**Practical No 1**

**Aim: Implementation of Logic programming using Prolog DFS for water jug problems.**

**Code:**

water\_jug(X,Y):- X>4,Y<3,write('4L jug overflow.'),nl.

water\_jug(X,Y):- X<4,Y>3,write('3L jug overflow.'),nl.

water\_jug(X,Y):- X>4,Y>3,write('Both jugs overflow.'),nl.

water\_jug(X,Y):- (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 State 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

untill 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: Fill 4L jug.)'),XX is 4,

water\_jug(XX,Y));

(X=:=4, Y=:=1,nl,write('4L:2 & 3L:3 (Action: Pour water from 4L to 3L jug untill 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));

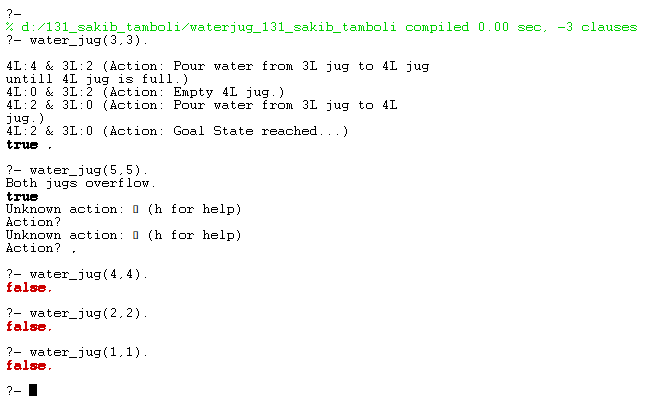
(X=:=4, Y=:=2,nl,write('4L:0 & 3L:2 (Action: Empty 4L jug.)'),XX is 0,

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)).

**Output:**



**Practical No 2**

**Aim: Implementation of Logic programing using PROLOG BFS for tic-tac toe problem**

**Code:**

Tic TacToe

% 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)

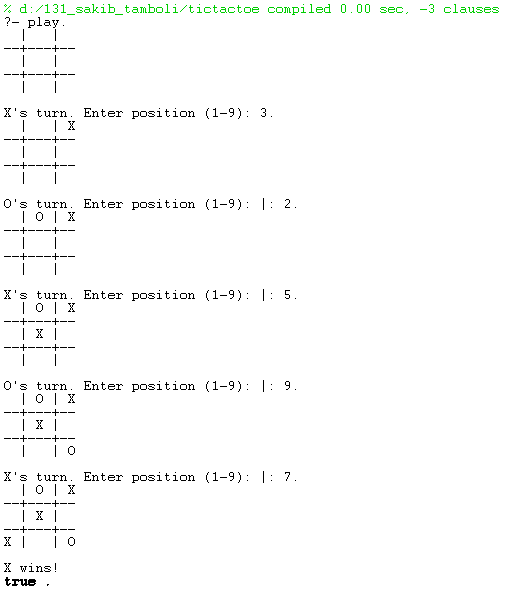
).

% Switch player

switch('X', 'O').

switch('O', 'X').

**Output:**

****

**Practical No 3**

**Aim: Implementation of Logic programming using PROLOG Hill-climbing to solve 8-puzzle problem**

**Code:**

Prolog 8puzzle

% Simple Prolog Planner for the 8 Puzzle Problem

/\* This predicate initialises the problem states. The first argument of solve is the initial state, the 2nd the goal state, and the third the plan that will be produced.\*/

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).

solve(State, Goal, Plan):-

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

% Determines whether Current and Destination tiles are a valid move.

is\_movable(X1,Y1) :- (1 is X1 - Y1) ; (-1 is X1 - Y1) ; (3 is X1 - Y1) ; (-3 is X1 - Y1).

/\* This predicate produces the plan. Once the Goal list is a subset of the current State the plan is complete and it is written to the screen using write\_sol \*/

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

is\_subset(Goal, State), nl,

write\_sol(Plan).

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

act(Action, Preconditions, Delete, Add),

is\_subset(Preconditions, State),

\+ member(Action, Sofar),

delete\_list(Delete, State, Remainder),

append(Add, Remainder, NewState),

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

/\* The problem has three operators.

1st arg = name

2nd arg = preconditions

3rd arg = delete list

4th arg = add list. \*/

% Tile can move to new position only if the destination tile is empty & Manhattan distance = 1

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)]).

% Utility predicates.

% Check is first list is a subset of the second

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

member(H, Set),

is\_subset(T, Set).

is\_subset([], \_).

% Remove all elements of 1st list from second to create third.

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).

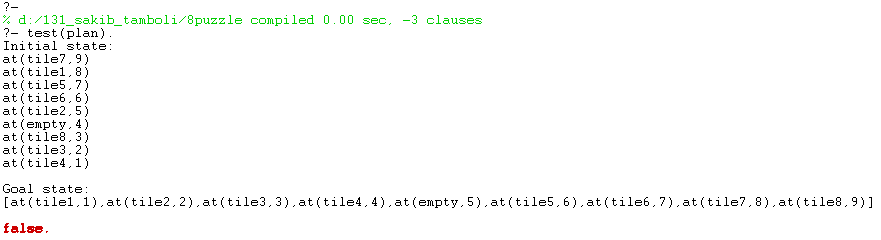
append([], L, L).

member(X, [X|\_]).

member(X, [\_|T]):-

member(X, T).

**Output:**

****

**Practical No 4**

**Aim: Introduction to python libraries - basic python libraries numpy, pandas**

# NUMPY

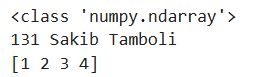
import numpy as np

x=np.array([1,2,3,4])

print("131 Sakib Tamboli")

print(type(x))

print(x)



x=np.array([1,2,'n',4])

x



x=np.array([1,2,'name',4])

x



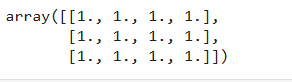
*#Generating array using arange*

d=np.arange(1,11,2)

d



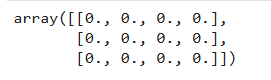
np**.**ones((3,4))



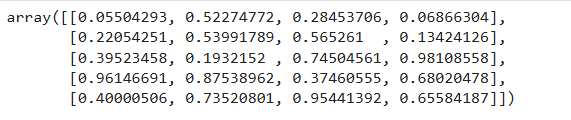
np**.**random**.**rand(4)



np**.**zeros((3,4))



np**.**random**.**rand(5,4)

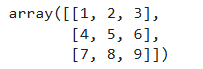


np.logspace(1,10,num=5,endpoint=True,base=10.0)



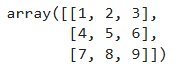
grid=np.arange(start=1,stop=10).reshape(3,3)

grid



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

mat

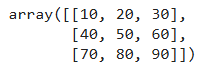


mat**.**shape

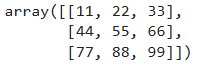


mat2**=**np**.**array([[10,20,30],[40,50,60],[70,80,90]])

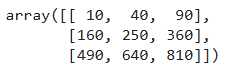
mat2



np**.**add(mat,mat2)



np**.**multiply(mat,mat2)



mat[1,2]



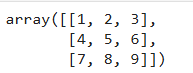
mat[1:2]



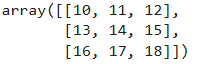
arr1**=**np**.**array([[1,2,3],[4,5,6],[7,8,9]])

arr2**=**np**.**arange(10,19,1)**.**reshape(3,3)

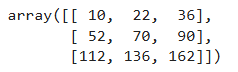
arr1



arr2



np**.**multiply(arr1,arr2)



arr2[:,0]



arr2[0,:]



arr1\_sub**=**arr1[:2,:2]

arr1\_sub

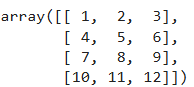


arr1



a\_row**=**np**.**append(arr1,[[10,11,12]],axis**=**0)

a\_row

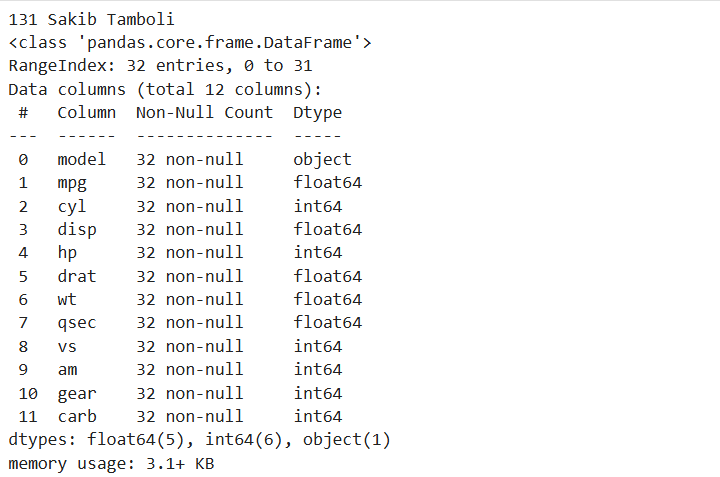


**#PANDAS**

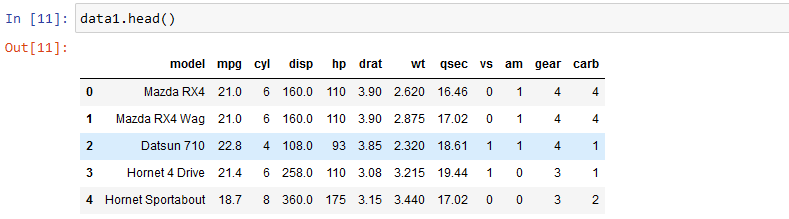
import pandas as pd

**data1=pd.read\_csv('D:/131\_Sakib\_Tamboli/mtcars.csv')**

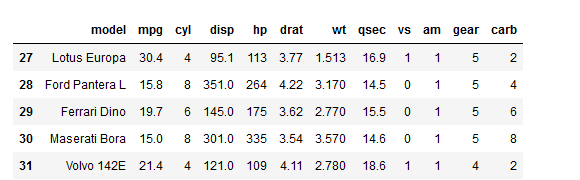
print("131 Sakib Tamboli")

data1.info()

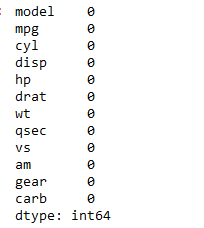
data1.head()



data1.tail()



data1.isnull().sum()



**d**ata1.isnull()



Data1.size

****

Data1.shape

****

Data1.ndim

****

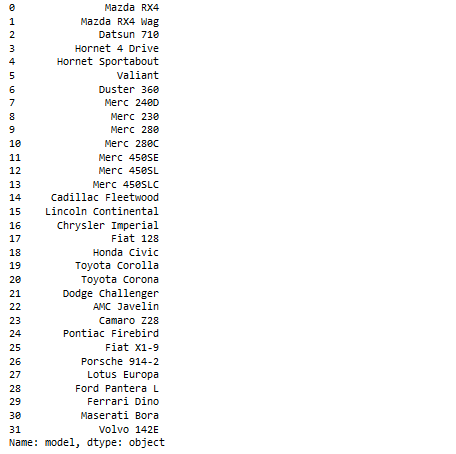
data1.at[4,'model']

****

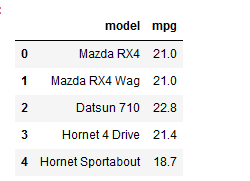
Data1.iat[4,3]

****

data1.loc[:,'model']

**\**

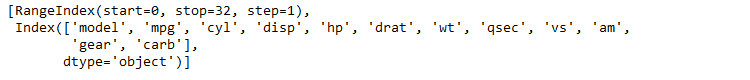
data1.iloc[0:5,0:2]

****

Data1['model'].dtype

****

Data1.axes

****

data1.columns



data1['hp'].std()



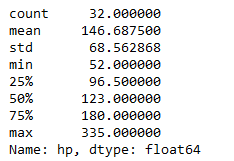
data1['mpg'].mean()



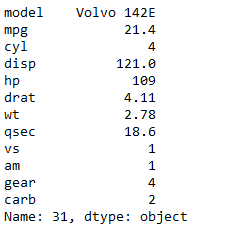
data1['mpg'].median()



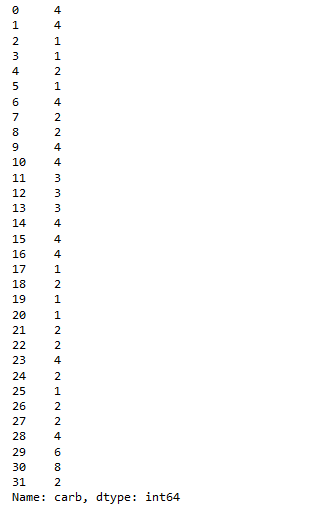
data1['hp'].describe()



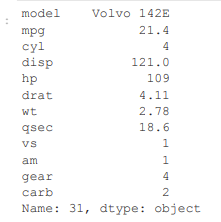
Data1.iloc[-1]



