Practical No: 01

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

a) Introduction to SWI- PROLOG Programming with the help of simple programs

Code:

```
parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).
```

Output:

```
?- parent(pam,bob).
true.

?- parent(ann,bob).
false.

?- parent(X,bob).
X = pam .

?- parent(pam,x).
false.

?- parent(pam,X).
X = bob.
```

b) Write a sample program to demonstrate Rules and facts

Code:

```
cat(tom).
cat(tom):- true.
animal(X):- cat(X).
```

Output:

```
animal(X).
X = tom .

?- cat(tom).
true .

?- cat(X).
tom
```

c) Write a sample program to demonstrate the relationship in prolog

Code:

X = Code style

```
X is 2+2.
X = 4.

X is 2*2+3.
X = 7.

X is 22/2-3.
X = 8.
```

P1: B Code

```
parent(z, x).
parent(x, y).
sister(x, y).
female(x).
```

Output:

```
sister(X,Y).
X = x,
Y = y.
```

d) Write a prolog program to demonstrate the use of function

Code:

```
true . %section A
result(rahim,3.6).
result(ajay,3.7).
result(rahul,3.8).
result(saurabh,3.9).

%section B
result(sam,4.9).
result(saurabh,4.1).
result(ram,4.2).
result(priyanka,4.3).

getresult:-
   write("Enter Selection A Student Name: "),
   read(X),
   result(X, Y),
   write("Selection A student result is: "),
   write(Y), nl,
```

```
    write("Enter Selection B Student Name: "),
    read(P),
    result(P, Q),
    write("Selection B student result is: "),
    write(Q), nl,

    compare(Y, Q).

compare(Y, Q):-
    (Y > Q ->
        write("Selection A student is the best");
     Y < Q ->
        write("Selection B student is the best");
     Y =:= Q ->
```

Output:

```
?- getresult.
Enter Selection A Student Name: |: rahim.
Selection A student result is: 3.6
Enter Selection B Student Name: |: ram.
Selection B student result is: 4.2
Selection B student is the best
true.
```

Practical No: 02

Aim: Implementation of Logic programming using PROLOG DFS for water jug problem.

Code:

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

Output:

```
% c:/users/aishwarya chavan/documents/prolog/waterjug problem compiled 0.00 sec, 0 clauses
?- main.
 Water Jug Game
Intial State: 4lit Jug- 0lit
              3lit Jug- 0lit
Final State:  4lit Jug- 2lit
              3lit Jug- 0lit
Follow the Rules:
Rule 1: Fill 4lit Jug
Rule 2: Fill 3lit Jug
Rule 3: Empty 4lit Jug
Rule 4: Empty 3lit Jug
Rule 5: Pour water from 3lit Jug to fill 4lit Jug
Rule 6: Pour water from 4lit Jug to fill 3lit Jug
Rule 7: Pour all of water from 3lit Jug to 4lit Jug
Rule 8: Pour all of water from 4lit Jug to 3lit Jug
4lit Jug:   0 | 3lit Jug:   0
Enter the move::2.

 4lit Jug:   0|  3lit Jug:    3|
 Enter the move::|: 7.
 4lit Jug:   3|  3lit Jug:    0|
 Enter the move::|: 2.
 4lit Jug:   3|  3lit Jug:    3|
 Enter the move::|: 5.
 4lit Jug:   4|  3lit Jug:    2|
 Enter the move::|: 3.
 4lit Jug:   0|  3lit Jug:    2|
 Enter the move::|: 7.
4lit Jug: 2 | 3lit Jug: 0|
~~~~~~~~~~~~~~
Goal Reached! Congrats!!
~~~~~~~~~~~~~~
true .
```

Practical No: 03

Aim: Implementation of Logic programming using PROLOG BFS for tic-tac-toe problem.

Code:

```
play :- my_turn([]).
my_turn(Game) :-
   valid_moves(ValidMoves, Game, x),
   any_valid_moves(ValidMoves, Game).
any_valid_moves([], _) :-
   write('It is a tie'), nl.
any_valid_moves([_|_], Game) :-
   findall(NextMove, game_analysis(x, Game, NextMove), MyMoves),
   do_a_decision(MyMoves, Game).
% This can only fail in the beginning.
do_a_decision(MyMoves, Game) :-
   not(MyMoves = []),
   length(MyMoves, MaxMove),
   random(0, MaxMove, ChosenMove),
   nth0(ChosenMove, MyMoves, X),
   NextGame = [X | Game],
   print_game(NextGame),
   (victory_condition(x, NextGame) ->
      (write('I won. You lose.'), nl);
      your_turn(NextGame), !).
      your_turn(Game) :-
   valid_moves(ValidMoves, Game, o),
   (ValidMoves = [] -> (write('It is a tie'), nl);
    (write('Available moves:'), write(ValidMoves), nl,
     ask_move(Y, ValidMoves),
     NextGame = [Y | Game],
     (victory_condition(o, NextGame) ->
      (write('I lose. You win.'), nl);
      my_turn(NextGame), !))).
ask_move(Move, ValidMoves) :-
   write('Give your move:'), nl,
   read(Move), member(Move, ValidMoves), !.
ask_move(Y, ValidMoves) :-
   write('not a move'), nl,
   ask_move(Y, ValidMoves).
movement_prompt(X, Y, ValidMoves) :-
   write('Give your X:'), nl, read(X), member(move(o, X, Y), ValidMoves), !,
   write('Give your Y:'), nl, read(Y), member(move(o, X, Y), ValidMoves).
% A routine for printing games.. Well you can use it.
print_game(Game) :-
   plot_row(0, Game), plot_row(1, Game), plot_row(2, Game).
```

```
plot_row(Y, Game) :-
   plot(Game, 0, Y), plot(Game, 1, Y), plot(Game, 2, Y), nl.
plot(Game, X, Y) :-
   (member(move(P, X, Y), Game), ground(P)) -> write(P) ; write('.').
% This system determines whether there's a perfect play available.
game_analysis(_, Game, _) :-
   victory_condition(Winner, Game),
   Winner = x. % We do not want to lose.
   % Winner = o. % We do not want to win. (egostroking mode).
   % true. % If you remove this constraint entirely, it may let you win.
game_analysis(Turn, Game, NextMove) :-
   not(victory_condition(_, Game)),
   game_analysis_continue(Turn, Game, NextMove).
game_analysis_continue(Turn, Game, NextMove) :-
   valid_moves(Moves, Game, Turn),
   game_analysis_search(Moves, Turn, Game, NextMove).
% Comment these away and the system refuses to play,
% because there are no ways to play this without a possibility of tie.
game_analysis_search([], o, _, _). % Tie on opponent's turn.
game_analysis_search([], x, _, _). % Tie on our turn.
game_analysis_search([X|Z], o, Game, NextMove) :- % Whatever opponent does,
   NextGame = [X | Game],                       % we desire not to lose.
   game_analysis_search(Z, o, Game, NextMove),
   game_analysis(x, NextGame, _), !.
game_analysis_search(Moves, x, Game, NextMove) :-
   game_analysis_search_x(Moves, Game, NextMove).
game_analysis_search_x([X|_], Game, X) :-
   NextGame = [X | Game],
   game_analysis(o, NextGame, _).
game_analysis_search_x([_|Z], Game, NextMove) :-
   game_analysis_search_x(Z, Game, NextMove).
% This thing describes all kinds of valid games.
valid_game(Turn, Game, LastGame, Result) :-
   victory_condition(Winner, Game) ->
      (Game = LastGame, Result = win(Winner)) ;
      valid_continuing_game(Turn, Game, LastGame, Result).
valid_continuing_game(Turn, Game, LastGame, Result) :-
   valid_moves(Moves, Game, Turn),
   tie_or_next_game(Moves, Turn, Game, LastGame, Result).
tie_or_next_game([], _, Game, Game, tie).
tie_or_next_game(Moves, Turn, Game, LastGame, Result) :-
   valid_gameplay_move(Moves, NextGame, Game),
   opponent(Turn, NextTurn),
   valid_game(NextTurn, NextGame, LastGame, Result).
% Victory conditions for tic tac toe.
victory(P, Game, Begin) :-
   valid_gameplay(Game, Begin),
   victory_condition(P, Game).
victory_condition(P, Game) :-
```

```
    (X = 0; X = 1; X = 2),
    member(move(P, X, 0), Game),
    member(move(P, X, 1), Game),
    member(move(P, X, 2), Game).
victory_condition(P, Game) :-
    (Y = 0; Y = 1; Y = 2),
    member(move(P, 0, Y), Game),
    member(move(P, 1, Y), Game),
    member(move(P, 2, Y), Game).
victory_condition(P, Game) :-
    member(move(P, 0, 2), Game),
    member(move(P, 1, 1), Game),
    member(move(P, 2, 0), Game).
victory_condition(P, Game) :-
    member(move(P, 0, 0), Game),
    member(move(P, 1, 1), Game),
    member(move(P, 2, 2), Game).
% This describes a valid form of gameplay.
% Which player did the move is disregarded.
valid_gameplay(Start, Start).
valid_gameplay(Game, Start) :-
    valid_gameplay(PreviousGame, Start),
    valid_moves(Moves, PreviousGame, _),
    valid_gameplay_move(Moves, Game, PreviousGame).
valid_gameplay_move([X|_], [X|PreviousGame], PreviousGame).
valid_gameplay_move([_|Z], Game, PreviousGame) :-
    valid_gameplay_move(Z, Game, PreviousGame).
% The set of valid moves must not be affected by the decision making
% of the prolog interpreter.
% Therefore we have to retrieve them like this.
% This is equivalent to the (∀x∈0..2)(∀y∈0..2)(....
% uh wait.. There's no way to represent this using those quantifiers.
valid_moves(Moves, Game, Turn) :-
    valid_moves_column(0, M1,    [], Game, Turn),
    valid_moves_column(1, M2,    M1, Game, Turn),
    valid_moves_column(2, Moves, M2, Game, Turn).
valid_moves_column(X, M3, M0, Game, Turn) :-
    valid_moves_cell(X, 0, M1, M0, Game, Turn),
    valid_moves_cell(X, 1, M2, M1, Game, Turn),
    valid_moves_cell(X, 2, M3, M2, Game, Turn).
valid_moves_cell(X, Y, M1, M0, Game, Turn) :-
    member(move(_, X, Y), Game) -> M0 = M1 ; M1 = [move(Turn,X,Y) | M0].
% valid_move(X, Y, Game) :-
%    (X = 0; X = 1; X = 2),
%    (Y = 0; Y = 1; Y = 2),
%    not(member(move(_, X, Y), Game)).
opponent(x, o).
opponent(o, x).
```

Output:

```
% c:/users/aishwarya chavan/documents/prolog/tictactoe compiled 0.02 sec, -1 clauses
?- play.
...
...
x..
Available moves:[move(o,2,2),move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,1),move(o,1,0),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,1,2).
...
...
xox
Available moves:[move(o,2,1),move(o,2,0),move(o,1,1),move(o,1,0),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,1,1).
.x.
.o.
xox
Available moves:[move(o,2,1),move(o,2,0),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,2,0).
.xo
.ox
xox
Available moves:[move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,0,0).
oxo
xox
xox
It is a tie
true.
```

Practical No: 04

Aim: Implementation of Logic programming using PROLOG Hill-climbing to solve 8- Puzzle Problem.

Code:

```
% Simple Prolog Planner for the 8 Puzzle Problem

% This predicate initialises the problem states. The first argument
% of solve/3 is the initial state, the 2nd the goal state, and the
% third the plan that will be produced.

test(Plan):-
write(&#39;Initial state:&#39;),nl,
Init= [at(tile4,1), at(tile3,2), at(tile8,3), at(empty,4), at(tile2,5), at(tile6,6), at(tile5,7),
at(tile1,8), at(tile7,9)],
write_sol(Init),
Goal= [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4), at(empty,5), at(tile5,6), at(tile6,7),
at(tile7,8), at(tile8,9)],
nl,write(&#39;Goal state:&#39;),nl,
write(Goal),nl,nl,
solve(Init,Goal,Plan).

solve(State, Goal, Plan):-
solve(State, Goal, [], Plan).

%Determines whether Current and Destination tiles are a valid move.
is_movable(X1,Y1) :- (1 is X1 - Y1) ; (-1 is X1 - Y1) ; (3 is X1 - Y1) ; (-3 is X1 - Y1).

% This predicate produces the plan. Once the Goal list is a subset
% of the current State the plan is complete and it is written to
% the screen using write_sol/1.

solve(State, Goal, Plan, Plan):-
is_subset(Goal, State), nl,
write_sol(Plan).

solve(State, Goal, Sofar, Plan):-
act(Action, Preconditions, Delete, Add),
is_subset(Preconditions, State),
\+ member(Action, Sofar),
delete_list(Delete, State, Remainder),
append(Add, Remainder, NewState),
solve(NewState, Goal, [Action|Sofar], Plan).

% The problem has three operators.
% 1st arg = name
% 2nd arg = preconditions
% 3rd arg = delete list
% 4th arg = add list.

% Tile can move to new position only if the destination tile is empty &amp; Manhattan distance = 1
```

```
act(move(X,Y,Z),
[at(X,Y), at(empty,Z), is_movable(Y,Z)],
[at(X,Y), at(empty,Z)],
[at(X,Z), at(empty,Y)]).

% Utility predicates.

% Check is first list is a subset of the second
is_subset([H|T], Set):-
member(H, Set),
is_subset(T, Set).
is_subset([], _).

% Remove all elements of 1st list from second to create third.

delete_list([H|T], Curstate, Newstate):-
remove(H, Curstate, Remainder),
delete_list(T, Remainder, Newstate).
delete_list([], Curstate, Curstate).

remove(X, [X|T], T).
remove(X, [H|T], [H|R]):-
remove(X, T, R).

write_sol([]).
write_sol([H|T]):-
write_sol(T),
write(H), nl.

append([H|T], L1, [H|L2]):-
append(T, L1, L2).
append([], L, L).

member(X, [X|_]).
member(X, [_|T]):-
member(X, T).
```

Output:

```
?- test(Plan).
Initial state:
at(tile7,9)
at(tile1,8)
at(tile5,7)
at(tile6,6)
at(tile2,5)
at(empty,4)
at(tile8,3)
at(tile3,2)
at(tile4,1)

Goal state:
[at(tile1,1),at(tile2,2),at(tile3,3),at(tile4,4),at(empty,5),at(tile5,6),at(tile
6,7),at(tile7,8),at(tile8,9)]

false.

?-
```

Practical No: 05

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

1. **NumPy (Numerical Python)**
    - Core library for numerical computations.
    - Used for handling large multidimensional arrays and matrices.
    - Provides mathematical functions and operations.

