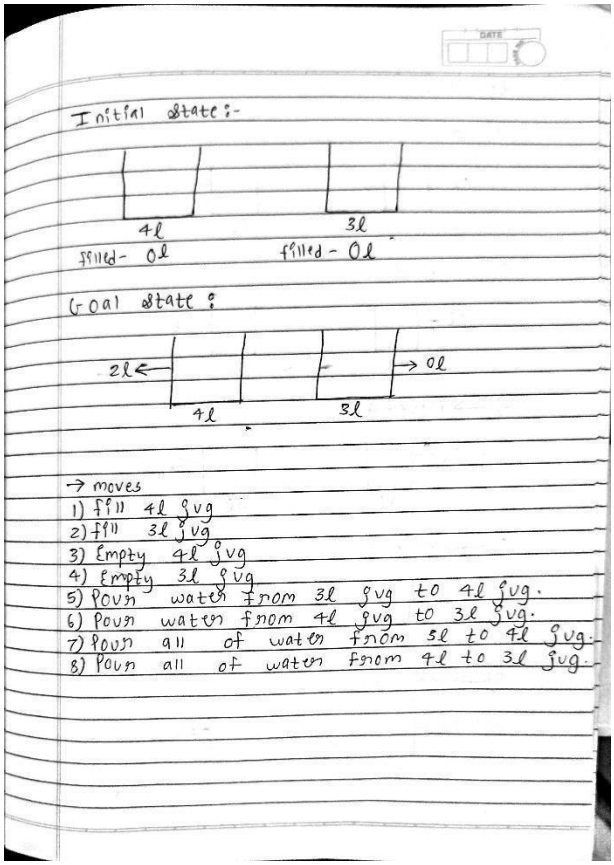
Practical No.1

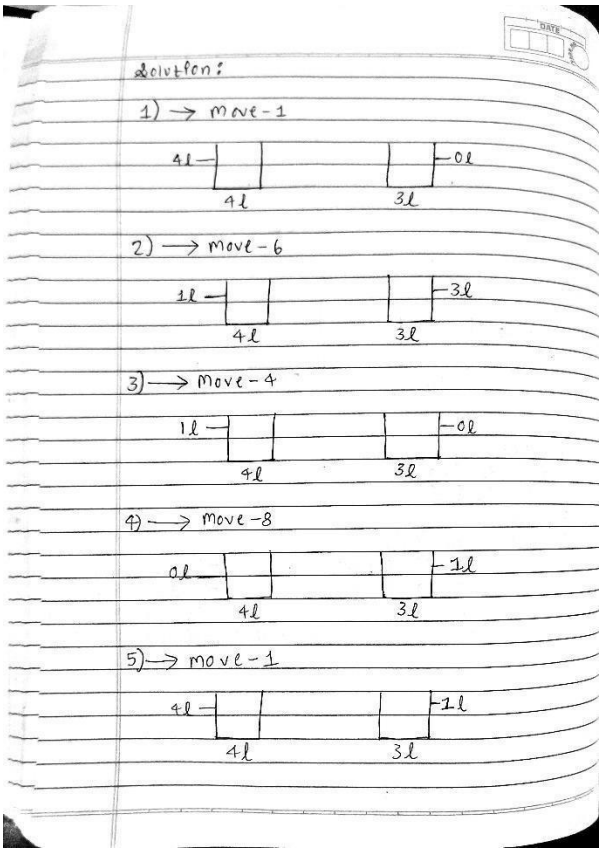
Aim: Implementation of logic Programming using PROLOG DFS for water jug problem.

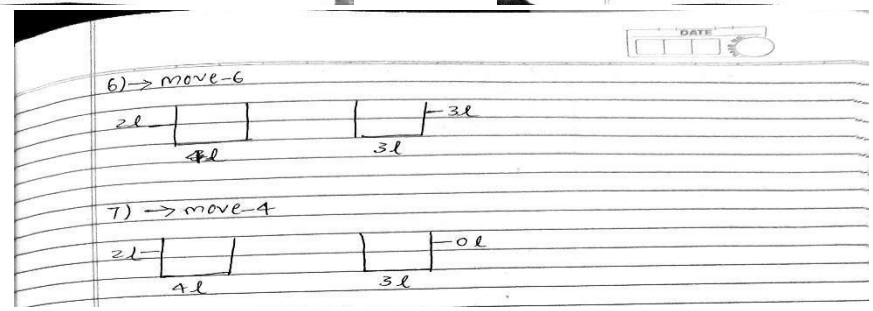
Objectives:

- Understanding of the DFS algorithm.
- Understanding of water-jug problem
- Analysis of water-jug Problem by using DFS.
- Implementation of DFS to solve the water jug problem.

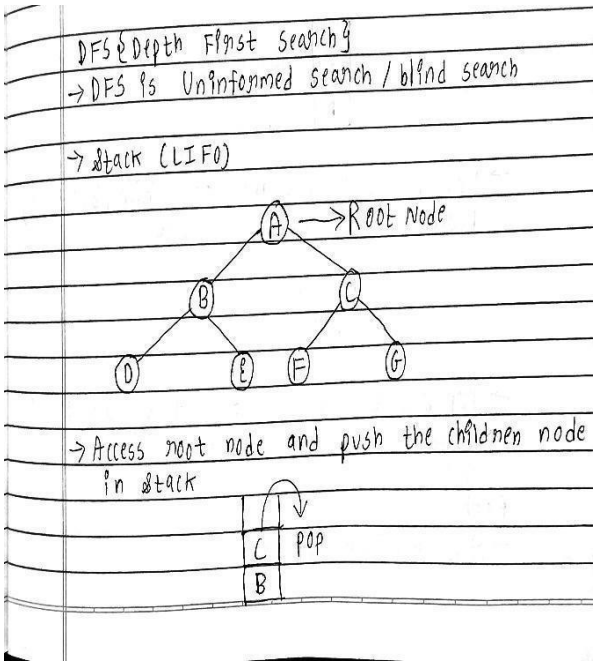
Theory:

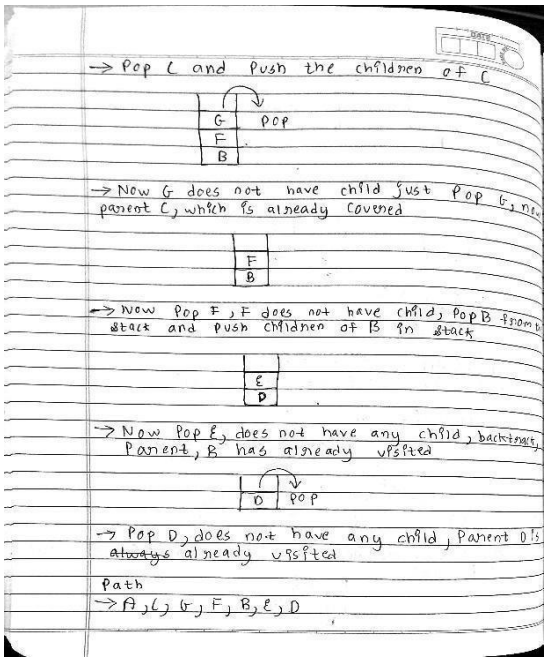


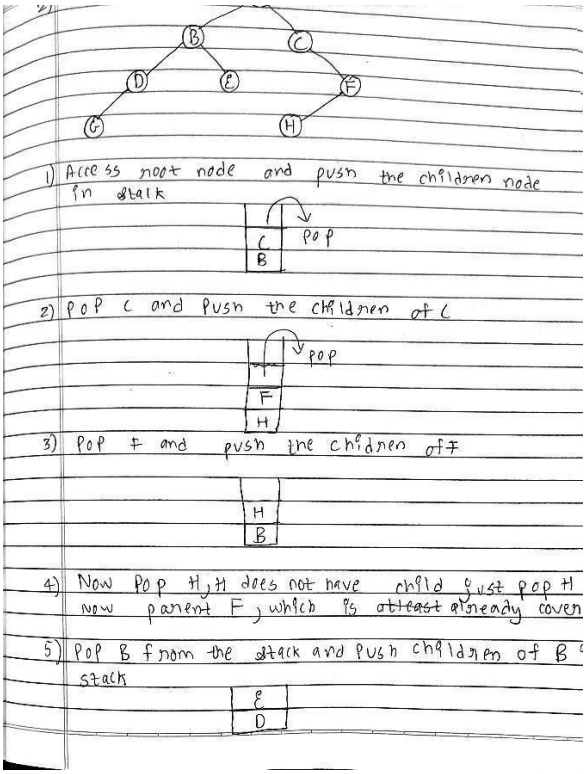


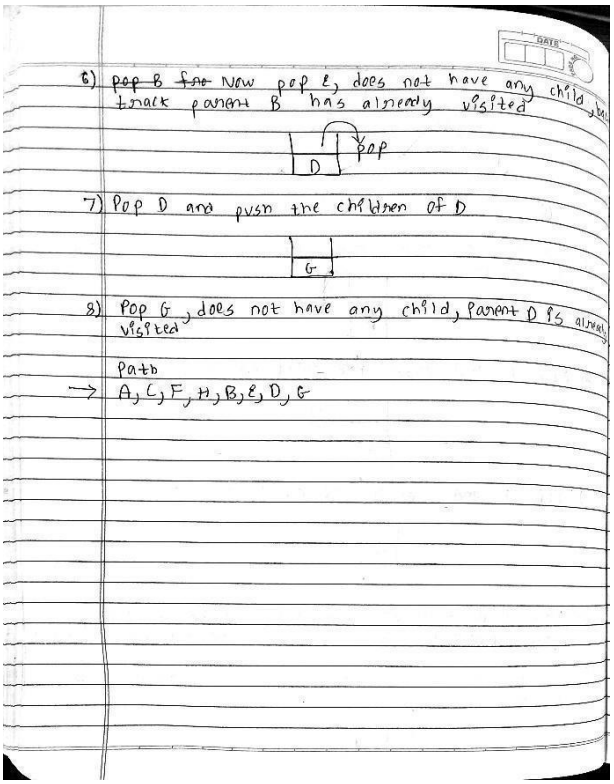


Problems based on DFS:







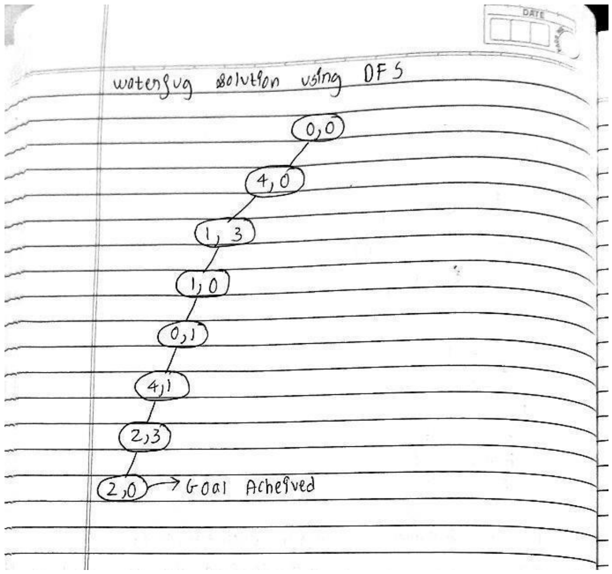


Code:

```
start(2,0):-write(' 4lit Jug:  2 | 3lit Jug:  0|\n'),
        write('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n'),
```

```
        write('Goal Reached! Congrats!!\n'),
        write('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n').
start(X,Y):-write(' 4lit Jug: '),write(X),write('| 3lit Jug: '),
        write(Y),write('|\n'),
        write(' Enter the
        move::'), read(N),
        contains(X,Y,N).

contains(_,Y,1):-start(4,Y).
contains(X,_,2):-start(X,3).
contains(_,Y,3):-start(0,Y).
contains(X,_,4):-start(X,0).
contains(X,Y,5):-N is Y-4+X, start(4,N).
contains(X,Y,6):-N is X-3+Y, start(N,3).
contains(X,Y,7):-N is X+Y, start(N,0).
contains(X,Y,8):-N is X+Y, start(0,N).

main():-write(' Water Jug Game \n'),
      write('Intial State: 4lit Jug- 0lit\n'),
      write('              3lit Jug- 0lit\n'),
      write('Final State: 4lit Jug- 2lit\n'),
      write('              3lit Jug- 0lit\n'),
      write('Follow the Rules: \n'),
      write('Rule 1: Fill 4lit Jug\n'),
      write('Rule 2: Fill 3lit Jug\n'),
      write('Rule 3: Empty 4lit Jug\n'),
      write('Rule 4: Empty 3lit Jug\n'),
      write('Rule 5: Pour water from 3lit Jug to fill 4lit Jug\n'),
      write('Rule 6: Pour water from 4lit Jug to fill 3lit Jug\n'),
      write('Rule 7: Pour all of water from 3lit Jug to 4lit Jug\n'),
      write('Rule 8: Pour all of water from 4lit Jug to 3lit Jug\n'),
      write(' 4lit Jug: 0 | 3lit Jug: 0'),nl,
      write(' Enter the move::'),
      read(N),nl,
      contains(0,0,N).
```

Output:




```
SWI-Prolog -- c:/Users/ADMIN/Downloads/waterjug (1).pl
File   Edit   Settings   Run   Debug   Help
Welcome to SWI-Prolog (threaded, 32 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?- main().
 Water Jug Game
Intial State: 4lit Jug- 0lit
              3lit Jug- 0lit
Final State:  4lit Jug- 2lit
              3lit Jug- 0lit
Follow the Rules:
Rule 1: Fill 4lit Jug
Rule 2: Fill 3lit Jug
Rule 3: Empty 4lit Jug
Rule 4: Empty 3lit Jug
Rule 5: Pour water from 3lit Jug to fill 4lit Jug
Rule 6: Pour water from 4lit Jug to fill 3lit Jug
Rule 7: Pour all of water from 3lit Jug to 4lit Jug
Rule 8: Pour all of water from 4lit Jug to 3lit Jug
 4lit Jug:   0 | 3lit Jug:   0
 Enter the move::1.

 4lit Jug:   4| 3lit Jug:   0|
 Enter the move::|: 6.
 4lit Jug:   1| 3lit Jug:   3|
 Enter the move::|: 4.
 4lit Jug:   1| 3lit Jug:   0|
 Enter the move::|: 8.
 4lit Jug:   0| 3lit Jug:   1|
 Enter the move::|: 1.
 4lit Jug:   4| 3lit Jug:   1|
 Enter the move::|: 6.
 4lit Jug:   2| 3lit Jug:   3|
 Enter the move::|: 4.
 4lit Jug:   2 | 3lit Jug:   0|
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Goal Reached! Congrats!!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
true .
```

Practical No.2

Program no. 1:

Aim:- Design and solve Tic-Tac-Toe using BFS

Objective –

- Understand Tic-Tac-Toe game Logic.
- Understand BFS Algorithm
- Solving Tic-Tac-Toe using BFS.
- Implementation of BFS for Tic-Tac-Toe

Theory:-

Tic-Tac-Toe is a 2-player game played on a 3x3 grid. Players take turns marking empty spaces with "X" or "O." The goal is to get three of your marks in a row, either horizontally, vertically, or diagonally. The first to do so wins, or the game ends in a draw if all spaces are filled with no winner. Simple, yet fun!

Introduction of BFS

The algorithm starts from a given source and explores all reachable vertices from the given source. It is similar to the Breadth-First Traversal of a tree. Like tree, we begin with the given source (in tree, we begin with root) and traverse vertices level by level using a queue data structure. The only catch here is that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.

Advantages of Breadth First Search:

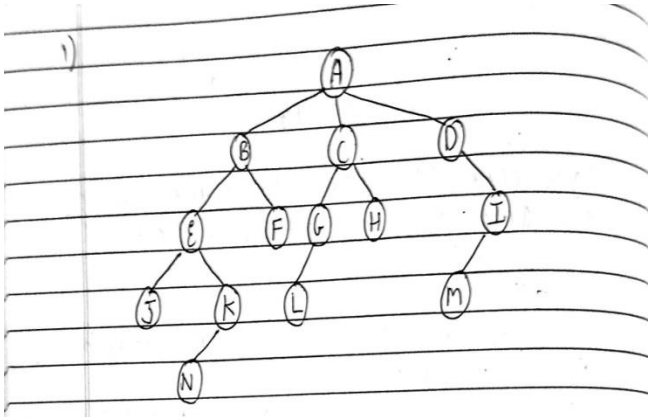
- BFS will never get trapped exploring the useful path forever.
- If there is a solution, BFS will definitely find it.
- If there is more than one solution then BFS can find the minimal one that requires less number of steps.
- Low storage requirement – linear with depth.
- Easily programmable.

Disadvantages of Breadth First Search:

The main drawback of BFS is its memory requirement. Since each level of the graph must be saved in order to generate the next level and the amount of memory is proportional to the number of nodes stored the space complexity of BFS is $\mathbf{O(b^d)}$, where **b** is the branching factor(the number of children at each node, the outdegree) and **d** is the depth. As a result, BFS is severely space-bound in practice so will exhaust the memory available on typical computers in a matter of minutes.

