# Data Wrangling: Join, Combine, and Reshape

Part 2

## Combining and Merging Datasets

Part 1

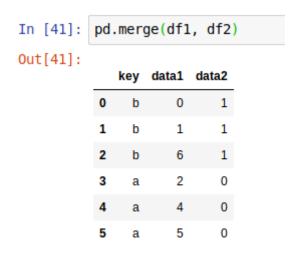
- Data contained in pandas objects can be combined together in a number of ways:
  - pandas.merge connects rows in DataFrames based on one or more keys. This will be familiar to users of SQL or other relational databases, as it implements database join operations.
  - pandas.concat concatenates or "stacks" 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

- *Merge* or *join* operations combine datasets by linking rows using one or more keys.
- These operations are central to relational databases (e.g., SQL-based).
- The merge function in pandas is the main entry point for using these algorithms on your data.

```
In [38]: df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
                   'data2': range(3)})
In [39]: df1
Out[39]:
        key data1
      2 a
             3
      4 a
             5
         a
      6 b
             6
In [40]: df2
Out[40]:
        key data2
             0
      2 d
             2
```

- This is an example of a many-to-one join; the data in df1 has multiple rows labeled a and b, whereas df2 has only one row for each value in the key column.
- Calling merge with these objects we obtain:

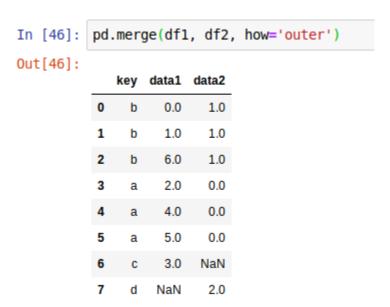


- Note that we didn't specify which column to join on.
- If that information is not specified, merge uses the overlapping column names as the keys.
- It's a good practice to specify explicitly, though:

[42]:	pd	.mer	ge(df1	, df2,
ut[42]:		kev	data1	data2
	0	b		1
	1	b	1	1
	2	b	6	1
	3	a	2	0
	4	a	4	0
	5	a	5	0

• If the column names are different in each object, you can specify them separately:

- You may notice that the 'c' and 'd' values and associated data are missing from the result.
- By default merge does an 'inner' join; the keys in the result are the intersection, or the common set found in both tables.
- Other possible options are 'left', 'right', and 'outer'.
- The outer join takes the union of the keys, combining the effect of applying both left and right joins:



Many-to-many merges have well-defined, though not necessarily intuitive, behavior.

```
In [47]: df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                              'data1': range(6)})
In [48]: df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
                              'data2': range(5)})
In [49]: df1
Out[49]:
            key data1
                    0
              а
              а
                    5
In [50]: df2
Out[50]:
            key data2
                    0
                    2
```

In [51]: pd.merge(df1, df2, on='key', how='left') Out[51]: key data1 data2 0 1.0 3.0 0 1 1.0 3.0 0.0 2 2 2.0 NaN 3 0.0 2.0 1.0

3.0

10

In [52]: pd.merge(df1, df2, how='inner')

#### Out[52]:

	key	data1	data2
C	) b	0	1
1	L b	0	3
2	2 b	1	1
3	<b>b</b>	1	3
4	b b	5	1
5	<b>b</b>	5	3
6	a a	2	0
7	a	2	2
8	a a	4	0
9	a	4	2

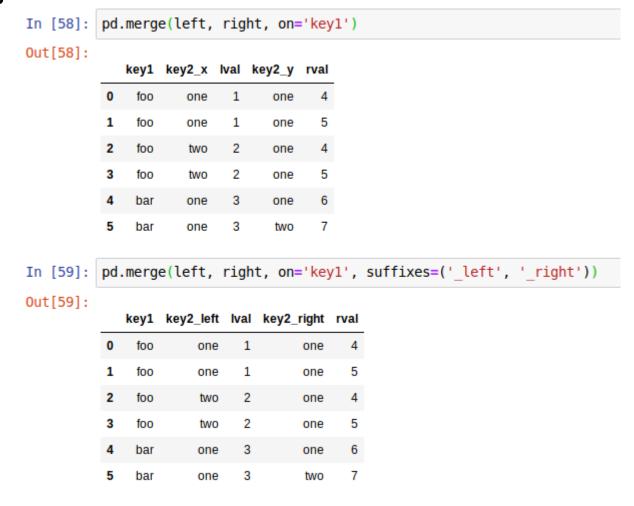
• To merge with multiple keys, pass a list of column names:

```
In [53]: left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                               'key2': ['one', 'two', 'one'],
                               'lval': [1, 2, 3]})
In [54]: right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                                'key2': ['one', 'one', 'one', 'two'],
                                'rval': [4, 5, 6, 7]})
In [55]: left
Out[55]:
            key1 key2 lval
                  two
             bar one
In [56]: right
Out[56]:
            key1 key2 rval
              foo one
                  one
             bar
                  two
```

In [57]:	pd	.merg	e(left	t, ri	ght,	on=['key1',	'key2'],	how='outer')
Out[57]:		key1	key2	lval	rval			
	0	foo	one	1.0	4.0			
	1	foo	one	1.0	5.0			

two NaN 7.0

- A last issue to consider in merge operations is the treatment of overlapping column names.
- While you can address the overlap, merge has a suffixes option for specifying strings to append to overlapping names in the left and right DataFrame objects:



## 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.

```
In [61]: right1 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
In [62]: left1
Out[62]:
        key value
             5
         C
In [63]: right1
Out[63]:
        group_val
            3.5
            7.0
       b
```

In [64]: pd.merge(left1, right1, left\_on='key', right\_index=True)

#### Out[64]:

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0

In [65]: pd.merge(left1, right1, left\_on='key', right\_index=True, how='outer')

#### Out[65]:

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

• With hierarchically indexed data, things are more complicated.

