**DAA**

1. **Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.**

def fibonacci\_iter(n):

if n < 0:

return -1, 1

if n == 0 or n == 1:

return n, 1

steps = 0

a = 0

b = 1

for i in range(2, n+1):

c = a + b

a = b

b = c

steps += 1

return c, steps+1

def fibonacci\_recur(n):

if n < 0:

return -1, 1

if n == 0 or n == 1:

return n, 1

fib1, steps1 = fibonacci\_recur(n-1)

fib2, steps2 = fibonacci\_recur(n-2)

return fib1 + fib2, steps1 + steps2 + 1

if \_\_name\_\_ == '\_\_main\_\_':

n = int(input("Enter a number: "))

print("Iterative:", fibonacci\_iter(n)[0])

print("Steps:", fibonacci\_iter(n)[1])

print("Recursive:", fibonacci\_recur(n)[0])

print("Steps:", fibonacci\_recur(n)[1])

1. **Write a program to implement Huffman Encoding using a greedy strategy.**

import heapq

class node:

def \_\_init\_\_(self, freq, symbol, left=None, right=None):

self.freq = freq

self.symbol = symbol

self.left = left

self.right = right

self.huff = ""

def \_\_lt\_\_(self, other):

return self.freq < other.freq

def printNodes(node, val=""):

newval = val + node.huff

if node.left:

printNodes(node.left, newval)

if node.right:

printNodes(node.right, newval)

else:

print(f"{node.symbol} -> {newval}")

chars = ["a", "b", "c", "d", "e", "f"]

freqs = [5, 9, 12, 13, 16, 45]

nodes = []

for i in range(len(chars)):

heapq.heappush(nodes, node(freqs[i], chars[i]))

while len(nodes) > 1:

left = heapq.heappop(nodes)

right = heapq.heappop(nodes)

left.huff = "0"

right.huff = "1"

newnode = node(left.freq + right.freq, left.symbol + right.symbol, left, right)

heapq.heappush(nodes, newnode)

printNodes(nodes[0])

1. **Write a program to solve a fractional Knapsack problem using a greedy method.**

class Item:

def \_\_init\_\_(self, profit, weight):

self.profit = profit

self.weight = weight

def fractionalKnapsack(w, arr):

arr.sort(key=lambda x: x.profit/x.weight, reverse=True)

finalValue = 0.0

for item in arr:

if w >= item.weight:

finalValue += item.profit

w -= item.weight

else:

finalValue += item.profit \* (w/item.weight)

break

return finalValue

if \_\_name\_\_ == "\_\_main\_\_":

n = int(input("Enter number of items-\n"))

arr = []

for i in range(n):

profit = int(input("Enter profit of item " + str(i + 1) + "-\n"))

weight = int(input("Enter weight of item " + str(i + 1) + "-\n"))

arr.append(Item(profit, weight))

w = int(input("Enter capacity of knapsack-\n"))

print("Maximum value in knapsack: ", fractionalKnapsack(w, arr))

1. **Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.**

def knapsack\_01(n, values, weights, W):

dp = [[0] \* (W+1) for \_ in range(n+1)]

for i in range(n+1):

for w in range(W+1):

if i == 0 or w == 0:

dp[i][w] = 0

elif weights[i-1] <= w:

dp[i][w] = max(dp[i-1][w], dp[i-1][w-weights[i-1]] + values[i-1])

else:

dp[i][w] = dp[i-1][w]

selected\_items = []

i, w = n, W

while i > 0 and w > 0:

if dp[i][w] != dp[i-1][w]:

selected\_items.append(i-1)

w -= weights[i-1]

i -= 1

return dp[n][W], selected\_items

if \_\_name\_\_ == "\_\_main\_\_":

n = 3

values = [60, 100, 120]

weights = [10, 20, 30]

W = 50

max\_value, selected\_items = knapsack\_01(n, values, weights, W)

print("Maximum value:", max\_value)

print("Selected items:", selected\_items)

1. **Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen’s matrix.**

def solveNQueens(n: int, first\_queen\_col: int):

col = set()

posDiag = set()

negDiag = set()

res = []

board = [["."] \* n for \_ in range(n)]

def backtrack(r):

if r == n:

res.append(["".join(row) for row in board])

return

for c in range(n):

if c in col or (r + c) in posDiag or (r - c) in negDiag:

continue

col.add(c)

posDiag.add(r + c)

negDiag.add(r - c)

board[r][c] = "Q"

backtrack(r + 1)

col.remove(c)

posDiag.remove(r + c)

negDiag.remove(r - c)

board[r][c] = "."

