Group A: Design and Analysis of Algorithms

A1. Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.

Iterative Program

```
# Program to display the Fibonacci sequence up to n-th term
nterms = int(input("Enter number of terms "))
# first two terms
n1, n2 = 0, 1
count = 0
# check if the number of terms is valid
if nterms <= 0:
  print("Please enter a positive integer")
# if there is only one term, return n1
elif nterms == 1:
  print("Fibonacci sequence upto", nterms,":")
  print(n1)
# generate fibonacci sequence
else:
print("Fibonacci sequence:")
  while count < nterms:
      print(n1)
      nth = n1 + n2
      # update values
      n1 = n2
      n2 = nth
      count += 1
```

Output

Enter number of terms 4 Fibonacci sequence:

0

1

1

2

Recursive Program

```
def fibonacci(n):
    if(n <= 1):
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))
n = int(input("Enter number of terms:"))
print("Fibonacci sequence:")
for i in range(n):
    print(fibonacci(i))
```

Output

```
Enter number of terms:4 Fibonacci sequence:
0
1
1
2
```

A2. Write a program to implement Huffman Encoding using a greedy strategy.

Implementation

```
def printNodes(node, val=''):

        newVal = val + str(node.huff)

        if(node.left):

                printNodes(node.left, newVal)

        if(node.right):

                printNodes(node.right, newVal)

        if(not node.left and not node.right):

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

- # characters for huffman tree
- chars = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
- # frequency of characters
- freq = [4, 7, 12, 14, 17, 43, 54]
- # list containing unused nodes
- nodes = []
- # converting characters and frequencies into huffman tree nodes
- for x in range(len(chars)):
- nodes.append(node(freq[x], chars[x]))
- while len(nodes) > 1:
- # sort all the nodes in ascending order based on their frequency
- nodes = sorted(nodes, key=lambda x: x.freq)
- # pick 2 smallest nodes
- left = nodes[0]
- right = nodes[1]
- # assign directional value to these nodes
- left.huff = 0
- right.huff = 1
- # combine the 2 smallest nodes to create new node as their parent
- newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)
- # remove the 2 nodes and add their parent as new node among others
- nodes.remove(left)

- nodes.remove(right)

  nodes.append(newNode)

- # Huffman Tree is ready!

- printNodes(nodes[0])

Output

a -> 0000 b -> 0001 c -> 001

d -> 010

e -> 011

f -> 10

g -> 11

A3. Write a program to solve a fractional Knapsack problem using a greedy method.

Implementation

```
def fractional_knapsack(value, weight, capacity):
    # index = [0, 1, 2, ..., n - 1] for n items
    index = list(range(len(value)))
    # contains ratios of values to weight
    ratio = [v/w for v, w in zip(value, weight)]
    # index is sorted according to value-to-weight ratio in decreasing order
    index.sort(key=lambda i: ratio[i], reverse=True)
    max_value = 0
    fractions = [0]*len(value)
for i in index:
        if weight[i] <= capacity:
            fractions[i] = 1
            max_value += value[i]
            capacity -= weight[i]
        else:
            fractions[i] = capacity/weight[i]
            max_value += value[i]*capacity/weight[i]
            break
    return max_value, fractions
n = int(input('Enter number of items: '))
value = input('Enter the values of the {} item(s) in order: '.format(n)).split()
value = [int(v) for v in value]
weight = input('Enter the positive weights of the {} item(s) in order: '.format(n)).split()
weight = [int(w) for w in weight]
capacity = int(input('Enter maximum weight: '))

max_value, fractions = fractional_knapsack(value, weight, capacity)
print('The maximum value of items that can be carried:', max_value)
print('The fractions in which the items should be taken:', fractions)
```

Output

Enter number of items: 3

Enter the values of the 3 item(s) in order: 24 15 25

Enter the positive weights of the 3 item(s) in order: 15 10 18 Enter maximum weight: 20

The maximum value of items that can be carried: 31.5

The fractions in which the items should be taken: [1, 0.5, 0]

A4. Write a Program for Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final 8-queen's matrix.

```
/* C++ program to solve N Queen Problem using
backtracking */

#include <bits/stdc++.h>
#define N 4
using namespace std;

/* A utility function to print solution */
void printSolution(int board[N][N])
{
for (int i = 0; i < N; i++) {
		for (int j = 0; j < N; j++)
			cout << " " << board[i][j] << " ";
		printf("\n");
	}
}

/* A utility function to check if a queen can
be placed on board[row][col]. Note that this
function is called when "col" queens are
already placed in columns from 0 to col -1.
So we need to check only left side for
attacking queens */
bool isSafe(int board[N][N], int row, int col)
{
	int i, j;

	/* Check this row on left side */
	for (i = 0; i < col; i++)
		if (board[row][i])
			return false;
```

```
    /* Check upper diagonal on left side */
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;
    /* Check lower diagonal on left side */
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;

    return true;
}

/* A recursive utility function to solve N
Queen problem */
bool solveNQUtil(int board[N][N], int col)
{
    /* base case: If all queens are placed
    then return true */
    if (col >= N)
        return true;

    /* Consider this column and try placing
    this queen in all rows one by one */
    for (int i = 0; i < N; i++) {
    /* Check if the queen can be placed on
        board[i][col] */
        if (isSafe(board, i, col)) {
            /* Place this queen in board[i][col] */
            board[i][col] = 1;

            /* recur to place rest of the queens */
            if (solveNQUtil(board, col + 1))
                return true;

            /* If placing queen in board[i][col]
```

```
                doesn't lead to a solution, then
                remove queen from board[i][col] */
                board[i][col] = 0; // BACKTRACK
            }
        }

        /* If the queen cannot be placed in any row in
            this column col then return false */
        return false;
}

