**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.

**Program:**

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))

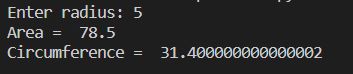
n = int(input("Enter radius: "))

c = Circle(n)

c.Area()

c.Circumference()

**OUTPUT:**



**2AIM:** 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.

**Program**:

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()

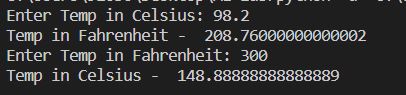
n1 = float(input("Enter Temp in Celsius: "))

T.ConverttoF(n1)

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

T.ConverttoC(n2)

**OUTPUT:**

****

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.

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)

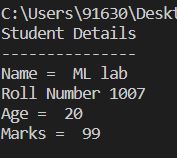
s = Student("ML Lab", 1007)

s.setAge(20)

s.setmarks(99)

s.display()

**OUTPUT:**

****

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.

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)

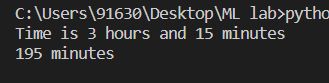
b = Time(1, 52)

c = Time.addTime(a,b)

c.displayTime()

c.displayMin()

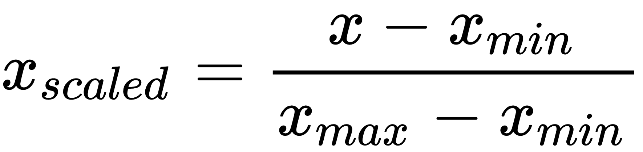
**OUTPUT:**

****

**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.(Note: refer .xls file)

**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:



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: 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 – μ) / σ

Here:

x: Original value

 μ: Mean of data

 σ: Standard deviation of data

Decimal scaling is a data normalization technique like [Z score](https://t4tutorials.com/z-score-normalization-data-mining/), [Min-Max](https://t4tutorials.com/min-max-normalization-of-data-in-data-mining/), and normalization with [standard deviation](https://t4tutorials.com/standard-deviation-normalization-of-data-in-data-mining/). 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  = **( vi / 10j )**

**Program:**

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

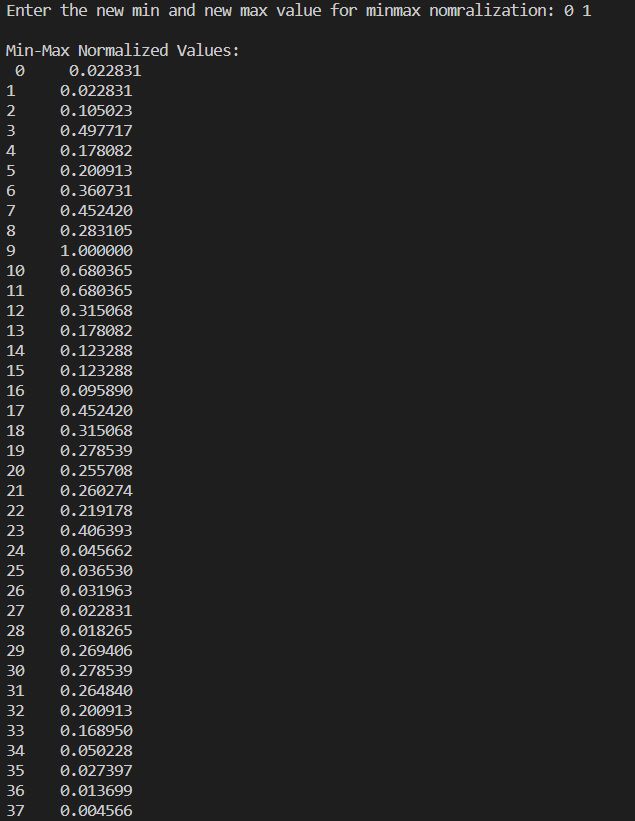
print("\nMin-Max Normalized Values:\n", min\_max\_normalized\_values)

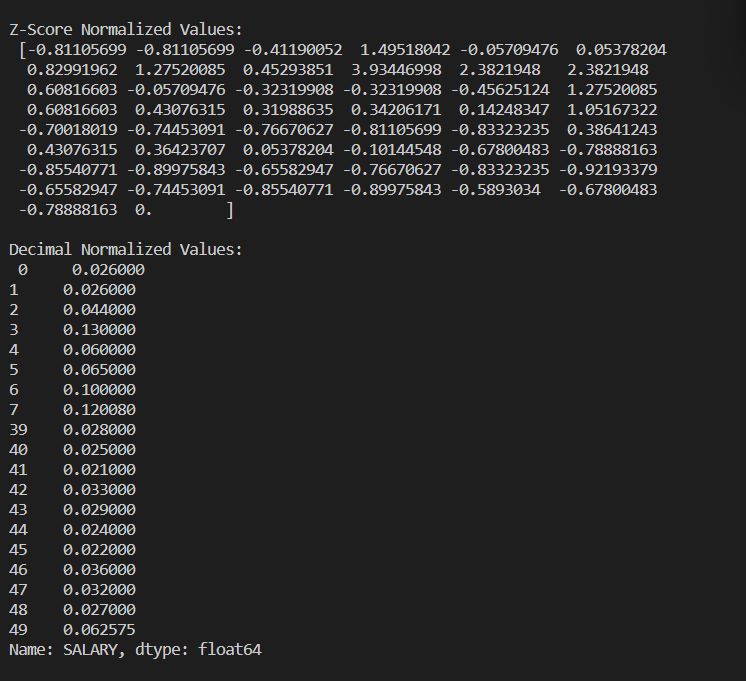
print("\nZ-Score Normalized Values:\n", z\_score\_normalized\_values)

print("\nDecimal Normalized Values:\n", decimal\_normalized\_values)

# print(column.min(), column.max())

**OUTPUT:**

****

****

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

**2.a** 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 dataframe, you can use the info() and describe() methods of the dataframe. The info() method provides a concise summary of the dataframe, 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 dataframe, including count, mean, standard deviation, minimum, maximum, and quartile values. Note that the describe() method only includes numerical columns by default.

**Program:**

# 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:**

****

**2.b** Count the number of missing values in each column?

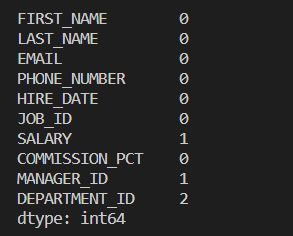
**DESCRIPTION:** To count the number of missing values in each column of a dataframe, you can use the isnull() method to create a boolean mask indicating which values are missing, and then use the sum() method to count the number of True values in each column.

