# MCAL22 Artificial Intelligence and Machine Learning Lab

## INDEX

| Sr. No | Artificial Intelligence Section | Date | CO | Sign |
|---|---|---|---|---|
| 1. | Implementation of Logic programming using PROLOG DFS for water jug problem | | CO1 | |
| 2. | Implementation of Logic programming using PROLOG BFS for tic-tac-toe problem | | CO1 | |
| 3. | Implementation of Logic programming using PROLOG Hill-climbing to solve 8- Puzzle Problem. | | CO1 | |
| 4. | Introduction to Python Programming: Learn the different libraries - NumPy, Pandas, SciPy, Matplotlib, Scikit Learn. | | CO2 | |
| 5. | Implement Perceptron algorithm for OR operation | | CO2 | |
| 6. | Improve the prediction accuracy by estimating the weight values for the training data using stochastic gradient descent. (Perceptron) | | CO2 | |
| 7. | Implement Adaline algorithm for AND operation | | CO2 | |

| Sr. No | Machine Learning Section | Date | CO | Sign |
|---|---|---|---|---|
| 1. | Implementation of Features Extraction and Selection, Normalization, Transformation, Principal Components Analysis. | | CO3 | |
| 2. | Implementation of Logistic regression | | CO4 | |
| 3. | Implementation of Classifying data using Support Vector Machine (SVM)- Linear and Non-Linear SVM Classification | | CO4 | |
| 4. | Implement Elbow method for K means Clustering | | CO4 | |
| 5. | Implementation of Bagging Algorithm: Random Forest | | CO4 | |
| 6. | Implementation of Boosting Algorithms: AdaBoost, Stochastic Gradient Boosting, Voting Ensemble | | CO4 | |

# Artificial Intelligence Section

## Practical 1: 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(' 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(' Current Quantity :'),
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).
```

## OUTPUT:

```
?- start(0,0).
 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
 Current Quantity : 4lit Jug: 0| 3lit Jug: 0|
```

```
 Enter the move::1.
 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
 Current Quantity : 4lit Jug: 4| 3lit Jug: 0|
```

```
 Enter the move::|: 6.
 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
 Current Quantity : 4lit Jug: 1| 3lit Jug: 3|
```

```
 Enter the move::|: 4.
 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
 Current Quantity : 4lit Jug: 1| 3lit Jug: 0|
```

```
 Enter the move::|: 8.
 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
 Current Quantity : 4lit Jug: 0| 3lit Jug: 1|
```

```
 Enter the move::|: 1.
 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
 Current Quantity : 4lit Jug: 4| 3lit Jug: 1|
```

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

# Practical 2: Implementation of Logic programming using PROLOG BFS for tic-tac-toe problem

## CODE:

```
% To play a game with the computer, type
% play.
% Predicates that define the winning conditions:
win(Board, Player) :- rowwin(Board, Player).
win(Board, Player) :- colwin(Board, Player).
win(Board, Player) :- diagwin(Board, Player).

rowwin(Board, Player) :- Board = [Player,Player,Player,_,_,_,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,Player,Player,Player,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,_,_,_,Player,Player,Player].

colwin(Board, Player) :- Board = [Player,_,_,Player,_,_,Player,_,_].
colwin(Board, Player) :- Board = [_,Player,_,_,Player,_,_,Player,_].
colwin(Board, Player) :- Board = [_,_,Player,_,_,Player,_,_,Player].

diagwin(Board, Player) :- Board = [Player,_,_,_,Player,_,_,_,Player].
diagwin(Board, Player) :- Board = [_,_,Player,_,Player,_,Player,_,_].
move([b,B,C,D,E,F,G,H,I], Player, [Player,B,C,D,E,F,G,H,I]).
move([A,b,C,D,E,F,G,H,I], Player, [A,Player,C,D,E,F,G,H,I]).
move([A,B,b,D,E,F,G,H,I], Player, [A,B,Player,D,E,F,G,H,I]).
move([A,B,C,b,E,F,G,H,I], Player, [A,B,C,Player,E,F,G,H,I]).
move([A,B,C,D,b,F,G,H,I], Player, [A,B,C,D,Player,F,G,H,I]).
move([A,B,C,D,E,b,G,H,I], Player, [A,B,C,D,E,Player,G,H,I]).
move([A,B,C,D,E,F,b,H,I], Player, [A,B,C,D,E,F,Player,H,I]).
move([A,B,C,D,E,F,G,b,I], Player, [A,B,C,D,E,F,G,Player,I]).
move([A,B,C,D,E,F,G,H,b], Player, [A,B,C,D,E,F,G,H,Player]).

display([A,B,C,D,E,F,G,H,I]) :- write([A,B,C]),nl,write([D,E,F]),nl,
 write([G,H,I]),nl,nl.

% Predicates to support playing a game with the user:
x_can_win_in_one(Board) :- move(Board, x, Newboard), win(Newboard, x).

% The predicate validategenerates the computer's (playing o) reponse
% from the current Board.
validate(Board,Newboard) :-
move(Board, o, Newboard),
win(Newboard, o),
  !.
validate(Board,Newboard) :-
move(Board, o, Newboard),
not(x_can_win_in_one(Newboard)).
validate(Board,Newboard) :-
move(Board, o, Newboard).