/* This function solves the N Queen problem using
Backtracking. It mainly uses solveNQUtil() to
solve the problem. It returns false if queens
cannot be placed, otherwise, return true and
prints placement of queens in the form of 1s.
Please note that there may be more than one
solutions, this function prints one of the
feasible solutions.*/
bool solveNQ()
{
        int board[N][N] = { { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 } };

        if (solveNQUtil(board, 0) == false) {
            cout << "Solution does not exist";
            return false;
        }

        printSolution(board);
        return true;
}
```

```
// driver program to test above function
int main()
{
    solveNQ();
    return 0;
}
```

Output

- 0 0 1 0
- 1 0 0 0
- 0 0 0 1
- 0 1 0 0

Group B: Machine Learning:

Assignment No. 1

```
import os
import math
import scipy
import numpy as np
import pandas as pd
import seaborn as sns
import datetime as dt
import geopy.distance
from tqdm import tqdm
from IPython.display import display

from statsmodels.formula import api
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.decomposition import PCA
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10,6]
import warnings
warnings.filterwarnings('ignore')
```

```
#Importing the dataset

df = pd.read_csv('../input/uber-fares-dataset/uber.csv')

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

target = 'fare_amount'
features = [i for i in df.columns if i not in [target]]

print('\n\033[1mInference:\033[0m The Datset consists of {} features & {} samples.'.format(df.shape[1], df.sha
```

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

Inference: The Datset consists of 7 features & 200000 samples.

```
#Check for empty elements

nvc = pd.DataFrame(df.isnull().sum().sort_values(), columns=['Total Null Values'])
nvc['Percentage'] = round(nvc['Total Null Values']/df.shape[0],3)*100
print(nvc)
df.dropna(inplace=True)
```

```
                   Total Null Values  Percentage
fare_amount                        0         0.0
pickup_datetime                    0         0.0
pickup_longitude                   0         0.0
pickup_latitude                    0         0.0
passenger_count                    0         0.0
dropoff_longitude                  1         0.0
dropoff_latitude                   1         0.0
```

```
# Reframing the columns

df = df[(df.pickup_latitude<90) & (df.dropoff_latitude<90) &
        (df.pickup_latitude>-90) & (df.dropoff_latitude>-90) &
        (df.pickup_longitude<180) & (df.dropoff_longitude<180) &
        (df.pickup_longitude>-180) & (df.dropoff_longitude>-180)]

df.pickup_datetime=pd.to_datetime(df.pickup_datetime)

df['year'] = df.pickup_datetime.dt.year
df['month'] = df.pickup_datetime.dt.month
df['weekday'] = df.pickup_datetime.dt.weekday
df['hour'] = df.pickup_datetime.dt.hour

df['Monthly_Quarter'] = df.month.map({1:'Q1',2:'Q1',3:'Q1',4:'Q2',5:'Q2',6:'Q2',7:'Q3',
                                      8:'Q3',9:'Q3',10:'Q4',11:'Q4',12:'Q4'})
df['Hourly_Segments'] = df.hour.map({0:'H1',1:'H1',2:'H1',3:'H1',4:'H2',5:'H2',6:'H2',7:'H2',8:'H3',
                                     9:'H3',10:'H3',11:'H3',12:'H4',13:'H4',14:'H4',15:'H4',16:'H5',
                                     17:'H5',18:'H5',19:'H5',20:'H6',21:'H6',22:'H6',23:'H6'})

df['Distance']=[round(geopy.distance.distance((df.pickup_latitude[i], df.pickup_longitude[i]),(df.dropoff_lati

df.drop(['pickup_datetime','month', 'hour',], axis=1, inplace=True)

original_df = df.copy(deep=True)

df.head()
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	year	weekday	Monthly_Quarter	Hourly_Segment
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1	2015	3	Q2	H
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1	2009	4	Q3	H
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1	2009	0	Q3	H
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3	2009	4	Q2	H
4	16.0	-73.925023	40.744085	-73.973082	40.761247	5	2014	3	Q3	H

```
#Checking the dtypes of all the columns

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199987 entries, 0 to 199999
Data columns (total 11 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   fare_amount        199987 non-null  float64
 1   pickup_longitude   199987 non-null  float64
 2   pickup_latitude    199987 non-null  float64
 3   dropoff_longitude  199987 non-null  float64
 4   dropoff_latitude   199987 non-null  float64
 5   passenger_count    199987 non-null  int64
 6   year               199987 non-null  int64
 7   weekday            199987 non-null  int64
 8   Monthly_Quarter    199987 non-null  object
 9   Hourly_Segments    199987 non-null  object
 10  Distance           199987 non-null  float64
dtypes: float64(6), int64(3), object(2)
memory usage: 22.3+ MB
```

```
#Checking number of unique rows in each feature

df.nunique().sort_values()
```

```
Monthly_Quarter          4
Hourly_Segments          6
year                     7
weekday                  7
passenger_count          8
fare_amount           1244
pickup_longitude     71055
dropoff_longitude    76890
pickup_latitude      83831
dropoff_latitude     90582
Distance            164542
dtype: int64
```

```
#Checking number of unique rows in each feature

nu = df.drop([target], axis=1).nunique().sort_values()
nf = []; cf = []; nnf = 0; ncf = 0; #numerical & categorical features

for i in range(df.drop([target], axis=1).shape[1]):
    if nu.values[i]<=24:cf.append(nu.index[i])
    else: nf.append(nu.index[i])

print('\n\033[1mInference:\033[0m The Datset has {} numerical & {} categorical features.'.format(len(nf),len(c
```

```
#Checking the stats of all the columns

display(df.describe())
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	year	weekday	Distance
count	199987.000000	199987.000000	199987.000000	199987.000000	199987.000000	199987.000000	199987.000000	199987.000000	1.999870e+05
mean	11.359849	-72.501786	39.917937	-72.511608	39.922031	1.684544	2011.742463	3.048383	2.056346e+04
std	9.901868	10.449955	6.130412	10.412192	6.117669	1.385999	1.856438	1.946960	3.796638e+05
min	-52.000000	-93.824668	-74.015515	-75.458979	-74.015750	0.000000	2009.000000	0.000000	0.000000e+00
25%	6.000000	-73.992064	40.734793	-73.991407	40.733823	1.000000	2010.000000	1.000000	1.215530e+03
50%	8.500000	-73.981822	40.752592	-73.980092	40.753042	1.000000	2012.000000	3.000000	2.121280e+03
75%	12.500000	-73.967154	40.767157	-73.963658	40.768000	2.000000	2013.000000	5.000000	3.874255e+03
max	499.000000	40.808425	48.018760	40.831932	45.031598	208.000000	2015.000000	6.000000	8.783594e+06

