BHARATI VIDYAPEETH'S

INSTITUTE OF COMPUTER APPLICATIONS & MANAGEMENT

(Affiliated to Guru Gobind Singh Indraprastha University,

Approved by AICTE, New Delhi)

Artificial Intelligence and Machine Learning

(MCA- 263)

Practical File

Submitted To:

Dr. Rakhee Sharma

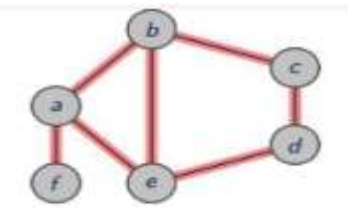
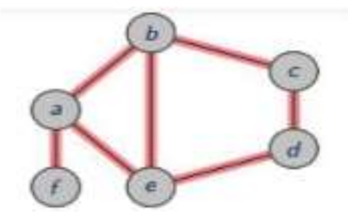
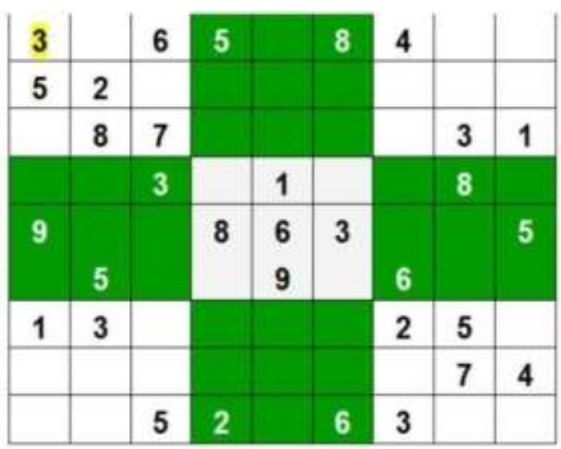
(Associate Professor)

Submitted By:

Gaurav Prakash(09811604422)

MCA 3rd Sem, Sec 2

INDEX

<table>
<tr><th>S. No.</th><th>Problem Description</th><th>Date of Execution</th><th>Sign.</th></tr>
<tr><td>AP1</td><td>Create a solution to solve the Graph Traversal using BFS?

</td><td>16-09-2022</td><td></td></tr>
<tr><td>AP2</td><td>Given a snake and ladder board, find the minimum number of dice throws to reach the destination cell starting from the source using BFS?</td><td>17-09-2022</td><td></td></tr>
<tr><td>AP3</td><td>Create a solution to solve the graph traversal using DFS?

</td><td>18-09-2022</td><td></td></tr>
<tr><td>AP4</td><td>Create a solution to solve the following Sudoku using DFS?

</td><td>19-09-2022</td><td></td></tr>
<tr><td>BP1</td><td>The Towers of Hanoi
Three vertical pegs (henceforth “towers”) stand tall. We will label them A, B, and C. Doughnut-shaped discs are around tower A. The widest disc is at the bottom, and we will call it disc 1. The rest of the discs above disc 1 are labelled with increasing numerals and get progressively narrower. For instance, if we were to work with three discs, the widest disc, the one on the bottom, would be 1. The next widest disc, disc 2, would sit on top of disc 1. And finally, the narrowest disc, disc 3, would sit on top of disc 2.
Our goal is to move all of the discs from tower A to tower C.
Given the following constraints: Only one disc can be moved at a time. The topmost disc of any tower is the only one available for moving. A wider disc</td><td>19-09-2022</td><td></td></tr>
</table>

can never be atop a narrower disc.

Solve this problem

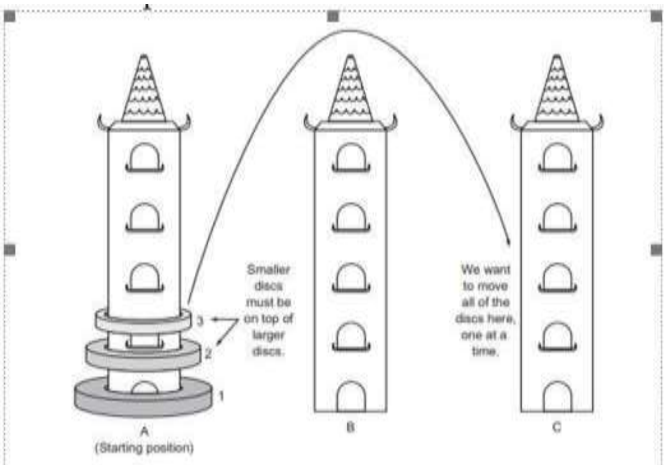
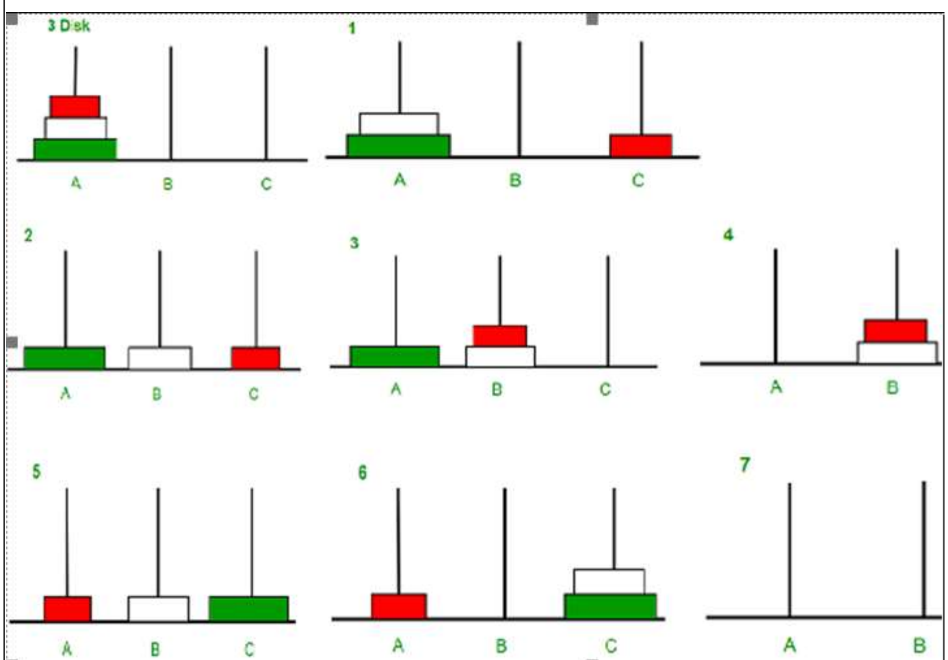
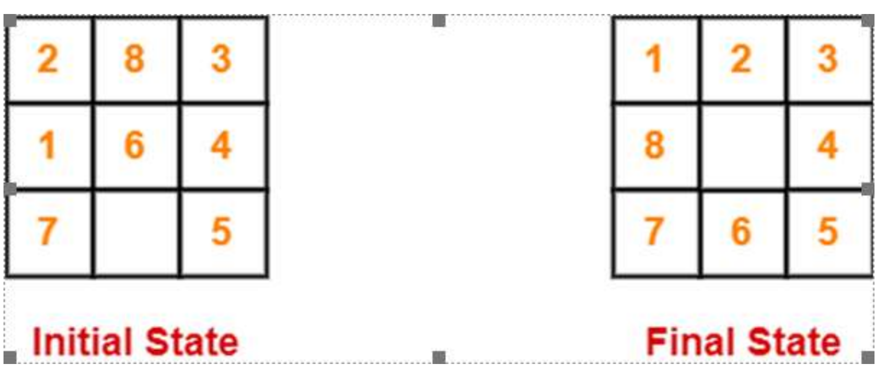


Figure 1.7 The challenge is to move the three discs, one at a time, from tower

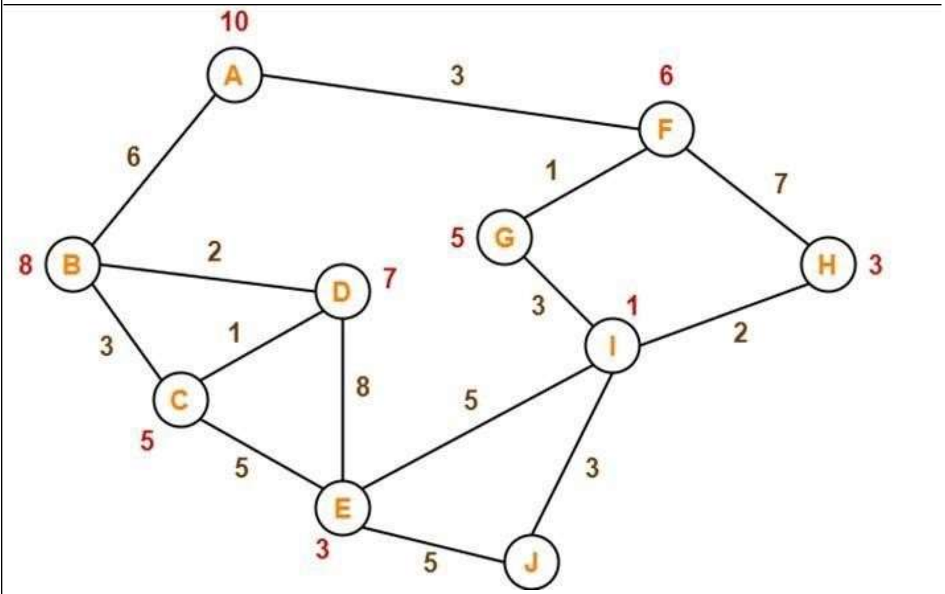


BP2 Given an initial state of a 8-puzzle problem and final state to be reached- 21-09-2022



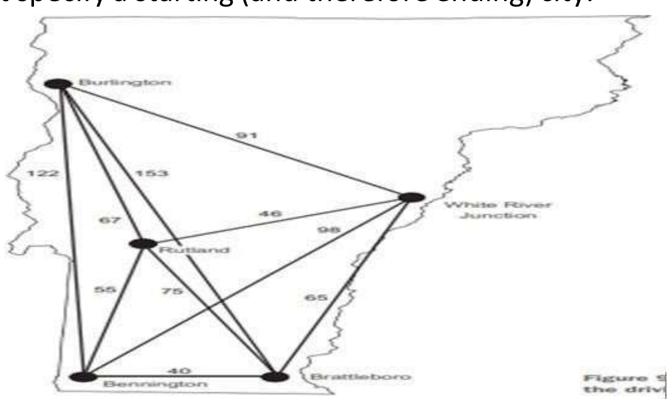
Find the most cost-effective path to reach the final state from initial state using A* Algorithm. F(n)=g(n)+h(n). Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.

BP3 | Consider the following graph | 25-09-2022



The numbers written on edges represent the distance between the nodes. The numbers written on nodes represent the heuristic value. Find the most cost-effective path to reach from start state A to final state J using A* Algorithm.

BP4 | The salesman is interested in visiting five of the major cities of Vermont. We will not specify a starting (and therefore ending) city. | 29-10-2022

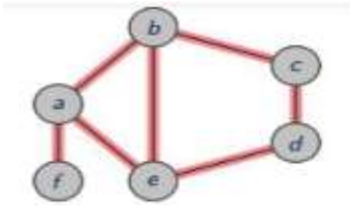


	Rutland	Burlington	White River Junction	Bennington	Brattleboro
Rutland	0	67	46	55	75
Burlington	67	0	91	122	153
White River Junction	46	91	0	98	65
Bennington	55	122	98	0	40
Brattleboro	75	153	65	40	0

CP1	Create a solution to load the IRIS dataset from the following URL: "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data". Prepare the data, evaluate the algorithms and present the results through suitable visualizations?	01-11-2022	
CP2	Using Scikit-learn, split the iris dataset into 80% train data and 20% test data. Train or fit the data into the model and using the K Nearest Neighbor Algorithm and create a plot of k values vs accuracy.	03-11-2022	
CP3	Clean the Oil Spill dataset from the following URL: https://github.com/jbrownlee/Datasets/blob/master/oilspill.csv. Clean the data of duplicate data, single value columns and low variance columns. Once the data is prepared, evaluate it on the classification algorithms in CP1 and present the result through suitable visualizations.	05-11-2022	
DP1	Load the Boston housing dataset directly via URL and split it into train and test sets, then estimate the mean squared error (MSE) for a linear regression model. Estimate the bias and variance for the linear regression model?	06-11-2022	
DP2	Use the Iris Dataset of CP1. The dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica and Iris versicolor).use KFolds cross-validation with 20 folds (K=20) to evaluate the generalization ability of our model. Within each fold we will estimate the training and test error using the training and test sets, respectively. Plot the MAE of the training phase and the MAE of the testing phase. Interpret the results and try to spot the overfitting and underfitting points?	11-11-2022	
EP1	Using linear regression predict the relationship between the experience of an individual and his salary. Predict the variance and bias for the same?	15-11-2022	
EP2	Predict the CO2 emission of a car based on the size of the engine, but use multiple regression so we can throw in more variables, like the weight of the car?	18-11-2022	
EP3	Plot the CO2 emission values wrt engine size using multiple linear regression?	20-11-2022	
EP4	Apply Linear Regression and build a model that studies the relationship between the head size and the brain weight of an individual? Evaluate by using least square regression method where RMSE (root mean squared error) and R-squared/R2 will be the model evaluation parameters	22-11-2022	
EP5	Modify EP1 to calculate MSE, RMSE and R2 as the model evaluation parameters.	23-11-2022	
EP6	Demonstrate odds ratio and log of odds on a dataframe for winning and losing?	24-11-2022	
EP7	Apply logistic regression to the load-digits dataset of the sklearn library? Create a confusion matrix for the model and also generate the classification report?	24-11-2022	
EP8	Generate univariate baby weight data and apply linear regression. Evaluate the model by calculating SSE, SST, and R2.	25-11-2022	

EP9	Apply logistic regression on userdata.csv dataset to predict the users who may be potential customers to purchase a SUV car? Also generate the confusion matrix to evaluate your model?	25-11-2022	
EP10	Apply logistic regression on handwritten digits dataset to classify the digits. Evaluate your model too?	26-11-2022	
FP1	Understand dimensionality reduction technique?	27-11-2022	
FP2	Implement dimensionality reduction on wines.csv using PCA?	28-11-2022	
FP3	Create a basic visualization of Iris dataset in question CP1 using PCA?	28-11-2022	
GP1	Create a random dataset using the make_blobs() function from sklearn and apply K-means on the same after deciding the number of clusters using the elbow method?	29-11-2022	
GP2	Create a mall_customer_dataset.csv dataset and apply the K-means on the same after deciding the number of clusters using the elbow method to uncover the patterns?	30-11-2022	
HP1	Use the Pima Indian diabetes database to perform ensemble predictions using the following bagging classifiers: Bagged Decision Trees, Random Forest Classifier and Extra trees?	01-01-2022	
HP2	Use the same Pima Indian diabetes database of HP1 to perform ensemble predictions using the following boosting classifiers: AdaBoost, Stochastic Gradient Boosting?	01-01-2022	
IP1	Implement a simple neuron using the sigmoid activation function and feed forward algorithm?	02-01-2022	
IP2	Implement a simple neural network with: - 2 inputs - A hidden layer with 2 neurons (h1, h2) - An output layer with 1 neuron (o1)	02-01-2022	
JP1	Build a simplified clone of IMDB Top 250 movies using metadata collection from IMDB. The following are the steps involved: -Decide on the metric or score to rate movies on -Calculate the score for every movie -Sort the movies based on the score and output the top results. -Use the Full Movie Lens Dataset.	03-01-2022	
JP2	Build a system that recommends movies that are similar to a particular movie. Compute pairwise cosine similarity scores for all movies based on that similarity score threshold. The plot description is available to you as the overview feature in your metadata dataset.	03-01-2022	

AP1. Create a solution to solve the Graph Traversal using BFS?




```
graph = {
  'a' : ['b','e','f'],
  'b' : ['a', 'e','c'],
  'c' : ['b','d'],
  'd' : ['c','e'],
  'e' : ['a','b','d'],
  'f' : ['a']
}

visited = []
queue = []

def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        m = queue.pop(0)
        print (m, end = " ")

        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

print("The Breadth-First Search Traversal(starting from b):")
 bfs(visited, graph, 'b')
```

OUTPUT

```
>>>
==================== RESTART: E:/MCA/sem 3/AI&ML/Ap1 dfs.py ====================
The Breadth-First Search Traversal(starting from b):
b a e c f d
>>>
```

AP2. Given a snake and ladder board, find the minimum number of dice throws to reach the destination cell starting from the source using BFS?

