Name: Sakib Tamboli
Roll no :131 Div :B

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,
```

Name: Sakib Tamboli
Roll no :131 Div :B

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:

```
?-
% d:/131_sakib_tamboli/waterjug_131_sakib_tamboli compiled 0.00 sec, -3 clauses
?- water_jug(3,3).

4L:4 & 3L:2 (Action: Pour water from 3L jug to 4L jug
untill 4L jug is full.)
4L:0 & 3L:2 (Action: Empty 4L jug.)
4L:2 & 3L:0 (Action: Pour water from 3L jug to 4L
jug.)
4L:2 & 3L:0 (Action: Goal State reached...)
true .

?- water_jug(5,5).
Both jugs overflow.
true
Unknown action: □ (h for help)
Action?
Unknown action: □ (h for help)
Action? .

?- water_jug(4,4).
false.

?- water_jug(2,2).
false.

?- water_jug(1,1).
false.

?- █
```

Name: Sakib Tamboli
Roll no :131 Div :B

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).
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
% 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').
```

Name: Sakib Tamboli
Roll no :131 Div :B

Output:

```
% d:/131_sakib_tamboli/tictactoe compiled 0.00 sec, -3 clauses
?- play.
  |   |
--+---+--
  |   |
--+---+--
  |   |

X's turn. Enter position (1-9): 3.
  |   | X
--+---+--
  |   |
--+---+--
  |   |

O's turn. Enter position (1-9): |: 2.
  | O | X
--+---+--
  |   |
--+---+--
  |   |

X's turn. Enter position (1-9): |: 5.
  | O | X
--+---+--
  | X |
--+---+--
  |   |

O's turn. Enter position (1-9): |: 9.
  | O | X
--+---+--
  | X |
--+---+--
  |   | O

X's turn. Enter position (1-9): |: 7.
  | O | X
--+---+--
  | X |
--+---+--
X |   | O

X wins!
true .
```

Name: Sakib Tamboli
Roll no :131 Div :B

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:

```
?-
% d:/131_sakib_tamboli/8puzzle compiled 0.00 sec, -3 clauses
?- test(plan).
Initial state:
at(tile7,9)
at(tile1,8)
at(tile5,7)
at(tile6,6)
at(tile2,5)
at(empty,4)
at(tile8,3)
at(tile3,2)
at(tile4,1)

Goal state:
[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)]

false.
```

Name: Sakib Tamboli
Roll no :131 Div :B

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)
```

```
<class 'numpy.ndarray'>
131 Sakib Tamboli
[1 2 3 4]
```

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

```
array(['1', '2', 'n', '4'], dtype='<U11')
```

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

```
array(['1', '2', 'name', '4'], dtype='<U11')
```

```
#Generating array using arange
d=np.arange(1,11,2)
d
```

```
array([1, 3, 5, 7, 9])
```

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

Name: Sakib Tamboli
Roll no :131 Div :B

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

np.random.rand(4)

```
array([0.01512685, 0.7664744 , 0.0110422 , 0.82456936])
```

np.zeros((3,4))

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

np.random.rand(5,4)

```
array([[0.05504293, 0.52274772, 0.28453706, 0.06866304],
       [0.22054251, 0.53991789, 0.565261  , 0.13424126],
       [0.39523458, 0.1932152 , 0.74504561, 0.98108558],
       [0.96146691, 0.87538962, 0.37460555, 0.68020478],
       [0.40000506, 0.73520801, 0.95441392, 0.65584187]])
```

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

```
array([1.00000000e+01, 1.77827941e+03, 3.16227766e+05, 5.62341325e+07,
       1.00000000e+10])
```

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

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

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

Name: Sakib Tamboli
Roll no :131 Div :B

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

mat.shape

```
(3, 3)
```

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

```
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])
```

np.add(mat,mat2)

```
array([[11, 22, 33],
       [44, 55, 66],
       [77, 88, 99]])
```

np.multiply(mat,mat2)

```
array([[ 10,  40,  90],
       [160, 250, 360],
       [490, 640, 810]])
```

mat[1,2]

```
6
```

mat[1:2]

```
array([[4, 5, 6]])
```

arr1=np.array([[1,2,3],[4,5,6],[7,8,9]])
arr2=np.arange(10,19,1).reshape(3,3)
arr1

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

arr2

```
array([[10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

np.multiply(arr1,arr2)

```
array([[ 10,  22,  36],
       [ 52,  70,  90],
       [112, 136, 162]])
```

arr2[:,0]

```
array([10, 13, 16])
```

arr2[0,:]

```
array([10, 11, 12])
```

arr1_sub=arr1[:2,:2]
arr1_sub

```
array([[1, 2],
       [4, 5]])
```

arr1

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

a_row=np.append(arr1,[[10,11,12]],axis=0)
a_row

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

#PANDAS

import pandas as pd
data1=pd.read_csv('D:/131_Sakib_Tamboli/mtcars.csv')
print("131 Sakib Tamboli")

Name: Sakib Tamboli
Roll no :131 Div :B

data1.info()

```
131 Sakib Tamboli
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   model   32 non-null     object
 1   mpg     32 non-null     float64
 2   cyl     32 non-null     int64
 3   disp    32 non-null     float64
 4   hp      32 non-null     int64
 5   drat    32 non-null     float64
 6   wt      32 non-null     float64
 7   qsec    32 non-null     float64
 8   vs      32 non-null     int64
 9   am      32 non-null     int64
 10  gear    32 non-null     int64
 11  carb    32 non-null     int64
dtypes: float64(5), int64(6), object(1)
memory usage: 3.1+ KB
```

data1.head()

Out[11]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

data1.tail()

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

Name: Sakib Tamboli
Roll no :131 Div :B

data1.isnull().sum()

```
model    0
mpg      0
cyl      0
disp     0
hp       0
drat     0
wt       0
qsec     0
vs       0
am       0
gear     0
carb     0
dtype: int64
```

data1.isnull()

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False	False	False	False	False	False
14	False	False	False	False	False	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False	False	False	False	False	False

Data1.size

```
384
```

Name: Sakib Tamboli
Roll no :131 Div :B

Data1.shape

```
(32, 12)
```

Data1.ndim

```
2
```

data1.at[4,'model']

```
'Hornet Sportabout'
```

Data1.iat[4,3]

```
360.0
```

data1.loc[:,'model']

```
0               Mazda RX4
1           Mazda RX4 Wag
2              Datsun 710
3          Hornet 4 Drive
4       Hornet Sportabout
5                 Valiant
6              Duster 360
7               Merc 240D
8                Merc 230
9                Merc 280
10              Merc 280C
11             Merc 450SE
12             Merc 450SL
13            Merc 450SLC
14     Cadillac Fleetwood
15    Lincoln Continental
16      Chrysler Imperial
17               Fiat 128
18            Honda Civic
19         Toyota Corolla
20          Toyota Corona
21       Dodge Challenger
22            AMC Javelin
23             Camaro Z28
24       Pontiac Firebird
25              Fiat X1-9
26          Porsche 914-2
27           Lotus Europa
28         Ford Pantera L
29           Ferrari Dino
30          Maserati Bora
31             Volvo 142E
Name: model, dtype: object
```

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

Name: Sakib Tamboli
Roll no :131 Div :B

	model	mpg
0	Mazda RX4	21.0
1	Mazda RX4 Wag	21.0
2	Datsun 710	22.8
3	Hornet 4 Drive	21.4
4	Hornet Sportabout	18.7

Data1['model'].dtype

```
dtype('O')
```

Data1.axes

```
[RangeIndex(start=0, stop=32, step=1),
 Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
        'gear', 'carb'],
       dtype='object')]
```

data1.columns

```
Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
       'gear', 'carb'],
      dtype='object')
```

data1['hp'].std()

```
68.56286848932059
```

data1['mpg'].mean()

```
20.090625000000003
```

data1['mpg'].median()

```
]: 19.2
```

data1['hp'].describe()

```
count     32.000000
mean     146.687500
std       68.562868
min       52.000000
25%       96.500000
50%      123.000000
75%      180.000000
max      335.000000
Name: hp, dtype: float64
```

Data1.iloc[-1]

Name: Sakib Tamboli
Roll no :131 Div :B

```
model    Volvo 142E
mpg            21.4
cyl               4
disp          121.0
hp              109
drat           4.11
wt             2.78
qsec           18.6
vs                1
am                1
gear              4
carb              2
Name: 31, dtype: object
```

Data1.iloc[:,-1]

```
0     4
1     4
2     1
3     1
4     2
5     1
6     4
7     2
8     2
9     4
10    4
11    3
12    3
13    3
14    4
15    4
16    4
17    1
18    2
19    1
20    1
21    2
22    2
23    4
24    2
25    1
26    2
27    2
28    4
29    6
30    8
31    2
Name: carb, dtype: int64
```

data1.iloc[-1]

Name: Sakib Tamboli
Roll no :131 Div :B

```
model    Volvo 142E
mpg            21.4
cyl               4
disp          121.0
hp              109
drat           4.11
wt             2.78
qsec           18.6
vs                1
am                1
gear              4
carb              2
Name: 31, dtype: object
```

data1_sorted=data1.sort_values(by='mpg')
data1_sorted

Name: Sakib Tamboli
Roll no :131 Div :B

Out[46]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
22	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
21	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
24	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
18	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1

data1_sorted.head()

