Vishnu Waman Thakur Charitable Trust's
VIVA INSTITUTE OF TECHNOLOGY
(Approved by AICTE, New Delhi, DTE, Govt. of Maharashtra and Affiliated to the University of Mumbai)

Academic Year: 2022-23

INDEX

Subject: AIML Lab **Course Code: MCAL21**

Class/Sem: MCA SEM-II (Batch-2) **Faculty Name:** Prof. Sonia Dubey

Sr. No.	Experiment List	Signature
1.	Introduction to SWI-PROLOG Programming with help of simple Programs and demonstration of Rules, Facts & Relationship.	
2.	Solve Water-Jug Problem Using DFS.	
3.	Design Tic Tac Toe Using BFS.	
4.	Design Hill-climbing algorithm to solve 8- Puzzle Problem.	
5.	**Introduction to Python Programming:** ➢ Study of Numpy, Scipy library and matplotlib library ➢ Study of Pandas library.	
6.	**Implementation of Linear Regression:** ➢ Read data using .csv File and Implement the parametric Regression algorithm in order to find the best fit straight line. ➢ Read data using .csv File and Implement the parametric Regression algorithm in order to find ROC. ➢ Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.	
7.	**Implementation of Dimensionality reduction techniques: -** ➢ Write a Program to demonstrate dimensionality reduction techniques (feature Selection, feature extraction) Using PCA.	

8.	**Implementation of Clustering algorithms:** ➢ Write a program to implement classification of iris dataset using K Means clustering algorithm with K=3. ➢ write a program to demonstrate K-medoid (PAM) Algorithm with K=3.	
9.	**Implementation of classifying data using Support Vector machine:** ➢ Write a program to find Hyperplane using Linear SVM for iris dataset. ➢ Write a program to demonstrate non-linear SVM with any one Sklearn dataset.	
10.	**Implementation of Bagging Algorithm:** ➢ Write a program to implement decision tree for iris dataset and plot the same. ➢ Write a program to demonstrate Random forest by using the concept of combination of fast and slow oscillation with regression. ➢ Write a program for classification of digits using Random forest.	
11.	**Implementation of Boosting Algorithm:** ➢ Write a program to demonstrate the concept of AdaBoost using toy dataset. ➢ Write a program to demonstrate the AdaBoost algorithm with Decision Tree as Weak Classifier and present Classification score. ➢ Write a program to demonstrate Gradient Descent Algorithm.	
12.	**Deployment of Machine Learning Models:** ➢ Create Web API to predict the salary of new join employee in an organization using Regression Classifier. ➢ Create Web API to make predictions about Flowers using iris dataset and logistic regression classifier.	

Practical No: 1

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

2.Objectives:

- Understand logical programming syntax and semantics
- Design programs in PROLOG language

3.Software Required: SWI-Prolog

4.Theory:

Prolog is a general-purpose logic programming language associated with artificial intelligence and computational linguistics. Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is declarative: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations. Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching databases, voice control systems, and filling templates.

Rules and facts:

Prolog programs describe relations, defined by means of clauses. Pure Prolog is restricted to Horn clauses.

There are two types of clauses: facts and rules.

A rule is of the form

Head:- Body.

and is read as "Head is true if Body is true". A rule's body consists of calls to predicates, which are called the rule's goals. The built-in predicate, /2 (meaning a 2arity operator with name,) denotes conjunction of goals, and; /2 denotes disjunction.

Conjunctions and disjunctions can only appear in the body, not in the head of a rule.

Clauses with empty bodies are called facts. An example of a fact is:

cat (tom).

which is equivalent to the rule:

cat(tom) :- true.

The built-in predicate true/0 is always true.

Given the above fact, one can ask:

is tom a cat?

?- cat(tom). Yes

what things are

cats?

? - cat(X).

X = tom

Clauses with bodies are called rules. An example of a rule is: animal(X) :- cat(X).

If we add that rule and ask what things are animals?
?- animal(X).
X = tom

Prolog must be able to handle arithmetic in order to be a useful general purpose programming language. However, arithmetic does not fit nicely into the logical scheme of things. That is, the concept of evaluating an arithmetic expression is in contrast to the straight pattern matching we have seen so far. For this reason, Prolog provides the built-in predicate 'is' that evaluates arithmetic expressions. Its syntax calls for the use of operators.

X is <arithmetic expression>
The variable X is set to the value of the arithmetic expression. On backtracking it is unassigned. The variable X is set to the value of the arithmetic expression. On backtracking it is unassigned. The arithmetic expression looks like an arithmetic expression in any other programming language.

Here is how to use Prolog as a calculator.

?- X is 2 + 2.

X = 4

?- X is 3 * 4 + 2.

X = 14

Parentheses clarify precedence.
?- X is 3 * (4 + 2).
X = 18
?- X is (8 / 4) / 2.
X = 1

In addition to 'is,' Prolog provides a number of operators that compare two numbers. These include 'greater than', 'less than', 'greater or equal than', and 'less or equal than.' They behave more logically, and succeed or fail according to whether the comparison is true or false. Notice the order of the symbols in the greater or equal than and less than or equal operators. They are specifically constructed not to look like an arrow, so that the use arrow symbols in programs is without confusion.
X > Y

X < Y

X >= Y
X =< Y

5.Procedure/ Program:

1 (A). Sample program to demonstrate Rules and facts.

Code:

likes(ram,mango).

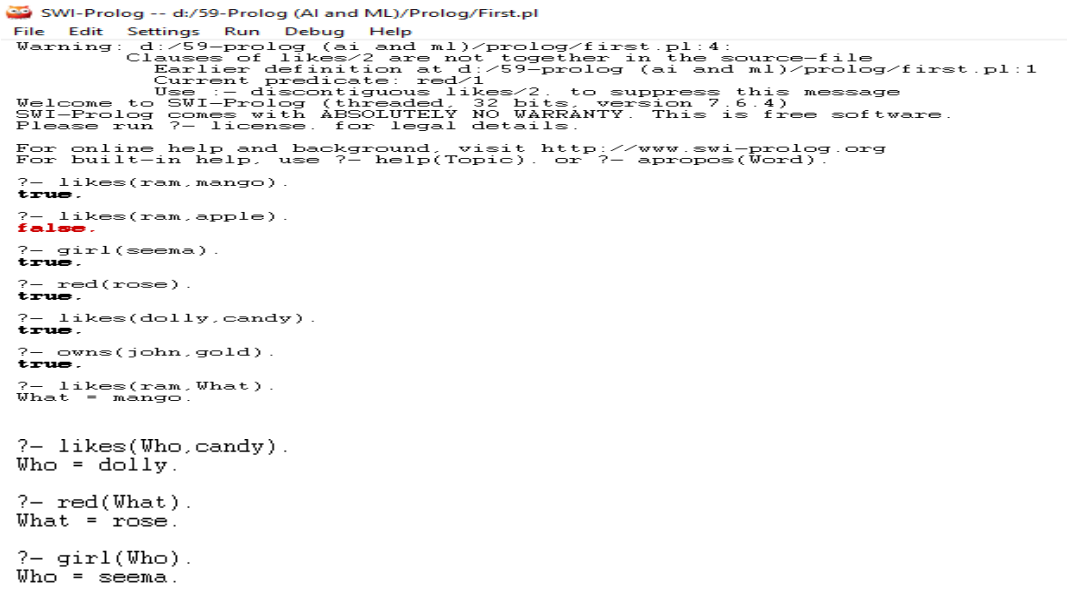
girl(seema).

red(rose).

likes(dolly,candy).

owns(john,gold).

Output:



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

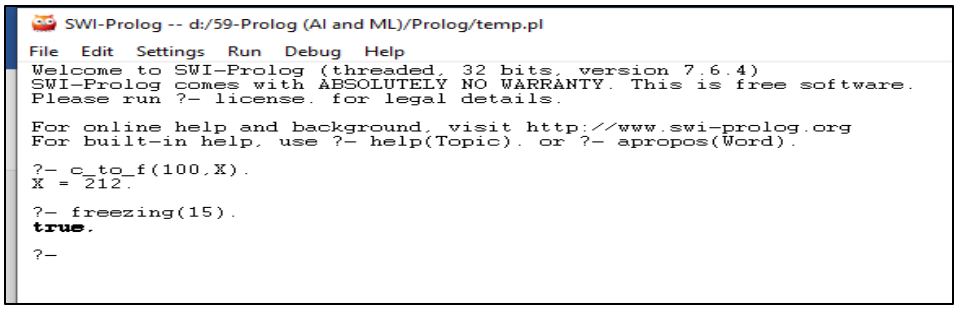
Code:

c_to_f(C, F):-

F is C * 9/5 +32.

freezing(F):- F=<32.

Output:

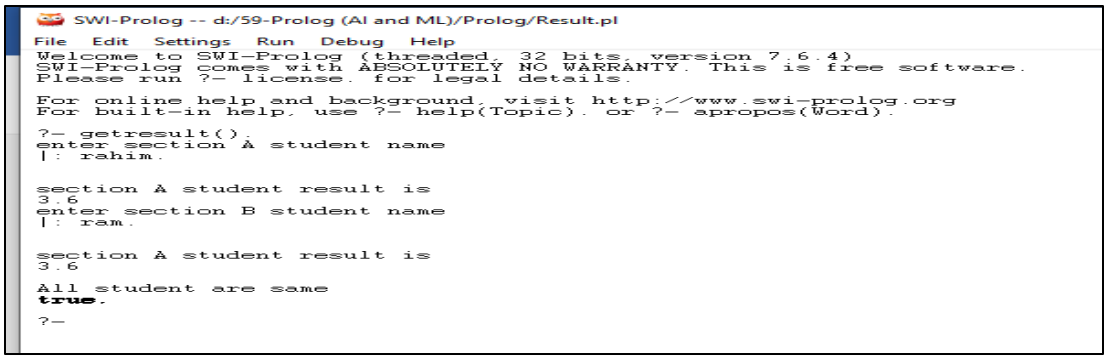


1.(C) Sample program to demonstrate the user-defined relationship in prolog.

Code:

```
%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 student 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"
```

Output:

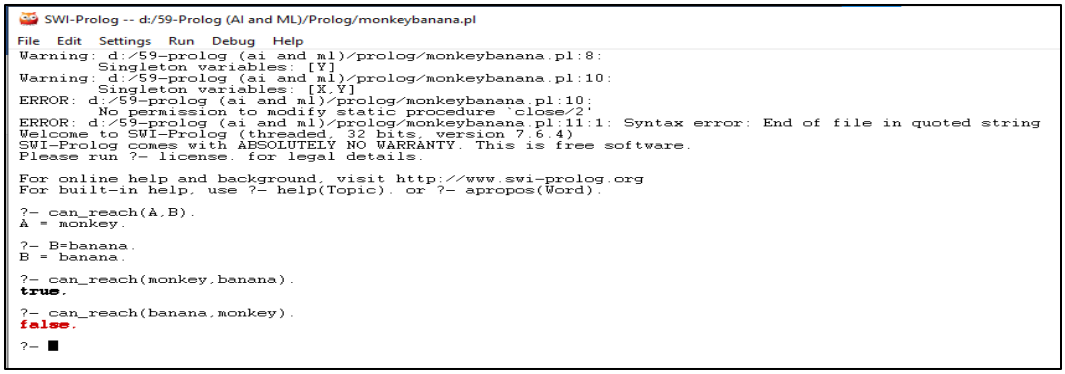


1.(D) Sample program to demonstrate the user-defined relationship in prolog.

Code:

```
in_room(bananas).
in_room(chair).
in_room(monkey).
clever(monkey).
can_climb(monkey,chair).
tall(chair).
can_move(monkey,chair,bananas).
can_reach(X,Y):-
clever(X).
close(X,Y).
under(Y,Z)"-
in_room(X),
in_room(Y),
in_room(Z),
can_y,
in_room(Z),
can_climb(X<Y<Z).
close(X,Y):-
get_on(X,Y),
under(Y,Z);
Tall(Y).
```

Output:



Conclusion:

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 problems are well understood.

Practical : 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: -**

We may sometimes search the goal along the largest depth of the tree, and move up only when further traversal along the depth is not possible. We then attempt to find alternative offspring of the parent of the node (state) last visited. If we visit the nodes of a tree using the above principles to search the goal, the traversal made is called depth first traversal and consequently the search strategy is called depth first search.

- **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:

c. 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

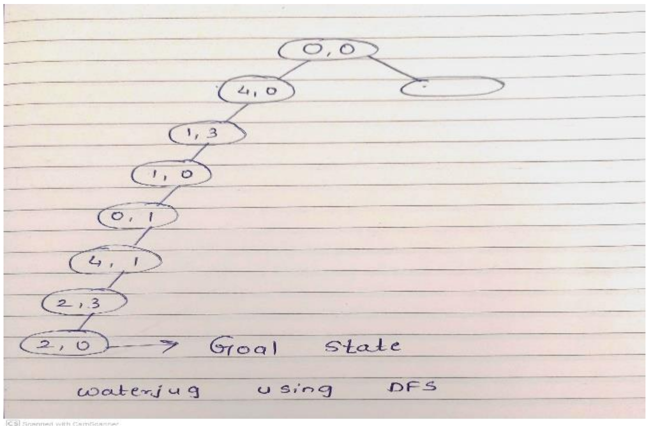
- **Water-Jug Problem**

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 liters, and jug B holds 3 liters. There is a pump, which can be used to fill either Jug. How can you get exactly 2 liters 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

5. Representational of algorithm



6. 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).

7.Output:

```
SWI-Prolog -- d:/59-AI and ML/Prolog/Waterjug.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 .

?-
```

8.Conclusion:

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

Practical No. 3

Tac Toe Using BFS

1. Aim: Design Tic Tac Toe Using BFS.

2. Objectives:

- Understand and implement the 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.

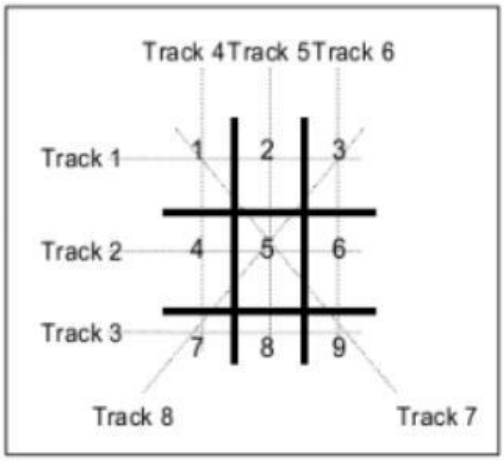
➢ It uses two queues for its implementation: open, close Queue ➢

Children are added from backend of queue.

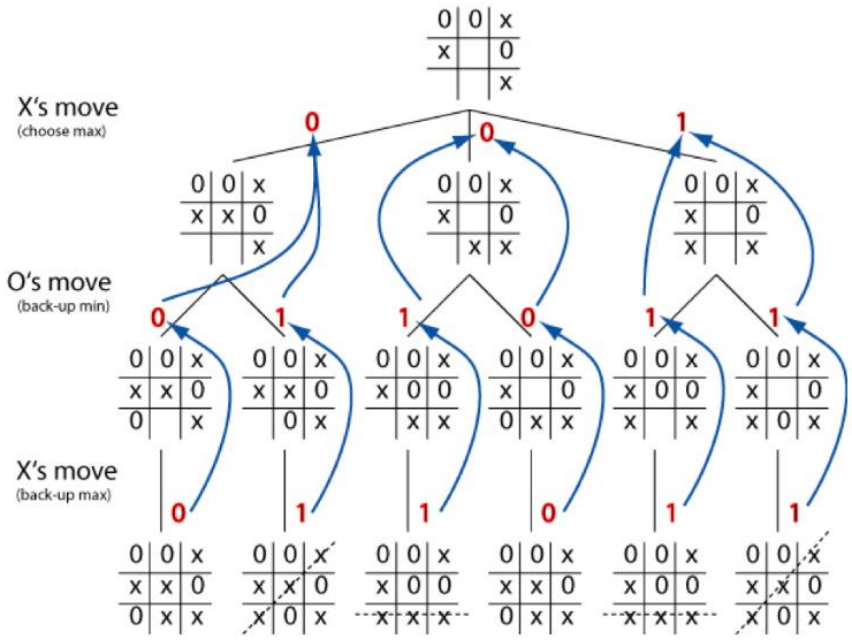
Algorithm

1. Create a single member queue comprising of root node.
2. If 1st Member of Queue is GOAL then goto Step 5.
3. If the 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 goto Step 2, If queue is empty then goto Step 6
5. Print "SUCCESS" and stop.
6. Print "FAILURE" 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 have 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 lays the foundation of the heuristics to be used in this study. These heuristics are somewhat similar to those proposed by Rich and Knight.



5.Code:

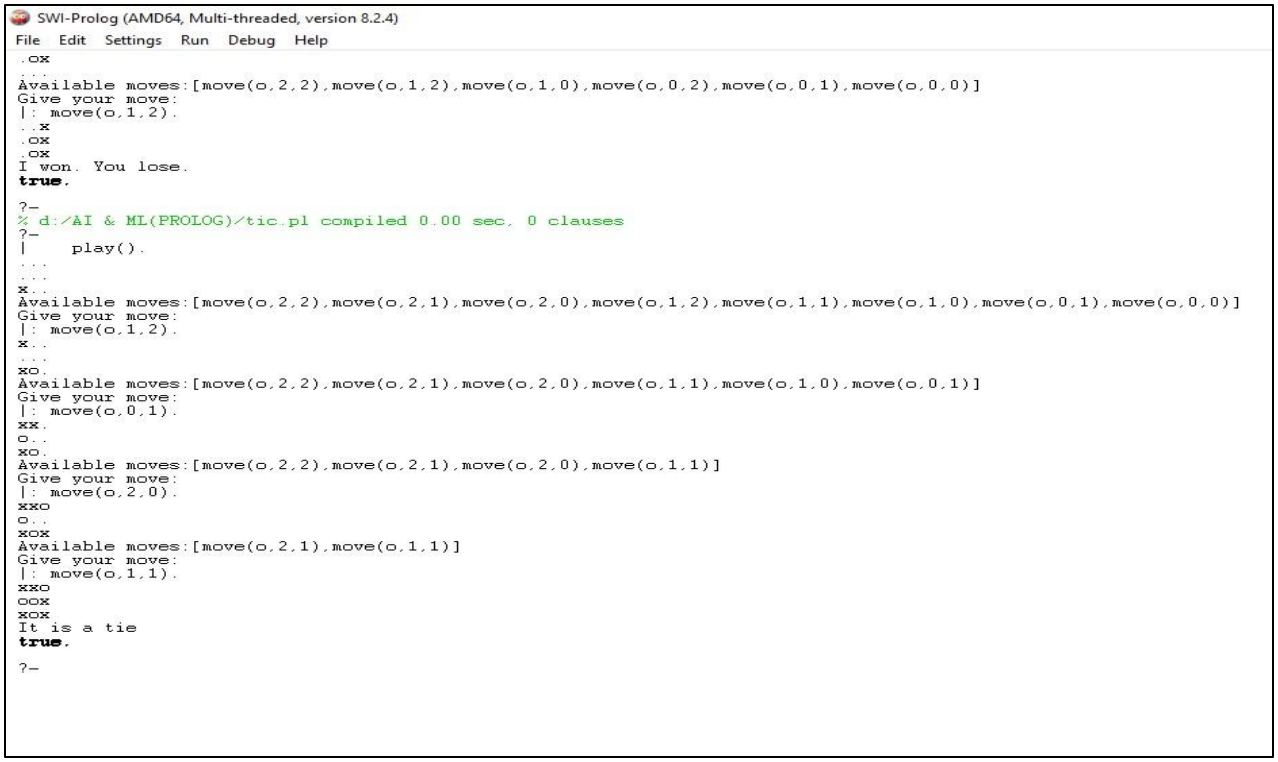
```
play :- my_turn([]).

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
```

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:/AI & ML(PROLOG)/tic.pl compiled 0.00 sec, 39 clauses
?-
|    play().
..x
...
...
Available moves:[move(o,2,2),move(o,2,1),move(o,1,2),move(o,1,1),move(o,1,0),move(o,0,2),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,1,1).
..x
.ox
...
Available moves:[move(o,2,2),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
.ox
.ox
I won. You lose.
true.

?-
```



7.Conclusion:

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

Practical No. 4

Hill Climbing

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

2. Objectives:

- Understand and Implement Hill climbing algorithm
- Understand Application Of hill climbing algorithm to solve 8 puzzle problem.

3. Software Required: SWI-Prolog

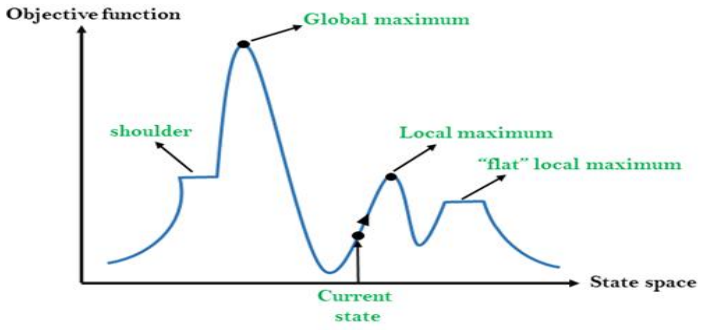
4. Theory:

• Hill Climbing algorithm: - Hill Climbing is a local search algorithm. The search algorithms that we have seen so far are 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 goal is irrelevant. For example, in the 8-queens problem, what matters is the final configuration of 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:

- they use very little memory-usually a constant amount.
- They can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable.

To understand local search, we will find it very useful to consider the state space landscape shown 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-a global 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:

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

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

● **8-PUZZLE Problem: -**

The eight puzzle consists of a 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. The objective of the puzzle is to find the sequence of tile movements that leads from a starting configuration to a goal configuration such as that shown in the figure a.

3	8	1
6	2	5
	4	7

A start configuration

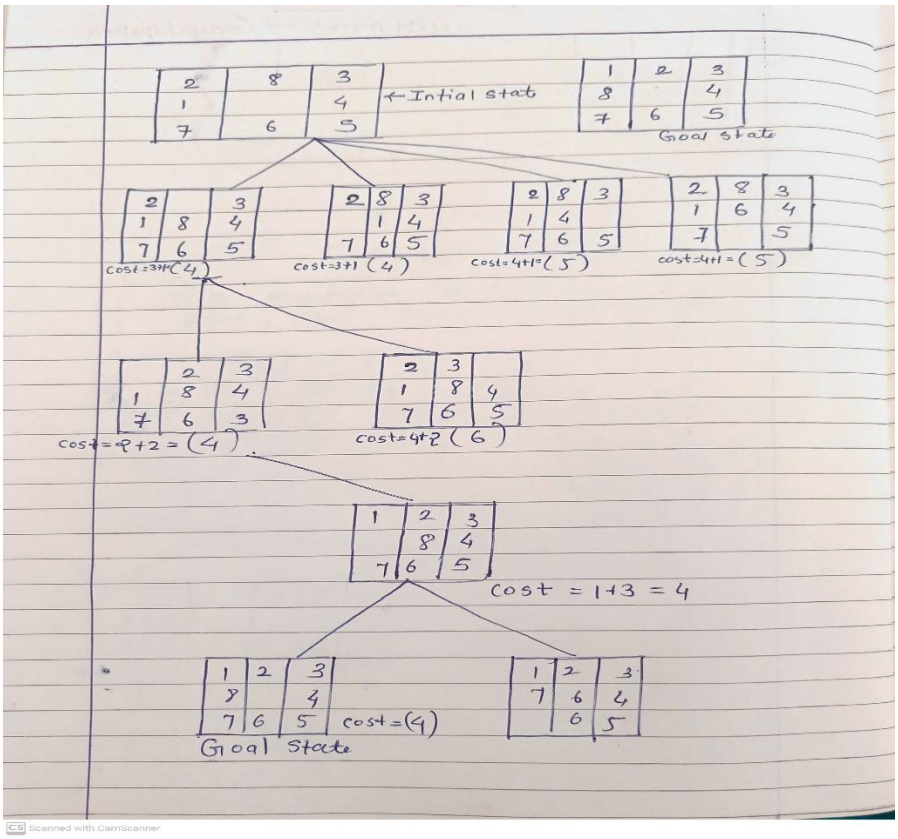
1	2	3
8		4
7	6	5

A goal configuration

Fig-a

The states of the eight puzzles are the different permutations of the tiles within the frame. The operations are 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.Representational of algorithm

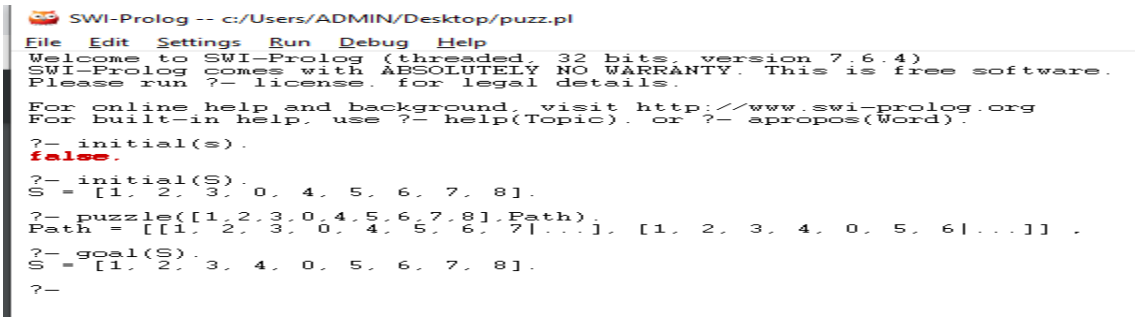


6.Code: -

```
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]).