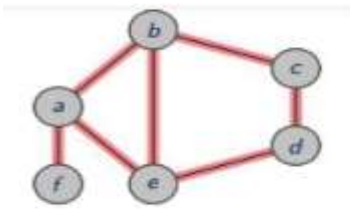
```
class QueueEntry(object):
        def __init_(self, v=0, dist=0):
                self.v = v
                self.dist = dist


def getMinDiceThrows(move, N):
        visited = [False] * N
        queue = []
        visited[0] = True
        queue.append(QueueEntry(0, 0))
        qe = QueueEntry()

        while queue:
                qe = queue.pop(0)
                v = qe.v
                if v == N - 1:
                        break
                j = v + 1

                while j <= v + 6 and j < N:
                        if visited[j] is False:
                                a = QueueEntry()
                                a.dist = qe.dist + 1
                                visited[j] = True
                                a.v = move[j] if move[j] != -1 else j

                                queue.append(a)

                        j += 1
        return qe.dist

N = 30
moves = [-1] * N
moves[2] = 21     #Ladder
moves[4] = 7      #Ladder
moves[10] = 25    #Ladder
moves[19] = 28    #Ladder
moves[26] = 0     #Snake
moves[20] = 8     #Snake
```

```
moves[16] = 3      #Snake
moves[18] = 6      #Snake

print("Mininum dice throws required is {0}".format(getMinDiceThrows(moves, N)))
```

OUTPUT

```
>>>
=================== RESTART: C:/Python310/Ap2 snake&ladder.py ==================
Mininum dice throws required is 3
>>>
```

AP3. Create a solution to solve the Graph Traversal using DFS?




```
graph = {
  'a' : ['b','e','f'],
  'b' : ['a', 'c','e'],
  'c' : ['b','d'],
  'd' : ['c','e'],
  'e' : ['a','b','d'],
  'f' : ['a']
}

visited = set()

def dfs(visited, graph, node):
    if node not in visited:
        print(node, end=" ")
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited,graph,neighbour)

print("The Depth-First Search Traversal(starting from b):")
dfs(visited, graph, 'b')
```

OUTPUT

```
>>>
====================== RESTART: C:/Python310/Ap3 dfs.py ======================
The Depth-First Search Traversal(starting from b):
b a e d c f
>>>
```

AP4. Create a solution to solve the following Sudoku using DFS?

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

```
N = 9
def printing(arr):
    for i in range(N):
        for j in range(N):
            print(arr[i][j], end = " ")
        print()
def isSafe(grid, row, col, num):
    for x in range(9):
        if grid[row][x] == num:
            return False
    for x in range(9):
        if grid[x][col] == num:
            return False
    startRow = row - row % 3
    startCol = col - col % 3
    for i in range(3):
        for j in range(3):
            if grid[i + startRow][j + startCol] == num:
                return False
    return True
def solveSudoku(grid, row, col):
    if (row == N - 1 and col == N):
        return True
    if col == N:
        row += 1
        col = 0
    if grid[row][col] > 0:
        return solveSudoku(grid, row, col + 1)
    for num in range(1, N + 1, 1):
        if isSafe(grid, row, col, num):
            grid[row][col] = num
            if solveSudoku(grid, row, col + 1):
                return True
        grid[row][col] = 0
    return False
```

```
grid = [[3, 0, 6, 5, 0, 8, 4, 0, 0],
        [5, 2, 0, 0, 0, 0, 0, 0, 0],
        [0, 8, 7, 0, 0, 0, 0, 3, 1],
        [0, 0, 3, 0, 1, 0, 0, 8, 0],
        [9, 0, 0, 8, 6, 3, 0, 0, 5],
        [0, 5, 0, 0, 9, 0, 6, 0, 0],
        [1, 3, 0, 0, 0, 0, 2, 5, 0],
        [0, 0, 0, 0, 0, 0, 0, 7, 4],
        [0, 0, 5, 2, 0, 6, 3, 0, 0]]

if (solveSudoku(grid, 0, 0)):
    printing(grid)
else:
    print("no solution exists ")
```

OUTPUT

```
>>>
================== RESTART: E:/MCA/sem 3/AI&ML/Ap4 suddko.py ==================
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
4 8 7 6 2 9 5 3 1
2 6 3 4 1 5 9 8 7
9 7 4 8 6 3 1 2 5
8 5 1 7 9 2 6 4 3
1 3 8 9 4 7 2 5 6
6 9 2 3 5 1 8 7 4
7 4 5 2 8 6 3 1 9
>>>
```

BP1. The Towers of Hanoi

Three vertical pegs (henceforth "towers") stand tall. We will label them A, B, and C. Doughnut-shaped discs are around tower A. The widest disc is at the bottom, and we will call it disc 1. The rest of the discs above disc 1 are labelled with increasing numerals and get progressively narrower. For instance, if we were to work with three discs, the widest disc, the one on the bottom, would be 1. The next widest disc, disc 2, would sit on top of disc 1. And finally, the narrowest disc, disc 3, would sit on top of disc 2.

Our goal is to move all of the discs from tower A to tower C.

Given the following constraints: Only one disc can be moved at a time. The topmost disc of any tower is the only one available for moving. A wider disc can never be atop a narrower disc.

Solve this problem

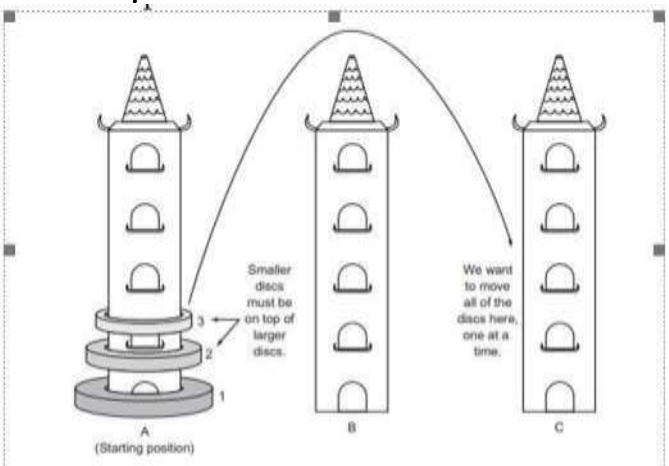
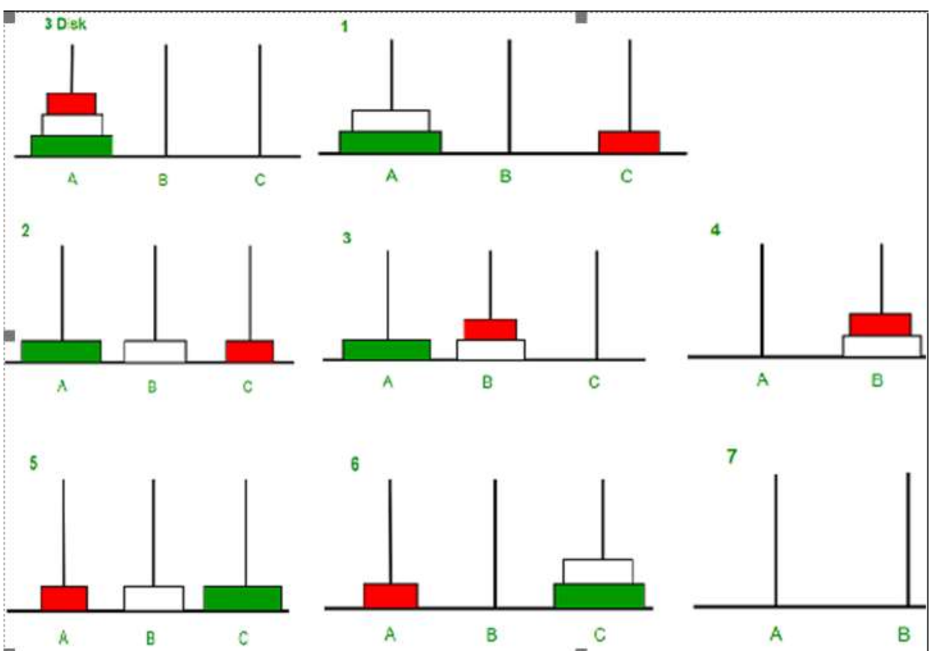


Figure 1.7 The challenge is to move the three discs, one at a time, from tower

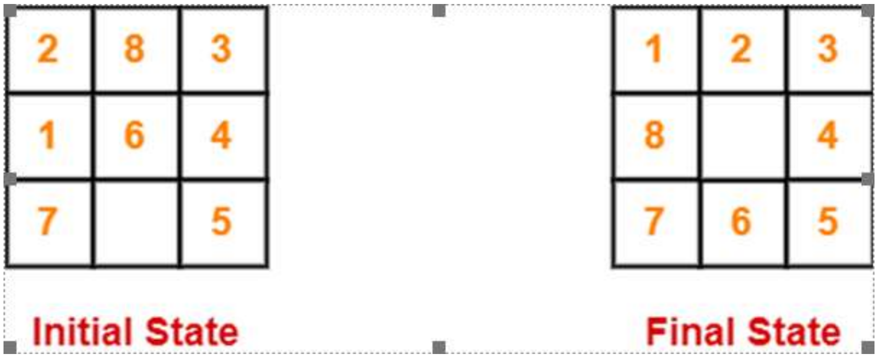


```
def tower_of_hanoi(disks, source, auxiliary, target):
   if(disks == 1):
      print('Move disk 1 from tower {} to tower {}.'.format(source, target))
      return
   tower_of_hanoi(disks - 1, source, target, auxiliary)
   print('Move disk {} from tower {} to tower {}.'.format(disks, source, target))
   tower_of_hanoi(disks - 1, auxiliary, source, target)


disks = int(input('Enter the number of disks: \n'))
tower_of_hanoi(disks, 'A', 'B', 'C')
```

OUTPUT

```
>>>
===================== RESTART: C:\Python310\Bp1_tower_of_hanoi.py =====================
Enter the number of disks:
3
Move disk 1 from tower A to tower C.
Move disk 2 from tower A to tower B.
Move disk 1 from tower C to tower B.
Move disk 3 from tower A to tower C.
Move disk 1 from tower B to tower A.
Move disk 2 from tower B to tower C.
Move disk 1 from tower A to tower C.
>>>
                                                                        Ln: 142   Col: 0
```

BP2. Given an initial state of 8-puzzle problem and final state to be reached-



Find the most cost-effective path to reach the final state from initial state using A* Algorithm. f(n)=g(n)+h(n). Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.

```
class Node:
    def__init _(self,data,level,fval):
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        x,y = self.find(self.data,'_')
        val_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
        children = []
        for i in val_list:
            child = self.shuffle(self.data,x,y,i[0],i[1])
            if child is not None:
                child_node = Node(child,self.level+1,0)
                children.append(child_node)
        return children

    def shuffle(self,puz,x1,y1,x2,y2):
        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):
            temp_puz = []
            temp_puz = self.copy(puz)
            temp = temp_puz[x2][y2]
            temp_puz[x2][y2] = temp_puz[x1][y1]
            temp_puz[x1][y1] = temp
            return temp_puz
        else:
            return None

    def copy(self,root):
        temp = []
        for i in root:
            t = []
```

```
            for j in i:
                t.append(j)
            temp.append(t)
        return temp

    def find(self,puz,x):
        for i in range(0,len(self.data)):
            for j in range(0,len(self.data)):
                if puz[i][j] == x:
                    return i,j

class Puzzle:
    def __init__(self,size):
        self.n = size
        self.open = []
        self.closed = []

    def accept(self):
        puz = []
        for i in range(0,self.n):
            temp = input().split(" ")
            puz.append(temp)
        return puz

    def f(self,initial,final):
        return self.h(initial.data,final)+initial.level

    def h(self,initial,final):
        temp = 0
        for i in range(0,self.n):
            for j in range(0,self.n):
                if initial[i][j] != final[i][j] and initial[i][j] != '_':
                    temp += 1
        return temp

    def process(self):
        print("Enter the initial state matrix \n")
        initial = self.accept()
        print("Enter the final state matrix \n")
        final = self.accept()
        initial = Node(initial,0,0)
        initial.fval = self.f(initial,final)
        self.open.append(initial)
        print("\nThe most cost-effective path to reach the final state from initial state using A* Algorithm:
\n")
        while True:
            cur = self.open[0]
            print("")
            print("  | ")
            print("  | ")
            print(" \\\'/ \n")
```

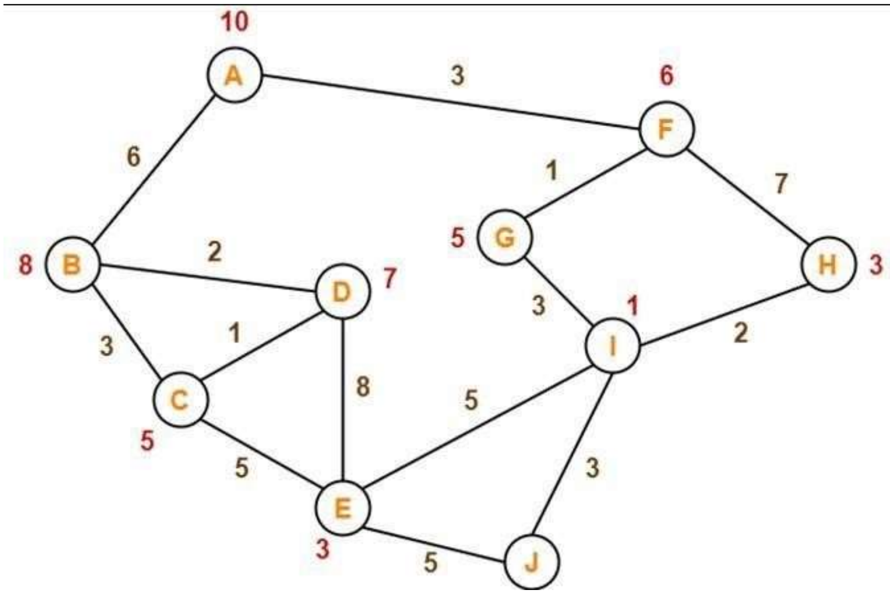
```
            for i in cur.data:
                for j in i:
                    print(j,end=" ")
                print("")
            if(self.h(cur.data,final) == 0):
                break
            for i in cur.generate_child():
                i.fval = self.f(i,final)
                self.open.append(i)
            self.closed.append(cur)
            del self.open[0]
            self.open.sort(key = lambda x:x.fval,reverse=False)

puz = Puzzle(3)
puz.process()
```