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4

Name: Sakib Tamboli
Roll no :131 Div :B

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

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1

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

```
model     7
mpg       7
cyl       7
disp      7
hp        7
drat      7
wt        7
qsec      7
vs        7
am        7
gear      7
carb      7
dtype: int64
```

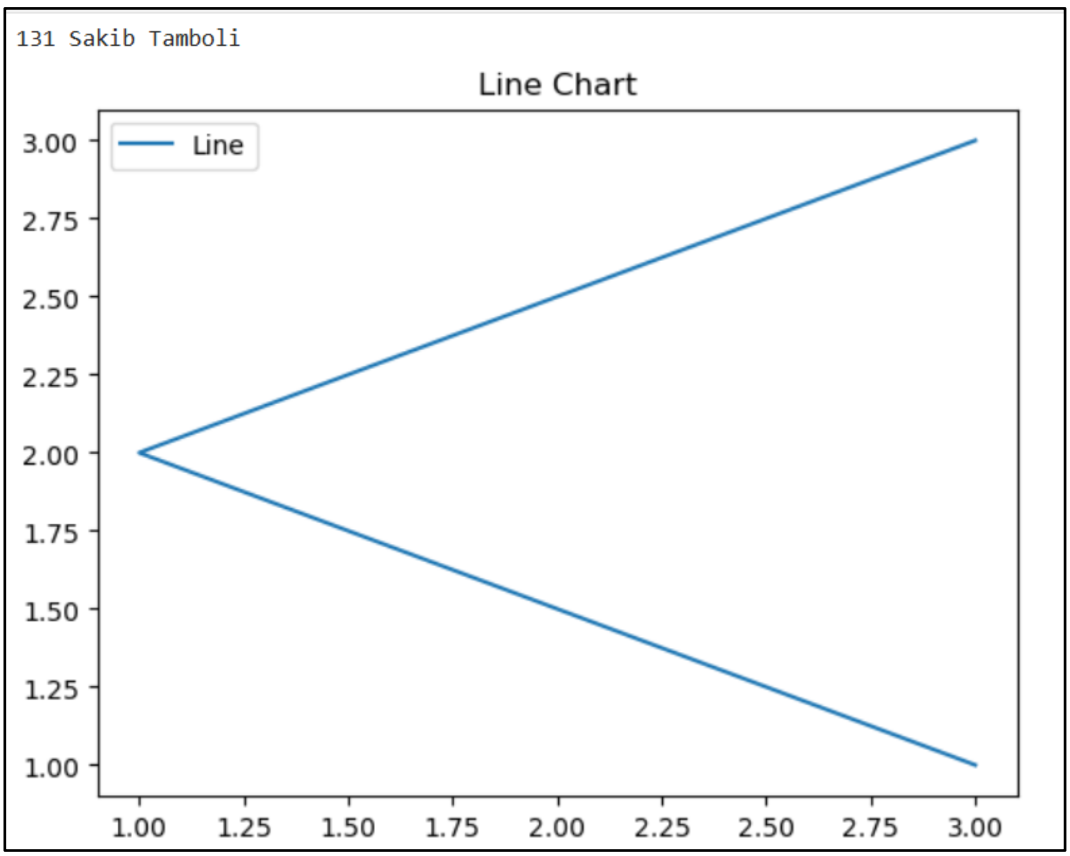
Name: Sakib Tamboli
Roll no :131 Div :B

Practical 5

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

<u>#Matplotlib</u>

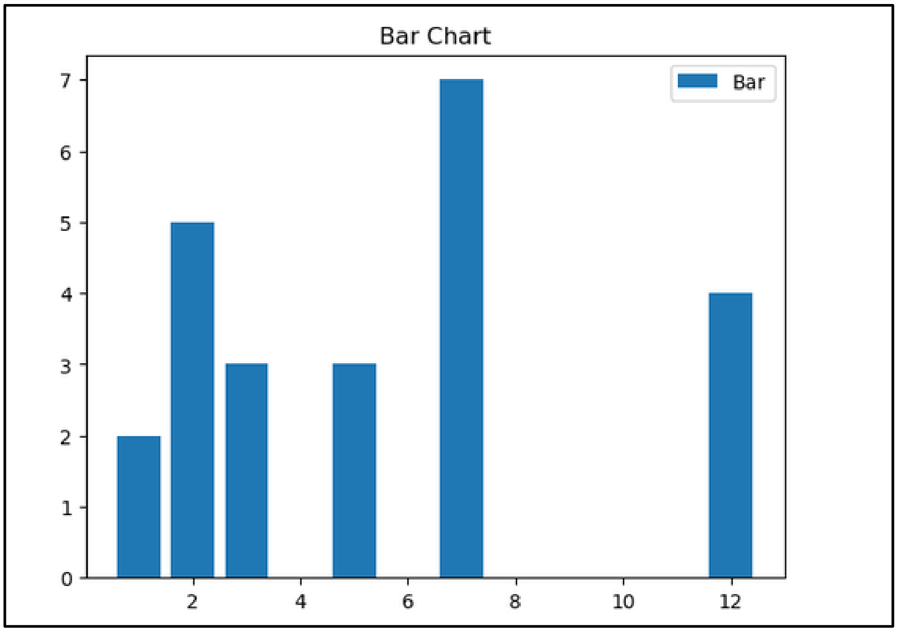
```
import matplotlib.pyplot as plt
x=[3,1,3]
y=[3,2,1]

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]
```

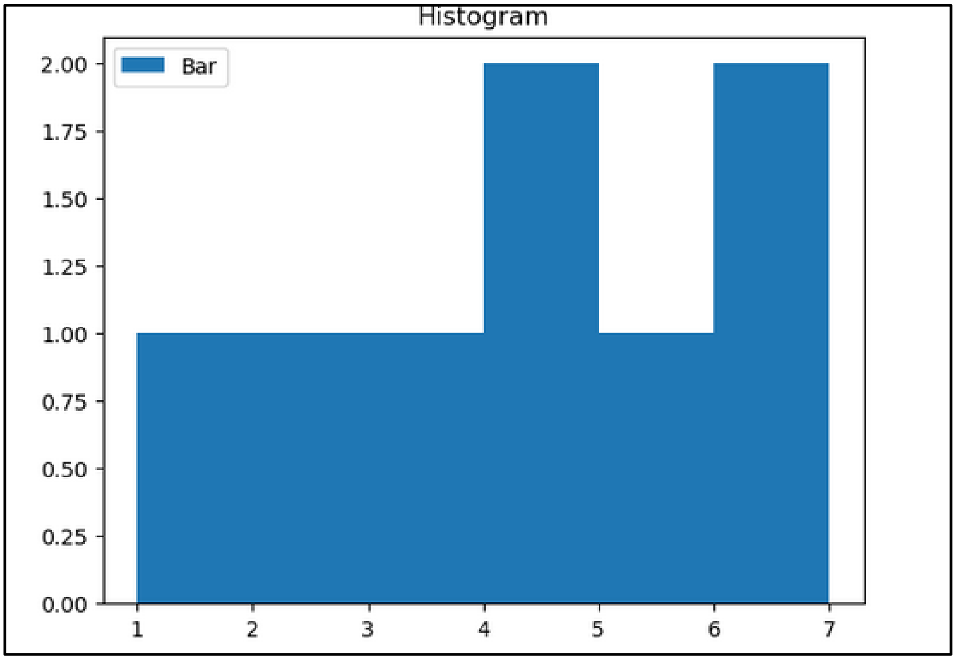
Name: Sakib Tamboli
Roll no :131 Div :B

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

plt.bar(x,y)
plt.title("Bar Chart")

plt.legend(["Bar"])
plt.show()
```

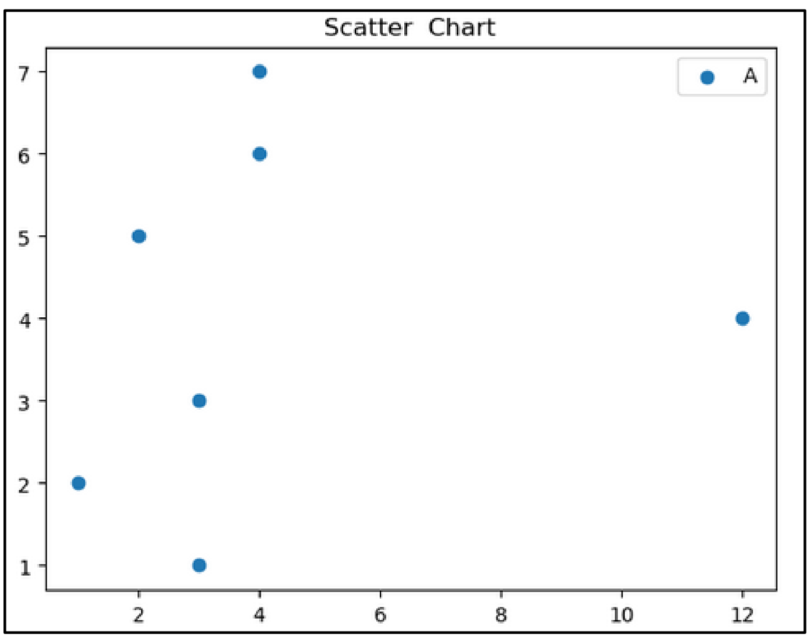



```
x=[1,2,3,4,5,6,7,4]