%% 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],
   [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).
```

7.Output:



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 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.

Practical No. 5

Introduction to Python Programming: Learn the different libraries.

- **NumPy**
- **Pandas**
- **Matplotlib**

Aim: Understanding of basic python programming.

Objectives:

1. Practical implementation of Basic Python Commands
2. Learning and Implementation of basic python libraries (NumPy, Pandas and Matplotlib).

Software Required: Jupyter Notebook

Theory:

1. **Definition of Python:** Python has a simple syntax similar to the English language. Python has syntax that allows developers to write programs with fewer lines than some other programming languages. Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
2. **NumPy**: NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.
3. **Pandas**: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.
4. **Matplotlib**: Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes, etc.

Conclusion:

Conclusion. Python is a powerful programming language for data science, and the availability of powerful libraries like Pandas, NumPy, and Matplotlib makes it even more powerful. With these libraries, data scientists can perform complex data analysis and visualization tasks quickly and easily.

MCAL21: Artificial Intelligence And Machine Learning

Practical No. 6

Aim: Understanding and implementation of classification and regression algorithm

Objectives:

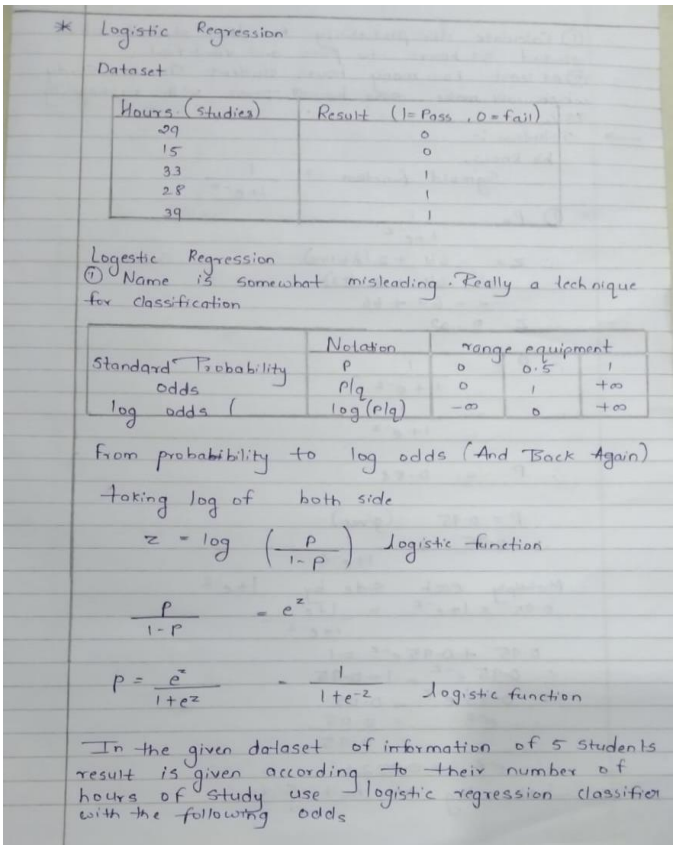
1. Understanding and Implementation of Linear Regression Algorithm
2 .Understanding and Implementation of Logistic Regression Algorithm
3 .Understanding and Implementation of Knearest Neighbor Classification Algorithm

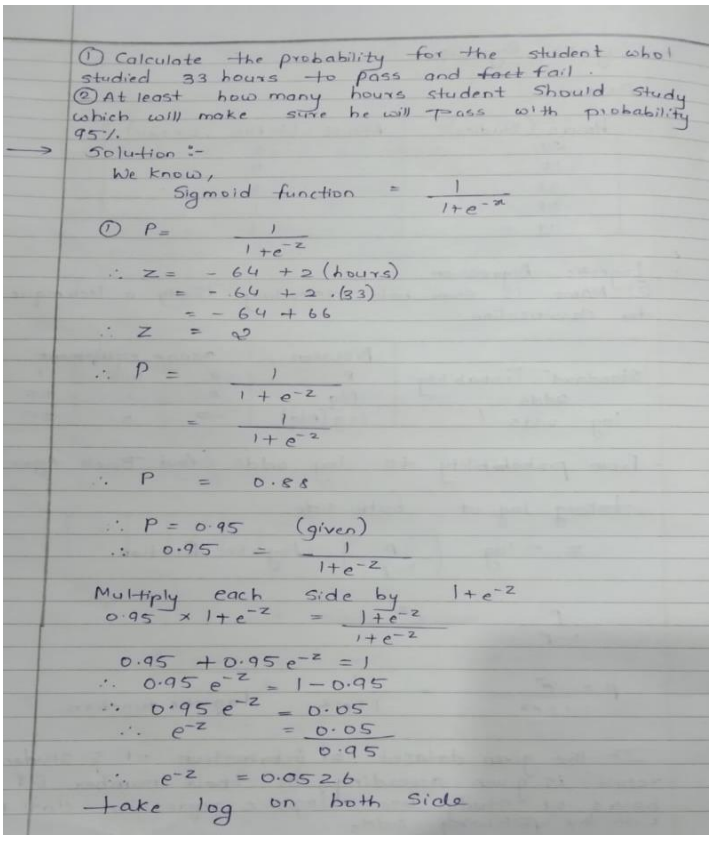
Software Required: Jupyter Notebook

Theory:

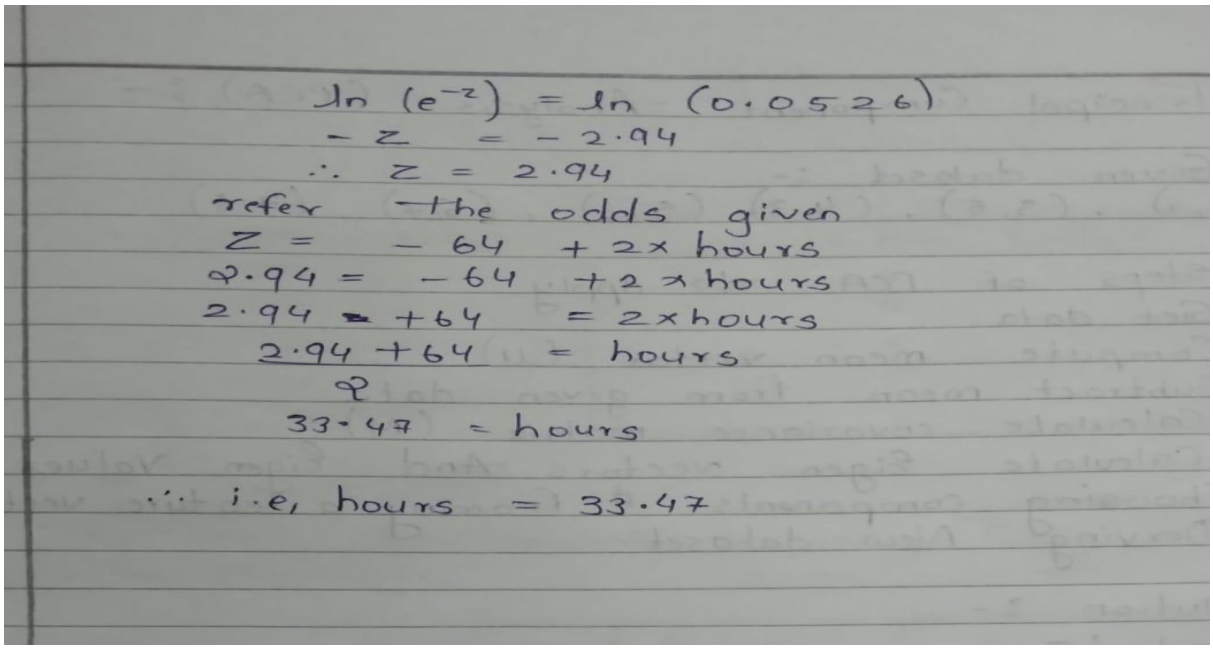
- **Linear Regression:** Linear regression strives to show the relationship between two variables by applying a linear equation to observed data. One variable is supposed tobe an independent variable, and the other is to be a dependent variable.
- **Logistic Regression:** Logistic regression is a process 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 thedata 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.

Program and output





Name: Abhishek Yadav **Roll No. 65**



Conclusion:

1. In Linear regression, if the coefficient of x is positive, then we can conclude that the relationship between the independent and the dependent variables is positive. Here, if the value of x increases, the value of y also increases.
2. Logistic regression predicts the probability of an event or class that is dependent on other factors. Thus the output of logistic regression always lies between 0 and 1. Because of this property it is commonly used for classification purpose.
3. The KNN algorithm is one of the simplest classification algorithms. Even with such simplicity, it can give highly competitive results. KNN algorithm can also be used for regression problems.

MCAL21: Artificial Intelligence And Machine Learning

Practical No. 7

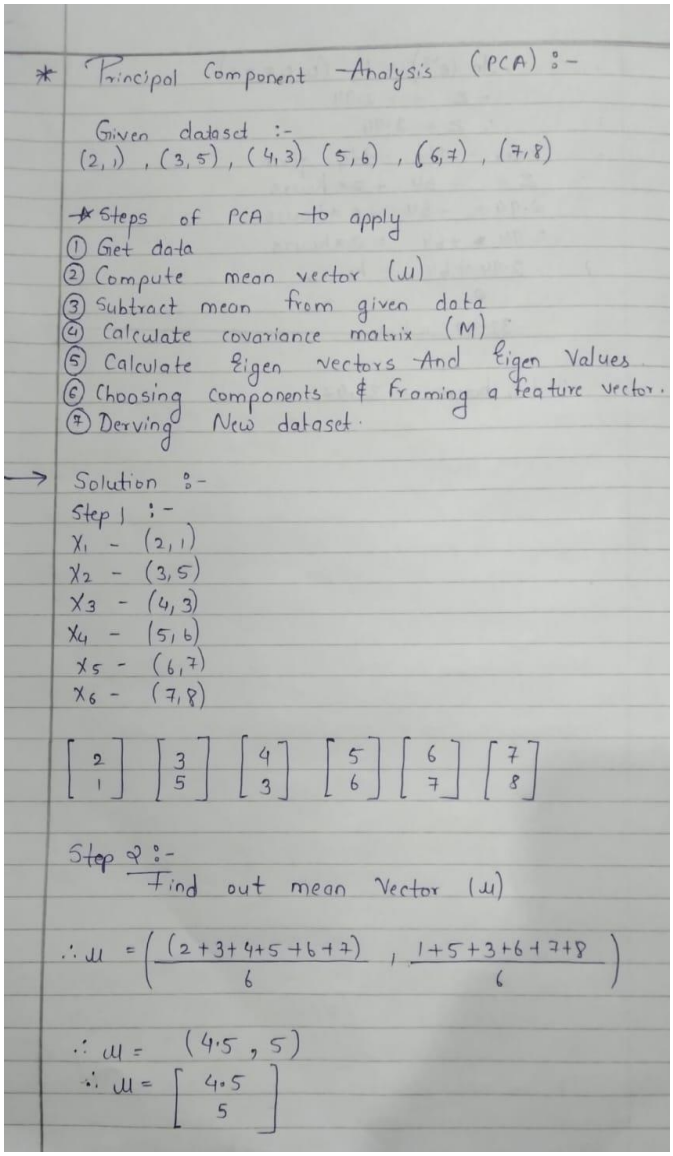
Aim: Write a program to demonstrate dimensionality reduction techniques (features selection, feature extraction) using PCA

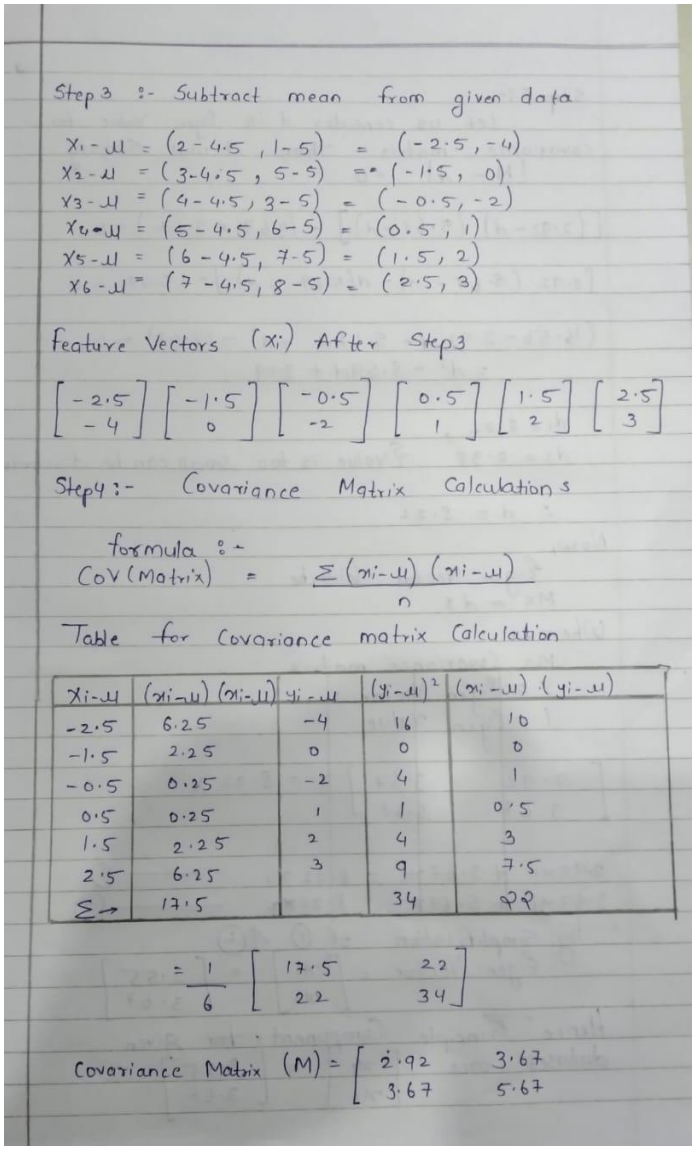
Objectives: To learn dimensionality reduction techniques using PCA.

Software Required: Jupyter Notebook

Theory definition of algorithm and sum:

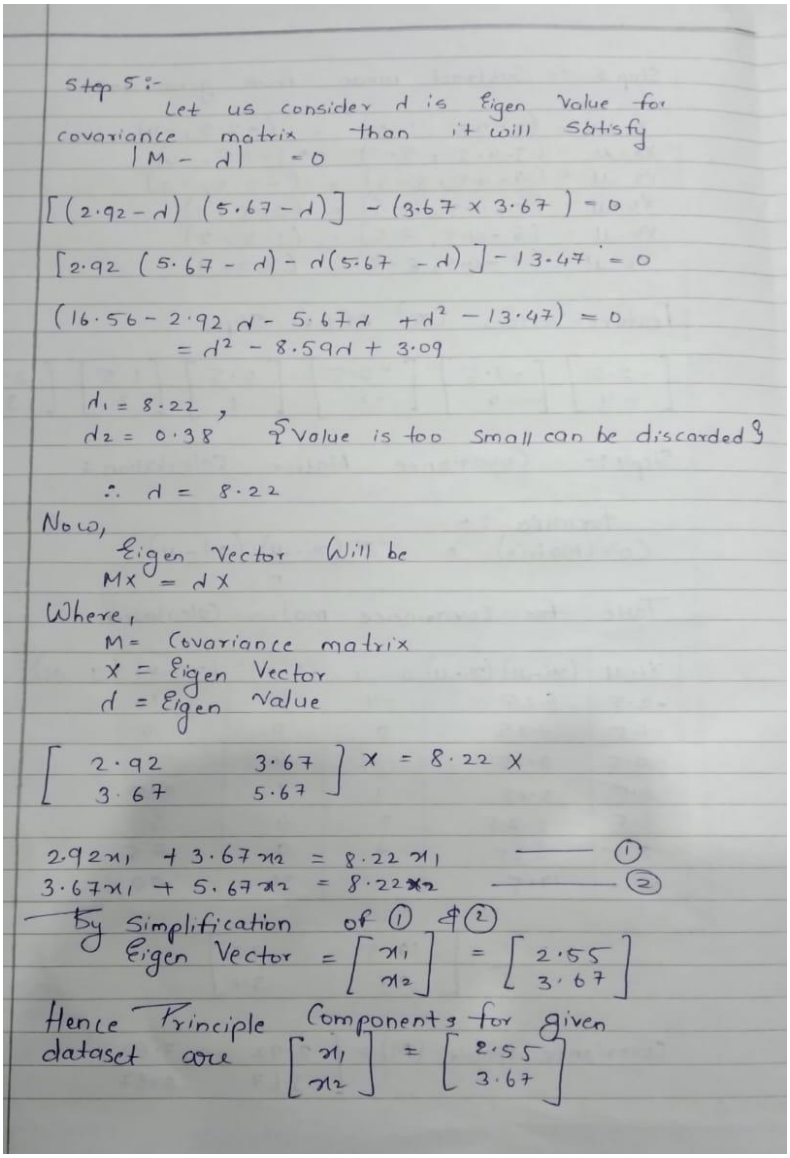
- **PCA: Principal component analysis** (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest.





xi-μ	(xi-μ)(xi-μ)	yi-μ	(yi-μ)²	(xi-μ)·(yi-μ)
-2.5	6.25	-4	16	10
-1.5	2.25	0	0	0
-0.5	0.25	-2	4	1
0.5	0.25	1	1	0.5
1.5	2.25	2	4	3
2.5	6.25	3	9	7.5
Σ →	17.5		34	22

Name: Abhishek Yadav **Roll No. 65**



Step 5:-
Let us consider d is Eigen Value for covariance matrix than it will satisfy
$|M - dI| = 0$

$[(2.92 - d)(5.67 - d)] - (3.67 \times 3.67) = 0$

$[2.92(5.67 - d) - d(5.67 - d)] - 13.47 = 0$

$(16.56 - 2.92d - 5.67d + d^2 - 13.47) = 0$
$= d^2 - 8.59d + 3.09$

$d_1 = 8.22$,
$d_2 = 0.38$ {Value is too small can be discarded}

$\therefore d = 8.22$

Now,
Eigen Vector Will be
$MX = dX$

Where,
M = Covariance matrix
X = Eigen Vector
d = Eigen Value

$$\begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix} X = 8.22\, X$$

$2.92x_1 + 3.67x_2 = 8.22x_1$ ——— (1)
$3.67x_1 + 5.67x_2 = 8.22x_2$ ——— (2)

By Simplification of (1) & (2)
Eigen Vector $= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$

Hence Principle Components for given dataset are $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$

Conclusion:

PCA identifies the intrinsic dimension of a dataset. In other words, it identifies the smallest number of features required to make an accurate prediction. A dataset may have a lot of features, but not all features are essential to the prediction. The features kept are the ones that have significant variance.

MCAL21: Artificial Intelligence And Machine Learning

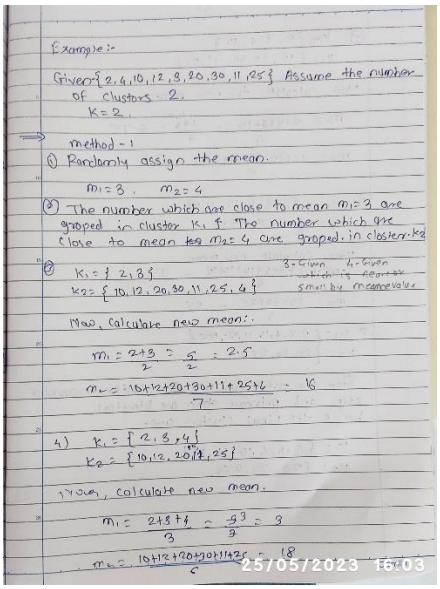
Practical No. 8

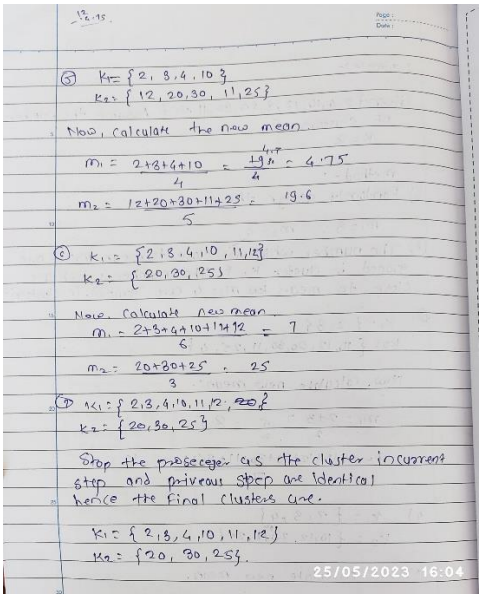
Aim: Write a program to implement classification of iris dataset using K-Means clustering algorithm with K=2.

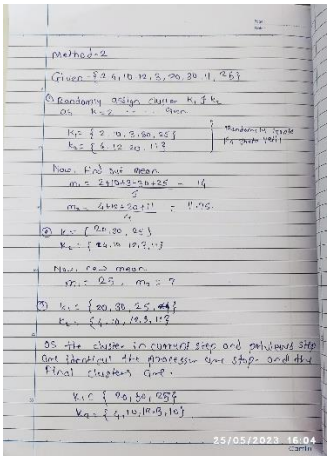
Objective: Understanding K-Means clustering algorithm.

Software Requirement: Jupyter Notebook

Theory: • K-Means Clustering: k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.







Conclusion: Even after multiple iterations, if the obtained centroids are same for all the clusters, it can be concluded that the algorithm is not learning any new pattern and gives a sign to stop its execution/training to a dataset.

Name: Abhishek Yadav **Roll No. 65**

MCAL21: Artificial Intelligence And Machine Learning

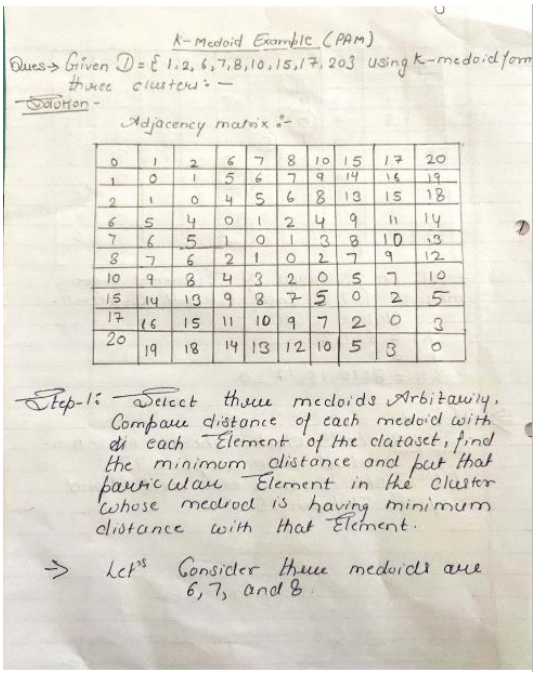
B) Aim: Write a program to demonstrate K-Medoid Algorithm.

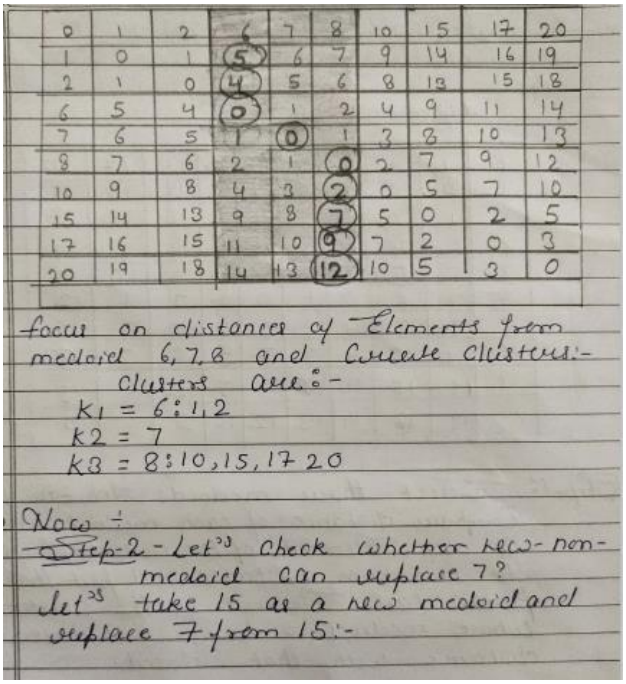
Objective: Understanding K-medoid clustering algorithm.

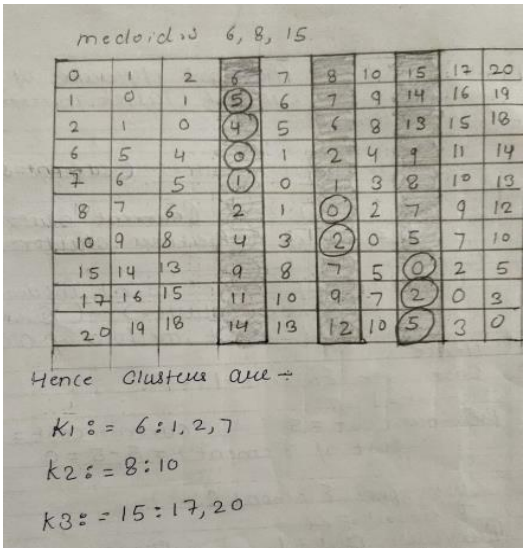
Software Requirement: Jupyter Notebook

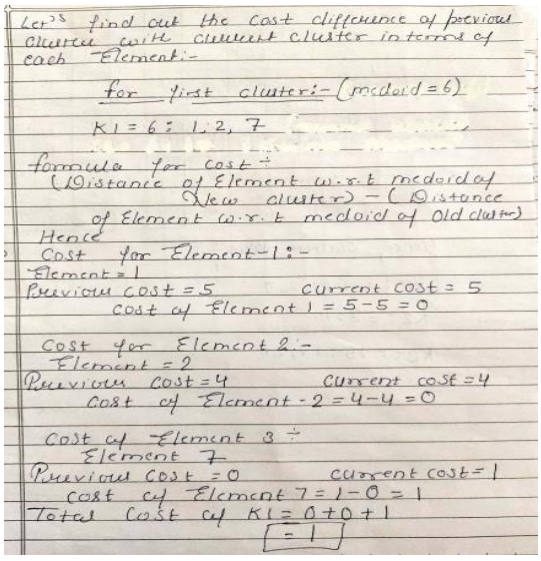
Theory:

- **K-Medoid:** k -medoids is a classical partitioning technique of clustering that splits the data set of n objects into k clusters, where the number k of clusters assumed known a priori (which implies that the programmer must specify k before the execution of a k -medoids algorithm).

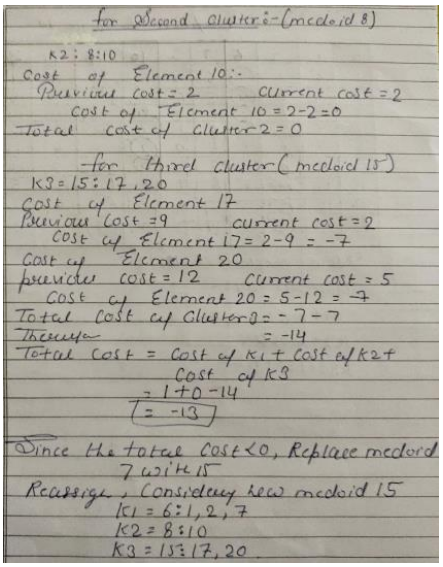








Name: Abhishek Yadav **Roll No. 65**



for Second cluster:-(medoid 8)

K2: 8:10

Cost of Element 10:-

Previous cost = 2 Current cost = 2

Cost of Element 10 = 2-2 = 0

Total cost of cluster 2 = 0

for third cluster (medoid 15)

K3 = 15: 17, 20

Cost of Element 17

Previous cost = 9 current cost = 2

Cost of Element 17 = 2-9 = -7

Cost of Element 20

previous cost = 12 Current cost = 5

Cost of Element 20 = 5-12 = -7

Total Cost of cluster 3 = -7-7

Therefore = -14

Total cost = Cost of K1 + Cost of K2 + Cost of K3

= 1+0-14

= -13

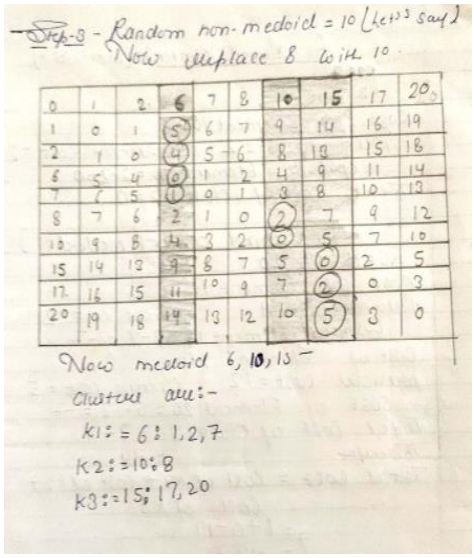
Since the total cost < 0, Replace medoid 7 with 15

Reassign, Considering new medoid 15

K1 = 6: 1, 2, 7

K2 = 8: 10

K3 = 15: 17, 20



Step-3 - Random non-medoid = 10 (Let's say)

Now replace 8 with 10

0	1	2	6	7	8	10	15	17	20
1	0	1	5	6	7	9	14	16	19
2	1	0	4	5	6	8	13	15	18
6	5	4	0	1	2	4	9	11	14
7	6	5	1	0	1	3	8	10	13
8	7	6	2	1	0	2	7	9	12
10	9	8	4	3	2	0	5	7	10
15	14	13	9	8	7	5	0	2	5
17	16	15	11	10	9	7	2	0	3
20	19	18	14	13	12	10	5	3	0

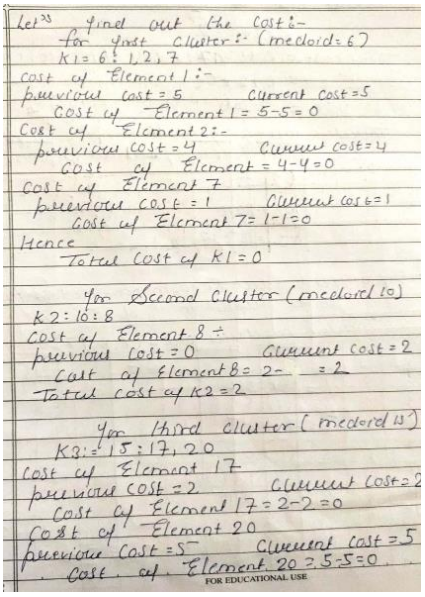
Now medoid 6, 10, 15 -

Clusters are:-

K1: = 6: 1, 2, 7

K2: = 10: 8

K3: = 15: 17, 20



Let's find out the cost:-

for first cluster: (medoid: 6)

K1 = 6: 1, 2, 7

cost of Element 1:-

previous cost = 5 Current cost = 5

Cost of Element 1 = 5-5 = 0

Cost of Element 2:-

previous cost = 4 Current cost = 4

Cost of Element = 4-4 = 0

Cost of Element 7

previous cost = 1 Current cost = 1

Cost of Element 7 = 1-1 = 0

Hence

Total Cost of K1 = 0

for Second cluster (medoid 10)

K2: 10: 8

Cost of Element 8 :-

previous cost = 0 Current cost = 2

Cost of Element 8 = 2- = 2

Total cost of K2 = 2

for third cluster (medoid 15)

K3: = 15: 17, 20

Cost of Element 17

previous cost = 2 Current cost = 2

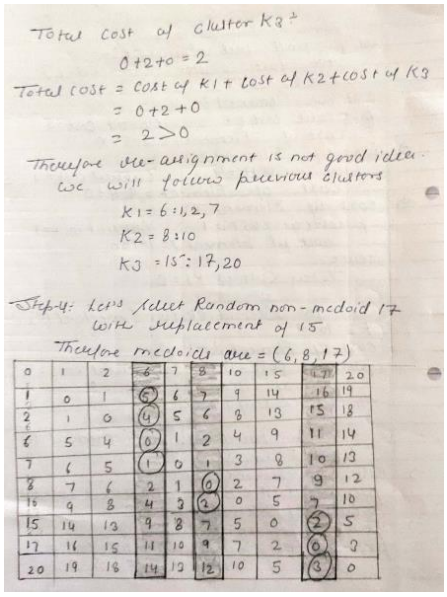
Cost of Element 17 = 2-2 = 0

Cost of Element 20

previous cost = 5 Current cost = 5

Cost of Element 20 = 5-5 = 0

FOR EDUCATIONAL USE



Total cost of cluster K3 =

0+2+0 = 2

Total cost = Cost of K1 + cost of K2 + cost of K3

= 0+2+0

= 2 > 0

Therefore re-assignment is not good idea. We will follow previous clusters

K1 = 6: 1, 2, 7

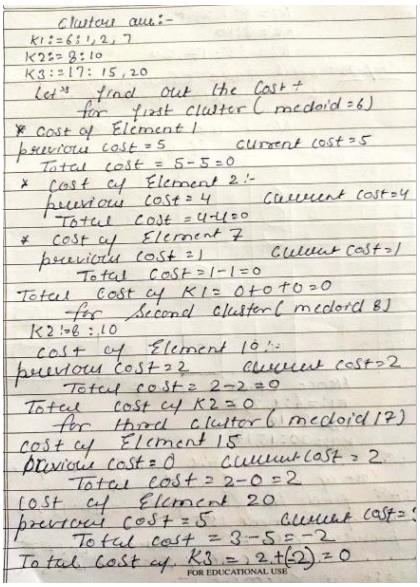
K2 = 8: 10

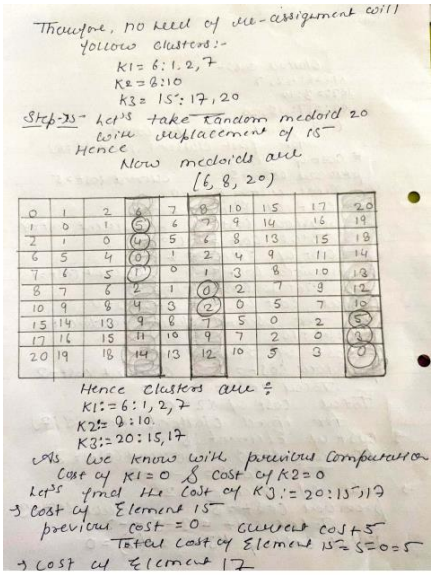
K3 = 15: 17, 20

Step-4: Let's select Random non-medoid 17 with replacement of 15

Therefore medoids are = (6, 8, 17)

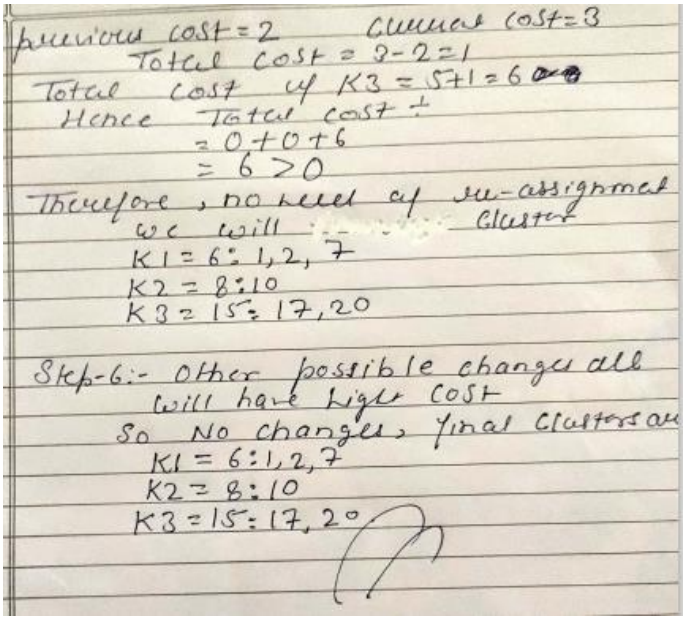
0	1	2	6	7	8	10	15	17	20
1	0	1	5	6	7	9	14	16	19
2	1	0	4	5	6	8	13	15	18
6	5	4	0	1	2	4	9	11	14
7	6	5	1	0	1	3	8	10	13
8	7	6	2	1	0	2	7	9	12
10	9	8	4	3	2	0	5	7	10
15	14	13	9	8	7	5	0	2	5
17	16	15	11	10	9	7	2	0	3
20	19	18	14	13	12	10	5	3	0


Clusters are:-
K1:=6:1,2,7
K2:=8:10
K3:=17: 15,20
Let's find out the Cost +
for first cluster (medoid =6)
* Cost of Element 1
previous cost = 5 current cost = 5
Total cost = 5-5=0
* Cost of Element 2:-
previous cost = 4 Current cost=4
Total cost = 4-4=0
* Cost of Element 7
previous cost = 1 Current cost=1
Total cost = 1-1=0
Total Cost of K1 = 0+0+0=0
for Second cluster (medoid 8)
K2:=8:10
cost of Element 10:-
previous cost = 2 Current cost=2
Total cost = 2-2=0
Total cost of K2 = 0
for Third cluster (medoid 17)
cost of Element 15
previous cost = 0 Current cost = 2
Total cost = 2-0=2
cost of Element 20
previous cost = 5 Current cost=
Total cost = 3-5=-2
Total Cost of K3 = 2+(-2) = 0
FOR EDUCATIONAL USE


Therefore, no need of re-assignment will follow clusters:-
K1 = 6:1,2,7
K2 = 8:10
K3 = 15:17,20
Step-7:- Let's take Random medoid 20 with replacement of 15
Hence
Now medoids are
(6, 8, 20)

0	1	2	(6)	7	8	10	15	17	20
1	0	1	(5)	6	7	9	14	16	19
2	1	0	(4)	5	6	8	13	15	18
6	5	4	(0)	1	2	4	9	11	14
7	6	5	(1)	0	1	3	8	10	13
8	7	6	2	1	(0)	2	7	9	12
10	9	8	4	3	(2)	0	5	7	10
15	14	13	9	8	7	5	0	2	(5)
17	16	15	11	10	9	7	2	0	(3)
20	19	18	14	13	12	10	5	3	(0)

Hence clusters are :-
K1:=6:1,2,7
K2:= 8:10
K3:=20:15,17
As we know with previous comparison
Cost of K1 = 0 & Cost of K2 = 0
Let's find the Cost of K3:=20:15,17
→ Cost of Element 15
previous cost = 0 Current cost = 5
Total cost of Element 15 = 5-0=5
→ Cost of Element 17


previous cost = 2 Current cost = 3
Total cost = 3-2=1
Total cost of K3 = 5+1=6
Hence Total cost :-
= 0+0+6
= 6 > 0
Therefore, no need of re-assignment
we will Cluster
K1 = 6:1,2,7
K2 = 8:10
K3 = 15:17,20

Step-6:- Other possible changes all will have High cost
So No changes, final clusters are
K1 = 6:1,2,7
K2 = 8:10
K3 = 15:17,20

Conclusion: the algorithm is not learning any new pattern and gives a sign to stop its execution/training to a dataset.

MCAL21: Artificial Intelligence And Machine Learning

Practical No. 9

A) Aim: Implementation of Classifying data using support vector machine (SVMs)

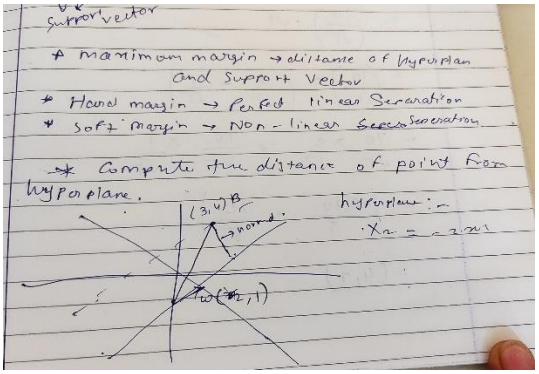
Objective: Understand of Following Terminologies

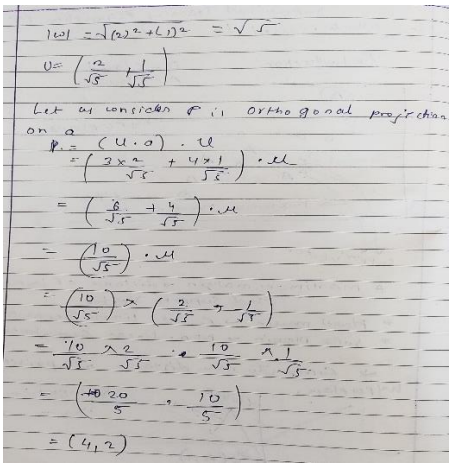
1. SVM Concept
2. Support Vector
3. Hyperplane
4. Soft Margin & Hard Margin
5. Computation of Distance of an arbitrary Point From Hyperplane.

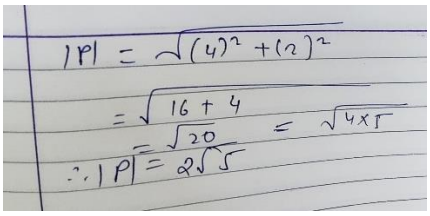
Software Requirement: Jupyter Notebook

Theory:

- **Linear SVM:** Linear SVM is **used for linearly separable data**, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.







Conclusion: SVM is a supervised learning algorithm which separates the data into different classes through the use of a hyperplane.

Name: Abhishek Yadav **Roll No. 65**

MCAL21: Artificial Intelligence And Machine Learning

B) Aim: Implementation of support vector machine (SVMs) Algorithm For Non Liner Case

Objective: Understand of Following Terminologies

1. Soft Margin
2. Cornel Trick
3. Non Liner Classification

Software Requirement: Jupyter Notebook

Theory:

- **Non-Linear SVM:** Nonlinear classification: SVM can be extended to solve nonlinear classification tasks when the set of samples cannot be separated linearly. By applying kernel functions, thesamples are mapped onto a high-dimensional feature space, in which thelinear classification is possible.

Conclusion: There are two types of SVM: linear and non-linear, they are used depending on the type of data. Non-linear SVM uses the Radial Basis Function Kernel that takes the data points to a higher dimension so that they are linearly separable in that dimension, and then the algorithm classifies them.

Name: Abhishek Yadav **Roll No. 65**

Practical No. 10

A) Aim: Implementation Decision Tree with its Accuracy

Objective: Understand The concept of decision tree

Understanding algorithms of (ID3, c4.5, CART)

Software Requirement: Jupyter Notebook

Theory:

1) Decision Tree:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.

Conclusion: Decision trees are powerful machine learning algorithms that can be used for both classification and regression tasks. They are popular due to their interpretability, ease of implementation, and ability to handle both numerical and categorical data. Here are some key conclusions about decision trees:

B) Aim: Implementation of Random Forrest Algorithm for Classification and regression

Objective: Understand The Following terminologies

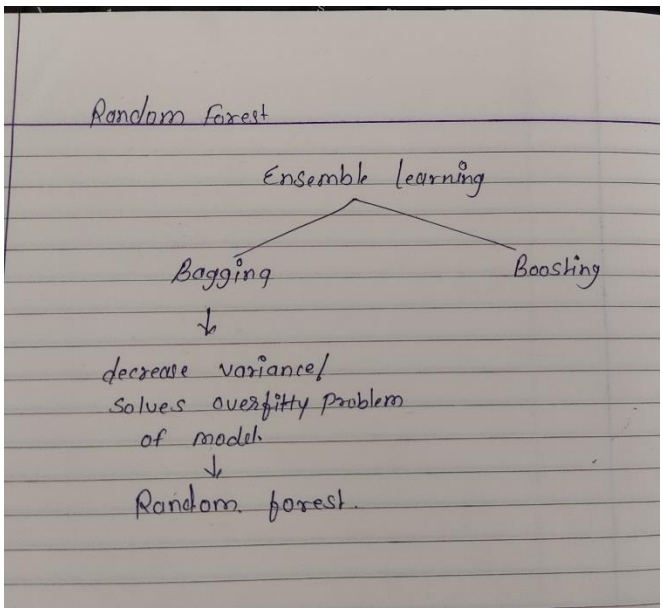
Ensemble learning, Bagging, Stump, Forest of tree

Software Requirement: Jupyter Notebook

Theory:

1) Random Forest:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML.



Conclusion: Random Forest is a powerful ensemble learning algorithm that combines the predictions of multiple decision trees to make more accurate predictions. It is widely used for both classification and regression tasks. Here are some conclusions about the Random Forest algorithm:

MCAL21: Artificial Intelligence And Machine Learning

Practical No. 11

Aim: Implement Adaboost Algorithm under ensemble learning

Objective: 1. Understanding of boosting concept

2. Implementation of strong learners with the help of multiple wick learners
3. Implementation of strong learners by focusing by Lerner

Software Requirement: Jupyter Notebook

Theory: AdaBoost algorithm, short for Adaptive Boosting, is a Boosting technique used as an Ensemble Method in Machine Learning. It is called Adaptive Boosting as the weights are re-assigned to each instance, with higher weights assigned to incorrectly classified instances.

Conclusion: AdaBoost helps in choosing the training set for each new classifier that is trained based on the results of the previous classifier. Also, while combining the results, it determines how much weight should be given to each classifier's proposed answer.

Name: Abhishek Yadav **Roll No. 65**

Practical No. 12

A) Aim: Deployment of Machine Learning Model on spyder

Objective: 1. Deploy A Machine Learning Model Which Can Predict salary of new employee on the basis of experience, test score, interview score.

2. Deploye a machine learning model which can predict a type of flower on the basis of sepal length, sepal with, Patel length, Patel with,

Software Requirement: spider

Code:

app.py

