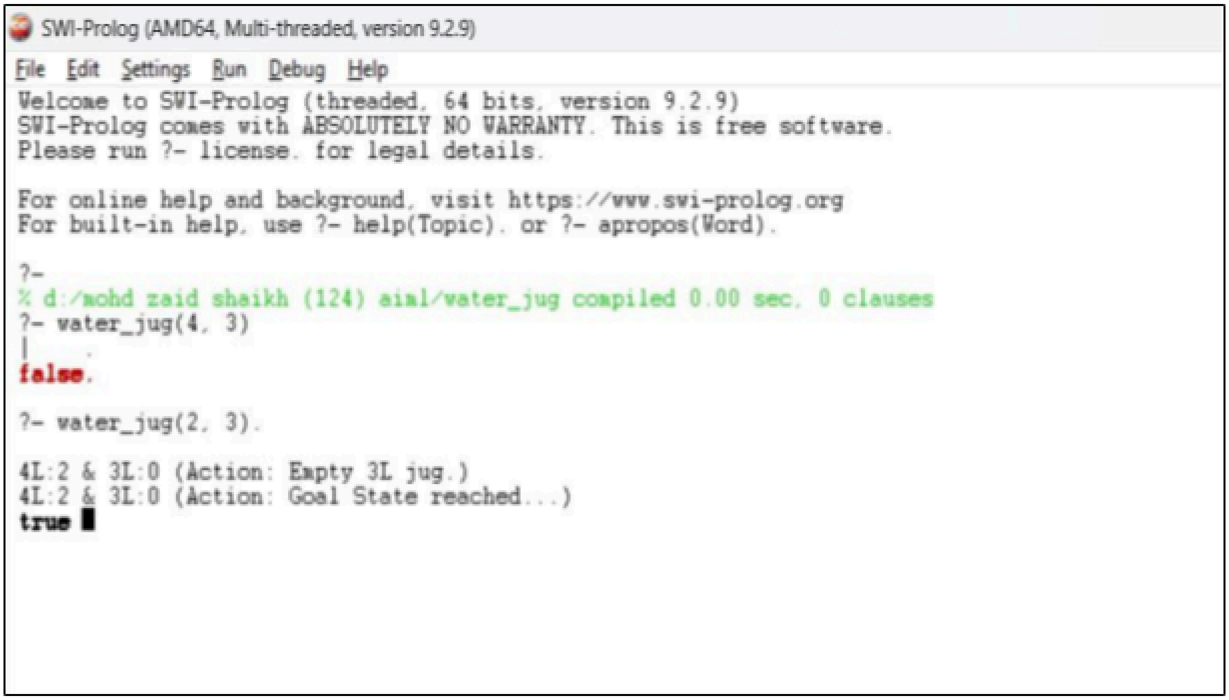
Practical 1

Aim: Implementation of logic programming using Prolog (DFS) – Water Jug Problem

Code: water_jug.pl

```
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 until 4L jug
is full.)'),
    XX is X + 1, YY is Y - 1, water_jug(XX, YY));

   (X =:= 1, Y =:= 0, nl, write('4L:0 & 3L:1 (Action: Pour water from 4L jug to 3L jug.)'),
    XX is Y, YY is X, water_jug(XX, YY));

   (X =:= 0, Y =:= 1, nl, write('4L:4 & 3L:1 (Action: Fill 4L jug.)'),
    XX is 4, water_jug(XX, Y));
```

Output:


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-
% d:/mohd zaid shaikh (124) aiml/water_jug compiled 0.00 sec, 0 clauses
?- water_jug(4, 3)
false.
?- water_jug(2, 3).
4L:2 & 3L:0 (Action: Empty 3L jug.)
4L:2 & 3L:0 (Action: Goal State reached...)
true

Practical 2

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

Code: tic-tac-toe.pl

```
% 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:

```
SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File  Edit  Settings  Run  Debug  Help
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% d:/mohd zaid shaikh (124) aiml/tic-tac-toe compiled 0.00 sec, -3 clauses
?- play.
   |   |
---+---+---
   |   |
---+---+---
   |   |

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

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

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

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

X's turn. Enter position (1-9): |: 4
|: .
X  | X | O
---+---+---
X  | O |
---+---+---
   |   |

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

O wins!
true .

?-
```

Practical 3

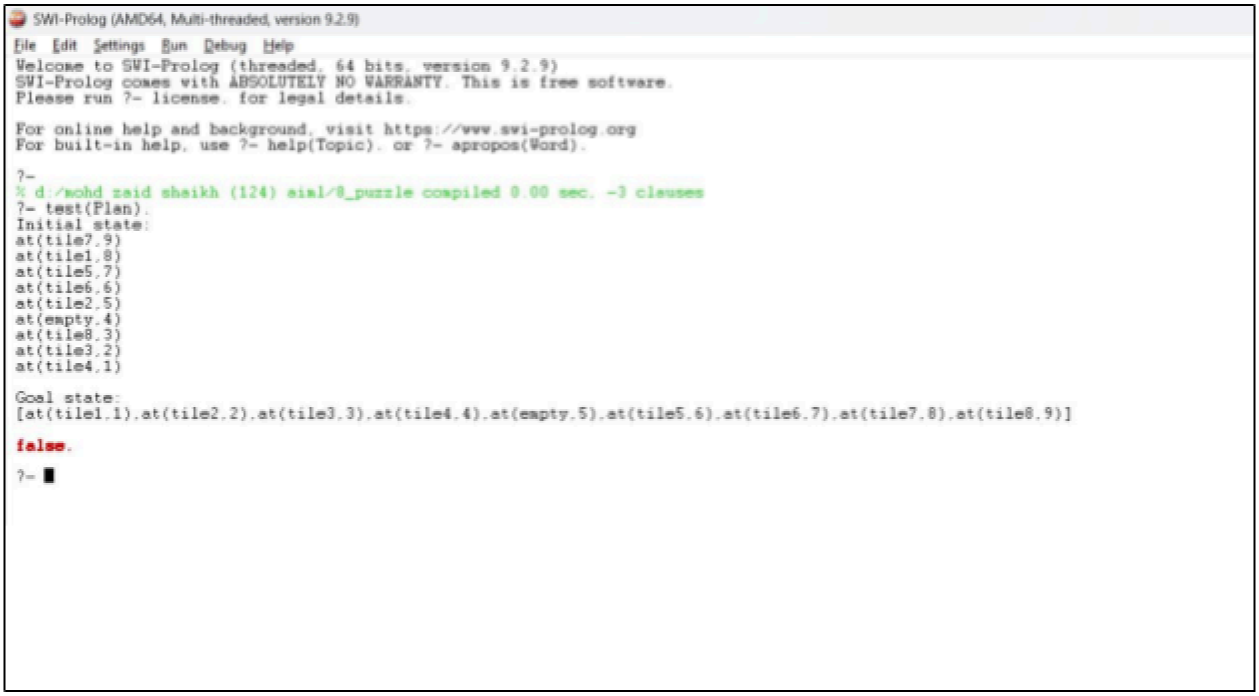
Aim: Implementation of logic programming using Prolog (Hill Climbing) – 8 Puzzle

Code: 8_puzzle.pl

```
% 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:



```
SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File  Edit  Settings  Run  Debug  Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% d:/mohd zaid shaikh (124) aiml/8_puzzle 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.

?-
```