```
plt.figure(figsize=[15,10])
a=plt.imread('https://raw.githubusercontent.com/Masterx-AI/Project_Uber_Fare_Prediction/main/wm.png')
plt.imshow(a, alpha=0.2)
plt.scatter( (df.pickup_longitude+180)*3,(df.pickup_latitude+215)*1.45555555,alpha=0.3, color='red')
#mdf.plot(kind='scatter',x='pickup_latitude',y='pickup_longitude',alpha=0.1)
plt.show()
```

```
#Let us first analyze the distribution of the target variable

plt.figure(figsize=[8,4])
sns.distplot(df[target], color='g',hist_kws=dict(edgecolor="black", linewidth=2), bins=30)
plt.title('Target Variable Distribution - Median Value of Homes ($1Ms)')
plt.show()
```

```
print('\033[1mVisualising Categorical Features:'.center(100))

n=2
plt.figure(figsize=[15,3*math.ceil(len(cf)/n)])

# for i in range(len(cf)):
#     if df[cf[i]].nunique()<=4:
#         plt.subplot(math.ceil(len(cf)/n),n,i+1)
#         sns.countplot(df[cf[i]])
#     else:
#         plt.subplot(math.ceil(len(cf)/2),2,i)
#         sns.countplot(df[cf[i]])

for i in range(len(cf)):
    if df[cf[i]].nunique()<=12:
        plt.subplot(math.ceil(len(cf)/n),n,i+1)
        sns.countplot(df[cf[i]])
    else:
        plt.subplot(3,1,i-3)
        sns.countplot(df[cf[i]])
        #plt.subplot(4,2,8)
        #sns.countplot(df[cf[i]])

plt.tight_layout()
plt.show()
```

```
#Visualising the numeric features

print('\033[1mNumeric Features Distribution'.center(100))

n=5

plt.figure(figsize=[15,5*math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
    sns.distplot(df[nf[i]],hist_kws=dict(edgecolor="black", linewidth=2), bins=10, color=list(np.random.randir
plt.tight_layout()
plt.show()

plt.figure(figsize=[15,5*math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
    df.boxplot(nf[i])
plt.tight_layout()
plt.show()
```

```
#Removal of any Duplicate rows (if any)

counter = 0
rs,cs = original_df.shape

df.drop_duplicates(inplace=True)
df.drop(['pickup_latitude','pickup_longitude',
         'dropoff_latitude','dropoff_longitude'],axis=1)

if df.shape==(rs,cs):
    print('\n\033[1mInference:\033[0m The dataset doesn\'t have any duplicates')
else:
    print(f'\n\033[1mInference:\033[0m Number of duplicates dropped/fixed ---> {rs-df.shape[0]}')
```

```
df1 = df.copy()
df3 = df1.copy()

ecc = nvc[nvc['Percentage']!=0].index.values
fcc = [i for i in cf if i not in ecc]
#One-Hot Binay Encoding
oh=True
dm=True
for i in fcc:
    #print(i)
    if df3[i].nunique()==2:
        if oh==True: print("\033[1mOne-Hot Encoding on features:\033[0m")
        print(i);oh=False
        df3[i]=pd.get_dummies(df3[i], drop_first=True, prefix=str(i))
    if (df3[i].nunique()>2 and df3[i].nunique()<17):
        if dm==True: print("\n\033[1mDummy Encoding on features:\033[0m")
        print(i);dm=False
        df3 = pd.concat([df3.drop([i], axis=1), pd.DataFrame(pd.get_dummies(df3[i], drop_first=True, prefix=st

df3.shape
```

```
#Removal of outlier:

df1 = df3.copy()

#features1 = [i for i in features if i not in ['CHAS','RAD']]
features1 = nf

for i in features1:
    Q1 = df1[i].quantile(0.25)
    Q3 = df1[i].quantile(0.75)
    IQR = Q3 - Q1
    df1 = df1[df1[i] <= (Q3+(1.5*IQR))]
    df1 = df1[df1[i] >= (Q1-(1.5*IQR))]
    df1 = df1.reset_index(drop=True)
display(df1.head())
print('\n\033[1mInference:\033[0m\nBefore removal of outliers, The dataset had {} samples.'.format(df3.shape[0
print('After removal of outliers, The dataset now has {} samples.'.format(df1.shape[0]))
```

```
#Final Dataset size after performing Preprocessing

df = df1.copy()
df.columns=[i.replace('-','_') for i in df.columns]

plt.title('Final Dataset')
plt.pie([df.shape[0], original_df.shape[0]-df.shape[0]], radius = 1, labels=['Retained','Dropped'], countercl
        autopct='%1.1f%%', pctdistance=0.9, explode=[0,0], shadow=True)
plt.pie([df.shape[0]], labels=['100%'], labeldistance=-0, radius=0.78)
plt.show()

print(f'\n\033[1mInference:\033[0m After the cleanup process, {original_df.shape[0]-df.shape[0]} samples were
while retaining {round(100 - (df.shape[0]*100/(original_df.shape[0])),2)}% of the data.')
```

