DAY-3 Pandas

1. Create any Series and print the output

In [4]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
aa=pd.Series([1,2,3])
aa
```

Out[3]:
```
0    1
1    2
2    3
dtype: int64
```

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

In [6]:
```
df=pd.DataFrame(np.random.randn(10,5))
df
```

Out[6]:

	0	1	2	3	4
0	-0.577553	-0.995943	-2.475729	0.907038	1.041748
1	-1.107139	-0.492423	-0.710511	-1.408060	-0.734869
2	0.718510	0.022415	1.378282	3.864170	0.794505
3	1.321792	0.993661	0.440777	-0.536389	-0.972916
4	0.190336	-0.429329	1.313691	-1.216513	-0.690703
5	0.009498	0.103781	-0.486682	-1.699863	-0.428188
6	-1.094483	0.545603	0.478801	0.744601	-0.960490
7	0.798164	0.533603	0.757341	0.729397	-0.211068
8	-1.075855	-0.335855	-1.655886	-0.850651	0.942511
9	0.009617	0.641107	-0.506304	-0.458219	-0.628776

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

In [7]:
```
df.head(7)
```

Out[7]:

	0	1	2	3	4
0	-0.577553	-0.995943	-2.475729	0.907038	1.041748
1	-1.107139	-0.492423	-0.710511	-1.408060	-0.734869
2	0.718510	0.022415	1.378282	3.864170	0.794505
3	1.321792	0.993661	0.440777	-0.536389	-0.972916
4	0.190336	-0.429329	1.313691	-1.216513	-0.690703
5	0.009498	0.103781	-0.486682	-1.699863	-0.428188
6	-1.094483	0.545603	0.478801	0.744601	-0.960490

In [8]:
```
df.tail(6)
```

Out[8]:

	0	1	2	3	4
4	0.190336	-0.429329	1.313691	-1.216513	-0.690703
5	0.009498	0.103781	-0.486682	-1.699863	-0.428188
6	-1.094483	0.545603	0.478801	0.744601	-0.960490
7	0.798164	0.533603	0.757341	0.729397	-0.211068
8	-1.075855	-0.335855	-1.655886	-0.850651	0.942511
9	0.009617	0.641107	-0.506304	-0.458219	-0.628776

4. Fill with a constant value and print the output

In [13]:
```
df1=pd.DataFrame(
{
    "A":1.0,
    "B":45,
    "c":pd.Series(index=list(range(4))),
}
)
df1
```

```
<ipython-input-13-b3db6660e54d>:5: DeprecationWarning: The default dtype for
empty Series will be 'object' instead of 'float64' in a future version. Speci
fy a dtype explicitly to silence this warning.
  "c":pd.Series(index=list(range(4))),
```

Out[13]:

	A	B	c
0	1.0	45	NaN
1	1.0	45	NaN
2	1.0	45	NaN
3	1.0	45	NaN

In [14]:
```
df1.fillna(value=100)
```

Out[14]:

	A	B	c
0	1.0	45	100.0
1	1.0	45	100.0
2	1.0	45	100.0
3	1.0	45	100.0

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

In [16]:
```
df1.dropna(axis=1,how='any')
```

Out[16]:

	A	B
0	1.0	45
1	1.0	45
2	1.0	45
3	1.0	45

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

In [17]:
```
d=pd.DataFrame(
{
    "A":1.0,
    "B":45,
    "C":pd.Series(index=list(range(4)))
}
)
d
```

```
<ipython-input-17-e8517b2d43bb>:5: DeprecationWarning: The default dtype for
empty Series will be 'object' instead of 'float64' in a future version. Speci
fy a dtype explicitly to silence this warning.
  "C":pd.Series(index=list(range(4)))
```

Out[17]:

	A	B	C
0	1.0	45	NaN
1	1.0	45	NaN
2	1.0	45	NaN
3	1.0	45	NaN

In [18]:
```
d.dropna()
```

Out[18]:

A	B	C

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

In [19]:
```
pd.isna(df1)
```

Out[19]:

	A	B	c
0	False	False	True
1	False	False	True
2	False	False	True
3	False	False	True

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

In [21]:
```
df1[df1["B"]>=23]
```

Out[21]:

	A	B	c
0	1.0	45	NaN
1	1.0	45	NaN
2	1.0	45	NaN
3	1.0	45	NaN

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

In [22]:
```
df1.loc[0:2]
```

Out[22]:

	A	B	c
0	1.0	45	NaN
1	1.0	45	NaN
2	1.0	45	NaN

In [23]:
```
df1.iloc[0:2]
```

Out[23]:

	A	B	c
0	1.0	45	NaN
1	1.0	45	NaN

```
In [24]: df1.index
```

```
Out[24]: Int64Index([0, 1, 2, 3], dtype='int64')
```

```
In [25]: df1.columns
```

```
Out[25]: Index(['A', 'B', 'c'], dtype='object')
```

10. Display the statistical summary of data

```
In [26]: df1.describe()
```

Out[26]:

	A	B	c
count	4.0	4.0	0.0
mean	1.0	45.0	NaN
std	0.0	0.0	NaN
min	1.0	45.0	NaN
25%	1.0	45.0	NaN
50%	1.0	45.0	NaN
75%	1.0	45.0	NaN
max	1.0	45.0	NaN

```
In [ ]:
```