Practical 4

Aim: Intro to Python Libraries – Basic Libraries, NumPy, Pandas

#numpy

```
[3]
x=np.array([2,3,4,5])
print(type(x))
```

```
... <class 'numpy.ndarray'>
```

```
[5]
print(x)
```

```
... [2 3 4 5]
```

```
[7]
x=np.array([2,3,'n',5])
print(x)
```

```
... ['2' '3' 'n' '5']
```

```
[9]
d=np.arange(start=1, stop=10, step=2)
print(d)
```

```
... [1 3 5 7 9]
```

```
grid = np.arange(start =1, stop=10).reshape(3,3)
print(grid)
```
[23]

```
... [[1 2 3]
     [4 5 6]
     [7 8 9]]
```

```
np.ones((3,4))
```
[11]

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

```
np.random.rand(5)
```
[13]

```
... array([0.85984462, 0.14581783, 0.28219656, 0.79902342, 0.60618392])
```

```
np.random.rand(5,4)
```
[15]

```
... array([[0.95404251, 0.62611004, 0.692988  , 0.53180799],
           [0.32360261, 0.58446612, 0.51718914, 0.32049731],
           [0.52852974, 0.7351838 , 0.63884682, 0.45631519],
           [0.04576068, 0.10476025, 0.04341695, 0.88704631],
           [0.77235022, 0.59780543, 0.80025621, 0.62047803]])
```

```
np.logspace(1,10, num=5,endpoint=True, base=10.0)
```
[19]

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

```
[25]  a= np.array([[1,2,3],[4,5,6],[7,8,9]])
      a.shape

...   (3, 3)
```

```
[45]  a[:,0]

...   array([1, 4, 7])

[47]  a[0,:]

...   array([1, 2, 3])
```

```
[41]  a[0,1]

...   2

[43]  a[1:3]

...   array([[4, 5, 6],
             [7, 8, 9]])
```

```
[51]
a_sub=a[:2,:2]
print(a_sub)
```

```
[[1 2]
 [4 5]]
```

```
[57]
a_sub[0,1]=10
print(a_sub)
```

```
[[ 1 10]
 [ 4  5]]
```

```
[93]
s_col=np.append(s_row,[[17],[18],[19],[20]],axis=1)
print(s_col)
```

```
[[ 1 10  3 17]
 [ 4  5  6 18]
 [ 7  8  9 19]
 [10 11 14 20]]
```

```
[97]
a_del=np.delete(s_col,1,axis=0)
print(a_del)
```

```
[[ 1 10  3 17]
 [ 7  8  9 19]
 [10 11 14 20]]
```

```
[59]
print(a)
```

```
[[ 1 10  3]
 [ 4  5  6]
 [ 7  8  9]]
```

```
[81]
s_row=np.append(a,[[10,11,14]],axis=0)
print(s_row)
```

```
[[ 1 10  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 14]]
```

#Panda

import pandas as pd

```
[19]
data1 = pd.read_csv('C:/Users/lab-2/mtcars.csv')
print(data1)
```

```
...
                   model   mpg  cyl   disp   hp  drat     wt   qsec  vs  am
0              Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1
1          Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1
2             Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1
3         Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0
4      Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0
5                Valiant  18.1    6  225.0  105  2.76  3.460  20.22   1   0
6             Duster 360  14.3    8  360.0  245  3.21  3.570  15.84   0   0
7              Merc 240D  24.4    4  146.7   62  3.69  3.190  20.00   1   0
8               Merc 230  22.8    4  140.8   95  3.92  3.150  22.90   1   0
9               Merc 280  19.2    6  167.6  123  3.92  3.440  18.30   1   0
10             Merc 280C  17.8    6  167.6  123  3.92  3.440  18.90   1   0
11            Merc 450SE  16.4    8  275.8  180  3.07  4.070  17.40   0   0
12            Merc 450SL  17.3    8  275.8  180  3.07  3.730  17.60   0   0
13           Merc 450SLC  15.2    8  275.8  180  3.07  3.780  18.00   0   0
14    Cadillac Fleetwood  10.4    8  472.0  205  2.93  5.250  17.98   0   0
15   Lincoln Continental  10.4    8  460.0  215  3.00  5.424  17.82   0   0
```

```
[21]
data1.info()
```

```
...  <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
```

[27]

```
data1.head()
```

	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

[33]

```
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

```
[47] data1.iat[8,4]
...  95
```

```
[37] data1.shape
...  (32, 12)
```

```
[39] data1.size
...  384
```

```
[41]  data1.ndim

...   2
```

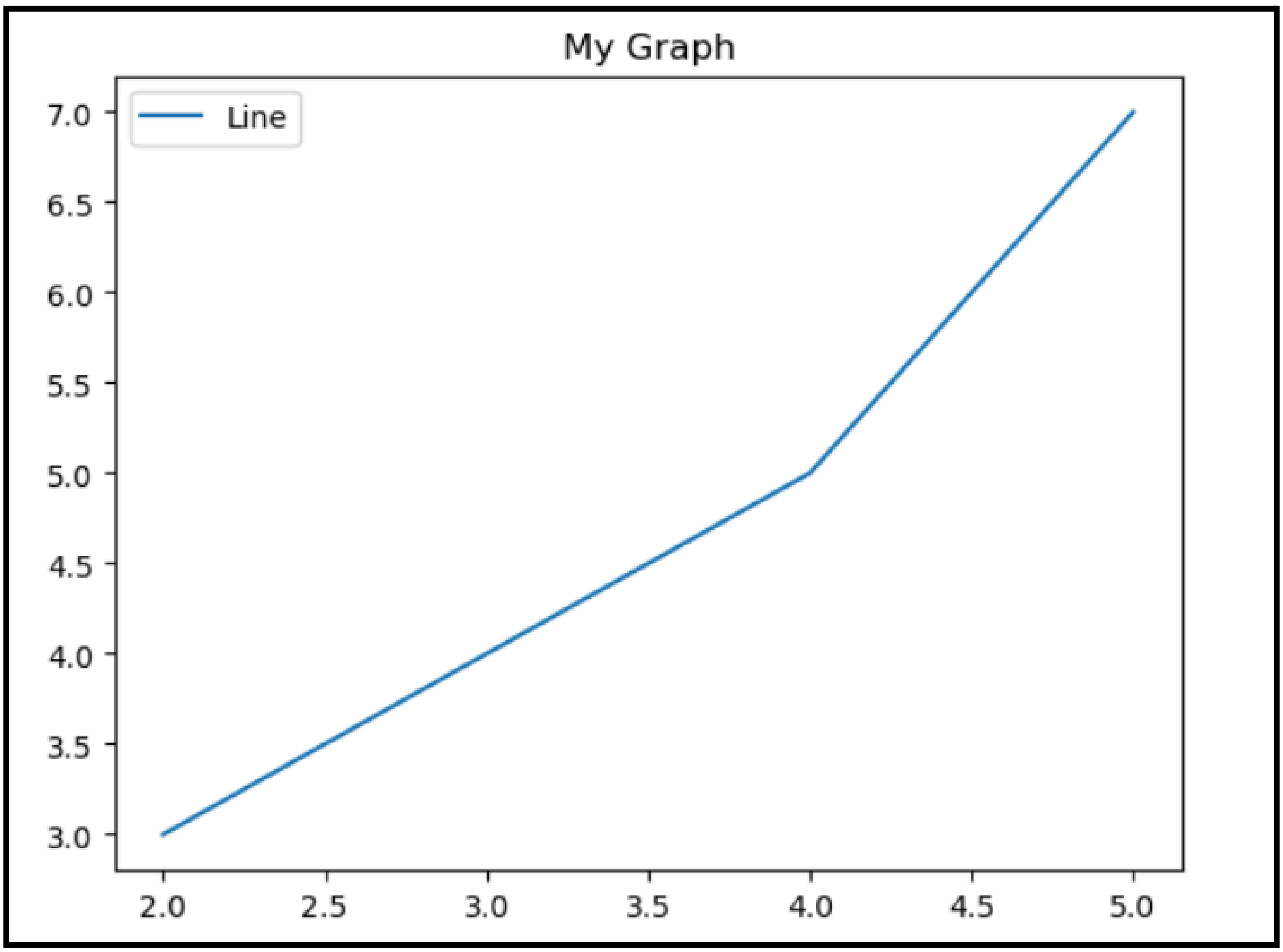
```
[45]  data1.at[8,"model"]

