

What is Pandas ?

- Pandas is a Python library used for working with data sets.
- Tool source
- A fast and efficient Data-Frame object for data manipulation.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- Reading and writing data structures and different format: CSV (comma separated value), TSV (Tab separated value), txt, XML, JSON, ZIP etc.

Why use Pandas ?

- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.

What can Pandas Do ?

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns ?
- What is average value ?
- Max value ?
- Min value ?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Pandas VS NumPy

NumPy array is used for implementatin of pandas data objects.

Pandas getting started:

Import Pandas

Once Pandas is installed, import it in your applications by adding the **import** keyword:

```
In [1]: import pandas
```

Now Pandas is imported and ready to use.

Pandas as pd

- Pandas is usually imported under the **pd** alias.
- In python alias are an alternate name for referring to the same thing.

Create an alias with the **as** keyword while importing:

```
In [2]: import pandas as pd
```

Now the Pandas package can be referred to as **pd** instead of Pandas.

Checking Pandas version:

The version string is stored under **__version__** attribute.

```
In [3]: import pandas as pd
print(pd.__version__)
```

1.1.3

Pandas Data structures

Pandas can read and write three types of data Structured

- Series
- DataFrame
- Panel

Series:

- A pandas Series is like a column in a table.

Series

	apples
0	3
1	2
2	0
3	1

- It is a one-dimensional array holding data of any type.

Create Series

We can create a series in Pandas by using the `Series()` function

```
In [4]: import pandas as pd
# Create a list
a = [1,7,2]

# Create a series
series_1 = pd.Series(a)

print(series_1)
```

```
0    1
1    7
2    2
dtype: int64
```

S is capital in series syntax

You can create the series **with different data types**:

```
In [5]: import pandas as pd

# Create a list with different data types
list_s = [2,-3,6.3,'Mrunal']

series_2 = pd.Series(list_s)

print(series_2)
```

```
0      2
1     -3
2     6.3
3  Mrunal
```

dtype: object

Create a series by short Method:

```
In [6]: import pandas as pd

series_3 = pd.Series(['apple', 'banana', 'Orange', 'Grapes'])
print(series_3)

0    apple
1   banana
2   Orange
3   Grapes
dtype: object
```

Create series by Dictionary:

```
In [7]: import pandas as pd
dict_s = pd.Series({'Football': 'Ronaldo', 'Cricket': 'Virat Kohli', 'Tennis': 'Novak. Djokovic'})
print(dict_s)

Football      Ronaldo
Cricket      Virat Kohli
Tennis    Novak. Djokovic
dtype: object
```

NOTE: If you want to find the syntax or parameter used in **Series()** function, just click in parenthesis of **Series()** function and click Shift + Tab

Labels/Index

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc. This label can be used to access a specified value.

```
In [8]: import pandas as pd
a = [10, 20, 30]
series_3 = pd.Series(a)

#Access the first index and put into a variable index_number
index_number = series_3[0]
print(index_number)

10
```

Create Labels:

With the **index** arguments, you can name your own labels.
Index value should be equal to data value

```
In [9]: import pandas as pd
list_s = [10, 20, 30, 40]

series_4 = pd.Series(list_s, index = ['A', 'B', 'C', 'D'])
print(series_4)

A    10
B    20
C    30
D    40
dtype: int64
```

You can provide different types of Data types for index.

Access the values

To access the value use syntax: **Series_name[Index value]**

```
In [10]: import pandas as pd
dict_s = pd.Series({'A':10,'B':20,'C':30,'D':40})

# Access the value.
value_1 = dict_s['B']
print(value_1)

20
```

Slicing

- Slicing is used when you want some part of the series.
- We pass slice instead of index like this: **[start:end]**
- We can also define the step, like this: **[start:end:step]**

```
In [11]: import pandas as pd
series_5 = pd.Series([10,20,30,40,50])

# Slice the values from 0 to 3
print(series_5[0:4])

0    10
1    20
2    30
3    40
dtype: int64
```

To change the Data type

To change the data type use **dtype** function.

```
In [12]: import pandas as pd
series_6 = pd.Series([10,20,30,40],index = ['A','B','C','D'], dtype = float)
print(series_6)

A    10.0
B    20.0
C    30.0
D    40.0
dtype: float64
```

Column Name

With the help of **name** parameter we can give the name to the column.

```
In [13]: import pandas as pd
series_6 = pd.Series([10,20,30,40],index = ['A','B','C','D'], dtype = float, name = 'data values')
print(series_6)

A    10.0
B    20.0
C    30.0
D    40.0
Name: data values, dtype: float64
```

Arithmetic Operations

Adding the two series with equal data.

```
In [14]: # Create two series
```

```

series_1 =pd.Series([10,20,30,40])
series_2 =pd.Series([1,2,3,4])

# add both the series
add_series = series_1 + series_2

print(add_series)

```

```

0    11
1    22
2    33
3    44
dtype: int64

```

Like this you can perform different types of operation also like devision, multiplication, subtractin etc

Adding the two series with unequal data.

```

In [15]: series_1 =pd.Series([10,20,30,40])
series_2 =pd.Series([1,2,3]) # we provide only three values

# add both the series
add_series = series_1 + series_2

print(add_series)

```

```

0    11.0
1    22.0
2    33.0
3     NaN
dtype: float64

```

In Pandas there is feature to handle the missing values.

DataFrame

- In Pandas DataFrame is mostly used.
- A Pandas DataFrame is a 2-dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Series			Series			DataFrame	
apples			oranges			apples	oranges
0	3	+	0	0	=	0	3
1	2		1	3		1	2
2	0		2	7		2	0
3	1		3	2		3	1
							2

Create DataFrame

Pandas have a **DataFrame()** function to create a DataFrame.

Empty DataFrame:

```

In [16]: import pandas as pd
empty_df = pd.DataFrame()
print(empty_df)

```

```

Empty DataFrame
Columns: []
Index: []

```

Create DataFrame with list:

```
In [17]: import pandas as pd

# Create a List
list_df = ['Apple', 'Mango', 'Orange', 'Banana']

# Create a DataFrame
df_1 = pd.DataFrame(list_df)
print(df_1)
```

	0
0	Apple
1	Mango
2	Orange
3	Banana

Create DataFrame with more columns:

Pandas for that we have to make list of list:

```
In [18]: import pandas as pd

# Create a list of list
lst_df = [[11,12,13],[21,22,23],[31,32,33]]

df_2 = pd.DataFrame(lst_df)
print(df_2)
```

	0	1	2
0	11	12	13
1	21	22	23
2	31	32	33

Create DataFrame with Dictionary:

```
In [19]: import pandas as pd
dict_4 = {'Customer Name':['Alex', 'Bumble', 'Kat', 'Kevin'], 'ID':[101,102,103,104]}
df_4 = pd.DataFrame(dict_4)
print(df_4)
```

	Customer Name	ID
0	Alex	101
1	Bumble	102
2	Kat	103
3	Kevin	104

NOTE: Value of the all array contain in the dictionary have to be same.

Create DataFrame with list of Dictionary:

```
In [20]: import pandas as pd
dict_6 = [{'a':1, 'b':2}, {'a':3, 'b':4}]
df_6 = pd.DataFrame(dict_6)
print(df_6)
```

	a	b
0	1	2
1	3	4

Create DataFrame with Dictionary of Series:

```
In [21]: import pandas as pd
dict_7 = {'ID':pd.Series(['A', 'B', 'C']), 'SN':pd.Series([11,22,33])}
df_7 = pd.DataFrame(dict_7)
print(df_7)
```

	ID	SN
0	A	11
1	B	22
2	C	33

NOTE: There are also other way to create DataFrame like zip function, list of tuple etc.

