PRACTICAL 03

Create a dataframe having at least 3 columns and 50 rows to store a numeric data generated by using random function Replace 10% of the values by null value whose indexex position are generated using random function. Do the following :

In [1]:
```
# importing numpy library
# importing pandas library
import numpy as np
import pandas as pd
```

a. Identify and count missing values in a dataframe

In [2]:
```
# creating a dataframe with 3 columns and 50 rows
df=pd.DataFrame(np.random.randint(0,100,size=(50,3)),
 columns=['col1','col2','col3'])
# Replacing 10% of the values with NaN
df=df.mask(np.random.random(df.shape)<.1)
#Identifying and count missing values in dataframe
missing_values=df.isnull().sum()
print('Missing Value in DataFrame is\n',missing_values)
print('Total Missing value ',missing_values.sum())
```

```
Missing Value in DataFrame is
 col1    4
col2    5
col3    6
dtype: int64
Total Missing value  15
```

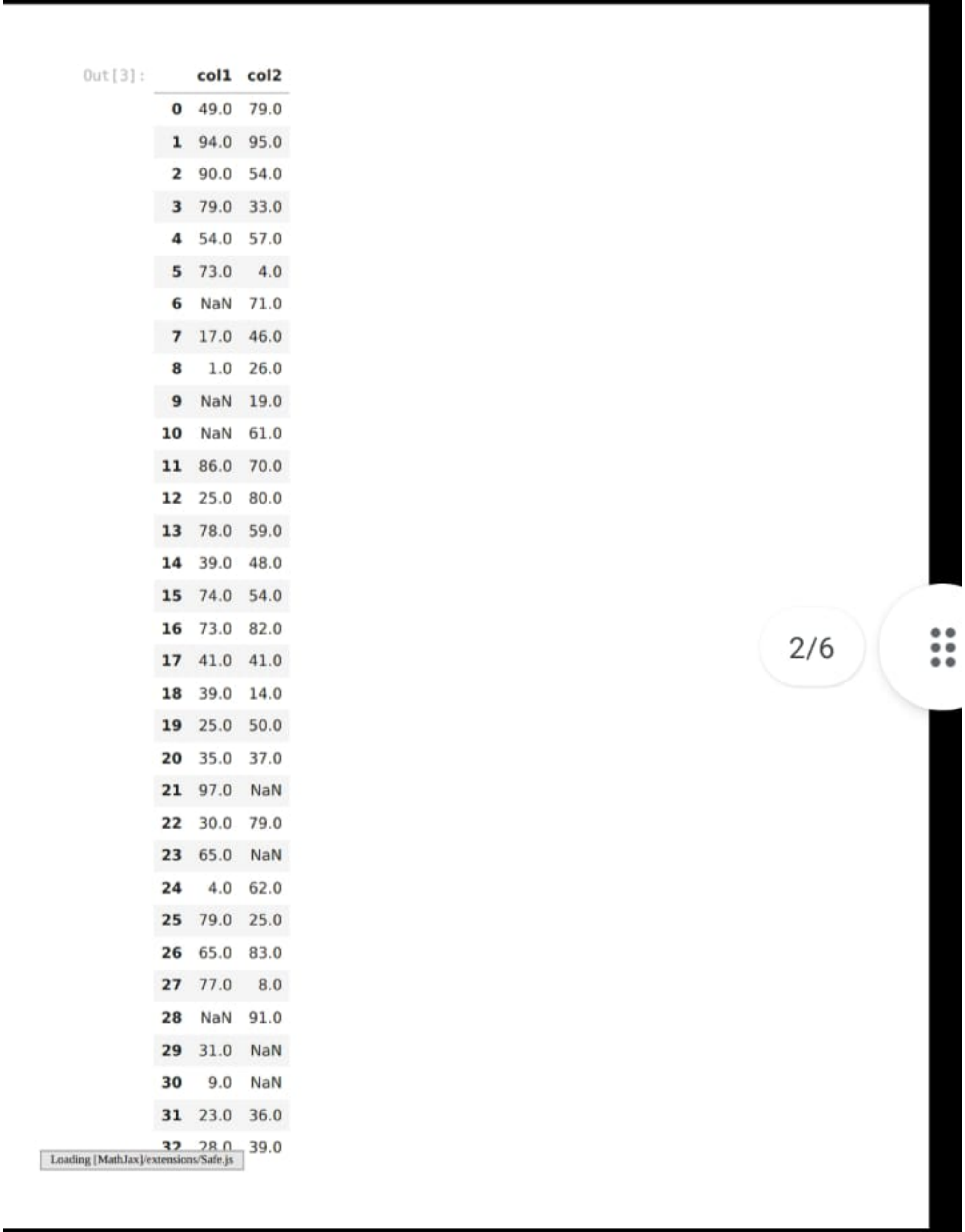
b. Drop the column having more than 5 null values

