Practical No: 01

**Aim: Implementation of logic programming using prolog-water jug problem (depth first search) DFS**

**Code:**

water\_jug(X, Y) :-

(X > 4, Y < 3, write('4L jug overflow'), nl);

(X > 4, Y > 3, write('3L jug overflow'), nl);

(X > 4, Y > 3, write('Both jug overflow'), nl);

(X =:= 0, Y =:= 0,

nl, write('4L:0 & 3L:3 (Action : Fill 3L jug.)'),

YY is 3, water\_jug(X, YY));

(X =:= 0, Y =:= 0,

nl, write('4L:4 & 3L:0 (Action : Fill 4L jug.)'),

XX is 4, water\_jug(XX, Y));

(X =:= 2, Y =:= 0,

nl, write('4L:2 & 3L:0 (Action : Goal Stage reached...)'));

(X =:= 4, Y =:= 0,

nl, write('4L:1 & 3L:3 (Action : Pour water from 4L to 3L jug.)'),

XX is X - 3, YY is 3, water\_jug(XX, YY));

(X =:= 0, Y =:= 3,

nl, write('4L:3 & 3L:0 (Action : Pour water from 3L to 4L jug.)'),

XX is 3, YY is 0, water\_jug(XX, YY));

(X =:= 1, Y =:= 3,

nl, write('4L:1 & 3L:0 (Action : Empty 3L jug.)'),

YY is 0, water\_jug(X, YY));

(X =:= 3, Y =:= 0,

nl, write('4L:3 & 3L:3 (Action : Fill 3L jug.)'),

YY is 3, water\_jug(X, YY));

(X =:= 3, Y =:= 3,

nl, write('4L:4 & 3L:2 (Action : Pour water from 3L jug to 4L jug until 4L jug is full.)'),

XX is X + 1, YY is Y - 1, water\_jug(XX, YY));

(X =:= 1, Y =:= 0,

nl, write('4L:0 & 3L:1 (Action : Pour water from 4L jug to 3L jug.)'),

XX is Y, YY is X, water\_jug(XX, YY));

(X =:= 0, Y =:= 1,

nl, write('4L:4 & 3L:1 (Action : Pour water from 4L.)'),

XX is 4, water\_jug(XX, Y));

(X =:= 4, Y =:= 1,

nl, write('4L:2 & 3L:3 (Action : Pour water from 4L jug to 3L jug until 3L jug is full.)'),

XX is X - 2, YY is Y + 2, water\_jug(XX, YY));

(X =:= 2, Y =:= 3,

nl, write('4L:2 & 3L:0 (Action : Empty 3L jug.)'),

YY is 0, water\_jug(X, YY));

(X =:= 4, Y =:= 2,

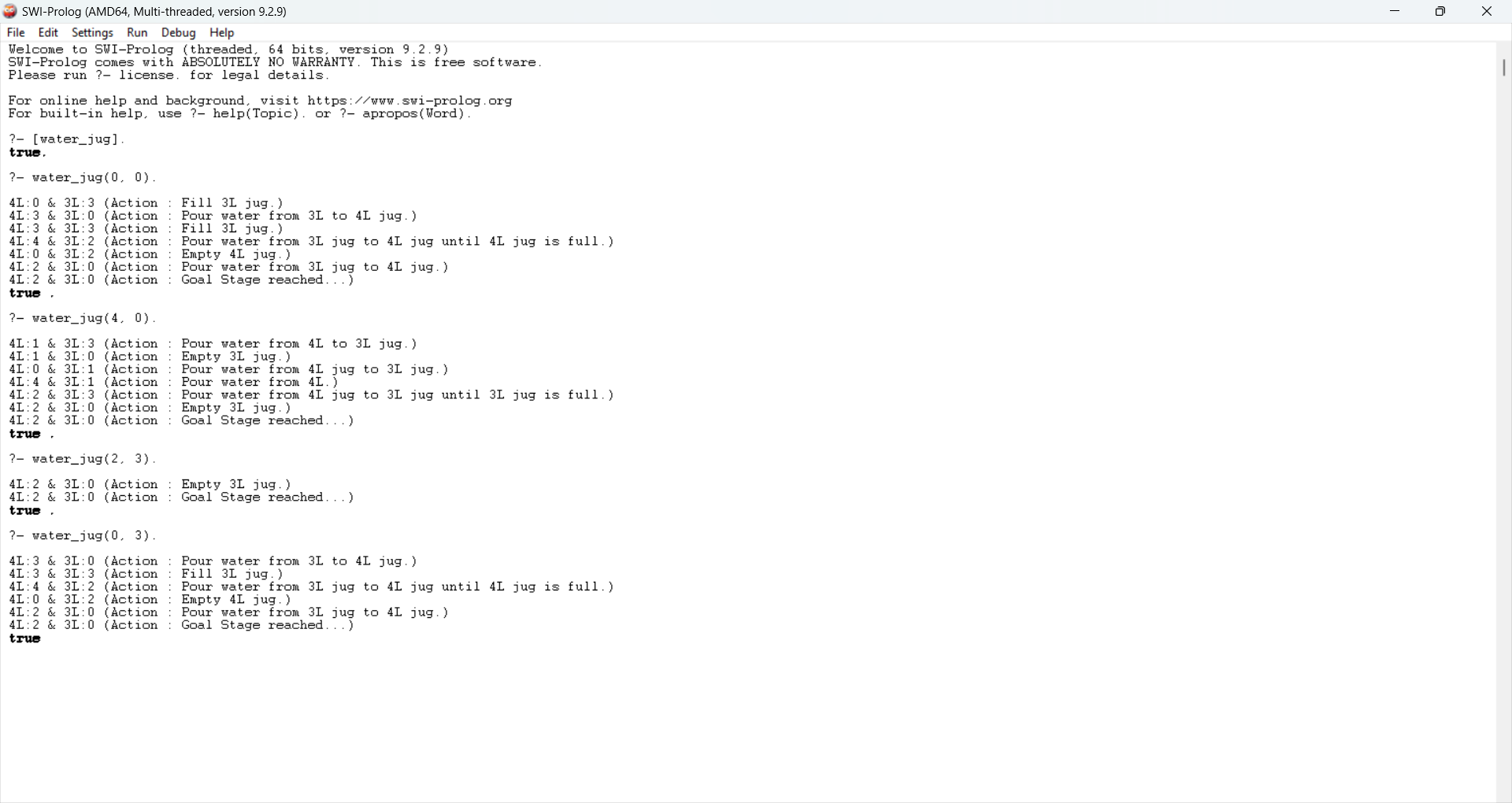
nl, write('4L:0 & 3L:2 (Action : Empty 4L jug.)'),

XX is 0, water\_jug(XX, Y));

(X =:= 0, Y =:= 2,

nl, write('4L:2 & 3L:0 (Action : Pour water from 3L jug to 4L jug.)'),

XX is Y, YY is X, water\_jug(XX, YY)).



Practical No: 02

**Aim: Implementation of logic programming using prolog (BFS) – Tic Tac Toe**

**Code:**

% Minimal Tic Tac Toe game in Prolog (2-player, terminal-based)

% Initial empty board

board([' ', ' ', ' ',

' ', ' ', ' ',

' ', ' ', ' ']).

% Display the board

display\_board([A,B,C,D,E,F,G,H,I]) :-

format('~w | ~w | ~w~n', [A,B,C]),

format('--+---+--~n'),

format('~w | ~w | ~w~n', [D,E,F]),

format('--+---+--~n'),

format('~w | ~w | ~w~n~n', [G,H,I]).

% Make a move: replace N-th position (1-indexed) with X or O

move(Board, Pos, Player, NewBoard) :-

nth1(Pos, Board, ' '), % Ensure the spot is empty

replace(Board, Pos, Player, NewBoard).

% Replace helper

replace([\_|T], 1, X, [X|T]).

replace([H|T], I, X, [H|R]) :-

I > 1, I1 is I - 1, replace(T, I1, X, R).

% Win conditions

win(Board, Player) :-

member([A,B,C], [[1,2,3], [4,5,6], [7,8,9],

[1,4,7], [2,5,8], [3,6,9],

[1,5,9], [3,5,7]]),

nth1(A, Board, Player),

nth1(B, Board, Player),

nth1(C, Board, Player).

% Start game

play :-

board(B), display\_board(B),

play\_turn(B, 'X').

% Alternate turns

play\_turn(Board, Player) :-

write(Player), write("'s turn. Enter position (1-9): "),

read(Pos),

move(Board, Pos, Player, NewBoard),

display\_board(NewBoard),

( win(NewBoard, Player) ->

write(Player), write(' wins!'), nl

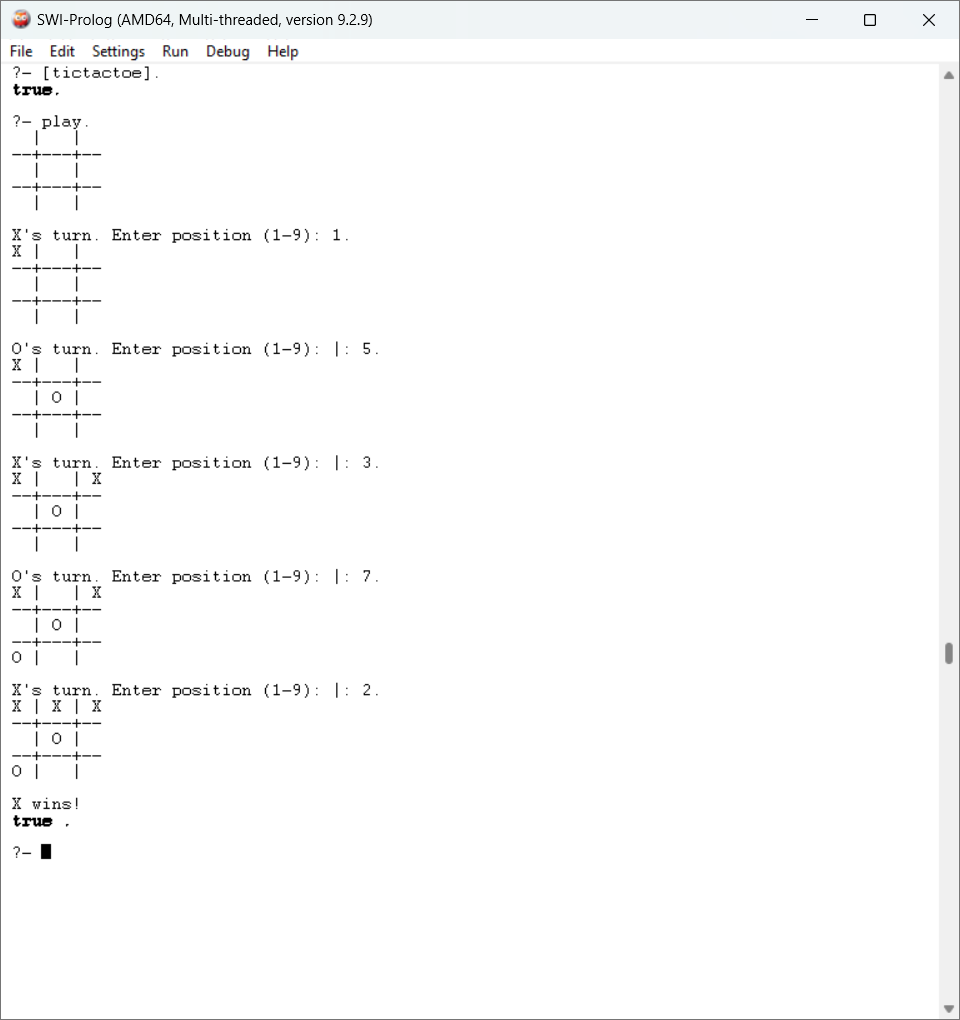
; switch(Player, Next), play\_turn(NewBoard, Next)

).

% Switch player

switch('X', 'O').

switch('O', 'X').