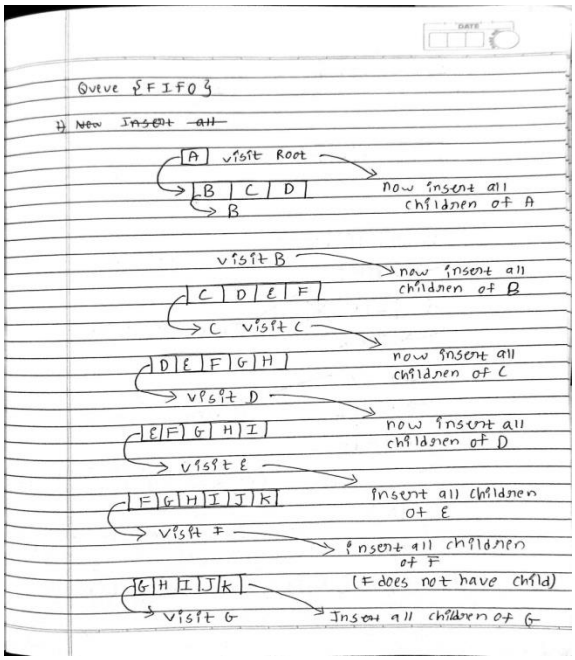
Problems based on BFS:

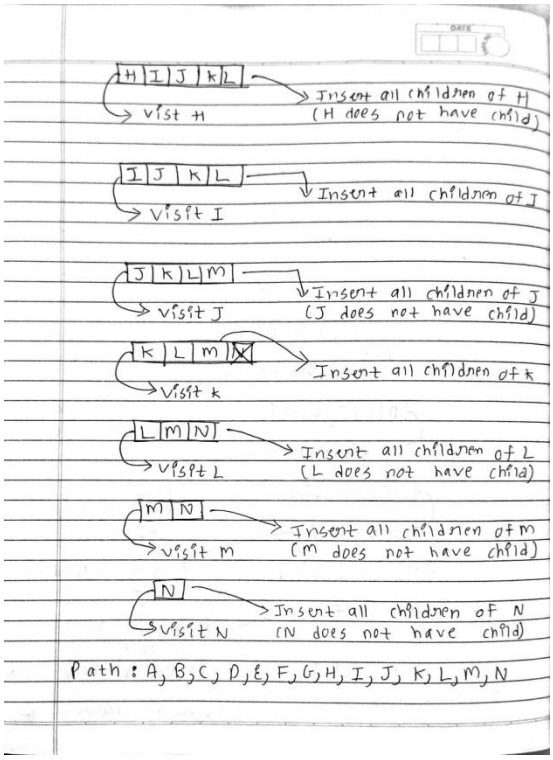
1)



Queue {FIFO}

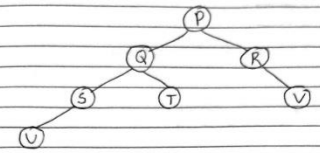
1) Now ~~Insert all~~



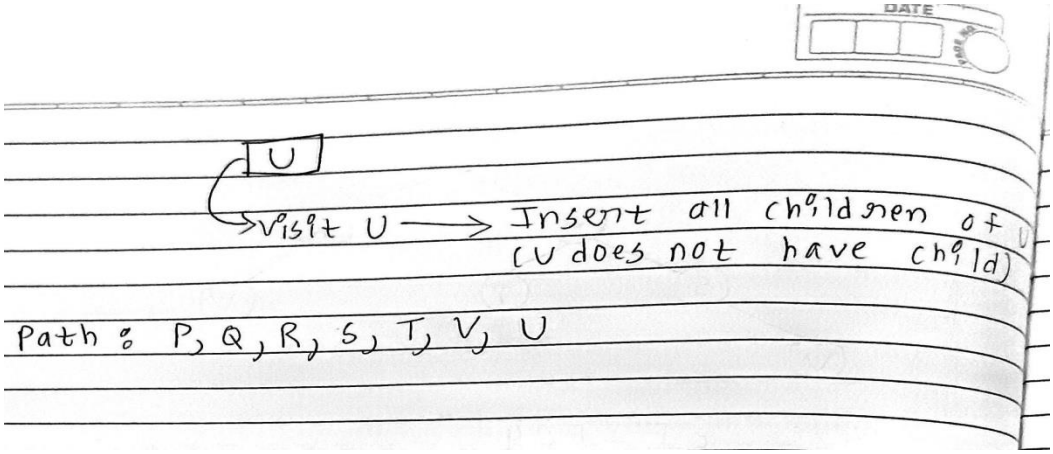


Path : A, B, C, D, E, F, G, H, I, J, K, L, M, N

2)



Queue {FIFO}



Code:

```
play :- my_turn([]).

my_turn(Game) :-
   valid_moves(ValidMoves, Game, x),
   any_valid_moves(ValidMoves, Game).

any_valid_moves([], _) :-
   write('It is a tie'), nl.
any_valid_moves([_|_], Game) :-
   findall(NextMove, game_analysis(x, Game, NextMove), MyMoves),
   do_a_decision(MyMoves, Game).

% This can only fail in the beginning.
do_a_decision(MyMoves, Game) :-
   not(MyMoves = []),
   length(MyMoves, MaxMove),
   random(0, MaxMove, ChosenMove),
   nth0(ChosenMove, MyMoves, X),
   NextGame = [X | Game],
   print_game(NextGame),
   (victory_condition(x, NextGame) ->
      (write('I won. You lose.'), nl);
      your_turn(NextGame), !).

your_turn(Game) :-
   valid_moves(ValidMoves, Game, o),
   (ValidMoves = [] -> (write('It is a tie'), nl);
    (write('Available moves:'), write(ValidMoves), nl,
     ask_move(Y, ValidMoves),
     NextGame = [Y | Game],
     (victory_condition(o, NextGame) ->
     (write('I lose. You win.'), nl);
       my_turn(NextGame), !))).

ask_move(Move, ValidMoves) :-
   write('Give your move:'), nl,
   read(Move), member(Move, ValidMoves), !.

ask_move(Y, ValidMoves) :-
   write('not a move'), nl,
   ask_move(Y,
```

ValidMoves).

```
movement_prompt(X, Y, ValidMoves) :-
   write('Give your X:'), nl, read(X), member(move(o, X, Y), ValidMoves), !,
   write('Give your Y:'), nl, read(Y), member(move(o, X, Y), ValidMoves).

% A routine for printing games.. Well you can use it.
print_game(Game) :-
   plot_row(0, Game), plot_row(1, Game), plot_row(2, Game).

plot_row(Y, Game) :-
   plot(Game, 0, Y), plot(Game, 1, Y), plot(Game, 2, Y), nl.

plot(Game, X, Y) :-
   (member(move(P, X, Y), Game), ground(P)) -> write(P) ; write('.').

% This system determines whether there's a perfect play available.
game_analysis(_, Game, _) :-
   victory_condition(Winner, Game),
   Winner = x. % We do not want to lose.
   % Winner = o. % We do not want to win. (egostroking mode).
   % true.
% If you remove this constraint entirely, it may let you win.
game_analysis(Turn, Game, NextMove) :-
   not(victory_condition(_, Game)),
   game_analysis_continue(Turn, Game, NextMove).

game_analysis_continue(Turn, Game, NextMove) :-
   valid_moves(Moves, Game, Turn),
   game_analysis_search(Moves, Turn, Game, NextMove).

% Comment these away and the system refuses to play,
% because there are no ways to play this without a possibility of tie.
game_analysis_search([], o, _, _). % Tie on opponent's turn.
game_analysis_search([], x, _, _). % Tie on our turn.

game_analysis_search([X|Z], o, Game, NextMove) :- % Whatever opponent does,
   NextGame = [X | Game],                  % we desire not to lose.
   game_analysis_search(Z, o, Game, NextMove),
   game_analysis(x, NextGame, _), !.

game_analysis_search(Moves, x, Game, NextMove) :-
   game_analysis_search_x(Moves, Game, NextMove).

game_analysis_search_x([X|_], Game, X) :-
   NextGame = [X | Game],
   game_analysis(o, NextGame, _).
game_analysis_search_x([_|Z], Game, NextMove) :-
   game_analysis_search_x(Z, Game, NextMove).

% This thing describes all kinds of valid games.
valid_game(Turn, Game, LastGame, Result) :-
   victory_condition(Winner, Game) ->
      (Game = LastGame, Result = win(Winner)) ;
      valid_continuing_game(Turn, Game, LastGame, Result).

valid_continuing_game(Turn, Game, LastGame, Result) :-
   valid_moves(Moves, Game, Turn),
   tie_or_next_game(Moves, Turn, Game, LastGame, Result).
```

```
tie_or_next_game([], _, Game, Game, tie).
tie_or_next_game(Moves, Turn, Game, LastGame, Result) :-
   valid_gameplay_move(Moves, NextGame, Game),
   opponent(Turn, NextTurn),
   valid_game(NextTurn, NextGame, LastGame, Result).

% Victory conditions for tic tac toe.
victory(P, Game, Begin) :-
   valid_gameplay(Game, Begin),
   victory_condition(P, Game).

victory_condition(P, Game) :-
   (X = 0; X = 1; X = 2),
   member(move(P, X, 0), Game),
   member(move(P, X, 1), Game),
   member(move(P, X, 2), Game).

victory_condition(P, Game) :-
   (Y = 0; Y = 1; Y = 2),
   member(move(P, 0, Y), Game),
   member(move(P, 1, Y), Game),
   member(move(P, 2, Y), Game).

victory_condition(P, Game) :-
   member(move(P, 0, 2), Game),
   member(move(P, 1, 1), Game),
   member(move(P, 2, 0), Game).

victory_condition(P, Game) :-
   member(move(P, 0, 0), Game),
   member(move(P, 1, 1), Game),
   member(move(P, 2, 2), Game).

% This describes a valid form of gameplay.
% Which player did the move is disregarded.
valid_gameplay(Start, Start).

valid_gameplay(Game, Start) :-
   valid_gameplay(PreviousGame, Start),
   valid_moves(Moves, PreviousGame, _),
   valid_gameplay_move(Moves, Game, PreviousGame).

valid_gameplay_move([X|_], [X|PreviousGame], PreviousGame).
valid_gameplay_move([_|Z], Game, PreviousGame) :-
   valid_gameplay_move(Z, Game, PreviousGame).

% The set of valid moves must not be affected by the decision making
% of the prolog interpreter.
% Therefore we have to retrieve them like this.
% This is equivalent to the (∀x∈0..2)(∀y∈0..2)(....
% uh wait.. There's no way to represent this using those quantifiers.
valid_moves(Moves, Game, Turn) :-
   valid_moves_column(0, M1, [], Game, Turn),
   valid_moves_column(1, M2, M1, Game, Turn),
   valid_moves_column(2, Moves, M2, Game, Turn).

valid_moves_column(X, M3, M0, Game, Turn) :-
```

```
valid_moves_cell(X, 0, M1, M0, Game, Turn), valid_moves_cell(X, 1, M2, M1, Game, Turn), valid_moves_cell(X, 2, M3, M2, Game, Turn).

valid_moves_cell(X, Y, M1, M0, Game, Turn) :-
  member(move(_, X, Y), Game) -> M0 = M1 ; M1 = [move(Turn,X,Y) | M0].

% valid_move(X, Y, Game) :-
%    (X = 0; X = 1; X = 2),
%    (Y = 0; Y = 1; Y = 2),
%    not(member(move(_, X, Y), Game)).

opponent(x, o).
opponent(o, x).
```

Output:

```
?- play().
...
.x.
...
Available moves:[move(o,2,2),move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,0),move(o,0,2),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,1,2).
.x.
.x.
.o.
Available moves:[move(o,2,2),move(o,2,1),move(o,2,0),move(o,0,2),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,2,2).
xx.
.x.
.oo
Available moves:[move(o,2,1),move(o,2,0),move(o,0,2),move(o,0,1)]
Give your move:
|: move(o,0,2).
I lose. You win.
true.


?- play().
...
.x.
...
Available moves:[move(o,2,2),move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,0),move(o,0,2),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,2,2).
...
.x.
x.o
Available moves:[move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,0),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,2,0).
..o
.xx
x.o
Available moves:[move(o,1,2),move(o,1,0),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,1,0).
xoo
.xx
x.o
Available moves:[move(o,1,2),move(o,0,1)]
Give your move:
|: move(o,1,2).
xoo
xxx
xoo
I won. You lose.
true.
```

Practical No. 3

Aim: Design Hill climbing Algorithm to solve 8-puzzle problem

Objectives:

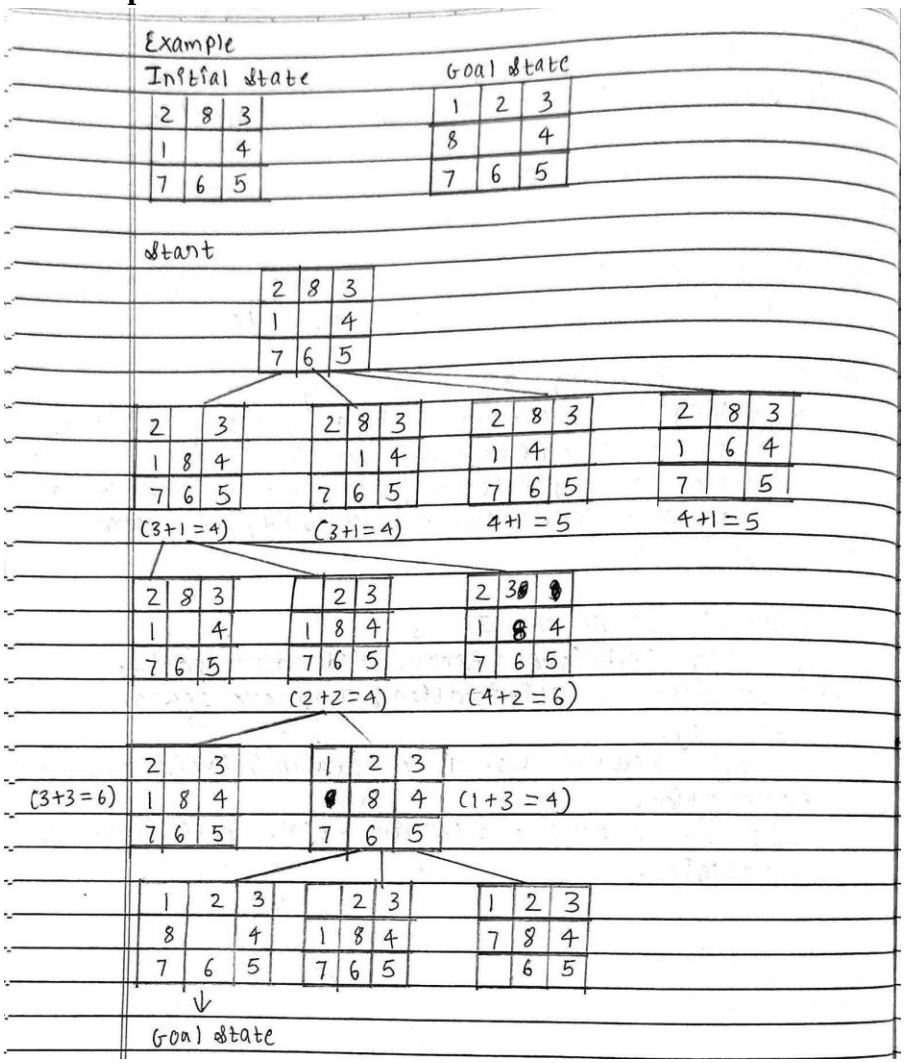
- Understand and Implement Hill Climbing algorithm
- Understand 8-puzzle problem and solve using Hill Climbing algorithm

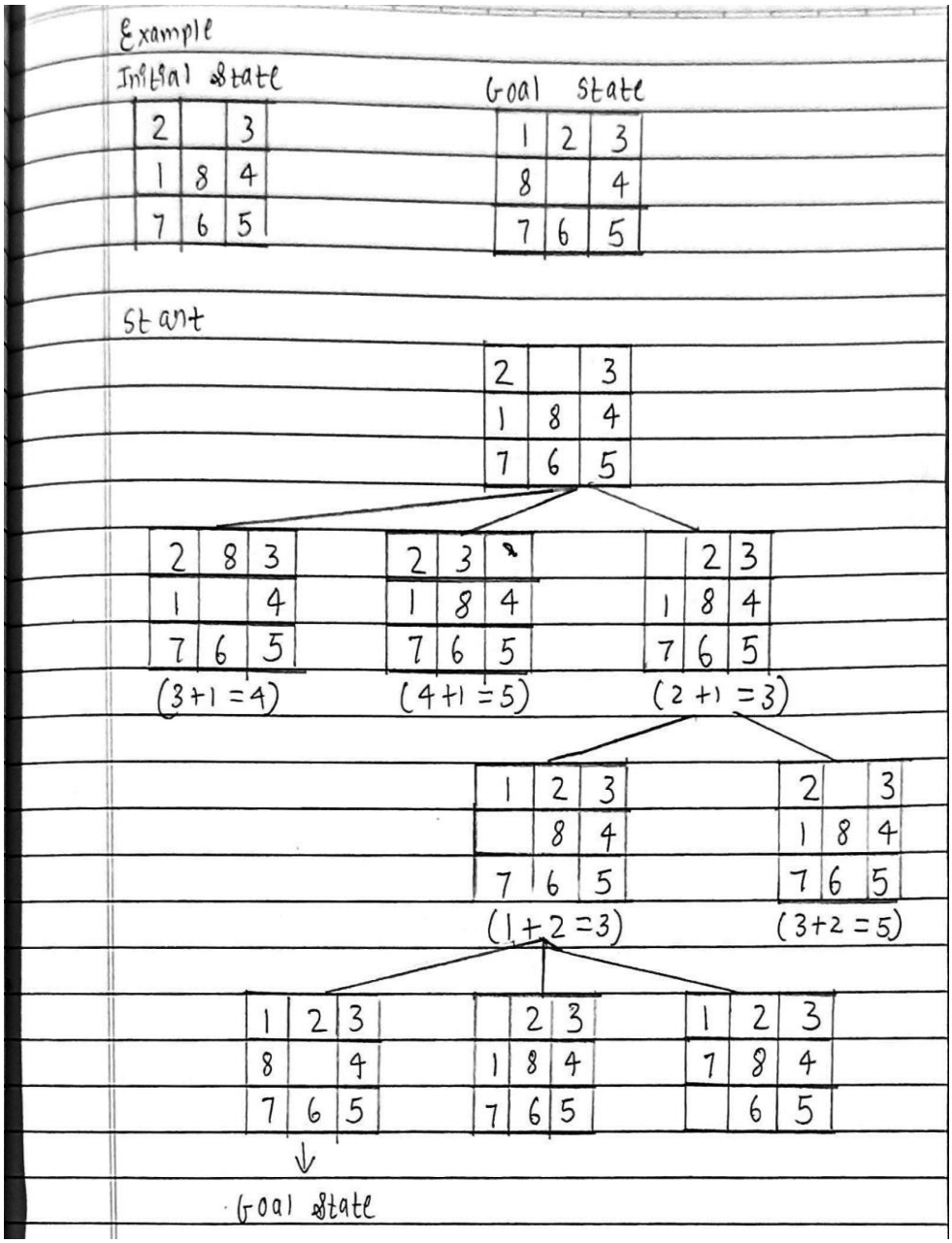
Theory:

Introduction:-

The 8 Puzzle Problem is a classic example which is often used in the field of Artificial Intelligence (AI) to test search algorithms and problem-solving techniques. This puzzle consists of a 3x3 grid with 8 numbered tiles and one empty space

Examples:





Advantages

- Efficient: It can quickly find local optima, which is useful when time is limited
- Simple: It's easy to understand and implement
- Memory efficient: It only needs to store the current state's data
- Low computational power: It doesn't require a lot of computational power or external resources

Disadvantages

- Local optima: It can get stuck at locally optimal solutions that aren't the best overall
- Limited exploration: It focuses on the immediate vicinity, which can cause it to miss globally optimal solutions
- Depends on initial state: The quality of the solution depends on where the algorithm starts
- No guarantee of optimal solution: It doesn't always find the best solution

