In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
mydataset = {
    'cars': ["BMW", "Volvo", "Ford"],
    'passings': [3, 7, 2]
}
myvar = pd.DataFrame(mydataset)
print(myvar)
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

In [3]:
```
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

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

In [4]:
```
a = [1, 7, 2]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar)
print(myvar["y"])
```

```
x    1
y    7
z    2
dtype: int64
7
```

In [5]:
```
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories)
print(myvar)
```

```
day1    420
day2    380
day3    390
dtype: int64
```

In [6]:

```
calories = {"day1": 420, "day2": 380, "day3": 390}

myvar = pd.Series(calories, index = ["day1", "day2"])

print(myvar)
```

```
day1    420
day2    380
dtype: int64
```

In [7]:

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
myvar = pd.DataFrame(data)
print(myvar)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

In [8]:

```
print(myvar)
print()
print(myvar.loc[0])
```

```
   calories  duration
0       420        50
1       380        40
2       390        45

calories    420
duration     50
Name: 0, dtype: int64
```

In [9]:

```
print(myvar)
print()
print(myvar.loc[[0,1]])
```

```
   calories  duration
0       420        50
1       380        40
2       390        45

   calories  duration
0       420        50
1       380        40
```

In [10]:

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45
```

In [11]:

```
print(df)
print()
print(df.loc["day2"])
```

```
      calories  duration
day1       420        50
day2       380        40
day3       390        45

calories    380
duration     40
Name: day2, dtype: int64
```

In [12]:

```
df = pd.read_csv('taxi.csv')
#print(df)
print(df.to_string())
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

In [13]:

```
df = pd.read_json('data.json')
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     300.5
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	300.5
37	60	100	120	300.1
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	300.1
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     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

In [14]:

```
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     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

In [15]:

```
data = {
  "Duration":{
    "0":60,
    "1":60,
    "2":60
  },
  "Pulse":{
    "0":110,
    "1":117,
```

```
    "2":103
  },
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135
  },
  "Calories":{
    "0":409,
    "1":479,
    "2":340
  }
}

df = pd.DataFrame(data)

print(df)
```

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130       409
1        60    117       145       479
2        60    103       135       340
```

In [16]:

```
df = pd.read_json('data.json')
print(df.head(10))
```

```
   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     300.5
6        60    110       136     374.0
7        45    104       134     253.3
8        30    109       133     195.1
9        60     98       124     269.0
```

In [17]:

```
df.tail(10)
```

Out[17]:

	Duration	Pulse	Maxpulse	Calories
159	30	80	120	240.9

	Duration	Pulse	Maxpulse	Calories
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	300.4
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Duration  169 non-null    int64  
 1   Pulse     169 non-null    int64  
 2   Maxpulse  169 non-null    int64  
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 6.6 KB
```

In [19]:
```
df = pd.read_csv('dumpdata.csv')
print(df)
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
```

```
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
18        45  '2020/12/18'     90       112       NaN
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
22        45           NaN    100       119     282.0
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
28        60  '2020/12/28'    103       132       NaN
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

In [20]:

```
new_df = df.dropna()
print(new_df.to_string())
```

```
    Duration          Date  Pulse  Maxpulse  Calories
0         60  '2020/12/01'    110       130     409.1
1         60  '2020/12/02'    117       145     479.0
2         60  '2020/12/03'    103       135     340.0
3         45  '2020/12/04'    109       175     282.4
4         45  '2020/12/05'    117       148     406.0
5         60  '2020/12/06'    102       127     300.0
6         60  '2020/12/07'    110       136     374.0
7        450  '2020/12/08'    104       134     253.3
8         30  '2020/12/09'    109       133     195.1
9         60  '2020/12/10'     98       124     269.0
10        60  '2020/12/11'    103       147     329.3
11        60  '2020/12/12'    100       120     250.7
```

```
12        60  '2020/12/12'    100       120     250.7
13        60  '2020/12/13'    106       128     345.3
14        60  '2020/12/14'    104       132     379.3
15        60  '2020/12/15'     98       123     275.0
16        60  '2020/12/16'     98       120     215.2
17        60  '2020/12/17'    100       120     300.0
19        60  '2020/12/19'    103       123     323.0
20        45  '2020/12/20'     97       125     243.0
21        60  '2020/12/21'    108       131     364.2
23        60  '2020/12/23'    130       101     300.0
24        45  '2020/12/24'    105       132     246.0
25        60  '2020/12/25'    102       126     334.5
26        60      20201226    100       120     250.0
27        60  '2020/12/27'     92       118     241.0
29        60  '2020/12/29'    100       132     280.0
30        60  '2020/12/30'    102       129     380.3
31        60  '2020/12/31'     92       115     243.0
```