# 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:**



**2.c** 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.

# Rename a specific column

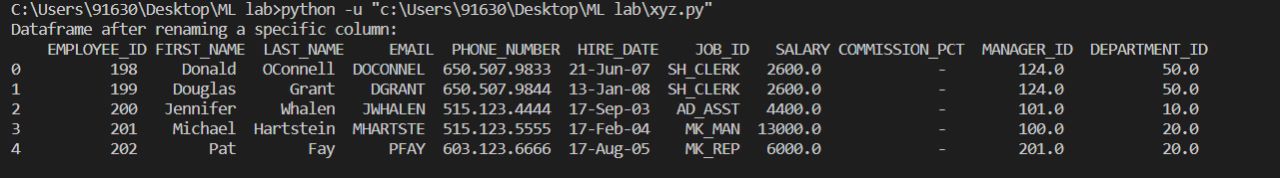
df = df.rename(columns={'number\_courses' : 'Number\_of\_Courses', 'time\_study' : 'Ti

me\_Allocated'})

# Verify that the column has been renamed

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

**OUTPUT:**



**2.d** 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.

import numpy as np

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

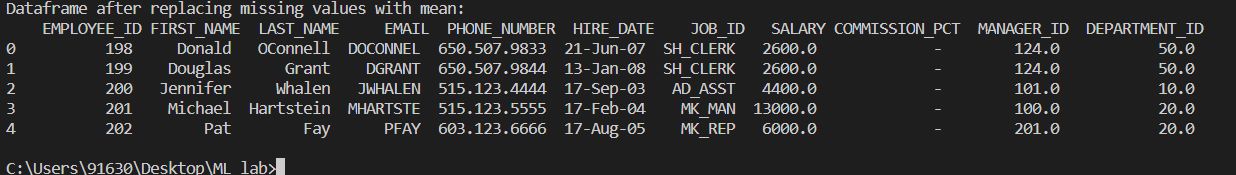
numeric\_columns = df.select\_dtypes(include=[np.number]).columns

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:**



**2.e** 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.

# Get the current column order

current\_column\_order = df.columns

# Define the desired column order

new\_column\_order = ['Marks', 'Number\_of\_Courses', 'Time\_Allocated']

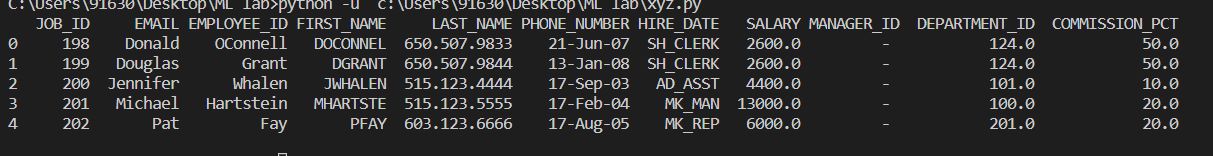
# Change the order of columns

df = df[new\_column\_order]

# Verify that the order of columns has been changed

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

**OUTPUT:**

****

**WEEK- 3**

**1)AIM:** Extract rows with missing values for a speci c 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.

**Program:**

import pandas as pd

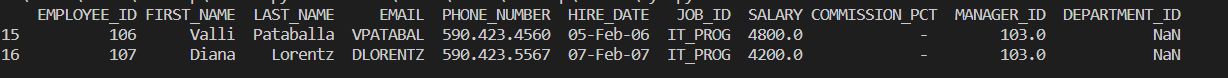
df=pd.read\_csv('employees.csv')

#print(df)

#1

#print(df[df['DEPARTMENT\_ID'].isnull()])

**OUTPUT:**



**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.

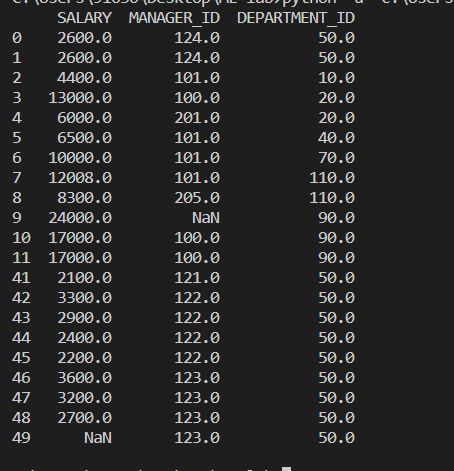
**Program:**

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

#print(df2)

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

**OUTPUT:**



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

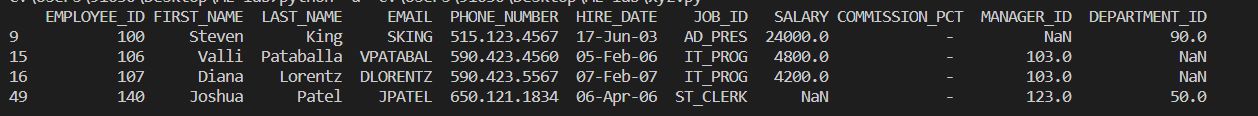
**DESCRIPTION**: The first step is to import the pandas library and read the dataset as a pandas DataFrame. The second step involves using the any() method to create a boolean mask that identifies the rows with at least one missing value. The any() method returns a boolean value indicating whether any 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.

**Program:**

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

**OUTPUT:**



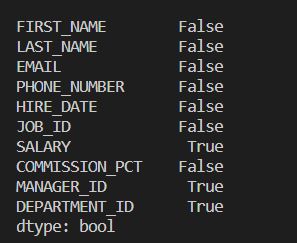
**4)AIM:** Find a list of columns with missing data

**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.

**Program:**

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

**OUTPUT:**



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

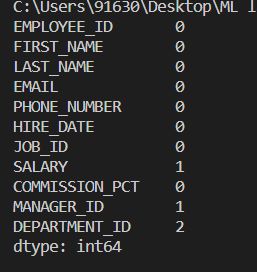
**DESCRIPTION:**

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.

**Program:**

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

**OUTPUT:**



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

**DESCRIPTION:**

1. 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.

**Program:**

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

**OUTPUT:**



**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.

**Program:**

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

**OUTPUT:**



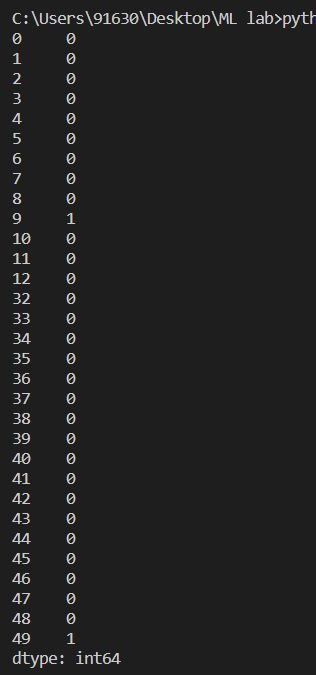
**8)AIM:**Find rows with missing data

**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.

**Program:**

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

**OUTPUT:**



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.

