.impute import SimpleImputer
import numpy as np
imputer = SimpleImputer(strategy='most_frequent', missing_values=np.nan)
imputer = imputer.fit(data[['SALARY']])
data['SALARY']=imputer.transform(data[['SALARY']])
print(data)
```

OUTPUT:

```
    EMPLOYEE_ID  FIRST_NAME  LAST_NAME     EMAIL  ...   SALARY COMMISSION_PCT MANAGER_ID  DEPARTMENT_ID
0           198      Donald   OConnell  DOCONNEL  ...   2600.0              -      124.0           50.0
1           199     Douglas      Grant    DGRANT  ...   2600.0              -      124.0           50.0
2           200    Jennifer     Whalen   JWHALEN  ...   4400.0              -      101.0           10.0
3           201     Michael  Hartstein  MHARTSTE  ...  13000.0              -      100.0           20.0
4           202         Pat        Fay      PFAY  ...   6000.0              -      201.0           20.0
5           203       Susan     Mavris   SMAVRIS  ...   6500.0              -      101.0           40.0
6           204     Hermann       Baer     HBAER  ...  10000.0              -      101.0           70.0
7           205     Shelley    Higgins  SHIGGINS  ...  12008.0              -      101.0          110.0
8           206     William      Gietz    WGIETZ  ...   8300.0              -      205.0          110.0
9           100      Steven       King     SKING  ...  24000.0              -        NaN           90.0
10          101       Neena    Kochhar  NKOCHHAR  ...  17000.0              -      100.0           90.0
11          102         Lex    De Haan   LDEHAAN  ...  17000.0              -      100.0           90.0
12          103   Alexander     Hunold   AHUNOLD  ...   9000.0              -      102.0           60.0
13          104       Bruce      Ernst    BERNST  ...   6000.0              -      103.0           60.0
14          105       David     Austin   DAUSTIN  ...   4800.0              -      103.0           60.0
15          106       Valli  Pataballa  VPATABAL  ...   4800.0              -      103.0            NaN
16          107       Diana    Lorentz  DLORENTZ  ...   4200.0              -      103.0            NaN
17          108       Nancy  Greenberg  NGREENBE  ...  12008.0              -      101.0          100.0
18          109      Daniel     Faviet   DFAVIET  ...   9000.0              -      108.0          100.0
19          110        John       Chen     JCHEN  ...   8200.0              -      108.0          100.0
20          111      Ismael    Sciarra  ISCIARRA  ...   7700.0              -      108.0          100.0
21          112 Jose Manuel      Urman   JMURMAN  ...   7800.0              -      108.0          100.0
22          113        Luis       Popp     LPOPP  ...   6900.0              -      108.0          100.0
23          114         Den   Raphaely  DRAPHEAL  ...  11000.0              -      100.0           30.0
24          115   Alexander       Khoo     AKHOO  ...   3100.0              -      114.0           30.0
25          116      Shelli      Baida    SBAIDA  ...   2900.0              -      114.0           30.0
46          137      Renske     Ladwig   RLADWIG  ...   3600.0              -      123.0           50.0
47          138     Stephen     Stiles   SSTILES  ...   3200.0              -      123.0           50.0
48          139        John        Seo      JSEO  ...   2700.0              -      123.0           50.0
49          140      Joshua      Patel    JPATEL  ...   2600.0              -      123.0           50.0

[50 rows x 11 columns]
```

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

DESCRIPTION: One hot encoding is a technique used to represent categorical variables as numerical values in a machine learning model. The advantages of using one hot encoding include:

- It allows the use of categorical variables in models that require numerical input.
- It can improve model performance by providing more information to the model about the categorical variable.
- It can help to avoid the problem of ordinality, which can occur when a categorical variable has a natural ordering (e.g. "small", "medium", "large").

CODE:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
df = pd.read_csv('employees.csv')
hot_encoded_data = pd.get_dummies(df,columns=['JOB_ID'])
print(hot_encoded_data)
```

