```
//fractional knapsack problem using a greedy method
def fractional_knapsack(items, max_weight):
 # Calculate the value-to-weight ratio for each item
 for item in items:
 item["value_per_weight"] = item["value"] / item["weight"]

 # Sort the items by value-to-weight ratio in descending order
 items.sort(key=lambda x: x["value_per_weight"], reverse=True)

 total_value = 0.0
 knapsack = []
 for item in items:
 if max_weight >= item["weight"]:
 # Add the whole item to the knapsack
 knapsack.append(item)
 total_value += item["value"]
 max_weight -= item["weight"]
 else:
 # Add a fraction of the item to the knapsack
 fraction = max_weight / item["weight"]
 knapsack.append({"name": item["name"], "weight": max_weight, "value":
item["value"] *
fraction})
 total_value += item["value"] * fraction
 break
 return knapsack, total_value
# Example usage
if __name__ == "__main__":
 items = [
 {"name": "item1", "weight": 2, "value": 10},
 {"name": "item2", "weight": 3, "value": 5},
 {"name": "item3", "weight": 5, "value": 15},
 {"name": "item4", "weight": 7, "value": 7},
 {"name": "item5", "weight": 1, "value": 6},
 ]
 max_weight = 10
 knapsack_items, total_value = fractional_knapsack(items, max_weight)
 print("Items in the knapsack:")
 for item in knapsack_items:
 print(f"{item['name']}, Weight: {item['weight']}, Value: {item['value']}")
 print(f"Total value in the knapsack: {total_value}")
//program to solve Sum of subset problem
def is_subset_sum_backtracking(nums, target_sum):
 def backtrack(index, current_sum):
 if current_sum == target_sum:
 return True
 if current_sum > target_sum or index == len(nums):
 return False

 # Include the current number in the subset
 if backtrack(index + 1, current_sum + nums[index]):
 return True

 # Exclude the current number from the subset
 if backtrack(index + 1, current_sum):
 return True

 return False
 return backtrack(0, 0)
# Example usage
nums = [1, 3, 4, 5, 2]
target_sum = 8
result = is_subset_sum_backtracking(nums, target_sum)
print("Backtracking: Subset with the given sum exists" if result else "Backtracking: No
such subset")
//job sequencing with deadlines using greedy method
def job_sequencing_with_deadlines(jobs):
 # Sort the jobs in decreasing order of their profits
 jobs.sort(key=lambda x: x[2], reverse=True)
 max_deadline = max(job[1] for job in jobs)
 schedule = [0] * max_deadline # Initialize a schedule array
 total_profit = 0
 for job in jobs:
 deadline, profit = job[1], job[2]
 for i in range(deadline - 1, -1, -1):
 if schedule[i] == 0:
 schedule[i] = job[0]
 total_profit += profit
 break
 return schedule, total_profit
# Example usage
if __name__ == "__main__":
 # Each job is represented as a tuple (job_id, deadline, profit)
 jobs = [
 (1, 2, 100),
 (2, 1, 19),
 (3, 2, 27),
 (4, 1, 25),
 (5, 3, 15),
 ]
 schedule, total_profit = job_sequencing_with_deadlines(jobs)
 print("Job schedule:")
 for job_id in schedule:
 if job_id != 0:
 print(f"Job {job_id}")
 print(f"Total profit: {total_profit}")
//graph coloring method by greedy and backtracking method
def greedy_graph_coloring(graph):
 color_map = {} # A dictionary to store the assigned colors for each vertex
 for vertex in graph:
 # Initialize the set of used colors for the current vertex's neighbors
 used_colors = set()
 for neighbor in graph[vertex]:
```

```
 if neighbor in color_map:
 used_colors.add(color_map[neighbor])
 # Find the smallest available color for the current vertex
 for color in range(len(graph)):
 if color not in used_colors:
 color_map[vertex] = color
 break
 return color_map
# Example usage
if __name__ == "__main__":
 graph = {
 'A': ['B', 'C'],
 'B': ['A', 'C', 'D'],
 'C': ['A', 'B', 'D'],
 'D': ['B', 'C']
 }
 color_map = greedy_graph_coloring(graph)
 for vertex, color in color_map.items():
 print(f"Vertex {vertex} is colored with color {color}")
```

```
//DeploythisasSmartContractonEtherumandObservethetransactionfeesand
GasValue.