**Program:**

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

or

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

**OUTPUT:**



**10)AIM:** Print the number of missing data per row

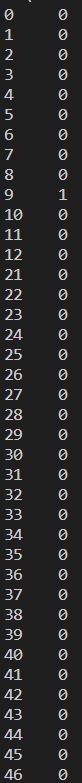
**DESCRIPTION:**

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.

**Program:**

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

**OUTPUT:**



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

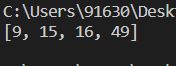
**DESCRIPTION:** To find the row with the largest number of missing data using pandas, you can first import the pandas library and load your data into a DataFrame. Then you can count the number of missing values in each row using the isna() method and sum them using the sum() method with axis=1. You 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.

**Program:**

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

t.loc[lambda x:x==t.max()].index.tolist()

**OUTPUT:**



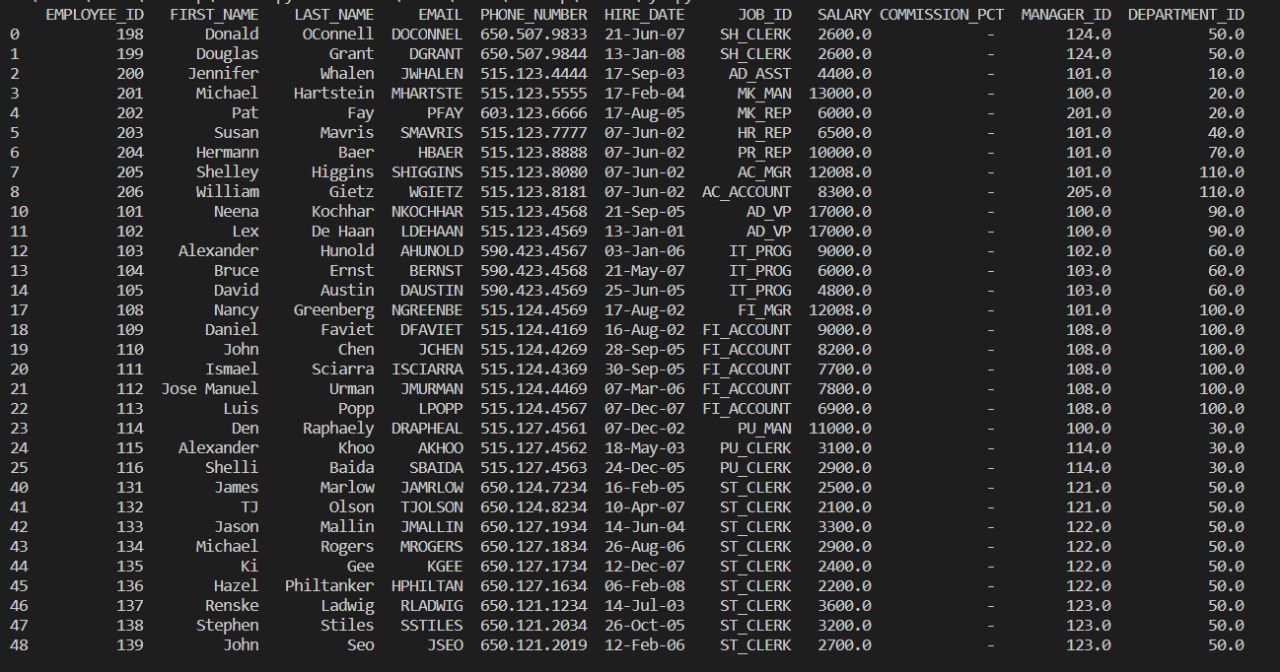
**12)AIM:** Remove rows with missing data

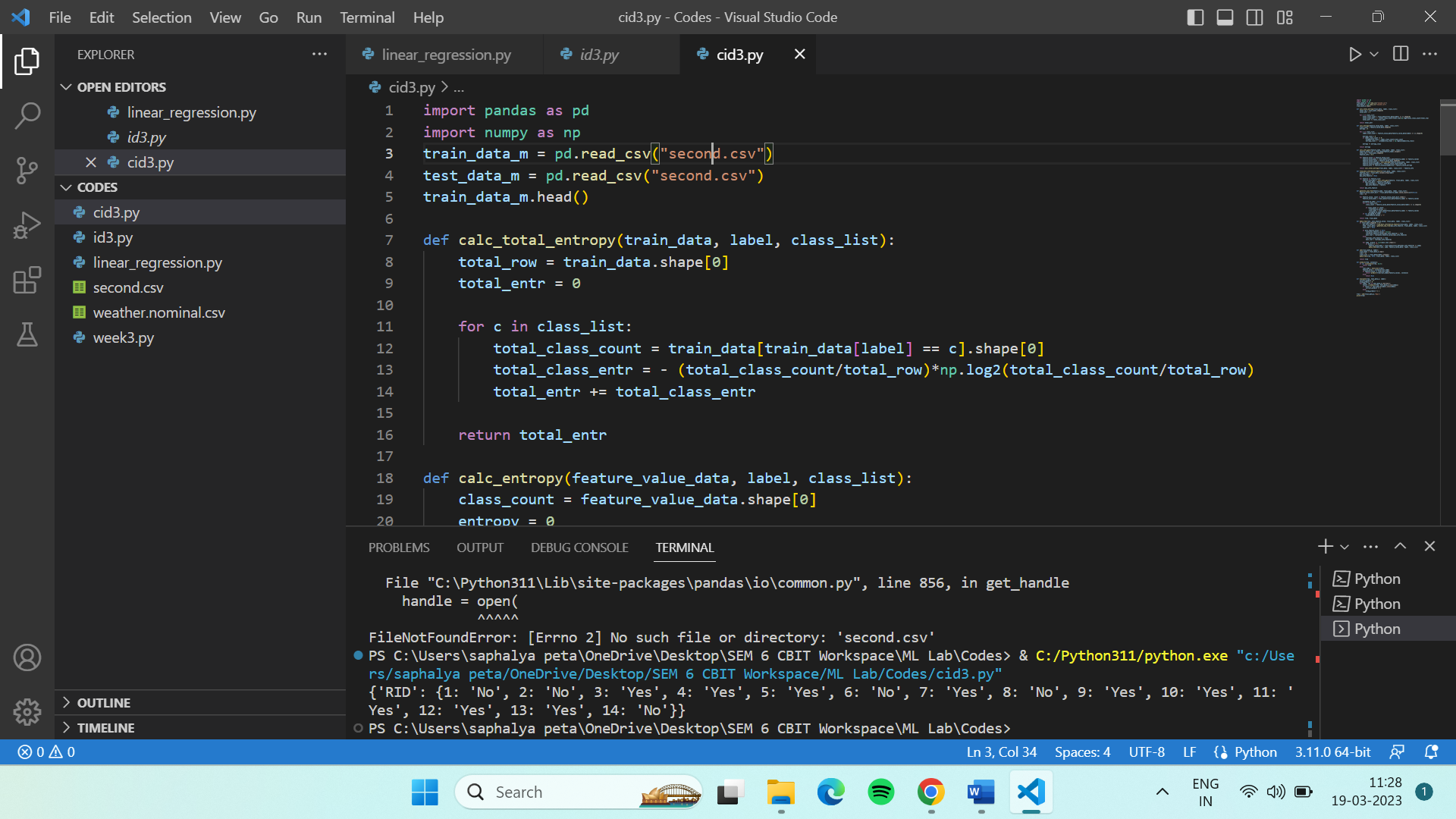
**DESCRIPTION:** To remove rows with missing data from a dataframe, you can use the dropna() method of the dataframe.