In [21]:

```
df = pd.read_csv('dumpdata.csv')
df['Date'] = pd.to_datetime(df['Date'])
print(df)
```

```
    Duration       Date  Pulse  Maxpulse  Calories
0         60 2020-12-01    110       130     409.1
1         60 2020-12-02    117       145     479.0
2         60 2020-12-03    103       135     340.0
3         45 2020-12-04    109       175     282.4
4         45 2020-12-05    117       148     406.0
5         60 2020-12-06    102       127     300.0
6         60 2020-12-07    110       136     374.0
7        450 2020-12-08    104       134     253.3
8         30 2020-12-09    109       133     195.1
9         60 2020-12-10     98       124     269.0
10        60 2020-12-11    103       147     329.3
11        60 2020-12-12    100       120     250.7
12        60 2020-12-12    100       120     250.7
13        60 2020-12-13    106       128     345.3
14        60 2020-12-14    104       132     379.3
15        60 2020-12-15     98       123     275.0
16        60 2020-12-16     98       120     215.2
17        60 2020-12-17    100       120     300.0
18        45 2020-12-18     90       112       NaN
19        60 2020-12-19    103       123     323.0
```

```
20        45 2020-12-20     97       125     243.0
21        60 2020-12-21    108       131     364.2
22        45        NaT    100       119     282.0
23        60 2020-12-23    130       101     300.0
24        45 2020-12-24    105       132     246.0
25        60 2020-12-25    102       126     334.5
26        60 2020-12-26    100       120     250.0
27        60 2020-12-27     92       118     241.0
28        60 2020-12-28    103       132       NaN
29        60 2020-12-29    100       132     280.0
30        60 2020-12-30    102       129     380.3
31        60 2020-12-31     92       115     243.0
```

In [77]:

```
sales = pd.read_csv('city_sales.csv')
print(sales)
```

```
                      date  num       city
0        2015-01-01 09:00:00    4     London
1        2015-01-01 09:01:00    4     London
2        2015-01-01 09:02:00    3     London
3        2015-01-01 09:03:00    3     London
4        2015-01-01 09:04:00    3     London
...                      ...  ...        ...
1795139  2019-01-31 15:56:00    3  Cambridge
1795140  2019-01-31 15:57:00    3  Cambridge
1795141  2019-01-31 15:58:00    3  Cambridge
1795142  2019-01-31 15:59:00    3  Cambridge
1795143  2019-01-31 16:00:00    2  Cambridge

[1795144 rows x 3 columns]
```

In [22]:

```
print(df.duplicated())
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
```

```
9     False
10    False
11    False
12     True
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
27    False
28    False
29    False
30    False
31    False
dtype: bool
```

In [23]:

```
df.corr()
```

Out[23]:

	Duration	Pulse	Maxpulse	Calories
Duration	1.000000	0.004410	0.049959	-0.114169
Pulse	0.004410	1.000000	0.276583	0.513186
Maxpulse	0.049959	0.276583	1.000000	0.357460
Calories	-0.114169	0.513186	0.357460	1.000000

In [79]:

```
students = pd.DataFrame({
    'name' : ['Ashish', "Manish", 'Shikha', 'Uma', 'Pradeep', 'Neha', 'Mahi'],
    'age' : [ 10, 22, 13, 21, 12, 11, 17],
    'section' : [ 'A', 'B', 'C', 'B', 'B', 'A', 'A'],
    'city' : [ 'Gurgaon', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai'],
```

```
    'gender' : [ 'M', 'F', 'F', 'M', 'M', 'M', 'F'],
    'favourite_color' : [ 'red', np.NaN , 'yellow', np.NAN, 'black', 'green', 'red']
})
students
```