Code:

```
% 8-Puzzle Problem using Hill Climbing in Prolog

% Define goal state
goal([[1,2,3], [8,0,4], [7,6,5]]).

% Move tile by swapping empty tile (0) with an adjacent tile
move(State, NewState) :-
    find_blank(State, X, Y),
    (move_up(State, X, Y, NewState) ;
    move_down(State, X, Y, NewState) ;
    move_left(State, X, Y, NewState) ;
    move_right(State, X, Y, NewState)).

% Find blank (0) position
find_blank(State, X, Y) :- nth0(X, State, Row), nth0(Y, Row, 0).

% Moves
move_up(State, X, Y, NewState) :-
    X > 0, X1 is X - 1,
    swap(State, X, Y, X1, Y, NewState).

move_down(State, X, Y, NewState) :-
    X < 2, X1 is X + 1,
    swap(State, X, Y, X1, Y, NewState).

move_left(State, X, Y, NewState) :-
    Y > 0, Y1 is Y - 1,
    swap(State, X, Y, X, Y1, NewState).

move_right(State, X, Y, NewState) :-
    Y < 2, Y1 is Y + 1,
    swap(State, X, Y, X, Y1, NewState).

% Swap tiles
swap(State, X1, Y1, X2, Y2, NewState) :-
    nth0(X1, State, Row1), nth0(Y1, Row1, Tile),
    nth0(X2, State, Row2), nth0(Y2, Row2, Tile2),
```

```
    replace(State, X1, Y1, Tile2, TempState),
    replace(TempState, X2, Y2, Tile, NewState).

% Replace element in 2D list
replace(State, X, Y, Val, NewState)
:-
    nth0(X, State, Row, RestRows),
    nth0(Y, Row, _, RestCols),
    nth0(Y, NewRow, Val,
    RestCols),
    nth0(X, NewState, NewRow, RestRows).

% Manhattan distance heuristic
heuristic(State, H) :-
    goal(Goal),
    findall(D, (nth0(X, State, Row), nth0(Y, Row, Tile), Tile \= 0,
            goal_position(Tile, Goal, GX, GY),
            D is abs(X - GX) + abs(Y - GY)), Distances),
    sumlist(Distances, H).

% Get goal position of a tile
goal_position(Tile, Goal, X, Y) :-
    nth0(X, Goal, Row), nth0(Y, Row, Tile).

% Hill Climbing search
solve_hill_climb(State, Path) :-
    solve_hill_climb_helper(State, [], Path).

solve_hill_climb_helper(State, Path, Path) :- goal(State).
solve_hill_climb_helper(State, Visited, Path) :-
    move(State, NextState),
    \+ member(NextState,
    Visited), heuristic(NextState,
    H),
    best_next_state(State, H, NextState, Visited, NewState),
    solve_hill_climb_helper(NewState, [NewState | Visited], Path).

% Choose best next state
best_next_state(State, BestH, BestState, Visited, NewState) :-
    findall(S, (move(State, S), \+ member(S, Visited)),
    Candidates), maplist(heuristic, Candidates, Heuristics),
    min_list(Heuristics, BestH),
    nth0(Index, Heuristics, BestH),
    nth0(Index, Candidates, NewState).

% Solve 8-puzzle with Hill Climbing
solve_puzzle(Start) :-
    solve_hill_climb(Start, Path),
```

```
    print_solution(Path).

% Print solution
```

```
print_solution([]).
print_solution([S|Rest]) :-
    write(S), nl, print_solution(Rest).
```

Output:

```
?- solve_puzzle([[2,8,3],[1,0,4],[7,6,5]]).
[[1,2,3],[8,0,4],[7,6,5]]
[[1,2,3],[0,8,4],[7,6,5]]
[[0,2,3],[1,8,4],[7,6,5]]
[[2,0,3],[1,8,4],[7,6,5]]
true
?- solve_puzzle([[2,0,3],[1,8,4],[7,6,5]]).
[[1,2,3],[8,0,4],[7,6,5]]
[[1,2,3],[0,8,4],[7,6,5]]
[[0,2,3],[1,8,4],[7,6,5]]
true
```

Practical No. 4

Aim: Understanding Basics of Python Programming.
Objective: Learn Python Libraries{NumPy,Pandas,SciPy,MatPlotlib,ScikitLearn}.

Theory:

NumPy is a library for numerical computing in Python. It provides support for multi-dimensional arrays and a variety of mathematical functions, making it essential for scientific computing. Its powerful array manipulation capabilities make data handling efficient and fast.

Pandas

Pandas is designed for data analysis and manipulation. It introduces structures like Series and DataFrame, which simplify data organization, filtering, and cleaning. It's widely used in data science and integrates well with other libraries.

Matplotlib

Matplotlib is a popular visualization library used to create a variety of plots, including line graphs, bar charts, and scatter plots. It provides detailed control over visual elements, making it ideal for clear and informative data presentation.

SciPy

SciPy extends NumPy by offering advanced scientific functions such as optimization, signal processing, and integration. It is particularly useful for complex mathematical computations.

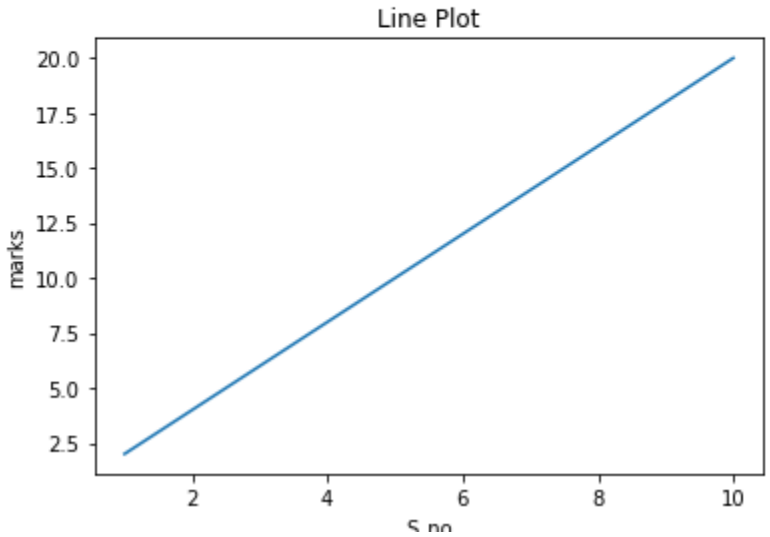
Scikit-learn

Scikit-learn is a machine learning library that simplifies tasks like classification, regression, and clustering. It includes powerful tools for model evaluation, data preprocessing, and feature selection, making it a go-to choice for building predictive models.

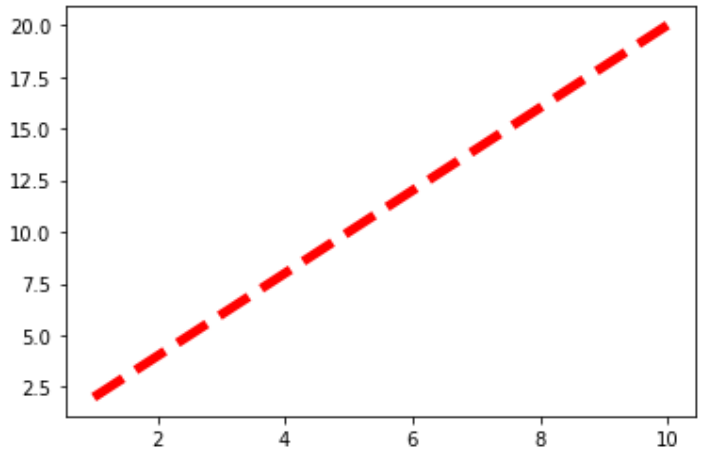
Implementation:

```
In [1]: import numpy as np

In [2]: n1=np.array([10,20,30,40])
        n1

Out[2]: array([10, 20, 30, 40])

In [3]: array([10, 20, 30, 40])

In [4]: n2=np.array([[10,20,30,40],[40,30,20,10]])
        n2

Out[4]: array([[10, 20, 30, 40],
               [40, 30, 20, 10]])
```

```
In [5]: plt.plot(x,y)
        plt.title("Line Plot")
        plt.xlabel("S_no")
        plt.ylabel("marks")
        plt.show()
```

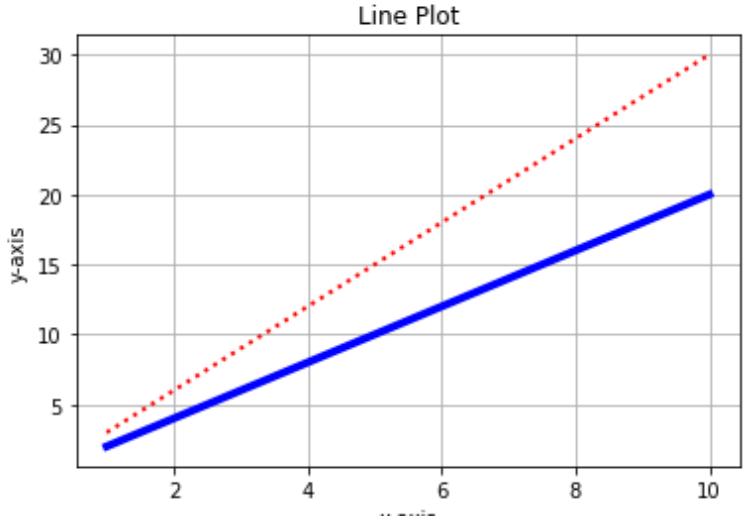


In [7]:
```
plt.plot(x,y,color='r',linestyle='--',linewidth=5)
plt.show()
```

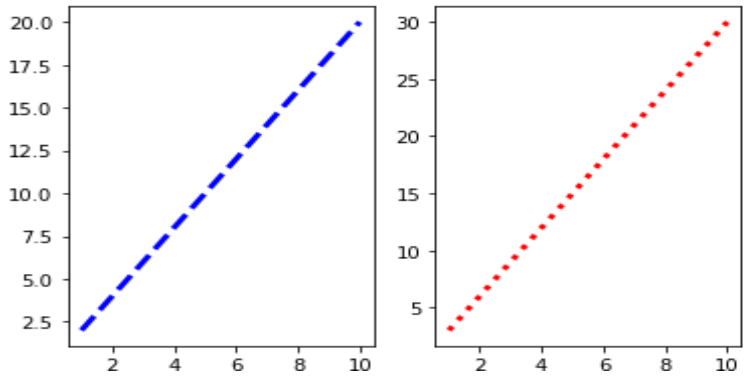


In [8]:
```
x=np.arange(1,11)
y1=2*x
y2=3*x
```

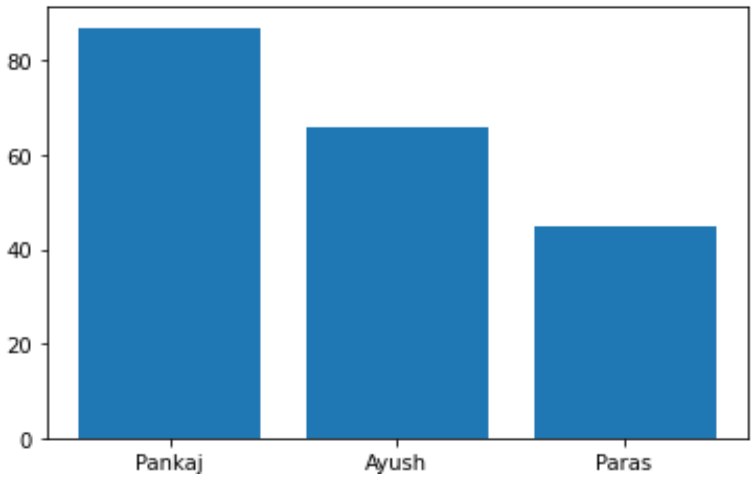
In [9]:
```
plt.plot(x,y1,color='b',linestyle='-',linewidth=4)
plt.plot(x,y2,color='r',linestyle=':',linewidth=2)
plt.title("Line Plot")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.grid(True)
plt.show()
```



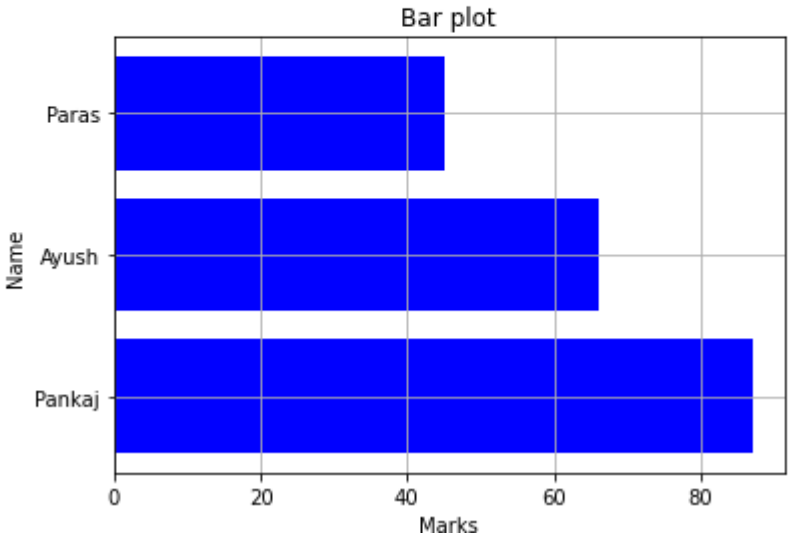
In [12]:
```
x=np.arange(1,11)
y1=2*x
y2=3*x


plt.subplot(1,2,1)
plt.plot(x,y1,color='b',linestyle='--',linewidth=3)


plt.subplot(1,2,2)
plt.plot(x,y2,color='r',linestyle=':',linewidth=3)

plt.show()
```



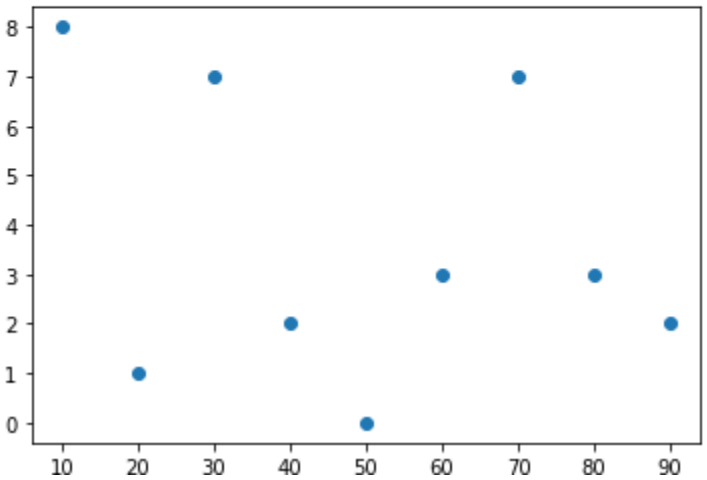
In [13]:
```
student={"Pankaj":87,"Ayush":66,"Paras":45}

names=list(student.keys())
values=list(student.values())

plt.bar(names,values)
plt.show()
```



In [15]:
```
plt.barh(names,values,color='b')
plt.title("Bar plot")
plt.xlabel("Marks")
plt.ylabel("Name")
plt.grid(True)
plt.show()
```



In [18]:
```
x=[10,20,30,40,50,60,70,80,90]
a=[8,1,7,2,0,3,7,3,2]

plt.scatter(x,a)
plt.show()
```



Numpy

```
In [1]: import numpy as np
```

```
In [2]: n1=np.array([10,20,30,40])
        n1
Out[2]: array([10, 20, 30, 40])

In [3]: n2=np.array([[10,20,30,40],[40,30,20,10]])
        n2
Out[3]: array([[10, 20, 30, 40],
               [40, 30, 20, 10]])

In [4]: type(n2)
Out[4]: numpy.ndarray

In [5]: n1=np.zeros((1,2))
        n1
Out[5]: array([[0., 0.]])

In [6]: n1=np.zeros((2,8))
        n1
Out[6]: array([[0., 0., 0., 0., 0., 0., 0., 0.],
               [0., 0., 0., 0., 0., 0., 0., 0.]])

In [9]: n1=np.full((3,4),10)
        n1
Out[9]: array([[10, 10, 10, 10],
               [10, 10, 10, 10],
               [10, 10, 10, 10]])
```

```
In [11]: n1=np.arange(10,21)
         n1
Out[11]: array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20])

In [12]: n1=np.arange(10,50,3)
         n1
Out[12]: array([10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49])

In [13]: n1=np.random.randint(1,80,7)
         n1
Out[13]: array([10, 19, 29, 72, 71, 32, 67])

In [14]: n1=np.random.randint(1,100,10)
         n1
Out[14]: array([62, 24, 79, 14, 30, 31, 85,  2, 87, 52])

In [17]: n1=np.array([[1,2,3,7],[4,5,6,9]])
         n1.shape
Out[17]: (2, 4)

In [20]: n1.shape=(4,2)
         n1.shape
Out[20]: (4, 2)

In [21]: n1=np.array([10,20,30])
         n2=np.array([40,50,60])
         np.vstack((n1,n2))
Out[21]: array([[10, 20, 30],
                [40, 50, 60]])
```

```
In [22]: n1=np.array([10,20,30])
         n2=np.array([40,50,60])
         np.hstack((n1,n2))
Out[22]: array([10, 20, 30, 40, 50, 60])

In [23]: n1=np.array([10,20,30,40])
         n2=np.array([40,50,60,70])
         np.column_stack((n1,n2))
Out[23]: array([[10, 40],
                [20, 50],
                [30, 60],
                [40, 70]])

In [24]: n1=np.array([10,20,30,40,50,60])
         n2=np.array([50,60,70,80,90])
         np.intersect1d(n1,n2)
Out[24]: array([50, 60])

In [25]: n1=np.array([10,20,30,40,50,60])
         n2=np.array([50,60,70,80,90])
         np.setdiff1d(n1,n2)
Out[25]: array([10, 20, 30, 40])

In [26]: n1=np.array([10,20,30,40,50,60,70])
         n2=np.array([50,60,70,80,90])
         np.setdiff1d(n2,n1)
Out[26]: array([80, 90])

In [27]: n1=np.array([10,20,30])
         n2=np.array([30,40,60])
         np.sum([n1,n2])
Out[27]: 190

In [28]: np.sum([n1,n2],axis=0)
Out[28]: array([40, 60, 90])

In [29]: np.sum([n1,n2],axis=1)
Out[29]: array([ 60, 130])

In [30]: n1=np.array([10,20,30])
         n1=n1+10
         n1
Out[30]: array([20, 30, 40])

In [31]: n1=np.array([10,20,30])
         n1=n1-19
         n1
Out[31]: array([-9,  1, 11])

In [32]: n1=np.array([10,20,30])
         n1=n1*20
         n1
Out[32]: array([200, 400, 600])

In [33]: n1=np.array([10,20,30])
         n1=n1/10
         n1
Out[33]: array([1., 2., 3.])
```

Pandas

```
In [1]: import pandas as pd
```

```
In [2]: s1=pd.Series([2,3,4,5,6,7])
        s1
```

```
Out[2]: 0    2
        1    3
        2    4
        3    5
        4    6
        5    7
        dtype: int64
```

```
In [3]: s1=pd.Series([1,2,3,4,5],index=['v','w','x','y','z'])
        s1
```

```
Out[3]: v    1
        w    2
        x    3
        y    4
        z    5
        dtype: int64
```

```
In [4]: pd.Series({'x':10,'y':20,'z':30})
```

```
Out[4]: x    10
        y    20
        z    30
        dtype: int64
```

```
In [5]: pd.Series({'a':10,'b':20,'c':30},index=['c','d','e','f'])
```

```
Out[5]: c    30.0
        d     NaN
        e     NaN
        f     NaN
        dtype: float64
```

```
In [6]: s1=pd.Series([1,2,3,4,5,6,7,8,9,10])
        s1[7]
```

```
Out[6]: 8
```

```
In [7]: s1=pd.Series([1,2,3,4,5,6,7,8,9,10])
        s1[:7]
```

```
Out[7]: 0    1
        1    2
        2    3
        3    4
        4    5
        5    6
        6    7
        dtype: int64
```

```
In [8]: s1=pd.Series([1,2,3,4,5,6,7,8,9,10])
        s1[-7:]
```

```
Out[8]: 3     4
        4     5
        5     6
        6     7
        7     8
        8     9
        9    10
        dtype: int64
```

```
In [10]: s1+10
```

```
Out[10]: 0    11
         1    12
         2    13
         3    14
         4    15
         5    16
         6    17
         7    18
         8    19
         9    20
         dtype: int64
```