```
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict',methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''
    int_features = [int(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    prediction = model.predict(final_features)
    output = round(prediction[0], 2)

    return render_template('index.html', prediction_text='Employee Salary should be $ {}'.format(output))

@app.route('/predict_api',methods=['POST'])
def predict_api():
    '''
    For direct API calls trought request
    '''
    data = request.get_json(force=True)
    prediction = model.predict([np.array(list(data.values()))])

    output = prediction[0]
    return jsonify(output)

if __name__ == "__main__":
    app.run(debug=True)
```

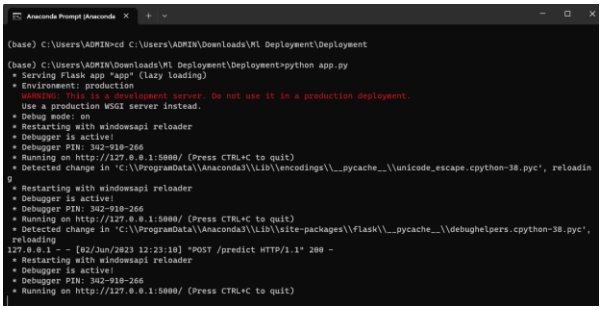
model.py

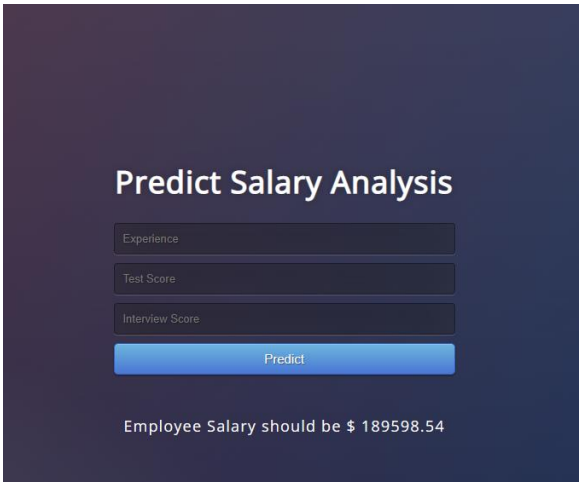
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle

dataset = pd.read_csv('hiring.csv')

dataset['experience'].fillna(0, inplace=True)

dataset['test_score'].fillna(dataset['test_score'].mean(), inplace=True)

X = dataset.iloc[:, :3]

#Converting words to integer values
def convert_to_int(word):
    word_dict = {'one':1, 'two':2, 'three':3, 'four':4, 'five':5, 'six':6, 'seven':7, 'eight':8,
                 'nine':9, 'ten':10, 'eleven':11, 'twelve':12, 'zero':0, 0: 0}
    return word_dict[word]

X['experience'] = X['experience'].apply(lambda x : convert_to_int(x))

y = dataset.iloc[:, -1]

#Splitting Training and Test Set
#Since we have a very small dataset, we will train our model with all availabe data.

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()

#Fitting model with trainig data
regressor.fit(X, y)

# Saving model to disk
pickle.dump(regressor, open('model.pkl','wb'))

# Loading model to compare the results
model = pickle.load(open('model.pkl','rb'))
print(model.predict([[2, 9, 6]]))
```

request.py

```
import requests

url = 'http://localhost:5000/predict_api'
r = requests.post(url,json={'experience':2, 'test_score':9, 'interview_score':6})

print(r.json())
```

OutPut:





B)Aim: IR Project of Machine Learning Model on spyder

Code:

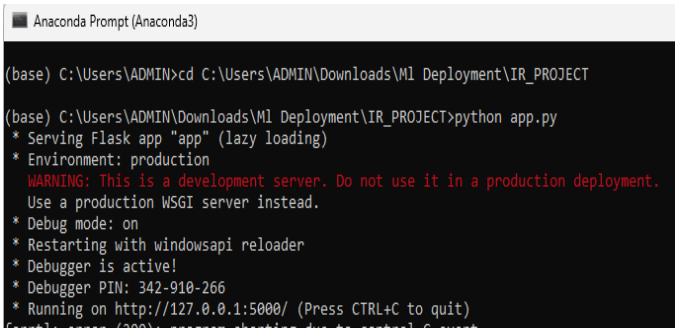
app.py

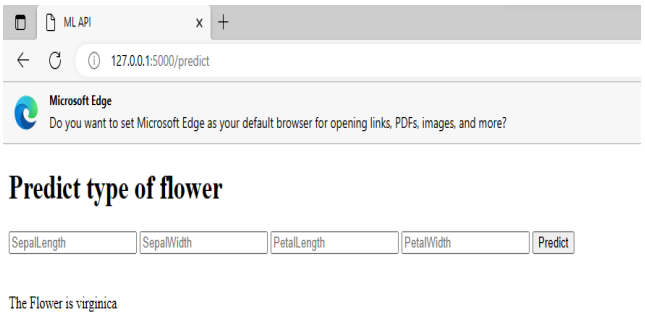
```
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle
model = pickle.load(open('model.pkl', 'rb'))
app = Flask(__name__)
@app.route('/')
def home():
    return render_template('index.html')
@app.route('/predict',methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''
    int_features = [float(x) for x in request.form.values()]

    final_features = [np.array(int_features)]
    prediction = model.predict(final_features)

    output =prediction[0]

    return render_template('index.html', prediction_text='The Flower is {}'.format(output))
if __name__ == "__main__":
    app.run(debug=True)
```

model.py

```
import pandas as pd

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

import pickle

data=pd.read_csv('iris.csv')

# X = feature values, all the columns except the last column

X = data.iloc[:, :-1]

# y = target values, last column of the data frame

y = data.iloc[:, -1]

#Split the data into 80% training and 20% testing

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Train the model

model = LogisticRegression()

model.fit(x_train, y_train) #Training the model

#Test the model

predictions = model.predict(x_test)

print( classification_report(y_test, predictions) )

print( accuracy_score(y_test, predictions))

pickle.dump(model,open('model.pkl','wb'))

p=model.predict([[5.1,3.5,1.4,0.2]])

print(p[0])
```

OutPut:





creation of numpy Array

In [4]:
```
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.]])
```

initialization of numpy array initialization with same number

In [6]:
```
n1=np.zeros((1,2))
n1
```

Out[6]:
```
array([[0., 0.]])
```

In [7]:
```
n1=np.zeros((5,5))
n1
```

Out[7]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

Intialization Numpy array with same number

In [8]:
```
n1=np.full((2,2),10)
n1
```

Out[8]:
```
array([[10, 10],
       [10, 10]])
```

Initilization numpy with range

In [9]:
```
n1=np.arange(10,20)
n1
```

Out[9]:
```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In [10]:
```
n1=np.arange(10,50,5)
n1
```

Out[10]:
```
array([10, 15, 20, 25, 30, 35, 40, 45])
```

Initialization numpy with random number

In [11]:
```
n1=np.random.randint(1,100,5)
n1
```

Out[11]:
```
array([37, 13,  7,  2, 36])
```

In [12]:
```
n1=np.random.randint(1,100,10)
n1
```

Out[12]:
```
array([58, 16, 87, 16, 87, 95, 76, 46, 45, 25])
```

Numpy Shape

In [13]:
```
n1=np.array([[1,2,3],[4,5,6]])
n1.shape
```

Out[13]:
```
(2, 3)
```

In [14]:
```
n1.shape=(3,2)
n1.shape
```

Out[14]:
```
(3, 2)
```

change shape of array

In [15]:
```
n1=np.array([[1,2,3,4],[4,3,2,1]])
n1
```

Out[15]:
```
array([[1, 2, 3, 4],
       [4, 3, 2, 1]])
```

In [16]:
```
n1.shape=(4,2)
n1
```

Out[16]:
```
array([[1, 2],
       [3, 4],
       [4, 3],
       [2, 1]])
```

joining Numpy Arrays vstack hstack colounm_satck()

In [18]:
```
n1=np.array([10,20,30])
n2=np.array([40,50,60])
np.vstack((n1,n2))
```

Out[18]:
```
array([[10, 20, 30],
       [40, 50, 60]])
```

In [19]:
```
n1=np.array([10,20,30])
n2=np.array([40,50,60])
np.hstack((n1,n2))
```

Out[19]:
```
array([10, 20, 30, 40, 50, 60])
```

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

Out[20]:
```
array([[10, 40],
       [20, 50],
       [30, 60]])
```

Numpy intersection and differences

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])
n2=np.array([50,60,70,80,90])
np.setdiff1d(n2,n1)
```