...   'Merc 230'
```

```
[49]  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
      ...
      28          Ford Pantera L
      29            Ferrari Dino
      30           Maserati Bora
      31              Volvo 142E
      Name: model, dtype: object
```

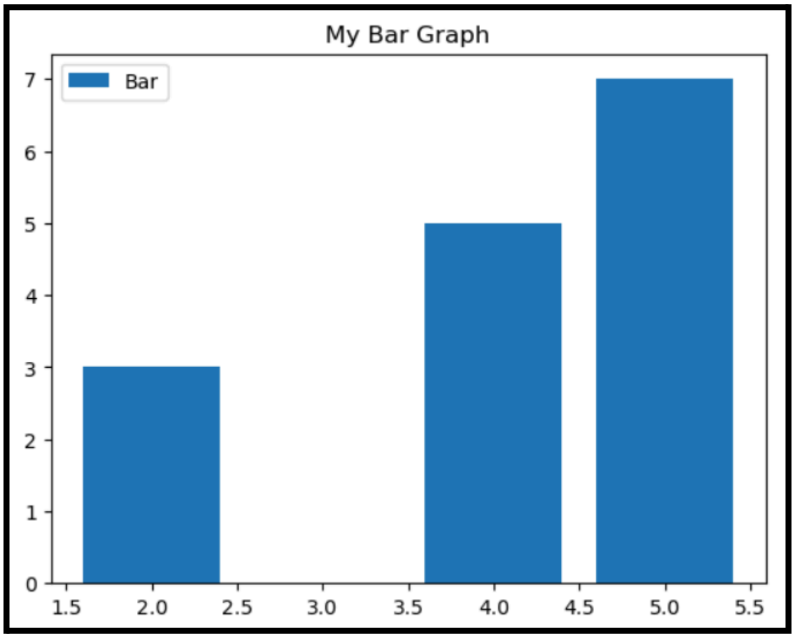
Practical 5

Aim: Intro to Python Libraries – Matplotlib, SciPy

```
import matplotlib.pyplot as plp
import pandas as pd

cars = pd.read_csv("C:/Users/USER/Downloads/mtcars.csv")
x = [2,4,5]
y = [3,5,7]
plp.plot(x,y)
plp.title("My Graph")
plp.legend(['Line'])
plp.show()
```

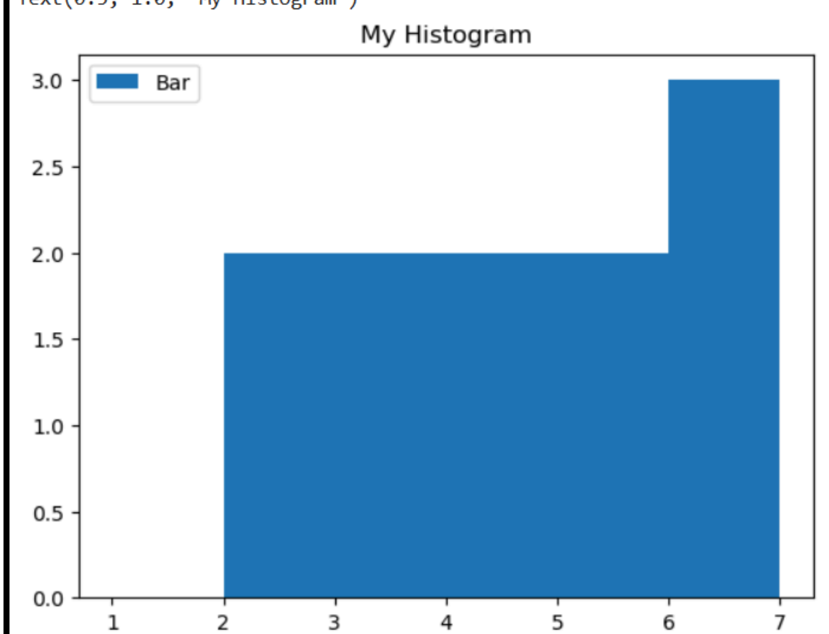


```
plp.bar(x,y)
plp.title("My Bar Graph")
plp.legend(["Bar"])
plp.show()
```

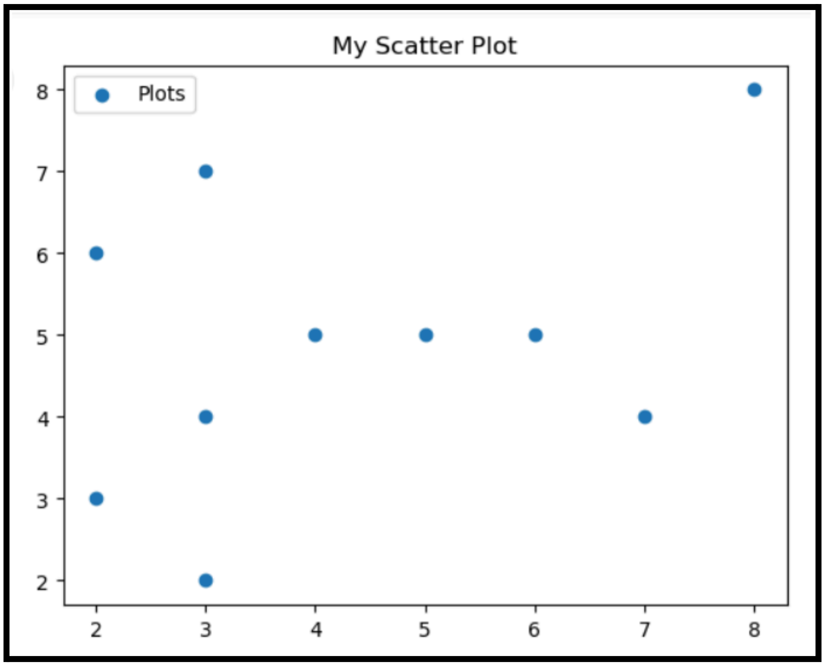