Locate Row

- As you can see from the result above, the DataFrame is like a table with rows and columns.
- Pandas use the **loc** attribute to return one or more specified rows.

```
In [22]: import pandas as pd
data = {
    'Calories': [430, 420, 390, 376],
    'Duration': [50, 45, 40, 38]
}

# Load data into a DataFrame object
data_df = pd.DataFrame(data)

# Print only first two rows of the DataFrame
print(data_df.loc[[0,1]]) # Notice there is double square bracket.
```

	Calories	Duration
0	430	50
1	420	45

Named Indexes

With the **index** argument, you can name your own indexes.

```
In [23]: # Add a list of names to give each roww a name:
import pandas as pd
data = {
    'Calories': [430, 420, 390, 376],
    'Duration': [50, 45, 40, 38]
}
data_df = pd.DataFrame(data, index = ['Day 1', 'Day 2', 'Day 3', 'Day 4'])
print(data_df)
```

	Calories	Duration
Day 1	430	50
Day 2	420	45
Day 3	390	40
Day 4	376	38

Locate row by Index

Use the named index in the **loc** attribute to return the specified rows.

```
In [24]: import pandas as pd
data = {
    'Calories': [430, 420, 390, 376],
    'Duration': [50, 45, 40, 38]
}
data_df = pd.DataFrame(data, index = ['Day 1', 'Day 2', 'Day 3', 'Day 4'])
print(data_df.loc['Day 2'])
```

	Calories	Duration
Day 2	420	45

Name: Day 2, dtype: int64

Pandas Read CSV

- 'Comma separated value'
- A simple way to store big data sets is to use CSV files (Comma separated files)
- CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

Advantages of CSV file

Advantages of CSV file

- Universal
- Easy to understand
- Quick to create.

In our example we will be using a CSV file called 'data.csv'

	A	B	C	D
1	Duration	Pulse	Maxpulse	Calories
2	60	110	130	409.1
3	60	117	145	479
4	60	103	135	340
5	45	109	175	282.4
6	45	117	148	406
7	60	102	127	300
8	60	110	136	374
9	45	104	134	253.3
10	30	109	133	195.1
11	60	98	124	269
12	60	103	147	329.3
13	60	100	120	250.7
14	60	106	128	345.3
15	60	104	132	379.3
16	60	98	123	275
17	60	98	120	215.2
18	60	100	120	300
19	45	90	112	
20	60	103	123	323
21	45	97	125	243

The above image is some part of that CSV file.

You can download above CSV file from below link.

<https://www.w3schools.com/python/pandas/data.csv>

How to read CSV file

Syntax: `pd.read_csv(file_path)`

```
In [25]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv')
# Put double slash during path
print(df)
```

```
   Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
4         45    117       148     406.0
..      ...    ...      ...      ...
164        60    105       140     290.8
165        60    110       145     300.0
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

[169 rows x 4 columns]

By default, when you print a DataFrame, you will only get the first 5 rows, and the last 5 rows.

But for entire data you can use `to_string()`

```
In [26]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv')
print(df.to_string())
```

```
   Duration  Pulse  Maxpulse  Calories
0         60    110       130     409.1
1         60    117       145     479.0
2         60    103       135     340.0
3         45    109       175     282.4
```


4	45	117	148	406.0
5	60	102	127	300.0
6	60	110	136	374.0
7	45	104	134	253.3
8	30	109	133	195.1
9	60	98	124	269.0
10	60	103	147	329.3
11	60	100	120	250.7
12	60	106	128	345.3
13	60	104	132	379.3
14	60	98	123	275.0
15	60	98	120	215.2
16	60	100	120	300.0
17	45	90	112	NaN
18	60	103	123	323.0
19	45	97	125	243.0
20	60	108	131	364.2
21	45	100	119	282.0
22	60	130	101	300.0
23	45	105	132	246.0
24	60	102	126	334.5
25	60	100	120	250.0
26	60	92	118	241.0
27	60	103	132	NaN
28	60	100	132	280.0
29	60	102	129	380.3
30	60	92	115	243.0
31	45	90	112	180.1
32	60	101	124	299.0
33	60	93	113	223.0
34	60	107	136	361.0
35	60	114	140	415.0
36	60	102	127	300.0
37	60	100	120	300.0
38	60	100	120	300.0
39	45	104	129	266.0
40	45	90	112	180.1
41	60	98	126	286.0
42	60	100	122	329.4
43	60	111	138	400.0
44	60	111	131	397.0
45	60	99	119	273.0
46	60	109	153	387.6
47	45	111	136	300.0
48	45	108	129	298.0
49	60	111	139	397.6
50	60	107	136	380.2
51	80	123	146	643.1
52	60	106	130	263.0
53	60	118	151	486.0
54	30	136	175	238.0
55	60	121	146	450.7
56	60	118	121	413.0
57	45	115	144	305.0
58	20	153	172	226.4
59	45	123	152	321.0
60	210	108	160	1376.0
61	160	110	137	1034.4
62	160	109	135	853.0
63	45	118	141	341.0
64	20	110	130	131.4
65	180	90	130	800.4
66	150	105	135	873.4
67	150	107	130	816.0
68	20	106	136	110.4
69	300	108	143	1500.2
70	150	97	129	1115.0
71	60	109	153	387.6
72	90	100	127	700.0
73	150	97	127	953.2
74	45	114	146	304.0
75	90	98	125	563.2
76	45	105	134	251.0
77	45	110	141	300.0
78	120	100	130	500.4
79	270	100	131	1729.0
80	30	159	182	319.2
81	45	149	169	344.0
82	30	103	139	151.1
83	120	100	130	500.0
84	45	100	120	225.3
85	30	151	170	300.0
86	45	102	136	234.0

87	120	100	157	1000.1
88	45	129	103	242.0
89	20	83	107	50.3
90	180	101	127	600.1
91	45	107	137	NaN
92	30	90	107	105.3
93	15	80	100	50.5
94	20	150	171	127.4
95	20	151	168	229.4
96	30	95	128	128.2
97	25	152	168	244.2
98	30	109	131	188.2
99	90	93	124	604.1
100	20	95	112	77.7
101	90	90	110	500.0
102	90	90	100	500.0
103	90	90	100	500.4
104	30	92	108	92.7
105	30	93	128	124.0
106	180	90	120	800.3
107	30	90	120	86.2
108	90	90	120	500.3
109	210	137	184	1860.4
110	60	102	124	325.2
111	45	107	124	275.0
112	15	124	139	124.2
113	45	100	120	225.3
114	60	108	131	367.6
115	60	108	151	351.7
116	60	116	141	443.0
117	60	97	122	277.4
118	60	105	125	NaN
119	60	103	124	332.7
120	30	112	137	193.9
121	45	100	120	100.7
122	60	119	169	336.7
123	60	107	127	344.9
124	60	111	151	368.5
125	60	98	122	271.0
126	60	97	124	275.3
127	60	109	127	382.0
128	90	99	125	466.4
129	60	114	151	384.0
130	60	104	134	342.5
131	60	107	138	357.5
132	60	103	133	335.0
133	60	106	132	327.5
134	60	103	136	339.0
135	20	136	156	189.0
136	45	117	143	317.7
137	45	115	137	318.0
138	45	113	138	308.0
139	20	141	162	222.4
140	60	108	135	390.0
141	60	97	127	NaN
142	45	100	120	250.4
143	45	122	149	335.4
144	60	136	170	470.2
145	45	106	126	270.8
146	60	107	136	400.0
147	60	112	146	361.9
148	30	103	127	185.0
149	60	110	150	409.4
150	60	106	134	343.0
151	60	109	129	353.2
152	60	109	138	374.0
153	30	150	167	275.8
154	60	105	128	328.0
155	60	111	151	368.5
156	60	97	131	270.4
157	60	100	120	270.4
158	60	114	150	382.8
159	30	80	120	240.9
160	30	85	120	250.4
161	45	90	130	260.4
162	45	95	130	270.0
163	45	100	140	280.9
164	60	105	140	290.8
165	60	110	145	300.0
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

Above code print your whole CSV file.

Pandas write CSV:

To write CSV file means make some changes in CSV file or do some modifications

Access the columns

To find the Columns and their names.

```
In [11]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv')
# To print columns name only
print(df.columns)
```

```
Index(['Duration', 'Pulse', 'Maxpulse', 'Calories'], dtype='object')
```

To print the number of columns, for that we have **usecols** parameter and give index arguments to print the columns.

```
In [28]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv', usecols = [0,1])
print(df)
```

	Duration	Pulse
0	60	110
1	60	117
2	60	103
3	45	109
4	45	117
..
164	60	105
165	60	110
166	60	115
167	75	120
168	75	125

```
[169 rows x 2 columns]
```

To create a column as a Index column.

