Chapter 8

Combining and Merging Datasets

- Database-Style DataFrame Joins
- Merging on Index
- Concatenating Along an Axis
- Combining Data with Overlap

```
* combine simple ye hota hai ke 1 ke peechay 1 lagado ya 1 ke aage 1 lagado just like concatenation
* merging sabko miladeta hai or unko sort bhi kardeta hai,uske darmiyaan values fit hojati hain

* Why we need to combine or merge ? Because we will receive data, or wo data zaruri nhi hai ke 1 hee table ho mult
iple table bhi hosakte hain
* data different sources sai bhi asakta hai
```

Combining and Merging Datasets

Data contained in pandas objects can be combined together in a number of ways:

- **pandas.merge** connects rows in DataFrames based on one or more keys. This will be familiar to users of SQL or other relational databases, as it implements database join operations.
- **pandas.concat** concatenates or “stacks” together objects along an axis.
- The **combine_first** instance method enables splicing together overlapping data to fill in missing values in one object with values from another.

Database-Style DataFrame Joins

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

In [29]:
```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],'data1': range(7)})
df1
```

Out[29]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

In [30]:
```
df2 = pd.DataFrame({'key': ['a', 'b', 'd','b'],'data2': range(4)})
df2
```

Out[30]:

	key	data2
0	a	0
1	b	1
2	d	2
3	b	3

In [31]:
```
# many to one join
pd.merge(df1, df2)

# Note that I didn't specify which column to join on. If that information is not specified, merge uses the..
# ..overlapping column names as the keys.

# ye usi col ki base par merge karega jis col ki values ki type same hogi, "key(numeric, string) >> impossibl
e"
# pehle same name choose karega col ka

# yahan c or d reh gya qk usko key nhi mila rahi merge karte waqt
```

Out[31]:

	key	data1	data2
0	b	0	1
1	b	0	3
2	b	1	1
3	b	1	3
4	b	6	1
5	b	6	3
6	a	2	0
7	a	4	0
8	a	5	0

In [32]:
```
pd.merge(df2, df1)
```

Out[32]:

	key	data2	data1
0	a	0	2
1	a	0	4
2	a	0	5
3	b	1	0
4	b	1	1
5	b	1	6
6	b	3	0
7	b	3	1
8	b	3	6

In [33]:
```
print(df1)
print()
print(df2)
print()
print(pd.merge(df1, df2, on='key'))
```

```
  key  data1
0   b      0
1   b      1
2   a      2
3   c      3
4   a      4
5   a      5
6   b      6

  key  data2
0   a      0
1   b      1
2   d      2
3   b      3

  key  data1  data2
0   b      0      1
1   b      0      3
2   b      1      1
3   b      1      3
4   b      6      1
5   b      6      3
6   a      2      0
7   a      4      0
8   a      5      0
```

In [34]:
```
#If the column names are different in each object, you can specify them separately:
df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],'data1': range(7)})
df3
```

Out[34]:

	lkey	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

In [35]:
```
df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],'data2': range(3)})
df4
```

Out[35]:

	rkey	data2
0	a	0
1	b	1
2	d	2

In [36]:
```
pd.merge(df3, df4, left_on='lkey', right_on='rkey',how='outer') #outer >>> union , inner >>> intersection
```

Out[36]:

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	c	3.0	NaN	NaN
7	NaN	NaN	d	2.0

Many To Many Join

Table 8-1. Different join types with how argument

Option	Behavior
`'inner'`	Use only the key combinations observed in both tables
`'left'`	Use all key combinations found in the left table
`'right'`	Use all key combinations found in the right table
`'output'`	Use all key combinations observed in both tables together

- Many-to-many joins form the Cartesian product of the rows.
- Suppose there were three 'b' rows in the left DataFrame and two in the right one, there are six 'b' rows in the result.
- The join method only affects the distinct key values appearing in the result.
- The default merge method is to intersect the join keys, you can instead form the union of them with an outer join

```
left join wohi data karega jo left mai hai agar right mai hai or left mai nhi to chor dega
```

In [39]:
```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],'data1': range(6)})
df1
```

Out[39]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	b	5

In [40]:
```
df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],'data2': range(5)})
df2
```

Out[40]:

	key	data2
0	a	0
1	b	1
2	a	2
3	b	3
4	d	4

In [41]:
```
pd.merge(df1, df2, on='key', how='left')
```

Out[41]:

	key	data1	data2
0	b	0	1.0
1	b	0	3.0
2	b	1	1.0
3	b	1	3.0
4	a	2	0.0
5	a	2	2.0
6	c	3	NaN
7	a	4	0.0
8	a	4	2.0
9	b	5	1.0
10	b	5	3.0

In [42]:
```
df1
```

Out[42]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	b	5

In [43]:
```
df2
```

Out[43]:

	key	data2
0	a	0
1	b	1
2	a	2
3	b	3
4	d	4

In [44]:
```
pd.merge(df1, df2, on='key', how='right')
```

Out[44]:

	key	data1	data2
0	b	0.0	1
1	b	1.0	1
2	b	5.0	1
3	b	0.0	3
4	b	1.0	3
5	b	5.0	3
6	a	2.0	0
7	a	4.0	0
8	a	2.0	2
9	a	4.0	2
10	d	NaN	4

In [45]:
```
pd.merge(df1, df2, on='key', how='inner')
```

Out[45]:

	key	data1	data2
0	b	0	1
1	b	0	3
2	b	1	1
3	b	1	3
4	b	5	1
5	b	5	3
6	a	2	0
7	a	2	2
8	a	4	0
9	a	4	2

In [46]:
```
df1
```

Out[46]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	b	5

In [47]:
```
df2
```

Out[47]:

	key	data2
0	a	0
1	b	1
2	a	2
3	b	3
4	d	4

In [48]:
```
pd.merge(df1, df2, on='key', how='outer')
```

Out[48]:

	key	data1	data2
0	b	0.0	1.0
1	b	0.0	3.0
2	b	1.0	1.0
3	b	1.0	3.0
4	b	5.0	1.0
5	b	5.0	3.0
6	a	2.0	0.0
7	a	2.0	2.0
8	a	4.0	0.0
9	a	4.0	2.0
10	c	3.0	NaN
11	d	NaN	4.0

- To merge with multiple keys, pass a list of column names
- To determine which key combinations will appear in the result depending on the choice of merge method, think of the multiple keys as forming an array of tuples to be used as a single join key (even though it's not actually implemented that way)

In [49]:
```
left = pd.DataFrame({'userId': ['foo', 'foo', 'bar'],'userName': ['one', 'two', 'one'],'lval': [1, 2, 3]})
left
```

Out[49]:

	userId	userName	lval
0	foo	one	1
1	foo	two	2
2	bar	one	3

In [50]:
```
right = pd.DataFrame({'userId': ['foo', 'foo', 'bar', 'bar'],'userName': ['one', 'one', 'one', 'two'],'rval': [4, 5, 6, 7]})
right
```

Out[50]:

	userId	userName	rval
0	foo	one	4
1	foo	one	5
2	bar	one	6
3	bar	two	7

In [51]:
```
pd.merge(left, right, on=['userId','userName'],how='right')
```

Out[51]:

	userId	userName	lval	rval
0	foo	one	1.0	4
1	foo	one	1.0	5
2	bar	one	3.0	6
3	bar	two	NaN	7

In [52]:
```
pd.merge(left, right, on=['userId','userName'],how='left')
```

Out[52]:

	userId	userName	lval	rval
0	foo	one	1	4.0
1	foo	one	1	5.0
2	foo	two	2	NaN
3	bar	one	3	6.0

In [53]:
```
pd.merge(right,left, on=['userId','userName'],how='right')
```

Out[53]:

	userId	userName	rval	lval
0	foo	one	4.0	1
1	foo	one	5.0	1
2	bar	one	6.0	3
3	foo	two	NaN	2

In [54]:
```
pd.merge(right,left, on=['userId','userName'],how='left')
```

Out[54]:

	userId	userName	rval	lval
0	foo	one	4	1.0
1	foo	one	5	1.0
2	bar	one	6	3.0
3	bar	two	7	NaN

Merging on Index

In some cases, the merge key(s) in a DataFrame will be found in its index. In this case, you can pass left_index=True or right_index=True (or both) to indicate that the index should be used as the merge key

```
There is no columns or values overlapping based on which we could do merging so in this case we will merge based o
n the index
```

In [55]:
```
left1 = pd.DataFrame({'key': ['c', 'd', 'a', 'a', 'b', 'c'],'value': range(6)})
left1
```

Out[55]:

	key	value
0	c	0
1	d	1
2	a	2
3	a	3
4	b	4
5	c	5

In [56]:
```
right1 = pd.DataFrame({'group_val': [3.5, 7]})
right1
```

Out[56]:

	group_val
0	3.5
1	7.0

In [57]:
```
pd.merge(left1, right1,left_index=True,right_index=True)
```

Out[57]:

	key	value	group_val
0	c	0	3.5
1	d	1	7.0

In [58]:
```
pd.merge(left1, right1, left_index=True, right_index=True, how='outer')  # outer mai union lega
```

Out[58]:

	key	value	group_val
0	c	0	3.5
1	d	1	7.0
2	a	2	NaN
3	a	3	NaN
4	b	4	NaN
5	c	5	NaN

In [60]:
```
pd.merge(left1, right1, left_index=True, right_index=True, how='inner')  # inner mai intersection lega
```

Out[60]:

	key	value	group_val
0	c	0	3.5
1	d	1	7.0

DataFrame has a convenient join instance for merging by index. It can also be used to combine together many DataFrame objects having the same or similar indexes but non-overlapping columns.

In [61]:
```
left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],index=['a', 'c', 'e'],columns=['Ohio', 'Nevada'])
left2
```

Out[61]:

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

In [62]:
```
right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14]],index=['b', 'c', 'd', 'e'],columns=['Missouri', 'Alabama'])
right2
```

Out[62]:

	Missouri	Alabama
b	7.0	8.0
c	9.0	10.0
d	11.0	12.0
e	13.0	14.0

In [63]:
```
left2.join(right2, how='inner')
```

Out[63]:

	Ohio	Nevada	Missouri	Alabama
c	3.0	4.0	9.0	10.0
e	5.0	6.0	13.0	14.0

In [64]:
```
another = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],index=['a', 'c', 'g', 'f'],columns=['New York', 'Oregon'])
another
```

Out[64]:

	New York	Oregon
a	7.0	8.0
c	9.0	10.0
g	11.0	12.0
f	16.0	17.0

In [65]:
```
result = left2.join([right2, another], how='outer')
```

In [66]:
```
result
```

Out[66]:

	Ohio	Nevada	Missouri	Alabama	New York	Oregon
a	1.0	2.0	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0	9.0	10.0
e	5.0	6.0	13.0	14.0	NaN	NaN
b	NaN	NaN	7.0	8.0	NaN	NaN
d	NaN	NaN	11.0	12.0	NaN	NaN
g	NaN	NaN	NaN	NaN	11.0	12.0
f	NaN	NaN	NaN	NaN	16.0	17.0

In [67]:
```
result = left2.join([right2, another], how='inner')
result
```

Out[67]:

	Ohio	Nevada	Missouri	Alabama	New York	Oregon
c	3.0	4.0	9.0	10.0	9.0	10.0

Combining Data with Overlap

In [68]:
```
df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],'b': [np.nan, 2., np.nan, 6.],'c': range(2, 18, 4)})
df1
```

Out[68]:

	a	b	c
0	1.0	NaN	2
1	NaN	2.0	6
2	5.0	NaN	10
3	NaN	6.0	14

In [69]:
```
df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],'b': [np.nan, 3., 4., 6., 8.]})
df2
```

Out[69]:

	a	b
0	5.0	NaN
1	4.0	3.0
2	NaN	4.0
3	3.0	6.0
4	7.0	8.0

In [70]:
```
df1.combine_first(df2)
```

Out[70]:

	a	b	c
0	1.0	NaN	2.0
1	4.0	2.0	6.0
2	5.0	4.0	10.0
3	3.0	6.0	14.0
4	7.0	8.0	NaN

In [71]:
```
# Example :
```

In [1]:
```
import requests
url = 'https://jsonplaceholder.typicode.com/todos/'
resp = requests.get(url)
```

In [2]:
```
resp
```

Out[2]:
```
<Response [200]>
```

In [3]:
```
data = resp.json()
len(data)
```

Out[3]:
```
200
```

In [6]:
```
pd.DataFrame(data)
```

Out[6]:

	userId	id	title	completed
0	1	1	delectus aut autem	False
1	1	2	quis ut nam facilis et officia qui	False
2	1	3	fugiat veniam minus	False
3	1	4	et porro tempora	True
4	1	5	laboriosam mollitia et enim quasi adipisci qui...	False
...	...	...	...	...
195	10	196	consequuntur aut ut fugit similique	True
196	10	197	dignissimos quo nobis earum saepe	True
197	10	198	quis eius est sint explicabo	True
198	10	199	numquam repellendus a magnam	True
199	10	200	ipsam aperiam voluptates qui	False

200 rows × 4 columns

In [7]:
```
new_data = pd.DataFrame(data[:3])
```

In [8]:
```
new_data
```

Out[8]:

	userId	id	title	completed
0	1	1	delectus aut autem	False
1	1	2	quis ut nam facilis et officia qui	False
2	1	3	fugiat veniam minus	False

In [9]:
```
import sqlite3
```

In [13]:
```
command = "CREATE TABLE test (id INTEGER, userName VARCHAR(20));"
```

In [14]:
```
con = sqlite3.connect('mydata2.sqlite')  # agar database nhi hai to create kardega
con
```

Out[14]:
```
<sqlite3.Connection at 0xa235b48>
```

In [15]:
```
con.execute(command)
```

Out[15]:
```
<sqlite3.Cursor at 0x2e756e0>
```

In [16]:
```
con.commit()
```

In [17]:
```
data = [(1, 'Jonathan'),(2, 'Saqib'),(3, 'Umair'), (222,2344)]
```

In [18]:
```
stmt = "INSERT INTO test VALUES(?, ?)"
con.executemany(stmt, (data))
con.commit()
```

In [19]:
```
data
```

Out[19]:
```
[(1, 'Jonathan'), (2, 'Saqib'), (3, 'Umair'), (222, 2344)]
```

In [20]:
```
cursor = con.execute('select * from test')
```

In [21]:
```
cursor
```

Out[21]:
```
<sqlite3.Cursor at 0x2e759e0>
```

In [22]:
```
rows = cursor.fetchall()
rows
```

Out[22]:
```
[(1, 'Jonathan'), (2, 'Saqib'), (3, 'Umair'), (222, '2344')]
```

In [23]:
```
another_data = pd.DataFrame(rows)
another_data
```

Out[23]:

	0	1
0	1	Jonathan
1	2	Saqib
2	3	Umair
3	222	2344

In [24]:
```
another_data.drop_duplicates(inplace=True)
```

In [25]:
```
new_data
```

Out[25]:

	userId	id	title	completed
0	1	1	delectus aut autem	False
1	1	2	quis ut nam facilis et officia qui	False
2	1	3	fugiat veniam minus	False

In [26]:
```
another_data
```

Out[26]:

	0	1
0	1	Jonathan
1	2	Saqib
2	3	Umair
3	222	2344

In [27]:
```
result = new_data.join(another_data)
```

In [28]:
```
result
```

Out[28]:

	userId	id	title	completed	0	1
0	1	1	delectus aut autem	False	1	Jonathan
1	1	2	quis ut nam facilis et officia qui	False	2	Saqib
2	1	3	fugiat veniam minus	False	3	Umair

In [29]:
```
con.close()
```

In [30]:
```
# another_data.rename(columns={0:'id',1:'name'},inplace=True)
```

In [31]:
```
# pd.merge(new_data,another_data,on=['id'])
```

In [ ]:
```

```