% The following translates from an integer description
```

```
% of x's move to a board transformation.

xmove([b,B,C,D,E,F,G,H,I], 1, [x,B,C,D,E,F,G,H,I]).
xmove([A,b,C,D,E,F,G,H,I], 2, [A,x,C,D,E,F,G,H,I]).
xmove([A,B,b,D,E,F,G,H,I], 3, [A,B,x,D,E,F,G,H,I]).
xmove([A,B,C,b,E,F,G,H,I], 4, [A,B,C,x,E,F,G,H,I]).
xmove([A,B,C,D,b,F,G,H,I], 5, [A,B,C,D,x,F,G,H,I]).
xmove([A,B,C,D,E,b,G,H,I], 6, [A,B,C,D,E,x,G,H,I]).
xmove([A,B,C,D,E,F,b,H,I], 7, [A,B,C,D,E,F,x,H,I]).
xmove([A,B,C,D,E,F,G,b,I], 8, [A,B,C,D,E,F,G,x,I]).
xmove([A,B,C,D,E,F,G,H,b], 9, [A,B,C,D,E,F,G,H,x]).
xmove(Board, _, Board) :- write('Illegal move.'), nl.

% The 0-place predicate playo starts a game with the user.

play :- explain, playfrom([b,b,b,b,b,b,b,b,b]).

explain :-
  write('You play X by entering integer positions followed by a period.'),
nl,
  display([1,2,3,4,5,6,7,8,9]).

playfrom(Board) :- win(Board, x), write('You win!').
playfrom(Board) :- win(Board, o), write('I win!').
playfrom(Board) :- read(N),
xmove(Board, N, Newboard),
display(Newboard),
validate(Newboard, Newnewboard),
display(Newnewboard),
playfrom(Newnewboard).
```

## OUTPUT:

```
?- play.
You play X by entering integer positions followed by a period.
[1,2,3]
[4,5,6]
[7,8,9]

|: 1.
[x,b,b]
[b,b,b]
[b,b,b]

[x,o,b]
[b,b,b]
[b,b,b]

|: 7.
[x,o,b]
[b,b,b]
[x,b,b]

[x,o,b]
[o,b,b]
[x,b,b]

|: 5.
[x,o,b]
[o,x,b]
[x,b,b]

[x,o,o]
[o,x,b]
[x,b,b]

|: 9.
[x,o,o]
[o,x,b]
[x,b,x]

[x,o,o]
[o,x,o]
[x,b,x]

You win!
true .
```

## Practical 3: Implementation of Logic programming using PROLOG Hill-climbing to solve 8- Puzzle Problem.

## CODE:

```
ids :-
   start(State),
length(Moves, N),
  hill([State], Moves, Path), !,
   show([start|Moves], Path),
format('~nmoves = ~w~n', [N]).

hill([State|States], [], Path) :-
goal(State), !,
   reverse([State|States], Path).

hill([State|States], [Move|Moves], Path) :-
move(State, Next, Move),
not(memberchk(Next, [State|States])),
   hill([Next,State|States],  Moves,  Path).

show([], _).
show([Move|Moves], [State|States]) :-
   State = state(A,B,C,D,E,F,G,H,J),
format('~n~w~n~n',  [Move]),
format('~w ~w  ~w~n',[A,B,C]),
format('~w ~w  ~w~n',[D,E,F]),
format('~w ~w ~w~n',[G,H,J]),
show(Moves, States).

% Empty position is marked with '*'
start( state(0,1,*,2,3,4,5,6,7) ).

goal(  state(*,0,1,2,3,4,5,6,7)  ).

move(  state(A,*,C,D,E,F,G,H,J),  state(*,A,C,D,E,F,G,H,J),  left  ).
move(  state(A,B,*,D,E,F,G,H,J),  state(A,*,B,D,E,F,G,H,J),  left  ).
move(  state(A,B,C,D,*,F,G,H,J),  state(A,B,C,*,D,F,G,H,J),  left  ).
move(  state(A,B,C,D,E,*,G,H,J),  state(A,B,C,D,*,E,G,H,J),  left  ).
move(  state(A,B,C,D,E,F,G,*,J),  state(A,B,C,D,E,F,*,G,J),  left  ).
move(  state(A,B,C,D,E,F,G,H,*),  state(A,B,C,D,E,F,G,*,H),  left  ).

move(  state(*,B,C,D,E,F,G,H,J),  state(B,*,C,D,E,F,G,H,J),  right).
move(  state(A,*,C,D,E,F,G,H,J),  state(A,C,*,D,E,F,G,H,J),  right).
move(  state(A,B,C,*,E,F,G,H,J),  state(A,B,C,E,*,F,G,H,J),  right).
move(  state(A,B,C,D,*,F,G,H,J),  state(A,B,C,D,F,*,G,H,J),  right).
move(  state(A,B,C,D,E,F,*,H,J),  state(A,B,C,D,E,F,H,*,J),  right).
move(  state(A,B,C,D,E,F,G,*,J),  state(A,B,C,D,E,F,G,J,*),  right).

move(  state(A,B,C,*,E,F,G,H,J),  state(*,B,C,A,E,F,G,H,J),  up).
move(  state(A,B,C,D,*,F,G,H,J),  state(A,*,C,D,B,F,G,H,J),  up).
```

```
move(  state(A,B,C,D,E,*,G,H,J),  state(A,B,*,D,E,C,G,H,J),  up).
move(  state(A,B,C,D,E,F,*,H,J),  state(A,B,C,*,E,F,D,H,J),  up).
move(  state(A,B,C,D,E,F,G,*,J),  state(A,B,C,D,*,F,G,E,J),  up).
move(  state(A,B,C,D,E,F,G,H,*),  state(A,B,C,D,E,*,G,H,F),  up).