In [3]:
```
 df = df.dropna(thresh=len(df)-5, axis=1)
 df
```

Loading [MathJax]/extensions/Safe.js

Out[3]:

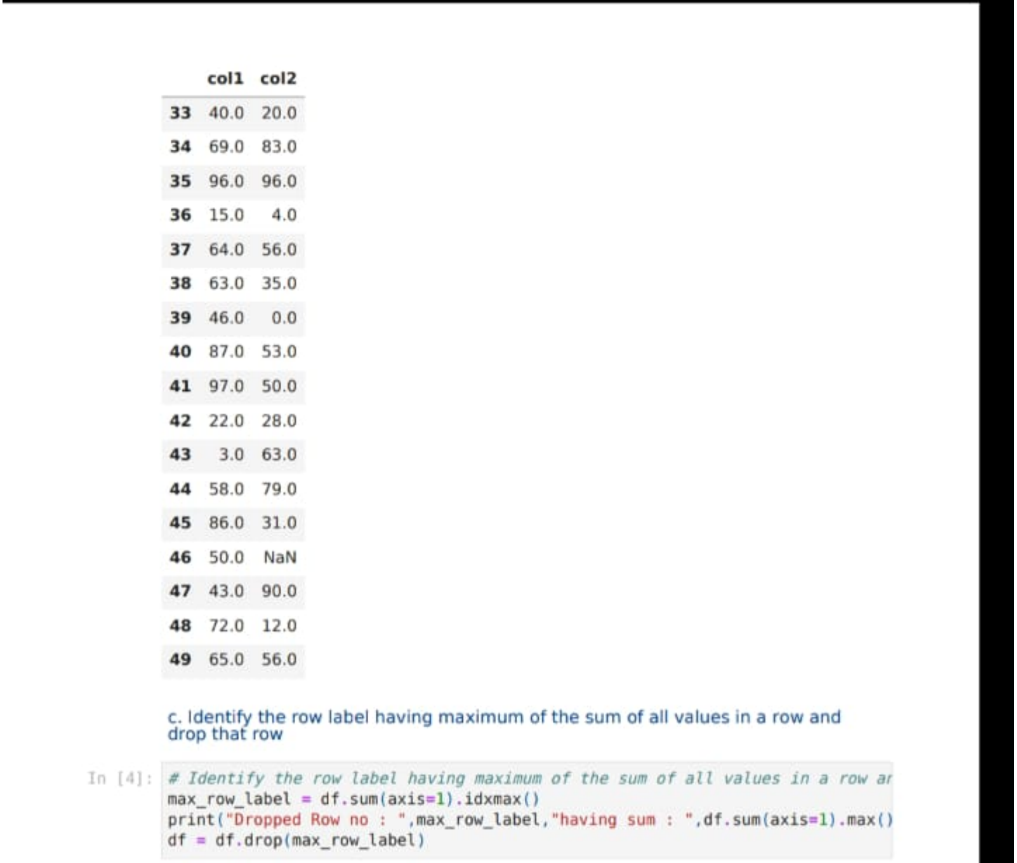
	col1	col2
0	49.0	79.0
1	94.0	95.0
2	90.0	54.0
3	79.0	33.0
4	54.0	57.0
5	73.0	4.0
6	NaN	71.0
7	17.0	46.0
8	1.0	26.0
9	NaN	19.0
10	NaN	61.0
11	86.0	70.0