```
x = [2,3,4,5,2,3,6,7,8,4,5,6,0]

plp.hist(x, bins = [1,2,3,4,5,6,7])
plp.legend(['Bar'])
plp.title("My Histogram")
```

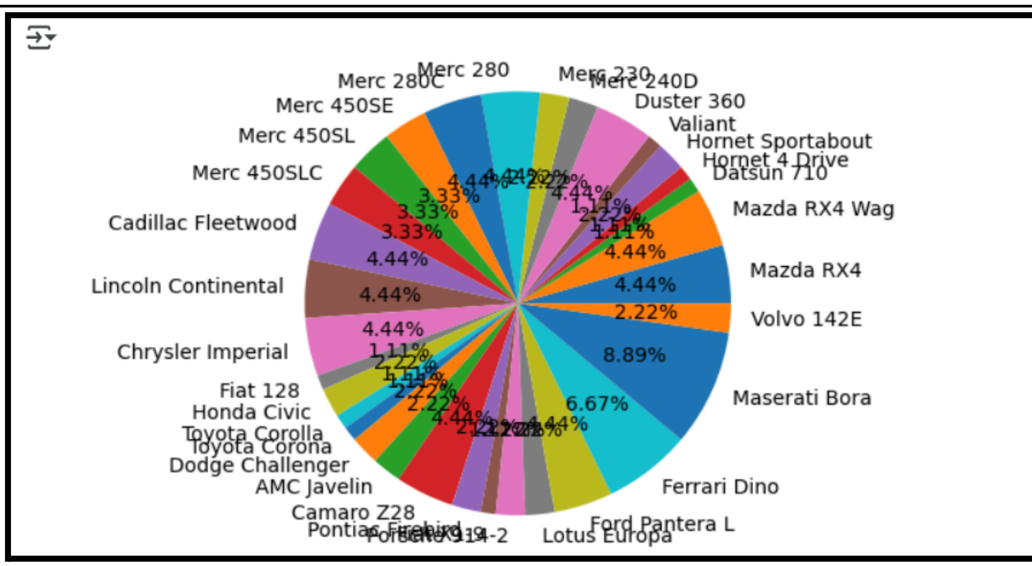
```
Text(0.5, 1.0, 'My Histogram')
```



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

plp.scatter(x,y)
plp.title("My Scatter Plot")
plp.legend(["Plots"])
plp.show()
```

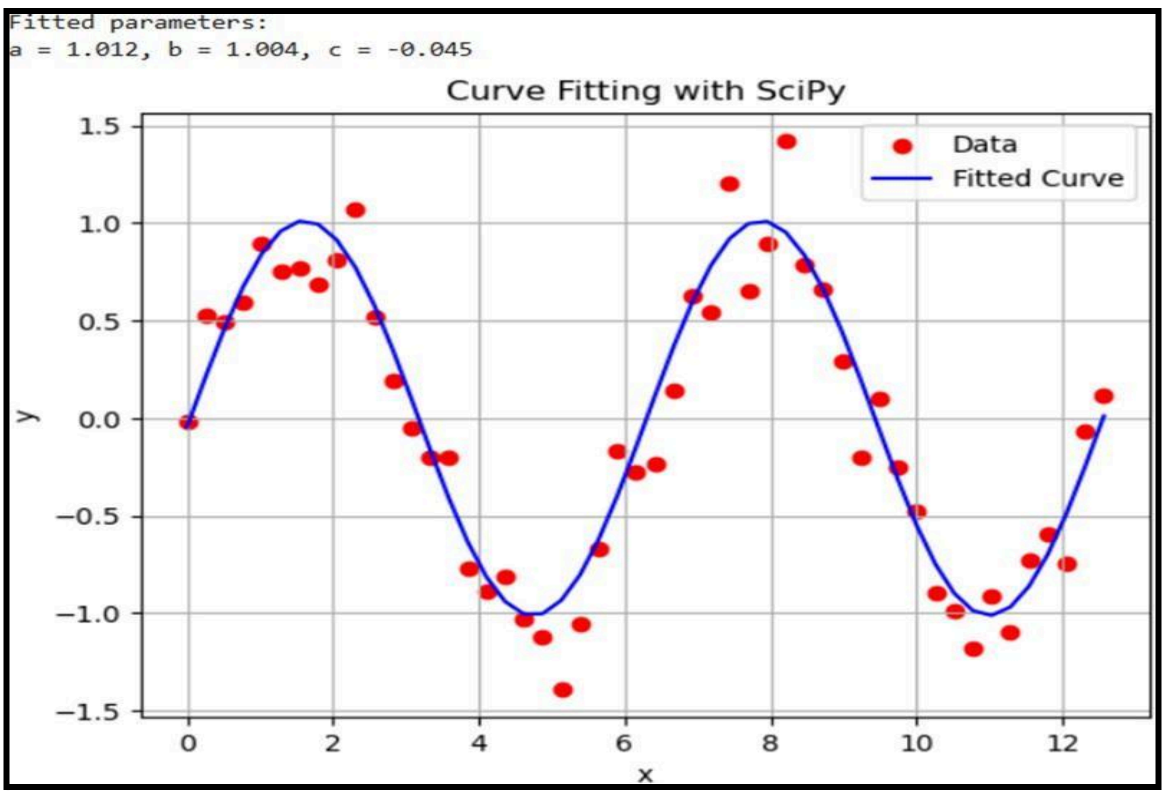



```
labels = ['Mango', 'Banana', 'Watermelon', 'Strawberry']
sizes = [10,30,20,40]
plp.pie(cars['carb'],labels=cars['model'],autopct='%1.2f%%')
plp.show()
```



```
#scipy

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
# Sample data (e.g., generated from a noisy sine wave) x_data = np.linspace(0, 4 * np.pi, 50)
y_data = np.sin(x_data) + 0.2 * np.random.normal(size=len(x_data)) # Define the model function to
fit
def model_func(x, a, b, c): return a * np.sin(b * x + c)
# Fit the model to the data
params, params_covariance = curve_fit(model_func, x_data, y_data, p0=[1, 1, 0]) # Print the
optimized parameters
print("Fitted parameters:")
print(f"a = {params[0]:.3f}, b = {params[1]:.3f}, c = {params[2]:.3f}")


# Plotting the data and the fitted curve plt.scatter(x_data, y_data, label='Data', color='red')
plt.plot(x_data, model_func(x_data, *params), label='Fitted Curve', color='blue') plt.legend()
plt.title('Curve Fitting with SciPy') plt.xlabel('x')
plt.ylabel('y') plt.grid(True) plt.show()
```

Output:



Practical 6

Aim: Intro to Python Libraries – Exploratory Data Analysis

Code:

```
import pandas as pd
from sklearn import metrics

df = pd.read_csv('C:/adeela/CreditRisk.csv')
print("DataFrame head:")
df.head()
```

Output

DataFrame head:

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

```
df.dtypes
num_cols = df.select_dtypes(include=np.number)
num_cols
```

Output:

```
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
```

	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
obj_cols

Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Property_Area'],
       dtype='object')
X = df.drop('Loan_Status', axis=1)
y =df['Loan_Status']
y.value_counts()
1     422
0     192
Name: Loan_Status, dtype: int64
pd.get_dummies(df,'Gender')
```

Output:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	Gender_LP001002	Gender_LP001003	Gender_LP001005	Gender_LP00
0	5849	0.0	0	360.0	1.0	1	1	0	0	
1	4583	1508.0	128	360.0	1.0	0	0	1	0	
2	3000	0.0	66	360.0	1.0	1	0	0	1	
3	2583	2358.0	120	360.0	1.0	1	0	0	0	
4	6000	0.0	141	360.0	1.0	1	0	0	0	
...	...	...	...	...	...	...	...	...	...	
609	2900	0.0	71	360.0	1.0	1	0	0	0	
610	4106	0.0	40	180.0	1.0	1	0	0	0	
611	8072	240.0	253	360.0	1.0	1	0	0	0	
612	7583	0.0	187	360.0	1.0	1	0	0	0	
613	4583	0.0	133	360.0	0.0	0	0	0	0	

614 rows × 635 columns

```
correlation_matrix =num_cols.corr()

correlation_matrix
```

Output:

	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

```
#Fill Null Values
100 * credit_df.isnull().sum() / credit_df.shape[0]
```

Output:

```
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()
```

Output:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Statu
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
#credit_df.columns[credit_df.dtypes == object]
#credit_df.columns[credit_df.dtypes == object]
for column in object_columns:
   majority = DF[column].value_counts().iloc[0]
   DF[column].fillna(majority, inplace=True)
for column in numeric_columns:
         mean = DF[column].mean()
   DF[column].fillna(mean, inplace=True)
# Impute
DF.head()
```

Output:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Statu
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	

Practical 7

Aim: Implementation of Perceptron Algorithm

Code: perceptron.ipynb

```
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("Output: ")
print(perceptron_or(0, 0))
print(perceptron_or(0, 1))
print(perceptron_or(1, 0))
print(perceptron_or(1, 1))
```

```
Output:
0
1
1
1
```

Practical 8

Aim: Implementation of Adaline for AND Operations

Code: adaline_and_operation.ipynb

```
import numpy as np

class Adaline:
def   init  (self, learning_rate=0.01, n_iter=100): self.learning_rate = learning_rate
self.n_iter = n_iter self.weights = None self.bias = None

def fit(self, X, y):
# Initialize weights and bias
self.weights = np.zeros(X.shape[1]) self.bias = 0

# Perform gradient descent
for _ in range(self.n_iter):
# Calculate net input (weighted sum of inputs) net_input = np.dot(X, self.weights) + self.bias

# Calculate error (difference between prediction and actual) error = y - net_input

# Update weights and bias using gradient descent
self.weights += self.learning_rate * np.dot(X.T, error) self.bias += self.learning_rate * np.sum(error)

def predict(self, X):
# Calculate net input
net_input = np.dot(X, self.weights) + self.bias

# Apply a threshold (0.0 for linear activation) return np.where(net_input >= 0.0, 1, 0)
# Example Usage (AND operation)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])

# Create and train Adaline model
ada = Adaline(learning_rate=0.1, n_iter=1000) ada.fit(X, y)

# Make predictions
predictions = ada.predict(X)
print("Output: ")
print("Predictions:", predictions)
```

```
Output:
Predictions: [0 1 1 1]
```

Practical 9

Aim: Implementation of Gradient Descent Algorithm

Code: gradient_descent.ipynb

```
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))
```

Output:

```
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
```

```
# 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
```

Output:

```
>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]
```

Practical 10

Aim: Implementation of Principal Component Analysis

Code:

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

#import the Datasets
data = pd.read_csv("driver-data.csv")
print(data.head())
print(data.info())
```

```
           id  mean_dist_day  mean_over_speed_perc
0  3423311935          71.24                    28
1  3423313212          52.53                    25
2  3423313724          64.54                    27
3  3423311373          55.69                    22
4  3423310999          54.58                    25
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 3 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    4000 non-null   int64
 1   mean_dist_day         4000 non-null   float64
 2   mean_over_speed_perc  4000 non-null   int64
dtypes: float64(1), int64(2)
memory usage: 93.9 KB
None
```

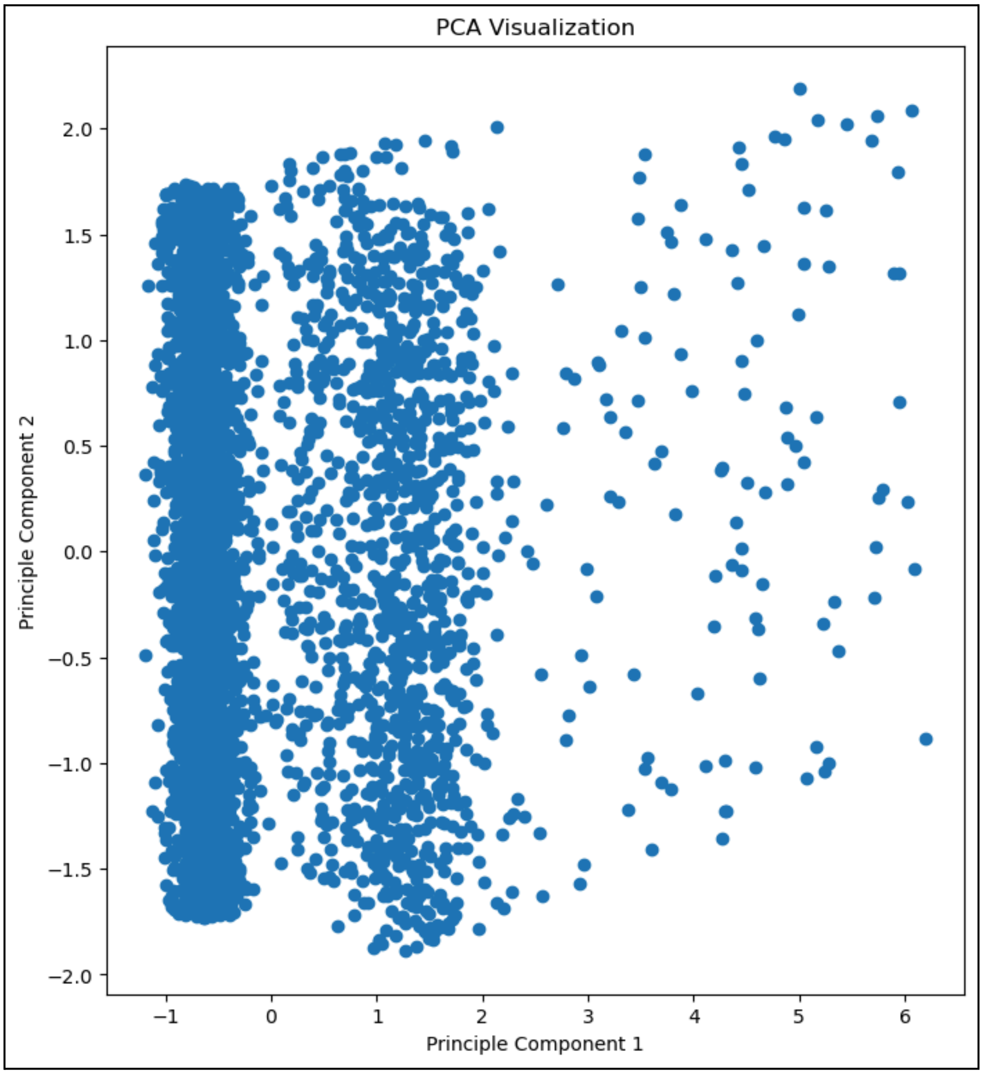
```
#Feature Scaling
x = data.values
print(x)
x = StandardScaler().fit_transform(x)
print(x)
```

```
[[3.42331194e+09 7.12400000e+01 2.80000000e+01]
 [3.42331321e+09 5.25300000e+01 2.50000000e+01]
 [3.42331372e+09 6.45400000e+01 2.70000000e+01]
 ...
 [3.42331292e+09 1.70910000e+02 1.20000000e+01]
 [3.42331363e+09 1.76140000e+02 5.00000000e+00]
 [3.42331153e+09 1.68030000e+02 9.00000000e+00]]
[[-0.44383803 -0.0898104   1.26061251]
 [ 0.66207644 -0.43977285  1.04174351]
 [ 1.10548146 -0.215131    1.18765617]
```

```
#Creating an instance of the PCA class with component=2
pca = PCA(n_components=2)
principle_component = pca.fit_transform(x)

#Creating a new dataset for PCA
principle_data = pd.DataFrame(data=principle_component, columns=['(Principle Component 1)','(Principle Component 2)'])
print(principle_data)
```

```
      (Principle Component 1)  (Principle Component 2)
0                    0.834991                -0.317476
1                    0.412104                 0.784128
2                    0.665867                 1.223204
3                    0.329080                -0.822333
4                    0.474999                -1.122316
...                       ...                      ...
3995                 1.103304                -1.626122
3996                 1.028275                -0.032743
3997                 1.314121                 0.297566
3998                 1.011236                 0.853422
3999                 1.143851                -0.913348

