MATH2319 Machine Learning - Assignment 1

(1) I hereby agree to follow any and all assignment rules and procedures as stated in Canvas for this course, MATH2319.

(2) In particular, I solemnly swear that I will not discuss/ have not discussed my assignment solutions with anyone in any way and the solutions I am submitting are my own personal work.

Full Name: Harini Mylanahally Sannaveeranna - s3755660 ¶

In [1]:

```
#importing packages

import pandas as pd
import numpy as np



from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

Question 1

The data set used is sourced from the UCI Machine Learning Repository (mentioned in the URL below). we are performing data preprocessing and exploration.

In [2]:

```
#here, attributeNames are the column names of crx.names file

attributeNames = [
    'A1',
    'A2',
    'A3',
    'A4',
    'A5',
    'A6',
    'A7',
    'A8',
    'A9',
    'A10',
    'A11',
    'A12',
    'A13',
    'A14',
    'A15',
    'A16',
]

url = (
    "http://archive.ics.uci.edu/ml/machine-learning-databases/credit-screening/c
rx.data"
)

# Read the dataset
df = pd.read_csv(url, sep = ',', names = attributeNames, header = None)

#df.head()

df
```

Out[2]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	b	21.08	10.085	y	p	e	h	1.25	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.00	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.00	f	t	1	t	g	00200	1	-
688	b	17.92	0.205	u	g	aa	v	0.04	f	f	0	f	g	00280	750	-
689	b	35.00	3.375	u	g	c	h	8.29	f	f	0	t	g	00000	0	-

690 rows × 16 columns

In [3]:

```
# Print summary statistics
df.describe()
```

Out[3]:

	A3	A8	A11	A15
count	690.000000	690.000000	690.00000	690.000000
mean	4.758725	2.223406	2.40000	1017.385507
std	4.978163	3.346513	4.86294	5210.102598
min	0.000000	0.000000	0.00000	0.000000
25%	1.000000	0.165000	0.00000	0.000000
50%	2.750000	1.000000	0.00000	5.000000
75%	7.207500	2.625000	3.00000	395.500000
max	28.000000	28.500000	67.00000	100000.000000

In [4]:

```
#checking the datatypes of the dataframe df

df.dtypes
```

Out[4]:

```
A1      object
A2      object
A3     float64
A4      object
A5      object
A6      object
A7      object
A8     float64
A9      object
A10     object
A11      int64
A12     object
A13     object
A14     object
A15      int64
A16     object
dtype: object
```

In [5]:

```
# Print DataFrame information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   A1      690 non-null    object 
 1   A2      690 non-null    object 
 2   A3      690 non-null    float64
 3   A4      690 non-null    object 
 4   A5      690 non-null    object 
 5   A6      690 non-null    object 
 6   A7      690 non-null    object 
 7   A8      690 non-null    float64
 8   A9      690 non-null    object 
 9   A10     690 non-null    object 
 10  A11     690 non-null    int64  
 11  A12     690 non-null    object 
 12  A13     690 non-null    object 
 13  A14     690 non-null    object 
 14  A15     690 non-null    int64  
 15  A16     690 non-null    object 
dtypes: float64(2), int64(2), object(12)
memory usage: 86.4+ KB
```

In [6]:

```
#checking missing values
df.isnull()
```

Out[6]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A1
0	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
1	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
2	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
3	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
4	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
685	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
686	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
687	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
688	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals
689	False	False	False	False	False	False	False	False	False	False	False	False	False	Fals

690 rows × 16 columns

In [7]:

```
# Inspect overall missing values in the dataset
print(df.isnull().values.sum())
```

```
0
```

In [8]:

```
# Inspect missing values in each column in the dataset
print(df.isnull().sum())
```

```
A1      0
A2      0
A3      0
A4      0
A5      0
A6      0
A7      0
A8      0
A9      0
A10     0
A11     0
A12     0
A13     0
A14     0
A15     0
A16     0
dtype: int64
```

In [9]:

```
# Inspect missing values in each column in the dataset
print(df.isna().sum())
```

```
A1      0
A2      0
A3      0
A4      0
A5      0
A6      0
A7      0
A8      0
A9      0
A10     0
A11     0
A12     0
A13     0
A14     0
A15     0
A16     0
dtype: int64
```

Checking the unique values for each column in the dataset to check whether the unusual or undefined values are present or not, by using unique() :

In [10]:

```
df['A1'].unique()
```

Out[10]:

```
array(['b', 'a', '?'], dtype=object)
```

inspected that in column A1 , found one unusual value ?

In [11]:

```
df['A2'].unique()
```

```
Out[11]:
```

```
array(['30.83', '58.67', '24.50', '27.83', '20.17', '32.08', '33.1
7',
       '22.92', '54.42', '42.50', '22.08', '29.92', '38.25', '48.0
8',
       '45.83', '36.67', '28.25', '23.25', '21.83', '19.17', '25.0
0',
       '47.75', '27.42', '41.17', '15.83', '47.00', '56.58', '57.4
2',
       '42.08', '29.25', '42.00', '49.50', '36.75', '22.58', '27.2
5',
       '23.00', '27.75', '54.58', '34.17', '28.92', '29.67', '39.5
8',
       '56.42', '54.33', '41.00', '31.92', '41.50', '23.92', '25.7
5',
       '26.00', '37.42', '34.92', '34.25', '23.33', '23.17', '44.3
3',
       '35.17', '43.25', '56.75', '31.67', '23.42', '20.42', '26.6
7',
       '36.00', '25.50', '19.42', '32.33', '34.83', '38.58', '44.2
5',
       '44.83', '20.67', '34.08', '21.67', '21.50', '49.58', '27.6
7',
       '39.83', '?', '37.17', '25.67', '34.00', '49.00', '62.50', '3
1.42',
       '52.33', '28.75', '28.58', '22.50', '28.50', '37.50', '35.2
5',
       '18.67', '54.83', '40.92', '19.75', '29.17', '24.58', '33.7
5',
       '25.42', '37.75', '52.50', '57.83', '20.75', '39.92', '24.7
5',
       '44.17', '23.50', '47.67', '22.75', '34.42', '28.42', '67.7
5',
       '47.42', '36.25', '32.67', '48.58', '33.58', '18.83', '26.9
2',
       '31.25', '56.50', '43.00', '22.33', '32.83', '40.33', '30.5
0',
       '52.83', '46.67', '58.33', '37.33', '23.08', '32.75', '68.6
7',
       '28.00', '44.00', '25.08', '32.00', '60.58', '40.83', '19.3
3',
       '41.33', '56.00', '49.83', '22.67', '27.00', '26.08', '18.4
2',
       '21.25', '57.08', '22.42', '48.75', '40.00', '40.58', '28.6
7',
       '33.08', '21.33', '41.75', '34.50', '48.17', '27.58', '24.0
8',
       '24.83', '36.33', '35.42', '71.58', '39.50', '39.33', '24.3
3',
       '60.08', '55.92', '53.92', '18.92', '50.08', '65.42', '17.5
8',
       '18.08', '19.67', '25.17', '33.50', '58.42', '26.17', '42.8
3',
       '38.17', '20.50', '48.25', '28.33', '18.75', '18.50', '45.0
0',
       '40.25', '41.42', '17.83', '18.17', '20.00', '52.17', '50.7
5',
       '17.08', '18.33', '59.67', '18.00', '37.58', '30.67', '18.5
8',
       '16.25', '21.17', '17.67', '16.50', '29.50', '21.75', '18.2
5',
       '35.75', '16.08', '69.17', '32.92', '16.33', '22.17', '57.5
8',
          '15.92', '31.75', '19.00', '17.50', '33.67', '30.17', '33.2
5',
          '25.25', '34.75', '47.33', '39.08', '42.75', '38.92', '62.7
5',
          '32.25', '26.75', '63.33', '30.75', '16.00', '19.50', '32.4
2',
          '30.25', '26.83', '16.92', '24.42', '39.42', '23.58', '21.4
2',
          '33.00', '26.33', '26.25', '28.17', '20.83', '43.17', '56.8
3',
          '15.17', '29.83', '31.00', '51.92', '69.50', '19.58', '22.2
5',
          '38.42', '26.58', '35.00', '29.42', '49.17', '51.83', '58.5
8',
          '53.33', '27.17', '25.92', '30.58', '17.25', '27.33', '36.5
0',
          '29.75', '52.42', '36.17', '34.58', '21.92', '36.58', '31.0
8',
          '30.42', '21.08', '17.42', '39.17', '26.50', '17.33', '23.7
5',
          '34.67', '74.83', '45.33', '47.25', '24.17', '39.25', '39.0
0',
          '64.08', '31.33', '21.00', '13.75', '46.00', '20.25', '60.9
2',
          '30.00', '22.83', '45.17', '41.58', '55.75', '25.33', '31.8
3',
          '33.92', '24.92', '80.25', '30.08', '48.33', '76.75', '51.3
3',
          '41.92', '29.58', '32.17', '51.42', '42.17', '43.08', '59.5
0',
          '65.17', '20.33', '48.50', '28.08', '73.42', '51.58', '38.6
7',
          '46.08', '20.08', '42.25', '16.17', '47.83', '22.00', '38.3
3',
          '25.58', '21.58', '36.08', '38.75', '35.58', '31.58', '15.7
5',
          '17.92', '30.33', '47.17', '25.83', '50.25', '36.42'], dtype=
object)
```

In [12]:

```
df['A3'].unique()
```

Out[12]:

```
array([ 0.   ,  4.46 ,  0.5  ,  1.54 ,  5.625,  4.   ,  1.04 , 11.58
5,
        4.915,  0.83 ,  1.835,  6.   ,  6.04 , 10.5  ,  4.415,  0.87
5,
        5.875,  0.25 ,  8.585, 11.25 ,  1.   ,  8.   , 14.5  ,  6.5
,
        0.585, 13.   , 18.5  ,  8.5  , 14.79 ,  9.79 ,  7.585,  5.12
5,
       10.75 ,  1.5  ,  1.585, 11.75 ,  9.415,  9.17 , 15.   ,  1.41
5,
       13.915, 28.   ,  6.75 ,  2.04 ,  0.665,  2.5  ,  3.   , 11.62
5,
        4.5  , 12.25 , 16.165,  0.79 ,  0.835,  4.25 ,  0.375, 25.12
5,
        7.5  ,  5.   ,  7.   ,  5.29 ,  1.165,  9.75 , 19.   ,  3.5
,
        0.625,  2.21 , 12.75 , 15.5  ,  1.375,  3.54 , 11.   ,  1.75
,
       16.5  , 12.   ,  2.25 ,  0.75 , 12.5  ,  1.25 ,  1.125,  7.04
,
       10.335,  6.21 ,  6.665,  9.   ,  5.5  ,  0.54 ,  2.75 ,  9.5
,
       13.5  ,  3.75 , 16.   ,  0.29 ,  1.665,  7.54 ,  0.46 , 10.
,
       11.5  ,  3.04 ,  2.   ,  0.08 ,  1.71 ,  3.25 ,  2.54 , 13.58
5,
        8.665,  9.25 ,  8.17 ,  2.335, 19.5  ,  5.665,  4.625,  0.20
5,
        0.96 ,  4.04 ,  5.04 ,  3.165,  7.625, 10.04 , 10.25 ,  2.12
5,
        9.335,  6.625,  2.71 ,  9.625, 12.54 ,  9.54 ,  8.46 , 13.75
,
       21.   , 10.125, 25.085,  0.21 , 21.5  , 11.125, 11.045,  1.33
5,
        0.085,  1.21 ,  0.165,  5.71 ,  5.415, 12.625,  0.58 ,  0.41
5,
        2.415,  0.335,  3.125, 12.125,  2.875, 13.665, 26.335, 10.29
,
        1.29 , 22.   ,  0.125,  1.085,  4.085,  4.71 ,  6.165,  4.58
5,
       11.46 , 14.585,  0.17 ,  1.625,  2.085,  5.085,  8.125,  2.83
5,
        1.79 ,  0.705,  2.165,  2.29 , 18.125,  3.085, 11.665,  4.12
5,
        1.08 , 13.335, 11.835,  4.79 ,  9.96 ,  7.08 , 25.21 ,  0.67
,
        3.79 , 22.29 ,  3.335,  0.42 ,  1.46 ,  0.04 , 12.33 , 12.33
5,
        0.915, 14.   , 17.75 , 20.   ,  5.25 ,  4.165, 10.915,  4.75
,
       10.415,  7.835,  0.71 ,  2.46 ,  9.585,  3.625,  2.665,  5.83
5,
       12.835, 10.665,  7.25 , 10.21 ,  3.29 , 10.085,  3.375])
```

In [13]:

```
df['A4'].unique()
```

Out[13]:

```
array(['u', 'y', '?', 'l'], dtype=object)
```

inspected that in column A4 , found one unusual value- ?

In [14]:

```
df['A5'].unique()
```

Out[14]:

```
array(['g', 'p', '?', 'gg'], dtype=object)
```

inspected that in column A5 , found one unusual value- ?

checking unusual values(?) for the remaining columns by using unique() :-

In [15]:

```
df['A6'].unique()
```

Out[15]:

```
array(['w', 'q', 'm', 'r', 'cc', 'k', 'c', 'd', 'x', 'i', 'e', 'aa',
'ff',
       'j', '?'], dtype=object)
```

In [16]:

```
df['A7'].unique()
```

Out[16]:

```
array(['v', 'h', 'bb', 'ff', 'j', 'z', '?', 'o', 'dd', 'n'], dtype=o
bject)
```

In [17]:

```
df['A8'].unique()
```

Out[17]:

```
array([ 1.25 ,  3.04 ,  1.5  ,  3.75 ,  1.71 ,  2.5  ,  6.5  ,  0.04
,
        3.96 ,  3.165,  2.165,  4.335,  1.   ,  5.   ,  0.25 ,  0.96
,
        3.17 ,  0.665,  0.75 ,  0.835,  7.875,  3.085,  0.5  ,  5.16
5,
       15.   ,  7.   ,  5.04 ,  7.96 ,  7.585,  0.415,  2.   ,  1.83
5,
       14.415,  4.5  ,  5.335,  8.625, 28.5  ,  2.625,  0.125,  6.04
,
        3.5  ,  0.165,  0.875,  1.75 ,  0.   ,  7.415,  0.085,  5.75
,
        6.   ,  3.   ,  1.585,  4.29 ,  1.54 ,  1.46 ,  1.625, 12.5
,
       13.5  , 10.75 ,  0.375,  0.585,  0.455,  4.   ,  8.5  ,  9.46
,
        2.25 , 10.   ,  0.795,  1.375,  1.29 , 11.5  ,  6.29 , 14.
,
        0.335,  1.21 ,  7.375,  7.5  ,  3.25 , 13.   ,  5.5  ,  4.25
,
        0.625,  5.085,  2.75 ,  2.375,  8.   ,  1.085,  2.54 ,  4.16
5,
        1.665, 11.   ,  9.   ,  1.335,  1.415,  1.96 ,  2.585,  5.12
5,
       15.5  ,  0.71 ,  5.665, 18.   ,  5.25 ,  8.665,  2.29 , 20.
,
        2.46 , 13.875,  2.085,  4.58 ,  2.71 ,  2.04 ,  0.29 ,  4.75
,
        0.46 ,  0.21 ,  0.54 ,  3.335,  2.335,  1.165,  2.415,  2.79
,
        4.625,  1.04 ,  6.75 ,  1.875, 16.   , 12.75 ,  5.375,  2.12
5,
       17.5  ,  3.125,  0.79 ,  8.29 ])
```

In [18]:

```
df['A9'].unique()
```

Out[18]:

```
array(['t', 'f'], dtype=object)
```

In [19]:

```
df['A10'].unique()
```

Out[19]:

```
array(['t', 'f'], dtype=object)
```

In [20]:

```
df['A11'].unique()
```

Out[20]:

```
array([ 1,  6,  0,  5,  7, 10,  3, 17,  2,  9,  8, 15, 11, 12, 40, 2
3,  4,
       20, 67, 14, 16, 13, 19])
```

In [21]:

```
df['A12'].unique()
```

Out[21]:

```
array(['f', 't'], dtype=object)
```

In [22]:

```
df['A13'].unique()
```

Out[22]:

```
array(['g', 's', 'p'], dtype=object)
```

In [23]:

```
df['A14'].unique()
```

Out[23]:

```
array(['00202', '00043', '00280', '00100', '00120', '00360', '0016
4',
       '00080', '00180', '00052', '00128', '00260', '00000', '0032
0',
       '00396', '00096', '00200', '00300', '00145', '00500', '0016
8',
       '00434', '00583', '00030', '00240', '00070', '00455', '0031
1',
       '00216', '00491', '00400', '00239', '00160', '00711', '0025
0',
       '00520', '00515', '00420', '?', '00980', '00443', '00140', '0
0094',
       '00368', '00288', '00928', '00188', '00112', '00171', '0026
8',
       '00167', '00075', '00152', '00176', '00329', '00212', '0041
0',
       '00274', '00375', '00408', '00350', '00204', '00040', '0018
1',
       '00399', '00440', '00093', '00060', '00395', '00393', '0002
1',
       '00029', '00102', '00431', '00370', '00024', '00020', '0012
9',
       '00510', '00195', '00144', '00380', '00049', '00050', '0038
1',
       '00150', '00117', '00056', '00211', '00230', '00156', '0002
2',
       '00228', '00519', '00253', '00487', '00220', '00088', '0007
3',
       '00121', '00470', '00136', '00132', '00292', '00154', '0027
2',
       '00340', '00108', '00720', '00450', '00232', '00170', '0116
0',
       '00411', '00460', '00348', '00480', '00640', '00372', '0027
6',
       '00221', '00352', '00141', '00178', '00600', '00550', '0200
0',
       '00225', '00210', '00110', '00356', '00045', '00062', '0009
2',
       '00174', '00017', '00086', '00454', '00254', '00028', '0026
3',
       '00333', '00312', '00290', '00371', '00099', '00252', '0076
0',
       '00560', '00130', '00523', '00680', '00163', '00208', '0038
3',
       '00330', '00422', '00840', '00432', '00032', '00186', '0030
3',
       '00349', '00224', '00369', '00076', '00231', '00309', '0041
6',
       '00465', '00256'], dtype=object)
```

In [24]:

```
df['A15'].unique()
```

```
Out[24]:
```

```
array([     0,    560,    824,      3,  31285,   1349,    314,   144
2,
          200,   2690,    245,   1208,   1260,     11,  10000,   500
0,
         4000,     35,    713,    551,    500,    300,    221,   228
3,
          100,     15,    284,   1236,   5800,    730,    400,  5000
0,
          456,  15108,   2954,      2,     20,     27,    225,
1,
           38,      5,    130,    147,    210,  11202,   1332,     5
0,
          258,    567,   1000,   2510,    809,    610,    150,  5110
0,
          367,    600,    247,    375,    278,    827,   2072,    58
2,
         2300,   3065,   2200,      6,   1602,   2184,   3376,   200
0,
         7544,  10561,    837,  11177,    639,   2028,   1065,    54
0,
          158,  15000,   3000,   3257,   1655,   1430,      7,    79
0,
          396,    678,   1187,   6590,    168,   1270,   1210,    74
2,
         8851,   7059,   1704,    857,   6700,   2503,   9800,    19
6,
           14,  26726,  18027,     99,    444,   1200,   2010,     1
3,
          120,     32,    722,     40,    484,    204,     98,   555
2,
          105,   2803,    126,      4,     21,    173,     10,     2
5,
           42, 100000,    113,      8,     44,   2732,    179,     1
6,
         1062,    251,    228,     67,     12,    122,   4208,   130
0,
          112,   1110,   1004,    286,   4500,   1212,    195,     8
7,
           17,    184,    140,     18,    146,     22,     55,     7
0,
           60,   1058,    769,   5200,     19,    316,    350,   355
2,
          687,   1950,     53,     41,     33,     80,    351,   210
0,
          475,    892,   4607,   2206,   5860,     28,   1391,   227
9,
          591,    960,    690,    234,    800,    990,   2197,     9
0,
          340,    347,    327,   4071,    109,   1249,    134,   134
4,
          321,    948,   2079,   2384,    458,   5298,    162,   158
3,
           58,     59,   1400,   1465,   8000,   4700,   1097,   329
0,
        13212,   5777,   5124,     23,   4159,    918,    768,    28
3,
          108,      9,     68,    587,    141,    501,    160,    39
0,
          154,    117,    246,    237,    364,    537,    394,    75
0])
```

```
In [25]:
```

```
df['A16'].unique()
```

```
Out[25]:
```

```
array(['+', '-'], dtype=object)
```

Now replacing the unusual values (?) with NaN by using replace() function:

```
In [26]:
```

```
# Replace the '?'s with NaN
df = df.replace("?",np.NaN)
df
```

```
Out[26]:
```

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	b	21.08	10.085	y	p	e	h	1.25	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.00	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.00	f	t	1	t	g	00200	1	-
688	b	17.92	0.205	u	g	aa	v	0.04	f	f	0	f	g	00280	750	-
689	b	35.00	3.375	u	g	c	h	8.29	f	f	0	t	g	00000	0	-

690 rows × 16 columns

In [27]:

```
#displaying rows ranging from 478 to 491

df[478:491]
```

Out[27]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A
478	b	22.75	11.500	u	g	i	v	0.415	f	f	0	f	g	00000	0	
479	NaN	26.50	2.710	y	p	NaN	NaN	0.085	f	f	0	f	s	00080	0	
480	a	16.92	0.500	u	g	i	v	0.165	f	t	6	t	g	00240	35	
481	b	23.50	3.165	y	p	k	v	0.415	f	t	1	t	g	00280	80	
482	a	17.33	9.500	u	g	aa	v	1.750	f	t	10	t	g	00000	10	
483	b	23.75	0.415	y	p	c	v	0.040	f	t	2	f	g	00128	6	
484	b	34.67	1.080	u	g	m	v	1.165	f	f	0	f	s	00028	0	
485	b	74.83	19.000	y	p	ff	ff	0.040	f	t	2	f	g	00000	351	
486	b	28.17	0.125	y	p	k	v	0.085	f	f	0	f	g	00216	2100	
487	b	24.50	13.335	y	p	aa	v	0.040	f	f	0	t	g	00120	475	
488	b	18.83	3.540	y	p	ff	ff	0.000	f	f	0	t	g	00180	1	
489	NaN	45.33	1.000	u	g	q	v	0.125	f	f	0	t	g	00263	0	
490	a	47.25	0.750	u	g	q	h	2.750	t	t	1	f	g	00333	892	

in the above, we can see many NaN values, which are replaced by ? in row 479 and 489

In [28]:

```
# Inspect the missing values again in the last 17 rows
df.tail(17)
```

Out[28]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
673	NaN	29.50	2.000	y	p	e	h	2.000	f	f	0	f	g	00256	17	-
674	a	37.33	2.500	u	g	i	h	0.210	f	f	0	f	g	00260	246	-
675	a	41.58	1.040	u	g	aa	v	0.665	f	f	0	f	g	00240	237	-
676	a	30.58	10.665	u	g	q	h	0.085	f	t	12	t	g	00129	3	-
677	b	19.42	7.250	u	g	m	v	0.040	f	t	1	f	g	00100	1	-
678	a	17.92	10.210	u	g	ff	ff	0.000	f	f	0	f	g	00000	50	-
679	a	20.08	1.250	u	g	c	v	0.000	f	f	0	f	g	00000	0	-
680	b	19.50	0.290	u	g	k	v	0.290	f	f	0	f	g	00280	364	-
681	b	27.83	1.000	y	p	d	h	3.000	f	f	0	f	g	00176	537	-
682	b	17.08	3.290	u	g	i	v	0.335	f	f	0	t	g	00140	2	-
683	b	36.42	0.750	y	p	d	v	0.585	f	f	0	f	g	00240	3	-
684	b	40.58	3.290	u	g	m	v	3.500	f	f	0	t	s	00400	0	-
685	b	21.08	10.085	y	p	e	h	1.250	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.000	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.000	f	t	1	t	g	00200	1	-
688	b	17.92	0.205	u	g	aa	v	0.040	f	f	0	f	g	00280	750	-
689	b	35.00	3.375	u	g	c	h	8.290	f	f	0	t	g	00000	0	-

In [29]:

```
# Inspect missing values in the dataset
print(df.isnull().sum())
```

```
A1      12
A2      12
A3       0
A4       6
A5       6
A6       9
A7       9
A8       0
A9       0
A10      0
A11      0
A12      0
A13      0
A14     13
A15      0
A16      0
dtype: int64
```

we can see that A1, A2, A4, A5, A6, A7 and A14 are having missing values

Missing Values (Nan) :

Now we will replace numerical features with median value for A2, A3, A8, A11, A14 and A15

In [30]:

```
# Impute the missing values with median imputation numerical features
df['A2'] = df['A2'].fillna(df['A2'].median())
df['A3'] = df['A3'].fillna(df['A3'].median())
df['A8'] = df['A8'].fillna(df['A8'].median())
df['A11'] = df['A11'].fillna(df['A11'].median())
df['A14'] = df['A14'].fillna(df['A14'].median())
df['A15'] = df['A15'].fillna(df['A15'].median())
```

In [31]:

```
# Count the number of NaNs in the dataset to verify
print(df.isnull().sum())
```

```
A1     12
A2      0
A3      0
A4      6
A5      6
A6      9
A7      9
A8      0
A9      0
A10     0
A11     0
A12     0
A13     0
A14     0
A15     0
A16     0
dtype: int64
```

A1, A4, A5, A6, A7 columns are having NaN values and these categorical features. We will now replace these NaN with mode()

In [32]:

```
#Remove the excessive white space in A1, A4, A5, A6 and A7 as these are conatini
ng categorical features
df['A1'] = df['A1'].str.strip()
```

In [33]:

```
df['A4'] = df['A4'].str.strip()
```

In [34]:

```
df['A5'] = df['A5'].str.strip()
```

In [35]:

```
df['A6'] = df['A6'].str.strip()
```

In [36]:

```
df['A7'] = df['A7'].str.strip()
```

In [37]:

```
df
```

Out[37]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	b	21.08	10.085	y	p	e	h	1.25	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.00	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.00	f	t	1	t	g	00200	1	-
688	b	17.92	0.205	u	g	aa	v	0.04	f	f	0	f	g	00280	750	-
689	b	35.00	3.375	u	g	c	h	8.29	f	f	0	t	g	00000	0	-

690 rows × 16 columns

In [38]:

```
# Impute the missing values with mode imputation for categorical features

df['A1'] = df['A1'].fillna(df['A1'].mode())
df['A4'] = df['A4'].fillna(df['A4'].mode())
df['A5'] = df['A5'].fillna(df['A5'].mode())
df['A6'] = df['A6'].fillna(df['A6'].mode())
df['A7'] = df['A7'].fillna(df['A7'].mode())
```

In [39]:

```
# Count the number of NaNs in the dataset to verify
print(df.isnull().values.sum())
```

```
42
```

In [40]:

```
# Count the number of NaNs in the dataset to verify
print(df.isna().sum())
```

```
A1      12
A2       0
A3       0
A4       6
A5       6
A6       9
A7       9
A8       0
A9       0
A10      0
A11      0
A12      0
A13      0
A14      0
A15      0
A16      0
dtype: int64
```

In [41]:

```
# Imputing missing observations in categorical columns with mode (alphabetically
occurs)

for col in df:
 if  df[col].isnull().any():
     impute_values = df[col].value_counts().index[0]
     df[col].fillna(impute_values, inplace = True)
```

In [42]:

```
# Count the number of NaNs in the dataset to verify
print(df.isnull().sum())
```

```
A1      0
A2      0
A3      0
A4      0
A5      0
A6      0
A7      0
A8      0
A9      0
A10     0
A11     0
A12     0
A13     0
A14     0
A15     0
A16     0
dtype: int64
```

Now all the NaN values are removed

In [43]:

```
# Counting the number of elements(values)in each column in the dataset, just to
see all values are replaced with NaN values
print(df.count())
```

```
A1      690
A2      690
A3      690
A4      690
A5      690
A6      690
A7      690
A8      690
A9      690
A10     690
A11     690
A12     690
A13     690
A14     690
A15     690
A16     690
dtype: int64
```

In [44]:

```
#displaying last 20 rows in the dataframe
df.tail(20)
```

Out[44]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
670	b	47.17	5.835	u	g	w	v	5.500	f	f	0	f	g	00465	150	-
671	b	25.83	12.835	u	g	cc	v	0.500	f	f	0	f	g	00000	2	-
672	a	50.25	0.835	u	g	aa	v	0.500	f	f	0	t	g	00240	117	-
673	b	29.50	2.000	y	p	e	h	2.000	f	f	0	f	g	00256	17	-
674	a	37.33	2.500	u	g	i	h	0.210	f	f	0	f	g	00260	246	-
675	a	41.58	1.040	u	g	aa	v	0.665	f	f	0	f	g	00240	237	-
676	a	30.58	10.665	u	g	q	h	0.085	f	t	12	t	g	00129	3	-
677	b	19.42	7.250	u	g	m	v	0.040	f	t	1	f	g	00100	1	-
678	a	17.92	10.210	u	g	ff	ff	0.000	f	f	0	f	g	00000	50	-
679	a	20.08	1.250	u	g	c	v	0.000	f	f	0	f	g	00000	0	-
680	b	19.50	0.290	u	g	k	v	0.290	f	f	0	f	g	00280	364	-
681	b	27.83	1.000	y	p	d	h	3.000	f	f	0	f	g	00176	537	-
682	b	17.08	3.290	u	g	i	v	0.335	f	f	0	t	g	00140	2	-
683	b	36.42	0.750	y	p	d	v	0.585	f	f	0	f	g	00240	3	-
684	b	40.58	3.290	u	g	m	v	3.500	f	f	0	t	s	00400	0	-
685	b	21.08	10.085	y	p	e	h	1.250	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.000	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.000	f	t	1	t	g	00200	1	-
688	b	17.92	0.205	u	g	aa	v	0.040	f	f	0	f	g	00280	750	-
689	b	35.00	3.375	u	g	c	h	8.290	f	f	0	t	g	00000	0	-

In [45]:

```
#checking datatypes in dataframe
df.dtypes
```

Out[45]:

```
A1       object
A2       object
A3      float64
A4       object
A5       object
A6       object
A7       object
A8      float64
A9       object
A10      object
A11       int64
A12      object
A13      object
A14      object
A15       int64
A16      object
dtype: object
```

In [46]:

```
#displaying the boxplot to check the outliers in the dataset df

import matplotlib.pyplot as plt

df.boxplot()
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

Equal-frequency binning and Integer encodig

In [47]:

```
#displaying A2 column

df['A2']
```

Out[47]:

```
0      30.83
1      58.67
2      24.50
3      27.83
4      20.17
       ...  
685    21.08
686    22.67
687    25.25
688    17.92
689    35.00
Name: A2, Length: 690, dtype: object
```

A2 is a object datatype, but its ahving numerical values in its column . Therefore change the dataype of A2 from objecct to numeric

In [48]:

```
# Change data type of column

df['A2']= pd.to_numeric(df.A2, errors='coerce')

df["A2"]
```

Out[48]:

```
0      30.83
1      58.67
2      24.50
3      27.83
4      20.17
       ...
685    21.08
686    22.67
687    25.25
688    17.92
689    35.00
Name: A2, Length: 690, dtype: float64
```

In [49]:

```
#checking the datatype
df.dtypes
```

Out[49]:

```
A1      object
A2     float64
A3     float64
A4      object
A5      object
A6      object
A7      object
A8     float64
A9      object
A10     object
A11      int64
A12     object
A13     object
A14     object
A15      int64
A16     object
dtype: object
```

Now, we see that the dattype of A2 is changed to numeric from object

In [50]:

```
df
```

Out[50]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	b	21.08	10.085	y	p	e	h	1.25	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.00	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.00	f	t	1	t	g	00200	1	-
688	b	17.92	0.205	u	g	aa	v	0.04	f	f	0	f	g	00280	750	-
689	b	35.00	3.375	u	g	c	h	8.29	f	f	0	t	g	00000	0	-

690 rows × 16 columns

In [51]:

```
#First, making a copy of the original dataset and give it a different name.


df_cat = df.copy()
```

In [52]:

```
#For the A2 numerical descriptive feature, discretize it via equal-frequency bin
ning with 3 bins named "low", "medium", and "high", and then use integer encodin
g for it.


df_cat['A2'] = pd.qcut( df_cat['A2'], q = 3, labels = ["low", "medium", "high"])

df_cat['A2']
```

Out[52]:

```
0      medium
1        high
2      medium
3      medium
4         low
        ...
685       low
686       low
687    medium
688       low
689      high
Name: A2, Length: 690, dtype: category
Categories (3, object): [low < medium < high]
```