Out[3]:

	col1	col2
0	49.0	79.0
1	94.0	95.0
2	90.0	54.0
3	79.0	33.0
4	54.0	57.0
5	73.0	4.0
6	NaN	71.0
7	17.0	46.0
8	1.0	26.0
9	NaN	19.0
10	NaN	61.0
11	86.0	70.0
12	25.0	80.0
13	78.0	59.0
14	39.0	48.0
15	74.0	54.0
16	73.0	82.0
17	41.0	41.0
18	39.0	14.0
19	25.0	50.0
20	35.0	37.0
21	97.0	NaN
22	30.0	79.0
23	65.0	NaN
24	4.0	62.0
25	79.0	25.0
26	65.0	83.0
27	77.0	8.0
28	NaN	91.0
29	31.0	NaN
30	9.0	NaN
31	23.0	36.0
32	28.0	39.0

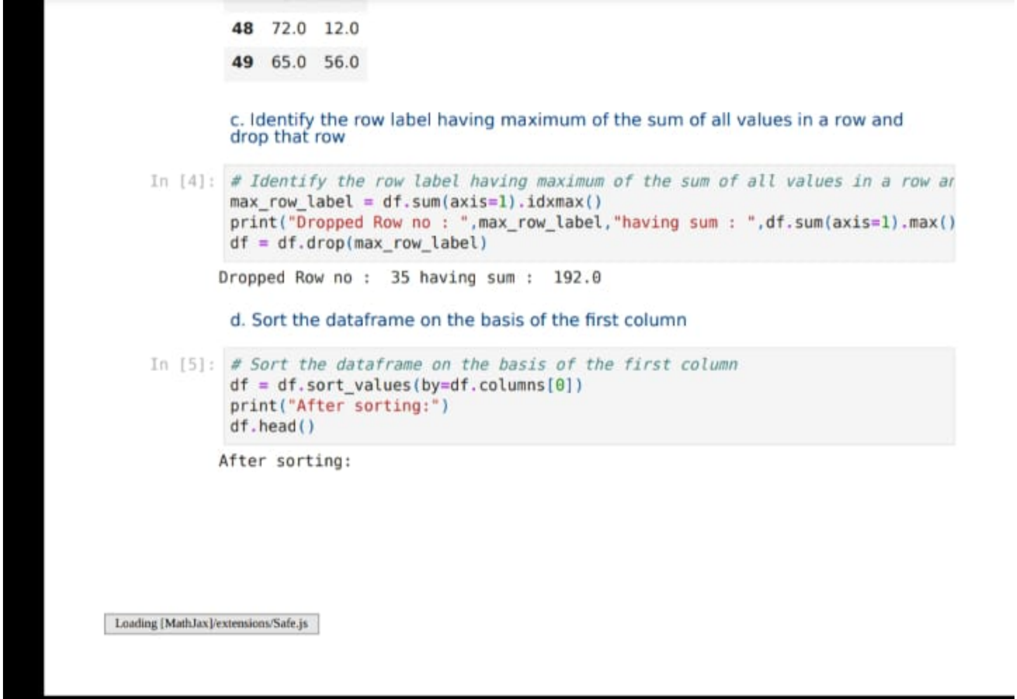
Loading [MathJax]/extensions/Safe.js



	col1	col2
33	40.0	20.0
34	69.0	83.0
35	96.0	96.0
36	15.0	4.0
37	64.0	56.0
38	63.0	35.0
39	46.0	0.0
40	87.0	53.0
41	97.0	50.0
42	22.0	28.0
43	3.0	63.0
44	58.0	79.0
45	86.0	31.0
46	50.0	NaN
47	43.0	90.0
48	72.0	12.0
49	65.0	56.0

c. Identify the row label having maximum of the sum of all values in a row and drop that row

In [4]:
```
# Identify the row label having maximum of the sum of all values in a row ar
max_row_label = df.sum(axis=1).idxmax()
print("Dropped Row no : ",max_row_label,"having sum : ",df.sum(axis=1).max()
df = df.drop(max_row_label)
```



