Create any Series and print the output

In [1]:

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

In [3]:

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

Out[3]:

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

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

In [6]:

```
a=np.random.rand(10, 5)
a[np.random.randint(0, 10, 5), np.random.randint(0, 5, 5)] = np.nan
columns = ['Col 1', 'Col 2', 'Col 3', 'Col 4', 'Col 5']
df = pd.DataFrame(a, columns=columns)
print(df)
```

```
      Col 1     Col 2     Col 3     Col 4     Col 5
0  0.910675  0.297451  0.895962  0.243356  0.629711
1  0.334462  0.363547  0.871059  0.506493       NaN
2  0.586605  0.823163  0.188622  0.448475  0.444310
3       NaN  0.660263  0.299419  0.400282  0.652786
4  0.625333  0.382358  0.447123  0.947852  0.328425
5  0.581316  0.045425       NaN  0.777970  0.832141
6  0.614330  0.765422  0.628513  0.902755  0.242338
7       NaN  0.081702  0.471967       NaN  0.511774
8  0.619253  0.497120  0.859813  0.238204  0.813465
9  0.548856  0.394432  0.457049  0.509623  0.911859
```

Display top 7 and last 6 rows and print the output

In [9]:

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

Out[9]:

	Col 1	Col 2	Col 3	Col 4	Col 5
0	0.910675	0.297451	0.895962	0.243356	0.629711
1	0.334462	0.363547	0.871059	0.506493	NaN
2	0.586605	0.823163	0.188622	0.448475	0.444310
3	NaN	0.660263	0.299419	0.400282	0.652786
4	0.625333	0.382358	0.447123	0.947852	0.328425
5	0.581316	0.045425	NaN	0.777970	0.832141
6	0.614330	0.765422	0.628513	0.902755	0.242338

In [10]:

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

Out[10]:

	Col 1	Col 2	Col 3	Col 4	Col 5
4	0.625333	0.382358	0.447123	0.947852	0.328425
5	0.581316	0.045425	NaN	0.777970	0.832141
6	0.614330	0.765422	0.628513	0.902755	0.242338
7	NaN	0.081702	0.471967	NaN	0.511774
8	0.619253	0.497120	0.859813	0.238204	0.813465
9	0.548856	0.394432	0.457049	0.509623	0.911859

Fill with a constant value and print the output

In [12]:

```
x=df
df.fillna(0)
```

Out[12]:

	Col 1	Col 2	Col 3	Col 4	Col 5
0	0.910675	0.297451	0.895962	0.243356	0.629711
1	0.334462	0.363547	0.871059	0.506493	0.000000
2	0.586605	0.823163	0.188622	0.448475	0.444310
3	0.000000	0.660263	0.299419	0.400282	0.652786
4	0.625333	0.382358	0.447123	0.947852	0.328425
5	0.581316	0.045425	0.000000	0.777970	0.832141
6	0.614330	0.765422	0.628513	0.902755	0.242338
7	0.000000	0.081702	0.471967	0.000000	0.511774
8	0.619253	0.497120	0.859813	0.238204	0.813465
9	0.548856	0.394432	0.457049	0.509623	0.911859

Drop the row with missing values and print the output

In [15]:

```
y=x
x.dropna()
```

Out[15]:

	Col 1	Col 2	Col 3	Col 4	Col 5
0	0.910675	0.297451	0.895962	0.243356	0.629711
2	0.586605	0.823163	0.188622	0.448475	0.444310
4	0.625333	0.382358	0.447123	0.947852	0.328425
6	0.614330	0.765422	0.628513	0.902755	0.242338
8	0.619253	0.497120	0.859813	0.238204	0.813465
9	0.548856	0.394432	0.457049	0.509623	0.911859

Drop the row with missing values and print the output

In [16]:

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

Out[16]:

	Col 2
0	0.297451
1	0.363547
2	0.823163
3	0.660263
4	0.382358
5	0.045425
6	0.765422
7	0.081702
8	0.497120
9	0.394432