In [12]:
```
s1=pd.Series([1,2,3,4,5,6,7,8,9,10])
s2=pd.Series([10,20,30,40,50,60,70,80,90])
s1+s2
```

Out[12]:
```
0    11.0
1    22.0
2    33.0
3    44.0
4    55.0
5    66.0
6    77.0
7    88.0
8    99.0
9     NaN
dtype: float64
```

In [13]:
```
pd.DataFrame({"Name":['Ayush','Yash','Harsh'],"Marks":[76,45,98]})
```

Out[13]:

	Name	Marks
0	Ayush	76
1	Yash	45
2	Harsh	98

In [14]:
```
df=pd.read_csv('Book1.csv')
df
```

Out[14]:

	S.no	Name	marks
0	1	priya	45
1	2	rohan	56
2	3	shina	78
3	4	rohit	67
4	5	anaya	68
5	6	neha	98
6	7	nimita	67
7	8	kunal	46
8	9	rajini	57
9	10	esha	34

In [16]:
```
df=pd.read_csv(r'C:\Users\ADMIN\Desktop\File123.csv')
df
```

Out[16]:

	S_no	name	marks
0	1	sanyukta	78
1	2	ayush	84
2	3	harsh	80
3	4	purnima	87
4	5	yash	76
5	6	paras	90
6	7	garrgi	96
7	8	mayur	89
8	9	vivekanad	74
9	10	pankaj	92

In [17]:
```
df.head()
```

Out[17]:

	S_no	name	marks
0	1	sanyukta	78
1	2	ayush	84
2	3	harsh	80
3	4	purnima	87
4	5	yash	76

In [18]:
```
df.tail()
```

Out[18]:

	S_no	name	marks
5	6	paras	90
6	7	garrgi	96
7	8	mayur	89
8	9	vivekanad	74
9	10	pankaj	92

In [19]:
```
df.describe()
```

Out[19]:

	S_no	marks
count	10.00000	10.000000
mean	5.50000	84.600000
std	3.02765	7.381659
min	1.00000	74.000000
25%	3.25000	78.500000
50%	5.50000	85.500000
75%	7.75000	89.750000
max	10.00000	96.000000

In [22]:
```
df.iloc[0:4,0:3]
```

Out[22]:

	S_no	name	marks
0	1	sanyukta	78
1	2	ayush	84
2	3	harsh	80
3	4	purnima	87

In [21]:
```
df.iloc[0:3,0:3]
```

Out[21]:

	S_no	name	marks
0	1	sanyukta	78
1	2	ayush	84
2	3	harsh	80

In [23]:
```
df.loc[0:3,("marks")]
```

Out[23]:
```
0    78
1    84
2    80
3    87
Name: marks, dtype: int64
```

In [24]:
```
df.drop('marks',axis=1)
```

Out[24]:

	S_no	name
0	1	sanyukta
1	2	ayush
2	3	harsh
3	4	purnima
4	5	yash
5	6	paras
6	7	garrgi
7	8	mayur
8	9	vivekanad
9	10	pankaj

In [26]:
```
df.drop([1,2],axis=0)
```

Out[26]:

	S_no	name	marks
0	1	sanyukta	78
3	4	purnima	87
4	5	yash	76
5	6	paras	90
6	7	garrgi	96
7	8	mayur	89
8	9	vivekanad	74
9	10	pankaj	92

In [27]:
```
df.mean()
```

Out[27]:
```
S_no      5.5
marks    84.6
dtype: float64
```

In [28]:
```
df.median()
```

Out[28]:
```
S_no      5.5
marks    85.5
dtype: float64
```

In [29]:
```
df2=df["marks"].median()
df2
```

Out[29]:
```
85.5
```

In [30]:
```
df1=df["marks"].min()
df1
```

Out[30]:
```
74
```

In [31]:
```
df.max()
```

Out[31]:
```
S_no       10
name     yash
marks      96
dtype: object
```

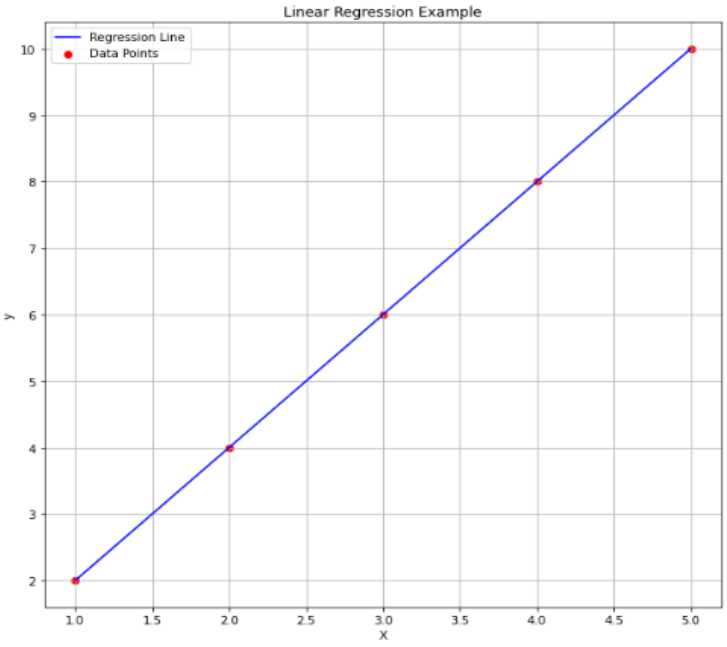
In [32]:
```
df2=df["marks"].max()
df2
```

Out[32]:
```
96
```

Scikit-Learn

In [1]:
```
#Scikit Learn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.cluster import KMeans
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [3]:
```
# Example 1: Linear Regression
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([2, 4, 6, 8, 10])
model = LinearRegression()
model.fit(X, y)
predictions = model.predict(X)

plt.figure(figsize=(10,10))
plt.scatter(X, y, color='red', label='Data Points')
plt.plot(X, predictions, color='blue', label='Regression Line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression Example')
plt.legend()
plt.grid()
plt.show()
```



MatPlotLib

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from scipy.integrate import quad
from scipy.stats import norm
```

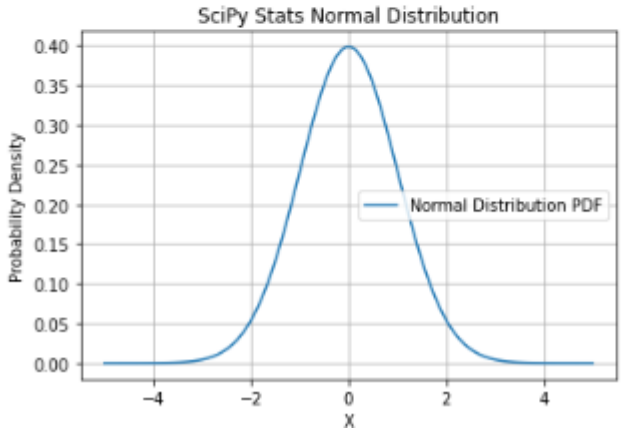
In [2]:
```
# Example 1: Optimization using scipy.optimize.minimize
def func(x):
    return (x - 3) ** 2 + 5

result = minimize(func, x0=0)
print("Optimization Result:", result)
```

```
Optimization Result:       fun: 5.000000000000001
 hess_inv: array([[0.5]])
      jac: array([5.96046448e-08])
  message: 'Optimization terminated successfully.'
     nfev: 6
      nit: 2
     njev: 3
   status: 0
  success: True
        x: array([3.00000003])
```

In [3]:
```
# Example 2: Numerical Integration using scipy.integrate.quad
def integrand(x):
    return np.exp(-x ** 2)

integral, error = quad(integrand, -np.inf, np.inf)
print("Integral of exp(-x^2) from -inf to inf:", integral)
```

```
Integral of exp(-x^2) from -inf to inf: 1.7724538509055159
```

In [4]:
```
# Example 3: Probability density function using scipy.stats.norm
x = np.linspace(-5, 5, 100)
pdf_values = norm.pdf(x, loc=0, scale=1)

plt.figure(figsize=(6,4))
plt.plot(x, pdf_values, label='Normal Distribution PDF')
plt.xlabel('X')
plt.ylabel('Probability Density')
plt.title('SciPy Stats Normal Distribution')
plt.legend()
plt.grid()
plt.show()
```



Conclusion: Successfully Learned all basic xPython Libraries

Practical No. 5

AIM:-Implementation of Perceptron algorithm for OR operation.

Objective:-

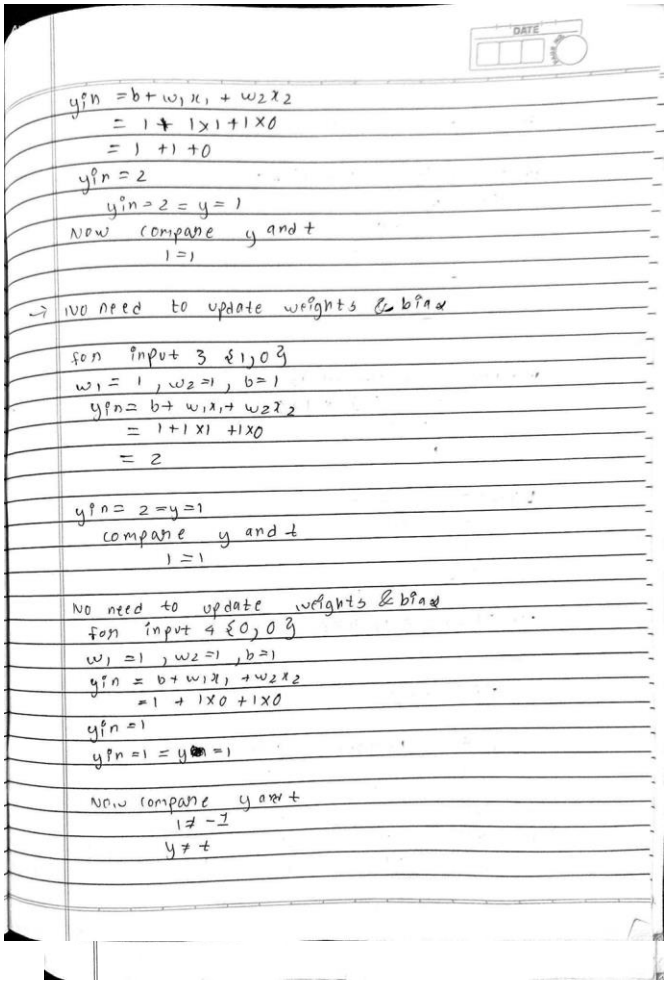
- Understanding of Perceptron.
- Understanding of OR Operation using Perceptron.
- Implementation of Perceptron for OR operation

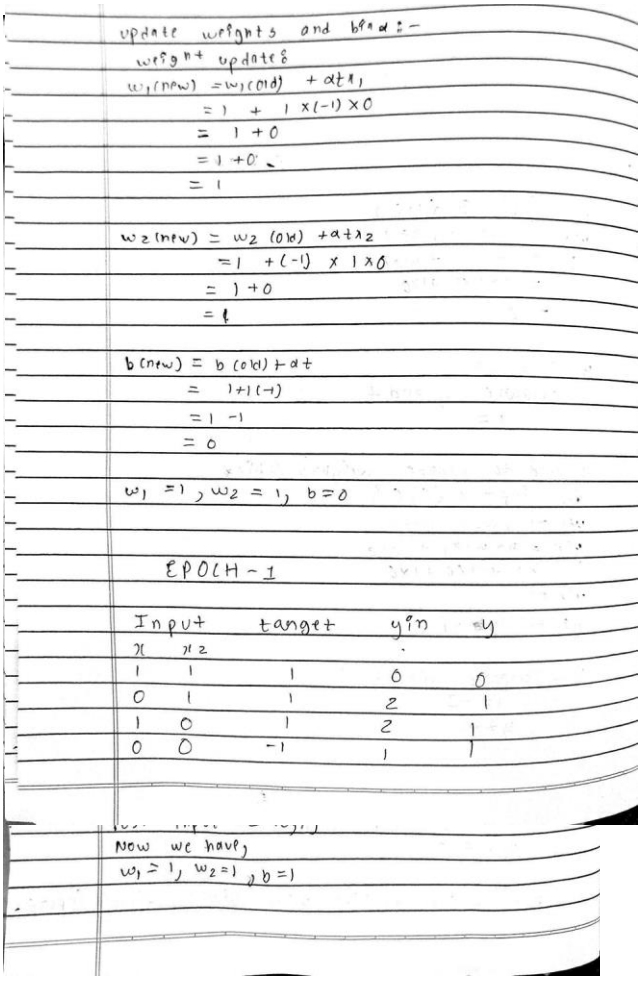
Theory:-

- Introduction of Perceptron

The perceptron is a fundamental artificial neural network model introduced by Frank Rosenblatt in 1958. It serves as a binary classifier that processes input features through weighted connections, applies an activation function, and predicts output. The perceptron is the foundation of modern deep learning and machine learning advancements.

- Perceptron algorithm for OR operation



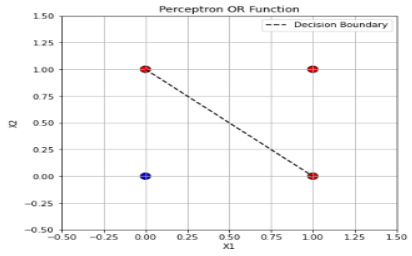


Code:

```
# required Libraries
import numpy as np
import matplotlib.pyplot as plt
# input features and targets (bipolar targets)
X = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
t = np.array([1, 1, 1, -1])
# Initialization of weights and bias
weights = np.zeros(2)
bias = 0
learning_rate = 0.1
# Perceptron training algorithm for 3 epochs
for epoch in range(3):
    print(f"Epoch {epoch + 1}")
    for i in range(len(X)):
        # Compute activation
        activation = np.dot(X[i], weights) + bias
        output = 1 if activation >= 0 else -1

        # Update weights and bias if there's an error
        if output != t[i]:
            weights += learning_rate * t[i] * X[i]
            bias += learning_rate * t[i]

        print(f"Input: {X[i]}, Target: {t[i]}, Output: {output}, Weights: {weights}, Bias:
    {bias}") print("-")

# Final weights and bias after training
print("Final Weights:", weights)
print("Final Bias:", bias)

# Plot decision boundary
x_values = np.array([0, 1])
y_values = (-weights[0] * x_values - bias) / weights[1]

plt.figure(figsize=(6,6))
plt.scatter(X[:, 0], X[:, 1], c=t, cmap='bwr', edgecolors='k',
s=100) plt.plot(x_values, y_values, 'k--', label='Decision
Boundary') plt.xlim(-0.5, 1.5)
plt.ylim(-0.5, 1.5)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Perceptron OR Function')
plt.legend()
plt.grid()
plt.show()
```

Output:

```
Epoch 1
Input: [1 1], Target: 1, Output: 1, Weights: [0. 0.], Bias: 0
Input: [1 0], Target: 1, Output: 1, Weights: [0. 0.], Bias: 0
Input: [0 1], Target: 1, Output: 1, Weights: [0. 0.], Bias: 0
Input: [0 0], Target: -1, Output: 1, Weights: [0. 0.], Bias: -0.1
-
Epoch 2
Input: [1 1], Target: 1, Output: -1, Weights: [0.1 0.1], Bias: 0.0
Input: [1 0], Target: 1, Output: 1, Weights: [0.1 0.1], Bias: 0.0
Input: [0 1], Target: 1, Output: 1, Weights: [0.1 0.1], Bias: 0.0
Input: [0 0], Target: -1, Output: 1, Weights: [0.1 0.1], Bias: -0.1
-
Epoch 3
Input: [1 1], Target: 1, Output: 1, Weights: [0.1 0.1], Bias: -0.1
Input: [1 0], Target: 1, Output: 1, Weights: [0.1 0.1], Bias: -0.1
Input: [0 1], Target: 1, Output: 1, Weights: [0.1 0.1], Bias: -0.1
Input: [0 0], Target: -1, Output: -1, Weights: [0.1 0.1], Bias: -0.1
-
Final Weights: [0.1 0.1]
Final Bias: -0.1
```



Conclusion: Successfully Implemented Perceptron algorithm for OR operation.

Practical No. 6

AIM:-Implementation of ADALINE algorithm for AND operation.