```
#Splitting the data intro training & testing sets

m=[]
for i in df.columns.values:
    m.append(i.replace(' ','_'))

df.columns = m
X = df.drop([target],axis=1)
Y = df[target]
Train_X, Test_X, Train_Y, Test_Y = train_test_split(X, Y, train_size=0.8, test_size=0.2, random_state=100)
Train_X.reset_index(drop=True,inplace=True)

print('Original set  ---> ',X.shape,Y.shape,'\nTraining set  ---> ',Train_X.shape,Train_Y.shape,'\nTesting set
```

```
#Feature Scaling (Standardization)

std = StandardScaler()

print('\033[1mStandardardization on Training set'.center(100))
Train_X_std = std.fit_transform(Train_X)
Train_X_std = pd.DataFrame(Train_X_std, columns=X.columns)
display(Train_X_std.describe())

print('\n','\033[1mStandardardization on Testing set'.center(100))
Test_X_std = std.transform(Test_X)
Test_X_std = pd.DataFrame(Test_X_std, columns=X.columns)
display(Test_X_std.describe())
```

```
#Checking the correlation

print('\033[1mCorrelation Matrix'.center(100))
plt.figure(figsize=[24,20])
sns.heatmap(df.corr(), annot=True, vmin=-1, vmax=1, center=0) #cmap='BuGn'
plt.show()
```

```
#Testing a Linear Regression model with statsmodels

Train_xy = pd.concat([Train_X,Train_Y.reset_index(drop=True)],axis=1)
a = Train_xy.columns.values

API = api.ols(formula='{} ~ {}'.format(target,' + '.join(i for i in Train_X.columns)), data=Train_xy).fit()
#print(API.conf_int())
#print(API.pvalues)
API.summary()
```

```python
from sklearn.preprocessing import PolynomialFeatures
Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
#Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
#Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)

DROP=[];b=[]

for i in tqdm(range(len(Train_X_std.columns)-1)):
    vif = pd.DataFrame()
    X = Train_X_std.drop(DROP,axis=1)
    vif['Features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    vif.reset_index(drop=True, inplace=True)
    if vif.loc[0][1]>=1.1:
        DROP.append(vif.loc[0][0])
        LR = LinearRegression()
        LR.fit(Train_X_std.drop(DROP,axis=1), Train_Y)

        pred1 = LR.predict(Train_X_std.drop(DROP,axis=1))
        pred2 = LR.predict(Test_X_std.drop(DROP,axis=1))

        Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))
        Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))
```

```python
print('Dropped Features --> ',DROP)
#plt.plot(b)
#plt.show()
#print(API.summary())

# plt.figure(figsize=[20,4])
# plt.subplot(1,3,1)
# sns.heatmap(Trd.loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max())
# plt.title('Train RMSE')
# plt.subplot(1,3,2)
# sns.heatmap(Tsd.loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max()+10)
# plt.title('Test RMSE')
# plt.subplot(1,3,3)
# sns.heatmap((Trd+Tsd).loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max()+25)
# plt.title('Total RMSE')
# plt.show()

plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
#plt.ylim([19.75,20.75])
plt.legend()
plt.grid()
plt.show()
```

```python
    pred2 = LR.predict(Test_X_std.loc[:,rfe.support_])

    Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))
    Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))

# plt.figure(figsize=[20,4])
# plt.subplot(1,3,1)
# sns.heatmap(Trd.loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max())
# plt.title('Train RMSE')
# plt.subplot(1,3,2)
# sns.heatmap(Tsd.loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max()+10)
# plt.title('Test RMSE')
# plt.subplot(1,3,3)
# sns.heatmap((Trd+Tsd).loc[:6], cmap='BuGn', annot=True, vmin=0, vmax=Trd.max().max()+25)
# plt.title('Total RMSE')
# plt.show()

plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
#plt.ylim([19.75,20.75])
plt.legend()
plt.grid()
plt.show()
```

```python
from sklearn.decomposition import PCA

pca = PCA().fit(Train_X_std)

