Artificial Intelligence and Machine Learning

Experiment No: - 1

Introduction to SWI- PROLOG

Programming with the help of simple programs

Name: Patel Arun Ramjanak **Roll No: 26**

1. **Aim:** Introduction to SWI- PROLOG Programming with the help of simple programs.

2. **Objectives:**
   • Understand logical programming syntax and semantics
   • Design programs in PROLOG language

3. **Software Required:** SWI-Prolog

4. **Theory:**
   SWI-Prolog is a versatile implementation of the Prolog language. Although SWI-Prolog gained its popularity primarily in education, its development is mostly driven by the needs for application development. This is facilitated by a rich interface to other IT components by supporting many document types and (network) protocols as well as a comprehensive low-level interface to C that is the basis for high-level interfaces to C++, Java (bundled), C#, Python, etc (externally available). Data type extensions such as dicts and strings as well as full support for Unicode and unbounded integers simplify smooth exchange of data with other components.
   SWI-Prolog aims at scalability. Its robust support for multi-threading exploits multi-core hardware efficiently and simplifies embedding in concurrent applications. It's Just in Time Indexing (JITI) provides transparent and efficient support for predicates with millions of clauses.
   SWI-Prolog unifies many extensions of the core language that have been developed in the Prolog community such as tabling, constraints, global variables, destructive assignment, delimited continuations and interactors.
   SWI-Prolog offers a variety of development tools, most of which may be combined at will. The native system provides an editor written in Prolog that is a close clone of Emacs. It provides semantic highlighting based on real time analysis of the code by the Prolog system itself. Complementary tools include a graphical debugger, profiler and cross-reference. Alternatively, there is a mode for GNU-Emacs and, Eclipse plugin called PDT and a VSC plugin, each of which may be combined with the native graphical tools. Finally, a computational notebook and web-based IDE is provided by SWISH. SWISH is a versatile tool that can be configured and extended to suit many different scenarios.
   SWI-Prolog provides an add-on distribution and installation mechanism called packs. A pack is a directory with minimal organizational conventions and a control file that describes the origin, version, dependencies and automatic upgrade support.

5. Procedure/ Program:

1-A). Sample program to demonstrate Rules and Facts.

weather - Notepad

File Edit Format View Help

```
weather(Pheonix,summer,hot).
weather(la,summer,warm).
weather(Pheonix,winter,warm).
```

1-B). Sample program to demonstrate the relationship in prolog.

weatherrelationship - Notepad

File Edit Format View Help

```
weather(Pheonix,hot,summer).
weather(la,warm,summer).
warmer_than(C1,C2):-
weather(C1,hot,summer),
weather(C2,warm,summer).
```

1-C). Demonstrate of relationship with user defined prolog program.

pract1c - Notepad

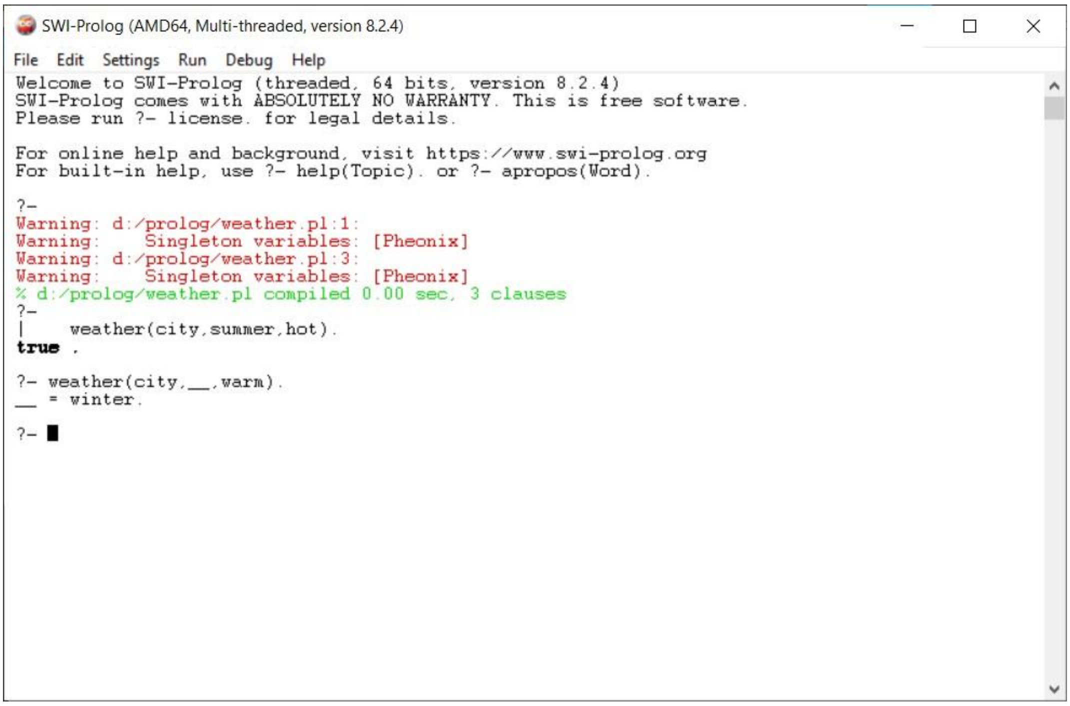
File Edit Format View Help

```
%section A
result(rahim,3.6).
result(ajay,3.7).
result(priya,2.8).
result(rahul,3.9).
result(kim,3.10).

%section B
result(sam,4.0).
result(vickey,3.9).
result(priyanka,3.8).
result(ram,3.6).
result(kunal,3.0).

getresult :-

write("enter section A student name"),nl,
read(X),nl,
result(X,Y),nl,
write("section A studen result is"),nl,
write(Y),nl,

write("enter section B student name"),nl,
read(P),nl,
result(P,Q),nl,
write("section B student result is"),nl,
write(Q),nl,

compare(Y,Q).

compare(Y,Q):-

Y>Q, nl,
write(" section A student is best");
Y<Q, nl,
write(" section B student is best");
Y=:=Q,nl,
write("all students are same").
```

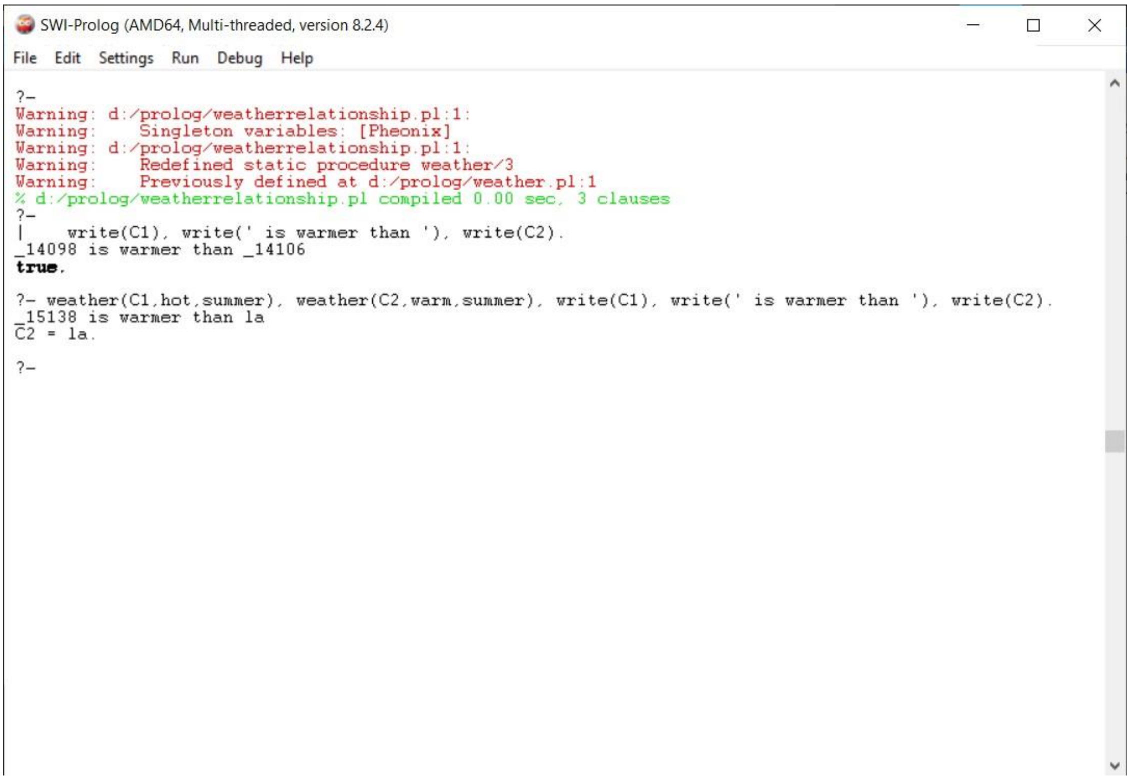
Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

6. Results:

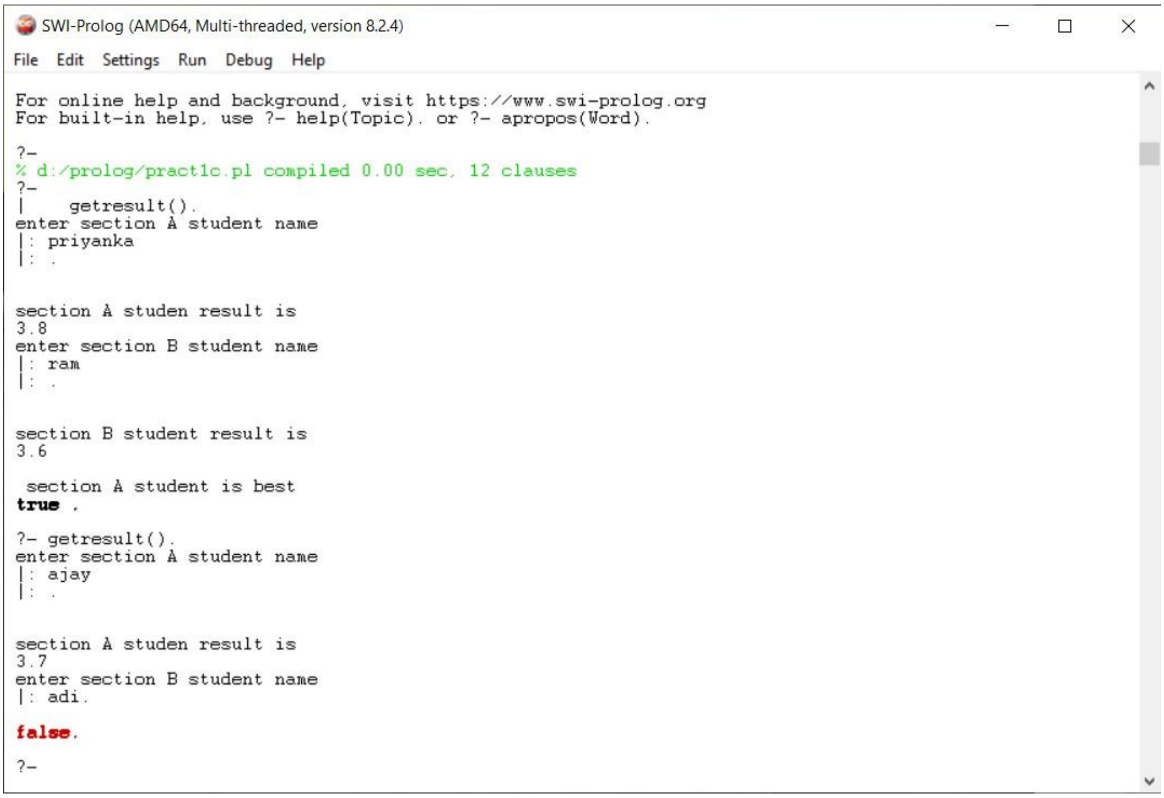
1-A). Sample program to demonstrate Rules and Facts.



1-B). Sample program to demonstrate the relationship in prolog.



1-C). Demonstrate of relationship with user defined prolong program.



7. <u>**Conclusion:**</u>

   Demonstration and implementation of rules and facts, relationship is done using SWI-Prolog software. Semantics and syntax of prolog language is well understood. Logical programming concepts required to execute artificial intelligence problem is well understood.

Artificial Intelligence and Machine Learning

Experiment No: - 2

Water Jug Problem Using DFS

1. **Aim:** Solve Water-Jug Problem Using DFS.

2. **Objectives:**
   • Understand DFS (State space search) & Water-Jug Problem.
   • Solve Water-Jug Problem using DFS in PROLOG language.

3. **Software Required:** SWI-Prolog

4. **Theory:**

   Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph.

   A standard DFS implementation puts each vertex of the graph into one of two categories:
   - Visited
   - Not Visited

   The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

   The DFS algorithm works as follows:
   - Start by putting any one of the graph's vertices on top of a stack.
   - Take the top item of the stack and add it to the visited list.
   - Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
   - Keep repeating steps 2 and 3 until the stack is empty.

   • **Algorithm:**
   1. Create a variable called NODE-LIST and set it to initial state
   2. Until a goal state is found or NODE-LIST is empty do
      a. Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit.
   b. For each way that each rule can match the state described in E do:
      i. Apply the rule to generate a new state
      ii. If the new state is a goal state, quit and return this state
      iii. Otherwise, add the new state in front of NODE-LIST

- **Water-Jug Problem:**

This is the jug problem using simple depth-first search of a graph.
The modified water-jug problem is as follows: Jug A holds 4 litres, and jug B holds 3 litres. There is a pump, which can be used to fill either Jug. How can you get exactly 2 litres of water into the 4-liter jug?

Assumptions:

We can fill a jug from the pump
We can pour water out of the jug onto the ground
We can pour water from one jug to another
There are no other measuring devices available
To solve the water jug problem, apart from problem statement we also need a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to state is made as described in corresponding right side and the resulting state is checked to see if it corresponds to a goal state. As long as it does not the cycle continue.

5. **Procedure/ Program:**

Rule	Current state	New state	Rules
1	(x, y) if x<4	(4,y)	Fill the 4-gallon jug.
2	(x, y) if y<3	(x,3)	Fill the 3-gallon jug.
3	(x, y) if x>0	(x-d, y)	Pour some water out of 4-gallon jug.
4	(x, y) if y>0	(x, y-d)	Pour some water out of 3-gallon jug.
5	(x, y) if x>0	(0, y)	Empty the 4-gallon jug on ground.
6	(x, y) if y>0	(x, 0)	Empty the 3-gallon jug on ground.
7	(x, y) if (x+y)>=4 &(y>0)	(4,y-(4-x))	Pour water from 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full.
8	if (x+y)>=3 &(x>0)	(x-(3-y),3)	Pour water from 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full.
9	(x, y) if (x+y)<=4 &(y>0)	(x+y,0)	Pour all the water from 3-gallon jug into the 4-gallon jug.
10	(x, y) if (x+y)<=3 &(x>0)	(0,x+y)	Pour all the water from 4-gallon jug into the 3-gallon jug.
11	(0,2)	(2,0)	Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug.
12	(2,y)	(0,y)	Empty 2 gallons in the 4-gallon jug on the ground.