**Program:**

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

**OUTPUT:**

****



**WEEK – 4**

1. **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)**

**AIM:** To implement perceptron learning algorithm and find the final weight vector.

**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.

**PROGRAM:**

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]

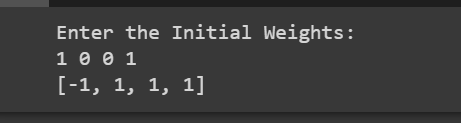
    if temp == w:

        break

    w = temp[:]

print(w)

**OUTPUT:**

****

1. **Implement AND, EX-OR truth table using perceptron learning algorithm**

**AIM:** To 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.

**PROGRAM:**

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)))

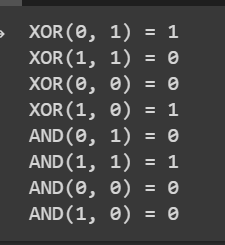
print("AND({}, {}) = {}".format(0, 0, AND\_logicFunction(test1)))

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:**



**WEEK – 5**

1. **As sometimes missing values are present in the data set. It can be handled in three ways.**
2. **Removing the whole line**
3. **Creating a sub model to predict those features**
4. **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.**

**AIM:** To input the missing values using Imputer class with mean, median and mode strategy for the given data set.

**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.

**PROGRAM:**

1. 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)

1. 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)

1. Using Mode

from sklearn.impute import SimpleImputer

import numpy as np

imputer = SimpleImputer(strategy='most\_frequent', missing\_values=np.nan)

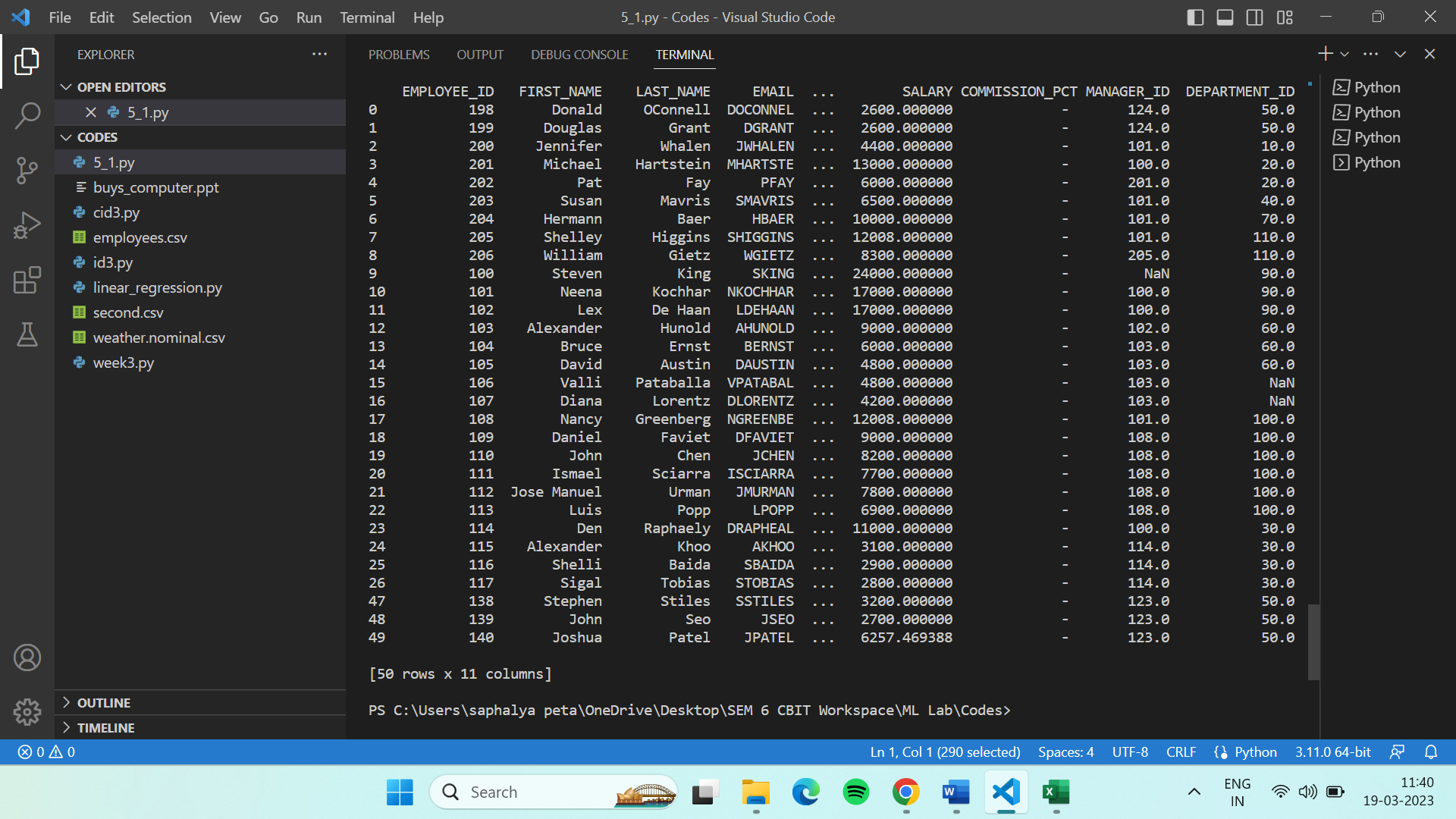
imputer = imputer.fit(data[['SALARY']])

data['SALARY']=imputer.transform(data[['SALARY']])

