

**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:** | | **Submitted By:** | |
| Dr. Rakhee | | Sourav Gusain ( 03511604422 ) | |
| (Assistant Professor) | | MCA 3rd Sem, Sec 1 | |

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Problem Description** | **Date of Execution** | **Sign.** |
| AP1 | Create a solution to solve the Graph Traversal using BFS? | 16-09-2022 |  |
| 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? | 17-09-2022 |  |
| AP3 | Create a solution to solve the graph traversal using DFS? | 18-09-2022 |  |
| AP4 | Create a solution to solve the following Sudoku using DFS? | 19-09-2022 |  |
| 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 | 19-09-2022 |  |

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

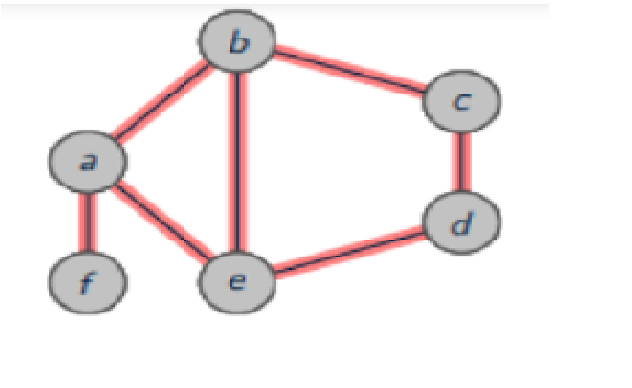
|  |  |  |  |
| --- | --- | --- | --- |
|  | Find the most cost-effective path to reach the final state from initial state using A\* Algorithm. F(n)=g(n)+h(n). Consider g(n) = Depth of node and h(n) = Number of misplaced tiles. |  |  |
| BP3 | Consider the following graph    The numbers written on edges 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. | 25-09-2022 |  |
| 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 |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 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 | 02-01-2022 |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | * A hidden layer with 2 neurons (h1, h2) * An output layer with 1 neuron (o1) |  |  |
| 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 = " ")

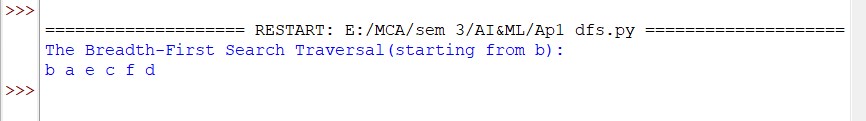
for neighbour in graph[m]:

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**



**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:

1. = QueueEntry()

a.dist = qe.dist + 1 visited[j] = True

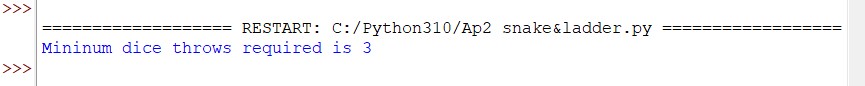
a.v = move[j] if move[j] != -1 else j queue.append(a)

j += 1

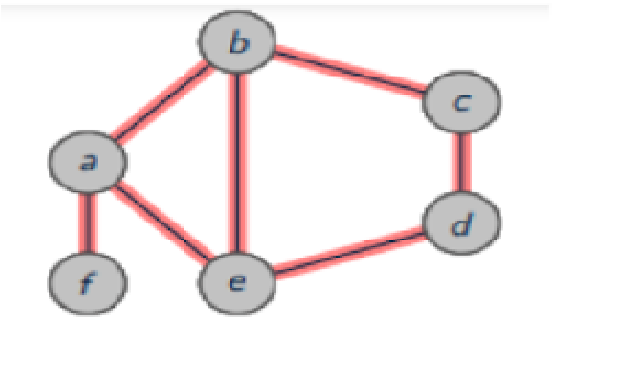
return qe.dist

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**



**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):

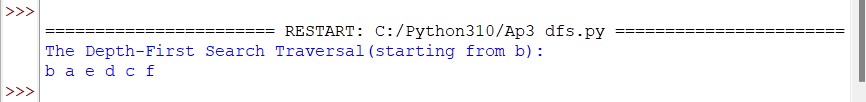
if node not in visited: print(node, end=" ") visited.add(node) for neighbour in graph[node]:

dfs(visited,graph,neighbour)

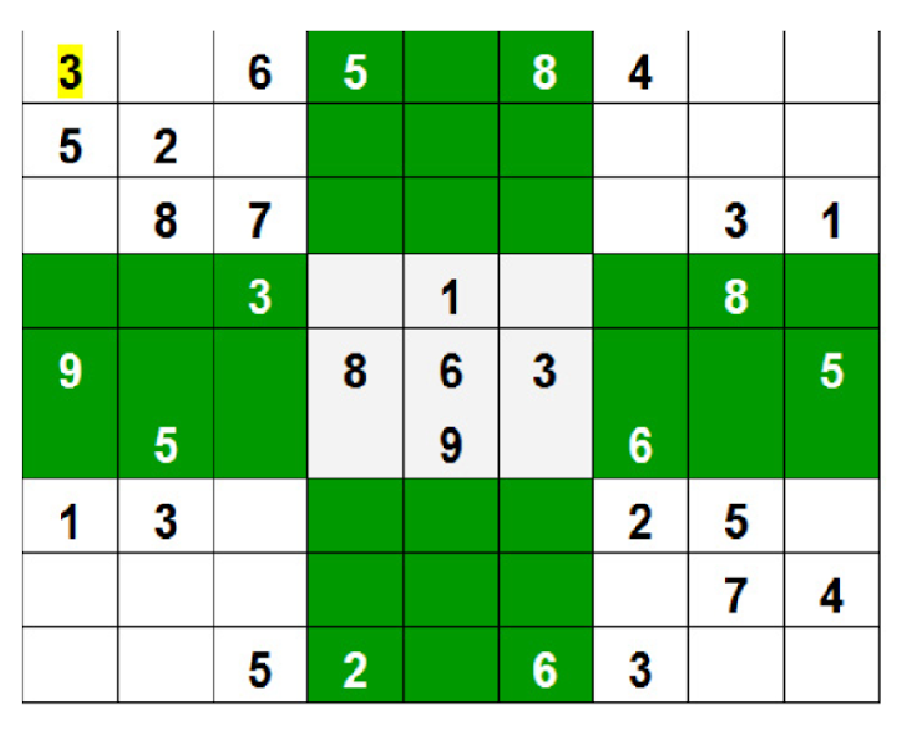
print("The Depth-First Search Traversal(starting from b):")

dfs(visited, graph, 'b')

**OUTPUT**



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



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]]

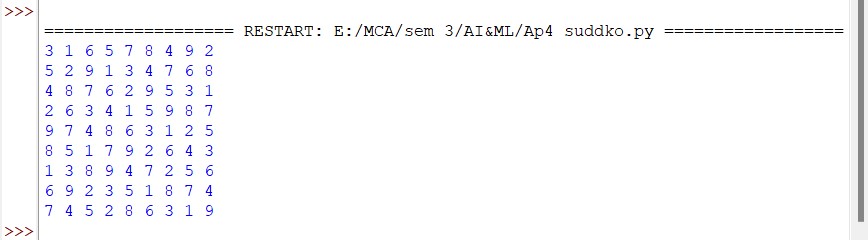
if (solveSudoku(grid, 0, 0)):

printing(grid)

else:

print("no solution exists ")

**OUTPUT**



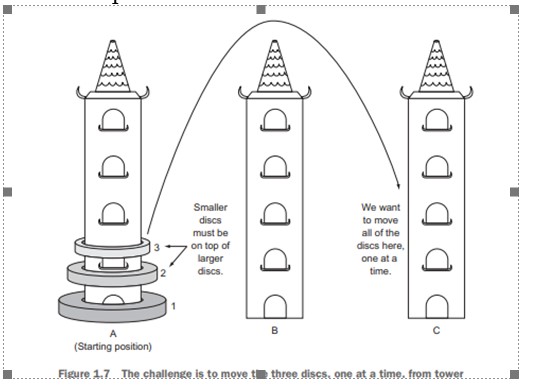
**BP1. The Towers of Hanoi**

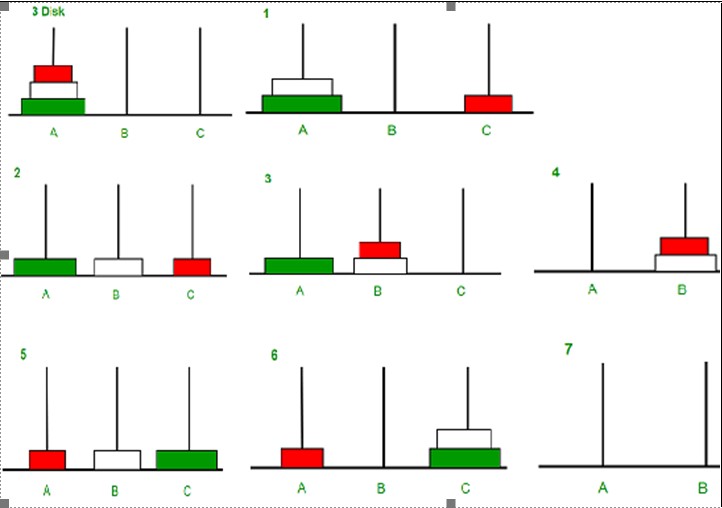
**Three vertical pegs (henceforth “towers”) stand tall. We will label them A, B, and C.**

**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**





def tower\_of\_hanoi(disks, source, auxiliary, target):

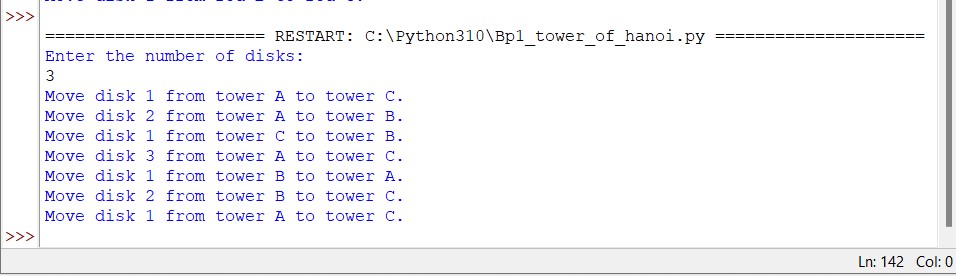
if(disks == 1): print('Move disk 1 from tower {} to tower {}.'.format(source, target)) return

tower\_of\_hanoi(disks - 1, source, target, auxiliary)