Practical No: 03

**Aim: Implementation of logic programming using prolog-hill climbing (to solve 8 puzzle problem)**

**Code:**

% Simple Prolog Planner for the 8 Puzzle Problem

% -----------------------------

% Test Predicate to Start Puzzle

% -----------------------------

test(Plan):-

write('Initial state:'), nl,

Init = [at(tile4,1), at(tile3,2), at(tile8,3), at(empty,4), at(tile2,5), at(tile6,6), at(tile5,7), at(tile1,8), at(tile7,9)],

write\_sol(Init),

Goal = [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4), at(empty,5), at(tile5,6), at(tile6,7), at(tile7,8), at(tile8,9)],

nl, write('Goal state:'), nl,

write(Goal), nl, nl,

solve(Init, Goal, Plan).

% -----------------------------

% Entry Point for the Solver

% -----------------------------

solve(State, Goal, Plan) :-

solve(State, Goal, [], Plan).

% -----------------------------

% Check if Goal is Reached

% -----------------------------

solve(State, Goal, Plan, Plan):-

is\_subset(Goal, State), nl,

write('Solution Plan:'), nl,

write\_sol(Plan).

% -----------------------------

% Recursive Solver with Action Selection

% -----------------------------

solve(State, Goal, Sofar, Plan):-

act(Action, Preconditions, Delete, Add),

exclude(is\_movable\_condition, Preconditions, RealPreconds),

is\_subset(RealPreconds, State),

check\_conditions(Preconditions),

\+ member(Action, Sofar),

delete\_list(Delete, State, Remainder),

append(Add, Remainder, NewState),

solve(NewState, Goal, [Action|Sofar], Plan).

% -----------------------------

% Define Move Action

% -----------------------------

act(move(X,Y,Z),

[at(X,Y), at(empty,Z), is\_movable(Y,Z)],

[at(X,Y), at(empty,Z)],

[at(X,Z), at(empty,Y)]).

% -----------------------------

% Condition Evaluation Helper

% -----------------------------

is\_movable(X1, Y1) :-

(X1 - Y1 =:= 1 ; X1 - Y1 =:= -1 ; X1 - Y1 =:= 3 ; X1 - Y1 =:= -3).

is\_movable\_condition(is\_movable(\_, \_)).

check\_conditions([]).

check\_conditions([is\_movable(Y,Z)|T]) :-

is\_movable(Y,Z),

check\_conditions(T).

check\_conditions([\_|T]) :-

check\_conditions(T).

% -----------------------------

% Utility Predicates

% -----------------------------

is\_subset([H|T], Set):-

member(H, Set),

is\_subset(T, Set).

is\_subset([], \_).

delete\_list([H|T], Curstate, Newstate):-

remove(H, Curstate, Remainder),

delete\_list(T, Remainder, Newstate).

delete\_list([], Curstate, Curstate).

remove(X, [X|T], T).

remove(X, [H|T], [H|R]) :-

remove(X, T, R).

write\_sol([]).

write\_sol([H|T]) :-

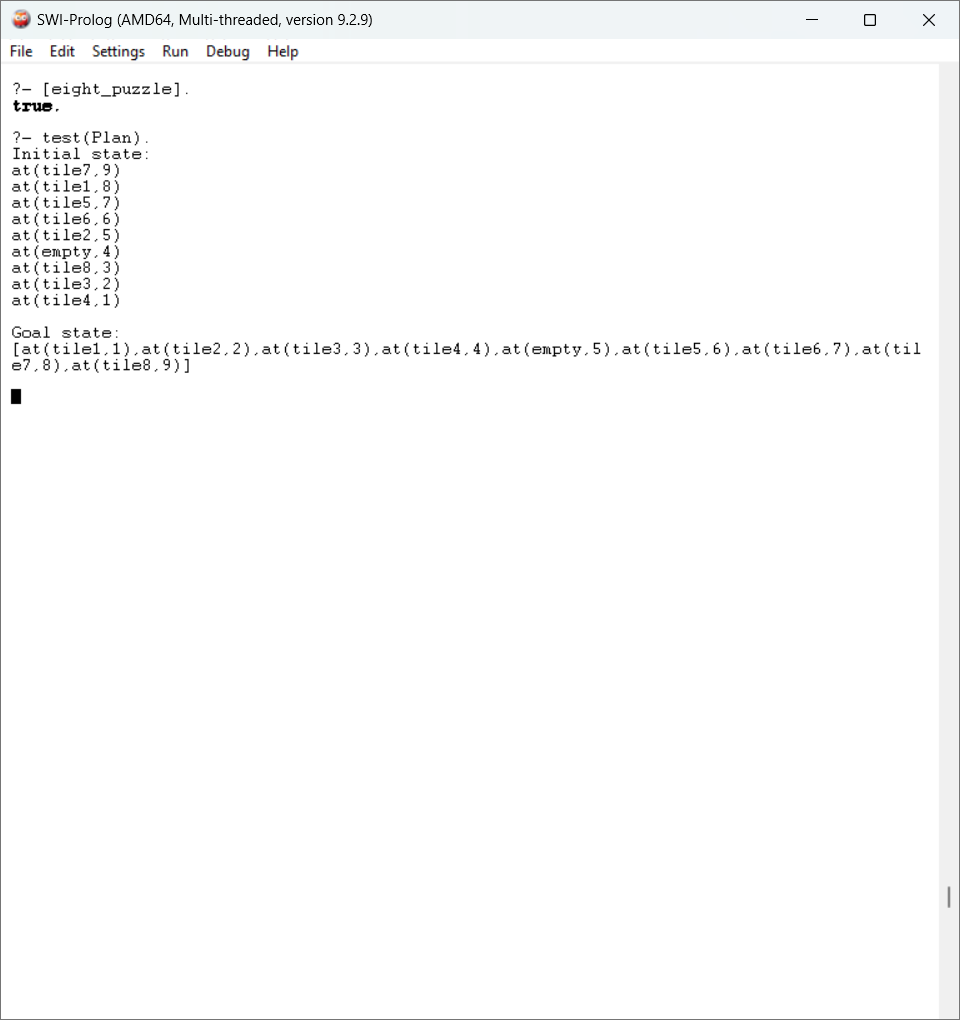
write\_sol(T),

write(H), nl.

append([H|T], L1, [H|L2]) :-

append(T, L1, L2).

append([], L, L).

****

Practical No: 04

**Aim: Introduction to python libraries - Pandas and NumPy**

**Code:**

**#Numpy Library**

import numpy as np

x = np.array([55,95,52,34,2])

print(type(x))



print(x)



x = np.array([4,5,'n',2000])

print(x)



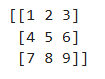
d = np.arange(start=1, stop=10, step=3)

print(d)

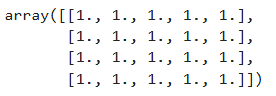


grid = np.arange(start=1, stop=10).reshape(3,3)

print(grid)



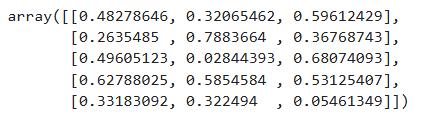
np.ones((4,5))



np.random.rand(5)



np.random.rand(5,3)



np.logspace(1,10, num=5, endpoint=True, base=10.0)



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

a.shape



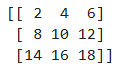
b = np.array([[1,2,3],[4,5,6],[7,8,9]])

b.shape



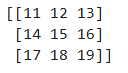
add = np.add(a,b)

print(add)



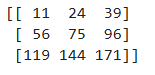
c = np.arange(start=11, stop=20).reshape(3,3)

print(c)



mul = np.multiply(b,c)

print(mul)



a[:,0]



a[0,:]



a[0,1]

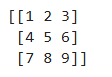


a[1:3]



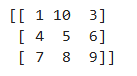
a\_sub = a[:3,:3]

print(a\_sub)



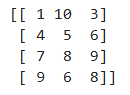
a\_sub[0,1]=10

print(a\_sub)



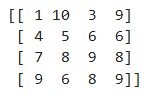
s\_row = np.append(a,[[9,6,8]],axis=0)

print(s\_row)



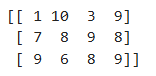
s\_col = np.append(s\_row,[[9],[6],[8],[9]], axis=1)

print(s\_col)

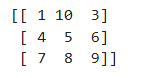


a\_del = np.delete(s\_col, 1, axis=0)

print(a\_del)



print(a)

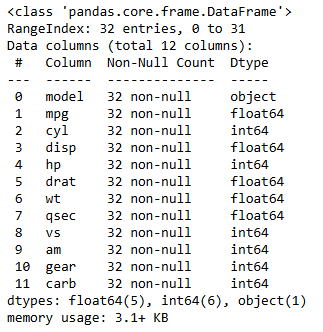


**#Pandas Library**

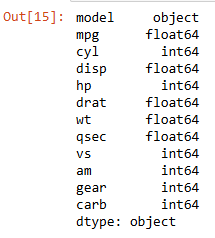
import pandas as pd

data1= pd.read\_csv('D:/Mriganka Adhikary 69/mtcars.csv')

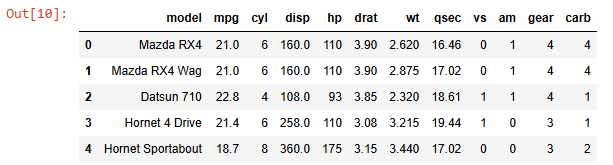
data1.info()



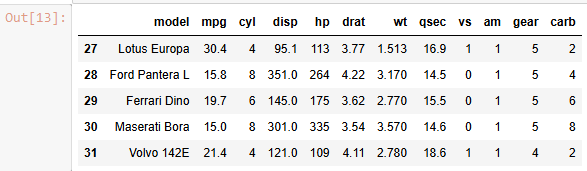
data1.dtypes



data1.head()

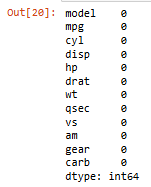


data1.tail()

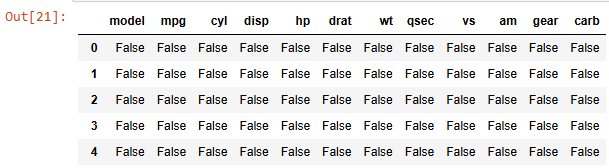
data1.size

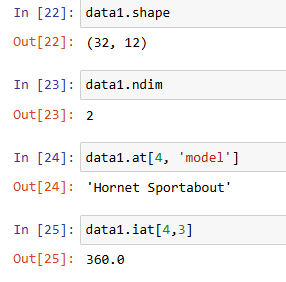


data1.isnull().sum()

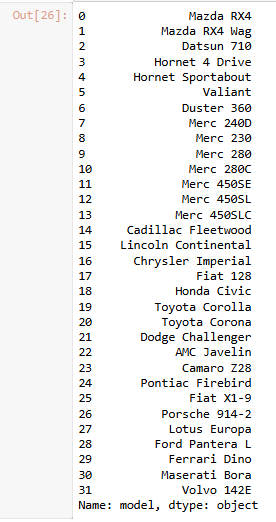


data1.isnull()

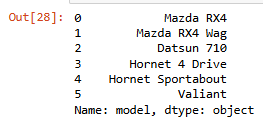




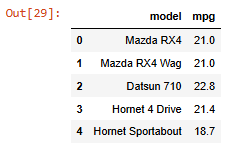
data1.loc[:,'model']



data1.loc[0:5,'model']



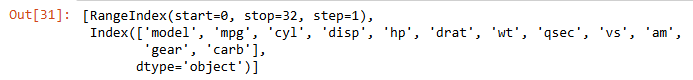
data1.iloc[0:5, 0:2]