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

plt.legend(["Bar"])
plt.show()
```

Name: Sakib Tamboli
Roll no :131 Div :B

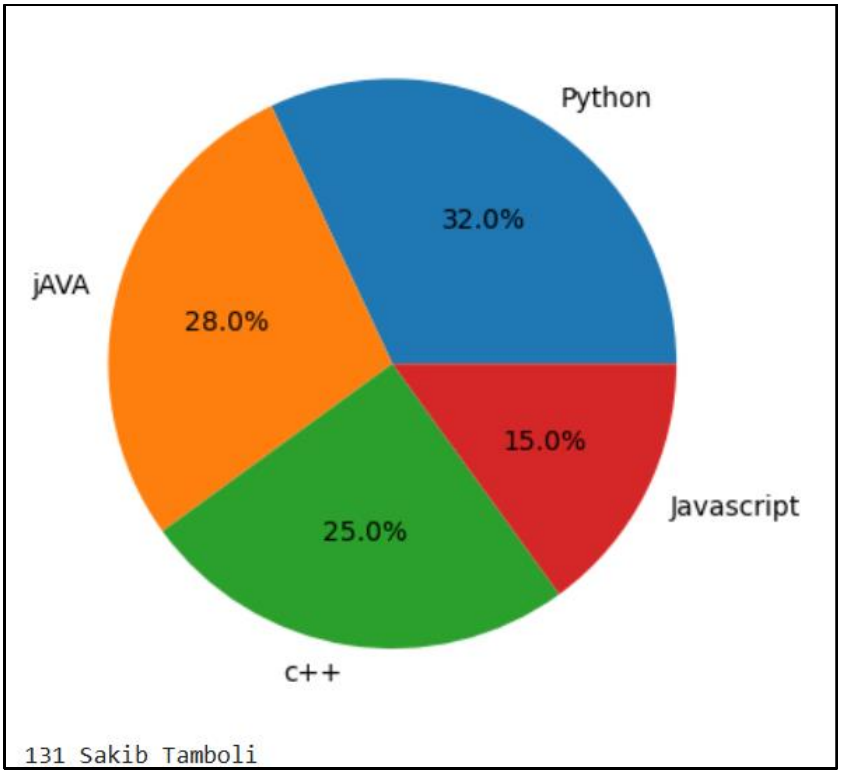



```
x=[3,1,3,12,2,4,4]
y=[3,2,1,4,5,6,7]

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)
```

```
131 Sakib Tamboli
Integral of x^2 from 0 to 2: 2.666666666666667
Minimum found at x = 2.9999999840660854
Mean: 5.833333333333333
Standard Deviation: 2.4094720491334933
Solution of linear equations: [1. 2.]
PDF of standard normal distribution at x=0: 0.3989422804014327
```

Name: Sakib Tamboli
Roll no :131 Div :B

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()

```
131 Sakib Tamboli
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   model   32 non-null     object
 1   mpg     32 non-null     float64
 2   cyl     32 non-null     int64
 3   disp    32 non-null     float64
 4   hp      32 non-null     int64
 5   drat    32 non-null     float64
 6   wt      32 non-null     float64
 7   qsec    32 non-null     float64
 8   vs      32 non-null     int64
 9   am      32 non-null     int64
 10  gear    32 non-null     int64
 11  carb    32 non-null     int64
dtypes: float64(5), int64(6), object(1)
memory usage: 3.1+ KB
```

data1.head()

Out[11]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

data1.tail()

Name: Sakib Tamboli
Roll no :131 Div :B

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	1	5	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5	0	1	5	4
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5	0	1	5	6
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6	0	1	5	8
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6	1	1	4	2

data1.isnull().sum()

```
model     0
mpg       0
cyl       0
disp      0
hp        0
drat      0
wt        0
qsec      0
vs        0
am        0
gear      0
carb      0
dtype: int64
```

data1.isnull()

Name: Sakib Tamboli
Roll no :131 Div :B

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False	False	False	False	False	False
14	False	False	False	False	False	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False	False	False	False	False	False

Data1.size

```
384
```

Data1.shape

```
(32, 12)
```

Data1.ndim

```
2
```

data1.at[4,'model']

```
'Hornet Sportabout'
```

Data1.iat[4,3]

```
360.0
```

Name: Sakib Tamboli
Roll no :131 Div :B

data1.loc[:,'model']

```
0              Mazda RX4
1          Mazda RX4 Wag
2             Datsun 710
3         Hornet 4 Drive
4      Hornet Sportabout
5                Valiant
6             Duster 360
7              Merc 240D
8               Merc 230
9               Merc 280
10             Merc 280C
11            Merc 450SE
12            Merc 450SL
13           Merc 450SLC
14    Cadillac Fleetwood
15   Lincoln Continental
16     Chrysler Imperial
17              Fiat 128
18           Honda Civic
19        Toyota Corolla
20         Toyota Corona
21      Dodge Challenger
22           AMC Javelin
23            Camaro Z28
24      Pontiac Firebird
25             Fiat X1-9
26         Porsche 914-2
27          Lotus Europa
28        Ford Pantera L
29          Ferrari Dino
30         Maserati Bora
31            Volvo 142E
Name: model, dtype: object
```

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

	model	mpg
0	Mazda RX4	21.0
1	Mazda RX4 Wag	21.0
2	Datsun 710	22.8
3	Hornet 4 Drive	21.4
4	Hornet Sportabout	18.7

Data1['model'].dtype

```
dtype('O')
```

Data1.axes

Name: Sakib Tamboli
Roll no :131 Div :B

```
[RangeIndex(start=0, stop=32, step=1),
 Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
        'gear', 'carb'],
       dtype='object')]
```

data1.columns

```
Index(['model', 'mpg', 'cyl', 'disp', 'hp', 'drat', 'wt', 'qsec', 'vs', 'am',
       'gear', 'carb'],
      dtype='object')
```

data1['hp'].std()

```
68.56286848932059
```

data1['mpg'].mean()

```
20.090625000000003
```

data1['mpg'].median()

```
]: 19.2
```

data1['hp'].describe()

```
count     32.000000
mean     146.687500
std       68.562868
min       52.000000
25%       96.500000
50%      123.000000
75%      180.000000
max      335.000000
Name: hp, dtype: float64
```

Data1.iloc[-1]

```
model    Volvo 142E
mpg            21.4
cyl               4
disp          121.0
hp              109
drat           4.11
wt             2.78
qsec           18.6
vs                1
am                1
gear              4
carb              2
Name: 31, dtype: object
```

Data1.iloc[:,-1]

Name: Sakib Tamboli
Roll no :131 Div :B

```
0     4
1     4
2     1
3     1
4     2
5     1
6     4
7     2
8     2
9     4
10    4
11    3
12    3
13    3
14    4
15    4
16    4
17    1
18    2
19    1
20    1
21    2
22    2
23    4
24    2
25    1
26    2
27    2
28    4
29    6
30    8
31    2
Name: carb, dtype: int64
```

data1.iloc[-1]

```
model    Volvo 142E
mpg            21.4
cyl               4
disp          121.0
hp              109
drat           4.11
wt             2.78
qsec           18.6
vs                1
am                1
gear              4
carb              2
Name: 31, dtype: object
```

data1_sorted=data1.sort_values(by='mpg')

Name: Sakib Tamboli
Roll no :131 Div :B

data1_sorted

Out[46]:

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
30	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
13	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
22	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
21	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
28	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
11	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
12	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
10	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
9	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
24	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
29	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
31	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
8	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
7	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
26	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
27	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
18	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1

data1_sorted.head()

Name: Sakib Tamboli
Roll no :131 Div :B

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
15	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
14	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
23	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
6	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
16	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4

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

	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
17	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
19	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
20	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
25	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1

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

```
model     7
mpg       7
cyl       7
disp      7
hp        7
drat      7
wt        7
qsec      7
vs        7
am        7
gear      7
carb      7
dtype: int64
```

Name: Sakib Tamboli
Roll no :131 Div :B

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:

```
0
1
1
1
131 Sakib Tamboli
```

Name: Sakib Tamboli
Roll no :131 Div :B

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")
```

Name: Sakib Tamboli
Roll no :131 Div :B

Output:

```
Predictions: [0 1 1 1]
131 Sakib Tamboli
```

Name: Sakib Tamboli
Roll no :131 Div :B

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))
```

```
131 Sakib Tamboli
Expected=0, Predicted=0
Expected=0, Predicted=0
Expected=0, Predicted=0
Expected=0, Predicted=0
Expected=0, Predicted=0
Expected=1, Predicted=1
Expected=1, Predicted=1
Expected=1, Predicted=1
Expected=1, Predicted=1
Expected=1, Predicted=1
```