col.add(first\_queen\_col)

posDiag.add(0 + first\_queen\_col)

negDiag.add(0 - first\_queen\_col)

board[0][first\_queen\_col] = "Q"

backtrack(1) # Start with the second row

return res

if \_\_name\_\_ == "\_\_main\_\_":

n = 8

first\_queen\_col = 1

board = solveNQueens(n, first\_queen\_col)[0]

for row in board:

print(" ".join(row))

1. **Write a program for analysis of quick sort by using deterministic and randomized variant.**

import random

import timeit

def deterministic\_partition(arr, low, high):

pivot = arr[high]

i = low - 1

for j in range(low, high):

if arr[j] < pivot:

i += 1

arr[i], arr[j] = arr[j], arr[i]

arr[i + 1], arr[high] = arr[high], arr[i + 1]

return i + 1

def randomized\_partition(arr, low, high):

pivot\_index = random.randint(low, high)

arr[pivot\_index], arr[high] = arr[high], arr[pivot\_index]

return deterministic\_partition(arr, low, high)

def quick\_sort(arr, low, high, pivot\_selector):

if low < high:

pivot\_index = pivot\_selector(arr, low, high)

quick\_sort(arr, low, pivot\_index - 1, pivot\_selector)

quick\_sort(arr, pivot\_index + 1, high, pivot\_selector)

if \_\_name\_\_ == "\_\_main\_\_":

arr\_sizes = [100, 1000, 10000, 100000]

for size in arr\_sizes:

arr = [random.randint(1, 1000) for \_ in range(size)]

arr.sort(reverse=True)

deterministic\_time = timeit.timeit("quick\_sort(arr.copy(), 0, len(arr) - 1, deterministic\_partition)",

globals=globals(),

number=10)

randomized\_time = timeit.timeit("quick\_sort(arr.copy(), 0, len(arr) - 1, randomized\_partition)",

globals=globals(),

number=10)

print(f"Array size: {size}")

print(f"Deterministic Quick Sort time: {deterministic\_time:.6f} seconds")

print(f"Randomized Quick Sort time: {randomized\_time:.6f} seconds")

print("-" \* 40)

**ML**

1. **Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks**

#Importing the required libraries

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

df = pd.read\_csv("uber.csv")

df.head()

df.info()

df.columns

df = df.drop(['Unnamed: 0', 'key'], axis= 1)

df.shape

df.dtypes

df.pickup\_datetime = pd.to\_datetime(df.pickup\_datetime)

df.dtypes

df.isnull().sum()

df['dropoff\_latitude'].fillna(value=df['dropoff\_latitude'].mean(),inplace = True)

df['dropoff\_longitude'].fillna(value=df['dropoff\_longitude'].median(),inplace = True)

df.isnull().sum()

df= df.assign(hour = df.pickup\_datetime.dt.hour,

day= df.pickup\_datetime.dt.day,

month = df.pickup\_datetime.dt.month,

year = df.pickup\_datetime.dt.year,

dayofweek = df.pickup\_datetime.dt.dayofweek)

df.head()

from math import \*

# function to calculate the travel distance from the longitudes and latitudes

def distance\_transform(longitude1, latitude1, longitude2, latitude2):

travel\_dist = []

for pos in range(len(longitude1)):

long1,lati1,long2,lati2 = map(radians,[longitude1[pos],latitude1[pos],longitude2[pos],latitude2[pos]])

dist\_long = long2 - long1

dist\_lati = lati2 - lati1

a = sin(dist\_lati/2)\*\*2 + cos(lati1) \* cos(lati2) \* sin(dist\_long/2)\*\*2

c = 2 \* asin(sqrt(a))\*6371

travel\_dist.append(c)

return travel\_dist

df['dist\_travel\_km'] = distance\_transform(df['pickup\_longitude'].to\_numpy(),

df['pickup\_latitude'].to\_numpy(),

df['dropoff\_longitude'].to\_numpy(),

df['dropoff\_latitude'].to\_numpy()

)

df.head()

df = df.drop('pickup\_datetime',axis=1)

df.head()

df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))

def remove\_outlier(df1 , col):

Q1 = df1[col].quantile(0.25)

Q3 = df1[col].quantile(0.75)