OUTPUT

```
>>>
========================= RESTART: C:/Python310/BP2_8_puzzle.py =========================
Enter the initial state matrix:

2 8 3
1 6 4
7 _ 5
Enter the final state matrix:

1 2 3
8 _ 4
7 6 5

The most cost-effective path to reach the final state from initial state using A* Algori
thm:


  |
  |
 \'/

2 8 3
1 6 4
7 _ 5

  |
  |
 \'/

2 8 3
1 _ 4
7 6 5

  |
  |
 \'/

2 8 3
_ 1 4
7 6 5
```

```
   |
   |
  \'/

2 _ 3
1 8 4
7 6 5

   |
   |
  \'/

_ 2 3
1 8 4
7 6 5

   |
   |
  \'/

1 2 3
_ 8 4
7 6 5

   |
   |
  \'/

1 2 3
8 _ 4
7 6 5
>>>
```

BP3. Consider the following graph



The numbers written on edges represent the distance between the nodes. The numbers written on nodes represent the heuristic value. Find the most cost-effective path to reach from start state A to final state J using A* Algorithm.

```
def aStarAlgo(start_state, final_state):
    open_set = set(start_state)
    closed_set = set()
    g = {}
    parents = {}
    g[start_state] = 0
    parents[start_state] = start_state
    while len(open_set) > 0:
        n = None
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == final_state or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] =  n
                    g[m] = g[n] + weight
                else:
                    if g[m] > g[n] + weight:
                        g[m]  = g[n]  + weight
                        parents[m] = n
                        if m in closed_set:
```

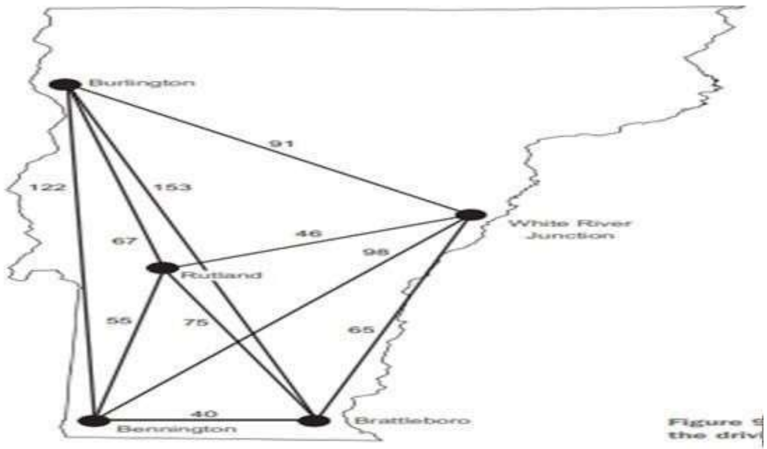
```
                    closed_set.remove(m)
                    open_set.add(m)
        if n == None:
            print('Path does not exist!')
            return None


        if n == final_state:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_state)
            path.reverse()
            print(' The most cost-effective path to reach from start state A to final state J using A* Algorithm:
{}'.format(path))
            return path
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None

def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None


def heuristic(n):
    H_dist = {
        'A': 10,
        'B': 8,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
    return H_dist[n]

Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
    'F': [('A', 3), ('G', 1), ('H', 7)],
```

```
    'G': [('F', 1), ('I', 3)],
    'H': [('F', 7), ('I', 2)],
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
}

aStarAlgo('A', 'J')
```

OUTPUT

```
>>>
========================== RESTART: C:/Python310/BP3_graph.py ==========================
The most cost-effective path to reach from start state A to final state J using A* Algor
ithm: ['A', 'F', 'G', 'I', 'J']
>>>
```

BP4. The salesman is interested in visiting five of the major cities of Vermont. We will not specify a starting (and therefore ending) city.



	Rutland	Burlington	White River Junction	Bennington	Brattleboro
Rutland	0	67	46	55	75
Burlington	67	0	91	122	153
White River Junction	46	91	0	98	65
Bennington	55	122	98	0	40
Brattleboro	75	153	65	40	0

```
routes = []


def find_paths(node, cities, path, distance):
    path.append(node)
    if len(path) > 1:
        distance += cities[path[-2]][node]
    if (len(cities) == len(path)) and (path[0] in cities[path[-1]]):
        global routes
        path.append(path[0])
        distance += cities[path[-2]][path[0]]
        #print (path, distance)
        routes.append([distance, path])
        return


    for city in cities:
        if (city not in path) and (node in cities[city]):
            find_paths(city, dict(cities), list(path), distance)



cities = {
        'Rutland': {'Rutland': 0, 'Burlington': 67, 'White River Junction': 46, 'Bennington': 55, 'Brattleboro': 75},
```

```
    'Burlington': {'Rutland': 67, 'Burlington': 0, 'White River Junction': 91, 'Bennington': 122,
'Brattleboro':153},
    'White River Junction': {'Rutland': 46, 'Burlington': 91, 'White River Junction': 0, 'Bennington': 98,
'Brattleboro':65},
    'Bennington': {'Rutland': 55, 'Burlington': 122, 'White River Junction': 98, 'Bennington': 0,
'Brattleboro': 40},
    'Brattleboro': {'Rutland': 75, 'Burlington': 153, 'White River Junction': 65, 'Bennington': 40,
'Brattleboro': 0},
  }

print ("Starting city: Burlington")
find_paths('Burlington', cities, [], 0)
print ("\n")
routes.sort()
if len(routes) != 0:
  print ("Minimum cost: {} \nShortest route: {}".format(routes[0][0],routes[0][1]))
else:
  print ("FAIL!")
```

OUTPUT

```
>>>
    ================== RESTART: C:/Python310/BP4_travelling_salesman.py ===================
    Starting city: Burlington


    Minimum cost: 318
    Shortest route: ['Burlington', 'Rutland', 'Bennington', 'Brattleboro', 'White River Junc
    tion', 'Burlington']
>>>
```

CP1. Create a solution to load the IRIS dataset from the following URL: "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data". Prepare the data, evaluate the algorithms and present the results through suitable visualizations?

```
#Load Libraries
from pandas import read_csv
from pandas.plotting import scatter_matrix
from pandas import set_option
from pandas import DataFrame
from pandas import concat
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

# Load dataset
#url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.cs
v"
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-
width', 'class']
dataset = read_csv(url, names=names)

# locate rows of duplicate data
# calculate duplicates
dups = dataset.duplicated()
# report if there are any duplicates
print(dups.any())
# list all duplicate rows
print(dataset[dups])
```

```
True
     sepal-length  sepal-width  petal-length  petal-width           class
34            4.9          3.1           1.5          0.1     Iris-setosa
37            4.9          3.1           1.5          0.1     Iris-setosa
142           5.8          2.7           5.1          1.9  Iris-virginica
```

```
# delete rows of duplicate data from the dataset
print(dataset.shape)
# delete duplicate rows
```

```
dataset.drop_duplicates(inplace=True)
print(dataset.shape)
```

```
(150, 5)
(147, 5)
```

```
# head, peek your dataset, see first 10 rows
print(dataset.head(10))
```

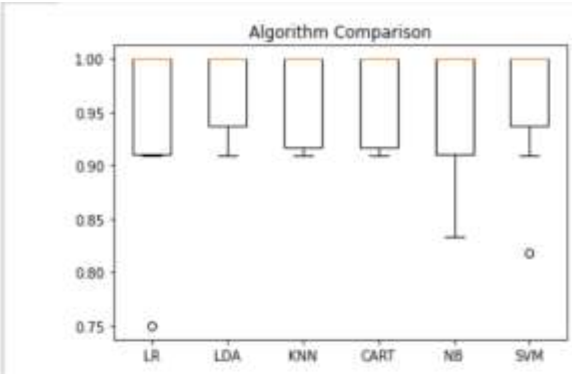
```
   sepal-length  sepal-width  petal-length  petal-width        class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
5           5.4          3.9           1.7          0.4  Iris-setosa
6           4.6          3.4           1.4          0.3  Iris-setosa
7           5.0          3.4           1.5          0.2  Iris-setosa
8           4.4          2.9           1.4          0.2  Iris-setosa
9           4.9          3.1           1.5          0.1  Iris-setosa
```

```
# Split-out validation dataset
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_
size=0.20, random_state=1)

# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ov
r')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
# evaluate each model in turn
results = []
names = []
for name, model in models:
  kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='
accuracy')
  results.append(cv_results)
  names.append(name)
  print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
```

```
LR: 0.948485 (0.077094)
LDA: 0.974242 (0.039394)
KNN: 0.965152 (0.042740)
CART: 0.965909 (0.041804)
NB: 0.949242 (0.067083)
SVM: 0.964394 (0.059056)
```

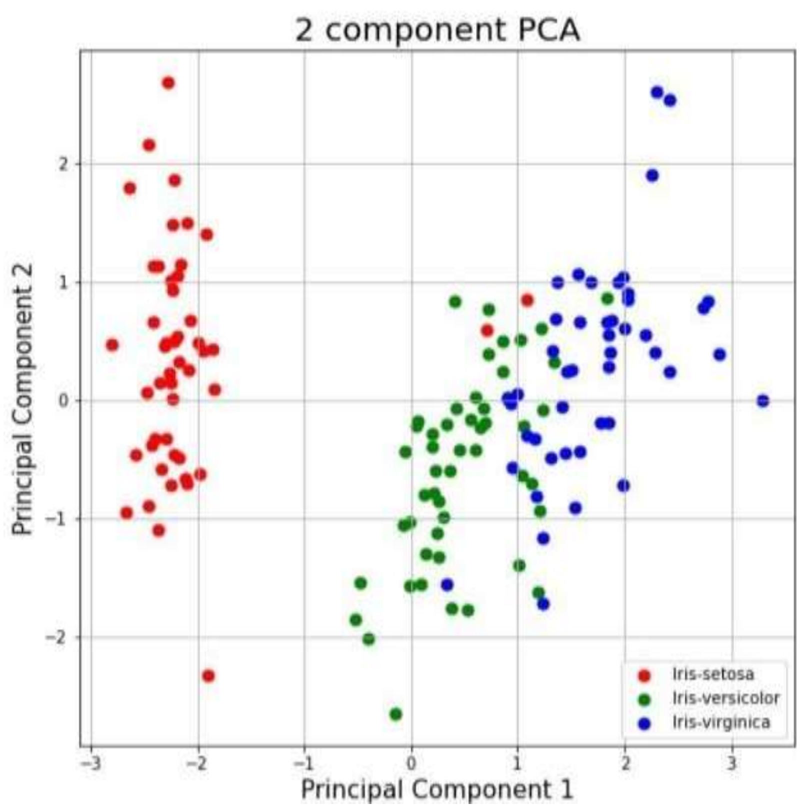
```
# Compare Algorithms
```

```
pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()
```




```
#PCA is effected by scale so you need to scale the features in your data be
fore applying PCA. Use StandardScaler to help you st
#the dataset's features onto unit scale (mean = 0 and variance = 1) which i
s a requirement for the optimal performance of many machine learning algori
thms.
from sklearn.preprocessing import StandardScaler
features = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width']
# Separating out the features
x = dataset.loc[:, features].values
# Separating out the target
y = dataset.loc[:,['class']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = DataFrame(data = principalComponents
              , columns = ['principal component 1', 'principal component 2']
)

#Concatenating DataFrame along axis = 1. finalDf is the final DataFrame bef
ore plotting the data
finalDf = concat([principalDf, dataset[['class']]], axis = 1)

fig = pyplot.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
```

```
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['class'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```




```
pca.explained_variance_ratio_
```

```
array([0.72738591, 0.23030014])
```

CP2. Using Scikit-learn, split the iris dataset into 80% train data and 20% test data. Train or fit the data into the model and using the K Nearest Neighbor Algorithm and create a plot of k values vs accuracy.

```
#Load Libraries
from pandas import read_csv
from pandas.plotting import scatter_matrix
from pandas import set_option
from pandas import DataFrame
from pandas import concat
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score


# Load dataset
#url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.cs
v"
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-
width', 'class']
dataset = read_csv(url, names=names)

# locate rows of duplicate data
# calculate duplicates
dups = dataset.duplicated()
# report if there are any duplicates
print(dups.any())
# list all duplicate rows
print(dataset[dups])
```

```
True
     sepal-length  sepal-width  petal-length  petal-width           class
34            4.9          3.1           1.5          0.1     Iris-setosa
37            4.9          3.1           1.5          0.1     Iris-setosa
142           5.8          2.7           5.1          1.9  Iris-virginica
```

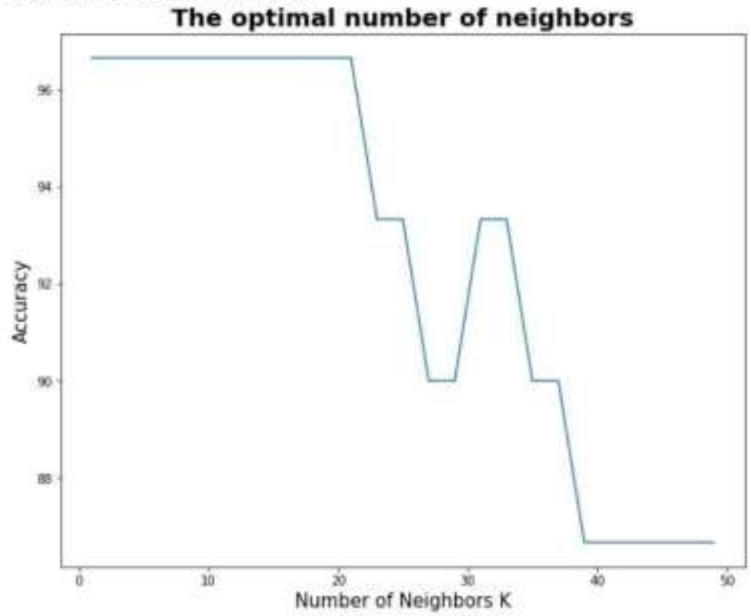
```
# delete rows of duplicate data from the dataset
print(dataset.shape)
# delete duplicate rows
dataset.drop_duplicates(inplace=True)
print(dataset.shape)
```

```
(150, 5)
(147, 5)
```

```
# head, peek your dataset, see first 10 rows
print(dataset.head(10))
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

```
# Split-out validation dataset
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_
size=0.20, random_state=1)

k_list = list(range(1,50,2))
# creating list of accuracy
accuracy = []

for k in k_list:
    classifier = KNeighborsClassifier(n_neighbors=k)

    # Fitting the model
    classifier.fit(X_train, Y_train)

    # Predicting the Test set results
    y_pred = classifier.predict(X_test)

    # Calculating the accuracy
    accuracy.append(accuracy_score(Y_test, y_pred)*100)


# plotting graph of k-values vs accuracy
pyplot.figure()
pyplot.figure(figsize=(10,8))
pyplot.title('The optimal number of neighbors', fontsize=20, fontweight='bo
ld')
pyplot.xlabel('Number of Neighbors K', fontsize=15)
pyplot.ylabel('Accuracy', fontsize=15)
#sns.set_style("whitegrid")
pyplot.plot(k_list, accuracy)

pyplot.show()
```


The optimal number of neighbors
Accuracy
Number of Neighbors K
96
94
92
90
88
0
10
20
30
40
50

CP3. Clean the Oil Spill dataset from the following URL: https://github.com/jbrownlee/Datasets/blob/master/oilspill.csv. Clean the data of duplicate data, single value columns and low variance columns. Once the data is prepared, evaluate it on the classification algorithms in CP1 and present the result through suitable visualizations

```
import pandas as pd
import numpy as np
## for plotting
import matplotlib.pyplot as plt
import seaborn as sns
## for statistical tests
import scipy
import statsmodels.formula.api as smf
import statsmodels.api as sm
## for machine learning
from sklearn import preprocessing, feature_selection, ensemble, decompositi
on
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

from google.colab import drive
drive.mount('/content/drive')

# read the data into a pandas Dataframe

dtf = pd.read_csv('/content/drive/My Drive/Colab Notebooks/titanic_data.csv
')
dtf.head()
```