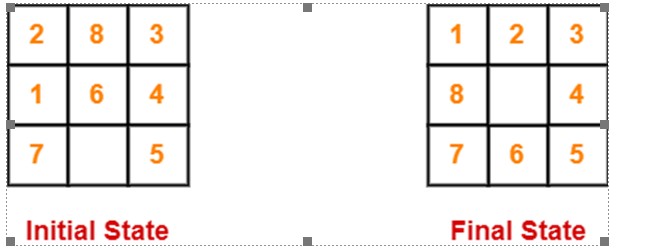
print('Move disk {} from tower {} to tower {}.'.format(disks, source, target)) tower\_of\_hanoi(disks - 1, auxiliary, source, target)

disks = int(input('Enter the number of disks: \n')) tower\_of\_hanoi(disks, 'A', 'B', 'C')

**OUTPUT**



**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():

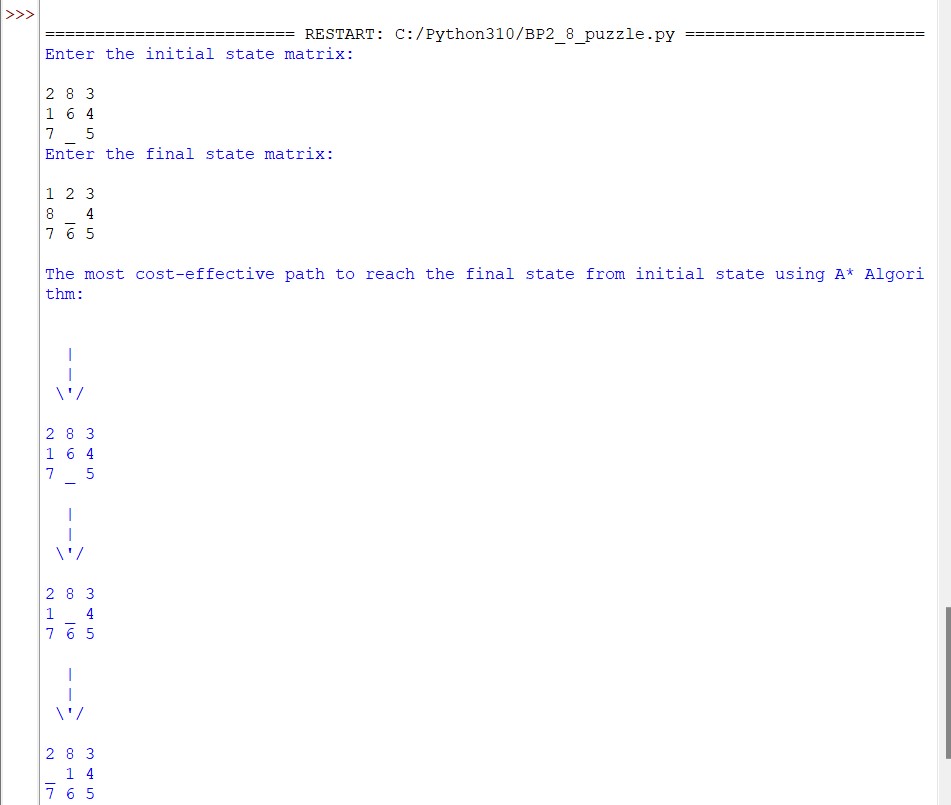
i.fval = self.f(i,final) self.open.append(i)

self.closed.append(cur) del self.open[0]

self.open.sort(key = lambda x:x.fval,reverse=False)

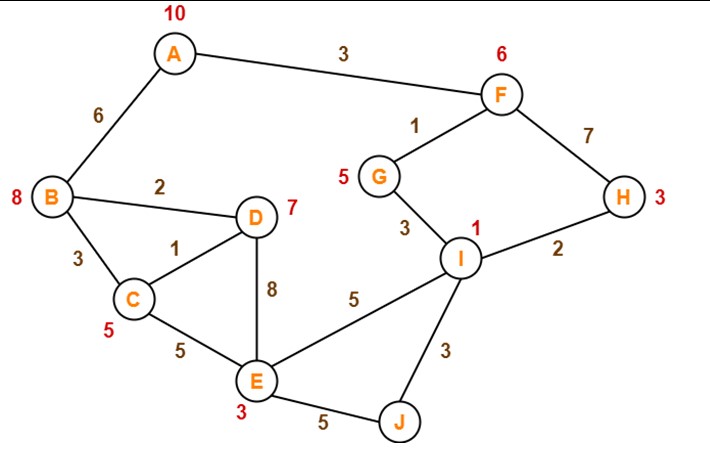
puz = Puzzle(3) puz.process()

**OUTPUT**





**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)],

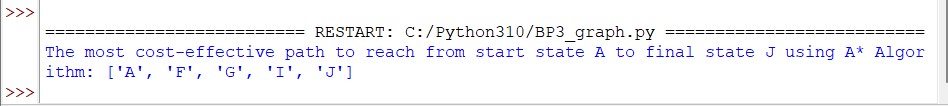
'F': [('A', 3), ('G', 1), ('H', 7)],

'G': [('F', 1), ('I', 3)], 'H': [('F', 7), ('I', 2)],

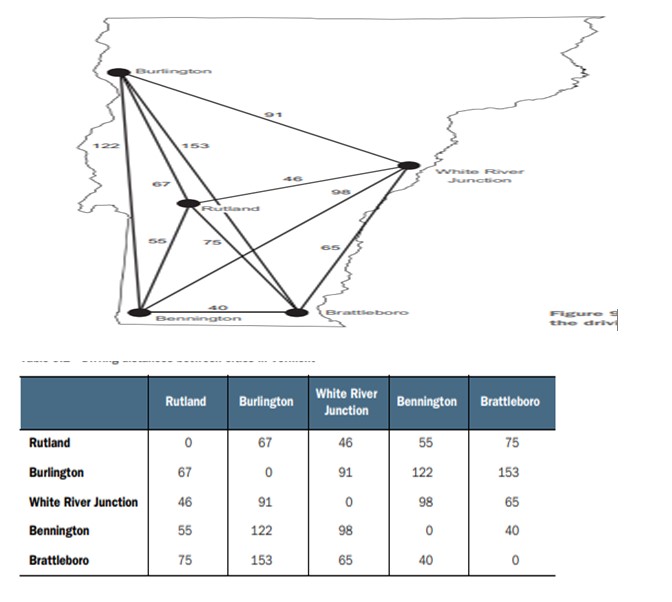
'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],

} aStarAlgo('A', 'J')

**OUTPUT**



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



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},

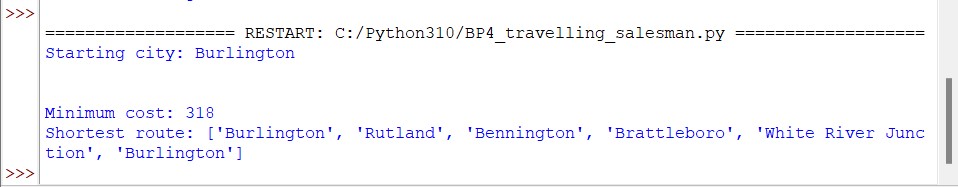
'Brattleboro': {'Rutland': 75, 'Burlington': 153, 'White River Junction': 65, 'Bennington': 40, 'Brattleboro': 0}, }

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**



**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', 'cla ss']

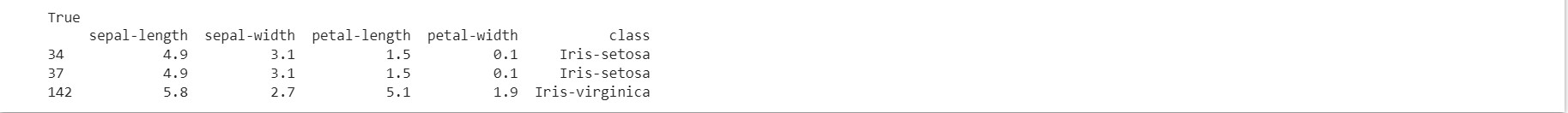
dataset = read\_csv(url, names=names)

# 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])



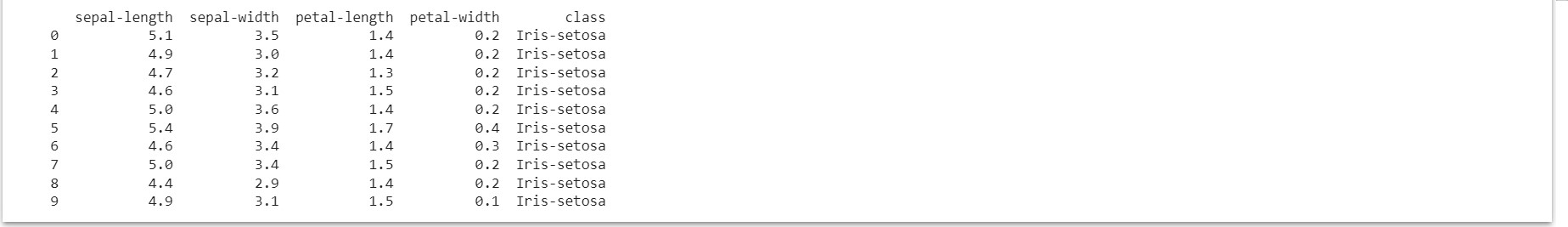
# delete rows of duplicate data from the dataset print(dataset.shape)

# delete duplicate rows

dataset.drop\_duplicates(inplace=True) print(dataset.shape)