Objective:-

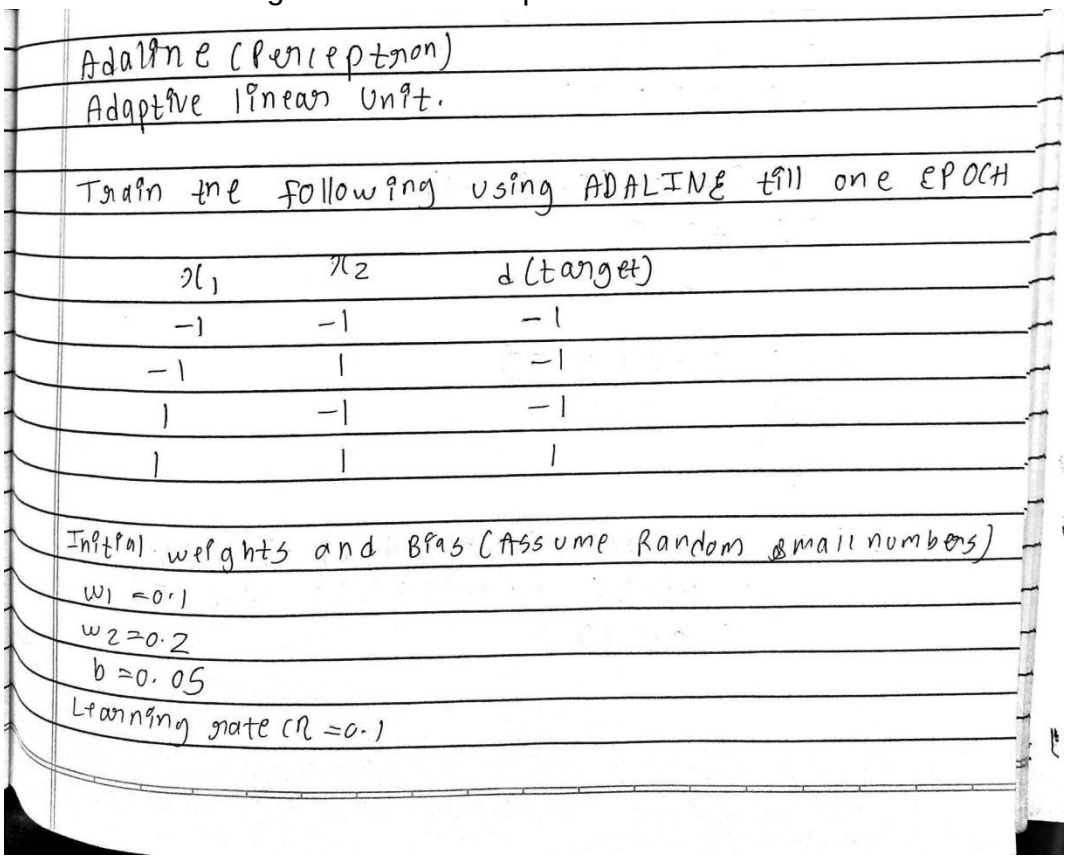
- Understanding of ADALINE.
- Understanding of AND Operation using ADALINE.
- Implementation of ADALINE for AND operation

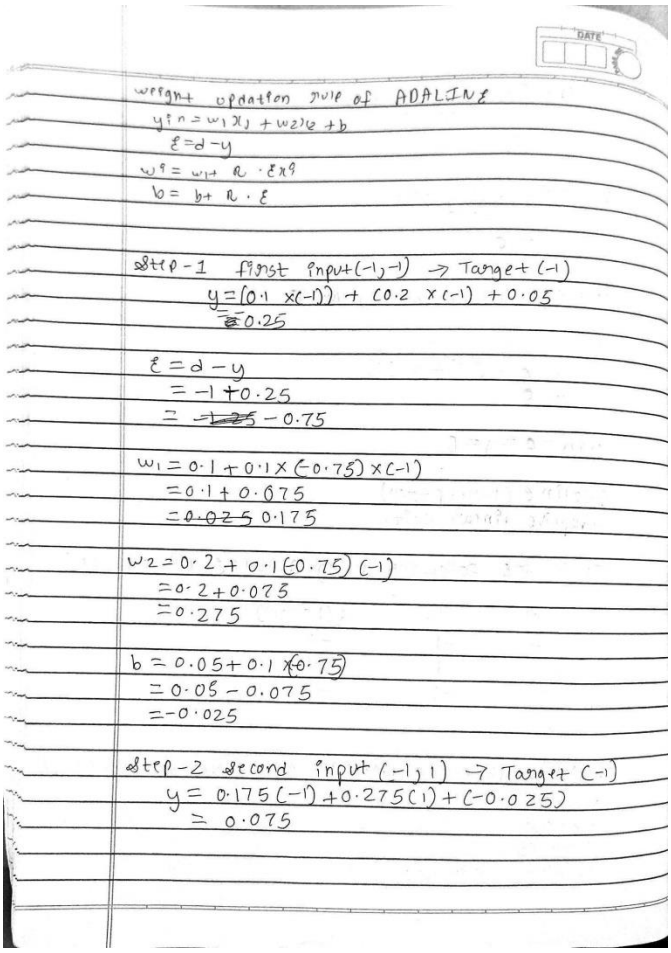
Theory:-

- Introduction of ADALINE

ADALINE (Adaptive Linear Neuron) is a single-layer neural network model developed by Bernard Widrow and Ted Hoff in 1960. It uses weighted inputs and applies the linear activation function before error correction via the Least Mean Squares (LMS) algorithm, making it effective in pattern recognition and signal processing tasks.

- ADALINE algorithm for AND operation





weight updation rule of ADALINE

$y_{in} = w_1 x_1 + w_2 x_2 + b$

$\varepsilon = d - y$

$w_i = w_{i+} \alpha \cdot \varepsilon x_i$

$b = b + \alpha \cdot \varepsilon$

Step-1 first input (-1, -1) → Target (-1)

$y = (0.1 \times (-1)) + (0.2 \times (-1)) + 0.05$

$= -0.25$

$\varepsilon = d - y$

$= -1 + 0.25$

$= -0.75$

$w_1 = 0.1 + 0.1 \times (-0.75) \times (-1)$

$= 0.1 + 0.075$

$= 0.175$

$w_2 = 0.2 + 0.1 (-0.75) (-1)$

$= 0.2 + 0.075$

$= 0.275$

$b = 0.05 + 0.1 (-0.75)$

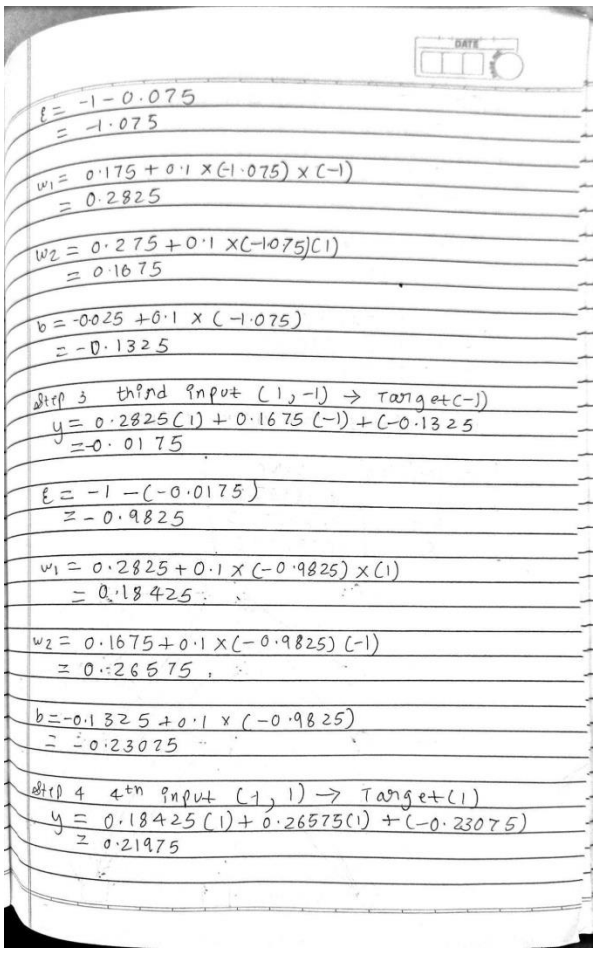
$= 0.05 - 0.075$

$= -0.025$

Step-2 second input (-1, 1) → Target (-1)

$y = 0.175 (-1) + 0.275 (1) + (-0.025)$

$= 0.075$



$\varepsilon = -1 - 0.075$

$= -1.075$

$w_1 = 0.175 + 0.1 \times (-1.075) \times (-1)$

$= 0.2825$

$w_2 = 0.275 + 0.1 \times (-1.075) (1)$

$= 0.1675$

$b = -0.025 + 0.1 \times (-1.075)$

$= -0.1325$

Step 3 third input (1, -1) → Target (-1)

$y = 0.2825 (1) + 0.1675 (-1) + (-0.1325$

$= -0.0175$

$\varepsilon = -1 - (-0.0175)$

$= -0.9825$

$w_1 = 0.2825 + 0.1 \times (-0.9825) \times (1)$

$= 0.18425$

$w_2 = 0.1675 + 0.1 \times (-0.9825) (-1)$

$= 0.26575$

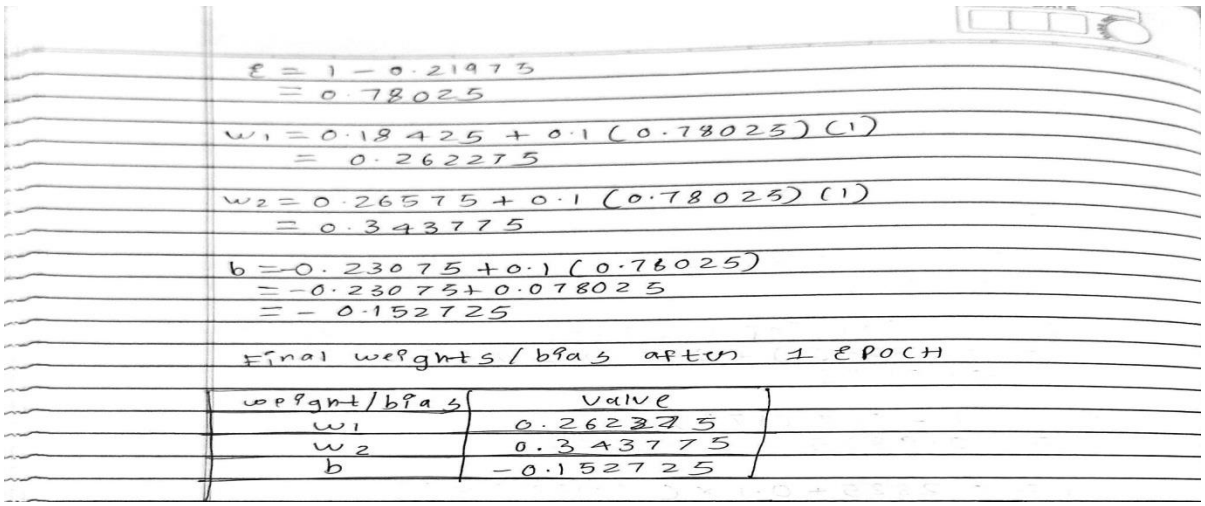
$b = -0.1325 + 0.1 \times (-0.9825)$

$= -0.23075$

Step 4 4th input (-1, 1) → Target (1)

$y = 0.18425 (1) + 0.26575 (1) + (-0.23075)$

$= 0.21975$



$\varepsilon = 1 - 0.21975$

$= 0.78025$

$w_1 = 0.18425 + 0.1 (0.78025) (1)$

$= 0.262275$

$w_2 = 0.26575 + 0.1 (0.78025) (1)$

$= 0.343775$

$b = -0.23075 + 0.1 (0.78025)$

$= -0.23075 + 0.078025$

$= -0.152725$

Final weights / bias after 1 epoch

weight / bias	value
w_1	0.262275
w_2	0.343775
b	-0.152725

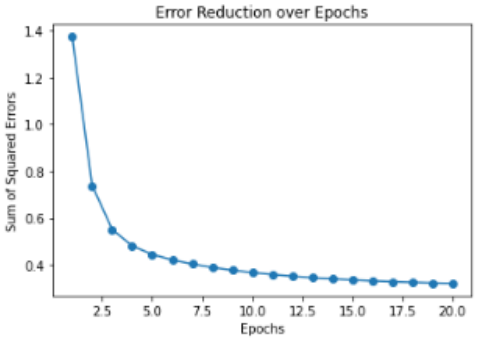
Code:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Define AND gate inputs and expected outputs
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # Input features
y = np.array([0, 0, 0, 1]) # Expected output

# Add bias term
X = np.c_[np.ones((X.shape[0], 1)), X] # Adding bias as the first column

# Initialize weights
weights = np.random.rand(X.shape[1])
learning_rate = 0.1
epochs = 20

# Activation function (Identity function for Adaline)
def activation_function(net_input):
    return net_input

# Training Adaline using gradient descent
errors = []
for epoch in range(epochs):
    total_error = 0
    for i in range(X.shape[0]):
        net_input = np.dot(X[i], weights) # Compute net input
        output = activation_function(net_input) # Compute activation output
        error = y[i] - output # Compute error
        weights += learning_rate * error * X[i] # Update weights
        total_error += error**2 # Sum of squared errors
    errors.append(total_error)
    print(f"Epoch {epoch+1}, Error: {total_error}")
# Plot error reduction
epochs_range = range(1, epochs + 1)
plt.plot(epochs_range, errors, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Sum of Squared Errors')
plt.title('Error Reduction over Epochs')
plt.show()
# Testing the trained Adaline model
print("Trained Weights:", weights)
print("Testing the AND operation:")
for i in range(X.shape[0]):
    net_input = np.dot(X[i], weights)
    output = 1 if net_input >= 0.5 else 0 # Threshold function for classification
    print(f"Input: {X[i][1:]}, Output: {output}")
```

Output:

```
Epoch 1, Error: 1.3741619888648047
Epoch 2, Error: 0.7387932160934976
Epoch 3, Error: 0.551395361033244
Epoch 4, Error: 0.48098200686831316
Epoch 5, Error: 0.4449623176241849
Epoch 6, Error: 0.42134073743721356
Epoch 7, Error: 0.4035463683681465
Epoch 8, Error: 0.38923803319456773
Epoch 9, Error: 0.37737809720752546
Epoch 10, Error: 0.3673988784343083
Epoch 11, Error: 0.3589347277894289
Epoch 12, Error: 0.35172281245596715
Epoch 13, Error: 0.345560813103316
Epoch 14, Error: 0.3402864277008161
Epoch 15, Error: 0.335766222794303
Epoch 16, Error: 0.33188888670261313
Epoch 17, Error: 0.3285607398564845
Epoch 18, Error: 0.32570250639968285
Epoch 19, Error: 0.3232468560873417
Epoch 20, Error: 0.3211364586391917
```




```
Trained Weights: [-0.1900164   0.5095328   0.44300605]
Testing the AND operation:
Input: [0. 0.], Output: 0
Input: [0. 1.], Output: 0
Input: [1. 0.], Output: 0
Input: [1. 1.], Output: 1
```

Conclusion: Successfully Implemented ADALINE algorithm for AND operation

Practical No. 7

AIM:- Improve the prediction Accuracy by estimating the weight values for the training data using Stochastic Gradient Descent(Perceptron)

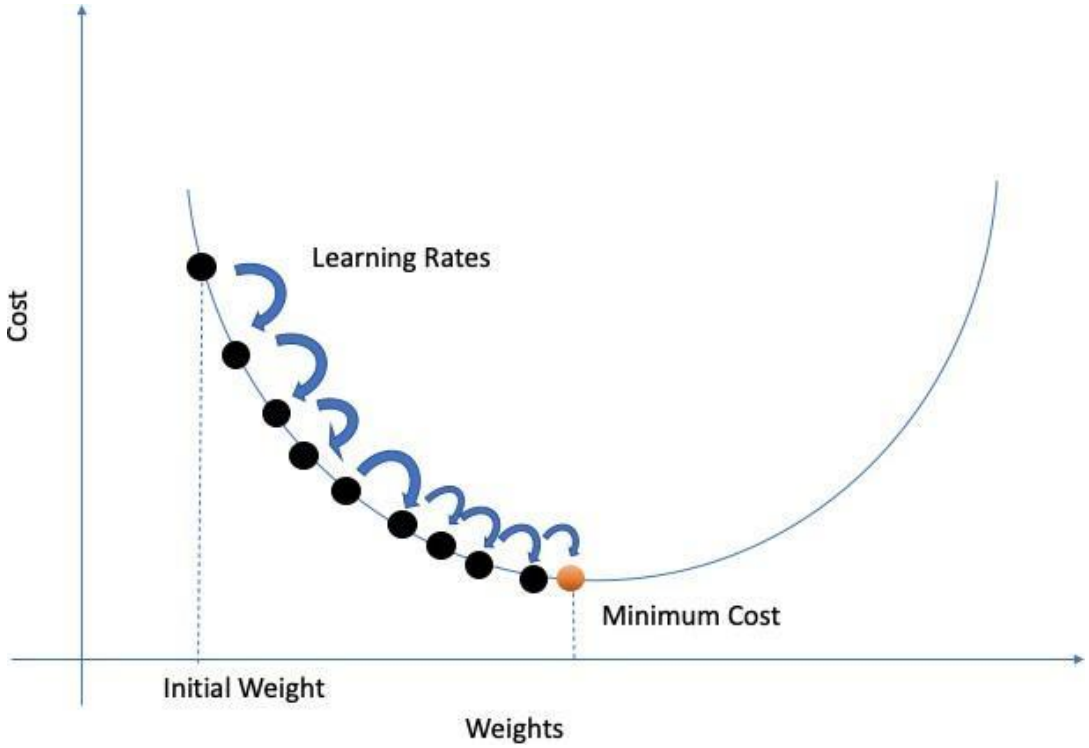
Objective:-

- Understanding of Stochasting Gradient Descent.
- Improving of prediction Accuracy using SGD.

Theory:-

- Stochastic Gradient Descent

  Stochastic Gradient Descent (SGD) is an optimization algorithm used in machine learning to minimize a model's error. Unlike standard gradient descent, SGD updates model parameters using a single data sample at a time, improving efficiency for large datasets and enhancing convergence speed, especially in deep learning applications.



Advantages of SGD

✓ Efficient for large datasets

✓ Reduces memory requirements since only one data point is processed at a time

✓ Can escape sharp local minima due to its noisy updates

Disadvantages of SGD

- ! Fluctuations in updates may slow convergence
- ! Requires careful tuning of the learning rate
- ! May struggle with very noisy data without techniques like momentum or learning rate schedules

Code:

[1]

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate a more complex dataset with noise and overlapping classes
X, y = make_classification(n_samples=200, n_features=2, n_classes=2,
                           n_redundant=0, n_clusters_per_class=1, class_sep=0.05, flip_y=0.3, random_state=42)

# Convert labels from {0,1} to {-1,1} for perceptron
y = np.where(y == 0, -1, 1)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

class PerceptronBatch:
    def _init_(self, learning_rate=0.0001, epochs=20): # Further reduced learning rate and epochs
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for epoch in range(self.epochs):
            total_error = 0
            for i in range(n_samples):
                linear_output = np.dot(X[i], self.weights) + self.bias
                y_predicted = np.where(linear_output >= 0, 1, -1)

                if y_predicted != y[i]:
                    total_error += y[i] - y_predicted

            # Apply weaker weight updates with decay
            decay = 1 / (epoch + 1) # Reduce learning rate over epochs
            self.weights += (self.learning_rate * total_error * np.mean(X, axis=0)) * decay
```

```
self.bias += (self.learning_rate * total_error) * decay
```

```
    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return np.where(linear_output >= 0, 1, -1)

# Train Perceptron with Batch Learning
perceptron = PerceptronBatch(learning_rate=0.0001, epochs=20)
perceptron.fit(X_train, y_train)

y_pred = perceptron.predict(X_test)
accuracy = np.mean(y_pred == y_test)
print(f"Prediction Accuracy with Batch Perceptron: {accuracy * 100:.2f}%")

# Plot decision boundary
def plot_decision_boundary(X, y, model, title):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k')
    plt.title(title)
    plt.show()

plot_decision_boundary(X_test, y_test, perceptron, "Perceptron with Batch Learning (Lower Accuracy)")
```

[2]

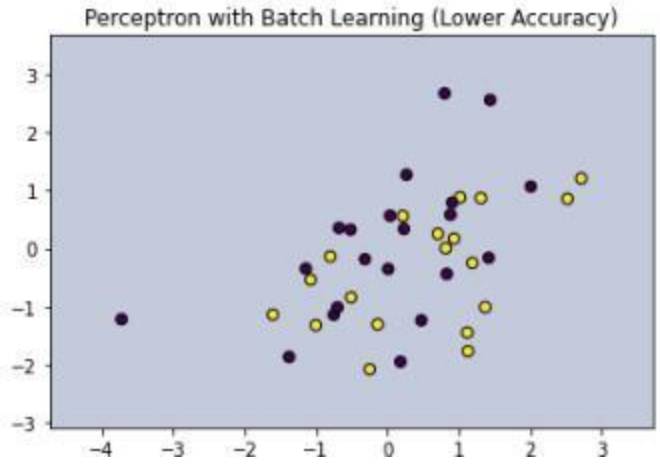
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate a synthetic dataset
X, y = make_classification(n_samples=200, n_features=2, n_classes=2,
                           n_redundant=0, n_clusters_per_class=1, class_sep=0.5, random_state=42)