```
# 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")
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
>epoch=0, lrate=0.100, error=2.000
>epoch=1, lrate=0.100, error=1.000
>epoch=2, lrate=0.100, error=0.000
>epoch=3, lrate=0.100, error=0.000
>epoch=4, lrate=0.100, error=0.000
[-0.1, 0.20653640140000007, -0.23418117710000003]
131 Sakib Tamboli
```

Name: Sakib Tamboli
Roll no :131 Div :B

Practical 10

Aim: Implementation of principal component analysis.

PCA

Code:

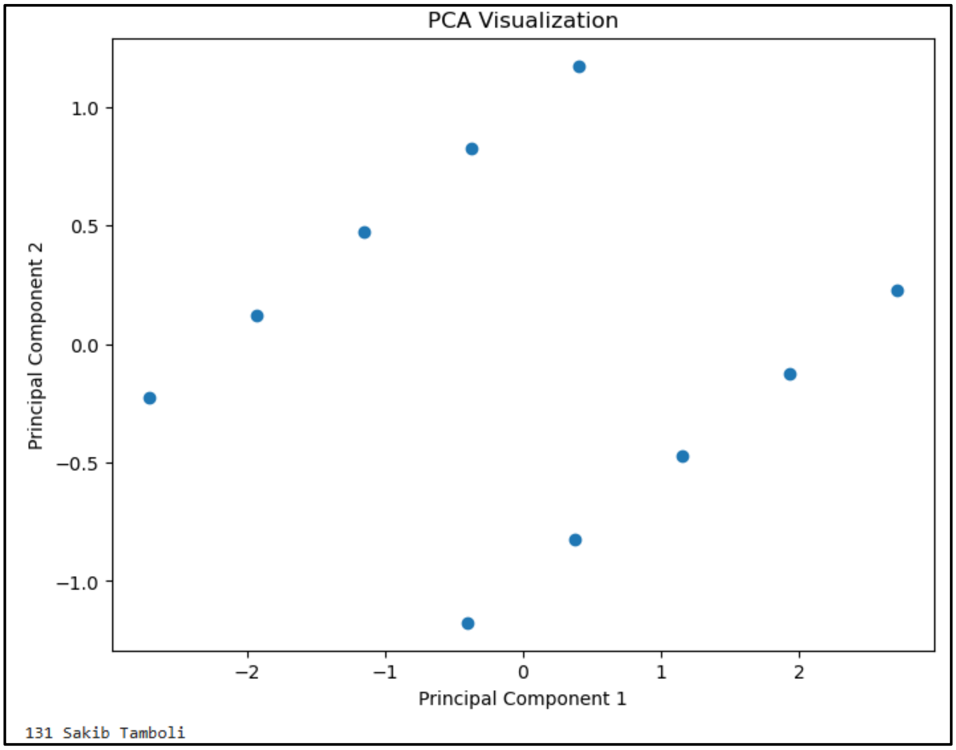
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

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)
```

```
131 Sakib Tamboli
array([[-1.5666989 ,  1.5666989 , -1.5666989 ],
       [-1.21854359,  1.21854359, -0.87038828],
       [-0.87038828,  0.87038828, -0.17407766],
       [-0.52223297,  0.52223297,  0.52223297],
       [-0.17407766,  0.17407766,  1.21854359],
       [ 0.17407766, -0.17407766, -1.21854359],
       [ 0.52223297, -0.52223297, -0.52223297],
       [ 0.87038828, -0.87038828,  0.17407766],
       [ 1.21854359, -1.21854359,  0.87038828],
       [ 1.5666989 , -1.5666989 ,  1.5666989 ]])
```

```
131 Sakib Tamboli
   principal component 1  principal component 2
0              -2.704116              -0.226706
1              -1.926645               0.123740
2              -1.149174               0.474186
3              -0.371704               0.824632
4               0.405767               1.175078
5              -0.405767              -1.175078
6               0.371704              -0.824632
7               1.149174              -0.474186
8               1.926645              -0.123740
9               2.704116               0.226706
```

```
# 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")
```

Name: Sakib Tamboli
Roll no :131 Div :B




```
# 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))
```

```
Explained Variance Ratio: [0.84317429 0.15682571]
Total Explained Variance: 1.0
```

Name: Sakib Tamboli
Roll no :131 Div :B

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()
```

131 Sakib Tamboli

	Index	Height(Inches)	Weight(Pounds)
count	25000.000000	25000.000000	25000.000000
mean	12500.500000	67.993114	127.079421
std	7217.022701	1.901679	11.660898
min	1.000000	60.278360	78.014760
25%	6250.750000	66.704397	119.308675
50%	12500.500000	67.995700	127.157750
75%	18750.250000	69.272958	134.892850
max	25000.000000	75.152800	170.924000

```
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]
```

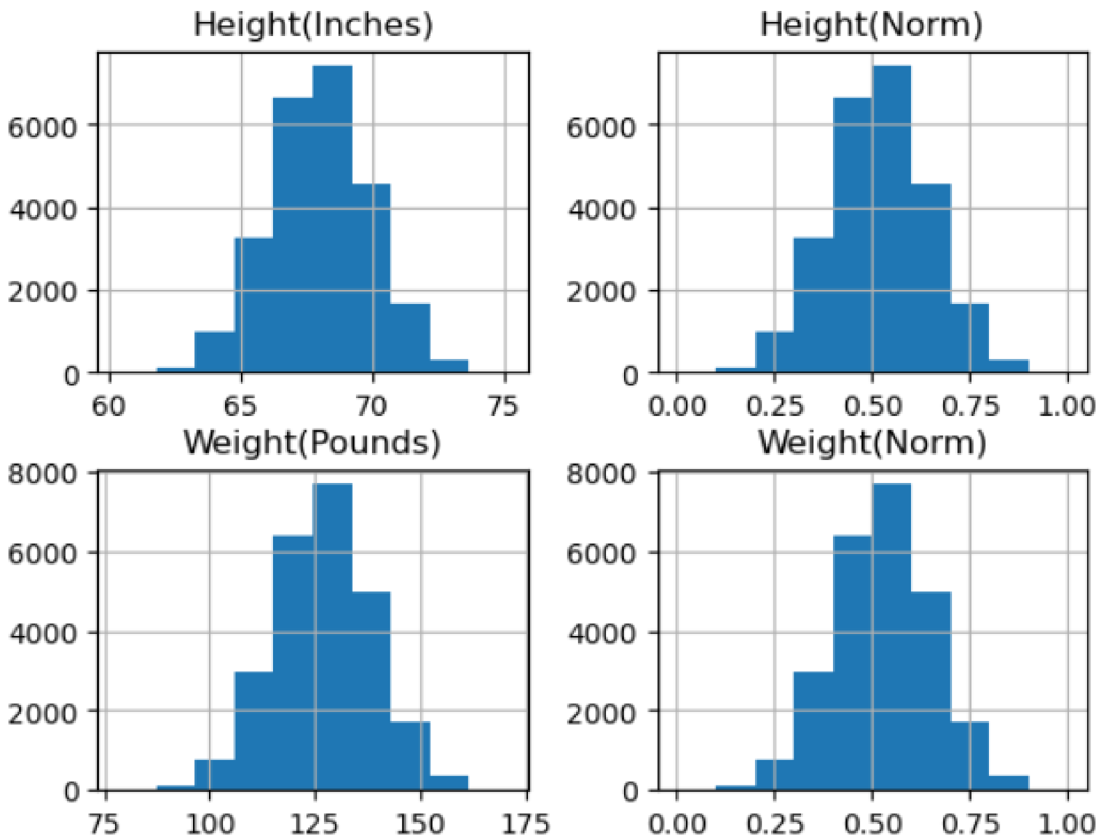
Name: Sakib Tamboli
Roll no :131 Div :B

```
hw_df['Weight(Norm)']=hw_scaled[:,1]

hw_df.describe()
```

	Index	Height(Inches)	Weight(Pounds)	Height(Norm)	Weight(Norm)
count	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000
mean	12500.500000	67.993114	127.079421	0.518658	0.528092
std	7217.022701	1.901679	11.660898	0.127849	0.125508
min	1.000000	60.278360	78.014760	0.000000	0.000000
25%	6250.750000	66.704397	119.308675	0.432019	0.444454
50%	12500.500000	67.995700	127.157750	0.518832	0.528935
75%	18750.250000	69.272958	134.892850	0.604702	0.612190
max	25000.000000	75.152800	170.924000	1.000000	1.000000

```
plt.figure(figsize=(10,10))
hw_df[['Height(Inches)','Height(Norm)','Weight(Pounds)','Weight(Norm)']].hist()
plt.show()
```