	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

```
#let's check how many cells are left empty in the table.
dtf.isnull().sum()
```

```
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
```

```
#Dropping the "Cabin" column from the data frame as it won't be of much imp
ortance
dtf=dtf.drop(columns='Cabin', axis=1)

#Replacing the missing values in the "Age" column with the mean value
dtf['Age'].fillna(dtf['Age'].mean(), inplace=True)

#Finding the mode value of the "Embarked" column as it will have occurred t
he maximum number of times
#Replacing the missing values in the "Embarked" column with mode value
print(dtf['Embarked'].mode())
dtf['Embarked'].fillna(dtf['Embarked'].mode()[0], inplace=True)

#convert string type values into numerical
dtf.replace({'Sex':{'male':0,'female':1}, 'Embarked':{'S':0,'C':1,'Q':2}},
inplace=True)

#cleaning data of duplicate data
# calculate duplicates
dups = dtf.duplicated()
# report if there are any duplicates
print(dups.any())
# list all duplicate rows
print(dtf[dups])
print(dtf.shape)
# delete duplicate rows
dtf.drop_duplicates(inplace=True)
print(dtf.shape)
```

```
False
Empty DataFrame
Columns: [PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Embarked]
Index: []
(891, 11)
(891, 11)
```

```
#cleaning data of single value column
print(dtf.shape)
```

```
# get number of unique values for each column
counts = dtf.nunique()
# record columns to delete
to_del = [i for i,v in enumerate(counts) if v == 1]
print(to_del)
# drop useless columns
dtf.drop(to_del, axis=1, inplace=True)
print(dtf.shape)
```

```
(891, 11)
[]
(891, 11)
```

```
#to implement ml split data in target and feature variables
# X is the feature variable, containing all the features like Pclass, Age, 
Sex, Embarked, etc. excluding the Survived column
X = dtf.drop(columns = ['PassengerId','Name','Ticket','Survived'],axis=1)

#Y, on the other hand, is the target variable, as that is the result that w
e want to determine,i.e, whether a person is alive.
Y =dtf['Survived']

#cleaning data of low variance
var_thr = VarianceThreshold(threshold = 0.1)
var_thr.fit(X)
concol = [column for column in X.columns
          if column not in X.columns[var_thr.get_support()]]

for features in concol:
    print(features)
X.drop(concol,axis=1)
```

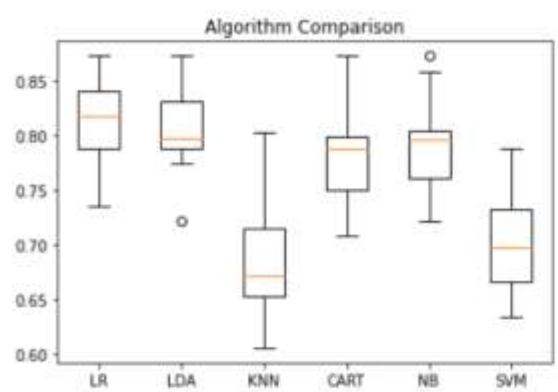
	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	0	22.000000	1	0	7.2500	0
1	1	1	38.000000	1	0	71.2833	1
2	3	1	26.000000	0	0	7.9250	0
3	1	1	35.000000	1	0	53.1000	0
4	3	0	35.000000	0	0	8.0500	0
...	...	...	...	...	...	...	...
886	2	0	27.000000	0	0	13.0000	0
887	1	1	19.000000	0	0	30.0000	0
888	3	1	29.699118	1	2	23.4500	0
889	1	0	26.000000	0	0	30.0000	1
890	3	0	32.000000	0	0	7.7500	2

891 rows × 7 columns

```
#split the data into four variables, namely, X_train, Y_train, X_test, Y_te
st
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2, ran
dom_state=2)

# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ov
r')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
# evaluate each model in turn
results = []
names = []
for name, model in models:
  kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
  cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='
accuracy')
  results.append(cv_results)
  names.append(name)
  print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
```

```
LR: 0.811913 (0.038648)
LDA: 0.804910 (0.040122)
KNN: 0.684038 (0.052450)
CART: 0.781084 (0.046911)
NB: 0.792214 (0.045227)
SVM: 0.703775 (0.050456)
```

```
# Compare Algorithms
plt.boxplot(results, labels=names)
plt.title('Algorithm Comparison')
plt.show()
```



DP1. Load the Boston housing dataset directly via URL and split it into train and test sets, then estimate the mean squared error (MSE) for a linear regression model. Estimate the bias and variance for the linear regression model?

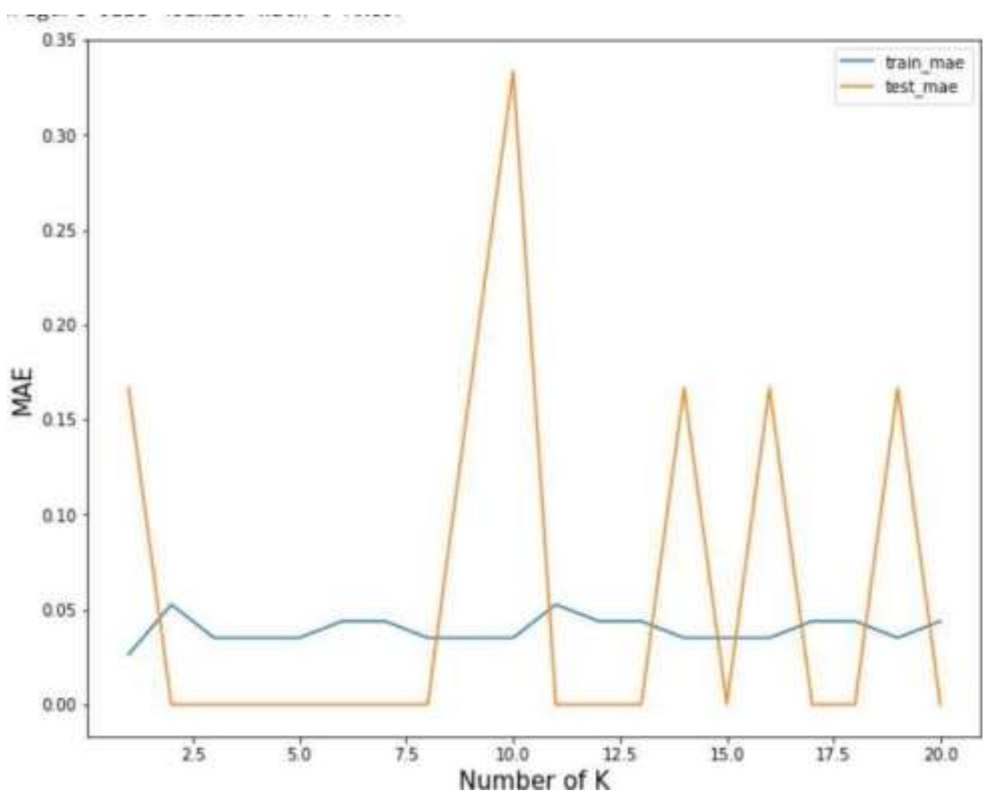
```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from mlxtend.evaluate import bias_variance_decomp

# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.
csv'
dataframe = read_csv(url, header=None)

# separate into inputs and outputs
data = dataframe.values
X, y = data[:, :-1], data[:, -1]

# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, r
andom_state=1)

# define the model
model = LinearRegression()

# estimate bias and variance
mse, bias, var = bias_variance_decomp(model, X_train, y_train, X_test, y_te
st, loss='mse', num_rounds=200, random_seed=1)

# summarize results
print('MSE: %.3f' % mse)
print('Bias: %.3f' % bias)
print('Variance: %.3f' % var)
```

```
MSE: 22.418
Bias: 20.744
Variance: 1.674
```

DP2. Use the Iris Dataset of CP1. The dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica and Iris versicolor).use KFolds cross-validation with 20 folds (K=20) to evaluate the generalization ability of our model. Within each fold we will estimate the training and test error using the training and test sets, respectively. Plot the MAE of the training phase and the MAE of the testing phase. Interpret the results and try to spot the overfitting and underfitting points?

```
#import datasets from sklearn library
from sklearn import datasets
data = datasets.load_iris()

#Import decision tree classification model and cross validation
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics  import  accuracy_score
from sklearn.metrics import mean_absolute_error

#Extract a holdout set at the very begining
X_train_set, X_holdout, y_train_set, y_holdout = train_test_split(data.data
, data.target, stratify = data.target, random_state = 42, test_size = .20)

#Get input and output datasets values in X and Y variables
X = X_train_set
y = y_train_set

#Initialize  k-fold  cross  validation  configurations
kf = KFold(n_splits=20, random_state=42,shuffle=True)


train_mae=[]
test_mae=[]
model = LogisticRegression(solver='liblinear', multi_class='ovr')
for train_index, test_index in kf.split(X):

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model.fit(X_train, y_train)
    X_train_pred = model.predict(X_train)
    train_mae.append(mean_absolute_error(y_train, X_train_pred))

    X_test_pred = model.predict(X_test)
    test_mae.append(mean_absolute_error(y_test, X_test_pred))


from matplotlib import pyplot
k = list(range(1,21))
```

```
pyplot.figure()
pyplot.figure(figsize=(10,8))

pyplot.xlabel('Number of K', fontsize=15)
pyplot.ylabel('MAE', fontsize=15)
pyplot.plot(k, train_mae,label="train_mae")
pyplot.plot(k, test_mae,label="test_mae")
pyplot.legend()
pyplot.show()
```

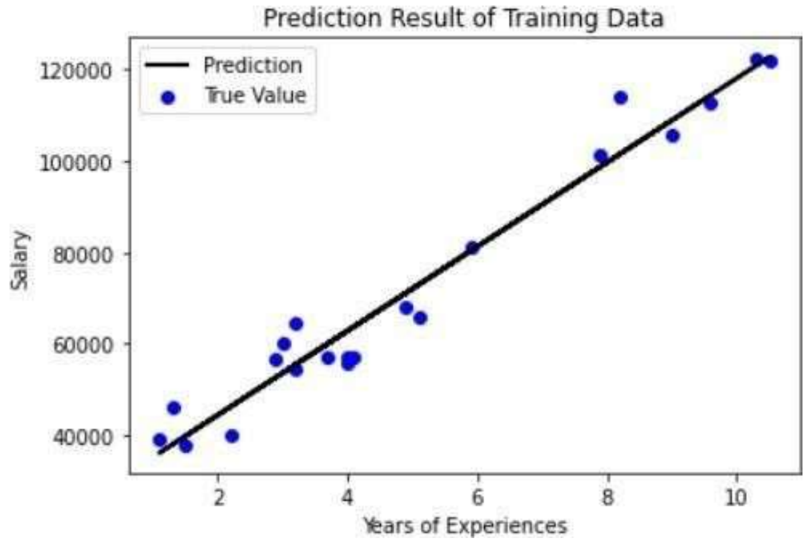


EP1. Using linear regression predict the relationship between the experience of an individual and his salary. Predict the variance and bias for the same?

```
import mlxtend.evaluate
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from mlxtend.evaluate import bias_variance_decomp

from google.colab import drive
drive.mount('/content/drive')
dtf = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Salary_Data.csv'
)
dtf.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

```
data = dtf.values
X, y = data[:, :-1], data[:, -1]
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
ndom_state=1)

model = LinearRegression()
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)

plt.figure()
plt.scatter(X_train, y_train, color='blue', label="True Value")
plt.plot(X_train, y_train_pred, color='black', linewidth=2, label="Predicti
on")
plt.xlabel("Years of Experiences")
plt.ylabel("Salary")
plt.title('Prediction Result of Training Data')
plt.legend()
plt.show()
```

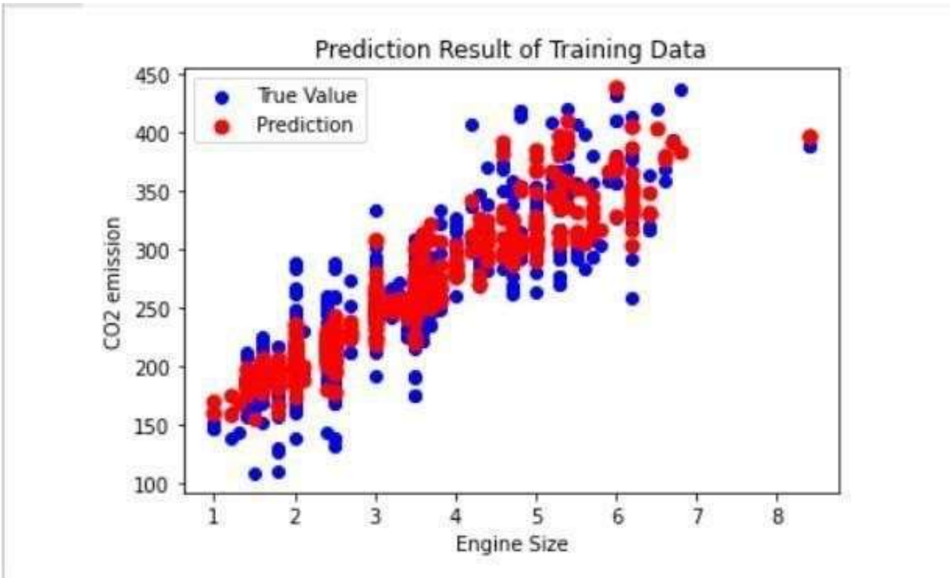



```
# estimate bias and variance
_, bias, var = bias_variance_decomp(model, X_train, y_train, X_test, y_test
, loss='mse', num_rounds=200, random_seed=1)
# summarize results
print('Bias: %.3f' % bias)
print('Variance: %.3f' % var)
```

```
Bias: 40477216.614
Variance: 2623028.095
```

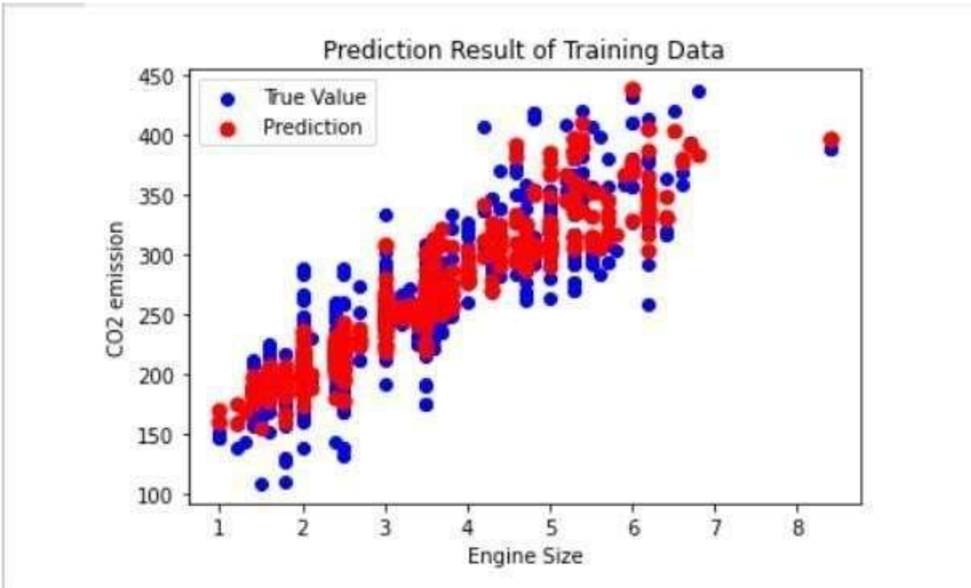
EP2. Predict the CO2 emission of a car based on the size of the engine, but use multiple regression so we can throw in more variables, like the weight of the car?

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

from google.colab import drive
drive.mount('/content/drive')
dtf = pd.read_csv('/content/drive/My Drive/Colab Notebooks/FuelConsumptionC
o2.csv')
data = dtf.values
X = np.asanyarray(dtf[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUE
LCONSUMPTION_HWY','FUELCONSUMPTION_COMB']])
y = np.asanyarray(dtf[['CO2EMISSIONS']])
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
ndom_state=1)

model = LinearRegression()
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)

plt.figure()
plt.scatter(X_train[:,:1], y_train, color='blue', label="True Value")
plt.scatter(X_train[:,:1], y_train_pred, color='red', linewidth=2, label="P
rediction")
plt.xlabel("Engine Size")
plt.ylabel("CO2 emission")
plt.title('Prediction Result of Training Data')
plt.legend()
plt.show()
```