[4000 rows x 2 columns]
```

```
plt.figure(figsize=(8,9))
plt.scatter(principle_data['(Principle Component 1)'], principle_data['(Principle Component 2)'])
plt.xlabel('Principle Component 1')
plt.ylabel('Principle Component 2')
plt.title('PCA Visualization')
plt.show()
```




```
# Explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
print("Explained Variance Ratio:", explained_variance_ratio)
print("Total Explained Variance:", np.sum(explained_variance_ratio))
```

```
Explained Variance Ratio: [0.4223297  0.33447006]
Total Explained Variance: 0.7567997587537549
```

Practical 11

Aim: Implementation of Normalization and 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]]
```

```
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)
```

Transformation

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

```
#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('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_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	0	360.0	1.0	Urban	1
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128	360.0	1.0	Rural	0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66	360.0	1.0	Urban	1
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120	360.0	1.0	Urban	1
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141	360.0	1.0	Urban	1

```
df.dtypes
```

Output:

```
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
```

Output:

	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()
```

Output:

```
np.float64(5403.459283387622)
```

```
obj_cols=df.select_dtypes(exclude=['number']).columns
obj_cols
```

Output:

```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Property_Area'],
      dtype='object')
```

```
X = df.drop('Loan_Status', axis=1)
y =df['Loan_Status']
y.value_counts()
```

Output:

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

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

Output:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	Gender_LP001002	Gender_LP001003	Gender_LP001005	Gender_LP001006	...
0	5849	0.0	0	360.0	1.0	1	True	False	False	False	...
1	4583	1508.0	128	360.0	1.0	0	False	True	False	False	...
2	3000	0.0	66	360.0	1.0	1	False	False	True	False	...
3	2583	2358.0	120	360.0	1.0	1	False	False	False	True	...
4	6000	0.0	141	360.0	1.0	1	False	False	False	False	...
...	...	...	...	...	...	...	...	...	...	...	...
609	2900	0.0	71	360.0	1.0	1	False	False	False	False	...
610	4106	0.0	40	180.0	1.0	1	False	False	False	False	...
611	8072	240.0	253	360.0	1.0	1	False	False	False	False	...
612	7583	0.0	187	360.0	1.0	1	False	False	False	False	...
613	4583	0.0	133	360.0	0.0	0	False	False	False	False	...

614 rows × 635 columns

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

Output:

	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

Practical 12

Aim: Implementation of Logistic Regression

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sb

data = pd.read_csv("/content/CreditRisk.csv")
print(data)
```

```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        ...     ...     ...        ...           ...           ...
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes
```

```
data.shape
```

(614, 13)

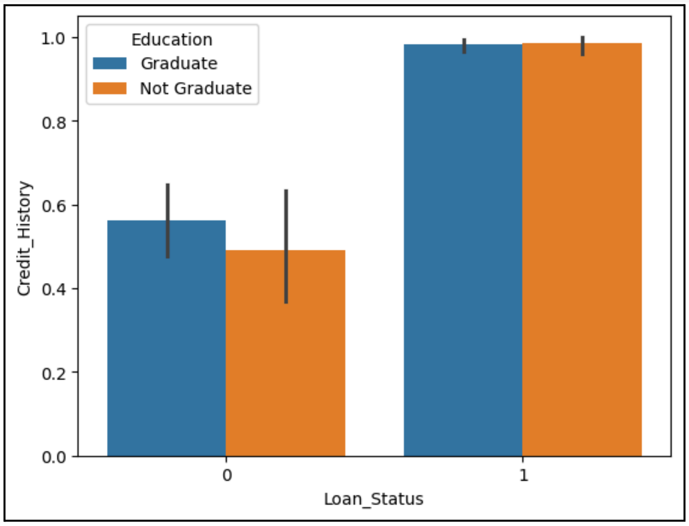
```
data.Loan_Status.value_counts()
```

Loan_Status	count
1	422
0	192

```
data.groupby(['Education','Loan_Status']).Education.count()
```

Education	Loan_Status	Education
Graduate	0	140
	1	340
Not Graduate	0	52
	1	82

```
sb.barplot(y = "Credit_History",x = "Loan_Status",hue = "Education",data = data)
```



#Finding the Null Values

```
data.isnull().sum()
```

	0
Loan_ID	0
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0

dtype: int64

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

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

```
#Segregetting the columns into the different category
data_object = data.select_dtypes(include=['object']).columns
data_numeric = data.select_dtypes(exclude=['object']).columns
print(data_object)
print(data_numeric)
```

```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'Property_Area'],
      dtype='object')
Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Loan_Status'],
      dtype='object')
```

```
#Impute
for columns in data_object:
    majority = data[columns].value_counts().iloc[0]
    data.fillna(majority, inplace = True)
for columns in data_numeric:
    majority = data[columns].value_counts().iloc[0]
    data.fillna(majority, inplace = True)
data.isnull().sum()
```

	0
Gender	0
Married	0
Dependents	0
Education	0
Self_Employed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	0
Credit_History	0
Property_Area	0
Loan_Status	0

dtype: int64

#Changing the String Values to Dummy Integer Values

```
df_dummy = pd.get_dummies(data, columns = data_object)
df_dummy.head()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Status	Gender_489	Gender_Female	Gender_Male	Married_489	...	Dependents_2	Dependents_3+
0	5849	0.0	0	360.0	1.0	1	False	False	True	False	...	False	False
1	4583	1508.0	128	360.0	1.0	0	False	False	True	False	...	False	False
2	3000	0.0	66	360.0	1.0	1	False	False	True	False	...	False	False
3	2583	2358.0	120	360.0	1.0	1	False	False	True	False	...	False	False
4	6000	0.0	141	360.0	1.0	1	False	False	True	False	...	False	False

5 rows × 25 columns

#Model Training

```
from sklearn.model_selection import train_test_split # For splitting the dataset into training and testing sets
from sklearn.linear_model import LogisticRegression # For logistic regression model
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report # For evaluating performance

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(max_iter=1000)
model.fit(train_x, train_y)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
y_pred = model.predict(test_x)

# Calculate accuracy
accuracy = accuracy_score(test_y, y_pred)
print(f"Accuracy: {accuracy}")

# Generate the confusion matrix
conf_matrix = confusion_matrix(test_y, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.6216216216216216
Confusion Matrix:
[[  3  62]
 [  8 112]]
```

```
# Generate the classification report
class_report = classification_report(test_y, y_pred)

print("Classification Report:")
print(class_report)
```

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

           0       0.27      0.05      0.08        65
           1       0.64      0.93      0.76       120

    accuracy                           0.62       185
   macro avg       0.46      0.49      0.42       185
weighted avg       0.51      0.62      0.52       185
```

Practical 13

Aim: Implementation of Support Vector Machine – RBF Kernel

Code: (*Here we are considering same CreditRisk.csv*)

```
#Model Training and Data Spliting
from sklearn.model_selection import train_test_split
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
svm_model = SVC(kernel='rbf', gamma=0.00001, C=1000)
svm_model.fit(train_x, train_y)
```

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