Code:

```
import numpy as np
 # Creating a NumPy
 array arr = np.array([1, 2, 3, 4, 5])
print("Array:", arr)
print("Type:", type(arr))
```

Output:

```
Array: [1 2 3 4 5]
Type: <class 'numpy.ndarray'>
```

2. **Pandas**
   Theory:
    - Used for data manipulation and analysis.
    - Supports structures like Series (1D) and DataFrame (2D table).
    - Built on top of NumPy

Code:

```
import pandas as pd
# Creating a DataFrame
data = {'Name': ['Riza', 'Aasiya'], 'Marks': [85, 90]}
df = pd.DataFrame(data)
print(df)
```

Output:

```
     Name  Marks
0    Riza     85
1  Aasiya     90
```

3. SciPy (Scientific Python)

Theory:

- Built on NumPy, used for scientific and technical computing.
- Includes modules for optimization, integration, statistics, and more.

Code:

```
from scipy import stats
# Finding mean and mode using SciPy
data = [1, 2, 2, 3, 4]
print("Mean:", stats.tmean(data))
print("Mode:", stats.mode(data))
```

Output:

```
Mean: 2.4
Mode: ModeResult(mode=2, count=2)
```

4. Matplotlib

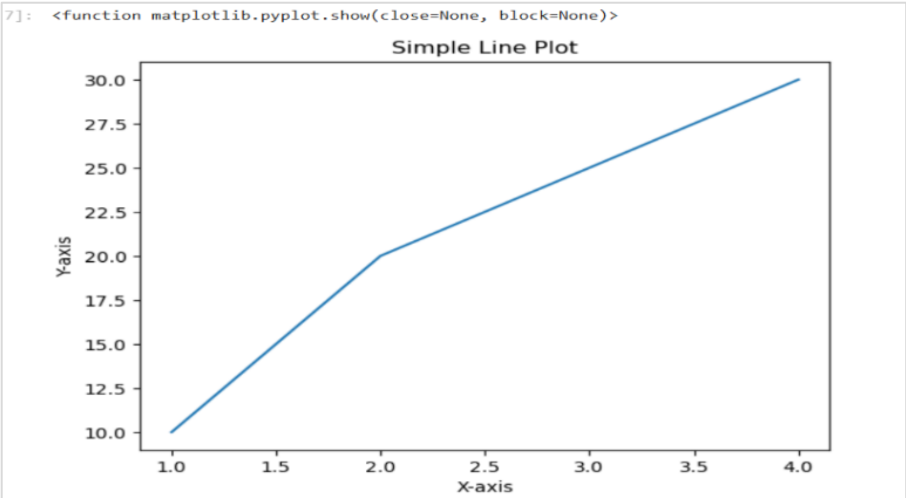
Theory:

- Used for plotting graphs and visualizing data.
- pyplot module is commonly used like MATLAB

Code:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.title("Simple Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show
```

Output:



5. SCIKIT LEARN

Theory:

- It is mainly used in machine learning.
- It has lot of statistics related tools.
- It is open source.
- By using the Scikit library the efficiency will improve tremendously as it is quite accurate.

Code:

```
pip install scikit-learn
from sklearn.datasets import load_iris
iris = load_iris()
A= iris.data
y = iris.target
21
feature_names = iris.feature_names
target_names = iris.target_names
print("Feature names:", feature_names)
print("Target names:", target_names)
print("\nFirst 10 rows of A:\n", A[:10])
```

Output:

```
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']

First 10 rows of A:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

Practical No: 06

Aim: Implement Perceptron algorithm for OR operation.

Code:

```
# importing Python library
import numpy as np

# define Unit Step Function
def unitStep(v):
	if v >= 0:
		return 1
	else:
		return 0

# design Perceptron Model
def perceptronModel(x, w, b):
	v = np.dot(w, x) + b
	y = unitStep(v)
	return y

# OR Logic Function
# w1 = 1, w2 = 1, b = -0.5
def OR_logicFunction(x):
	w = np.array([1, 1])
	b = -0.5
	return perceptronModel(x, w, b)

# testing the Perceptron Model
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("OR({}, {}) = {}".format(0, 1, OR_logicFunction(test1)))
print("OR({}, {}) = {}".format(1, 1, OR_logicFunction(test2)))
print("OR({}, {}) = {}".format(0, 0, OR_logicFunction(test3)))
print("OR({}, {}) = {}".format(1, 0, OR_logicFunction(test4)))
```

Output:

```
OR(0, 1) = 1
OR(1, 1) = 1
OR(0, 0) = 0
OR(1, 0) = 1
```

Practical No: 07

Aim: Improve the prediction accuracy by estimating the weight values for the training data using stochastic gradient descent (Perceptron)

Code:

```
import numpy as np

def perceptron_sgd(X, y, learning_rate=0.01, epochs=100):
    # Initialize weights and bias
    weights = np.zeros(X.shape[1])
    bias = 0

    for epoch in range(epochs):
        for i in range(len(X)):
            # Predict the label
            y_pred = np.sign(np.dot(X[i], weights) + bias)

            # Update weights if there's a misclassification
            if y_pred != y[i]:
                weights += learning_rate * y[i] * X[i]
                bias += learning_rate * y[i]

    return weights, bias

# Example usage:
X = np.array([[2, 3], [1, -1], [-1, -2], [-2, 1]])  # Training data
y = np.array([1, -1, -1, 1])  # Labels (binary classification)
weights, bias = perceptron_sgd(X, y)
print("Weights:", weights)
print("Bias:", bias)
```

Output:

```
Weights: [0.01 0.04]
Bias: 0.0
```

Practical No: 08

Aim: Implement Adaline algorithm for AND operation.

Code:

```
import numpy as np

# Define the activation function (linear for Adaline)
def activation_function(x):
    return x

# Adaline algorithm for AND operation
def adaline_and_operation():
    # Input data (AND truth table)
    inputs = np.array( [
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1]
    ] )

    # Target outputs
    targets = np.array([0, 0, 0, 1])

    # Initialize weights and bias
    weights = np.random.rand(2)  # Random values between 0 and 1
    bias = np.random.rand(1)     # Random values between 0 and 1

    # Learning rate
    lr = 0.1

    # Number of epochs
    epochs = 1000

    # Training loop
    for epoch in range(epochs):
        for i in range(len(inputs)):
            # Compute the net input
            net_input = np.dot(inputs[i], weights) + bias

            # Compute the output
            output = activation_function(net_input)

            # Calculate the error
            error = targets[i] - output
```

```
            # Update weights and bias
            weights += lr * error * inputs[i]
            bias += lr * erro

    print("Trained weights:", weights)
    print("Trained bias:", bias)

    # Test the model
    print("\nTesting the trained model:")
    for input_data in inputs:
        net_input = np.dot(input_data, weights) + bias
        output = activation_function(net_input)
        print(f"Input: {input_data}, Output: {output}")

# Call the function
adaline_and_operation()
```

Output:

```
Trained weights: [0.55555556 0.52777778]
Trained bias: [-0.27777778]

Testing the trained model:
Input: [0 0], Output: [-0.27777778]
Input: [0 1], Output: [0.25]
Input: [1 0], Output: [0.27777778]
Input: [1 1], Output: [0.80555556]
```

Practical No: 09

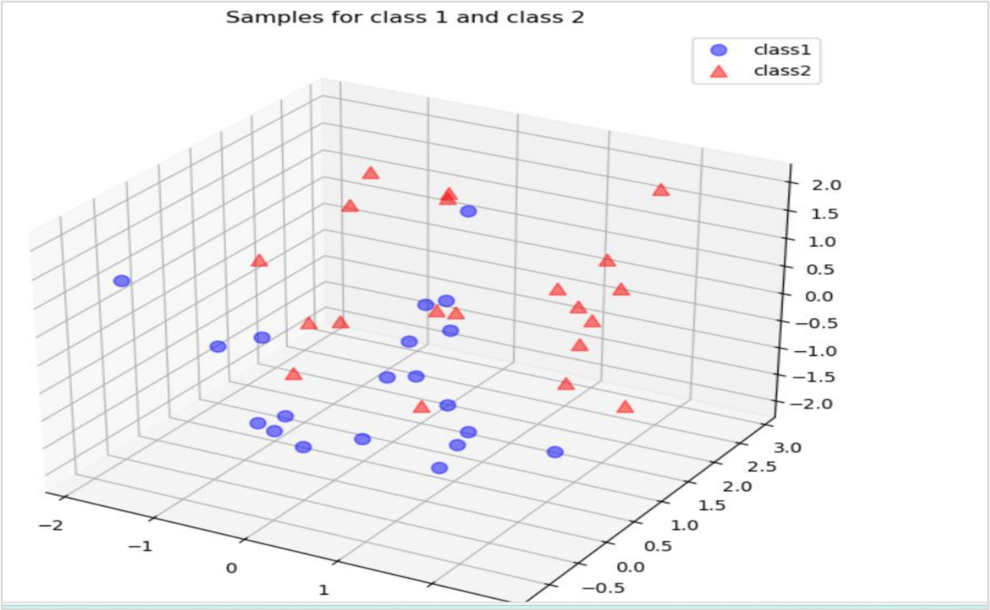
Aim: Implementation of Features Extraction and Selection, Normalization, Transformation, Principal Components Analysis.

Extraction and Selection

Code: Selection

```
import numpy as np
#np.random.seed(234234782384239784) # random seed for consistency
# A reader pointed out that Python 2.7 would raise a
# "ValueError: object of too small depth for desired array".
# This can be avoided by choosing a smaller random seed, e.g. 1
# or by completely omitting this line, since I just used the random seed for
# consistency.

mu_vec1 = np.array([0,0,0])
cov_mat1 = np.array([[1,0,0],[0,1,0],[0,0,1]])
class1_sample = np.random.multivariate_normal(mu_vec1, cov_mat1, 20).T
assert class1_sample.shape == (3,20), "The matrix has not the dimensions 3x20"

mu_vec2 = np.array([1,1,1])
cov_mat2 = np.array([[1,0,0],[0,1,0],[0,0,1]])
class2_sample = np.random.multivariate_normal(mu_vec2, cov_mat2, 20).T
assert class2_sample.shape == (3,20), "The matrix has not the dimensions 3x20"
%pylab inline
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import proj3d

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
plt.rcParams['legend.fontsize'] = 10
ax.plot(class1_sample[0,:], class1_sample[1,:], class1_sample[2,:], 'o', markersize=8, color='blue',
alpha=0.5, label='class1')
ax.plot(class2_sample[0,:], class2_sample[1,:], class2_sample[2,:], '^', markersize=8, alpha=0.5,
color='red', label='class2')

plt.title('Samples for class 1 and class 2')
ax.legend(loc='upper right')

plt.show()
```

Output:



Code:

```
l_samples = np.concatenate((class1_sample, class2_sample), axis=1)
assert all_samples.shape == (3,40), "The matrix has not the dimensions 3x40"
mean_x = np.mean(all_samples[0,:])
mean_y = np.mean(all_samples[1,:])
mean_z = np.mean(all_samples[2,:])

mean_vector = np.array([[mean_x],[mean_y],[mean_z]])
```

Output:

```
Scatter Matrix:
 [[33.71266861 10.08651371  8.12777968]
 [10.08651371 48.14712763 18.31418846]
 [ 8.12777968 18.31418846 53.04123406]]
```

Code:

```
cov_mat = np.cov([all_samples[0,:],all_samples[1,:],all_samples[2,:]])
print('Covariance Matrix:\n', cov_mat)
```

Output:

```
Covariance Matrix:
 [[0.8644274  0.25862856 0.20840461]
 [0.25862856 1.23454173 0.46959458]
 [0.20840461 0.46959458 1.36003164]]
```

