Lecture 05. Data Wrangling: Combine and Merge

Instructor: Luping Yu

Mar 28, 2023

In many applications, data may be spread across a number of files or databases or be arranged in a form that is not easy to analyze. This lecture focuses on tools to help **combine** and **merge** data.

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.

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 [2]:
```
import pandas as pd
```

In [3]:
```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})

df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
                    'data2': range(3)})

display_side_by_side(df1, df2)
```

	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

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:

In [4]:
```
df_merged = pd.merge(df1, df2)

display_side_by_side(df1, df2, df_merged)
```

	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

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

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.

It's a good practice to specify explicitly, though:

In [5]:
```
df_merged = pd.merge(df1, df2, on='key')

display_side_by_side(df1, df2, df_merged)
```

	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

	key	data1	data2
0	b	0	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:

In [6]:
```
df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})

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

display_side_by_side(df3, df4)
```

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

	rkey	data2
0	a	0
1	b	1
2	d	2

In [7]:
```
df_merged = pd.merge(df3, df4, left_on='lkey', right_on='rkey')

display_side_by_side(df3, df4, df_merged)
```

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

	rkey	data2
0	a	0
1	b	1
2	d	2

	lkey	data1	rkey	data2
0	b	0	b	1
1	b	1	b	1
2	b	6	b	1
3	a	2	a	0
4	a	4	a	0
5	a	5	a	0

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 join` .

The `outer join` takes the **union** of the keys, combining the effect of applying both left and right joins:

In [8]:
```
df_merged_inner = pd.merge(df1, df2, how='inner')
df_merged_outer = pd.merge(df1, df2, how='outer')

display_side_by_side(df1, df2, df_merged_inner, df_merged_outer)
```

	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

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

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

In [9]:
```
df_merged_left = pd.merge(df1, df2, how='left')
df_merged_right = pd.merge(df1, df2, how='right')

display_side_by_side(df1, df2, df_merged_left, df_merged_right)
```

	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

	key	data1	data2
0	b	0	1.0
1	b	1	1.0
2	a	2	0.0
3	c	3	NaN
4	a	4	0.0
5	a	5	0.0
6	b	6	1.0

	key	data1	data2
0	a	2.0	0
1	a	4.0	0
2	a	5.0	0
3	b	0.0	1
4	b	1.0	1
5	b	6.0	1
6	d	NaN	2

See following table for a summary of the options for `how=` :

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
'outer'	Use all key combinations observed in both tables together

Many-to-many merges have well-defined behavior. Here's an example:

In [10]:
```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                    'data1': range(6)})

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

In [11]:
```
df_merged = pd.merge(df1, df2)

display_side_by_side(df1, df2, df_merged)
```

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

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

	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

Many-to-many joins form the **Cartesian product** of the rows. Since there were three 'b' rows in df1 and two in df2, there are six 'b' rows in the result.

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

In [12]:
```
df1 = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                    'key2': ['one', 'two', 'one'],
                    'lval': [1, 2, 3]})

df2 = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                    'key2': ['one', 'one', 'one', 'two'],
                    'rval': [4, 5, 6, 7]})
```

In [13]:
```
df_merged = pd.merge(df1, df2, on=['key1', 'key2'])

display_side_by_side(df1, df2, df_merged)
```

	key1	key2	lval
0	foo	one	1
1	foo	two	2
2	bar	one	3

	key1	key2	rval
0	foo	one	4
1	foo	one	5
2	bar	one	6
3	bar	two	7

	key1	key2	lval	rval
0	foo	one	1	4
1	foo	one	1	5
2	bar	one	3	6

See the following table for an argument reference on merge:

Argument	Description
how	One of'inner','outer','left', or'right'; defaults to'inner'.
on	Column names to join on. Must be found in both DataFrame objects. If not specified and no other join keys given, will use the intersection of the column names inleftandrightas the join keys.
left_on	Columns in left DataFrame to use as join keys.
right_on	Analogous to left_on for left DataFrame.
left_index	Use row index in left as its join key (or keys, if a MultiIndex).
right_index	Analogous to left_index.

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` to indicate that the index should be used as the merge key:

In [14]:
```
df1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'], 'value': range(6)})
df2 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
```

In [15]:
```
df_merged = pd.merge(df1, df2, left_on='key', right_index=True)

display_side_by_side(df1, df2, df_merged)
```

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

	group_val
a	3.5
b	7.0

	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

Since the default merge method is to intersect the join keys, you can instead form the union of them with an outer join:

In [16]:
```
df_merged = pd.merge(df1, df2, left_on='key', right_index=True, how='outer')

display_side_by_side(df1, df2, df_merged)
```

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

	group_val
a	3.5
b	7.0

	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	c	5	NaN

Concatenating Along an Axis

Another kind of data combination operation is referred to interchangeably as concatenation, binding, or stacking.

The `concat` function in pandas provides a consistent way to concat the datasets. Suppose we have three Series with no index overlap:

In [17]:
```
s1 = pd.Series([0, 1], index=['a', 'b'])
s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])
s3 = pd.Series([5, 6], index=['f', 'g'])
```

Calling concat with these objects in a list glues together the values and indexes:

In [18]:
```
pd.concat([s1, s2, s3])
```

Out[18]:
```
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
```

By default concat works along **axis=0** (index), producing another Series.

If you pass **axis=1** (columns), the result will instead be a DataFrame:

In [19]:
```
pd.concat([s1, s2, s3], axis=1)
```

Out[19]:

	0	1	2
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

The same logic extends to DataFrame objects:

In [20]:
```
df1 = pd.DataFrame([[0, 1],[2, 3],[4, 5]],
                   index=['a', 'b', 'c'],
                   columns=['one', 'two'])
df2 = pd.DataFrame([[5, 6],[7, 8]],
                   index=['a', 'c'],
                   columns=['two', 'three'])
```

In [21]:
```
df_concated = pd.concat([df1, df2])

display_side_by_side(df1, df2, df_concated)
```

	one	two
a	0	1
b	2	3
c	4	5

	two	three
a	5	6
c	7	8

	one	two	three
a	0.0	1	NaN
b	2.0	3	NaN
c	4.0	5	NaN
a	NaN	5	6.0
c	NaN	7	8.0

In [22]:
```
df_concated = pd.concat([df1, df2], axis=1)

display_side_by_side(df1, df2, df_concated)
```

	one	two
a	0	1
b	2	3
c	4	5

	two	three
a	5	6
c	7	8

	one	two	two	three
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0