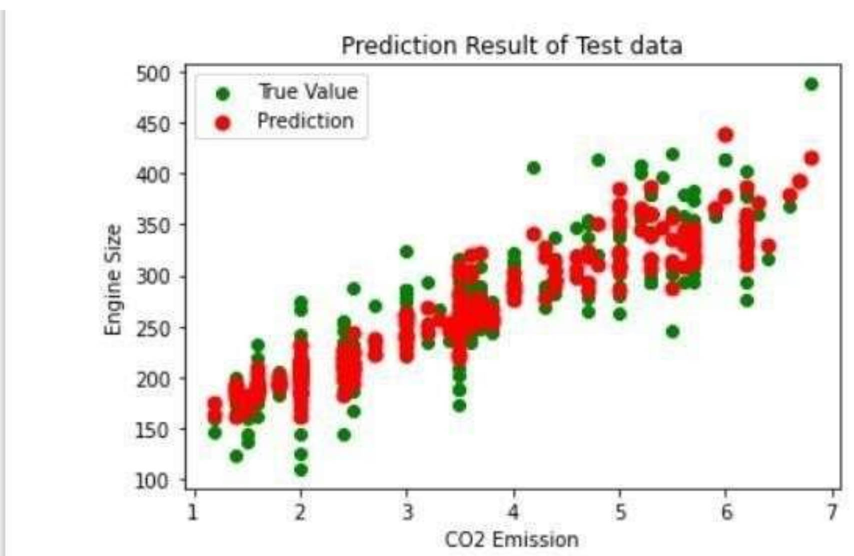

Prediction Result of Training Data
True Value
Prediction
CO2 emission
Engine Size
450
400
350
300
250
200
150
100
1
2
3
4
5
6
7
8

EP3. Plot the CO2 emission values wrt engine size using multiple linear regression?

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

from google.colab import drive
drive.mount('/content/drive')
dtf = pd.read_csv('/content/drive/My Drive/Colab Notebooks/FuelConsumptionC
o2.csv')
data = dtf.values
X = np.asanyarray(dtf[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUE
LCONSUMPTION_HWY','FUELCONSUMPTION_COMB']])
y = np.asanyarray(dtf[['CO2EMISSIONS']])
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
ndom_state=1)

model = LinearRegression()
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)

plt.figure()
plt.scatter(X_train[:,:1], y_train, color='blue', label="True Value")
plt.scatter(X_train[:,:1], y_train_pred, color='red', linewidth=2, label="P
rediction")
plt.xlabel("Engine Size")
plt.ylabel("CO2 emission")
plt.title('Prediction Result of Training Data')
plt.legend()
plt.show()
```




```
y_test_pred = model.predict(X_test)

plt.figure()
plt.scatter(X_test[:,:1], y_test, color='green', label='True Value')
plt.scatter(X_test[:,:1], y_test_pred, color='red', linewidth=2, label='Pre
diction')
plt.xlabel("CO2 Emission")
plt.ylabel("Engine Size")
plt.title('Prediction Result of Test data')
plt.legend()
plt.show()
```



EP4. Apply Linear Regression and build a model that studies the relationship between the head size and the brain weight of an individual? Evaluate by using least square regression method where RMSE (root mean squared error) and R-squared/R2 will be the model evaluation parameters.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from google.colab import drive
drive.mount('/content/drive')
dtf = pd.read_csv('/content/drive/My Drive/Colab Notebooks/headbrain.csv')
dtf.head()
```

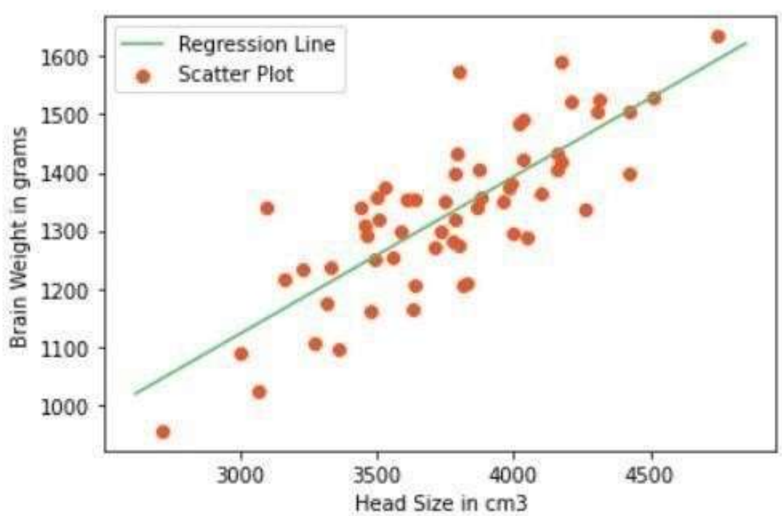
	Gender	Age Range	Head Size	Brain Weight (grams)
0	1	1	4512	1530
1	1	1	3738	1297
2	1	1	4261	1335
3	1	1	3777	1282
4	1	1	4177	1590

```
X = dtf['Head Size'].values
Y = dtf['Brain Weight (grams)'].values

# Mean X and Y
mean_x = np.mean(X)
mean_y = np.mean(Y)

# Total number of values
n = len(X)

# Using the formula to calculate 'm' and 'c'
numer = 0
denom = 0
for i in range(n):
  numer += (X[i] - mean_x) * (Y[i] - mean_y)
  denom += (X[i] - mean_x) ** 2
m = numer / denom
c = mean_y - (m * mean_x)
```

```
# Printing coefficients
print("Coefficients")
print(m, c)
```

```
Coefficients
0.2710171952076625 308.69902672534613
```

```
# Plotting Values and Regression Line

max_x = np.max(X) + 100
min_x = np.min(X) - 100

# Calculating line values x and y
x = np.linspace(min_x, max_x, 1000)
y = c + m * x

# Ploting Line
plt.plot(x, y, color='#58b970', label='Regression Line')
# Ploting Scatter Points
plt.scatter(X, Y, c='#ef5423', label='Scatter Plot')

plt.xlabel('Head Size in cm3')
plt.ylabel('Brain Weight in grams')
plt.legend()
plt.show()
```




```
# Calculating Root Mean Squares Error
rmse = 0
for i in  range(n):
    y_pred = c + m * X[i]
    rmse += (Y[i] - y_pred) ** 2
```

```
rmse = np.sqrt(rmse/n)
print("RMSE: ",rmse)
```

```
RMSE:  82.20224448520042
```

```
# Calculating R2 Score
ss_tot = 0
ss_res = 0
for i in  range(n):
    y_pred = c + m * X[i]
    ss_tot += (Y[i] - mean_y) ** 2
    ss_res += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ss_tot)
print("R2 Score: ",r2)
```

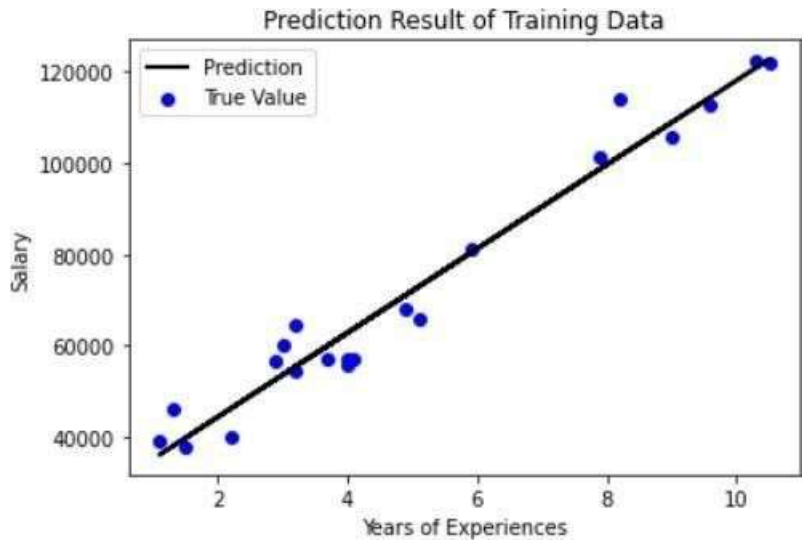
```
R2 Score:  0.6378327820399066
```

EP5. Modify EP1 to calculate MSE, RMSE and R2 as the model evaluation parameters.

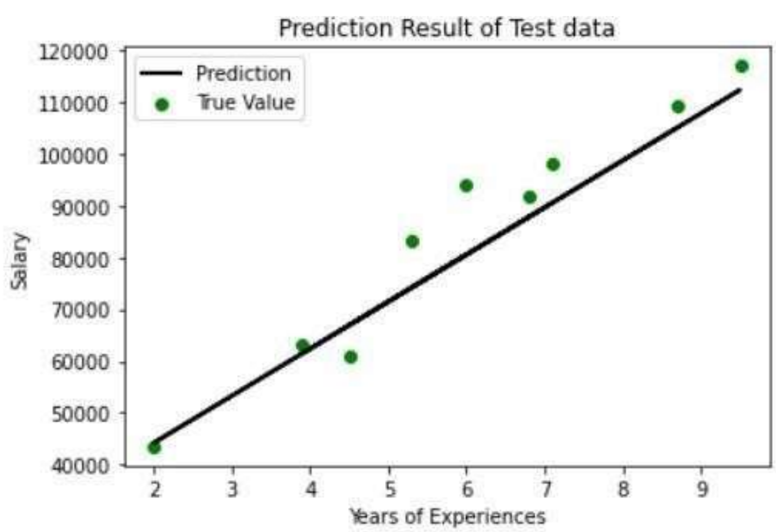
```
import mlxtend.evaluate
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

from google.colab import drive
drive.mount('/content/drive')
dtf = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Salary_Data.csv'
)
dtf.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

```
data = dtf.values
X, y = data[:, :-1], data[:, -1]
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
ndom_state=1)

model = LinearRegression()
model.fit(X_train, y_train)
y_train_pred = model.predict(X_train)

plt.figure()
plt.scatter(X_train, y_train, color='blue', label="True Value")
plt.plot(X_train, y_train_pred, color='black', linewidth=2, label="Predicti
on")
plt.xlabel("Years of Experiences")
plt.ylabel("Salary")
plt.title('Prediction Result of Training Data')
plt.legend()
plt.show()
```




```
y_test_pred = model.predict(X_test)
plt.figure()
plt.scatter(X_test, y_test, color='green', label='True Value')
plt.plot(X_test,y_test_pred,color='black', linewidth=2, label='Prediction')
plt.xlabel("Years of Experiences")
plt.ylabel("Salary")
plt.title('Prediction Result of Test data')
plt.legend()
plt.show()
```




```
mse = mean_squared_error(y_test,y_test_pred)
print("MSE =", mse)
```

```
MSE = 45664016.935905606
```

```
rmse = np.sqrt(mean_squared_error(y_test,y_test_pred))
print("RMSE =", rmse)
```

```
RMSE = 6757.515589024239
```

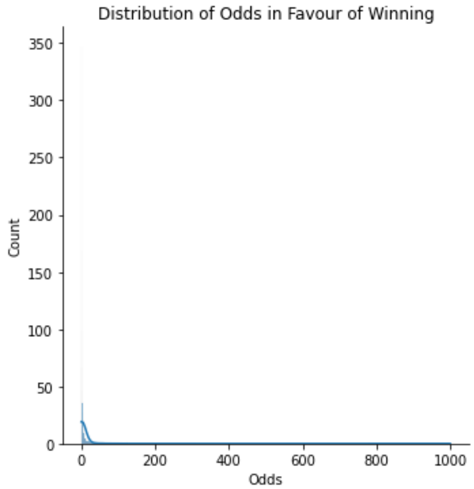
```
r2_Score=r2_score(y_test, y_test_pred)
print("R2 square =", r2_Score)
```

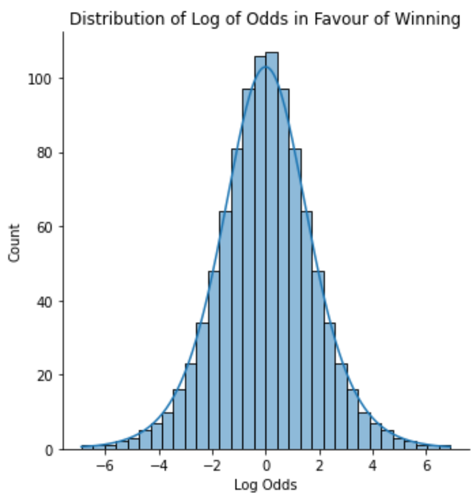
```
R2 square = 0.9123312937146872
```

EP6. Demonstrate odds ratio and log of odds on a dataframe for winning and losing?

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

win=list(range(1,1000,1))
lose=list(range(999,0,-1))

df=pd.DataFrame()
df['Win']=win
df['Lose']=lose

df['Odds_win']=df['Win']/df['Lose']
df['Odds_lose']=df['Lose']/df['Win']

df['Log_Odds_Win']=np.log(df['Odds_win'])
df['Log_Odds_Lose']=np.log(df['Odds_lose'])
sns.displot(df['Odds_win'],kde=True)
plt.title("Distribution of Odds in Favour of Winning")
plt.xlabel("Odds")
plt.ylabel("Count")

sns.displot(df['Log_Odds_Win'],kde=True)
plt.title("Distribution of Log of Odds in Favour of Winning")
plt.xlabel("Log Odds")
plt.ylabel("Count")
```

```
Text(3.799999999999997, 0.5, 'Count')
```





EP7. Generate univariate baby weight data and apply linear regression. Evaluate the model by calculating SSE, SST, and R2.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

from google.colab import drive
drive.mount('/content/drive')
dtf = pd.read_csv('/content/drive/My Drive/Colab Notebooks/Baby Weight.csv'
)
dtf.head()
```

	Birth.Weight	Gestational.Days	Maternal.Age	Maternal.Height	Maternal.Pregnancy.Weight	Maternal.Smoker	Unnamed: 6
0	120	284	27	62	100	False	NaN
1	113	282	33	64	135	False	NaN
2	128	279	28	64	115	True	NaN
3	108	282	23	67	125	True	NaN
4	136	286	25	62	93	False	NaN

