In [1]:

```
import pandas as pd
import numpy as np
```

1. Create any Series and print the output

In [2]:

```
a=pd.Series([1,2,3,4,5])
a
```

Out[2]:

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

2. Create any dataframe of 10x5 with few nan values and print the output

In [3]:

```
df=np.random.randn(10,5)
df[df<0]=np.nan
df=pd.DataFrame(df,columns=['a','b','c','d','e'])
df
```

Out[3]:

	a	b	c	d	e
0	NaN	0.052204	0.767535	NaN	NaN
1	NaN	NaN	0.198013	NaN	NaN
2	1.192508	0.598544	0.768638	0.175535	1.519276
3	0.753986	NaN	0.748025	0.941423	NaN
4	0.864428	NaN	NaN	NaN	1.539462
5	0.230474	0.863690	NaN	0.353322	NaN
6	NaN	NaN	0.396256	0.836536	NaN
7	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	1.786162	0.783210
9	NaN	1.646917	0.554242	NaN	NaN

3.Display top 7 and last 6 rows and print the output

In [10]:

```
df.head(7)
```

Out[10]:

	a	b	c	d	e
0	NaN	NaN	NaN	0.091634	0.713894
1	NaN	NaN	0.563225	NaN	NaN
2	NaN	NaN	NaN	NaN	1.328040
3	2.119494	NaN	NaN	NaN	NaN
4	0.733538	0.485432	NaN	NaN	0.102413
5	NaN	0.139734	NaN	1.243937	NaN
6	0.233309	NaN	NaN	0.551457	0.779743

In [11]:

```
df.tail(6)
```

Out[11]:

	a	b	c	d	e
4	0.733538	0.485432	NaN	NaN	0.102413
5	NaN	0.139734	NaN	1.243937	NaN
6	0.233309	NaN	NaN	0.551457	0.779743
7	0.191300	0.242987	0.204863	NaN	0.066691
8	2.047426	NaN	NaN	0.004437	1.035493
9	1.114153	0.258183	0.478120	1.226438	NaN

4. Fill with a constant value and print the output

In [15]:

```
df_fill= df.fillna(1)
df_fill
```

Out[15]:

	a	b	c	d	e
0	1.000000	1.000000	1.000000	0.091634	0.713894
1	1.000000	1.000000	0.563225	1.000000	1.000000
2	1.000000	1.000000	1.000000	1.000000	1.328040
3	2.119494	1.000000	1.000000	1.000000	1.000000
4	0.733538	0.485432	1.000000	1.000000	0.102413
5	1.000000	0.139734	1.000000	1.243937	1.000000
6	0.233309	1.000000	1.000000	0.551457	0.779743
7	0.191300	0.242987	0.204863	1.000000	0.066691
8	2.047426	1.000000	1.000000	0.004437	1.035493
9	1.114153	0.258183	0.478120	1.226438	1.000000

5. Drop the column with missing values and print the output

In [18]:

```
df.dropna(axis=1)
```

Out[18]:

0
1
2
3
4
5
6
7
8
9

6. Drop the row with missing values and print the output

In [19]:

```
df.dropna(axis=0)
```

Out[19]:

a	b	c	d	e

7. To check the presence of missing values in your dataframe

In [21]:

```
missing = df.isnull()
missing
```

Out[21]:

	a	b	c	d	e
0	True	True	True	False	False
1	True	True	False	True	True
2	True	True	True	True	False
3	False	True	True	True	True
4	False	False	True	True	False
5	True	False	True	False	True
6	False	True	True	False	False
7	False	False	False	True	False
8	False	True	True	False	False
9	False	False	False	False	True

8. Use operators and check the condition and print the output

In [22]:

```
has_missing_values = df.isnull().any().any()
if has_missing_values:
    print("\nDataFrame contains missing values (NaN).")
else:
    print("\nDataFrame does not contain any missing values (NaN).")
```

```
DataFrame contains missing values (NaN).
```

9. Display your output using loc and iloc, row and column heading

In [23]:

```
print(df.loc[:4, :'Column_C'])
```

```
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4]
```