move(  state(*,B,C,D,E,F,G,H,J),  state(D,B,C,*,E,F,G,H,J),  down  ).
move(  state(A,*,C,D,E,F,G,H,J),  state(A,E,C,D,*,F,G,H,J),  down  ).
move(  state(A,B,*,D,E,F,G,H,J),  state(A,B,F,D,E,*,G,H,J),  down  ).
move(  state(A,B,C,*,E,F,G,H,J),  state(A,B,C,G,E,F,*,H,J),  down  ).
move(  state(A,B,C,D,*,F,G,H,J),  state(A,B,C,D,H,F,G,*,J),  down  ).
move(  state(A,B,C,D,E,*,G,H,J),  state(A,B,C,D,E,J,G,H,*),  down  ).
```

## OUTPUT:

```
?- ids.

start

0 1 *
2 3 4
5 6 7

left

0 * 1
2 3 4
5 6 7

left

* 0 1
2 3 4
5 6 7

moves = 2
true.
```

**Practical 4: Introduction to Python Programming: Learn the different libraries - NumPy, Pandas, SciPy, Matplotlib, Scikit Learn.**

- **NumPy**

```
[1]: !pip install numpy
     Requirement already satisfied: numpy in c:\users\kapil\anaconda3\lib\site-packages (1.26.4)

[5]: import numpy as np

[9]: #creating an array
     digits=np.array([[2,4,6],
                      [1,3,5],
                      [7,8,9]
                      ])

[11]: digits

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

[13]: #Addition
      a=15
      b=25
      c=a+b
      c

[13]: 40
```

```
[15]: #Addition of two arrays
      arr1=np.array([2,3,4])
      arr2=np.array([1,5,2])
      arr3=np.add(arr1,arr2)
      arr3

[15]: array([3, 8, 6])

[17]: #Transpose
      digits.transpose()

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

[21]: #Sort Array
      sortArray=np.array([
          [1,4,2],
          [3,8,5],
          [6,9,7]
      ])
      np.sort(sortArray,axis=None)

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

```
[21]: #Sort Array
      sortArray=np.array([
          [1,4,2],
          [3,8,5],
          [6,9,7]
      ])
      np.sort(sortArray,axis=None)

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

[23]: #Array type
      arr=np.array([1,4,6,8], dtype='S')
      print(arr)
      print(arr.dtype)

      [b'1' b'4' b'6' b'8']
      |S1
```

## ➢ Pandas

```
[26]: import pandas as pd
```

```
[28]: #Creating Dataframe
data={
    'India':[7,4,9],
    'Austria':[1,5,8]
}
num=pd.DataFrame(data)
num
```

[28]:

| | India | Austria |
|---|---|---|
| 0 | 7 | 1 |
| 1 | 4 | 5 |
| 2 | 9 | 8 |

```
[30]: num=pd.DataFrame(data,index=['Food','Education','People'])
num
```

[30]:

| | India | Austria |
|---|---|---|
| Food | 7 | 1 |
| Education | 4 | 5 |
| People | 9 | 8 |

## ➢ SciPy

```
import numpy as np
```

[35]:

```
A=np.array([[3,2],[6,3]])

#To find determinant
from scipy import linalg
linalg.det(A)
```

[35]:

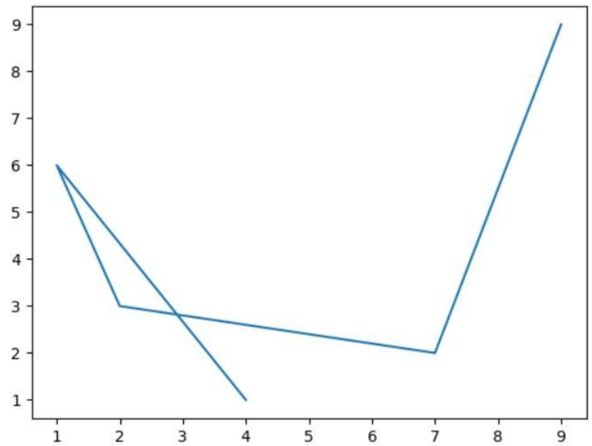
```
-3.0
```

## ➢ Matplotlib

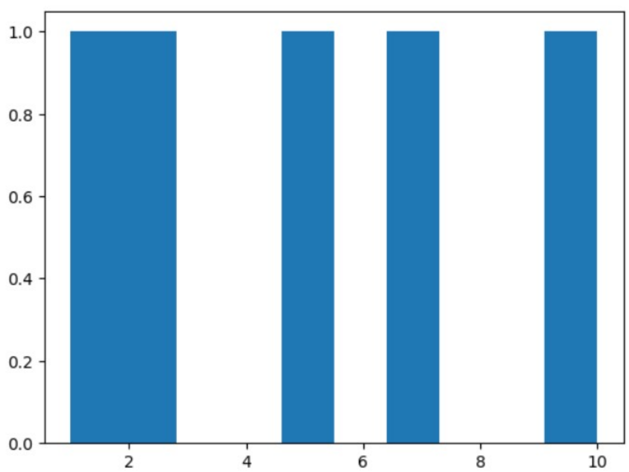
```
[38]: #Matplotlib
      from matplotlib import pyplot as plt
```

```
[40]: #x-axis values
      x=[4,1,2,7,9]
      #y-axis values
      y=[1,6,3,2,9]

      plt.plot(x,y)
```

```
[40]: [<matplotlib.lines.Line2D at 0x22114a42030>]
```