Code:

```
# eigenvectors and eigenvalues for the from the scatter matrix
eig_val_sc, eig_vec_sc = np.linalg.eig(scatter_matrix)

# eigenvectors and eigenvalues for the from the covariance matrix
eig_val_cov, eig_vec_cov = np.linalg.eig(cov_mat)

for i in range(len(eig_val_sc)):
    eigvec_sc = eig_vec_sc[:,i].reshape(1,3).T
    eigvec_cov = eig_vec_cov[:,i].reshape(1,3).T
    assert eigvec_sc.all() == eigvec_cov.all(), 'Eigenvectors are not identical'

    print('Eigenvector {}: \n{}'.format(i+1, eigvec_sc))
    print('Eigenvalue {} from scatter matrix: {}'.format(i+1, eig_val_sc[i]))
    print('Eigenvalue {} from covariance matrix: {}'.format(i+1, eig_val_cov[i]))
    print('Scaling factor: ', eig_val_sc[i]/eig_val_cov[i])
    print(40 * '-')
```

Output:

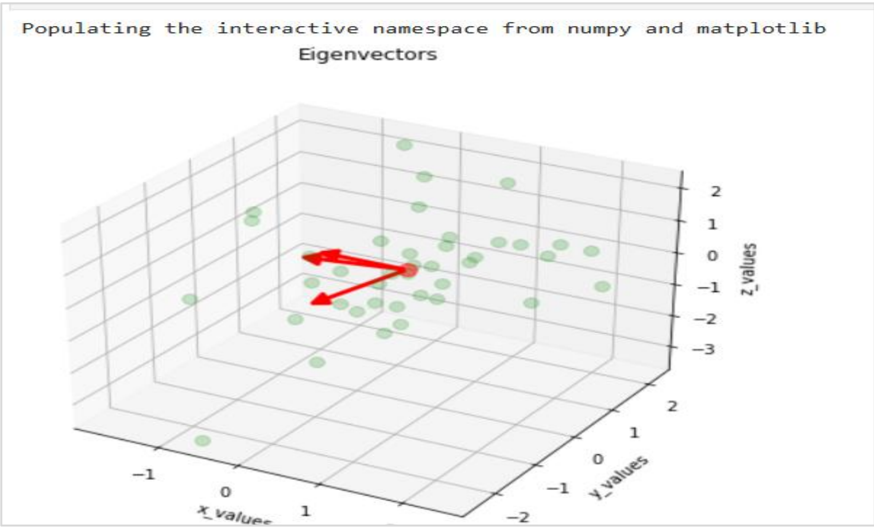
```
Eigenvector 1:
[[-0.30824235]
 [-0.63907963]
 [-0.70467289]]
Eigenvalue 1 from scatter matrix: 73.20598118090528
Eigenvalue 1 from covariance matrix: 1.8770764405360343
Scaling factor:  38.99999999999997
----------------------------------------
Eigenvector 2:
[[-0.82885341]
 [ 0.54396773]
 [-0.13077128]]
Eigenvalue 2 from scatter matrix: 28.37534621609244
Eigenvalue 2 from covariance matrix: 0.7275729798998062
Scaling factor:  39.0
----------------------------------------
Eigenvector 3:
[[-0.46689258]
 [-0.54376128]
 [ 0.69737722]]
Eigenvalue 3 from scatter matrix: 33.31970289768161
Eigenvalue 3 from covariance matrix: 0.8543513563508106
Scaling factor:  38.99999999999999
----------------------------------------
```

Code:

```
 for i in range(len(eig_val_sc)):
    eigv = eig_vec_sc[:,i].reshape(1,3).T
    np.testing.assert_array_almost_equal(scatter_matrix.dot(eigv), eig_val_sc[i] * eigv,
                                         decimal=6, err_msg='', verbose=True)
%pylab inline

from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
from mpl_toolkits.mplot3d import proj3d
from matplotlib.patches import FancyArrowPatch


class Arrow3D(FancyArrowPatch):
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)

fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(111, projection='3d')

ax.plot(all_samples[0,:], all_samples[1,:], all_samples[2,:], 'o', markersize=8, color='green', alpha=0.2)
ax.plot([mean_x], [mean_y], [mean_z], 'o', markersize=10, color='red', alpha=0.5)
for v in eig_vec_sc.T:
    a = Arrow3D([mean_x, v[0]], [mean_y, v[1]], [mean_z, v[2]], mutation_scale=20, lw=3,
arrowstyle="-|>", color="r")
    ax.add_artist(a)
ax.set_xlabel('x_values')
ax.set_ylabel('y_values')
ax.set_zlabel('z_values')

plt.title('Eigenvectors')

plt.show()
```

Output:



Code:

```
  for ev in eig_vec_sc:
    numpy.testing.assert_array_almost_equal(1.0, np.linalg.norm(ev))
    # instead of 'assert' because of rounding errors
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_val_sc[i]), eig_vec_sc[:,i]) for i in range(len(eig_val_sc))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)

# Visually confirm that the list is correctly sorted by decreasing eigenvalues
for i in eig_pairs:
    print(i[0])
```

Output:

```
73.20598118090528
33.31970289768161
28.37534621609244
```

Code:

```
matrix_w = np.hstack((eig_pairs[0][1].reshape(3,1), eig_pairs[1][1].reshape(3,1)))
print('Matrix W:\n', matrix_w)
```

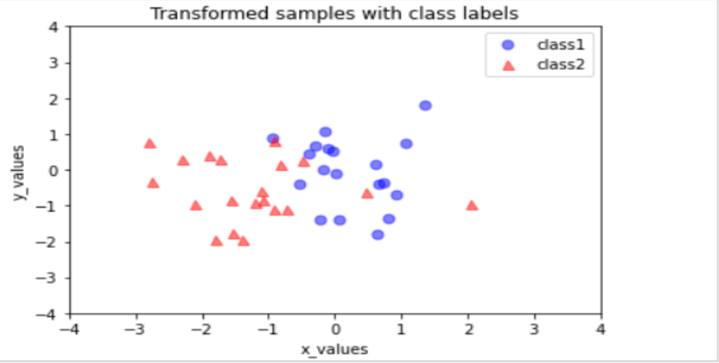
Output:

```
Matrix W:
 [[-0.30824235 -0.46689258]
 [-0.63907963 -0.54376128]
 [-0.70467289  0.69737722]]
```

Code:

```
transformed = matrix_w.T.dot(all_samples)
assert transformed.shape == (2,40), "The matrix is not 2x40 dimensional."
plt.plot(transformed[0,0:20], transformed[1,0:20], 'o', markersize=7, color='blue', alpha=0.5,
label='class1')
plt.plot(transformed[0,20:40], transformed[1,20:40], '^', markersize=7, color='red', alpha=0.5,
label='class2')
plt.xlim([-4,4])
plt.ylim([-4,4])
plt.xlabel('x_values')
plt.ylabel('y_values')
plt.legend()
plt.title('Transformed samples with class labels')
plt.show()
```

Output:



Code: Extraction

```
# Import packages
import numpy as np
from sklearn import decomposition, datasets
from sklearn.preprocessing import StandardScaler
# Load the breast cancer dataset
dataset = datasets.load_breast_cancer()
# Load the features
X = dataset.data
# View the shape of the dataset
X.shape
```

Output:

```
(569, 30)
```

Code:

```
# View the data
X
```

Output:

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

Code:

```
X_std_pca.shape# Create a scaler object
sc = StandardScaler()

# Fit the scaler to the features and transform
X_std = sc.fit_transform(X)
# Create a pca object with the 2 components as a parameter
pca = decomposition.PCA(n_components=2)

# Fit the PCA and transform the data
X_std_pca = pca.fit_transform(X_std)
# View the new feature data's shape
```

Output:

```
(569, 2)
```

Code:

```
# View the new feature data
X_std_pca
```

Output:

```
array([[ 9.19283683,  1.94858307],
       [ 2.3878018 , -3.76817174],
       [ 5.73389628, -1.0751738 ],
       ...,
       [ 1.25617928, -1.90229671],
       [10.37479406,  1.67201011],
       [-5.4752433 , -0.67063679]])
```

Normalization, Transformation

Code:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load dataset
data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
#Univariate Selection in Python with Scikit-Learn
from sklearn.feature_selection import SelectKBest, chi2
# Apply SelectKBest with chi2
select_k_best = SelectKBest(score_func=chi2, k=2)
X_train_k_best = select_k_best.fit_transform(X_train, y_train)

print("Selected features:", X_train.columns[select_k_best.get_support()])
```

Output:

```
Selected features: Index(['petal length (cm)', 'petal width (cm)'], dtype='object')
```

Code:

```
#Recursive Feature Elimination
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# Apply RFE with logistic regression
model = LogisticRegression()
rfe = RFE(model, n_features_to_select=2)
X_train_rfe = rfe.fit_transform(X_train, y_train)

print("Selected features:", X_train.columns[rfe.get_support()])
```

Output:

```
Selected features: Index(['petal length (cm)', 'petal width (cm)'], dtype='object')
```

Practical No: 10

Aim: Implementation of Logistic regression.

Code:

```
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')
# 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()
```

Output:

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

Code:

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

Output:

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

Code:

```
# check missing values in train data
train_df.isnull().sum()
```

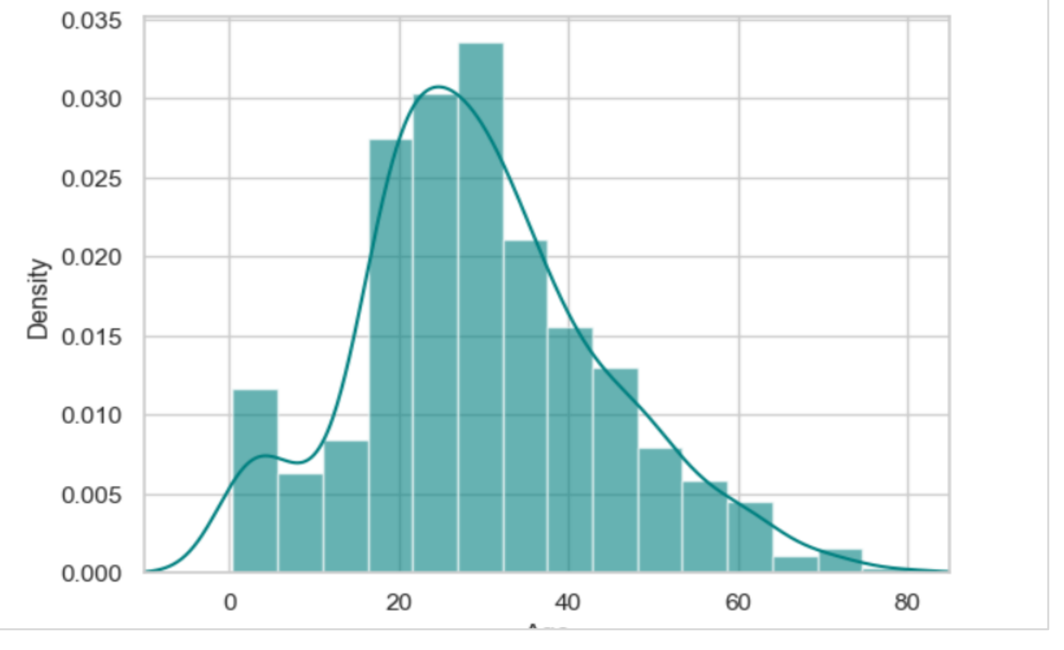
Output:

```
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
```

Code:

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

Output:



Code:

```
# 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)))
```

Output:

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

Code:

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

Output:

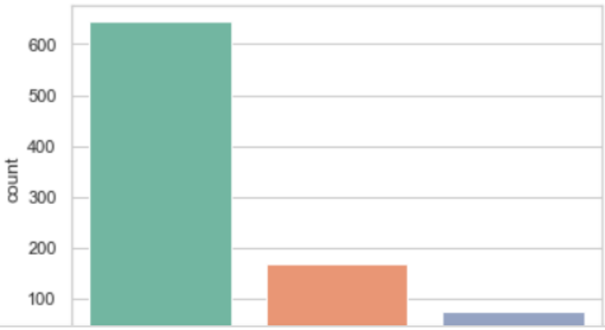
```
Percent of missing "Cabin" records is 77.10%
```

Code:

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

Output:

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



Code:

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

Output:

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

Code:

```
train_data = train_df.copy()
train_data["Age"].fillna(train_df["Age"].median(skipna=True), inplace=True)
train_data["Embarked"].fillna(train_df['Embarked'].value_counts().idxmax(), inplace=True)
train_data.drop('Cabin', axis=1, inplace=True)
# check missing values in adjusted train data
train_data.isnull().sum()
```

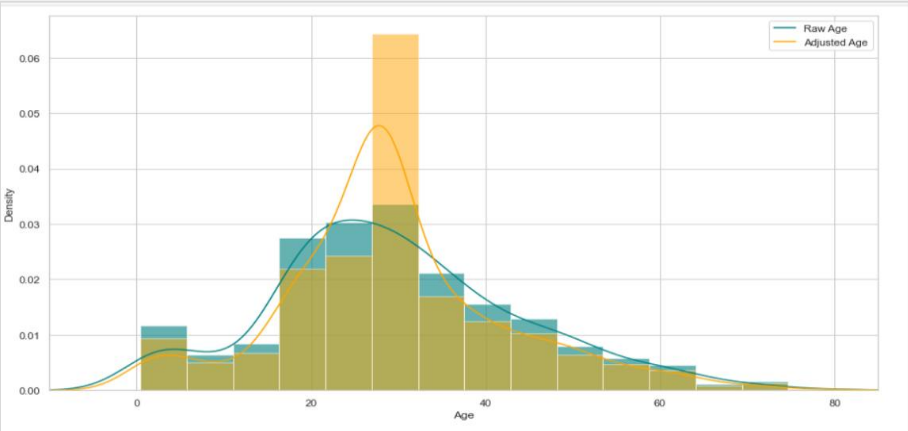
Output:

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

Code:

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

Output:



Code:

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

Output:

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

Code:

```
test_df.isnull().sum()
```

Output:

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

Code:

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

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

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

final_test = testing
final_test.head()
```

Output:

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

Code:

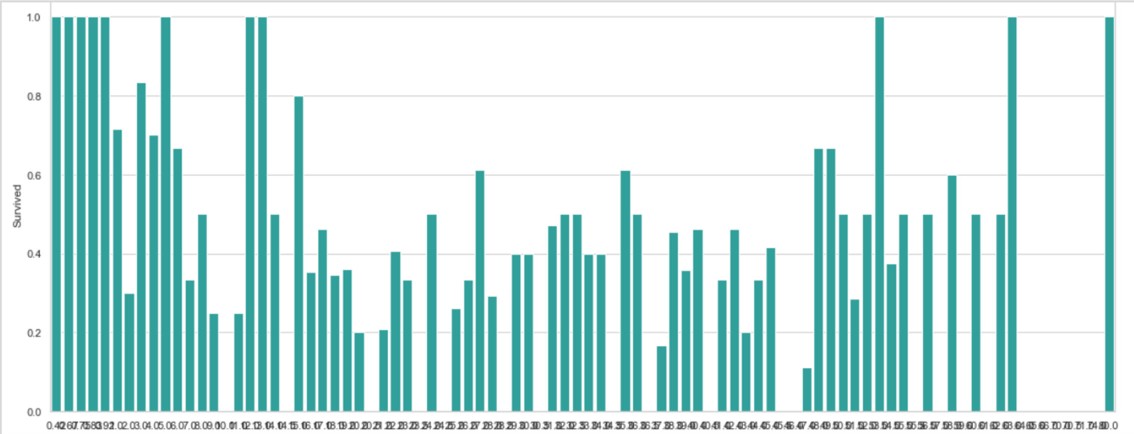
```
plt.figure(figsize=(15,8))
ax = sns.kdeplot(final_train["Age"][final_train.Survived == 1], color="darkturquoise", shade=True)
sns.kdeplot(final_train["Age"][final_train.Survived == 0], color="lightcoral", shade=True)
plt.legend(['Survived', 'Died'])
plt.title('Density Plot of Age for Surviving Population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

Output:

Code:

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

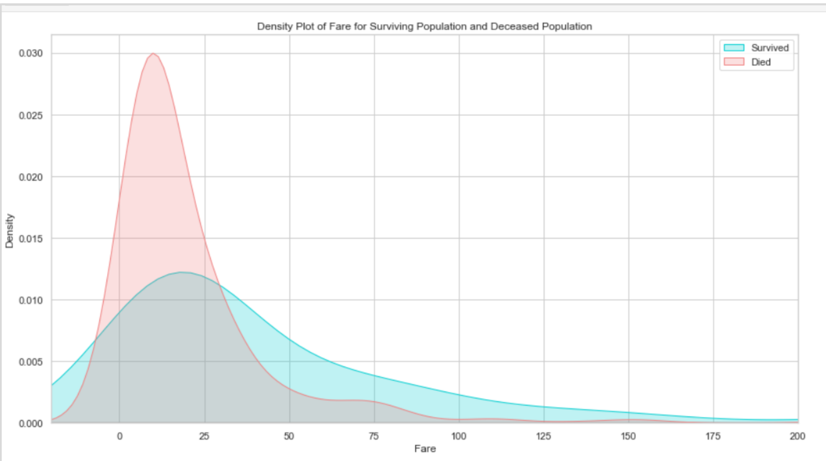
Output:



Code:

```
final_train['IsMinor']=np.where(final_train['Age']<=16, 1, 0)
final_test['IsMinor']=np.where(final_test['Age']<=16, 1, 0)
#Exploration of Fare
plt.figure(figsize=(15,8))
ax = sns.kdeplot(final_train["Fare"][final_train.Survived == 1], color="darkturquoise", shade=True)
sns.kdeplot(final_train["Fare"][final_train.Survived == 0], color="lightcoral", shade=True)
plt.legend(['Survived', 'Died'])
plt.title('Density Plot of Fare for Surviving Population and Deceased Population')
ax.set(xlabel='Fare')
plt.xlim(-20,200)
plt.show()
```

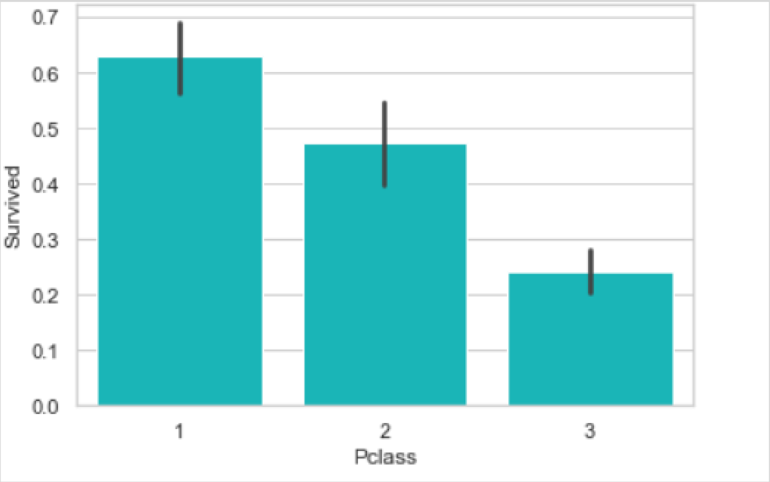
Output:



Code:

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

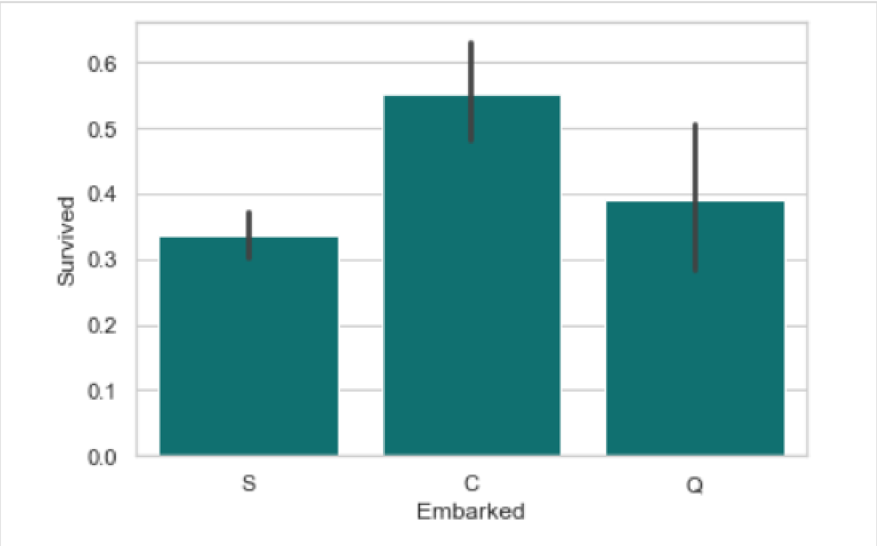
Output:



Code:

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

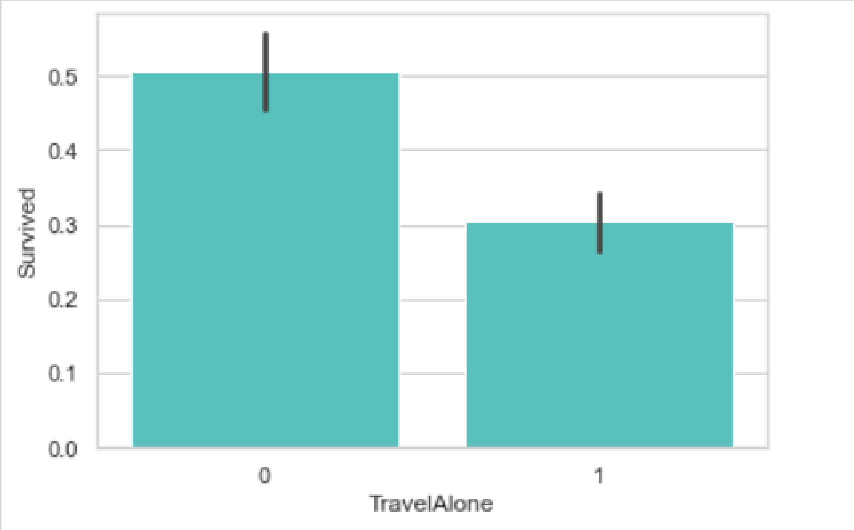
Output:



Code:

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

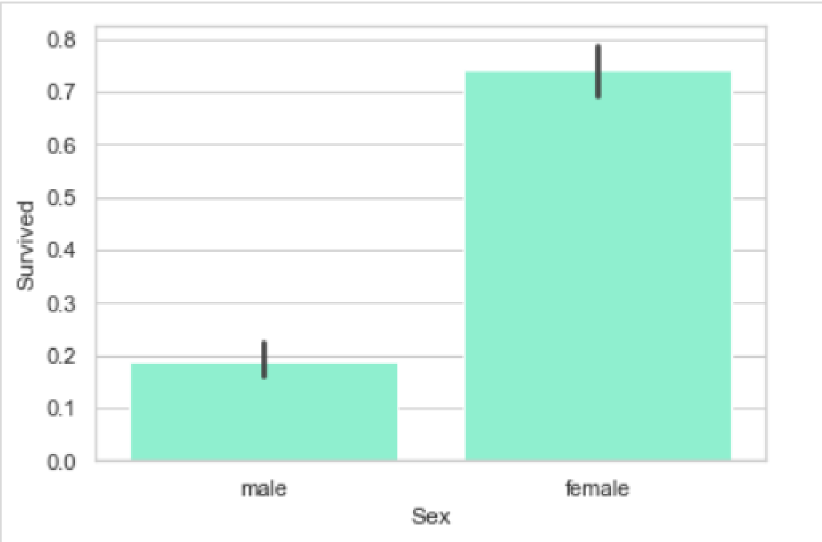
Output:



Code:

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

Output:



Code:

```
#Logistic Regression and Results
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

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

```
y = final_train['Survived']
# Build a logreg and compute the feature importances
model = LogisticRegression()
# create the RFE model and select 8 attributes
rfe = RFE(model, 8)
rfe = rfe.fit(X, y)
# summarize the selection of the attributes
print('Selected features: %s' % list(X.columns[rfe.support_]))
```

Output:

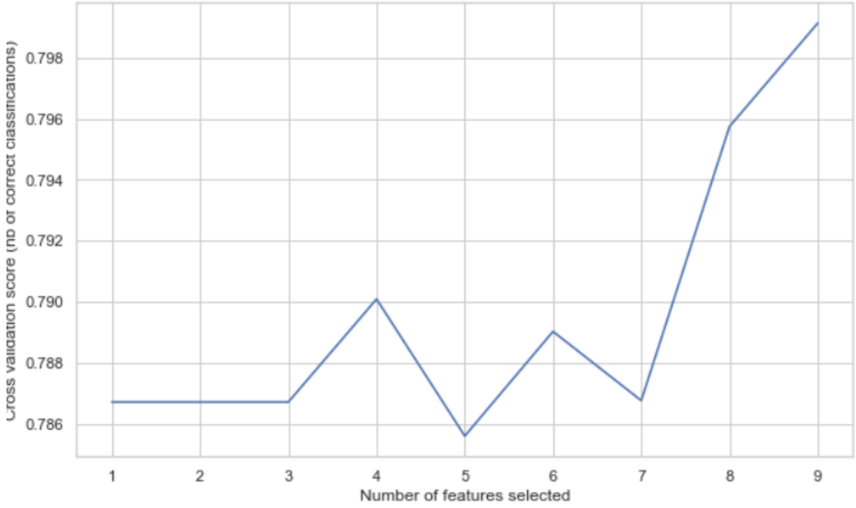
```
Selected features: ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C', 'Embarked_S', 'Sex_male', 'IsMinor']
```

Code:

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

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

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

Output:

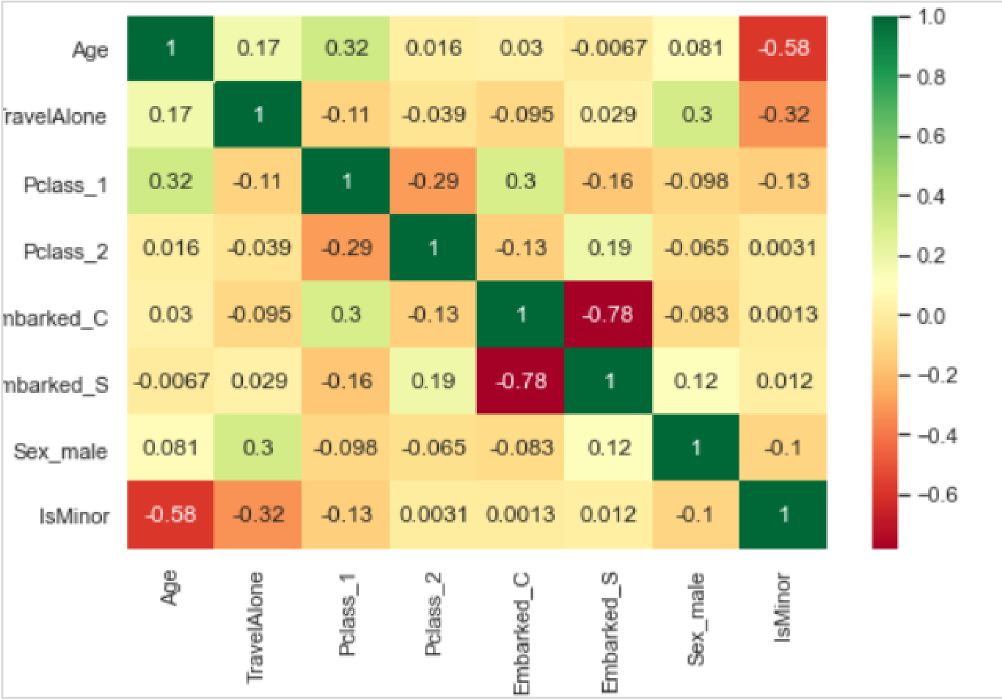
```
Optimal number of features: 9
Selected features: ['Age', 'Fare', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C', 'Embarked_S', 'Sex_male', 'IsMino
```



Code:

```
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()
```

Output:



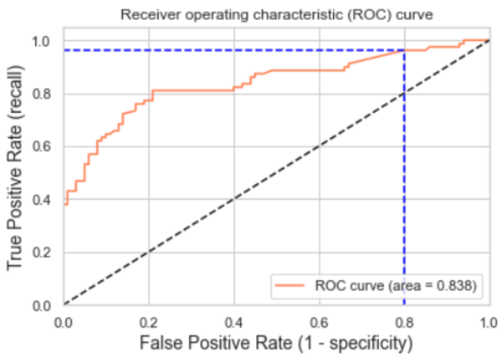
Code:

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

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

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

# check classification scores of logistic regression
logreg = LogisticRegression()
```

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

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

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

Output:

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




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

Code:

```
# 10-fold cross-validation logistic regression
logreg = LogisticRegression()
# Use cross_val_score function
# We are passing the entirety of X and y, not X_train or y_train, it takes care of splitting the data
```

```
# cv=10 for 10 folds
# scoring = {'accuracy', 'neg_log_loss', 'roc_auc'} for evaluation metric - althought they are many
scores_accuracy = cross_val_score(logreg, X, y, cv=10, scoring='accuracy')
scores_log_loss = cross_val_score(logreg, X, y, cv=10, scoring='neg_log_loss')
scores_auc = cross_val_score(logreg, X, y, cv=10, scoring='roc_auc')
print('K-fold cross-validation results:')
print(logreg.__class__.__name__+" average accuracy is %2.3f" % scores_accuracy.mean())
print(logreg.__class__.__name__+" average log_loss is %2.3f" % -scores_log_loss.mean())
print(logreg.__class__.__name__+" average auc is %2.3f" % scores_auc.mean())
```

Output:

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

Code:

```
from sklearn.model_selection import cross_validate

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

modelCV = LogisticRegression()

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

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

Output:

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

Code:

```
#What happens when we add the feature "Fare"?

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

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

```
modelCV = LogisticRegression()

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

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

Output:

```
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)
```

Code:

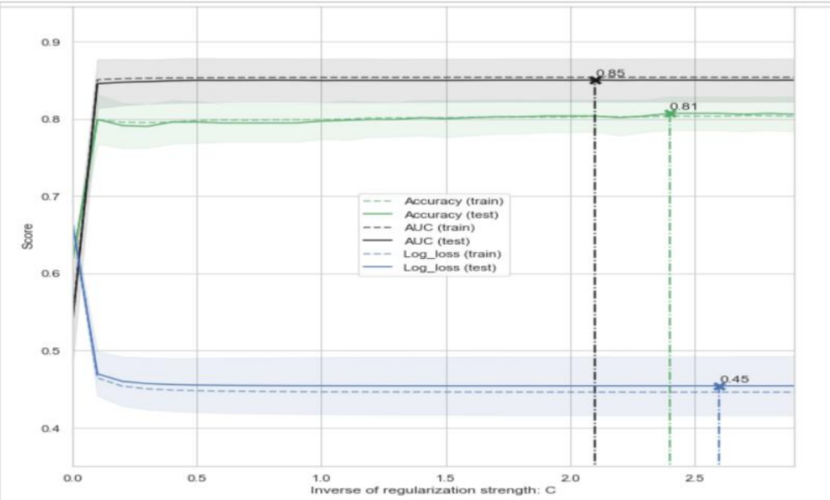
```
from sklearn.model_selection import GridSearchCV
X = final_train[Selected_features]

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

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

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

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

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

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

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

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

for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):
```

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

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

    # 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()
```

Output:

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

GridSearchCV evaluating using multiple scorers simultaneously



Code:

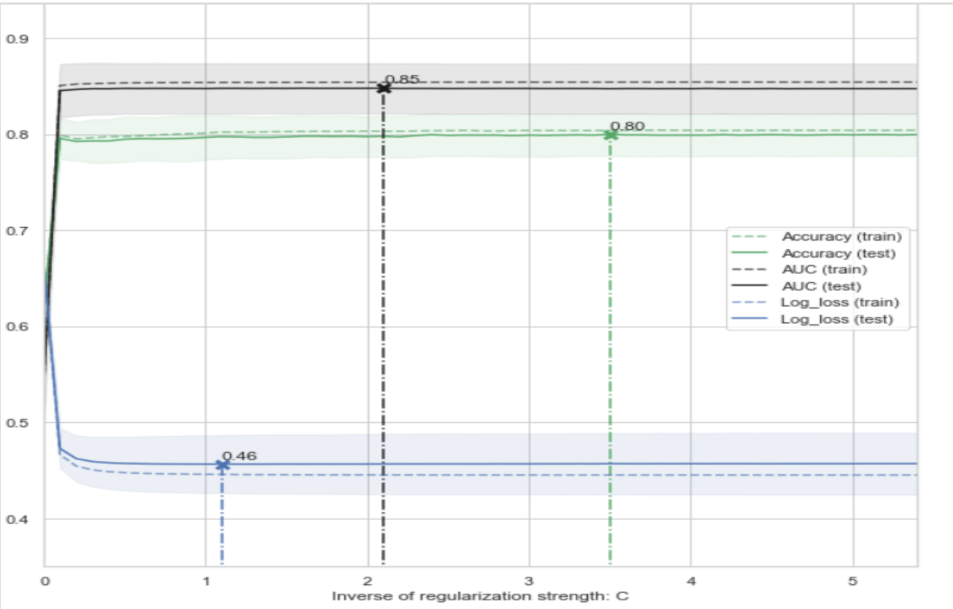
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline

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

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

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

n_folds=5
n_repeats=5

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

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

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

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

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

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

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

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

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

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

for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):
```

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

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

    # 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()
```

Output:

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

GridSearchCV evaluating using multiple scorers simultaneously



Code:

```
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()
```

Output:

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

Practical No: 11

Aim: Implementation of Classifying data using Support Vector Machine (SVM).
a. Linear SVM
b. Non-linear SVM

a. Linear SVM

Code:

```
%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)
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=0.01)
svm_clf.fit(X, y)
```

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

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

Output:

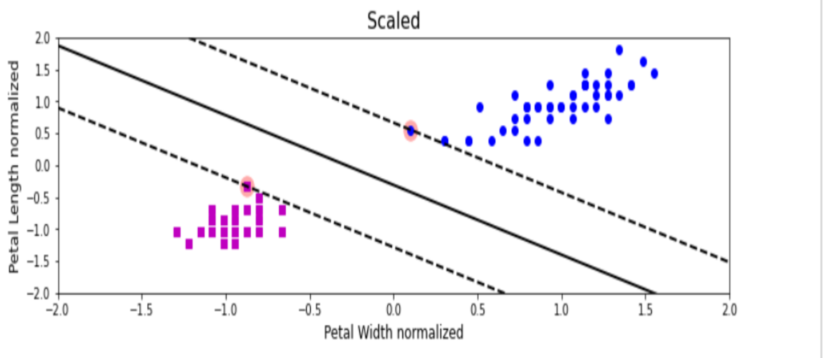
```
array([1])
```

Code:

```
#plot the decision boundaries
import numpy as np

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

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

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

Output:

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



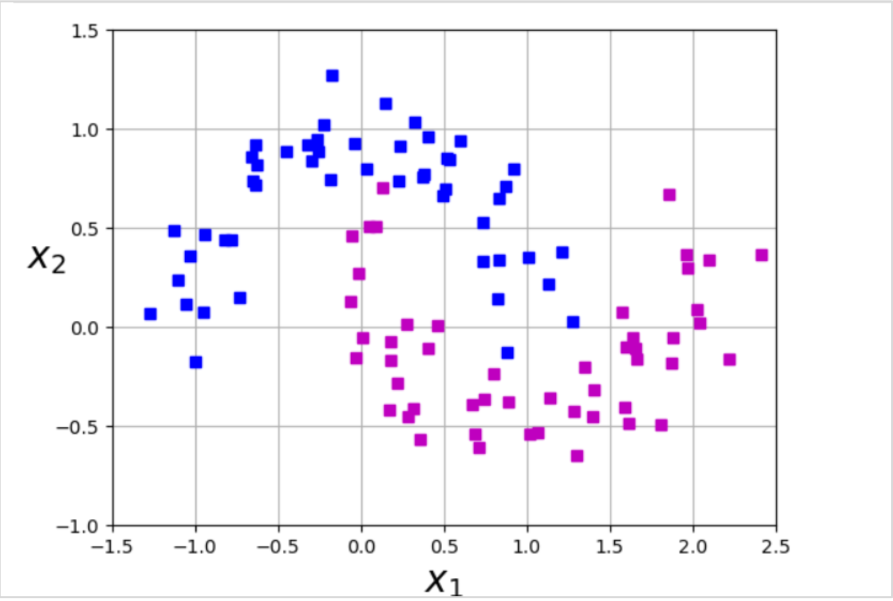
b. Non-linear SVM

Code:

```
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
import numpy as np
```

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

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()
```

Output:



Code:

```
#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)

Output:

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

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Pipeline?Documentation for PipelineiFitted

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

PolynomialFeatures?Documentation for PolynomialFeatures

```
PolynomialFeatures(degree=3)
```

StandardScaler?Documentation for StandardScaler

```
StandardScaler()
```

SVC?Documentation for SVC

```
SVC(C=5, coef0=1, degree=10, kernel='poly')
```

Code:

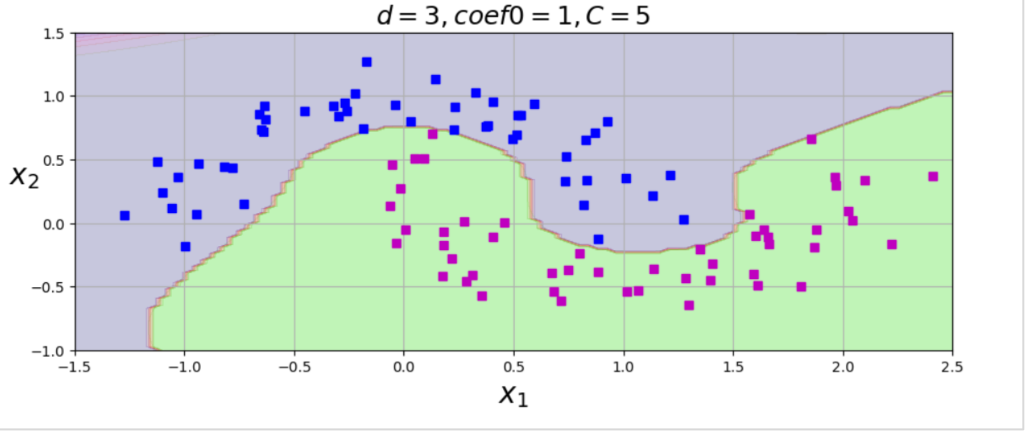
#plot the decision boundaries
plt.figure(figsize=(11, 4))

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

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

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

Output:



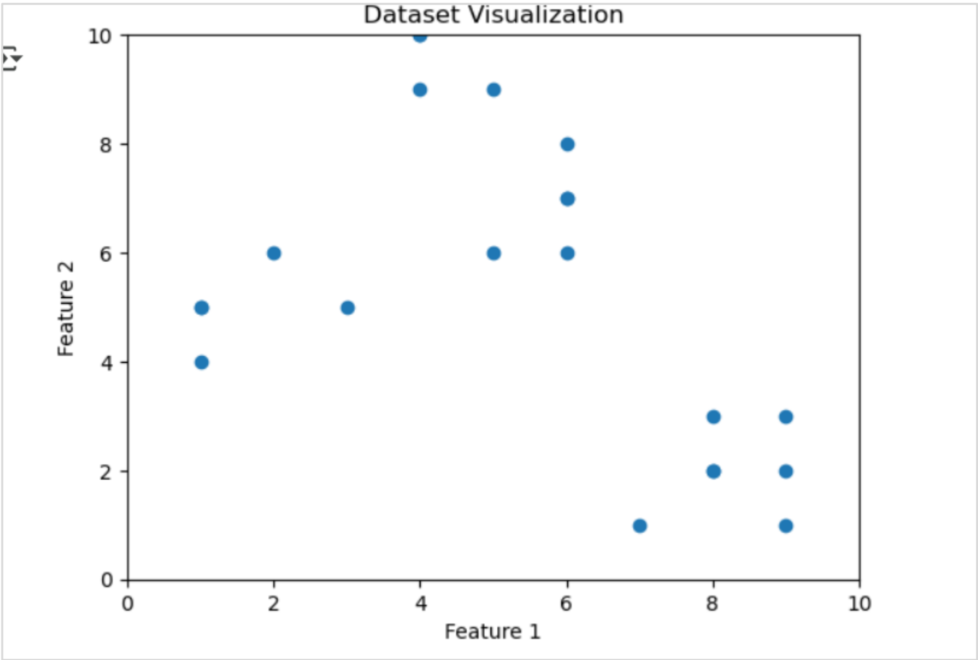
Practical No: 12

Aim: Implement Elbow method for K means Clustering.