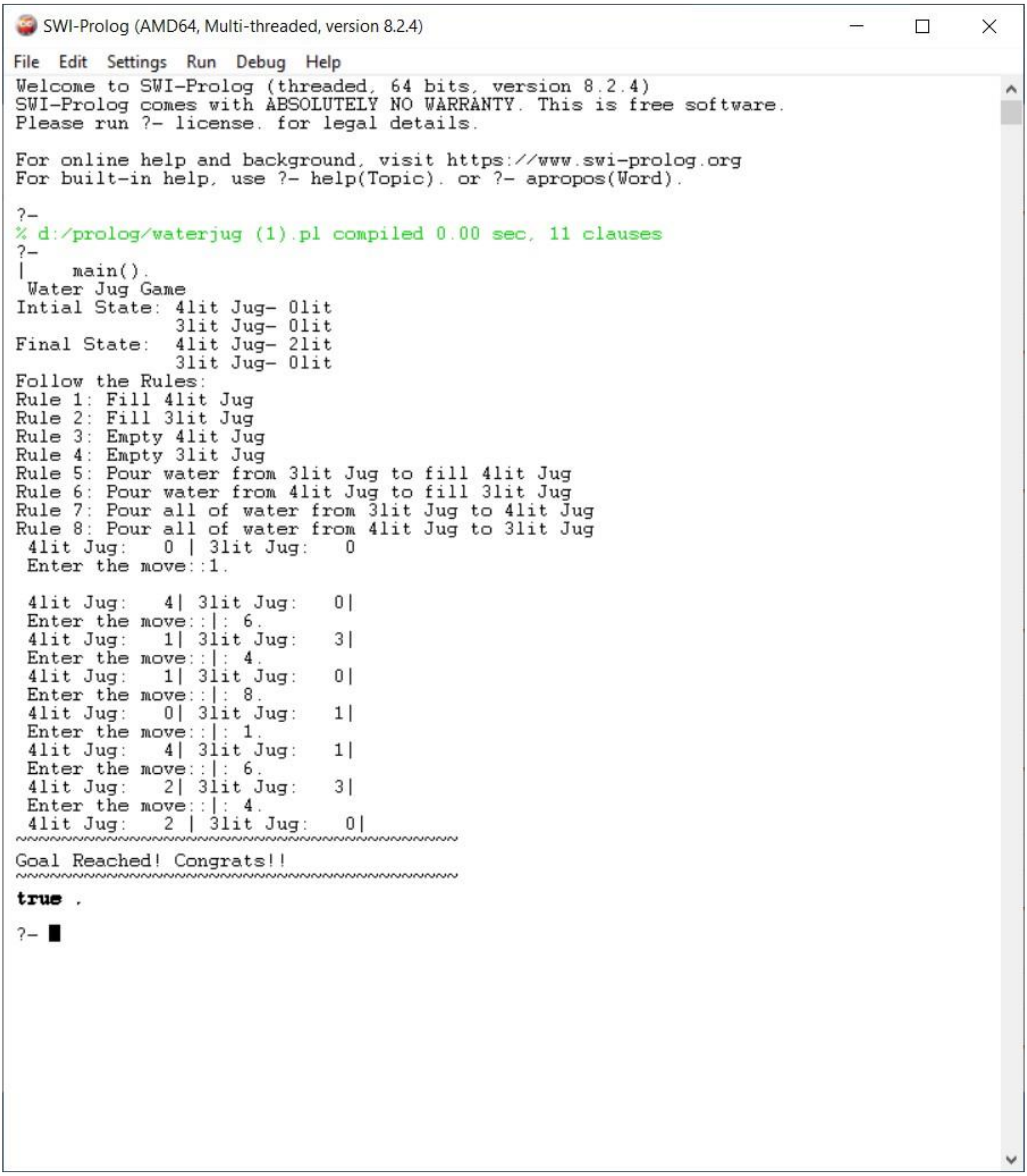
waterjug (1) - Notepad

File Edit Format View Help

```
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).
```

Ln 1, Col 1 | 100% | Unix (LF) | UTF-8

6. Output:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.2.4)
File  Edit  Settings  Run  Debug  Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?-
% d:/prolog/waterjug (1).pl compiled 0.00 sec, 11 clauses
?-
|     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 .

?-
```

7. Conclusion:

In state space problem, the problem consists of four components: initial state, a set of actions, a goal test functions and a path cost function is analysed. The environment of the problem is represented by a state space and path from initial state to goal state is been analysed.

Artificial Intelligence and Machine Learning

Experiment No: 3

Tic Tac Toe Using BFS

1. Aim: Design Tic Tac Toe Using BFS.

2. Objectives:

• Understand and implement BFS algorithm.

• Understand gaming using BFS in Prolog Language.

3. Software Required: SWI-Prolog

4. Theory:

• Breadth-first search

Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.

➢ Breadth-first search can be implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out (FIFO) queue, assuring that the nodes that are visited first will be expanded first.

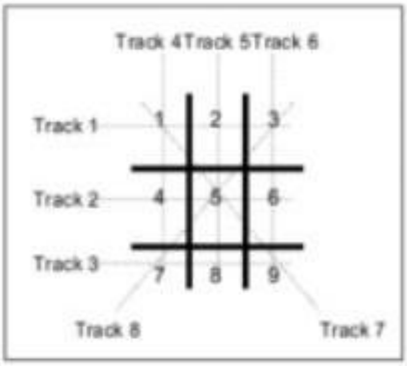
➢ It uses two queues for its implementation: open, close Queue

➢ Children are added from backend of queue.

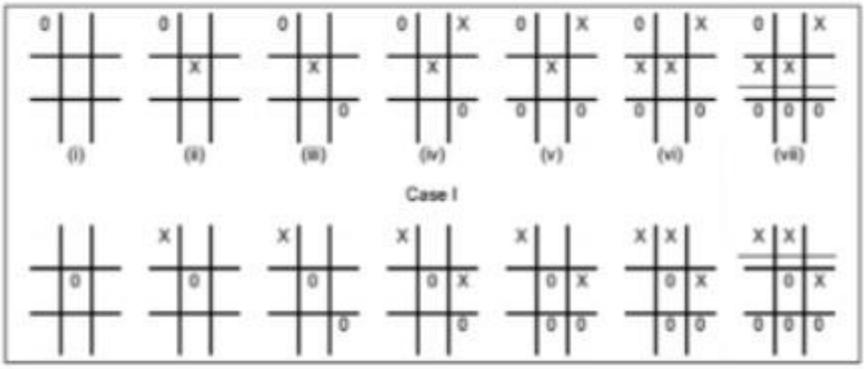
Algorithm

1. Create single member queue comprising of root node.
2. If 1st Member of Queue is GOAL, then go to Step 5.
3. If first member of queue is not GOAL then remove it and add to CLOSE or Visited Queue. Consider its Children/ successor, if any add them from BACK/REAR [FIFO]
4. If queue is not empty then go to Step 2, If queue is empty then go to Step 6
5. Print “SUCCESS” and stop.
6. Print “FALIURE” and stop.

A Formal Definition of the Tic Tac Toe Game:



The board used to play the Tic-Tac-Toe game consists of 9 cells laid out in the form of a 3x3 matrix . The game is played by 2 players and either of them can start. Each of the two players is assigned a unique symbol (generally 0 and X). Each player alternately gets a turn to make a move. Making a move is compulsory and cannot be deferred. In each move a player places the symbol assigned to him/her in a hitherto blank cell. Let a track be defined as any row, column or diagonal on the board. Since the board is a square matrix with 9 cells, all rows, columns and diagonals have exactly 3 cells. It can be easily observed that there are 3 rows, 3 columns and 2 diagonals, and hence a total of 8 tracks on the board (Fig. 1). The goal of the game is to fill all the three cells of any track on the board with the symbol assigned to one before the opponent does the same with the symbol assigned to him/her. At any point of the game, if there exists a track whose all three cells have been marked by the same symbol, then the player to whom that symbol has been assigned wins and the game terminates. If there exist no track whose cells have been marked by the same symbol when there is no more blank cell on the board then the game is drawn. Let the priority of a cell be defined as the number of tracks passing through it. The priorities of the nine cells on the board according to this definition are tabulated in Table 1. Alternatively, let the priority of a track be defined as the sum of the priorities of its three cells. The priorities of the eight tracks on the board according to this definition are tabulated in Table 2. The prioritization of the cells and the tracks lay the foundation of the heuristics to be used in this study. These heuristics are somewhat similar to those proposed by Rich and Knight.



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),
```

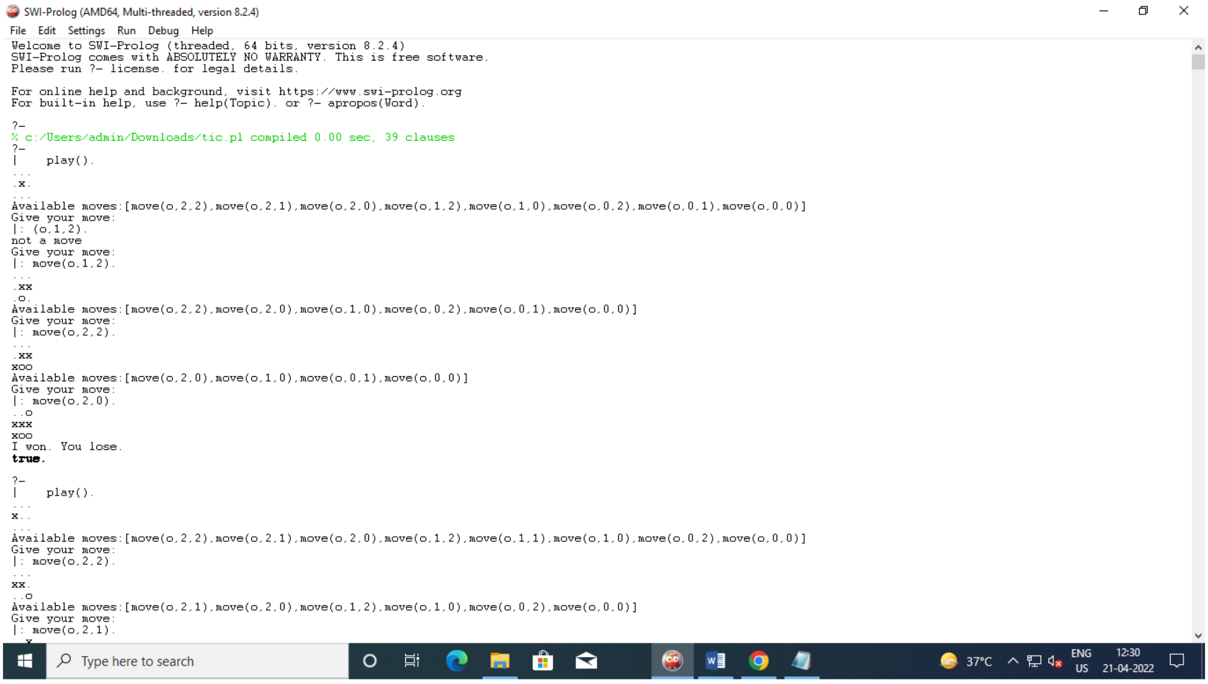
```
    (ValidMoves = [] -> (write('It is a tie'), nl);
     (write('Available moves:'), write(ValidMoves), nl,
     ask_move(Y, ValidMoves),
      NextGame = [Y | Game],
      (victory_condition(o, NextGame) ->
      (write('I lose. You win.'), nl);
        my_turn(NextGame), !))).

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).
```