IQR = Q3 - Q1

lower\_whisker = Q1-1.5\*IQR

upper\_whisker = Q3+1.5\*IQR

df[col] = np.clip(df1[col] , lower\_whisker , upper\_whisker)

return df1

def treat\_outliers\_all(df1 , col\_list):

for c in col\_list:

df1 = remove\_outlier(df , c)

return df1

df = treat\_outliers\_all(df , df.iloc[: , 0::])

df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))

df= df.loc[(df.dist\_travel\_km >= 1) | (df.dist\_travel\_km <= 130)]

print("Remaining observastions in the dataset:", df.shape)

incorrect\_coordinates = df.loc[(df.pickup\_latitude > 90) |(df.pickup\_latitude < -90) |

(df.dropoff\_latitude > 90) |(df.dropoff\_latitude < -90) |

(df.pickup\_longitude > 180) |(df.pickup\_longitude < -180) |

(df.dropoff\_longitude > 90) |(df.dropoff\_longitude < -90)

]

df.drop(incorrect\_coordinates, inplace = True, errors = 'ignore')

df.head()

df.isnull().sum()

sns.heatmap(df.isnull())

corr = df.corr()

corr

fig,axis = plt.subplots(figsize = (10,6))

sns.heatmap(df.corr(),annot = True)

x = df[['pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude','passenger\_count','hour','day','month','year','dayofweek','dist\_travel\_km']]

y = df['fare\_amount']

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size = 0.33)

from sklearn.linear\_model import LinearRegression

regression = LinearRegression()

regression.fit(X\_train,y\_train)

regression.intercept\_

regression.coef\_

prediction = regression.predict(X\_test)

print(prediction)

y\_test

from sklearn.metrics import r2\_score

r2\_score(y\_test,prediction)

from sklearn.metrics import mean\_squared\_error

MSE = mean\_squared\_error(y\_test,prediction)

MSE

RMSE = np.sqrt(MSE)

RMSE

from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n\_estimators=100) #Here n\_estimators means number of trees you want to build before making the prediction

rf.fit(X\_train,y\_train)

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

y\_pred

R2\_Random = r2\_score(y\_test,y\_pred)

R2\_Random

MSE\_Random = mean\_squared\_error(y\_test,y\_pred)

MSE\_Random

RMSE\_Random = np.sqrt(MSE\_Random)

RMSE\_Random

**2.**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn import metrics

df=pd.read\_csv('emails.csv')

df.head()

df.columns

df.isnull().sum()

df.dropna(inplace = True)

df.drop(['Email No.'],axis=1,inplace=True)

X = df.drop(['Prediction'],axis = 1)

y = df['Prediction']

from sklearn.preprocessing import scale

X = scale(X)

*# split into train and test*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 42)

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=7)

knn.fit(X\_train, y\_train)

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

print("Prediction",y\_pred)

print("KNN accuracy = ",metrics.accuracy\_score(y\_test,y\_pred))

print("Confusion matrix",metrics.confusion\_matrix(y\_test,y\_pred))

*# cost C = 1*

model = SVC(C = 1)

*# fit*

model.fit(X\_train, y\_train)

*# predict*

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

metrics.confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)

print("SVM accuracy = ",metrics.accuracy\_score(y\_test,y\_pred))

**3.**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt #Importing the libraries

df = pd.read\_csv("Churn\_Modelling.csv")

df.head()

df.shape

df.describe()

df.isnull()

df.isnull().sum()

df.info()

df.dtypes

df.columns

df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unnecessary columns

df.head()

def visualization(x, y, xlabel):

plt.figure(figsize=(10,5))

plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not\_exit'])

plt.xlabel(xlabel,fontsize=20)

plt.ylabel("No. of customers", fontsize=20)

plt.legend()

df\_churn\_exited = df[df['Exited']==1]['Tenure']

df\_churn\_not\_exited = df[df['Exited']==0]['Tenure']

visualization(df\_churn\_exited, df\_churn\_not\_exited, "Tenure")

df\_churn\_exited2 = df[df['Exited']==1]['Age']

df\_churn\_not\_exited2 = df[df['Exited']==0]['Age']

visualization(df\_churn\_exited2, df\_churn\_not\_exited2, "Age")

X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary']]

states = pd.get\_dummies(df['Geography'],drop\_first = True)

gender = pd.get\_dummies(df['Gender'],drop\_first = True)

df = pd.concat([df,gender,states], axis = 1)

df.head()

X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','Male','Germany','Spain']]

y = df['Exited']