In [53]:

```
#performed the dicretization correctly using the value_counts method

df['A2'] .value_counts()
```

Out[53]:

```
28.46    12
22.67     9
20.42     7
24.50     6
20.67     6
         ..
17.83     1
44.83     1
60.58     1
50.08     1
28.33     1
Name: A2, Length: 350, dtype: int64
```

Performing Integer encoding and assigning low with 0, medium with 1 and high with 2

In [54]:

```
#Integer Encoding

level_mapping = {'low': 0, 'medium': 1, 'high': 2}
```

In [55]:

```
#copying df_cat to df

df = df_cat.copy()
```

In [56]:

```
#performing the integer-encoding using the replace() function. After the encodin
g, we notice that the "A2" feature is now of integer data type.

df['A2'] = df['A2'].replace(level_mapping)
```

In [57]:

```
#dislaying first 5 rows of the datset df
df.head(5)
```

Out[57]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16
0	b	1	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	2	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	1	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	1	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	0	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+

In [58]:

```
df['A2'].dtype
```

Out[58]:

```
dtype('int64')
```

In [59]:

```
#checking value counts in A2
df['A2'] .value_counts()
```

Out[59]:

```
1    231
0    230
2    229
Name: A2, dtype: int64
```

we can see that low is replaced with 0, medium - 1 and high with 2

In [60]:

```
df.dtypes
```

Out[60]:

```
A1      object
A2       int64
A3     float64
A4      object
A5      object
A6      object
A7      object
A8     float64
A9      object
A10     object
A11      int64
A12     object
A13     object
A14     object
A15      int64
A16     object
dtype: object
```

In [61]:

```
#changing A14 to numeric from object datatype
df['A14']= pd.to_numeric(df.A14, errors='coerce')

df["A14"]
```

Out[61]:

```
0      202.0
1       43.0
2      280.0
3      100.0
4      120.0
       ...  
685    260.0
686    200.0
687    200.0
688    280.0
689      0.0
Name: A14, Length: 690, dtype: float64
```

Encoding The Target Feature

In [62]:

```
#dropping the last column(A16) in the dataframe and assigning to a target variab
le

Data = df.drop(columns = 'A16').values

#seperating A16 column from df and storing in target
target = df['A16']
```

Now target is our A16 column

In [63]:

```
# counting the number of instances each label has in the target feature in the d
f dataset.
np.unique(target, return_counts=True)
```

Out[63]:

```
(array(['+', '-'], dtype=object), array([307, 383]))
```

LabelEncoder labels in an alphabetical order. That is, "+" is labeled as 0 whereas "-" as labeled as 1.

As expected, "+" and "-" have 307 and 383 observations respectively. Next, let's encode these as 0 and 1 using LabelEncoder from the sklearn preprocessing module.

In [64]:

```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
le_fit = le.fit(target)
target_encoded_le = le_fit.transform(target)
```

In [65]:

```
import numpy as np

print("Target Type:", type(target))

print("Counts Using NumPy:")
print(np.unique(target_encoded_le, return_counts = True))

print("Counts Using Pandas:")
print(pd.Series(target_encoded_le).value_counts())
```

```
Target Type: <class 'pandas.core.series.Series'>
Counts Using NumPy:
(array([0, 1]), array([307, 383]))
Counts Using Pandas:
1    383
0    307
dtype: int64
```

In taget (A16), values of + and - are replaced by 0 and 1 respectively with 383 counts and 307 counts respectively

One-Hot-Encoding

In [66]:

```
# Selecting features of columns A1 - A15 from the dataset df

X = df.iloc[:,0:15]
X
```

Out[66]:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
0	b	1	0.000	u	g	w	v	1.25	t	t	1	f	g	202.0	0
1	a	2	4.460	u	g	q	h	3.04	t	t	6	f	g	43.0	560
2	a	1	0.500	u	g	q	h	1.50	t	f	0	f	g	280.0	824
3	b	1	1.540	u	g	w	v	3.75	t	t	5	t	g	100.0	3
4	b	0	5.625	u	g	w	v	1.71	t	f	0	f	s	120.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	b	0	10.085	y	p	e	h	1.25	f	f	0	f	g	260.0	0
686	a	0	0.750	u	g	c	v	2.00	f	t	2	t	g	200.0	394
687	a	1	13.500	y	p	ff	ff	2.00	f	t	1	t	g	200.0	1
688	b	0	0.205	u	g	aa	v	0.04	f	f	0	f	g	280.0	750
689	b	2	3.375	u	g	c	h	8.29	f	f	0	t	g	0.0	0

690 rows × 15 columns

In [67]:

```
# get the list of categorical descriptive features
X_cat = X.columns[X.dtypes==object].tolist()
X_cat
```

Out[67]:

```
['A1', 'A4', 'A5', 'A6', 'A7', 'A9', 'A10', 'A12', 'A13']
```

using one-hot-encoding for encoding all the categorical descriptive features(A1,A4,A5,A6,A7,A9,A10,A12 and A13) in the dataset

In [68]:

```
# if a categorical descriptive feature has only 2 levels,
for col in X_cat:
    n = len(X[col].unique())
    if (n == 2):
        X[col] = pd.get_dummies(X[col], drop_first=True)
```

In [69]:

```
# for other categorical features (with > 2 levels), using regular one-hot-encodi
ng
# if a feature is numeric, it will be untouched
X = pd.get_dummies(X)
X.head()
```

Out[69]:

	A1	A2	A3	A8	A9	A10	A11	A12	A14	A15	...	A7_ff	A7_h	A7_j	A7_n	A7_o	A
0	1	1	0.000	1.25	1	1	1	0	202.0	0	...	0	0	0	0	0	
1	0	2	4.460	3.04	1	1	6	0	43.0	560	...	0	1	0	0	0	
2	0	1	0.500	1.50	1	0	0	0	280.0	824	...	0	1	0	0	0	
3	1	1	1.540	3.75	1	1	5	1	100.0	3	...	0	0	0	0	0	
4	1	0	5.625	1.71	1	0	0	0	120.0	0	...	0	0	0	0	0	

5 rows × 42 columns

Now we can see that new column names are added for all categorical descriptive features (A1,A4,A5,A6,A7,A9,A10,A12 and A13)

In [70]:

```
#checking column names of X, from A1 to A15 after one hot encoding
X.columns
```

Out[70]:

```
Index(['A1', 'A2', 'A3', 'A8', 'A9', 'A10', 'A11', 'A12', 'A14', 'A1
5', 'A4_l',
       'A4_u', 'A4_y', 'A5_g', 'A5_gg', 'A5_p', 'A6_aa', 'A6_c', 'A6
_cc',
       'A6_d', 'A6_e', 'A6_ff', 'A6_i', 'A6_j', 'A6_k', 'A6_m', 'A6_
q', 'A6_r',
       'A6_w', 'A6_x', 'A7_bb', 'A7_dd', 'A7_ff', 'A7_h', 'A7_j', 'A
7_n',
       'A7_o', 'A7_v', 'A7_z', 'A13_g', 'A13_p', 'A13_s'],
      dtype='object')
```

In [71]:

```
#checking the datatypes of new columns of X dataset
X.dtypes
```

Out[71]:

```
A1          uint8
A2          int64
A3        float64
A8        float64
A9          uint8
A10         uint8
A11         int64
A12         uint8
A14       float64
A15         int64
A4_l        uint8
A4_u        uint8
A4_y        uint8
A5_g        uint8
A5_gg       uint8
A5_p        uint8
A6_aa       uint8
A6_c        uint8
A6_cc       uint8
A6_d        uint8
A6_e        uint8
A6_ff       uint8
A6_i        uint8
A6_j        uint8
A6_k        uint8
A6_m        uint8
A6_q        uint8
A6_r        uint8
A6_w        uint8
A6_x        uint8
A7_bb       uint8
A7_dd       uint8
A7_ff       uint8
A7_h        uint8
A7_j        uint8
A7_n        uint8
A7_o        uint8
A7_v        uint8
A7_z        uint8
A13_g       uint8
A13_p       uint8
A13_s       uint8
dtype: object
```

Standard Scaling the Descriptive Features using the preprocessing module in sklearn

In [72]:

```
from sklearn import preprocessing

#applying standard scaling to X dataset
X_std = preprocessing.StandardScaler().fit_transform(X)
```

In [73]:

```
#roundig the 2 decimal places in X_std dataset
#X_std = X_std.round(2)
```