To check the presence of missing values in your dataframe

In [20]:

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

Out[20]:

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

Use operators and check the condition and print the output

In [21]:

```
df[df>0.5]
```

Out[21]:

	Col 1	Col 2	Col 3	Col 4	Col 5
0	0.910675	NaN	0.895962	NaN	0.629711
1	NaN	NaN	0.871059	0.506493	NaN
2	0.586605	0.823163	NaN	NaN	NaN
3	NaN	0.660263	NaN	NaN	0.652786
4	0.625333	NaN	NaN	0.947852	NaN
5	0.581316	NaN	NaN	0.777970	0.832141
6	0.614330	0.765422	0.628513	0.902755	NaN
7	NaN	NaN	NaN	NaN	0.511774
8	0.619253	NaN	0.859813	NaN	0.813465
9	0.548856	NaN	NaN	0.509623	0.911859

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

In [22]:

```
df.loc[0:3]
```

Out[22]:

	Col 1	Col 2	Col 3	Col 4	Col 5
0	0.910675	0.297451	0.895962	0.243356	0.629711
1	0.334462	0.363547	0.871059	0.506493	NaN
2	0.586605	0.823163	0.188622	0.448475	0.444310
3	NaN	0.660263	0.299419	0.400282	0.652786

In [23]:

```
df.iloc[0:4]
```

Out[23]:

	Col 1	Col 2	Col 3	Col 4	Col 5
0	0.910675	0.297451	0.895962	0.243356	0.629711
1	0.334462	0.363547	0.871059	0.506493	NaN
2	0.586605	0.823163	0.188622	0.448475	0.444310
3	NaN	0.660263	0.299419	0.400282	0.652786

Display the statistical summary of data

In [24]:

```
df.describe()
```

Out[24]:

	Col 1	Col 2	Col 3	Col 4	Col 5
count	8.000000	10.000000	9.000000	9.000000	9.000000
mean	0.602604	0.431088	0.568836	0.552779	0.596312
std	0.156450	0.262325	0.259909	0.265074	0.232905
min	0.334462	0.045425	0.188622	0.238204	0.242338
25%	0.573201	0.313975	0.447123	0.400282	0.444310
50%	0.600467	0.388395	0.471967	0.506493	0.629711
75%	0.620773	0.619478	0.859813	0.777970	0.813465
max	0.910675	0.823163	0.895962	0.947852	0.911859

MINI-PROJECT

DATASET 1

In [26]:

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

Out[26]:

	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 [28]:

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

Out[28]:

	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 [29]:

```
df.tail()
```

Out[29]:

	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 [31]:

```
df.describe()
```

Out[31]:

	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 [33]:

```
df.isnull()
```

Out[33]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
1544	True	True	True	True	True	True	True	False	False
1545	True	True	True	True	True	True	True	False	False
1546	True	True	True	True	True	True	True	False	False
1547	True	True	True	True	True	True	True	False	False
1548	True	True	True	True	True	True	True	False	False

1549 rows × 11 columns

In [34]:

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

Out[34]:

```
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 [35]:

```
df.dropna()
```

Out[35]:

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

In [12]:

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

Out[12]:

	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 [38]:

```
df.shape
```

Out[38]:

```
(1549, 11)
```

In [39]:

```
df.size
```

Out[39]:

```
17039
```

DATASET 2

In [2]:

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

Out[2]:

	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 [3]:

```
df.head()
```

Out[3]:

	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	
4	Canada	North America	5	7.427	0.03553	1.32629	1.32261	0.90563	

In [4]:

```
df.tail()
```

Out[4]:

	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 [5]:

```
df.isnull()
```

Out[5]:

	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 [6]:

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

Out[6]:

```
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 [8]:

```
df.describe()
```

Out[8]:

	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 [9]:

```
df.shape
```

Out[9]:

```
(158, 12)
```

In [10]:

```
df.dropna()
```

Out[10]:

	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 [11]:

```
df.size
```

Out[11]:

```
1896
```

In [ ]:

```

```