```
<Figure size 1000x1000 with 0 Axes>
```



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)
```

```
[[0.11785113 0.1767767  0.29462783 0.35355339 0.41247896 0.23570226
  0.47140452 0.41247896 0.35355339]]
```

```
#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)
```

```
Original DataFrame:
      Name  Age      City
0    Alice   25  New York
1      Bob   30    London
2  Charlie   22     Paris
3    David   28     Tokyo
```

```
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)
```

```
DataFrame with Age Group column:
      Name  Age      City Age_Group
0    Alice   25  New York     Young
1      Bob   30    London     Adult
2  Charlie   22     Paris     Young
3    David   28     Tokyo     Adult
```

```
# 2. Creating dummy variables for categorical features
df = pd.get_dummies(df, columns=['City'])
print("\nDataFrame after creating dummy variables for City:")
print(df)
```

```
DataFrame after creating dummy variables for City:
      Name  Age Age_Group  City_London  City_New York  City_Paris  City_Tokyo
0    Alice   25     Young        False           True       False       False
1      Bob   30     Adult         True          False       False       False
2  Charlie   22     Young        False          False        True       False
3    David   28     Adult        False          False       False        True
```

```
import pandas as pd
from sklearn import metrics


df = pd.read_csv("D:/131_Sakib_Tamboli/CreditRisk.csv')
print("DataFrame head:")
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
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())
```

```
DataFrame head:
2.0
```

```
import pandas as pd

df = pd.read_csv('D:/131_Sakib_Tamboli/CreditRisk.csv')
print("DataFrame head:")
df.head()
```

```
DataFrame head:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property
0	LP001002	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	

Name: Sakib Tamboli
Roll no :131 Div :B

Df.dtypes

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount             int64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status            int64
dtype: object
```

#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

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status
0	5849	0.0	0	360.0	1.0	1
1	4583	1508.0	128	360.0	1.0	0
2	3000	0.0	66	360.0	1.0	1
3	2583	2358.0	120	360.0	1.0	1
4	6000	0.0	141	360.0	1.0	1
...	...	...	...	...	...	...
609	2900	0.0	71	360.0	1.0	1
610	4106	0.0	40	180.0	1.0	1
611	8072	240.0	253	360.0	1.0	1
612	7583	0.0	187	360.0	1.0	1
613	4583	0.0	133	360.0	0.0	0

614 rows × 6 columns

df['ApplicantIncome'].mean()

```
5403.459283387622
```

```
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()
```

```
Categorical Columns:
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Property_Area'],
      dtype='object')
Loan_Status
1    422
0    192
Name: count, dtype: int64
```

```
pd.get_dummies(df,'Gender')
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	Gender_LP001002	Gender_LP001003	Gender_LP001005	Gend
0	5849	0.0	0	360.0	1.0	1	True	False	False	
1	4583	1508.0	128	360.0	1.0	0	False	True	False	
2	3000	0.0	66	360.0	1.0	1	False	False	True	
3	2583	2358.0	120	360.0	1.0	1	False	False	False	
4	6000	0.0	141	360.0	1.0	1	False	False	False	
...	...	...	...	...	...	...	...	...	...	
609	2900	0.0	71	360.0	1.0	1	False	False	False	
610	4106	0.0	40	180.0	1.0	1	False	False	False	
611	8072	240.0	253	360.0	1.0	1	False	False	False	
612	7583	0.0	187	360.0	1.0	1	False	False	False	
613	4583	0.0	133	360.0	0.0	0	False	False	False	

614 rows × 635 columns

```
correlation_matrix =num_cols.corr()
correlation_matrix
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status
ApplicantIncome	1.000000	-0.116605	0.538290	-0.045306	-0.014715	-0.004710
CoapplicantIncome	-0.116605	1.000000	0.190377	-0.059878	-0.002056	-0.059187
LoanAmount	0.538290	0.190377	1.000000	0.040539	-0.002197	-0.010631
Loan_Amount_Term	-0.045306	-0.059878	0.040539	1.000000	0.001470	-0.021268
Credit_History	-0.014715	-0.002056	-0.002197	0.001470	1.000000	0.561678
Loan_Status	-0.004710	-0.059187	-0.010631	-0.021268	0.561678	1.000000

Name: Sakib Tamboli
Roll no :131 Div :B

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
```

```
]:  (614, 13)
```

```
print("131 Sakib Tamboli")
credit_df.info()
```

```
131  Sakib Tamboli
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         614 non-null    int64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    int64
dtypes: float64(3), int64(3), object(7)
memory usage: 62.5+ KB
```

```
credit_df.head()
```

Name: Sakib Tamboli
Roll no :131 Div :B

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property
0	LP001002	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	

credit_df.tail()

8]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Prope
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71	360.0	1.0	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40	180.0	1.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253	360.0	1.0	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187	360.0	1.0	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133	360.0	0.0	S

credit_df.describe()

]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status
count	614.000000	614.000000	614.000000	600.00000	564.000000	614.000000
mean	5403.459283	1621.245798	141.166124	342.00000	0.842199	0.687296
std	6109.041673	2926.248369	88.340630	65.12041	0.364878	0.463973
min	150.000000	0.000000	0.000000	12.00000	0.000000	0.000000
25%	2877.500000	0.000000	98.000000	360.00000	1.000000	0.000000
50%	3812.500000	1188.500000	125.000000	360.00000	1.000000	1.000000
75%	5795.000000	2297.250000	164.750000	360.00000	1.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000	1.000000

credit_df.Loan_Status.value_counts()

```
1    422
0    192
Name: Loan_Status, dtype: int64
```

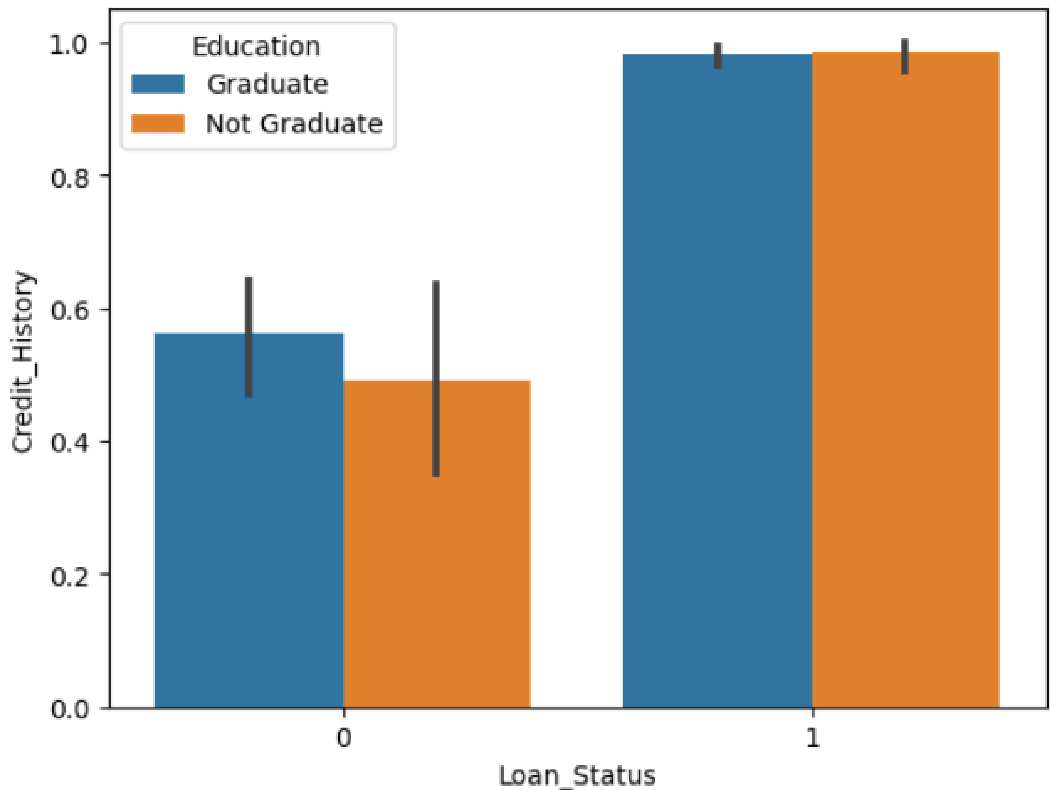
credit_df.groupby(['Education','Loan_Status']).Education.count()

Name: Sakib Tamboli
Roll no :131 Div :B

```
Education      Loan_Status
Graduate       0              140
               1              340
Not Graduate   0               52
               1               82
Name: Education, dtype: int64
```

sns.barplot(y='Credit_History',x='Loan_Status',hue='Education' ,data=credit_df)

```
<Axes: xlabel='Loan_Status', ylabel='Credit_History'>
```



#Fill Null Values

100 * credit_df.isnull().sum() / credit_df.shape[0]

```
]:  Loan_ID              0.000000
    Gender               2.117264
    Married              0.488599
    Dependents           2.442997
    Education            0.000000
    Self_Employed        5.211726
    ApplicantIncome      0.000000
    CoapplicantIncome    0.000000
    LoanAmount           0.000000
    Loan_Amount_Term     2.280130
    Credit_History       8.143322
    Property_Area        0.000000
    Loan_Status          0.000000
    dtype: float64
```

```
DF=credit_df.drop(credit_df.columns[0],axis=1)
DF.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Lo
0	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	Urban	
1	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	Rural	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	Urban	
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	Urban	
4	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	Urban	

```
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()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Lo
0	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	Urban	
1	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	Rural	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	Urban	
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	Urban	
4	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	Urban	

Categorical columns

DF[object_columns].Property_Area