In [74]:

```
#displaying values of X_std
pd.DataFrame(X_std).head()
```

Out[74]:

	0	1	2	3	4	5	6	7	
0	0.661438	0.001777	-0.956613	-0.291083	0.95465	1.157144	-0.288101	-0.919195	0.1071
1	-1.511858	1.227857	-0.060051	0.244190	0.95465	1.157144	0.740830	-0.919195	-0.8169
2	-1.511858	0.001777	-0.856102	-0.216324	0.95465	-0.864196	-0.493887	-0.919195	0.5604
3	0.661438	0.001777	-0.647038	0.456505	0.95465	1.157144	0.535044	1.087908	-0.4856
4	0.661438	-1.224303	0.174141	-0.153526	0.95465	-0.864196	-0.493887	-0.919195	-0.3694

5 rows × 42 columns

Now we can see that standard scaling is applied and all the values are changed in the dataset

In [75]:

```
#checking X column names (one hot encoded colmn nmaes are displayed)
X.columns
```

Out[75]:

```
Index(['A1', 'A2', 'A3', 'A8', 'A9', 'A10', 'A11', 'A12', 'A14', 'A1
5', 'A4_l',
       'A4_u', 'A4_y', 'A5_g', 'A5_gg', 'A5_p', 'A6_aa', 'A6_c', 'A6
_cc',
       'A6_d', 'A6_e', 'A6_ff', 'A6_i', 'A6_j', 'A6_k', 'A6_m', 'A6_
q', 'A6_r',
       'A6_w', 'A6_x', 'A7_bb', 'A7_dd', 'A7_ff', 'A7_h', 'A7_j', 'A
7_n',
       'A7_o', 'A7_v', 'A7_z', 'A13_g', 'A13_p', 'A13_s'],
      dtype='object')
```

In [76]:

```
#concatenating X_std and X datasets and naming as df_clean
df_clean = pd.DataFrame(X_std,columns=X.columns)
```

In [77]:

```
#assigning the results of target_encoded_le (0's and 1's) to target column
df_clean ['target'] = target_encoded_le
```

In [78]:

```
#displaying the cleaned dataset
df_clean.head()
```

Out[78]:

	A1	A2	A3	A8	A9	A10	A11	A12	A
0	0.661438	0.001777	-0.956613	-0.291083	0.95465	1.157144	-0.288101	-0.919195	0.1071
1	-1.511858	1.227857	-0.060051	0.244190	0.95465	1.157144	0.740830	-0.919195	-0.8169
2	-1.511858	0.001777	-0.856102	-0.216324	0.95465	-0.864196	-0.493887	-0.919195	0.5604
3	0.661438	0.001777	-0.647038	0.456505	0.95465	1.157144	0.535044	1.087908	-0.4856
4	0.661438	-1.224303	0.174141	-0.153526	0.95465	-0.864196	-0.493887	-0.919195	-0.3694

5 rows × 43 columns

In [79]:

```
#checking the shape of cleaned datset
df_clean.shape
```

Out[79]:

```
(690, 43)
```

In [80]:

```
# Print summary statistics of all columns in the cleaned dataset, df_clean
df_clean.describe(include='all').round(3)
```

Out[80]:

	A1	A2	A3	A8	A9	A10	A11	A12	A14	A1
count	690.000	690.000	690.000	690.000	690.000	690.000	690.000	690.000	690.000	690.00
mean	0.000	-0.000	0.000	0.000	0.000	-0.000	0.000	-0.000	0.000	-0.00
std	1.001	1.001	1.001	1.001	1.001	1.001	1.001	1.001	1.001	1.00
min	-1.512	-1.224	-0.957	-0.665	-1.048	-0.864	-0.494	-0.919	-1.067	-0.19
25%	-1.512	-1.224	-0.756	-0.616	-1.048	-0.864	-0.494	-0.919	-0.602	-0.19
50%	0.661	0.002	-0.404	-0.366	0.955	-0.864	-0.494	-0.919	-0.137	-0.19
75%	0.661	1.228	0.492	0.120	0.955	1.157	0.123	1.088	0.514	-0.11
max	0.661	1.228	4.672	7.858	0.955	1.157	13.294	1.088	10.557	19.01

8 rows × 43 columns

In [81]:

```
#displaying the first 5 values of df_clean cleaned dataset
df_clean.head(5)
```

Out[81]:

	A1	A2	A3	A8	A9	A10	A11	A12	A
0	0.661438	0.001777	-0.956613	-0.291083	0.95465	1.157144	-0.288101	-0.919195	0.1071
1	-1.511858	1.227857	-0.060051	0.244190	0.95465	1.157144	0.740830	-0.919195	-0.816
2	-1.511858	0.001777	-0.856102	-0.216324	0.95465	-0.864196	-0.493887	-0.919195	0.560
3	0.661438	0.001777	-0.647038	0.456505	0.95465	1.157144	0.535044	1.087908	-0.485
4	0.661438	-1.224303	0.174141	-0.153526	0.95465	-0.864196	-0.493887	-0.919195	-0.369

5 rows × 43 columns

In [82]:

```
#Saving final clean dataset as "df_clean.csv".

df_clean.to_csv("/Users/harini/Downloads/df_clean.csv")
```

Question 2

In [83]:

```
#importing packages

import pandas as pd
import numpy as np



# Read the dataset
df_q2 = pd.read_csv("/Users/harini/Downloads/Asignment1_Q2.csv")
df_q2
```

Out[83]:

	COUNTRY_ID	LIFE_EXP	TOP10_INCOME	INFANT_MORT	MIL_SPEND	SCHOOL_YEARS	
0	Afghanistan	59.61	23.21	74.3	4.44	0.4	1
1	Haiti	45.00	47.67	73.1	0.09	3.4	1
2	Nigeria	51.30	38.23	82.6	1.07	4.1	2
3	Egypt	70.48	26.58	19.6	1.86	5.3	2
4	Argentina	75.77	32.30	13.3	0.76	10.1	2
5	China	74.87	29.98	13.7	1.95	6.4	3
6	Brazil	73.12	42.93	14.5	1.43	7.2	3
7	Israel	81.30	28.80	3.6	6.77	12.5	5
8	U.S.A	78.51	29.85	6.3	4.72	13.7	7
9	Ireland	80.15	27.23	3.5	0.60	11.5	
10	U.K.	80.09	28.49	4.4	2.59	13.0	7
11	Germany	80.24	22.07	3.5	1.31	12.0	8
12	Canada	80.99	24.79	4.9	1.42	14.2	8
13	Australia	82.09	25.40	4.2	1.86	11.5	8
14	Sweden	81.43	22.18	2.4	1.27	12.8	9
15	New Zealand	80.67	27.81	4.9	1.13	12.3	9
16	Russia	67.62	31.68	10.0	3.87	12.9	

In [84]:

```
#counting number of values in each column
df_q2.count()
```

Out[84]:

```
COUNTRY_ID      17
LIFE_EXP        17
TOP10_INCOME    17
INFANT_MORT     17
MIL_SPEND       17
SCHOOL_YEARS    17
CPI             17
dtype: int64
```

In [85]:

```
#checking the missing values in df
df_q2.isnull().sum()
```

Out[85]:

```
COUNTRY_ID      0
LIFE_EXP        0
TOP10_INCOME    0
INFANT_MORT     0
MIL_SPEND       0
SCHOOL_YEARS    0
CPI             0
dtype: int64
```

In [86]:

```
#checking the missing values in df
df_q2.isna().sum()
```

Out[86]:

```
COUNTRY_ID      0
LIFE_EXP        0
TOP10_INCOME    0
INFANT_MORT     0
MIL_SPEND       0
SCHOOL_YEARS    0
CPI             0
dtype: int64
```

In [87]:

```
#taking an empty list- y, looping life_exp, top10_income, infant_mort, mil_spend
columns and appending the new variable- distance to the list

y = []
for i in range(0,17):
    dist = 0
    for j in ['LIFE_EXP', 'TOP10_INCOME', 'INFANT_MORT', 'MIL_SPEND', 'SCHOOL_YE
ARS']:
        dist += abs(df_q2[j][i]-df_q2[j][16])
    y.append(dist)
df_q2['M_Distance'] = y
```

In [88]:

```
#displaying first 10 rows of the df_q2 dataset
df_q2.head(10)
```

Out[88]:

	COUNTRY_ID	LIFE_EXP	TOP10_INCOME	INFANT_MORT	MIL_SPEND	SCHOOL_YEARS	
0	Afghanistan	59.61	23.21	74.3	4.44	0.4	1.
1	Haiti	45.00	47.67	73.1	0.09	3.4	1.
2	Nigeria	51.30	38.23	82.6	1.07	4.1	2.
3	Egypt	70.48	26.58	19.6	1.86	5.3	2.
4	Argentina	75.77	32.30	13.3	0.76	10.1	2.
5	China	74.87	29.98	13.7	1.95	6.4	3.
6	Brazil	73.12	42.93	14.5	1.43	7.2	3.
7	Israel	81.30	28.80	3.6	6.77	12.5	5.
8	U.S.A	78.51	29.85	6.3	4.72	13.7	7.
9	Ireland	80.15	27.23	3.5	0.60	11.5	7

In [89]:

```
#sorting the distance by using sort_values() function
df_q2 = df_q2.sort_values( by = ['M_Distance'], ascending = True)
```

In [90]:

```
#displaying first 10 rows of the df_q2 dataset
df_q2.head(10)
```

Out[90]:

	COUNTRY_ID	LIFE_EXP	TOP10_INCOME	INFANT_MORT	MIL_SPEND	SCHOOL_YEARS	
16	Russia	67.62	31.68	10.0	3.87	12.9	
4	Argentina	75.77	32.30	13.3	0.76	10.1	2
8	U.S.A	78.51	29.85	6.3	4.72	13.7	7
5	China	74.87	29.98	13.7	1.95	6.4	3
10	U.K.	80.09	28.49	4.4	2.59	13.0	7
15	New Zealand	80.67	27.81	4.9	1.13	12.3	9
7	Israel	81.30	28.80	3.6	6.77	12.5	5
3	Egypt	70.48	26.58	19.6	1.86	5.3	2
9	Ireland	80.15	27.23	3.5	0.60	11.5	
12	Canada	80.99	24.79	4.9	1.42	14.2	8

we can see that unused value "?" is in Russia's CPI column

a. 3-nearest neighbor prediction model using Manhattan distance return for the CPI of Russia

In [91]:

```
#replacing unusual values- ? with Nan value

df_q2['CPI'] = df_q2['CPI'].replace({'?':np.nan})
```

In [92]:

```
df_q2['CPI']
```

Out[92]:

```
16       NaN
4     2.9961
8     7.1357
5     3.6356
10    7.7751
15    9.4627
7     5.8069
3     2.8622
9      7.536
12    8.6725
6     3.7741
13    8.8442
11    8.0461
14    9.2985
0     1.5171
2     2.4493
1     1.7999
Name: CPI, dtype: object
```

In [93]:

```
#changing the datatype of CPI to float
df_q2['CPI'] = df_q2['CPI'].astype('float')
```

In [94]:

```
#displaying the values of CPI
df_q2['CPI']
```

Out[94]:

```
16       NaN
4     2.9961
8     7.1357
5     3.6356
10    7.7751
15    9.4627
7     5.8069
3     2.8622
9     7.5360
12    8.6725
6     3.7741
13    8.8442
11    8.0461
14    9.2985
0     1.5171
2     2.4493
1     1.7999
Name: CPI, dtype: float64
```

for k = 3, 1st 3-nearest neighbors- [1:4] :

In [95]:

```
#calculating the 1st 3 nearest neighbors in the CPI column by using mean() funct
ion

df_q2['CPI'][1:4].mean()
```

Out[95]:

```
4.589133333333334
```

3-nearest neighbor prediction model using Manhattan distance return for the CPI of Russia is 4.589133333333334

b. weighted k-NN prediction model return for the CPI of Russia? Use k = 16

In [96]:

```
#calculating Weighted k-NN.  we know that for manhattan distance, Weights = 1/(M
anhattan Distance)^2

df_q2['Weight'] = 1/df_q2['M_Distance']**2
```

In [97]:

```
#displaying the rows of df_q2 dataset
df_q2.head(10)
```

Out[97]:

	COUNTRY_ID	LIFE_EXP	TOP10_INCOME	INFANT_MORT	MIL_SPEND	SCHOOL_YEARS	
16	Russia	67.62	31.68	10.0	3.87	12.9	
4	Argentina	75.77	32.30	13.3	0.76	10.1	2
8	U.S.A	78.51	29.85	6.3	4.72	13.7	7
5	China	74.87	29.98	13.7	1.95	6.4	3
10	U.K.	80.09	28.49	4.4	2.59	13.0	7
15	New Zealand	80.67	27.81	4.9	1.13	12.3	9
7	Israel	81.30	28.80	3.6	6.77	12.5	5
3	Egypt	70.48	26.58	19.6	1.86	5.3	2
9	Ireland	80.15	27.23	3.5	0.60	11.5	7
12	Canada	80.99	24.79	4.9	1.42	14.2	8

In [98]:

```
#multiplying CPI and Weights and storing it in CPT*WT variable
df_q2['CPI*Weight'] = df_q2['CPI']*df_q2['Weight']
```

In [99]:

```
#displaying the first 5 rows in df_q2 datset
df_q2.head(5)
```

Out[99]:

	COUNTRY_ID	LIFE_EXP	TOP10_INCOME	INFANT_MORT	MIL_SPEND	SCHOOL_YEARS	
16	Russia	67.62	31.68	10.0	3.87	12.9	
4	Argentina	75.77	32.30	13.3	0.76	10.1	2
8	U.S.A	78.51	29.85	6.3	4.72	13.7	7
5	China	74.87	29.98	13.7	1.95	6.4	3
10	U.K.	80.09	28.49	4.4	2.59	13.0	7

In [100]:

```
#checking the mean from rows 1 to 4 in CPI for 1st 16-nearest neighbors

#df_q2['CPI'][1:17].mean()
```

In [101]:

```
#we know that sum of  both CPI and weight divide by sum of weight from 1st 16 ro
ws  is our (k = 16), 16 nearest neighbors
sum(df_q2['CPI*Weight'][1:17])/sum(df_q2['Weight'][1:17])
```

Out[101]:

```
6.121484701869464
```

16-nearest neighbor prediction model using Manhattan distance return for the CPI of Russia is 6.121484701869464

In [102]:

```
#chceking the shape of the dataset
df_q2.shape
```

Out[102]:

```
(17, 10)
```

c. 3-nearest neighbor prediction model using Euclidean distance return for the CPI of Russia when the descriptive features have been normalized using range normalization

In [103]:

```
#loading the dataset again and copying to new variable, df_q2_copy

df_q2_copy = pd.read_csv('/Users/harini/Downloads/Asignment1_Q2.csv',header=0)
df_q2_copy.head()
```

Out[103]:

	COUNTRY_ID	LIFE_EXP	TOP10_INCOME	INFANT_MORT	MIL_SPEND	SCHOOL_YEARS	
0	Afghanistan	59.61	23.21	74.3	4.44	0.4	1.
1	Haiti	45.00	47.67	73.1	0.09	3.4	1.
2	Nigeria	51.30	38.23	82.6	1.07	4.1	2.
3	Egypt	70.48	26.58	19.6	1.86	5.3	2.
4	Argentina	75.77	32.30	13.3	0.76	10.1	2.

In [104]:

```
#hecking the datatypes
df_q2_copy.dtypes
```

Out[104]:

```
COUNTRY_ID       object
LIFE_EXP        float64
TOP10_INCOME    float64
INFANT_MORT     float64
MIL_SPEND       float64
SCHOOL_YEARS    float64
CPI              object
dtype: object
```

In [105]:

```
# features have been normalized using range normalization

for lst in ['LIFE_EXP','TOP10_INCOME','INFANT_MORT','MIL_SPEND','SCHOOL_YEARS']:
    e_lst = []
    for y in range(0,17):
        val = (df_q2_copy[lst][y]-df_q2_copy[lst].min())/(df_q2_copy[lst].max()-
df_q2_copy[lst].min())
        e_lst.append(val)
    df_q2_copy[lst] = e_lst
df_q2_copy.head(10)
```

Out[105]:

	COUNTRY_ID	LIFE_EXP	TOP10_INCOME	INFANT_MORT	MIL_SPEND	SCHOOL_YEARS	
0	Afghanistan	0.393907	0.044531	0.896509	0.651198	0.000000	1.
1	Haiti	0.000000	1.000000	0.881546	0.000000	0.217391	1.
2	Nigeria	0.169857	0.631250	1.000000	0.146707	0.268116	2.
3	Egypt	0.686978	0.176172	0.214464	0.264970	0.355072	2.
4	Argentina	0.829604	0.399609	0.135910	0.100299	0.702899	2.
5	China	0.805338	0.308984	0.140898	0.278443	0.434783	3.
6	Brazil	0.758156	0.814844	0.150873	0.200599	0.492754	3.
7	Israel	0.978700	0.262891	0.014963	1.000000	0.876812	5.
8	U.S.A	0.903478	0.303906	0.048628	0.693114	0.963768	7.
9	Ireland	0.947695	0.201563	0.013716	0.076347	0.804348	7

Once the normalization(scaling) is done.

