

C: Appending datasets

Appending vs merging or joining

Merging: Result is **wider**: adds **columns** using **keys to align rows**

Appending: Result is **longer**: Adds **rows aligning columns**

Merge via left dataframe's `.merge()` method:

left:

Index	Name	L1	L2
0	A	10	34
1	B	5	54
2	C	16	43

right:

Index	Name	R1
0	B	67
1	C	87
2	D	45

```
result = left.merge( right, how="outer" )
```

Aside: Without explicit keys, will use **all common columns**.
In general, better to be explicit via `on="Name"`.

result:

--	--	--	--	--

Index	Name	L1	L2	R1
0	A	10	34	
1	B	5	54	67
2	C	16	43	87
3	D			45

Append via `pd.concat()` function:

Example dataframes:

top:

Index	Name	C1	C2
0	A	10	34
1	B	5	54
2	C	16	43

bottom:

Index	Name	C1	C2
0	F	21	53
1	G	13	34

Concatenating a **list** of dataframes:

```
result = pd.concat( [top, bottom] )
```

result:

Index	Name	C1	C2
-------	------	----	----

0	A	10	34
1	B	5	54
2	C	16	43
0	F	21	53
1	G	13	34

Note 1: **first** argument specifies what to combine (a **list** of dataframes)

⚠ Cannot give items as separate arguments: `pd.concat(top, bottom)` ❌

Note 2: index values carried through without modification

If not desired, use argument: `ignore_index=True`

Concatenating a **dictionary** of dataframes:

- Can pass a dictionary as the first argument instead of a list
- Dictionary keys added as an index column

```
datasets = {
    'top': top,
    'bot': bottom,
}
result = pd.concat( datasets )
```

result:

Index	Index	Name	C1	C2
top	0	A	10	34
top	1	B	5	54
top	2	C	16	43
bot	0	F	21	53
bot	1	G	13	34

- Handy when collecting data from separate files

Appending mismatched dataframes:

Can also get **extra columns** if column names don't match:

top:

Index	Name	C1	C2
0	A	10	34
1	B	5	54
2	C	16	43

wide:

Index	Name	C1	C2	C3
0	F	21	53	67
1	G	13	34	89

```
result = pd.concat( [top, wide] )
```

result:

Index	Name	C1	C2	C3
0	A	10	34	
1	B	5	54	
2	C	16	43	
0	F	21	53	67
1	G	13	34	89

Defaults to an outer join on columns