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



# 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')))

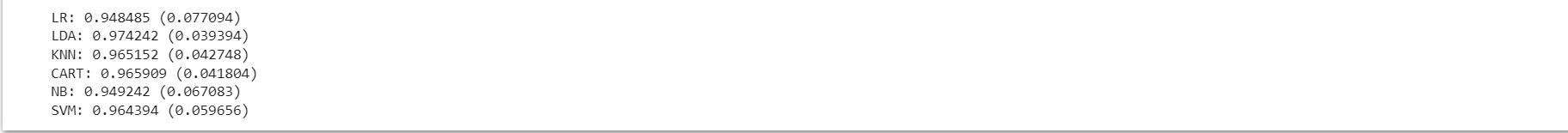
models.append(('LDA', LinearDiscriminantAnalysis())) models.append(('KNN', KNeighborsClassifier())) models.append(('CART', DecisionTreeClassifier())) models.append(('NB', GaussianNB())) models.append(('SVM', SVC(gamma='auto')))

# 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')

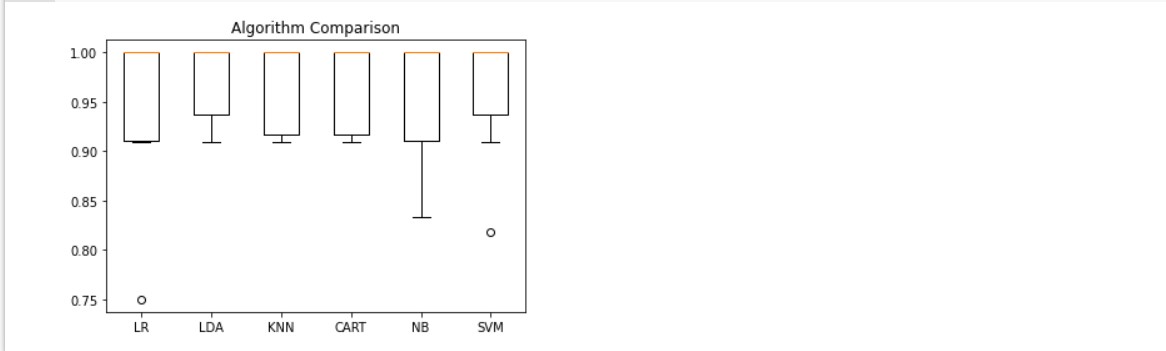
results.append(cv\_results) names.append(name)

print('%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std()))



# 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)

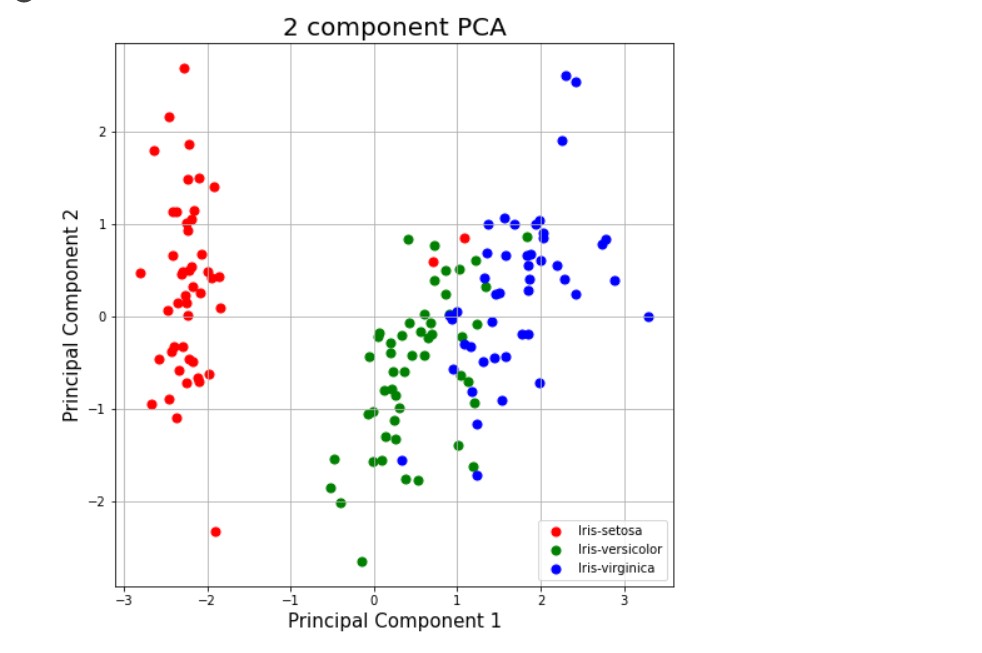
targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'] colors = ['r', 'g', 'b'] for target, color in zip(targets,colors):

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\_



**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', 'cla ss']

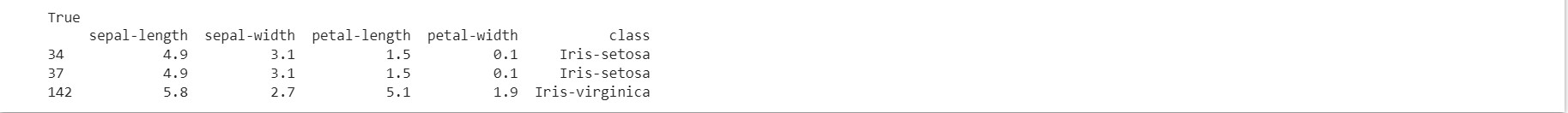
dataset = read\_csv(url, names=names)

# 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])



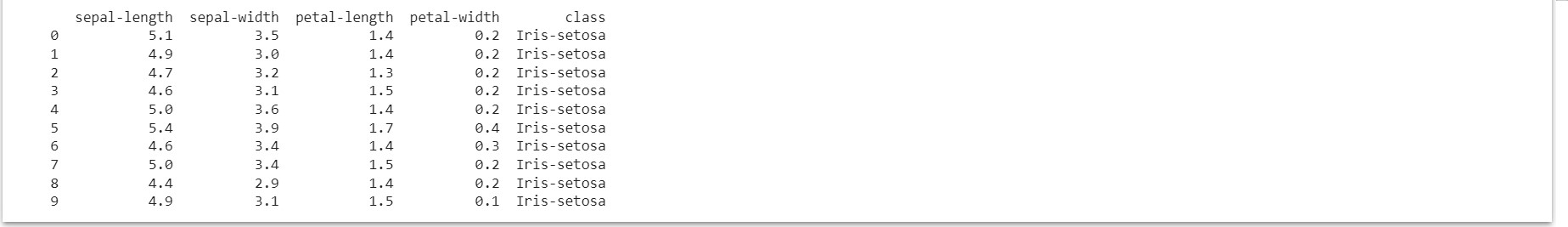
# delete rows of duplicate data from the dataset print(dataset.shape)

# delete duplicate rows

dataset.drop\_duplicates(inplace=True) print(dataset.shape)



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



# 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)

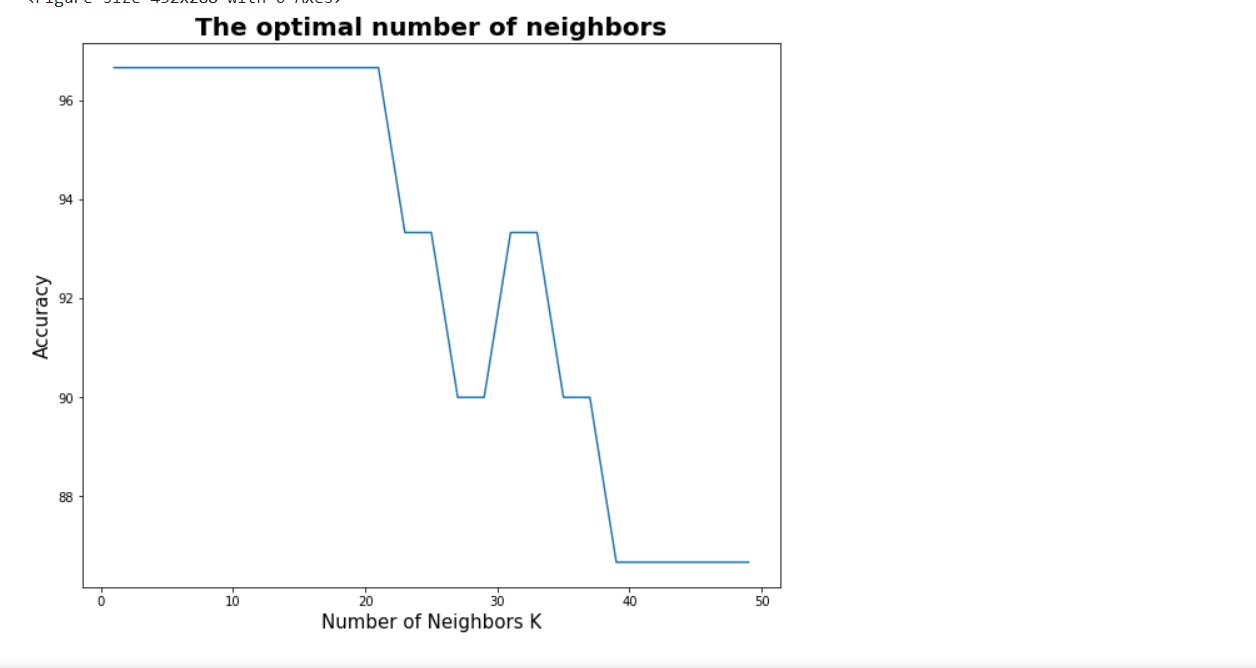
# plotting graph of k-values vs accuracy pyplot.figure()

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()



**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

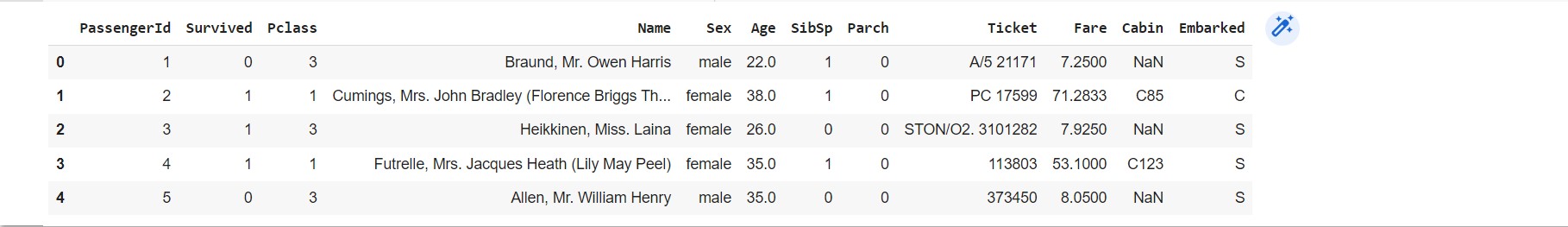
from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression from sklearn.metrics import accuracy\_score from sklearn.feature\_selection import VarianceThreshold from sklearn.model\_selection import cross\_val\_score from sklearn.model\_selection import 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()