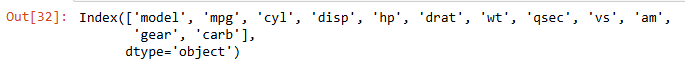
data1['model'].dtype

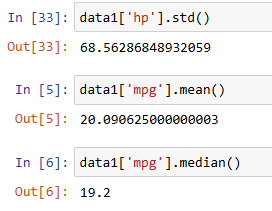


data1.axes

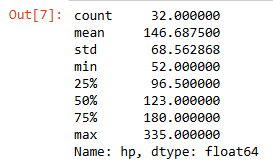


data1.columns

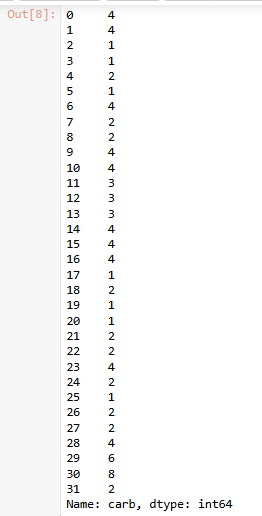




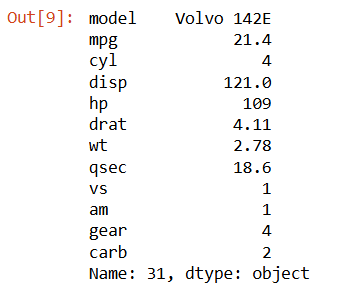
data1['hp'].describe()



data1.iloc[:,-1]

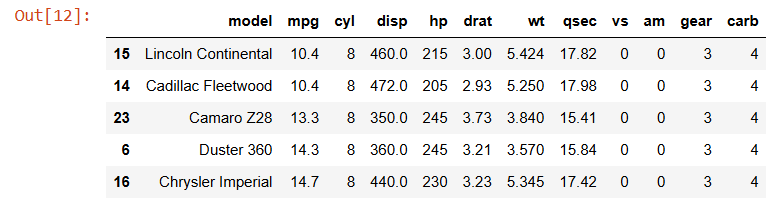


data1.iloc[-1]

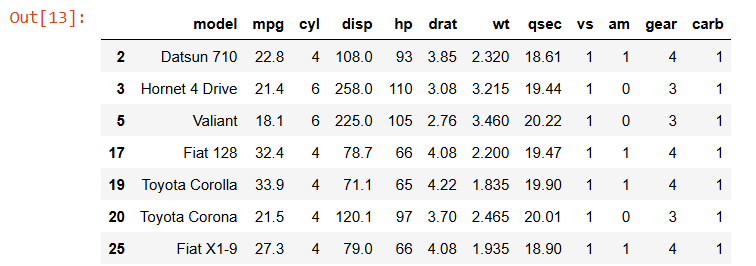


data1\_sorted=data1.sort\_values(by='mpg')

data1\_sorted.head()



data1[data1['carb']==1]



Practical No: 05

**Aim: Introduction to python libraries – matplotlib, SciPy, scikit-learn**

**Code:**

**#matplotlib**

import matplotlib.pyplot as plp

import pandas as pd

cars = pd.read\_csv(“C:/Users/dell/Mriganka Adhikary 69/mtcars.csv”)

print("Mriganka Adhikary 69")

x = [2,4,5]

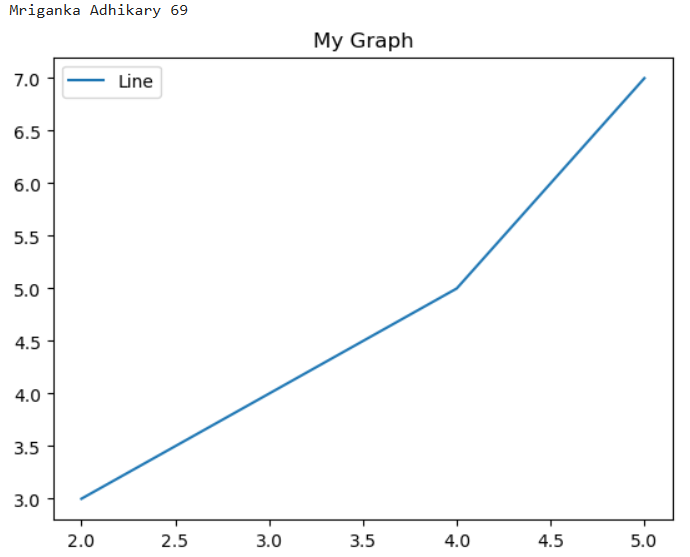
y = [3,5,7]

plp.plot(x,y)

plp.title("My Graph")

plp.legend(['Line'])

plp.show()



print("Mriganka Adhikary 69")

plp.bar(x,y)

plp.title("My Bar Graph")

plp.legend(["Bar"])

plp.show()

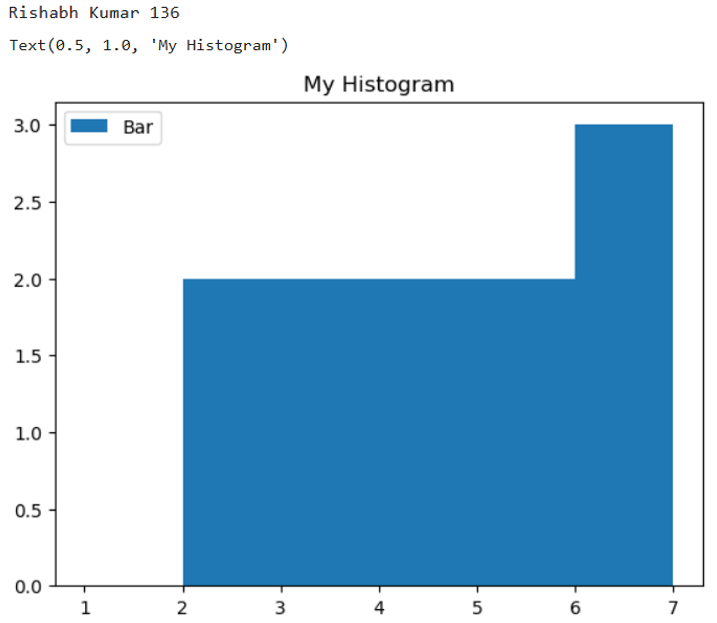
print("Mriganka Adhikary 69")

x = [2,3,4,5,2,3,6,7,8,4,5,6,0]

plp.hist(x, bins = [1,2,3,4,5,6,7])

plp.legend(['Bar'])

plp.title("My Histogram")



print("Mriganka Adhikary 69")

x = [3,4,2,3,2,5,3,7,6,8]

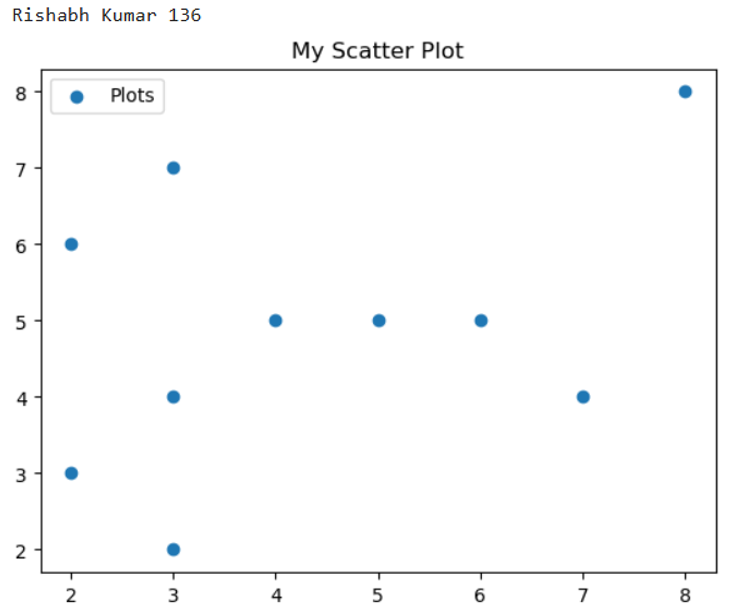
y = [4,5,6,7,3,5,2,4,5,8]

plp.scatter(x,y)

plp.title("My Scatter Plot")

plp.legend(["Plots"])

plp.show()



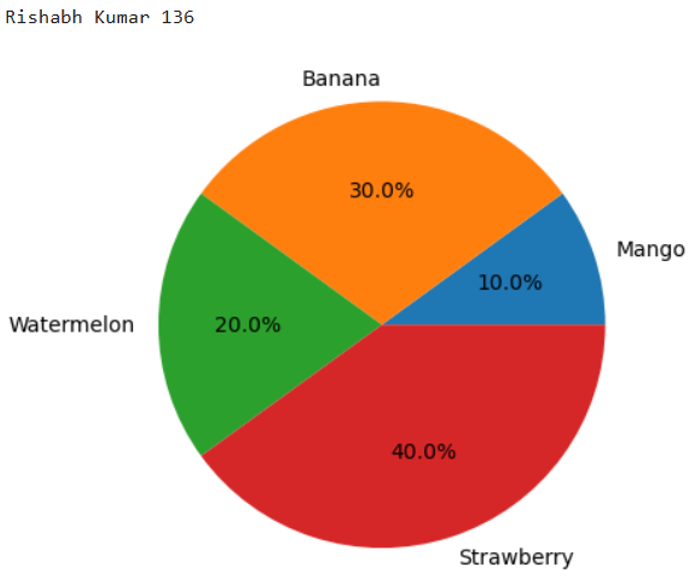
print("Mriganka Adhikary 69")

labels = ['Mango', 'Banana', 'Watermelon', 'Strawberry']

sizes = [10,30,20,40]

plp.pie(sizes, labels=labels, autopct='%1.1f%%')

plp.show()



**# SciPy**

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import curve\_fit

# Generate sample data: a noisy sine wave

x\_data = np.linspace(0, 4 \* np.pi, 50)

y\_data = np.sin(x\_data) + 0.2 \* np.random.normal(size=len(x\_data))

# Define the model function to fit

def model\_func(x, a, b, c):

return a \* np.sin(b \* x + c)

# Fit the model to the data using initial guess for parameters: a=1, b=1, c=0

params, params\_covariance = curve\_fit(model\_func, x\_data, y\_data, p0=[1, 1, 0])

# Print the optimized parameters

print("Fitted parameters:")

print(f"a = {params[0]:.3f}, b = {params[1]:.3f}, c = {params[2]:.3f}")

# Plotting the data and the fitted curve

plt.scatter(x\_data, y\_data, label='Data', color='red')

plt.plot(x\_data, model\_func(x\_data, \*params), label='Fitted Curve', color='blue')

plt.legend()

plt.title('Curve Fitting with SciPy')

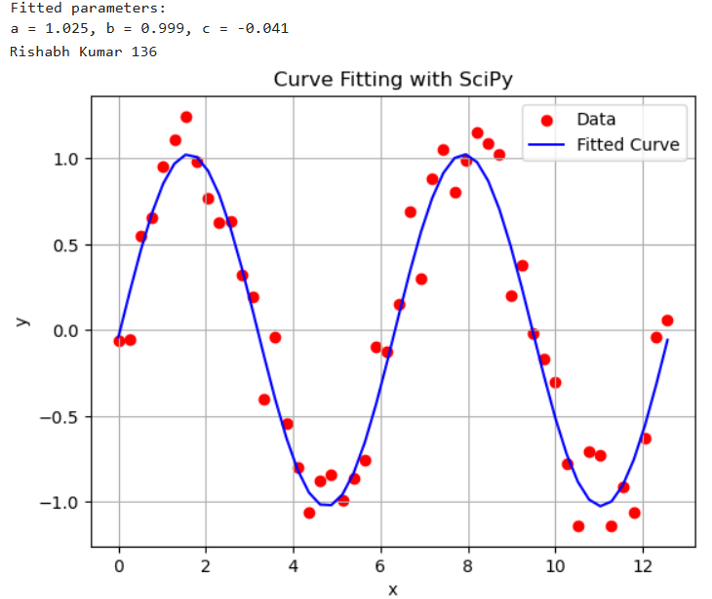
plt.xlabel('x')

plt.ylabel('y')

plt.grid(True)

print("Mriganka Adhikary 69")

plt.show()



**# Scikit Learn**

from sklearn.datasets import load\_iris

iris = load\_iris()

print("Mriganka Adhikary 69")

print(iris)