Out[26]:
```
array([70, 80, 90])
```

Numpy Array Mathematics

In [3]:
```
import numpy as np
```

In [5]:
```
n1=np.array([10,20])
n2=np.array([30,40])
np.sum([n1,n2])
```

Out[5]:
```
100
```

In [6]:
```
np.sum([n1,n2],axis=0)
```

Out[6]:
```
array([40, 60])
```

In [7]:
```
np.sum([n1,n2],axis=1)
```

Out[7]:
```
array([30, 70])
```

Basic Addition

In [8]:
```
n1=np.array([10,20,30])
n1=n1+1
n1
```

Out[8]:
```
array([11, 21, 31])
```

Basic Subtraction

In [9]:
```
n1=np.array([10,20,30])
n1=n1-1
n1
```

Out[9]:
```
array([ 9, 19, 29])
```

Basic Multiplication

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

Out[10]:
```
array([20, 40, 60])
```

Basic Divison

In [11]:
```
n1=np.array([10,20,30])
n1=n1/10
n1
```

Out[11]:
```
array([1., 2., 3.])
```

In [ ]:
```

```

#Matplotlib is a python Library used for data visualization

LINE PLOT

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
```

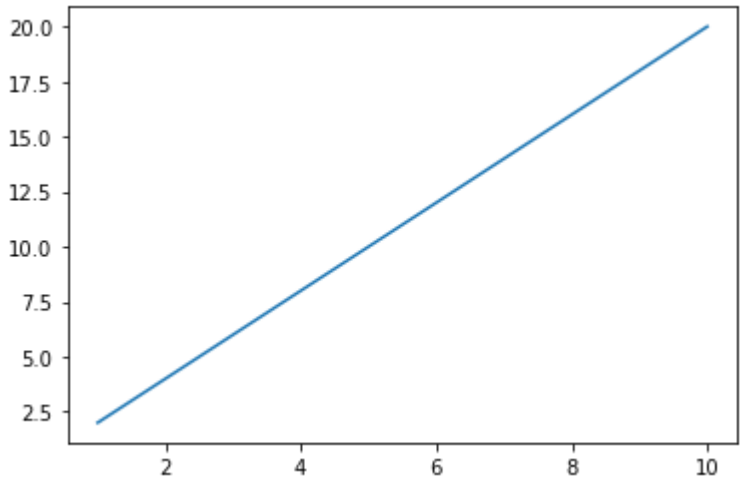
In [2]:
```
x=np.arange(1,11)
x
```

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

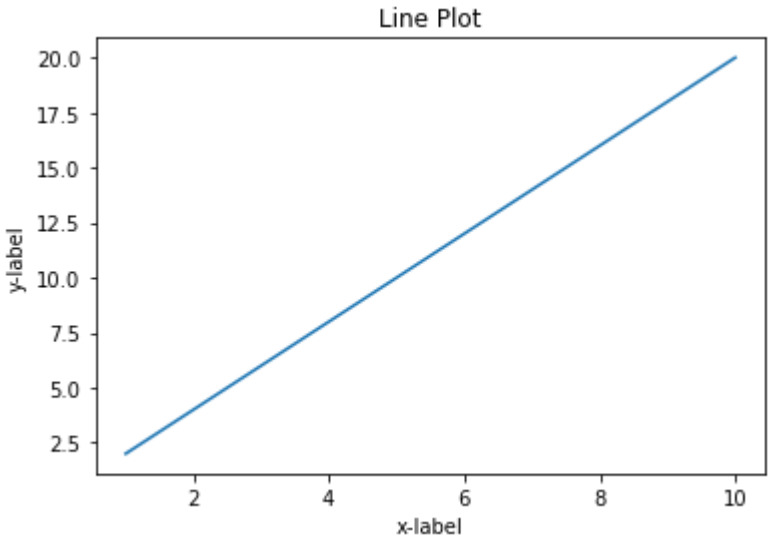
In [3]:
```
y=2*x
y
```

Out[3]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [4]:
```
plt.plot(x,y)
plt.show()
```

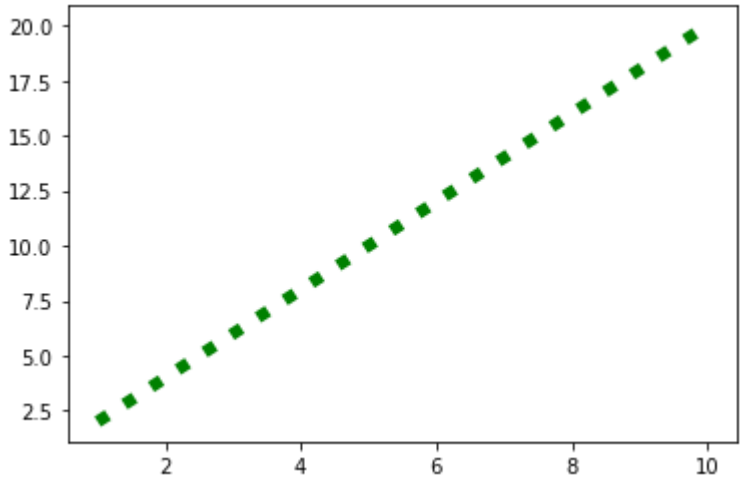


In [5]:
```
plt.plot(x,y)
plt.title("Line Plot")
plt.xlabel("x-label")
plt.ylabel("y-label")
plt.show()
```



Changing line aesthetics

In [8]:
```
plt.plot(x,y,color='g',linestyle=':',linewidth=6)
plt.show()
```

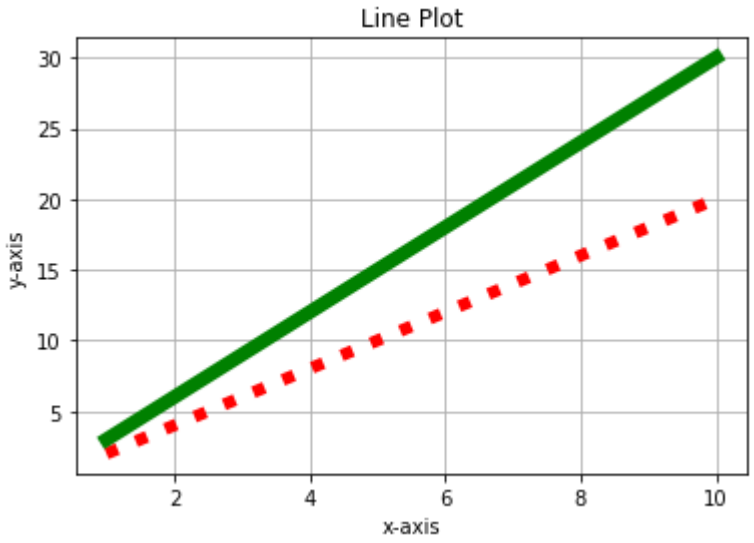


Addition of two lines in same plot

In [9]:
```
x=np.arange(1,11)
y1=2*x
y2=3*x
```

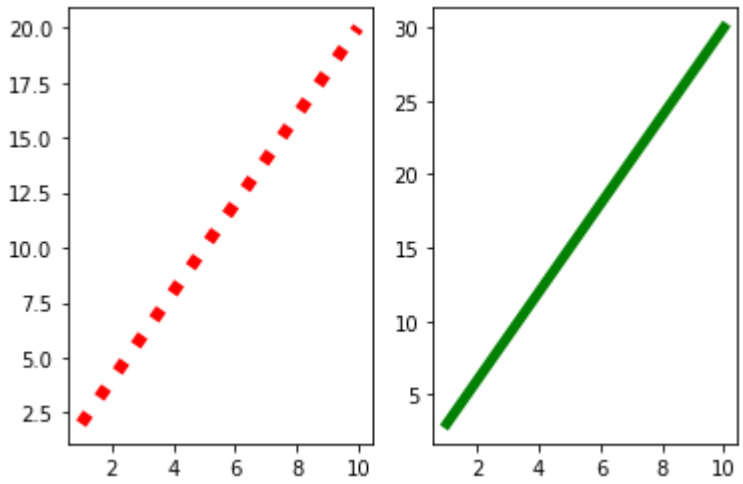
plt.plot(x,y1,color='r',linestyle=':',linewidth=6) plt.plot(x,y2,color='g',linestyle='-',linewidth=7) plt.title("Line Plot") plt.xlabel("x-axis") plt.ylabel("y-axis") plt.grid(True) plt.show()

In [13]:
```
plt.plot(x,y1,color='r',linestyle=':',linewidth=6)
plt.plot(x,y2,color='g',linestyle='-',linewidth=7)
plt.title("Line Plot")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.grid(True)
plt.show()
```



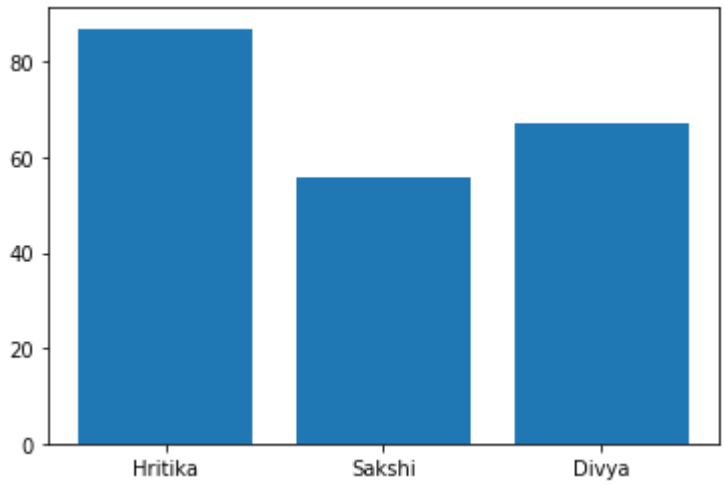
Adding sub-plots

In [16]:
```
x=np.arange(1,11)
y1=2*x
y2=3*x

plt.subplot(1,2,1)
plt.plot(x,y1,color='r',linestyle=':',linewidth=6)

plt.subplot(1,2,2)
plt.plot(x,y2,color='g',linestyle='-',linewidth=5)

plt.show()
```



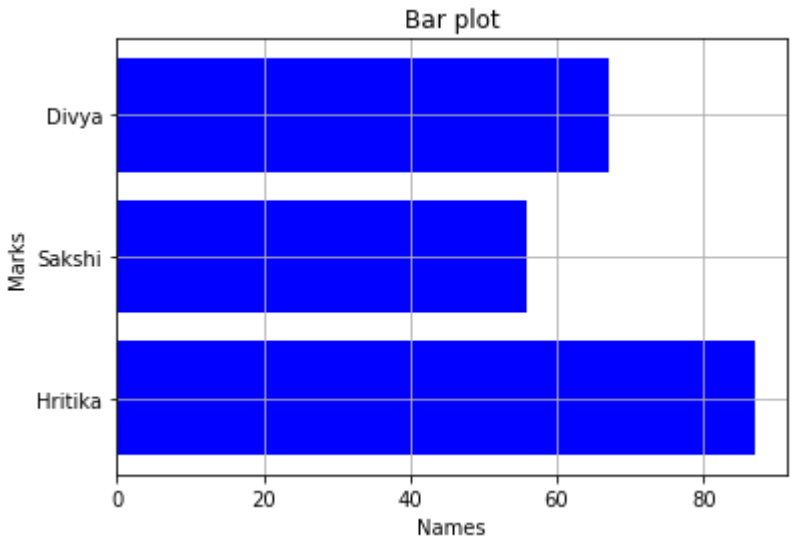
Bar plot

In [18]:
```
student={"Hritika":87,"Sakshi":56,"Divya":67}

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

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



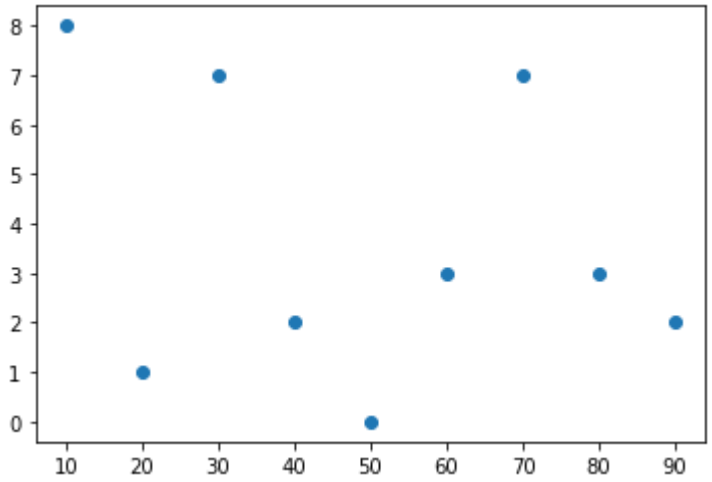
Horizontal bar plot

In [19]:
```
plt.barh(names,values,color='b')
plt.title("Bar plot")
plt.xlabel("Names")
plt.ylabel("Marks")
plt.grid(True)
plt.show()
```



Scatter Plot

In [23]:
```
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()
```



In [ ]:

In [18]:
```
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 [19]:
```
# 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[19]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cal
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N
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	N
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	N

In [20]:
```
print('The number of samples into the train data is {}.'.format(train_df.shape[
```

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

In [21]:
```
test_df.head()
```

Out[21]:

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

In [22]:
```
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.
```

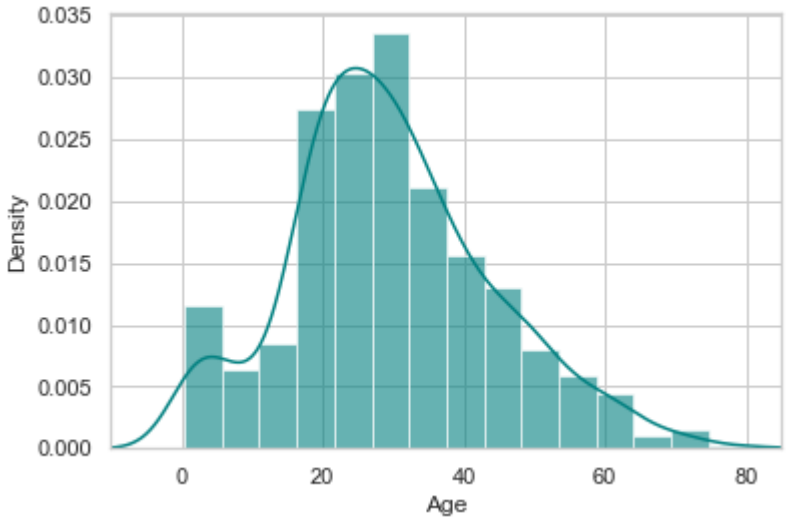
In [23]:
```
# check missing values in train data
train_df.isnull().sum()
```

Out[23]:
```
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 [24]:
```
# percent of missing "Age"
print('Percent of missing "Age" records is %.2f%%' %((train_df['Age'].isnull().
```

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

In [25]:
```
ax = train_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', al
train_df["Age"].plot(kind='density', color='teal')
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```



In [26]:
```
# 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 [27]:
```
# percent of missing "Cabin"
print('Percent of missing "Cabin" records is %.2f%%' %((train_df['Cabin'].isnul
```

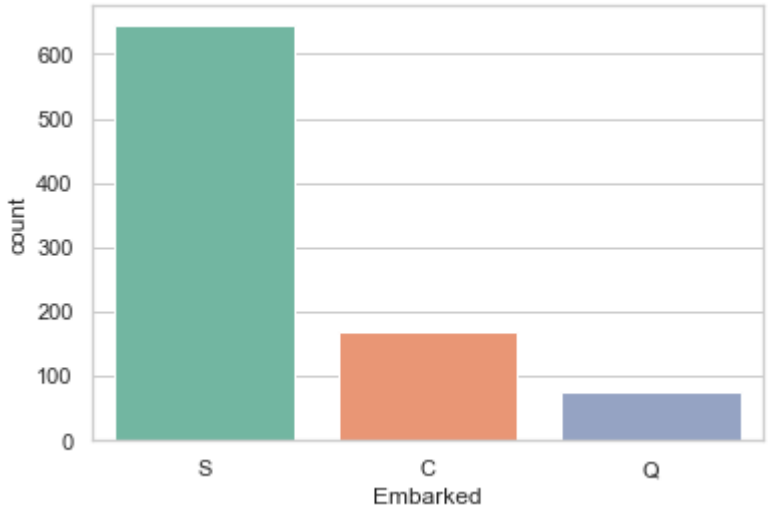
```
Percent of missing "Cabin" records is 77.10%
```

In [28]:
```
 #percent of missing "Embarked"
print('Percent of missing "Embarked" records is %.2f%%' %((train_df['Embarked']
```

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

In [29]:
```
print('Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Qu
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 = Queenst
own, S = Southampton):
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```



In [30]:
```
print('The most common boarding port of embarkation is %s.' %train_df['Embarkec
```

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

In [31]:
```
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(), inp
train_data.drop('Cabin', axis=1, inplace=True)
```

In [32]:
```
# check missing values in adjusted train data
train_data.isnull().sum()
```

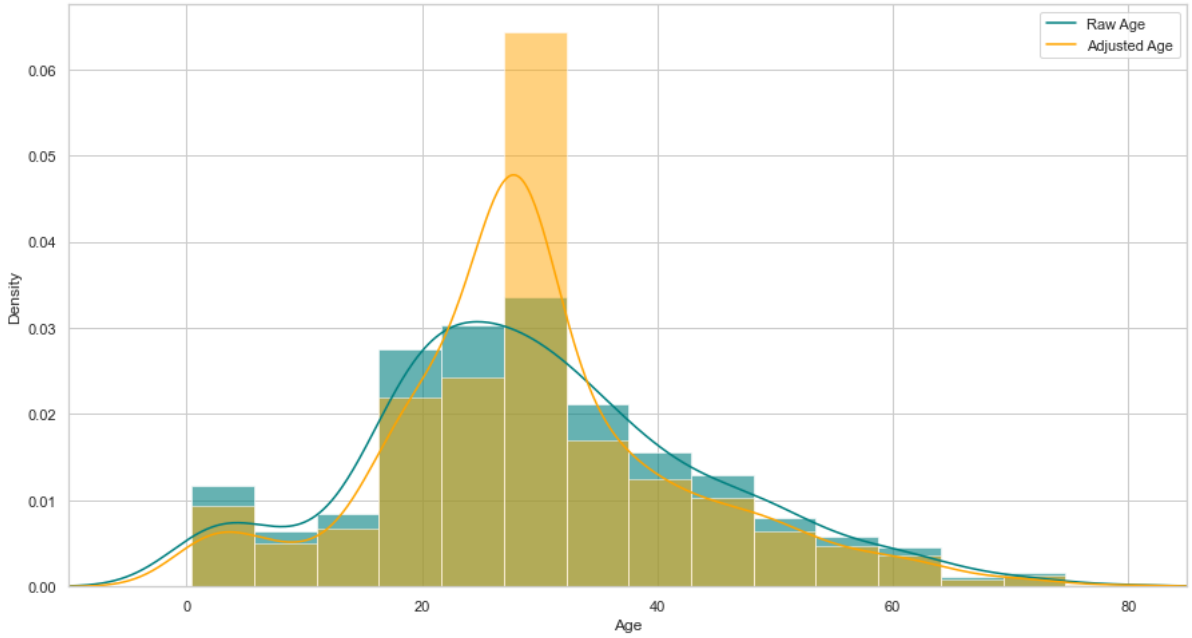
Out[32]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

In [33]:
```
# preview adjusted train data
train_data.head()
```

Out[33]:

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

In [34]:
```
plt.figure(figsize=(15,8))
ax = train_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', al
train_df["Age"].plot(kind='density', color='teal')
ax = train_data["Age"].hist(bins=15, density=True, stacked=True, color='orange'
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 [35]:
```
# Create categorical variable for traveling alone
train_data['TravelAlone']=np.where((train_data["SibSp"]+train_data["Parch"])>0,
train_data.drop('SibSp', axis=1, inplace=True)
train_data.drop('Parch', axis=1, inplace=True)
```

In [36]:
```
#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[36]:

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

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 [37]:
```
test_df.isnull().sum()
```

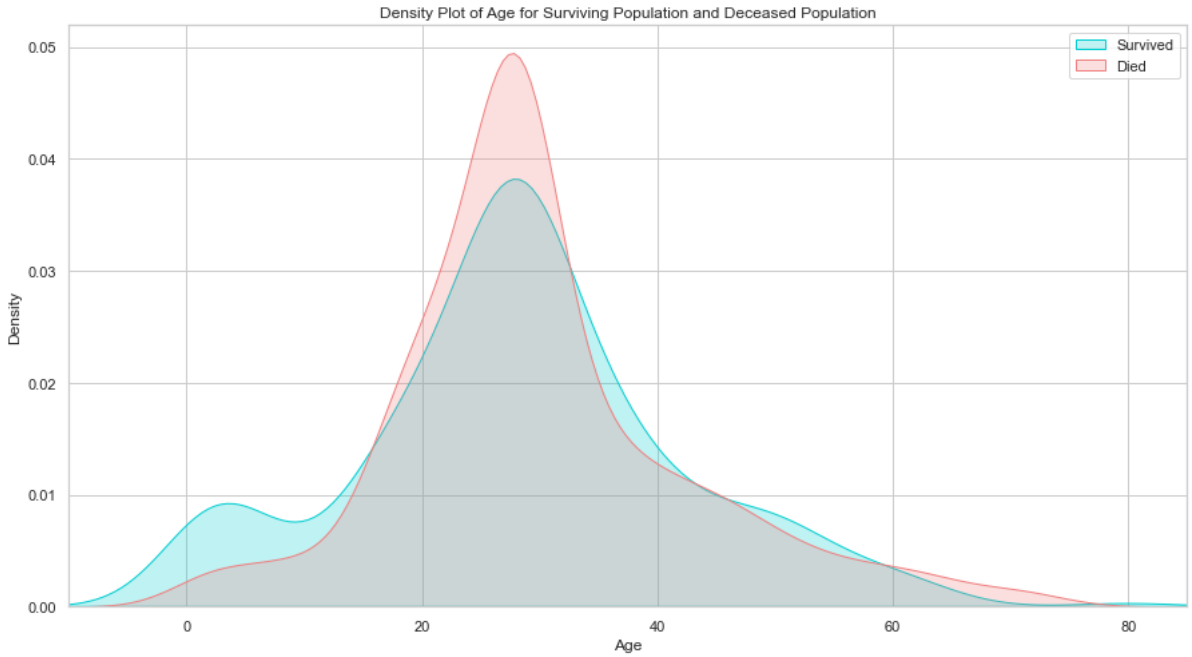
Out[37]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [38]:
```
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,

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()
```

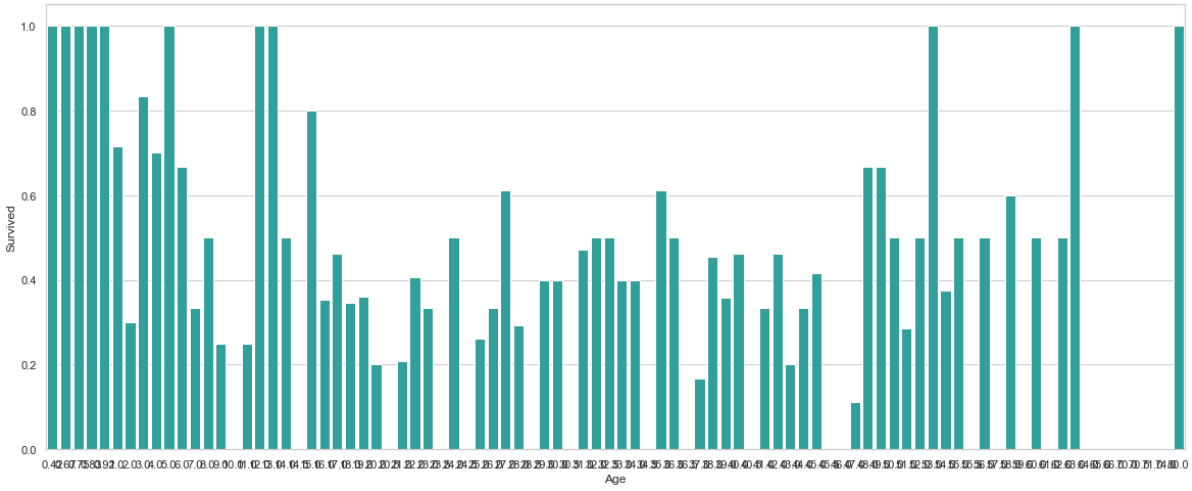
Out[38]:

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

In [39]:
```
plt.figure(figsize=(15,8))
ax = sns.kdeplot(final_train["Age"][final_train.Survived == 1], color="darkturq
sns.kdeplot(final_train["Age"][final_train.Survived == 0], color="lightcoral",
plt.legend(['Survived', 'Died'])
plt.title('Density Plot of Age for Surviving Population and Deceased Populatior
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```

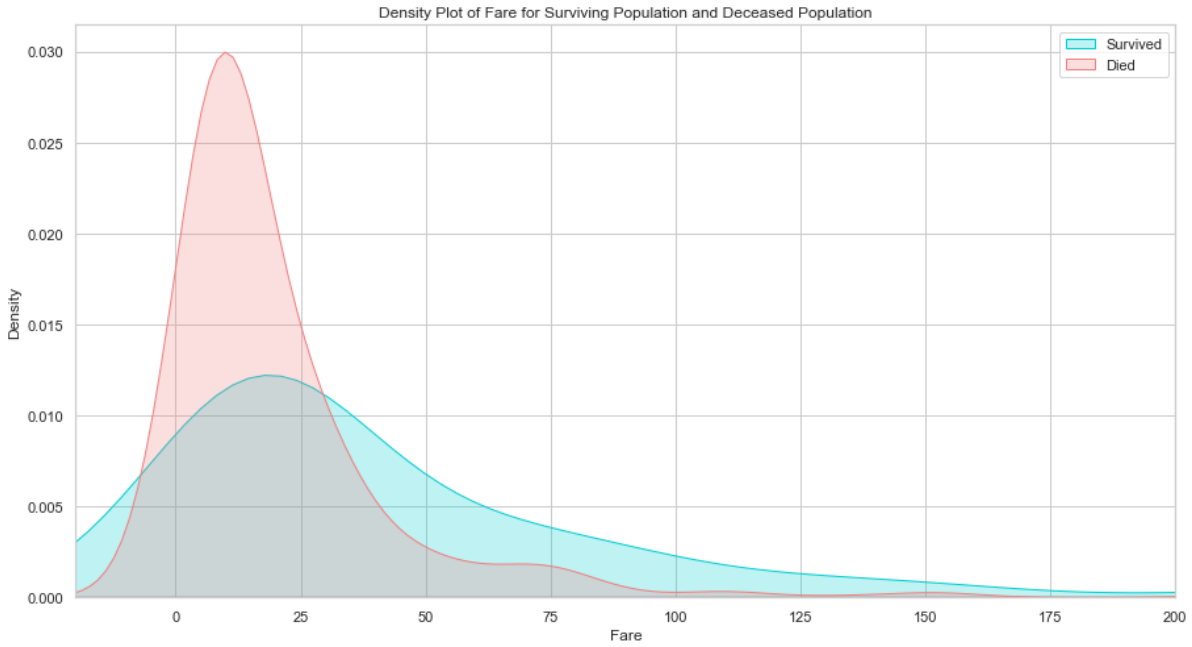


In [40]:
```
plt.figure(figsize=(20,8))
avg_survival_byage = final_train[["Age", "Survived"]].groupby(['Age'], as_index
g = sns.barplot(x='Age', y='Survived', data=avg_survival_byage, color="LightSea
plt.show()
```

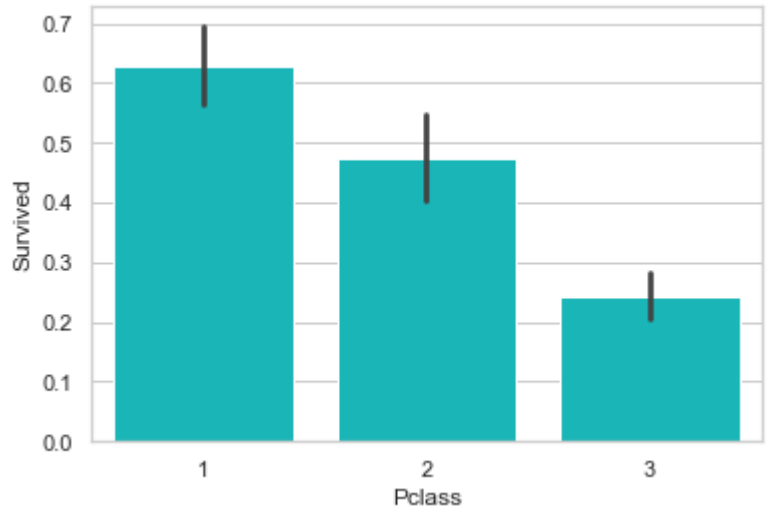


In [41]:
```
final_train['IsMinor']=np.where(final_train['Age']<=16, 1, 0)