fig, ax = plt.subplots(figsize=(8,6))
x_values = range(1, pca.n_components_+1)
ax.bar(x_values, pca.explained_variance_ratio_, lw=2, label='Explained Variance')
ax.plot(x_values, np.cumsum(pca.explained_variance_ratio_), lw=2, label='Cumulative Explained Variance', color
plt.plot([0,pca.n_components_+1],[0.9,0.9],'g--')
ax.set_title('Explained variance of components')
ax.set_xlabel('Principal Component')
ax.set_ylabel('Explained Variance')
plt.legend()
plt.grid()
plt.show()
```

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import PolynomialFeatures
Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
m=df.shape[1]-4

for i in tqdm(range(m)):
    pca = PCA(n_components=Train_X_std.shape[1]-i)
    Train_X_std_pca = pca.fit_transform(Train_X_std)
    Test_X_std_pca = pca.fit_transform(Test_X_std)

    LR = LinearRegression()
    LR.fit(Train_X_std_pca, Train_Y)

    pred1 = LR.predict(Train_X_std_pca)
    pred2 = LR.predict(Test_X_std_pca)

    Trr.append(round(np.sqrt(mean_squared_error(Train_Y, pred1)),2))
    Tss.append(round(np.sqrt(mean_squared_error(Test_Y, pred2)),2))
```

```python
plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
#plt.ylim([19.5,20.75])
plt.legend()
plt.grid()
plt.show()
```

```python
#Shortlisting the selected Features (with RFE)

lm = LinearRegression()
rfe = RFE(lm,n_features_to_select=df.shape[1]-23)
rfe = rfe.fit(Train_X_std, Train_Y)

LR = LinearRegression()
LR.fit(Train_X_std.loc[:,rfe.support_], Train_Y)

#print(Train_X_std.loc[:,rfe.support_].columns)

pred1 = LR.predict(Train_X_std.loc[:,rfe.support_])
pred2 = LR.predict(Test_X_std.loc[:,rfe.support_])

print(np.sqrt(mean_squared_error(Train_Y, pred1)))
print(np.sqrt(mean_squared_error(Test_Y, pred2)))

Train_X_std = Train_X_std.loc[:,rfe.support_]
Test_X_std = Test_X_std.loc[:,rfe.support_]
```

```python
#Let us first define a function to evaluate our models

Model_Evaluation_Comparison_Matrix = pd.DataFrame(np.zeros([5,8]), columns=['Train-R2','Test-R2','Train-RSS','
                                                  'Train-MSE','Test-MSE','Train-RMSE
rc=np.random.choice(Train_X_std.loc[:,Train_X_std.nunique()>50].columns,3)
def Evaluate(n, pred1,pred2):
    #Plotting predicted predicteds alongside the actual datapoints
    plt.figure(figsize=[15,6])
    for e,i in enumerate(rc):
        plt.subplot(2,3,e+1)
        plt.scatter(y=Train_Y, x=Train_X_std[i], label='Actual')
        plt.scatter(y=pred1, x=Train_X_std[i], label='Prediction')
        plt.legend()
    plt.show()

    #Evaluating the Multiple Linear Regression Model

    print('\n\n{}Training Set Metrics{}'.format('-'*20, '-'*20))
    print('\nR2-Score on Training set --->',round(r2_score(Train_Y, pred1),20))
    print('Residual Sum of Squares (RSS) on Training set  --->',round(np.sum(np.square(Train_Y-pred1)),20))
    print('Mean Squared Error (MSE) on Training set       --->',round(mean_squared_error(Train_Y, pred1),20))
    print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(mean_squared_error(Train_Y, pred
```

```python
    print('\n\n{}Training Set Metrics{}'.format('-'*20, '-'*20))
    print('\nR2-Score on Training set --->',round(r2_score(Train_Y, pred1),20))
    print('Residual Sum of Squares (RSS) on Training set  --->',round(np.sum(np.square(Train_Y-pred1)),20))
    print('Mean Squared Error (MSE) on Training set       --->',round(mean_squared_error(Train_Y, pred1),20))
    print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(mean_squared_error(Train_Y, pred

    print('\n{}Testing Set Metrics{}'.format('-'*20, '-'*20))
    print('\nR2-Score on Testing set --->',round(r2_score(Test_Y, pred2),20))
    print('Residual Sum of Squares (RSS) on Training set  --->',round(np.sum(np.square(Test_Y-pred2)),20))
    print('Mean Squared Error (MSE) on Training set       --->',round(mean_squared_error(Test_Y, pred2),20))
    print('Root Mean Squared Error (RMSE) on Training set --->',round(np.sqrt(mean_squared_error(Test_Y, pred2
    print('\n{}Residual Plots{}'.format('-'*20, '-'*20))

    Model_Evaluation_Comparison_Matrix.loc[n,'Train-R2']  = round(r2_score(Train_Y, pred1),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-R2']   = round(r2_score(Test_Y, pred2),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-RSS'] = round(np.sum(np.square(Train_Y-pred1)),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-RSS']  = round(np.sum(np.square(Test_Y-pred2)),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-MSE'] = round(mean_squared_error(Train_Y, pred1),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-MSE']  = round(mean_squared_error(Test_Y, pred2),20)
    Model_Evaluation_Comparison_Matrix.loc[n,'Train-RMSE']= round(np.sqrt(mean_squared_error(Train_Y, pred1)),
    Model_Evaluation_Comparison_Matrix.loc[n,'Test-RMSE'] = round(np.sqrt(mean_squared_error(Test_Y, pred2)),2
```

```
    # Plotting y_test and y_pred to understand the spread.
    plt.figure(figsize=[15,4])

    plt.subplot(1,2,1)
    sns.distplot((Train_Y - pred1))
    plt.title('Error Terms')
    plt.xlabel('Errors')

    plt.subplot(1,2,2)
    plt.scatter(Train_Y,pred1)
    plt.plot([Train_Y.min(),Train_Y.max()],[Train_Y.min(),Train_Y.max()], 'r--')
    plt.title('Test vs Prediction')
    plt.xlabel('y_test')
    plt.ylabel('y_pred')
    plt.show()
```

```
#Linear Regression

MLR = LinearRegression().fit(Train_X_std,Train_Y)
pred1 = MLR.predict(Train_X_std)
pred2 = MLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Multiple Linear Regression Model \033[0m{}{}\n'.format('<'*3,'-'*25 ,'-'*25,'>'*
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(0, pred1, pred2)
```

```
#Creating a Ridge Regression model

RLR = Ridge().fit(Train_X_std,Train_Y)
pred1 = RLR.predict(Train_X_std)
pred2 = RLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Ridge Regression Model \033[0m{}{}\n'.format('<'*3,'-'*25 ,'-'*25,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(1, pred1, pred2)
```

```
#Creating a Ridge Regression model

LLR = Lasso().fit(Train_X_std,Train_Y)
pred1 = LLR.predict(Train_X_std)
pred2 = LLR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Lasso Regression Model \033[0m{}{}\n'.format('<'*3,'-'*25 ,'-'*25,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(2, pred1, pred2)
```

```
#Creating a ElasticNet Regression model

ENR = ElasticNet().fit(Train_X_std,Train_Y)
pred1 = ENR.predict(Train_X_std)
pred2 = ENR.predict(Test_X_std)

print('{}{}\033[1m Evaluating Elastic-Net Regression Model \033[0m{}{}\n'.format('<'*3,'-'*25 ,'-'*25,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(3, pred1, pred2)
```

```
Trr=[]; Tss=[]
n_degree=6

for i in range(2,n_degree):
    #print(f'{i} Degree')
    poly_reg = PolynomialFeatures(degree=i)
    X_poly = poly_reg.fit_transform(Train_X_std)
    X_poly1 = poly_reg.fit_transform(Test_X_std)
    LR = LinearRegression()
    LR.fit(X_poly, Train_Y)

    pred1 = LR.predict(X_poly)
    Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))

    pred2 = LR.predict(X_poly1)
    Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))

plt.figure(figsize=[15,6])
plt.subplot(1,2,1)
plt.plot(range(2,n_degree),Trr, label='Training')
plt.plot(range(2,n_degree),Tss, label='Testing')
#plt.plot([1,4],[1,4],'b--')
plt.title('Polynomial Regression Fit')
#plt.ylim([0,5])
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.grid()
plt.legend()
```

```
plt.subplot(1,2,2)
plt.plot(range(2,n_degree),Trr, label='Training')
plt.plot(range(2,n_degree),Tss, label='Testing')
plt.title('Polynomial Regression Fit')
plt.ylim([3,4.1])
plt.xlabel('Degree')
plt.ylabel('RMSE')
plt.grid()
plt.legend()
#plt.xticks()
plt.show()
```

```
#Using the 5th Order Polynomial Regression model (degree=5)

poly_reg = PolynomialFeatures(degree=5)
X_poly = poly_reg.fit_transform(Train_X_std)
X_poly1 = poly_reg.fit_transform(Test_X_std)
PR = LinearRegression()
PR.fit(X_poly, Train_Y)

pred1 = PR.predict(X_poly)
pred2 = PR.predict(X_poly1)

print('{}{}\033[1m Evaluating Polynomial Regression Model \033[0m{}{}\n'.format('<'*3,'-'*25 ,'-'*25,'>'*3))
print('The Coeffecient of the Regresion Model was found to be ',MLR.coef_)
print('The Intercept of the Regresion Model was found to be ',MLR.intercept_)

Evaluate(4, pred1, pred2)
```

```
# R2-Scores Comparison for different Regression Models

R2 = EMC['Train-R2'].sort_values(ascending=True)
plt.hlines(y=R2.index, xmin=0, xmax=R2.values)
plt.plot(R2.values, R2.index,'o')
plt.title('R2-Scores Comparison for various Regression Models')
plt.xlabel('R2-Score')
#plt.ylabel('Regression Models')
for i, v in enumerate(R2):
    plt.text(v+0.02, i-0.05, str(v*100), color='blue')
plt.xlim([0,1.1])
plt.show()
```

```
# Root Mean SquaredError Comparison for different Regression Models

cc = Model_Evaluation_Comparison_Matrix.columns.values
# baxes = brokenaxes(ylims=((0,4),(524,532)))
# baxes.bar(np.arange(s), Model_Evaluation_Comparison_Matrix[cc[-2]].values, width=0.3, label='RMSE (Training)
# baxes.bar(np.arange(s)+0.3, Model_Evaluation_Comparison_Matrix[cc[-1]].values, width=0.3, label='RMSE (Testi
# for index, value in enumerate(Model_Evaluation_Comparison_Matrix[cc[-2]].values):
#     plt.text(round(value,2), index, str(round(value,2)))
# for index, value in enumerate(Model_Evaluation_Comparison_Matrix[cc[-1]].values):
#     plt.text(round(value,2), index, str(round(value,2)))
plt.bar(np.arange(5), Model_Evaluation_Comparison_Matrix[cc[6]].values, width=0.3, label='RMSE (Training)')
plt.bar(np.arange(5)+0.3, Model_Evaluation_Comparison_Matrix[cc[7]].values, width=0.3, label='RMSE (Testing)')
plt.xticks(np.arange(5),EMC.index, rotation=35)
plt.legend()
#plt.ylim([0,10])
plt.show()
```