//SPDX-License-Identifier:MIT
pragmasolidity^0.8.18;
contractBankContract{
structclient_account{
intclient_id;
addressclient_address;
uintclient_balance_in_ether;
}
client_account[]clients;
intclientCounter;
addresspayablemanager;
mapping(address=>uint)publicinterestDate;
modifieronlyManager(){
require(msg.sender==manager,"onlymanagercancallthis!");
_;
}
modifieronlyClients()
{
boolisclient=false;
for(uinti=0;i<clients.length;i++)
{
if(clients[i].client_address==msg.sender)
{
isclient=true;
break;
}
}
require(isclient,"Onlyclientscancallthis!");
_;
}
constructor()payable
{
clientCounter=0;
}
receive()externalpayable{}
functionsetManager(addressmanagerAddress)publicreturns(stringmemory)
{
manager=payable(managerAddress);
return"";
}
functionjoinAsClient()publicpayablereturns(stringmemory)
{
interestDate[msg.sender]=block.timestamp;
clients.push(client_account(clientCounter++,
msg.sender,address(msg.sender).balance));
return"";
}
functiondeposit()publicpayableonlyClients
{
payable(address(this)).transfer(msg.value);
}
functionwithdraw(uintamount)publicpayableonlyClients{
payable(msg.sender).transfer(amount*1ether);
}
functionsendInterest()publicpayableonlyManager{
for(uinti=0;i<clients.length;i++)
{
addressinitialAddress=clients[i].client_address;
uintlastInterestDate=interestDate[initialAddress];
if(block.timestamp<lastInterestDate+10seconds)
{
revert("It'sjustbeenlessthan10seconds!");
}
payable(initialAddress).transfer(1ether);
interestDate[initialAddress]=block.timestamp;
}
}
functiongetContractBalance()publicviewreturns(uint)
{
returnaddress(this).balance;
}
}
//Solidityprogramtodemonstratehowtowriteasmartcontract
//SPDX-License-Identifier:MIT
pragmasolidity^0.8.18;
```

```
contractStorage
{
uintpublicsetData;
functionset(uintx)public
{setData=x;}
functionget(
)publicviewreturns(uint){
returnsetData;
}
}
```

ExperimentNo.04

```
//CodeusingFallback
//SPDX-License-Identifier:MIT
pragmasolidity^0.4.0;
//Creatingacontract
contractfback
{
//Declaringthestatevariable
uintx;
//Mappingofaddressestotheirbalances
mapping(address=>uint)balance;
//Creatingaconstructor
constructor()public
{//Setxtodefault
//valueof10
x=10;
}
//Creatingafunction
functionSetX(uint_x)publicreturns(bool)
{
//Setxtothe
//valuesent
x=_x;
returntrue;
}
//ThisfallbackfunctionwillkeepalltheEther
function()publicpayable
{
balance[msg.sender]+=msg.value;
}
}
//Creatingthesendercontract
contractSender
{functiontransfer()publicpayable
{//AddressofFbackcontract
address_receiver=
0xbcD310867F1b74142c2f5776404b6bd97165FA56;
//Transfers100EthtoabovecontractS
_receiver.transfer(100);
}
}
```

ExperimentNo.03

```
//SPDX-License-Identifier:MIT
pragmasolidity^0.8.18;
contractMyContract{
addressprivateowner;
constructor(){
owner=msg.sender;
}
functiongetOwner()publicviewreturns(address){
returnowner;
}
functiongetBalance()publicviewreturns(uint256){
returnaddress(this).balance;
}
functiondeposit()externalpayable{
require(msg.value==2ether,"Pleasesendtwoethers");
}
functionwithdraw()external{
require(msg.sender==owner,"Onlytheownercanwithdraw");
payable(msg.sender).transfer(address(this).balance);
}
}
```

Ml 1

```
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 sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
df = pd.read_csv("D:/uber.csv")
df.head()
df.isnull().sum
df = df.drop(['Unnamed: 0', 'key'], axis=1)
df.dropna(axis=0, inplace=True)
df.isnull().sum()
df.info()
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
df.describe()
sns.boxplot(df['fare_amount'])
plt.show()
a1 = df['fare_amount'].quantile(0.25)
a2 = df['fare_amount'].quantile(0.75)
iqr = a2 - a1
lower_limit = (a1 - 1.5 * iqr)
upper_limit = (a1 + 1.5 * iqr)
df2 = df[(df['fare_amount'] > lower_limit) & (df['fare_amount'] < upper_limit)]
```

```
sns.boxplot(df2['fare_amount'])
plt.show()
def distance(lon1, lon2, lat1, lat2):
lon1, lon2, lat1, lat2 = map(np.radians, [lon1, lon2, lat1, lat2])
dlon = lon2 - lon1
dlat = lat2 - lat1
R = 6371
a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2
c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
distance = R * c
return distance
df["distance"] = distance(df["pickup_longitude"], df["dropoff_longitude"],
df["pickup_latitude"],
df["dropoff_latitude"])
sns.scatterplot(x=df["distance"], y = df['fare_amount'])
plt.show()