```
#dropping extra column
dtf=dtf.drop(columns='Unnamed: 6', axis=1)
dtf.head()
```

	Birth.Weight	Gestational.Days	Maternal.Age	Maternal.Height	Maternal.Pregnancy.Weight	Maternal.Smoker
0	120	284	27	62	100	False
1	113	282	33	64	135	False
2	128	279	28	64	115	True
3	108	282	23	67	125	True
4	136	286	25	62	93	False

```
#convert boolean type values into numerical
dtf.replace({'Maternal.Smoker':{False:0,True:1}},inplace=True)
dtf.head()
```

	Birth.Weight	Gestational.Days	Maternal.Age	Maternal.Height	Maternal.Pregnancy.Weight	Maternal.Smoker
0	120	284	27	62	100	0
1	113	282	33	64	135	0
2	128	279	28	64	115	1
3	108	282	23	67	125	1
4	136	286	25	62	93	0

```
X = dtf.iloc[:, 1:].values
y = dtf.iloc[:, 0].values
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
ndom_state=1)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

model = LinearRegression()
model.fit(X_train, y_train)

y_train_pred = model.predict(X_train)

#calculate sse
sse = np.sum((y_train_pred - y_train)**2)
print(sse)
```

```
197370.78335708228
```

```
#calculate ssr
ssr = np.sum((y_train_pred - y_train.mean())**2)
print(ssr)
```

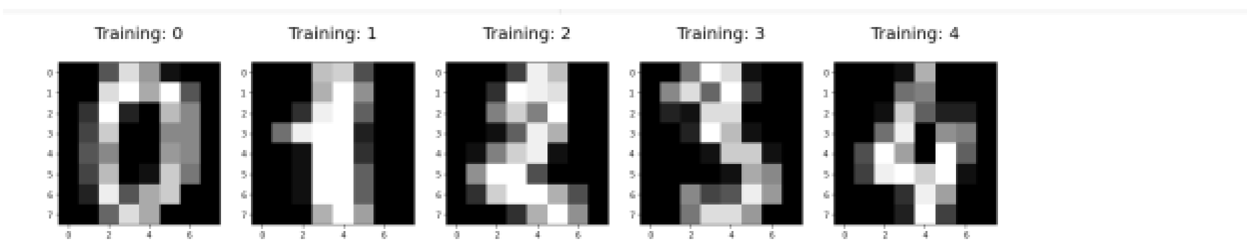
```
74863.97425558524
```

```
#calculate sst
sst = ssr + sse
print(sst)
```

```
272234.7576126675
```

EP8. Apply logistic regression to the load-digits dataset of the sklearn library? Create a confusion matrix for the model and also generate the classification report?

```
from sklearn.datasets import load_digits
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

digits = load_digits()
plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:5], digits.target[
0:5])):
    plt.subplot(1, 5, index + 1)
    plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
    plt.title('Training: %i\n' % label, fontsize = 20)
```




```
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.tar
get, test_size=0.25, random_state=0)
logisticRegr = LogisticRegression()
logisticRegr.fit(x_train, y_train)
predictions = logisticRegr.predict(x_test)
score = logisticRegr.score(x_test, y_test)
print(score)
```

```
0.9511111111111111
```

```
cm = metrics.confusion_matrix(y_test, predictions)
print(cm)
```

```
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 40  0  0  0  0  0  0  2  1]
 [ 0  1 40  3  0  0  0  0  0  0]
 [ 0  0  0 43  0  0  0  0  1  1]
 [ 0  0  0  0 37  0  0  1  0  0]
 [ 0  0  0  0  0 46  0  0  0  2]
 [ 0  1  0  0  0  0 51  0  0  0]
 [ 0  0  0  1  1  0  0 46  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  0  0  1  0  0  1 45]]
```

```
print(f"{metrics.classification_report(y_test, predictions)}\n")
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        37
           1       0.89      0.93      0.91        43
           2       0.98      0.91      0.94        44
           3       0.91      0.96      0.93        45
           4       0.97      0.97      0.97        38
           5       0.98      0.96      0.97        48
           6       1.00      0.98      0.99        52
           7       0.98      0.96      0.97        48
           8       0.91      0.90      0.91        48
           9       0.90      0.96      0.93        47

    accuracy                           0.95       450
   macro avg       0.95      0.95      0.95       450
weighted avg       0.95      0.95      0.95       450
```

EP9. Apply logistic regression on userdata.csv dataset to predict the users who may be potential customers to purchase a SUV car? Also generate the confusion matrix to evaluate your model?

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

from google.colab import drive
drive.mount('/content/drive')
dtf = pd.read_csv('/content/drive/My Drive/Colab Notebooks/User_Data.csv')
dtf.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
data = dtf.values
X, y = data[:, [2,3]], data[:, 4]
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
ndom_state=1)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

classifier = LogisticRegression()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
print('Accuracy score of test data : ',metrics.accuracy_score(y_test,y_pred
))
```

```
Accuracy score of test data :  0.825
```

```
cm = metrics.confusion_matrix(y_test, y_pred)
print('Confision matrix: ')
print(cm)
```

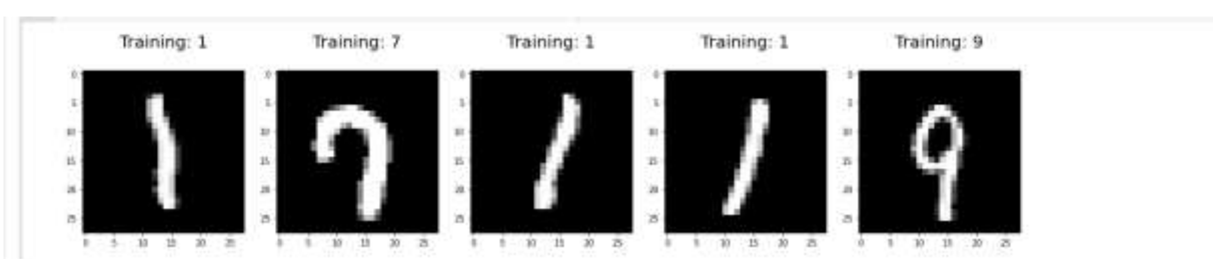
```
Confision matrix:
[[65  7]
 [14 34]]
```

E10. Apply logistic regression on handwritten digits dataset to classify the digits. Evaluate your model too?

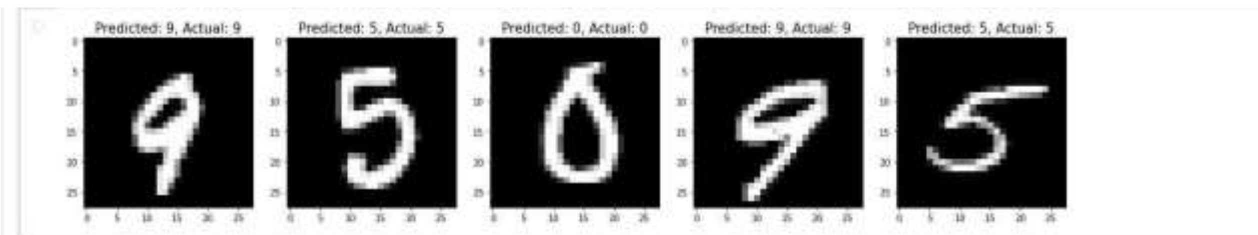
```
import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import check_random_state
import matplotlib.pyplot as plt

X, y = fetch_openml('mnist_784', version=1, return_X_y=True, as_frame=False
)
# Print to show there are 1797 images (8 by 8 images for a dimensionality o
f 64)
print("Image Data Shape" , X.shape)
# Print to show there are 1797 labels (integers from 0-9)
print("Label Data Shape", y.shape)
```

```
Image Data Shape (70000, 784)
Label Data Shape (70000,)
```

```
train_img, test_img, train_lbl, test_lbl = train_test_split(X, y, test_size
=1/7.0, random_state=0)

plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(train_img[0:5], train_lbl[0:5]))
:
    plt.subplot(1, 5, index + 1)
    plt.imshow(np.reshape(image, (28,28)), cmap=plt.cm.gray)
    plt.title('Training: %s\n' % label, fontsize = 20)
```



```
logisticRegr = LogisticRegression(solver = 'lbfgs')
logisticRegr.fit(train_img, train_lbl)
# Make predictions on entire test data
predictions = logisticRegr.predict(test_img)

plt.figure(figsize=(20,4))
for i in range(0,5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(np.reshape(test_img[i], (28,28)), cmap=plt.cm.gray)
    plt.title('Predicted: {}, Actual: {}'.format(predictions[i], test_lbl[i
]), fontsize = 15)
```




```
score = logisticRegr.score(test_img, test_lbl)
print(score)
```

```
0.9246
```

FP1. Understand dimensionality reduction technique?

```
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig
# define a 3*2 matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

```
# calculate the mean of each column
M = mean(A.T, axis=1)
print(M)
```

```
[3. 4.]
```

```
# center columns by subtracting column means
C = A - M
print(C)
```

```
[[-2. -2.]
 [ 0.  0.]
 [ 2.  2.]]
```

```
# calculate covariance matrix of centered matrix
V = cov(C.T)
print(V)
```

```
[[4. 4.]
 [4. 4.]]
```

```
# eigendecomposition of covariance matrix
values, vectors = eig(V)
print(vectors)
print(values)
# project data
P = vectors.T.dot(C.T)
print(P.T)
```

```
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
[8. 0.]
[[-2.82842712  0.        ]
 [ 0.          0.        ]
 [ 2.82842712  0.        ]]
```

```
# Principal Component Analysis
from numpy import array
from sklearn.decomposition import PCA
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# create the PCA instance
pca = PCA(2)
# fit on data
pca.fit(A)
# access values and vectors
print(pca.components_)
print(pca.explained_variance_)
# transform data
B = pca.transform(A)
print(B)
```

```
[[1 2]
 [3 4]
 [5 6]]
[[ 0.70710678  0.70710678]
 [-0.70710678  0.70710678]]
[8. 0.]
[[-2.82842712e+00 -2.22044605e-16]
 [ 0.00000000e+00  0.00000000e+00]
 [ 2.82842712e+00  2.22044605e-16]]
```

FP2. Implement dimensionality reduction on wines.csv using PCA?

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')
dataset = pd.read_csv('/content/drive/My Drive/Colab Notebooks/wine.csv')
dataset.head()
```

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

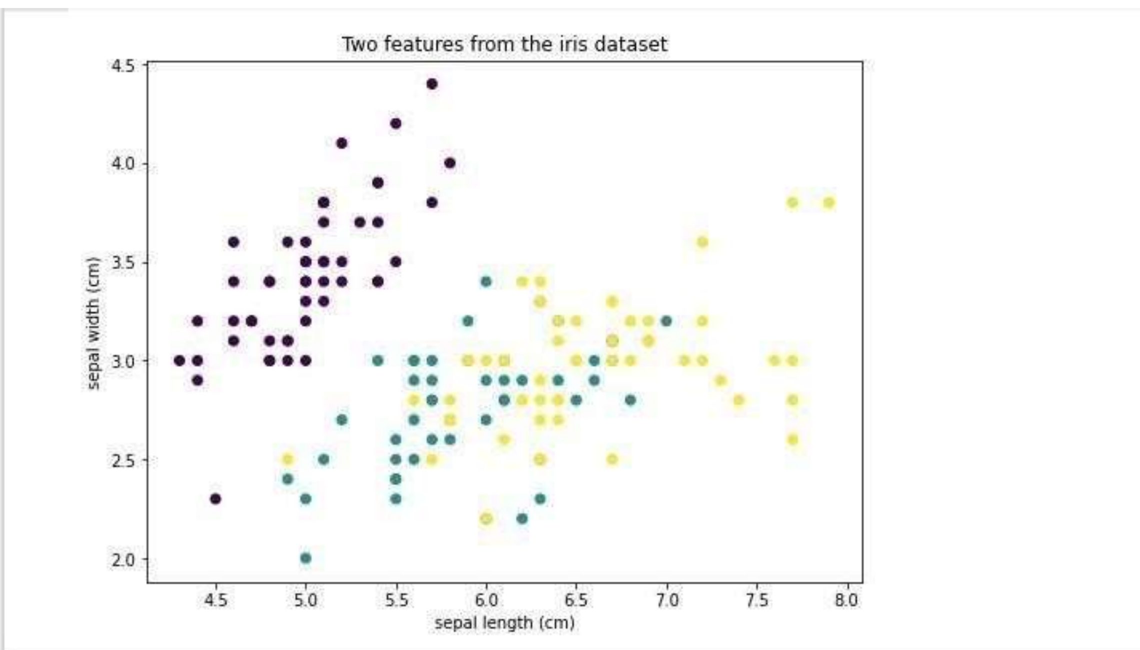
```
X = dataset.iloc[:, 1:].values
y = dataset.iloc[:, 0].values

# splitting the data into the training and test set.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 0)

# Feature scaling
stndS = StandardScaler()
X_train = stndS.fit_transform(X_train)
X_test = stndS.transform(X_test)

# create a PCA object
pca = PCA(n_components = 2)# extracted features we want to end up within ou
r new dataset(2).
# Apply the above object to our training dataset using the fit method.
X_train = pca.fit_transform(X_train)
# Apply the PCA object to the test set only to transform this set
X_test = pca.transform(X_test)

# create object of the above classifier
clfy = LogisticRegression()
clfy.fit(X_train, y_train)
```

```
y_pred = clfy.predict(X_test)

# creating a confussion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# model score
print('Accuracy score of test data : ',accuracy_score(y_test, y_pred))
```

```
[[14  0  0]
 [ 1 15  0]
 [ 0  0  6]]
Accuracy score of test data :  0.9722222222222222
```

FP3. Create a basic visualization of Iris dataset in question CP1 using PCA?