```
In [29]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv', index_col = 'Duration')
print(df)
```

	Pulse	Maxpulse	Calories
Duration			
60	110	130	409.1
60	117	145	479.0
60	103	135	340.0
45	109	175	282.4
45	117	148	406.0
...
60	105	140	290.8
60	110	145	300.0
60	115	145	310.2
75	120	150	320.4
75	125	150	330.4

```
[169 rows x 3 columns]
```

Access the Rows

To print the number of rows use **nrows** parameter. It takes arguments as a number like 1,2,5 etc. and prints that many rows.

```
In [30]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv', nrows = 3)
print(df)
```

```
Duration  Pulse  Maxpulse  Calories
```

	duration	pulse	maxpulse	calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0

You can skip the rows by using **skiprows** parameter.

```
In [31]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv', skiprows = [0])
print(df)
```

	60	110	130	409.1
0	60	117	145	479.0
1	60	103	135	340.0
2	45	109	175	282.4
3	45	117	148	406.0
4	60	102	127	300.0
..
163	60	105	140	290.8
164	60	110	145	300.0
165	60	115	145	310.2
166	75	120	150	320.4
167	75	125	150	330.4

[168 rows x 4 columns]

It skip the first row that is our columns name.

Parameters of Pandas read_csv() function

Header

Chnage the header according to Index

```
In [32]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv', header = 2)
print(df)
```

	60	117	145	479.0
0	60	103	135	340.0
1	45	109	175	282.4
2	45	117	148	406.0
3	60	102	127	300.0
4	60	110	136	374.0
..
162	60	105	140	290.8
163	60	110	145	300.0
164	60	115	145	310.2
165	75	120	150	320.4
166	75	125	150	330.4

[167 rows x 4 columns]

It set our second row as a header.

NOTE:If you want no header then type header = None

Prefix

- Prefix parameter is use to give the Header to the columns.
- It looks only one strngs.
- To use prefix it is required to use **header = None**

```
In [33]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv', header = None, prefix = 'Data')
print(df)
```

	Data0	Data1	Data2	Data3
0	60	110	130	409.1

1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
..
164	60	105	140	290.8
165	60	110	145	300.0
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

[169 rows x 4 columns]

names

- In prefix we can't give the individuals names to the columns.
- **names** parameter can provide to that to give individuals names to the columns.

```
In [34]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv', skiprows=[0], names=
print(df)
```

	A	B	C	D
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
..
164	60	105	140	290.8
165	60	110	145	300.0
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

[169 rows x 4 columns]

Pandas Methods

head()

To print first rows. By default it print first 5 rows.

```
In [35]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv')
print(df.head())
```

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0

You can also give the parameters to the **head()** function.

```
In [36]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv')
print(df.head(3))
```

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0

tail()

Same as head parameter it print rows but from bottom.

```
In [37]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv')
print(df.tail(3))
```

	Duration	Pulse	Maxpulse	Calories
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

dtype()

To change the type of columns

Syntax: `dtype = {'Column_name': 'data_type'}`

```
In [38]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\data.csv', dtype = {'Duration': float})
print(df)
```

	Duration	Pulse	Maxpulse	Calories
0	60.0	110	130	409.1
1	60.0	117	145	479.0
2	60.0	103	135	340.0
3	45.0	109	175	282.4
4	45.0	117	148	406.0
..
164	60.0	105	140	290.8
165	60.0	110	145	300.0
166	60.0	115	145	310.2
167	75.0	120	150	320.4
168	75.0	125	150	330.4

[169 rows x 4 columns]

we changed the data type of column 'Duration' **int** to **float**

Handling missing values

In pandas there is a functionality to print missing values as NaN.(Not a number)

Panda also consider different types of strings as a NaN, like

All These Strings Are Considered as Default NaN Values by Pandas

#N/A	-NaN	null
#N/A N/A	-nan	n/a
#NA	N/A	nan
-1.#IND	NA	1.#IND
-1.#QNAN	NULL	1.#QNAN

We will use different data set for next parameters. You can download that data set from below link.

<https://www.w3schools.com/python/pandas/dirtydata.csv>

```
In [114]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
print(df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
2	60.0	'2020/12/03'	103.0	NaN	340.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
5	NaN	'2020/12/05'	102.0	127.0	NaN

6	60.0	'2020/12/07'	110.0	136.0	374.0
7	450.0	'2020/12/08'	NaN	134.0	253.3
8	30.0	'2020/12/09'	109.0	NaN	195.1
9	NaN	'2020/12/10'	98.0	124.0	269.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
12	60.0	'2020/12/12'	NaN	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

na_values = 'string'

If you want some string in your CSV file as consider NaN, then use `na_values = 'string'`

```
In [113]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv', na_values = 'Not available')
print(df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
2	60.0	'2020/12/03'	103.0	NaN	340.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
5	NaN	'2020/12/05'	102.0	127.0	NaN
6	60.0	'2020/12/07'	110.0	136.0	374.0
7	450.0	'2020/12/08'	NaN	134.0	253.3
8	30.0	'2020/12/09'	109.0	NaN	195.1
9	NaN	'2020/12/10'	98.0	124.0	269.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
12	60.0	'2020/12/12'	NaN	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

This code find the string 'Not available' in your CSV file and change that string into 'NaN'

keep_default_na = False

If you want to not change the NaN values, keep as it is as default, suppose in your CSV there is N/A, null strings. But you want to keep that as it is, then use `keep_default_na=False`

```
In [41]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv', keep_default_na=False)
print(df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479
2	60	'2020/12/03'	103	NaN	340
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406
5		'2020/12/05'	102	127	
6	60	'2020/12/07'	110	136	374
7	450	'2020/12/08'	NaN	134	253.3
8	30	'2020/12/09'	109		195.1
9		'2020/12/10'	98	124	269
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'		120	250.7
13	60	'2020/12/13'	106	128	345.3

Methods and Functions in Pandas

isnull()

To check the missing values.

```
In [115]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
```

```
print(df.isnull())
```

	Duration	Date	Pulse	Maxpulse	Calories
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	True	False
3	False	False	False	False	False
4	False	False	False	False	False
5	True	False	False	False	True
6	False	False	False	False	False
7	False	False	True	False	False
8	False	False	False	True	False
9	True	False	False	False	False
10	False	False	False	False	False
11	False	False	False	False	False
12	False	False	True	False	False
13	False	False	False	False	False

False= Value

True= Null value

NOTE: notnull() is opposite of the isnull(). It shows the True if there is no null value and false for null value.

If you want to find the how many total number of null value in the column, then use **sum()**

```
In [43]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
print(df.isnull().sum())
```

```
Duration    2
Date        0
Pulse       2
Maxpulse    2
Calories    1
dtype: int64
```

It is showing the null values per column.

If you want to find that the total Null values in whole data set then use **DataFrame.isnull.sum().sum()**

```
In [44]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
print(df.isnull().sum().sum())
```

```
7
```

You can use **isnull()** for series. Use following syntax for series **series_name.isnull()**

dropna()

- One way to deal with empty cells is to remove rows that contain empty cells.
- This is usually Ok, since data set can be very big, and removing a feww rows will not have a big impact on the result.
- For that we have **dropna()** method. This method removes the rows or column that contain the Null values.

Let's print our original Data set.

```
In [45]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
print(df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
2	60.0	'2020/12/03'	103.0	NaN	340.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
5	NaN	'2020/12/05'	102.0	127.0	NaN
6	60.0	'2020/12/07'	110.0	136.0	374.0
7	450.0	'2020/12/08'	NaN	134.0	253.3
8	30.0	'2020/12/09'	109.0	NaN	195.1

9	NaN	'2020/12/10'	98.0	124.0	269.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
12	60.0	'2020/12/12'	NaN	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

See, there are 2,5,7,8,9 rows that contain the null vales

```
In [46]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
new_df = df.dropna()
print(new_df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
6	60.0	'2020/12/07'	110.0	136.0	374.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

2,5,7,8,9 rows are dropped.

NOTE: By default, the dropna() method returns a new DataFrame, and will not change the original.