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size = 0.30)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

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

X\_train

X\_test

import keras #Keras is the wrapper on the top of tenserflow

#Can use Tenserflow as well but won't be able to understand the errors initially.

from keras.models import Sequential #To create sequential neural network

from keras.layers import Dense #To create hidden layers

classifier = Sequential()

#To add the layers

#Dense helps to contruct the neurons

#Input Dimension means we have 11 features

# Units is to create the hidden layers

#Uniform helps to distribute the weight uniformly

classifier.add(Dense(activation = "relu",input\_dim = 11,units = 6,kernel\_initializer = "uniform"))

classifier.add(Dense(activation = "relu",units = 6,kernel\_initializer = "uniform")) #Adding second hidden layers

classifier.add(Dense(activation = "sigmoid",units = 1,kernel\_initializer = "uniform"))

classifier.compile(optimizer="adam",loss = 'binary\_crossentropy',metrics = ['accuracy'])

classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer and 1 neuron in last

classifier.fit(X\_train,y\_train,batch\_size=10,epochs=50) #Fitting the ANN to training

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

y\_pred = (y\_pred > 0.5) #Predicting the result

from sklearn.metrics import confusion\_matrix,accuracy\_score,classification\_report

cm = confusion\_matrix(y\_test,y\_pred)

cm

accuracy = accuracy\_score(y\_test,y\_pred)

accuracy

plt.figure(figsize = (10,7))

sns.heatmap(cm,annot = True)

plt.xlabel('Predicted')

plt.ylabel('Truth')

print(classification\_report(y\_test,y\_pred))

**4.**

import numpy as np

import matplotlib.pyplot as plt

def f(x):

return (x+3)\*\*2

def df(x):

return 2\*x + 6

def gradient\_descent(initial\_x, learning\_rate, num\_iterations):

x = initial\_x

x\_history = [x]

for i in range(num\_iterations):

gradient = df(x)

x = x - learning\_rate \* gradient

x\_history.append(x)

return x, x\_history

initial\_x = 2

learning\_rate = 0.1

num\_iterations = 50

x, x\_history = gradient\_descent(initial\_x, learning\_rate, num\_iterations)

print("Local minimum: {:.2f}".format(x))

#Create a range of x values to plot

x\_vals = np.linspace(-1, 5, 100)

#Plot the function f(x)

plt.plot(x\_vals, f(x\_vals))

# Plot the values of x at each iteration

plt.plot(x\_history, f(np.array(x\_history)), 'rx')

#Label the axes and add a title

plt.xlabel('x')

plt.ylabel('f(x)')

plt.title('Gradient Descent')

#Show the plot

plt.show()

**5.**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans, k\_means

from sklearn.decomposition import PCA

df = pd.read\_csv("sales\_data\_sample.csv")

df.head()

df.shape

df.describe()

df.info()

df.isnull().sum()

df.dtypes

df\_drop = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATUS','POSTALCODE', 'CITY', 'TERRITORY', 'PHONE', 'STATE', 'CONTACTFIRSTNAME', 'CONTACTLASTNAME', 'CUSTOMERNAME', 'ORDERNUMBER']df = df.drop(df\_drop, axis=1)

df.isnull().sum()

df.dtypes

df['COUNTRY'].unique()

df['PRODUCTLINE'].unique()

df['DEALSIZE'].unique()

productline = pd.get\_dummies(df['PRODUCTLINE']) *#Converting the categorical columns.* Dealsize = pd.get\_dummies(df['DEALSIZE'])

df = pd.concat([df,productline,Dealsize], axis = 1)

df\_drop = ['COUNTRY','PRODUCTLINE','DEALSIZE'] *#Dropping Country too as there are alot of countries.* df = df.drop(df\_drop, axis=1)

df['PRODUCTCODE'] = pd.Categorical(df['PRODUCTCODE']).codes

df.drop('ORDERDATE', axis=1, inplace=True)

df.dtypes

distortions = []

K = range(1,10)

for k in K: kmeanModel = KMeans(n\_clusters=k)

kmeanModel.fit(df)

distortions.append(kmeanModel.inertia\_)

plt.figure(figsize=(16,8))

plt.plot(K, distortions, 'bx-')

plt.xlabel('k')

plt.ylabel('Distortion')

plt.title('The Elbow Method showing the optimal k')

plt.show()