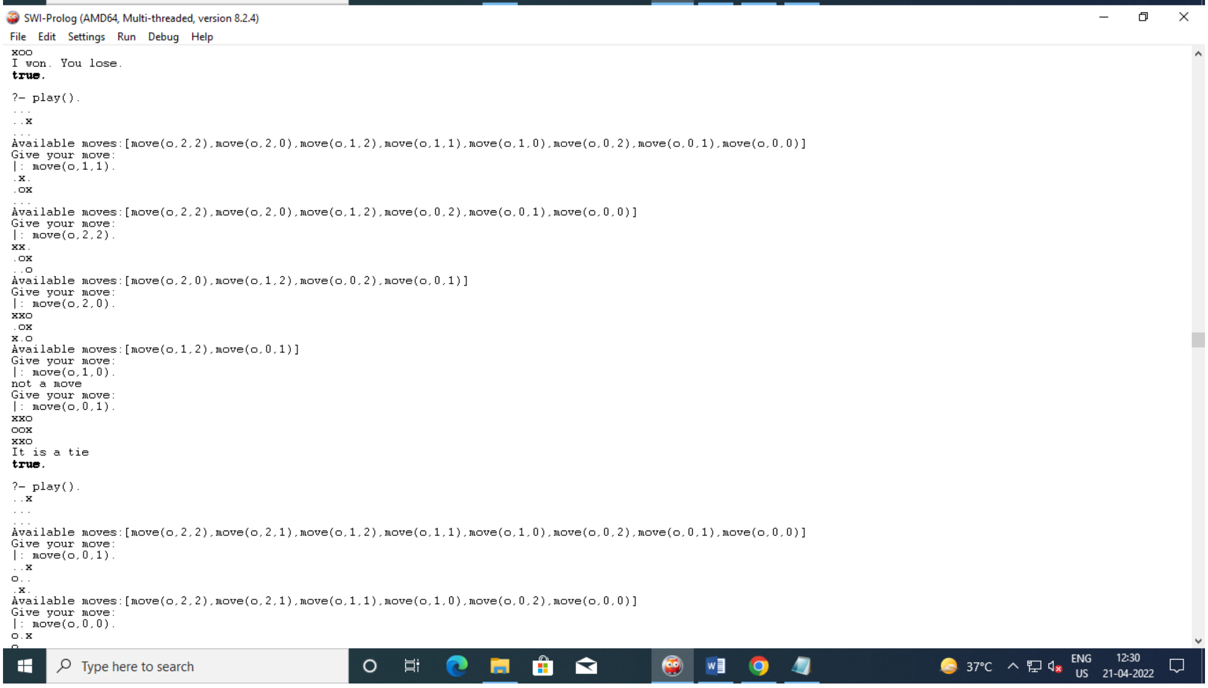
```
game_analysis_continue(Turn, Game, NextMove) :-
   valid_moves(Moves, Game, Turn),
   game_analysis_search(Moves, Turn, Game, NextMove).

% 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: -





Conclusion:

BFS is a uniformed search technique. It selects the shallowest unexpanded node inthe search tree for expansion. It is complete, optimal for unit step costs and has timeand space
complexity of O(bd).

Artificial Intelligence and Machine Learning

Experiment No: 4

Hill-climbing to solve 8- Puzzle Problem

Name: Patel Arun Ramjanak Roll No: 26

1. **Aim:** Design Hill-climbing algorithm to solve 8- Puzzle Problem.

2. **Objectives:**
   - Understand and Implement Hill climbing algorithm
   - Understand 8-puzzle Problem and solve it using Hill climbing algorithm.

3. **Software Required:** SWI-Prolog

4. **Theory:**

- **Hill Climbing algorithm: -**

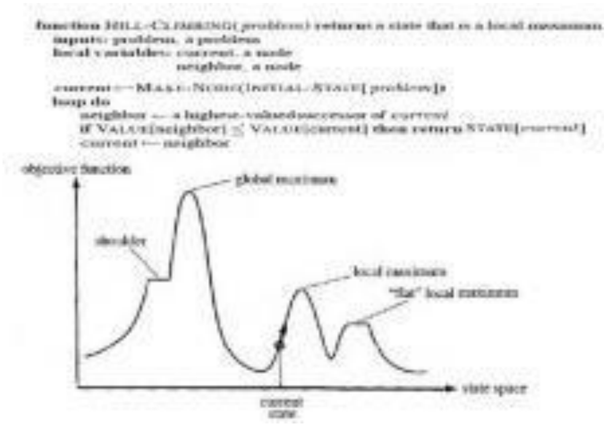
Hill Climbing is a local search algorithm. The search algorithms that we have seen so farare designed to explore search spaces systematically. This is achieved by keeping one or more paths in memory and by recording which alternatives have been explored at each point along the path and which have not. In many problems, however, the path to the goalis irrelevant. For example, in the 8-queens problem, what matters is the final configurationof queens, not the order in which they are added. It is used for continuous state space problem or when numbers of states are very large. Search algorithms operate using a single current state (rather than multiple paths) and generally move only to neighbors of that state.

They have two key advantages:

I. They use very little memory-usually a constant amount

II. They can often find reasonable solutions in large or infinite (continuous) state spaces forwhich systematic algorithms are unsuitable.

To understand local search, we will find it very useful to consider the state space landscapeshown in Figure. A landscape has both "location" (defined by the state) and "elevation" (defined by the value of the heuristic cost function or objective function). If elevation corresponds to cost, then the aim is to find the lowest valley-a global minimum. If elevation corresponds to an objective function, then the aim is to find the highest peak-aglobal maximum.



The hill-climbing search algorithm is shown in Figure. It is simply a loop that continually moves in the direction of increasing value-that is, uphill. It terminates when it reaches a "peak" where no neighbour has a higher value.
The algorithm does not maintain a search tree, so the current node data structure need only record the state and its objective function value. Unfortunately, hill climbing often gets stuck for the following reasons:

1. **Local maxima:**
   A local maximum is a peak that is higher than each of its neighbouring states, but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upwards towards the peak, but will then be stuck with nowhere else to go.

2. **Ridges:**
   Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.

3. **Plateau:**

   A plateau is an area of the state space landscape where the evaluation function isflat. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which it is possible to make progress.

4. 8-PUZZLE Problem: -

The eight puzzle consists of 3-by-3 square frame which holds eight movable square tiles which are numbered from 1 to 8. One square is empty, permitting tiles to be shifted. Theobjective of the puzzle is to find the sequence of tile movements that leads from a startingconfiguration to a goal configuration such as that shown in the figure a.

3	8	1
6	2	5
	4	7

A start configuration

1	2	3
8		4
7	6	5

A goal configuration

Fig-a

The states of the eight puzzles are the different permutations of the tiles within the frame. The operationsare the permissible moves (one may consider the empty space as being movable rather than the tiles): up, down, left and right. An optimal or good solution is one that maps an initial arrangement of tiles to the goal configuration with the smallest number of moves.

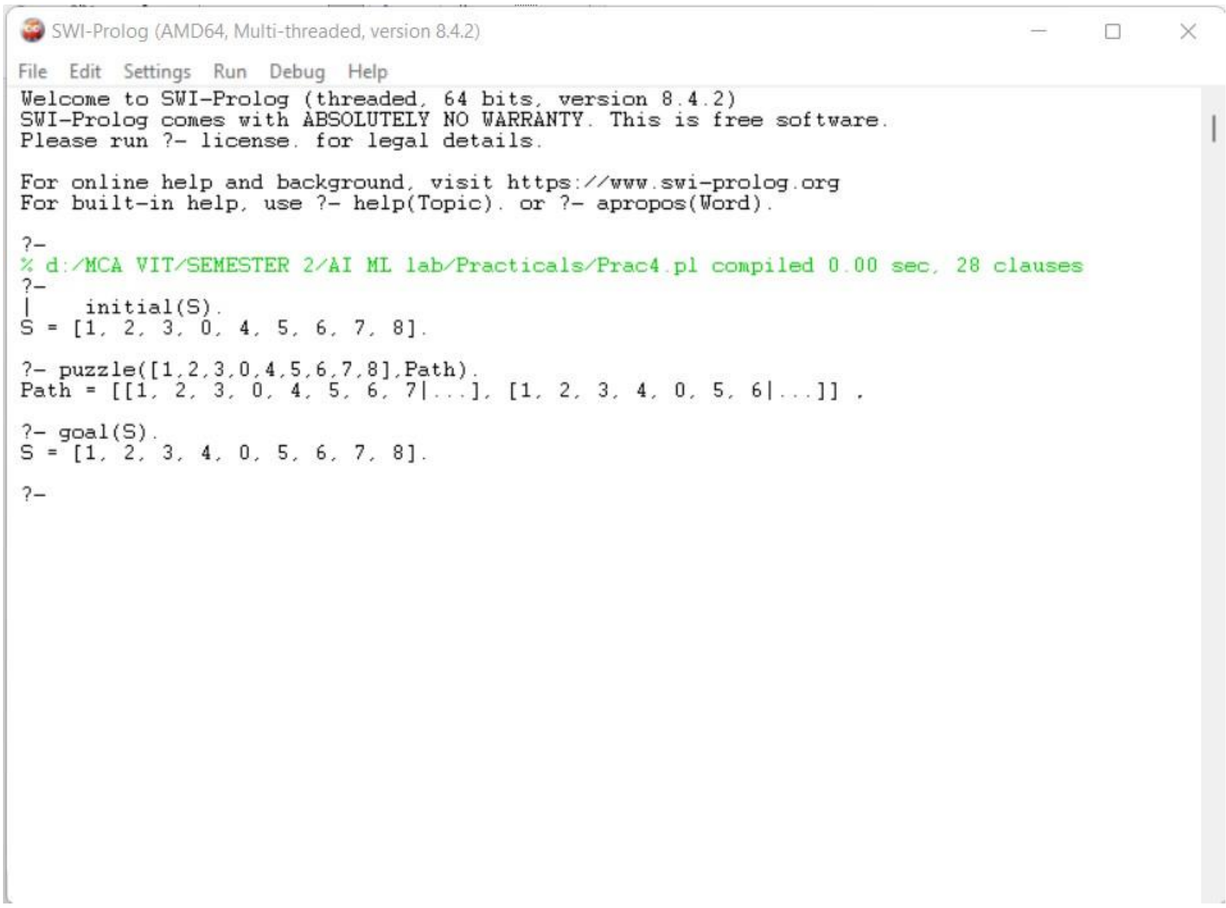
5. Code:

```
initial([1,2,3,
        0,4,5,
        6,7,8]).

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

move([X1,0,X3, X4,X5,X6, X7,X8,X9],
    [0,X1,X3, X4,X5,X6, X7,X8,X9]).
move([X1,X2,0, X4,X5,X6, X7,X8,X9],
    [X1,0,X2, X4,X5,X6, X7,X8,X9]).

%% move left in the middle row
move([X1,X2,X3, X4,0,X6,X7,X8,X9],
    [X1,X2,X3, 0,X4,X6,X7,X8,X9]).
move([X1,X2,X3, X4,X5,0,X7,X8,X9],
    [X1,X2,X3, X4,0,X5,X7,X8,X9]).

%% move left in the bottom row
move([X1,X2,X3, X4,X5,X6, X7,0,X9],
    [X1,X2,X3, X4,X5,X6, 0,X7,X9]).
move([X1,X2,X3, X4,X5,X6, X7,X8,0],
    [X1,X2,X3, X4,X5,X6, X7,0,X8]).

%% move right in the top row
move([0,X2,X3, X4,X5,X6, X7,X8,X9],
    [X2,0,X3, X4,X5,X6, X7,X8,X9]).
move([X1,0,X3, X4,X5,X6, X7,X8,X9],
    [X1,X3,0, X4,X5,X6, X7,X8,X9]).

%% move right in the middle row
move([X1,X2,X3, 0,X5,X6, X7,X8,X9],
    [X1,X2,X3, X5,0,X6, X7,X8,X9]).
move([X1,X2,X3, X4,0,X6, X7,X8,X9],
    [X1,X2,X3, X4,X6,0, X7,X8,X9]).
```

```
Prac4.pl - Notepad
File    Edit    View

%% move right in the bottom row
move([X1,X2,X3, X4,X5,X6,0,X8,X9],
    [X1,X2,X3, X4,X5,X6,X8,0,X9]).
move([X1,X2,X3, X4,X5,X6,X7,0,X9],
    [X1,X2,X3, X4,X5,X6,X7,X9,0]).

%% move up from the middle row
move([X1,X2,X3, 0,X5,X6, X7,X8,X9],
    [0,X2,X3, X1,X5,X6, X7,X8,X9]).
move([X1,X2,X3, X4,0,X6, X7,X8,X9],
    [X1,0,X3, X4,X2,X6, X7,X8,X9]).
move([X1,X2,X3, X4,X5,0, X7,X8,X9],
    [X1,X2,0, X4,X5,X3, X7,X8,X9]).

%% move up from the bottom row
move([X1,X2,X3, X4,X5,X6, X7,0,X9],
    [X1,X2,X3, X4,0,X6, X7,X5,X9]).
move([X1,X2,X3, X4,X5,X6, X7,X8,0],
    [X1,X2,X3, X4,X5,0, X7,X8,X6]).
move([X1,X2,X3, X4,X5,X6, 0,X8,X9],
    [X1,X2,X3, 0,X5,X6, X4,X8,X9]).

%% move up from the top row
move([0,X2,X3, X4,X5,X6, X7,X8,X9],
    [X4,X2,X3, 0,X5,X6, X7,X8,X9]).
move([X1,0,X3, X4,X5,X6, X7,X8,X9],
    [X1,X5,X3, X4,0,X6, X7,X8,X9]).
move([X1,X2,0, X4,X5,X6, X7,X8,X9],
    [X1,X2,X6, X4,X5,0, X7,X8,X9]).

%% move down from the middle row
move([X1,X2,X3, 0,X5,X6, X7,X8,X9],

Ln 1, Col 1    150%    Windows (CRLF)    UTF-8
```

```
move([X1,X2,X3, X4,X5,X6, X7,X8,0],
    [X1,X2,X3, X4,X5,0, X7,X8,X6]).
move([X1,X2,X3, X4,X5,X6, 0,X8,X9],
    [X1,X2,X3, 0,X5,X6, X4,X8,X9]).

%% move up from the top row
move([0,X2,X3, X4,X5,X6, X7,X8,X9],
    [X4,X2,X3, 0,X5,X6, X7,X8,X9]).
move([X1,0,X3, X4,X5,X6, X7,X8,X9],
    [X1,X5,X3, X4,0,X6, X7,X8,X9]).
move([X1,X2,0, X4,X5,X6, X7,X8,X9],
    [X1,X2,X6, X4,X5,0, X7,X8,X9]).

%% move down from the middle row
move([X1,X2,X3, 0,X5,X6, X7,X8,X9],
    [X1,X2,X3, X7,X5,X6, 0,X8,X9]).
move([X1,X2,X3, X4,0,X6, X7,X8,X9],
    [X1,X2,X3, X4,X8,X6, X7,0,X9]).
move([X1,X2,X3, X4,X5,0, X7,X8,X9],
    [X1,X2,X3, X4,X5,X9, X7,X8,0]).

puzzle(S, [S]) :- goal(S).
puzzle(S, [S|Rest]) :- move(S, S2), puzzle(S2, Rest).
```

6. Output:



7. Conclusion:

- Informed search covers algorithms that perform purely local search in the state space, evaluating and modifying one or more current states. These algorithms are suitable for the problem in which the path cost is irrelevant and all that matters is the solution state itself. One of the informed search methods that is hill climbing search algorithm is executed.
- 8-puzzle is a simple game consisting of a 3*3 grid containing 9 squares. One of the squares is empty. From the given states a program is executed to reach the goal state. It is analysed and implemented.

Artificial Intelligence & Machin Learning

Experiment No. 5

Introduction to Python Programming: Learn the different libraries – NumPy, Pandas, SciPy, Matplotlib .

Name: Patel Arun Ramjanak Roll No. 26

PYTHON

PRACTICALS

PRACTICAL NO. 5

Aim: Introduction to Python Programming: Learn the different libraries – NumPy, Pandas, SciPy,Matplotlib .

Objective: To learn different libraries in python.

Software Requirement:

- **Anaconda Navigator:** Anaconda Navigator is a desktop graphical user interface included in Anaconda that allows you to launch applications and easily manage conda packages, environments and channels without the need to use command line commands.

Theory:

- NumPy: NumPy can be used **to perform a wide variety of mathematical operations on arrays.**
- Pandas: **Pandas** is a **Python** library. **Pandas** is used to analyze data.
- SciPy: SciPy is **a scientific computation library that uses NumPy underneath**. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats andsignal processing.
- Matplotlib: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plotsinto applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

Code & Output:

1. **NumPy:**

Import numpy

```
In [74]: import numpy as np
```

```
In [82]: # create an array
         digits = np.array([
               [1, 2, 3],
               [4, 5, 6],
             [6, 7, 9],
             ])
```

```
In [3]:  digits
Out[3]: array([[1, 2, 3],
               [4, 5, 6],
               [6, 7, 9]])
```

```
In [83]: # addition of two integers
         a=2
         b=4
```

```
In [5]: c=a+b
        c
Out[5]: 6
```

```
In [85]: # Study shape and axes of an array.
         temperatures = np.array([
               29.3, 42.1, 18.8, 16.1, 38.0, 12.5,
                12.6, 49.9, 38.6, 31.3, 9.2, 22.2
             ]).reshape(2, 2, 3)

In [7]: In [3]: temperatures.shape

Out[7]: (2, 2, 3)

In [9]: In [4]: temperatures

Out[9]: array([[[29.3, 42.1, 18.8],
                [16.1, 38. , 12.5]],

               [[12.6, 49.9, 38.6],
                [31.3,  9.2, 22.2]]])

In [10]: In [5]: np.swapaxes(temperatures, 1, 2)

Out[10]: array([[[29.3, 16.1],
                 [42.1, 38. ],
                 [18.8, 12.5]],

                [[12.6, 31.3],
                 [49.9,  9.2],
                 [38.6, 22.2]]])

In [11]: table = np.array([
            ...:     [5, 3, 7, 1],
            ...:     [2, 6, 7 ,9],
            ...:     [1, 1, 1, 1],
            ...:     [4, 3, 2, 0],
            ...: ])

In [12]: table.max()

Out[12]: 9

In [13]:  table.max(axis=0)

Out[13]: array([5, 6, 7, 9])

In [14]:  table.max(axis=1)

Out[14]: array([7, 9, 1, 4])
```

```
In [89]: #Study of Broadcasting with an array.
         A= np.arange(32).reshape(4, 1, 8)

In [16]: A

Out[16]: array([[[ 0,  1,  2,  3,  4,  5,  6,  7]],

                [[ 8,  9, 10, 11, 12, 13, 14, 15]],

                [[16, 17, 18, 19, 20, 21, 22, 23]],

                [[24, 25, 26, 27, 28, 29, 30, 31]]])

In [17]: B = np.arange(48).reshape(1, 6, 8)

