What is the difference between join and merge in Pandas?

Asked 6 years, 4 months ago Active 5 months ago Viewed 135k times

212

64

Suppose I have two DataFrames like so:

```
left = pd.DataFrame({'key1': ['foo', 'bar'], 'lval': [1, 2]})

right = pd.DataFrame({'key2': ['foo', 'bar'], 'rval': [4, 5]})
```

I want to merge them, so I try something like this:

```
pd.merge(left, right, left_on='key1', right_on='key2')
```

And I'm happy

```
    key1    lval    key2    rval
0   foo     1       foo     4
1   bar     2       bar     5
```

But I'm trying to use the join method, which I've been lead to believe is pretty similar.

```
left.join(right, on=['key1', 'key2'])
```

And I get this:

```
//anaconda/lib/python2.7/site-packages/pandas/tools/merge.pyc in
_validate_specification(self)
    406             if self.right_index:
    407                 if not ((len(self.left_on) == self.right.index.nlevels)):
--> 408                     raise AssertionError()
    409                 self.right_on = [None] * n
    410         elif self.right_on is not None:

AssertionError:
```

What am I missing?

python pandas dataframe join

edited Dec 8 '18 at 2:53
cs95
223k 56 369 440

asked Mar 27 '14 at 0:42
munk
9,766 4 42 65

4 The specific problem here is that `merge` joins columns of `left` to columns of `right`, which is what you want, but `join(... on=[...])` joins columns of `left` to index keys of `right`, which is not what you

`other` 's indexes. Remember, indexes for join. While merge() is a more generic method. – Jiapeng Zhang Mar 4 '19 at 22:30

7 Answers

Active Oldest Votes

89

I always use `join` on indices:

```
import pandas as pd
left = pd.DataFrame({'key': ['foo', 'bar'], 'val': [1, 2]}).set_index('key')
right = pd.DataFrame({'key': ['foo', 'bar'], 'val': [4, 5]}).set_index('key')
left.join(right, lsuffix='_l', rsuffix='_r')
```

```
     val_l  val_r
key
foo      1      4
bar      2      5
```

The same functionality can be had by using `merge` on the columns follows:

```
left = pd.DataFrame({'key': ['foo', 'bar'], 'val': [1, 2]})
right = pd.DataFrame({'key': ['foo', 'bar'], 'val': [4, 5]})
left.merge(right, on=('key'), suffixes=('_l', '_r'))
```

```
   key  val_l  val_r
0  foo      1      4
1  bar      2      5
```

edited Jan 17 '17 at 21:54

answered Mar 27 '14 at 0:55

Paul H
44.9k 13 124 117

The error seems to be saying that it expects the multi index on `right` that is the same depth as the length on `on` . That makes sense to me sort of. I can accept that the semantics are different. But I'd like to know if I can get that same behavior with df.join – munk Mar 27 '14 at 0:59

352

`pandas.merge()` is the underlying function used for all merge/join behavior.

DataFrames provide the `pandas.DataFrame.merge()` and `pandas.DataFrame.join()` methods as a convenient way to access the capabilities of `pandas.merge()` . For example, `df1.merge(right=df2, ...)` is equivalent to `pandas.merge(left=df1, right=df2, ...)` .

These are the main differences between `df.join()` and `df.merge()` :

1. lookup on right table: `df1.join(df2)` always joins via the index of `df2` , but `df1.merge(df2)` can join to one or more columns of `df2` (default) or to the index of `df2` (with `right_index=True`).
2. lookup on left table: by default, `df1.join(df2)` uses the index of `df1` and `df1.merge(df2)`

3. left vs inner join: `df1.join(df2)` does a left join by default (keeps all rows of `df1`), but `df.merge` does an inner join by default (returns only matching rows of `df1` and `df2`).

So, the generic approach is to use `pandas.merge(df1, df2)` or `df1.merge(df2)`. But for a number of common situations (keeping all rows of `df1` and joining to an index in `df2`), you can save some typing by using `df1.join(df2)` instead.

Some notes on these issues from the documentation at http://pandas.pydata.org/pandas-docs/stable/merging.html#database-style-dataframe-joining-merging:

> `merge` is a function in the pandas namespace, and it is also available as a DataFrame instance method, with the calling DataFrame being implicitly considered the left object in the join.
>
> The related `DataFrame.join` method, uses `merge` internally for the index-on-index and index-on-column(s) joins, but joins on indexes by default rather than trying to join on common columns (the default behavior for `merge`). If you are joining on index, you may wish to use `DataFrame.join` to save yourself some typing.

...