Code:

```
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist
import numpy as np
import matplotlib.pyplot as plt

# Creating the dataset
x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6,
               7, 8, 9, 8, 9, 9, 8, 4, 4, 5, 4])
x2 = np.array([5, 4, 5, 6, 5, 8, 6, 7, 6, 7,
               1, 2, 1, 2, 3, 2, 3, 9, 10, 9, 10])
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)

# Visualizing the data
plt.scatter(x1, x2, marker='o')
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.title('Dataset Visualization')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Output:

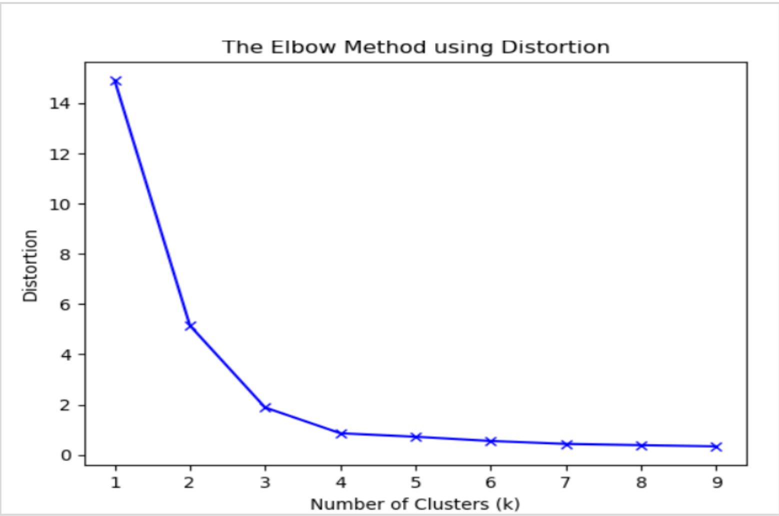


Code:

```
distortions = []
inertias = []
mapping1 = {}
mapping2 = {}
K = range(1, 10)

for k in K:
    kmeanModel = KMeans(n_clusters=k, random_state=42).fit(X)

    distortions.append(sum(np.min(cdist(X, kmeanModel.cluster_centers_, 'euclidean'), axis=1)**2) / X.shape[0])

    inertias.append(kmeanModel.inertia_)

    mapping1[k] = distortions[-1]
    mapping2[k] = inertias[-1]
```

Output:

```
Distortion values:
1 : 14.90249433106576
2 : 5.146258503401359
3 : 1.8817838246409675
4 : 0.856122448979592
5 : 0.7166666666666667
6 : 0.5484126984126984
7 : 0.4325396825396825
8 : 0.3817460317460318
9 : 0.3341269841269841
```

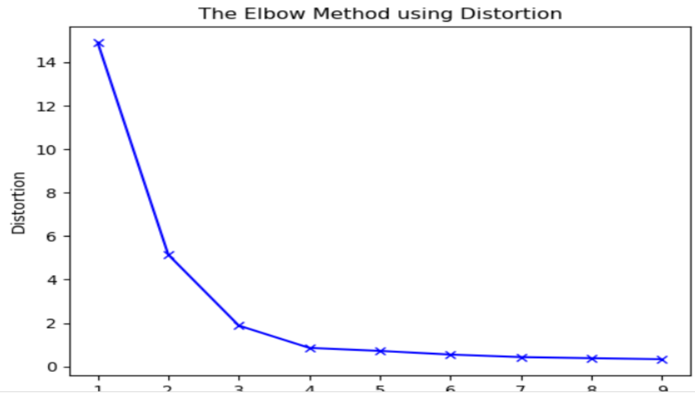


Code:

```
print("Distortion values:")
for key, val in mapping1.items():
    print(f'{key} : {val}')

plt.plot(K, distortions, 'bx-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Distortion')
plt.title('The Elbow Method using Distortion')
plt.show()
```

Output:

```
Distortion values:
1 : 14.90249433106576
2 : 5.146258503401359
3 : 1.8817838246409675
4 : 0.856122448979592
5 : 0.7166666666666667
6 : 0.5484126984126984
7 : 0.4325396825396825
8 : 0.3817460317460318
9 : 0.3341269841269841
```

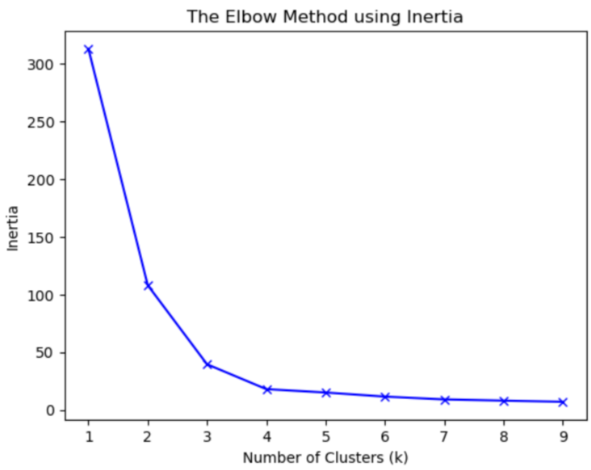


Code:

```
print("Inertia values:")
for key, val in mapping2.items():
    print(f'{key} : {val}')

plt.plot(K, inertias, 'bx-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('The Elbow Method using Inertia')
plt.show()
```

Output:

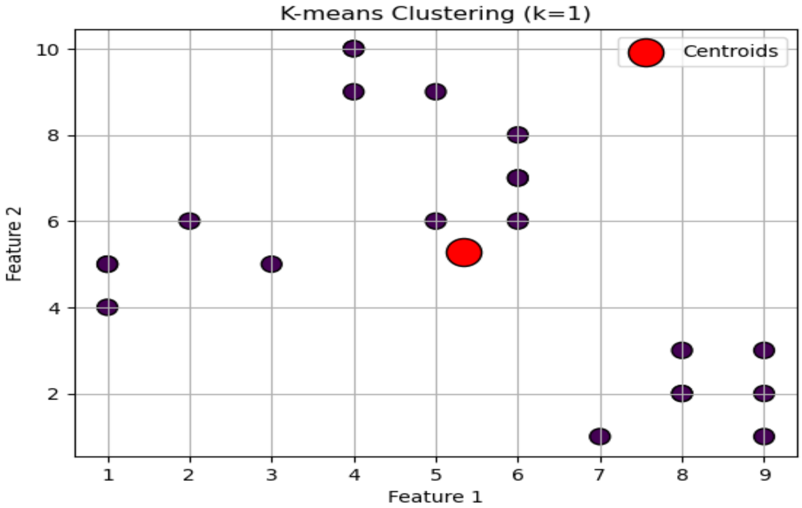
```
Inertia values:
1 : 312.95238095238096
2 : 108.07142857142856
3 : 39.51746031746031
4 : 17.978571428571428
5 : 15.05
6 : 11.516666666666666
7 : 9.083333333333332
8 : 8.016666666666667
9 : 7.0166666666666675
```

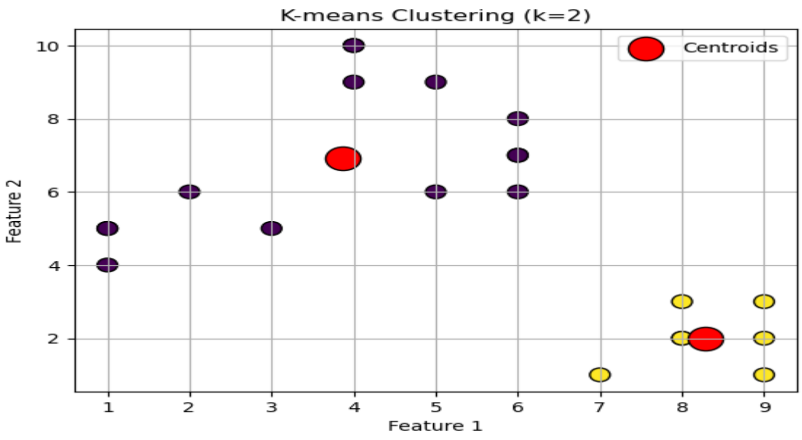


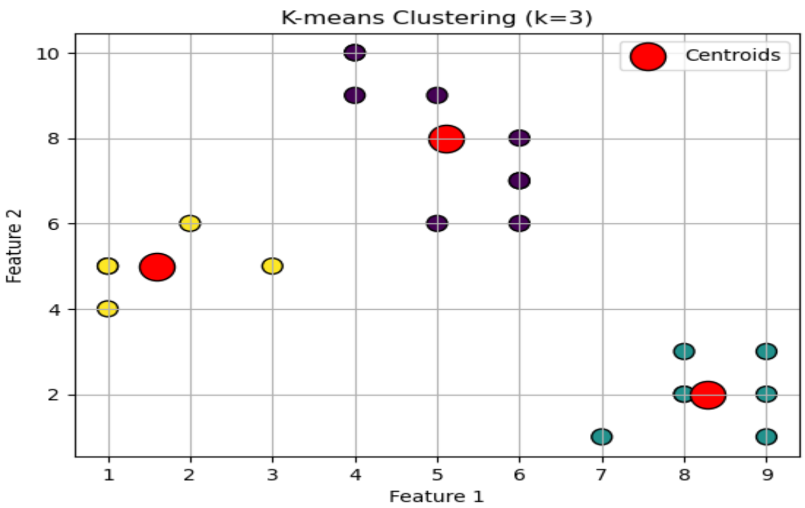
Code:

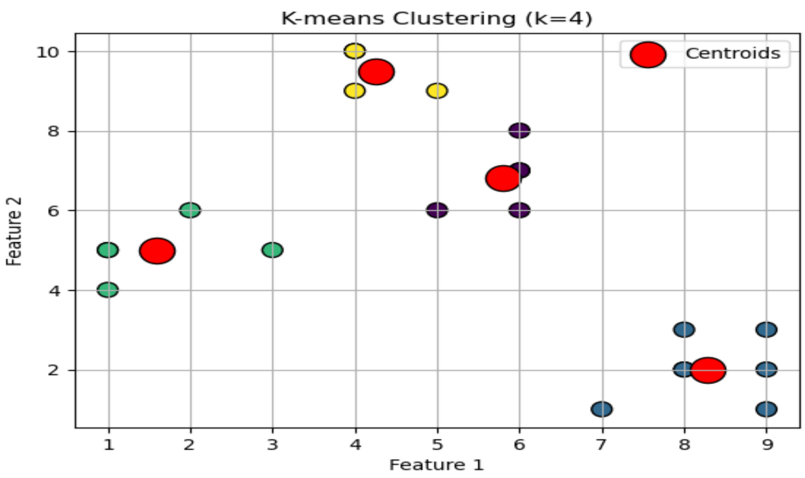
```
k_range = range(1, 5)

for k in k_range:
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42)
    y_kmeans = kmeans.fit_predict(X)

    plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis', marker='o', edgecolor='k', s=100)
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
                s=300, c='red', label='Centroids', edgecolor='k')
    plt.title(f'K-means Clustering (k={k})')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.grid()
    plt.show()
```

Output:






K-means Clustering (k=3)
Centroids
Feature 1
Feature 2
1
2
3
4
5
6
7
8
9
2
4
6
8
10


K-means Clustering (k=4)
Centroids
Feature 1
Feature 2
1
2
3
4
5
6
7
8
9
2
4
6
8
10

Practical No: 13

Aim: Implementation of Bagging Algorithm: Random Forest

Code:

```
#Implementing Random Forest for Classification Tasks
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings('ignore')

# Corrected URL for the dataset
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
titanic_data = pd.read_csv(url)

# Drop rows with missing 'Survived' values
titanic_data = titanic_data.dropna(subset=['Survived'])

# Features and target variable
X = titanic_data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
y = titanic_data['Survived']

# Encode 'Sex' column
X.loc[:, 'Sex'] = X['Sex'].map({'female': 0, 'male': 1})

# Fill missing 'Age' values with the median
X.loc[:, 'Age'].fillna(X['Age'].median(), inplace=True)

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

# Initialize RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the classifier to the training data
rf_classifier.fit(X_train, y_train)