```
0          Urban
1          Rural
2          Urban
3          Urban
4          Urban
         ...
609        Rural
610        Rural
611        Urban
612        Urban
613    Semiurban
Name: Property_Area, Length: 614, dtype: object
```

DF[object_columns].Property_Area.head()

```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

DF_dummy=pd.get_dummies(DF, columns=object_columns)
DF_dummy.shape

```
(614, 25)
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
DF_dummy.head()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	Gender_489	Gender_Female	Gender_Male	Married_398	...	De
0	5849	0.0	0	360.0	1.0	1	0	0	1	0	...	
1	4583	1508.0	128	360.0	1.0	0	0	0	1	0	...	
2	3000	0.0	66	360.0	1.0	1	0	0	1	0	...	
3	2583	2358.0	120	360.0	1.0	1	0	0	1	0	...	
4	6000	0.0	141	360.0	1.0	1	0	0	1	0	...	

5 rows × 25 columns

```
#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
```

```
((429, 24), (185, 24))
```

```
model = LogisticRegression()
model.fit(train_x, train_y)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
train_y_hat = model.predict(train_x)
test_y_hat = model.predict(test_x)
```

Name: Sakib Tamboli
Roll no :131 Div :B

print('train_accuracy', accuracy_score(train_y, train_y_hat))
print('test accuracy', accuracy_score(test_y, test_y_hat))

```
train_accuracy 0.703962703962704
test accuracy 0.6432432432432432
```

print(confusion_matrix(train_y, train_y_hat))

```
[[ 12 115]
 [ 12 290]]
```

print(confusion_matrix(test_y, test_y_hat))

```
[[  5  60]
 [  6 114]]
```

test_y.value_counts()

```
1    120
0     65
Name: Loan_Status, dtype: int64
```

pd.Series(test_y_hat).value_counts()

```
1    174
0     11
dtype: int64
```

Accuracy for train
(57 + 295) / train_y.shape[0]

```
0.8205128205128205
```

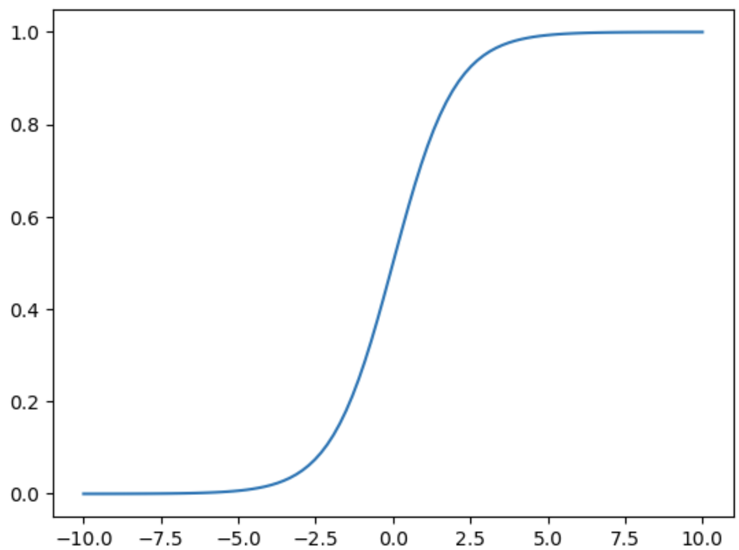
print(classification_report(test_y, test_y_hat))

Name: Sakib Tamboli
Roll no :131 Div :B

```
              precision    recall  f1-score   support

           0       0.45      0.08      0.13        65
           1       0.66      0.95      0.78       120

    accuracy                           0.64       185
   macro avg       0.55      0.51      0.45       185
weighted avg       0.58      0.64      0.55       185
```

```
x = np.linspace(-10, 10, 100)
y = 1 / ( 1 + np.exp(-x)) # sigmoid
plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x2dc05844f10>]
```




```
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))
```

```
[[  5  60]
 [  6 114]]
[[30 35]
 [48 72]]
[[  0  65]
 [  0 120]]
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
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))
```

```
              precision    recall  f1-score   support

           0       0.45      0.08      0.13        65
           1       0.66      0.95      0.78       120

    accuracy                           0.64       185
   macro avg       0.55      0.51      0.45       185
weighted avg       0.58      0.64      0.55       185

              precision    recall  f1-score   support

           0       0.38      0.46      0.42        65
           1       0.67      0.60      0.63       120

    accuracy                           0.55       185
   macro avg       0.53      0.53      0.53       185
weighted avg       0.57      0.55      0.56       185

              precision    recall  f1-score   support

           0       0.00      0.00      0.00        65
           1       0.65      1.00      0.79       120

    accuracy                           0.65       185
   macro avg       0.32      0.50      0.39       185
weighted avg       0.42      0.65      0.51       185
```

Name: Sakib Tamboli
Roll no :131 Div :B

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

```
]:  (614, 13)
```

```
print("131 Sakib Tamboli")
credit_df.info()
```

```
131  Sakib Tamboli
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         614 non-null    int64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    int64
dtypes: float64(3), int64(3), object(7)
memory usage: 62.5+ KB
```

credit_df.head()

Name: Sakib Tamboli
Roll no :131 Div :B

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property
0	LP001002	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	

credit_df.tail()

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Prope
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71	360.0	1.0	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40	180.0	1.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253	360.0	1.0	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187	360.0	1.0	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133	360.0	0.0	S

credit_df.describe()

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status
count	614.000000	614.000000	614.000000	600.00000	564.000000	614.000000
mean	5403.459283	1621.245798	141.166124	342.00000	0.842199	0.687296
std	6109.041673	2926.248369	88.340630	65.12041	0.364878	0.463973
min	150.000000	0.000000	0.000000	12.00000	0.000000	0.000000
25%	2877.500000	0.000000	98.000000	360.00000	1.000000	0.000000
50%	3812.500000	1188.500000	125.000000	360.00000	1.000000	1.000000
75%	5795.000000	2297.250000	164.750000	360.00000	1.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000	1.000000

credit_df.Loan_Status.value_counts()

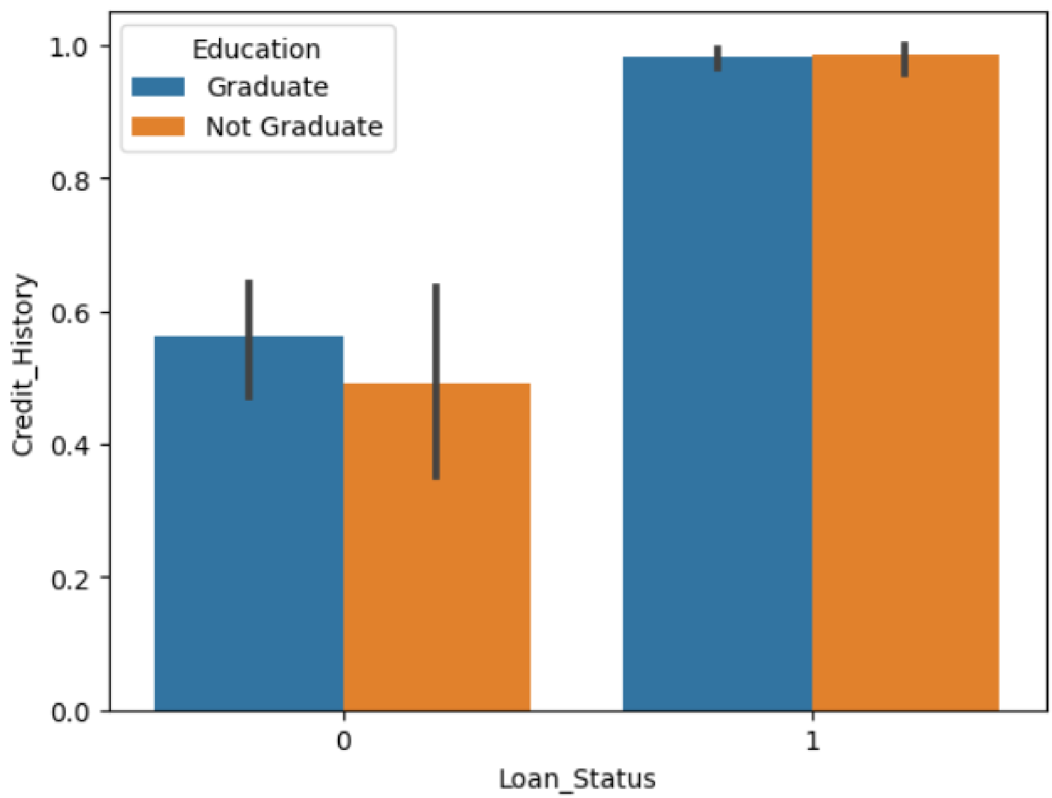
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

credit_df.groupby(['Education','Loan_Status']).Education.count()

```
]:  Education     Loan_Status
    Graduate      0              140
                  1              340
    Not Graduate  0               52
                  1               82
    Name: Education, dtype: int64
```

sns.barplot(y='Credit_History',x='Loan_Status',hue='Education' ,data=credit_df)

```
]:  <Axes: xlabel='Loan_Status', ylabel='Credit_History'>
```



#Fill Null Values

100 * credit_df.isnull().sum() / credit_df.shape[0]