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



#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)

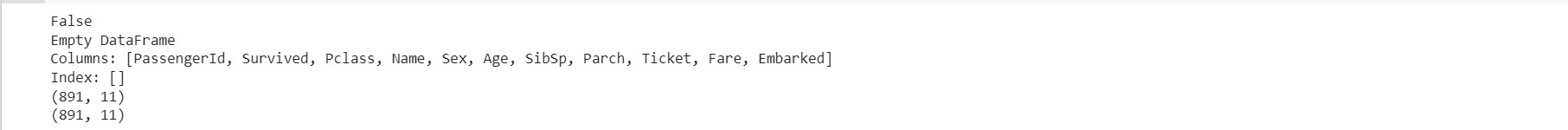
#cleaning data of duplicate data

# 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)



#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)



#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']

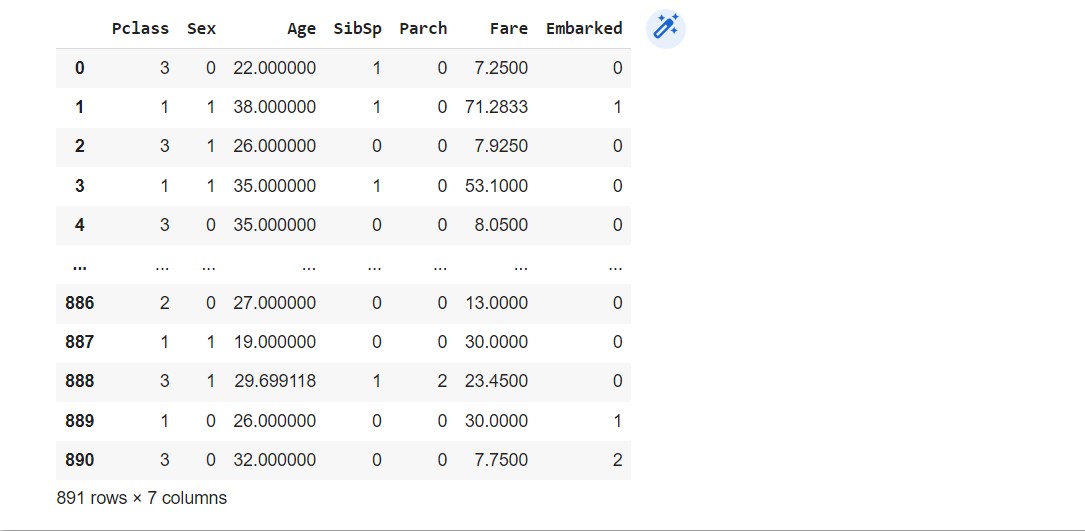
#cleaning data of low variance

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)



#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')))

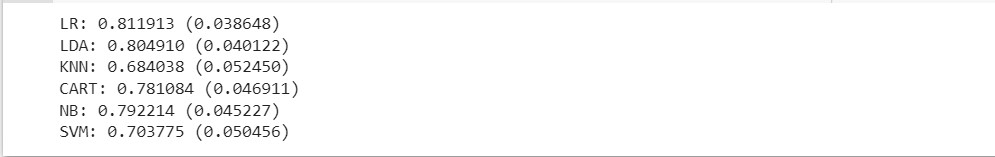
models.append(('LDA', LinearDiscriminantAnalysis())) models.append(('KNN', KNeighborsClassifier())) models.append(('CART', DecisionTreeClassifier())) models.append(('NB', GaussianNB())) models.append(('SVM', SVC(gamma='auto')))

# 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')

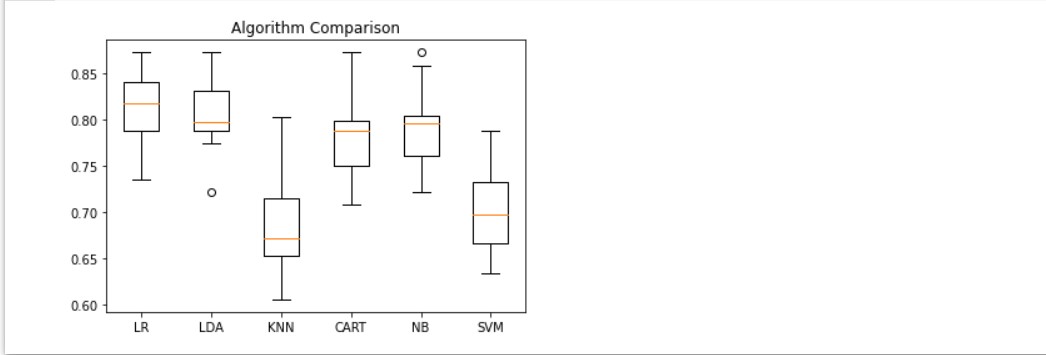
results.append(cv\_results) names.append(name)

print('%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std()))



# 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)



**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))

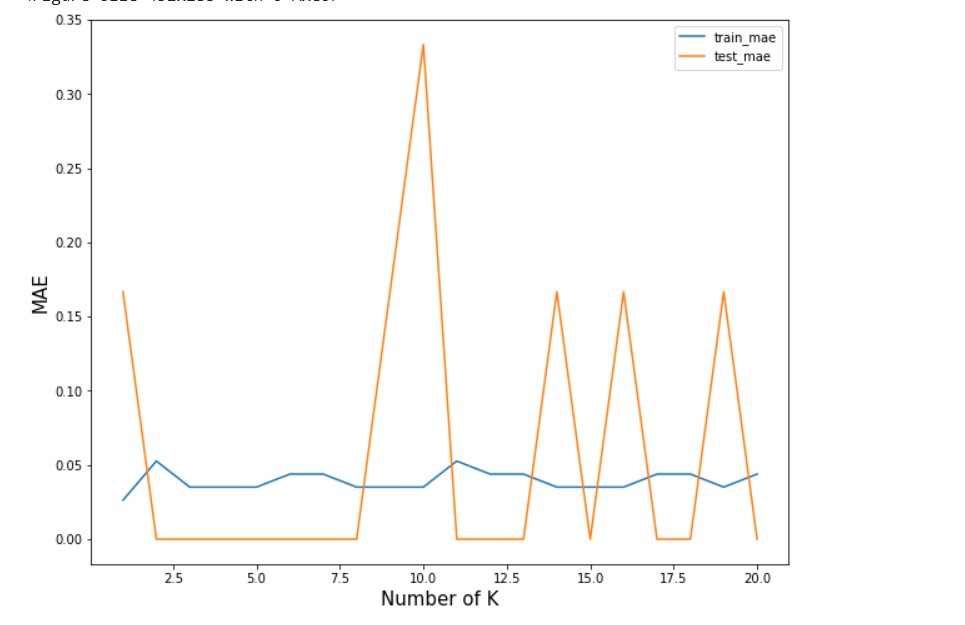
X\_test\_pred = model.predict(X\_test)

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()



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)



**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)

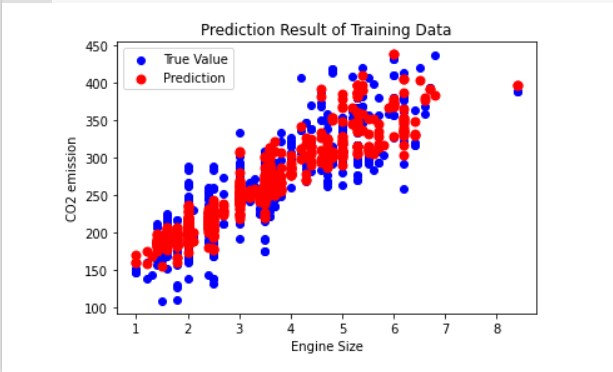
model = LinearRegression() model.fit(X\_train, y\_train) y\_train\_pred = model.predict(X\_train)

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()



**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)

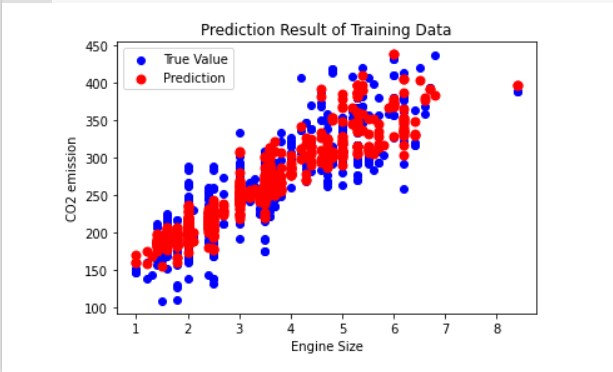
model = LinearRegression() model.fit(X\_train, y\_train) y\_train\_pred = model.predict(X\_train)

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()



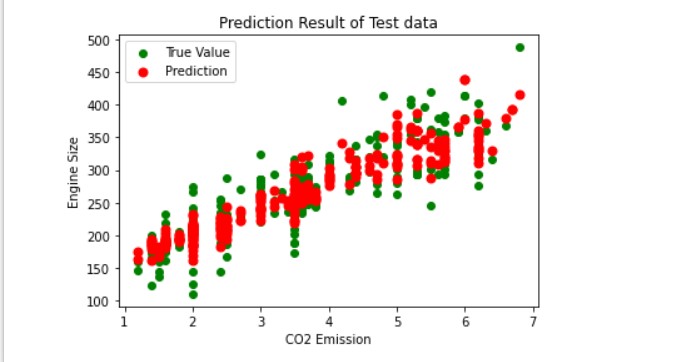
y\_test\_pred = model.predict(X\_test)

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()