In [18]: B

Out[18]: array([[[ 0,  1,  2,  3,  4,  5,  6,  7],
                 [ 8,  9, 10, 11, 12, 13, 14, 15],
                 [16, 17, 18, 19, 20, 21, 22, 23],
                 [24, 25, 26, 27, 28, 29, 30, 31],
                 [32, 33, 34, 35, 36, 37, 38, 39],
                 [40, 41, 42, 43, 44, 45, 46, 47]]])
```


```
In [89]: #Study of Broadcasting with an array.
         A= np.arange(32).reshape(4, 1, 8)

In [16]: A

Out[16]: array([[[ 0,  1,  2,  3,  4,  5,  6,  7]],

                [[ 8,  9, 10, 11, 12, 13, 14, 15]],

                [[16, 17, 18, 19, 20, 21, 22, 23]],

                [[24, 25, 26, 27, 28, 29, 30, 31]]])

In [17]: B = np.arange(48).reshape(1, 6, 8)

In [18]: B

Out[18]: array([[[ 0,  1,  2,  3,  4,  5,  6,  7],
                 [ 8,  9, 10, 11, 12, 13, 14, 15],
                 [16, 17, 18, 19, 20, 21, 22, 23],
                 [24, 25, 26, 27, 28, 29, 30, 31],
                 [32, 33, 34, 35, 36, 37, 38, 39],
                 [40, 41, 42, 43, 44, 45, 46, 47]]])
```

```
In [90]: # Addition of two Arrays.
         A+B

Out[90]: array([[[ 0,  2,  4,  6,  8, 10, 12, 14],
                 [ 8, 10, 12, 14, 16, 18, 20, 22],
                 [16, 18, 20, 22, 24, 26, 28, 30],
                 [24, 26, 28, 30, 32, 34, 36, 38],
                 [32, 34, 36, 38, 40, 42, 44, 46],
                 [40, 42, 44, 46, 48, 50, 52, 54]],

                [[ 8, 10, 12, 14, 16, 18, 20, 22],
                 [16, 18, 20, 22, 24, 26, 28, 30],
                 [24, 26, 28, 30, 32, 34, 36, 38],
                 [32, 34, 36, 38, 40, 42, 44, 46],
                 [40, 42, 44, 46, 48, 50, 52, 54],
                 [48, 50, 52, 54, 56, 58, 60, 62]],

                [[16, 18, 20, 22, 24, 26, 28, 30],
                 [24, 26, 28, 30, 32, 34, 36, 38],
                 [32, 34, 36, 38, 40, 42, 44, 46],
                 [40, 42, 44, 46, 48, 50, 52, 54],
                 [48, 50, 52, 54, 56, 58, 60, 62],
                 [56, 58, 60, 62, 64, 66, 68, 70]],

                [[24, 26, 28, 30, 32, 34, 36, 38],
                 [32, 34, 36, 38, 40, 42, 44, 46],
                 [40, 42, 44, 46, 48, 50, 52, 54],
                 [48, 50, 52, 54, 56, 58, 60, 62],
                 [56, 58, 60, 62, 64, 66, 68, 70],
                 [64, 66, 68, 70, 72, 74, 76, 78]]])
```

```
In [91]: #Find the Square of an array.
         square = np.array([
                [16, 3, 2, 13],
                [5, 10, 11, 8],
                [9, 6, 7, 12],
                [4, 15, 14, 1]
            ])

In [21]: for i in range(4):
            ...:     assert square[:, i].sum() == 34
            ...:     assert square[i, :].sum() == 34
            ...:

In [22]: assert square[:2, :2].sum() == 34

In [23]: assert square[2:, :2].sum() == 34

In [24]:  assert square[:2, 2:].sum() == 34

In [25]: assert square[2:, 2:].sum() == 34

In [92]: #Study of masking and filtering.
         numbers = np.linspace(5, 50, 24, dtype=int).reshape(4, -1)

In [27]: numbers

Out[27]: array([[ 5,  6,  8, 10, 12, 14],
                [16, 18, 20, 22, 24, 26],
                [28, 30, 32, 34, 36, 38],
                [40, 42, 44, 46, 48, 50]])
```

```
In [28]: mask = numbers % 4 == 0

In [29]: mask

Out[29]: array([[False, False,  True, False,  True, False],
                [ True, False,  True, False,  True, False],
                [ True, False,  True, False,  True, False],
                [ True, False,  True, False,  True, False]])

In [30]: numbers[mask]

Out[30]: array([ 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48])

In [31]: by_four = numbers[numbers % 4 == 0]

In [32]: by_four

Out[32]: array([ 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48])

In [33]: from numpy.random import default_rng

In [34]: rng = default_rng()

In [35]: values = rng.standard_normal(10000)

In [36]:  values[:5]

Out[36]: array([-1.78348667e-01, -1.41216557e+00, -2.25700591e+00,  1.80981733e-03,
                 3.64960139e-01])

In [37]: std = values.std()

In [38]: std

Out[38]: 1.0010075480221816

In [39]: filtered = values[(values > -2 * std) & (values < 2 * std)]

In [40]: filtered.size

Out[40]: 9532

In [41]:  values.size

Out[41]: 10000

In [42]: filtered.size / values.size

Out[42]: 0.9532

In [94]: a = np.array([
                [1, 2],
           [3, 4],
           [5, 6],
          ])

In [95]: #Transposing, Sorting, and Concatenating of arrays.
         a.T

Out[95]: array([[1, 3, 5],
                [2, 4, 6]])
```

```
In [45]: a.transpose()

Out[45]: array([[1, 3, 5],
                [2, 4, 6]])

In [46]: data = np.array([
            ...:     [7, 1, 4],
            ...:     [8, 6, 5],
            ...:     [1, 2, 3]
            ...: ])

In [47]: np.sort(data)

Out[47]: array([[1, 4, 7],
                [5, 6, 8],
                [1, 2, 3]])

In [48]: np.sort(data, axis=None)

Out[48]: array([1, 1, 2, 3, 4, 5, 6, 7, 8])

In [49]: np.sort(data, axis=0)

Out[49]: array([[1, 1, 3],
                [7, 2, 4],
                [8, 6, 5]])

In [50]: a = np.array([
            ...:     [4, 8],
            ...:     [6, 1]
            ...: ])

In [51]: b = np.array([
            ...:     [3, 5],
            ...:     [7, 2],
            ...: ])

In [52]: np.hstack((a, b))

Out[52]: array([[4, 8, 3, 5],
                [6, 1, 7, 2]])

In [53]: np.vstack((b, a))

Out[53]: array([[3, 5],
                [7, 2],
                [4, 8],
                [6, 1]])

In [54]: np.concatenate((a, b))

Out[54]: array([[4, 8],
                [6, 1],
                [3, 5],
                [7, 2]])

In [55]: np.concatenate((a, b), axis=None)

Out[55]: array([4, 8, 6, 1, 3, 5, 7, 2])
```

```
In [96]: #Implementation of Maclaurin Series.
         from math import e, factorial
         fac = np.vectorize(factorial)

         def e_x(x, terms=10):
             """Approximates e^x using a given number of terms of
             the Maclaurin series
             """
             n = np.arange(terms)
             return np.sum((x ** n) / fac(n))

         if __name__ == "__main__":
             print("Actual:", e ** 3)  # Using e from the standard library

             print("N (terms)\tMaclaurin\tError")

             for n in range(1, 14):
                 maclaurin = e_x(3, terms=n)
                 print(f"{n}\t\t{maclaurin:.03f}\t\t{e**3 - maclaurin:.03f}")

         Actual: 20.085536923187664
         N (terms)       Maclaurin       Error
         1               1.000           19.086
         2               4.000           16.086
         3               8.500           11.586
         4               13.000          7.086
         5               16.375          3.711
         6               18.400          1.686
         7               19.412          0.673
         8               19.846          0.239
         9               20.009          0.076
         10              20.063          0.022
         11              20.080          0.006
         12              20.084          0.001
         13              20.085          0.000
```

```
In [97]: #Study of different Datatypes(numerical,String)
         a = np.array([1, 3, 5.5, 7.7, 9.2], dtype=np.single)
         a

Out[97]: array([1. , 3. , 5.5, 7.7, 9.2], dtype=float32)

In [60]: b = np.array([1, 3, 5.5, 7.7, 9.2], dtype=np.uint8)
         b

Out[60]: array([1, 3, 5, 7, 9], dtype=uint8)

In [61]: names = np.array(["bob", "amy", "han"], dtype=str)

In [62]: names

Out[62]: array(['bob', 'amy', 'han'], dtype='<U3')

In [63]: names.itemsize

Out[63]: 12

In [64]: names = np.array(["bob", "amy", "han"])

In [65]: names

Out[65]: array(['bob', 'amy', 'han'], dtype='<U3')

In [66]: more_names = np.array(["bobo", "jehosephat"])

In [67]: np.concatenate((names, more_names))

Out[67]: array(['bob', 'amy', 'han', 'bobo', 'jehosephat'], dtype='<U10')
```

```
In [68]: names[2] = "jasica"

In [69]: names

Out[69]: array(['bob', 'amy', 'jas'], dtype='<U3')

In [98]: #Study of structured array
         data = np.array([
              ("joe", 32, 6),
              ("mary", 15, 20),
             ("felipe", 80, 100),
             ("beyonce", 38, 9001),
            ], dtype=[("name", str, 10), ("age", int), ("power", int)])

In [71]: data[0]

Out[71]: ('joe', 32, 6)

In [72]: data["name"]

Out[72]: array(['joe', 'mary', 'felipe', 'beyonce'], dtype='<U10')

In [73]: data[data["power"] > 9000]["name"]

Out[73]: array(['beyonce'], dtype='<U10')

In [108]: #numpy version
          print("Numpy Version:",np.version.version)

          Numpy Version: 1.19.5
```

2. Pandas:

In [4]:
```
# import Pandas Print version
import pandas as pd
print(pd.__version__)
```

```
1.2.4
```

In [5]:
```
#create a dataframe
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}
```

In [6]:
```
purchases = pd.DataFrame(data)

purchases
```

Out[6]:

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

In [7]:
```
purchases = pd.DataFrame(data, index=['June', 'Robert', 'Lily', 'David'])

purchases
```

Out[7]:

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

In [8]:
```
purchases.loc['June']
```

Out[8]:
```
apples     3
oranges    0
Name: June, dtype: int64
```

In [10]:
```
#read csv file
df = pd.read_csv('f.csv')

df
```

Out[10]:

	1	ram	7
0	2	sonali	8
1	3	teena	9
2	4	rahul	0

In [11]:
```
#read csv with index
df = pd.read_csv('f.csv', index_col=0)

df
```

Out[11]:

	ram	7
1		
2	sonali	8
3	teena	9
4	rahul	0

In [1]:
```
#Create Dataframe:
import pandas as pd
df = pd.DataFrame({'X':[78,85,96,80,86], 'Y':[84,94,89,83,86],'Z':[86,97,96,72,83]});
print(df)
```

```
    X   Y   Z
0  78  84  86
1  85  94  97
2  96  89  96
3  80  83  72
4  86  86  83
```

In [2]:
```
#create series
s = pd.Series([2, 4, 6, 8, 10])
print(s)
```

```
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

In [5]:
```
#Load the data and make sure to change the path for your Localdirectory
data = pd.read_csv('project_data.csv')
```

In [6]:
```
#first 5 rows
data.head()
```

Out[6]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
201701	1982	Graduation	Single	58138.0	9/4/2012	58	8	4	0	3	11
201702	1950	Graduation	Married	46344.0	3/8/2014	38	1	2	0	3	11
201703	1965	Graduation	Divorced	71613.0	8/21/2013	26	8	10	0	3	11
201704	1984	Graduation	Relationship	26646.0	2/10/2014	26	2	4	0	3	11
201705	1981	PhD	Widowed	58293.0	1/19/2014	94	5	6	0	3	11

In [7]:
```
#last 5 rows
data.tail()
```

Out[7]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
202195	1944	PhD	Divorced	55614.0	11/27/2013	85	9	6	0	3	11
202196	1962	Master	Divorced	59432.0	4/13/2013	88	5	11	0	3	11
202197	1978	Graduation	Divorced	55563.0	4/5/2014	22	2	3	0	3	11
202198	1971	PhD	Relationship	43624.0	4/21/2013	83	4	4	0	6	11
202199	1949	PhD	Relationship	41461.0	5/22/2014	63	6	11	0	6	11

In [8]:
```
#check the basic information of the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 499 entries, 0 to 498
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   customer_id        499 non-null    int64
 1   year_of_birth      499 non-null    int64
 2   educational_level  499 non-null    object
 3   marital_status     499 non-null    object
 4   annual_income      486 non-null    float64
 5   purhcase_date      499 non-null    object
 6   recency            499 non-null    int64
 7   online_purchases   499 non-null    int64
 8   store_purchases    499 non-null    int64
 9   complaints         499 non-null    int64
 10  calls              499 non-null    int64
 11  intercoms          499 non-null    int64
dtypes: float64(1), int64(8), object(3)
memory usage: 46.9+ KB
```

In [9]:
```
#extract the shape of the data
data.shape
```

Out[9]: (499, 12)

In [10]:
```
data['marital_status'].unique()
```

Out[10]:
```
array(['Single', 'Married', 'Divorced', 'Relationship', 'Widowed',
       'Widow'], dtype=object)
```

In [12]:
```
#count educational level
round(data['educational_level'].value_counts(normalize=True),2)
```

Out[12]:
```
Graduation     0.52
PhD            0.23
Master         0.16
High School    0.08
Basic          0.01
Name: educational_level, dtype: float64
```

In [13]:
```
#missing values or duplicate values
data.isnull()
data.duplicated().sum()
data['educational_level'].isnull().sum()
#specifying Education as a variable where we should look for the sum of missing values
```

Out[13]: 0

In [14]:
```
#Select and filter data: loc and iloc
subset_data = data[['year_of_birth ', 'educational_level', 'annual_income']]
subset_data
```

Out[14]:

	year_of_birth	educational_level	annual_income
0	1982	Graduation	58138.0
1	1950	Graduation	46344.0
2	1965	Graduation	71613.0
3	1984	Graduation	26646.0
4	1981	PhD	58293.0
...	...	...	...
494	1944	PhD	55614.0
495	1962	Master	59432.0
496	1978	Graduation	55563.0
497	1971	PhD	43624.0
498	1949	PhD	41461.0

499 rows × 3 columns

In [15]:
```
#select a unique category of the education by specifying that only "Master" should be returned from the data frame.
data[data["educational_level"] == "Master"]
```

Out[15]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
:01706	1967	Master	Relationship	62000.0	9/9/2013	16	6	10	5	3	11
:01714	1952	Master	Single	59354.0	11/15/2013	53	6	5	0	3	11
:01719	1980	Master	Single	76995.0	3/28/2013	91	11	9	4	3	11
:01731	1989	Master	Divorced	10979.0	5/22/2014	34	3	3	8	3	11
:01732	1963	Master	Single	38620.0	5/11/2013	56	2	3	0	3	11
...	...	...	...	...	...	...	...	...	...	...	...
:02182	1972	Master	Divorced	37760.0	8/11/2013	54	2	3	0	3	11
:02185	1960	Master	Relationship	29027.0	10/10/2012	93	5	4	0	0	11
:02187	1976	Master	Relationship	56290.0	11/14/2013	4	3	7	0	11	11
:02194	1964	Master	Single	58308.0	1/12/2013	77	2	3	0	3	11
:02196	1962	Master	Divorced	59432.0	4/13/2013	88	5	11	0	3	11