Data1.iloc[:,-1]

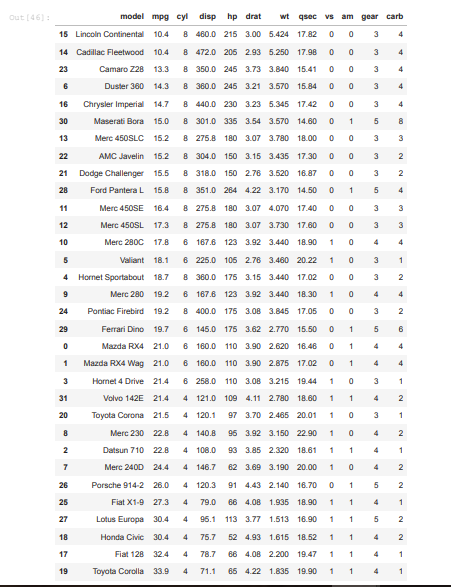


data1.iloc[-1]

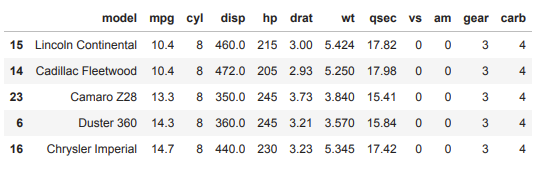


data1\_sorted=data1.sort\_values(by='mpg')

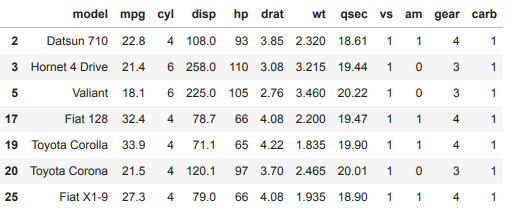
data1\_sorted

****

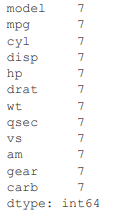
data1\_sorted.head()

****

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

****

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

****

**Practical 5**

**Aim: Introduction to python libraries - basic python libraries matplotlib, scipy**

**#Matplotlib**

import matplotlib.pyplot as plt

x=[3,1,3]

y=[3,2,1]

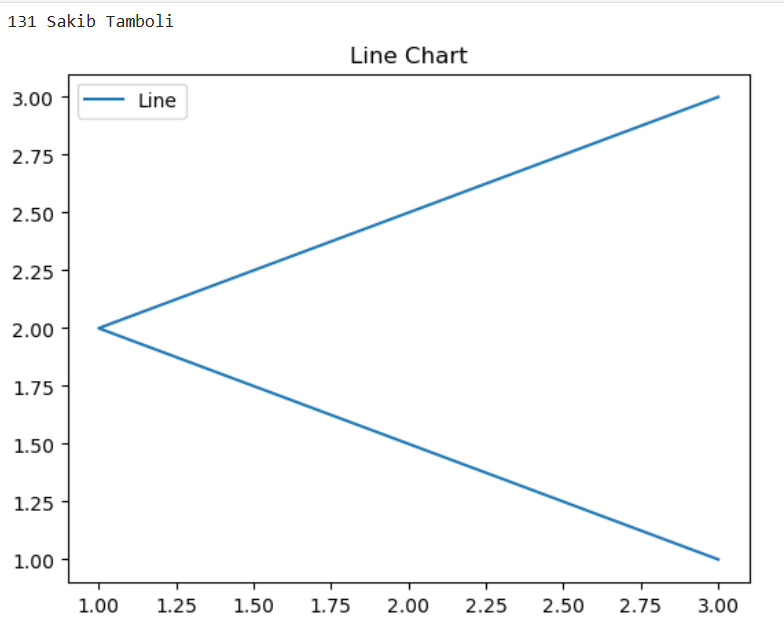
plt.plot(x,y)

plt.title("Line Chart")

plt.legend(["Line"])

Plt.show

print("131 Sakib Tamboli")



x=[3,1,3,12,2,5,7]

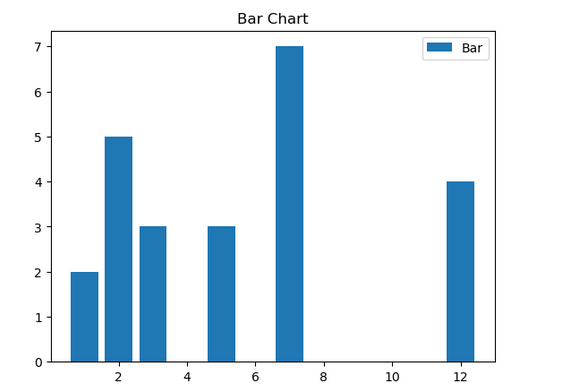
y=[3,2,1,4,5,3,7]

plt.bar(x,y)

plt.title("Bar Chart")

plt.legend(["Bar"])

plt.show()



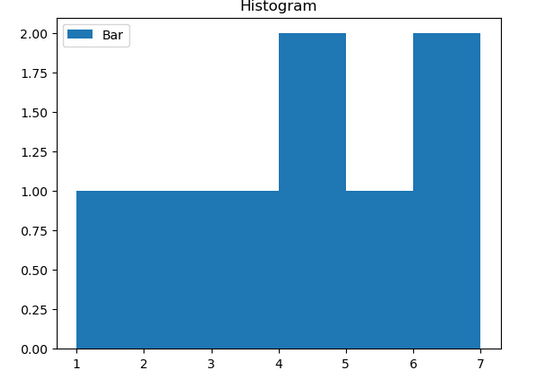
x=[1,2,3,4,5,6,7,4]

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

plt.title("Histogram")

plt.legend(["Bar"])

plt.show()



x=[3,1,3,12,2,4,4]

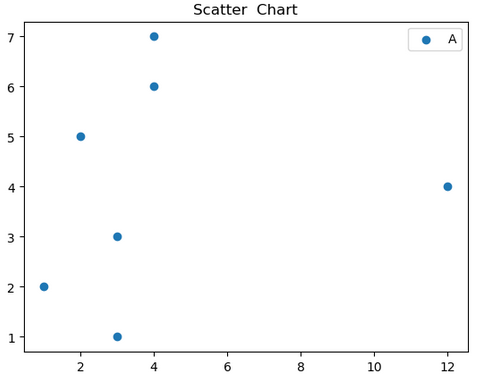
y=[3,2,1,4,5,6,7]

plt.scatter(x,y)

plt.title("Scatter Chart")

plt.legend(["A"])

plt.show()



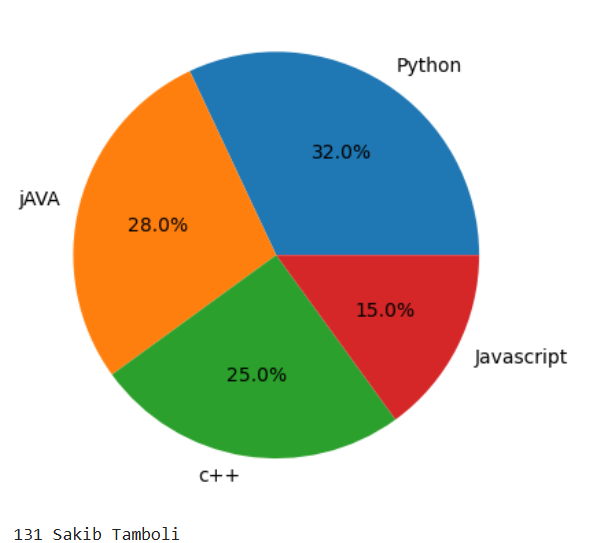
labels=['Python','jAVA','c++','Javascript']

sizes=[40,30,20,10]

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

plt.show()

print("131 Sakib Tamboli")



**#SCIPY**

import numpy as np

from scipy import integrate, optimize, stats, linalg

print("131 Sakib Tamboli")

# Example 1: Numerical Integration

# Integrate the function f(x) = x^2 from 0 to 2

result, error = integrate.quad(lambda x: x\*\*2, 0, 2)

print("Integral of x^2 from 0 to 2:", result)

# Example 2: Optimization (Finding Minimum)

# Minimize the function f(x) = (x - 3)^2

min\_result = optimize.minimize(lambda x: (x - 3)\*\*2, x0=0)

print("Minimum found at x =", min\_result.x[0])

# Example 3: Statistics - Mean and Standard Deviation of a sample

data = np.array([2, 5, 8, 9, 4, 7])

mean = np.mean(data)

std\_dev = np.std(data)

print("Mean:", mean)

print("Standard Deviation:", std\_dev)

# Example 4: Linear Algebra - Solve a system of equations

# 2x + 3y = 8 and 3x + 4y = 11

A = np.array([[2, 3], [3, 4]])

b = np.array([8, 11])

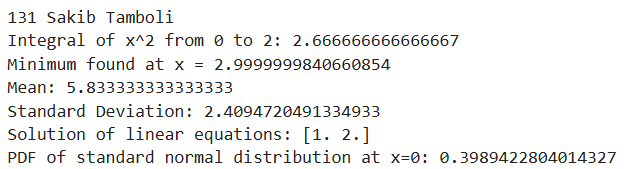
x = linalg.solve(A, b)

print("Solution of linear equations:", x)

# Example 5: Probability - PDF of Normal Distribution

pdf\_val = stats.norm.pdf(0, loc=0, scale=1) # Standard normal at x=0

print("PDF of standard normal distribution at x=0:", pdf\_val)

****

**Practical 6**

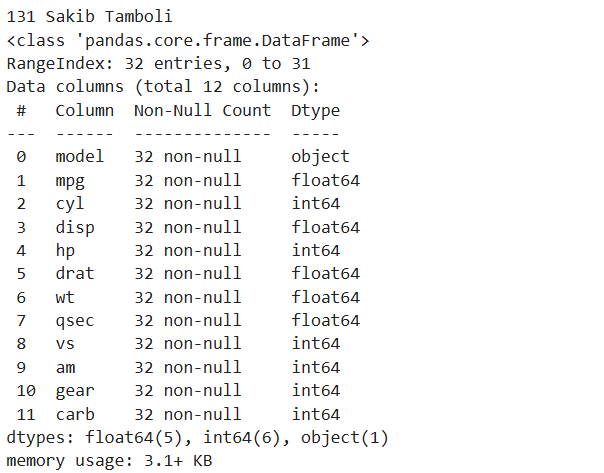
**Aim: Exploratory Data Analysis Using python.**

import pandas as pd

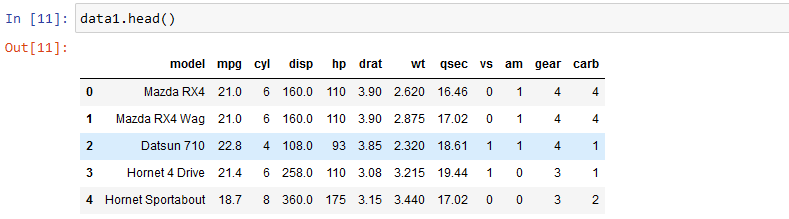
**data1=pd.read\_csv('D:/131\_Sakib\_Tamboli/mtcars.csv')**

print("131 Sakib Tamboli")

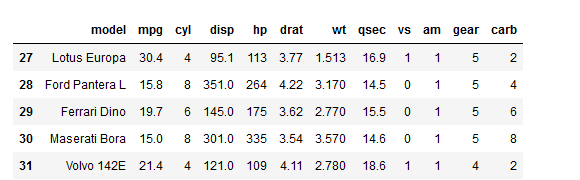
data1.info()



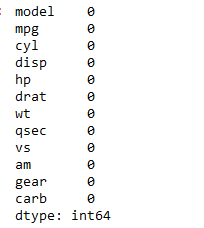
data1.head()



data1.tail()



data1.isnull().sum()



**d**ata1.isnull()



Data1.size

****

Data1.shape

****

Data1.ndim

****

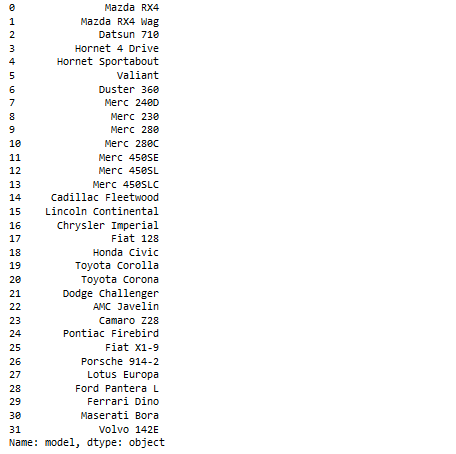
data1.at[4,'model']

****

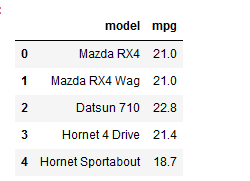
Data1.iat[4,3]

****

data1.loc[:,'model']