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.metrics import f1_score
from sklearn.svm import SVC
import matplotlib.pyplot as plt

# Load iris dataset
irisdata = load_iris()
X, y = irisdata['data'], irisdata['target']
plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X[:,1], c=y)
plt.xlabel(irisdata["feature_names"][0])
plt.ylabel(irisdata["feature_names"][1])
plt.title("Two features from the iris dataset")
plt.show()
```

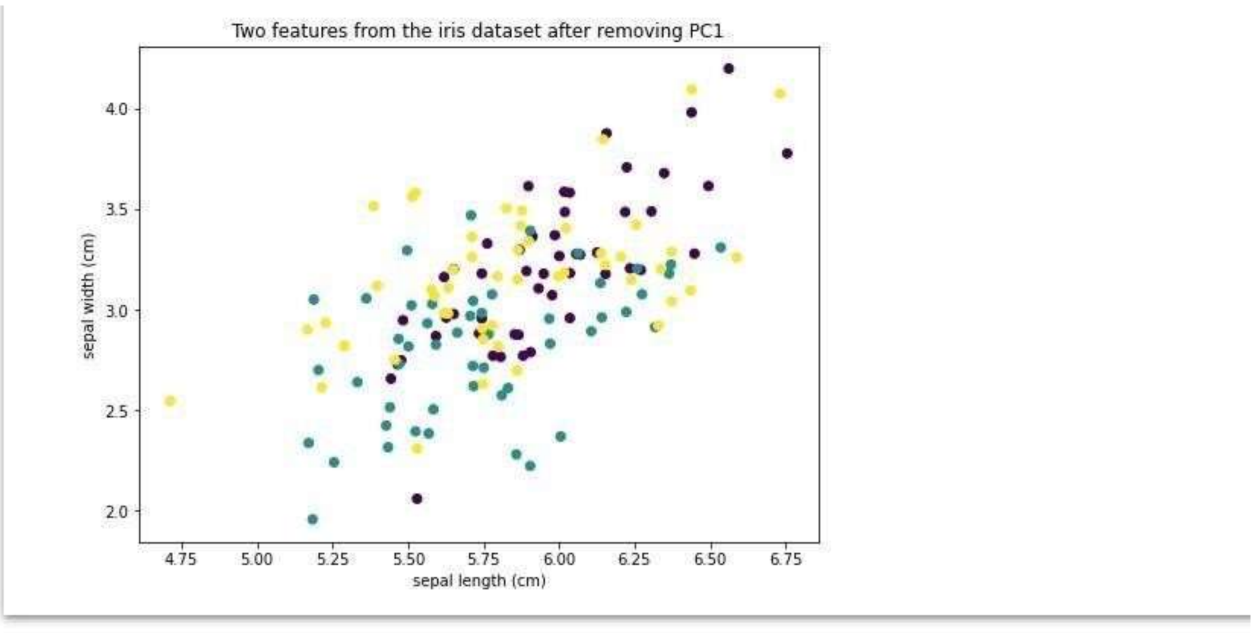



```
# Show the principal components
pca = PCA().fit(X)
print("Principal components:")
print(pca.components_)
```

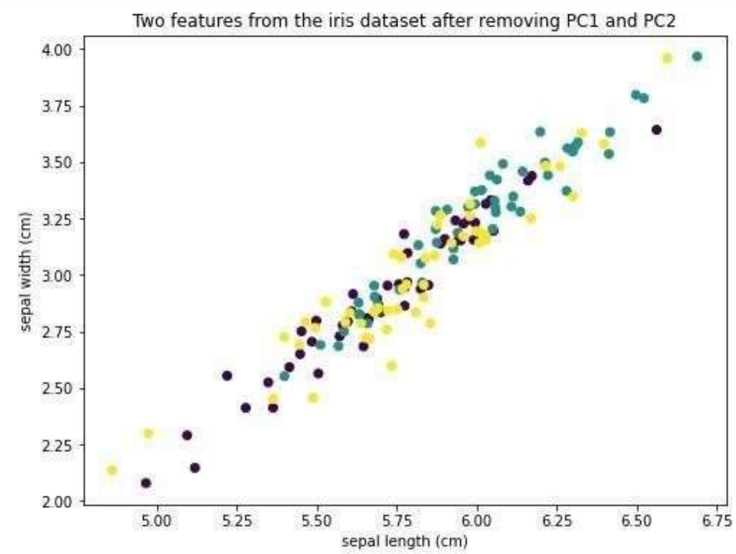
```
Principal components:
[[ 0.36138659 -0.08452251  0.85667061  0.3582892 ]
 [ 0.65658877  0.73016143 -0.17337266 -0.07548102]
 [-0.58202985  0.59791083  0.07623608  0.54583143]
 [-0.31548719  0.3197231   0.47983899 -0.75365743]]
```

```
# Remove PC1
```

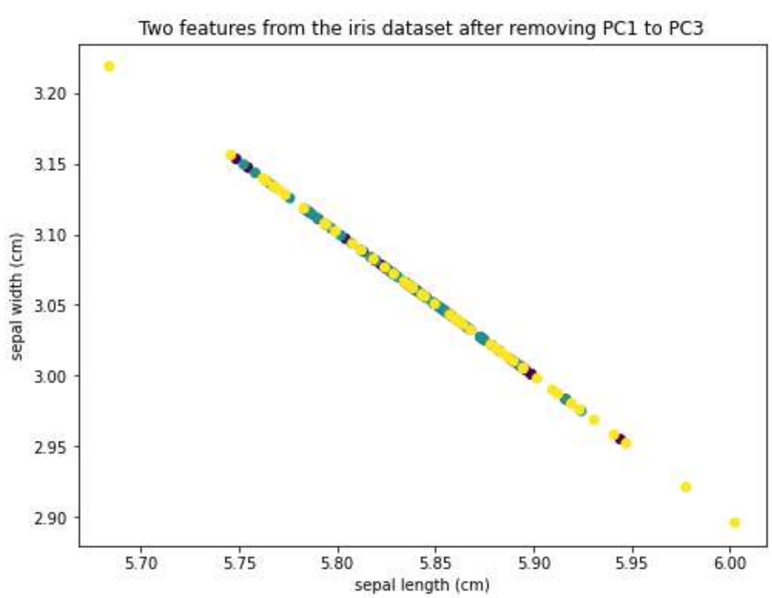
```
Xmean = X - X.mean(axis=0)
value = Xmean @ pca.components_[0]
pc1 = value.reshape(-1,1) @ pca.components_[0].reshape(1,-1)
Xremove = X - pc1
plt.figure(figsize=(8,6))
plt.scatter(Xremove[:,0], Xremove[:,1], c=y)
plt.xlabel(irisdata["feature_names"][0])
plt.ylabel(irisdata["feature_names"][1])
plt.title("Two features from the iris dataset after removing PC1")
plt.show()
```




```
# Remove PC2
Xmean = X - X.mean(axis=0)
value = Xmean @ pca.components_[1]
pc2 = value.reshape(-1,1) @ pca.components_[1].reshape(1,-1)
Xremove = Xremove - pc2
plt.figure(figsize=(8,6))
plt.scatter(Xremove[:,0], Xremove[:,1], c=y)
plt.xlabel(irisdata["feature_names"][0])
plt.ylabel(irisdata["feature_names"][1])
plt.title("Two features from the iris dataset after removing PC1 and PC2")
plt.show()
```




```
# Remove PC3
Xmean = X - X.mean(axis=0)
value = Xmean @ pca.components_[2]
pc3 = value.reshape(-1,1) @ pca.components_[2].reshape(1,-1)
Xremove = Xremove - pc3
plt.figure(figsize=(8,6))
plt.scatter(Xremove[:,0], Xremove[:,1], c=y)
plt.xlabel(irisdata["feature_names"][0])
plt.ylabel(irisdata["feature_names"][1])
plt.title("Two features from the iris dataset after removing PC1 to PC3")
plt.show()
```




```
# Print the explained variance ratio
print("Explainedd variance ratios:")
```

```
print(pca.explained_variance_ratio_)
```

```
Explainedd variance ratios:
[0.92461872 0.05306648 0.01710261 0.00521218]
```

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

# Run classifer on all features
clf = SVC(kernel="linear", gamma='auto').fit(X_train, y_train)
print("Using all features, accuracy: ", clf.score(X_test, y_test))
print("Using all features, F1: ", f1_score(y_test, clf.predict(X_test), ave
rage="macro"))
```

```
Using all features, accuracy:  0.98
Using all features, F1:  0.980952380952381
```

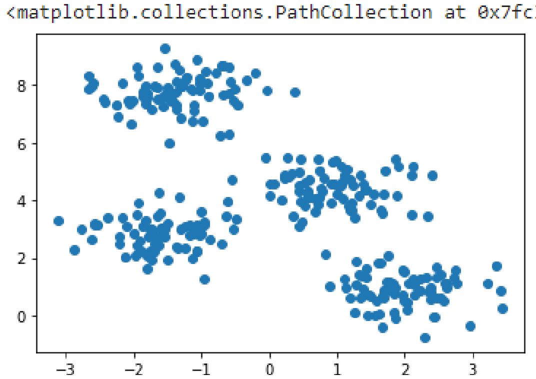
```
# Run classifier on PC1
mean = X_train.mean(axis=0)
X_train2 = X_train - mean
X_train2 = (X_train2 @ pca.components_[0]).reshape(-1,1)
clf = SVC(kernel="linear", gamma='auto').fit(X_train2, y_train)
X_test2 = X_test - mean
X_test2 = (X_test2 @ pca.components_[0]).reshape(-1,1)
print("Using PC1, accuracy: ", clf.score(X_test2, y_test))
print("Using PC1, F1: ", f1_score(y_test, clf.predict(X_test2), average="ma
cro"))
```

```
Using PC1, accuracy:  0.9
Using PC1, F1:  0.9044499044499045
```

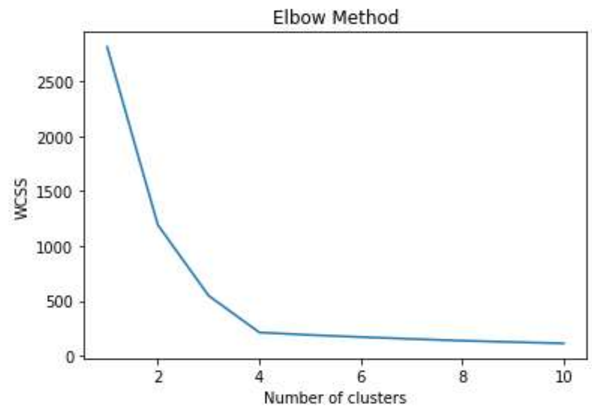
GP1. Create a random dataset using the make_blobs() function from sklearn and apply K-means on the same after deciding the number of clusters using the elbow method?

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

X,y=make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

plt.scatter(X[:,0], X[:,1])
```

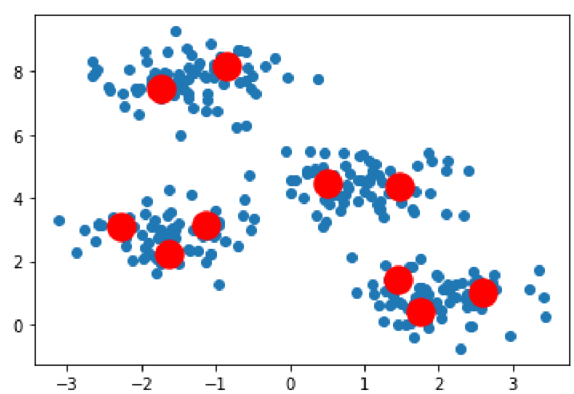
```
<matplotlib.collections.PathCollection at 0x7fc30f71bcd0>
```



```
wcss=[]

for i in range (1,11):
  kmeans=KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
  kmeans.fit(X)
  wcss.append(kmeans.inertia_)


plt.plot(range(1,11),wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

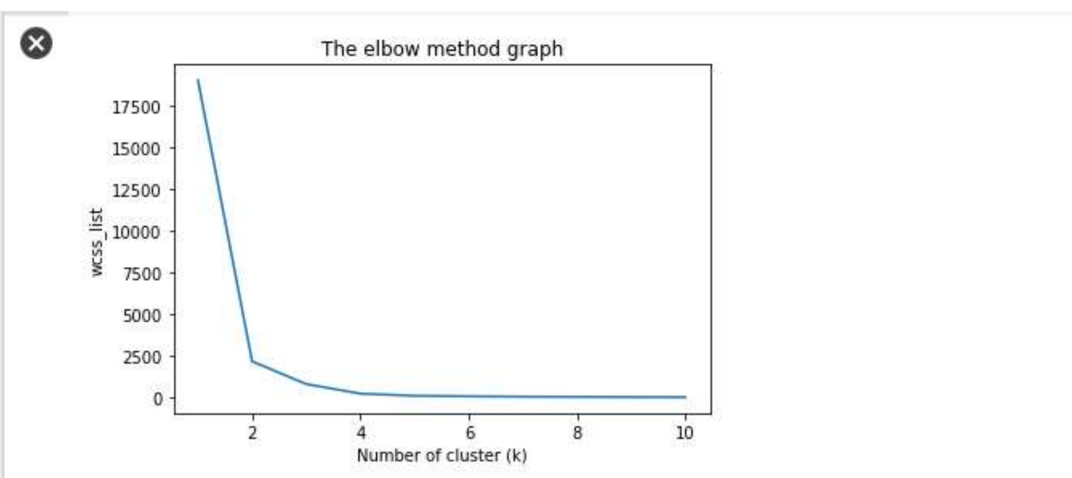



```
pred_y=kmeans.fit_predict(X)

plt.scatter(X[:,0], X[:,1])
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:, 1], s=
300,c='red')
plt.show()
```



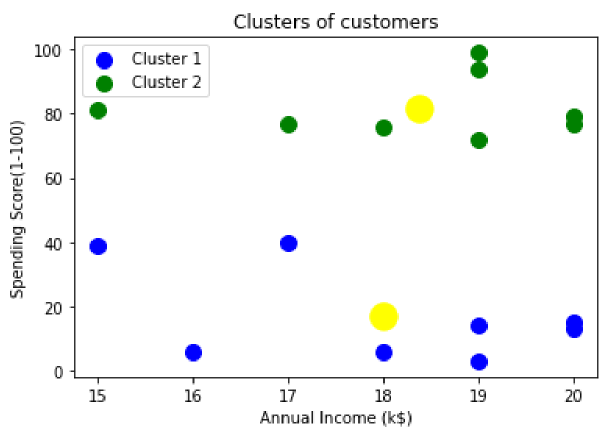
GP2. Create a mall_customer_dataset.csv dataset and apply the K-means on the same after deciding the number of clusters using the elbow method to uncover the patterns?

```
#import libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
dataset=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Mall_Customers
_data.csv')

#extract the independent variables
x=dataset.iloc[:,[3,4]].values

#finding optimal number of clusters using elbow method
from sklearn.cluster import KMeans
wcss_list=[]#initializing the list for the values of WCSS
for i in range(1,11):
  kmeans=KMeans(n_clusters=i, init='k-means++', random_state=42)
  kmeans.fit(x)
  wcss_list.append(kmeans.inertia_)
mtp.plot(range(1,11), wcss_list)
mtp.title('The elbow method graph')
mtp.xlabel('Number of cluster (k)')
mtp.ylabel('wcss_list')
mtp.show()
```




```
#training the K-mean model on a dataset
kmeans=KMeans(n_clusters=2,init='k-means++',random_state=42)
y_predict=kmeans.fit_predict(x)

#visualize the clusters
```

```
mtp.scatter(x[y_predict == 0,0],x[y_predict ==0,1], s=100, c='blue',label='
Cluster 1')
mtp.scatter(x[y_predict == 1,0],x[y_predict ==1,1], s=100, c='green',label=
'Cluster 2')
mtp.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=300
,c='yellow')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score(1-100)')
mtp.legend()
mtp.show()
```



HP1. Use the Pima Indian diabetes database to perform ensemble predictions using the following bagging classifiers: Bagged Decision Trees, Random Forest Classifier and Extra trees?

```
import pandas
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-
indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'cl
ass']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7

kfold = model_selection.KFold(n_splits=10, random_state=seed , shuffle=True
)

# Bagged Decision Trees for Classification
cart = DecisionTreeClassifier()
num_trees = 100

model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, rand
om_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.7578263841421736
```

```
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
max_features=3
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_fea
tures)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.7694805194805195
```

```
# Extra Trees Classification
from sklearn.ensemble import ExtraTreesClassifier
num_trees = 100
max_features = 7
```

```
model = ExtraTreesClassifier(n_estimators=num_trees, max_features=max_featu
res)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

```
0.7629528366370473
```

HP2. Use the same Pima Indian diabetes database of HP1 to perform ensemble predictions using the following boosting classifiers: AdaBoost, Stochastic Gradient Boosting?

```
import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier

url="https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names=['preg','plas','pres','skin','test','mass','pedi','age','class']
dataframe=pandas.read_csv(url,names=names)
array=dataframe.values
X=array[:,0:8]
Y=array[:,8]
seed=7
num_trees=100
kfold=model_selection.KFold(n_splits=12)

#AdaBoost boosting classifier
model=AdaBoostClassifier(n_estimators=num_trees,random_state=seed)
results=model_selection.cross_val_score(model,X,Y,cv=kfold)
print(results)
print(results.mean())
```

```
[0.703125 0.765625 0.75     0.71875  0.703125 0.78125  0.765625 0.734375
 0.84375  0.828125 0.734375 0.78125 ]
0.7591145833333334
```

```
# Stochastic Boosting Classifier boosting classifier
from sklearn.ensemble import GradientBoostingClassifier
model=GradientBoostingClassifier(n_estimators=num_trees,random_state=seed)
results=model_selection.cross_val_score(model,X,Y,cv=kfold)
print(results)
print(results.mean())
```

```
[0.703125 0.8125   0.859375 0.703125 0.625    0.78125  0.765625 0.8125
 0.859375 0.875    0.71875  0.78125 ]
0.7747395833333334
```

IP1. Implement a simple neuron using the sigmoid activation function and feed forward algorithm?

