```
import pandas as pd
s1 = pd.Series([1,2,3,4,5,6,7,8,9])
s1
```

```
0    1
1    2
2    3
3    4
4    5
5    6
6    7
7    8
8    9
dtype: int64
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remo
```

```
s1 = s1 + 5
s1
```

```
0     6
1     7
2     8
3     9
4    10
5    11
6    12
7    13
8    14
dtype: int64
```

```
s1 = pd.Series([1,2,3,4,5,6,7,8,9])
s2 = pd.Series([10,20,30,40,50,60,70,80,90])
```

```
s3 = s2 - s1
s3
```

```
0     9
1    18
2    27
3    36
4    45
5    54
6    63
7    72
8    81
dtype: int64
```

```
a = [10,20,30]
b = [1,2,3]
c = ['one', 'two','three']

d = pd.Series(data = a, index = c)
d
```

```
one      10
two      20
three    30
dtype: int64
```

```
e = pd.Series(a,c)
e
```

```
one      10
two      20
three    30
dtype: int64
```

```
f = pd.Series(c,b)
```

```
f = pd.Series(c,b)
```

```
f
```

```
1      one
2      two
3    three
dtype: object
```

```
import numpy as np
```

```
df3 = pd.DataFrame(np.random.randn(2,2))
```

```
df3
```

	0	1
0	-0.231088	0.557198
1	-1.074062	-1.245593

```
a = [5,4,3,2,1]
b = [10,20,30,40,50]
c = ['a','b','c','d','e']
```

```
s1 = pd.Series(a,c)
s2 = pd.Series(b,c)
```

```
s1
```

```
a    5
b    4
c    3
d    2
e    1
dtype: int64
```

```
s2
```

```
a    10
b    20
c    30
d    40
e    50
dtype: int64
```

```
df1 = pd.concat([s1,s2], axis=1)
df1
```

	0	1
a	5	10
b	4	20
c	3	30
d	2	40
e	1	50

```
df2 = pd.concat([s1,s2], axis=0)
df2
```

```
a     5
b     4
c     3
d     2
e     1
a    10
b    20
c    30
```

```
d    40
e    50
dtype: int64
```

```
col = ['1st Column','2nd Column','3rd Column','4th Column']
row = ['1st Row','2nd Row','3rd Row','4th Row']
```

```
df = pd.DataFrame(data = np.random.randn(4,4) ,index=row,columns=col)
df
```

	1st Column	2nd Column	3rd Column	4th Column
1st Row	-0.386015	0.480619	0.577173	-0.500527
2nd Row	0.746682	0.065077	-0.910529	-1.436253
3rd Row	-0.198009	-0.819249	-1.907990	0.292286
4th Row	0.678752	1.222685	0.251319	-0.877580

```
df = pd.DataFrame(data = np.random.randn(9,7))
df
```

	0	1	2	3	4	5	6
0	0.104146	-0.461835	-1.646687	-0.539209	1.574557	0.214392	1.950016
1	0.040326	1.312195	0.798537	-1.585383	1.236870	0.575894	0.926934
2	-0.005453	2.018860	1.048645	0.095882	0.351339	-1.283047	1.307240
3	-0.289372	-1.363703	-0.435930	-0.542922	-1.204901	-0.236885	-1.424151
4	1.209881	0.056989	1.311003	0.910693	-0.368672	2.582254	-0.232278
5	1.332670	1.281214	-0.138750	0.541094	0.460149	0.149889	0.558065
6	0.223138	-1.603381	0.466316	-0.824562	-0.295815	0.717483	-0.678217
7	1.226311	1.172624	-0.885811	-0.462835	1.446936	0.657902	-0.161637
8	-1.685427	-0.979780	0.042185	-0.904505	-0.051983	0.059969	-0.534669

```
df.head()
```

	0	1	2	3	4	5	6
0	0.104146	-0.461835	-1.646687	-0.539209	1.574557	0.214392	1.950016
1	0.040326	1.312195	0.798537	-1.585383	1.236870	0.575894	0.926934
2	-0.005453	2.018860	1.048645	0.095882	0.351339	-1.283047	1.307240
3	-0.289372	-1.363703	-0.435930	-0.542922	-1.204901	-0.236885	-1.424151
4	1.209881	0.056989	1.311003	0.910693	-0.368672	2.582254	-0.232278

```
df.tail()
```