# Make predictions
y_pred = rf_classifier.predict(X_test)

# Calculate accuracy and classification report
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", classification_rep)

# Sample prediction
sample = X_test.iloc[0:1]  # Keep as DataFrame to match model input format
prediction = rf_classifier.predict(sample)
```

```
# Retrieve and display the sample
sample_dict = sample.iloc[0].to_dict()
print(f"\nSample Passenger: {sample_dict}")
print(f"Predicted Survival: {'Survived' if prediction[0] == 1 else 'Did Not Survive'}")
```

Output:

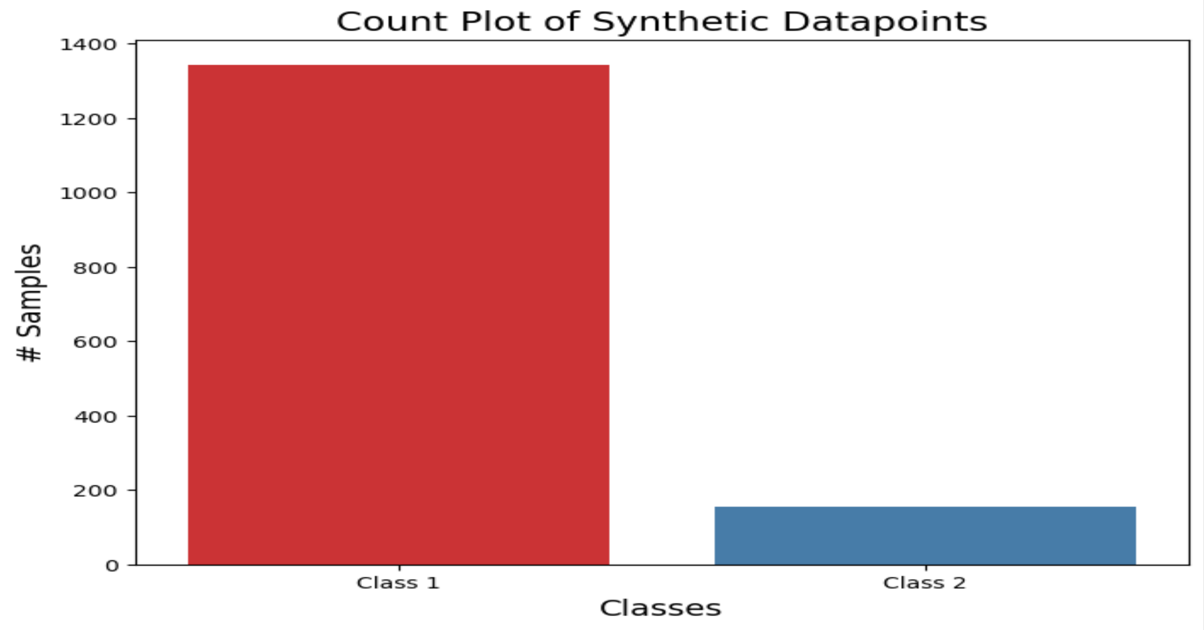
```
Accuracy: 0.80

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.85      0.83       105
           1       0.77      0.73      0.75        74

    accuracy                           0.80       179
   macro avg       0.79      0.79      0.79       179
weighted avg       0.80      0.80      0.80       179


Sample Passenger: {'Pclass': 3, 'Sex': 1, 'Age': 28.0, 'SibSp': 1, 'Parch': 1, 'Fare': 15.2458}
Predicted Survival: Did Not Survive
```

Code:

```
#Bagging and Random Forest for Imbalanced Classification
# Import Required Libraries
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Create synthetic dataset
X, y = make_classification(n_samples=1500, n_features=15, n_informative=5, n_redundant=1, n_classes=2,
weights=[0.90, 0.10])

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Count occurrences of each label
label_counts = np.bincount(y)

# Visualize the imbalanced data
fig, ax = plt.subplots(figsize=(8, 6))
ax = sns.barplot(x=np.arange(2), y=label_counts, palette="Set1")
ax.set_xticks(np.arange(2))
ax.set_xticklabels(['Class 1', 'Class 2'])
ax.set_title("Count Plot of Synthetic Datapoints", fontsize=16)
ax.set_xlabel("Classes", fontsize=14)
ax.set_ylabel("# Samples", fontsize=14)
plt.show()
```

Output:

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_8924\1856569737.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same ef

  ax = sns.barplot(x=np.arange(2), y=label_counts, palette="Set1")
```



Code:

```
#Standard Bagging
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score

# Create a bagging classifier
bagging_clf = BaggingClassifier()

# Train the bagging classifier on the training data
bagging_clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = bagging_clf.predict(X_test)

# Calculate the accuracy of the model
acc_bag = accuracy_score(y_test, y_pred)
print("Bagging Classifier - Test Accuracy:", round(acc_bag, 2))
```

Output:

```
Bagging Classifier - Test Accuracy: 0.93
```

Practical No: 14

Aim: Implementation of Boosting Algorithms

a. AdaBoost
b. Stochastic Gradient Boosting
c. Voting Ensemble (Soft voting, Voting Hard, Voting Regression)

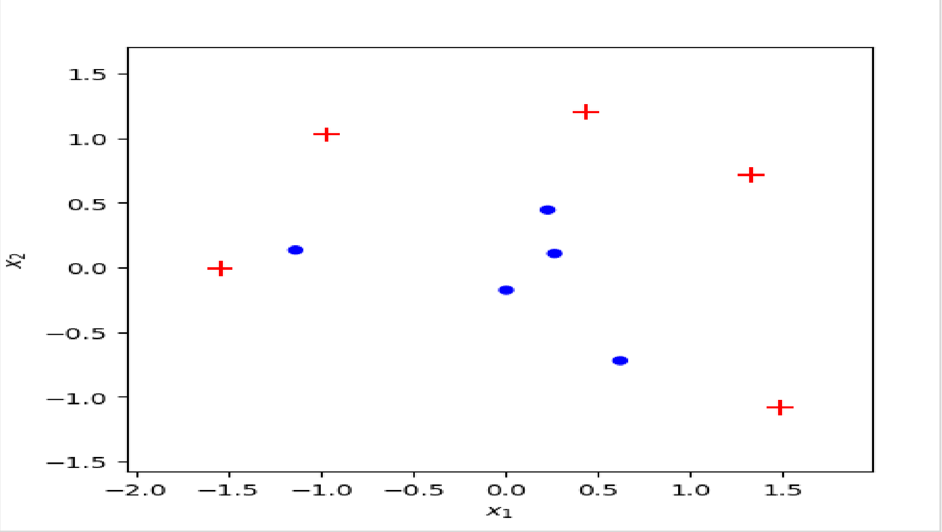
a. Adaboost

Code:

```
from typing import Optional
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.ensemble import AdaBoostClassifier

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('$x_2$')
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)
```

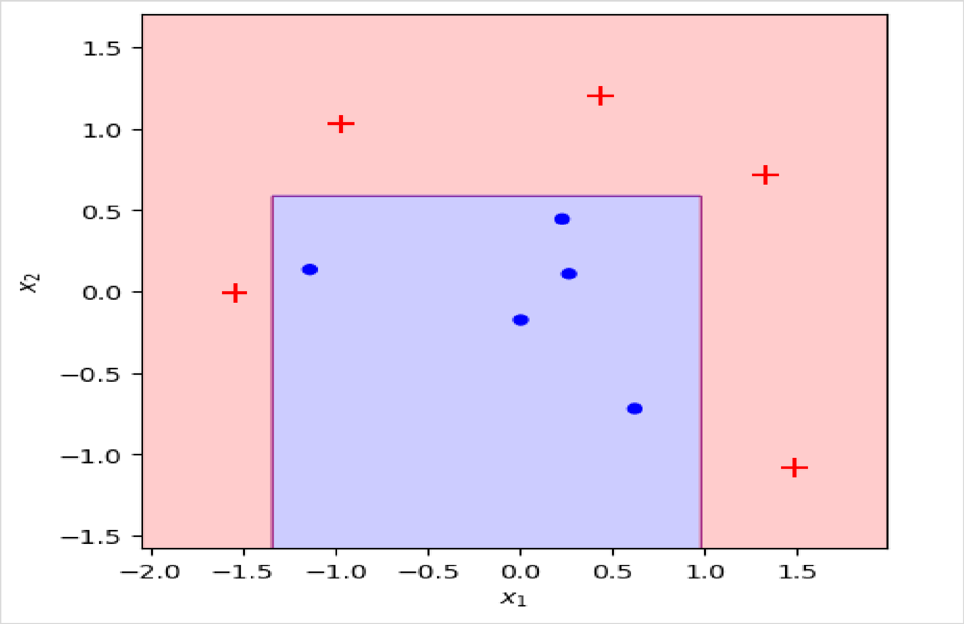
Output:



Code:

```
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%}')
```

Output:

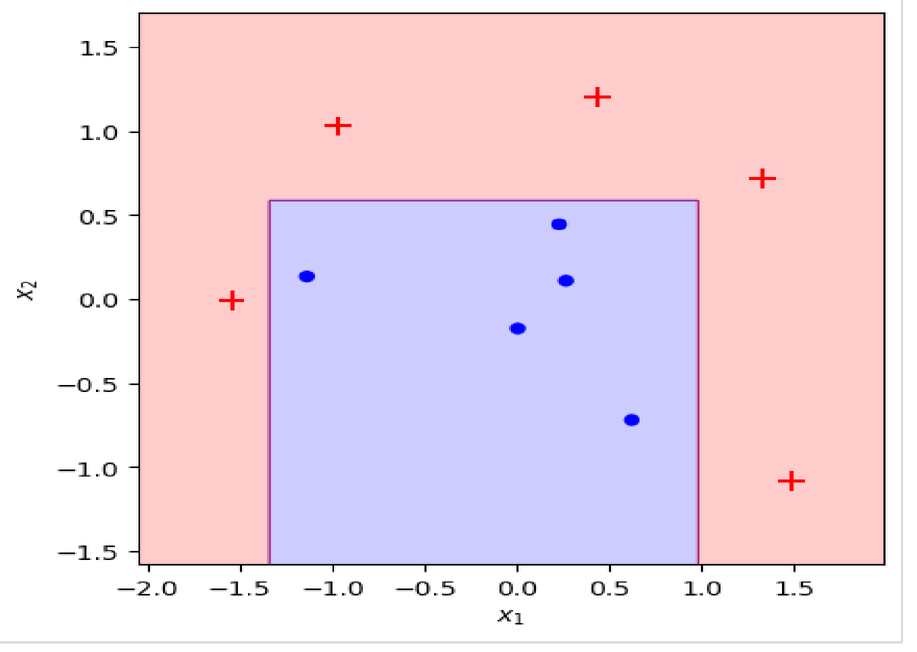


Code:

```
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
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 prediction and its corresponding stump weight.

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

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))
# 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%}')
```

Output:

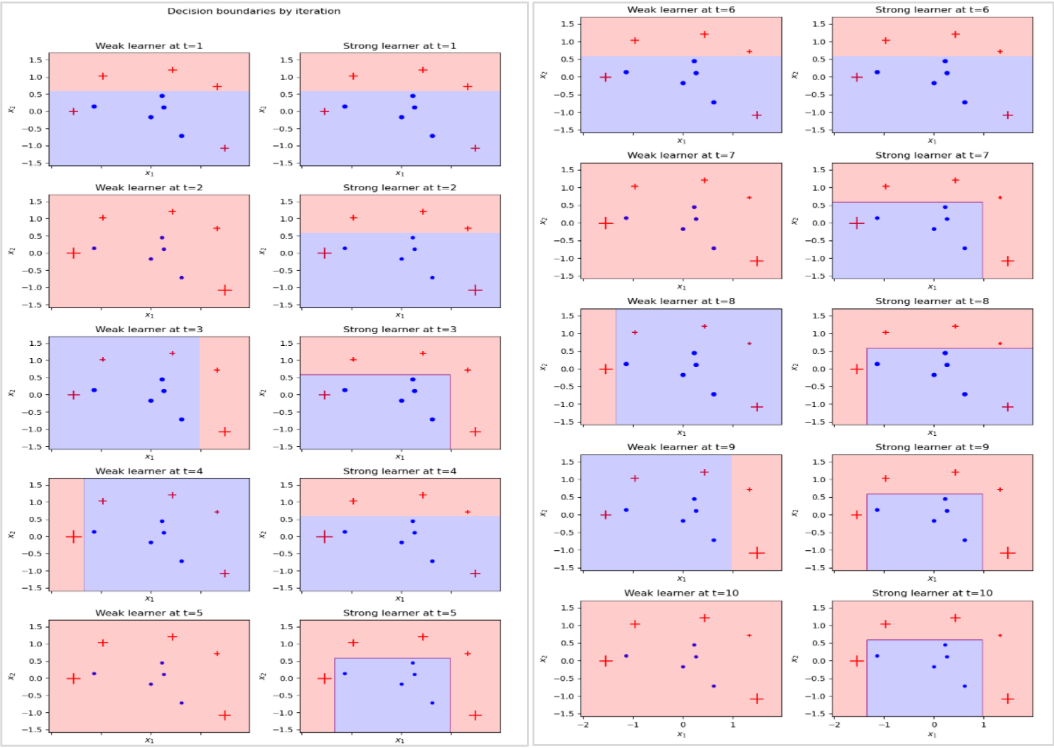


Code:

```
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)
```

Output:



b. Stochastic Gradient Boosting.