48	72.0	12.0
49	65.0	56.0

c. Identify the row label having maximum of the sum of all values in a row and drop that row

```
In [4]: # Identify the row label having maximum of the sum of all values in a row ar
        max_row_label = df.sum(axis=1).idxmax()
        print("Dropped Row no : ",max_row_label,"having sum : ",df.sum(axis=1).max()
        df = df.drop(max_row_label)
```

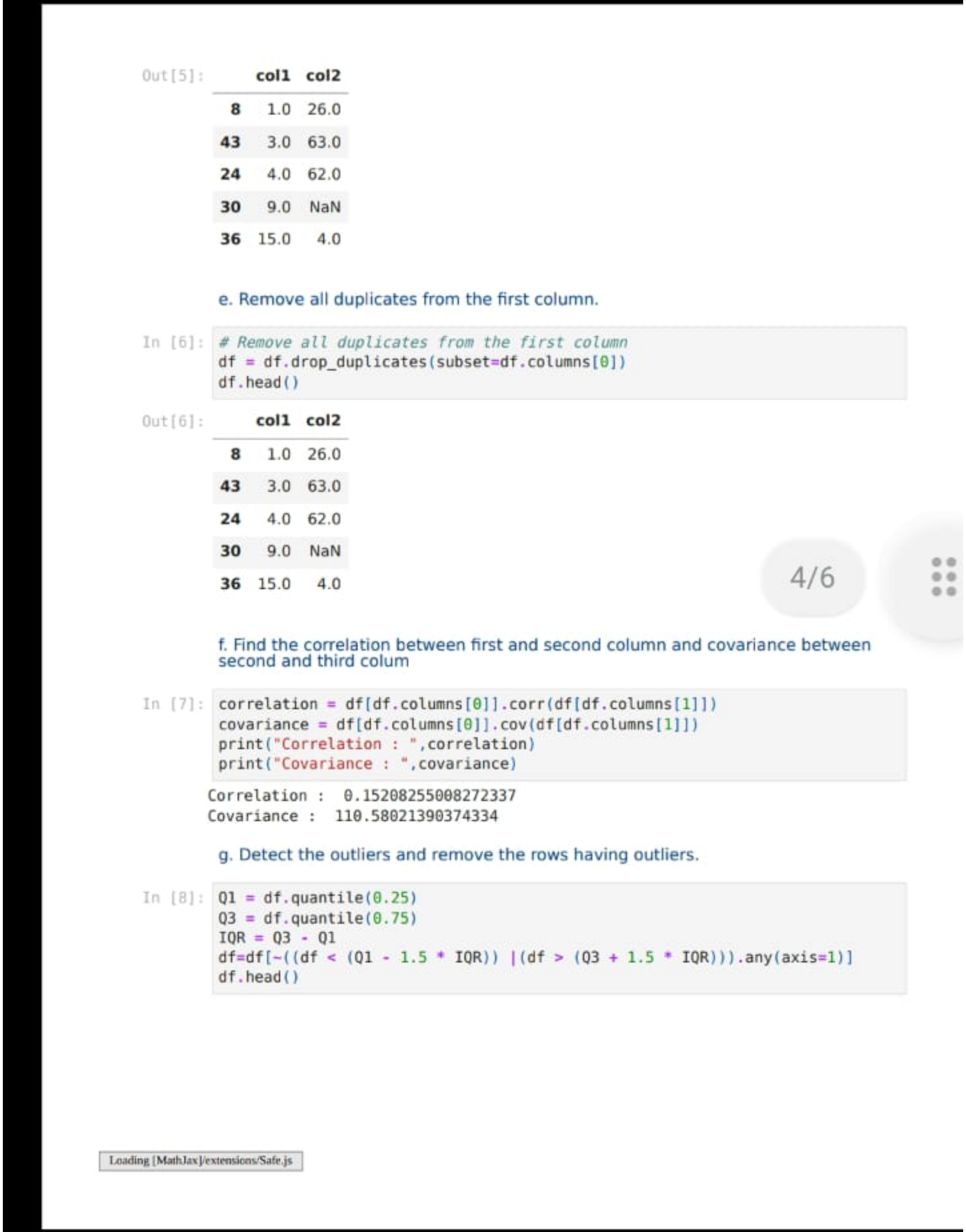
```
Dropped Row no :  35 having sum :  192.0
```

d. Sort the dataframe on the basis of the first column

```
In [5]: # Sort the dataframe on the basis of the first column
        df = df.sort_values(by=df.columns[0])
        print("After sorting:")
        df.head()
```

```
After sorting:
```