Out[79]:

	name	age	section	city	gender	favourite_color
0	Ashish	10	A	Gurgaon	M	red
1	Manish	22	B	Delhi	F	NaN
2	Shikha	13	C	Mumbai	F	yellow
3	Uma	21	B	Delhi	M	NaN
4	Pradeep	12	B	Mumbai	M	black
5	Neha	11	A	Delhi	M	green
6	Mahi	17	A	Mumbai	F	red

In [25]:

```
print(students.loc[1])
```

```
name               Manish
age                    22
section                 B
city                Delhi
gender                  F
favourite_color       NaN
Name: 1, dtype: object
```

In [26]:

```
#print selected rows
students.loc[0:2]
```

Out[26]:

	name	age	section	city	gender	favourite_color
0	Ashish	10	A	Gurgaon	M	red
1	Manish	22	B	Delhi	F	NaN
2	Shikha	13	C	Mumbai	F	yellow

In [80]:
```
#print selected columns and and selected rows
students.loc[0:2,['name','age','city']]
```

Out[80]:

	name	age	city
0	Ashish	10	Gurgaon
1	Manish	22	Delhi
2	Shikha	13	Mumbai

In [82]:
```
#print rows with condition
print(students)
students.age >= 15
```

```
      name  age section     city gender favourite_color
0   Ashish   10       A  Gurgaon      M             red
1   Manish   22       B    Delhi      F             NaN
2   Shikha   13       C   Mumbai      F          yellow
3      Uma   21       B    Delhi      M             NaN
4  Pradeep   12       B   Mumbai      M           black
5     Neha   11       A    Delhi      M           green
6     Mahi   17       A   Mumbai      F             red
```

Out[82]:
```
0    False
1     True
2    False
3     True
4    False
5    False
6     True
Name: age, dtype: bool
```

In [83]:
```
#use above condition as selection criteria
students.loc[students.age >= 15,['name','city']]
```

Out[83]:

	name	city
1	Manish	Delhi
3	Uma	Delhi

	name	city
6	Mahi	Mumbai

In [30]:
```
#multiple conditions
students.loc[(students.age >= 15) & (students.city == "Delhi")]
```

Out[30]:

	name	age	section	city	gender	favourite_color
1	Manish	22	B	Delhi	F	NaN
3	Uma	21	B	Delhi	M	NaN

In [31]:
```
#update gender of second record
students.loc[1,['gender']] = ['M']
#update multiple rows with same
students.loc[0:2, ['section', 'favourite_color']] = ['A','Red']
students
```

Out[31]:

	name	age	section	city	gender	favourite_color
0	Ashish	10	A	Gurgaon	M	Red
1	Manish	22	A	Delhi	M	Red
2	Shikha	13	A	Mumbai	F	Red
3	Uma	21	B	Delhi	M	NaN
4	Pradeep	12	B	Mumbai	M	black
5	Neha	11	A	Delhi	M	green
6	Mahi	17	A	Mumbai	F	red

In [32]:
```
#update multiple rows with condition
students.loc[(students.age >= 20), ['section', 'city']] = ['S','Pune']
students
```

Out[32]:

	name	age	section	city	gender	favourite_color
0	Ashish	10	A	Gurgaon	M	Red
1	Manish	22	S	Pune	M	Red
2	Shikha	13	A	Mumbai	F	Red
3	Uma	21	S	Pune	M	NaN
4	Pradeep	12	B	Mumbai	M	black
5	Neha	11	A	Delhi	M	green
6	Mahi	17	A	Mumbai	F	red

When we are using iloc, we need to specify the rows and columns by their integer index. If we want to select only the first and third row, we simply need to put this into a list in the iloc statement with our dataframe

In [87]:
```
# Selecting two rows number 1 and number 3
students.iloc[[1,3,5]]
```

Out[87]:

	name	age	section	city	gender	favourite_color
1	Manish	22	B	Delhi	F	NaN
3	Uma	21	B	Delhi	M	NaN
5	Neha	11	A	Delhi	M	green

In [89]:
```
students.iloc[[0,6],[0,1,2]]
```

Out[89]:

	name	age	section
0	Ashish	10	A
6	Mahi	17	A

In [34]:
```
# Select rows with particular indices and particular columns
# Selecting rows 0 and 2 and selecting column number 1 and 3
# first we provide row numbers and after column that column numbers
students.iloc[[0,2,3],[0,3,4]]
```