2 columns

In [16]:
```
#specify the rows and columns as labels
data.loc[:6, ['educational_level', 'recency']]
```

Out[16]:

	educational_level	recency
0	Graduation	58
1	Graduation	38
2	Graduation	26
3	Graduation	26
4	PhD	94
5	Master	16
6	Graduation	34

In [17]:
```
#speciy rows and columns as integer based values
data.iloc[:6, [2,6]]
```

Out[17]:

	educational_level	recency
0	Graduation	58
1	Graduation	38
2	Graduation	26
3	Graduation	26
4	PhD	94
5	Master	16

In [22]:
```
#choosing the customers with an income higher than 75,000 and with a master's degree.
data.iloc[list((data.annual_income > 75000) & (data.educational_level == 'Master')),  :,]
```

Out[22]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
:01719	1980	Master	Single	76995.0	3/28/2013	91	11	9	4	3	11
:01752	1964	Master	Single	79143.0	8/11/2012	2	6	13	0	3	11
:01756	1955	Master	Married	82384.0	11/19/2012	55	3	13	0	3	11
:01761	1982	Master	Single	75777.0	7/4/2013	12	3	11	0	3	11
:01777	1993	Master	Married	75251.0	8/27/2012	34	7	5	0	3	11
:01810	1993	Master	Single	89058.0	12/7/2012	18	5	4	0	3	7
:01821	1957	Master	Relationship	88193.0	6/20/2013	65	6	10	0	5	11
:01841	1987	Master	Single	92859.0	10/19/2012	46	5	12	0	3	2
:01918	1985	Master	Widowed	83790.0	11/15/2013	81	8	6	0	3	11
:01978	1981	Master	Single	77882.0	4/30/2014	29	3	5	0	3	11
:02006	1983	Master	Widowed	80950.0	3/28/2013	44	6	9	0	3	11
:02124	1973	Master	Relationship	82584.0	6/4/2013	26	3	8	0	3	11
:02136	1983	Master	Relationship	82634.0	6/21/2013	49	1	3	0	0	11

#Apply data operations: index, new variables, data types

In [26]:
```
#set the index as customer_id
data.set_index("customer_id")
```

Out[26]:

	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
ner_id											
201701	1982	Graduation	Single	58138.0	9/4/2012	58	8	4	0	3	11
201702	1950	Graduation	Married	46344.0	3/8/2014	38	1	2	0	3	11
201703	1965	Graduation	Divorced	71613.0	8/21/2013	26	8	10	0	3	11
201704	1984	Graduation	Relationship	26646.0	2/10/2014	26	2	4	0	3	11
201705	1981	PhD	Widowed	58293.0	1/19/2014	94	5	6	0	3	11
...	...	...	...	...	...	...	...	...	...	...	...
202195	1944	PhD	Divorced	55614.0	11/27/2013	85	9	6	0	3	11
202196	1962	Master	Divorced	59432.0	4/13/2013	88	5	11	0	3	11
202197	1978	Graduation	Divorced	55563.0	4/5/2014	22	2	3	0	3	11
202198	1971	PhD	Relationship	43624.0	4/21/2013	83	4	4	0	6	11
202199	1949	PhD	Relationship	41461.0	5/22/2014	63	6	11	0	6	11

ws × 11 columns

In [27]:
```
#sort the data by year_of_birth, ascending is default;
data.sort_values(by = ['year_of_birth '], ascending = True)
# if we want it in descending we should set ascending = False
```

Out[27]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
202040	1899	PhD	Single	83532.0	9/26/2013	36	4	4	0	3	11
201733	1940	Graduation	Married	40548.0	10/10/2012	31	2	4	0	3	11
202059	1943	Master	Married	65073.0	8/20/2013	65	5	5	1	3	11
201740	1943	PhD	Divorced	48948.0	2/1/2013	53	7	5	0	3	11
202195	1944	PhD	Divorced	55614.0	11/27/2013	85	9	6	0	3	11
...	...	...	...	...	...	...	...	...	...	...	...
201717	2000	Graduation	Married	41850.0	12/24/2012	51	3	3	7	3	11
202119	2000	Graduation	Single	91065.0	2/22/2013	33	7	9	0	3	11
201817	2000	Graduation	Relationship	90765.0	1/24/2014	25	4	5	0	3	11
201886	2000	Graduation	Single	25271.0	12/5/2012	45	1	2	0	3	11
202153	2000	Master	Single	36230.0	10/17/2013	17	2	4	0	3	11

12 columns

In [28]:
```
#create a new variable which is the sum of all purchases performed by customers
data['sum_purchases'] = data.online_purchases + data.store_purchases
data['sum_purchases']
```

Out[28]:
```
0      12
1       3
2      18
3       6
4      11
       ..
494    15
495    16
496     5
497     8
498    17
Name: sum_purchases, Length: 499, dtype: int64
```

In [29]:
```
#create an income category (low, meduim, high) based on the income variable
income_categories = ['Low', 'Meduim', 'High'] #set the categories
bins = [0,75000,120000,600000] #set the income boundaries
cats= pd.cut(data['annual_income'],bins, labels=income_categories) #apply the pd.cut method
data['Income_Category'] = cats #assign the categories based on income
data[['annual_income', 'Income_Category']]
```

Out[29]:

	annual_income	Income_Category
0	58138.0	Low
1	46344.0	Low
2	71613.0	Low
3	26646.0	Low
4	58293.0	Low
...	...	...
494	55614.0	Low
495	59432.0	Low
496	55563.0	Low
497	43624.0	Low
498	41461.0	Low

499 rows × 2 columns

In [46]:
```
#apply groupby to find the mean of income, recency, number of web and store purchases by educational group
aggregate_view = pd.DataFrame(data.groupby(by='educational_level')[['annual_income', 'recency', 'store_purchases',  'online_purch
aggregate_view
```

Out[46]:

	educational_level	annual_income	recency	store_purchases	online_purchases
0	Basic	19514.571429	53.571429	2.857143	1.571429
1	Graduation	51607.827309	47.171206	5.840467	3.887160
2	High School	44154.717949	58.400000	4.600000	3.450000
3	Master	51191.700000	45.000000	5.691358	4.049383
4	PhD	55878.990991	49.008772	6.298246	4.429825

In [49]:
```
#apply pivot table to find the aggregated sum of purchases and mean of recency per education and marital status group
import numpy as np
pivot_table = pd.DataFrame(pd.pivot_table(data, values=['sum_purchases', 'recency'], index=['marital_status'],
columns=['educational_level'], aggfunc={'recency': np.mean, 'sum_purchases': np.sum}, fill_value=0)).reset_index()
pivot_table
```

Out[49]:

	marital_status	recency					sum_purchases				
educational_level		Basic	Graduation	High School	Master	PhD	Basic	Graduation	High School	Master	PhD
0	Divorced	68.333333	54.897959	64.666667	60.083333	41.350000	13	481	31	134	232
1	Married	0.000000	42.701493	66.866667	50.315789	60.000000	0	652	120	193	236
2	Relationship	39.333333	48.196078	49.615385	36.800000	43.161290	15	464	99	159	364
3	Single	52.000000	44.278689	49.000000	42.761905	49.315789	3	623	54	218	173
4	Widow	0.000000	61.000000	96.000000	14.000000	25.000000	0	34	8	14	6
5	Widowed	0.000000	46.760000	52.000000	40.000000	53.684211	0	246	10	71	212

3. SciPy:

```
In [4]: #Data Analysis with SciPy
        # import numpy library
        import numpy as np
        A = np.array([[1,2,3],[4,5,6],[7,8,8]])
```

```
In [5]: #Linear Algebra
        #Determinant of a Matrix
        # importing linalg function from scipy
        from scipy import linalg

        # Compute the determinant of a matrix
        linalg.det(A)

Out[5]: 2.999999999999997
```

```
In [6]: #pivoted LU decomposition of a matrix
        P, L, U = linalg.lu(A)
        print(P)
        print(L)
        print(U)
        # print LU decomposition
        print(np.dot(L,U))

        [[0. 1. 0.]
         [0. 0. 1.]
         [1. 0. 0.]]
        [[1.         0.         0.        ]
         [0.14285714 1.         0.        ]
         [0.57142857 0.5        1.        ]]
        [[7.         8.         8.        ]
         [0.         0.85714286 1.85714286]
         [0.         0.         0.5       ]]
        [[7. 8. 8.]
         [1. 2. 3.]
         [4. 5. 6.]]
```

```
In [7]: #Eigen values and eigen vectors of above matrix
        eigen_values, eigen_vectors = linalg.eig(A)
        print(eigen_values)
        print(eigen_vectors)

        [15.55528261+0.j -1.41940876+0.j -0.13587385+0.j]
        [[-0.24043423 -0.67468642  0.51853459]
         [-0.54694322 -0.23391616 -0.78895962]
         [-0.80190056  0.70005819  0.32964312]]
```

```
In [8]: #linear equations
        v = np.array([[2],[3],[5]])
        print(v)
        s = linalg.solve(A,v)
        print(s)

        [[2]
         [3]
         [5]]
        [[-2.33333333]
         [ 3.66666667]
         [-1.        ]]
```

```
In [9]: #Sparse Linear Algebra
        from scipy import sparse
        # Row-based linked list sparse matrix
        A = sparse.lil_matrix((1000, 1000))
        print(A)

        A[0,:100] = np.random.rand(100)
        A[1,100:200] = A[0,:100]
        A.setdiag(np.random.rand(1000))
        print(A)
```

```
  (0, 0)	0.11616892671917378
  (0, 1)	0.13503257433879967
  (0, 2)	0.9618747187565171
  (0, 3)	0.02899256849300469
  (0, 4)	0.850262131087913
  (0, 5)	0.9346351616745983
  (0, 6)	0.21428777850603808
  (0, 7)	0.7398978235086023
  (0, 8)	0.09159219936893082
  (0, 9)	0.21523310318480082
  (0, 10)	0.9050708143647447
  (0, 11)	0.8348462936615604
  (0, 12)	0.9042726075329924
  (0, 13)	0.5666525054114153
  (0, 14)	0.27382290310454094
  (0, 15)	0.8697402189342641
  (0, 16)	0.3328942783310157
  (0, 17)	0.382150244305717
```

```
In [10]: #Integration
         import scipy.integrate
         f= lambda x:np.exp(-x**2)
         # print results
         i = scipy.integrate.quad(f, 0, 1)
         print(i)
```

```
(0.7468241328124271, 8.291413475940725e-15)
```

```
In [11]: #Double Integrals
         from scipy import integrate
         f = lambda y, x: x*y**2
         i = integrate.dblquad(f, 0, 2, lambda x: 0, lambda x: 1)
         # print the results
         print(i)
```

```
(0.6666666666666667, 7.401486830834377e-15)
```

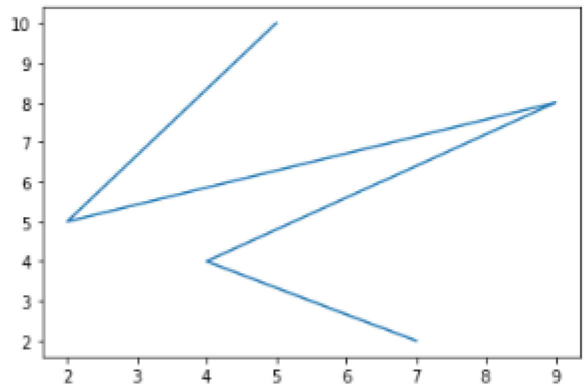
4. Matplotlib:

```
In [1]: # importing matplotlib module
        from matplotlib import pyplot as plt
```

```
In [2]: # x-axis values
        x = [5, 2, 9, 4, 7]
```

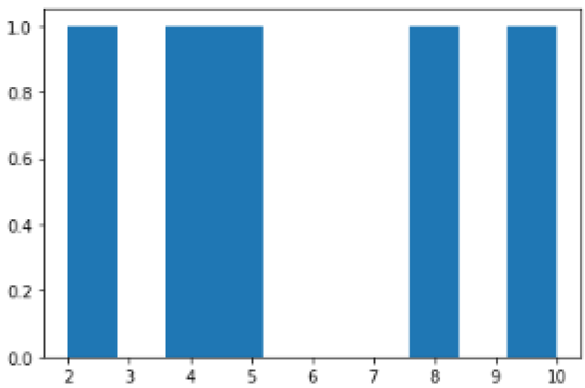
```
In [3]: # Y-axis values
        y = [10, 5, 8, 4, 2]
```

```
In [4]: # Function to plot
        plt.plot(x, y)

Out[4]: [<matplotlib.lines.Line2D at 0x1f57f64f460>]
```

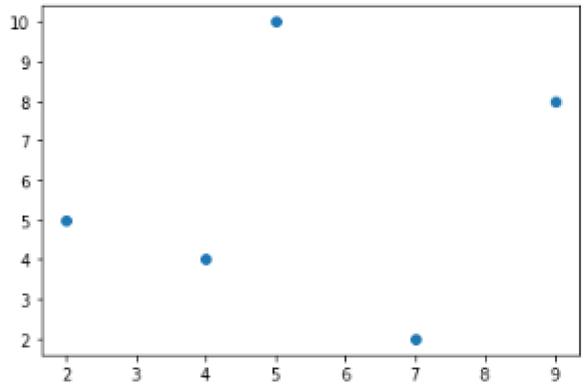



```
In [5]: # function to show the plot
        plt.show()
```

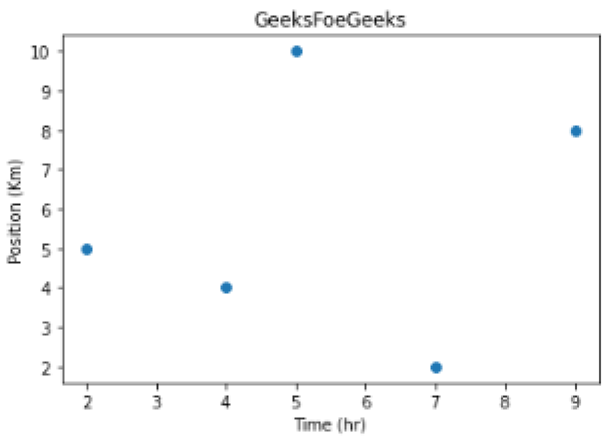
```
In [1]: #Histogram
        from matplotlib import pyplot as plt

        # Y-axis values
        y = [10, 5, 8, 4, 2]

        # Function to plot histogram
        plt.hist(y)

        # Function to show the plot
        plt.show()
```



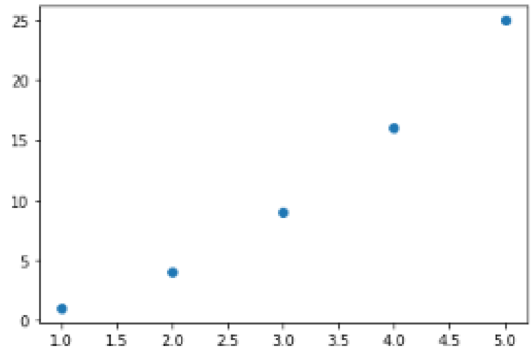
In [2]:
```
#Scatter Plot
# x-axis values
x = [5, 2, 9, 4, 7]

# Y-axis values
y = [10, 5, 8, 4, 2]

# Function to plot scatter
plt.scatter(x, y)

# function to show the plot
plt.show()
```

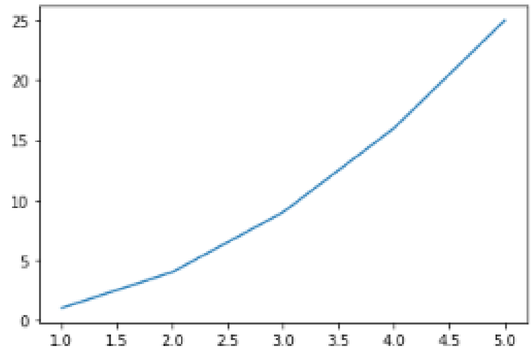


In [3]:
```
#Adding title and Labeling the Axes in the graph
# x-axis values
x = [5, 2, 9, 4, 7]

# Y-axis values
y = [10, 5, 8, 4, 2]

# Function to plot
plt.scatter(x, y)

# Adding Title
plt.title("GeeksFoeGeeks")

# Labeling the axes
plt.xlabel("Time (hr)")
plt.ylabel("Position (Km)")

# function to show the plot
plt.show()
```



In [4]:
```
#Multiple Graphs
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
plt.scatter(x, y)

# function to show the plot
plt.show()

plt.plot(x, y)

# function to show the plot
plt.show()
```





Artificial Intelligence & Machin Learning

Experiment No. 6

Introduction to Linear Regression, Logistic regression, KNN- classification.