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

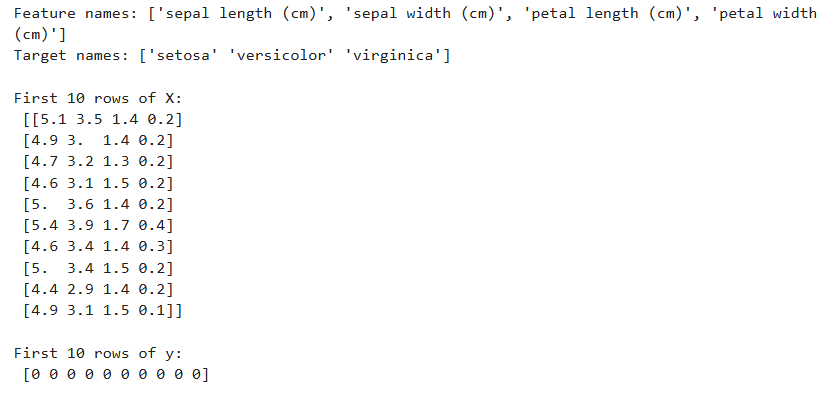
target\_names = iris.target\_names

print("Feature names:", feature\_names)

print("Target names:", target\_names)

print("\nFirst 10 rows of X:\n", X[:10])

print("\nFirst 10 rows of y:\n", y[:10])



import numpy as np

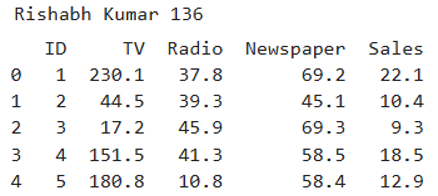
import pandas as pd

from sklearn.linear\_model import LogisticRegression

df = pd.read\_csv('C:/Users/Asus/Mriganka Adhikary 69/Advertising.csv')

print("Mriganka Adhikary 69")

print(df.head())



Practical No: 06

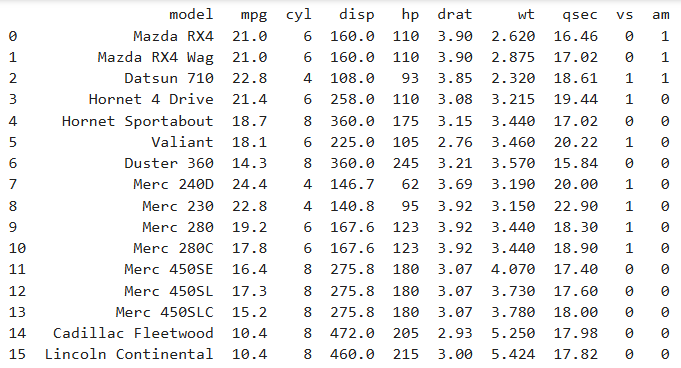
**Aim: Implementation of Exploratory Data Analysis (EDA) using Python**

**Code:**

import pandas as pd

data1 = pd.read\_csv(“C:/Users/dell/Mriganka Adhikary 69/mtcars.csv”)

print(data1)



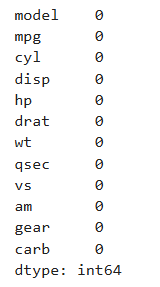
data1.head()



data1.size



data1.isnull().sum()



data1.shape



data1.ndim



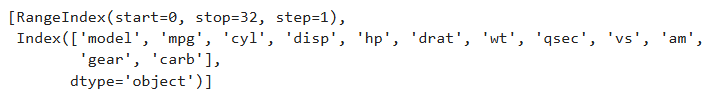
data1.at[4,'model']

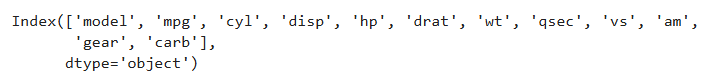


data1.iat[4,3]



data1.axes

  
data1.columns



data1['hp'].std()



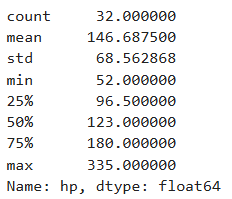
data1['mpg'].mean()



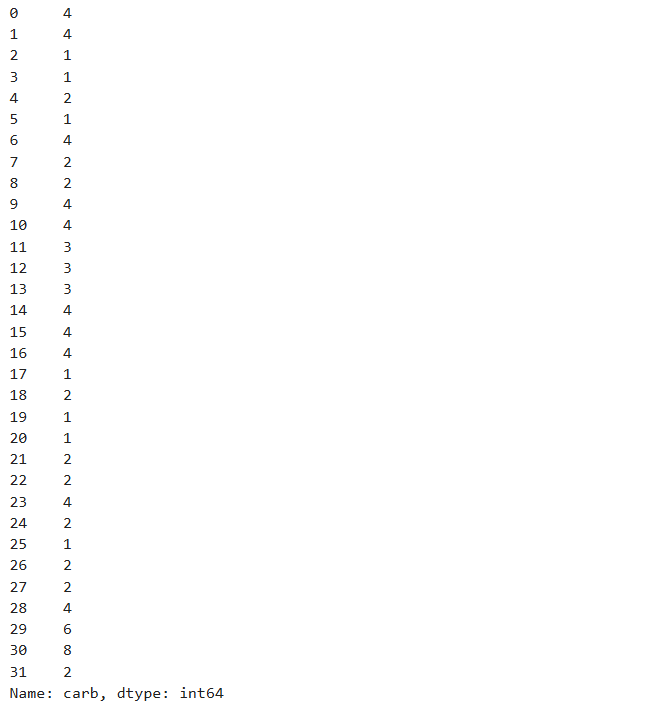
data1['mpg'].median()



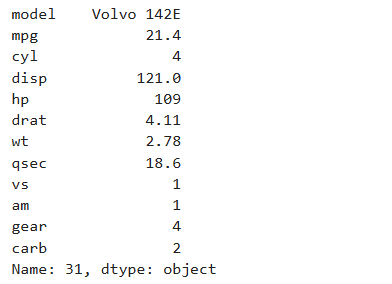
data1['hp'].describe()



data1.iloc[:,-1]

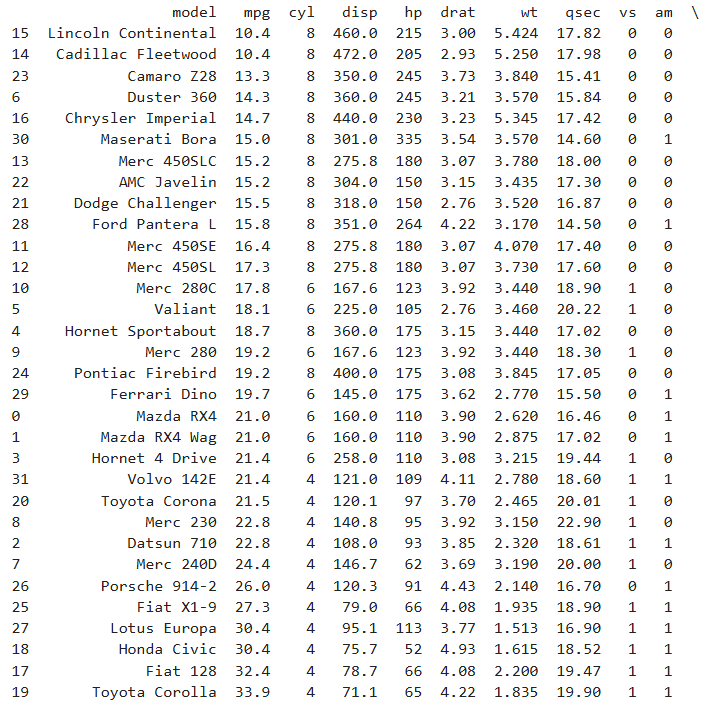


data1.iloc[-1]

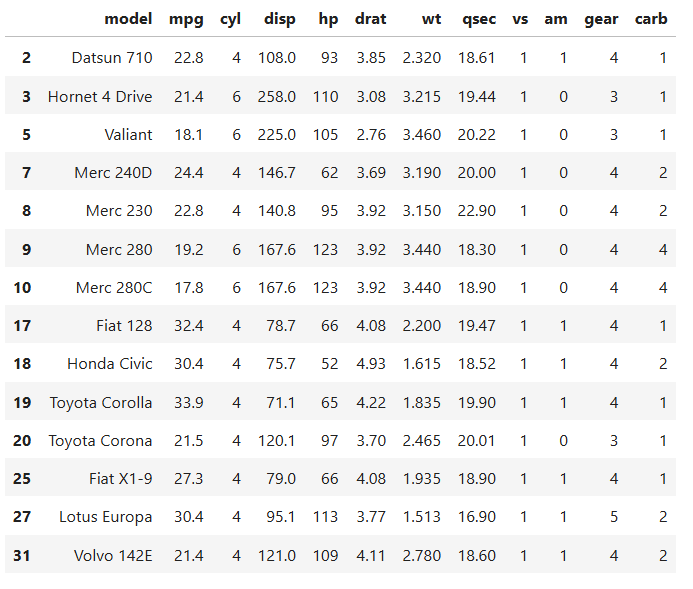


data1\_sorted = data1.sort\_values(by='mpg')

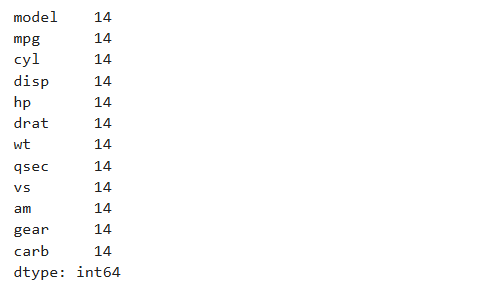
print(data1\_sorted)



data1[data1['vs']==1]



data1[data1['vs']==1].count()



data1['carb'].replace(4,'Four')



Practical No: 07

**Aim: Implementation of perceptron algorithm**

**Code:**

import numpy as np

def perceptron\_or(x1, x2):

w1 = 1

w2 = 1

b = -0.5

result = w1 \* x1 + w2 \* x2 + b

if result >= 0:

return 1

else:

return 0

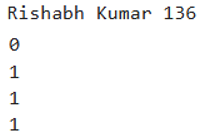
print("Mriganka Adhikary 69")

print(perceptron\_or(0, 0))

print(perceptron\_or(0, 1))

print(perceptron\_or(1, 0))

print(perceptron\_or(1, 1))



Practical No: 08

**Aim: Implementation of Adaline algo for AND operation**

**Code:**

import numpy as np

class Adaline:

def \_\_init\_\_(self, learning\_rate=0.01, n\_iter=100):

self.learning\_rate = learning\_rate

self.n\_iter = n\_iter

self.weights = None

self.bias = None

def fit(self, X, y):

# Initialize weights and bias

self.weights = np.zeros(X.shape[1])

self.bias = 0

# Perform gradient descent

for \_ in range(self.n\_iter):

# Calculate net input (weighted sum of inputs)

net\_input = np.dot(X, self.weights) + self.bias

# Calculate error (difference between prediction and actual)

error = y - net\_input

# Update weights and bias using gradient descent

self.weights += self.learning\_rate \* np.dot(X.T, error)

self.bias += self.learning\_rate \* np.sum(error)

def predict(self, X):

# Calculate net input

net\_input = np.dot(X, self.weights) + self.bias

# Apply a threshold (0.0 for linear activation)

return np.where(net\_input >= 0.0, 1, 0)

# Example Usage (AND operation)

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

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

# Create and train Adaline model

ada = Adaline(learning\_rate=0.1, n\_iter=1000)

ada.fit(X, y)

# Make predictions

predictions = ada.predict(X)

print("Mriganka Adhikary 69")

print("Predictions:", predictions)