**\**

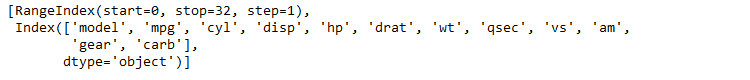
data1.iloc[0:5,0:2]

****

Data1['model'].dtype

****

Data1.axes

****

data1.columns



data1['hp'].std()



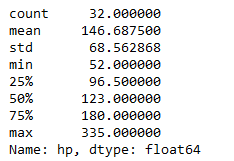
data1['mpg'].mean()



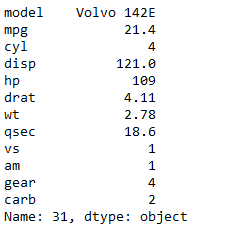
data1['mpg'].median()



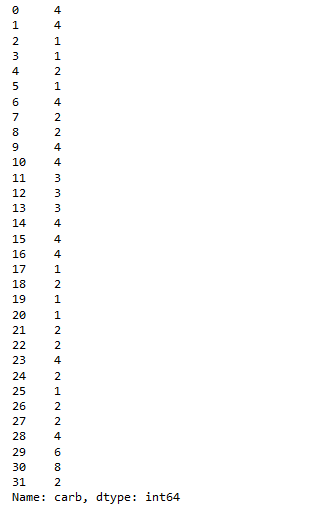
data1['hp'].describe()



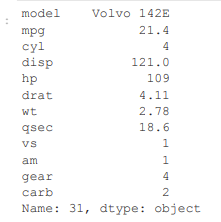
Data1.iloc[-1]



Data1.iloc[:,-1]

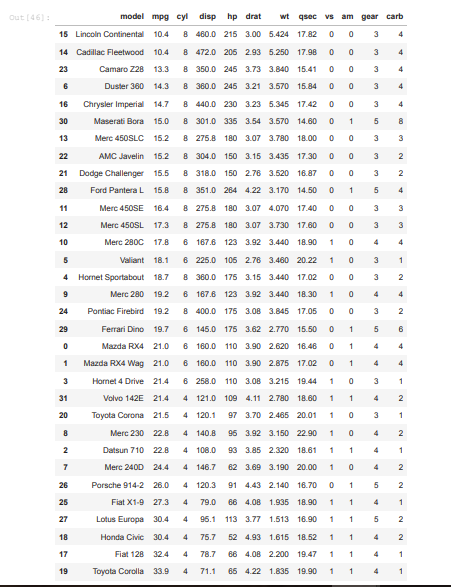


data1.iloc[-1]

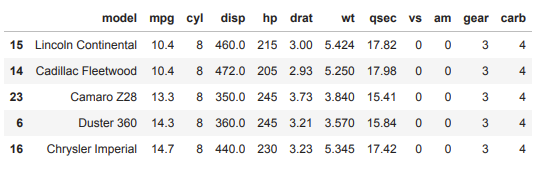


data1\_sorted=data1.sort\_values(by='mpg')

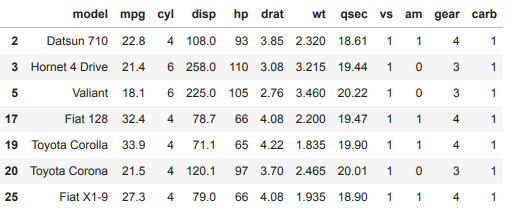
data1\_sorted

****

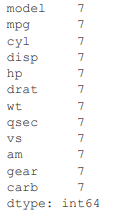
data1\_sorted.head()

****

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

****

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

****

**Practical 7**

**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(perceptron\_or(0,0))

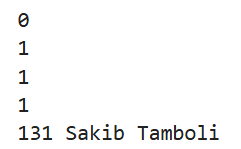
print(perceptron\_or(0,1))

print(perceptron\_or(1,0))

print(perceptron\_or(1,1))

print("131 Sakib Tamboli")

**Output:**

****

**Practical 8**

**Aim: Implementation of Adaline algorithm 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])

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

ada.fit(X, y)

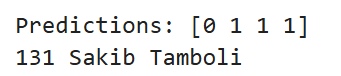
# Make predictions

predictions = ada.predict(X)

print("Predictions:", predictions)

print("131 Sakib Tamboli")

**Output:**

****

**Practical 9**

**Aim: Implementation of gradient Descent Algorithm.**

**Code:**

print("131 Sakib Tamboli")

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]

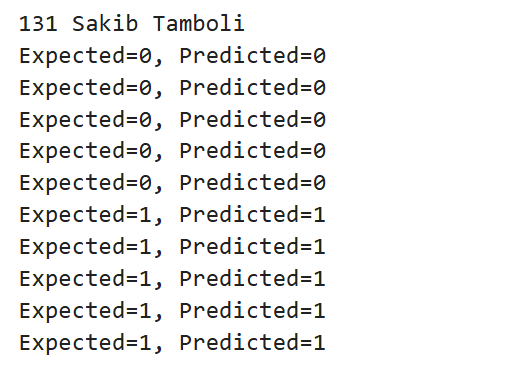
]

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

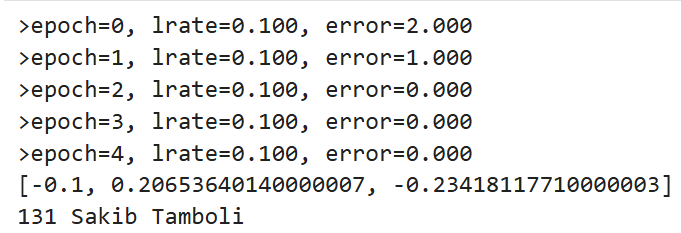
n\_epoch = 5 # Number of epochs for training

# Train weights using the dataset

weights = train\_weights(dataset, l\_rate, n\_epoch)

print(weights) # Print the trained weights

print("131 Sakib Tamboli")



**Practical 10**

**Aim: Implementation of principal component analysis.**

**PCA**

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

print("131 Sakib Tamboli")

# 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

print("131 Sakib Tamboli")

# Apply PCA with 2 components

pca = PCA(n\_components=2)

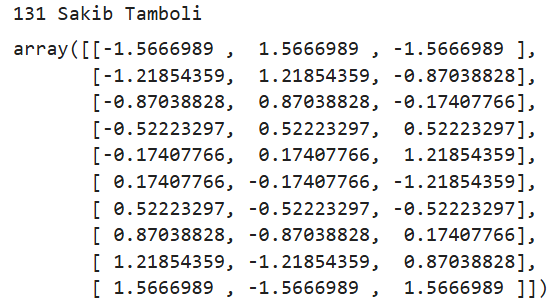
principal\_components = pca.fit\_transform(x)

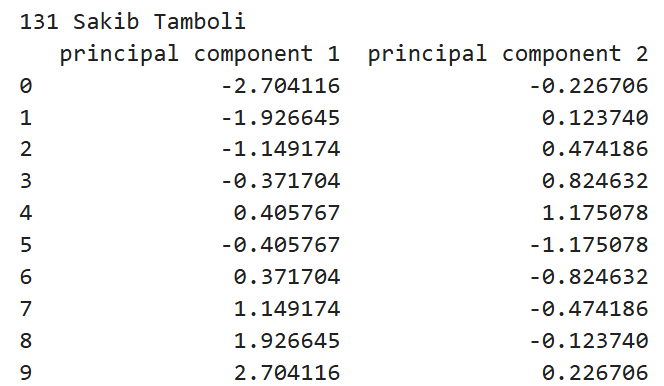
# Create a new DataFrame with the principal components

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'])

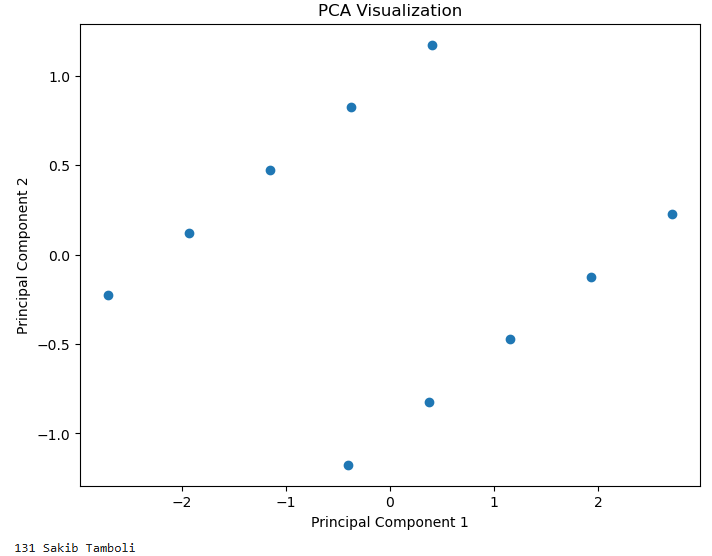
plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('PCA Visualization')

plt.show()

print("131 Sakib Tamboli")



# 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 11**

**Aim: Implementation of Normalization and transformation.**

**Normalization**

**Code:**

import pandas as pd

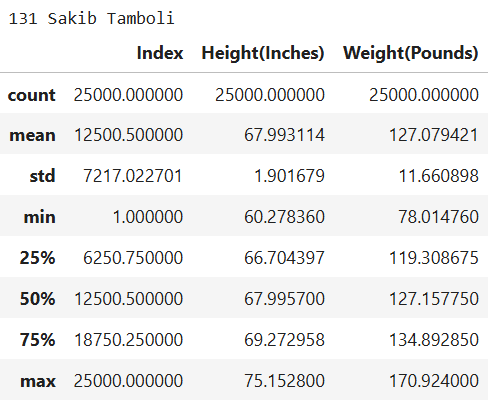
import numpy as np

import matplotlib.pyplot as plt

print("131 Sakib Tamboli")

hw\_df=pd.read\_csv("D:/131\_Sakib\_Tamboli/H-W-Index.csv")

hw\_df.describe()

****

from sklearn.preprocessing import minmax\_scale

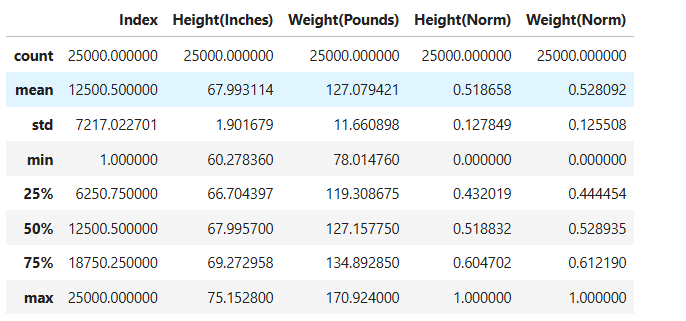
import pandas as pd

hw\_scaled=minmax\_scale(hw\_df[['Height(Inches)','Weight(Pounds)']],feature\_range=(0,1))

hw\_df['Height(Norm)']=hw\_scaled[:,0]

hw\_df['Weight(Norm)']=hw\_scaled[:,1]

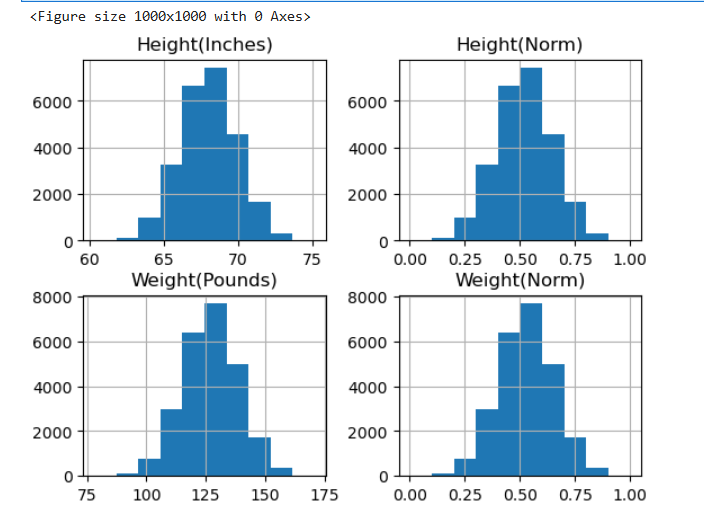
hw\_df.describe()

****

plt.figure(figsize=(10,10))

hw\_df[['Height(Inches)','Height(Norm)','Weight(Pounds)','Weight(Norm)']].hist()

plt.show()

****

**Transformation**

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'],

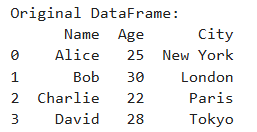
'Age': [25, 30, 22, 28],

'City': ['New York', 'London', 'Paris', 'Tokyo']}

df = pd.DataFrame(data)

print("Original DataFrame:")

print(df)

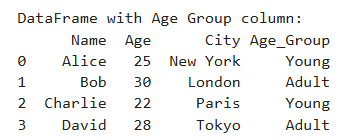


1. #Adding a new column

df['Age\_Group'] = pd.cut(df['Age'], bins=[18, 25, 30, 100], labels=['Young', 'Adult', 'Senior'])

print("\nDataFrame with Age Group column:")

print(df)