```
]:  Loan_ID              0.000000
    Gender               2.117264
    Married              0.488599
    Dependents           2.442997
    Education            0.000000
    Self_Employed        5.211726
    ApplicantIncome      0.000000
    CoapplicantIncome    0.000000
    LoanAmount           0.000000
    Loan_Amount_Term     2.280130
    Credit_History       8.143322
    Property_Area        0.000000
    Loan_Status          0.000000
    dtype: float64
```

```
DF=credit_df.drop(credit_df.columns[0],axis=1)
DF.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Lo
0	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	Urban	
1	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	Rural	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	Urban	
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	Urban	
4	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	Urban	

```
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()
```

Name: Sakib Tamboli
Roll no :131 Div :B

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Lo
0	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	Urban	
1	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	Rural	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	Urban	
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	Urban	
4	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	Urban	

Categorical columns

DF[object_columns].Property_Area

```
0          Urban
1          Rural
2          Urban
3          Urban
4          Urban
         ...
609        Rural
610        Rural
611        Urban
612        Urban
613    Semiurban
Name: Property_Area, Length: 614, dtype: object
```

DF[object_columns].Property_Area.head()

```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

DF_dummy=pd.get_dummies(DF, columns=object_columns)
DF_dummy.shape

```
(614, 25)
```

Name: Sakib Tamboli
Roll no :131 Div :B

DF_dummy.head()

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	Gender_489	Gender_Female	Gender_Male	Married_398	...	De
0	5849	0.0	0	360.0	1.0	1	0	0	1	0	...	
1	4583	1508.0	128	360.0	1.0	0	0	0	1	0	...	
2	3000	0.0	66	360.0	1.0	1	0	0	1	0	...	
3	2583	2358.0	120	360.0	1.0	1	0	0	1	0	...	
4	6000	0.0	141	360.0	1.0	1	0	0	1	0	...	

5 rows × 25 columns

```
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
```

```
((429, 24), (185, 24))
```

```
svm_model = SVC(kernel='rbf', gamma=0.00001, C=1000)
svm_model.fit(train_x, train_y)
```

```
SVC
SVC(C=1000, gamma=1e-05)
```

```
train_y_hat = svm_model.predict(train_x)
test_y_hat = svm_model.predict(test_x)
print('-'*20, 'Train', '-'*20)
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
print(classification_report(train_y, train_y_hat))
print('-'*20, 'Test', '-'*20)
print(classification_report(test_y, test_y_hat))
```

```
-------------------- Train --------------------
              precision    recall  f1-score   support

           0       0.97      0.94      0.96       127
           1       0.98      0.99      0.98       302

    accuracy                           0.97       429
   macro avg       0.97      0.97      0.97       429
weighted avg       0.97      0.97      0.97       429

-------------------- Test --------------------
              precision    recall  f1-score   support

           0       0.33      0.20      0.25        65
           1       0.64      0.78      0.71       120

    accuracy                           0.58       185
   macro avg       0.49      0.49      0.48       185
weighted avg       0.53      0.58      0.55       185
```

```
confusion_matrix(test_y, test_y_hat)
```

```
array([[13, 52],
       [26, 94]], dtype=int64)
```

Name: Sakib Tamboli
Roll no :131 Div :B

Practical 14

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

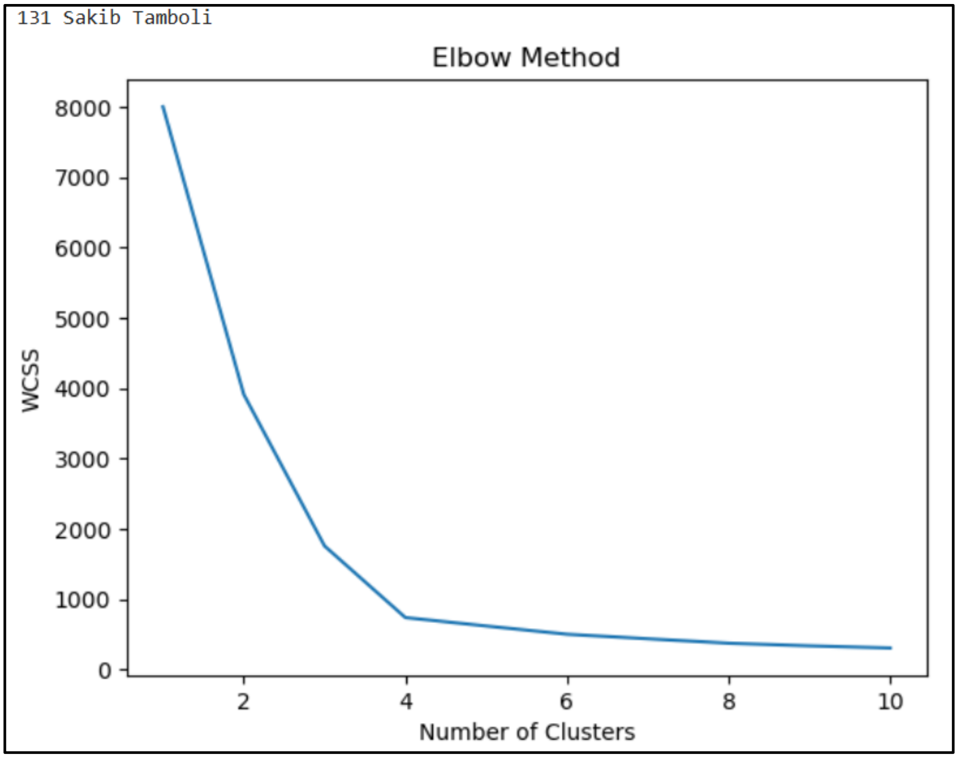
Elbow

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()
```

Name: Sakib Tamboli
Roll no :131 Div :B

Output:



Name: Sakib Tamboli
Roll no :131 Div :B

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
```

```
131 Sakib Tamboli
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

```
print(df.shape)
```

```
(891, 12)
```

```
#checking for missing data
NAs=pd.concat([df.isnull().sum()],axis=1,keys=["Train"])
NAs[NAs.sum(axis=1)>0]
```

Name: Sakib Tamboli
Roll no :131 Div :B

	Train
Age	177
Cabin	687
Embarked	2

```
df.pop("Cabin")
df.pop("Name")
df.pop("Ticket")
```

```
0              A/5 21171
1               PC 17599
2       STON/O2. 3101282
3                 113803
4                 373450
             ...
886               211536
887               112053
888           W./C. 6607
889               111369
890               370376
Name: Ticket, Length: 891, dtype: object
```

```
#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()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	1	0	3	22.0	1	0	7.2500	0	1	0	0	1
1	2	1	1	38.0	1	0	71.2833	1	0	1	0	0
2	3	1	3	26.0	0	0	7.9250	1	0	0	0	1
3	4	1	1	35.0	1	0	53.1000	1	0	0	0	1
4	5	0	3	35.0	0	0	8.0500	0	1	0	0	1

```
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)
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
RandomForestClassifier
RandomForestClassifier(n_estimators=10)
```

```
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)
```

```
0.8161434977578476
```

```
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)
```

```
131 Sakib Tamboli
Bagging Accuracy : 0.820627802690583
```

```
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)
```

```
131 Sakib Tamboli
Voting Accuracy : 0.820627802690583
```

Name: Sakib Tamboli
Roll no :131 Div :B

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
```

```
(150, 4)
```

```
Y.shape
```

```
(150,)
```

```
#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)
```

```
Bagging Accuracy : 1.0
```

```
#2. Boosting (using Adaboost with decision tree)
boosting_model=AdaBoostClassifier(estimator=DecisionTreeClassifier(),n_estimators=50,rando
m_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)
```

```
boosting Accuracy : 1.0
```

```
# 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)
```

```
Stacking Accuracy : 1.0
```

```
# 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)
```

```
Voting Accuracy : 1.0
```

Name: Sakib Tamboli
Roll no :131 Div :B

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
```

131 Sakib Tamboli

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

```
print("131 Sakib Tamboli")
print(df.shape)
```

```
131 Sakib Tamboli
(891, 12)
```

```
#checking for missing data
NAs=pd.concat([df.isnull().sum()],axis=1,keys=["Train"])
NAs[NAs.sum(axis=1)>0]
```

Name: Sakib Tamboli
Roll no :131 Div :B

	Train
Age	177
Cabin	687
Embarked	2

```
df.pop("Cabin")
df.pop("Name")
df.pop("Ticket")
```

```
0             A/5 21171
1              PC 17599
2      STON/O2. 3101282
3                113803
4                373450
             ...
886              211536
887              112053
888          W./C. 6607
889              111369
890              370376
Name: Ticket, Length: 891, dtype: object
```

```
#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()
```

Name: Sakib Tamboli
Roll no :131 Div :B

	PassengerId	Survived	Age	SibSp	Parch	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	1	0	22.0	1	0	7.2500	0	0	1	0	1	0	0	1
1	2	1	38.0	1	0	71.2833	1	0	0	1	0	1	0	0
2	3	1	26.0	0	0	7.9250	0	0	1	1	0	0	0	1
3	4	1	35.0	1	0	53.1000	1	0	0	1	0	0	0	1
4	5	0	35.0	0	0	8.0500	0	0	1	0	1	0	0	1