Practical No: 09

**Aim: Implementation of Gradient descent algorithm**

**Code:**

| def predict(row, weights):  activation = weights[0]  for i in range(len(row)-1):  activation += weights[i + 1] \* row[i]  return 1.0 if activation >= 0.0 else 0.0  def predict(row, weights):  activation = weights[0]  for i in range(len(row)-1):  activation += weights[i + 1] \* row[i]  return 1.0 if activation >= 0.0 else 0.0  # test predictions  dataset = [  [2.7810836, 2.550537003, 0],  [1.465489372, 2.362125076, 0],  [3.396561688, 4.400293529, 0],  [1.38807019, 1.850220317, 0],  [3.06407232, 3.005305973, 0],  [7.627531214, 2.759262235, 1],  [5.332441248, 2.088626775, 1],  [6.922596716, 1.77106367, 1],  [8.675418651, -0.242068655, 1],  [7.673756466, 3.508563011, 1]  ]  weights = [-0.1, 0.20653640140000007, -0.23418117710000003]  for row in dataset:  prediction = predict(row, weights)  print("Expected=%d, Predicted=%d" % (row[-1], prediction))    # Estimate Perceptron weights using stochastic gradient descent  def train\_weights(train, l\_rate, n\_epoch):  weights = [0.0 for i in range(len(train[0]))] # Initialize weights to 0.0  for epoch in range(n\_epoch):  sum\_error = 0.0    for row in train:  prediction = predict(row, weights) # Make a prediction using current weights  error = row[-1] - prediction # Calculate the error as the difference between actual and predicted    sum\_error += error \*\* 2 # Accumulate the sum of squared errors for this epoch    weights[0] = weights[0] + l\_rate \* error # Update the bias (weights[0])    for i in range(len(row) - 1):  weights[i + 1] = weights[i + 1] + l\_rate \* error \* row[i] # Update weights for features  print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l\_rate, sum\_error)) # Print epoch details  return weights # Return the trained weights after all epochs  # Function to make predictions using weights  def predict(row, weights):  activation = weights[0]  for i in range(len(row) - 1):  activation += weights[i + 1] \* row[i]  return 1.0 if activation >= 0.0 else 0.0  # Function to train perceptron weights using stochastic gradient descent  def train\_weights(train, l\_rate, n\_epoch):  weights = [0.0 for i in range(len(train[0]))] # Initialize weights to 0.0 for each feature  for epoch in range(n\_epoch):  sum\_error = 0.0  for row in train:  prediction = predict(row, weights) # Make prediction using current weights  error = row[-1] - prediction # Calculate error as actual - predicted  sum\_error += error \*\* 2 # Accumulate squared error  weights[0] = weights[0] + l\_rate \* error # Update bias (weights[0])  for i in range(len(row) - 1):  weights[i + 1] = weights[i + 1] + l\_rate \* error \* row[i] # Update weights for features  print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l\_rate, sum\_error)) # Print epoch details  return weights # Return trained weights  # Dataset for training  dataset = [  [2.7810836, 2.550537003, 0],  [1.465489372, 2.362125076, 0],  [3.396561688, 4.400293529, 0],  [1.38807019, 1.850220317, 0],  [3.06407232, 3.005305973, 0],  [7.627531214, 2.759262235, 1],  [5.332441248, 2.088626775, 1],  [6.922596716, 1.77106367, 1],  [8.675418651, -0.242068655, 1],  [7.673756466, 3.508563011, 1]  ]  l\_rate = 0.1 # Learning rate  n\_epoch = 5 # Number of epochs for training  # Train weights using the dataset  weights = train\_weights(dataset, l\_rate, n\_epoch)  print(weights) # Print the trained weights |
| --- |

Practical No: 10

**Aim: Implementation of Principle Component Analysis**

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Sample data (replace with your actual dataset)

data = {'Feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Feature2': [10, 9, 8, 7, 6, 5, 4, 3, 2, 1],

'Feature3': [1, 3, 5, 7, 9, 2, 4, 6, 8, 10]

}

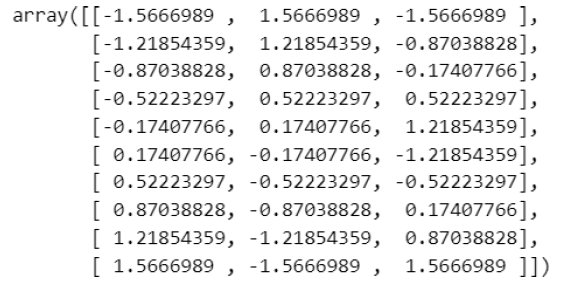
df = pd.DataFrame(data)

# Feature scaling

x = df.values

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

x



# Apply PCA with 2 components

pca = PCA(n\_components=2)

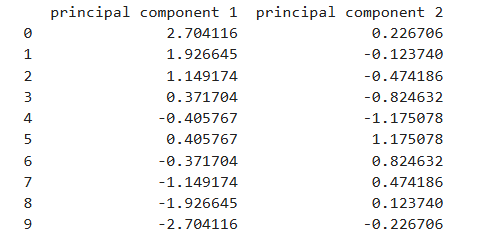
principal\_components = pca.fit\_transform(x)

# Create a new DataFrame with the principal components

principal\_df = pd.DataFrame(data=principal\_components,

columns=['principal component 1', 'principal component 2'])

print(principal\_df)



# Visualize the results

plt.figure(figsize=(8, 6))

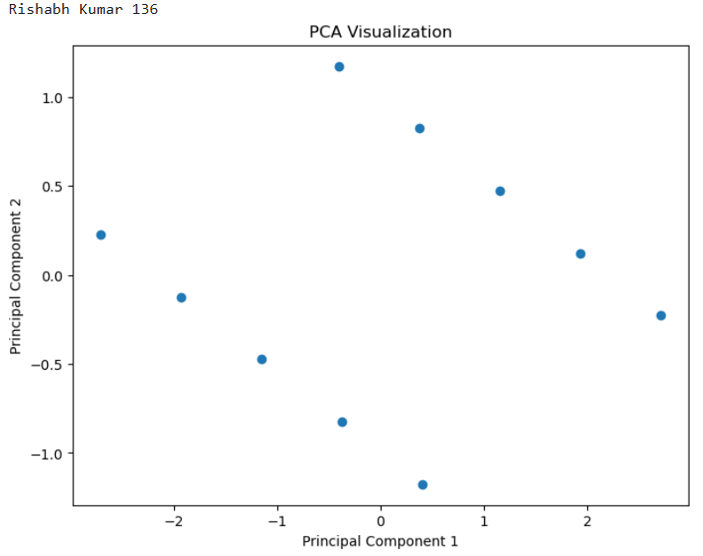
plt.scatter(principal\_df['principal component 1'], principal\_df['principal component 2'])

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('PCA Visualization')

plt.show()

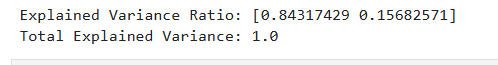


# Explained variance ratio

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

print("Explained Variance Ratio:", explained\_variance\_ratio)

print("Total Explained Variance:", np.sum(explained\_variance\_ratio))



Practical No: 11

**Aim: Implementation of Normalization and Transformation**

**Code:**

| from sklearn import preprocessing  import numpy as np  x\_array = np.array([2,3,5,6,7,4,8,7,6])  normalized\_arr = preprocessing.normalize([x\_array])  print(normalized\_arr)    #Transformation  import pandas as pd  import numpy as np  # Sample data  data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],  'Age': [25, 30, 22, 28],  'City': ['New York', 'London', 'Paris', 'Tokyo']}  df = pd.DataFrame(data)  print("Original DataFrame:")  print(df)    #Adding a new column  df['Age\_Group'] = pd.cut(df['Age'], bins=[18, 25, 30, 100], labels=['Young', 'Adult', 'Senior'])  print("\nDataFrame with Age Group column:")  print(df)    # 2. Creating dummy variables for categorical features  df = pd.get\_dummies(df, columns=['City'])  print("\nDataFrame after creating dummy variables for City:")  print(df)    import pandas as pd  from sklearn import metrics  df = pd.read\_csv('D:/76/CreditRisk.csv')  print("DataFrame head:")  df.head()    df.dtypes    #Example feature extraction:  # 1. Calculate the mean of a numerical column  #num\_cols=df.select\_dtypes(include=['number']).columns  num\_cols = df.select\_dtypes(include=np.number)  num\_cols    df['ApplicantIncome'].mean()    obj\_cols=df.select\_dtypes(exclude=['number']).columns  obj\_cols    X = df.drop('Loan\_Status', axis=1)  y =df['Loan\_Status']  y.value\_counts()    pd.get\_dummies(df,'Gender')    correlation\_matrix =num\_cols.corr()  correlation\_matrix |
| --- |

Practical No: 12

**Aim: Implementation of Logistic Regression**

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

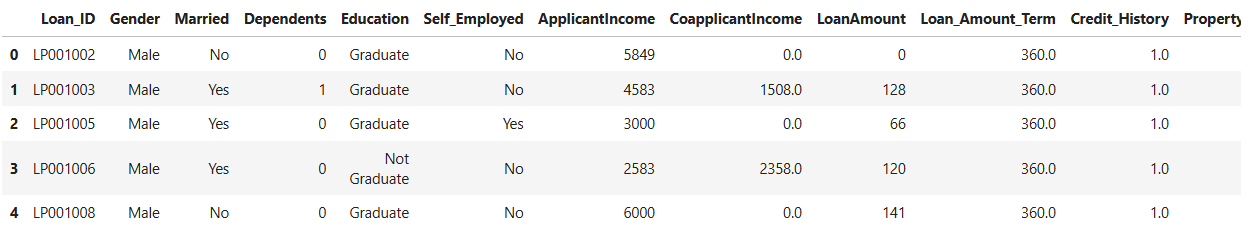
import seaborn as sns

credit\_df= pd.read\_csv(r"C:\Users\Asus\Mriganka Adhikary 69\CreditRisk.csv")

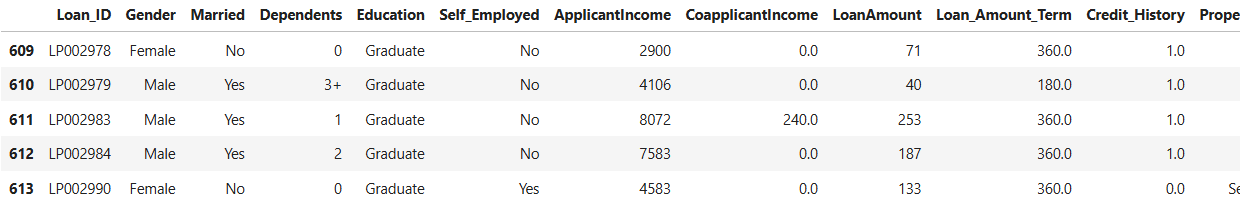
credit\_df.shape



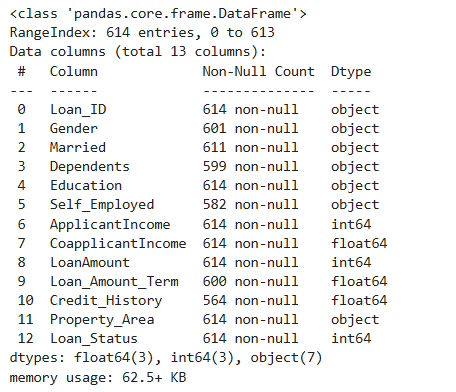
credit\_df.head()



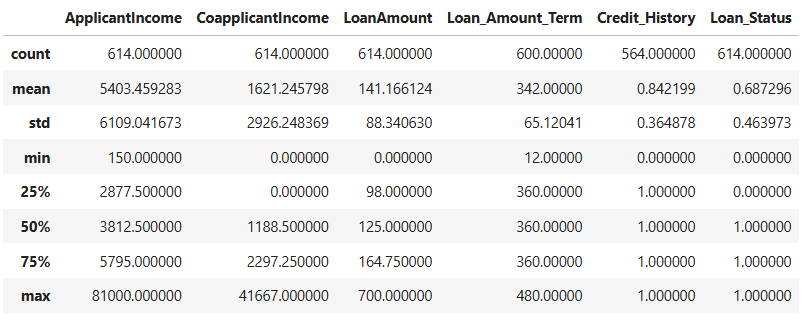
credit\_df.tail()



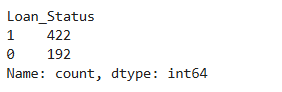
credit\_df.info()