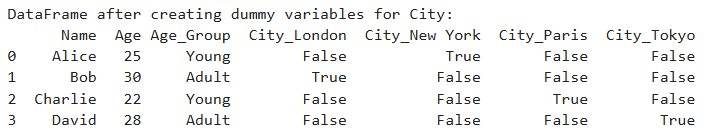


# 2. Creating dummy variables for categorical features

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:/131\_Sakib\_Tamboli/CreditRisk.csv')**

print("DataFrame head:")

df.head()

# Example feature extraction:

# 1. Calculate the mean of a numerical column

# if 'numerical\_column' in df.columns:

# mean\_value = df['numerical\_column'].mean()

# print(f"\nMean of 'numerical\_column': {mean\_value}")

2.# Correlation method

# 3. One-hot encode categorical features

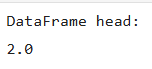
# categorical\_cols = ['categorical\_column\_1', 'categorical\_column\_2']

# if all(col in df.columns for col in categorical\_cols):

# df = pd.get\_dummies(df, columns=categorical\_cols)

# print("\nDataFrame after one-hot encoding:")

# print(df.head())

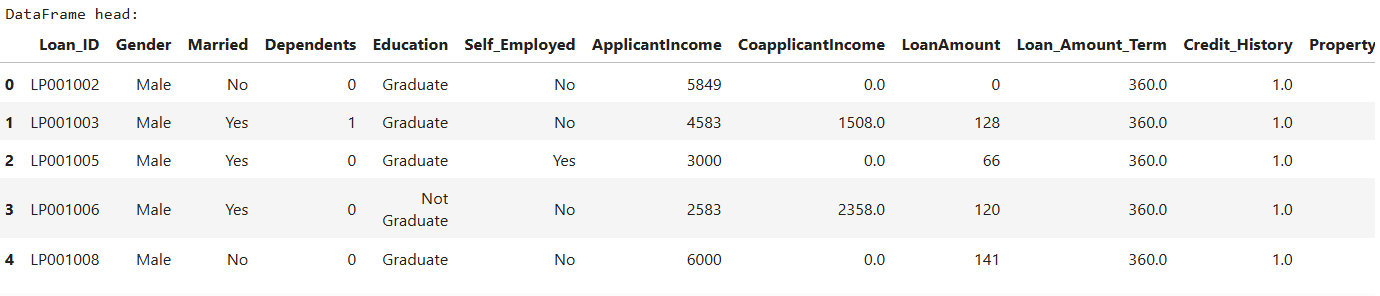


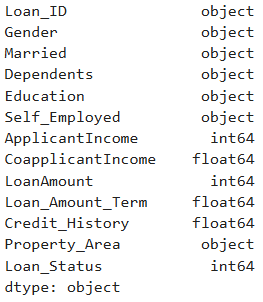
import pandas as pd

**df = pd.read\_csv(''D:/131\_Sakib\_Tamboli/CreditRisk.csv')**

print("DataFrame head:")

df.head()



Df.dtypes  


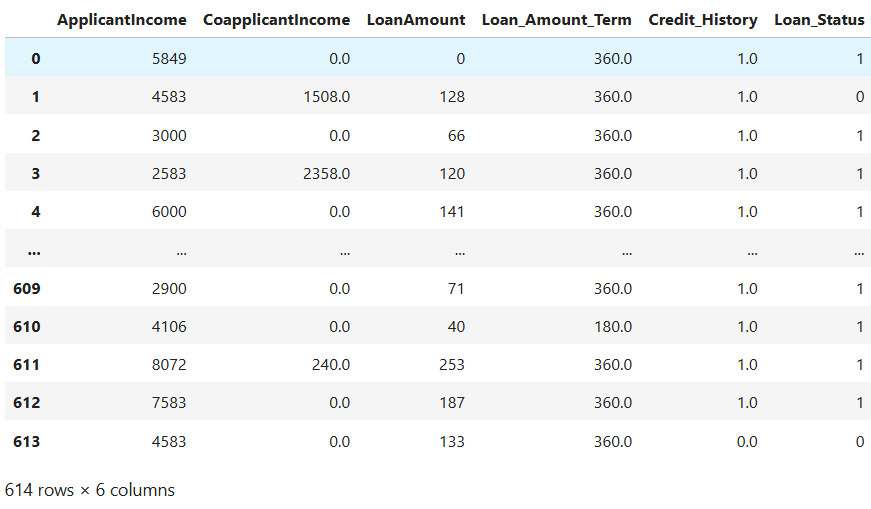
#Example feature extraction:

# 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

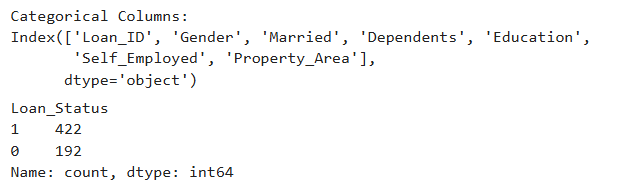
print("Categorical Columns:")

print(obj\_cols)

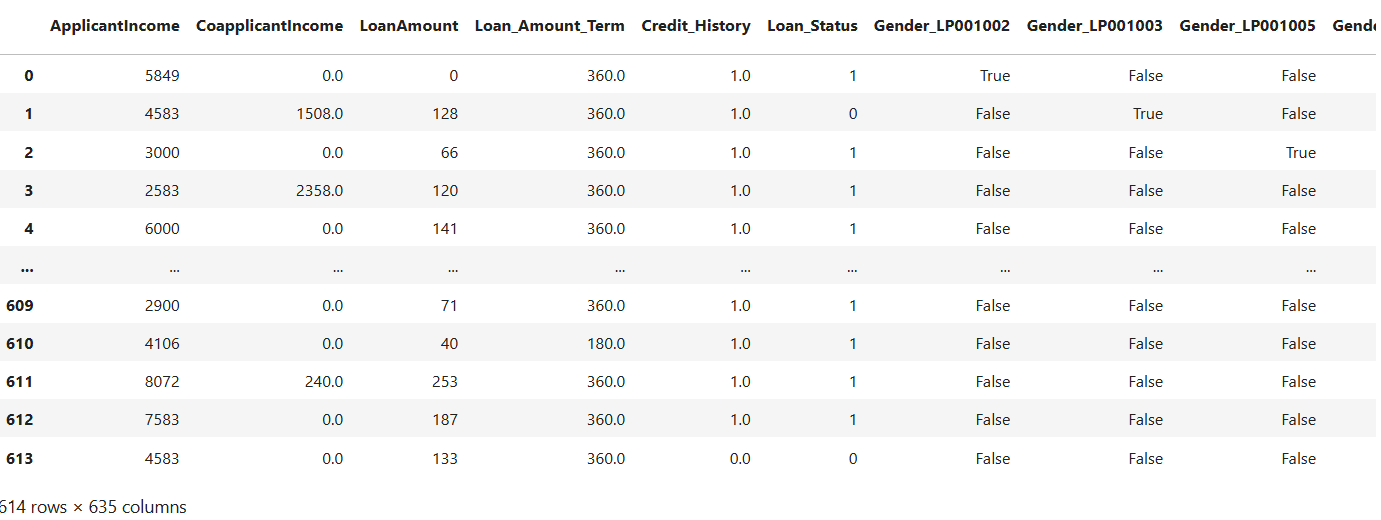
X = df.drop('Loan\_Status', axis=1)

y =df['Loan\_Status']

y.value\_counts()

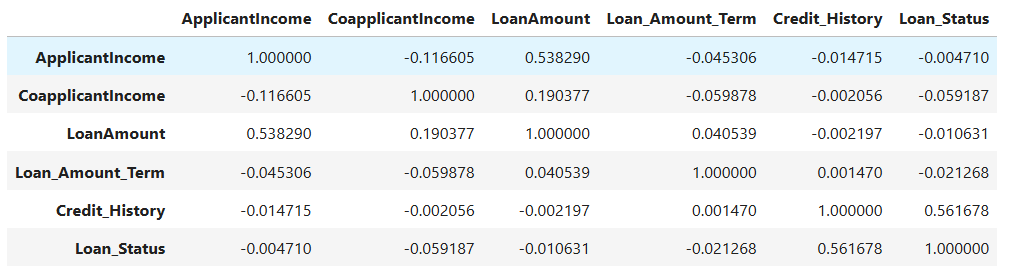


pd.get\_dummies(df,'Gender')



correlation\_matrix =num\_cols.corr()

correlation\_matrix



**Practical 12**

**Aim: Implementation of Logistic Regression.**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

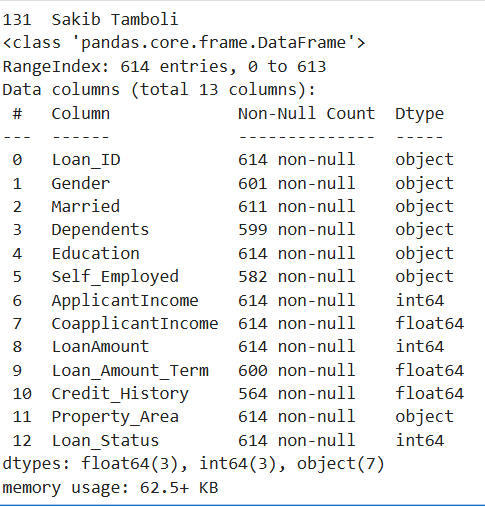
credit\_df=pd.read\_csv("D:/131\_Sakib\_Tamboli/CreditRisk.csv")

credit\_df.shape

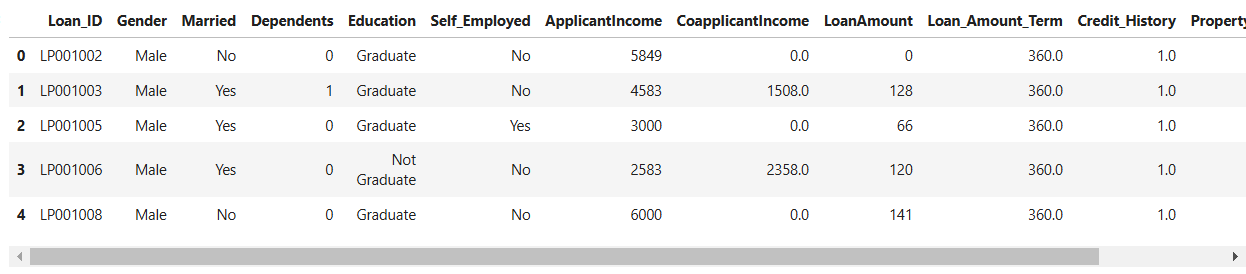


print("131 Sakib Tamboli")

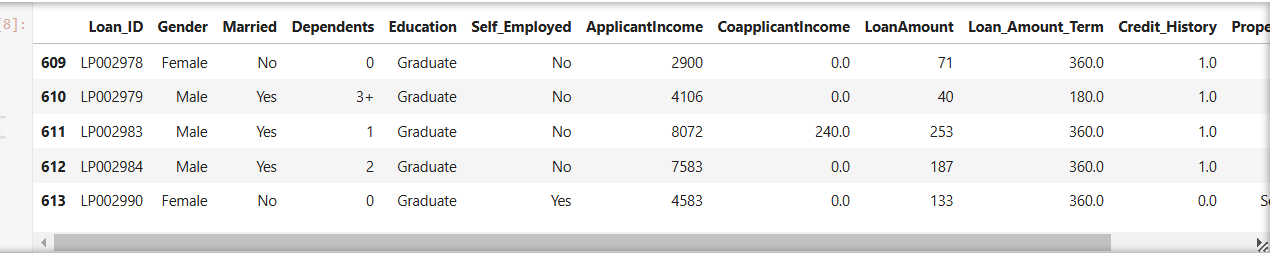
credit\_df.info()



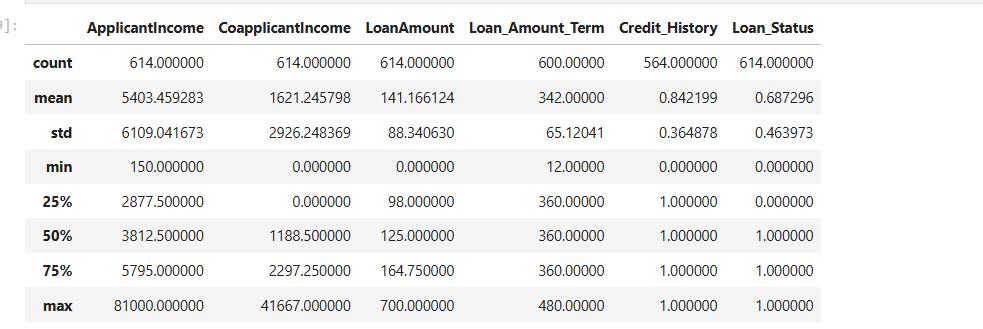
credit\_df.head()