```
import numpy as np

def sigmoid(x):
  return 1/(1+np.exp(-x))

class Neuron:
  def_init_(self,weights,bias):
    self.weights=weights
    self.bias=bias
  def feedforward(self,inputs):
    total=np.dot(self.weights,inputs)+self.bias
    return sigmoid(total)

weights=np.array([0,1])
bias=4

n=Neuron(weights,bias)
x=np.array([2,3])
print(n.feedforward(x))
```

```
0.9990889488055994
```

IP2. Implement a simple neural network with:
- **2 inputs**
- **A hidden layer with 2 neurons (h1, h2)**
- **An output layer with 1 neuron (o1)**

```
import numpy as np

def sigmoid(x):
  # Sigmoid activation function: f(x) = 1 / (1 + e^(-x))
  return 1 / (1 + np.exp(-x))

def deriv_sigmoid(x):
  # Derivative of sigmoid: f'(x) = f(x) * (1 - f(x))
  fx = sigmoid(x)
  return fx * (1 - fx)

def mse_loss(y_true, y_pred):
  # y_true and y_pred are numpy arrays of the same length.
  return ((y_true - y_pred) ** 2).mean()

class OurNeuralNetwork:

  def__init__(self):
    # Weights
    self.w1  =  np.random.normal()
    self.w2  =  np.random.normal()
    self.w3  =  np.random.normal()
    self.w4  =  np.random.normal()
    self.w5  =  np.random.normal()
    self.w6 = np.random.normal()

    # Biases
    self.b1  =  np.random.normal()
    self.b2  =  np.random.normal()
    self.b3 = np.random.normal()

  def feedforward(self, x):
    # x is a numpy array with 2 elements.
    h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
    h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
    o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
    return o1

  def train(self, data, all_y_trues):

    learn_rate = 0.1
    epochs = 1000 # number of times to loop through the entire dataset
```

```
for epoch in range(epochs):
  for x, y_true in zip(data, all_y_trues):
    # --- Do a feedforward (we'll need these values later)
    sum_h1 = self.w1 * x[0] + self.w2 * x[1] + self.b1
    h1 = sigmoid(sum_h1)

    sum_h2 = self.w3 * x[0] + self.w4 * x[1] + self.b2
    h2 = sigmoid(sum_h2)

    sum_o1 = self.w5 * h1 + self.w6 * h2 + self.b3
    o1 = sigmoid(sum_o1)
    y_pred = o1

    # --- Calculate partial derivatives.
    # --- Naming: d_L_d_w1 represents "partial L / partial w1"
    d_L_d_ypred = -2 * (y_true - y_pred)

    # Neuron o1
    d_ypred_d_w5 = h1 * deriv_sigmoid(sum_o1)
    d_ypred_d_w6 = h2 * deriv_sigmoid(sum_o1)
    d_ypred_d_b3 = deriv_sigmoid(sum_o1)

    d_ypred_d_h1 = self.w5 * deriv_sigmoid(sum_o1)
    d_ypred_d_h2 = self.w6 * deriv_sigmoid(sum_o1)

    # Neuron h1
    d_h1_d_w1 = x[0] * deriv_sigmoid(sum_h1)
    d_h1_d_w2 = x[1] * deriv_sigmoid(sum_h1)
    d_h1_d_b1 = deriv_sigmoid(sum_h1)

    # Neuron h2
    d_h2_d_w3 = x[0] * deriv_sigmoid(sum_h2)
    d_h2_d_w4 = x[1] * deriv_sigmoid(sum_h2)
    d_h2_d_b2 = deriv_sigmoid(sum_h2)

    # --- Update weights and biases
    # Neuron h1
    self.w1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w1
    self.w2 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_w2
    self.b1 -= learn_rate * d_L_d_ypred * d_ypred_d_h1 * d_h1_d_b1

    # Neuron h2
    self.w3 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w3
    self.w4 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_w4
    self.b2 -= learn_rate * d_L_d_ypred * d_ypred_d_h2 * d_h2_d_b2

    # Neuron o1
    self.w5 -= learn_rate * d_L_d_ypred * d_ypred_d_w5
```

```
        self.w6 -= learn_rate * d_L_d_ypred * d_ypred_d_w6
        self.b3 -= learn_rate * d_L_d_ypred * d_ypred_d_b3

      # --- Calculate total loss at the end of each epoch
      if epoch % 10 == 0:
        y_preds = np.apply_along_axis(self.feedforward, 1, data)
        loss = mse_loss(all_y_trues, y_preds)
        print("Epoch %d loss: %.3f" % (epoch, loss))

# Define dataset
data = np.array([
  [-2, -1],   # Alice
  [25, 6],    # Bob
  [17, 4],    # Charlie
  [-15, -6], # Diana
])
all_y_trues = np.array([
  1, # Alice
  0, # Bob
  0, # Charlie
  1, # Diana
])

# Train our neural network!
network = OurNeuralNetwork()
network.train(data, all_y_trues)
```

```
Epoch 0 loss: 0.295
Epoch 10 loss: 0.174
Epoch 20 loss: 0.127
Epoch 30 loss: 0.103
Epoch 40 loss: 0.085
Epoch 50 loss: 0.071
Epoch 60 loss: 0.060
Epoch 70 loss: 0.051
Epoch 80 loss: 0.044
Epoch 90 loss: 0.039
Epoch 100 loss: 0.035
Epoch 110 loss: 0.031
Epoch 120 loss: 0.028
Epoch 130 loss: 0.025
Epoch 140 loss: 0.023
Epoch 150 loss: 0.021
Epoch 160 loss: 0.020
Epoch 170 loss: 0.018
Epoch 180 loss: 0.017
Epoch 190 loss: 0.016
Epoch 200 loss: 0.015
Epoch 210 loss: 0.014
Epoch 220 loss: 0.013
Epoch 230 loss: 0.013
Epoch 240 loss: 0.012
Epoch 250 loss: 0.011
Epoch 260 loss: 0.011
```

```
Epoch 270 loss: 0.010
Epoch 280 loss: 0.010
Epoch 290 loss: 0.010
Epoch 300 loss: 0.009
Epoch 310 loss: 0.009
Epoch 320 loss: 0.009
Epoch 330 loss: 0.008
Epoch 340 loss: 0.008
Epoch 350 loss: 0.008
Epoch 360 loss: 0.007
Epoch 370 loss: 0.007
Epoch 380 loss: 0.007
Epoch 390 loss: 0.007
Epoch 400 loss: 0.007
Epoch 410 loss: 0.006
Epoch 420 loss: 0.006
Epoch 430 loss: 0.006
Epoch 440 loss: 0.006
Epoch 450 loss: 0.006
Epoch 460 loss: 0.006
Epoch 470 loss: 0.005
Epoch 480 loss: 0.005
Epoch 490 loss: 0.005
Epoch 500 loss: 0.005
Epoch 510 loss: 0.005
Epoch 520 loss: 0.005
Epoch 530 loss: 0.005
Epoch 540 loss: 0.005
Epoch 550 loss: 0.005
Epoch 560 loss: 0.004
Epoch 570 loss: 0.004
Epoch 580 loss: 0.004
Epoch 590 loss: 0.004
Epoch 600 loss: 0.004
Epoch 610 loss: 0.004
Epoch 620 loss: 0.004
Epoch 630 loss: 0.004
Epoch 640 loss: 0.004
Epoch 650 loss: 0.004
Epoch 660 loss: 0.004
Epoch 670 loss: 0.004
Epoch 680 loss: 0.004
Epoch 690 loss: 0.004
Epoch 700 loss: 0.003
Epoch 710 loss: 0.003
Epoch 720 loss: 0.003
Epoch 730 loss: 0.003
Epoch 740 loss: 0.003
Epoch 750 loss: 0.003
Epoch 760 loss: 0.003
Epoch 770 loss: 0.003
Epoch 780 loss: 0.003
Epoch 790 loss: 0.003
Epoch 800 loss: 0.003
Epoch 810 loss: 0.003
Epoch 820 loss: 0.003
Epoch 830 loss: 0.003
```

```
Epoch 840 loss: 0.003
Epoch 850 loss: 0.003
Epoch 860 loss: 0.003
Epoch 870 loss: 0.003
Epoch 880 loss: 0.003
Epoch 890 loss: 0.003
Epoch 900 loss: 0.003
Epoch 910 loss: 0.003
Epoch 920 loss: 0.003
Epoch 930 loss: 0.003
Epoch 940 loss: 0.002
Epoch 950 loss: 0.002
Epoch 960 loss: 0.002
Epoch 970 loss: 0.002
Epoch 980 loss: 0.002
Epoch 990 loss: 0.002
```

```
# Make some predictions
emily = np.array([-7, -3]) # 128 pounds, 63 inches
frank = np.array([20, 2]) # 155 pounds, 68 inches
print("Emily: %.3f" % network.feedforward(emily)) # 0.951 - F
print("Frank: %.3f" % network.feedforward(frank)) # 0.039 - M
```

```
Emily: 0.949
Frank: 0.039
```

JP1. Build a simplified clone of IMDB Top 250 movies using metadata collection from IMDB. The following are the steps involved: -Decide on the metric or score to rate movies on -Calculate the score for every movie -Sort the movies based on the score and output the top results. -Use the Full Movie Lens Dataset.

```
#Importing relevant libraries
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')

# Load Movies Metadata
metadata = pd.read_csv('/content/drive/My Drive/Colab Notebooks/movies_meta
data.csv', low_memory=False)
# Print the first three rows
metadata.head(3)
```

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overview	...	release_date	revenue	runtime
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody, Andy's toys live happily in his ...	...	1995-10-30	373554033.0	81.0
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an encha...	...	1995-12-15	262797249.0	104.0
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[{'id': 10749, 'name': 'Romance'}, {'id': 35,...	NaN	15602	tt0113228	en	Grumpier Old Men	A family wedding reignites the ancient feud be...	...	1995-12-22	0.0	101.0

revenue	runtime	spoken_languages	status	tagline	title	video	vote_average	vote_count
373554033.0	81.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	NaN	Toy Story	False	7.7	5415.0
262797249.0	104.0	[{'iso_639_1': 'en', 'name': 'English'}, {'iso...	Released	Roll the dice and unleash the excitement!	Jumanji	False	6.9	2413.0
0.0	101.0	[{'iso_639_1': 'en', 'name': 'English'}]	Released	Still Yelling. Still Fighting. Still Ready for...	Grumpier Old Men	False	6.5	92.0

```
# Calculate mean of vote average column
C = metadata['vote_average'].mean()
print(C)
```

```
5.618207215134185
```

```
# Calculate the minimum number of votes required to be in the chart, m
m = metadata['vote_count'].quantile(0.90)
print(m)
```

```
160.0
```

```
# Filter out all qualified movies into a new DataFrame
q_movies = metadata.copy().loc[metadata['vote_count'] >= m]
q_movies.shape
```

```
(4555, 24)
```

```
metadata.shape
```

```
(45466, 24)
```

```
# Function that computes the weighted rating of each movie
def weighted_rating(x, m=m, C=C):
    v  =  x['vote_count']
    R = x['vote_average']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)

# Define a new feature 'score' and calculate its value with `weighted_ratin
g()`
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)

#Sort movies based on score calculated above
q_movies = q_movies.sort_values('score', ascending=False)

#Print the top 20 movies
q_movies[['title', 'vote_count', 'vote_average', 'score']].head(20)
```

	title	vote_count	vote_average	score
314	The Shawshank Redemption	8358.0	8.5	8.445869
834	The Godfather	6024.0	8.5	8.425439
10309	Dilwale Dulhania Le Jayenge	661.0	9.1	8.421453
12481	The Dark Knight	12269.0	8.3	8.265477
2843	Fight Club	9678.0	8.3	8.256385
292	Pulp Fiction	8670.0	8.3	8.251406
522	Schindler's List	4436.0	8.3	8.206639
23673	Whiplash	4376.0	8.3	8.205404
5481	Spirited Away	3968.0	8.3	8.196055
2211	Life Is Beautiful	3643.0	8.3	8.187171
1178	The Godfather: Part II	3418.0	8.3	8.180076
1152	One Flew Over the Cuckoo's Nest	3001.0	8.3	8.164256
351	Forrest Gump	8147.0	8.2	8.150272
1154	The Empire Strikes Back	5998.0	8.2	8.132919
1176	Psycho	2405.0	8.3	8.132715
18465	The Intouchables	5410.0	8.2	8.125837
40251	Your Name.	1030.0	8.5	8.112532
289	Leon: The Professional	4293.0	8.2	8.107234
3030	The Green Mile	4166.0	8.2	8.104511
1170	GoodFellas	3211.0	8.2	8.077459

JP2. Build a system that recommends movies that are similar to a particular movie. Compute pairwise cosine similarity scores for all movies based on that similarity score threshold. The plot description is available to you as the overview feature in your metadata dataset.

```
#Importing relevant libraries
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')

# Load Movies Metadata
metadata = pd.read_csv('/content/drive/My Drive/Colab Notebooks/movies_meta
data.csv', low_memory=False)
#Print plot overviews of the first 5 movies.
metadata['overview'].head()
```

```
0    Led by Woody, Andy's toys live happily in his ...
1    When siblings Judy and Peter discover an encha...
2    A family wedding reignites the ancient feud be...
3    Cheated on, mistreated and stepped on, the wom...
4    Just when George Banks has recovered from his ...
Name: overview, dtype: object
```

```
#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-
IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')

#Replace NaN with an empty string
metadata['overview'] = metadata['overview'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(metadata['overview'])

#Output the shape of tfidf_matrix
tfidf_matrix.shape
```

```
(45466, 75827)
```

```
#Array mapping from feature integer indices to feature name.
tfidf.get_feature_names()[5000:5010]
```

```
['avails',
 'avaks',
 'avalanche',
 'avalanches',
 'avallone',
 'avalon',
 'avant',
 'avanthika',
 'avanti',
 'avaracious']
```

```
# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

#Construct a reverse map of indices and movie titles
indices = pd.Series(metadata.index, index=metadata['title']).drop_duplicate
s()

indices[:10]
```

```
title
Toy Story                      0
Jumanji                        1
Grumpier Old Men               2
Waiting to Exhale              3
Father of the Bride Part II    4
Heat                           5
Sabrina                        6
Tom and Huck                   7
Sudden Death                   8
GoldenEye                      9
dtype: int64
```

```
# Function that takes in movie title as input and outputs most similar movi
es
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]
```

```
    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return metadata['title'].iloc[movie_indices]

get_recommendations('The Dark Knight Rises')
```

```
12481                                      The Dark Knight
150                                         Batman Forever
1328                                        Batman Returns
15511                            Batman: Under the Red Hood
585                                                 Batman
21194    Batman Unmasked: The Psychology of the Dark Kn...
9230                     Batman Beyond: Return of the Joker
18035                                     Batman: Year One
19792              Batman: The Dark Knight Returns, Part 1
3095                           Batman: Mask of the Phantasm
Name: title, dtype: object
```