Loading [MathJax]/extensions/Safe.js



Out[5]:

	col1	col2
8	1.0	26.0
43	3.0	63.0
24	4.0	62.0
30	9.0	NaN
36	15.0	4.0

e. Remove all duplicates from the first column.

```
In [6]: # Remove all duplicates from the first column
        df = df.drop_duplicates(subset=df.columns[0])
        df.head()
```

Out[6]:

	col1	col2
8	1.0	26.0
43	3.0	63.0
24	4.0	62.0
30	9.0	NaN
36	15.0	4.0

4/6

f. Find the correlation between first and second column and covariance between second and third colum

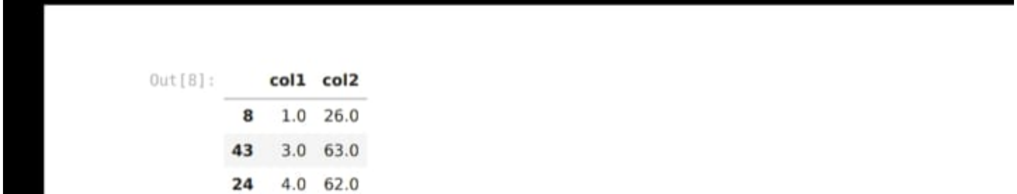
```
In [7]: correlation = df[df.columns[0]].corr(df[df.columns[1]])
        covariance = df[df.columns[0]].cov(df[df.columns[1]])
        print("Correlation : ",correlation)
        print("Covariance : ",covariance)
```

```
Correlation :  0.15208255008272337
Covariance :  110.58021390374334
```

g. Detect the outliers and remove the rows having outliers.

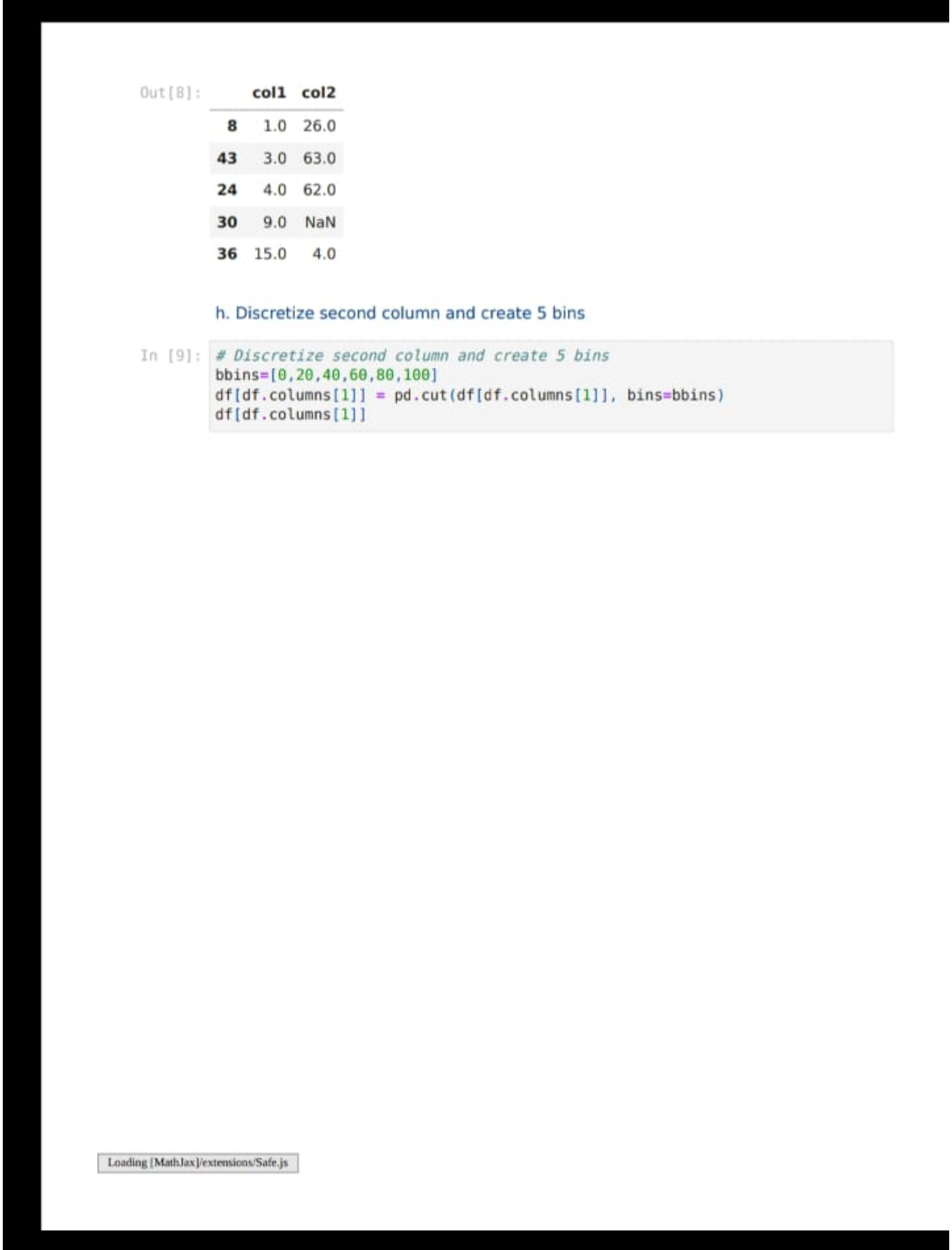
```
In [8]: Q1 = df.quantile(0.25)
        Q3 = df.quantile(0.75)
        IQR = Q3 - Q1
        df=df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
        df.head()
```