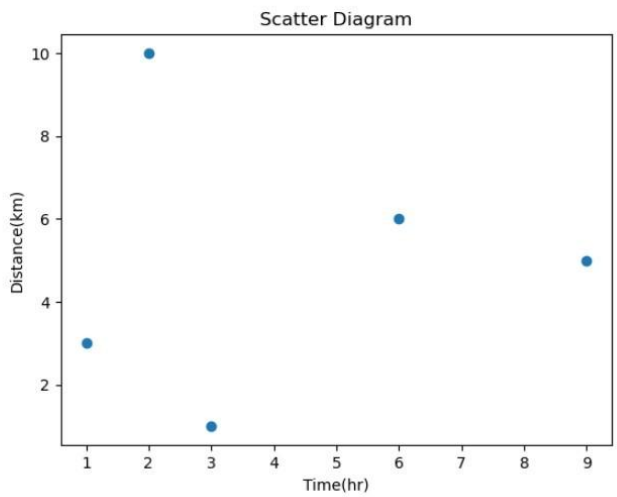



```
[42]: #Histogram

      h=[5,10,2,7,1]
      plt.hist(h)
      plt.show()
```



```
[48]: #Scatter Plot
      x=[3,6,1,2,9]
      y=[1,6,3,10,5]

      plt.scatter(x,y)
      plt.title("Scatter Diagram")

      #Labels
      plt.xlabel("Time(hr)")
      plt.ylabel("Distance(km)")
```

```
[48]: Text(0, 0.5, 'Distance(km)')
```




## ➢ Scikit

```
[53]: import pandas as pd
      from sklearn.datasets import load_wine

      wine_data=load_wine()

      #Conversion to pandas DataFrame
      wine_df=pd.DataFrame(wine_data.data,columns=wine_data.feature_names)

      #Add target label
      wine_df["target"]=wine_data.target

      #Preview
      wine_df.head()
```

[53]:

| | alcohol | malic_acid | ash | alcalinity_of_ash | magnesium | total_phenols | flavanoids | nonflavanoid_phenols | proanthocyanins | color_intensity | hue | od280/od315_of_dilute |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 14.23 | 1.71 | 2.43 | 15.6 | 127.0 | 2.80 | 3.06 | 0.28 | 2.29 | 5.64 | 1.04 | |
| 1 | 13.20 | 1.78 | 2.14 | 11.2 | 100.0 | 2.65 | 2.76 | 0.26 | 1.28 | 4.38 | 1.05 | |
| 2 | 13.16 | 2.36 | 2.67 | 18.6 | 101.0 | 2.80 | 3.24 | 0.30 | 2.81 | 5.68 | 1.03 | |
| 3 | 14.37 | 1.95 | 2.50 | 16.8 | 113.0 | 3.85 | 3.49 | 0.24 | 2.18 | 7.80 | 0.86 | |
| 4 | 13.24 | 2.59 | 2.87 | 21.0 | 118.0 | 2.80 | 2.69 | 0.39 | 1.82 | 4.32 | 1.04 | |

## Practical 5: Implement Perceptron algorithm for OR operation

```
[4]: import numpy as np

class Perceptron:
    def __init__(self, learning_rate=0.01, n_iterations=1):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.weights = None
        self.bias = None

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

        y_ = np.array([1 if i > 0 else 0 for i in y])

        for _ in range(self.n_iterations):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation_function(linear_output)

                update = self.learning_rate * (y_[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update

    def activation_function(self, x):
        return np.where(x >= 0, 1, 0)

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation_function(linear_output)
        return y_predicted


# OR gate inputs and outputs
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 1])

# Initialize and train the perceptron
perceptron = Perceptron(learning_rate=0.1, n_iterations=6)
perceptron.fit(X, y)

# Test the perceptron
predictions = perceptron.predict(X)
print(predictions)
```

```
[0 1 1 1]
```

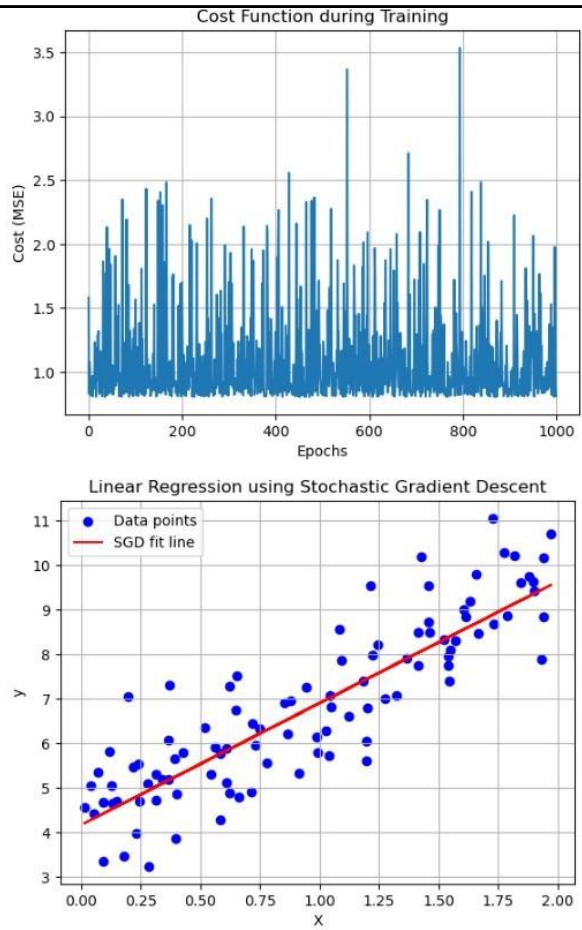
**Practical 6: Improve the prediction accuracy by estimating the weight values for the training data using stochastic gradient descent. (Perceptron)**