Assignment No. 2

cur_x = 3

The algorithm starts at x=3 rate = 0.01 # Learning rate

precision = 0.000001 #This tells us when to stop the algorithm previous_step_size = 1

max_iters = 10000

maximum number of iterations iters = 0

#iteration counter

df = lambda x: 2*(x+5)

#Gradient of our function

while previous_step_size > precision and iters < max_iters: prev_x = cur_x

#Store current x value in prev_x

cur_x = cur_x - rate * df(prev_x)

#Grad descent

previous_step_size = abs(cur_x - prev_x)

#Change in x iters = iters+1 #iteration count

print("Iteration",iters,"\nX value is",cur_x)

#Print iterations print("The local minimum occurs at", cur_x)

```
X value is -4.998919090416489
Iteration 442
X value is -4.99894070860816
Iteration 443
X value is -4.998961894435997
Iteration 444
X value is -4.998982656547277
Iteration 445
X value is -4.999003003416331
Iteration 446
X value is -4.999022943348004
Iteration 447
X value is -4.999042484481044
Iteration 448
X value is -4.999061634791423
             Iteration 449
X value is -4.999080402095594
Iteration 450
X value is -4.999098794053682
Iteration 451
X value is -4.999116818172609
```

```
Iteration 452
X value is -4.999134481809157
Iteration 453
X value is -4.999151792172974
Iteration 454
X value is -4.999168756329515
Iteration 455
X value is -4.999185381202924
Iteration 456
X value is -4.999201673578866
Iteration 457
X value is -4.999217640107289
Iteration 458
X value is -4.999233287305143
Iteration 459
X value is -4.9992486215590395
Iteration 460
X value is -4.999263649127859
Iteration 461
X value is -4.999278376145302
Iteration 462
X value is -4.999292808622396
Iteration 463
X value is -4.999306952449948
Iteration 464
X value is -4.999320813400949
Iteration 465
X value is -4.99933439713293
Iteration 466
X value is -4.999347709190272
Iteration 467
X value is -4.9993607550064665
Iteration 468
X value is -4.999373539906337
Iteration 469
X value is -4.99938606910821
Iteration 470
```

Assignment 3

KNN algorithm on diabetes dataset

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('diabetes.csv')

df.columns

Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'Pedigree', 'Age', 'Outcome'],

dtype='object')

Check for null values. If present remove null values from the dataset

df.isnull().sum()

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

```
BMI                     0
Pedigree                0
Age                     0
Outcome                 0
dtype: int64
```

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)

Confusion matrix: [[123 28]

[37 43]]

print("Acccuracy ",metrics.accuracy_score(y_test,y_pred))

Acccuracy 0.7186147186147186

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.