X\_train = df.values

X\_train.shape

model = KMeans(n\_clusters=3,random\_state=2)

model = model.fit(X\_train)

predictions = model.predict(X\_train)

unique,counts = np.unique(predictions,return\_counts=True)

counts = counts.reshape(1,3)

counts\_df = pd.DataFrame(counts,columns=['Cluster1','Cluster2','Cluster3'])

counts\_df.head()

pca = PCA(n\_components=2)

reduced\_X = pd.DataFrame(pca.fit\_transform(X\_train),columns=['PCA1','PCA2'])

reduced\_X.head()

plt.figure(figsize=(14,10))

plt.scatter(reduced\_X['PCA1'],reduced\_X['PCA2'])

model.cluster\_centers\_

reduced\_centers = pca.transform(model.cluster\_centers\_)

reduced\_centers

plt.figure(figsize=(14,10))

plt.scatter(reduced\_X['PCA1'],reduced\_X['PCA2']) plt.scatter(reduced\_centers[:,0],reduced\_centers[:,1],color='black',marker='x',s=300)

reduced\_X['Clusters'] = predictions

reduced\_X.head()

plt.figure(figsize=(14,10))

plt.scatter(reduced\_X[reduced\_X['Clusters'] == 0].loc[:,'PCA1'],reduced\_X[reduced\_X['Clusters'] == 0].loc[:,'PCA2'],color='slateblue')

plt.scatter(reduced\_X[reduced\_X['Clusters'] == 1].loc[:,'PCA1'],reduced\_X[reduced\_X['Clusters'] == 1].loc[:,'PCA2'],color='springgreen')

plt.scatter(reduced\_X[reduced\_X['Clusters'] == 2].loc[:,'PCA1'],reduced\_X[reduced\_X['Clusters'] == 2].loc[:,'PCA2'],color='indigo')

plt.scatter(reduced\_centers[:,0],reduced\_centers[:,1],color='black',marker='x',s=300)

**BT**

**3.**

// SPDX-License-Identifier: MIT

//https://betterprogramming.pub/developing-a-smart-contract-by-using-re mix-ide-81ff6f44ba2f

pragma solidity >=0.7.0 <0.9.0;

contract SimpleBank

{

struct client\_account{

int client\_id;

address client\_address;

uint client\_balance\_in\_ether;

}

client\_account[] clients;

int clientCounter;

address payable manager;

modifier onlyManager() {

require(msg.sender == manager, "Only manager can call this!");

\_;

}

modifier onlyClients() {

bool isclient = false;

for(uint i=0;i<clients.length;i++){

if(clients[i].client\_address == msg.sender){

isclient = true;

break;

}

}

require(isclient, "Only clients can call this!");

\_;

}

constructor() {

clientCounter = 0;

}

receive() external payable { }

function setManager(address managerAddress) public returns(string memory){

manager = payable(managerAddress);

return "";

}

function joinAsClient() public payable returns(string memory){

clients.push(client\_account(clientCounter++, msg.sender, address(msg.sender).balance));

return "";

}

function deposit() public payable onlyClients{

payable(address(this)).transfer(msg.value);

}

function withdraw(uint amount) public payable onlyClients{

payable(msg.sender).transfer(amount \* 1 ether);

}

function sendInterest() public payable onlyManager{

for(uint i=0;i<clients.length;i++){

address initialAddress = clients[i].client\_address;

payable(initialAddress).transfer(1 ether);

}

}

function getContractBalance() public view returns(uint){

return address(this).balance;

}

}

**4.**

// SPDX-License-Identifier: MIT

//https://betterprogramming.pub/developing-a-smart-contract-by-using-remix-ide-81ff6f44ba2f

pragma solidity ^0.5.0;

contract Crud {

struct User {

uint id;

string name;

}

User[] public users;

uint public nextId = 0;

function Create(string memory name) public {

users.push(User(nextId, name));

nextId++;

}

function Read(uint id) view public returns(uint, string memory) {

for(uint i=0; i<users.length; i++) {

if(users[i].id == id) {

return(users[i].id, users[i].name);

}

}

}

function Update(uint id, string memory name) public {

for(uint i=0; i<users.length; i++) {

if(users[i].id == id) {

users[i].name =name;

}

}

}

function Delete(uint id) public {

delete users[id];

}

function find(uint id) view internal returns(uint) {

for(uint i=0; i< users.length; i++) {

if(users[i].id == id) {

return i;

}

}

// if user does not exist then revert back

revert("User does not exist");

}

}