```
#Prediction
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))
print(accuracy_score(test_y, test_y_hat))

confusion_matrix(test_y, test_y_hat)
```

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

           0       0.97      0.95      0.96       127
           1       0.98      0.99      0.98       302

    accuracy                           0.98       429
   macro avg       0.97      0.97      0.97       429
weighted avg       0.98      0.98      0.98       429

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

           0       0.33      0.22      0.26        65
           1       0.64      0.77      0.70       120

    accuracy                           0.57       185
   macro avg       0.49      0.49      0.48       185
weighted avg       0.53      0.57      0.55       185

0.572972972972973
```

```
array([[14, 51],
       [28, 92]])
```

Practical 14

Aim: Implementing Elbow Method for Choosing Number of Clusters

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('driver-data.csv')
print(df.info())
print(df.shape)

data = df[['mean_dist_day', 'mean_over_speed_perc']]
print(data)

data.fillna(data.mean(), inplace=True)
print(data)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 3 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    4000 non-null   int64
 1   mean_dist_day         4000 non-null   float64
 2   mean_over_speed_perc  4000 non-null   int64
dtypes: float64(1), int64(2)
memory usage: 93.9 KB
None
(4000, 3)
      mean_dist_day  mean_over_speed_perc
0             71.24                    28
1             52.53                    25
2             64.54                    27
3             55.69                    22
4             54.58                    25
...             ...                   ...
3995         160.04                    10
3996         176.17                     5
3997         170.91                    12
3998         176.14                     5
3999         168.03                     9
```

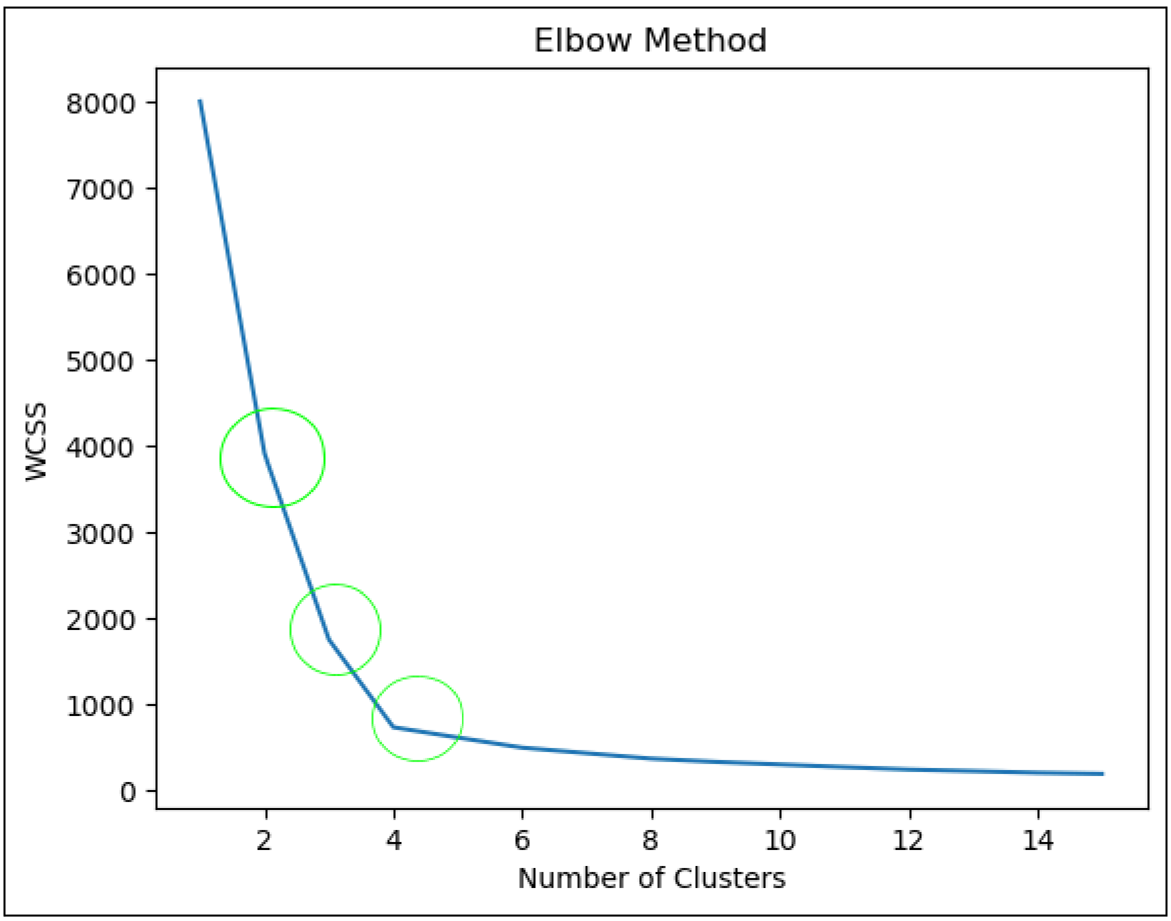
```
[4000 rows x 2 columns]
      mean_dist_day  mean_over_speed_perc
0             71.24                    28
1             52.53                    25
2             64.54                    27
3             55.69                    22
4             54.58                    25
...             ...                   ...
3995         160.04                    10
3996         176.17                     5
3997         170.91                    12
3998         176.14                     5
3999         168.03                     9

[4000 rows x 2 columns]
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(data)

wcss = []
for i in range(1,16):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42, n_init=10)
```

```
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
print(wcss)
```

```
[8000.0, 3911.9263904284157, 1756.5445821314272, 739.153450864558,
619.4037594867996, 502.03685490351063, 437.8780702842313, 374.8024199852508,
337.1456744336048, 306.97337203496494, 276.4393166945794, 248.09311615011552,
229.74448771226085, 210.66993580924535, 198.0806841488063]
```

```
# Plot the Elbow method graph
plt.plot(range(1, 16), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



Hence 3 Number of Clusters would be great.

Practical 15

Aim: Ensemble Techniques – Bagging, Boosting, Stacking, Voting

Code:

```
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier, StackingClassifier,
VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
```

```
iris = load_iris()
X, y = iris.data, iris.target

X.shape
```

```
(150, 4)
```

```
y.shape
```

```
(150,)
```

```
# Splitting Data into training and testing
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)
```

```
Baggin Accuracy: 1.0
```

```
# 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)
```

```
Boosting Accuracy: 1.0
```

```
estimators = [
    ('dt', DecisionTreeClassifier()),
    ('lr', LogisticRegression()),
    ('knn', KNeighborsClassifier())
]
```

```
# 3. Stacking (using Decision Tree, logistic Regression, and KNN as base estimators)
stacking_model = StackingClassifier(estimators=estimators,
final_estimator=LogisticRegression())
stacking_model.fit(x_train, y_train)
stacking_predictions = stacking_model.predict(x_test)
stacking_accuracy = accuracy_score(y_test, stacking_predictions)
print("Stacking Accuracy:", stacking_accuracy)
```

```
Stacking Accuracy: 1.0
```

```
# Voting (using Decision Tree, Logistic Regression and KNN)
voting_model = VotingClassifier(estimators=estimators, 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)
```

```
Voting Accuracy: 1.0
```

Practical 16

Aim: Implementing Bagging and Voting Algorithm Using Random Forest as Base Estimator

Code:

```
# Import necessary libraries
 import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, VotingClassifier
 from sklearn.model_selection import train_test_split
```

```
# Load the Titanic dataset
train = pd.read_csv("titanic.csv") print(train.shape) # Output:
(891, 12)
```

```
(891, 12)
```

```
# Checking for missing data
NAs = pd.concat([train.isnull().sum()], axis=1, keys=["Train"])
 NAs[NAs.sum(axis=1) > 0] # Display columns with missing values
```