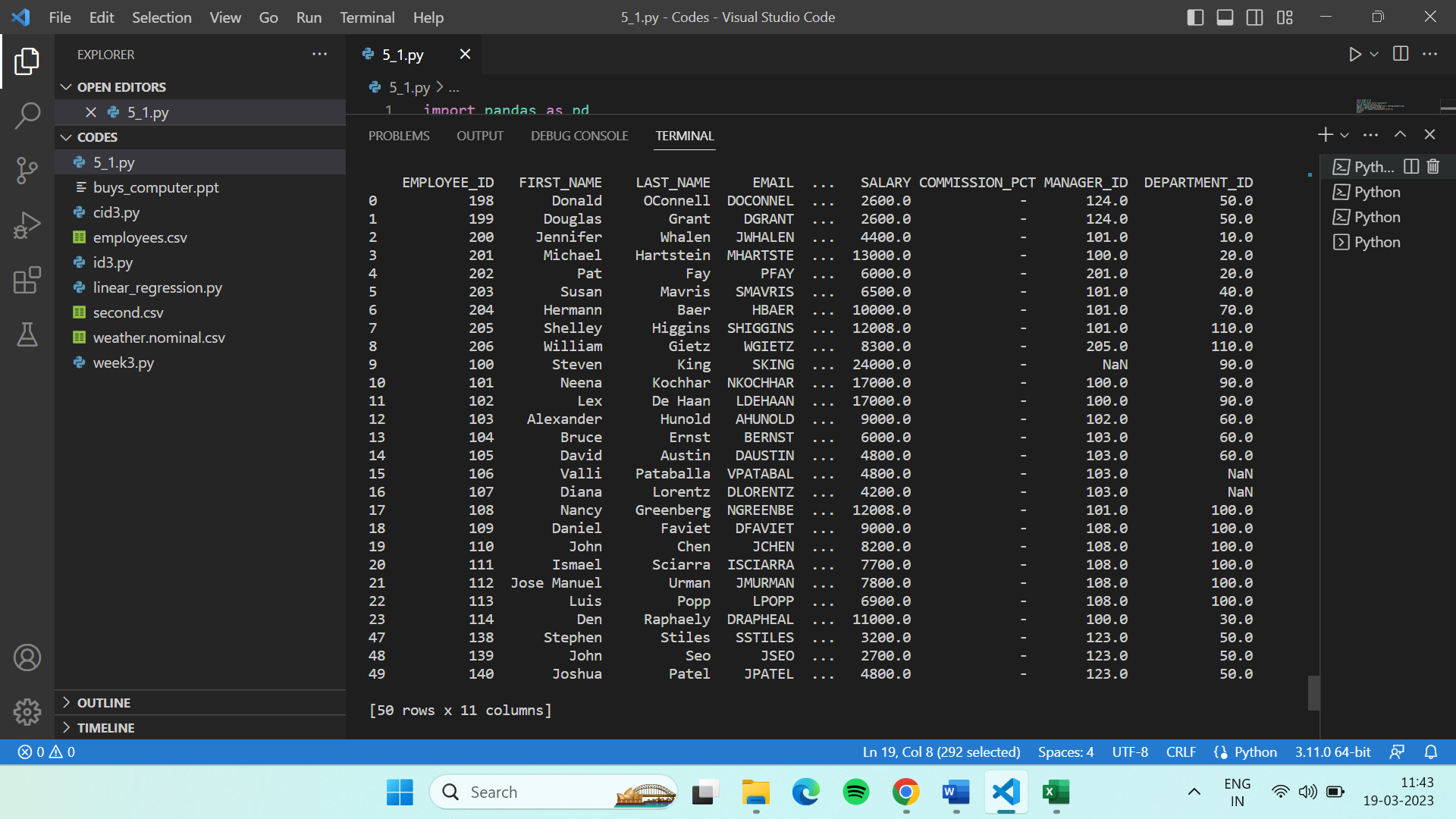
print(data)