Loading [MathJax]/extensions/Safe.js



Out[8]:

	col1	col2
8	1.0	26.0
43	3.0	63.0
24	4.0	62.0



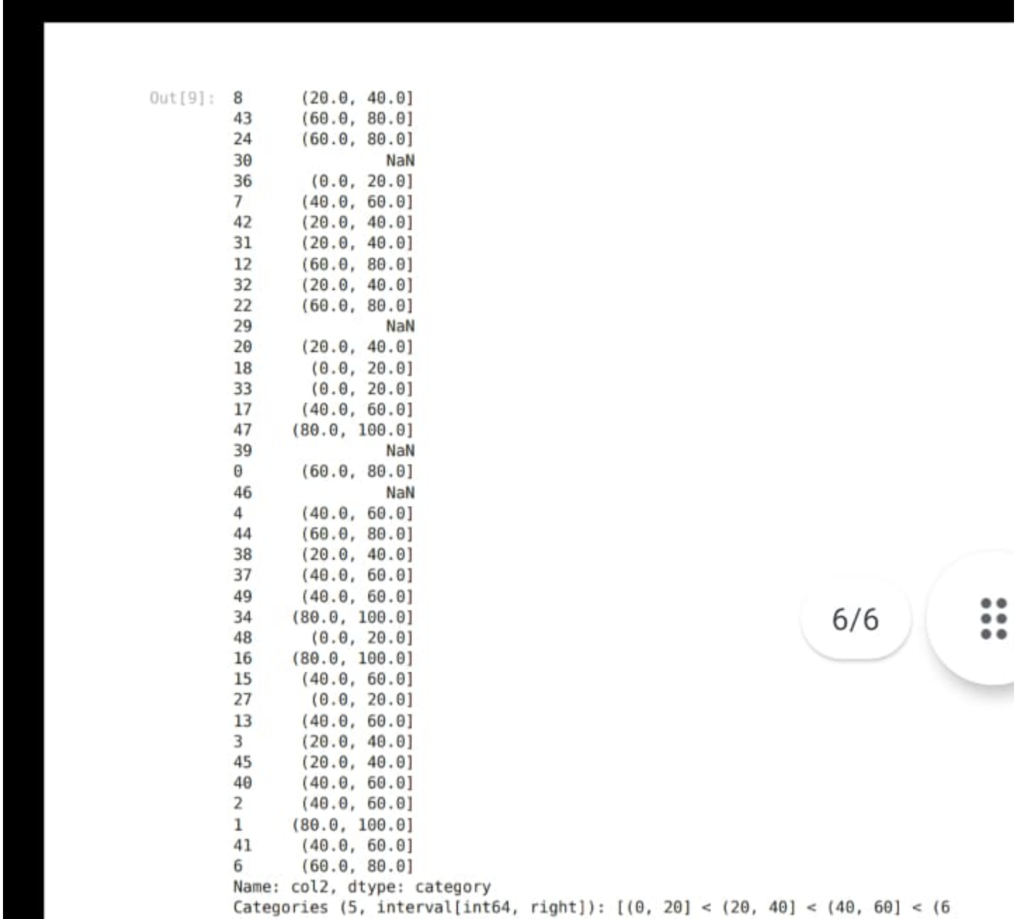
Out[8]:

	col1	col2
8	1.0	26.0
43	3.0	63.0
24	4.0	62.0
30	9.0	NaN
36	15.0	4.0

h. Discretize second column and create 5 bins

In [9]:
```
# Discretize second column and create 5 bins
bbins=[0,20,40,60,80,100]
df[df.columns[1]] = pd.cut(df[df.columns[1]], bins=bbins)
df[df.columns[1]]
```

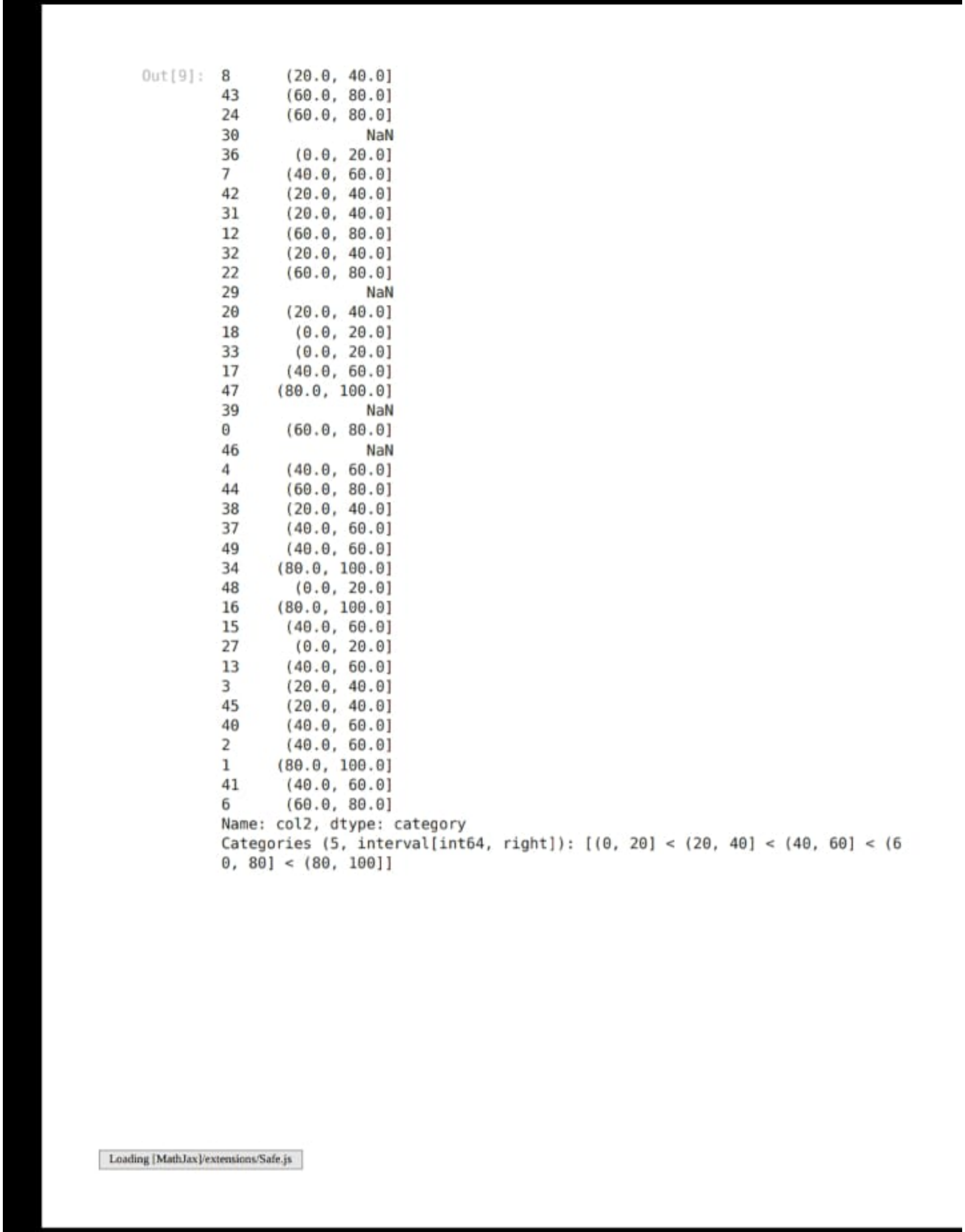
Loading [MathJax]/extensions/Safe.js



Out[9]:
```
8       (20.0, 40.0]
43      (60.0, 80.0]
24      (60.0, 80.0]
30               NaN
36       (0.0, 20.0]
7       (40.0, 60.0]
42      (20.0, 40.0]
31      (20.0, 40.0]
12      (60.0, 80.0]
32      (20.0, 40.0]
22      (60.0, 80.0]
29               NaN
20      (20.0, 40.0]
18       (0.0, 20.0]
33       (0.0, 20.0]
17      (40.0, 60.0]
47     (80.0, 100.0]
39               NaN
0       (60.0, 80.0]
46               NaN
4       (40.0, 60.0]
44      (60.0, 80.0]
38      (20.0, 40.0]
37      (40.0, 60.0]
49      (40.0, 60.0]
34     (80.0, 100.0]
48       (0.0, 20.0]
16     (80.0, 100.0]
15      (40.0, 60.0]
27       (0.0, 20.0]
13      (40.0, 60.0]
3       (20.0, 40.0]
45      (20.0, 40.0]
40      (40.0, 60.0]
2       (40.0, 60.0]
1      (80.0, 100.0]
41      (40.0, 60.0]
6       (60.0, 80.0]
Name: col2, dtype: category
Categories (5, interval[int64, right]): [(0, 20] < (20, 40] < (40, 60] < (6
```

6/6

Loading [MathJax]/extensions/Safe.js