# Convert labels from {0,1} to {-1,1} for perceptron
y = np.where(y == 0, -1, 1)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

class PerceptronSGD:
    def __init__(self, learning_rate=0.01, epochs=50):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None
```

```
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for epoch in range(self.epochs):
            for i in range(n_samples):
                linear_output = np.dot(X[i], self.weights) + self.bias
                y_predicted = np.where(linear_output >= 0, 1, -1)

                # Update rule
                update = self.learning_rate * (y[i] - y_predicted)
                self.weights += update * X[i]
                self.bias += update

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return np.where(linear_output >= 0, 1, -1)

# Train Perceptron with SGD
perceptron = PerceptronSGD(learning_rate=0.01, epochs=50)
perceptron.fit(X_train, y_train)

y_pred = perceptron.predict(X_test)
accuracy = np.mean(y_pred == y_test)
print(f"Prediction Accuracy with Stochastic Gradient Descent: {accuracy * 100:.2f}%")

# Plot decision boundary
def plot_decision_boundary(X, y, model, title):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k')
    plt.title(title)
    plt.show()

plot_decision_boundary(X_test, y_test, perceptron, "Perceptron with Stochastic Gradient Descent")
```

Output:

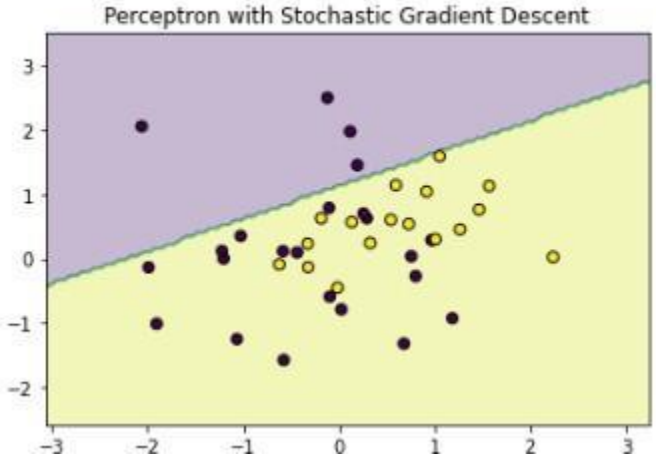
[1]

```
Prediction Accuracy with Batch Perceptron: 47.50%
```



[2]

```
Prediction Accuracy with Stochastic Gradient Descent: 52.50%
```



Conclusion: Successfully Improved accuracy by using Stochastic Gradient Descent

Practical No. 8

Aim:-Implementation of feature Extraction,selection,Normalization,Transformation,Principal Component Analysis

Objectives:-

- Understanding of features Extraction,selection,Normalization,Transformation and Reduction.
- Understanding Principal component Analysis(PCA)
- Implementation of above techniques in Python.

Theory:-all above mentioned terms Definition with Example

1) **Feature Extraction**

- The process of deriving new features from raw data to improve model performance. It involves transforming data into a format that is more meaningful for machine learning algorithms. Examples include edge detection in images or word embeddings in NLP.
- Example:

2) **Feature Selection**

- The process of selecting the most relevant features from a dataset while removing redundant or irrelevant ones. This improves model performance and reduces overfitting. Common methods include filter methods (e.g., correlation), wrapper methods (e.g., recursive feature elimination), and embedded methods (e.g., LASSO regression).

3) **Feature Normalization**

- The process of scaling numerical features to a common range, typically [0,1] or [-1,1], to ensure that all features contribute equally to the model. Common techniques include Min-Max Scaling and Z-score Standardization.

4) **Feature Transformation**

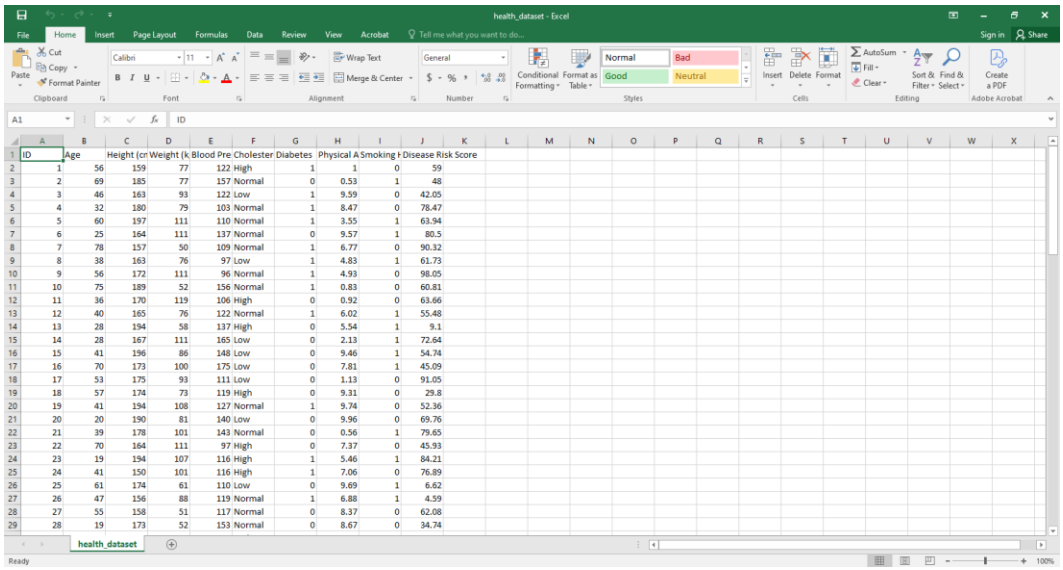
- The process of modifying or encoding features to make them more suitable for machine learning models. This includes techniques like logarithmic scaling, polynomial transformations, encoding categorical variables, and applying mathematical functions.

5) **Principal Component Analysis (PCA)**

- A dimensionality reduction technique that transforms a high-dimensional dataset into a lower-dimensional space by identifying the principal components (new axes) that capture the most variance in the data. PCA helps reduce complexity while retaining important information.

Code & Output:-

Dataset Description:



ID	Age	Height (cm	Weight (k	Blood Pre	Cholester	Diabetes	Physical A	Smoking H	Disease Risk Score
1	56	159	77	122	High	1	1	0	59
2	69	185	77	157	Normal	0	0.53	1	48
3	46	163	93	122	Low	1	9.59	0	42.05
4	32	180	79	103	Normal	1	8.47	0	78.47
5	60	197	111	110	Normal	1	3.55	1	63.94
6	25	164	111	137	Normal	0	9.57	1	80.5
7	78	157	50	109	Normal	1	6.77	0	90.32
8	38	163	76	97	Low	1	4.83	1	61.73
9	56	172	111	96	Normal	1	4.93	0	98.05
10	75	189	52	156	Normal	1	0.83	0	60.81
11	36	170	119	106	High	0	0.92	0	63.66
12	40	165	76	122	Normal	1	6.02	1	55.48
13	28	194	58	137	High	0	5.54	1	9.1
14	28	167	111	165	Low	0	2.13	1	72.64
15	41	196	86	148	Low	0	9.46	1	54.74
16	70	173	100	175	Low	0	7.81	1	45.09
17	53	175	93	111	Low	0	1.13	0	91.05
18	57	174	73	119	High	0	9.31	0	29.8
19	41	194	108	127	Normal	1	9.74	0	52.36
20	20	190	81	140	Low	0	9.96	0	69.76
21	39	178	101	143	Normal	0	0.56	1	79.65
22	70	164	111	97	High	0	7.37	0	45.93
23	19	194	107	116	High	1	5.46	1	84.21
24	41	150	101	116	High	1	7.06	0	76.89
25	61	174	61	110	Low	0	9.69	1	6.62
26	47	156	88	119	Normal	1	6.88	1	4.59
27	55	158	51	117	Normal	0	8.37	0	62.08
28	19	173	52	153	Normal	0	8.67	0	34.74

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder, PowerTransformer
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
```

In [2]:
```
df = pd.read_csv("health_dataset.csv")
```

In [3]:
```
num_cols = df.select_dtypes(include=['number']).columns.tolist()
cat_cols = df.select_dtypes(include=['object']).columns.tolist()
```

In [4]:
```
num_imputer = SimpleImputer(strategy="mean")
cat_imputer = SimpleImputer(strategy="most_frequent")
```

In [5]:
```
df[num_cols] = num_imputer.fit_transform(df[num_cols])
df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])
```

In [6]:
```
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 10 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   ID                              100 non-null    float64
 1   Age                             100 non-null    float64
 2   Height (cm)                     100 non-null    float64
 3   Weight (kg)                     100 non-null    float64
 4   Blood Pressure (BP)             100 non-null    float64
 5   Cholesterol Level               100 non-null    object
 6   Diabetes                        100 non-null    float64
 7   Physical Activity (hours/week)  100 non-null    float64
 8   Smoking Habit                   100 non-null    float64
 9   Disease Risk Score              100 non-null    float64
dtypes: float64(9), object(1)
memory usage: 7.9+ KB
```

Out[6]:

	ID	Age	Height (cm)	Weight (kg)	Blood Pressure (BP)	Cholesterol Level	Diabetes	Physical Activity (hours/week)	Smoking Habit	Disease Risk Score
0	1.0	56.0	159.0	77.0	122.0	High	1.0	1.00	0.0	59.00
1	2.0	69.0	185.0	77.0	157.0	Normal	0.0	0.53	1.0	48.00
2	3.0	46.0	163.0	93.0	122.0	Low	1.0	9.59	0.0	42.05
3	4.0	32.0	180.0	79.0	103.0	Normal	1.0	8.47	0.0	78.47
4	5.0	60.0	197.0	111.0	110.0	Normal	1.0	3.55	1.0	63.94

In [7]:
```
le = LabelEncoder()
for col in cat_cols:
    df[col] = le.fit_transform(df[col])
```

In [8]:
```
df.head()
```

Out[8]:

	ID	Age	Height (cm)	Weight (kg)	Blood Pressure (BP)	Cholesterol Level	Diabetes	Physical Activity (hours/week)	Smoking Habit	Disease Risk Score
0	1.0	56.0	159.0	77.0	122.0	0	1.0	1.00	0.0	59.00
1	2.0	69.0	185.0	77.0	157.0	2	0.0	0.53	1.0	48.00
2	3.0	46.0	163.0	93.0	122.0	1	1.0	9.59	0.0	42.05
3	4.0	32.0	180.0	79.0	103.0	2	1.0	8.47	0.0	78.47
4	5.0	60.0	197.0	111.0	110.0	2	1.0	3.55	1.0	63.94

In [9]:
```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

In [10]:
```
k_best = min(5, X.shape[1])
```

In [11]:
```
selector = SelectKBest(score_func=f_classif, k=k_best)
X_selected = selector.fit_transform(X, y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\feature_selection\_univariate_selection.py:111: RuntimeWarning: invalid valu
e encountered in true_divide
  msw = sswn / float(dfwn)
```

In [12]:
```
selected_features = X.columns[selector.get_support()]
df_selected = pd.DataFrame(X_selected, columns=selected_features)
```

In [13]:
```
df_selected.head()
```

Out[13]:

	Blood Pressure (BP)	Cholesterol Level	Diabetes	Physical Activity (hours/week)	Smoking Habit
0	122.0	0.0	1.0	1.00	0.0
1	157.0	2.0	0.0	0.53	1.0
2	122.0	1.0	1.0	9.59	0.0
3	103.0	2.0	1.0	8.47	0.0
4	110.0	2.0	1.0	3.55	1.0

In [14]:
```
scaler_minmax = MinMaxScaler()
X_minmax = scaler_minmax.fit_transform(df_selected)
```

In [15]:
```
scaler_std = StandardScaler()
X_standard = scaler_std.fit_transform(df_selected)
```

In [16]:
```
df_minmax = pd.DataFrame(X_minmax, columns=selected_features)
df_standard = pd.DataFrame(X_standard, columns=selected_features)
```

In [17]:
```
df_minmax.head(), df_standard.head()
```

Out[17]:
```
(   Blood Pressure (BP)  Cholesterol Level  Diabetes  \
0             0.352273                0.0       1.0
1             0.750000                1.0       0.0
2             0.352273                0.5       1.0
3             0.136364                1.0       1.0
4             0.215909                1.0       1.0

   Physical Activity (hours/week)  Smoking Habit
0                        0.070539            0.0
1                        0.021784            1.0
2                        0.961618            0.0
3                        0.845436            0.0
4                        0.335062            1.0  ,
   Blood Pressure (BP)  Cholesterol Level  Diabetes  \
0            -0.358213          -1.451747  1.083473
1             1.161477           1.073030 -0.922958
2            -0.358213          -0.189358  1.083473
3            -1.183187           1.073030  1.083473
4            -0.879249           1.073030  1.083473

   Physical Activity (hours/week)  Smoking Habit
0                       -1.420558      -0.850963
1                       -1.582102       1.175139
2                        1.531920      -0.850963
3                        1.146964      -0.850963
4                       -0.544095       1.175139  )
```

In [18]:
```
df_log = pd.DataFrame(np.log1p(df_selected), columns=selected_features)
```

In [19]:
```
pt = PowerTransformer(method='yeo-johnson')
df_boxcox = pd.DataFrame(pt.fit_transform(df_selected), columns=selected_features)
```

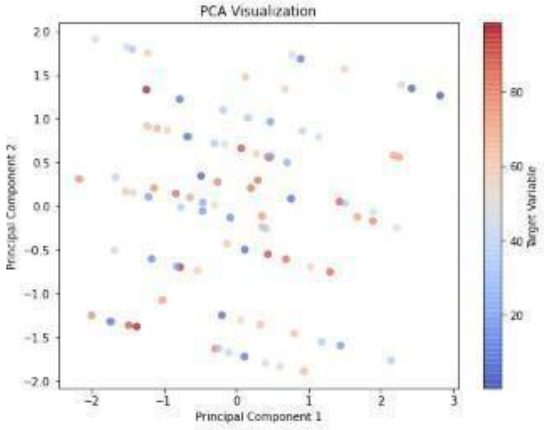
In [20]:
```
df_log.head(), df_boxcox.head()
```

Out[20]:
```
(   Blood Pressure (BP)  Cholesterol Level  Diabetes  \
 0              4.812184           0.000000  0.693147
 1              5.062595           1.098612  0.000000
 2              4.812184           0.693147  0.693147
 3              4.644391           1.098612  0.693147
 4              4.709530           1.098612  0.693147

    Physical Activity (hours/week)  Smoking Habit
 0                        0.693147       0.000000
 1                        0.425268       0.693147
 2                        2.359910       0.000000
 3                        2.248129       0.000000
 4                        1.515127       0.693147  ,
    Blood Pressure (BP)  Cholesterol Level  Diabetes  \
 0            -0.268326          -1.436146  1.083473
 1             1.145294           1.082518 -0.922958
 2            -0.268326          -0.211344  1.083473
 3            -1.256694           1.082518  1.083473
 4            -0.868983           1.082518  1.083473

    Physical Activity (hours/week)  Smoking Habit
 0                       -1.528636      -0.850963
 1                       -1.768575       1.175139
 2                        1.426669      -0.850963
 3                        1.112370      -0.850963
 4                       -0.464438       1.175139  )
```

In [21]:
```
pca_components = min(2, df_selected.shape[1])
pca = PCA(n_components=pca_components)
X_pca = pca.fit_transform(df_standard)
```

In [22]:
```
df_pca = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(X_pca.shape[1])])
```

In [25]:
```
if df_pca.shape[1] >= 2:
    plt.figure(figsize=(8,6))
    plt.scatter(df_pca.iloc[:,0], df_pca.iloc[:,1], c=y, cmap='coolwarm', alpha=0.7)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.title('PCA Visualization')
    plt.colorbar(label='Target Variable')
    plt.show()
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
```

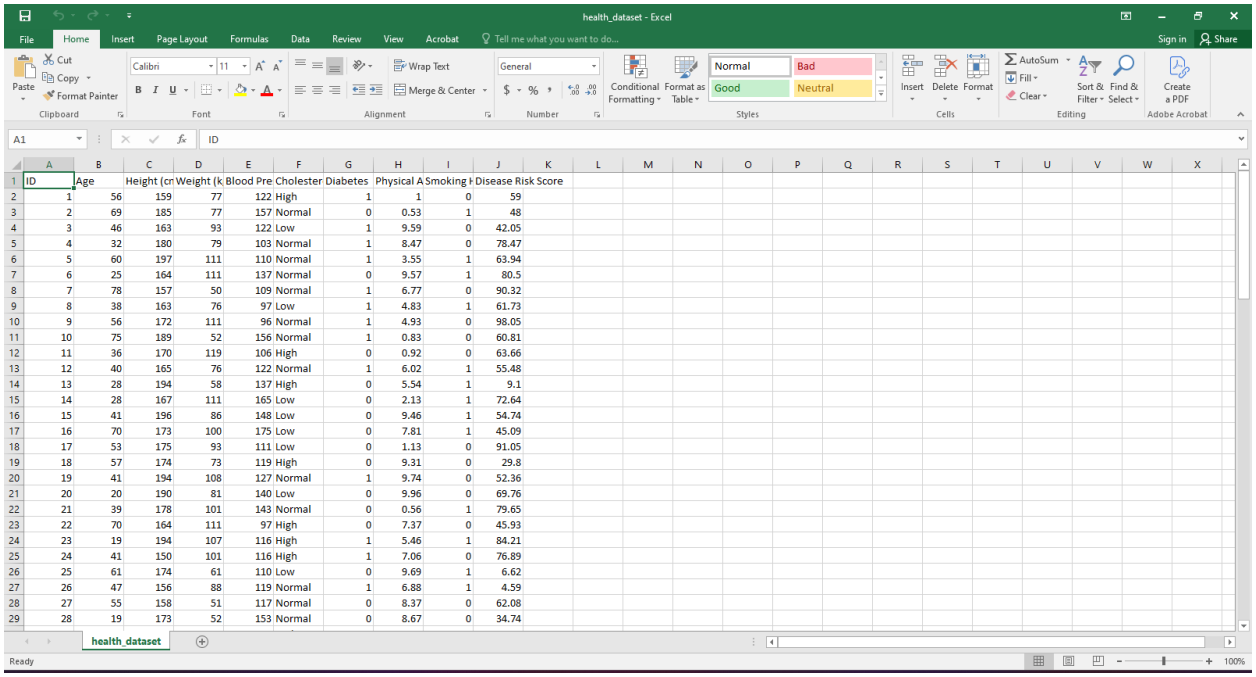



```
Explained Variance Ratio: [0.2730915  0.21075584]
```

In [26]:
```
processed_df = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(X_pca.shape[1])])
processed_df['Disease Risk Score'] = y.reset_index(drop=True)

processed_df.to_csv("processed_health_dataset.csv", index=False)
print("Processed dataset saved successfully.")
```