Out[34]:

	name	city	gender
0	Ashish	Gurgaon	M
2	Shikha	Mumbai	F
3	Uma	Pune	M

In [35]:
```
# selecting range using iloc
# high number is exclusive
students.iloc[0:3]
```

Out[35]:

	name	age	section	city	gender	favourite_color
0	Ashish	10	A	Gurgaon	M	Red
1	Manish	22	S	Pune	M	Red
2	Shikha	13	A	Mumbai	F	Red

In [36]:
```
students.iloc[0:2, 0:3]
```

Out[36]:

	name	age	section
0	Ashish	10	A
1	Manish	22	S

In [37]:
```
columns = list(students.columns)
print(columns)
```

```
['name', 'age', 'section', 'city', 'gender', 'favourite_color']
```

In [38]:
```
#rename columns name
columns[1] = "Student Age"
students.columns = columns
students
```

Out[38]:

	name	Student Age	section	city	gender	favourite_color
0	Ashish	10	A	Gurgaon	M	Red
1	Manish	22	S	Pune	M	Red
2	Shikha	13	A	Mumbai	F	Red
3	Uma	21	S	Pune	M	NaN
4	Pradeep	12	B	Mumbai	M	black
5	Neha	11	A	Delhi	M	green
6	Mahi	17	A	Mumbai	F	red

GroupBy Example

In [39]:
```
students
```

Out[39]:

	name	Student Age	section	city	gender	favourite_color
0	Ashish	10	A	Gurgaon	M	Red
1	Manish	22	S	Pune	M	Red
2	Shikha	13	A	Mumbai	F	Red
3	Uma	21	S	Pune	M	NaN
4	Pradeep	12	B	Mumbai	M	black
5	Neha	11	A	Delhi	M	green
6	Mahi	17	A	Mumbai	F	red

In [90]:
```
# it will group the data but will not do anything
students.groupby('city').count()
```

Out[90]:

city	name	age	section	gender	favourite_color
Delhi	3	3	3	3	1
Gurgaon	1	1	1	1	1
Mumbai	3	3	3	3	3

In [41]:
```
students.groupby('gender').count()
```

Out[41]:

gender	name	Student Age	section	city	favourite_color
F	2	2	2	2	2
M	5	5	5	5	4

In [42]:
```
# Group by on multiple columns
students.groupby(['city','gender']).count()
```

Out[42]:

city	gender	name	Student Age	section	favourite_color
Delhi	**M**	1	1	1	1
Gurgaon	**M**	1	1	1	1
Mumbai	**F**	2	2	2	2
	M	1	1	1	1
Pune	**M**	2	2	2	1

In [43]:
```
# Find count for specific column
#students.groupby('gender')['age'].mean()  -> it will not work as we have changes its name
students.groupby('gender')['Student Age'].mean()
```

Out[43]:
```
gender
F    15.0
M    15.2
Name: Student Age, dtype: float64
```

In [44]:
```
grp = students.groupby('city')
grp.groups
```

Out[44]:
```
{'Delhi': [5], 'Gurgaon': [0], 'Mumbai': [2, 4, 6], 'Pune': [1, 3]}
```

In [45]:
```
#printing group data in readable form
for name,group in grp:
    print(name,'contains',group.shape[0],'rows')
```

```
Delhi contains 1 rows
Gurgaon contains 1 rows
Mumbai contains 3 rows
Pune contains 2 rows
```

In [46]:
```
# We can get even specifc group
grp.get_group('Delhi')
```

Out[46]:

	name	Student Age	section	city	gender	favourite_color
5	Neha	11	A	Delhi	M	green

In [47]:
```
grp.get_group('Mumbai')
```

Out[47]:

	name	Student Age	section	city	gender	favourite_color
2	Shikha	13	A	Mumbai	F	Red
4	Pradeep	12	B	Mumbai	M	black

	name	Student Age	section	city	gender	favourite_color
6	Mahi	17	A	Mumbai	F	red

In [48]:
```
students
```

Out[48]:

	name	Student Age	section	city	gender	favourite_color
0	Ashish	10	A	Gurgaon	M	Red
1	Manish	22	S	Pune	M	Red
2	Shikha	13	A	Mumbai	F	Red
3	Uma	21	S	Pune	M	NaN
4	Pradeep	12	B	Mumbai	M	black
5	Neha	11	A	Delhi	M	green
6	Mahi	17	A	Mumbai	F	red