**OUTPUT:**

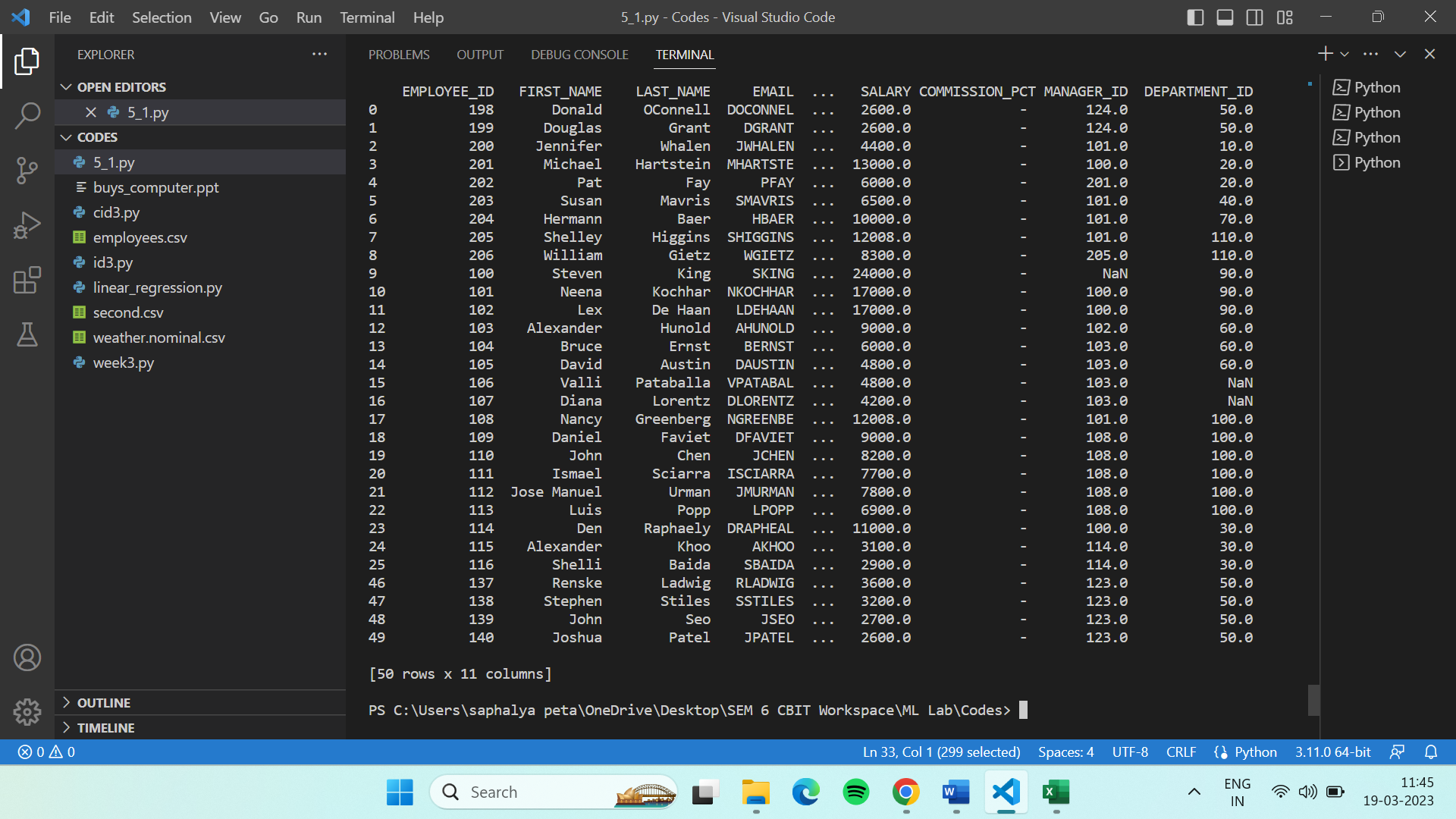
1. **Mean**



1. **Median**



1. **Most frequent**



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

**AIM:** To download any data which contains one categorical field, and applying 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”).

**PROGRAM:**

import pandas as pd

import numpy as np

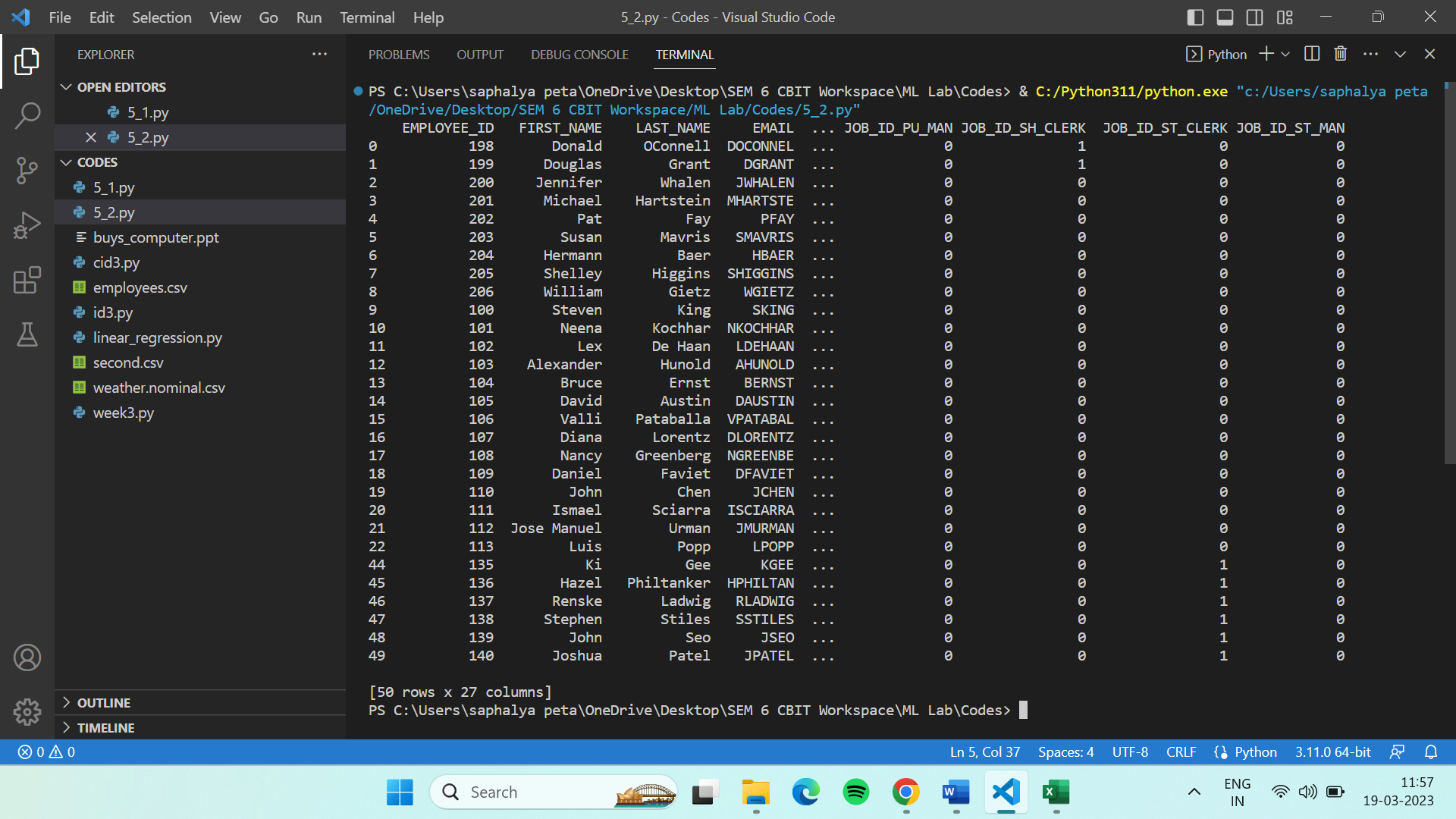
from sklearn.impute import SimpleImputer

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

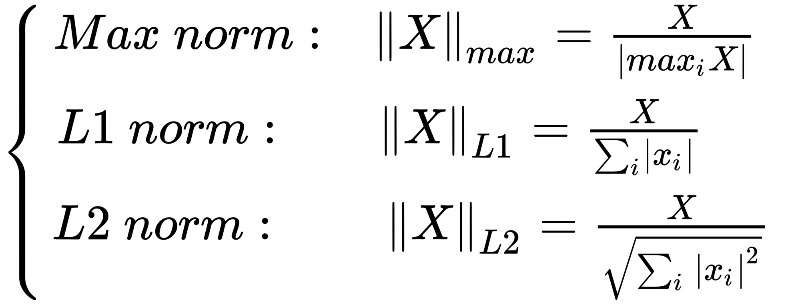
hot\_encoded\_data = pd.get\_dummies(df,columns=['JOB\_ID'])

print(hot\_encoded\_data)

**OUTPUT:**



1. **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**



**AIM:** To download a data set and apply L1, L2 and Max Normalization on the continuous data column and replace the data with new data.