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

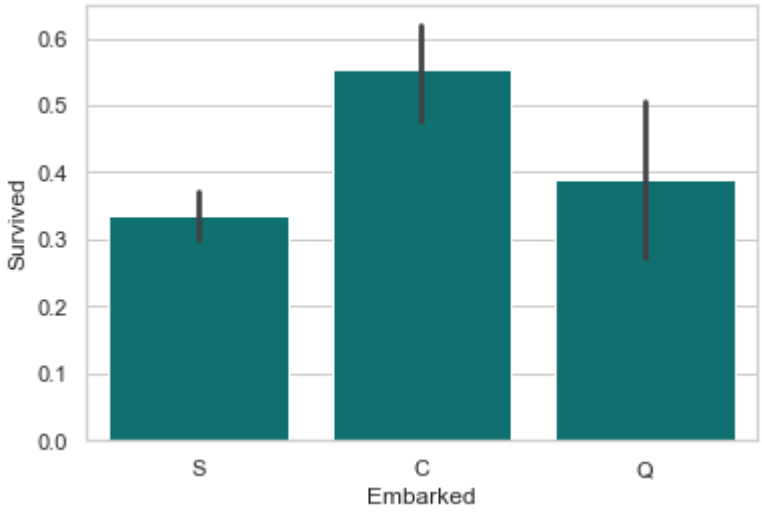
In [42]:
```
#Exploration of Fare
plt.figure(figsize=(15,8))
ax = sns.kdeplot(final_train["Fare"][final_train.Survived == 1], color="darktur
sns.kdeplot(final_train["Fare"][final_train.Survived == 0], color="lightcoral",
plt.legend(['Survived', 'Died'])
plt.title('Density Plot of Fare for Surviving Population and Deceased Populatic
ax.set(xlabel='Fare')
plt.xlim(-20,200)
plt.show()
```



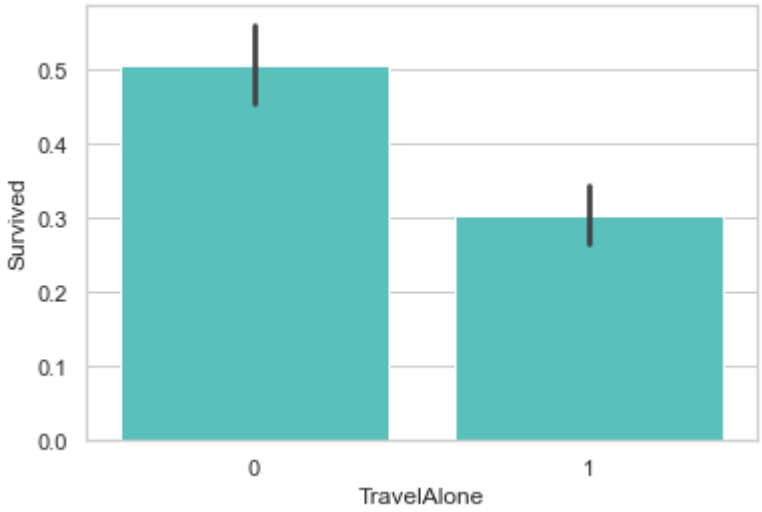
In [43]:
```
#Exploration of Passenger Class
sns.barplot('Pclass', 'Survived', data=train_df, color="darkturquoise")
plt.show()
```



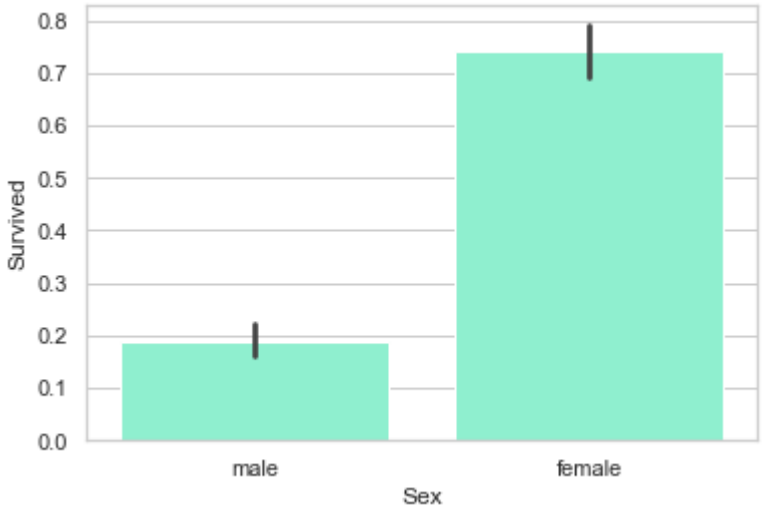
In [44]:
```
#Exploration of Embarked Port
sns.barplot('Embarked', 'Survived', data=train_df, color="teal")
plt.show()
```



In [45]:
```
#Exploration of Traveling Alone vs. With Family
sns.barplot('TravelAlone', 'Survived', data=final_train, color="mediumturquoise
plt.show()
```



In [46]:
```
#Exploration of Gender Variable
sns.barplot('Sex', 'Survived', data=train_df, color="aquamarine")
plt.show()
```

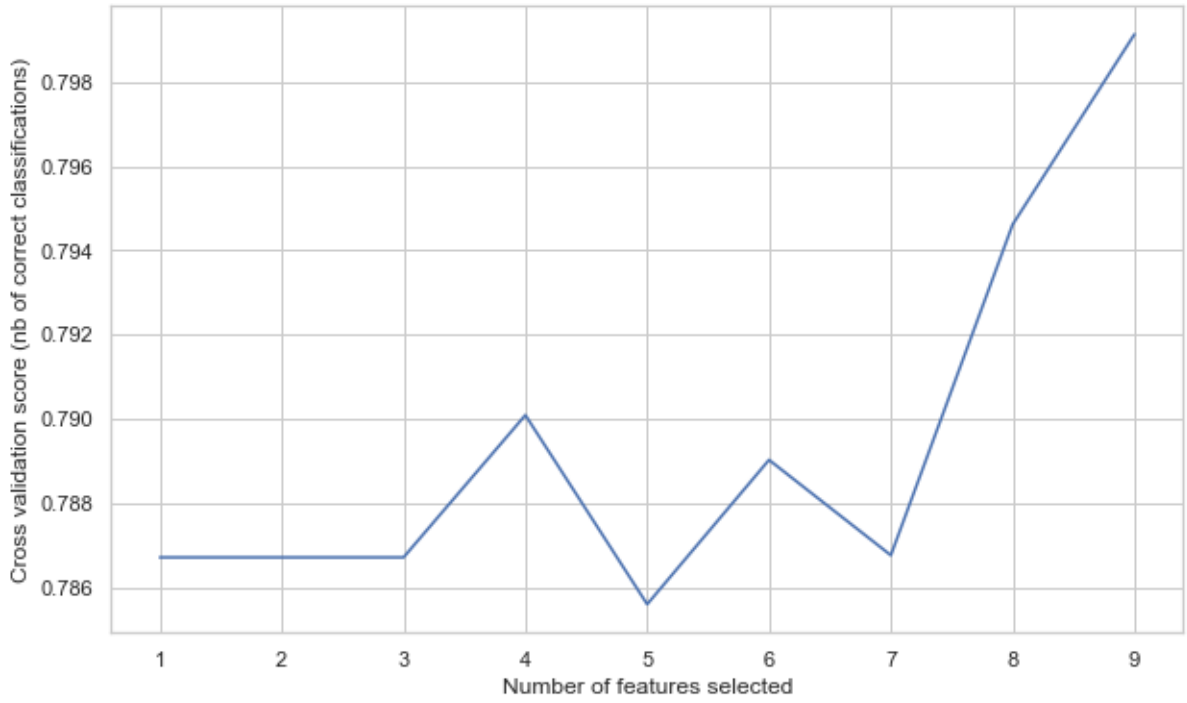


In [47]:
```
#Logistic Regression and Results
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

cols = ["Age","Fare","TravelAlone","Pclass_1","Pclass_2","Embarked_C","Embarked
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_]))
```

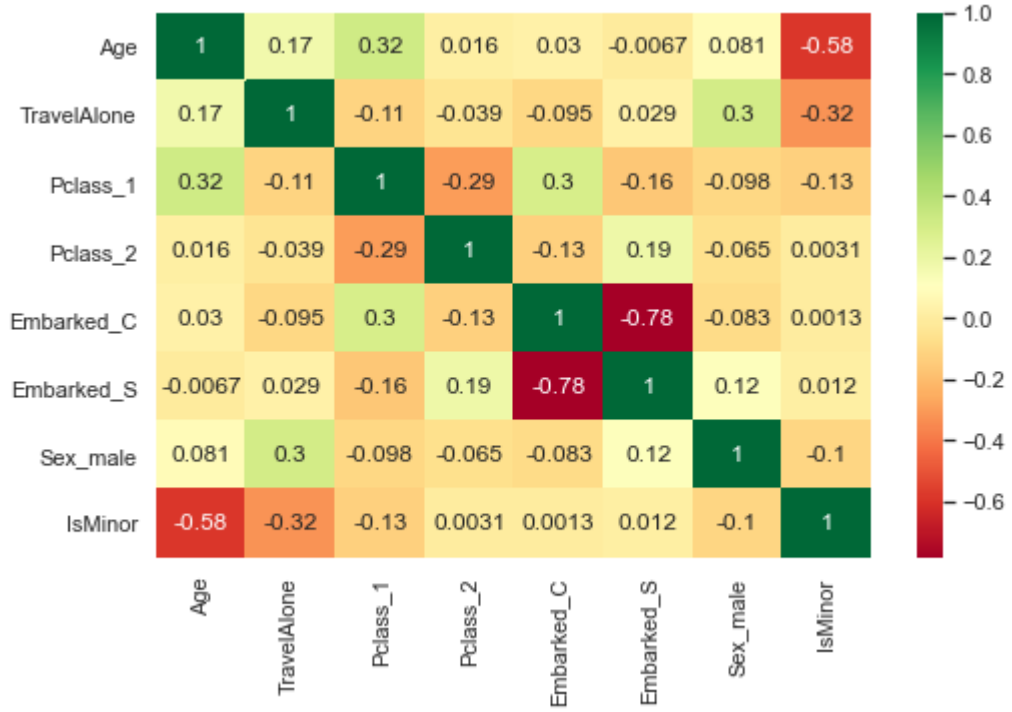
```
Selected features: ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_
C', 'Embarked_S', 'Sex_male', 'IsMinor']
```

In [48]:
```
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 classificatio
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', 'Em
barked_C', 'Embarked_S', 'Sex_male', 'IsMinor']
```

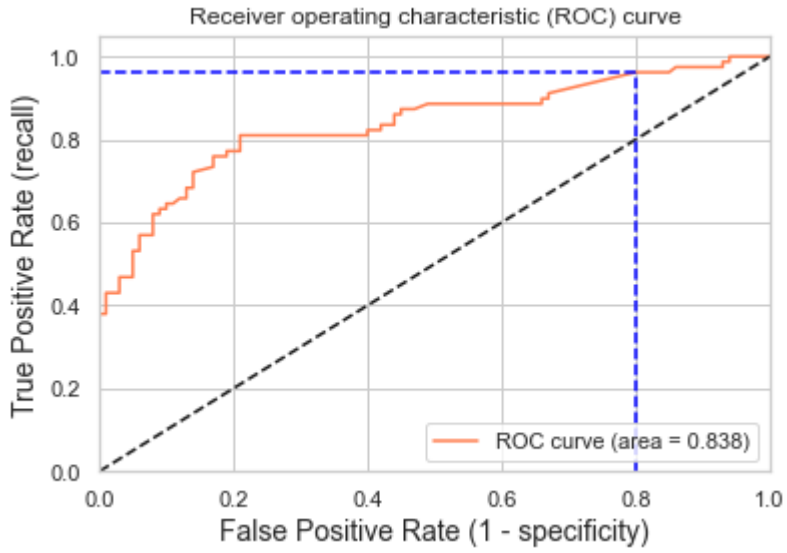


In [49]:
```
Selected_features = ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C'
                     'Embarked_S', 'Sex_male', 'IsMinor']
X = final_train[Selected_features]

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



In [50]:
```
#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_sc
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_curve

# 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 estima
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# 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
print(logreg.__class__.__name__+" log_loss is %2.3f" % log_loss(y_test, y_pred_
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

plt.figure()
plt.plot(fpr, tpr, color='coral', label='ROC curve (area = %0.3f)' % auc(fpr, t
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 %.
      "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 specificit
y of 0.200, i.e. a false positive rate of 80.00%.

In [51]:
```
# 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
# cv=10 for 10 folds
# scoring = {'accuracy', 'neg_log_loss', 'roc_auc'} for evaluation metric - alt
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.
print(logreg.__class__.__name__+" average log_loss is %2.3f" % -scores_log_loss
print(logreg.__class__.__name__+" average auc is %2.3f" % scores_auc.mean())
```

```
K-fold cross-validation results:
LogisticRegression average accuracy is 0.795
LogisticRegression average log_loss is 0.454
LogisticRegression average auc is 0.850
```

In [52]:
```
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(scor
                                if list(scoring.values())[sc]=='neg_log_loss'
                                else results['test_%s' % list(scoring.values())[
                                results['test_%s' % list(scoring.values())[sc]].
```

```
K-fold cross-validation results:
LogisticRegression average accuracy: 0.795 (+/-0.025)
LogisticRegression average log_loss: 0.454 (+/-0.037)
LogisticRegression average auc: 0.850 (+/-0.028)
```

In [53]:
```
#What happens when we add the feature "Fare"?

cols = ["Age","Fare","TravelAlone","Pclass_1","Pclass_2","Embarked_C","Embarked
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(sco
                         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(scor
                                if list(scoring.values())[sc]=='neg_log_loss'
                                else results['test_%s' % list(scoring.values())[
                                results['test_%s' % list(scoring.values())[sc]].
```

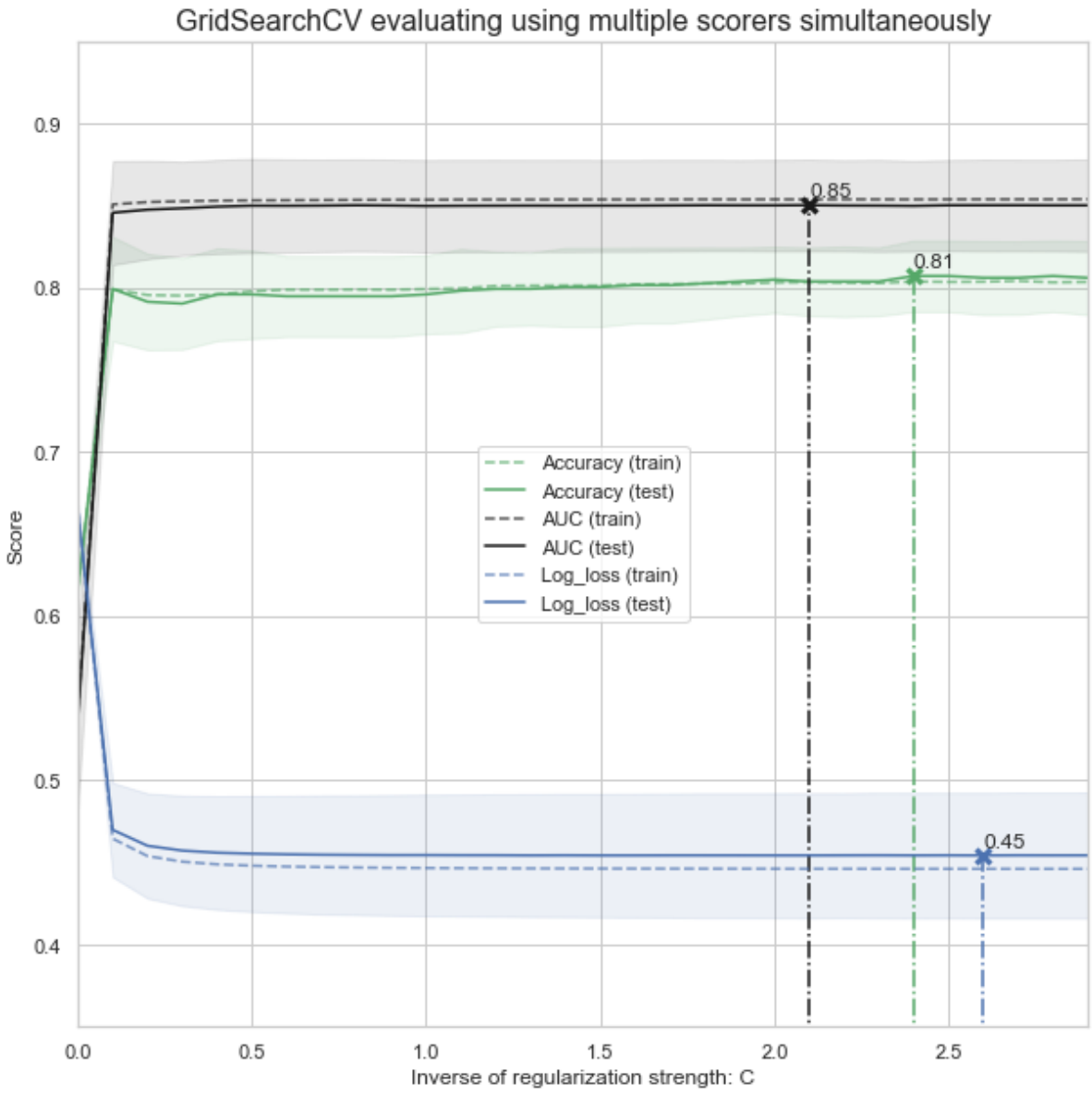
```
K-fold cross-validation results:
LogisticRegression average accuracy: 0.799 (+/-0.028)
LogisticRegression average log_loss: 0.455 (+/-0.037)
LogisticRegression average auc: 0.849 (+/-0.028)
```

In [54]:

```
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='Accurac

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",fonts

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 scorin
        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[score

    # 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 [*]:

```
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

#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",fonts

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 scorin
        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[score

    # 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()
```

In [*]:
```
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()
```

In [ ]:
```

```

In [1]:
```
print("welcom to PYTHON_PROGRAMMING")
```

```
welcom to PYTHON_PROGRAMMING
```

In [2]:
```
a=input("Enter a value")
b=input("Enter a value")
c=int(a) + int(b)
print ("the sum is",c)
```

```
Enter a value5
Enter a value10
the sum is 15
```

In [3]:
```
a = 5
print(a, "is of type", type(a))
a= 2.0
print(a, "is of type", type(a))
a = 1+2J
print(a, "is complex number", isinstance(1+2J,complex))
```

```
5 is of type <class 'int'>
2.0 is of type <class 'float'>
(1+2j) is complex number True
```

In [4]:
```
a =34
print(type(a))
b=45545454545
print(type(b))
print(b)
```

```
<class 'int'>
<class 'int'>
45545454545
```

In [6]:
```
f=3.1455
print(f)
print(type(f))
f1=1.255
print(f1)
```

```
3.1455
<class 'float'>
1.255
```

In [7]:
```
c=3+4j
print(type(c))
print(c.real)
print(c.imag)
```

```
<class 'complex'>
3.0
4.0
```

In [8]:
```
b=True
print(type(b))
a=5
b=8
c=a<b
print(c)
```

```
<class 'bool'>
True
```

In [9]:
```
s = "This is a string"
print(s)
s =" " 'A multiline String'""
print(s)
```

```
This is a string
 A multiline String
```

In [10]:
```
s1 = 'python'
print(s1)
s2 = "record"
print(s2)
```

```
python
record
```

In [11]:
```
id(1025)
```

Out[11]: 1653791110608

In [12]:
```
id(1025)
```

Out[12]: 1653791111184

In [13]:
```
id("RGM")
```

Out[13]: 1653791340848

In [17]:
```
s = "RGM"
print(id(s))
```

```
1653791341808
```

In [18]:
```
s1 = "RGM"
print(id(s1))
```

```
1653791341808
```

In [20]:
```
print(id(s)==id(s1))
```

```
True
```

In [21]:
```
list1 = ["python", "java", "c++"]
print(id(list1[0]))
print(id(list1[2]))
```

```
1653759382320
1653792005552
```

In [24]:
```
print(id(list1[0])==id(list1[2]))
```

```
False
```

In [25]:
```
print('id of 5 =',id(5))
a=5
print('id of a =',id(a))
b=a
print('id of a =',id(b))
c=2.3
print('id of a =',id(c))
```

```
id of 5 = 140708949665712
id of a = 140708949665712
id of a = 140708949665712
id of a = 1653791110384
```

In [26]:
```
r = range(10)
print(type(r))
print(r)
```

```
<class 'range'>
range(0, 10)
```

In [27]:
```
r = range(10)
print(type(r))
print(r)
for i in r:
    print(i)
```

```
<class 'range'>
range(0, 10)
0
1
2
3
4
5
6
7
8
9
```

In [28]:
```
r = range(10)
print(type(r))
print(r)
for i in r:
    print(i,end = ' ')
```

```
<class 'range'>
range(0, 10)
0 1 2 3 4 5 6 7 8 9
```

In [29]:
```
r = range(10)
print(type(r))
print(r)
for i in r:
    print(i,end = '\t')
```

```
<class 'range'>
range(0, 10)
0	1	2	3	4	5	6	7	8	9
```

In [30]:
```
r = range(10,-25)
print(type(r))
print(r)
for i in r:
    print(i,end = ' ')
```

```
<class 'range'>
range(10, -25)
```

In [31]:
```
r = range(-25,10)
print(type(r))
print(r)
for i in r:
    print(i,end = ' ')
```

```
<class 'range'>
range(-25, 10)
-25 -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -
5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9
```

In [32]:
```
r = range(1,21,1)
for i in r:
    print(i,end = ' ')
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

In [33]:
```
r = range(1,21,2)
for i in r:
    print(i,end = ' ')
```

```
1 3 5 7 9 11 13 15 17 19
```

In [34]:
```
r = range(1,21,3)
for i in r:
    print(i,end = ' ')
```

```
1 4 7 10 13 16 19
```

In [35]:
```
r = range(10,20)
print(r[0])
print(r[-1])
r1 = r[1:5]
print(r1)
for i in r:
    print(i)
```

```
10
19
range(11, 15)
10
11
12
13
14
15
16
17
18
19
```

In [36]:
```
print(bin(12))
print(bin(0XA11))
print(bin(0o2345))
print(bin(54))
```

```
0b1100
0b101000010001
0b10011100101
0b110110
```

In [37]:
```
print(oct(12))
print(oct(0XA11))
print(oct(0o2345))
print(oct(54))
```

```
0o14
0o5021
0o2345
0o66
```

In [38]:
```
print(hex(12))
print(hex(0XA11))
print(hex(0o2345))
print(hex(54))
```

```
0xc
0xa11
0x4e5
0x36
```

In [39]:
```
print(int(45.80))
print(int(True))
print(int(False))
```

```
45
1
0
```

In [40]:
```
print(bool(0))
print(bool(1))
print(bool(4))
print(bool(2.0))
print(bool(2+3j))
print(bool(True))
print(bool(False))
```

```
False
True
True
True
True
True
False
```

In [41]:
```
x = 2
y = 3
print("addition result :", x + y)
```

```
addition result : 5
```

In [42]:
```
x = 2
y = 3.3
print("addition result :", x + y)
```

```
addition result : 5.3
```

In [43]:
```
x = 2.2
y = 3.3
print("addition result :", x + y)
```

```
addition result : 5.5
```

In [48]:
```
x = "2"
y = "3"
print("addition result :", x + y)
```

```
addition result : 23
```

In [45]:
```
x = 2
y = 3
print("subtraction result :", x - y)
```

```
subtraction result : -1
```

In [47]:
```
x = 10
y = 5.0
print("subtraction result :", x - y)
```

```
subtraction result : 5.0
```

In [49]:
```
x = 10
y = 5
print("multiplication result :", x*y)
```

```
multiplication result : 50
```

In [50]:
```
x = 10
y = 5.0
print("multiplication result :", x*y)
```

```
multiplication result : 50.0
```

In [51]:
```
x = "10"
y = 5
print("multiplication result :", x*y)
```

```
multiplication result : 1010101010
```

In [52]:
```
x = 10
y = 5
print("division result :", x/y)
```

```
division result : 2.0
```

In [53]:
```
x = 10
y = complex((30))
print("multiplication result :", x*y)
```

```
multiplication result : (300+0j)
```

In [54]:
```
a = 3
b = 4
print(a%b)
print(b%a)
```

```
3
1
```

In [55]:
```
a = 10
b = 20
print('a < b is', a<b)

print('a <= b is', a<=b)

print('a > b is', a>b)

print('a >= b is', a>=b)
```

```
a < b is True
a <= b is True
a > b is False
a >= b is False
```

In [57]:
```
print(ord('b'))
print(ord('B'))
```

```
98
66
```

In [58]:
```
print(chr(97))
print(chr(65))
```

```
a
A
```

In [59]:
```
print(10==20)
print(10!=20)
```

```
False
True
```

In [60]:
```
print(10==True)
print(10==10.0)
print('Tejas'=='Tejas')
```

```
False
True
True
```