Also error_rate = 1- accuracy

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))

65

231

Error rate 0.2813852813852814

Error rate 0.2813852813852814

print("Precision score",metrics.precision_score(y_test,y_pred))

Precision score 0.6056338028169014

print("Recall score ",metrics.recall_score(y_test,y_pred))

Recall score 0.5375

```
print("Classification report ",metrics.classification_report(y_test,y_pred))
```

Classification report	precision	recall	f1-score	support
0	0.77	0.81	0.79	151
1	0.61	0.54	0.57	80
accuracy			0.72	231
macro avg	0.69	0.68	0.68	231
weighted avg	0.71	0.72	0.71	231

Assignment No : 4

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization
import seaborn as sns # for statistical data visualization
%matplotlib inline

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```python
import warnings

warnings.filterwarnings('ignore')
```

```python
df = pd.read_csv("sales_data_sample.csv", sep=",", encoding='Latin-1')
```

```python
df.shape
```

```
(7050, 16)
```

```python
df.head()
```

	status_id	status_type	status_published	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_l
0	246675545449582_1649696485147474	video	4/22/2018 6:00	529	512	262	432	92	3	
1	246675545449582_1649426988507757	photo	4/21/2018 22:45	150	0	0	150	0	0	
2	246675545449582_1648730588577397	video	4/21/2018 6:17	227	236	57	204	21	1	
3	246675545449582_1648576705259452	photo	4/21/2018 2:29	111	0	0	111	0	0	
4	246675545449582_1645700502213739	photo	4/18/2018 3:22	213	0	0	204	9	0	

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 16 columns):
status_id          7050 non-null object
status_type        7050 non-null object
status_published   7050 non-null object
num_reactions      7050 non-null int64
num_comments       7050 non-null int64
num_shares         7050 non-null int64
num_likes          7050 non-null int64
num_loves          7050 non-null int64
num_wows           7050 non-null int64
num_hahas          7050 non-null int64
num_sads           7050 non-null int64
num_angrys         7050 non-null int64
Column1            0 non-null float64
Column2            0 non-null float64
Column3            0 non-null float64
Column4            0 non-null float64
dtypes: float64(4), int64(9), object(3)
memory usage: 881.4+ KB
```

```python
df.isnull().sum()
```

```
status_id            0
status_type          0
status_published     0
num_reactions        0
num_comments         0
num_shares           0
num_likes            0
num_loves            0
num_wows             0
num_hahas            0
num_sads             0
num_angrys           0
Column1           7050
Column2           7050
Column3           7050
Column4           7050
dtype: int64
```

```python
df.drop(['Column1', 'Column2', 'Column3', 'Column4'], axis=1, inplace=True)
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 12 columns):
status_id           7050 non-null object
status_type         7050 non-null object
status_published    7050 non-null object
num_reactions       7050 non-null int64
num_comments        7050 non-null int64
num_shares          7050 non-null int64
num_likes           7050 non-null int64
num_loves           7050 non-null int64
num_wows            7050 non-null int64
num_hahas           7050 non-null int64
num_sads            7050 non-null int64
num_angrys          7050 non-null int64
dtypes: int64(9), object(3)
memory usage: 661.1+ KB
```

```python
df.describe()
```

	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
count	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000	7050.000000
mean	230.117163	224.356028	40.022553	215.043121	12.728652	1.289362	0.696454	0.243688	0.113191
std	462.625309	889.636820	131.599965	449.472357	39.972930	8.719650	3.957183	1.597156	0.726812
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	17.000000	0.000000	0.000000	17.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	59.500000	4.000000	0.000000	58.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	219.000000	23.000000	4.000000	184.750000	3.000000	0.000000	0.000000	0.000000	0.000000
max	4710.000000	20990.000000	3424.000000	4710.000000	657.000000	278.000000	157.000000	51.000000	31.000000

```python
# view the labels in the variable

df['status_id'].unique()
```

```
array(['246675545449582_1649696485147474',
       '246675545449582_1649426988507757',
       '246675545449582_1648730588577397', ...,
       '1050855161656896_1060126464063099',
       '1050855161656896_1058663487542730',
       '1050855161656896_1050858841656528'], dtype=object)
```

```
# view how many different types of variables are there
len(df['status_id'].unique())
```

```
6997
```

```
# view the labels in the variable
df['status_published'].unique()
```

```
array(['4/22/2018 6:00', '4/21/2018 22:45', '4/21/2018 6:17', ...,
       '9/21/2016 23:03', '9/20/2016 0:43', '9/10/2016 10:30'],
      dtype=object)
```

```
# view how many different types of variables are there
len(df['status_published'].unique())
```

```
6913
```

```
# view the labels in the variable
df['status_type'].unique()
```

```
array(['video', 'photo', 'link', 'status'], dtype=object)
```

```
# view how many different types of variables are there
len(df['status_type'].unique())
```

```
4
```

```
df.drop(['status_id', 'status_published'], axis=1, inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 10 columns):
status_type      7050 non-null object
num_reactions    7050 non-null int64
num_comments     7050 non-null int64
num_shares       7050 non-null int64
num_likes        7050 non-null int64
num_loves        7050 non-null int64
num_wows         7050 non-null int64
num_hahas        7050 non-null int64
num_sads         7050 non-null int64
num_angrys       7050 non-null int64
dtypes: int64(9), object(1)
memory usage: 550.9+ KB
```

```
df.head()
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	video	529	512	262	432	92	3	1	1	0
1	photo	150	0	0	150	0	0	0	0	0
2	video	227	236	57	204	21	1	1	0	0
3	photo	111	0	0	111	0	0	0	0	0
4	photo	213	0	0	204	9	0	0	0	0