**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.

**PROGRAM:**

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)

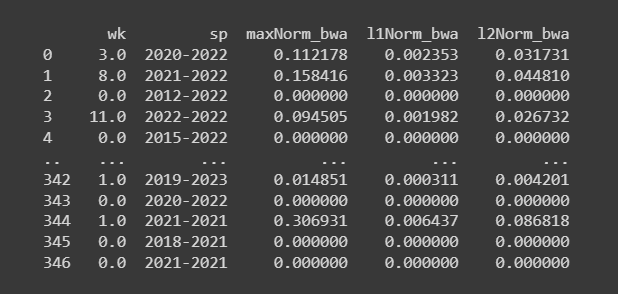
data['maxNorm\_bwa']=maxnorm

data['l1Norm\_bwa']=l1norm

data['l2Norm\_bwa']=l2norm

print(data)

**OUTPUT:**

****

**WEEK – 6**

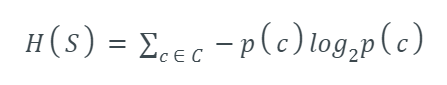
1. **Implement ID3 algorithm on**
2. **‘weather.csv’ dataset**
3. **‘buys\_computer’ dataset**

**AIM:** To implement ID3 algorithm on the given data sets.

**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).

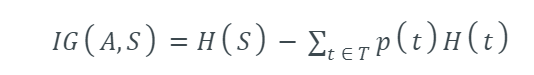
Entropy is a measure of the amount of uncertainty in the dataset S. Mathematical Representation of Entropy is shown here –



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 –



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.

**PROGRAM:**

import pandas as pd

import math

import numpy as np

data = pd.read\_csv('weather.nominal.csv')

features = [feat for feat in data]

features.remove("play")

#df = pd.DataFrame(data)

# print(df)

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)

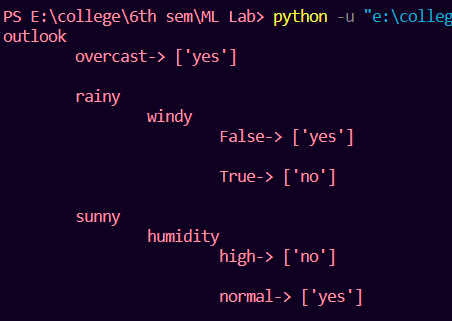
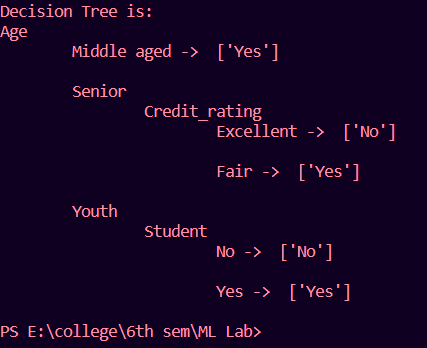
root = ID3(data, features)

print("Decision Tree is:")

printTree(root)

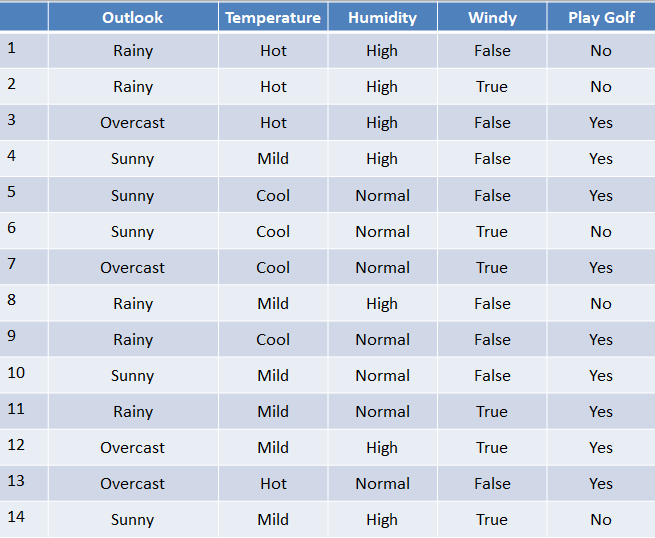
**OUTPUT:**

* 1. **For ‘weather.csv’ data set b. For ‘buys\_computer.csv’ data set**

****

**WEEK – 7**

1. **Apply Gaussian Naïve Bayes Theorem on the following dataset:**

****

**AIM:** To implement Naïve Bayes Algorithm

**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.



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.

**PROGRAM:**

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)

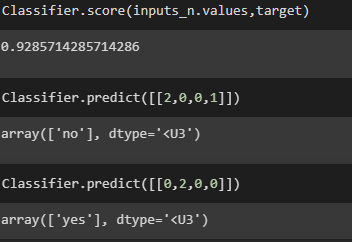
classifier = GaussianNB()

classifier.fit(inputs\_n,target)

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

classifier.predict([ [0,0,0,0] ])

**OUTPUT:**

****

**WEEK – 8**

1. **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)**

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

**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.

**PROGRAM:**

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)

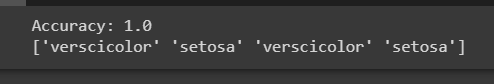
print("Accuracy:", accuracy)

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:**

****

1. **Apply clustering algorithms – k – means, agglomerative and DBSCAN to classify for any standard datasets.**