```
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)
```

```
RandomForestClassifier
RandomForestClassifier()
```

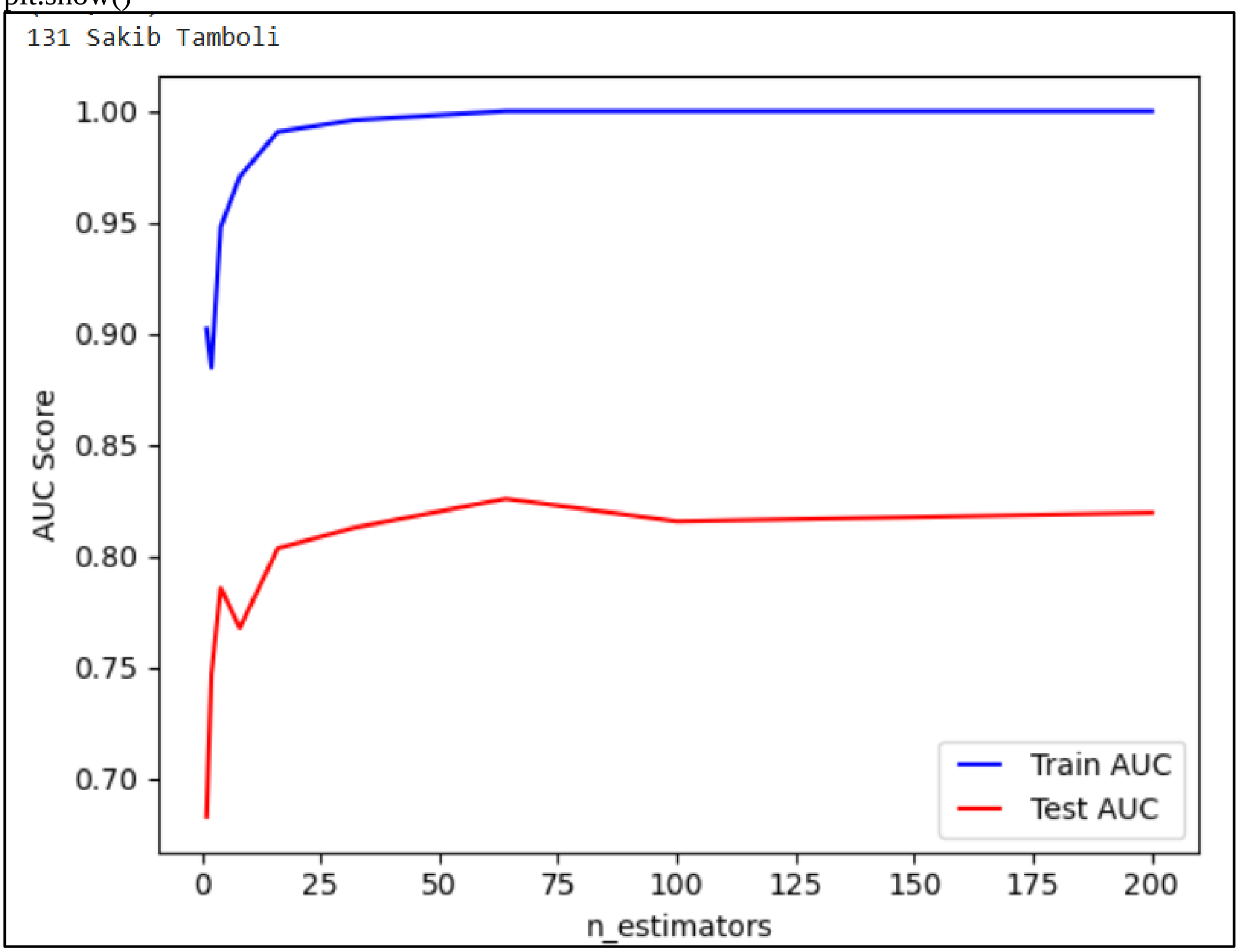
```
y_pred=rf.predict(x_test)

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
```

```
0.8036425489545304
```

```
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)
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
    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)

```
RandomForestClassifier
RandomForestClassifier(n_estimators=200)
```

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

print(df.shape)

```
131 Sakib Tamboli
(891, 12)
```

#checking for missing data
NAs=pd.concat([df.isnull().sum()],axis=1,keys=["Train"])
NAs[NAs.sum(axis=1)>0]

Name: Sakib Tamboli
Roll no :131 Div :B

	Train
Age	177
Cabin	687
Embarked	2

```
df.pop("Cabin")
df.pop("Name")
df.pop("Ticket")
```

```
0            A/5 21171
1             PC 17599
2     STON/O2. 3101282
3               113803
4               373450
            ...
886             211536
887             112053
888         W./C. 6607
889             111369
890             370376
Name: Ticket, Length: 891, dtype: object
```

```
#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()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	1	0	3	22.0	1	0	7.2500	0	1	0	0	1
1	2	1	1	38.0	1	0	71.2833	1	0	1	0	0
2	3	1	3	26.0	0	0	7.9250	1	0	0	0	1
3	4	1	1	35.0	1	0	53.1000	1	0	0	0	1
4	5	0	3	35.0	0	0	8.0500	0	1	0	0	1

```
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
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
rf=RandomForestClassifier(n_estimators=10)
rf.fit(x_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=10)
```

```
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)
```

```
131 Sakib Tamboli
0.8430493273542601
```

```
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)
```

```
131 Sakib Tamboli
Bagging Accuracy:  0.8834080717488789
```

Name: Sakib Tamboli
Roll no :131 Div :B

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
```

131 Sakib Tamboli

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

```
print(df.shape)
```

```
(891, 12)
```

```
#checking for missing data
NAs=pd.concat([df.isnull().sum()],axis=1,keys=["Train"])
NAs[NAs.sum(axis=1)>0]
```

Name: Sakib Tamboli
Roll no :131 Div :B

	Train
Age	177
Cabin	687
Embarked	2

```
df.pop("Cabin")
df.pop("Name")
df.pop("Ticket")
```

```
0             A/5 21171
1              PC 17599
2      STON/O2. 3101282
3                113803
4                373450
             ...
886              211536
887              112053
888          W./C. 6607
889              111369
890              370376
Name: Ticket, Length: 891, dtype: object
```

```
#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()
```

	PassengerId	Survived	Age	SibSp	Parch	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	1	0	22.0	1	0	7.2500	0	0	1	0	1	0	0	1
1	2	1	38.0	1	0	71.2833	1	0	0	1	0	1	0	0
2	3	1	26.0	0	0	7.9250	0	0	1	1	0	0	0	1
3	4	1	35.0	1	0	53.1000	1	0	0	1	0	0	0	1
4	5	0	35.0	0	0	8.0500	0	0	1	0	1	0	0	1

```
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)
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
df_model=DecisionTreeClassifier(max_depth=14)
df_model.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=14)
```

```
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
```

```
0.7668161434977578
```

```
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)
```

```
boosting Accuracy : 0.757847533632287
```

```
from sklearn.metrics import classification_report
print("\n Classification Report:\n" ,classification_report(y_test,y_pred))
```

```
 Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.79      0.80       136
           1       0.69      0.74      0.71        87

    accuracy                           0.77       223
   macro avg       0.76      0.76      0.76       223
weighted avg       0.77      0.77      0.77       223
```

Name: Sakib Tamboli
Roll no :131 Div :B

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
```

131 Sakib Tamboli

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

```
print(df.shape)
print("131 Sakib Tamboli")
```

```
(891, 12)
131 Sakib Tamboli
```

```
#checking for missing data
NAs=pd.concat([df.isnull().sum()],axis=1,keys=["Train"])
NAs[NAs.sum(axis=1)>0]
```

Name: Sakib Tamboli
Roll no :131 Div :B

	Train
Age	177
Cabin	687
Embarked	2

```
df.pop("Cabin")
df.pop("Name")
df.pop("Ticket")
```

```
0             A/5 21171
1              PC 17599
2      STON/O2. 3101282
3                113803
4                373450
             ...
886              211536
887              112053
888          W./C. 6607
889              111369
890              370376
Name: Ticket, Length: 891, dtype: object
```

```
#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()
```

	PassengerId	Survived	Age	SibSp	Parch	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	1	0	22.0	1	0	7.2500	0	0	1	0	1	0	0	1
1	2	1	38.0	1	0	71.2833	1	0	0	1	0	1	0	0
2	3	1	26.0	0	0	7.9250	0	0	1	1	0	0	0	1
3	4	1	35.0	1	0	53.1000	1	0	0	1	0	0	0	1
4	5	0	35.0	0	0	8.0500	0	0	1	0	1	0	0	1

```
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)
```

Name: Sakib Tamboli
Roll no :131 Div :B

```
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)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42, subsample=0.8)
```

```
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))
```

```
131 Sakib Tamboli

 Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.87      0.87       139
           1       0.79      0.80      0.79        84

    accuracy                           0.84       223
   macro avg       0.83      0.83      0.83       223
weighted avg       0.84      0.84      0.84       223
```