In [68]: lefth

Out[68]:

	key1	key2	data
0	Ohio	2000	0.0
1	Ohio	2001	1.0
2	Ohio	2002	2.0
3	Nevada	2001	3.0
4	Nevada	2002	4.0

In [69]: righth

Out[69]:

		event1	event2
Nevada	2001	0	1
	2000	2	3
Ohio	2000	4	5
	2000	6	7
	2001	8	9
	2002	10	11

In [70]: pd.merge(lefth, righth, left\_on=['key1', 'key2'], right\_index=True)

Out[70]:

	k	ey1	key2	data	event1	event2
(	0 (	Ohio	2000	0.0	4	5
(	0 (	Ohio	2000	0.0	6	7
:	1 (	Ohio	2001	1.0	8	9
2	2 (	Ohio	2002	2.0	10	11
:	3 Nev	ada	2001	3.0	0	1

Out[71]:

	key1	key2	data	event1	event2
0	Ohio	2000	0.0	4.0	5.0
0	Ohio	2000	0.0	6.0	7.0
1	Ohio	2001	1.0	8.0	9.0
2	Ohio	2002	2.0	10.0	11.0
3	Nevada	2001	3.0	0.0	1.0
4	Nevada	2002	4.0	NaN	NaN
4	Nevada	2000	NaN	2.0	3.0

• Using the indexes of both sides of the merge is also possible:

b NaN

c 3.0

d NaN

e 5.0

NaN

4.0

NaN

6.0

```
In [72]: left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],
                               index=['a', 'c', 'e'],
                               columns=['Ohio', 'Nevada'])
In [73]: right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14]],
                                index=['b', 'c', 'd', 'e'],
                                columns=['Missouri', 'Alabama'])
In [74]: left2
Out[74]:
             Ohio Nevada
          a 1.0
                     2.0
              3.0
                     4.0
          e 5.0
                     6.0
In [75]: right2
Out[75]:
             Missouri Alabama
                 7.0
                         8.0
                 9.0
                        10.0
                11.0
                        12.0
                13.0
                        14.0
```

In [76]: pd.merge(left2, right2, how='outer', left\_index=True, right\_index=True)
Out[76]:
 Ohio Nevada Missouri Alabama
 a 1.0 2.0 NaN NaN

8.0

10.0

12.0

14.0

7.0

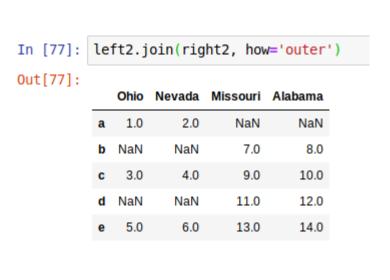
9.0

11.0

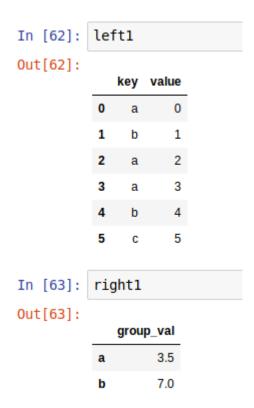
13.0

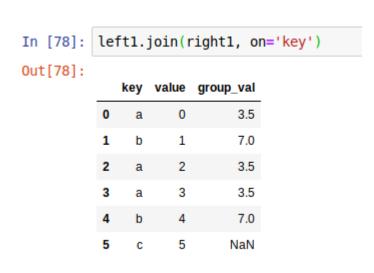
• DataFrame has a convenient join instance for merging by index.





• DataFrame's join method also supports joining the index of the passed DataFrame on one of the columns of the calling DataFrame:





• Lastly, for simple index-on-index merges, you can pass a list of DataFrames to join as an alternative to using the more general concat function:

```
In [79]: another = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],
                                   index=['a', 'c', 'e', 'f'],
                                   columns=['New York', 'Oregon'])
In [80]: another
Out[80]:
             New York Oregon
                          8.0
                         10.0
                  9.0
                 11.0
                         12.0
                 16.0
                         17.0
In [81]: left2
Out[81]:
             Ohio Nevada
              1.0
                      2.0
              3.0
                      4.0
              5.0
                      6.0
In [82]: right2
Out[82]:
             Missouri Alabama
                  7.0
                          8.0
                  9.0
                         10.0
                 11.0
                         12.0
                 13.0
                         14.0
```

In [83]:	33]: left2.join([right2, another])									
Out[83]:		Ohio	Nevada	Missouri	Alabama	New York	Oregon			
	a	1.0	2.0	NaN	NaN	7.0	8.0			
	С	3.0	4.0	9.0	10.0	9.0	10.0			
	е	5.0	6.0	13.0	14.0	11.0	12.0			
In [84]:	/h	<pre>left2.join([right2, another], how='outer') /home/joshua/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py nation axis is not aligned. A future version</pre>								
	То	of pandas will change to not sort by default.  To accept the future behavior, pass 'sort=False'.  To retain the current behavior and silence the warning, pass 'sort=True								
		verify_integrity=True)								
Out[84]:										

Ohio Nevada Missouri Alabama New York Oregon

NaN

8.0

10.0

12.0

14.0

NaN

7.0

NaN

9.0

NaN

11.0

16.0

8.0

NaN

10.0

NaN

12.0

17.0

NaN

7.0

9.0

11.0

13.0

NaN

2.0

NaN

4.0

NaN

6.0

NaN

a 1.0

b NaN

c 3.0

d NaN

e 5.0

f NaN