**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)

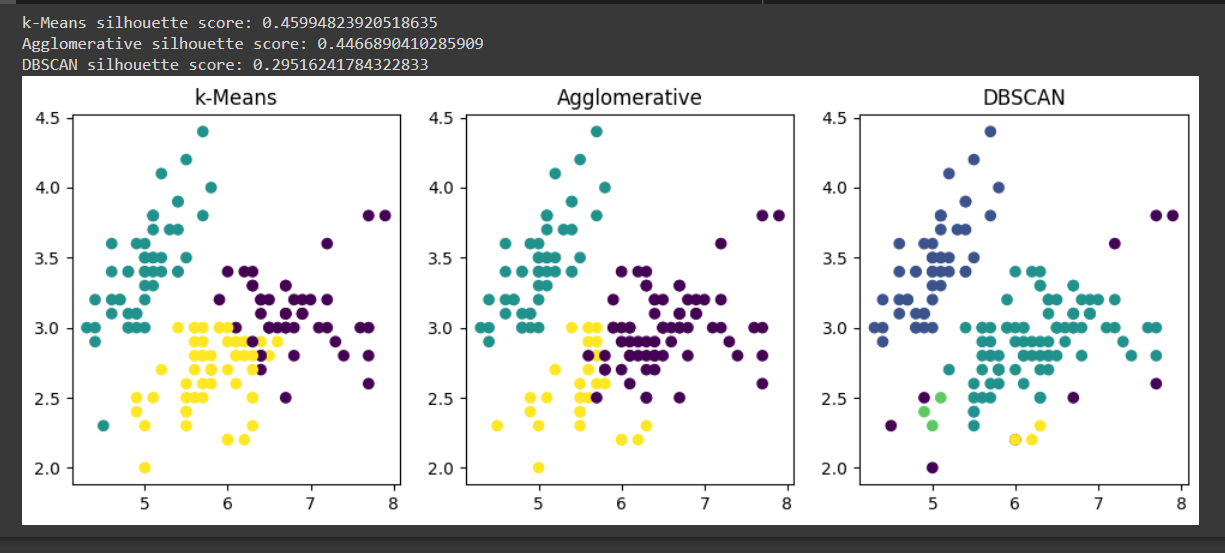
ax2.set\_title("Agglomerative")

ax3.scatter(X[:, 0], X[:, 1], c=dbscan\_labels)

ax3.set\_title("DBSCAN")

plt.show()

**OUTPUT:**

****

**WEEK – 9**

1. **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.**

**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:

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:

Precision = TP / (TP + FP)

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

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\_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)

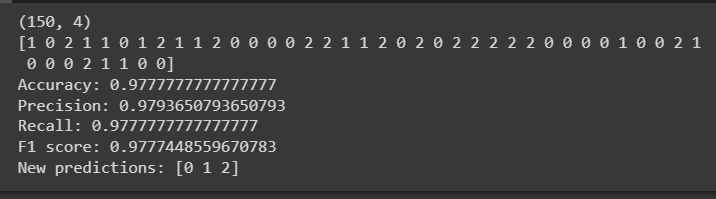
print("F1 score:", f1)

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:**

****

**WEEK – 10**

1. **Down any one data set from Kaggle or any other repository. Apply the following supervised learning algorithms**
2. **Logistic regression**
3. **ID3**
4. **Random forest**
5. **XG Boost**
6. **Naïve Bayes Algorithm**

**Compare the accuracy and plot appropriate graphs**

**AIM:** To apply the above mentioned 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")

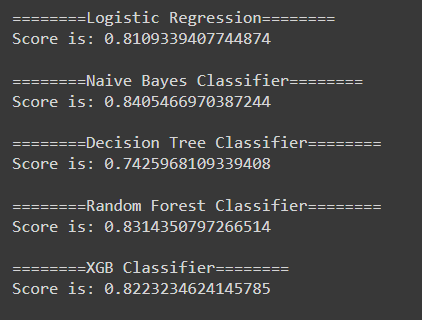
  plt.show()

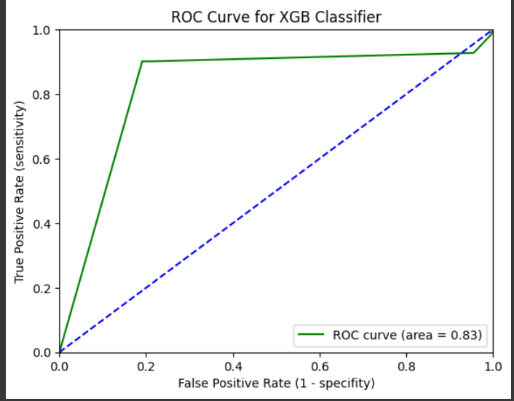
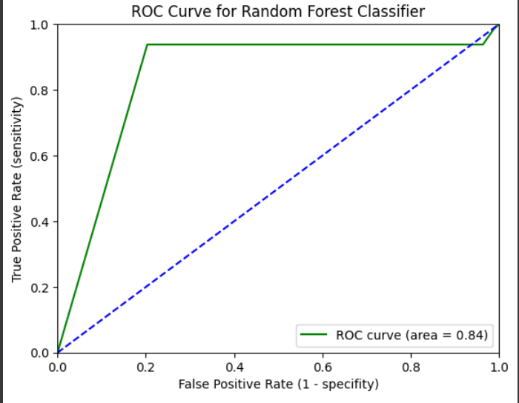
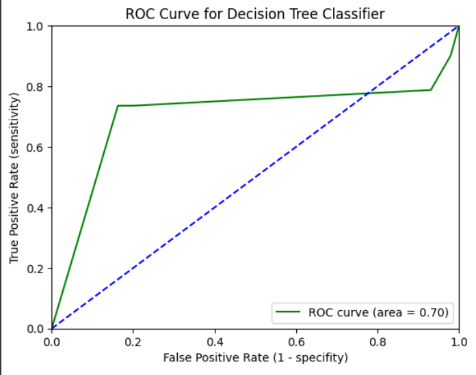
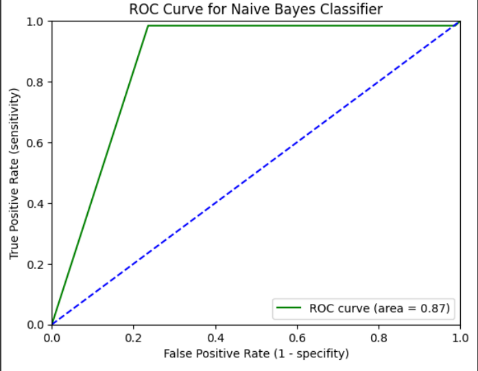
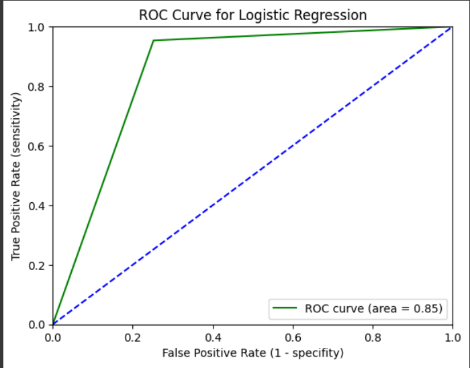
  roc\_curve(y\_test,pred\_res,pos\_label=4)

for model\_name, model in model\_dict.items():

  Rocplot(x\_train,y\_train,x\_test,y\_test,model,model\_name)

**OUTPUT:**

****

****