DAY-3-Pandas ¶

1. Create any Series and print the output

In [3]:

```
import pandas as pd
```

In [6]:

```
import numpy as np
```

In [2]:

```
a=pd.Series([1,2,3,4,5,6])
a
```

Out[2]:

```
0    1
1    2
2    3
3    4
4    5
5    6
dtype: int64
```

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

In [10]:

```
b=pd.DataFrame(np.random.rand(10,5))
b
```

Out[10]:

	0	1	2	3	4
0	0.844187	0.284480	0.479730	0.274145	0.813674
1	0.749949	0.846854	0.944369	0.333499	0.554023
2	0.081618	0.853207	0.600953	0.877377	0.084339
3	0.092729	0.314577	0.183360	0.297615	0.931951
4	0.235653	0.119510	0.247345	0.554040	0.647371
5	0.787439	0.717846	0.150214	0.224654	0.837183
6	0.957915	0.374390	0.096364	0.702563	0.075246
7	0.656318	0.295225	0.433599	0.576759	0.489606
8	0.449529	0.968953	0.321071	0.836905	0.104911
9	0.536675	0.508257	0.071998	0.619168	0.816137

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

In [12]:

```
b.head(7)
```

Out[12]:

	0	1	2	3	4
0	0.844187	0.284480	0.479730	0.274145	0.813674
1	0.749949	0.846854	0.944369	0.333499	0.554023
2	0.081618	0.853207	0.600953	0.877377	0.084339
3	0.092729	0.314577	0.183360	0.297615	0.931951
4	0.235653	0.119510	0.247345	0.554040	0.647371
5	0.787439	0.717846	0.150214	0.224654	0.837183
6	0.957915	0.374390	0.096364	0.702563	0.075246

In [13]:

```
b.tail(6)
```

Out[13]:

	0	1	2	3	4
4	0.235653	0.119510	0.247345	0.554040	0.647371
5	0.787439	0.717846	0.150214	0.224654	0.837183
6	0.957915	0.374390	0.096364	0.702563	0.075246
7	0.656318	0.295225	0.433599	0.576759	0.489606
8	0.449529	0.968953	0.321071	0.836905	0.104911
9	0.536675	0.508257	0.071998	0.619168	0.816137

4. Fill with a constant value and print the output

In [22]:

```
df1=pd.DataFrame(
{
    "a":1.0,
    "b":45,
    "c":pd.Series(index=list(range(4)))
})
df1
```

```
<ipython-input-22-be6a5c6dc2dd>:5: DeprecationWarning: The default dtype f
or empty Series will be 'object' instead of 'float64' in a future version.
Specify a dtype explicitly to silence this warning.
  "c":pd.Series(index=list(range(4)))
```

Out[22]:

	a	b	c
0	1.0	45	NaN
1	1.0	45	NaN
2	1.0	45	NaN
3	1.0	45	NaN

In [25]:

```
df1.fillna(value=1000)
```

Out[25]:

	a	b	c
0	1.0	45	1000.0
1	1.0	45	1000.0
2	1.0	45	1000.0
3	1.0	45	1000.0

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

In [26]:

```
df1.dropna(axis=1,how='any')
```

Out[26]:

	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 [27]:

```
d=pd.DataFrame(
{
    "a":1.0,
    "b":45,
    "c":pd.Series(index=list(range(4)))
})
d
```

```
<ipython-input-27-83302b0d01d9>:5: DeprecationWarning: The default dtype f
or empty Series will be 'object' instead of 'float64' in a future version.
Specify a dtype explicitly to silence this warning.
  "c":pd.Series(index=list(range(4)))
```

Out[27]:

	a	b	c
0	1.0	45	NaN
1	1.0	45	NaN
2	1.0	45	NaN
3	1.0	45	NaN

In [29]:

```
d.dropna()
```

Out[29]:

	a	b	c

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

In [30]:

```
pd.isna(df1)
```

Out[30]:

	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 [33]:

```
df1[df1["b"]>=23]
```

Out[33]:

	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 [35]:

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

Out[35]:

	a	b	c
0	1.0	45	NaN
1	1.0	45	NaN
2	1.0	45	NaN

In [36]:

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

Out[36]:

	a	b	c
0	1.0	45	NaN
1	1.0	45	NaN

In [37]:

```
df1.index
```

Out[37]:

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

In [38]:

```
df1.columns
```

Out[38]:

```
Index(['a', 'b', 'c'], dtype='object')
```

10. Display the statistical summary of data

In [39]:

```
df1.describe()
```

Out[39]:

	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 [ ]:

```

```