In [61]:
```
print(True and True)
print(True and False)
print(False and True)
print(False and False)
```

```
True
False
False
False
```

In [62]:
```
print(True or True)
print(True or False)
print(False or True)
print(False or False)
```

```
True
True
True
False
```

In [63]:
```
print(not True)
print(not False)
```

```
False
True
```

In [68]:
```
username = input('Enter User Name :')
Password = input('Enter Password :')
if username == 'Tejas2425' and Password == '242503':
    print("valid user")
else:
    print('invalide user')
```

```
Enter User Name :Tejas2425
Enter Password :242503
valid user
```

In [69]:
```
username = input('Enter User Name :')
Password = input('Enter Password :')
if username == 'Tejas2425' and Password == '242503':
    print("valid user")
else:
    print('invalide user')
```

```
Enter User Name :abc
Enter Password :123
invalide user
```

#Assignment Operators

In [66]:
```
x = 10
x += 20
print(x)
```

```
30
```

#Ternary Operator (or) Conditional Operator

In [67]:
```
a,b=25,43
c = 50 if a>b else 100
print(c)
```

```
100
```

In [71]:
```
x=int(input("Enter First Number :"))
y=int(input("Enter Second Number"))
min=x if x<y else y
print("Mimimum Value :",min)
```

```
Enter First Number :50
Enter Second Number100
Mimimum Value : 50
```

In [72]:
```
x=int(input("Enter First Number :"))
y=int(input("Enter Second Number"))
min=x if x<y else y
print("Mimimum Value :",min)
```

```
Enter First Number :100
Enter Second Number60
Mimimum Value : 60
```

In [73]:
```
x=int(input("Enter First Number :"))
y=int(input("Enter Second Number"))
z=int(input("Enter third Number"))
min=x if x<y and x<z else y if y<z else z
print("Mimimum Value :",min)
```

```
Enter First Number :234
Enter Second Number134
Enter third Number34
Mimimum Value : 34
```

In [74]:
```
x=int(input("Enter First Number :"))
y=int(input("Enter Second Number"))
z=int(input("Enter third Number"))
min=x if x>y and x>z else y if y>z else z
print("Mimimum Value :",min)
```

```
Enter First Number :234
Enter Second Number134
Enter third Number34
Mimimum Value : 234
```

In [75]:
```
x="Hello learning python is very easy!!"
print('h' in x)
print('d' in x)
print('d' not in x)
print('python' in x)
print('Python' in x)
```

```
True
False
True
True
False
```

In [76]:
```
list1=["sunny", "bunny", "chinny", "pinny"]
print("sunny" in list1)
print("tunny" in list1)
print("sunny" not in list1)
```

```
True
False
False
```

In [77]:
```
print(3+10*2)
print((3+10)*2)
```

```
23
26
```

In [1]:
```
a = 30
b=20
c=10
d=5
print((a+b)*c/d)
print((a+b)*(c/d))
```

```
100.0
100.0
```

In [ ]:
```
# program to read Employee data from the ketboard and print that data
```

In [ ]:
```
eno = int(input("Enter Employee No:"))
ename = input("Enter Employee Name:")
esal = float(input("Enter Employee salary:"))
eadd = input("Enter Employee Address:")
married = bool(input("Enter Married ?[ True | False]:"))
print("Plese Confirm your provided Information")
print("Employee No :", eno)
print("Employee Name :", ename)
print("Employee Salary :", esal)
print("Employee Address :", eadd)
print("Employee Married ? :", married)
```

In [ ]:
```
a,b,c = 10,20,30
print(a,b,c)
print(a,b,c sep=',')
print(a,b,c sep=',')
print(a,b,c sep=',')
```

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

In [3]:
```
s1=pd.Series([1,2,3,4,5])
s1
```

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

In [5]:
```
s1=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
s1
```

Out[5]:
```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

In [6]:
```
pd.Series({'a':10,'b':20,'c':30})
```

Out[6]:
```
a    10
b    20
c    30
dtype: int64
```

In [7]:
```
pd.Series({'a':10,'b':20,'c':30},index=['b','c','d','a'])
s1[3]
```

Out[7]:
```
4
```

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

Out[8]:
```
4
```

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

Out[9]:
```
0    1
1    2
2    3
3    4
dtype: int64
```

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

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

In [11]:
```
s1+5
```

Out[11]:
```
0     6
1     7
2     8
3     9
4    10
5    11
6    12
7    13
8    14
9    15
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 [14]:
```
pd.DataFrame({"Name":['Bob','Sam','Anne'],"Marks":[76,45,98]})
```

Out[14]:

	Name	Marks
0	Bob	76
1	Sam	45
2	Anne	98

In [22]:
```
df=pd.read_csv('Book1.csv')
df
```

Out[22]:

	S.no	Name	marks
0	1	priya	45
1	2	rohan	55
2	3	shina	78
3	4	rohit	67
4	5	anaya	68
5	6	neha	98
6	7	nimita	67
7	8	raj	46
8	9	rajni	65
9	10	esha	57

In [23]:
```
df.head()
```

Out[23]:

	S.no	Name	marks
0	1	priya	45
1	2	rohan	55
2	3	shina	78
3	4	rohit	67
4	5	anaya	68

In [24]:
```
df.tail()
```

Out[24]:

	S.no	Name	marks
5	6	neha	98
6	7	nimita	67
7	8	raj	46
8	9	rajni	65
9	10	esha	57

In [25]:
```
df.describe()
```

Out[25]:

	S.no	marks
count	10.00000	10.000000
mean	5.50000	64.600000
std	3.02765	15.629033
min	1.00000	45.000000
25%	3.25000	55.500000
50%	5.50000	66.000000
75%	7.75000	67.750000
max	10.00000	98.000000

In [26]:
```
df.iloc[0:3,0:2]
```

Out[26]:

	S.no	Name
0	1	priya
1	2	rohan
2	3	shina

In [27]:
```
df.iloc[0:3,0:3]
```

Out[27]:

	S.no	Name	marks
0	1	priya	45
1	2	rohan	55
2	3	shina	78

In [28]:
```
df.loc[0:3,{"Name","marks"}]
```

Out[28]:

	marks	Name
0	45	priya
1	55	rohan
2	78	shina
3	67	rohit

In [29]:
```
df.drop('marks', axis=1)
```

Out[29]:

	S.no	Name
0	1	priya
1	2	rohan
2	3	shina
3	4	rohit
4	5	anaya
5	6	neha
6	7	nimita
7	8	raj
8	9	rajni
9	10	esha

In [30]:
```
df.drop([1,2,3],axis=0)
```

Out[30]:

	S.no	Name	marks
0	1	priya	45
4	5	anaya	68
5	6	neha	98
6	7	nimita	67
7	8	raj	46
8	9	rajni	65
9	10	esha	57

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

Out[31]:
```
S.no      5.5
marks    64.6
dtype: float64
```

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

Out[32]:
```
64.6
```

In [33]:
```
df.median()
```

Out[33]:
```
S.no      5.5
marks    66.0
dtype: float64
```

In [34]:
```
df2=df["marks"].median()
df2
```

Out[34]:
```
66.0
```

In [35]:
```
df.min()
```

Out[35]:
```
S.no         1
Name     anaya
marks       45
dtype: object
```

In [36]:
```
df2=df["marks"].min()
df2
```

Out[36]:
```
45
```

In [37]:
```
df.max()
```

Out[37]:
```
S.no        10
Name     shina
marks       98
dtype: object
```

In [38]:
```
df2=df["marks"].max()
df2
```

Out[38]:
```
98
```

In [ ]:
```

```

Pandas stands for panel data and is the core library for data Manipulation and data Analysis
Pandas Data-Structures Series-dimensional-Series Object,Multidimensional-Data Frame

Series Object is one-Dimensional labeled array

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

In [2]:
```
s1=pd.Series([1,2,3,4,5])
s1
```

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

Changing Index

In [4]:
```
s1=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
s1
```

Out[4]:
```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

Series Object from Dictionary

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

Out[5]:
```
a    10
b    20
c    30
dtype: int64
```

Changing index Position

In [6]:
```
pd.Series({'a':10,'b':20,'c':30},index=['b','c','d','a'])
```

Out[6]:
```
b    20.0
c    30.0
d     NaN
a    10.0
dtype: float64
```

Extracting Individual element

Extracting a single element

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

Out[7]:
```
4
```

Extracting a Sequence of element

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

Out[8]:
```
0    1
1    2
2    3
3    4
dtype: int64
```

Extracting element from back

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

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

Basic Math Operation on Series

Adding a scalar value to Series Elements

In [11]:
```
s1+5
```

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

Addition two series Objects

In [14]:
```
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[14]:
```
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
```

pandas Dataframe

Data Frame is a Two Dimensional labelled Data Structure

In [15]:
```
pd.DataFrame({"Name":['Bob','Sam','Anne'],"Marks":[45,85,98]})
```

Out[15]:

	Name	Marks
0	Bob	45
1	Sam	85
2	Anne	98

DataFRame in Build Function

Shape(),head(),tail(),describe()

In [20]:
```
df=pd.read_csv('Book1.csv')
df
```

Out[20]:

	S.no	Name	marks
0	1	priya	20
1	2	rohan	15
2	3	tejas	30
3	4	manish	25
4	5	vanchita	21
5	6	sachin	21
6	7	swarupa	23
7	8	ashwini	21
8	9	pratik	20
9	10	sumit	21

In [21]:
```
df=pd.read_csv(r'D:\Book1.csv')
df
```

Out[21]:

	S.no	Name	marks
0	1	priya	20
1	2	rohan	15
2	3	tejas	30
3	4	manish	25
4	5	vanchita	21
5	6	sachin	21
6	7	swarupa	23
7	8	ashwini	21
8	9	pratik	20
9	10	sumit	21

Read First Five tuples of csv file

In [23]:
```
df.tail()
```

Out[23]:

	S.no	Name	marks
5	6	sachin	21
6	7	swarupa	23
7	8	ashwini	21
8	9	pratik	20
9	10	sumit	21

In [24]:
```
df.describe()
```

Out[24]:

	S.no	marks
count	10.00000	10.000000
mean	5.50000	21.700000
std	3.02765	3.860052
min	1.00000	15.000000
25%	3.25000	20.250000
50%	5.50000	21.000000
75%	7.75000	22.500000
max	10.00000	30.000000

In [25]:
```
df.iloc[0:3,0:2]
```

Out[25]:

	S.no	Name
0	1	priya
1	2	rohan
2	3	tejas

In [26]:
```
df.iloc[0:3,0:3]
```

Out[26]:

	S.no	Name	marks
0	1	priya	20
1	2	rohan	15
2	3	tejas	30

In [27]:
```
df.loc[0:3,("Name","marks")]
```

Out[27]:

	Name	marks
0	priya	20
1	rohan	15
2	tejas	30
3	manish	25

Droping Coloumns

In [28]:
```
df.drop('marks',axis=1)
```

Out[28]:

	S.no	Name
0	1	priya
1	2	rohan
2	3	tejas
3	4	manish
4	5	vanchita
5	6	sachin
6	7	swarupa
7	8	ashwini
8	9	pratik
9	10	sumit

Droping Row

In [30]:
```
df.drop([1,2,3],axis=0)
```

Out[30]:

	S.no	Name	marks
0	1	priya	20
4	5	vanchita	21
5	6	sachin	21
6	7	swarupa	23
7	8	ashwini	21
8	9	pratik	20
9	10	sumit	21

More Pandas Function

mean,median,minimum,maximum

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

Out[31]:
```
S.no      5.5
marks    21.7
dtype: float64
```

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

Out[32]:
```
21.7
```

In [33]:
```
df.median()
```

Out[33]:
```
S.no      5.5
marks    21.0
dtype: float64
```

In [39]:
```
df2=df["marks"].median()
df2
```

Out[39]:
```
21.0
```

In [40]:
```
df.min()
```

Out[40]:
```
S.no           1
Name     ashwini
marks         15
dtype: object
```

In [36]:
```
df2=df["marks"].min()
df2
```

Out[36]:
```
15
```

In [37]:
```
df.max()
```

Out[37]:
```
S.no           10
Name      vanchita
marks          30
dtype: object
```

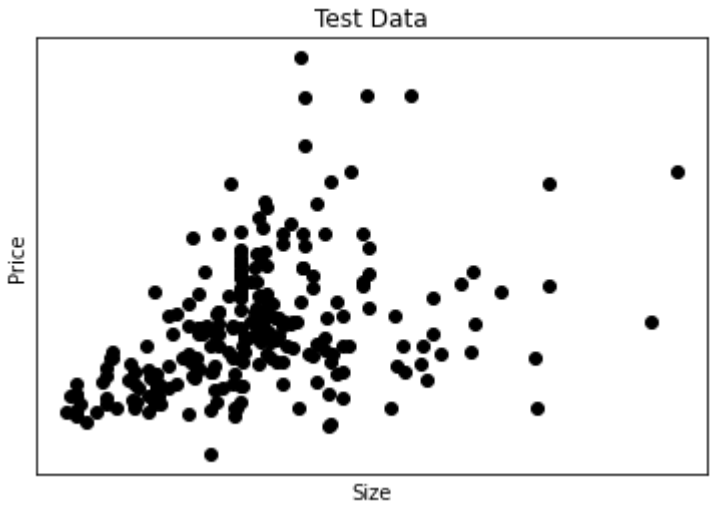
In [38]:
```
df2=df["marks"].max()
df2
```

Out[38]:
```
30
```

In [ ]:

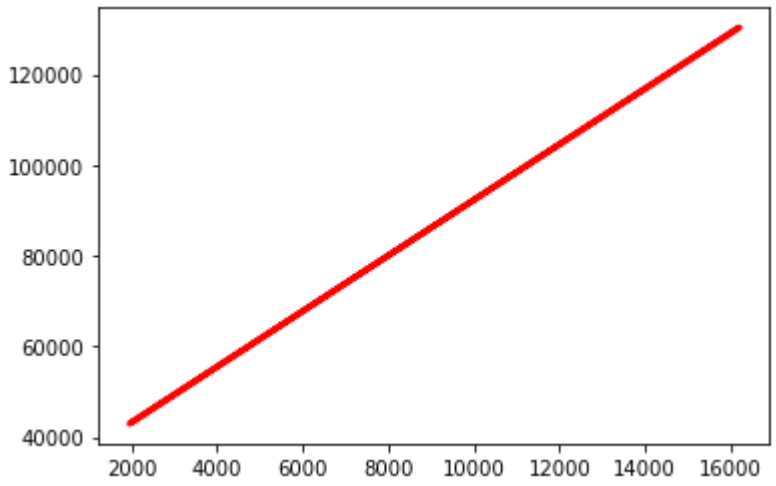
In [1]:

```
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 [2]:
```
# 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[2]:
```
[<matplotlib.lines.Line2D at 0x13673298af0>]
```



In [ ]:

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)
```

```
 [5.1 2.5 3.  1.1]
 [5.7 2.8 4.1 1.3]
 [6.3 3.3 6.  2.5]
 [5.8 2.7 5.1 1.9]
 [7.1 3.  5.9 2.1]
 [6.3 2.9 5.6 1.8]
 [6.5 3.  5.8 2.2]
 [7.6 3.  6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
```

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 [9]:
```
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
Confusion Matrix
[[11  0  0]
 [ 0 14  1]
 [ 0  0 19]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      0.93      0.97        15
           2       0.95      1.00      0.97        19

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

In [ ]:

In [1]:

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

plt.style.use("seaborn-whitegrid")
```

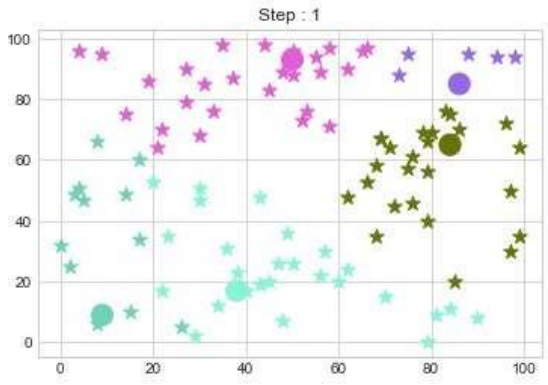
In [2]:

```
class KMedoidsClass:
    def __init__(self,data,k,iters):
        self.data= data
        self.k = k
        self.iters = iters
        self.medoids = np.array([data[i] for i in range(self.k)])
        self.colors = np.array(np.random.randint(0, 255, size =(self.k, 4)))/255
        self.colors[:,3]=1

    def manhattan(self,p1, p2):
        return np.abs((p1[0]-p2[0])) + np.abs((p1[1]-p2[1]))

    def get_costs(self, medoids, data):
        tmp_clusters = {i:[] for i in range(len(medoids))}
        cst = 0
        for d in data:
            dst = np.array([self.manhattan(d, md) for md in medoids])
            c = dst.argmin()
            tmp_clusters[c].append(d)
            cst+=dst.min()

        tmp_clusters = {k:np.array(v) for k,v in tmp_clusters.items()}
        return tmp_clusters, cst

    def fit(self):

        samples,_ = self.data.shape

        self.clusters, cost = self.get_costs(data=self.data, medoids=self.medoids)
        count = 0

        colors =  np.array(np.random.randint(0, 255, size =(self.k, 4)))/255
        colors[:,3]=1

        plt.title(f"Step : 0")
        [plt.scatter(self.clusters[t][:, 0], self.clusters[t][:, 1], marker="*", s=100,
                                    color = colors[t]) for t in range(self.k)]
        plt.scatter(self.medoids[:, 0], self.medoids[:, 1], s=200, color=colors)
        plt.show()

        while True:
            swap = False
            for i in range(samples):
                if not i in self.medoids:
                    for j in range(self.k):
                        tmp_meds = self.medoids.copy()
                        tmp_meds[j] = i
                        clusters_, cost_ = self.get_costs(data=self.data, medoids=tmp_meds)

                        if cost_<cost:
                            self.medoids = tmp_meds
                            cost = cost_
                            swap = True
                            self.clusters = clusters_
                            print(f"Medoids Changed to: {self.medoids}.")
                            plt.title(f"Step : {count+1}")
                            [plt.scatter(self.clusters[t][:, 0], self.clusters[t][:, 1], marker="*", s=100,
                                        color = colors[t]) for t in range(self.k)]
                            plt.scatter(self.medoids[:, 0], self.medoids[:, 1], s=200, color=colors)
                            plt.show()
            count+=1

            if count>=self.iters:
                print("End of the iterations.")
                break
            if not swap:
                print("No changes.")
                break
```

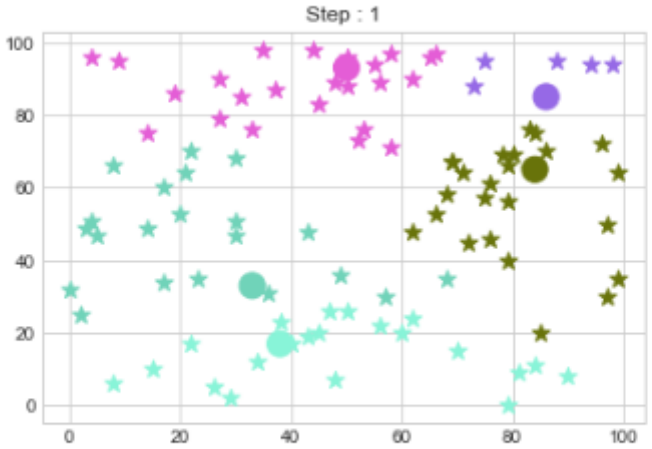
In [3]:

```
dt = np.random.randint(0,100, (100,2))
kmedoid = KMedoidsClass(dt,5,5)
kmedoid.fit()
```

```
[38 17]
[86 85]
[ 9  9]].
```

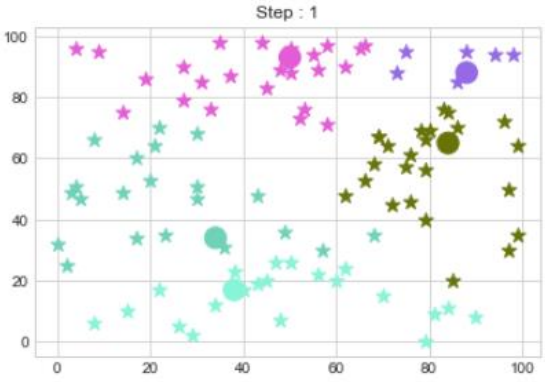



```
[33 33]].
```




```
Medoids Changed to: [[84 65]
 [50 93]
```

```
[88 88]
[34 34]].
```

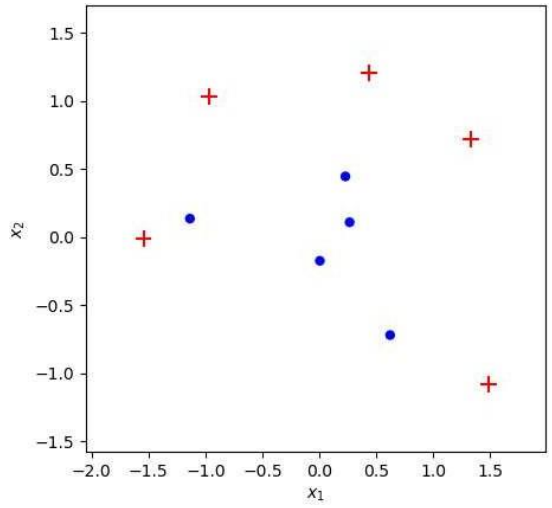



```
No changes.
```

In [1]:
```
from typing import Optional
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
```

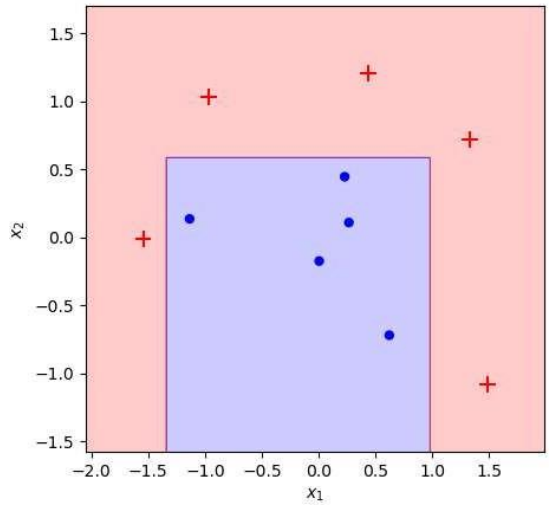
In [2]:
```
def plot_adaboost(X: np.ndarray,
                  y: np.ndarray,
                  clf=None,
                  sample_weights: Optional[np.ndarray] = None,
                  annotate: bool = False,
                  ax: Optional[mpl.axes.Axes] = None) -> None:
    """ Plot ± samples in 2D, optionally with decision boundary """

    assert set(y) == {-1, 1}, 'Expecting response labels to be ±1'

    if not ax:
        fig, ax = plt.subplots(figsize=(5, 5), dpi=100)
        fig.set_facecolor('white')

    pad = 1
    x_min, x_max = X[:, 0].min() - pad, X[:, 0].max() + pad
    y_min, y_max = X[:, 1].min() - pad, X[:, 1].max() + pad

    if sample_weights is not None:
        sizes = np.array(sample_weights) * X.shape[0] * 100
    else:
        sizes = np.ones(shape=X.shape[0]) * 100

    X_pos = X[y == 1]
    sizes_pos = sizes[y == 1]
    ax.scatter(*X_pos.T, s=sizes_pos, marker='+', color='red')

    X_neg = X[y == -1]
    sizes_neg = sizes[y == -1]
    ax.scatter(*X_neg.T, s=sizes_neg, marker='.', c='blue')

    if clf:
        plot_step = 0.01
        xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                             np.arange(y_min, y_max, plot_step))

        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)

        # If all predictions are positive class, adjust color map acordingly
        if list(np.unique(Z)) == [1]:
            fill_colors = ['r']
        else:
            fill_colors = ['b', 'r']

        ax.contourf(xx, yy, Z, colors=fill_colors, alpha=0.2)

    if annotate:
        for i, (x, y) in enumerate(X):
            offset = 0.05
            ax.annotate(f'$x_{i + 1}$', (x + offset, y - offset))

    ax.set_xlim(x_min+0.5, x_max-0.5)
    ax.set_ylim(y_min+0.5, y_max-0.5)
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')
```

In [3]:
```
from sklearn.datasets import make_gaussian_quantiles
from sklearn.model_selection import train_test_split

def make_toy_dataset(n: int = 100, random_seed: int = None):
    """ Generate a toy dataset for evaluating AdaBoost classifiers """

    n_per_class = int(n/2)

    if random_seed:
        np.random.seed(random_seed)

    X, y = make_gaussian_quantiles(n_samples=n, n_features=2, n_classes=2)

    return X, y*2-1

X, y = make_toy_dataset(n=10, random_seed=10)
plot_adaboost(X, y)
```



In [4]:
```
from sklearn.ensemble import AdaBoostClassifier

bench = AdaBoostClassifier(n_estimators=10, algorithm='SAMME').fit(X, y)
plot_adaboost(X, y, bench)

train_err = (bench.predict(X) != y).mean()
print(f'Train error: {train_err:.1%}')
```

```
Train error: 0.0%
```

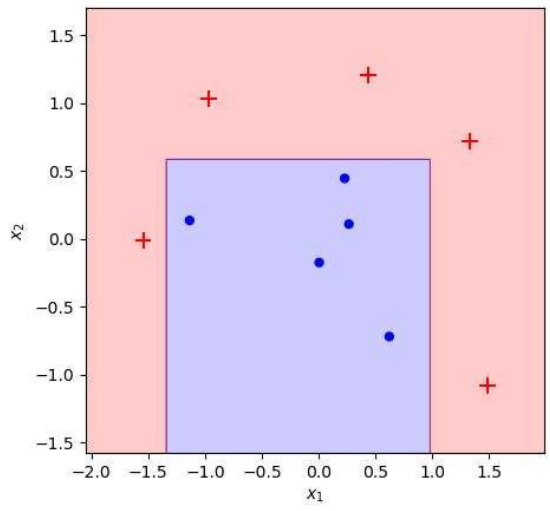


In [5]:
```
class AdaBoost:


    def __init__(self):
        self.stumps = None
        self.stump_weights = None
        self.errors = None
        self.sample_weights = None

    def _check_X_y(self, X, y):
        """ Validate assumptions about format of input data"""
        assert set(y) == {-1, 1}, 'Response variable must be ±1'
        return X, y
```

In [6]:
```
from sklearn.tree import DecisionTreeClassifier

def fit(self, X: np.ndarray, y: np.ndarray, iters: int):
    """ Fit the model using training data """