Code:

```
def gradient_descent(gradient, start, learn_rate, n_iter):
    vector = start
    for _ in range(n_iter):
        diff = -learn_rate * gradient(vector)
        vector += diff
    return vector
import numpy as np
```

```
def gradient_descent(
    gradient, start, learn_rate, n_iter=50, tolerance=1e-06
):
    vector = start
    for _ in range(n_iter):
        diff = -learn_rate * gradient(vector)
        if np.all(np.abs(diff) <= tolerance):
            break
        vector += diff
    return vector
 gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.2
... )
```

Output:

```
2.210739197207331e-06
```

Code:

```
gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.8
... )
```

Output:

```
-4.77519666596786e-07
```

Code:

```
gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.005
... )
```

Output:

```
6.050060671375367
```

Code:

```
gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.005,
...     n_iter=100
... )
3.660323412732294
>>> gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.005,
...     n_iter=1000
... )
```

```
0.0004317124741065828
>>> gradient_descent(
...     gradient=lambda v: 2 * v, start=10.0, learn_rate=0.005,
...     n_iter=2000
... )
```

Output:

```
9.952518849647663e-05
```

Code:

```
gradient_descent(
...     gradient=lambda v: 4 * v**3 - 10 * v - 3, start=0,
...     learn_rate=0.2
... )
```

Output:

```
-1.4207567437458342
```

Code:

```
gradient_descent(
...     gradient=lambda v: 4 * v**3 - 10 * v - 3, start=0,
...     learn_rate=0.1
... )
```

Output:

```
1.285401330315467
```

c. Voting Ensemble (Soft voting, Voting Hard, Voting Regression)

a. Soft voting

Code:

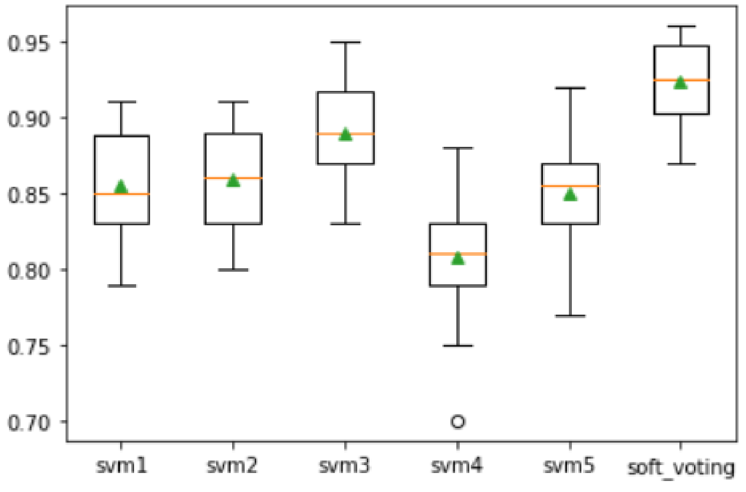
```
# get a voting ensemble of models
def get_voting():
  # define the base models
  models = list()
  models.append(('svm1', SVC(probability=True, kernel='poly', degree=1)))
  models.append(('svm2', SVC(probability=True, kernel='poly', degree=2)))
  models.append(('svm3', SVC(probability=True, kernel='poly', degree=3)))
  models.append(('svm4', SVC(probability=True, kernel='poly', degree=4)))
  models.append(('svm5', SVC(probability=True, kernel='poly', degree=5)))
  # define the voting ensemble
```

```
  ensemble = VotingClassifier(estimators=models, voting='soft')
  return ensemble

# get a list of models to evaluate
def get_models():
  models = dict()
  models['svm1'] = SVC(probability=True, kernel='poly', degree=1)
  models['svm2'] = SVC(probability=True, kernel='poly', degree=2)
  models['svm3'] = SVC(probability=True, kernel='poly', degree=3)
  models['svm4'] = SVC(probability=True, kernel='poly', degree=4)
  models['svm5'] = SVC(probability=True, kernel='poly', degree=5)
  models['soft_voting'] = get_voting()
  return models
# compare soft voting ensemble to standalone classifiers
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
from matplotlib import pyplot

# get the dataset
def get_dataset():
  X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
n_redundant=5, random_state=2)
  return X, y

# get a voting ensemble of models
def get_voting():

  # define the base models
  models = list()
  models.append(('svm1', SVC(probability=True, kernel='poly', degree=1)))
  models.append(('svm2', SVC(probability=True, kernel='poly', degree=2)))
  models.append(('svm3', SVC(probability=True, kernel='poly', degree=3)))
  models.append(('svm4', SVC(probability=True, kernel='poly', degree=4)))
  models.append(('svm5', SVC(probability=True, kernel='poly', degree=5)))
  # define the voting ensemble
  ensemble = VotingClassifier(estimators=models, voting='soft')
  return ensemble

# get a list of models to evaluate
def get_models():
  models = dict()
  models['svm1'] = SVC(probability=True, kernel='poly', degree=1)
  models['svm2'] = SVC(probability=True, kernel='poly', degree=2)
  models['svm3'] = SVC(probability=True, kernel='poly', degree=3)
  models['svm4'] = SVC(probability=True, kernel='poly', degree=4)
```

```
  models['svm5'] = SVC(probability=True, kernel='poly', degree=5)
  models['soft_voting'] = get_voting()
  return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
  cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
  scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')
  return scores

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
  scores = evaluate_model(model, X, y)
  results.append(scores)
  names.append(name)
  print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

Output:

```
>svm1 0.855 (0.035)
>svm2 0.859 (0.034)
>svm3 0.890 (0.035)
>svm4 0.808 (0.037)
>svm5 0.850 (0.037)
>soft_voting 0.923 (0.027)
```



b. Hard voting

Code:

```
# make a prediction with a soft voting ensemble
from sklearn.datasets import make_classification
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC
# define dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
n_redundant=5, random_state=2)
# define the base models
models = list()
models.append(('svm1', SVC(probability=True, kernel='poly', degree=1)))
models.append(('svm2', SVC(probability=True, kernel='poly', degree=2)))
models.append(('svm3', SVC(probability=True, kernel='poly', degree=3)))
models.append(('svm4', SVC(probability=True, kernel='poly', degree=4)))
models.append(('svm5', SVC(probability=True, kernel='poly', degree=5)))
# define the soft voting ensemble
ensemble = VotingClassifier(estimators=models, voting='soft')
# fit the model on all available data
ensemble.fit(X, y)
# make a prediction for one example
data = [[5.88891819,2.64867662,-0.42728226,-1.24988856,-0.00822,-
3.57895574,2.87938412,-1.55614691,-0.38168784,7.50285659,-1.16710354,-
5.02492712,-0.46196105,-0.64539455,-1.71297469,0.25987852,-0.193401,-
5.52022952,0.0364453,-1.960039]]
yhat = ensemble.predict(data)
print('Predicted Class: %d' % (yhat))
```

Output:

```
Predicted Class: 1
```

c. Regression Voting

Code:

```
# test regression dataset
from sklearn.datasets import make_regression
# define dataset
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, noise=0.1,
random_state=1)
# summarize the dataset
print(X.shape, y.shape)
```

Output:

```
(1000, 20) (1000,)
```

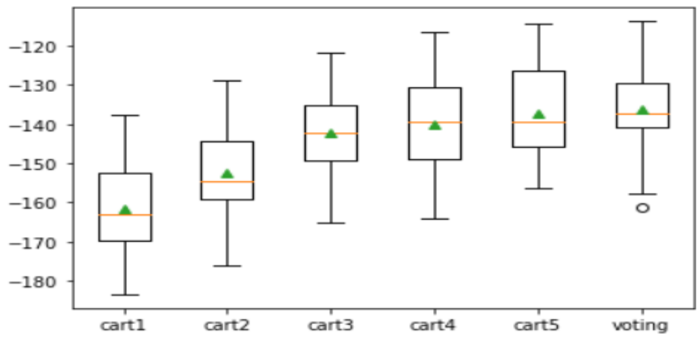
Code:

```
# get a voting ensemble of models
def get_voting():
  # define the base models
  models = list()
  models.append(('cart1', DecisionTreeRegressor(max_depth=1)))
  models.append(('cart2', DecisionTreeRegressor(max_depth=2)))
  models.append(('cart3', DecisionTreeRegressor(max_depth=3)))
  models.append(('cart4', DecisionTreeRegressor(max_depth=4)))
  models.append(('cart5', DecisionTreeRegressor(max_depth=5)))
  # define the voting ensemble
  ensemble = VotingRegressor(estimators=models)
  return ensemble

# get a list of models to evaluate
def get_models():
  models = dict()
  models['cart1'] = DecisionTreeRegressor(max_depth=1)
  models['cart2'] = DecisionTreeRegressor(max_depth=2)
  models['cart3'] = DecisionTreeRegressor(max_depth=3)
  models['cart4'] = DecisionTreeRegressor(max_depth=4)
  models['cart5'] = DecisionTreeRegressor(max_depth=5)
  models['voting'] = get_voting()
  return models
# compare voting ensemble to each standalone models for regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import VotingRegressor
from matplotlib import pyplot

# get the dataset
def get_dataset():
  X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, noise=0.1,
random_state=1)
  return X, y

# get a voting ensemble of models
def get_voting():
  # define the base models
  models = list()
  models.append(('cart1', DecisionTreeRegressor(max_depth=1)))
  models.append(('cart2', DecisionTreeRegressor(max_depth=2)))
  models.append(('cart3', DecisionTreeRegressor(max_depth=3)))
  models.append(('cart4', DecisionTreeRegressor(max_depth=4)))
  models.append(('cart5', DecisionTreeRegressor(max_depth=5)))
  # define the voting ensemble
  ensemble = VotingRegressor(estimators=models)
```

```
  return ensemble

# get a list of models to evaluate
def get_models():
  models = dict()
  models['cart1'] = DecisionTreeRegressor(max_depth=1)
  models['cart2'] = DecisionTreeRegressor(max_depth=2)
  models['cart3'] = DecisionTreeRegressor(max_depth=3)
  models['cart4'] = DecisionTreeRegressor(max_depth=4)
  models['cart5'] = DecisionTreeRegressor(max_depth=5)
  models['voting'] = get_voting()
  return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
  cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
  scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error', cv=cv,
n_jobs=-1, error_score='raise')
  return scores

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
  scores = evaluate_model(model, X, y)
  results.append(scores)
  names.append(name)
  print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

Output:

```
>cart1 -161.519 (11.414)
>cart2 -152.596 (11.271)
>cart3 -142.378 (10.900)
>cart4 -140.086 (12.469)
>cart5 -137.145 (12.222)
>voting -136.347 (11.231)
```

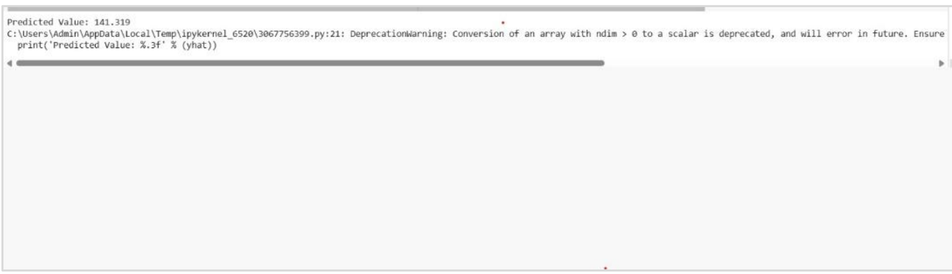


Code:

```
# make a prediction with a voting ensemble
from sklearn.datasets import make_regression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import VotingRegressor
# define dataset
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, noise=0.1, random_state=1)
# define the base models
models = list()
models.append(('cart1', DecisionTreeRegressor(max_depth=1)))
models.append(('cart2', DecisionTreeRegressor(max_depth=2)))
models.append(('cart3', DecisionTreeRegressor(max_depth=3)))
models.append(('cart4', DecisionTreeRegressor(max_depth=4)))
models.append(('cart5', DecisionTreeRegressor(max_depth=5)))
# define the voting ensemble
ensemble = VotingRegressor(estimators=models)
# fit the model on all available data
ensemble.fit(X, y)
# make a prediction for one example
data = [[0.59332206,-0.56637507,1.34808718,-0.57054047,-0.72480487,1.05648449,0.77744852,0.07361796,0.88398267,2.02843157,1.01902732,0.11227799,0.94218853,0.26741783,0.91458143,-0.72759572,1.08842814,-0.61450942,-0.69387293,1.69169009]]
yhat = ensemble.predict(data)
print('Predicted Value: %.3f' % (yhat))
```

Output:

```
Predicted Value: 141.319
C:\Users\Admin\AppData\Local\Temp\ipykernel_6520\3067756399.py:21: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  print('Predicted Value: %.3f' % (yhat))
```



```
Predicted Value: 141.319
C:\Users\Admin\AppData\Local\Temp\ipykernel_6520\3067756399.py:21: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure
  print('Predicted Value: %.3f' % (yhat))
```