The next step is to calculate Manhattan distance from it

In [106]:

```
#calculating manhattan distance to the normalized dataset df_q2_copy

lst = []
for i in range(0,17):
    distance = 0
    for j in ['LIFE_EXP','TOP10_INCOME','INFANT_MORT','MIL_SPEND','SCHOOL_YEAR
S']:
        distance += abs(df_q2_copy[j][i]-df_q2_copy[j][16])
    lst.append(distance)
df_q2_copy['M_Distance'] = lst
```

In [107]:

```
#displaying the first 17 rows from the dataset
df_q2_copy.head(17)
```

Out[107]:

	COUNTRY_ID	LIFE_EXP	TOP10_INCOME	INFANT_MORT	MIL_SPEND	SCHOOL_YEARS	
0	Afghanistan	0.393907	0.044531	0.896509	0.651198	0.000000	1
1	Haiti	0.000000	1.000000	0.881546	0.000000	0.217391	1
2	Nigeria	0.169857	0.631250	1.000000	0.146707	0.268116	2
3	Egypt	0.686978	0.176172	0.214464	0.264970	0.355072	2
4	Argentina	0.829604	0.399609	0.135910	0.100299	0.702899	2
5	China	0.805338	0.308984	0.140898	0.278443	0.434783	3
6	Brazil	0.758156	0.814844	0.150873	0.200599	0.492754	3
7	Israel	0.978700	0.262891	0.014963	1.000000	0.876812	5
8	U.S.A	0.903478	0.303906	0.048628	0.693114	0.963768	7
9	Ireland	0.947695	0.201563	0.013716	0.076347	0.804348	
10	U.K.	0.946077	0.250781	0.024938	0.374251	0.913043	7
11	Germany	0.950121	0.000000	0.013716	0.182635	0.840580	8
12	Canada	0.970342	0.106250	0.031172	0.199102	1.000000	8
13	Australia	1.000000	0.130078	0.022444	0.264970	0.804348	8
14	Sweden	0.982205	0.004297	0.000000	0.176647	0.898551	9
15	New Zealand	0.961715	0.224219	0.031172	0.155689	0.862319	9
16	Russia	0.609868	0.375391	0.094763	0.565868	0.905797	

In [108]:

```
#sorting the manhattan distance in ascending order
df_q2_copy = df_q2_copy.sort_values(by=['M_Distance'],ascending = True)
```

In [109]:

```
df_q2_copy
```

Out[109]:

	COUNTRY_ID	LIFE_EXP	TOP10_INCOME	INFANT_MORT	MIL_SPEND	SCHOOL_YEARS	
16	Russia	0.609868	0.375391	0.094763	0.565868	0.905797	
8	U.S.A	0.903478	0.303906	0.048628	0.693114	0.963768	7
10	U.K.	0.946077	0.250781	0.024938	0.374251	0.913043	7
4	Argentina	0.829604	0.399609	0.135910	0.100299	0.702899	2
15	New Zealand	0.961715	0.224219	0.031172	0.155689	0.862319	9
7	Israel	0.978700	0.262891	0.014963	1.000000	0.876812	5
5	China	0.805338	0.308984	0.140898	0.278443	0.434783	3
13	Australia	1.000000	0.130078	0.022444	0.264970	0.804348	8
12	Canada	0.970342	0.106250	0.031172	0.199102	1.000000	8
9	Ireland	0.947695	0.201563	0.013716	0.076347	0.804348	
14	Sweden	0.982205	0.004297	0.000000	0.176647	0.898551	9
11	Germany	0.950121	0.000000	0.013716	0.182635	0.840580	8
3	Egypt	0.686978	0.176172	0.214464	0.264970	0.355072	2
6	Brazil	0.758156	0.814844	0.150873	0.200599	0.492754	3
0	Afghanistan	0.393907	0.044531	0.896509	0.651198	0.000000	1
2	Nigeria	0.169857	0.631250	1.000000	0.146707	0.268116	2
1	Haiti	0.000000	1.000000	0.881546	0.000000	0.217391	1

We can see that "?" is in Russia's CPI

In [110]:

```
# replacing "?" with nan values in CPI
df_q2_copy['CPI'] = df_q2_copy['CPI'].replace({'?':np.nan})
```

```
In [111]:
```

```
#displaying the values of CPI
df_q2_copy['CPI']
```

```
Out[111]:
```

```
16       NaN
8     7.1357
10    7.7751
4     2.9961
15    9.4627
7     5.8069
5     3.6356
13    8.8442
12    8.6725
9      7.536
14    9.2985
11    8.0461
3     2.8622
6     3.7741
0     1.5171
2     2.4493
1     1.7999
Name: CPI, dtype: object
```

```
In [112]:
```

```
#changing the datatpe of CPI to float from object
df_q2_copy['CPI'] = df_q2_copy['CPI'].astype('float')
```

```
In [113]:
```

```
#checking the datatype of CPI
df_q2_copy['CPI'].dtype
```

```
Out[113]:
```

```
dtype('float64')
```

```
In [114]:
```

```
#calculating the 1st 3 nearest neighbors in the CPI column by using mean() funct
ion
df_q2_copy['CPI'][1:4].mean()
```

```
Out[114]:
```

```
5.968966666666667
```

3-nearest neighbor prediction model using Manhattan distance return for the CPI of Russia is 5.968966666666667

d. weighted k-NN prediction model—with k = 16 applied to the range-normalized data

The same activity of b) will be performed again to this question too to the range-normalized data.

In [115]:

```
#calculating the weight for manhattan distance
df_q2_copy['Weight'] = 1/df_q2_copy['M_Distance']**2
```

In [116]:

```
#calculating the CPI*Weight for manhattan distance
df_q2_copy['CPI*Weight'] = df_q2_copy['CPI']*df_q2_copy['Weight']
```

In [117]:

```
#displaying the rows of df_q2_copy datset
df_q2_copy.head()
```

Out[117]:

	COUNTRY_ID	LIFE_EXP	TOP10_INCOME	INFANT_MORT	MIL_SPEND	SCHOOL_YEARS	
16	Russia	0.609868	0.375391	0.094763	0.565868	0.905797	
8	U.S.A	0.903478	0.303906	0.048628	0.693114	0.963768	7
10	U.K.	0.946077	0.250781	0.024938	0.374251	0.913043	7
4	Argentina	0.829604	0.399609	0.135910	0.100299	0.702899	2
15	New Zealand	0.961715	0.224219	0.031172	0.155689	0.862319	9

In [118]:

```
#we know that sum of  both CPI and weight divide by sum of weight from 1st 16 ro
ws  is our (k = 16), 16 nearest neighbors
sum(df_q2_copy['CPI*Weight'][1:17])/sum(df_q2_copy['Weight'][1:17])
```

Out[118]:

```
6.609054843134554
```

16-nearest neighbor prediction model using Manhattan distance return for the CPI of Russia is 6.609054843134554 in the range normalized data

e. Which of the predictions made was the most accurate? Why do you think this was?

Range Normalized data (Normalized knn) predictions are most accurate.

Because, Normalized KNN is far better as the data is distributed and also data is most accurate and volume of training set is very less.

Before range-normization, 3-nearest neighbor prediction model using Manhattan distance return for the CPI of Russia is 4.589133333333334 and for 16-nearest neighbor prediction model using Manhattan distance return for the CPI of Russia is 6.121484701869464

After range normalization, 3-nearest neighbor prediction model using Manhattan distance return for the CPI of Russia is 5.968966666666667 and for 16-nearest neighbor prediction model using Manhattan distance return for the CPI of Russia is 6.609054843134554