OUTPUT:

```
/OneDrive/Desktop/SEM 6 CBIT Workspace/ML Lab/Codes/5_2.py"
    EMPLOYEE_ID   FIRST_NAME    LAST_NAME     EMAIL  ... JOB_ID_PU_MAN JOB_ID_SH_CLERK  JOB_ID_ST_CLERK JOB_ID_ST_MAN
0           198       Donald     OConnell  DOCONNEL  ...             0               1                0             0
1           199      Douglas        Grant    DGRANT  ...             0               1                0             0
2           200     Jennifer       Whalen   JWHALEN  ...             0               0                0             0
3           201      Michael    Hartstein  MHARTSTE  ...             0               0                0             0
4           202          Pat          Fay      PFAY  ...             0               0                0             0
5           203        Susan       Mavris   SMAVRIS  ...             0               0                0             0
6           204      Hermann         Baer     HBAER  ...             0               0                0             0
7           205      Shelley      Higgins  SHIGGINS  ...             0               0                0             0
8           206      William        Gietz    WGIETZ  ...             0               0                0             0
9           100       Steven         King     SKING  ...             0               0                0             0
10          101        Neena      Kochhar  NKOCHHAR  ...             0               0                0             0
11          102          Lex      De Haan   LDEHAAN  ...             0               0                0             0
12          103    Alexander       Hunold   AHUNOLD  ...             0               0                0             0
13          104        Bruce        Ernst    BERNST  ...             0               0                0             0
14          105        David       Austin   DAUSTIN  ...             0               0                0             0
15          106        Valli    Pataballa  VPATABAL  ...             0               0                0             0
16          107        Diana      Lorentz  DLORENTZ  ...             0               0                0             0
17          108        Nancy    Greenberg  NGREENBE  ...             0               0                0             0
18          109       Daniel       Faviet   DFAVIET  ...             0               0                0             0
19          110         John         Chen     JCHEN  ...             0               0                0             0
20          111       Ismael      Sciarra  ISCIARRA  ...             0               0                0             0
21          112  Jose Manuel        Urman   JMURMAN  ...             0               0                0             0
22          113         Luis         Popp     LPOPP  ...             0               0                0             0
44          135           Ki          Gee      KGEE  ...             0               0                1             0
45          136        Hazel   Philtanker  HPHILTAN  ...             0               0                1             0
46          137       Renske       Ladwig   RLADWIG  ...             0               0                1             0
47          138      Stephen       Stiles   SSTILES  ...             0               0                1             0
48          139         John          Seo      JSEO  ...             0               0                1             0
49          140       Joshua        Patel    JPATEL  ...             0               0                1             0

[50 rows x 27 columns]
```

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. Refer the formulas

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

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. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges.

CODE:

```
from math import sqrt
x=list(data['bwa'])
maxval=max(x)
maxnorm=[]
l1norm=[]
l2norm=[]
s=sum(x)
sq=0
for i in x:
  maxnorm.append(i/maxval)
  l1norm.append(i/s)
  sq+=i*i
for i in x:
  l2norm.append(i/sqrt(sq))
print(maxnorm)
data['maxNorm_bwa']=maxnorm
data['l1Norm_bwa']=l1norm
data['l2Norm_bwa']=l2norm
print(data)
```

OUTPUT:

```
       wk         sp  maxNorm_bwa  l1Norm_bwa  l2Norm_bwa
0     3.0  2020-2022     0.112178    0.002353    0.031731
1     8.0  2021-2022     0.158416    0.003323    0.044810
2     0.0  2012-2022     0.000000    0.000000    0.000000
3    11.0  2022-2022     0.094505    0.001982    0.026732
4     0.0  2015-2022     0.000000    0.000000    0.000000
..    ...        ...          ...         ...         ...
342   1.0  2019-2023     0.014851    0.000311    0.004201
343   0.0  2020-2022     0.000000    0.000000    0.000000
344   1.0  2021-2021     0.306931    0.006437    0.086818
345   0.0  2018-2021     0.000000    0.000000    0.000000
346   0.0  2021-2021     0.000000    0.000000    0.000000
```

WEEK 6

1.AIM: Implement ID3 algorithm on 'weather.csv' dataset and 'buys_computer' dataset.

DESCRIPTION: ID3 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).

Entropy is a measure of the amount of uncertainty in the dataset S. Mathematical Representation of Entropy is shown here –

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

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.

Information Gain IG(A) tells us how much uncertainty in S was reduced after splitting set S on attribute A. Mathematical representation of Information gain is shown here –

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

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:

Calculate entropy for dataset.

For each attribute/feature.
2.1. Calculate entropy for all its categorical values.
2.2. Calculate information gain for the feature.

Find the feature with maximum information gain.

Repeat it until we get the desired tree.