Parameters of `dropna()`

- `axis=0,1`
- `how='any'`,
- `thresh=None`,
- `subset=None`,
- `inplace=False`

axis

- `axis` parameter removes the row or column that contain the Null values.
- It take the 2 arguments {0,1}.
- 0 = Row (By default value is 0)
- 1 = Column
- syntax: `DataFrame.dropna(axis = 1,0)`
- Pass tuple or list to drop on multiple axes.

```
In [47]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
new_df = df.dropna(axis = 1)
print(new_df)
```

	Date
0	'2020/12/01'
1	'2020/12/02'
2	'2020/12/03'
3	'2020/12/04'
4	'2020/12/05'
5	'2020/12/05'
6	'2020/12/07'
7	'2020/12/08'
8	'2020/12/09'
9	'2020/12/10'
10	'2020/12/11'
11	'2020/12/12'
12	'2020/12/12'
13	'2020/12/13'

Because we give argument 1, it drop all the columns that contain the Null value

how

- Determine if row or column is removed from DataFrame, when we have at least one NA or all NA.
- arguments = {'any', 'all'}
- 'any' = If any NA values are present, drop that row or column.
- 'all' = If all values are NA, drop that row or column.

```
In [48]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
new_df = df.dropna(how = 'any')
print(new_df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
6	60.0	'2020/12/07'	110.0	136.0	374.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

See, it removes the all rows that contain the null value.

thresh

- Whenever we want specific row that contain specific Not null value. Like suppose we want all rows that contain only 2 Null values, that's where we can use the **thresh** parameter.
- arguments: intergers (1,2,3,5 etc)

Let's print the rows that contain 5 Not null values.

```
In [49]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
new_df = df.dropna(thresh = 5)
print(new_df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
6	60.0	'2020/12/07'	110.0	136.0	374.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

subset

- **subset** parameter drop the rows that contain Null value of specific columns.
- arguments: Column name (It take a arguments in square bracket)

EXAMPLE: Drop the rows that contain the null value of 'Maxpulse' column.

```
In [50]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
new_df = df.dropna(subset = ['Maxpulse'])
print(new_df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
5	NaN	'2020/12/05'	102.0	127.0	NaN
6	60.0	'2020/12/07'	110.0	136.0	374.0
7	450.0	'2020/12/08'	NaN	134.0	253.3

9	NaN	'2020/12/10'	98.0	124.0	269.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
12	60.0	'2020/12/12'	NaN	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

It drop the 2 and 8 row, because in that row the maxpulse column contain the Null values.

inplace

If you want to change the original DataFrame, use the **inplace = True** argument.

```
In [51]: # Remove all rows with null values
import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
df.dropna(inplace = True)
print(df.to_string())
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
6	60.0	'2020/12/07'	110.0	136.0	374.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

fillna()

- Fillna fills the NaN values with given values (input)
- Syntax: **DataFrame.fillna()**

Parameters of fillna()

- **value**
- **method**
- **axis**
- **inplace**
- **limit**

value

- By using the **value** parameter you fill the Null value with a specified Number, string.
- arguments: {scalar, dict, Series, or DataFrame}

```
In [52]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
update_df = df.fillna(0.0)
print(update_df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
2	60.0	'2020/12/03'	103.0	0.0	340.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
5	0.0	'2020/12/05'	102.0	127.0	0.0
6	60.0	'2020/12/07'	110.0	136.0	374.0
7	450.0	'2020/12/08'	0.0	134.0	253.3
8	30.0	'2020/12/09'	109.0	0.0	195.1
9	0.0	'2020/12/10'	98.0	124.0	269.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
12	60.0	'2020/12/12'	0.0	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

It fills the all the Null Spaces with 0.0

method

- To fill the previous and next value in null space.
- arguments: {'ffill', 'bfill'}
- ffill: To fill the forward value.
- bfill: To fill the backward value.
- Syntax: `DataFrame.fillna(method = 'ffill')`

```
In [53]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
update_df = df.fillna(method = 'ffill')
print(update_df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
2	60.0	'2020/12/03'	103.0	145.0	340.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
5	45.0	'2020/12/05'	102.0	127.0	406.0
6	60.0	'2020/12/07'	110.0	136.0	374.0
7	450.0	'2020/12/08'	110.0	134.0	253.3
8	30.0	'2020/12/09'	109.0	134.0	195.1
9	30.0	'2020/12/10'	98.0	124.0	269.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
12	60.0	'2020/12/12'	100.0	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

It fills the Null value with previous values.

NOTE: You can use 'pad' instead of 'ffill'

axis

- By using axis we can make changes according to the rows(0) and columns(1).
- To use the **axis** it is required to use **method** or **value**.

```
In [54]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
update_df = df.fillna(method = 'ffill', axis = 1)
print(update_df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60	'2020/12/01'	110	130	409.1
1	60	'2020/12/02'	117	145	479
2	60	'2020/12/03'	103	103	340
3	45	'2020/12/04'	109	175	282.4
4	45	'2020/12/05'	117	148	406
5	NaN	'2020/12/05'	102	127	127
6	60	'2020/12/07'	110	136	374
7	450	'2020/12/08'	'2020/12/08'	134	253.3
8	30	'2020/12/09'	109	109	195.1
9	NaN	'2020/12/10'	98	124	269
10	60	'2020/12/11'	103	147	329.3
11	60	'2020/12/12'	100	120	250.7
12	60	'2020/12/12'	'2020/12/12'	120	250.7
13	60	'2020/12/13'	106	128	345.3

It fills the value from forward Column(1 represents the column).

limit

If there is 4 cell empty in one column but you want to fill only first 2 cell, there we can use **limit**.

With **limit** parameter it is required to use **value** or **method**

```
In [55]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\dirtydata.csv')
update_df = df.fillna(1000, limit = 1)
print(update_df)
```

	Duration	Date	Pulse	Maxpulse	Calories
0	60.0	'2020/12/01'	110.0	130.0	409.1
1	60.0	'2020/12/02'	117.0	145.0	479.0
2	60.0	'2020/12/03'	103.0	1000.0	340.0
3	45.0	'2020/12/04'	109.0	175.0	282.4
4	45.0	'2020/12/05'	117.0	148.0	406.0
5	1000.0	'2020/12/05'	102.0	127.0	1000.0
6	60.0	'2020/12/07'	110.0	136.0	374.0
7	450.0	'2020/12/08'	1000.0	134.0	253.3
8	30.0	'2020/12/09'	109.0	NaN	195.1
9	NaN	'2020/12/10'	98.0	124.0	269.0
10	60.0	'2020/12/11'	103.0	147.0	329.3
11	60.0	'2020/12/12'	100.0	120.0	250.7
12	60.0	'2020/12/12'	NaN	120.0	250.7
13	60.0	'2020/12/13'	106.0	128.0	345.3

It fills only first cell of empty cell of every column.

inplace

We already learn about **inplace**, with use of **inplace** parameter we can directly change the original DataFrame.

arguments: {'True', 'False'}

replace()

- It replace the values of DataFrame with other values dynamically.
- Syntax: **DataFrame.replace()**

We will use another Data set, you can download from given link.

https://indianaiproduction.com/wp-content/uploads/2019/06/Fortune_10.csv

```
In [56]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\fortune_10.csv')
print(df)
```

	ID	Name	Industry	Profit	Growth
0	1	Lamtone	IT Services	0	30%
1	2	Stripfind	Financial Services	0	20%
2	3	Canecorporation	Health	0	7%
3	4	Mattouch	IT Services	6597557	26%
4	5	Techdrill	Health	3138627	8%
5	6	Techline	Health	8427816	23%
6	7	Cityace	Health	3005116	6%
7	8	Kayelectronics	Health	5573830	4%
8	9	Ganzlax	IT Services	11901180	18%
9	10	Trantraxlax	Government Services	5453060	7%

This is our data set 'Fortune_10'

Parameters of **replace()**

- **to_replace**=None,
- **value** =None,
- **inplace**=False,
- **limit**=None,
- **regex**=False,
- **method**='pad',

to_replace or value

to_replace or value

- To replace from string1 to value
- arguments: {str, regex, list, dict, Series, int, float, or None}
- Syntax: `DartaFrame.replace(to_replace = 'string_1', value='string_2')`
- You can use directly also like this syntax.
- `DartaFrame.replace('string_1', 'string_2')`