	0	1	2	3	4	5	6
4	1.209881	0.056989	1.311003	0.910693	-0.368672	2.582254	-0.232278
5	1.332670	1.281214	-0.138750	0.541094	0.460149	0.149889	0.558065
6	0.223138	-1.603381	0.466316	-0.824562	-0.295815	0.717483	-0.678217
7	1.226311	1.172624	-0.885811	-0.462835	1.446936	0.657902	-0.161637
8	-1.685427	-0.979780	0.042185	-0.904505	-0.051983	0.059969	-0.534669

```
df.tail(3)
```

	0	1	2	3	4	5	6
6	0.223138	-1.603381	0.466316	-0.824562	-0.295815	0.717483	-0.678217

```
df.shape
```

```
(9, 7)
```

```
df.describe()
```

	0	1	2	3	4	5	6
count	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000
mean	0.239580	0.159243	0.062168	-0.367972	0.349831	0.381984	0.190145
std	0.949590	1.331963	0.958255	0.768432	0.936373	1.025210	1.073011
min	-1.685427	-1.603381	-1.646687	-1.585383	-1.204901	-1.283047	-1.424151
25%	-0.005453	-0.979780	-0.435930	-0.824562	-0.295815	0.059969	-0.534669
50%	0.104146	0.056989	0.042185	-0.539209	0.351339	0.214392	-0.161637
75%	1.209881	1.281214	0.798537	0.095882	1.236870	0.657902	0.926934
max	1.332670	2.018860	1.311003	0.910693	1.574557	2.582254	1.950016

```
df = pd.DataFrame([[1,2,3],[5,5,5],[7,8,9]])
df
```

	0	1	2
0	1	2	3
1	5	5	5
2	7	8	9

```
df.describe()
```

	0	1	2
count	3.000000	3.0	3.000000
mean	4.333333	5.0	5.666667
std	3.055050	3.0	3.055050
min	1.000000	2.0	3.000000
25%	3.000000	3.5	4.000000
50%	5.000000	5.0	5.000000
75%	6.000000	6.5	7.000000
max	7.000000	8.0	9.000000

```
col = ['0th Column', '1st Column','2nd Column','3rd Column','4th Column']
row = ['0th Row','1st Row','2nd Row','3rd Row','4th Row']
df = pd.DataFrame(data = np.random.randn(5,5), index = row, columns= col)
df
```

	0th Column	1st Column	2nd Column	3rd Column	4th Column
0th Row	-1.285326	-1.280802	-1.026640	1.022448	-1.222339
1st Row	0.913776	-0.438280	0.681848	-1.130522	-0.717311
2nd Row	0.868023	-0.280843	0.612143	0.943146	-1.567509
3rd Row	0.781885	0.248122	0.132217	-0.451155	-1.482056
4th Row	-0.001240	-1.576827	-0.738853	0.172198	-0.459346

```
df['5th Column'] = np.random.randn(5,1)
```

```
df
```

	0th Column	1st Column	2nd Column	3rd Column	4th Column	5th Column
0th Row	-1.285326	-1.280802	-1.026640	1.022448	-1.222339	-0.580515
1st Row	0.913776	-0.438280	0.681848	-1.130522	-0.717311	0.476923
2nd Row	0.868023	-0.280843	0.612143	0.943146	-1.567509	-0.637460
3rd Row	0.781885	0.248122	0.132217	-0.451155	-1.482056	-0.900889
4th Row	-0.001240	-1.576827	-0.738853	0.172198	-0.459346	1.130743

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

	0th Column	1st Column	2nd Column	3rd Column	4th Column	5th Column
0th Row	-1.285326	-1.280802	-1.026640	1.022448	-1.222339	-0.580515
4th Row	-0.001240	-1.576827	-0.738853	0.172198	-0.459346	1.130743

```
df.iloc[[0,2],[1,4]]
```

	1st Column	4th Column
0th Row	-1.280802	-1.222339
2nd Row	-0.280843	-1.567509

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

	0th Column	1st Column	2nd Column	3rd Column	4th Column	5th Column
0th Row	-1.285326	-1.280802	-1.026640	1.022448	-1.222339	-0.580515
1st Row	0.913776	-0.438280	0.681848	-1.130522	-0.717311	0.476923
2nd Row	0.868023	-0.280843	0.612143	0.943146	-1.567509	-0.637460

```
df.iloc[1:4 ,3:5]
```