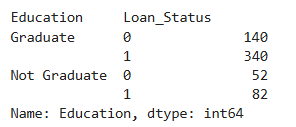
credit\_df.describe()



credit\_df.Loan\_Status.value\_counts()

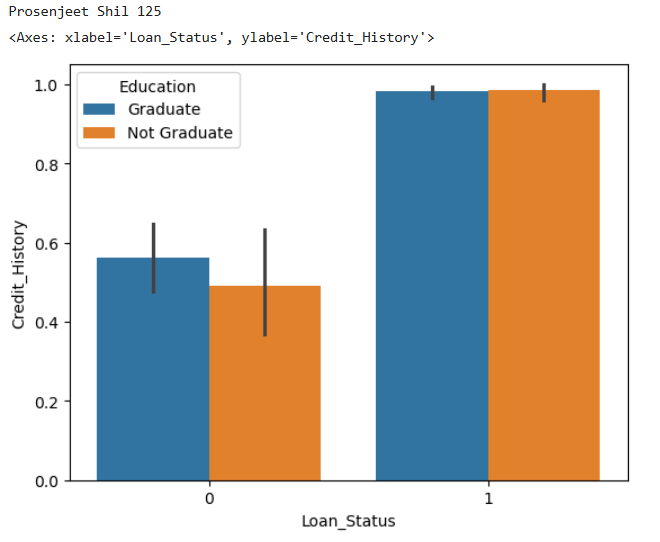


credit\_df.groupby(['Education','Loan\_Status']).Education.count()

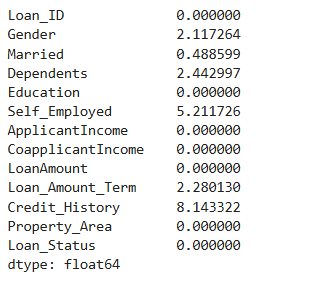


print("Mriganka Adhikary 69")

sns.barplot(y='Credit\_History',x='Loan\_Status',hue='Education',data=credit\_df)

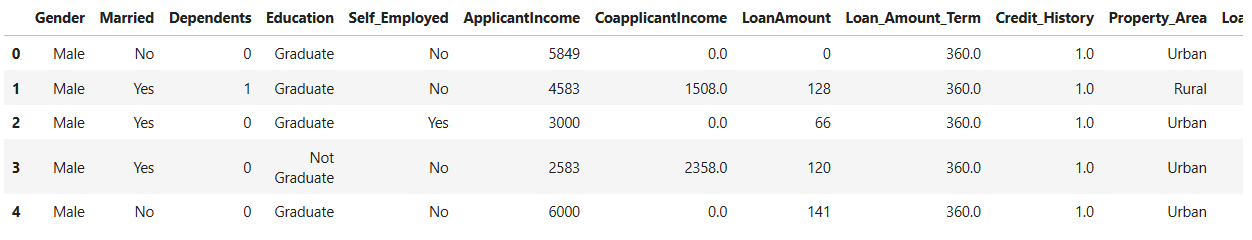


100\*credit\_df.isnull().sum()/credit\_df.shape[0]



DF=credit\_df.drop(credit\_df.columns[0],axis=1)

DF.head()



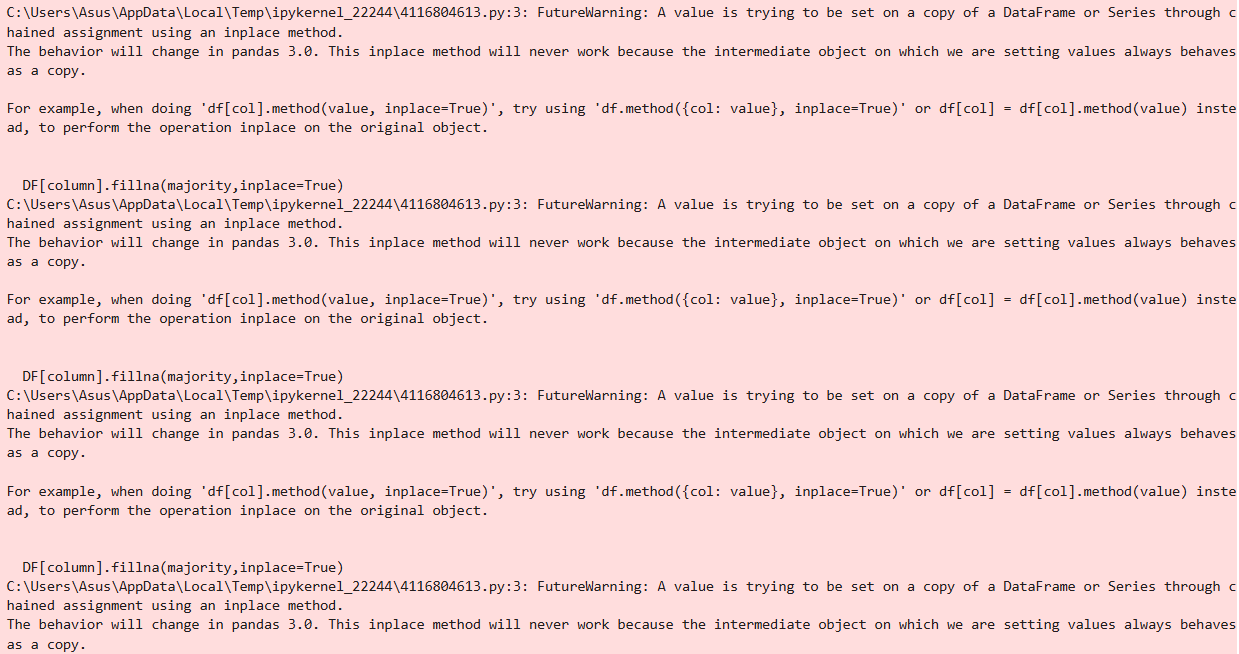
object\_columns=DF.select\_dtypes(include=['object']).columns

numeric\_columns=DF.select\_dtypes(exclude=['object']).columns

for column in object\_columns:

majority=DF[column].value\_counts().iloc[0]

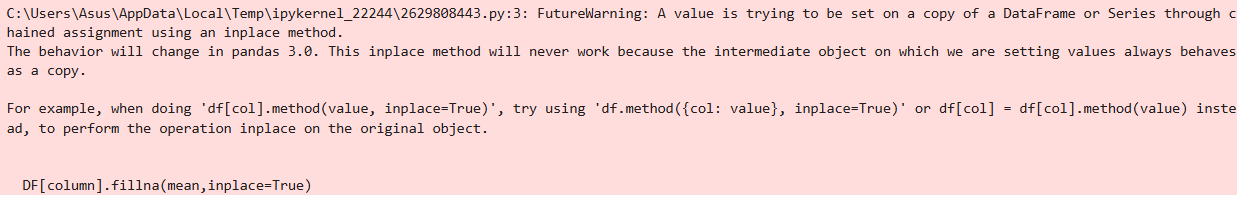
DF[column].fillna(majority,inplace=True)



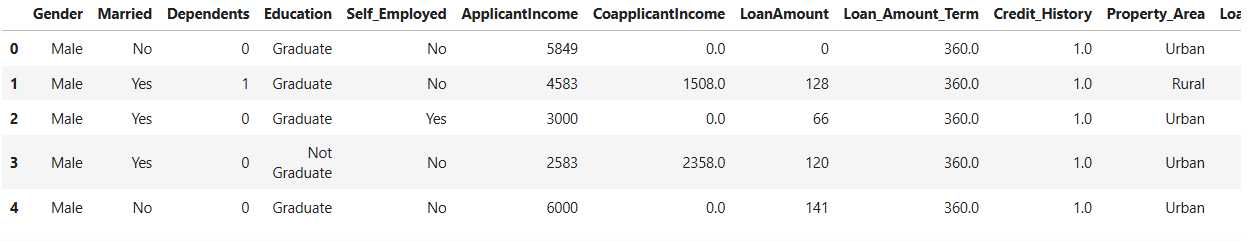
for column in numeric\_columns:

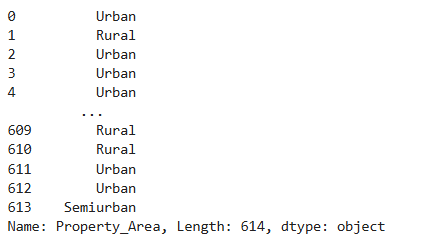
mean=DF[column].mean()

DF[column].fillna(mean,inplace=True)



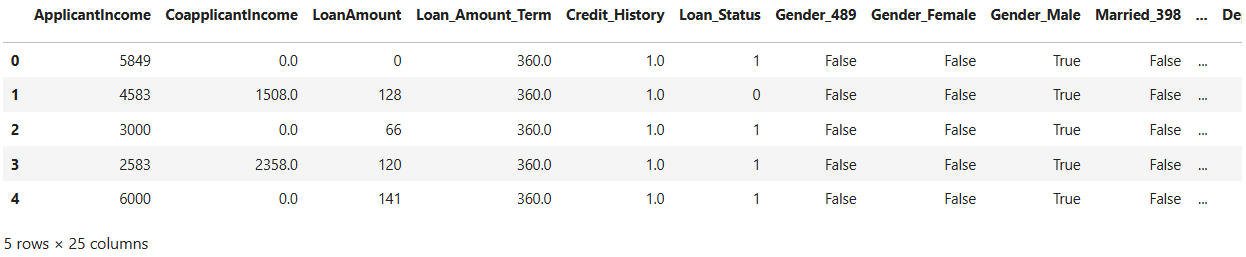
DF.head()



DF[object\_columns].Property\_Area 

DF\_dummy=pd.get\_dummies(DF,columns=object\_columns)

DF\_dummy.head()



DF\_dummy.shape



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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report

X=DF\_dummy.drop('Loan\_Status',axis=1)

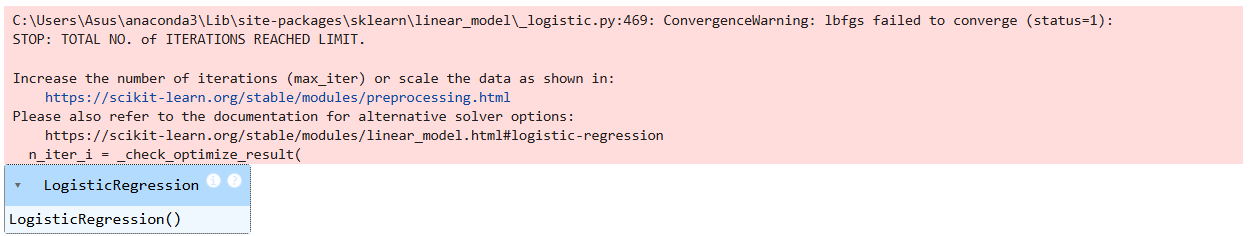
y=DF\_dummy.Loan\_Status

train\_X,test\_X,train\_y,test\_y=train\_test\_split(X,y,test\_size=0.3,random\_state=42) train\_X.shape,test\_X.shape



model=LogisticRegression()