```
X = df

y = df['status_type']
```

```python
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

X['status_type'] = le.fit_transform(X['status_type'])

y = le.transform(y)
```

```python
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7050 entries, 0 to 7049
Data columns (total 10 columns):
status_type      7050 non-null int64
num_reactions    7050 non-null int64
num_comments     7050 non-null int64
num_shares       7050 non-null int64
num_likes        7050 non-null int64
num_loves        7050 non-null int64
num_wows         7050 non-null int64
num_hahas        7050 non-null int64
num_sads         7050 non-null int64
num_angrys       7050 non-null int64
dtypes: int64(10)
memory usage: 550.9 KB
```

```python
X.head()
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	3	529	512	262	432	92	3	1	1	0
1	1	150	0	0	150	0	0	0	0	0
2	3	227	236	57	204	21	1	1	0	0
3	1	111	0	0	111	0	0	0	0	0
4	1	213	0	0	204	9	0	0	0	0

```python
cols = X.columns
```

```python
from sklearn.preprocessing import MinMaxScaler

ms = MinMaxScaler()

X = ms.fit_transform(X)
```

```python
X = pd.DataFrame(X, columns=[cols])
```

```python
X.head()
```

	status_type	num_reactions	num_comments	num_shares	num_likes	num_loves	num_wows	num_hahas	num_sads	num_angrys
0	1.000000	0.112314	0.024393	0.076519	0.091720	0.140030	0.010791	0.006369	0.019608	0.0
1	0.333333	0.031847	0.000000	0.000000	0.031847	0.000000	0.000000	0.000000	0.000000	0.0
2	1.000000	0.048195	0.011243	0.016647	0.043312	0.031963	0.003597	0.006369	0.000000	0.0
3	0.333333	0.023567	0.000000	0.000000	0.023567	0.000000	0.000000	0.000000	0.000000	0.0
4	0.333333	0.045223	0.000000	0.000000	0.043312	0.013699	0.000000	0.000000	0.000000	0.0

```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=0)

kmeans.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

```python
kmeans.cluster_centers_
```

```
array([[3.28506857e-01, 3.90710874e-02, 7.54854864e-04, 7.53667113e-04,
        3.85438884e-02, 2.17448568e-03, 2.43721364e-03, 1.20039760e-03,
        2.75348016e-03, 1.45313276e-03],
       [9.54921576e-01, 6.46330441e-02, 2.67028654e-02, 2.93171709e-02,
        5.71231462e-02, 4.71007076e-02, 8.18581889e-03, 9.65207685e-03,
        8.04219428e-03, 7.19501847e-03]])
```

```python
kmeans.inertia_
```

```
237.75726404419564
```

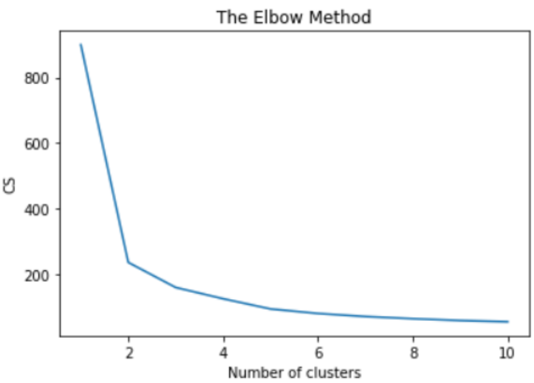
```python
labels = kmeans.labels_

# check how many of the samples were correctly labeled
correct_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
```

```
Result: 63 out of 7050 samples were correctly labeled.
```

```python
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Accuracy score: 0.01
```

```python
from sklearn.cluster import KMeans
cs = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    cs.append(kmeans.inertia_)
plt.plot(range(1, 11), cs)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('CS')
plt.show()
```




```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2,random_state=0)

kmeans.fit(X)

labels = kmeans.labels_

# check how many of the samples were correctly labeled

correct_labels = sum(y == labels)

print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))

print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 63 out of 7050 samples were correctly labeled.
Accuracy score: 0.01
```

```python
kmeans = KMeans(n_clusters=3, random_state=0)

kmeans.fit(X)

# check how many of the samples were correctly labeled
labels = kmeans.labels_

correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 138 out of 7050 samples were correctly labeled.
Accuracy score: 0.02
```

```python
kmeans = KMeans(n_clusters=4, random_state=0)

kmeans.fit(X)

# check how many of the samples were correctly labeled
labels = kmeans.labels_

correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 4340 out of 7050 samples were correctly labeled.
Accuracy score: 0.62
```

GROUP C

Assignment: 3

Loops in Solidity:

```
pragma solidity >= 0.5.0 < 0.9.0; contract Loops {

uint [3] public arr; uint public count;

function Whileloop() public { while(count < arr.length) {

arr[count] = count; count++;

}

}

function Forloop() public {

for(uint i=count; i<arr.length; i++) { arr[count] = count;

count++;

}

}
```

If-Else in Solodity :

```
pragma solidity >= 0.5.0 < 0.9.0; contract Array {

function check(int a) public pure returns(string memory) { string memory value;

if(a > 0) {

value = "Greater Than zero";

}

else if(a == 0) {

value = "Equal to zero";

}

else {

value = "Less than zero";

}

return value;

}

}
```

Create a Smart Contract with CRUD Functionality

```
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].nam
                       e =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");

        }

   }
```