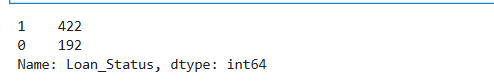
credit\_df.tail()



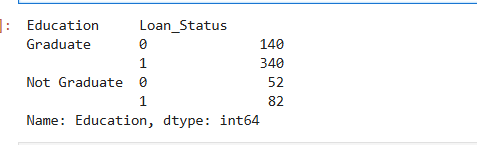
credit\_df.describe()



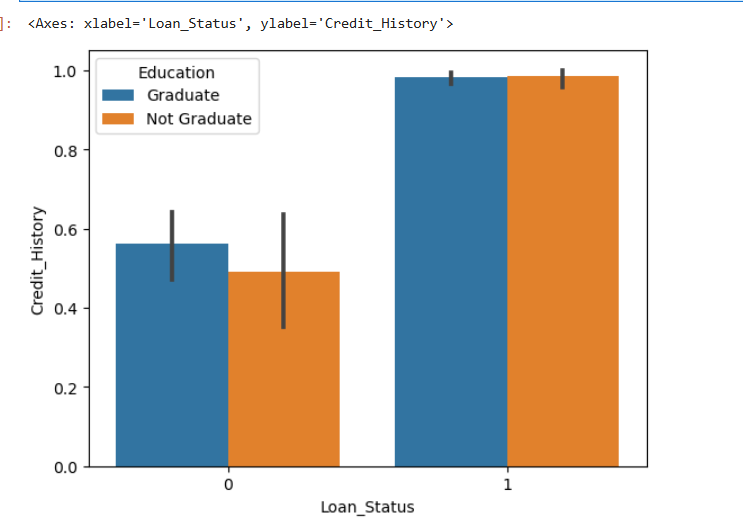
credit\_df.Loan\_Status.value\_counts()



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

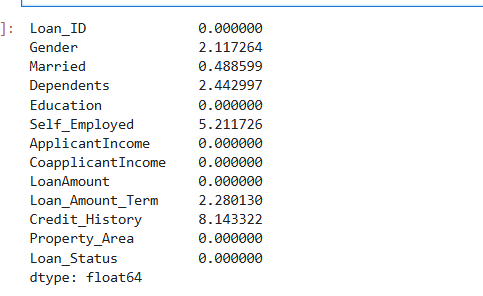


sns.barplot(y='Credit\_History',x='Loan\_Status',hue='Education' ,data=credit\_df)



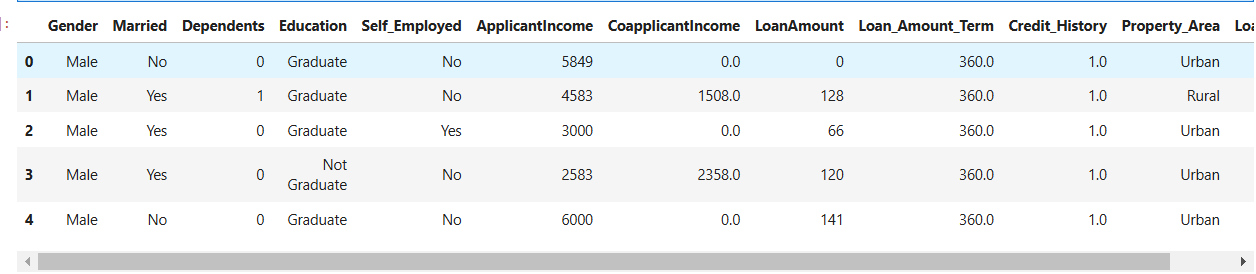
#Fill Null Values

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]

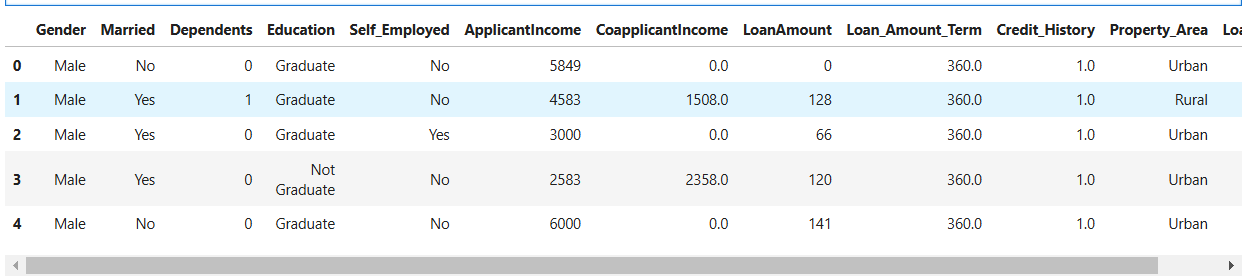
DF[column].fillna(majority, inplace=True)

for column in numeric\_columns:

mean=DF[column].mean()

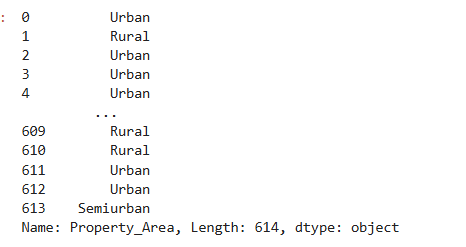
DF[column].fillna(majority, inplace=True)

DF.head()

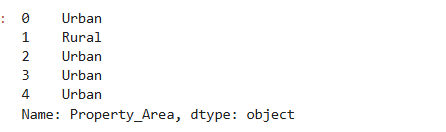


# Categorical columns

DF[object\_columns].Property\_Area



DF[object\_columns].Property\_Area.head()

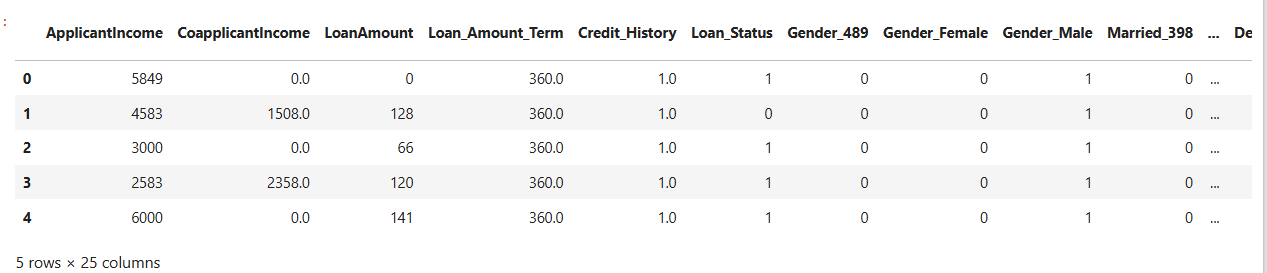


DF\_dummy=pd.get\_dummies(DF, columns=object\_columns)

DF\_dummy.shape



DF\_dummy.head()



#Model

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

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

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

y = DF\_dummy.Loan\_Status

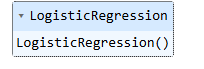
train\_x, test\_x, train\_y, test\_y = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

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))



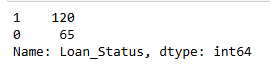
print(confusion\_matrix(train\_y, train\_y\_hat))



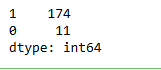
print(confusion\_matrix(test\_y, test\_y\_hat))



test\_y.value\_counts()



pd.Series(test\_y\_hat).value\_counts()

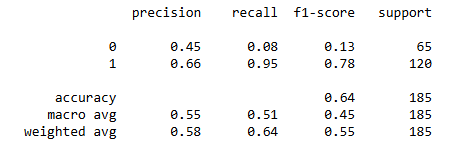


# Accuracy for train

(57 + 295) / train\_y.shape[0]



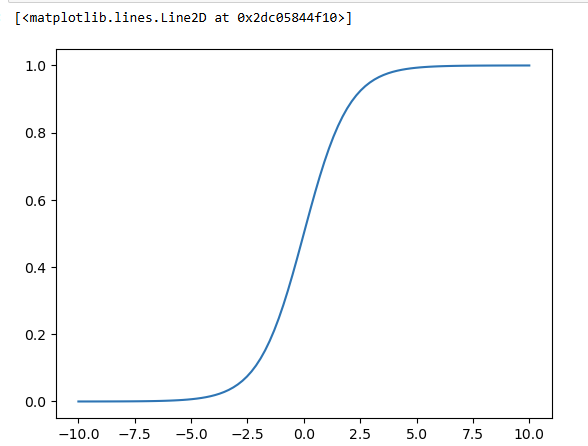
print(classification\_report(test\_y, test\_y\_hat))



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)

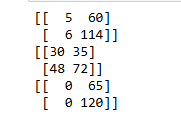
test\_y\_hat\_7 = (model.predict\_proba(test\_x)[:, 1] > 0.7).astype(int)

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))

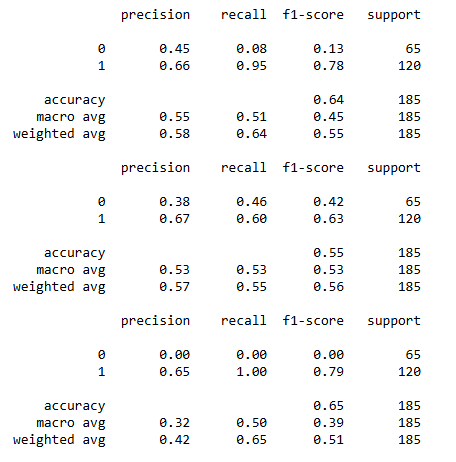
print(confusion\_matrix(test\_y, test\_y\_hat\_3))



print(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 13**

**Aim: Implementation of Support Vector Machine - RBF kernel**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

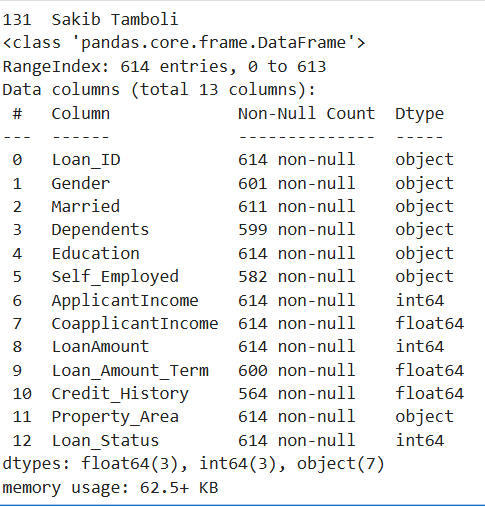
**credit\_df=pd.read\_csv("D:/131\_Sakib\_Tamboli/CreditRisk.csv")**

credit\_df.shape

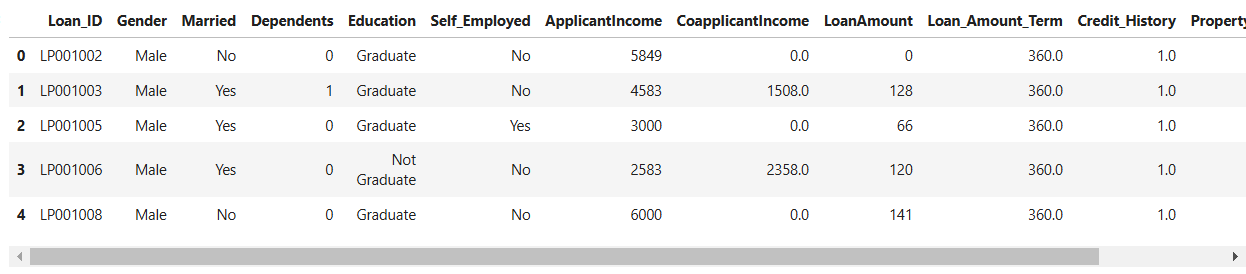


print("131 Sakib Tamboli")

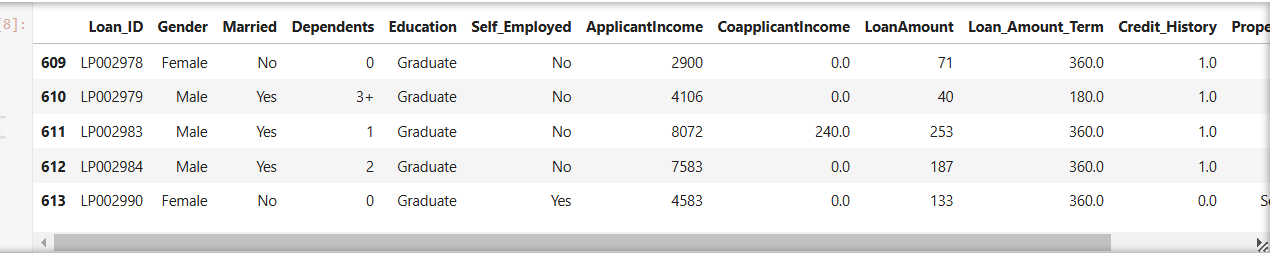
credit\_df.info()