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

     # Generate synthetic data
     np.random.seed(42)
     X = 2 * np.random.rand(100, 1)
     y = 4 + 3 * X + np.random.randn(100, 1)

     def sgd(X, y, learning_rate=0.1, epochs=1000, batch_size=1):
         m = len(X)
         theta = np.random.randn(2, 1)  # Initialize parameters randomly

         # Add a bias term to X (X_0 = 1)
         X_bias = np.c_[np.ones((m, 1)), X]
         cost_history = []

         for epoch in range(epochs):
             # Shuffle the data at the beginning of each epoch
             indices = np.random.permutation(m)
             X_shuffled = X_bias[indices]
             y_shuffled = y[indices]

             for i in range(0, m, batch_size):
                 # Select a mini-batch or a single sample
                 X_batch = X_shuffled[i:i+batch_size]
                 y_batch = y_shuffled[i:i+batch_size]
```

```
                 # Compute the gradient
                 gradients = 2 / batch_size * X_batch.T.dot(X_batch.dot(theta) - y_batch)

                 # Update the parameters (theta)
                 theta -= learning_rate * gradients

             # Calculate and record the cost (Mean Squared Error) after each epoch
             predictions = X_bias.dot(theta)
             cost = np.mean((predictions - y) ** 2)
             cost_history.append(cost)

             # Print progress every 100 epochs
             if epoch % 100 == 0:
                 print(f"Epoch {epoch}, Cost: {cost:.4f}")

         return theta, cost_history

     # Train the model using SGD
     theta_final, cost_history = sgd(X, y, learning_rate=0.1, epochs=1000, batch_size=1)

     # Plot the cost history
     plt.plot(cost_history)
     plt.xlabel('Epochs')
     plt.ylabel('Cost (MSE)')
     plt.title('Cost Function during Training')
     plt.grid(True)
     plt.show()
```

```
# Plot the data and the regression line
plt.scatter(X, y, color='blue', label='Data points')
X_plot = np.c_[np.ones((X.shape[0], 1)), X]
plt.plot(X, X_plot.dot(theta_final), color='red', label='SGD fit line')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression using Stochastic Gradient Descent')
plt.legend()
plt.grid(True)
plt.show()
```

```
Epoch 0, Cost: 1.5818
Epoch 100, Cost: 1.5665
Epoch 200, Cost: 1.4445
Epoch 300, Cost: 1.7038
Epoch 400, Cost: 0.9102
Epoch 500, Cost: 0.8184
Epoch 600, Cost: 0.8352
Epoch 700, Cost: 0.8543
Epoch 800, Cost: 1.0508
Epoch 900, Cost: 0.8262
```



## Practical 7: Implement Adaline algorithm for AND operation

```
[1]: import numpy as np

class Adaline:
    def __init__(self, input_size, learning_rate=0.1, epochs=100):
        self.weights = np.zeros(input_size)
        self.bias = 0
        self.learning_rate = learning_rate
        self.epochs = epochs

    def activation(self, X):  # X is input
        return X

    def predict(self, X):
        return self.activation(np.dot(X, self.weights) + self.bias)

    def train(self, X, y):
        for epoch in range(self.epochs):
            for i in range(len(X)):
                prediction = self.predict(X[i])
                error = y[i] - prediction
                self.weights += self.learning_rate * error * X[i]
                self.bias += self.learning_rate * error

    def evaluate(self, X):
        return np.where(self.predict(X) >= 0.5, 1, 0)

# Training data for AND gate
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 0, 0, 1])

adaline = Adaline(input_size=2, learning_rate=0.1, epochs=100)
adaline.train(X, y)
predictions = adaline.evaluate(X)

for i, prediction in enumerate(predictions):
    print(f"Input: {X[i]} => Predicted: {prediction} => Actual: {y[i]}")
```

```
Input: [0 0] => Predicted: 0 => Actual: 0
Input: [0 1] => Predicted: 0 => Actual: 0
Input: [1 0] => Predicted: 0 => Actual: 0
Input: [1 1] => Predicted: 1 => Actual: 1
```

# Machine Learning Section

**Practical 1: Implementation of Features Extraction and Selection, Normalization, Transformation, Principal Components Analysis.**

## 1. Feature Extraction

```
[1]: from sklearn.feature_extraction.text import TfidfVectorizer

documents = ["machine learning is amazing", "deep learning is a part of machine learning"]
vectorizer = TfidfVectorizer()
X_tfidf = vectorizer.fit_transform(documents)

print("TF-IDF shape:", X_tfidf.shape)

TF-IDF shape: (2, 7)
```

## 2. Feature Selection

```
[2]: from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest, chi2

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

# Select top 2 features based on chi-square test
X_selected = SelectKBest(chi2, k=2).fit_transform(X, y)

print("Selected Features shape:", X_selected.shape)

Selected Features shape: (150, 2)
```

## 3. Normalization

```
[3]: from sklearn.preprocessing import Normalizer

normalizer = Normalizer()
X_normalized = normalizer.fit_transform(X)

print("Normalized data (first sample):", X_normalized[0])

Normalized data (first sample): [0.80377277 0.55160877 0.22064351 0.0315205 ]
```

## 4. Transformation

```
[4]: from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)

     print("Standardized data (first sample):", X_scaled[0])

     Standardized data (first sample): [-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
```

## 5. Principal Component Analysis

```
[5]: from sklearn.decomposition import PCA

     # Reduce to 2 principal components
     pca = PCA(n_components=2)
     X_pca = pca.fit_transform(X_scaled)

     print("PCA transformed shape:", X_pca.shape)