Name: Patel Arun Ramjanak Roll No. 26

PRACTICAL NO. 6

Aim: Implementation of linear regression, logistic regression, KNN,- classification.**Objective:** Understand linear regression, logistic regression, KNN,- classification. **Software Requirement:**

- **Anaconda Navigator:** Anaconda Navigator is a desktop graphical user interface included in Anaconda that allows you to launch applications and easily manage conda packages, environments and channels without the need to use command line commands.

Theory:

- **Linear Regression:** Linear regression strives to show the relationship between two variables by applying a linear equation to observed data. One variable is supposed to be an independent variable, and the other is to be a dependent variable.
- **Logistic Regression:** Logistic regression is a process of **modeling the probability of a discrete outcome given an input variable**. The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on.
- **KNN Classification:** k-nearest neighbours (knn) is a **non-parametric classification method**, i.e. we do not have to assume a parametric model for the data of the classes Calculate the distance between the query-instance (new observation) and all the training samples Sort the distances and determine the nearest neighbours based on the k-th minimum distance.

24/5/22

\+ KNN Classification

• Given Data Set

Name	Age	Gender	Sports	Dist
Ajay	32	M	Football	27.02
Mark	40	M	Neither	35.01
Sam	16	F	Cricket	11.00
Tom	34	F	Cricket	29.00
Sachin	55	M	Neither	50.01
Rahul	40	M	Cricket	35.01
Pooja	20	F	Neither	15.00
Smith	15	M	Cricket	10.00
Laxmi	55	F	Football	50.00
Jolly	15	M	Football	
Angelina	5	F		

Consider K = 3 (We will find 3 closest neighbours)
Consider Male = 0 female = 1

Apply Euclidean Distance formula to find distance between Angelina and other people.

$= \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

1st person Ajay Age = 32 Gender = male = 0

$= \sqrt{(5-32)^2+(1-0)^2}$

$= \sqrt{729+1}$

$= 27.024$

2nd person Mark Age = 40 Gender = male = 0

$= \sqrt{(5-43)^2+(1-0)^2}$

$= \sqrt{1444+1}$

$= 35.01$

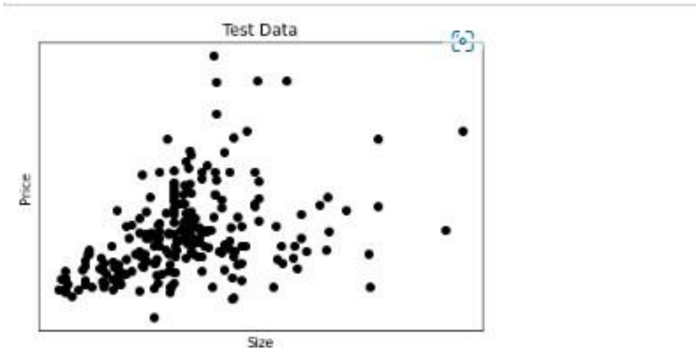
Date:

As we have decided
K=3, find out 3 closest neighbour
Sam = 11.00 Cricket
~~Tom = 9/1000~~ ~~Cricket~~
Smith = 10.00 Cricket
Jolly = 10.05 Football

Code & Output:

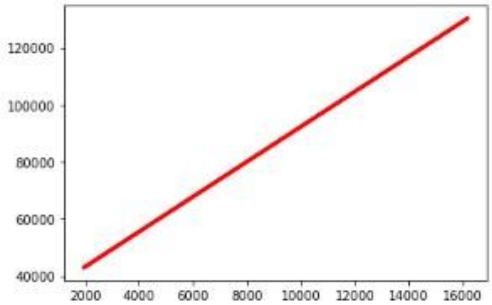
Linear Regression:

In [10]:
```
import matplotlib

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
import pandas as pd

# Load CSV and columns
df = pd.read_csv("Housing.csv")

Y = df['price']
X = df['lotsize']

X=X.values.reshape(len(X),1)
Y=Y.values.reshape(len(Y),1)

# Split the data into training/testing sets
X_train = X[:-250]
X_test = X[-250:]

# Split the targets into training/testing sets
Y_train = Y[:-250]
Y_test = Y[-250:]

# Plot outputs
plt.scatter(X_test, Y_test,  color='black')
plt.title('Test Data')
plt.xlabel('Size')
plt.ylabel('Price')
plt.xticks(())
plt.yticks(())

plt.show()
```



In [11]:
```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(X_train, Y_train)

# Plot outputs
plt.plot(X_test, regr.predict(X_test), color='red',linewidth=3)
```

Out[11]: [<matplotlib.lines.Line2D at 0x2cc4ed76430>]



Logistic Regression:

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

from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
import seaborn as sns
sns.set(style="white") #white background style for seaborn plots
sns.set(style="whitegrid", color_codes=True)

import warnings
warnings.simplefilter(action='ignore')
```

In [4]:
```
# Read CSV train data file into DataFrame
train_df = pd.read_csv("titanic_train.csv")

# Read CSV test data file into DataFrame
test_df = pd.read_csv("titanic_test.csv")

# preview train data
train_df.head()
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [5]:
```
print('The number of samples into the train data is {}.'.format(train_df.shape[0]))
```

```
The number of samples into the train data is 891.
```

In [6]:
```
test_df.head()
```

Out[6]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

In [7]:
```
print('The number of samples into the test data is {}.'.format(test_df.shape[0]))
```

```
The number of samples into the test data is 418.
```

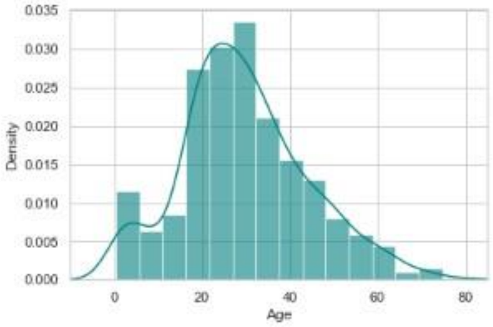
In [8]:
```
# check missing values in train data
train_df.isnull().sum()
```

Out[8]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [9]:
```
# percent of missing "Age"
print('Percent of missing "Age" records is %.2f%%' %((train_df['Age'].isnull().sum()/train_df.shape[0])*100))
```

```
Percent of missing "Age" records is 19.87%
```

In [10]:
```python
ax = train_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', alpha=0.6)
train_df["Age"].plot(kind='density', color='teal')
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```



In [11]:
```python
# mean age
print('The mean of "Age" is %.2f' %(train_df["Age"].mean(skipna=True)))
# median age
print('The median of "Age" is %.2f' %(train_df["Age"].median(skipna=True)))
```

```
The mean of "Age" is 29.70
The median of "Age" is 28.00
```

In [12]:
```python
# percent of missing "Cabin"
print('Percent of missing "Cabin" records is %.2f%%' %((train_df['Cabin'].isnull().sum()/train_df.shape[0])*100))
```

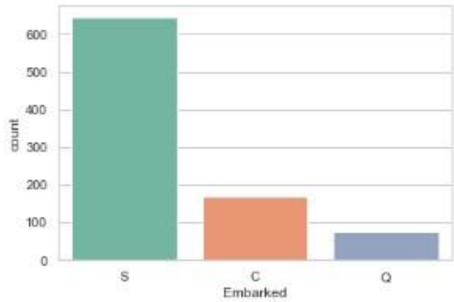
```
Percent of missing "Cabin" records is 77.10%
```

In [13]:
```python
 #percent of missing "Embarked"
print('Percent of missing "Embarked" records is %.2f%%' %((train_df['Embarked'].isnull().sum()/train_df.shape[0])*100))
```

```
Percent of missing "Embarked" records is 0.22%
```

In [14]:
```python
print('Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton):')
print(train_df['Embarked'].value_counts())
sns.countplot(x='Embarked', data=train_df, palette='Set2')
plt.show()
```

```
Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton):
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```



In [15]:
```python
print('The most common boarding port of embarkation is %s.' %train_df['Embarked'].value_counts().idxmax())
```

```
The most common boarding port of embarkation is S.
```

In [16]:
```
train_data = train_df.copy()
train_data["Age"].fillna(train_df["Age"].median(skipna=True), inplace=True)
train_data["Embarked"].fillna(train_df['Embarked'].value_counts().idxmax(), inplace=True)
train_data.drop('Cabin', axis=1, inplace=True)
```

In [17]:
```
# check missing values in adjusted train data
train_data.isnull().sum()
```

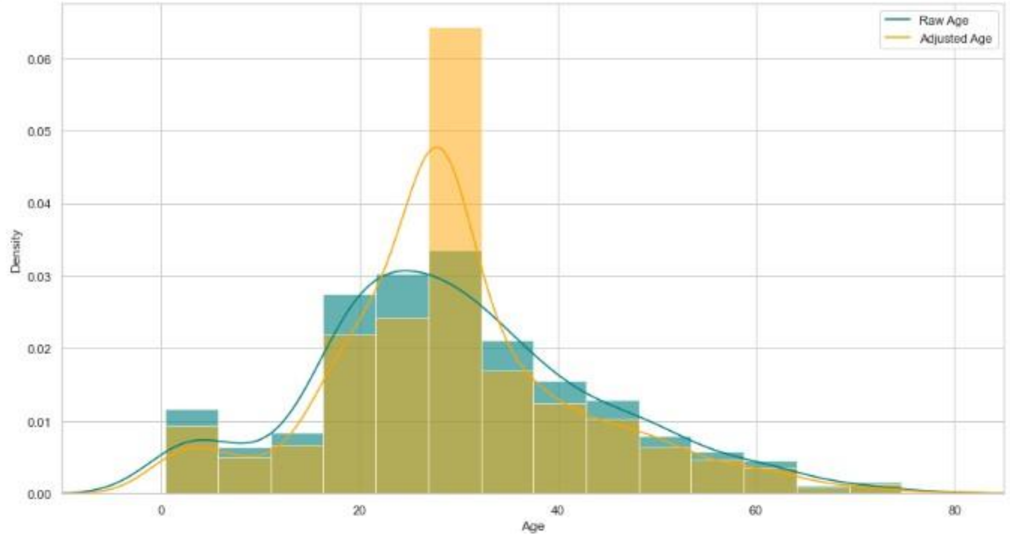
Out[17]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

In [18]:
```
# preview adjusted train data
train_data.head()
```

Out[18]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S

In [19]:
```
plt.figure(figsize=(15,8))
ax = train_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', alpha=0.6)
train_df["Age"].plot(kind='density', color='teal')
ax = train_data["Age"].hist(bins=15, density=True, stacked=True, color='orange', alpha=0.5)
train_data["Age"].plot(kind='density', color='orange')
ax.legend(['Raw Age', 'Adjusted Age'])
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```



In [20]:
```
# Create categorical variable for traveling alone
train_data['TravelAlone']=np.where((train_data["SibSp"]+train_data["Parch"])>0, 0, 1)
train_data.drop('SibSp', axis=1, inplace=True)
train_data.drop('Parch', axis=1, inplace=True)
```

In [21]:
```
#create categorical variables and drop some variables
training=pd.get_dummies(train_data, columns=["Pclass","Embarked","Sex"])
training.drop('Sex_female', axis=1, inplace=True)
training.drop('PassengerId', axis=1, inplace=True)
training.drop('Name', axis=1, inplace=True)
training.drop('Ticket', axis=1, inplace=True)

final_train = training
final_train.head()
```

Out[21]:

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

Now, apply the same changes to the test data. I will apply to same imputation for "Age" in the Test data as I did for my Training data (if missing, Age = 28). I'll also remove the "Cabin" variable from the test data, as I've decided not to include it in my analysis. There were no missing values in the "Embarked" port variable. I'll add the dummy variables to finalize the test set. Finally, I'll impute the 1 missing value for "Fare" with the median, 14.45.

In [22]:
```
test_df.isnull().sum()
```

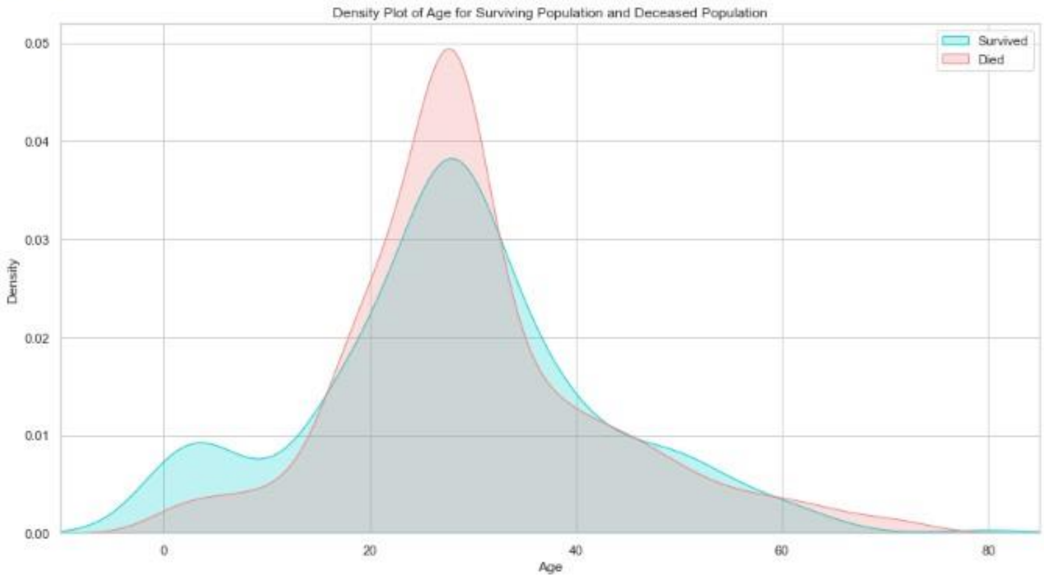
Out[22]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [23]:
```
test_data = test_df.copy()
test_data["Age"].fillna(train_df["Age"].median(skipna=True), inplace=True)
test_data["Fare"].fillna(train_df["Fare"].median(skipna=True), inplace=True)
test_data.drop('Cabin', axis=1, inplace=True)