```
Out[9]: 8       (20.0, 40.0]
        43      (60.0, 80.0]
        24      (60.0, 80.0]
        30               NaN
        36       (0.0, 20.0]
        7       (40.0, 60.0]
        42      (20.0, 40.0]
        31      (20.0, 40.0]
        12      (60.0, 80.0]
        32      (20.0, 40.0]
        22      (60.0, 80.0]
        29               NaN
        20      (20.0, 40.0]
        18       (0.0, 20.0]
        33       (0.0, 20.0]
        17      (40.0, 60.0]
        47     (80.0, 100.0]
        39               NaN
        0       (60.0, 80.0]
        46               NaN
        4       (40.0, 60.0]
        44      (60.0, 80.0]
        38      (20.0, 40.0]
        37      (40.0, 60.0]
        49      (40.0, 60.0]
        34     (80.0, 100.0]
        48       (0.0, 20.0]
        16     (80.0, 100.0]
        15      (40.0, 60.0]
        27       (0.0, 20.0]
        13      (40.0, 60.0]
        3       (20.0, 40.0]
        45      (20.0, 40.0]
        40      (40.0, 60.0]
        2       (40.0, 60.0]
        1      (80.0, 100.0]
        41      (40.0, 60.0]
        6       (60.0, 80.0]
        Name: col2, dtype: category
        Categories (5, interval[int64, right]): [(0, 20] < (20, 40] < (40, 60] < (6
        0, 80] < (80, 100]]
```

Loading [MathJax]/extensions/Safe.js