```
In [57]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\fortune_10.csv')
update_fortune_10 = df.replace('Health', 'Finance')
print(update_fortune_10)
```

	ID	Name	Industry	Profit	Growth
0	1	Lamtone	IT Services	0	30%
1	2	Stripfind	Financial Services	0	20%
2	3	Canecorporation	Finance	0	7%
3	4	Mattouch	IT Services	6597557	26%
4	5	Techdrill	Finance	3138627	8%
5	6	Techline	Finance	8427816	23%
6	7	Cityace	Finance	3005116	6%
7	8	Kayelectronics	Finance	5573830	4%
8	9	Ganzlax	IT Services	11901180	18%
9	10	Trantraxlax	Government Services	5453060	7%

It changed string 'Health' with 'Finance'.

You can change multiple values by using list:

```
In [58]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\fortune_10.csv')
update_fortune_10 = df.replace([1,2,3,4,5,6,7,8,9,10],['a','b','c','d','e','f','h','i','j','k'])
print(update_fortune_10)
```

	ID	Name	Industry	Profit	Growth
0	a	Lamtone	IT Services	0	30%
1	b	Stripfind	Financial Services	0	20%
2	c	Canecorporation	Health	0	7%
3	d	Mattouch	IT Services	6597557	26%
4	e	Techdrill	Health	3138627	8%
5	f	Techline	Health	8427816	23%
6	h	Cityace	Health	3005116	6%
7	i	Kayelectronics	Health	5573830	4%
8	j	Ganzlax	IT Services	11901180	18%
9	k	Trantraxlax	Government Services	5453060	7%

We changed the whole 'ID' column.

If you want to changed the specific value in specific column, then use dictionary.

```
In [59]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\fortune_10.csv')
update_fortune_10 = df.replace({'Industry': 'Health'}, 'none')
print(update_fortune_10)
```

	ID	Name	Industry	Profit	Growth
0	1	Lamtone	IT Services	0	30%
1	2	Stripfind	Financial Services	0	20%
2	3	Canecorporation	none	0	7%
3	4	Mattouch	IT Services	6597557	26%
4	5	Techdrill	none	3138627	8%
5	6	Techline	none	8427816	23%
6	7	Cityace	none	3005116	6%
7	8	Kayelectronics	none	5573830	4%
8	9	Ganzlax	IT Services	11901180	18%
9	10	Trantraxlax	Government Services	5453060	7%

We changed the "Health" with "none" string in 'Industry' column.

regex

- To convert String into int.
- Syntax: `DataFrame.replace('A-Za-z', 0)`
- Above code convert all strings into 0.
- To convert only for specific column:
- Syntax: `DataFrame.replace('Column_Name': 'A-Za-z', 0, regex = True)`

```
In [60]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\fortune_10.csv')
update_fortune_10 = df.replace({'Industry': 'A-Za-z'}, 0, regex = True)
print(update_fortune_10)
```

	ID	Name	Industry	Profit	Growth
0	1	Lamtone	0	0	30%
1	2	Stripfind	0	0	20%
2	3	Canecorporation	0	0	7%
3	4	Mattouch	0	6597557	26%
4	5	Techdrill	0	3138627	8%
5	6	Techline	0	8427816	23%
6	7	Cityace	0	3005116	6%
7	8	Kayelectronics	0	5573830	4%
8	9	Ganzlax	0	11901180	18%
9	10	Trantraxlax	0	5453060	7%

method

- To replace with forward and backward words
- arguments: {'ffill', 'bfill'}
- ffill: To fill the forward value.
- bfill: To fill the backward value.
- Syntax: `DataFrame.replace('Value_name', method = 'ffill')`

```
In [61]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\fortune_10.csv')
update_fortune_10 = df.replace('Health', method='ffill')
print(update_fortune_10)
```

	ID	Name	Industry	Profit	Growth
0	1	Lamtone	IT Services	0	30%
1	2	Stripfind	Financial Services	0	20%
2	3	Canecorporation	Financial Services	0	7%
3	4	Mattouch	IT Services	6597557	26%
4	5	Techdrill	IT Services	3138627	8%
5	6	Techline	IT Services	8427816	23%
6	7	Cityace	IT Services	3005116	6%
7	8	Kayelectronics	IT Services	5573830	4%
8	9	Ganzlax	IT Services	11901180	18%
9	10	Trantraxlax	Government Services	5453060	7%

limit

- How many values you want to replace
- arguments: integers (1,2,3,4 etc)
- Syntax: `DataFrame.replace(limit = 1,2 etc)`
- With **limit** parameter it is required to use **to_replace** or **method**

```
In [62]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\fortune_10.csv')
update_fortune_10 = df.replace(0, method='bfill', limit=3)
print(update_fortune_10)
```

	ID	Name	Industry	Profit	Growth
0	1	Lamtone	IT Services	6597557	30%
1	2	Stripfind	Financial Services	6597557	20%
2	3	Canecorporation	Health	6597557	7%
3	4	Mattouch	IT Services	6597557	26%
4	5	Techdrill	Health	3138627	8%
5	6	Techline	Health	8427816	23%
6	7	Cityace	Health	3005116	6%
7	8	Kayelectronics	Health	5573830	4%
8	9	Ganzlax	IT Services	11901180	18%

We replace all three zero in the profit column with backward value.

interpolate()

- Pandas interpolate() function is basically used to fill NaN values in dataframe or series.
- It fill only numeric value, not string.
- Syntax: `DataFrame.interpolate()`

But we have already `fillna()` function to fill the values then why `interpolate()`, because:

- Very powerful function
- Uses various interpolation techniques.

We will take our previous Data set 'dirtydata'

Let's print the Fortune_10 Data set

Parameters of interpolate()

- **method**: str = 'linear',
- **axis**: Union[str, int] = 0,
- **limit**: Union[int, NoneType] = None,
- **inplace**: bool = False,
- **limit_direction**: Union[str, NoneType] = None,
- **limit_area**: Union[str, NoneType] = None,
- **downcast**: Union[str, NoneType] = None,

loc()

- Access a group of rows and columns by label(s) or a boolean array.
- Basically it return the desired output according what type of input we give.
- Syntax: `DataFrame.loc[]`

We will use different data set "Student_result", lets print that data set.

```
In [63]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
print(df)
```

	Student ID	Class	Study hrs	Sleeping hrs	Percentage
0	1001	10	2	9	50
1	1002	10	6	8	80
2	1003	10	6	8	91
3	1004	11	0	8	82
4	1005	11	4	7	60
5	1006	11	6	7	96
6	1007	12	4	6	80
7	1008	12	10	6	90
8	1009	12	2	8	60
9	1010	12	6	9	85

Parameters of loc[]

- A single label
- A list or array of labels
- A slice object with labels

A single label

- It return the rows of given index.
- arguments: Index number like 1,2,3,etc
- Syntax: **DataFrame.loc[Index]**

```
In [64]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
loc_df = df.loc[2]
print(loc_df)
```

```
Student ID      1003
Class           10
Study hrs       6
Sleeping hrs    8
Percentage      91
Name: 2, dtype: int64
```

It return the 2 row.

A list of array of labels

- It can take a multiple index at one time and gives all that rows.
- Syntax: **DataFrame.loc[[Multiple Index]]**

```
In [65]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
loc_df = df.loc[[2,4]]
print(loc_df)
```

```
Student ID  Class  Study hrs  Sleeping hrs  Percentage
2          1003    10         6              8          91
4          1005    11         4              7          60
```

If we want a specific value from specific column, then use following syntax:

DataFrame.loc[index number, Column name]

```
In [66]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
loc_df = df.loc[4,"Class"]
print(loc_df)
```

```
11
```

A slice object with labels

- It gives you some part of DataFrame.
- Syntax: **DataFrame.loc[initial value:End value, Column name]**

```
In [67]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
loc_df = df.loc[0:3,"Class"]
print(loc_df)
```

```
0      10
1      10
2      10
3      11
Name: Class, dtype: int64
```

We can use conditional operator also with **loc[]** method

```
In [68]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
loc_df = df.loc[df['Class'] < 60]
print(loc_df)
```