	3rd Column	4th Column
1st Row	-1.130522	-0.717311
2nd Row	0.943146	-1.567509
3rd Row	-0.451155	-1.482056

```
row = ['Prince','Varun','Nivedita','Jaadu']
col = ['Age','Qualification','Gender','State']
friends = pd.DataFrame([[24,'B.Tech','M','Bihar'],[22,'B.Tech','M','Haryana'],[21,'B.Tech','F','UK'],[23,'M.Tech','F','Delhi
friends
```

	Age	Qualification	Gender	State
Prince	24	B.Tech	M	Bihar
Varun	22	B.Tech	M	Haryana
Nivedita	21	B.Tech	F	UK
Jaadu	23	M.Tech	F	Delhi

```
friends.loc[['Prince','Nivedita']]
```

	Age	Qualification	Gender	State
Prince	24	B.Tech	M	Bihar

```
friends.loc[['Varun','Nivedita'],['Age','Gender']]
```

	Age	Gender
Varun	22	M
Nivedita	21	F

```
friends.loc['Varun':'Jaadu']
```

	Age	Qualification	Gender	State
Varun	22	B.Tech	M	Haryana
Nivedita	21	B.Tech	F	UK
Jaadu	23	M.Tech	F	Delhi

```
friends.loc['Prince':'Nivedita','Gender':'State']
```

	Gender	State
Prince	M	Bihar
Varun	M	Haryana
Nivedita	F	UK

```
friends['Age']
```

```
Prince      24
Varun       22
Nivedita    21
Jaadu       23
Name: Age, dtype: int64
```

```
friends[['Age']]
```

	Age
Prince	24
Varun	22
Nivedita	21
Jaadu	23

```
row = ['Number','Square','Cube']
col = ['1st','2nd','3rd','4th','5th']
df = pd.DataFrame([[1,2,3,4,5],[1,4,9,16,25],[1,8,27,64,125]], index = row, columns = col)
df
```

	1st	2nd	3rd	4th	5th
Number	1	2	3	4	5
Square	1	4	9	16	25
Cube	1	8	27	64	125

```
df['3rd']
```

```
Number     3
Square     9
Cube      27
Name: 3rd, dtype: int64
```

```
df[['3rd']]
```

	3rd
Number	3
Square	9
Cube	27

```
df[['3rd','1st','5th']]
```

	3rd	1st	5th
Number	3	1	5
Square	9	1	25
Cube	27	1	125

```
df['5th - 4th'] = df['5th'] - df['4th']
df
```

	1st	2nd	3rd	4th	5th	5th - 4th
Number	1	2	3	4	5	1
Square	1	4	9	16	25	9
Cube	1	8	27	64	125	61

```
df['2nd - 1st'] = df['2nd'] - df['1st']
df
```

	1st	2nd	3rd	4th	5th	5th - 4th	2nd - 1st
Number	1	2	3	4	5	1	1
Square	1	4	9	16	25	9	3
Cube	1	8	27	64	125	61	7

```
df.iloc[0]
```

```
1st          1
2nd          2
3rd          3
4th          4
5th          5
5th - 4th    1
2nd - 1st    1
Name: Number, dtype: int64
```

```
df.iloc[2]
```

```
1st            1
2nd            8
3rd           27
4th           64
5th          125
5th - 4th     61
2nd - 1st      7
Name: Cube, dtype: int64
```

```
df.iloc[[2]]
```

	1st	2nd	3rd	4th	5th	5th - 4th	2nd - 1st
Cube	1	8	27	64	125	61	7

```
df.iloc[[2]].T
```

	Cube
1st	1
2nd	8
3rd	27
4th	64
5th	125
5th - 4th	61
2nd - 1st	7

```
df.iloc[[0,2]]
```

	1st	2nd	3rd	4th	5th	5th - 4th	2nd - 1st
Number	1	2	3	4	5	1	1
Cube	1	8	27	64	125	61	7

```
df.iloc[[0,2]].T
```

	Number	Cube
1st	1	1
2nd	2	8
3rd	3	27
4th	4	64
5th	5	125
5th - 4th	1	61
2nd - 1st	1	7

```
df
```

	1st	2nd	3rd	4th	5th	5th - 4th	2nd - 1st
Number	1	2	3	4	5	1	1
Square	1	4	9	16	25	9	3
Cube	1	8	27	64	125	61	7

```
df.drop('5th - 4th', axis = 1, inplace=True)
df
```