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



train\_y\_hat=model.predict(train\_X)

test\_y\_hat=model.predict(test\_X)

print('train accuracy',accuracy\_score(train\_y,train\_y\_hat))

print('test accuracy',accuracy\_score(test\_y,test\_y\_hat))



| from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report  X=DF\_dummy.drop('Loan\_Status',axis=1)  y=DF\_dummy.Loan\_Status  train\_X,test\_X,train\_y,test\_y=train\_test\_split(X,y,test\_size=0.3,random\_state=42)  train\_X.shape,test\_X.shape    model=LogisticRegression()  model.fit(train\_X,train\_y)    train\_y\_hat=model.predict(train\_X)  test\_y\_hat=model.predict(test\_X)  print('train accuracy',accuracy\_score(train\_y,train\_y\_hat))  print('test accuracy',accuracy\_score(test\_y,test\_y\_hat))    print(confusion\_matrix(train\_y,train\_y\_hat))    print(confusion\_matrix(test\_y,test\_y\_hat))    test\_y.value\_counts()    pd.Series(test\_y\_hat).value\_counts()    # Accuracy for train  (57+295)/train\_y.shape[0]    print(classification\_report(test\_y,test\_y\_hat))    print("Mriganka Adhikary 69")  X=np.linspace(-10,10,100)  y=1/(1+np.exp(-X))# sigmoid  plt.plot(X,y)    test\_y\_hat\_5=(model.predict\_proba(test\_X)[:,1]>0.5).astype(int)  test\_y\_hat\_7=(model.predict\_proba(test\_X)[:,1]>0.7).astype(int)  test\_y\_hat\_3=(model.predict\_proba(test\_X)[:,1]>0.3).astype(int)  print(confusion\_matrix(test\_y,test\_y\_hat\_5))  print(confusion\_matrix(test\_y,test\_y\_hat\_7))  print(confusion\_matrix(test\_y,test\_y\_hat\_3))    print("Mriganka Adhikary 69")  print(classification\_report(test\_y,test\_y\_hat\_5))  print(classification\_report(test\_y,test\_y\_hat\_7))  print(classification\_report(test\_y,test\_y\_hat\_3)) |
| --- |

# 

Practical No: 13

**Aim: Implementation of Support Vector Machine – RBF Kernel**

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

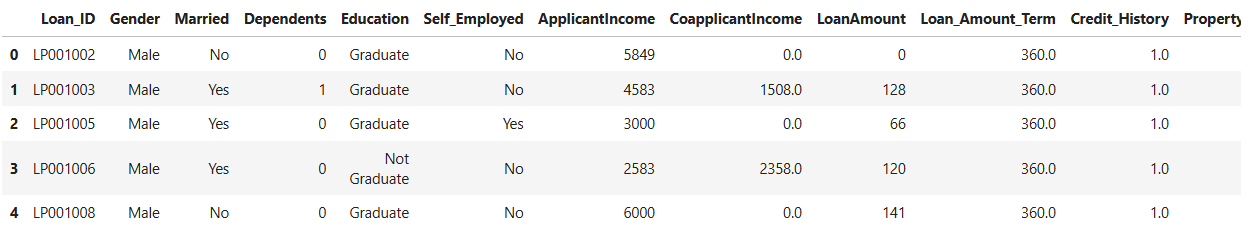
import seaborn as sns

credit\_df= pd.read\_csv(r"C:\Users\Asus\Mriganka Adhikary 69\CreditRisk.csv")

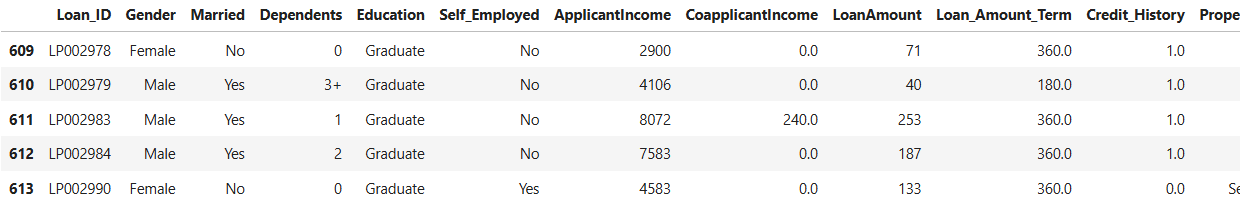
credit\_df.shape



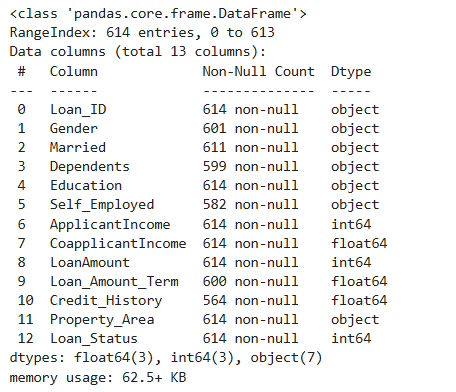
credit\_df.head()



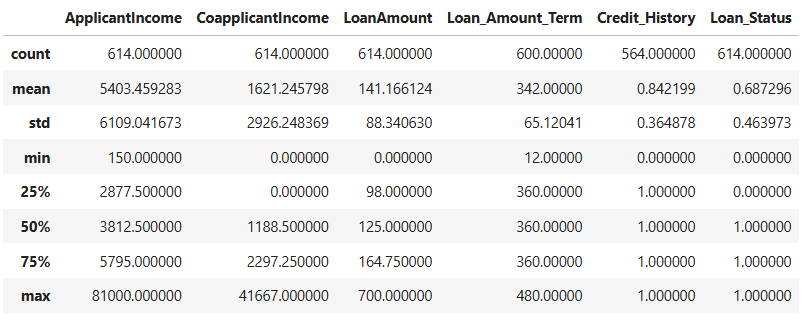
credit\_df.tail()



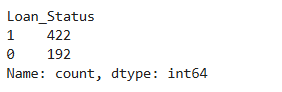
credit\_df.info()



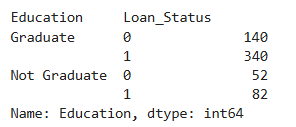
credit\_df.describe()



credit\_df.Loan\_Status.value\_counts()

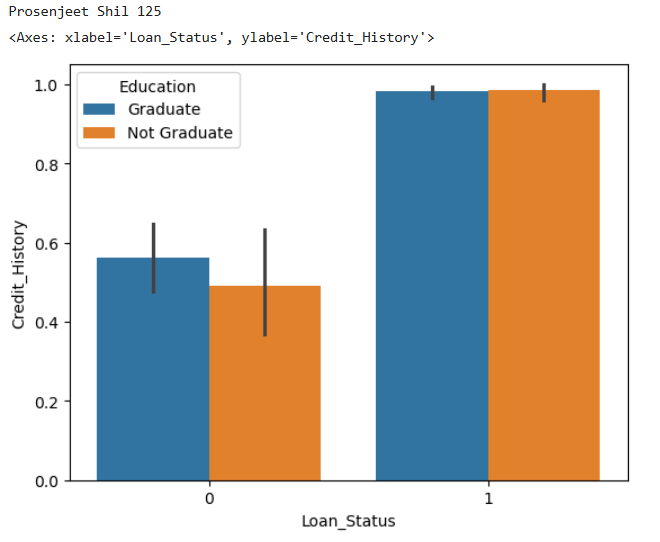


credit\_df.groupby(['Education','Loan\_Status']).Education.count()

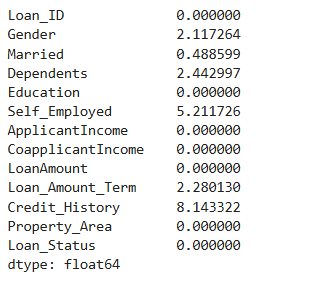


print("Mriganka Adhikary 69")

sns.barplot(y='Credit\_History',x='Loan\_Status',hue='Education',data=credit\_df)



100\*credit\_df.isnull().sum()/credit\_df.shape[0]



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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report

X=DF\_dummy.drop('Loan\_Status',axis=1)

y=DF\_dummy.Loan\_Status

train\_X,test\_X,train\_y,test\_y=train\_test\_split(X,y,test\_size=0.3,random\_state=42) train\_X.shape,test\_X.shape



#SVM Model

from sklearn.svm import SVC

svm\_model = SVC(kernel='rbf', gamma=0.00001, C=1000)

svm\_model.fit(train\_x, train\_y)



train\_y\_hat = svm\_model.predict(train\_x)

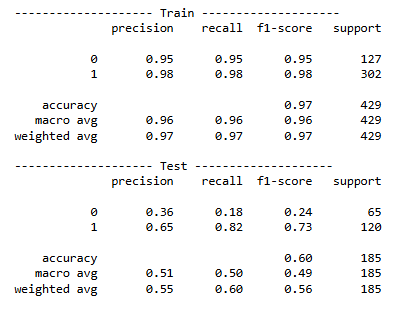
test\_y\_hat = svm\_model.predict(test\_x)

print('-'\*20, 'Train', '-'\*20)

print(classification\_report(train\_y, train\_y\_hat))

print('-'\*20, 'Test', '-'\*20)

print(classification\_report(test\_y, test\_y\_hat))



confusion\_matrix(test\_y, test\_y\_hat)



Practical No: 14

**Aim: Implementing Elbow method for choosing number of clusters**

**Code:**

| import pandas as pd  import matplotlib.pyplot as plt  from sklearn.cluster import KMeans  from sklearn.preprocessing import StandardScaler  df = pd.read\_csv('D:/76/driver-data.csv')  features = ['mean\_dist\_day', 'mean\_over\_speed\_perc']  X = df[features]  X.fillna(X.mean(), inplace=True)    scaler = StandardScaler()  X\_scaled = scaler.fit\_transform(X)  wcss = []  for i in range(1, 11): # Test clusters from 1 to 10  kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42,n\_init=10)  kmeans.fit(X\_scaled)  wcss.append(kmeans.inertia\_)  # Plot the Elbow method graph  print("Mriganka Adhikary 69")  plt.plot(range(1, 11), wcss)  plt.title('Elbow Method')  plt.xlabel('Number of Clusters')  plt.ylabel('WCSS') # Within-Cluster Sum of Squares  plt.show() |
| --- |

# 

Practical No: 15

**Aim: Ensemble Techniques - Implementing Bagging, boosting, stacking and Voting techniques**

**Code:**

from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier, StackingClassifier, VotingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.datasets import load\_iris

from sklearn.metrics import accuracy\_score

iris = load\_iris()

x, y = iris.data, iris.target

X.shape



Y.shape



x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

bagging\_model = BaggingClassifier(base\_estimator=DecisionTreeClassifier(), n\_estimators=50, random\_state=42)

bagging\_model.fit(x\_train, y\_train)

bagging\_predictions = bagging\_model.predict(x\_test)

bagging\_accuracy = accuracy\_score(y\_test, bagging\_predictions)

print("Bagging Accuracy: ", bagging\_accuracy)



boosting\_model = BaggingClassifier(base\_estimator = DecisionTreeClassifier(), n\_estimators=50, random\_state=42)

boosting\_model.fit(x\_train, y\_train)

boosting\_predictions = boosting\_model.predict(x\_test)

boosting\_accuracy = accuracy\_score(y\_test, boosting\_predictions)

print("Bagging Accuracy: ", bagging\_accuracy)



estimators = [

('dt', DecisionTreeClassifier()),

('lr', LogisticRegression()),

('knn', KNeighborsClassifier())

]

stacking\_model = StackingClassifier(estimators = estimators, final\_estimator = LogisticRegression())

stacking\_model.fit(x\_train, y\_train)

stacking\_predictions = stacking\_model.predict(x\_test)

stacking\_accuracy = accuracy\_score(y\_test, stacking\_predictions)

print("stacking Accuracy: ", stacking\_accuracy)



voting\_model = VotingClassifier(estimators = [

('dt', DecisionTreeClassifier()),

('lr', LogisticRegression()),

('knn', KNeighborsClassifier())

], voting='hard')

voting\_model.fit(x\_train, y\_train)

voting\_predictions = voting\_model.predict(x\_test)

voting\_accuracy = accuracy\_score(y\_test, voting\_predictions)

print("stacking Accuracy: ", voting\_accuracy)



voting\_model = VotingClassifier(estimators=[('rf', RandomForestClassifier())], voting='hard')

voting\_model.fit(x\_train, y\_train)

voting\_predictions = voting\_model.predict(x\_test)

voting\_accuracy= accuracy\_score(y\_test, voting\_predictions)

print("Voting Accuracy: ", voting\_accuracy)



Practical No: 16

**Aim: Implementing Bagging algorithm taking Random Forest as Base estimator**

**Code:**

import numpy as np

import pandas as pd

import matplotlib as mpl

import matplotlib.pyplot as plt

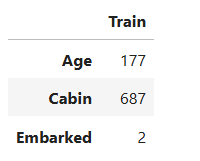
train = pd.read\_csv(r"C:\Users\Asus\Mriganka Adhikary 69\titanic.csv")

print(train.shape)