	Train
Age	177
Cabin	687
Embarked	2

```
# Removing unnecessary columns
train.pop("Cabin")
train.pop("Name")
train.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 values with mean
train['Age'] = train['Age'].fillna(train['Age'].mean())

# Filling missing Embarked values with most common value
train["Embarked"] = train["Embarked"].fillna(train["Embarked"].mode()[0])

# Converting Pclass to string
train["Pclass"] = train["Pclass"].apply(str)

# Getting Dummies for categorical variables
for col in train.dtypes[train.dtypes == "object"].index:
    for_dummy = train.pop(col)
    train = pd.concat([train, pd.get_dummies(for_dummy, prefix=col)], axis=1)

train.head()
```

	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	False	False	True	False	True	False	False	True
1	2	1	38.0	1	0	71.2833	True	False	False	True	False	True	False	False
2	3	1	26.0	0	0	7.9250	False	False	True	True	False	False	False	True
3	4	1	35.0	1	0	53.1000	True	False	False	True	False	False	False	True
4	5	0	35.0	0	0	8.0500	False	False	True	False	True	False	False	True

```
# Separating labels
labels = train.pop("Survived")

# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(train, labels, test_size=0.25)

# Importing and training RandomForestClassifier
rf = RandomForestClassifier(n_estimators=10)
rf.fit(x_train, y_train)
```

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

```
# Making predictions
y_pred = rf.predict(x_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

```
0.8071748878923767
```

```
# 1. Bagging (using RandomForestClassifier as base estimator)
bagging_model = BaggingClassifier(estimator=RandomForestClassifier(), n_estimators=10)
bagging_model.fit(x_train, y_train)
bagging_prediction = bagging_model.predict(x_test)
bagging_accuracy = accuracy_score(y_test, bagging_prediction)
print("Bagging Accuracy", bagging_accuracy)
```

```
Bagging Accuracy 0.8026905829596412
```

```
# 4. Voting (using RandomForestClassifier)
voting_model = VotingClassifier(estimators=[
    ('rf', RandomForestClassifier())
], voting='hard') # 'hard' for majority vote, 'soft' for weighted average probabilities
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)
```

```
Voting Accuracy: 0.7982062780269058
```

Practical 17

Aim: Implementing AdaBoost Algorithm

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
 from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
train = pd.read_csv('titanic.csv') print(train)
```

```
     PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0              1         0       3  ...   7.2500   NaN         S
1              2         1       1  ...  71.2833   C85         C
2              3         1       3  ...   7.9250   NaN         S
3              4         1       1  ...  53.1000  C123         S
4              5         0       3  ...   8.0500   NaN         S
..           ...       ...     ...  ...      ...   ...       ...
886          887         0       2  ...  13.0000   NaN         S
887          888         1       1  ...  30.0000   B42         S
888          889         0       3  ...  23.4500   NaN         S
889          890         1       1  ...  30.0000  C148         C
890          891         0       3  ...   7.7500   NaN         Q

[891 rows x 12 columns]
```

```
# Checking for missing data
NAs = pd.concat([train.isnull().sum()], axis=1, keys=["Train"])
 NAs[NAs.sum(axis=1) > 0]# Display columns with missing values
```

	Train
Age	177
Cabin	687
Embarked	2

```python
# Removing unnecessary columns
 train.pop("Cabin")
train.pop("Name")
train.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
```

```python
# Filling missing Age values with mean
train['Age'] = train['Age'].fillna(train['Age'].mean())
 # Filling missing Embarked values with most common value
train["Embarked"] = train["Embarked"].fillna(train["Embarked"].mode()[0])
 # Converting Pclass to string
train["Pclass"] = train["Pclass"].apply(str)

# Getting Dummies for categorical variables
 for col in train.dtypes[train.dtypes == "object"].index:
     for_dummy = train.pop(col)
     train = pd.concat([train, pd.get_dummies(for_dummy, prefix=col)], axis=1)
 train.head()
```

	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	False	False	True	False	True	False	False	True
1	2	1	38.0	1	0	71.2833	True	False	False	True	False	True	False	False
2	3	1	26.0	0	0	7.9250	False	False	True	True	False	False	False	True
3	4	1	35.0	1	0	53.1000	True	False	False	True	False	False	False	True
4	5	0	35.0	0	0	8.0500	False	False	True	False	True	False	False	True

```
# Separating labels
labels = train.pop("Survived")

# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(train, labels,
 test_size=0.25)

dt_model = DecisionTreeClassifier(max_depth=14)

dt_model.fit(x_train, y_train)
```

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

```
y_pred = dt_model.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
 print("Accuracy:", accuracy)

_classification_report = classification_report(y_test, y_pred)
 print("Classification Report \n", _classification_report)
```

```
Accuracy 0.7399103139013453
Classification Report
               precision    recall  f1-score   support

           0       0.78      0.81      0.80       140
           1       0.66      0.63      0.64        83

    accuracy                           0.74       223
   macro avg       0.72      0.72      0.72       223
weighted avg       0.74      0.74      0.74       223
```

```
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)
print("Boosting Classification Report \n", classification_report(y_test,
 boosting_predictions))
```

```
Boosting Accuracy: 0.7040358744394619
Boosting Classification Report
              precision    recall  f1-score   support

           0       0.77      0.76      0.76       140
           1       0.60      0.61      0.61        83

    accuracy                           0.70       223
   macro avg       0.68      0.69      0.68       223
weighted avg       0.71      0.70      0.70       223
```

Practical 18

Aim: Implementation of Gradient Boosting Algorithm

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
train = pd.read_csv('titanic.csv') print(train.shape)
```

```
(891, 12)
```

```
# Checking for missing data
NAs = pd.concat([train.isnull().sum()], axis=1, keys=["Train"])
 NAs[NAs.sum(axis=1) > 0]# Display columns with missing values
```

	Train
Age	177
Cabin	687
Embarked	2

```
# Removing unnecessary columns
 train.pop("Cabin")
train.pop("Name")
train.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 values with mean
train['Age'] = train['Age'].fillna(train['Age'].mean())

# Filling missing Embarked values with most common value
train["Embarked"] = train["Embarked"].fillna(train["Embarked"].mode()[0])

# Converting Pclass to string
train["Pclass"] = train["Pclass"].apply(str)

# Getting Dummies for categorical variables
 for col in train.dtypes[train.dtypes == "object"].index:
     for_dummy = train.pop(col)
train = pd.concat([train, pd.get_dummies(for_dummy, prefix=col)], axis=1)

train.head()
```

	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	False	False	True	False	True	False	False	True
1	2	1	38.0	1	0	71.2833	True	False	False	True	False	True	False	False
2	3	1	26.0	0	0	7.9250	False	False	True	True	False	False	False	True
3	4	1	35.0	1	0	53.1000	True	False	False	True	False	False	False	True
4	5	0	35.0	0	0	8.0500	False	False	True	False	True	False	False	True

```python
# Separating labels
labels = train.pop("Survived")

# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(train, labels,
 test_size=0.25)

 # gradient boosting
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)
gb_accuracy = accuracy_score(y_test, y_pred)
print("Gradient Boosting Accuracy:", gb_accuracy)
print("Gradient Boosting Report \n", classification_report(y_test, y_pred))
```

```
Gradient Boosting Accuracy: 0.8251121076233184
Gradient Boosting Report
               precision    recall  f1-score   support

           0       0.87      0.84      0.86       139
           1       0.75      0.80      0.77        84

    accuracy                           0.83       223
   macro avg       0.81      0.82      0.82       223
weighted avg       0.83      0.83      0.83       223
```