	1st	2nd	3rd	4th	5th	2nd - 1st
Number	1	2	3	4	5	1
Square	1	4	9	16	25	3
Cube	1	8	27	64	125	7

```
df
```

	1st	2nd	3rd	4th	5th	2nd - 1st
Number	1	2	3	4	5	1
Square	1	4	9	16	25	3
Cube	1	8	27	64	125	7

```
df.drop('2nd - 1st', axis=1,inplace=True)
df
```

	1st	2nd	3rd	4th	5th
Number	1	2	3	4	5
Square	1	4	9	16	25
Cube	1	8	27	64	125

```
df
```

	1st	2nd	3rd	4th	5th
Number	1	2	3	4	5
Square	1	4	9	16	25
Cube	1	8	27	64	125

```
df = df.T
```

```
df
```

	Number	Square	Cube
1st	1	1	1
2nd	2	4	8
3rd	3	9	27
4th	4	16	64
5th	5	25	125

```
df1 = df.transpose()
```

```
df1
```

	1st	2nd	3rd	4th	5th
Number	1	2	3	4	5
Square	1	4	9	16	25
Cube	1	8	27	64	125

```
df
```

	Number	Square	Cube
1st	1	1	1
2nd	2	4	8
3rd	3	9	27
4th	4	16	64
5th	5	25	125

```
#Creation of a data frame from scratch
df1 = pd.DataFrame({'Laptop':['Dell','Lenovo','HP','Asus','Acer','Apple'],'Rating Out of 5':[3.5,3,4,4.25,3,5]})
df1
```

Laptop Rating Out of 5

```
df2 = pd.DataFrame({'CPU':['Core i7','Core i3','Core i5','Core i9','AMD Ryzen','M1/M2'],'GPU':['GTX','GTE','GTE3080','GTE306
df2
```

	CPU	GPU
0	Core i7	GTX
1	Core i3	GTE
2	Core i5	GTE3080
3	Core i9	GTE3060
4	AMD Ryzen	GTE3040
5	M1/M2	M1/M2

```
#Concatenate / Join of a data frame through columns
df = pd.concat([df1,df2],axis=1)
df
```

	Laptop	Rating Out of 5	CPU	GPU
0	Dell	3.50	Core i7	GTX
1	Lenovo	3.00	Core i3	GTE
2	HP	4.00	Core i5	GTE3080
3	Asus	4.25	Core i9	GTE3060
4	Acer	3.00	AMD Ryzen	GTE3040
5	Apple	5.00	M1/M2	M1/M2

```
#Converting a data frame into a csv file
df.to_csv('PC_Data.csv')
```

```
df
```

	Laptop	Rating Out of 5	CPU	GPU
0	Dell	3.50	Core i7	GTX
1	Lenovo	3.00	Core i3	GTE
2	HP	4.00	Core i5	GTE3080
3	Asus	4.25	Core i9	GTE3060
4	Acer	3.00	AMD Ryzen	GTE3040
5	Apple	5.00	M1/M2	M1/M2

```
#Converting a data frame into a excel file
df.to_excel('PC_Data.xlsx')
```

Import and Read a file (csv or excel) to google colab:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remo
```

```
import pandas as pd
```

1. Copy the path of that file, you want to import and read.

Example:

path: /content/drive/MyDrive/Notes/Iris.csv

2. Use .read_csv or .read_xlsx to read that file.

```
iris = pd.read_csv('/content/drive/MyDrive/Notes/Iris.csv')
```

OR

```
path = '/content/drive/MyDrive/Notes/Iris.csv'
iris = pd.read_csv(path)
```

Returning that csv file:

```
iris
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

MORE PANDAS FUNCTIONS

.head() => The .head() method returns first five record.
=> we can also pass the value to return specific record.

```
iris.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
iris.head(3)          # It will return first 3 record.
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa

.tail() => The .tail() method returns last five records.
=> we can also pass the value to return specific record.

```
iris.tail()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

```
iris.tail(7)        # It will returns last 7 record.
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
143	144	6.8	3.2	5.9	2.3	Iris-virginica
144	145	6.7	3.3	5.7	2.5	Iris-virginica
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

.info() => The .info() method returns the information of data.

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

.describe() => The describe() method returns description of the data in the DataFrame.