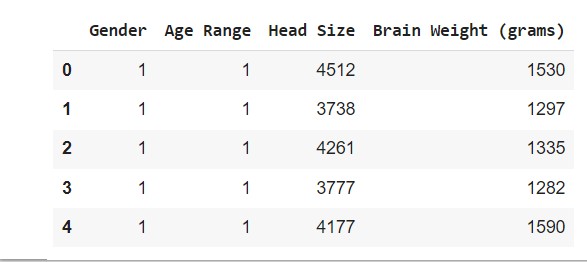
**EP4. Apply Linear Regression and build a model that studies the relationship between the head size and the brain weight of an individual? Evaluate by using least square regression method where RMSE (root mean squared error) and R-squared/R2 will be the model evaluation parameters.**

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()



1. = dtf['Head Size'].values
2. = 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)



# 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

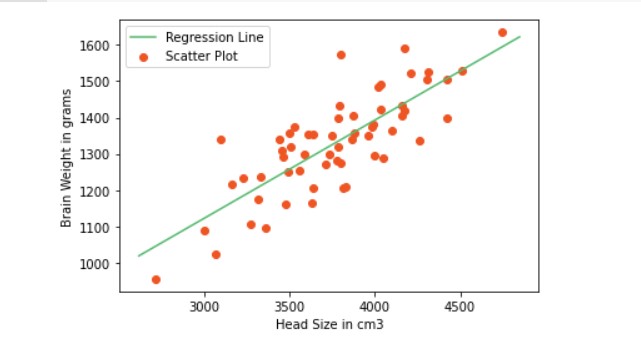
# Ploting Line

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)



# 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)



**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()



data = dtf.values

X, y = data[:, :-1], data[:, -1]

# split the data

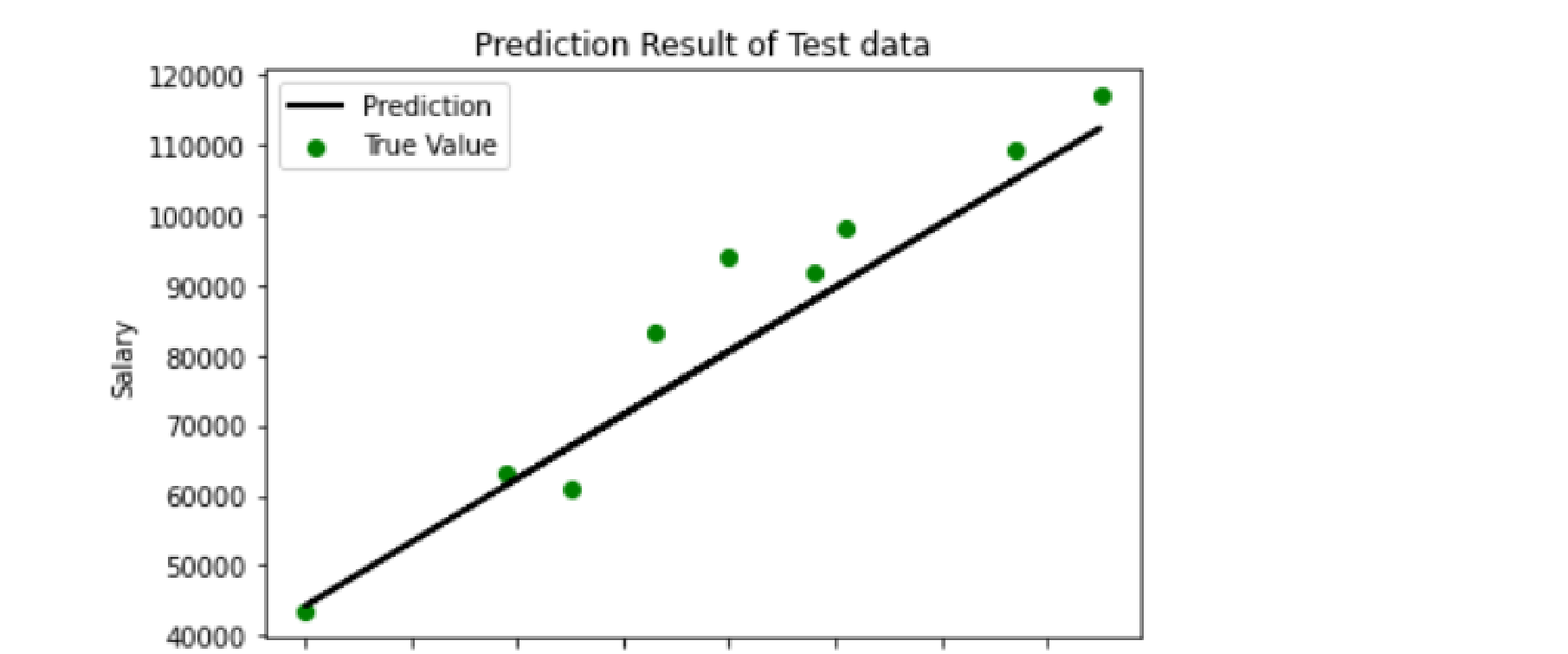
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, ra ndom\_state=1)

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)



rmse = np.sqrt(mean\_squared\_error(y\_test,y\_test\_pred)) print("RMSE =", rmse)



r2\_Score=r2\_score(y\_test, y\_test\_pred) print("R2 square =", r2\_Score)



**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

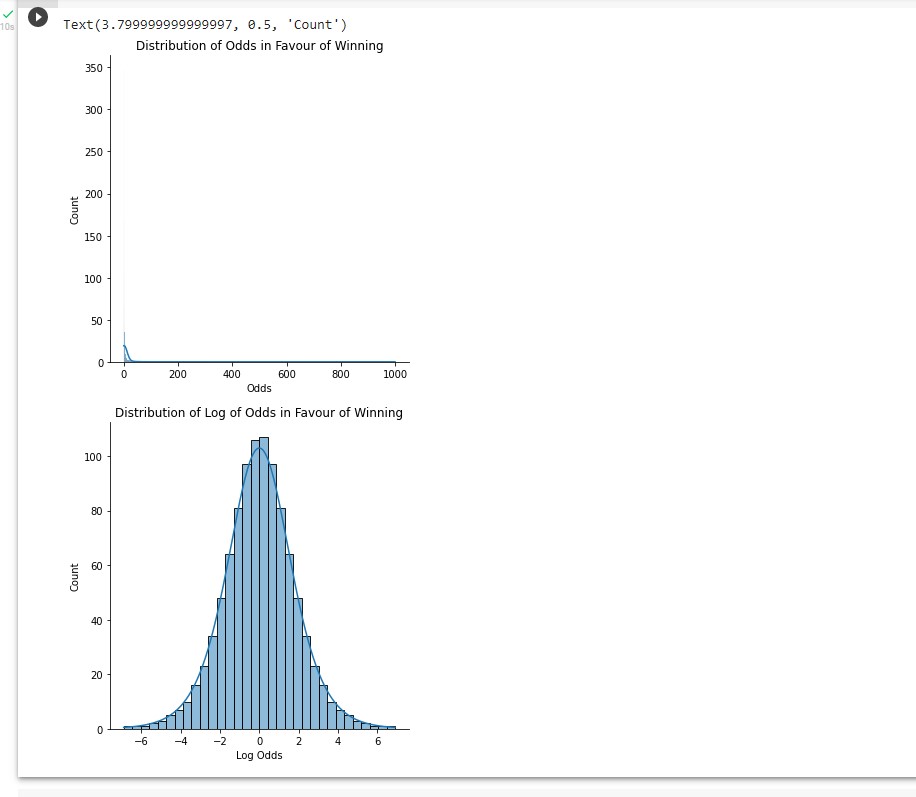
df['Odds\_win']=df['Win']/df['Lose'] df['Odds\_lose']=df['Lose']/df['Win']

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")



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

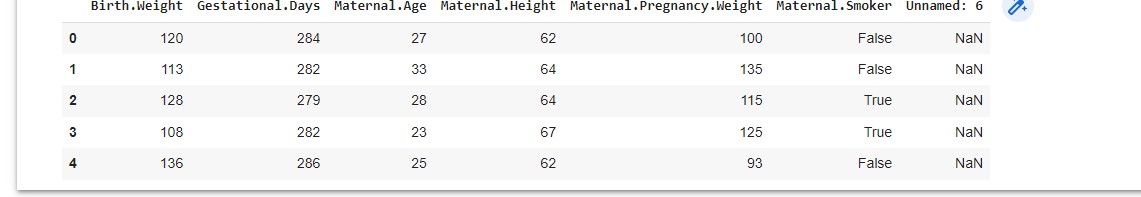
import pandas as pd import numpy as np

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')

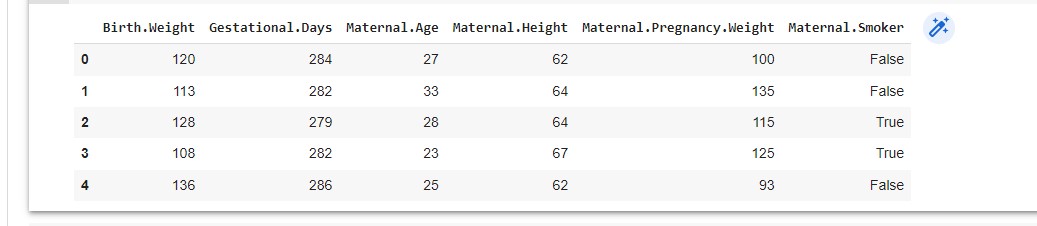
dtf = pd.read\_csv('/content/drive/My Drive/Colab Notebooks/Baby Weight.csv'

) dtf.head()



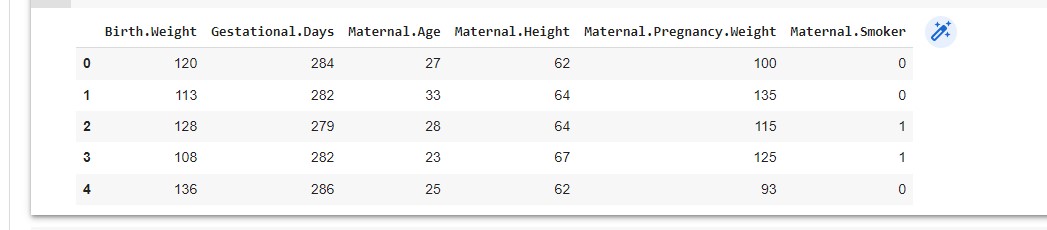
#dropping extra column

dtf=dtf.drop(columns='Unnamed: 6', axis=1) dtf.head()