> These two function calls are completely equivalent:

```
left.join(right, on=key_or_keys)
pd.merge(left, right, left_on=key_or_keys, right_index=True, how='left', sort=False)
```

edited Apr 11 '18 at 19:49

answered Jun 17 '16 at 22:51

Matthias Fripp
11.4k 3 22 33

20 This should definitely be the accepted answer ! Thanks for the thourough explanation – Yohan Obadia Dec 26 '18 at 17:20

@Matthias Fripp, Perhaps for the more experienced it goes without saying, but it could also be said that "lookup on right table: df1.join(df2) can be overridden to df1.join(df2, on=key_or_keys? – spacedustpi Feb 19 '19 at 0:00

@spacedustpi, I think you are saying that you can use `on=key_or_keys` to change the way rows are found in the right table. However, that is not actually the case. The `on` argument changes the lookup on the *left* table (`df1`) from index to column(s). However, even with this argument, the right table (`df2`) will be matched via its index. (See the last example above.) – Matthias Fripp Feb 19 '19 at 5:28

1 Pandas has several methods to deal with these situations, among them merge, join, append, concat, combine, combine_first. Take a look at each of these to have a glimpse about which one would be the best fit for your situation – xiaxio Mar 29 at 19:03

I believe that `join()` is just a convenience method. Try `df1.merge(df2)` instead, which allows you

```
In [30]: left.merge(right, left_on="key1", right_on="key2")
Out[30]:
  key1  lval key2  rval
0  foo     1  foo     4
1  bar     2  bar     5
```

answered Mar 27 '14 at 1:03

Noah
15k 7 54 66

12

From this documentation

> pandas provides a single function, merge, as the entry point for all standard database join operations between DataFrame objects:
>
> ```
> merge(left, right, how='inner', on=None, left_on=None, right_on=None,
>  left_index=False, right_index=False, sort=True,
>  suffixes=('_x', '_y'), copy=True, indicator=False)
> ```

And :

> `DataFrame.join` is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame. Here is a very basic example: The data alignment here is on the indexes (row labels). This same behavior can be achieved using merge plus additional arguments instructing it to use the indexes:
>
> ```
> result = pd.merge(left, right, left_index=True, right_index=True,
> how='outer')
> ```

edited Feb 17 at 16:00

mmrmartin
1,328 11 21

answered Jun 12 '16 at 10:34

Romain Jouin
2,922 31 57

9

One of the difference is that `merge` is creating a new index, and `join` is keeping the left side index. It can have a big consequence on your later transformations if you wrongly assume that your index isn't changed with `merge` .

For example:

```
import pandas as pd

df1 = pd.DataFrame({'org_index': [101, 102, 103, 104],
                    'date': [201801, 201801, 201802, 201802],
                    'val': [1, 2, 3, 4]}, index=[101, 102, 103, 104])
```

```
101  201801        101    1
102  201801        102    2
103  201802        103    3
104  201802        104    4
```

-

```
df2 = pd.DataFrame({'date': [201801, 201802], 'dateval': ['A', 'B']}).set_index('date')
df2
```

```
       dateval
date
201801       A
201802       B
```

-

```
df1.merge(df2, on='date')
```

```
     date  org_index  val dateval
0  201801        101    1       A
1  201801        102    2       A
2  201802        103    3       B
3  201802        104    4       B
```

-

```
df1.join(df2, on='date')
       date  org_index  val dateval
101  201801        101    1       A
102  201801        102    2       A
103  201802        103    3       B
104  201802        104    4       B
```

answered Dec 19 '18 at 10:31

steco
678 6 14

That is correct. If we merge the two data frames on columns other than indices we will get a new index but if we merge on the indices of both data frames we will get the a data frame with the same index. So, in order to get the same index after merge we can make the columns our index (on which we want to merge) for both data frames and then merge the data frames on the newly created index. – hasan najeeb Nov 28 '19 at 12:17

Very insightful. I've never needed the indexing (I normally just reset the index) but this could make a big difference in some cases. – irene Jan 30 at 12:15

4

- Join: Default Index (If any same column name then it will throw an error in default mode because u have not defined lsuffix or rsuffix))

- Merge: Default Same Column Names (If no same column name it will throw an error in default mode)

```
df_1.merge(df_2)
```

- `on` parameter has different meaning in both cases

```
df_1.merge(df_2, on='column_1')

df_1.join(df_2, on='column_1') // It will throw error
df_1.join(df_2.set_index('column_1'), on='column_1')
```

answered Mar 2 '19 at 5:53

Harsh
41 2

2

To put it analogously to SQL "Pandas merge is to outer/inner join and Pandas join is to natural join". Hence when you use merge in pandas, you want to specify which kind of sqlish join you want to use whereas when you use pandas join, you really want to have a matching column label to ensure it joins

answered Apr 15 '19 at 6:29

Kaustubh J
370 2 7