NAs = pd.concat([train.isnull().sum()], axis=1,keys=["Train"])

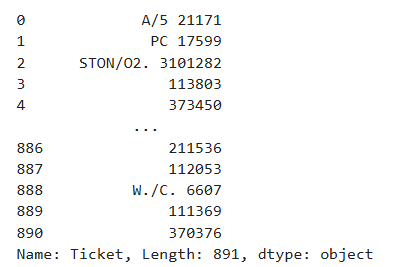
NAs[NAs.sum(axis=1)>0]



train.pop("Cabin")

train.pop("Name")

train.pop("Ticket")



Train.shape



train["Age"] =train["Age"].fillna(train["Age"].mean());

train["Embarked"]= train["Embarked"].fillna(train["Embarked"].mode()[0]);

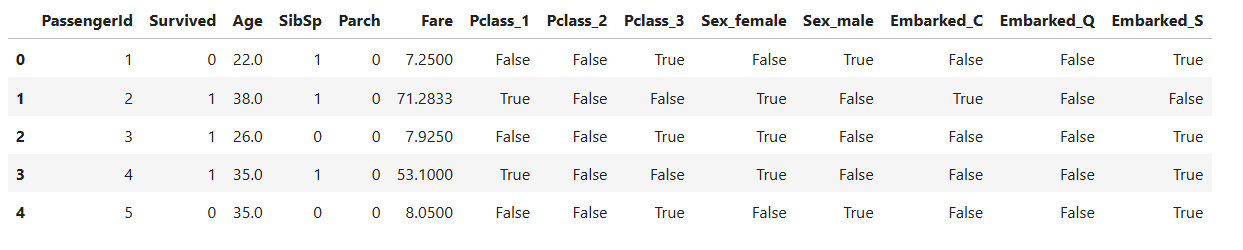
train["Pclass"] = train["Pclass"].apply(str)

for col in train.dtypes[train.dtypes=="object"].index:

for\_dummy=train.pop(col)

train= pd.concat([train,pd.get\_dummies(for\_dummy,prefix=col)],axis=1)

train.head()



labels= train.pop("Survived")

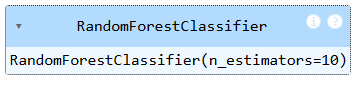
from sklearn.model\_selection import train\_test\_split

X\_train, X\_test,Y\_train,Y\_test = train\_test\_split(train,labels,test\_size=0.25)

from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(n\_estimators=10)

rf.fit(X\_train,Y\_train)



Y\_pred=rf.predict(X\_test)

from sklearn.metrics import accuracy\_score

accuracy = accuracy\_score(Y\_test,Y\_pred)

accuracy



from sklearn.ensemble import BaggingClassifier

bagging\_model = BaggingClassifier(estimator = RandomForestClassifier(),n\_estimators=10)

bagging\_model.fit(X\_train,Y\_train)

bagging\_predictions=bagging\_model.predict(X\_test)

bagging\_accuracy = accuracy\_score(Y\_test,bagging\_predictions)

print("Bagging Accuracy:" , bagging\_accuracy)



Practical No: 17

**Aim: Implementing Adaboost Algorithm**

**Code:**

import pandas as pd

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

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

data = pd.read\_csv('C:/Users/Asus/Mriganka Adhikary 69/titanic.csv')

data.shape

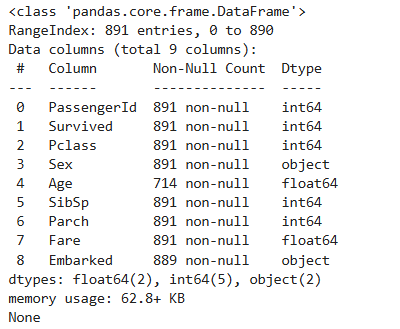


data.pop("Cabin")

data.pop("Name")

data.pop("Ticket")

print(data.info())



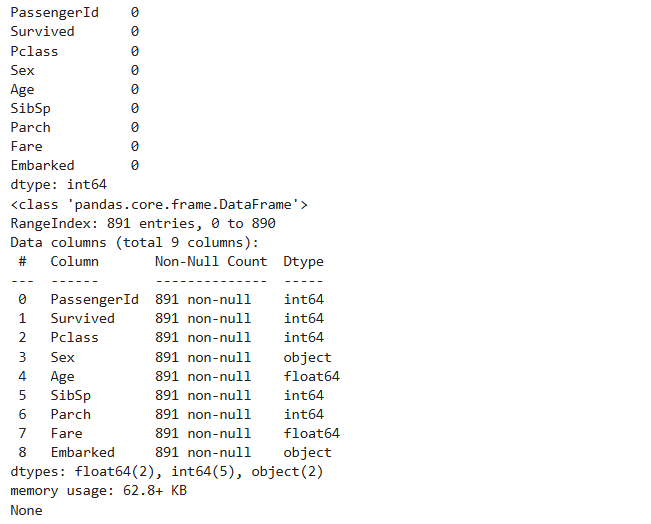
# filling missing values

data["Age"]=data["Age"].fillna(data["Age"].mean())

data["Embarked"]=data["Embarked"].fillna(data["Embarked"].mode()[0])

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

print(data.info())



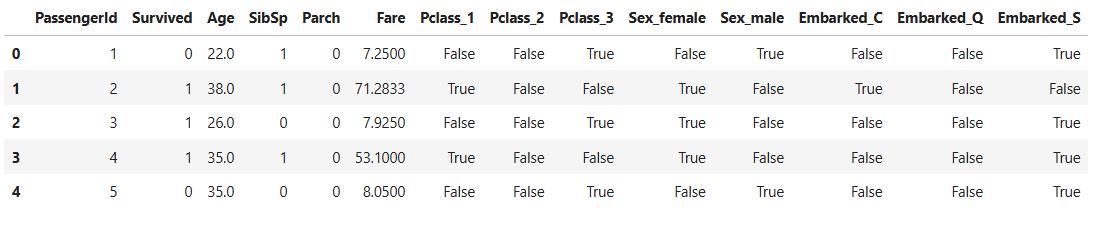
data["Pclass"]=data["Pclass"].apply(str)

for col in data.dtypes[data.dtypes=="object"].index :

for\_dummy = data.pop(col)

data = pd.concat([data,pd.get\_dummies(for\_dummy, prefix=col)], axis=1)

data.head()



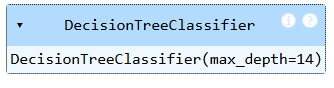
labels = data.pop("Survived")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data,labels,test\_size=0.25)

#Model Training

data\_model = DecisionTreeClassifier(max\_depth=14)

data\_model.fit(X\_train, y\_train)



train\_y\_hat = data\_model.predict(X\_train)

test\_y\_hat = data\_model.predict(X\_test)

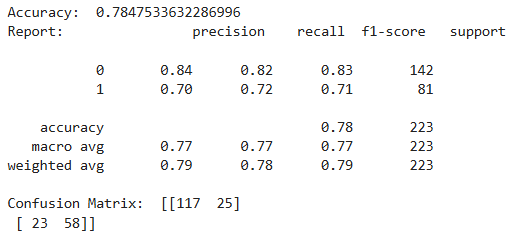
y\_pred = data\_model.predict(X\_test)

#Evaluating the Model

print("Accuracy: ",accuracy\_score(y\_test,y\_pred))

print("Report: ",classification\_report(y\_test,y\_pred))

print("Confusion Matrix: ",confusion\_matrix(y\_test,y\_pred))



# AdaBoost Method

boosting\_model = AdaBoostClassifier(estimator=DecisionTreeClassifier(), n\_estimators=50, random\_state=42)

boosting\_model.fit(X\_train, y\_train)

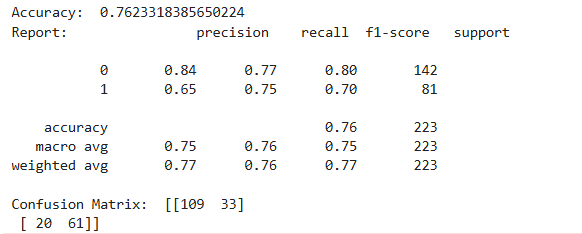
boosting\_pred = boosting\_model.predict(X\_test)

#Evaluating the Model

print("Accuracy: ",accuracy\_score(y\_test,boosting\_pred))

print("Report: ",classification\_report(y\_test,boosting\_pred))

print("Confusion Matrix: ",confusion\_matrix(y\_test,boosting\_pred))



Practical No: 18

**Aim: Implementation of Gradient Boosting Algorithm**

**Code:**

import numpy as np

import pandas as pd

import matplotlib as mpl

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.ensemble import GradientBoostingClassifier

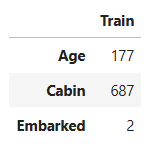
train = pd.read\_csv('C:/Users/Asus/Mriganka Adhikary 69/titanic.csv')

print(train.shape)



NAs = pd.concat([train.isnull().sum()], axis=1, keys=["Train"])

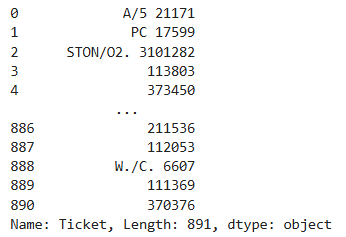
NAs[NAs.sum(axis=1) > 0]



train.pop("Cabin")

train.pop("Name")

train.pop("Ticket")



train.shape



train["Age"] = train["Age"].fillna(train["Age"].mean())

train["Embarked"] = train["Embarked"].fillna(train["Embarked"].mode()[0])

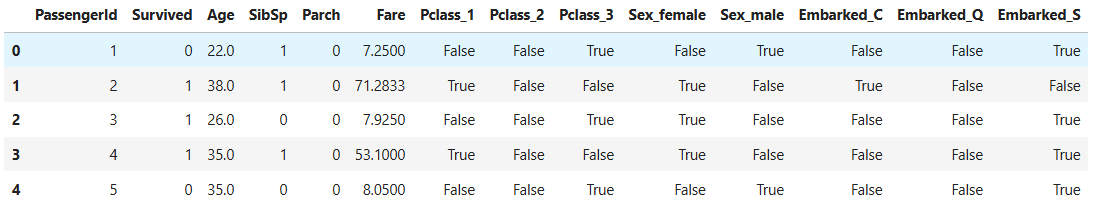
train["Pclass"] = train["Pclass"].apply(str)

for col in train.dtypes[train.dtypes=="object"].index:

for\_dummy=train.pop(col)

train=pd.concat([train, pd.get\_dummies(for\_dummy, prefix=col)], axis=1)

train.head()



labels = train.pop("Survived")

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(train, labels, test\_size = 0.25)

gb\_classifier = GradientBoostingClassifier(

n\_estimators = 100,

learning\_rate = 0.1,

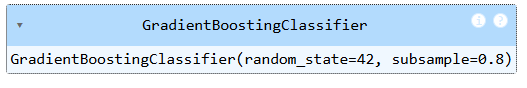
max\_depth = 3,

subsample = 0.8,

random\_state = 42

)

gb\_classifier.fit(x\_train,y\_train)



y\_pred=gb\_classifier.predict(x\_test)

accuracy=accuracy\_score(y\_test,y\_pred)

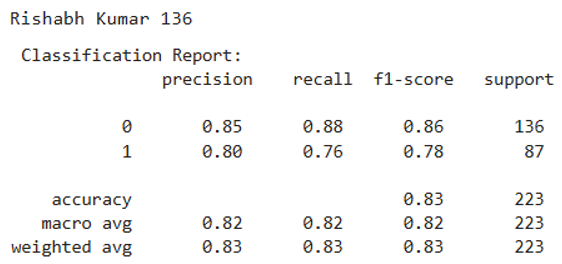
accuracy



from sklearn.metrics import classification\_report

print("Mriganka Adhikary 69")

print("\n Classification Report: \n",classification\_report(y\_test, y\_pred))



print("Confusion Matrix: ",confusion\_matrix(y\_test,y\_pred))