```
iris.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000

```
iris.head(15)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa
10	11	5.4	3.7	1.5	0.2	Iris-setosa
11	12	4.8	3.4	1.6	0.2	Iris-setosa
12	13	4.8	3.0	1.4	0.1	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa
14	15	5.8	4.0	1.2	0.2	Iris-setosa

.iloc() method:

Using the iloc() function in python, we can easily retrieve any particular value from a row or column using index values.

```
.iloc[row,column]
.iloc[start:stop:jump,start:stop:jump]
```

Where Rows - start:stop - index start is inclusive and index stop is exclusive.

```
iris.iloc[0:5,1:4]
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm
0	5.1	3.5	1.4
1	4.9	3.0	1.4
2	4.7	3.2	1.3
3	4.6	3.1	1.5
4	5.0	3.6	1.4

Question:

We have to show rows from index 5 to 10 and columns from index 3 to last from data ' iris.csv '.

Answer:

```
iris.iloc[5:11,2:]
```

	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
5	3.9	1.7	0.4	Iris-setosa
6	3.4	1.4	0.3	Iris-setosa
7	3.4	1.5	0.2	Iris-setosa
8	2.9	1.4	0.2	Iris-setosa
9	3.1	1.5	0.1	Iris-setosa
10	3.7	1.5	0.2	Iris-setosa

.loc() method:

The loc() property is used to access a group of rows and columns by label(s) or a boolean array.

```
.loc[start:stop:jump('label1','label2',.....,'labelN')]
```

Rows : From index start to stop.

Columns: label1, label2,.....,labelN

```
.loc[start:stop:jump,'label1':'labelN']
```

Rows : From index start to stop.

Columns: From label1 to labelN.

```
.loc[[index1,index2,.....,indexN],'label1':'labelN':jump]
```

Rows : index1,index2,.....,indexN

Columns: From label1 to labelN.

It means rows have no label only indexes, while columns have labels.

```
.loc['starting_label:'stopping_label':jump, start:stop:jump]
```

Rows : From index starting_label to stopping_label.

Columns: From index start to stop.

It means columns have no label only indexes, while rows have labels.

Note:

1. One of the rows or columns must have lebels.
2. In loc() method start and stop both indexes are inclusive.

Examples:

```
row = ['Number','Square','Cube']
df = pd.DataFrame([[1,2,3,4,5],[1,4,9,16,25],[1,8,27,64,125]], index = row)
df
```

	0	1	2	3	4
Number	1	2	3	4	5
Square	1	4	9	16	25
Cube	1	8	27	64	125

```
df.loc['Number':'Cube', 0:3]
```

	0	1	2	3
Number	1	2	3	4

```
iris.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
iris.loc[0:5,("SepalLengthCm","PetalLengthCm")]
```

	SepalLengthCm	PetalLengthCm
0	5.1	1.4
1	4.9	1.4
2	4.7	1.3
3	4.6	1.5
4	5.0	1.4
5	5.4	1.7

```
iris.loc[0:5,"SepalLengthCm":"PetalLengthCm"]
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm
0	5.1	3.5	1.4
1	4.9	3.0	1.4
2	4.7	3.2	1.3
3	4.6	3.1	1.5
4	5.0	3.6	1.4
5	5.4	3.9	1.7

```
iris.loc[[1,3,15],"SepalLengthCm":"PetalLengthCm"]
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm
1	4.9	3.0	1.4
3	4.6	3.1	1.5
15	5.7	4.4	1.5

.drop()

The drop() method is used to drop rows or column.

- To drop row, set axis = 0.

```
.drop('Rows Name', axis=0)                          -If rows have labels.
.drop([rows_index1,...row_indexN], axis=0)          -If rows have indexes.
```

- To drop column, set axis = 1.

```
.drop('Columns Name', axis=1)                          -If columns have labels.
.drop([columns_index1,...,column_indexN], axis=1)      -If columns have indexes.
```

Examples:

```
iris.head(10)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

Dropping Rows:

```
iris.drop([1,2,5],axis=0)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

147 rows × 6 columns

```
iris1 = iris.drop([2,5,9],axis=0)
iris1.head(10)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa

Dropping Columns:

```
iris.drop('Species', axis=1)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2
...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3
146	147	6.3	2.5	5.0	1.9
147	148	6.5	3.0	5.2	2.0
148	149	6.2	3.4	5.4	2.3
149	150	5.9	3.0	5.1	1.8