#convert boolean type values into numerical

dtf.replace({'Maternal.Smoker':{False:0,True:1}},inplace=True) dtf.head()



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)



#calculate ssr

ssr = np.sum((y\_train\_pred - y\_train.mean())\*\*2) print(ssr)



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



**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

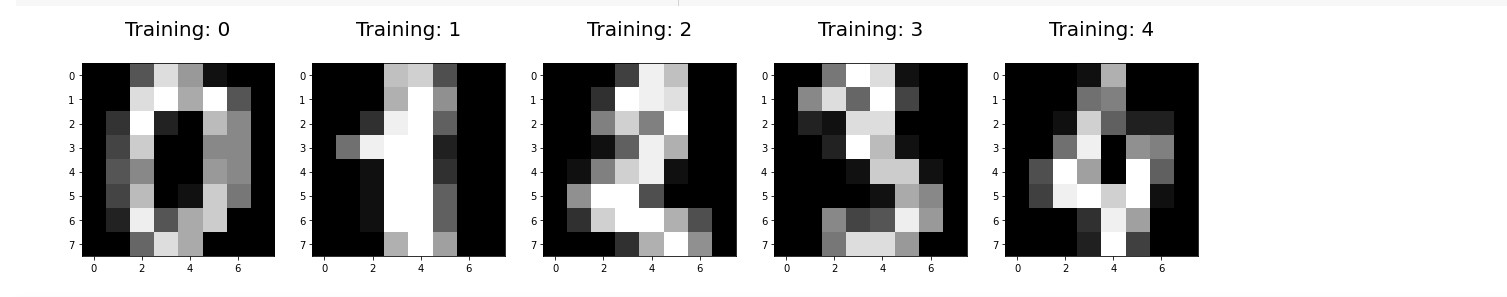
from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression from sklearn import metrics

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)

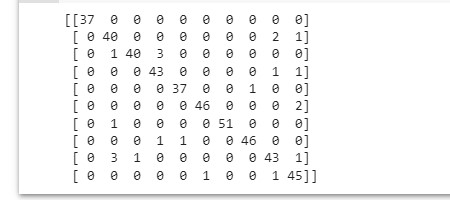
plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray) plt.title('Training: %i\n' % label, fontsize = 20)



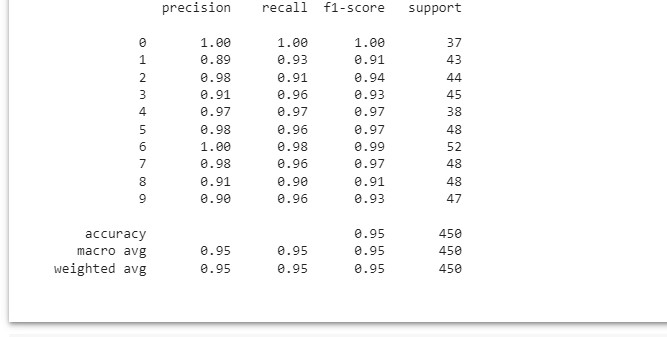
x\_train, x\_test, y\_train, y\_test = train\_test\_split(digits.data, digits.tar get, test\_size=0.25, random\_state=0) logisticRegr = LogisticRegression() logisticRegr.fit(x\_train, y\_train) predictions = logisticRegr.predict(x\_test) score = logisticRegr.score(x\_test, y\_test) print(score)



cm = metrics.confusion\_matrix(y\_test, predictions) print(cm)



print(f"{metrics.classification\_report(y\_test, predictions)}\n")

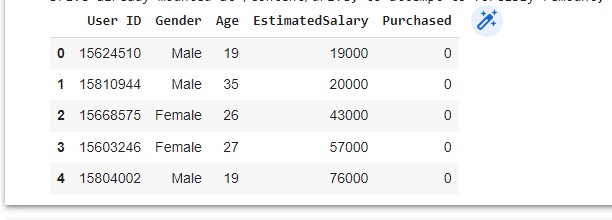


**EP9. Apply logistic regression on userdata.csv dataset to predict the users who may be potential customers to purchase a SUV car? Also generate the confusion matrix to evaluate your model?**

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()



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)

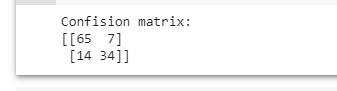
classifier = LogisticRegression() classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

print('Accuracy score of test data : ',metrics.accuracy\_score(y\_test,y\_pred ))



cm = metrics.confusion\_matrix(y\_test, y\_pred) print('Confision matrix: ') print(cm)



**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)



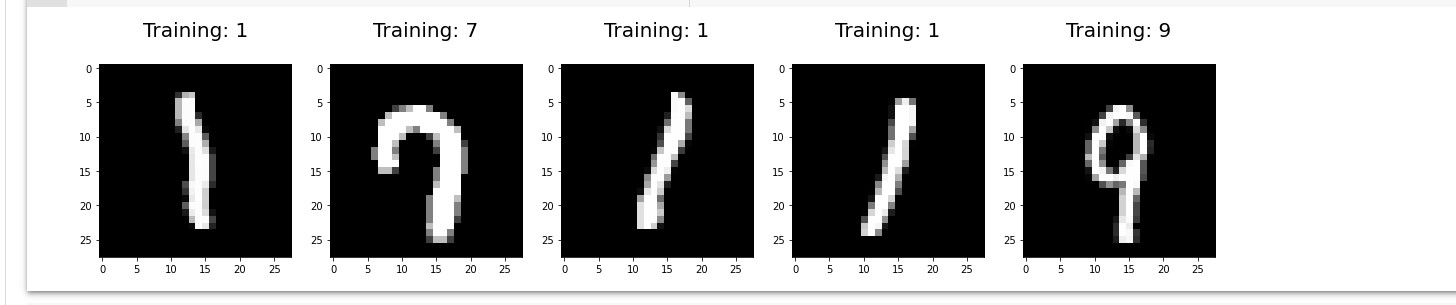
train\_img, test\_img, train\_lbl, test\_lbl = train\_test\_split(X, y, test\_size =1/7.0, random\_state=0)

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)

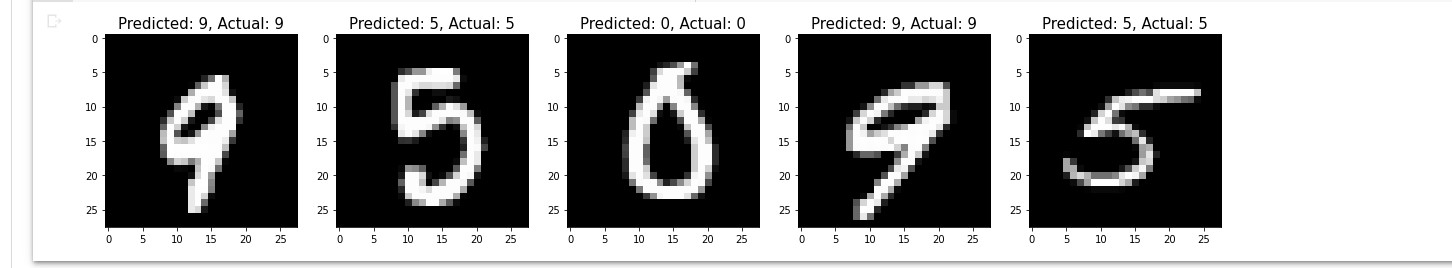


logisticRegr = LogisticRegression(solver = 'lbfgs') logisticRegr.fit(train\_img, train\_lbl) # Make predictions on entire test data predictions = logisticRegr.predict(test\_img)

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)

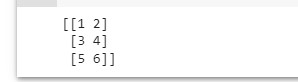


**FP1. Understand dimensionality reduction technique?**

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)



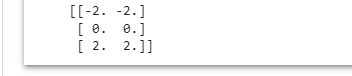
# calculate the mean of each column

M = mean(A.T, axis=1) print(M)



# center columns by subtracting column means

C = A - M print(C)



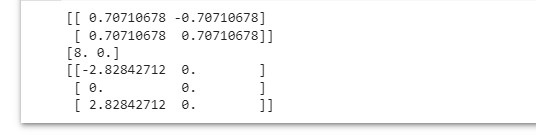
# calculate covariance matrix of centered matrix

V = cov(C.T) print(V)



# eigendecomposition of covariance matrix values, vectors = eig(V) print(vectors) print(values) # project data

P = vectors.T.dot(C.T) print(P.T)



# Principal Component Analysis from numpy import array from sklearn.decomposition import PCA

# define a matrix

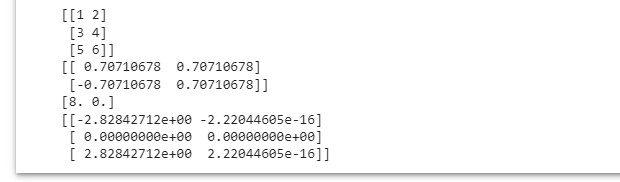
1. = array([[1, 2], [3, 4], [5, 6]]) print(A)

# 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

1. = pca.transform(A)print(B)



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

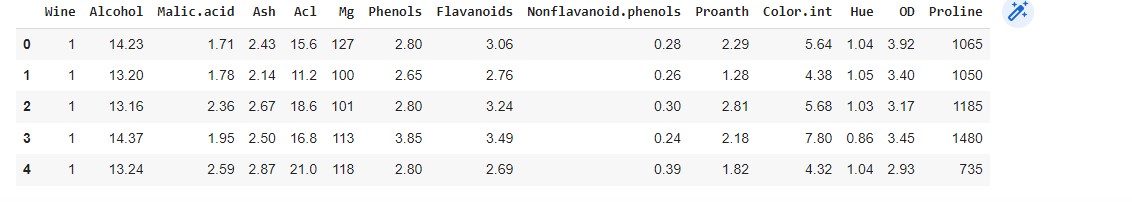
import numpy as np import pandas as pd

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()