```
Processed dataset saved successfully.
```

Processed Dataset:



ID	Age	Height (cm	Weight (k	Blood Pre	Cholester	Diabetes	Physical A	Smoking H	Disease Risk Score
1	56	159	77	122	High	1	1	0	59
2	69	185	77	157	Normal	0	0.53	1	48
3	46	163	93	122	Low	1	9.59	0	42.05
4	32	180	79	103	Normal	1	8.47	0	78.47
5	60	197	111	110	Normal	1	3.55	1	63.94
6	25	164	111	137	Normal	0	9.57	1	80.5
7	78	157	50	109	Normal	1	6.77	0	90.32
8	38	163	76	97	Low	1	4.83	1	61.73
9	56	172	111	96	Normal	1	4.93	0	98.05
10	75	189	52	156	Normal	1	0.83	0	60.81
11	36	170	119	106	High	0	0.92	0	63.66
12	40	165	76	122	Normal	1	6.02	1	55.48
13	28	194	58	137	High	0	5.54	1	9.1
14	28	167	111	165	Low	0	2.13	1	72.64
15	41	196	86	148	Low	0	9.46	1	54.74
16	70	173	100	175	Low	0	7.81	1	45.09
17	53	175	93	111	Low	0	1.13	0	91.05
18	57	174	73	119	High	0	9.31	0	29.8
19	41	194	108	127	Normal	1	9.74	0	52.36
20	20	190	81	140	Low	0	9.96	0	69.76
21	39	178	101	143	Normal	0	0.56	1	79.65
22	70	164	111	97	High	0	7.37	0	45.93
23	19	194	107	116	High	1	5.46	1	84.21
24	41	150	101	116	High	1	7.06	0	76.89
25	61	174	61	110	Low	0	9.69	1	6.62
26	47	156	88	119	Normal	1	6.88	1	4.59
27	55	158	51	117	Normal	0	8.37	0	62.08
28	19	173	52	153	Normal	0	8.67	0	34.74

Conclusion:-Successfully Implemented feature Extraction, Selection, Normalization, Transformation, PCA.

Practical No. 9

Aim: Implement a python program to demonstrate Logistic regression

Objectives:

Understanding of Non-Linear Regression.

Understanding of Logistic Regression.

Implementation of Logistic Regression using PYTHON

Theory:

Nonlinear Regression

A type of regression where the relationship between the independent and dependent variables is modeled using a **nonlinear function**.

Use: When data doesn't follow a straight-line pattern.

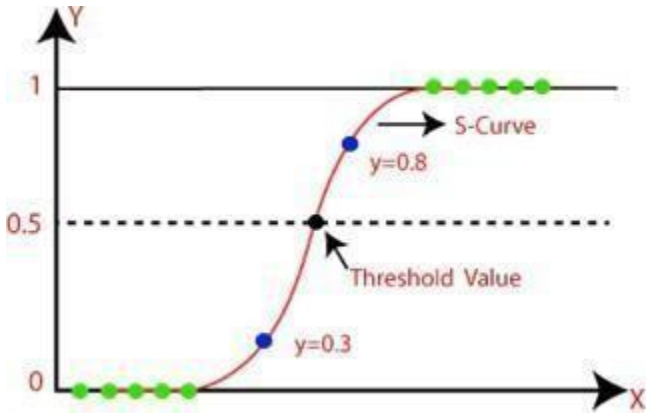
Example: Modeling population growth using an exponential curve:

1. $y=a \cdot e^{bx}$,where y is population and x is time.

Logistic Regression

A **classification algorithm**, not a true regression, used to predict **binary outcomes** (e.g., yes/no, 0/1).Uses the logistic (sigmoid) function to map predicted values to a probability between 0 and 1.

Formula:



$P(y=1) = 1 /$

$1+e^{-(b0+b1x)}$

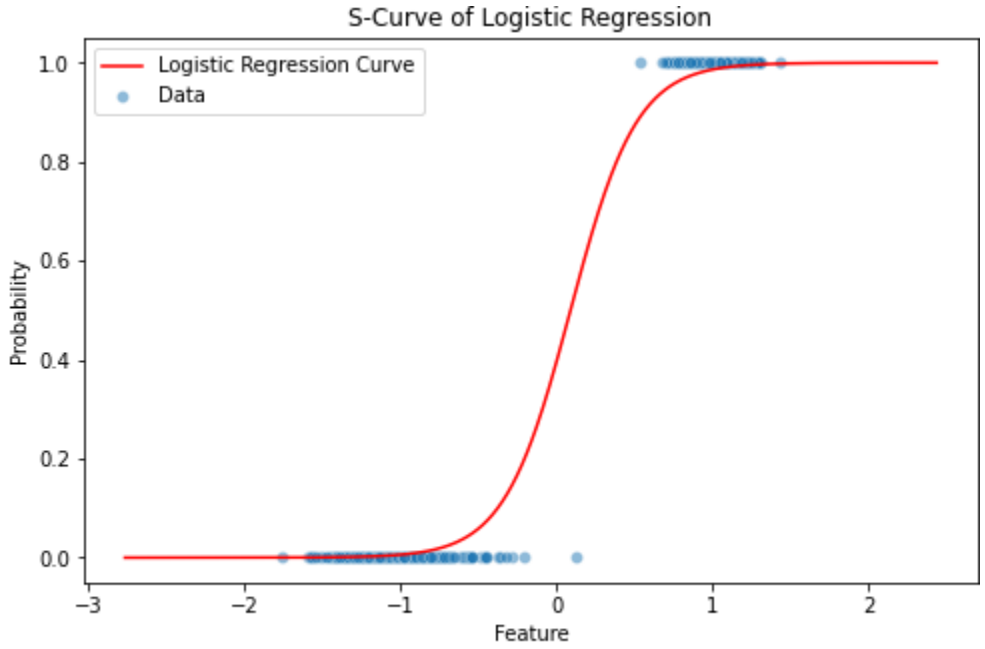
Example: Predicting whether an email is spam (1) or not spam (0) based on word features.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score

# Generate a simple dataset for S-curve visualization
X, y = make_classification(n_samples=500, n_features=1, n_informative=1, n_redundant=0,
                           n_clusters_per_class=1, random_state=42)

# Save dataset to CSV file
df = pd.DataFrame({'Feature': X.ravel(), 'Target': y})
df.to_csv('logistic_regression_data.csv', index=False)

# Load dataset from CSV file
data = pd.read_csv('logistic_regression_data.csv')
X = data[['Feature']].values
y = data['Target'].values

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict probabilities
X_range = np.linspace(X.min() - 1, X.max() + 1, 500).reshape(-1, 1)
y_prob = model.predict_proba(X_range)[:, 1]

# Plot S-curve
plt.figure(figsize=(8,
5))
sns.scatterplot(x=X.ravel(), y=y, label='Data', alpha=0.5)
plt.plot(X_range, y_prob, color='red', label='Logistic Regression Curve')
plt.xlabel('Feature')
plt.ylabel('Probability')
```

```
plt.title('S-Curve of Logistic Regression')
plt.legend()
plt.show()

# Evaluate model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

Output:




```
Accuracy: 1.00
```

jupyter logistic_regression_data.csv ✔ 3 minutes ago

File Edit View Language

```
Feature,Target
0.8925397543928608,1
1.0212892344790485,1
-0.9709182954229663,0
1.0107306362302557,1
-0.9054211224522328,0
1.1934900866905687,1
-1.1640948902040096,0
0.8687868653514965,1
1.1273443081270595,1
-1.1901594503467066,0
-0.484779575122009,0
-1.2131461681216362,0
-1.1398395658071145,0
-1.2579400776417822,0
1.0007422803524917,1
0.998871419927753,1
0.8913768808139252,1
-0.9248379149637164,0
-0.9716694236078032,0
-1.092006436909234,0
-1.0210150365779112,0
-0.8945323004296972,0
1.3099810783443147,1
0.9432045904029194,1
-1.2312848797003664,0
-0.5686995560218319,0
-0.7785066101800533,0
-0.8955853107893259,0
-1.356284275001949,0
0.9314534665192088,1
-1.2341321718157379,0
1.1120650866966404,1
-1.3592086510150996,0
1.0440110208035305,1
1.0696344099867323,1
1.0937289732237656,1
0.8233680435722951,1
```

Conclusion: Successfully demonstrated Logistic Regression.

Practical No. 10

Aim: Implement a python program to find Hyperplane using Linear SVM.

Objectives:

Understanding of HyperPlane.

Understanding of Linear SVM.

Implementation of Linear SVM using PYTHON

Theory:

Linear SVM (Support Vector Machine):

A **Linear SVM** is a supervised learning algorithm used for binary classification. It finds the best hyperplane that separates data points of different classes.

Hyperplane:

A hyperplane is a decision boundary that divides the feature space into two classes.

- In 2D: it's a line
- In 3D: it's a plane
- In higher dimensions: it's a hyperplane

Goal of SVM: To maximize the margin — the distance between the hyperplane and the nearest data points from each class (called support vectors).

Example: Classifying emails as "spam" or "not spam" using word frequencies. SVM will find the line (hyperplane) that best separates the two types of emails with the widest margin.

Example:

Classifying circular data — imagine two classes shaped like concentric circles. A linear hyperplane won't work in 2D, but after using RBF kernel, SVM can draw a circular boundary that separates the classes.

Code:

In [1]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

def plot_svc_decision_boundary(svm_clf, xmin, xmax):
    w = svm_clf.coef_[0]
    b = svm_clf.intercept_[0]

    # At the decision boundary, w0*x0 + w1*x1 + b = 0
    # => x1 = -w0/w1 * x0 - b/w1
    x0 = np.linspace(xmin, xmax, 200)
    decision_boundary = -w[0]/w[1] * x0 - b/w[1]

    margin = 1/w[1]
    gutter_up = decision_boundary + margin
    gutter_down = decision_boundary - margin

    svs = svm_clf.support_vectors_
    plt.scatter(svs[:, 0], svs[:, 1], s=180, facecolors='#FFAAAA')
    plt.plot(x0, decision_boundary, "k-", linewidth=2)
    plt.plot(x0, gutter_up, "k--", linewidth=2)
    plt.plot(x0, gutter_down, "k--", linewidth=2)
```

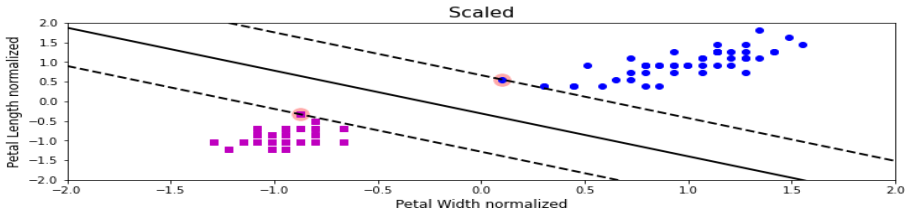
In [2]:
```
from sklearn.svm import SVC
from sklearn import datasets
import numpy as np

# Load Iris dataset
iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # Select petal length and petal width
y = iris["target"]

# Select only Setosa and Versicolor classes
setosa_or_versicolor = (y == 0) | (y == 1)
X = X[setosa_or_versicolor]
y = y[setosa_or_versicolor]

# SVM Classifier model with a large but finite C value
svm_clf = SVC(kernel="linear", C=1e10)  # Large C approximates a hard margin
svm_clf.fit(X, y)

# Make a prediction
prediction = svm_clf.predict([[2.4, 3.1]])
print("Predicted class:", prediction[0])
```

```
Predicted class: 1
```

In [3]:
```
#plot the decision boundaries
import numpy as np

plt.figure(figsize=(12,3.2))

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
svm_clf.fit(X_scaled, y)

plt.plot(X_scaled[:, 0][y==1], X_scaled[:, 1][y==1], "bo")
plt.plot(X_scaled[:, 0][y==0], X_scaled[:, 1][y==0], "ms")
plot_svc_decision_boundary(svm_clf, -2, 2)
plt.xlabel("Petal Width normalized", fontsize=12)
plt.ylabel("Petal Length normalized", fontsize=12)
plt.title("Scaled", fontsize=16)
plt.axis([-2, 2, -2, 2])
```

Output:

Out[3]: (-2.0, 2.0, -2.0, 2.0)



Conclusion: Successfully demonstrated Linear SVM.

Practical No. 11

Aim: Implement a python program to find Hyperplane using Non-Linear SVM.

Objectives:

Understanding of HyperPlane.

Understanding of Non-Linear SVM.

Implementation of Non-Linear SVM using PYTHON

Theory:

Non-Linear SVM

A Non-Linear SVM is used when data cannot be separated by a straight line (or hyperplane). It uses a kernel trick to transform the data into a higher dimension where a linear hyperplane can separate the classes.

Hyperplane in Non-Linear SVM

Although the separation isn't linear in original space, SVM still finds a linear hyperplane — but in the transformed feature space, not in the original one.

So, in the original space, this hyperplane may look curved or complex.

Kernel Trick:

Mathematical technique used to project data into a higher-dimensional space without explicitly computing coordinates.

Common kernels:

- Polynomial
- Radial Basis Function (RBF/Gaussian)

Code:

In [1]:
```
from sklearn.datasets import make_moons
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\__init__.py:138: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required
for this version of SciPy (detected version 1.23.5)
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion} is required for this version of "
```

In [2]:
```
import numpy as np
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
```

In [4]:
```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, noise=0.15, random_state=42)

#define a function to plot the dataset
def plot_dataset(X, y, axes):
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs")
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "ms")
    plt.axis(axes)
    plt.grid(True, which='both')
    plt.xlabel(r"$x_1$", fontsize=20)
    plt.ylabel(r"$x_2$", fontsize=20, rotation=0)

#Let's have a look at the data we have generated
plot_dataset(X, y, [-1.5, 2.5, -1, 1.5])
plt.show()
```

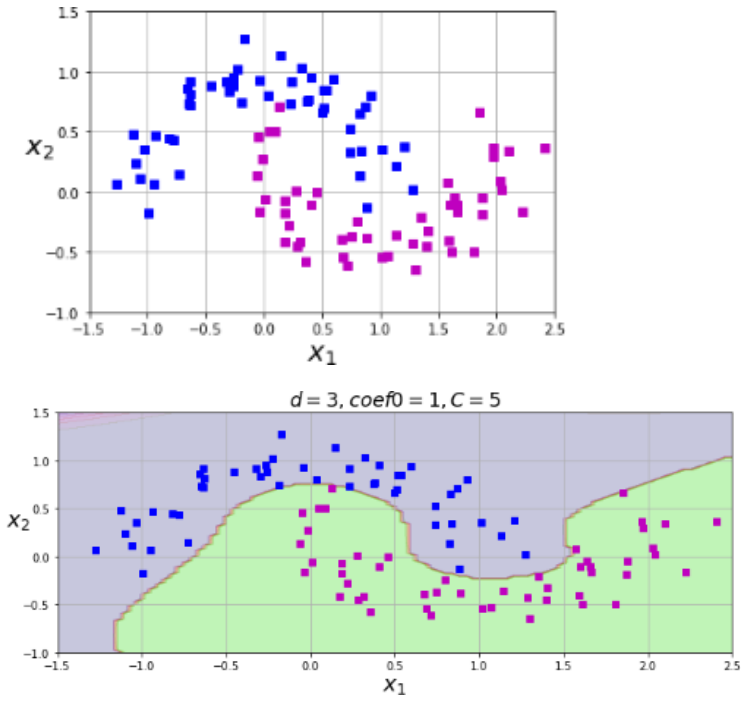
In [5]:
```
#define a function plot the decision boundaries
def plot_predictions(clf, axes):
    #create data in continous linear space
    x0s = np.linspace(axes[0], axes[1], 100)
    x1s = np.linspace(axes[2], axes[3], 100)
    x0, x1 = np.meshgrid(x0s, x1s)
    X = np.c_[x0.ravel(), x1.ravel()]
    y_pred = clf.predict(X).reshape(x0.shape)
    y_decision = clf.decision_function(X).reshape(x0.shape)
    plt.contourf(x0, x1, y_pred, cmap=plt.cm.brg, alpha=0.2)
    plt.contourf(x0, x1, y_decision, cmap=plt.cm.brg, alpha=0.1)
```

In [6]:
```
#C controls the width of the street
#Degree of data

#create a pipeline to create features, scale data and fit the model
polynomial_svm_clf = Pipeline((
    ("poly_features", PolynomialFeatures(degree=3)),
    ("scalar", StandardScaler()),
    ("svm_clf", SVC(kernel="poly", degree=10, coef0=1, C=5))
))

#call the pipeline
polynomial_svm_clf.fit(X,y)
```

Out[6]:
```
Pipeline(steps=[('poly_features', PolynomialFeatures(degree=3)),
                ('scalar', StandardScaler()),
                ('svm_clf', SVC(C=5, coef0=1, degree=10, kernel='poly'))])
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [7]:
```
#plot the decision boundaries
plt.figure(figsize=(11, 4))

#plot the decision boundaries
plot_predictions(polynomial_svm_clf, [-1.5, 2.5, -1, 1.5])