150 rows × 5 columns

Some Basic Functions of Pandas

:

.mean() - Calculating Mean: Pandas dataframe.mean() function returns the mean of the values for the requested axis.

```
iris.mean()
```

```
<ipython-input-199-7eed97565d6e>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric
  iris.mean()
Id               75.500000
SepalLengthCm     5.843333
SepalWidthCm      3.054000
PetalLengthCm     3.758667
PetalWidthCm      1.198667
dtype: float64
```

.median() - Calculating Median: The median() method returns a Series with the median value of each column.

```
iris.median()
```

```
<ipython-input-200-7f9b4600b8ea>:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric
  iris.median()
Id               75.50
SepalLengthCm     5.80
SepalWidthCm      3.00
PetalLengthCm     4.35
PetalWidthCm      1.30
dtype: float64
```

.min() - The min() method returns the minimum of the values over the requested axis.

```
iris.min()
```

```
Id                          1
SepalLengthCm             4.3
SepalWidthCm              2.0
PetalLengthCm             1.0
PetalWidthCm              0.1
Species           Iris-setosa
dtype: object
```

.max() - The max() method returns the maximum of the values over the requested axis.

```
iris.max()
```

```
Id                          150
SepalLengthCm               7.9
SepalWidthCm                4.4
PetalLengthCm               6.9
PetalWidthCm                2.5
Species          Iris-virginica
dtype: object
```

Make a Function and Use it

1. Make a function, which calculates the required calculation.
2. Use .apply to return.

Example

Make half of requested axis:

```
def make_half(s):
  return s*0.5
iris[['SepalWidthCm','PetalWidthCm']].apply(make_half)
```

	SepalWidthCm	PetalWidthCm
0	1.75	0.10
1	1.50	0.10
2	1.60	0.10
3	1.55	0.10
4	1.80	0.10
...	...	...
145	1.50	1.15
146	1.25	0.95
147	1.50	1.00
148	1.70	1.15
149	1.50	0.90

150 rows × 2 columns

Example

Make Double:

```
def make_double(s):
  return s*2
```

```
iris[['SepalLengthCm','PetalLengthCm']].apply(make_double)
```

	SepalLengthCm	PetalLengthCm
0	10.2	2.8
1	9.8	2.8
2	9.4	2.6
3	9.2	3.0
4	10.0	2.8
...	...	...
145	13.4	10.4
146	12.6	10.0
147	13.0	10.4
148	12.4	10.8
149	11.8	10.2

150 rows × 2 columns

.value_counts()

value_counts() function returns a Series containing counts of unique values.

Example

```
iris['Species'].value_counts()
```

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

.sort_values(by='')

Pandas sort_values() function sorts a data frame in Ascending or Descending order of passed Column. It's different than the sorted Python function since it cannot sort a data frame and particular column cannot be selected.

Example

```
iris.sort_values(by='SepalLengthCm')
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
13	14	4.3	3.0	1.1	0.1	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa
38	39	4.4	3.0	1.3	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
41	42	4.5	2.3	1.3	0.3	Iris-setosa
...	...	...	...	...	...	...
122	123	7.7	2.8	6.7	2.0	Iris-virginica
118	119	7.7	2.6	6.9	2.3	Iris-virginica
117	118	7.7	3.8	6.7	2.2	Iris-virginica
135	136	7.7	3.0	6.1	2.3	Iris-virginica
131	132	7.9	3.8	6.4	2.0	Iris-virginica

150 rows × 6 columns

```
iris1 = iris.sort_values(by='SepalLengthCm')
iris1
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
13	14	4.3	3.0	1.1	0.1	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa
38	39	4.4	3.0	1.3	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
41	42	4.5	2.3	1.3	0.3	Iris-setosa
...	...	...	...	...	...	...
122	123	7.7	2.8	6.7	2.0	Iris-virginica
118	119	7.7	2.6	6.9	2.3	Iris-virginica
117	118	7.7	3.8	6.7	2.2	Iris-virginica
135	136	7.7	3.0	6.1	2.3	Iris-virginica
131	132	7.9	3.8	6.4	2.0	Iris-virginica

150 rows × 6 columns

```
iris1.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
13	14	4.3	3.0	1.1	0.1	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa
38	39	4.4	3.0	1.3	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
41	42	4.5	2.3	1.3	0.3	Iris-setosa