CODE:

```
import pandas as pd
import numpy as np
train_data_m = pd.read_csv("second.csv")
test_data_m = pd.read_csv("second.csv")
train_data_m.head()
```

```
def calc_total_entropy(train_data, label, class_list):
    total_row = train_data.shape[0]
    total_entr = 0
    for c in class_list:
        total_class_count = train_data[train_data[label] == c].shape[0]
        total_class_entr = - (total_class_count/total_row)*np.log2(total_class_count/total_row)
        total_entr += total_class_entr
    return total_entr

def calc_entropy(feature_value_data, label, class_list):
    class_count = feature_value_data.shape[0]
    entropy = 0
    for c in class_list:
        label_class_count = feature_value_data[feature_value_data[label] == c].shape[0]
        entropy_class = 0
        if label_class_count != 0:
            probability_class = label_class_count/class_count
            entropy_class = - probability_class * np.log2(probability_class)

        entropy += entropy_class

    return entropy

def calc_info_gain(feature_name, train_data, label, class_list):
    feature_value_list = train_data[feature_name].unique()
    total_row = train_data.shape[0]
    feature_info = 0.0

    for feature_value in feature_value_list:
        feature_value_data = train_data[train_data[feature_name] == feature_value]
        feature_value_count = feature_value_data.shape[0]
        feature_value_entropy = calc_entropy(feature_value_data, label, class_list)
        feature_value_probability = feature_value_count/total_row
        feature_info += feature_value_probability * feature_value_entropy

    return calc_total_entropy(train_data, label, class_list) - feature_info

def find_most_informative_feature(train_data, label, class_list):
    feature_list = train_data.columns.drop(label)
    max_info_gain = -1
    max_info_feature = None

    for feature in feature_list:
        feature_info_gain = calc_info_gain(feature, train_data, label, class_list)
        if max_info_gain < feature_info_gain:
            max_info_gain = feature_info_gain
            max_info_feature = feature
```

```
    return max_info_feature

def generate_sub_tree(feature_name, train_data, label, class_list):
    feature_value_count_dict = train_data[feature_name].value_counts(sort=False)
    tree = {}

    for feature_value, count in feature_value_count_dict.items():
        feature_value_data = train_data[train_data[feature_name] == feature_value]

        assigned_to_node = False
        for c in class_list:
            class_count = feature_value_data[feature_value_data[label] == c].shape[0]

            if class_count == count:
                tree[feature_value] = c
                train_data = train_data[train_data[feature_name] != feature_value]
                assigned_to_node = True
        if not assigned_to_node:
            tree[feature_value] = "?"

    return tree, train_data

def make_tree(root, prev_feature_value, train_data, label, class_list):
    if train_data.shape[0] != 0:
        max_info_feature = find_most_informative_feature(train_data, label, class_list)
        tree, train_data = generate_sub_tree(max_info_feature, train_data, label, class_list)
        next_root = None

        if prev_feature_value != None:
            root[prev_feature_value] = dict()
            root[prev_feature_value][max_info_feature] = tree
            next_root = root[prev_feature_value][max_info_feature]
        else:
            root[max_info_feature] = tree
            next_root = root[max_info_feature]

        for node, branch in list(next_root.items()):
            if branch == "?":
                feature_value_data = train_data[train_data[max_info_feature] == node]
                make_tree(next_root, node, feature_value_data, label, class_list)

def id3(train_data_m, label):
    train_data = train_data_m.copy()
    tree = {}
    class_list = train_data[label].unique()
    make_tree(tree, None, train_data, label, class_list)

    return tree
```

```
def predict(tree, instance):
    if not isinstance(tree, dict):
        return tree
    else:
        root_node = next(iter(tree))
        feature_value = instance[root_node]
        if feature_value in tree[root_node]:
            return predict(tree[root_node][feature_value], instance)
        else:
            return None

def evaluate(tree, test_data_m, label):
    correct_predict = 0
    wrong_predict = 0
    for index, row in test_data_m.iterrows():
        result = predict(tree, test_data_m.iloc[index])
        if result == test_data_m[label].iloc[index]:
            correct_predict += 1
        else:
            wrong_predict += 1

tree = id3(train_data_m,'Buys')
print(tree)
```

OUTPUT:

a.For ‘weather.csv’ data set

```
{'outlook': {'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}, 'overcast': 'yes', 'rainy': {'windy': {Fal
se: 'yes', True: 'no'}}}}
```

b.For ‘buys_computer.csv’ data set

```
{'Age': {'Youth': {'Student': {'No': 'No', 'Yes': 'Yes'}}, 'Middle aged': 'Yes', 'Senior': {'Credit_rating': {'Fair': 'Yes', 'Excellent': 'No'}}}}
```

WEEK 7

AIM: Implement naive bayes algorithm on 'weather.csv' dataset. You can use the python notebook uploaded here. Answer the following on weather data set.

	Outlook	Temperature	Humidity	Windy	Play Golf
1	Rainy	Hot	High	False	No
2	Rainy	Hot	High	True	No
3	Overcast	Hot	High	False	Yes
4	Sunny	Mild	High	False	Yes
5	Sunny	Cool	Normal	False	Yes
6	Sunny	Cool	Normal	True	No
7	Overcast	Cool	Normal	True	Yes
8	Rainy	Mild	High	False	No
9	Rainy	Cool	Normal	False	Yes
10	Sunny	Mild	Normal	False	Yes
11	Rainy	Mild	Normal	True	Yes
12	Overcast	Mild	High	True	Yes
13	Overcast	Hot	Normal	False	Yes
14	Sunny	Mild	High	True	No

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

DESCRIPTION: Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. Bayes' Theorem is distinguished by its use of sequential events, where additional information later acquired impacts the initial probability. These probabilities are denoted as the prior probability and the posterior probability. The prior probability is the initial probability of an event before it is contextualized under a certain condition, or the marginal probability. The posterior probability is the probability of an event after observing a piece of data.