     PCA transformed shape: (150, 2)
```

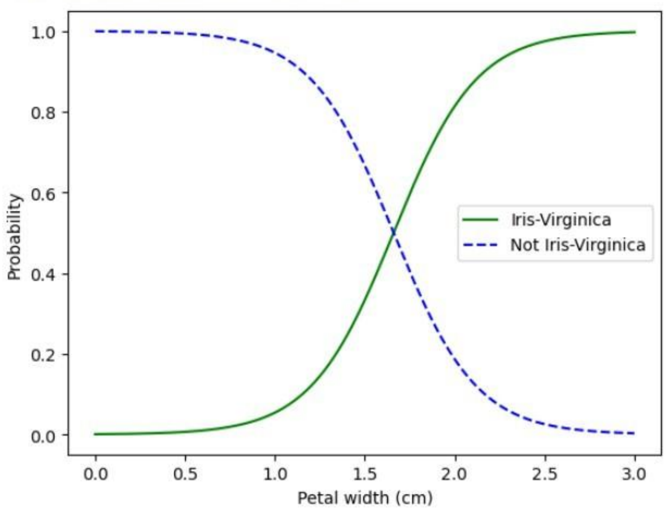
# Practical 2: Implementation of Logistic regression

```
[21]: #Problem Statement 1: Build and train a Logistic Regression Model to do binary classification of iris flowers using the iris dataset.

import numpy as np
from sklearn import datasets

iris = datasets.load_iris()
print(type(iris))
print(list(iris.keys()))
X = iris["data"][:,3:] # petal width
y = (iris["target"] == 2).astype(np.int64) # 1 if Iris-Virginica, else 0

<class 'sklearn.utils._bunch.Bunch'>
['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module']
```

```
[5]: from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(solver="lbfgs", random_state=42)
log_reg.fit(X,y)

[5]: LogisticRegression
LogisticRegression(random_state=42)
```

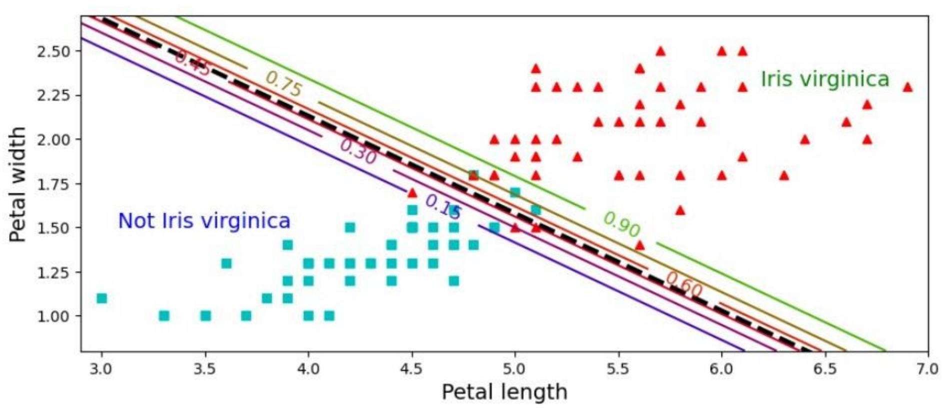
```
[17]: import matplotlib.pyplot as plt
X_new = np.linspace(0,3,1000).reshape(-1,1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:,1],"g-")
plt.plot(X_new, y_proba[:,0], "b--")
plt.xlabel('Petal width (cm)')
plt.ylabel('Probability')
plt.legend(['Iris-Virginica','Not Iris-Virginica'])
```

```
[17]: <matplotlib.legend.Legend at 0x1926b602390>
```




```
[9]: log_reg.predict([[1.7],[1.5]])

[9]: array([1, 0], dtype=int64)
```

```
[19]: #Problem Statement 2: Logistic Regression for predicting class using two features: Petal length and width.

from sklearn.linear_model import LogisticRegression

X = iris["data"][:, (2, 3)]  # petal length, petal width
y = (iris["target"] == 2).astype(np.int64)

log_reg2 = LogisticRegression(solver="lbfgs", C=10**10, random_state=42)
log_reg2.fit(X, y)

x0, x1 = np.meshgrid(
        np.linspace(2.9, 7, 500).reshape(-1, 1),
        np.linspace(0.8, 2.7, 200).reshape(-1, 1),
    )
X_new = np.c_[x0.ravel(), x1.ravel()]
print(X_new.shape)

y_proba = log_reg2.predict_proba(X_new)

plt.figure(figsize=(10, 4))
plt.plot(X[y==0, 0], X[y==0, 1], "cs")
plt.plot(X[y==1, 0], X[y==1, 1], "r^")
zz = y_proba[:, 1].reshape(x0.shape)
contour = plt.contour(x0, x1, zz, cmap=plt.cm.brg)
left_right = np.array([2.9, 7])
boundary = -(log_reg2.coef_[0][0] * left_right + log_reg2.intercept_[0]) / log_reg2.coef_[0][1]
plt.clabel(contour, inline=1, fontsize=12)
plt.plot(left_right, boundary, "k--", linewidth=3)
plt.text(3.5, 1.5, "Not Iris virginica", fontsize=14, color="b", ha="center")
plt.text(6.5, 2.3, "Iris virginica", fontsize=14, color="g", ha="center")
plt.xlabel("Petal length", fontsize=14)
plt.ylabel("Petal width", fontsize=14)
plt.axis([2.9, 7, 0.8, 2.7])

(100000, 2)