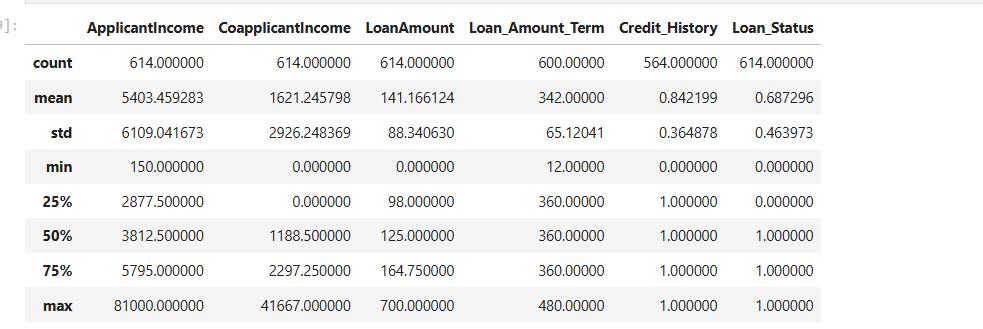
credit\_df.head()



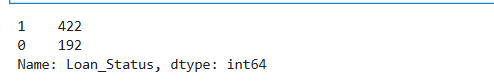
credit\_df.tail()



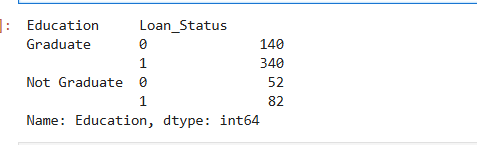
credit\_df.describe()



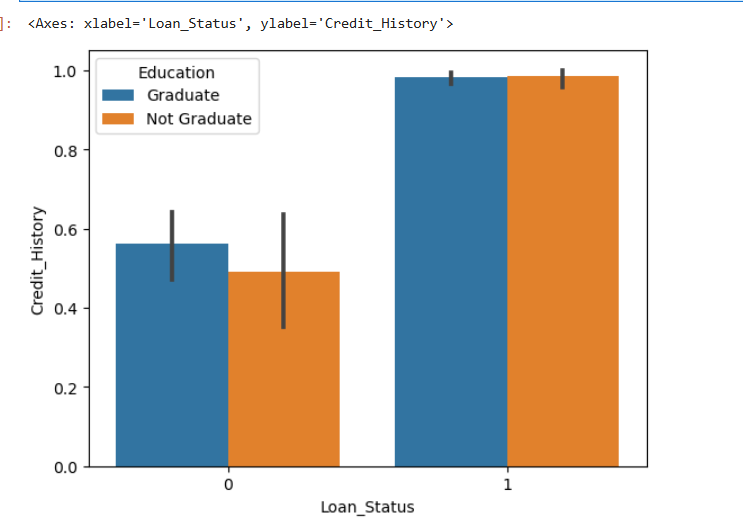
credit\_df.Loan\_Status.value\_counts()



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

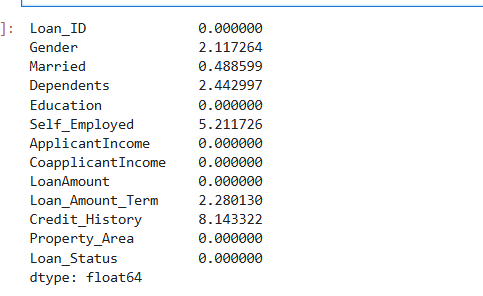


sns.barplot(y='Credit\_History',x='Loan\_Status',hue='Education' ,data=credit\_df)



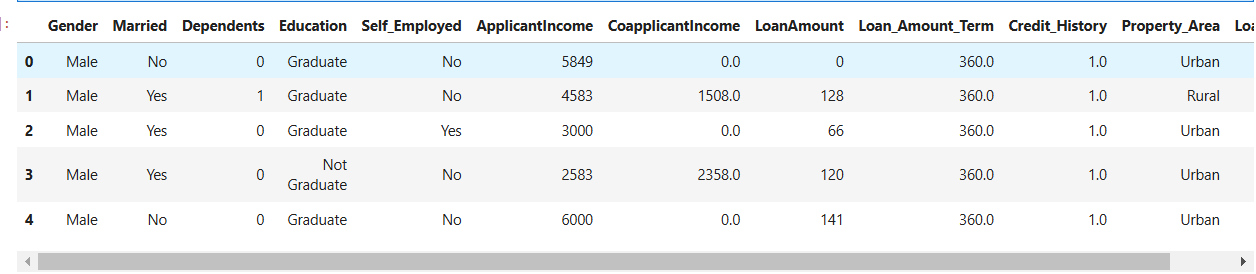
#Fill Null Values

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]

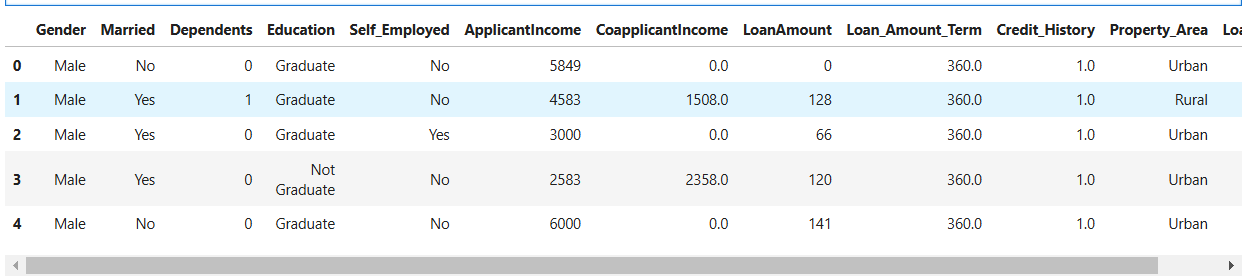
DF[column].fillna(majority, inplace=True)

for column in numeric\_columns:

mean=DF[column].mean()

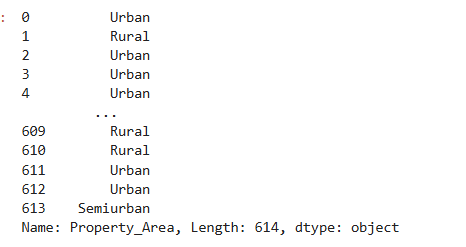
DF[column].fillna(majority, inplace=True)

DF.head()

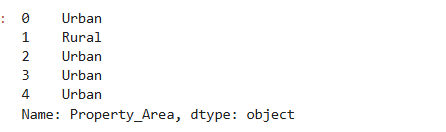


# Categorical columns

DF[object\_columns].Property\_Area



DF[object\_columns].Property\_Area.head()

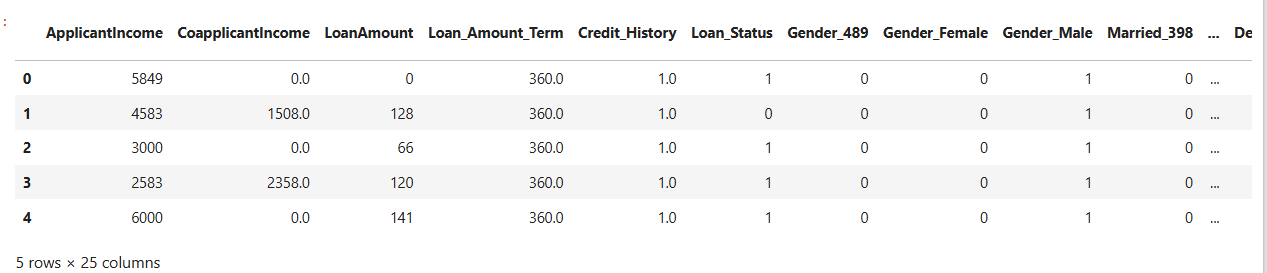


DF\_dummy=pd.get\_dummies(DF, columns=object\_columns)

DF\_dummy.shape



DF\_dummy.head()

****

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

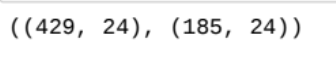
from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

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

y = credit\_df\_dummy.Loan\_Status

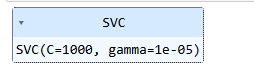
train\_x, test\_x, train\_y, test\_y = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

train\_x.shape, test\_x.shape

****

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

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



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

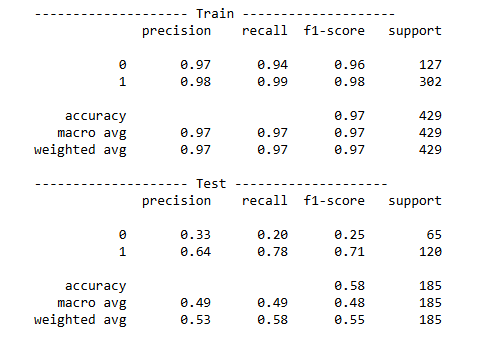
test\_y\_hat = svm\_model.predict(test\_x)

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 14**

**Aim: Implementing elbow method for choosing No. of clusters.**

**Elbow**

**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:/131\_Sakib\_Tamboli/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\_)

print("131 Sakib Tamboli")

# Plot the Elbow method graph

plt.plot(range(1, 11), wcss)

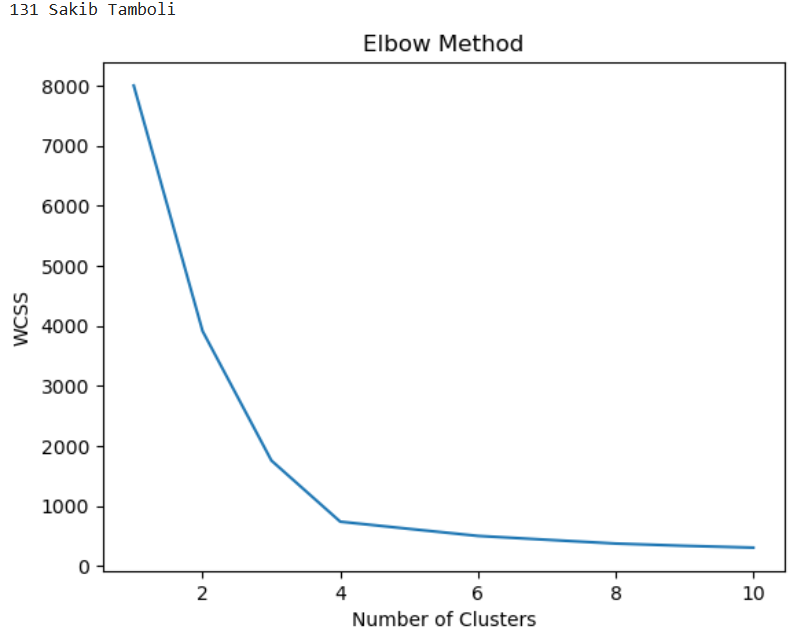
plt.title('Elbow Method')

plt.xlabel('Number of Clusters')

plt.ylabel('WCSS') # Within-Cluster Sum of Squares

plt.show()

**Output:**



**Practical 15**

**Aim: General Ensemble techniques - Implementing Bagging, Stacking, Voting technique.**

**Ensemble Learning using Random Forest, Bagging, and Voting Classifiers**

import pandas as pd

import numpy as np

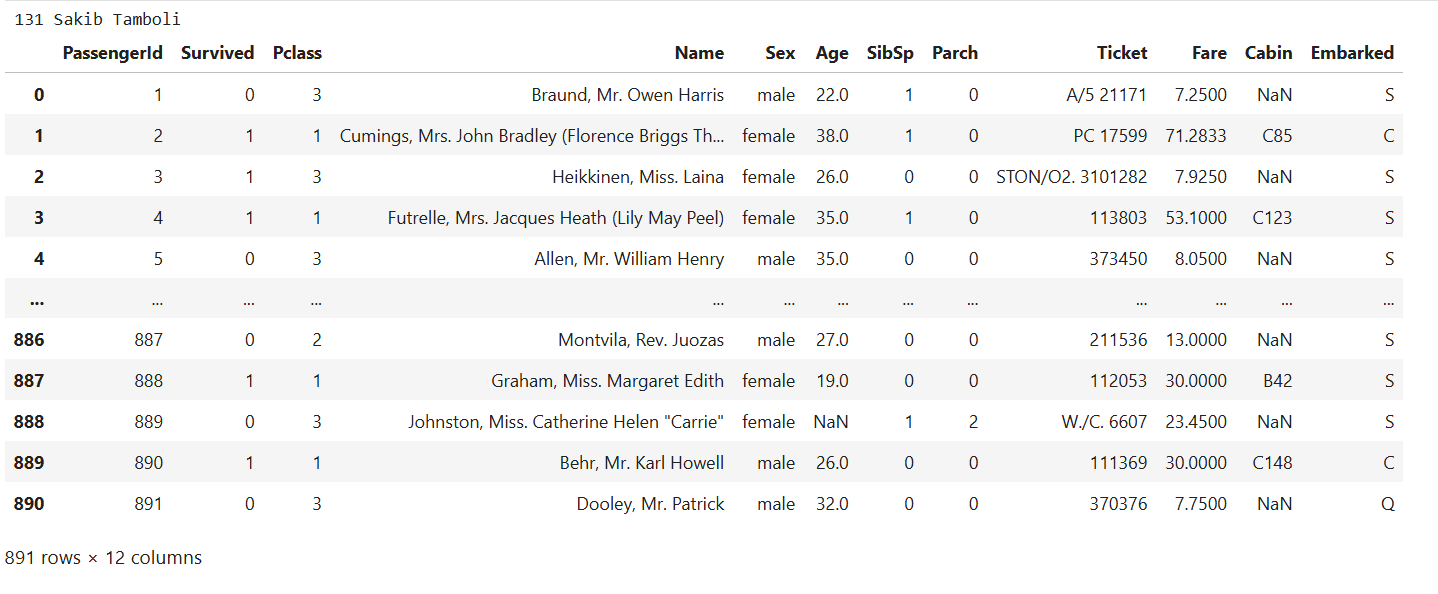
import matplotlib as mpl

import matplotlib.pyplot as plt

df=pd.read\_csv("D:/131\_Sakib\_Tamboli/Titanic-Dataset.csv")

print("131 Sakib Tamboli")

df



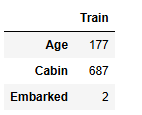
print(df.shape)