    X, y = self._check_X_y(X, y)
    n = X.shape[0]

    # init numpy arrays
    self.sample_weights = np.zeros(shape=(iters, n))
    self.stumps = np.zeros(shape=iters, dtype=object)
    self.stump_weights = np.zeros(shape=iters)
    self.errors = np.zeros(shape=iters)

    # initialize weights uniformly
    self.sample_weights[0] = np.ones(shape=n) / n

    for t in range(iters):
        # fit  weak learner
        curr_sample_weights = self.sample_weights[t]
        stump = DecisionTreeClassifier(max_depth=1, max_leaf_nodes=2)
        stump = stump.fit(X, y, sample_weight=curr_sample_weights)

        # calculate error and stump weight from weak learner prediction
        stump_pred = stump.predict(X)
        err = curr_sample_weights[(stump_pred != y)].sum()# / n
        stump_weight = np.log((1 - err) / err) / 2

        # update sample weights
        new_sample_weights = (
            curr_sample_weights * np.exp(-stump_weight * y * stump_pred)
        )

        new_sample_weights /= new_sample_weights.sum()

        # If not final iteration, update sample weights for t+1
        if t+1 < iters:
            self.sample_weights[t+1] = new_sample_weights

        # save results of iteration
        self.stumps[t] = stump
        self.stump_weights[t] = stump_weight
        self.errors[t] = err

    return self
#Making predictions
#We make a final prediction by taking a “weighted majority vote”, calculated as the sign (+) of the linear combination of each stump’s predi

#$$ H_t(x) = \text{sign} \Big( \sum_{t=1}^T a_t h_t(x) \Big) $$

def predict(self, X):
    """ Make predictions using already fitted model """
    stump_preds = np.array([stump.predict(X) for stump in self.stumps])
    return np.sign(np.dot(self.stump_weights, stump_preds))
```

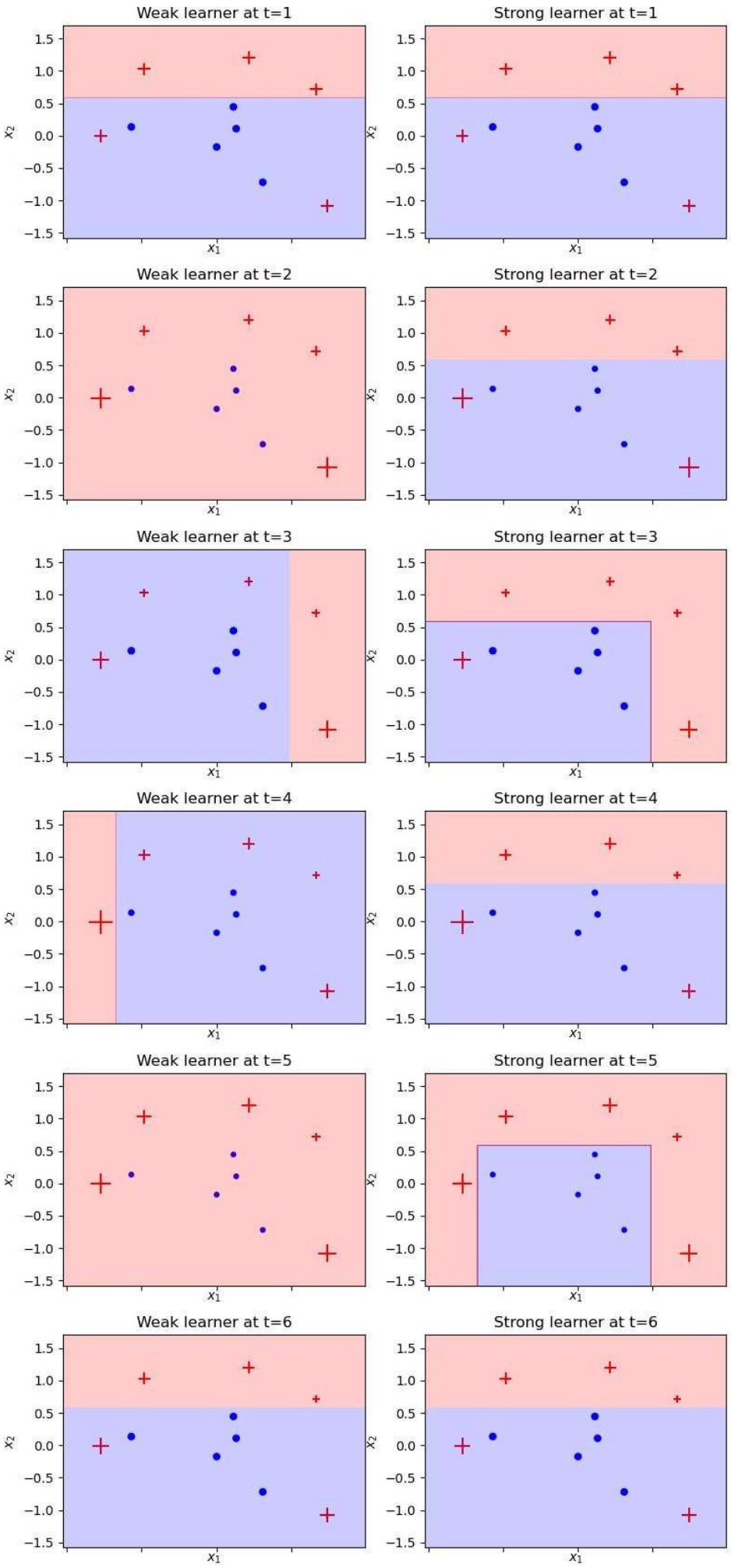
In [7]:
```
# assign our individually defined functions as methods of our classifier
AdaBoost.fit = fit
AdaBoost.predict = predict

clf = AdaBoost().fit(X, y, iters=10)
plot_adaboost(X, y, clf)

train_err = (clf.predict(X) != y).mean()
print(f'Train error: {train_err:.1%}')
```

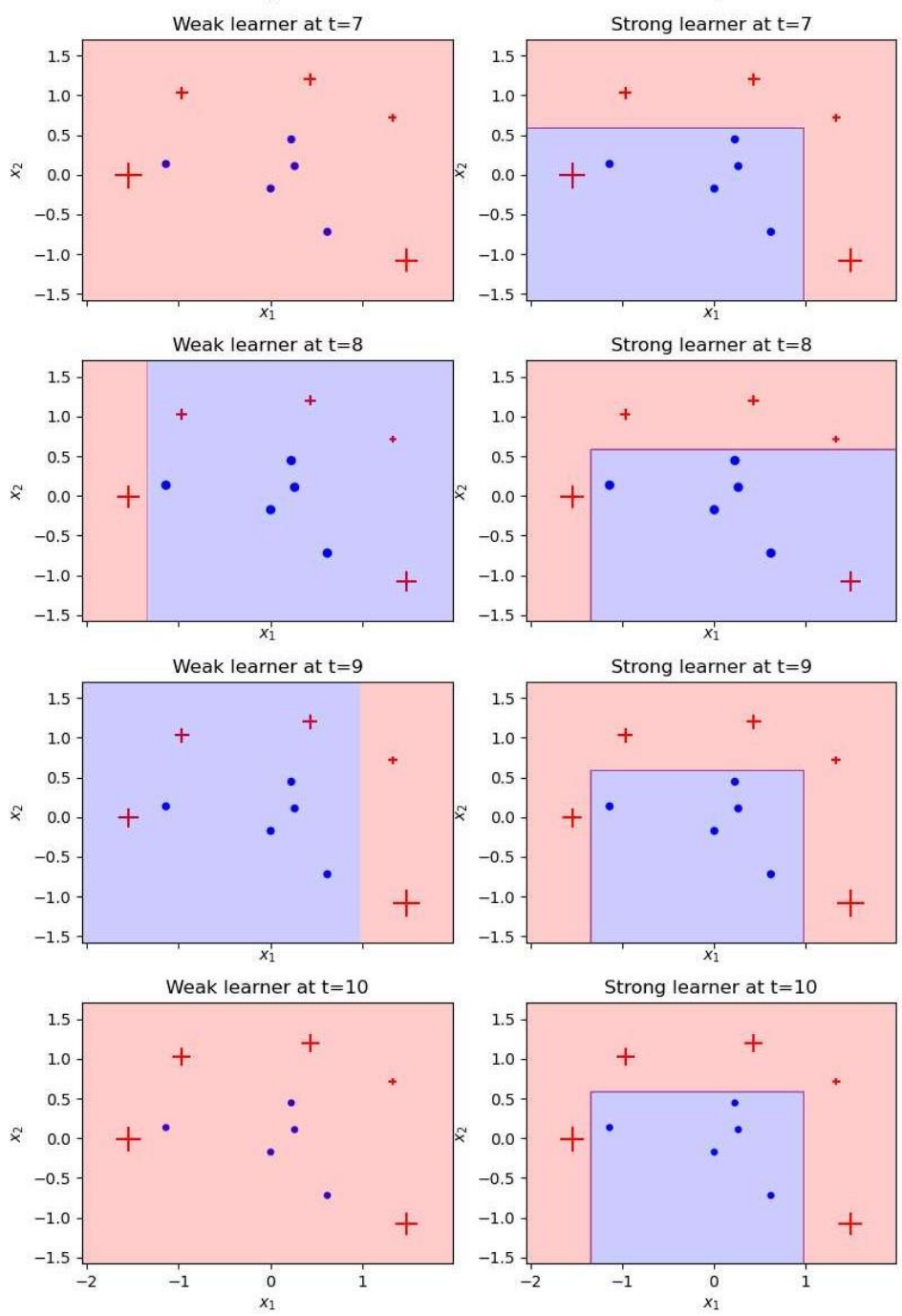
```
Train error: 0.0%
```



In [8]:
```
def truncate_adaboost(clf, t: int):
    """ Truncate a fitted AdaBoost up to (and including) a particular iteration """
    assert t > 0, 't must be a positive integer'
    from copy import deepcopy
    new_clf = deepcopy(clf)
    new_clf.stumps = clf.stumps[:t]
    new_clf.stump_weights = clf.stump_weights[:t]
    return new_clf


def plot_staged_adaboost(X, y, clf, iters=10):
    """ Plot weak learner and cumulaive strong learner at each iteration. """

    # Larger grid
    fig, axes = plt.subplots(figsize=(8, iters*3),
                             nrows=iters,
                             ncols=2,
                             sharex=True,
                             dpi=100)

    fig.set_facecolor('white')

    _ = fig.suptitle('Decision boundaries by iteration')
    for i in range(iters):
        ax1, ax2 = axes[i]

        # Plot weak learner
        _ = ax1.set_title(f'Weak learner at t={i + 1}')
        plot_adaboost(X, y, clf.stumps[i],
                      sample_weights=clf.sample_weights[i],
                      annotate=False, ax=ax1)

        # Plot strong learner
        trunc_clf = truncate_adaboost(clf, t=i + 1)
        _ = ax2.set_title(f'Strong learner at t={i + 1}')
        plot_adaboost(X, y, trunc_clf,
                      sample_weights=clf.sample_weights[i],
                      annotate=False, ax=ax2)

    plt.tight_layout()
    plt.subplots_adjust(top=0.95)
    plt.show()

clf = AdaBoost().fit(X, y, iters=10)
plot_staged_adaboost(X, y, clf)
```

Decision boundaries by iteration




Weak learner at t=7
Strong learner at t=7
Weak learner at t=8
Strong learner at t=8
Weak learner at t=9
Strong learner at t=9
Weak learner at t=10
Strong learner at t=10
x_1
x_2

In [19]:
```
#importing libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from random import sample
import random
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn import tree
from math import log,exp
from sklearn.datasets import load_iris
```

In [20]:
```
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
```

In [85]:
```
#importing file
iris = pd.read_csv("iris.csv")
```

In [86]:
```
iris.head(1)
```

Out[86]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa

In [87]:
```
#considering only two classes
example = iris[(iris['species'] == 'versicolor') | (iris['species'] == 'virginica')]
```

In [88]:
```
example.head(2)
```

Out[88]:

	sepal_length	sepal_width	petal_length	petal_width	species
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor

In [27]:
```
#replacing the two classes with +1 and -1
example['Label'] = example['species'].replace(to_replace = ['versicolor','virginica'], value=[1,-1])
```

```
<ipython-input-27-0a1dde03b3fa>:2: 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-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  example['Label'] = example['species'].replace(to_replace = ['versicolor','virginica'], value=[1,-1])
```

In [28]:
```
example = example.drop('species', axis = 1)
```

In [29]:
```
#Initially assign same weights to each records in the dataset
example['probR1'] = 1/(example.shape[0])
```

In [30]:
```
example.head(5)
```

Out[30]:

	sepal_length	sepal_width	petal_length	petal_width	Label	probR1
50	7.0	3.2	4.7	1.4	1	0.01
51	6.4	3.2	4.5	1.5	1	0.01
52	6.9	3.1	4.9	1.5	1	0.01
53	5.5	2.3	4.0	1.3	1	0.01
54	6.5	2.8	4.6	1.5	1	0.01

In [31]:
```
#simple random sample with replacement
random.seed(10)
example1 = example.sample(len(example), replace = True, weights = example['probR1'])
```

In [32]:
```
example1
```

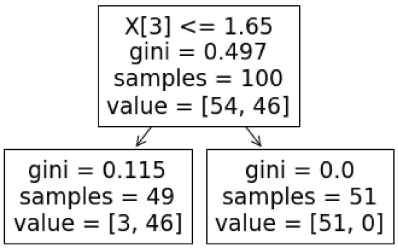
116	6.5	3.0	5.5	1.8	-1	0.01
110	6.5	3.2	5.1	2.0	-1	0.01
126	6.2	2.8	4.8	1.8	-1	0.01
137	6.4	3.1	5.5	1.8	-1	0.01
73	6.1	2.8	4.7	1.2	1	0.01
100	6.3	3.3	6.0	2.5	-1	0.01
78	6.0	2.9	4.5	1.5	1	0.01
64	5.6	2.9	3.6	1.3	1	0.01
67	5.8	2.7	4.1	1.0	1	0.01
144	6.7	3.3	5.7	2.5	-1	0.01
146	6.3	2.5	5.0	1.9	-1	0.01
101	5.8	2.7	5.1	1.9	-1	0.01
57	4.9	2.4	3.3	1.0	1	0.01

In [33]:
```
#X_train and Y_train split
X_train = example1.iloc[0:len(iris),0:4]
y_train = example1.iloc[0:len(iris),4]
```

In [34]:
```
#fitting the DT model with depth one
clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100, max_depth=1)
clf = clf_gini.fit(X_train, y_train)
```

In [35]:
```
#plotting tree for round 1 boosting
tree.plot_tree(clf)
```

Out[35]:
```
[Text(167.4, 163.07999999999998, 'X[3] <= 1.65\ngini = 0.497\nsamples = 100\nvalue = [54, 46]'),
 Text(83.7, 54.360000000000014, 'gini = 0.115\nsamples = 49\nvalue = [3, 46]'),
 Text(251.10000000000002, 54.360000000000014, 'gini = 0.0\nsamples = 51\nvalue = [51, 0]')]
```



In [36]:
```
#prediction
y_pred = clf_gini.predict(example.iloc[0:len(iris),0:4])
y_pred
```

Out[36]:
```
array([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1, -1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1, -1, -1, -1,  1,  1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],
      dtype=int64)
```

In [37]:
```
#adding a column pred1 after the first round of boosting
example['pred1'] = y_pred
```

In [38]:
```
example
```

Out[38]:

	sepal_length	sepal_width	petal_length	petal_width	Label	probR1	pred1
50	7.0	3.2	4.7	1.4	1	0.01	1
51	6.4	3.2	4.5	1.5	1	0.01	1
52	6.9	3.1	4.9	1.5	1	0.01	1
53	5.5	2.3	4.0	1.3	1	0.01	1
54	6.5	2.8	4.6	1.5	1	0.01	1
55	5.7	2.8	4.5	1.3	1	0.01	1
56	6.3	3.3	4.7	1.6	1	0.01	1
57	4.9	2.4	3.3	1.0	1	0.01	1
58	6.6	2.9	4.6	1.3	1	0.01	1
59	5.2	2.7	3.9	1.4	1	0.01	1
60	5.0	2.0	3.5	1.0	1	0.01	1

In [39]:
```
#misclassified = 0 if the label and prediction are same
example.loc[example.Label != example.pred1, 'misclassified'] = 1
example.loc[example.Label == example.pred1, 'misclassified'] = 0
```

In [40]:
```
#error calculation
e1 = sum(example['misclassified'] * example['probR1'])
```

In [41]:
```
e1
```

Out[41]:
```
0.060000000000000005
```

In [42]:
```
#calculation of alpha (performance)
alpha1 = 0.5*log((1-e1)/e1)
```

In [43]:
```
#update weight
new_weight = example['probR1']*np.exp(-1*alpha1*example['Label']*example['pred1'])
```

In [44]:
```
#normalized weight
z = sum(new_weight)
normalized_weight = new_weight/sum(new_weight)
```

In [45]:
```
example['prob2'] = round(normalized_weight,4)
```

In [46]:
```
example
```

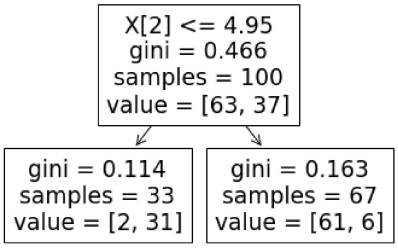
Out[46]:

	sepal_length	sepal_width	petal_length	petal_width	Label	probR1	pred1	misclassified	prob2
50	7.0	3.2	4.7	1.4	1	0.01	1	0.0	0.0053
51	6.4	3.2	4.5	1.5	1	0.01	1	0.0	0.0053
52	6.9	3.1	4.9	1.5	1	0.01	1	0.0	0.0053
53	5.5	2.3	4.0	1.3	1	0.01	1	0.0	0.0053
54	6.5	2.8	4.6	1.5	1	0.01	1	0.0	0.0053
55	5.7	2.8	4.5	1.3	1	0.01	1	0.0	0.0053
56	6.3	3.3	4.7	1.6	1	0.01	1	0.0	0.0053
57	4.9	2.4	3.3	1.0	1	0.01	1	0.0	0.0053
58	6.6	2.9	4.6	1.3	1	0.01	1	0.0	0.0053
59	5.2	2.7	3.9	1.4	1	0.01	1	0.0	0.0053
60	5.0	2.0	3.5	1.0	1	0.01	1	0.0	0.0053

In [47]:
```
#round 2
random.seed(20)
example2 = example.sample(len(example), replace = True, weights = example['prob2'])
example2 = example2.iloc[:,0:5]
X_train = example2.iloc[0:len(iris),0:4]
y_train = example2.iloc[0:len(iris),4]

clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100, max_depth=1)
clf = clf_gini.fit(X_train, y_train)

y_pred = clf_gini.predict(example.iloc[0:len(iris),0:4])
#adding a column pred2 after the second round of boosting
example['pred2'] = y_pred
```

In [48]:
```
#plotting tree for round 2 boosting
tree.plot_tree(clf)
```

Out[48]:
```
[Text(167.4, 163.07999999999998, 'X[2] <= 4.95\ngini = 0.466\nsamples = 100\nvalue = [63, 37]'),
 Text(83.7, 54.360000000000014, 'gini = 0.114\nsamples = 33\nvalue = [2, 31]'),
 Text(251.10000000000002, 54.360000000000014, 'gini = 0.163\nsamples = 67\nvalue = [61, 6]')]
```



In [49]:
```
example
```

Out[49]:

	sepal_length	sepal_width	petal_length	petal_width	Label	probR1	pred1	misclassified	prob2	pred2
50	7.0	3.2	4.7	1.4	1	0.01	1	0.0	0.0053	1
51	6.4	3.2	4.5	1.5	1	0.01	1	0.0	0.0053	1
52	6.9	3.1	4.9	1.5	1	0.01	1	0.0	0.0053	1
53	5.5	2.3	4.0	1.3	1	0.01	1	0.0	0.0053	1
54	6.5	2.8	4.6	1.5	1	0.01	1	0.0	0.0053	1
55	5.7	2.8	4.5	1.3	1	0.01	1	0.0	0.0053	1
56	6.3	3.3	4.7	1.6	1	0.01	1	0.0	0.0053	1
57	4.9	2.4	3.3	1.0	1	0.01	1	0.0	0.0053	1
58	6.6	2.9	4.6	1.3	1	0.01	1	0.0	0.0053	1
59	5.2	2.7	3.9	1.4	1	0.01	1	0.0	0.0053	1
60	5.0	2.0	3.5	1.0	1	0.01	1	0.0	0.0053	1

In [50]:
```
#adding a field misclassified2
example.loc[example.Label != example.pred2, 'misclassified2'] = 1
example.loc[example.Label == example.pred2, 'misclassified2'] = 0
```

In [51]:
```
# calculation of error
e2 = sum(example['misclassified2'] * example['prob2'])
e2
```

Out[51]:
```
0.1204
```

In [52]:
```
#calculation of alpha
alpha2 = 0.5*log((1-e2)/e2)
alpha2
```

Out[52]:
```
0.9943238629029427
```

In [53]:
```
#update weight
new_weight = example['prob2']*np.exp(-1*alpha2*example['Label']*example['pred2'])
z = sum(new_weight)
normalized_weight = new_weight/sum(new_weight)
```

In [54]:
```
example['prob3'] = round(normalized_weight,4)
```

In [55]:
```
example
```

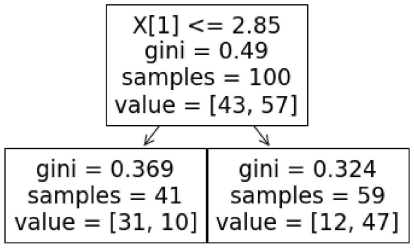
Out[55]:

	sepal_length	sepal_width	petal_length	petal_width	Label	probR1	pred1	misclassified	prob2	pred2	misclassified2	prob3
50	7.0	3.2	4.7	1.4	1	0.01	1	0.0	0.0053	1	0.0	0.0030
51	6.4	3.2	4.5	1.5	1	0.01	1	0.0	0.0053	1	0.0	0.0030
52	6.9	3.1	4.9	1.5	1	0.01	1	0.0	0.0053	1	0.0	0.0030
53	5.5	2.3	4.0	1.3	1	0.01	1	0.0	0.0053	1	0.0	0.0030
54	6.5	2.8	4.6	1.5	1	0.01	1	0.0	0.0053	1	0.0	0.0030
55	5.7	2.8	4.5	1.3	1	0.01	1	0.0	0.0053	1	0.0	0.0030
56	6.3	3.3	4.7	1.6	1	0.01	1	0.0	0.0053	1	0.0	0.0030
57	4.9	2.4	3.3	1.0	1	0.01	1	0.0	0.0053	1	0.0	0.0030
58	6.6	2.9	4.6	1.3	1	0.01	1	0.0	0.0053	1	0.0	0.0030
59	5.2	2.7	3.9	1.4	1	0.01	1	0.0	0.0053	1	0.0	0.0030
60	5.0	2.0	3.5	1.0	1	0.01	1	0.0	0.0053	1	0.0	0.0030

In [56]:
```
#round 3
random.seed(30)
example3 = example.sample(len(example), replace = True, weights = example['prob3'])
example3 = example3.iloc[:,0:5]
X_train = example3.iloc[0:len(iris),0:4]
y_train = example3.iloc[0:len(iris),4]

clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100, max_depth=1)
clf = clf_gini.fit(X_train, y_train)

#adding a column pred3 after the third round of boosting
y_pred = clf_gini.predict(example.iloc[0:len(iris),0:4])
example['pred3'] = y_pred
```

In [57]:
```
#plotting tree for round 3 boosting
tree.plot_tree(clf)
```

Out[57]:
```
[Text(167.4, 163.07999999999998, 'X[1] <= 2.85\ngini = 0.49\nsamples = 100\nvalue = [43, 57]'),
 Text(83.7, 54.360000000000014, 'gini = 0.369\nsamples = 41\nvalue = [31, 10]'),
 Text(251.10000000000002, 54.360000000000014, 'gini = 0.324\nsamples = 59\nvalue = [12, 47]')]
```



In [58]:
```
#adding a field misclassified3
example.loc[example.Label != example.pred3, 'misclassified3'] = 1
example.loc[example.Label == example.pred3, 'misclassified3'] = 0
```

In [59]:
```
#weighted error calculation
e3 = sum(example['misclassified3'] * example['prob3']) #/len(example)
e3
```

Out[59]:
```
0.2754000000000001
```

In [60]:
```
#calculation of performance(alpha)
alpha3 = 0.5*log((1-e3)/e3)
```

In [61]:
```
#update weight
new_weight = example['prob3']*np.exp(-1*alpha3*example['Label']*example['pred3'])
z = sum(new_weight)
normalized_weight = new_weight/sum(new_weight)
```

In [62]:
```
example['prob4'] = round(normalized_weight,4)
```

In [63]:
```
example
```

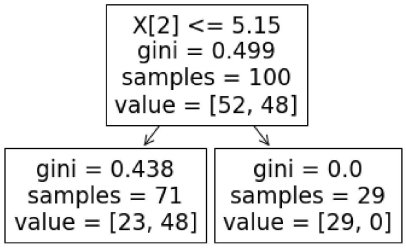
Out[63]:

	sepal_length	sepal_width	petal_length	petal_width	Label	probR1	pred1	misclassified	prob2	pred2	misclassified2	prob3	pred3	misclassified3	prob4
50	7.0	3.2	4.7	1.4	1	0.01	1	0.0	0.0053	1	0.0	0.0030	1	0.0	0.0021
51	6.4	3.2	4.5	1.5	1	0.01	1	0.0	0.0053	1	0.0	0.0030	1	0.0	0.0021
52	6.9	3.1	4.9	1.5	1	0.01	1	0.0	0.0053	1	0.0	0.0030	1	0.0	0.0021
53	5.5	2.3	4.0	1.3	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055
54	6.5	2.8	4.6	1.5	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055
55	5.7	2.8	4.5	1.3	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055
56	6.3	3.3	4.7	1.6	1	0.01	1	0.0	0.0053	1	0.0	0.0030	1	0.0	0.0021
57	4.9	2.4	3.3	1.0	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055
58	6.6	2.9	4.6	1.3	1	0.01	1	0.0	0.0053	1	0.0	0.0030	1	0.0	0.0021
59	5.2	2.7	3.9	1.4	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055
60	5.0	2.0	3.5	1.0	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055

In [64]:
```
#Round 4
random.seed(40)
example4 = example.sample(len(example), replace = True, weights = example['prob4'])
example4 = example4.iloc[:,0:5]
X_train = example4.iloc[0:len(iris),0:4]
y_train = example4.iloc[0:len(iris),4]

clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100, max_depth=1)
clf = clf_gini.fit(X_train, y_train)