[19]: (2.9, 7.0, 0.8, 2.7)
```



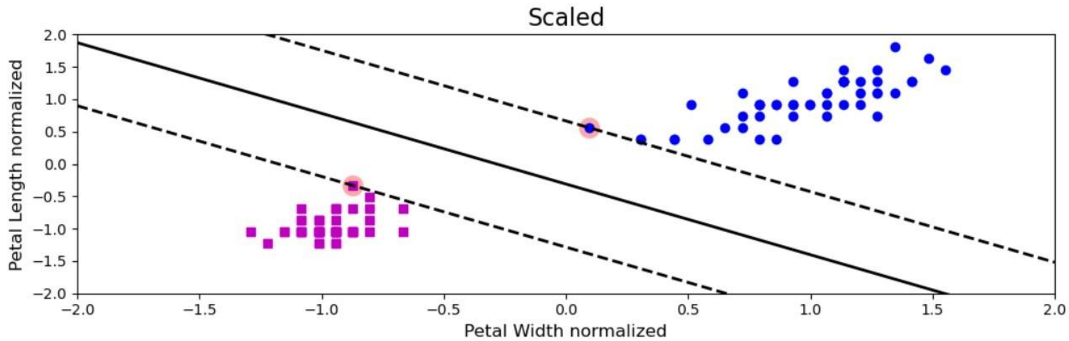
## Practical 3: Implementation of Classifying data using Support Vector Machine (SVM)- Linear and Non-Linear SVM Classification

### ➢ Linear SVM

```
[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)
```

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

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

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

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

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

Predicted class: 1
```

```
[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])

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

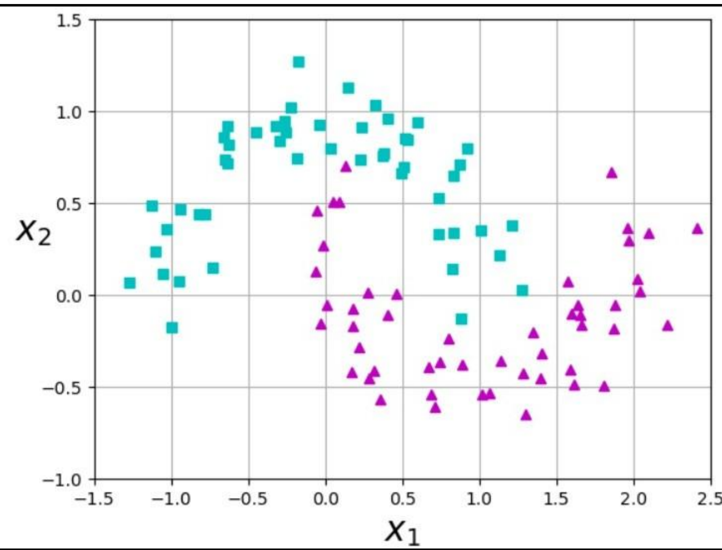



## ➢ Non-Linear SVM

```
[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
```

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

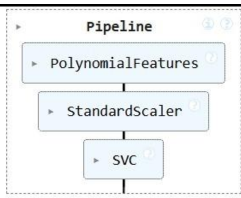
```
[4]: from sklearn.datasets import make_moons
     X, y = make_moons(n_samples=100, noise=0.15, random_state=42)
     #define a function to plot the dataset
     def plot_dataset(X, y, axes):
         plt.plot(X[:, 0][y==0], X[:, 1][y==0], "cs")
         plt.plot(X[:, 0][y==1], X[:, 1][y==1], "m^")
         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()
```



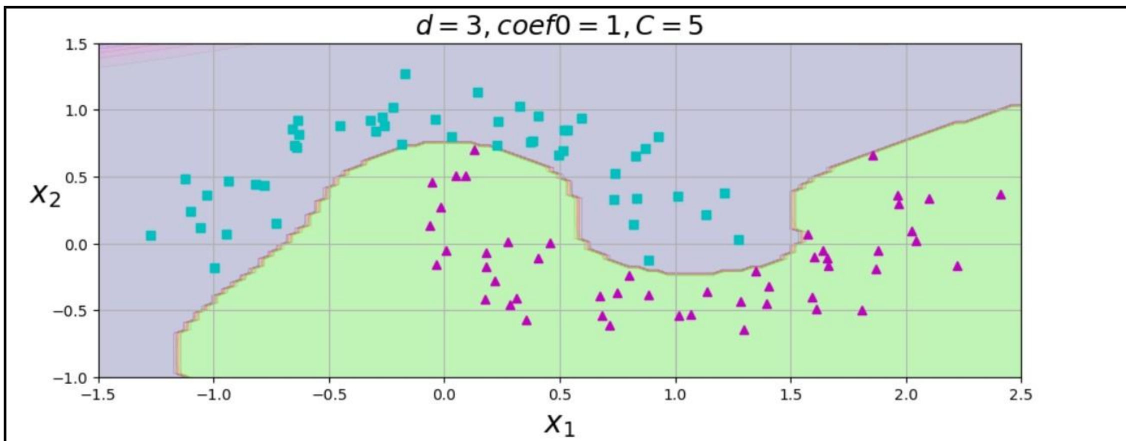
```
[7]: #define a function plot the decision boundaries
     def plot_predictions(clf, axes):
         #create data in continous linear space
         x0s = np.linspace(axes[0], axes[1], 100)
         x1s = np.linspace(axes[2], axes[3], 100)
         x0, x1 = np.meshgrid(x0s, x1s)
         X = np.c_[x0.ravel(), 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)
```

```
[9]: #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)
```

[9]:




```
[11]: #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()
```



## Practical 4: Implement Elbow method for K means Clustering

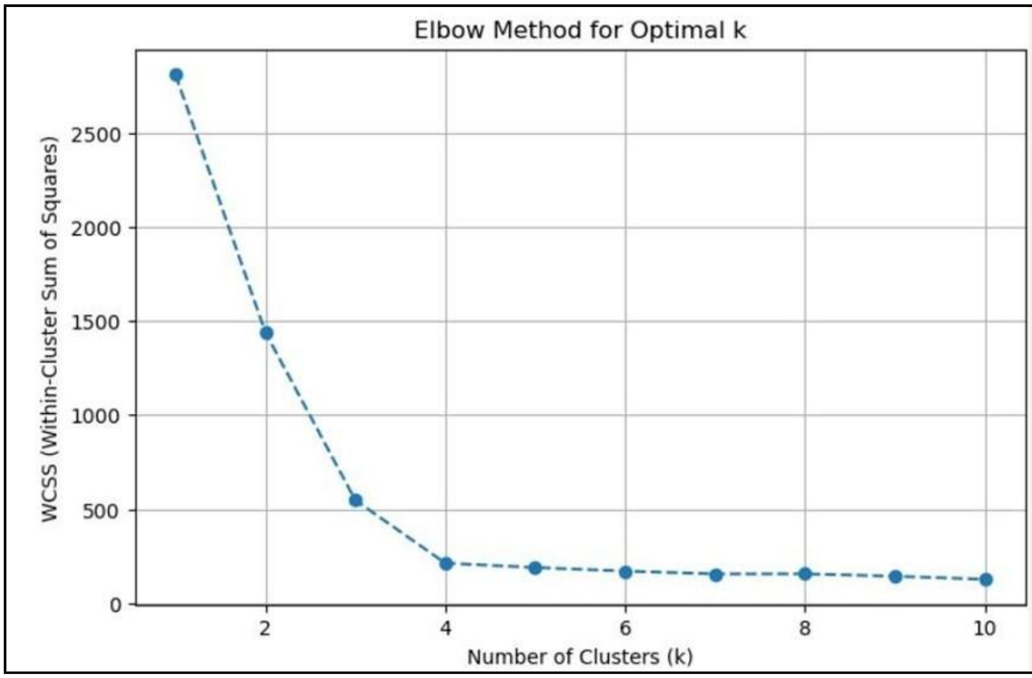
```
[1]: !pip install --user threadpoolctl==3.1.0

Collecting threadpoolctl==3.1.0
  Downloading threadpoolctl-3.1.0-py3-none-any.whl.metadata (9.2 kB)
Downloading threadpoolctl-3.1.0-py3-none-any.whl (14 kB)
Installing collected packages: threadpoolctl
Successfully installed threadpoolctl-3.1.0
```

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

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

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

# Drop missing values
df_cleaned = df.dropna()

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

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

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

# Plot the Elbow Method

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

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

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

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

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

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

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

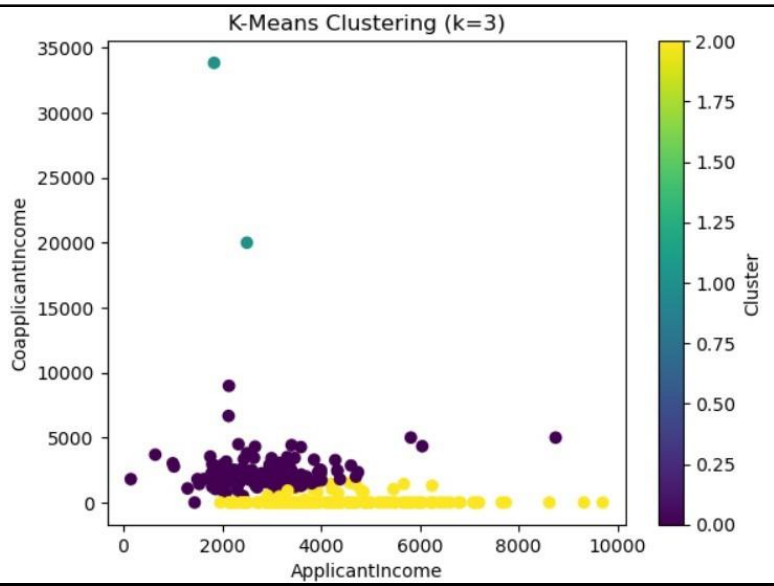
```
   Credit_History Property_Area Loan_Status
0             1.0         Rural           N
1             1.0         Urban           Y
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
Numerical columns used for clustering: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




Number of Clusters

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

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

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



## Practical 5: Implementation of Bagging Algorithm: Random Forest

```
[1]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.datasets import load_iris
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score
     from sklearn.decomposition import PCA

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

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

     # Initialize and train the Random Forest Classifier
     rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
     rf_classifier.fit(X_train, y_train)

     # Make predictions
     y_pred = rf_classifier.predict(X_test)

     # Evaluate the model
     accuracy = accuracy_score(y_test, y_pred)
     print(f'Accuracy of Random Forest Classifier: {accuracy * 100:.2f}%')
```

```
Accuracy of Random Forest Classifier: 100.00%
```

## Practical 6: Implementation of Boosting Algorithms: AdaBoost, Stochastic Gradient Boosting, Voting Ensemble

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

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

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

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

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

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

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

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

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

```
Accuracy of Random Forest Classifier: 100.00%
Accuracy of AdaBoost Classifier: 100.00%
Accuracy of Gradient Boosting Classifier: 100.00%
```

```
Accuracy of Voting Classifier: 100.00%
```