#checking for missing data

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

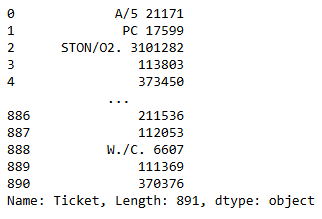
NAs[NAs.sum(axis=1)>0]



df.pop("Cabin")

df.pop("Name")

df.pop("Ticket")



#Filling missing Age value with mean

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

#Filling missing

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

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

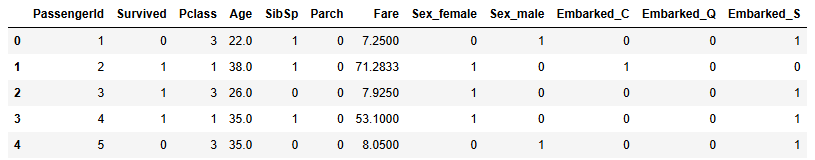
# getting dummies

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

for\_dummy=df.pop(col)

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

df.head()



labels=df.pop("Survived")

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

x\_train,x\_test,y\_train,y\_test=train\_test\_split(df,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 roc\_curve, auc

from sklearn.metrics import accuracy\_score

#false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_pred)

#roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

#roc\_auc

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

print(accuracy)



from sklearn.ensemble import BaggingClassifier,VotingClassifier

print("131 Sakib Tamboli")

#1.Bagging (using RandomForestClassifier as base estimator)

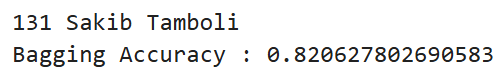
bagging\_model=BaggingClassifier(estimator=RandomForestClassifier(),n\_estimators=10,random\_state=42)

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)



print("131 Sakib Tamboli")

# 2. Voting (using RandomForestClassifier)

voting\_model=VotingClassifier(estimators=[

('rf', RandomForestClassifier()),

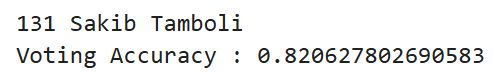
], voting='hard' )

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)



**Implementation and Comparison of Ensemble Learning Techniques -Bagging, Boosting, Stacking, and Voting**

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



#Split thet data inot training and testing sets

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

#1.Bagging (using decision tree as base\_estimator)

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



#2. Boosting (using Adaboost with decision tree)

boosting\_model=AdaBoostClassifier(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("boosting Accuracy :", boosting\_accuracy)



# 3. Stacking (using decision tree, Logistic Regression, and KNN as base estimators)

estimator=[

('dt', DecisionTreeClassifier()),

('lr', LogisticRegression()),

('knn', KNeighborsClassifier())

]

stacking\_model=StackingClassifier(estimators=estimator,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)



# 3. Voting (using decision tree, Logistic Regression, and KNN as base estimators)

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("Voting Accuracy :", stacking\_accuracy)



**Practical 16**

**Aim: Implementing bagging algorithm taking random forest as the base estimator.**

**Basic Random Forest**

import pandas as pd

import numpy as np

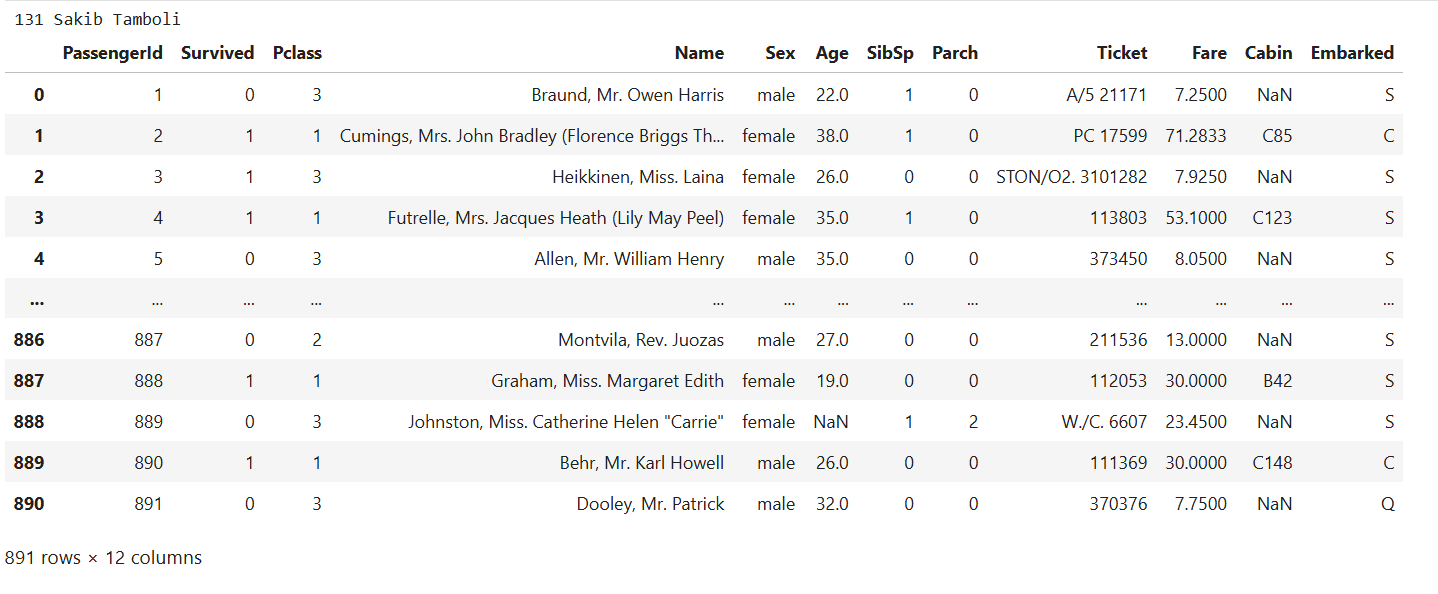
import matplotlib as mpl

import matplotlib.pyplot as plt

**df=pd.read\_csv("D:/131\_Sakib\_Tamboli/Titanic-Dataset.csv")**

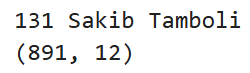
print("131 Sakib Tamboli")

Df



print("131 Sakib Tamboli")

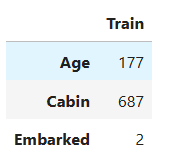
print(df.shape)



#checking for missing data

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

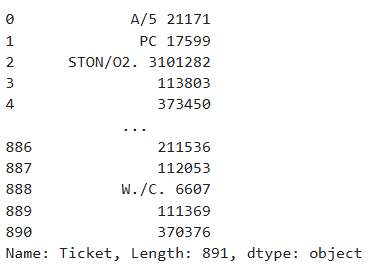
NAs[NAs.sum(axis=1)>0]



df.pop("Cabin")

df.pop("Name")

df.pop("Ticket")



#Filling missing Age value with mean

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

#Filling missing

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

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

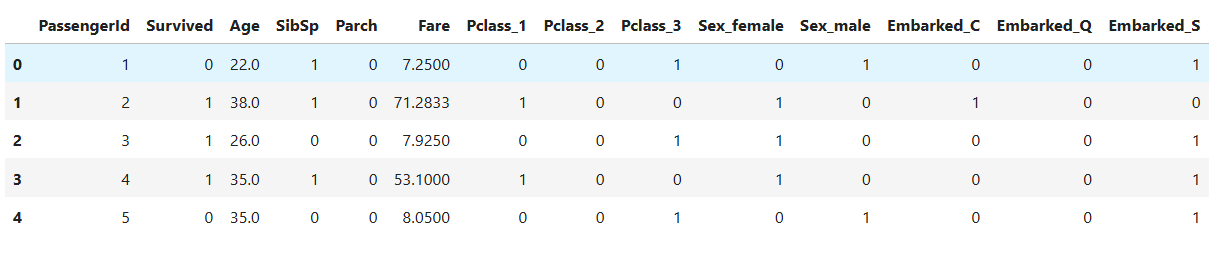
# getting dummies

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

for\_dummy=df.pop(col)

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

df.head()



labels=df.pop("Survived")

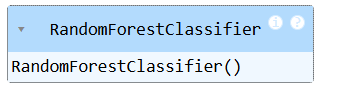
from sklearn.model\_selection import train\_test\_split

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

from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(n\_estimators=100)

rf.fit(x\_train,y\_train)



y\_pred=rf.predict(x\_test)

from sklearn.metrics import roc\_curve,auc

false\_positive\_rate, true\_positive\_rate, thresholds=roc\_curve(y\_test,y\_pred)

roc\_auc=auc(false\_positive\_rate,true\_positive\_rate)

roc\_auc



n\_estimators=[1,2,4,8,16,32,64,100,200]

train\_results=[]

test\_results=[]

for estimator in n\_estimators:

rf=RandomForestClassifier(n\_estimators=estimator,n\_jobs=-1)

rf.fit(x\_train,y\_train)

train\_pred=rf.predict(x\_train)

false\_positive\_rate,true\_positive\_rate,thresholds=roc\_curve(y\_train,train\_pred)

roc\_auc=auc(false\_positive\_rate,true\_positive\_rate)

train\_results.append(roc\_auc)

y\_pred=rf.predict(x\_test)

false\_positive\_rate, true\_positive\_rate,thresholds=roc\_curve(y\_test,y\_pred)

roc\_auc=auc(false\_positive\_rate, true\_positive\_rate)

test\_results.append(roc\_auc)

from matplotlib.legend\_handler import HandlerLine2D

line\_1, = plt.plot(n\_estimators, train\_results, "b", label="Train AUC")

line\_2, = plt.plot(n\_estimators, test\_results, "r", label="Test AUC")

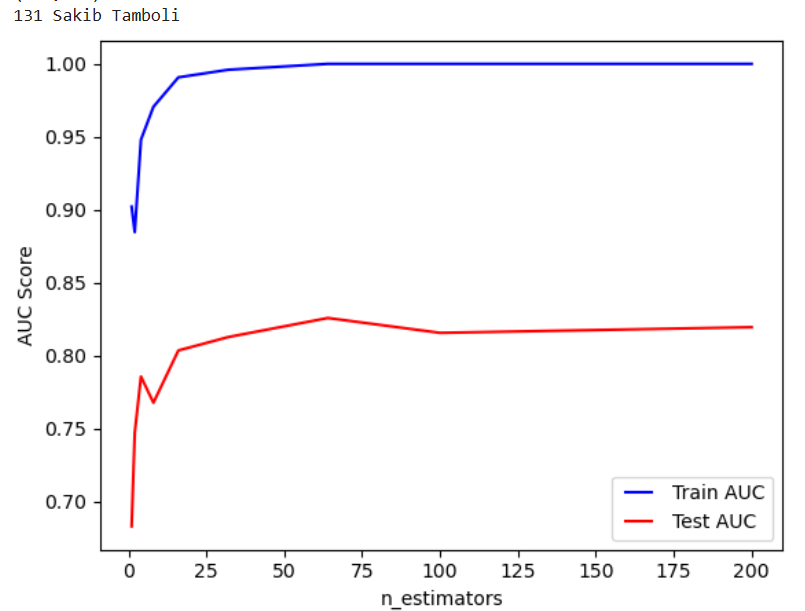
plt.legend(handler\_map={line\_1: HandlerLine2D(numpoints=2), line\_2: HandlerLine2D(numpoints=2)})

plt.ylabel("AUC Score")

plt.xlabel("n\_estimators")

print("131 Sakib Tamboli")