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

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**. A Gaussian distribution is also called Normal Distr.

CODE:

```
from sklearn.naive_bayes import GaussianNB
import pandas as pd
from sklearn.preprocessing import LabelEncoder
df=pd.read_csv('weather.nominal.csv')
Numerics=LabelEncoder()
inputs=df.drop('play',axis='columns')
target=df['play']
print(target)
print(inputs)
```

```
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)
print(inputs_n)
classifier = GaussianNB()
classifier.fit(inputs_n,target)
classifier.predict([ [2,2,0,1] ])
classifier.predict([ [0,0,0,0] ])
```

OUTPUT:

```
Classifier.score(inputs_n.values,target)

0.9285714285714286

Classifier.predict([[2,0,0,1]])

array(['no'], dtype='<U3')

Classifier.predict([[0,2,0,0]])

array(['yes'], dtype='<U3')
```

WEEK 8

1.AIM: Apply KNN classifier on the following data set for the input

Sepal Length	Sepal width	Species
5.3	3.7	setosa
5.1	3.8	Setosa
7.2	3.0	Virginica
5.4	3.4	Setosa
5.1	3.3	Setosa
5.4	3.9	Setosa
7.4	2.8	Virginica
6.1	2.8	Verscicolor
7.3	2.9	Virginica
6.0	2.7	Verscicolor
5.8	2.8	Virginica
6.3	2.3	Verscicolor
5.1	2.5	Verscicolor
6.3	2.5	Verscicolor
5.5	2.4	Verscicolor

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
data = pd.read_csv('knndata.csv')
X = data.iloc[:, :-1].values
```

```
y = data.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_data = [[6.0, 2.7], [5.2, 2.8], [5.6, 2.7],[4.9, 2.5]]
predictions = knn.predict(new_data)
print(predictions)
```

OUTPUT:

```
Accuracy: 1.0
['verscicolor' 'setosa' 'verscicolor' 'setosa']
```

2.AIM: Apply clustering algorithms – k – means, agglomerative and DBSCAN to classify for any standard datasets.

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.

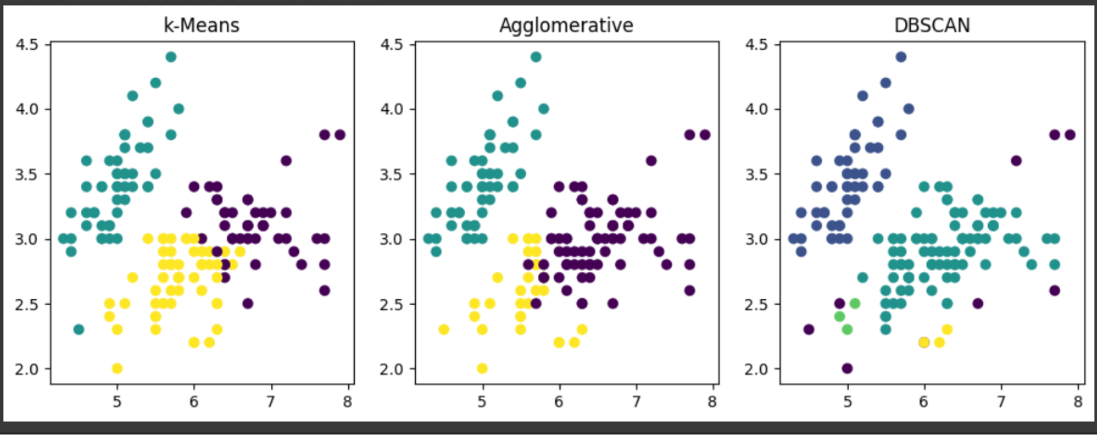
CODE:

```
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

AIM: Demonstration of Naïve Bayesian classifier for a sample training data set stored as a .CSV file. You can down load any data set of your choice. Calculate the accuracy, precision, and recall for your 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.

CODE:

```
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_state=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:

```
(150, 4)
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 2 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0 0 0 2 1 1 0 0]
Accuracy: 0.9777777777777777
Precision: 0.9793650793650793
Recall: 0.9777777777777777
F1 score: 0.9777448559670783
New predictions: [0 1 2]
```

WEEK 10

AIM: Case study on supervised learning algorithms. Down any one data set from Kaggle or any other repository. Apply the following supervised learning algorithms:

1. Linear Regression
2. Logistic regression
3. ID3
4. Random forest
5. XG Boost
6. Naïve Bayes Algorithm

Compare the accuracy and plot 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.

CODE:

```
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
```