IOTA ACADEMY
Become Job Ready...

Merging, Joining, and Concatenating

pandas provides various facilities for easily combining together Series or DataFrame : Merging, Joining and Concatenating.

In [39]:
```python
import pandas as pd
from IPython.display import display
```

1. Concatenating

`concat()`

The concat function does all of the heavy lifting of performing concatenation operations along an axis.

- **Syntax:** pd.concat(objs, *, axis=0, join='outer', ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, sort=False, copy=True)

In [40]:
```python
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])

print("DataFrame:1")
display(df1)

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])
print("\nDataFrame:2")
display(df2)

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])

print("\nDataFrame:3")
display(df3)
```

DataFrame:1

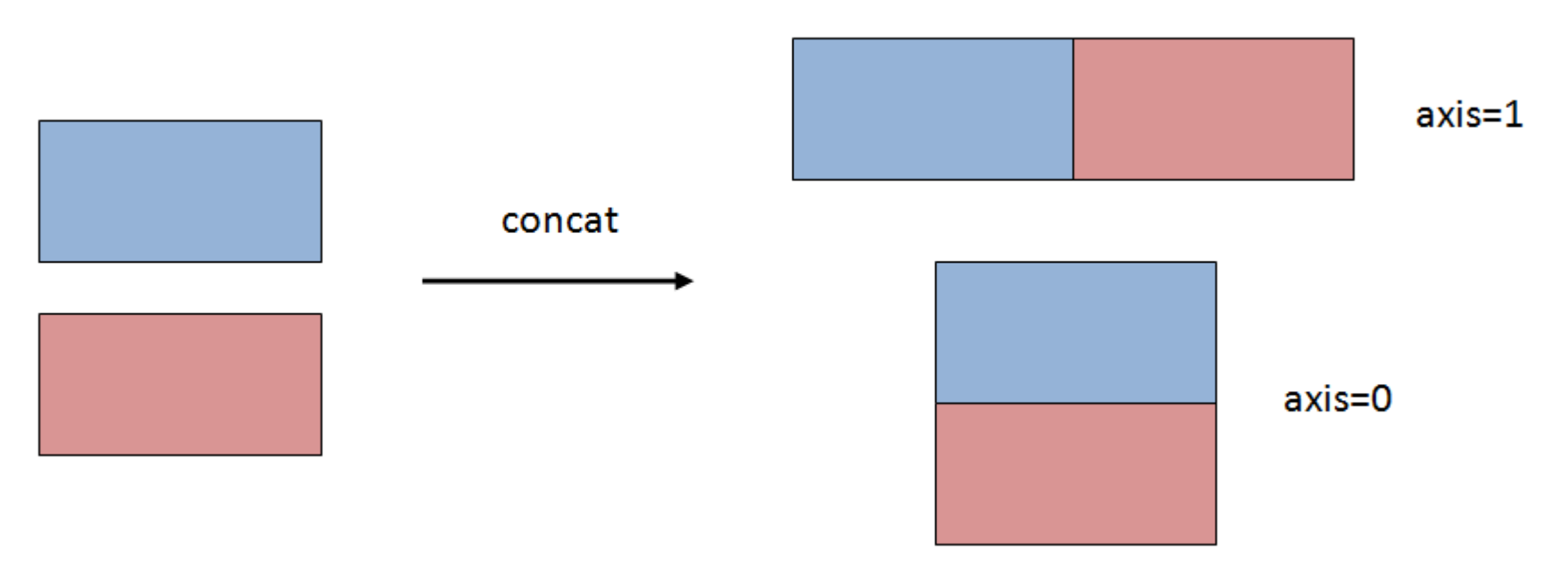
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

DataFrame:2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

DataFrame:3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11



!

In [41]:
```python
# apply concat method
# It will concatenate all DataFrames vertically.
pd.concat([df1,df2,df3])  # default axis is 0
```

Out[41]:

	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 [42]:
```python
# change axis
# It will concatenate all DataFrames horizontally.
pd.concat([df1,df2,df3],axis=1)  # axis : {0/'index/rows', 1/'columns'}
```

Out[42]:

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

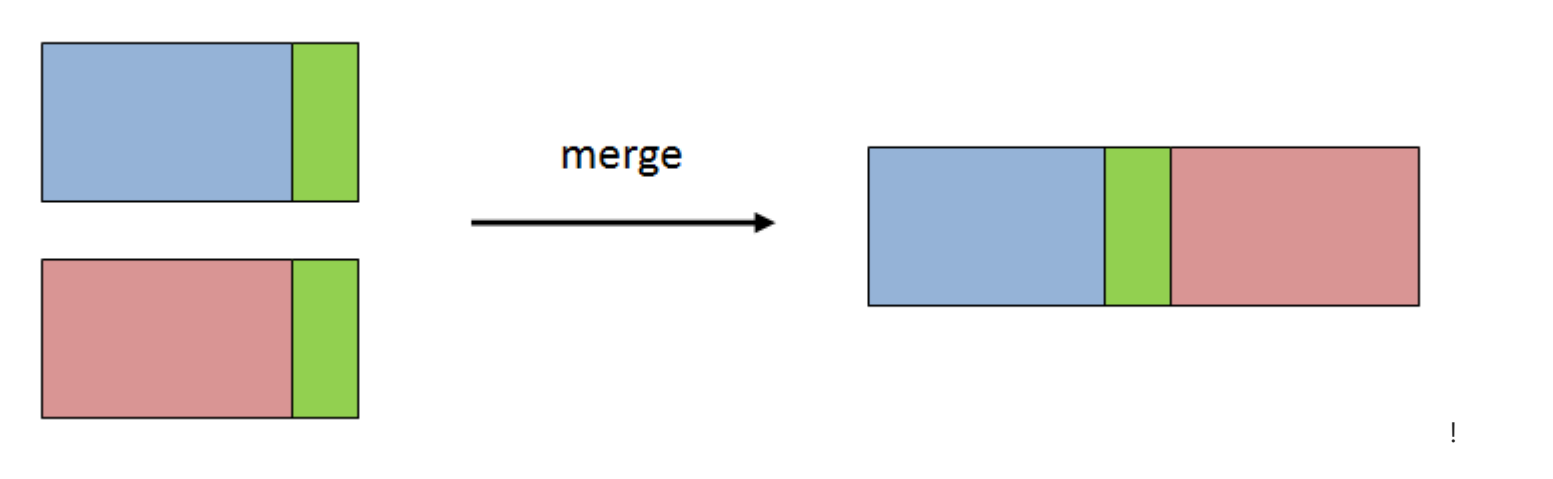
In [43]:
```python
# Use the key argument to give each DataFrame its own name.
pd.concat([df1,df2,df3],keys=["DF1","DF2","DF3"]) # multiindexing
```

Out[43]:

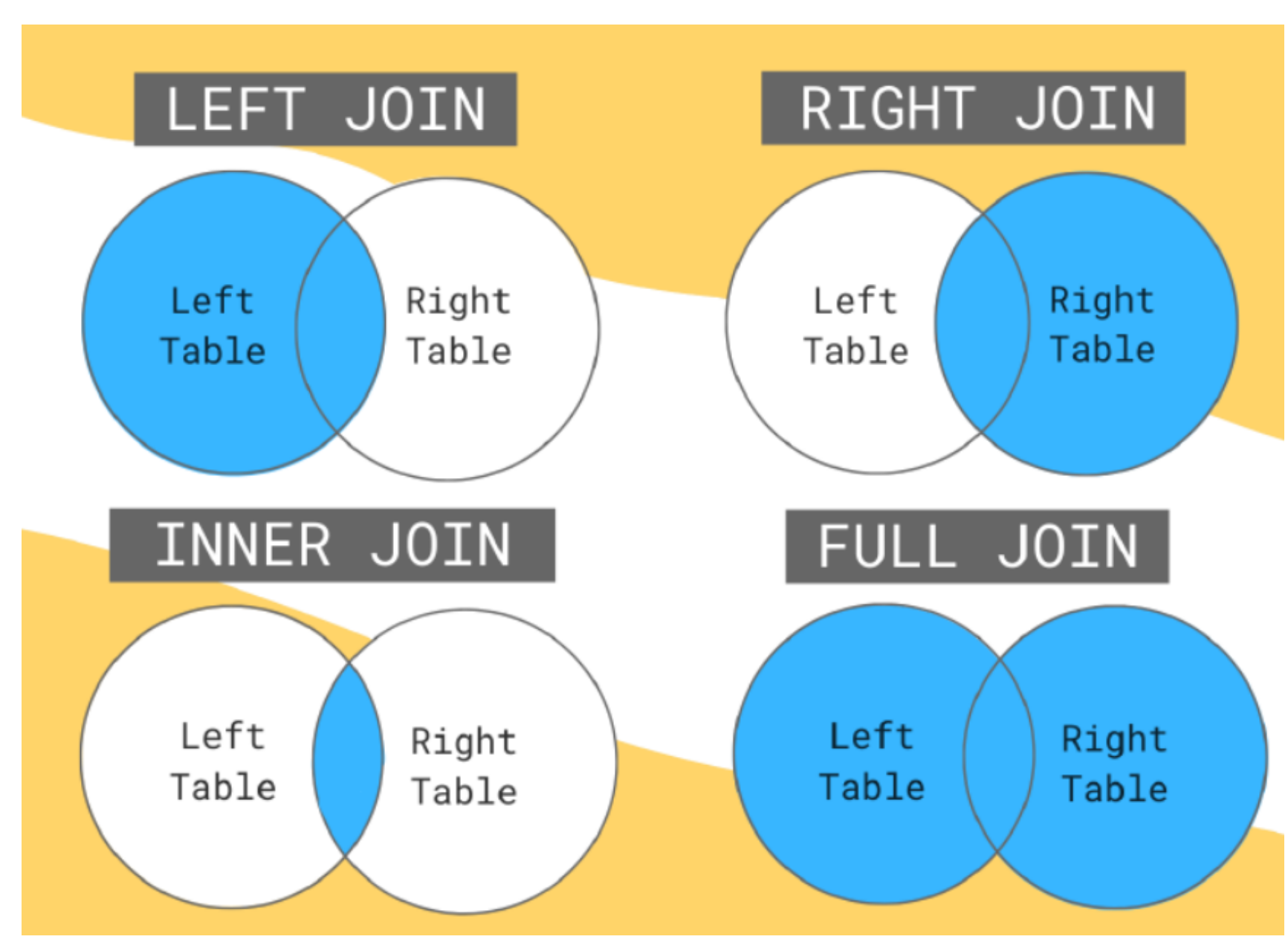
		A	B	C	D
DF1	0	A0	B0	C0	D0
	1	A1	B1	C1	D1
	2	A2	B2	C2	D2
	3	A3	B3	C3	D3
DF2	4	A4	B4	C4	D4
	5	A5	B5	C5	D5
	6	A6	B6	C6	D6
	7	A7	B7	C7	D7
DF3	8	A8	B8	C8	D8
	9	A9	B9	C9	D9
	10	A10	B10	C10	D10
	11	A11	B11	C11	D11

2.Merging

The **merge** function allows you to merge DataFrames together using a similar logic as merging SQL Tables together. For example:



!



`merge()`

Merge DataFrame or named Series objects with a database-style join.

- **Syntax:** pd.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=True, indicator=False, validate=None)

In [44]:
```python
# Create DataFrame
student = pd.DataFrame({"Name":list("ABCDEF"),
                        "Age":list(range(23,29)),
                        "Roll_num":[100,200,300,400,1000,900,],
                       "Subject":["Math","Math","Biology","Art","Commerece","Biology"]})

display(student)

region = pd.DataFrame({"Roll_num":[100,200,300,400,500,600,700,800],
                       "city":["Indore","Bhopal"]*4})

display(region)
```

	Name	Age	Roll_num	Subject
0	A	23	100	Math
1	B	24	200	Math
2	C	25	300	Biology
3	D	26	400	Art
4	E	27	1000	Commerece
5	F	28	900	Biology

	Roll_num	city
0	100	Indore
1	200	Bhopal
2	300	Indore
3	400	Bhopal
4	500	Indore
5	600	Bhopal
6	700	Indore
7	800	Bhopal

a) Inner join

- inner join is the most common type of join you'll be working with. It returns a dataframe with only rows where the merge "on" value present in both the left and right dataframes.

In [45]:
```python
# merging
pd.merge(student,region,how='inner',on='Roll_num') # Default Merging - inner join
```

Out[45]:

	Name	Age	Roll_num	Subject	city
0	A	23	100	Math	Indore
1	B	24	200	Math	Bhopal
2	C	25	300	Biology	Indore
3	D	26	400	Art	Bhopal

b) left join, or left merge,

- keeps every row from the left dataframe.Rows in the left dataframe that have no corresponding join value in the right dataframe are left with NaN values.

In [46]:
```python
pd.merge(student,region,on="Roll_num",how="left")

# There isn't any data for roll number 1000 and 900 in the region table.
# so there will be NaN
```

Out[46]:

	Name	Age	Roll_num	Subject	city
0	A	23	100	Math	Indore
1	B	24	200	Math	Bhopal
2	C	25	300	Biology	Indore
3	D	26	400	Art	Bhopal
4	E	27	1000	Commerece	NaN
5	F	28	900	Biology	NaN

c) right merge / right join

keeps every row from the right dataframe.Rows in the right dataframe that have no corresponding join value in the left dataframe are left with NaN values.

In [47]:
```python
pd.merge(student,region,on="Roll_num",how="right")
```

Out[47]:

	Name	Age	Roll_num	Subject	city
0	A	23.0	100	Math	Indore
1	B	24.0	200	Math	Bhopal
2	C	25.0	300	Biology	Indore
3	D	26.0	400	Art	Bhopal
4	NaN	NaN	500	NaN	Indore
5	NaN	NaN	600	NaN	Bhopal
6	NaN	NaN	700	NaN	Indore
7	NaN	NaN	800	NaN	Bhopal

d) Outer join

Outer join is also called Full Outer Join that returns all rows from both pandas DataFrames. Where join expression doesn't match it returns null on respective cells.

In [48]:
```python
pd.merge(student,region,on="Roll_num",how="outer")
```

Out[48]:

	Name	Age	Roll_num	Subject	city
0	A	23.0	100	Math	Indore
1	B	24.0	200	Math	Bhopal
2	C	25.0	300	Biology	Indore
3	D	26.0	400	Art	Bhopal
4	E	27.0	1000	Commerece	NaN
5	F	28.0	900	Biology	NaN
6	NaN	NaN	500	NaN	Indore
7	NaN	NaN	600	NaN	Bhopal
8	NaN	NaN	700	NaN	Indore
9	NaN	NaN	800	NaN	Bhopal

merge on the basis of two common column

In [49]:
```python
df_users = pd.DataFrame({
    'Year': [2019,2019,2019,2019,2019,2019,2019,2019,2019,2019,2019,2019,2020,2020,2020],
    'Quarter': ['Q1','Q1','Q1','Q2','Q2','Q2','Q3','Q3','Q3','Q4','Q4','Q4','Q1','Q1','Q1'],
    'Month': ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec','Jan','Feb','Mar'],
    'Users': [150,170,160,200,190,196,210,225,260,210,212,219,630,598,321]
})

# advertising partners
df_ad_partners = pd.DataFrame({
    'Year': [2019,2019,2019,2019,2020],
    'Quarter': ['Q1','Q2','Q3','Q4','Q1'],
    'adv_pat':["A","A","A","A","B"]
})
```

In [50]:
```python
pd.merge(df_users, df_ad_partners, on=['Year', 'Quarter'])  # default is inner
```

Out[50]:

	Year	Quarter	Month	Users	adv_pat
0	2019	Q1	Jan	150	A
1	2019	Q1	Feb	170	A
2	2019	Q1	Mar	160	A
3	2019	Q2	Apr	200	A
4	2019	Q2	May	190	A
5	2019	Q2	Jun	196	A
6	2019	Q3	Jul	210	A
7	2019	Q3	Aug	225	A
8	2019	Q3	Sep	260	A
9	2019	Q4	Oct	210	A
10	2019	Q4	Nov	212	A
11	2019	Q4	Dec	219	A
12	2020	Q1	Jan	630	B
13	2020	Q1	Feb	598	B
14	2020	Q1	Mar	321	B

if the column names in the two tables are different

In [51]:
```python
# Create DataFrame
customer=pd.DataFrame({
    'id':[1,2,3,4,5,6,7,8,9],
    'name':['Olivia','Aditya','Cory','Isabell','Dominic','Tyler','Samuel','Daniel','Jeremy'],
    'age':[20,25,15,10,30,65,35,18,23],
    'Product_ID':[101,0,106,0,103,104,0,0,107],
    'Product_name':['Watch','NA','Oil','NA','Shoes','Smartphone','NA','NA','Laptop'],
    'City':['Mumbai','Delhi','Bangalore','Chennai','Chennai','Delhi','Kolkata','Delhi','Mumbai']
})

display(customer)

product_dup=pd.DataFrame({
'Product_ID':[101,102,103,104,105,106,107,103,107],

'Purchased_Product':['Watch','Bag','Shoes','Smartphone','Books','Oil','Laptop','Shoes','Laptop'],
'Category':['Fashion','Fashion','Fashion','Electronics','Study','Grocery','Electronics','Fashion','Electronics'
'Price':[299.0,1350.50,2999.0,14999.0,145.0,110.0,79999.0,2999.0,79999.0],
'Seller_City':['Delhi','Mumbai','Chennai','Kolkata','Delhi','Chennai','Bengalore','Chennai','Bengalore']})

display(product_dup)
```

	id	name	age	Product_ID	Product_name	City
0	1	Olivia	20	101	Watch	Mumbai
1	2	Aditya	25	0	NA	Delhi
2	3	Cory	15	106	Oil	Bangalore
3	4	Isabell	10	0	NA	Chennai
4	5	Dominic	30	103	Shoes	Chennai
5	6	Tyler	65	104	Smartphone	Delhi
6	7	Samuel	35	0	NA	Kolkata
7	8	Daniel	18	0	NA	Delhi
8	9	Jeremy	23	107	Laptop	Mumbai

	Product_ID	Purchased_Product	Category	Price	Seller_City
0	101	Watch	Fashion	299.0	Delhi
1	102	Bag	Fashion	1350.5	Mumbai
2	103	Shoes	Fashion	2999.0	Chennai
3	104	Smartphone	Electronics	14999.0	Kolkata
4	105	Books	Study	145.0	Delhi
5	106	Oil	Grocery	110.0	Chennai
6	107	Laptop	Electronics	79999.0	Bengalore
7	103	Shoes	Fashion	2999.0	Chennai
8	107	Laptop	Electronics	79999.0	Bengalore

if the column names are different in the two dataframes that contain same data, Then,we have to explicitly mention both the column names. We have two parameters for that: "left on" and "right on."

In [52]:
```python
pd.merge(customer,product_dup,left_on="Product_name",right_on="Purchased_Product")
```

Out[52]:

	id	name	age	Product_ID_x	Product_name	City	Product_ID_y	Purchased_Product	Category	Price	Seller_City
0	1	Olivia	20	101	Watch	Mumbai	101	Watch	Fashion	299.0	Delhi
1	3	Cory	15	106	Oil	Bangalore	106	Oil	Grocery	110.0	Chennai
2	5	Dominic	30	103	Shoes	Chennai	103	Shoes	Fashion	2999.0	Chennai
3	5	Dominic	30	103	Shoes	Chennai	103	Shoes	Fashion	2999.0	Chennai
4	6	Tyler	65	104	Smartphone	Delhi	104	Smartphone	Electronics	14999.0	Kolkata
5	9	Jeremy	23	107	Laptop	Mumbai	107	Laptop	Electronics	79999.0	Bengalore
6	9	Jeremy	23	107	Laptop	Mumbai	107	Laptop	Electronics	79999.0	Bengalore

In [53]:
```python
pd.merge(product_dup,customer,on='Product_ID',how='outer',indicator=True)

# We just have to mention the indicator argument as True in the function,
#  and a new column of name _merge will be created in the resulting dataframe:
```

Out[53]:

	Product_ID	Purchased_Product	Category	Price	Seller_City	id	name	age	Product_name	City	_merge
0	101	Watch	Fashion	299.0	Delhi	1.0	Olivia	20.0	Watch	Mumbai	both
1	102	Bag	Fashion	1350.5	Mumbai	NaN	NaN	NaN	NaN	NaN	left_only
2	103	Shoes	Fashion	2999.0	Chennai	5.0	Dominic	30.0	Shoes	Chennai	both
3	103	Shoes	Fashion	2999.0	Chennai	5.0	Dominic	30.0	Shoes	Chennai	both
4	104	Smartphone	Electronics	14999.0	Kolkata	6.0	Tyler	65.0	Smartphone	Delhi	both
5	105	Books	Study	145.0	Delhi	NaN	NaN	NaN	NaN	NaN	left_only
6	106	Oil	Grocery	110.0	Chennai	3.0	Cory	15.0	Oil	Bangalore	both
7	107	Laptop	Electronics	79999.0	Bengalore	9.0	Jeremy	23.0	Laptop	Mumbai	both
8	107	Laptop	Electronics	79999.0	Bengalore	9.0	Jeremy	23.0	Laptop	Mumbai	both
9	0	NaN	NaN	NaN	NaN	2.0	Aditya	25.0	NA	Delhi	right_only
10	0	NaN	NaN	NaN	NaN	4.0	Isabell	10.0	NA	Chennai	right_only
11	0	NaN	NaN	NaN	NaN	7.0	Samuel	35.0	NA	Kolkata	right_only
12	0	NaN	NaN	NaN	NaN	8.0	Daniel	18.0	NA	Delhi	right_only

3. Joining

Joining is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame.

In [54]:
```python
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                     'B': ['B0', 'B1', 'B2']},
                      index=['K0', 'K1', 'K2'])

right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                    'D': ['D0', 'D2', 'D3']},
                      index=['K0', 'K2', 'K3'])
```

In [55]:
```python
left
```

Out[55]:

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

In [56]:
```python
right
```

Out[56]:

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

In [57]:
```python
left.join(right)
```

Out[57]:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

In [58]:
```python
left.join(right, how='outer')
```

Out[58]:

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

Both join and merge can be used to combines two dataframes but the join method combines two dataframes on the basis of their indexes whereas the merge method is more versatile and allows us to specify columns beside the index to join on for both dataframes.

Great Job!