Working with Missing Data in Pandas

In Pandas missing data is represented by two value:

None: None is a Python singleton object that is often used for missing data in Python code. NaN : NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation

isnull() notnull() dropna() fillna() replace() interpolate()

In [49]:
```
# dictionary of lists
dict = {'First Score':['Male', 'Female', np.nan, 'Female'],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from list
df = pd.DataFrame(dict)
df
```

Out[49]:

	First Score	Second Score	Third Score
0	Male	30.0	NaN
1	Female	45.0	40.0
2	NaN	56.0	80.0
3	Female	NaN	98.0

In [50]:
```
# Check if any value in DF is null
df.isnull()
```

Out[50]:

	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

In [51]:
```
# Get count of NaN values from all columns
df.isnull().sum()
```

Out[51]:
```
First Score     1
Second Score    1
Third Score     1
dtype: int64
```

In [52]:
```
# Check for NaN in one specific column
pd.isnull(df['First Score'])
```

Out[52]:
```
0    False
1    False
2     True
3    False
Name: First Score, dtype: bool
```

In [53]:
```
# opposite of isnull
df.notnull()
```

Out[53]:

	First Score	Second Score	Third Score
0	True	True	False
1	True	True	True
2	False	True	True
3	True	False	True

Filling missing values

In [54]:
```
# Fill missing value with 0
# df.fillna(0)
# Fill missing value with average
print(df['Second Score'].mean())
avg_second = df['Second Score'].mean()
# it will fill all values of all columns
df.fillna(avg_second)
```

```
43.666666666666664
```

Out[54]:

	First Score	Second Score	Third Score
0	Male	30.000000	43.666667
1	Female	45.000000	40.000000
2	43.666667	56.000000	80.000000
3	Female	43.666667	98.000000

In [55]:
```
avg_second = df['Second Score'].mean()
#df['Second Score'] = df['Second Score'].fillna(avg_second)

df
```

Out[55]:

	First Score	Second Score	Third Score
0	Male	30.0	NaN
1	Female	45.0	40.0
2	NaN	56.0	80.0
3	Female	NaN	98.0

In [56]:

```
avg_second = round(df['Second Score'].mean())
avg_third = round(df['Third Score'].mean())
print(avg_second, avg_third)
#df = df.fillna({'Third Score':avg_third})
df = df.fillna({'Second Score':avg_second,'Third Score':avg_third})

print(df)
```

```
44 73
  First Score  Second Score  Third Score
0        Male          30.0         73.0
1      Female          45.0         40.0
2         NaN          56.0         80.0
3      Female          44.0         98.0
```

Merging, Joining, and Concatenating DataFrame

Concatenating DataFrame using .concat() :

The concat() function (in the main pandas namespace) does all of the heavy lifting of performing concatenation operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes. Note that I say "if any" because there is only a single possible axis of concatenation for Series.

In [57]:

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])
```

```
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                    index=[4, 5, 6, 7])


df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                    index=[8, 9, 10, 11])

#Putting all DF's in list
frames = [df1, df2, df3]
#Concatinating them
result = pd.concat(frames)

print(result)
```

```
     A    B    C    D
0    A0   B0   C0   D0
1    A1   B1   C1   D1
2    A2   B2   C2   D2
3    A3   B3   C3   D3
4    A4   B4   C4   D4
5    A5   B5   C5   D5
6    A6   B6   C6   D6
7    A7   B7   C7   D7
8    A8   B8   C8   D8
9    A9   B9   C9   D9
10  A10  B10  C10  D10
11  A11  B11  C11  D11
```

In [91]:

```
print(df1)
```

```
    A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
2  A2  B2  C2  D2
3  A3  B3  C3  D3
```

In [94]:
```
#Common indexes
df4 = pd.DataFrame({'E': ['B2', 'B3', 'B6', 'B7'],
                    'F': ['D2', 'D3', 'D6', 'D7'],
                    'G': ['F2', 'F3', 'F6', 'F7']},
                   index=[2, 3, 6, 7])
print(df4)
```

```
    E   F   G
2  B2  D2  F2
3  B3  D3  F3
6  B6  D6  F6
7  B7  D7  F7
```

In [95]:
```
result = pd.concat([df1, df4], axis=1)
print(result)
```