In [24]:

```
print(df.iloc[:5, :3])
```

```
          a         b         c
0       NaN       NaN       NaN
1       NaN       NaN  0.563225
2       NaN       NaN       NaN
3  2.119494       NaN       NaN
4  0.733538  0.485432       NaN
```

10. Display the statistical summary of data

In [4]:

```
df.describe()
```

Out[4]:

	a	b	c	d	e
count	4.000000	4.000000	6.000000	5.000000	3.000000
mean	0.760349	0.790339	0.572118	0.818596	1.280650
std	0.399333	0.663510	0.236200	0.628808	0.430913
min	0.230474	0.052204	0.198013	0.175535	0.783210
25%	0.623108	0.461959	0.435753	0.353322	1.151243
50%	0.809207	0.731117	0.651133	0.836536	1.519276
75%	0.946448	1.059497	0.762657	0.941423	1.529369
max	1.192508	1.646917	0.768638	1.786162	1.539462

MINI-PROJECT 1 :

Analyse using two given datasets and perform basic analysis using numpy and pandas

a)Import library

b)Import dataset

c)head

d)tail

e)describe

f)shape

g)size

h)find missing values

i)fill/drop

In [5]:

```
df=pd.read_csv("fiat500_VehicleSelection_Dataset.csv")
df
```

Out[5]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.611
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.24
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	1
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.63
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.49
...	...	...	...	...	...	...	...	
1544	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1546	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nul
1547	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

1549 rows × 11 columns

In [6]:

```
df.head(4)
```

Out[6]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	l
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.6115598
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.241889
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11.417
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.634609

In [7]:

```
df.tail()
```

Out[7]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
1544	NaN	NaN	NaN	NaN	NaN	NaN	NaN	length	5
1545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	concat	lonprice
1546	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Null values	NO
1547	NaN	NaN	NaN	NaN	NaN	NaN	NaN	find	1
1548	NaN	NaN	NaN	NaN	NaN	NaN	NaN	search	1

In [8]:

```
df.describe()
```

Out[8]:

	ID	engine_power	age_in_days	km	previous_owners	l
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.00000
mean	769.500000	51.904421	1650.980494	53396.011704	1.123537	43.54136
std	444.126671	3.988023	1289.522278	40046.830723	0.416423	2.13351
min	1.000000	51.000000	366.000000	1232.000000	1.000000	36.85583
25%	385.250000	51.000000	670.000000	20006.250000	1.000000	41.80299
50%	769.500000	51.000000	1035.000000	39031.000000	1.000000	44.39409
75%	1153.750000	51.000000	2616.000000	79667.750000	1.000000	45.46796
max	1538.000000	77.000000	4658.000000	235000.000000	4.000000	46.79561

In [9]:

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

Out[9]:

```
ID                    11
model                 11
engine_power          11
age_in_days           11
km                    11
previous_owners       11
lat                   11
lon                    0
price                  0
Unnamed: 9          1549
Unnamed: 10         1548
dtype: int64
```

In [10]:

```
df.dropna()
```

Out[10]:

ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price	Unnamed: 9	U

In [11]:

```
df.fillna(0)
```

Out[11]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	
0	1.0	lounge	51.0	882.0	25000.0	1.0	44.907242	8.6115
1	2.0	pop	51.0	1186.0	32500.0	1.0	45.666359	12.241
2	3.0	sport	74.0	4658.0	142228.0	1.0	45.503300	11.
3	4.0	lounge	51.0	2739.0	160000.0	1.0	40.633171	17.634
4	5.0	pop	73.0	3074.0	106880.0	1.0	41.903221	12.495
...	...	...	...	...	...	...	...	
1544	0.0	0	0.0	0.0	0.0	0.0	0.000000	
1545	0.0	0	0.0	0.0	0.0	0.0	0.000000	(
1546	0.0	0	0.0	0.0	0.0	0.0	0.000000	Null \
1547	0.0	0	0.0	0.0	0.0	0.0	0.000000	
1548	0.0	0	0.0	0.0	0.0	0.0	0.000000	ξ