#adding a column pred4 after the fourth round of boosting
y_pred = clf_gini.predict(example.iloc[0:len(iris),0:4])
example['pred4'] = y_pred
```

In [65]:
```
#plotting tree for round 4 boosting
tree.plot_tree(clf)
```

Out[65]:
```
[Text(167.4, 163.07999999999998, 'X[2] <= 5.15\ngini = 0.499\nsamples = 100\nvalue = [52, 48]'),
 Text(83.7, 54.360000000000014, 'gini = 0.438\nsamples = 71\nvalue = [23, 48]'),
 Text(251.10000000000002, 54.360000000000014, 'gini = 0.0\nsamples = 29\nvalue = [29, 0]')]
```



In [66]:
```
#adding a field misclassified4
example.loc[example.Label != example.pred4, 'misclassified4'] = 1
example.loc[example.Label == example.pred4, 'misclassified4'] = 0
```

In [67]:
```
#error calculation
e4 = sum(example['misclassified4'] * example['prob4'])
e4
```

Out[67]:
```
0.23320000000000002
```

In [68]:
```
# calculation of performance (alpha)
alpha4 = 0.5*log((1-e4)/e4)
```

In [69]:
```
#printing the alpha value which is used in each round of boosting
print(alpha1)
print(alpha2)
print(alpha3)
print(alpha4)
```

```
1.3757676565209744
0.9943238629029427
0.483697596083211
0.595164778347576
```

In [70]:
```
#final prediction
t = alpha1 * example['pred1'] + alpha2 * example['pred2'] + alpha3 * example['pred3'] + alpha4 * example['pred4']
```

In [71]:
```
#sign of the final prediction
np.sign(list(t))
```

Out[71]:
```
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1., -1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1.,  1., -1., -1., -1., -1., -1., -1., -1.,  1.,
       -1.,  1., -1., -1., -1.,  1., -1., -1., -1., -1.,  1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1.])
```

In [72]:
```
example['final_pred'] = np.sign(list(t))
```

In [73]:
```
example
```

Out[73]:

	sepal_length	sepal_width	petal_length	petal_width	Label	probR1	pred1	misclassified	prob2	pred2	misclassified2	prob3	pred3	misclassified3	prob4	pred
50	7.0	3.2	4.7	1.4	1	0.01	1	0.0	0.0053	1	0.0	0.0030	1	0.0	0.0021	
51	6.4	3.2	4.5	1.5	1	0.01	1	0.0	0.0053	1	0.0	0.0030	1	0.0	0.0021	
52	6.9	3.1	4.9	1.5	1	0.01	1	0.0	0.0053	1	0.0	0.0030	1	0.0	0.0021	
53	5.5	2.3	4.0	1.3	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055	
54	6.5	2.8	4.6	1.5	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055	
55	5.7	2.8	4.5	1.3	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055	
56	6.3	3.3	4.7	1.6	1	0.01	1	0.0	0.0053	1	0.0	0.0030	1	0.0	0.0021	
57	4.9	2.4	3.3	1.0	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055	
58	6.6	2.9	4.6	1.3	1	0.01	1	0.0	0.0053	1	0.0	0.0030	1	0.0	0.0021	
59	5.2	2.7	3.9	1.4	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055	
60	5.0	2.0	3.5	1.0	1	0.01	1	0.0	0.0053	1	0.0	0.0030	-1	1.0	0.0055	

In [74]:
```
#Confusion matrix
c=confusion_matrix(example['Label'], example['final_pred'])
c
```

Out[74]:
```
array([[45,  5],
       [ 1, 49]], dtype=int64)
```

In [75]:
```
#Overall Accuracy
(c[0,0]+c[1,1])/np.sum(c)*100
```

Out[75]:
```
94.0
```

In [76]:
```
#Fitting the model using the adaboost classifier library
```

In [77]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [83]:
```
iris = pd.read_csv("iris.csv")
iris = iris[(iris['species'] == 'versicolor') | (iris['species'] == 'virginica')]
```

In [79]:
```
#X_train and Y_train split
X_train = iris.iloc[0:len(iris),0:4]
y_train = iris.iloc[0:len(iris),4]
```

In [80]:
```
clf = AdaBoostClassifier(n_estimators=4, random_state=0)
clf.fit(X_train, y_train)
```

Out[80]:
```
AdaBoostClassifier(n_estimators=4, random_state=0)
```

In [81]:
```
clf.predict([[5.5, 2.5, 4.0, 1.3]])
```

Out[81]:
```
array(['versicolor'], dtype=object)
```

In [82]:
```
clf.score(X_train, y_train)
```

Out[82]:
```
0.9733333333333334
```

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
credit_df = pd.read_csv('CreditRisk.csv')
```

In [3]:
```
credit_df.shape
```

Out[3]: (614, 13)

In [4]:
```
credit_df.head()
```

Out[4]:

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

In [5]:
```
credit_df.tail()
```

Out[5]:

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

In [6]:
```
credit_df.info()
```

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

In [7]:
```
credit_df.describe()
```

Out[7]:

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

In [8]:
```
credit_df.Loan_Status.value_counts()
```

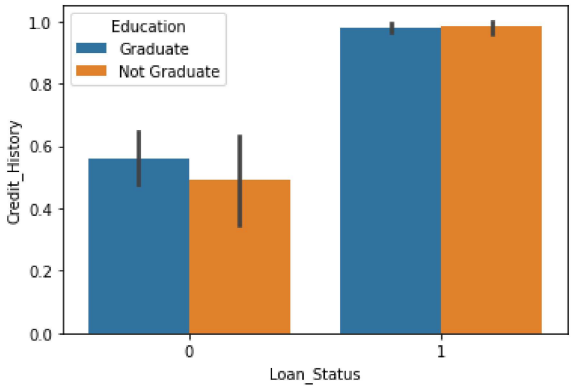
Out[8]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

In [9]:
```
credit_df.groupby(['Education', 'Loan_Status']).Education.count()
```

Out[9]:
```
Education     Loan_Status
Graduate      0              140
              1              340
Not Graduate  0               52
              1               82
Name: Education, dtype: int64
```

In [10]:
```
sns.barplot(y = 'Credit_History', x = 'Loan_Status', hue='Education', data = credit_df)
```

Out[10]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='Credit_History'>
```



In [11]:
```
100 * credit_df.isnull().sum() / credit_df.shape[0]
```

Out[11]:
```
Loan_ID              0.000000
Gender               2.117264
Married              0.488599
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           0.000000
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [12]:
```
object_columns = credit_df.select_dtypes(include=['object']).columns
numeric_columns = credit_df.select_dtypes(exclude=['object']).columns
```

In [13]:
```
for column in object_columns:
    majority = credit_df[column].value_counts().iloc[0]
    credit_df[column].fillna(majority, inplace=True)
```

In [14]:
```
for column in numeric_columns:
    mean = credit_df[column].mean()
    credit_df[column].fillna(mean, inplace=True)
```

In [15]:
```
credit_df.head()
```

Out[15]:

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

In [16]:
```
credit_df.drop('Loan_ID', axis=1, inplace=True)
```

In [17]:
```
object_columns = credit_df.select_dtypes(include=['object']).columns
```

In [18]:
```
credit_df.head()
```

Out[18]:

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

In [19]:
```
credit_df[object_columns].Property_Area
```

Out[19]:
```
0          Urban
1          Rural
2          Urban
3          Urban
4          Urban
         ...
609        Rural
610        Rural
611        Urban
612        Urban
613    Semiurban
Name: Property_Area, Length: 614, dtype: object
```

In [20]:
```
credit_df[object_columns].Property_Area.head()
```

Out[20]:
```
0    Urban
1    Rural
2    Urban
3    Urban
4    Urban
Name: Property_Area, dtype: object
```

In [21]:
```
credit_df_dummy = pd.get_dummies(credit_df, columns=object_columns)
```

In [22]:
```
credit_df_dummy.shape
```

Out[22]:
```
(614, 25)
```

In [23]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [24]:
```
X = credit_df_dummy.drop('Loan_Status', axis=1)
y = credit_df_dummy.Loan_Status
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [25]:
```
train_x.shape, test_x.shape
```

Out[25]:
```
((429, 24), (185, 24))
```

In [26]:
```
dt_model = DecisionTreeClassifier(max_depth=14)
```

In [27]:
```
dt_model.fit(train_x, train_y)
```

Out[27]:
```
DecisionTreeClassifier(max_depth=14)
```

In [28]:
```
train_y_hat = dt_model.predict(train_x)
test_y_hat = dt_model.predict(test_x)
```

In [29]:
```
print('-'*20, 'Train', '-'*20)
print(classification_report(train_y, train_y_hat))
print('-'*20, 'Test', '-'*20)
print(classification_report(test_y, test_y_hat))
```

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

           0       1.00      1.00      1.00       127
           1       1.00      1.00      1.00       302

    accuracy                           1.00       429
   macro avg       1.00      1.00      1.00       429
weighted avg       1.00      1.00      1.00       429

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

           0       0.57      0.52      0.54        65
           1       0.75      0.78      0.77       120

    accuracy                           0.69       185
   macro avg       0.66      0.65      0.66       185
weighted avg       0.69      0.69      0.69       185
```

In [30]:
```
print('-'*20, 'Train', '-'*20)
print(classification_report(train_y, train_y_hat))
print('-'*20, 'Test', '-'*20)
print(classification_report(test_y, test_y_hat))
```

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

           0       1.00      1.00      1.00       127
           1       1.00      1.00      1.00       302

    accuracy                           1.00       429
   macro avg       1.00      1.00      1.00       429
weighted avg       1.00      1.00      1.00       429

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

           0       0.57      0.52      0.54        65
           1       0.75      0.78      0.77       120

    accuracy                           0.69       185
   macro avg       0.66      0.65      0.66       185
weighted avg       0.69      0.69      0.69       185
```

In [1]:
```
from matplotlib import pyplot as plt
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

In [2]:
```
# Prepare the data data
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

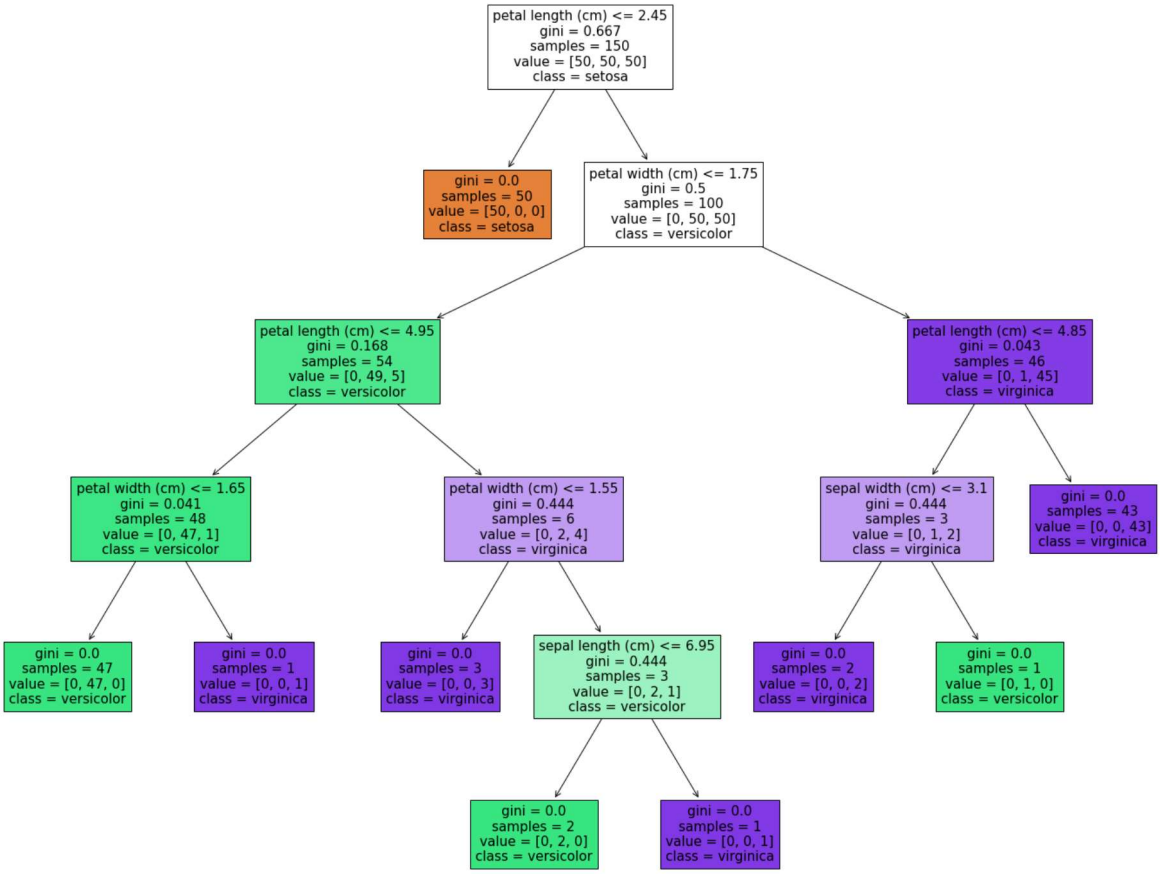
In [3]:
```
# Fit the classifier with default hyper-parameters
clf = DecisionTreeClassifier(random_state=1234)
model = clf.fit(X, y)
```

In [4]:
```
text_representation = tree.export_text(clf)
print(text_representation)
```

```
|--- feature_2 <= 2.45
|   |--- class: 0
|--- feature_2 >  2.45
|   |--- feature_3 <= 1.75
|   |   |--- feature_2 <= 4.95
|   |   |   |--- feature_3 <= 1.65
|   |   |   |   |--- class: 1
|   |   |   |--- feature_3 >  1.65
|   |   |   |   |--- class: 2
|   |   |--- feature_2 >  4.95
|   |   |   |--- feature_3 <= 1.55
|   |   |   |   |--- class: 2
|   |   |   |--- feature_3 >  1.55
|   |   |   |   |--- feature_0 <= 6.95
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_0 >  6.95
|   |   |   |   |   |--- class: 2
|   |--- feature_3 >  1.75
|   |   |--- feature_2 <= 4.85
|   |   |   |--- feature_1 <= 3.10
|   |   |   |   |--- class: 2
|   |   |   |--- feature_1 >  3.10
|   |   |   |   |--- class: 1
|   |   |--- feature_2 >  4.85
|   |   |   |--- class: 2
```

In [5]:
```
with open("decistion_tree.log", "w") as fout:
    fout.write(text_representation)
```

In [20]:
```
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)
```



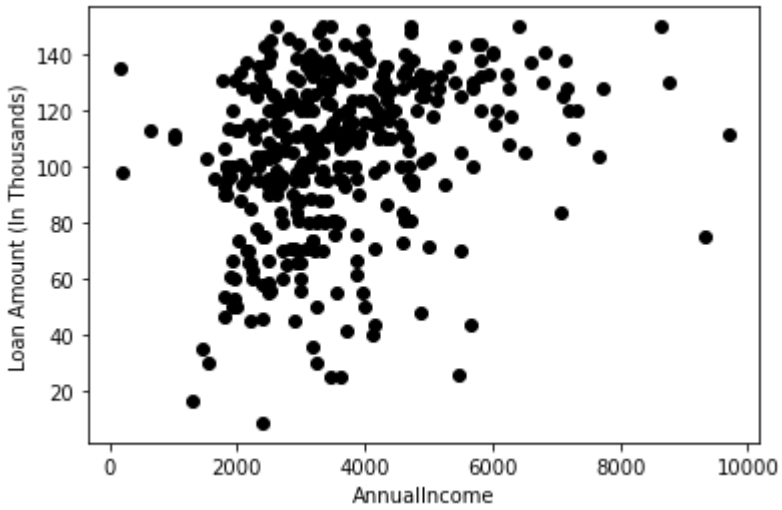
In [1]:
```
#import libraries
import pandas as pd
import numpy as np
import random as rd
import matplotlib.pyplot as plt
```

In [2]:
```
data = pd.read_csv('clustering.csv')
data.head()
```

Out[2]:

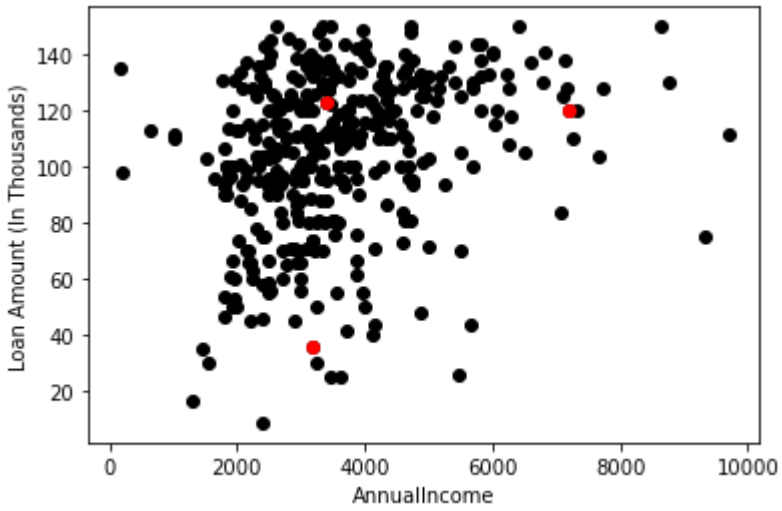
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappli
0	LP001003	Male	Yes	1	Graduate	No	4583	
1	LP001005	Male	Yes	0	Graduate	Yes	3000	
2	LP001006	Male	Yes	0	Not Graduate	No	2583	
3	LP001008	Male	No	0	Graduate	No	6000	
4	LP001013	Male	Yes	0	Not Graduate	No	2333	

In [3]:
```
X = data[["LoanAmount","ApplicantIncome"]]
#Visualise data points
plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')
plt.xlabel('AnnualIncome')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
```



Steps 1 and 2 of K-Means were about choosing the number of clusters (k) and selecting random centroids for each cluster. We will pick 3 clusters and then select random observations from the data as the centroids:

In [4]:
```
K=3

# Select random observation as centroids
Centroids = (X.sample(n=K))
plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')
plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')
plt.xlabel('AnnualIncome')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
```



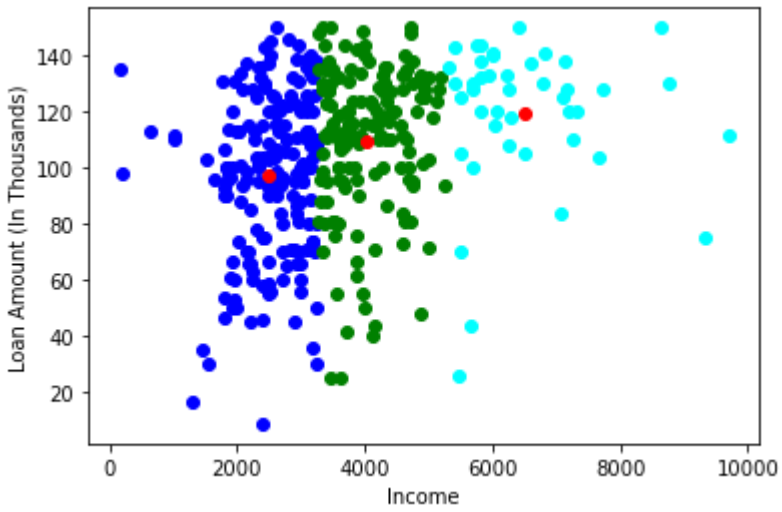
In [5]:
```
# Step 3 - Assign all the points to the closest cluster centroid
# Step 4 - Recompute centroids of newly formed clusters
# Step 5 - Repeat step 3 and 4

diff = 1
j=0

while(diff!=0):
    XD=X
    i=1
    for index1,row_c in Centroids.iterrows():
        ED=[]
        for index2,row_d in XD.iterrows():
            d1=(row_c["ApplicantIncome"]-row_d["ApplicantIncome"])**2
            d2=(row_c["LoanAmount"]-row_d["LoanAmount"])**2
            d=np.sqrt(d1+d2)
            ED.append(d)
        X[i]=ED
        i=i+1

    C=[]
    for index,row in X.iterrows():
        min_dist=row[1]
        pos=1
        for i in range(K):
            if row[i+1] < min_dist:
                min_dist = row[i+1]
                pos=i+1
        C.append(pos)
    X["Cluster"]=C
    Centroids_new = X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantInco
    if j == 0:
        diff=1
        j=j+1
    else:
        diff = (Centroids_new['LoanAmount'] - Centroids['LoanAmount']).sum() +
        print(diff.sum())
    Centroids = X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]
```

```
-9.237706177129652
0.0
```

```
<ipython-input-5-912b3f05973f>:18: 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-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  X[i]=ED
<ipython-input-5-912b3f05973f>:30: 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-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  X["Cluster"]=C
```

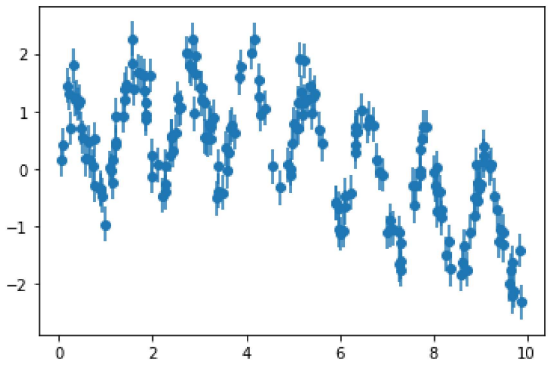
When this difference is 0, we are stopping the training. Let's now visualize the clusters we have got:

In [6]:
```
color=['blue','green','cyan']
for k in range(K):
    data=X[X["Cluster"]==k+1]
    plt.scatter(data["ApplicantIncome"],data["LoanAmount"],c=color[k])
plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')
plt.xlabel('Income')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
```

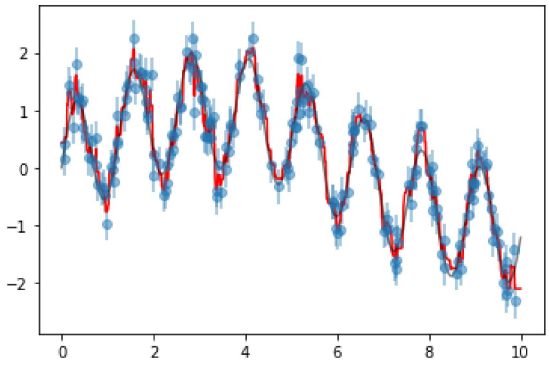


In [1]:
```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import numpy as np
```

In [2]:
```
rng = np.random.RandomState(42)
x = 10 * rng.rand(200)

def model(x, sigma=0.3):
    fast_oscillation = np.sin(5 * x)
    slow_oscillation = np.sin(0.5 * x)
    noise = sigma * rng.randn(len(x))

    return slow_oscillation + fast_oscillation + noise

y = model(x)
plt.errorbar(x, y, 0.3, fmt='o');
```



In [3]:
```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(200)
forest.fit(x[:, None], y)

xfit = np.linspace(0, 10, 1000)
yfit = forest.predict(xfit[:, None])
ytrue = model(xfit, sigma=0)

plt.errorbar(x, y, 0.3, fmt='o', alpha=0.5)
plt.plot(xfit, yfit, '-r');
plt.plot(xfit, ytrue, '-k', alpha=0.5);
```



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

iris = datasets.load_iris()
#print(iris)
X = iris["data"][:, (2, 3)]  # petal length, petal width
#print(X)

y = iris["target"]

setosa_or_versicolor = (y == 0) | (y == 1)
X = X[setosa_or_versicolor]
y = y[setosa_or_versicolor]

# SVM Classifier model
#the hyperparameter control the margin violations
#smaller C leads to more margin violations but wider street
#C can be inferred
svm_clf = SVC(kernel="linear", C=float("inf"))
svm_clf.fit(X, y)

svm_clf.predict([[2.4, 3.1]])

#SVM classifiers do not output a probability like logistic regression classifiers
```

Out[2]:

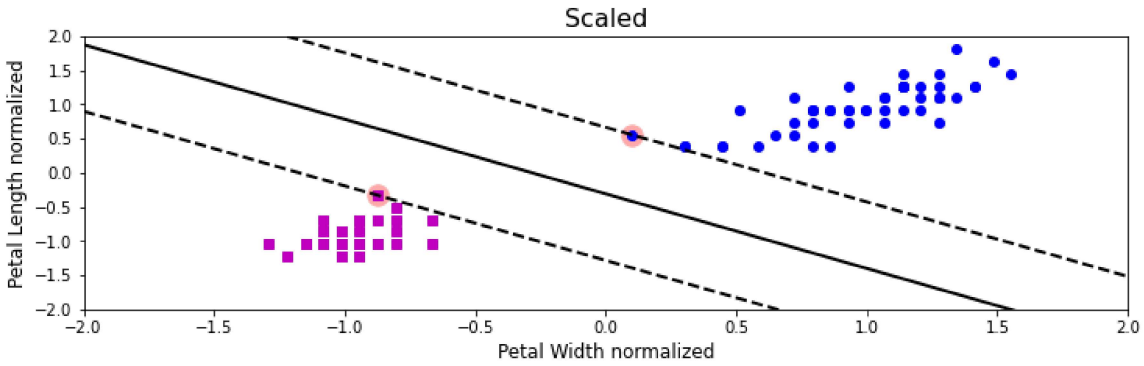
```
array([1])
```

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])
```

Out[3]:

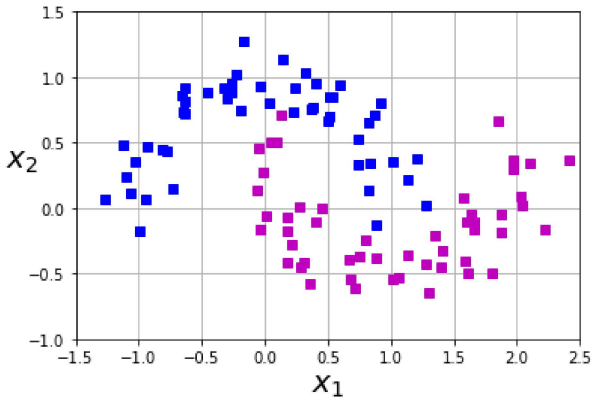
```
(-2.0, 2.0, -2.0, 2.0)
```



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
```

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

In [3]:
```
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 [4]:
```
#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 [5]:
```
#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[5]:
```
Pipeline(steps=[('poly_features', PolynomialFeatures(degree=3)),
                ('scalar', StandardScaler()),
                ('svm_clf', SVC(C=5, coef0=1, degree=10, kernel='poly'))])
```

In [6]:
```
#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()
```