df.drop(df[df['distance'] > 60].index, inplace = True)
df.drop(df[df['distance'] == 0].index, inplace = True)
df.drop(df[df['fare_amount'] == 0].index, inplace = True)
df.drop(df[df['fare_amount'] < 0].index, inplace = True)
df.drop(df[(df['fare_amount'] > 10) & (df['distance'] < 1)].index, inplace = True)
df.drop(df[(df['fare_amount'] < 10) & (df['distance'] > 10)].index, inplace = True)
sns.scatterplot(x=df["distance"], y = df['fare_amount'])
plt.show()
Ml 2
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('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)
# ##KNN classifier
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))
# ## SVM classifier
# 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))
Ml 3
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('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('diabetes.csv')
df.columns
# Check for null values. If present remove null values from the dataset
df.isnull().sum()
# Outcome is the label/target, other columns are features
X = df.drop('Outcome',axis = 1)
y = df['Outcome']
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("Confusion matrix: ")
cs = metrics.confusion_matrix(y_test,y_pred)
print(cs)
print("Acccuracy ",metrics.accuracy_score(y_test,y_pred))
# Classification error rate: proportion of instances misclassified over the whole set of instances.
# Error rate is calculated as the total number of two incorrect predictions (FN + FP) divided by the
total number of a dataset (examples in the dataset.
total_misclassified = cs[0,1] + cs[1,0]
print(total_misclassified)
total_examples = cs[0,0]+cs[0,1]+cs[1,0]+cs[1,1]
print(total_examples)
print("Error rate",total_misclassified/total_examples)
```

```
print("Error rate ",1-metrics.accuracy_score(y_test,y_pred))
print("Precision score",metrics.precision_score(y_test,y_pred))
print("Recall score ",metrics.recall_score(y_test,y_pred))
print("Classification report ",metrics.classification_report(y_test,y_pred))
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Importing the required libraries.
from sklearn.cluster import KMeans, k_means #For clustering
from sklearn.decomposition import PCA #Linear Dimensionality reduction.
df = pd.read_csv("sales_data_sample.csv") #Loading the dataset.
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) #Dropping the categorical uneccessary columns along
with columns having null values. Can't fill the null values are there are alot of
null values.
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 #Converting the
datatype.
df.drop('ORDERDATE', axis=1, inplace=True) #Dropping the Orderdate as Month is
already included.
df.dtypes #All the datatypes are converted into numeric
distortions = [] # Within Cluster Sum of Squares from the centroid
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(df)
    distortions.append(kmeanModel.inertia_)   #Appeding the intertia to the
Distortions
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 #Returns a numpy array.
X_train.shape
model = KMeans(n_clusters=3,random_state=2) #Number of cluster = 3
model = model.fit(X_train) #Fitting the values to create a model.
predictions = model.predict(X_train) #Predicting the cluster values (0,1,or 2)
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) #Converting all the features into 2 columns to make it
easy to visualize using Principal COmponent Analysis.
reduced_X = pd.DataFrame(pca.fit_transform(X_train),columns=['PCA1','PCA2'])
#Creating a DataFrame.
reduced_X.head()
#Plotting the normal Scatter Plot
plt.figure(figsize=(14,10))
plt.scatter(reduced_X['PCA1'],reduced_X['PCA2'])
model.cluster_centers_ #Finding the centriods. (3 Centriods in total. Each Array
contains a centroids for particular feature )
reduced_centers = pca.transform(model.cluster_centers_) #Transforming the
centroids into 3 in x and y coordinates
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) #Plotting the centriods
```

```
reduced_X['Clusters'] = predictions #Adding the Clusters to the reduced
dataframe.
reduced_X.head()
#Plotting the clusters
plt.figure(figsize=(14,10))
#                   taking the cluster number and first column           taking
the same cluster number and second column      Assigning the color
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)
```