1549 rows × 11 columns

In [12]:

```
df.shape
```

Out[12]:

```
(1549, 11)
```

In [13]:

```
df.size
```

Out[13]:

```
17039
```

MINI-PROJECT 2 :

Analyse using two given datasets and perform basic analysis using numpy and pandas

a)Import library

b)Import dataset

c)head

d)tail

e)describe

f)shape

g)size

h)find missing values

i)fill/drop

In [14]:

```
import pandas as pd
df=pd.read_csv("VE.CSV.csv")
df
```

Out[14]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563
...	...	...	...	...	...	...	...	...
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193
156	Burundi	Sub-Saharan Africa	157	2.905	0.08658	0.01530	0.41587	0.22396
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443

158 rows × 12 columns

In [15]:

```
df.head(4)
```

Out[15]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	F
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143	
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784	
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464	
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521	

In [16]:

```
df.tail()
```

Out[16]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Fr
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864	
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910	
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193	
156	Burundi	Sub-Saharan Africa	157	2.905	0.08658	0.01530	0.41587	0.22396	
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443	

In [17]:

```
df.isnull()
```

Out[17]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Fre
0	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	
153	False	False	False	False	False	False	False	False	
154	False	False	False	False	False	False	False	False	
155	False	False	False	False	False	False	False	False	
156	False	False	False	False	False	False	False	False	
157	False	False	False	False	False	False	False	False	

158 rows × 12 columns

In [18]:

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

Out[18]:

```
Country                          0
Region                           0
Happiness Rank                   0
Happiness Score                  0
Standard Error                   0
Economy (GDP per Capita)         0
Family                           0
Health (Life Expectancy)         0
Freedom                          0
Trust (Government Corruption)    0
Generosity                       0
Dystopia Residual                0
dtype: int64
```

In [19]:

```
df.describe()
```

Out[19]:

	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)	Freedom
count	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000	158.000000
mean	79.493671	5.375734	0.047885	0.846137	0.991046	0.630259	0.428615
std	45.754363	1.145010	0.017146	0.403121	0.272369	0.247078	0.150693
min	1.000000	2.839000	0.018480	0.000000	0.000000	0.000000	0.000000
25%	40.250000	4.526000	0.037268	0.545808	0.856823	0.439185	0.328330
50%	79.500000	5.232500	0.043940	0.910245	1.029510	0.696705	0.435515
75%	118.750000	6.243750	0.052300	1.158448	1.214405	0.811013	0.549092
max	158.000000	7.587000	0.136930	1.690420	1.402230	1.025250	0.669730

In [20]:

```
df.shape
```

Out[20]:

```
(158, 12)
```

In [21]:

```
df.dropna()
```

Out[21]:

	Country	Region	Happiness Rank	Happiness Score	Standard Error	Economy (GDP per Capita)	Family	Health (Life Expectancy)
0	Switzerland	Western Europe	1	7.587	0.03411	1.39651	1.34951	0.94143
1	Iceland	Western Europe	2	7.561	0.04884	1.30232	1.40223	0.94784
2	Denmark	Western Europe	3	7.527	0.03328	1.32548	1.36058	0.87464
3	Norway	Western Europe	4	7.522	0.03880	1.45900	1.33095	0.88521
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563
...	...	...	...	...	...	...	...	...
153	Rwanda	Sub-Saharan Africa	154	3.465	0.03464	0.22208	0.77370	0.42864
154	Benin	Sub-Saharan Africa	155	3.340	0.03656	0.28665	0.35386	0.31910
155	Syria	Middle East and Northern Africa	156	3.006	0.05015	0.66320	0.47489	0.72193
156	Burundi	Sub-Saharan Africa	157	2.905	0.08658	0.01530	0.41587	0.22396
157	Togo	Sub-Saharan Africa	158	2.839	0.06727	0.20868	0.13995	0.28443

158 rows × 12 columns

In [22]:

```
df.size
```

Out[22]:

1896