```
     A    B    C    D    E    F    G
0   A0   B0   C0   D0  NaN  NaN  NaN
1   A1   B1   C1   D1  NaN  NaN  NaN
2   A2   B2   C2   D2   B2   D2   F2
3   A3   B3   C3   D3   B3   D3   F3
6  NaN  NaN  NaN  NaN   B6   D6   F6
7  NaN  NaN  NaN  NaN   B7   D7   F7
```

Concatenating with Series

In [59]:
```
s2 = pd.Series(['_0', '_1', '_2', '_3'])
result = pd.concat([df1, s2, s2], axis=1)

print(result)
```

```
    A   B   C   D   0   1
0  A0  B0  C0  D0  _0  _0
1  A1  B1  C1  D1  _1  _1
2  A2  B2  C2  D2  _2  _2
3  A3  B3  C3  D3  _3  _3
```

Concatenating DataFrame using .append() :

A useful shortcut to concat() are the append() instance methods on Series and DataFrame. These methods actually predated concat. They concatenate along axis=0, namely the index.

In [60]:

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])


df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                    index=[4, 5, 6, 7])

result = df1.append(df2)

print(result)
```

```
    A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
2  A2  B2  C2  D2
3  A3  B3  C3  D3
4  A4  B4  C4  D4
5  A5  B5  C5  D5
6  A6  B6  C6  D6
7  A7  B7  C7  D7
```

Concatenating DataFrame using .merge() :

When you want to combine data objects based on one or more keys in a similar way to a relational database, merge() is the tool you need. More specifically, merge() is most useful when you want to combine rows that share data.

**pandas provides a single function, merge(), as the entry point for all standard database join operations between DataFrame or named Series objects. **

There are THREE types of operation in merge

one-to-one joins: for example when joining two DataFrame objects on their indexes (which must contain unique values).
many-to-one joins: for example when joining an index (unique) to one or more columns in a different DataFrame.
many-to-many joins: joining columns on columns.

In [96]:

```
left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                     'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']})

print(left)

right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})
print(right)

result = pd.merge(left, right, on='key')

print(result)
```

```
  key   A   B
0  K0  A0  B0
1  K1  A1  B1
2  K2  A2  B2
3  K3  A3  B3
  key   C   D
0  K0  C0  D0
1  K1  C1  D1
2  K2  C2  D2
3  K3  C3  D3
  key   A   B   C   D
0  K0  A0  B0  C0  D0
1  K1  A1  B1  C1  D1
2  K2  A2  B2  C2  D2
3  K3  A3  B3  C3  D3
```

Concatinating Dataframe using .join():

In [62]:

```
# Define a dictionary containing employee data
data1 = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32]}
```

```
# Define a dictionary containing employee data
data2 = {'Address':['Allahabad', 'Kannuaj', 'Allahabad', 'Kannuaj'],
        'Qualification':['MCA', 'Phd', 'Bcom', 'B.hons']}

# Convert the dictionary into DataFrame
name_age = pd.DataFrame(data1,index=['K0', 'K1', 'K2', 'K3'])
name_age
```

Out[62]:

	Name	Age
K0	Jai	27
K1	Princi	24
K2	Gaurav	22
K3	Anuj	32

In [63]:

```
# Convert the dictionary into DataFrame
add_qua = pd.DataFrame(data2, index=['K0', 'K2', 'K3', 'K4'])
add_qua
```

Out[63]:

	Address	Qualification
K0	Allahabad	MCA
K2	Kannuaj	Phd
K3	Allahabad	Bcom
K4	Kannuaj	B.hons

In [64]:

```
# Joining Dataframe
# Based on initial DF, you will see indexes
name_age.join(add_qua)
```

Out[64]:

	Name	Age	Address	Qualification
K0	Jai	27	Allahabad	MCA

	Name	Age	Address	Qualification
K1	Princi	24	NaN	NaN
K2	Gaurav	22	Kannuaj	Phd
K3	Anuj	32	Allahabad	Bcom

In [65]:
```
add_qua.join(name_age)
```

Out[65]:

	Address	Qualification	Name	Age
K0	Allahabad	MCA	Jai	27.0
K2	Kannuaj	Phd	Gaurav	22.0
K3	Allahabad	Bcom	Anuj	32.0
K4	Kannuaj	B.hons	NaN	NaN