```
Student ID  Class  Study hrs  Sleeping hrs  Percentage
0          1001    10         2              9          50
1          1002    10         6              8          80
```

2	1003	10	6	8	91
3	1004	11	0	8	82
4	1005	11	4	7	60
5	1006	11	6	7	96
6	1007	12	4	6	80
7	1008	12	10	6	90
8	1009	12	2	8	60
9	1010	12	6	9	85

As we can see it print the rows that contain value less than 11.

And if you want to print above values only for specific column, then use:

```
In [69]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
loc_df = df.loc[df['Class'] < 11, ['Percentage']]
print(loc_df)
```

```
Percentage
0      50
1      80
2      91
```

iloc[]

- Integer location-based indexing
- Syntax: DataFrame.**iloc[]**

Parameters of **iloc[]**

- An integer
- A list or array of integers
- A slice object with ints
- A boolean array

An integer

```
In [70]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
loc_df = df.iloc[0]
print(loc_df)
```

```
Student ID      1001
Class           10
Study hrs        2
Sleeping hrs     9
Percentage       50
Name: 0, dtype: int64
```

A list or array of integers

```
In [71]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
loc_df = df.iloc[[2,1]]
print(loc_df)
```

```
Student ID  Class  Study hrs  Sleeping hrs  Percentage
2      1003     10         6             8         91
1      1002     10         6             8         80
```

A slice object with ints

```
In [72]: import pandas as pd
```

```
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
loc_df = df.iloc[:, 0]
print(loc_df)
```

```
0    1001
1    1002
2    1003
3    1004
4    1005
5    1006
6    1007
7    1008
8    1009
9    1010
Name: Student ID, dtype: int64
```

```
In [73]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
loc_df = df.iloc[[0,1]]
print(loc_df)
```

	Student ID	Class	Study hrs	Sleeping hrs	Percentage
0	1001	10	2	9	50
1	1002	10	6	8	80

groupby()

- Pandas groupby function is used to split the data into groups based on some criteria.
- Syntax: **DataFrame.groupby()**
- Any groupby operation involves one of the following operations on the original object:
 - Splitting the object
 - Applying a function
 - Combining the result

Our Original Data set

```
In [74]: import pandas as pd
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')
print(df)
```

	Student ID	Class	Study hrs	Sleeping hrs	Percentage
0	1001	10	2	9	50
1	1002	10	6	8	80
2	1003	10	6	8	91
3	1004	11	0	8	82
4	1005	11	4	7	60
5	1006	11	6	7	96
6	1007	12	4	6	80
7	1008	12	10	6	90
8	1009	12	2	8	60
9	1010	12	6	9	85

Parameters of groupby()

- **by**=None,
- **axis**=0,
- **level**=None,
- **as_index**: bool = True,
- **sort**: bool = True,
- **group_keys**: bool = True,
- **squeeze**: bool = <object object at 0x000001B633BA87F0>,
- **observed**: bool = False,
- **dropna**: bool = True,

by

- Used to determine the groups for the groupby.

- arguments:{mapping, function, label, or list of labels}

```
In [75]: import pandas as pda
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')

# Make a groups for 'Study hrs' column
groupby_df = df.groupby(by = 'Study hrs')

# Print that groups
print(groupby_df.groups)

{0: [3], 2: [0, 8], 4: [4, 6], 6: [1, 2, 5, 9], 10: [7]}
```

It made a group according to study hrs, like 0 hrs have 3 index, 2 hrs have 0 and 8 index etc.

Let's try more than one labels with shortcut method.

```
In [76]: import pandas as pda
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')

# Make a groups for 'Study hrs' column
groupby_df = df.groupby(['Study hrs', 'Class'])

# Print that groups
print(groupby_df.groups)

{(0, 11): [3], (2, 10): [0], (2, 12): [8], (4, 11): [4], (4, 12): [6], (6, 10): [1, 2], (6, 11): [5], (6, 12): [9], (10, 12): [7]}
```

It made a group according to class and study hrs

We can split our data by using the `get_group()` function.

Example: Print the data that contain only class 11 student.

```
In [77]: import pandas as pda
df = pd.read_csv('C:\\Users\\Mrunal Wankhede\\Desktop\\Study Material\\Excel Sheets\\student_results.csv')

# Make a groups for 'Study hrs' column
groupby_df = df.groupby('Class').get_group(11)

# Print that groups
print(groupby_df)
```

	Student ID	Class	Study hrs	Sleeping hrs	Percentage
3	1004	11	0	8	82
4	1005	11	4	7	60
5	1006	11	6	7	96

As we can see it print data only for class 11.

merge()

- Pandas merge connects columns or indexes in DataFrame based on one or more keys.
- With the help of pandas `merge()` function we can connect multiple DataFrame.

Let's create the two DataFrame

```
In [78]: # Create 1st dataframe
import pandas as pd
df_1 = pd.DataFrame({'ID': [1,2,3,4], 'Class': [9,10,11,12 ]})
print(df_1)

   ID  Class
0   1     9
1   2    10
2   3    11
3   4    12
```

```
In [79]: # Create 2nd dataframe
import pandas as pd
df_2 = pd.DataFrame({'ID':[1,2,3,4], 'Name':['A', 'B', 'C', 'D']})
print(df_2)
```

	ID	Name
0	1	A
1	2	B
2	3	C
3	4	D

Parameters of `merge()`

- `left`,
- `right`,
- `how`= 'inner',
- `on`=None,
- `left_on`=None,
- `right_on`=None,
- `left_index`= False,
- `right_index`= False,
- `sort`= False,
- `suffixes`=('_x', '_y'),
- `copy`: bool = True,
- `indicator`= False,
- `validate`=None,

left,right

```
In [80]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})
df_2 = pd.DataFrame({'ID':[1,2,3,4], 'Name':['A', 'B', 'C', 'D']})

merge_1 = pd.merge(df_1,df_2)

print(merge_1)
```

	ID	Class	Name
0	1	9	A
1	2	10	B
2	3	11	C
3	4	12	D

Because we put df_1 on left side that's why the 'Class' column shows first.
If we put the df_2 on left side the 'Name' column will appear first, let's try

```
In [81]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})
df_2 = pd.DataFrame({'ID':[1,2,3,4], 'Name':['A', 'B', 'C', 'D']})

merge_1 = pd.merge(df_2,df_1)

print(merge_1)
```

	ID	Name	Class
0	1	A	9
1	2	B	10
2	3	C	11
3	4	D	12

Sometimes in DataFrames there are same Column and we don't need to print that column two times, that's where parameter `on` can use.

on

- `on` parameter is used to join the same column or index and print only once time.
- arguments:{label or list}

- label: If there is only one column same then give only that column name.
- list: If there are more than one column same then make a list and put all that column name in that list.

```
In [82]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ], 'Div':['A','b','C','D']})
df_2 = pd.DataFrame({'ID':[1,2,3,4], 'Name':['A','B','C','D'], 'Div':['A','b','C','D']})

merge_1 = pd.merge(df_2,df_1, on = ['ID','Div'])

print(merge_1)
```

	ID	Name	Div	Class
0	1	A	A	9
1	2	B	b	10
2	3	C	C	11
3	4	D	D	12

It print only one time 'ID' and 'Div' column.

If there are not similar values in column,like

```
In [83]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})

# we change the last value in 'Id' column
df_2 = pd.DataFrame({'ID':[1,2,3,5], 'Name':['A','B','C','D']})

merge_1 = pd.merge(df_2,df_1, on = 'ID')

print(merge_1)
```

	ID	Name	Class
0	1	A	9
1	2	B	10
2	3	C	11

It print only similar value, but we want all vlaues that's where **how** parameter came.

how

- It print the all the values even if there are not similar values.
- arguments: {'left', 'right', 'outer', 'inner'}, default 'inner'
 - 'left': It print all the values from left DataFrame
 - 'right': It print all the values from right DataFrame
 - 'outer': It print all the values from both DataFrame
 - 'inner': It print only similar values.

Example: Print all the values from left DataFrame.

```
In [84]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})

# we change the last value in 'Id' column
df_2 = pd.DataFrame({'ID':[1,2,3,5], 'Name':['A','B','C','D']})

merge_1 = pd.merge(df_2,df_1, on = 'ID',how = 'left')

print(merge_1)
```

	ID	Name	Class
0	1	A	9.0
1	2	B	10.0
2	3	C	11.0
3	5	D	NaN

It print the all values from left DataFrame, It print the 5 which is present in left DataFrame.

Example: Print all the values from right DataFrame.