test_data['TravelAlone']=np.where((test_data["SibSp"]+test_data["Parch"])>0, 0, 1)

test_data.drop('SibSp', axis=1, inplace=True)
test_data.drop('Parch', axis=1, inplace=True)

testing = pd.get_dummies(test_data, columns=["Pclass","Embarked","Sex"])
testing.drop('Sex_female', axis=1, inplace=True)
testing.drop('PassengerId', axis=1, inplace=True)
testing.drop('Name', axis=1, inplace=True)
testing.drop('Ticket', axis=1, inplace=True)

final_test = testing
final_test.head()
```

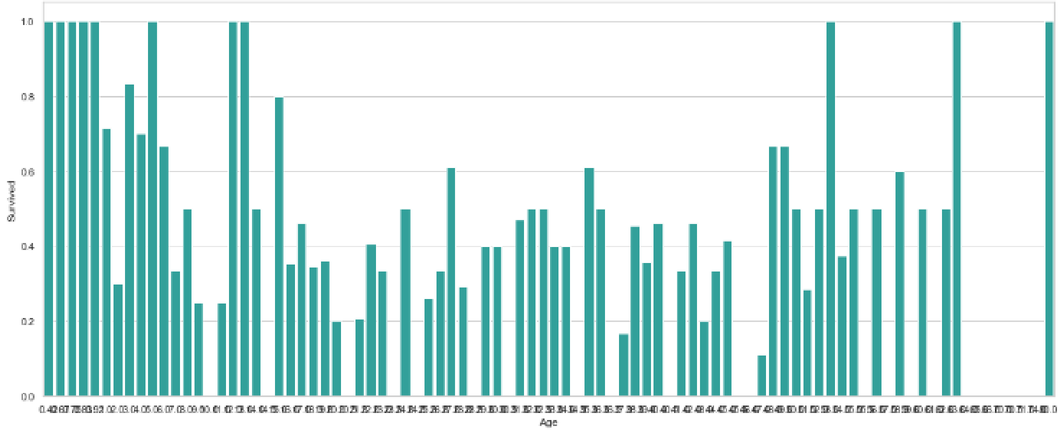
Out[23]:

	Age	Fare	TravelAlone	Pclass_1	Pclass_2	Pclass_3	Embarked_C	Embarked_Q	Embarked_S	Sex_male
0	34.5	7.8292	1	0	0	1	0	1	0	1
1	47.0	7.0000	0	0	0	1	0	0	1	0
2	62.0	9.6875	1	0	1	0	0	1	0	1
3	27.0	8.6625	1	0	0	1	0	0	1	1
4	22.0	12.2875	0	0	0	1	0	0	1	0

In [24]:
```
plt.figure(figsize=(15,8))
ax = sns.kdeplot(final_train["Age"][final_train.Survived == 1], color="darkturquoise", shade=True)
sns.kdeplot(final_train["Age"][final_train.Survived == 0], color="lightcoral", shade=True)
plt.legend(['Survived', 'Died'])
plt.title('Density Plot of Age for Surviving Population and Deceased Population')
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```

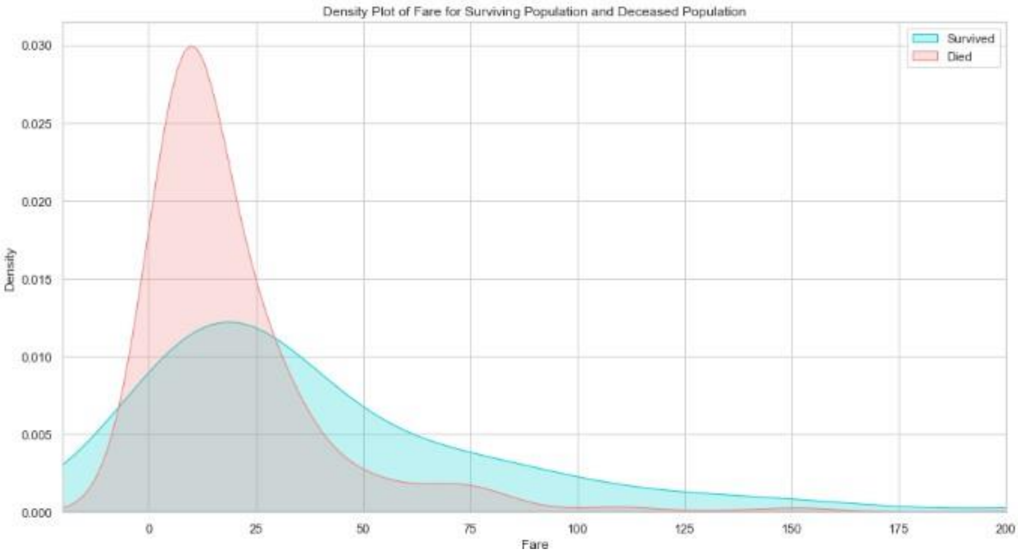


In [25]:
```
plt.figure(figsize=(20,8))
avg_survival_byage = final_train[["Age", "Survived"]].groupby(['Age'], as_index=False).mean()
g = sns.barplot(x='Age', y='Survived', data=avg_survival_byage, color="LightSeaGreen")
plt.show()
```

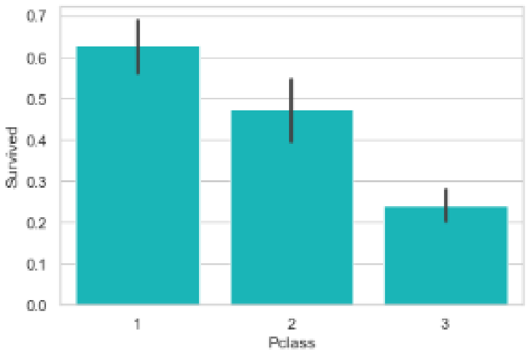


In [26]:
```
final_train['IsMinor']=np.where(final_train['Age']<=16, 1, 0)

final_test['IsMinor']=np.where(final_test['Age']<=16, 1, 0)
```

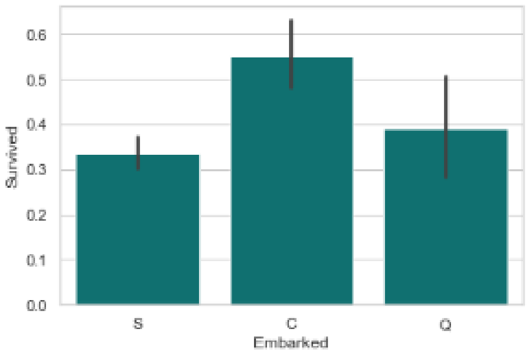
In [27]:
```
#Exploration of Fare
plt.figure(figsize=(15,8))
ax = sns.kdeplot(final_train["Fare"][final_train.Survived == 1], color="darkturquoise", shade=True)
sns.kdeplot(final_train["Fare"][final_train.Survived == 0], color="lightcoral", shade=True)
plt.legend(['Survived', 'Died'])
plt.title('Density Plot of Fare for Surviving Population and Deceased Population')
ax.set(xlabel='Fare')
plt.xlim(-20,200)
plt.show()
```



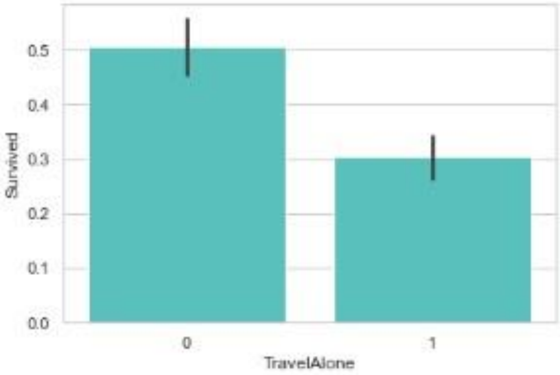
In [28]:
```
#Exploration of Passenger Class
sns.barplot('Pclass', 'Survived', data=train_df, color="darkturquoise")
plt.show()
```



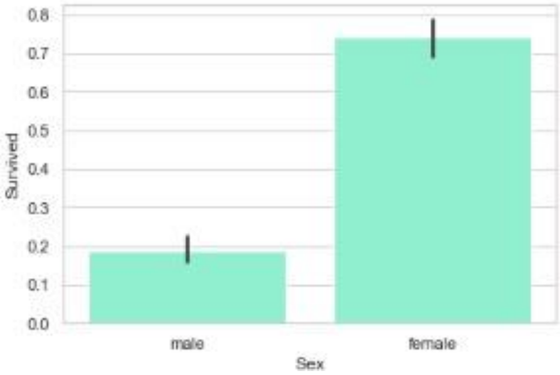
In [30]:
```
#Exploration of Embarked Port
sns.barplot('Embarked', 'Survived', data=train_df, color="teal")
plt.show()
```



```
In [31]: #Exploration of Traveling Alone vs. With Family
sns.barplot('TravelAlone', 'Survived', data=final_train, color="mediumturquoise")
plt.show()
```



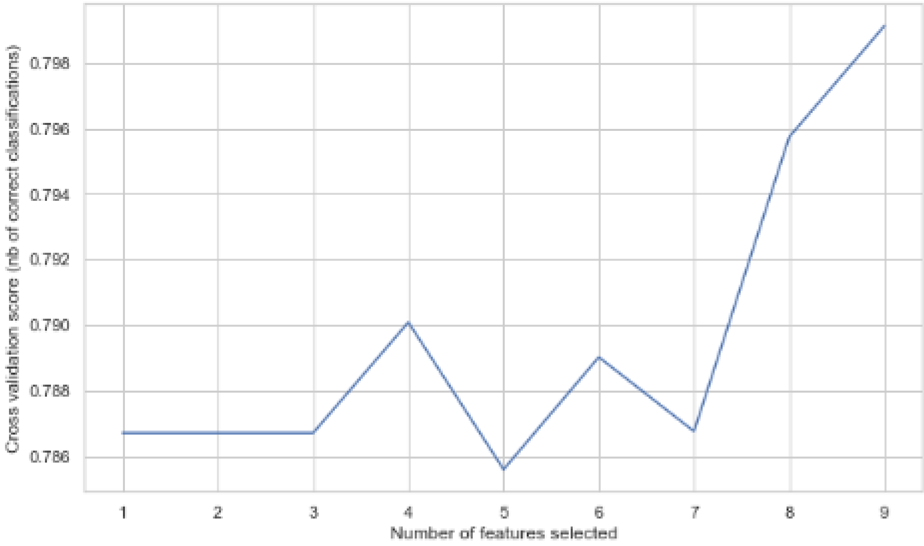

```
In [32]: #Exploration of Gender Variable
sns.barplot('Sex', 'Survived', data=train_df, color="aquamarine")
plt.show()
```



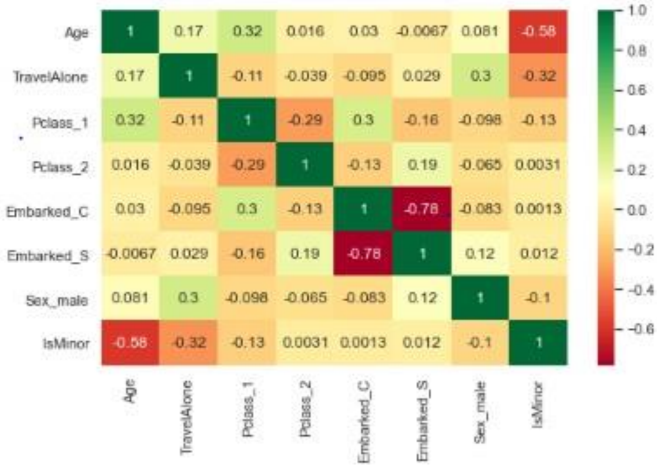
Sex

```
In [33]: #Logistic Regression and Results
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

cols = ["Age","Fare","TravelAlone","Pclass_1","Pclass_2","Embarked_C","Embarked_S","Sex_male","IsMinor"]
X = final_train[cols]
y = final_train['Survived']
# Build a logreg and compute the feature importances
model = LogisticRegression()
# create the RFE model and select 8 attributes
rfe = RFE(model, 8)
rfe = rfe.fit(X, y)
# summarize the selection of the attributes
print('Selected features: %s' % list(X.columns[rfe.support_]))

Selected features: ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C', 'Embarked_S', 'Sex_male', 'IsMinor']
```

```
In [34]: from sklearn.feature_selection import RFECV
# Create the RFE object and compute a cross-validated score.
# The "accuracy" scoring is proportional to the number of correct classifications
rfecv = RFECV(estimator=LogisticRegression(), step=1, cv=10, scoring='accuracy')
rfecv.fit(X, y)

print("Optimal number of features: %d" % rfecv.n_features_)
print('Selected features: %s' % list(X.columns[rfecv.support_]))

# Plot number of features VS. cross-validation scores
plt.figure(figsize=(10,6))
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()

Optimal number of features: 9
Selected features: ['Age', 'Fare', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C', 'Embarked_S', 'Sex_male', 'IsMinor']
```



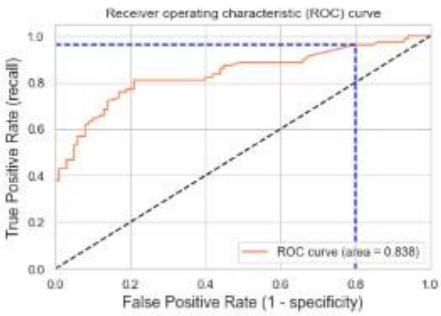

```
In [35]: Selected_features = ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C',
                             'Embarked_S', 'Sex_male', 'IsMinor']
         X = final_train[Selected_features]

         plt.subplots(figsize=(8, 5))
         sns.heatmap(X.corr(), annot=True, cmap="RdYlGn")
         plt.show()
```