In [66]:
```
# Outer Join
name_age.join(add_qua, how='outer')
```

Out[66]:

	Name	Age	Address	Qualification
K0	Jai	27.0	Allahabad	MCA
K1	Princi	24.0	NaN	NaN
K2	Gaurav	22.0	Kannuaj	Phd
K3	Anuj	32.0	Allahabad	Bcom
K4	NaN	NaN	Kannuaj	B.hons

In [67]:
```
add_qua.join(name_age,how="outer")
```

Out[67]:

	Address	Qualification	Name	Age
K0	Allahabad	MCA	Jai	27.0

	Address	Qualification	Name	Age
K1	NaN	NaN	Princi	24.0
K2	Kannuaj	Phd	Gaurav	22.0
K3	Allahabad	Bcom	Anuj	32.0
K4	Kannuaj	B.hons	NaN	NaN

In [68]:
```
# Inner Join
name_age.join(add_qua, how='inner')
```

Out[68]:

	Name	Age	Address	Qualification
K0	Jai	27	Allahabad	MCA
K2	Gaurav	22	Kannuaj	Phd
K3	Anuj	32	Allahabad	Bcom

Pivot & Pivot Table

In [98]:
```
df = pd.read_csv("weather.csv")
df
```

Out[98]:

	date	city	temperature	humidity
0	5/1/2017	new york	65	56
1	5/2/2017	new york	66	58
2	5/3/2017	new york	68	60
3	5/1/2017	mumbai	75	80
4	5/2/2017	mumbai	78	83
5	5/3/2017	mumbai	82	85
6	5/1/2017	beijing	80	26

	date	city	temperature	humidity
7	5/2/2017	beijing	77	30
8	5/3/2017	beijing	79	35

In [99]:
```
df.pivot(index='city',columns='date')
```

Out[99]:

	temperature			humidity		
date	5/1/2017	5/2/2017	5/3/2017	5/1/2017	5/2/2017	5/3/2017
city						
beijing	80	77	79	26	30	35
mumbai	75	78	82	80	83	85
new york	65	66	68	56	58	60

In [100…
```
df.pivot(index='city',columns='date',values="temperature")
```

Out[100…

date	5/1/2017	5/2/2017	5/3/2017
city			
beijing	80	77	79
mumbai	75	78	82
new york	65	66	68

In [72]:
```
df.pivot(index='date',columns='city')
```

Out[72]:

	temperature			humidity		
city	beijing	mumbai	new york	beijing	mumbai	new york
date						
5/1/2017	80	75	65	26	80	56
5/2/2017	77	78	66	30	83	58
5/3/2017	79	82	68	35	85	60

Pivot Table

In [73]:

```
df = pd.read_csv("weather2.csv")
df
```

Out[73]:

	date	city	temperature	humidity
0	5/1/2017	new york	65	56
1	5/1/2017	new york	61	54
2	5/2/2017	new york	70	60
3	5/2/2017	new york	72	62
4	5/1/2017	mumbai	75	80
5	5/1/2017	mumbai	78	83
6	5/2/2017	mumbai	82	85
7	5/2/2017	mumbai	80	26

In [74]:

```
# by default it calculate mean
df.pivot_table(index="city",columns="date")
# df.pivot_table(index="city",columns="date", aggfunc="mean")
```

Out[74]:

	humidity		temperature	
date	5/1/2017	5/2/2017	5/1/2017	5/2/2017
city				
mumbai	81.5	55.5	76.5	81.0
new york	55.0	61.0	63.0	71.0

In [75]:
```
df.pivot_table(index="city",columns="date", margins = "true")
```

Out[75]:

	humidity			temperature		
date	5/1/2017	5/2/2017	All	5/1/2017	5/2/2017	All
city						
mumbai	81.50	55.50	68.50	76.50	81.0	78.750
new york	55.00	61.00	58.00	63.00	71.0	67.000
All	68.25	58.25	63.25	69.75	76.0	72.875

In [104…
```
df.pivot_table(index="city",columns="date", aggfunc="mean")
```

Out[104…

	humidity			temperature		
date	5/1/2017	5/2/2017	5/3/2017	5/1/2017	5/2/2017	5/3/2017
city						
beijing	26	30	35	80	77	79
mumbai	80	83	85	75	78	82
new york	56	58	60	65	66	68