```
In [85]: import pandas as pd
```

```
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})

# we change the last value in 'Id' column
df_2 = pd.DataFrame({'ID':[1,2,3,5], 'Name':['A','B','C','D']})

merge_1 = pd.merge(df_2,df_1, on = 'ID',how = 'right')

print(merge_1)
```

	ID	Name	Class
0	1	A	9
1	2	B	10
2	3	C	11
3	4	NaN	12

It print the 4 in 'ID' column.

If we want to know which value is belongs from which column, then use **indicator** parameter.

indicator

- It shows the which value is belongs from which parameter.
- arguments:{bool or str, default False}
 - If True, adds a column to the output DataFrame called "_merge" with information on the source of each row

```
In [86]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})

# we change the last value in 'Id' column
df_2 = pd.DataFrame({'ID':[1,2,3,5], 'Name':['A','B','C','D']})

merge_1 = pd.merge(df_2,df_1, on = 'ID',how = 'outer', indicator = True)

print(merge_1)
```

	ID	Name	Class	_merge
0	1	A	9.0	both
1	2	B	10.0	both
2	3	C	11.0	both
3	5	D	NaN	left_only
4	4	NaN	12.0	right_only

If suppose all the values are differnt in 'ID' column, like

```
In [87]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})
print(df_1)
# we change all the values in 'ID' column
df_2 = pd.DataFrame({'ID':[5,6,7,8], 'Name':['A','B','C','D']})
print(df_2)
```

	ID	Class
0	1	9
1	2	10
2	3	11
3	4	12

	ID	Name
0	5	A
1	6	B
2	7	C
3	8	D

Let's try to merge them

```
In [88]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})

# we change all the values in 'ID' column
df_2 = pd.DataFrame({'ID':[5,6,7,8], 'Name':['A','B','C','D']})

merge_1 = pd.merge(df_2,df_1)
print(merge_1)
```

Empty DataFrame
Columns: [ID, Name, Class]

Index: []

It didn't show any output, then we can use **left_index**, **right_index** parameters to merge that both dataframe.

left_index, right_index

- It merge the DataFrame even if there are not similar values in the DataFrames.
- arguments:{bool, default False}
 - 'True': It print the
- It is required to use both **left_index**, **right_index** parameters together.

```
In [89]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})

# we change all the values in 'ID' column
df_2 = pd.DataFrame({'ID':[5,6,7,8], 'Name':['A', 'B', 'C', 'D']})

merge_1 = pd.merge(df_2, df_1, left_index = True, right_index = True)
print(merge_1)
```

	ID_x	Name	ID_y	Class
0	5	A	1	9
1	6	B	2	10
2	7	C	3	11
3	8	D	4	12

If we have same column name not a 'ID', like suppose we have same 'Class' name column, like

```
In [90]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})

# We took same column in both DataFrame 'Class'
df_2 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})

merge_1 = pd.merge(df_2, df_1, on = 'ID')
print(merge_1)
```

	ID	Class_x	Class_y
0	1	9	9
1	2	10	10
2	3	11	11
3	4	12	12

It print them with different name, but if you want to give different name to that columns, then use **suffixes** parameter.

suffixes

- If there is same columns in different DataFrame then it helps you to give different names to them.
- arguments:{list-like, default is ("_x", "_y")}
 - 'list-like'- Means you can give them different name in the list.

```
In [91]: import pandas as pd
df_1 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})

# We took same column in both DataFrame 'Class'
df_2 = pd.DataFrame({'ID':[1,2,3,4], 'Class':[9,10,11,12 ]})

merge_1 = pd.merge(df_2, df_1, on = 'ID', suffixes = ['_Higher', '_Middle'])
print(merge_1)
```

	ID	Class_Higher	Class_Middle
0	1	9	9
1	2	10	10
2	3	11	11
3	4	12	12

Concat()

- Pandas provides various facilities for easily combining together Series, DataFrame and Panel Objects.
- Syntax: **Pandas.concat()**

Parameters of **concat()**

- **objs**
- **axis=0**,
- **join='outer'**,
- **ignore_index= False**,
- **keys=None**,
- **levels=None**,
- **names=None**,
- **verify_integrity= False**,
- **sort= False**,
- **copy= True**,

Firstly create the two series to concatenate:

```
In [92]: import pandas as pd
sr_1 = pd.Series([0,1,2])
sr_2 = pd.Series([3,4,5])

print(sr_1)
print(sr_2)
```

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

Let's concatenate the series:

```
In [93]: import pandas as pd
sr_1 = pd.Series([0,1,2])
sr_2 = pd.Series([3,4,5])

concat_sr = pd.concat([sr_1,sr_2])
print(concat_sr)
```

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

We concatenate the series that's same, let's concatenate the different sizes series.

```
In [94]: import pandas as pd
sr_1 = pd.Series([0,1,2])
sr_2 = pd.Series([3,4,5,6,7,8])

concat_sr = pd.concat([sr_1,sr_2])
print(concat_sr)
```

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

Let's use concat for DataFrame

Firstly we make two DataFrame for Concat with Dictionary.

```
In [95]: df_1 = pd.DataFrame({'ID': [1,2,3,4],
                             'Name': ['A', 'B', 'C', 'D'],
                             'Class': [5,6,7,8]})
df_2 = pd.DataFrame({'ID': [5,6,7,8],
                     'Name': ['E', 'F', 'G', 'H'],
                     'Class': [9,10,11,12]})

print(df_1)
print(df_2)
```

	ID	Name	Class
0	1	A	5
1	2	B	6
2	3	C	7
3	4	D	8

	ID	Name	Class
0	5	E	9
1	6	F	10
2	7	G	11
3	8	H	12

Let's concat the both DataFrame

```
In [96]: df_1 = pd.DataFrame({'ID': [1,2,3,4],
                             'Name': ['A', 'B', 'C', 'D'],
                             'Class': [5,6,7,8]})
df_2 = pd.DataFrame({'ID': [5,6,7,8],
                     'Name': ['E', 'F', 'G', 'H'],
                     'Class': [9,10,11,12]})

concat_df = pd.concat([df_1,df_2])
print(concat_df)
```

	ID	Name	Class
0	1	A	5
1	2	B	6
2	3	C	7
3	4	D	8
0	5	E	9
1	6	F	10
2	7	G	11
3	8	H	12

If you want to change the position of DataFrame like suppose you want print df_2 above, then just switch there name in concat function.

We concat the both DataFrame but the index are not in sequence, they start from 0 when 2nd DataFrame start. If you want to solve that prolem then use **ignore_index** parameter.

ignore_index

- **ignore_index** parameter use for print the index sequence wise.
- arguments:{bool, default False}

```
In [97]: df_1 = pd.DataFrame({'ID': [1,2,3,4],
                             'Name': ['A', 'B', 'C', 'D'],
                             'Class': [5,6,7,8]})
df_2 = pd.DataFrame({'ID': [5,6,7,8],
                     'Name': ['E', 'F', 'G', 'H'],
                     'Class': [9,10,11,12]})

concat_df = pd.concat([df_1,df_2],ignore_index = True)
print(concat_df)
```

	ID	Name	Class
0	1	A	5
1	2	B	6
2	3	C	7
3	4	D	8
4	5	E	9
5	6	F	10

6	7	G	11
7	8	H	12

If you want to `concat()` by column wise then use `axis` parameter.

axis

- `axis` parameter is used to concat the DataFrame, Series column wise.
- arguments: {0/'index', 1/'columns'}, default 0

```
In [98]: df_1 = pd.DataFrame({'ID': [1,2,3,4],
                          'Name': ['A', 'B', 'C', 'D'],
                          'Class': [5,6,7,8]})
df_2 = pd.DataFrame({'ID': [5,6,7,8],
                     'Name': ['E', 'F', 'G', 'H'],
                     'Class': [9,10,11,12]})

concat_df = pd.concat([df_1, df_2], axis = 1)
print(concat_df)
```

	ID	Name	Class	ID	Name	Class
0	1	A	5	5	E	9
1	2	B	6	6	F	10
2	3	C	7	7	G	11
3	4	D	8	8	H	12

join

- It gives the union or intersection of DataFrames.
- arguments : {'inner', 'outer'}, default 'outer'
 - 'inner': Intersection
 - 'outer': Union