	Age	TravelAlone	Pclass_1	Pclass_2	Embarked_C	Embarked_S	Sex_male	IsMinor
Age	1	0.17	0.32	0.016	0.03	-0.0067	0.081	-0.58
TravelAlone	0.17	1	-0.11	-0.039	-0.095	0.029	0.3	-0.32
Pclass_1	0.32	-0.11	1	-0.29	0.3	-0.16	-0.098	-0.13
Pclass_2	0.016	-0.039	-0.29	1	-0.13	0.19	-0.065	0.0031
Embarked_C	0.03	-0.095	0.3	-0.13	1	-0.78	-0.083	0.0013
Embarked_S	-0.0067	0.029	-0.16	0.19	-0.78	1	0.12	0.012
Sex_male	0.081	0.3	-0.098	-0.065	-0.083	0.12	1	-0.1
IsMinor	-0.58	-0.32	-0.13	0.0031	0.0013	0.012	-0.1	1

```
In [37]: #Review of model evaluation procedures
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_curve, auc, log_loss

# create X (features) and y (response)
X = final_train[Selected_features]
y = final_train['Survived']

# use train/test split with different random_state values
# we can change the random_state values that changes the accuracy scores
# the scores change a lot, this is why testing scores is a high-variance estimate
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)

# check classification scores of logistic regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
y_pred_proba = logreg.predict_proba(X_test)[:, 1]
[fpr, tpr, thr] = roc_curve(y_test, y_pred_proba)
print('Train/Test split results:')
print(logreg.__class__.__name__+" accuracy is %2.3f" % accuracy_score(y_test, y_pred))
print(logreg.__class__.__name__+" log_loss is %2.3f" % log_loss(y_test, y_pred_proba))
print(logreg.__class__.__name__+" auc is %2.3f" % auc(fpr, tpr))

idx = np.min(np.where(tpr > 0.95)) # index of the first threshold for which the sensibility > 0.95

plt.figure()
plt.plot(fpr, tpr, color='coral', label='ROC curve (area = %0.3f)' % auc(fpr, tpr))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot([0,fpr[idx]], [tpr[idx],tpr[idx]], 'k--', color='blue')
plt.plot([fpr[idx],fpr[idx]], [0,tpr[idx]], 'k--', color='blue')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - specificity)', fontsize=14)
plt.ylabel('True Positive Rate (recall)', fontsize=14)
plt.title('Receiver operating characteristic (ROC) curve')
plt.legend(loc="lower right")
plt.show()
print("Using a threshold of %.3f " % thr[idx] + "guarantees a sensitivity of %.3f " % tpr[idx] +
      "and a specificity of %.3f" % (1-fpr[idx]) +
      ", i.e. a false positive rate of %.2f%%." % (np.array(fpr[idx])*100))
```

```
Train/Test split results:
LogisticRegression accuracy is 0.782
LogisticRegression log_loss is 0.504
LogisticRegression auc is 0.838
```




```
Using a threshold of 0.070 guarantees a sensitivity of 0.962 and a specificity of 0.200, i.e. a false positive rate of 80.00%.
```

```
In [38]: # 10-fold cross-validation logistic regression
logreg = LogisticRegression()
# Use cross_val_score function
# We are passing the entirety of X and y, not X_train or y_train, it takes care of splitting the data
# cv=10 for 10 folds
# scoring = {'accuracy', 'neg_log_loss', 'roc_auc'} for evaluation metric - althought they are many
scores_accuracy = cross_val_score(logreg, X, y, cv=10, scoring='accuracy')
scores_log_loss = cross_val_score(logreg, X, y, cv=10, scoring='neg_log_loss')
scores_auc = cross_val_score(logreg, X, y, cv=10, scoring='roc_auc')
print('K-fold cross-validation results:')
print(logreg.__class__.__name__+" average accuracy is %2.3f" % scores_accuracy.mean())
print(logreg.__class__.__name__+" average log_loss is %2.3f" % -scores_log_loss.mean())
print(logreg.__class__.__name__+" average auc is %2.3f" % scores_auc.mean())
```

```
K-fold cross-validation results:
LogisticRegression average accuracy is 0.796
LogisticRegression average log_loss is 0.454
LogisticRegression average auc is 0.850
```

In [39]:
```
from sklearn.model_selection import cross_validate

scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}

modelCV = LogisticRegression()

results = cross_validate(modelCV, X, y, cv=10, scoring=list(scoring.values()),
                         return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print(modelCV.__class__.__name__+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -results['test_%s' % list(scorin
                              if list(scoring.values())[sc]=='neg_log_loss'
                              else results['test_%s' % list(scoring.values())[sc]].mean(),
                              results['test_%s' % list(scoring.values())[sc]].std()))
```

```
K-fold cross-validation results:
LogisticRegression average accuracy: 0.796 (+/-0.024)
LogisticRegression average log_loss: 0.454 (+/-0.037)
LogisticRegression average auc: 0.850 (+/-0.028)
```

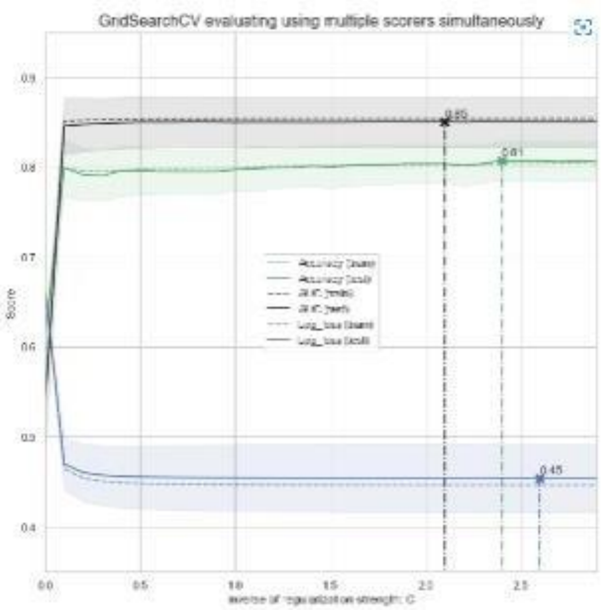
In [41]:
```
#What happens when we add the feature "Fare"?

cols = ["Age","Fare","TravelAlone","Pclass_1","Pclass_2","Embarked_C","Embarked_S","Sex_male","IsMinor"]
X = final_train[cols]

scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}

modelCV = LogisticRegression()

results = cross_validate(modelCV, final_train[cols], y, cv=10, scoring=list(scoring.values()),
                         return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print(modelCV.__class__.__name__+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -results['test_%s' % list(scorin
                              if list(scoring.values())[sc]=='neg_log_loss'
                              else results['test_%s' % list(scoring.values())[sc]].mean(),
                              results['test_%s' % list(scoring.values())[sc]].std()))
```

```
K-fold cross-validation results:
LogisticRegression average accuracy: 0.799 (+/-0.028)
LogisticRegression average log_loss: 0.455 (+/-0.037)
LogisticRegression average auc: 0.849 (+/-0.028)

K-fold cross-validation results:
LogisticRegression average accuracy: 0.799 (+/-0.028)
LogisticRegression average log_loss: 0.455 (+/-0.037)
LogisticRegression average auc: 0.849 (+/-0.028)
```

In [44]:

```
from sklearn.model_selection import GridSearchCV
X = final_train[Selected_features]

param_grid = {'C': np.arange(1e-05, 3, 0.1)}
scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}

gs = GridSearchCV(LogisticRegression(), return_train_score=True,
                  param_grid=param_grid, scoring=scoring, cv=10, refit='Accuracy')

gs.fit(X, y)
results = gs.cv_results_

print('='*20)
print("best params: " + str(gs.best_estimator_))
print("best params: " + str(gs.best_params_))
print('best score:', gs.best_score_)
print('='*20)

plt.figure(figsize=(10, 10))
plt.title("GridSearchCV evaluating using multiple scorers simultaneously",fontsize=16)

plt.xlabel("Inverse of regularization strength: C")
plt.ylabel("Score")
plt.grid()

ax = plt.axes()
ax.set_xlim(0, param_grid['C'].max())
ax.set_ylim(0.35, 0.95)

# Get the regular numpy array from the MaskedArray
X_axis = np.array(results['param_C'].data, dtype=float)

for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):
    for sample, style in (('train', '--'), ('test', '-')):
        sample_score_mean = -results['mean_%s_%s' % (sample, scorer)] if scoring[scorer]=='neg_log_loss' else results['mean_%s_%
        sample_score_std = results['std_%s_%s' % (sample, scorer)]
        ax.fill_between(X_axis, sample_score_mean - sample_score_std,
                        sample_score_mean + sample_score_std,
                        alpha=0.1 if sample == 'test' else 0, color=color)
        ax.plot(X_axis, sample_score_mean, style, color=color,
                alpha=1 if sample == 'test' else 0.7,
                label="%s (%s)" % (scorer, sample))

    best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
    best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer]=='neg_log_loss' else results['mean_test_%s' %

    # Plot a dotted vertical line at the best score for that scorer marked by x
    ax.plot([X_axis[best_index], ] * 2, [0, best_score],
            linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)

    # Annotate the best score for that scorer
    ax.annotate("%0.2f" % best_score,
                (X_axis[best_index], best_score + 0.005))

plt.legend(loc="best")
plt.grid('off')
plt.show()
```

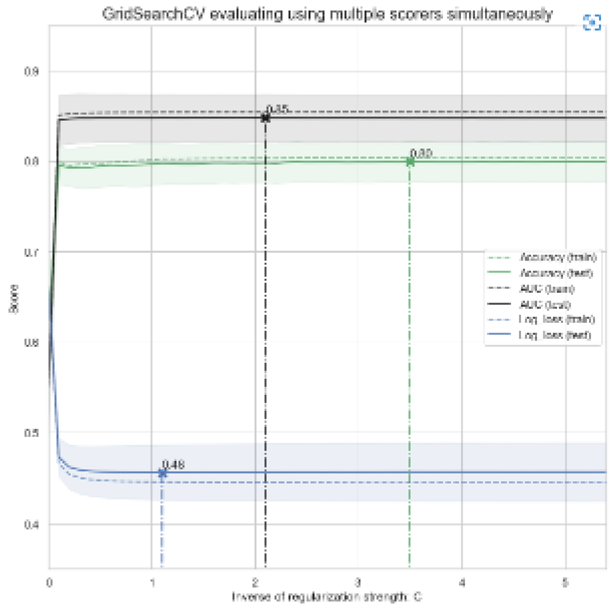
```
====================
best params: LogisticRegression(C=2.4000100000000004)
best params: {'C': 2.4000100000000004}
best score: 0.8069662921348316
====================
```



```
In [45]: from sklearn.preprocessing import StandardScaler
         from sklearn.model_selection import RepeatedStratifiedKFold
         from sklearn.pipeline import Pipeline

         #Define simple model
         ###############################################################################
         C = np.arange(1e-05, 5.5, 0.1)
         scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}
         log_reg = LogisticRegression()

         #Simple pre-processing estimators
         ###############################################################################
         std_scale = StandardScaler(with_mean=False, with_std=False)
         #std_scale = StandardScaler()

         #Defining the CV method: Using the Repeated Stratified K Fold
         ###############################################################################

         n_folds=5
         n_repeats=5

         rskfold = RepeatedStratifiedKFold(n_splits=n_folds, n_repeats=n_repeats, random_state=2)

         #Creating simple pipeline and defining the gridsearch
         ###############################################################################

         log_clf_pipe = Pipeline(steps=[('scale',std_scale), ('clf',log_reg)])

         log_clf = GridSearchCV(estimator=log_clf_pipe, cv=rskfold,
                       scoring=scoring, return_train_score=True,
                       param_grid=dict(clf__C=C), refit='Accuracy')

         log_clf.fit(X, y)
         results = log_clf.cv_results_

         print('='*20)
         print("best params: " + str(log_clf.best_estimator_))
         print("best params: " + str(log_clf.best_params_))
         print('best score:', log_clf.best_score_)
         print('='*20)

         plt.figure(figsize=(10, 10))
         plt.title("GridSearchCV evaluating using multiple scorers simultaneously",fontsize=16)
```

```
plt.xlabel("Inverse of regularization strength: C")
plt.ylabel("Score")
plt.grid()

ax = plt.axes()
ax.set_xlim(0, C.max())
ax.set_ylim(0.35, 0.95)

# Get the regular numpy array from the MaskedArray
X_axis = np.array(results['param_clf__C'].data, dtype=float)

for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):
    for sample, style in (('train', '--'), ('test', '-')):
        sample_score_mean = -results['mean_%s_%s' % (sample, scorer)] if scoring[scorer]=='neg_log_loss' else results['mean_%s_%
        sample_score_std = results['std_%s_%s' % (sample, scorer)]
        ax.fill_between(X_axis, sample_score_mean - sample_score_std,
                        sample_score_mean + sample_score_std,
                        alpha=0.1 if sample == 'test' else 0, color=color)
        ax.plot(X_axis, sample_score_mean, style, color=color,
                alpha=1 if sample == 'test' else 0.7,
                label="%s (%s)" % (scorer, sample))

    best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
    best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer]=='neg_log_loss' else results['mean_test_%s' %

    # Plot a dotted vertical line at the best score for that scorer marked by x
    ax.plot([X_axis[best_index], ] * 2, [0, best_score],
            linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)

    # Annotate the best score for that scorer
    ax.annotate("%0.2f" % best_score,
                (X_axis[best_index], best_score + 0.005))

plt.legend(loc="best")
plt.grid('off')
plt.show()
```

```
====================
best params: Pipeline(steps=[('scale', StandardScaler(with_mean=False, with_std=False)),
                ('clf', LogisticRegression(C=3.50001))])
best params: {'clf__C': 3.50001}
best score: 0.7995518172117255
====================
```



In [46]:
```
final_test['Survived'] = log_clf.predict(final_test[Selected_features])
final_test['PassengerId'] = test_df['PassengerId']

submission = final_test[['PassengerId','Survived']]

submission.to_csv("submission.csv", index=False)

submission.tail()
```

Out[46]:

	PassengerId	Survived
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

3.KNN Classification:

```
In [1]: from sklearn.model_selection import train_test_split
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import classification_report, confusion_matrix
        from sklearn import datasets

In [2]: iris=datasets.load_iris()

In [3]: x = iris.data
        y = iris.target

In [4]: print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
        print(x)
        print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
        print(y)

        sepal-length sepal-width petal-length petal-width
        [[5.1 3.5 1.4 0.2]
         [4.9 3.  1.4 0.2]
         [4.7 3.2 1.3 0.2]
         [4.6 3.1 1.5 0.2]
         [5.  3.6 1.4 0.2]
         [5.4 3.9 1.7 0.4]
         [4.6 3.4 1.4 0.3]
         [5.  3.4 1.5 0.2]
         [4.4 2.9 1.4 0.2]
         [4.9 3.1 1.5 0.1]
         [5.4 3.7 1.5 0.2]
         [4.8 3.4 1.6 0.2]
         [4.8 3.  1.4 0.1]
         [4.3 3.  1.1 0.1]
         [5.8 4.  1.2 0.2]
         [5.7 4.4 1.5 0.4]
         [5.4 3.9 1.3 0.4]
         [5.1 3.5 1.4 0.3]

In [5]: x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

In [6]: #To Training the model and Nearest nighbors K=5
        classifier = KNeighborsClassifier(n_neighbors=5)
        classifier.fit(x_train, y_train)

Out[6]: KNeighborsClassifier()

In [7]: #To make predictions on our test data
        y_pred=classifier.predict(x_test)

In [8]: print('Confusion Matrix')
        print(confusion_matrix(y_test,y_pred))
        print('Accuracy Metrics')
        print(classification_report(y_test,y_pred))

        Confusion Matrix
        [[12  0  0]
         [ 0 12  1]
         [ 0  1 19]]
        Accuracy Metrics
                      precision    recall  f1-score   support

                   0       1.00      1.00      1.00        12
                   1       0.92      0.92      0.92        13
                   2       0.95      0.95      0.95        20

            accuracy                           0.96        45
           macro avg       0.96      0.96      0.96        45
        weighted avg       0.96      0.96      0.96        45
```