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

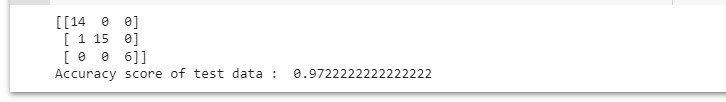
X\_test = pca.transform(X\_test)

# 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))

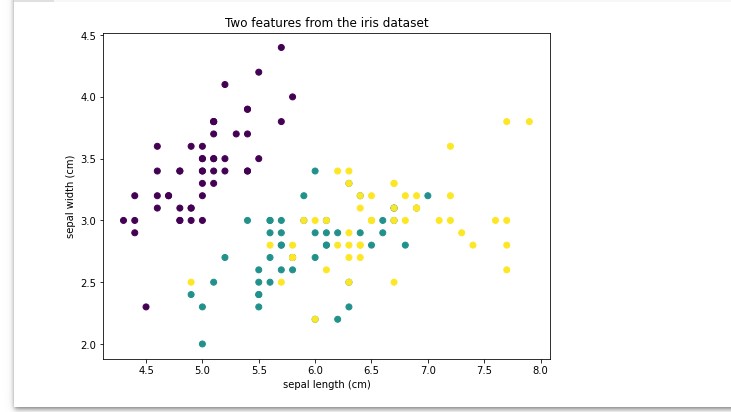


**FP3. Create a basic visualization of Iris dataset in question CP1 using PCA?**

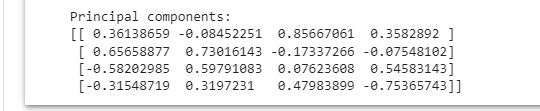
from sklearn.datasets import load\_iris from sklearn.model\_selection import train\_test\_split from sklearn.decomposition import PCA from sklearn.metrics import f1\_score from sklearn.svm import SVC import matplotlib.pyplot as plt

# 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\_)



# Remove PC1

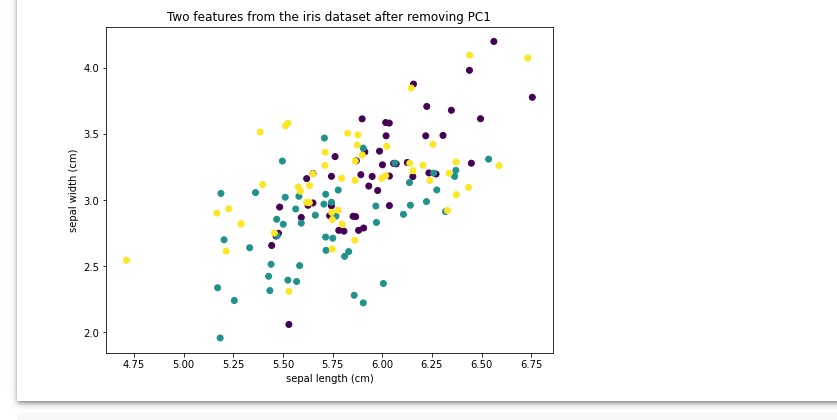
Xmean = X - X.mean(axis=0) value = Xmean @ pca.components\_[0]

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

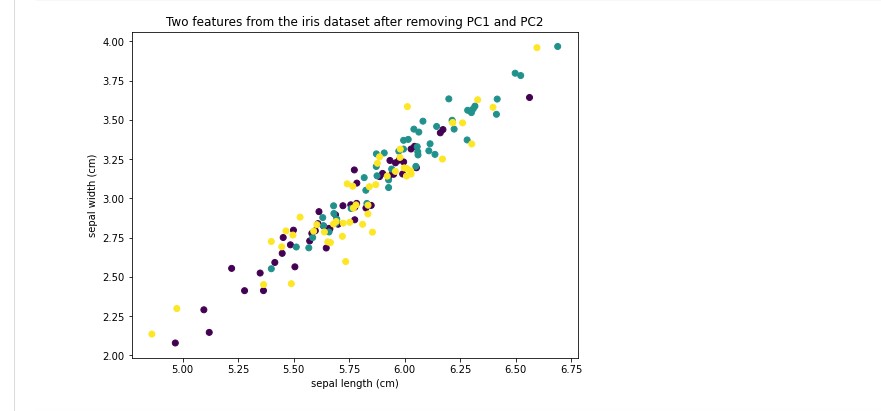
Xmean = X - X.mean(axis=0) value = Xmean @ pca.components\_[1]

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

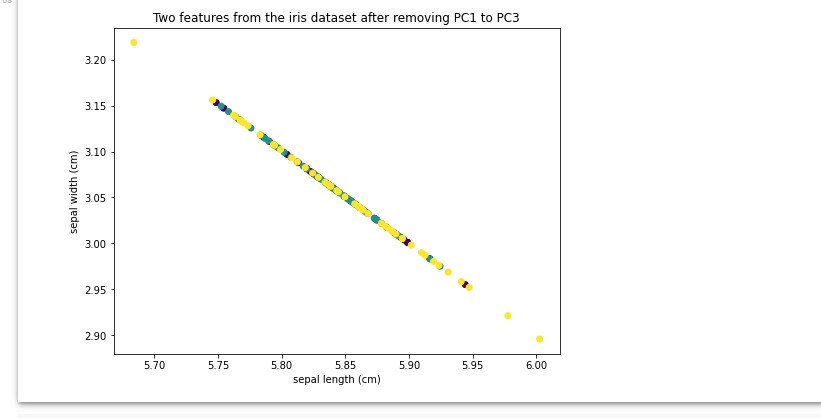
Xmean = X - X.mean(axis=0) value = Xmean @ pca.components\_[2]

pc3 = value.reshape(-1,1) @ pca.components\_[2].reshape(1,-1)

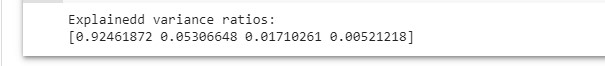
Xremove = Xremove - pc3 plt.figure(figsize=(8,6))

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\_)

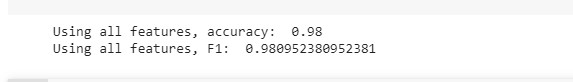


# Split data

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"))



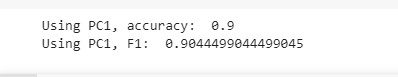
# Run classifier on PC1 mean = X\_train.mean(axis=0) X\_train2 = X\_train - mean

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"))

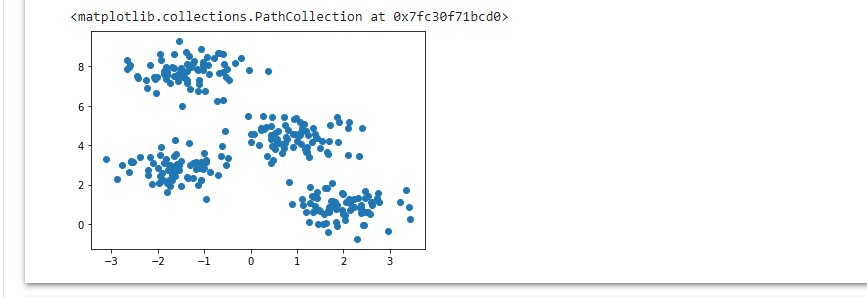


**GP1. Create a random dataset using the make\_blobs() function from sklearn and apply K-means on the same after deciding the number of clusters using the elbow method?**

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])



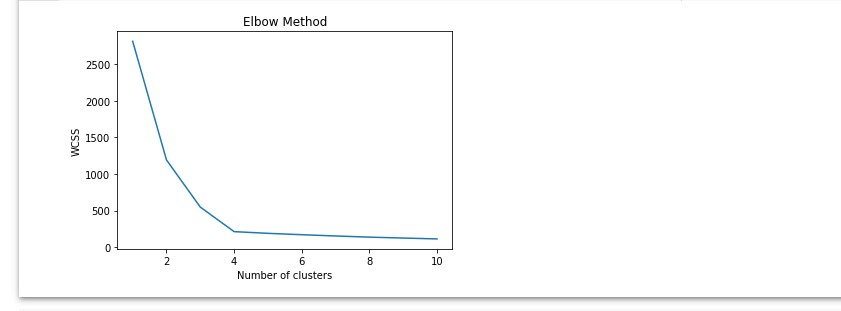
wcss=[]

for i in range (1,11):

kmeans=KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, ra ndom\_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()

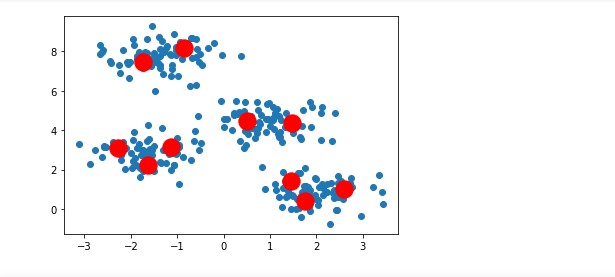


pred\_y=kmeans.fit\_predict(X)

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')

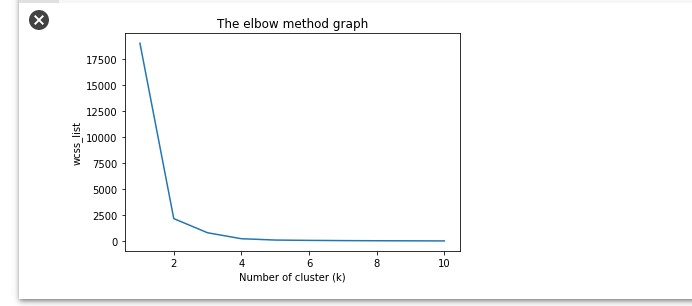
dataset=pd.read\_csv('/content/drive/My Drive/Colab Notebooks/Mall\_Customers \_data.csv')

#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')

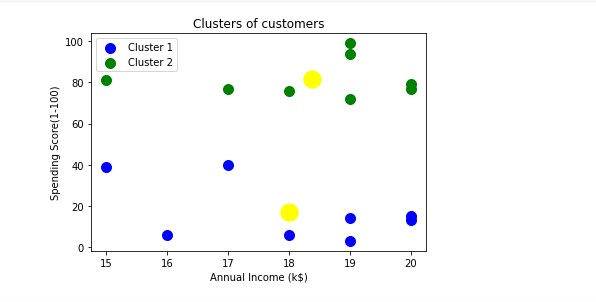
mtp.scatter(x[y\_predict == 1,0],x[y\_predict ==1,1], s=100, c='green',label=