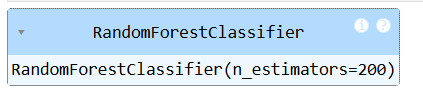
plt.show()



from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(n\_estimators=200)

rf.fit(x\_train,y\_train)



**Bagging on Random Forest**

import pandas as pd

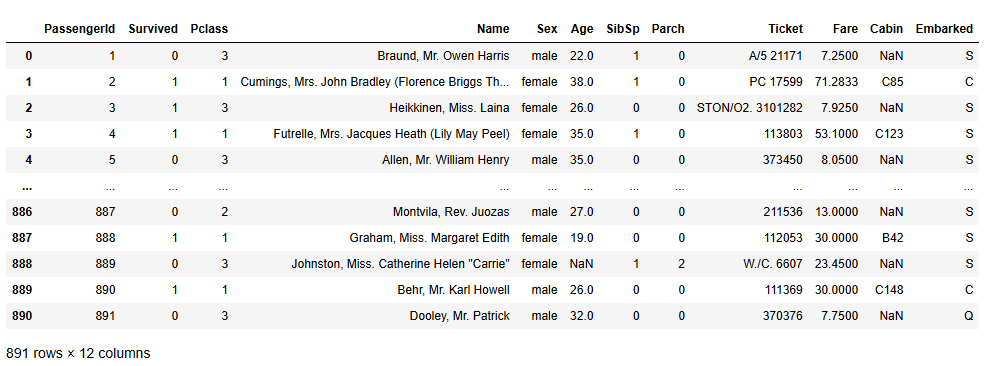
import numpy as np

import matplotlib as mpl

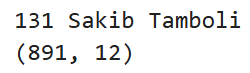
import matplotlib.pyplot as plt

**df=pd.read\_csv("D:/131\_Sakib\_Tamboli/Titanic-Dataset.csv")**

df



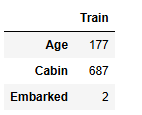
print(df.shape)



#checking for missing data

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

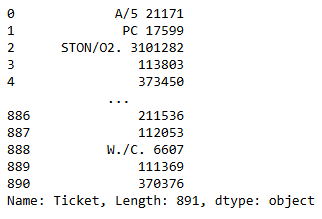
NAs[NAs.sum(axis=1)>0]



df.pop("Cabin")

df.pop("Name")

df.pop("Ticket")



#Filling missing Age value with mean

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

#Filling missing

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

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

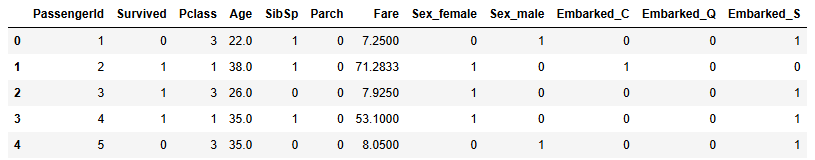
# getting dummies

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

for\_dummy=df.pop(col)

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

df.head()



labels=df.pop("Survived")

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

x\_train,x\_test,y\_train,y\_test=train\_test\_split(df,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 roc\_curve, auc

from sklearn.metrics import accuracy\_score

#false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test, y\_pred)

#roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

#roc\_auc

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

print("131 Sakib Tamboli")

print(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("131 Sakib Tamboli")

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



**Practical 17**

**Aim: Implementation of AdaBoost algorithm.**

**Adaboost with Decision Tree**

import pandas as pd

import numpy as np

import matplotlib as mpl

import matplotlib.pyplot as plt

from sklearn.ensemble import AdaBoostClassifier

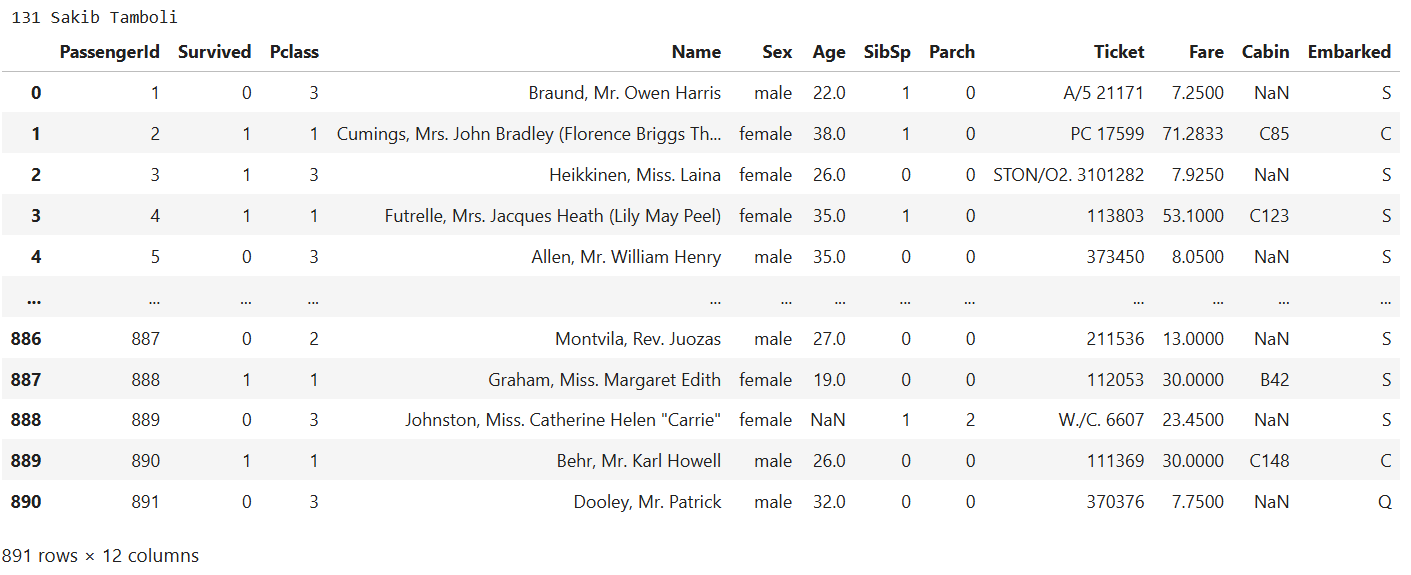
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

**df=pd.read\_csv("D:/131\_Sakib\_Tamboli/Titanic-Dataset.csv")**

print("131 Sakib Tamboli")

df

****

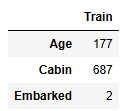
print(df.shape)

****

#checking for missing data

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

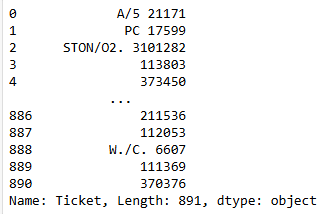
NAs[NAs.sum(axis=1)>0]

****

df.pop("Cabin")

df.pop("Name")

df.pop("Ticket")



#Filling missing Age value with mean

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

#Filling missing

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

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

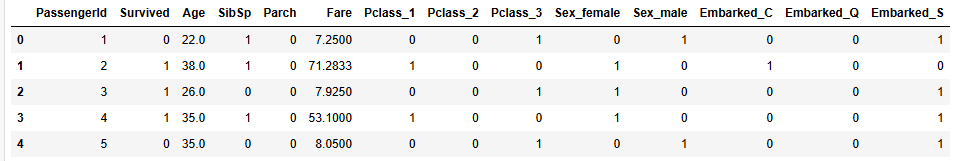
# getting dummies

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

for\_dummy=df.pop(col)

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

df.head()

****

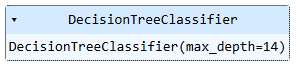
labels=df.pop("Survived")

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

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

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

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

****

train\_y\_hat=df\_model.predict(x\_train)

test\_y\_hat=df\_model.predict(x\_test)

y\_pred=df\_model.predict(x\_test)

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

accuracy

****

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

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

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

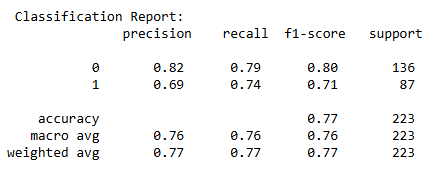
boosting\_accuracy=accuracy\_score(y\_test,boosting\_predictions)

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

****

from sklearn.metrics import classification\_report

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

****

**Practical 18**

**Aim: Implementation gradient boosting algorithm.**

**Stochastic Gradient Boosting**

import pandas as pd

import numpy as np

import matplotlib as mpl

import matplotlib.pyplot as plt

from sklearn.ensemble import GradientBoostingClassifier

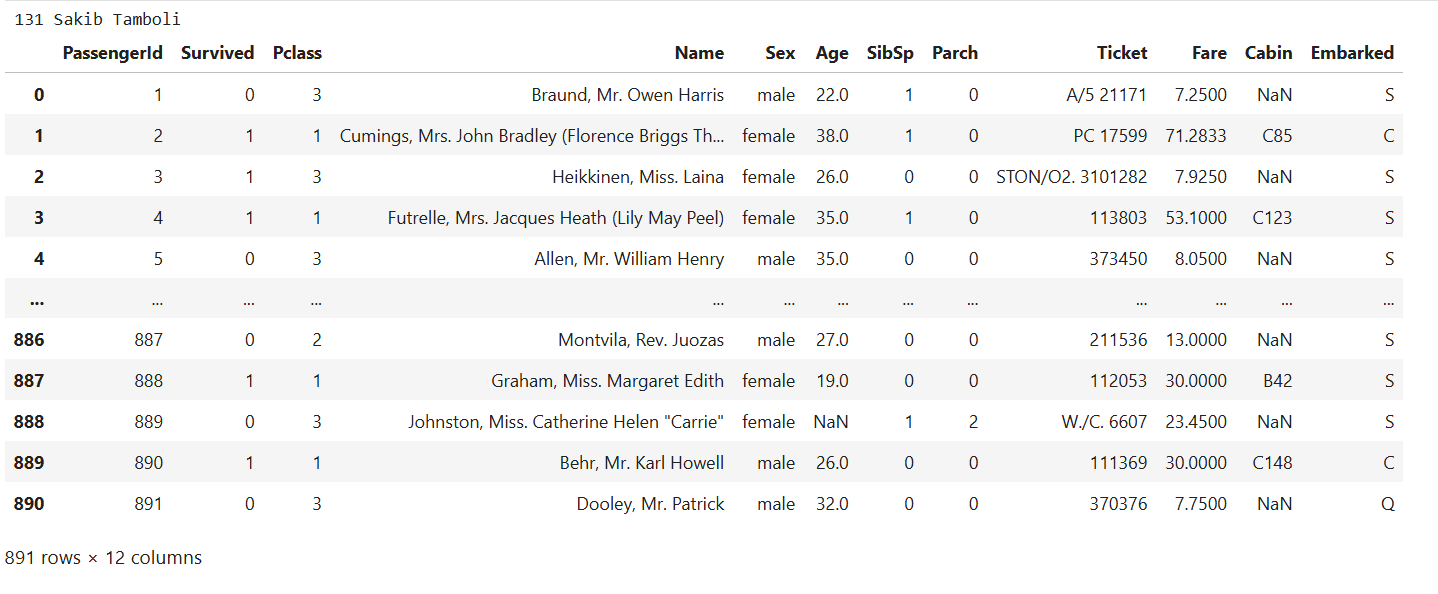
from sklearn.model\_selection import train\_test\_split

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

**df=pd.read\_csv("D:/131\_Sakib\_Tamboli/titanic.csv")**

print("131 Sakib Tamboli")

df

****

print(df.shape)

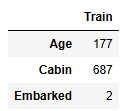
print(“131 Sakib Tamboli”)

****

#checking for missing data

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

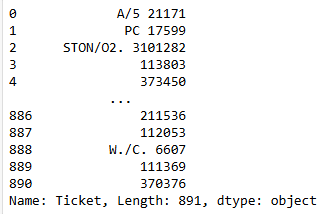
NAs[NAs.sum(axis=1)>0]

****

df.pop("Cabin")

df.pop("Name")

df.pop("Ticket")



#Filling missing Age value with mean

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

#Filling missing

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

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

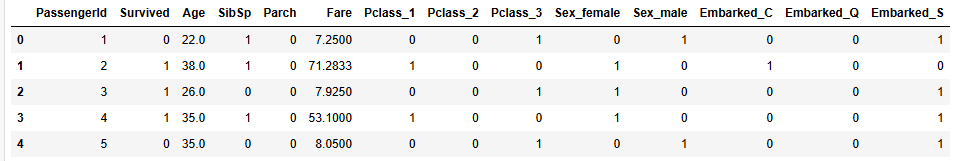
# getting dummies

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

for\_dummy=df.pop(col)

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

df.head()

****

labels=df.pop("Survived")

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

x\_train,x\_test,y\_train,y\_test=train\_test\_split(df,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)

print("Accuarcy:",accuracy)

print("131 Sakib Tamboli")

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