#plot the dataset
plot_dataset(X, y, [-1.5, 2.5, -1, 1.5])

plt.title(r"$d=3, coef0=1, C=5$", fontsize=18)
plt.show()
```

Output:



Conclusion: Successfully demonstrated Non-Linear SVM.

Practical No. 12

Aim: Implementation of k-means algorithm using Elbow method

Objectives:

- Understandingof k-means algorithm.
- Understanding of Elbow method
- Understanding of Elbow method for k-means
- Implementation of k-means using elbow method

Theory:

K-means:-

What is K-Means Clustering?

K-Means is an unsupervised machine learning algorithm used to group similar data points into k clusters. It finds groupings by minimizing the distance between points and their corresponding cluster center (centroid).

Example

Suppose we have the following 6 data points:
(1, 2), (1, 4), (1, 0), (10, 2), (10, 4), (10, 0)
And we want to group them into **2 clusters (k=2)**.

Steps of K-Means

Step 1: Choose the number of clusters k
Let's choose **k = 2**

Step 2: Initialize centroids randomly
Pick two random points as initial centroids. For example:
Centroid 1 = (1, 2)
Centroid 2 = (10, 2)

Step 3: Assign points to the nearest centroid
Calculate the Euclidean distance between each point and the centroids:

- (1, 2) → closer to (1, 2)
- (1, 4) → closer to (1, 2)
- (1, 0) → closer to (1, 2)
- (10, 2) → closer to (10, 2)
- (10, 4) → closer to (10, 2)
- (10, 0) → closer to (10, 2)

So, the clusters are:
Cluster 1: (1, 2), (1, 4), (1, 0)
Cluster 2: (10, 2), (10, 4), (10, 0)

Step 4: Recalculate the centroids

Take the average of the points in each cluster:

- Cluster 1 centroid:
  ((1+1+1)/3,(2+4+0)/3)=(1,2)((1+1+1)/3 , (2+4+0)/3) = (1, 2)
- Cluster 2 centroid:
  ((10+10+10)/3,(2+4+0)/3)=(10,2)((10+10+10)/3 , (2+4+0)/3) = (10, 2)

Centroids didn't change, so the algorithm converged!

Final Output

Two clusters:

- Cluster 1: All points with x = 1
- Cluster 2: All points with x = 10

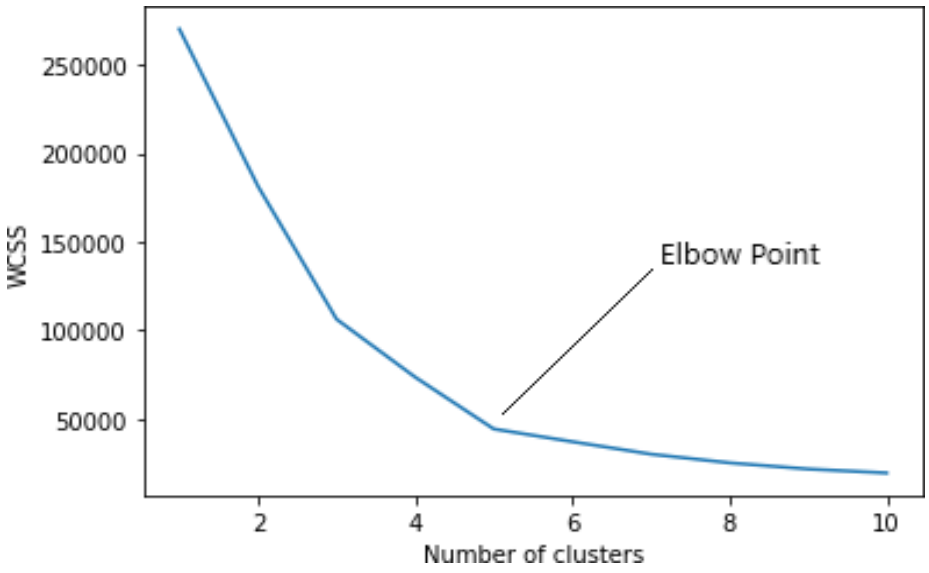
Elbow method:-

The Elbow Method is a graphical approach used to determine the optimal number of clusters (k) in K-Means Clustering.

Step-by-Step Explanation:

1. **Run K-Means** clustering for a range of k values (e.g., 1 to 10).
2. For each k, calculate the **WCSS (Within-Cluster Sum of Squares)**:
   - It measures the **compactness** of the clusters.
   - Lower WCSS = tighter, more defined clusters.
3. Plot **k vs. WCSS** on a graph.
4. Look for the point where the **rate of decrease sharply changes** — this is the **"elbow" point**.
5. The elbow indicates the **optimal number of clusters** — adding more clusters beyond this doesn't improve the model much.

Diagram for Elbow Method:



Code and Output:

In [4]:
```
!pip install --user threadpoolctl==3.1.0
```

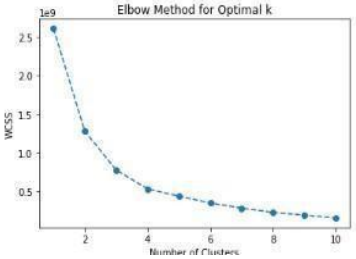
```
Requirement already satisfied: threadpoolctl==3.1.0 in c:\users\admin\appdata\roaming\python\python38\site-packages (3.1.0)
```

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Load the dataset
df = pd.read_csv("clustering.csv")

# Display first few rows of the dataset
print(df.head())

# Drop missing values
df_cleaned = df.dropna()

# Selecting numerical columns for clustering
numerical_cols = df_cleaned.select_dtypes(include=[np.number]).columns
print("Numerical columns used for clustering:", numerical_cols.tolist())

# Feature selection for clustering (Modify as needed)
X = df_cleaned[numerical_cols]

# Apply the Elbow Method
wcss = []  # Within-cluster sum of squares
for i in range(1, 11):  # Trying different cluster numbers from 1 to 10
    kmeans = KMeans(n_clusters=i, random_state=42, n_init=10)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot the Elbow Method

plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('Elbow Method for Optimal k')
plt.show()

# Choose optimal k (Modify based on the elbow plot observation)
k_optimal = 3  # Example choice, change based on your dataset

# Apply K-Means with the optimal number of clusters
kmeans = KMeans(n_clusters=k_optimal, random_state=42, n_init=10)
df_cleaned['Cluster'] = kmeans.fit_predict(X)

# Display clustered data
print(df_cleaned.head())
```

```
# Plot the clusters (for 2D visualization, choose two relevant features)
plt.scatter(df_cleaned[numerical_cols[0]], df_cleaned[numerical_cols[1]], c=df_cleaned['Cluster'], cmap='viridis')
plt.xlabel(numerical_cols[0])
plt.ylabel(numerical_cols[1])
plt.title(f'K-Means Clustering (k={k_optimal})')
plt.colorbar(label='Cluster')
plt.show()
```

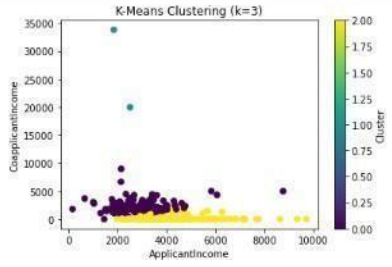
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001003   Male     Yes          1      Graduate            No
1  LP001005   Male     Yes          0      Graduate           Yes
2  LP001006   Male     Yes          0  Not Graduate            No
3  LP001008   Male      No          0      Graduate            No
4  LP001013   Male     Yes          0  Not Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             4583             1508.0       128.0             360.0
1             3000                0.0        66.0             360.0
2             2583             2358.0       120.0             360.0
3             6000                0.0       141.0             360.0
4             2333             1516.0        95.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Rural           N
1             1.0         Urban           Y
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
Numerical columns used for clustering: ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_Histo
ry']
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak o
n Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=2.
  warnings.warn(
```




```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001003   Male     Yes          1      Graduate            No
1  LP001005   Male     Yes          0      Graduate           Yes
2  LP001006   Male     Yes          0  Not Graduate            No
3  LP001008   Male      No          0      Graduate            No
4  LP001013   Male     Yes          0  Not Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             4583             1508.0       128.0             360.0
1             3000                0.0        66.0             360.0
2             2583             2358.0       120.0             360.0
3             6000                0.0       141.0             360.0
4             2333             1516.0        95.0             360.0

   Credit_History Property_Area Loan_Status  Cluster
0             1.0         Rural           N        2
1             1.0         Urban           Y        2
2             1.0         Urban           Y        0
3             1.0         Urban           Y        2
4             1.0         Urban           Y        0
```

```
<ipython-input-5-4d27906d6adb>:42: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  df_cleaned['Cluster'] = kmeans.fit_predict(X)
```



Conclusion: Successfully applied Elbow method and Implemented k-means algorithm.

Practical No. 13

Aim: Implementation of Bagging algorithm:-Random Forest

Objectives:

- Understanding of bagging
- Understanding of Random forest
- Implementation of Random Forest

Theory:

1. Bagging (Bootstrap Aggregating):

Definition:

Bagging is a machine learning ensemble technique that aims to improve the stability and accuracy of algorithms by reducing variance and overfitting. It works by training multiple models (typically the same type of model) on different random subsets of the training data, then combining their predictions to make a final decision. The main idea is to average the predictions (in regression) or take a vote (in classification) to achieve a more robust model.

How Bagging Works:

- **Bootstrap Sampling:** Bagging uses a technique called **bootstrap sampling** to create multiple subsets of the original dataset. Each subset is created by sampling data points **with replacement**, meaning that some points may appear multiple times in a subset, and some may not appear at all.

- **Model Training:** Each model in the ensemble is trained on one of these bootstrap samples.

- **Aggregating Results:**

    - For regression problems, the final prediction is typically the **average** of all the model predictions.

    - For classification problems, the final prediction is made by taking the **mode (majority vote)** of all the model predictions.

Advantages of Bagging:

- **Reduces Overfitting:** By averaging multiple models, bagging reduces the likelihood of overfitting, especially with high-variance models like decision trees.

- **Improves Accuracy:** The averaging or voting process generally results in a more accurate prediction than a single model.

Example of Bagging: Imagine you have a dataset to predict house prices (a regression problem). You could use bagging as follows:

1. **Step 1:** Create multiple bootstrap samples of the original dataset. For example, you might create five different subsets of the original data by sampling with replacement.

2. **Step 2:** Train a regression model (like a decision tree) on each of these bootstrap samples.

3. **Step 3:** Make predictions using each of the trained models.

4. **Step 4:** Average the predictions from each model to get the final predicted house price.

2. Random Forest:

Definition:
Random Forest is an extension of bagging that uses decision trees as the base learners and introduces an additional level of randomness in the model training process. It is one of the most powerful and widely used machine learning algorithms, especially for classification and regression problems.

How Random Forest Works: Random Forest is built on top of the bagging concept but adds an extra randomization step when splitting the nodes in the decision trees. Instead of considering all the features when making a split, Random Forest only considers a random subset of features at each node. This helps make the trees more diverse and reduces correlation between them, leading to a more robust ensemble model.

- **Bootstrap Sampling:** Similar to bagging, Random Forest uses bootstrap sampling to create different subsets of the training data.

- **Random Feature Selection:** When constructing each decision tree, instead of evaluating all features at each split, Random Forest randomly selects a subset of features to consider for splitting. This ensures that the trees are diverse and helps reduce overfitting.

- **Tree Construction:** Each tree in the forest is trained on a different bootstrap sample, and at each node, only a subset of features is used for the split. This leads to more diverse trees in the forest.

- **Aggregating Results:**

    - For regression problems, the final prediction is the **average** of the predictions from all the trees.

    - For classification problems, the final prediction is the **majority vote** from all the trees.

Advantages of Random Forest:

- **High Accuracy:** Random Forest generally provides very high accuracy due to the diversity of the trees and the random feature selection.
- **Robust to Overfitting:** By averaging multiple trees, Random Forest reduces the risk of overfitting, even with deep trees.
- **Handles High-Dimensional Data:** Random Forest is good at handling datasets with many features, as it performs random feature selection.

Example of Random Forest:

Let's say you're working with a dataset for classifying whether a customer will buy a product (binary classification).

1. **Step 1:** Create multiple bootstrap samples from the original dataset.
2. **Step 2:** For each bootstrap sample, train a decision tree. However, when constructing each tree, at each decision point (node), only a random subset of features is considered for splitting the data.
3. **Step 3:** Once all trees are built, use them to make predictions on new data. For each new observation, each tree in the forest provides a classification (e.g., "Will Buy" or "Won't Buy").
4. **Step 4:** The final classification is determined by a **majority vote** among all the trees.

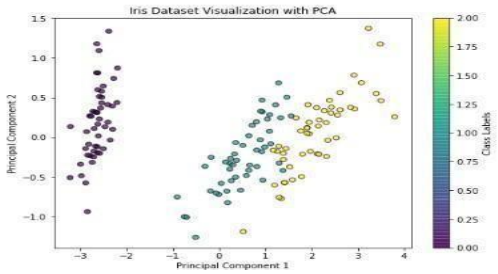
For example, if you have 100 trees in the forest and 60 trees predict "Will Buy" while 40 trees predict "Won't Buy," the final prediction will be "Will Buy."

Code and Output:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred_rf = rf_classifier.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Accuracy of Random Forest Classifier: {accuracy_rf * 100:.2f}%')

# AdaBoost Classifier
adaboost = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1), n_estimators=50, random_state=42)
adaboost.fit(X_train, y_train)
y_pred_adaboost = adaboost.predict(X_test)
accuracy_adaboost = accuracy_score(y_test, y_pred_adaboost)
print(f'Accuracy of AdaBoost Classifier: {accuracy_adaboost * 100:.2f}%')

# Gradient Boosting Classifier
gb_classifier = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
gb_classifier.fit(X_train, y_train)
y_pred_gb = gb_classifier.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print(f'Accuracy of Gradient Boosting Classifier: {accuracy_gb * 100:.2f}%')

# Voting Classifier (Ensemble of Logistic Regression, Decision Tree, and Random Forest)
voting_classifier = VotingClassifier(estimators=[
    ('lr', LogisticRegression()),
    ('dt', DecisionTreeClassifier()),
    ('rf', RandomForestClassifier(n_estimators=100))
], voting='hard')
voting_classifier.fit(X_train, y_train)
y_pred_voting = voting_classifier.predict(X_test)
accuracy_voting = accuracy_score(y_test, y_pred_voting)
print(f'Accuracy of Voting Classifier: {accuracy_voting * 100:.2f}%')
```

```
# Reduce dimensions for visualization
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Scatter plot of the dataset
plt.figure(figsize=(8, 6))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis', edgecolor='k', alpha=0.7)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Iris Dataset Visualization with PCA')
plt.colorbar(label='Class Labels')
plt.show()
```

```
Accuracy of Random Forest Classifier: 100.00%
Accuracy of AdaBoost Classifier: 100.00%
Accuracy of Gradient Boosting Classifier: 100.00%
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Accuracy of Voting Classifier: 100.00%
```



Conclusion: Successfully Implemented a Bagging algorithm:Random Forest

Practical No. 14

Aim: Implementation of boosting algorithm AdaBoost, stochastic gradient boosting and voting ensemble

Objectives:

- Understanding the concept of boosting
- Understanding of Adaboost algorithm
- Understanding of Ensemble learning Method
- Implementation of boosting algorithm

Theory:

What is Boosting?

Boosting is a machine learning technique used to improve the accuracy of a model by combining several weak models to create a stronger model. The key idea behind boosting is to focus on the mistakes made by previous models and correct them in subsequent iterations. This process is done by adjusting the weights of the misclassified data points, so the model learns to focus more on them.

In boosting, each model is built sequentially, and each new model tries to correct the errors made by the previous ones. Boosting is particularly effective because it helps improve the performance of weak models (e.g., decision trees with limited depth).

What is Ensemble Learning?

Ensemble learning is a method that combines multiple individual models (learners) to produce a stronger and more accurate prediction than any single model could achieve alone. The idea is that combining several models can reduce variance (overfitting) and bias (underfitting) while improving the overall predictive performance.

There are two main types of ensemble learning:

1. **Bagging** (Bootstrap Aggregating): Multiple models are trained independently on different subsets of the data, and their predictions are averaged or voted on (e.g., Random Forest).

2. **Boosting**: Models are trained sequentially, with each new model trying to correct the mistakes of the previous one (e.g., AdaBoost, Gradient Boosting).

AdaBoost with Example:

AdaBoost (Adaptive Boosting) is one of the most popular boosting algorithms. It combines weak learners (usually decision trees) to create a strong classifier. In AdaBoost, each new model focuses on the errors of the previous models.

Steps in AdaBoost:

1. Start with a weak model (e.g., a decision stump, which is a decision tree of height 1).
2. Assign equal weights to all training samples.
3. Train the weak model on the weighted dataset.
4. Increase the weights of the misclassified samples, so the next model focuses more on them.
5. Add the new model to the ensemble.
6. Repeat the process for a fixed number of iterations or until the model reaches the desired accuracy.

Example:

Imagine you have a dataset of email messages that need to be classified as spam or not spam.

1. **First iteration**: You train the first decision stump (weak model). It correctly classifies most of the emails but makes some mistakes.
2. **Weight update**: Increase the weights for the misclassified emails, so the next model will focus more on them.
3. **Second iteration**: A second decision stump is trained, and it focuses on the mistakes made by the first stump.
4. **Repeat**: This process continues until a pre-specified number of weak models are created.

Finally, AdaBoost combines the predictions of all the weak learners to make the final prediction, often by taking a weighted vote of the models' predictions.

Voting Ensemble Method:

The **Voting Ensemble** method combines multiple models to improve prediction accuracy by taking a vote (majority rule) for classification problems or averaging predictions for regression problems.

There are two main types of voting:

1. **Hard Voting**: The final prediction is based on the majority vote of the individual models. Each model casts a vote for a class, and the class with the most votes is chosen as the final prediction.

2. **Soft Voting**: Instead of hard class labels, models predict probabilities for each class. The probabilities from all models are averaged, and the class with the highest average probability is chosen.

Example of Voting Ensemble (Classification):

Suppose you have three classifiers: a Decision Tree, a K-Nearest Neighbors (KNN) model, and a Logistic Regression model.

- Model 1 (Decision Tree) predicts: **Class A**
- Model 2 (KNN) predicts: **Class B**
- Model 3 (Logistic Regression) predicts: **Class A**

Hard Voting: Since Class A has the majority of votes (2 out of 3), the final prediction will be **Class A**.

Soft Voting: If the models provide probabilities like:

- Decision Tree: Class A: 0.7, Class B: 0.3
- KNN: Class A: 0.4, Class B: 0.6
- Logistic Regression: Class A: 0.5, Class B: 0.5

The average probabilities would be:

- Class A: (0.7 + 0.4 + 0.5) / 3 = 0.53
- Class B: (0.3 + 0.6 + 0.5) / 3 = 0.47

In this case, the final prediction would be **Class A**, because it has the highest average probability.

To sum up:

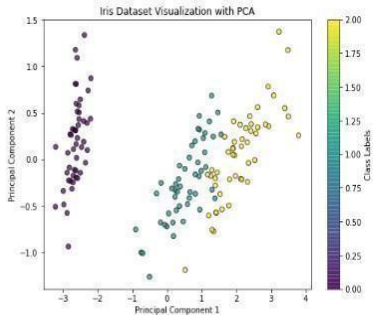
- **Boosting** focuses on correcting errors from previous models.
- **Ensemble learning** combines multiple models to improve performance.
- **AdaBoost** is a boosting method that adapts based on the errors of previous models.
- **Voting Ensemble** combines predictions from different models using majority voting (hard voting) or probability averaging (soft voting).

Code & Output:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred_rf = rf_classifier.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Accuracy of Random Forest Classifier: {accuracy_rf * 100:.2f}%')

# AdaBoost Classifier
adaboost = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1), n_estimators=50, random_state=42)
adaboost.fit(X_train, y_train)
y_pred_adaboost = adaboost.predict(X_test)
accuracy_adaboost = accuracy_score(y_test, y_pred_adaboost)
print(f'Accuracy of AdaBoost Classifier: {accuracy_adaboost * 100:.2f}%')

# Gradient Boosting Classifier
gb_classifier = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
gb_classifier.fit(X_train, y_train)
y_pred_gb = gb_classifier.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print(f'Accuracy of Gradient Boosting Classifier: {accuracy_gb * 100:.2f}%')

# Voting Classifier (Ensemble of Logistic Regression, Decision Tree, and Random Forest)
voting_classifier = VotingClassifier(estimators=[
    ('lr', LogisticRegression()),
    ('dt', DecisionTreeClassifier()),
    ('rf', RandomForestClassifier(n_estimators=100))
], voting='hard')
voting_classifier.fit(X_train, y_train)
y_pred_voting = voting_classifier.predict(X_test)
accuracy_voting = accuracy_score(y_test, y_pred_voting)
print(f'Accuracy of Voting Classifier: {accuracy_voting * 100:.2f}%')
```

```
# Reduce dimensions for visualization
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Scatter plot of the dataset
plt.figure(figsize=(8, 6))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis', edgecolor='k', alpha=0.7)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Iris Dataset Visualization with PCA')
plt.colorbar(label='Class Labels')
plt.show()
```

```
Accuracy of Random Forest Classifier: 100.00%
Accuracy of AdaBoost Classifier: 100.00%
Accuracy of Gradient Boosting Classifier: 100.00%
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Accuracy of Voting Classifier: 100.00%
```



Conclusion: Successfully understood and implemented various Boosting algorithm