'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-ind ians-diabetes.data.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'cl ass']

dataframe = pandas.read\_csv(url, names=names)

array = dataframe.values

1. = array[:,0:8]
2. = 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())



# 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())



# 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())



**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-india ns-diabetes.data.csv"

names=['preg','plas','pres','skin','test','mass','pedi','age','class'] dataframe=pandas.read\_csv(url,names=names) array=dataframe.values

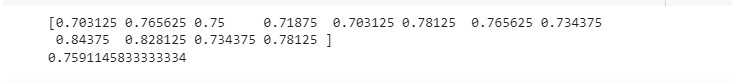
X=array[:,0:8]

Y=array[:,8] seed=7 num\_trees=100

kfold=model\_selection.KFold(n\_splits=12)

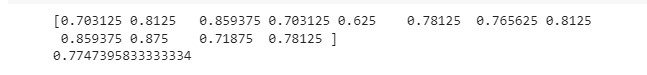
#AdaBoost boosting classifier

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())



# 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())



**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))

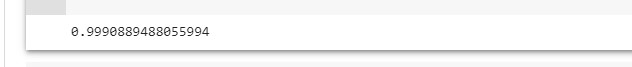
class Neuron: def \_\_init\_\_(self,weights,bias):

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))



**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

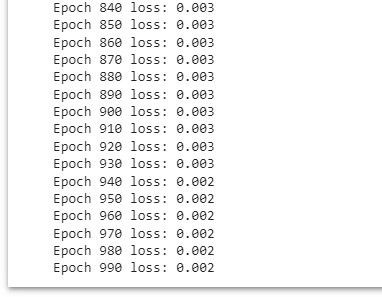
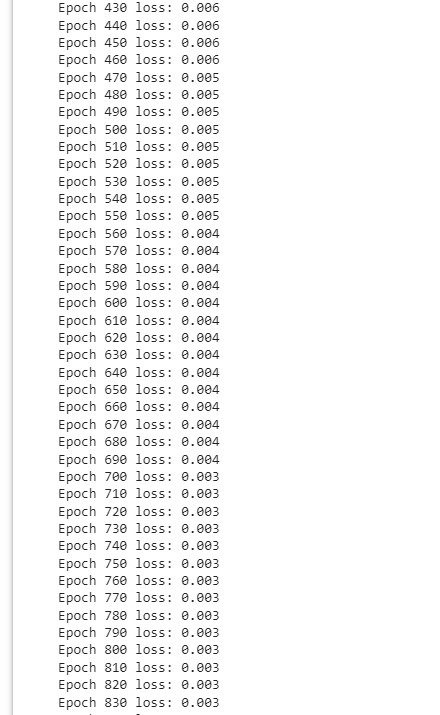
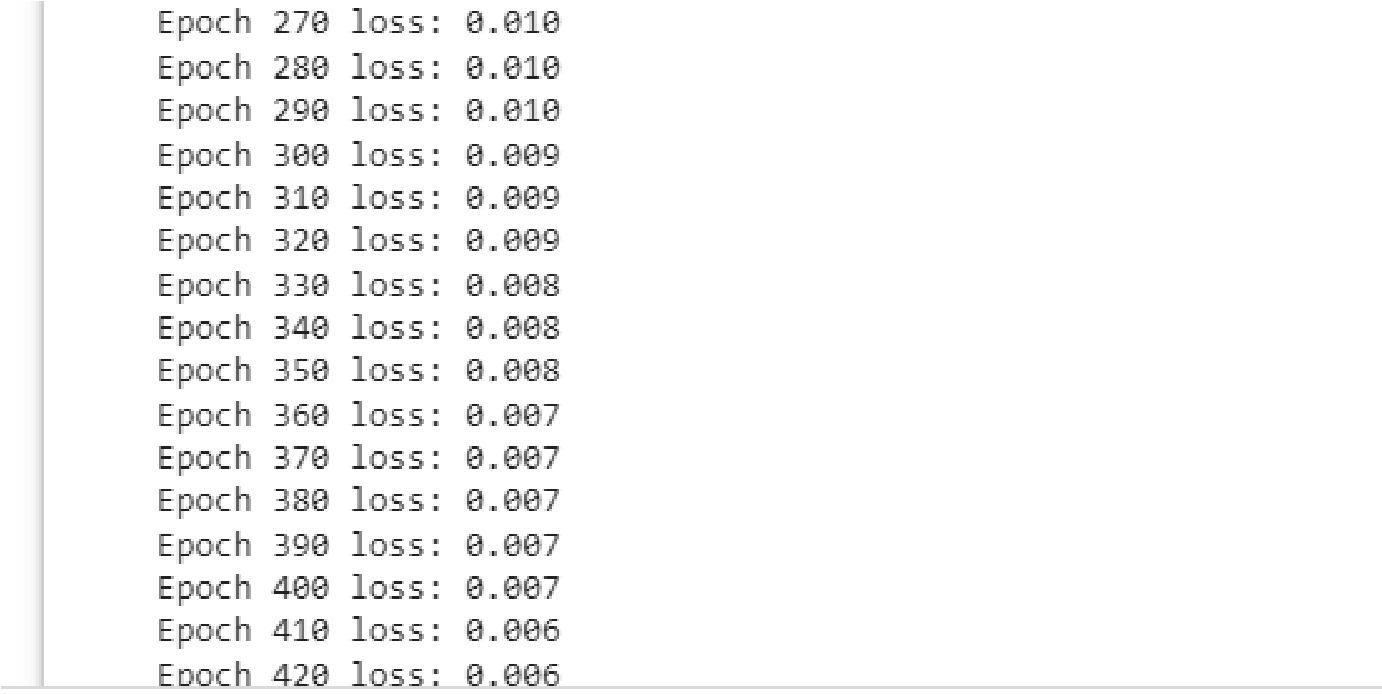
0, # Bob

0, # Charlie

1, # Diana

])

# Train our neural network! network = OurNeuralNetwork() network.train(data, all\_y\_trues)



# 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



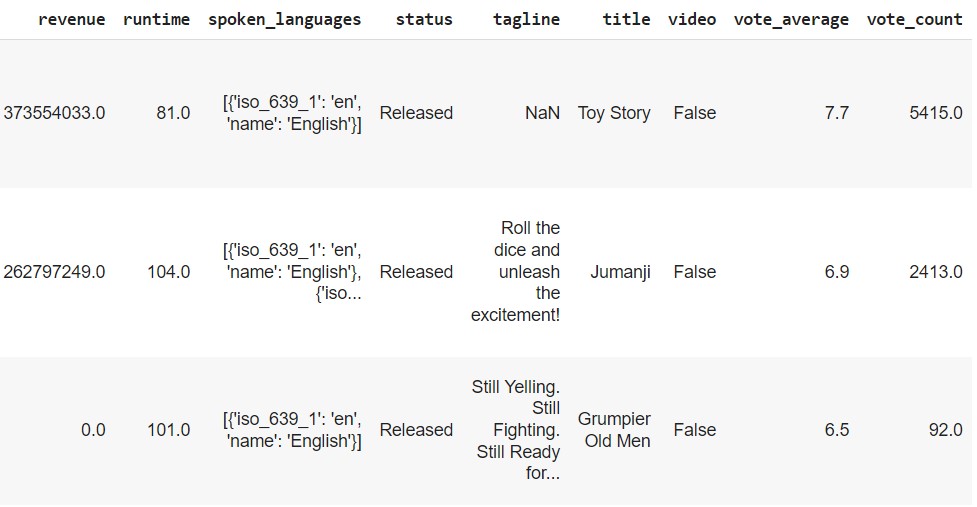
**JP1. Build a simplified clone of IMDB Top 250 movies using metadata collection from IMDB. The following are the steps involved: -Decide on the metric or score to rate movies on -Calculate the score for every movie -Sort the movies based on the score and output the top results. -Use the Full Movie Lens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





# Calculate mean of vote average column

C = metadata['vote\_average'].mean() print(C)



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



# Filter out all qualified movies into a new DataFrame q\_movies = metadata.copy().loc[metadata['vote\_count'] >= m] q\_movies.shape



metadata.shape



# 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)



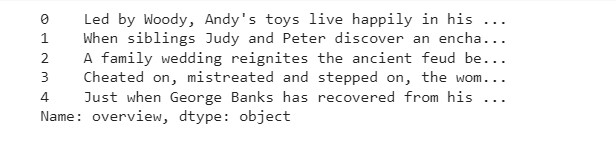
**JP2. Build a system that recommends movies that are similar to a particular movie. Compute pairwise cosine similarity scores for all movies based on that similarity score threshold. The plot description is available to you as the overview feature in your metadata dataset.**

#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()



#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('')

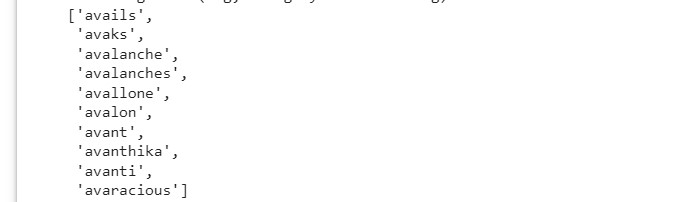
#Construct the required TF-IDF matrix by fitting and transforming the data tfidf\_matrix = tfidf.fit\_transform(metadata['overview'])

#Output the shape of tfidf\_matrix tfidf\_matrix.shape



#Array mapping from feature integer indices to feature name.

tfidf.get\_feature\_names()[5000:5010]



# Import linear\_kernel

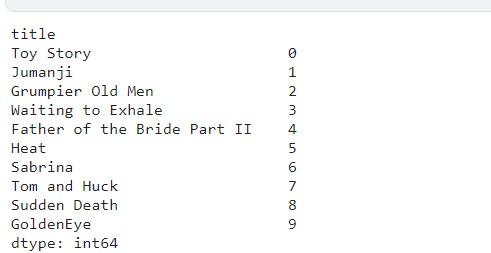
from sklearn.metrics.pairwise import linear\_kernel

# 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]



# 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')