```
In [99]: df_1 = pd.DataFrame({'ID': [1,2,3,4],
                          'Name': ['A', 'B', 'C', 'D'],
                          'Class': [5,6,7,8]})
df_2 = pd.DataFrame({'ID': [3,4],
                     'Name': ['C', 'D'],
                     'Class': [7,8]})

concat_df = pd.concat([df_1, df_2], axis = 1, join = 'inner')
print(concat_df)
```

	ID	Name	Class	ID	Name	Class
0	1	A	5	3	C	7
1	2	B	6	4	D	8

It prints only the union values, the first values of 'ID' column is same.

keys

- With the help of `keys` parameter we can give the label to the DataFrame.
- arguments: {sequence, default None}

```
In [100]: df_1 = pd.DataFrame({'ID': [1,2,3,4],
                          'Name': ['A', 'B', 'C', 'D'],
                          'Class': [5,6,7,8]})
df_2 = pd.DataFrame({'ID': [5,6,7,8],
                     'Name': ['E', 'F', 'G', 'H'],
                     'Class': [9,10,11,12]})

concat_df = pd.concat([df_1, df_2], keys = ['First_df', 'Second_df'])
print(concat_df)
```

		ID	Name	Class
First_df	0	1	A	5
	1	2	B	6
	2	3	C	7
	3	4	D	8
Second_df	0	5	E	9
	1	6	F	10

2	7	G	11
3	8	H	12

let's use with **axis** parameter.

```
In [101]: df_1 = pd.DataFrame({'ID': [1,2,3,4],
                             'Name': ['A', 'B', 'C', 'D'],
                             'Class': [5,6,7,8]})
df_2 = pd.DataFrame({'ID': [5,6,7,8],
                     'Name': ['E', 'F', 'G', 'H'],
                     'Class': [9,10,11,12]})

concat_df = pd.concat([df_1,df_2],axis = 1, keys = ['First_df', 'Second_df'])
print(concat_df)
```

	First_df			Second_df		
	ID	Name	Class	ID	Name	Class
0	1	A	5	5	E	9
1	2	B	6	6	F	10
2	3	C	7	7	G	11
3	4	D	8	8	H	12

sort

- **sort** parameter used to concat the DataFrame of different columns.
- arguments:{ bool, default False}

```
In [102]: df_1 = pd.DataFrame({'ID': [1,2,3,4],
                             'Name': ['A', 'B', 'C', 'D'],
                             'Class': [5,6,7,8]})
df_2 = pd.DataFrame({"Marks": [40,63,91,34]})

concat_df = pd.concat([df_1,df_2], sort = False )
print(concat_df)
```

	ID	Name	Class	Marks
0	1.0	A	5.0	NaN
1	2.0	B	6.0	NaN
2	3.0	C	7.0	NaN
3	4.0	D	8.0	NaN
0	NaN	NaN	NaN	40.0
1	NaN	NaN	NaN	63.0
2	NaN	NaN	NaN	91.0
3	NaN	NaN	NaN	34.0

join()

- DataFrame join is a convenient method for combining the columns of two potentially differently-indexed.
- Syntax: **DataFrame.join()**

Parameters of join()

other,
on=None,
how='left',
lsuffix="",
rsuffix="",
sort=False

```
In [103]: # Create two DataFrame
df_1 = pd.DataFrame({'A': [1,2,3,],
                     'B': [10,20,30,]})
df_2 = pd.DataFrame({'C': [4,5,6],
                     'D': [40,50,60]})

# Join them by using join function

join_df = df_1.join(df_2)
```

```
print(join_df)
```

	A	B	C	D
0	1	10	4	40
1	2	20	5	50
2	3	30	6	60

To join the DataFrame it is required that the index must be same of the both DataFrame.

how

- It print the all the values even if there are not similar values.
- arguments: {'left', 'right', 'outer', 'inner'}, default 'left'
 - 'left': It print all the values from left DataFrame
 - 'right': It print all the values from right DataFrame
 - 'outer': It print all the values from both DataFrame
 - 'inner': It print only similar values.

```
In [104]: df_1 = pd.DataFrame({'A': [1, 2, 3, ],
                             'B': [10, 20, 30, ]},
                             index = ['a', 'b', 'c'])

df_2 = pd.DataFrame({'C': [4, 5],
                     'D': [40, 50]},
                     index = ['a', 'b'])

join_df = df_1.join(df_2, how = 'right')

print(join_df)
```

	A	B	C	D
a	1	10	4	40
b	2	20	5	50

```
In [105]: df_1 = pd.DataFrame({'A': [1, 2, 3, ],
                             'B': [10, 20, 30, ]},
                             index = ['a', 'b', 'c'])

df_2 = pd.DataFrame({'C': [4, 5],
                     'D': [40, 50]},
                     index = ['a', 'b'])

join_df = df_1.join(df_2, how = 'inner')

print(join_df)
```

	A	B	C	D
a	1	10	4	40
b	2	20	5	50

lsuffix, rsuffix

- This parameter is used to give the name of the Column if there are similar columns in DataFrame.
- **lsuffix**: To change the left DataFrame name.
- **rsuffix**: To change the right DataFrame name
- arguments: {str, default ""}

```
In [106]: df_1 = pd.DataFrame({'A': [1, 2, 3, ],
                             'B': [10, 20, 30, ]},
                             index = ['a', 'b', 'c'])

# We make a same column name 'A'
df_2 = pd.DataFrame({'A': [4, 5],
                     'D': [40, 50]},
                     index = ['a', 'b'])

join_df = df_1.join(df_2, lsuffix = '_1')

print(join_df)
```

	A_1	B	A	D
a	1	10	4.0	40.0
b	2	20	5.0	50.0
c	3	30	NaN	NaN

```
In [107]: df_1 = pd.DataFrame({'A':[1,2,3,],
                             'B':[10,20,30,]},
                             index = ['a','b','c'])

# We make a same column name 'A'
df_2 = pd.DataFrame({"A": [4,5],
                     'D': [40,50]},
                     index = ['a','b'])

join_df = df_1.join(df_2, rsuffix = '_1')

print(join_df)
```

	A	B	A_1	D
a	1	10	4.0	40.0
b	2	20	5.0	50.0
c	3	30	NaN	NaN

append()

- Pandas append function is used to append rows of other DataFrame to the end of the given DataFrame, returning a new DataFrame object.
- Syntax: **DataFrame.append()**

Parameters of **append()**

- **other,**
- **ignore_index=False,**
- **verify_integrity=False,**
- **sort=False**

```
In [108]: df_1 = pd.DataFrame({'A':[1,2,3,],
                             'B':[10,20,30]})

df_2 = pd.DataFrame({"A": [4,5,6],
                     'B': [40,50,60]})

append_df = df_1.append(df_2)

print(append_df)
```

	A	B
0	1	10
1	2	20
2	3	30
0	4	40
1	5	50
2	6	60

ignore_index

- **ignore_index** parameter use for print the index sequence wise.
- arguments:{bool, default False}

```
In [109]: df_1 = pd.DataFrame({'A':[1,2,3,],
                             'B':[10,20,30]})

df_2 = pd.DataFrame({"A": [4,5,6],
                     'B': [40,50,60]})

append_df = df_1.append(df_2, ignore_index = True)
```

```
print(append_df)
```

	A	B
0	1	10
1	2	20
2	3	30
3	4	40
4	5	50
5	6	60

sort

- **sort** parameter used to concat the DataFrame of different columns.
- arguments:{ bool, default False}

In [110..

```
df_1 = pd.DataFrame({'A':[1,2,3,],  
                     'B':[10,20,30]})
```

```
df_2 = pd.DataFrame({'C':[4,5,6],  
                     'B':[40,50,60]})
```

```
append_df = df_1.append(df_2, ignore_index = True, sort = False)
```

```
print(append_df)
```

	A	B	C
0	1.0	10	NaN
1	2.0	20	NaN
2	3.0	30	NaN
3	NaN	40	4.0
4	NaN	50	5.0
5	NaN	60	6.0

In []: