☰   ⦚┇⦚ pandas   🔍   ⫷

pandas.DataFrame.merge

DataFrame.**merge**(*right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=None, indicator=False, validate=None*)   [source]

Merge DataFrame or named Series objects with a database-style join.

A named Series object is treated as a DataFrame with a single named column.

The join is done on columns or indexes. If joining columns on columns, the DataFrame indexes *will be ignored*. Otherwise if joining indexes on indexes or indexes on a column or columns, the index will be passed on. When performing a cross merge, no column specifications to merge on are allowed.

> ⚠ **Warning**
>
> If both key columns contain rows where the key is a null value, those rows will be matched against each other. This is different from usual SQL join behaviour and can lead to unexpected results.

Parameters:

> **right** : *DataFrame or named Series*
>
>> Object to merge with.
>
> **how** : *{'left', 'right', 'outer', 'inner', 'cross'}, default 'inner'*
>
>> Type of merge to be performed.
>> - left: use only keys from left frame, similar to a SQL left outer join; preserve key order.
>> - right: use only keys from right frame, similar to a SQL right outer join; preserve key order.
>> - outer: use union of keys from both frames, similar to a SQL full outer join; sort keys

Skip to main content

- inner: use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys.

- cross: creates the cartesian product from both frames, preserves the order of the left keys.

on : *label or list*

Column or index level names to join on. These must be found in both DataFrames. If *on* is None and not merging on indexes then this defaults to the intersection of the columns in both DataFrames.

left_on : *label or list, or array-like*

Column or index level names to join on in the left DataFrame. Can also be an array or list of arrays of the length of the left DataFrame. These arrays are treated as if they are columns.

right_on : *label or list, or array-like*

Column or index level names to join on in the right DataFrame. Can also be an array or list of arrays of the length of the right DataFrame. These arrays are treated as if they are columns.

left_index : *bool, default False*

Use the index from the left DataFrame as the join key(s). If it is a MultiIndex, the number of keys in the other DataFrame (either the index or a number of columns) must match the number of levels.

right_index : *bool, default False*

Use the index from the right DataFrame as the join key. Same caveats as left_index.

sort : *bool, default False*

Sort the join keys lexicographically in the result DataFrame. If False, the order of the join keys depends on the join type (how keyword).

suffixes : *list-like, default is ("_x", "_y")*

A length-2 sequence where each element is optionally a string indicating the suffix to add to overlapping column names in *left* and *right* respectively. Pass a value of *None* instead of a string to indicate that the column name from *left* or *right* should be left as-is, with no suffix. At least one of the values must not be None.

copy : *bool, default True*

If False, avoid copy if possible.

Skip to main content

> **ⓘ Note**
>
> The *copy* keyword will change behavior in pandas 3.0. [Copy-on-Write](#) will be
> enabled by default, which means that all methods with a *copy* keyword will
> use a lazy copy mechanism to defer the copy and ignore the *copy* keyword.
> The *copy* keyword will be removed in a future version of pandas.
> You can already get the future behavior and improvements through
> enabling copy on write `pd.options.mode.copy_on_write = True`

indicator : *bool or str, default False*

If True, adds a column to the output DataFrame called "_merge" with information on
the source of each row. The column can be given a different name by providing a
string argument. The column will have a Categorical type with the value of "left_only"
for observations whose merge key only appears in the left DataFrame, "right_only" for
observations whose merge key only appears in the right DataFrame, and "both" if the
observation's merge key is found in both DataFrames.

validate : *str, optional*

If specified, checks if merge is of specified type.
- "one_to_one" or "1:1": check if merge keys are unique in both left and right
  datasets.
- "one_to_many" or "1:m": check if merge keys are unique in left dataset.
- "many_to_one" or "m:1": check if merge keys are unique in right dataset.
- "many_to_many" or "m:m": allowed, but does not result in checks.

Returns:

DataFrame

A DataFrame of the two merged objects.

> **↪ See also**
>
> `merge_ordered`
>   Merge with optional filling/interpolation.
> `merge_asof`
>   Merge on nearest keys.
> `DataFrame.join`

Examples

```
>>> df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
...                     'value': [1, 2, 3, 5]})
>>> df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
...                     'value': [5, 6, 7, 8]})
>>> df1
    lkey value
0   foo      1
1   bar      2
2   baz      3
3   foo      5
>>> df2
    rkey value
0   foo      5
1   bar      6
2   baz      7
3   foo      8
```

Merge df1 and df2 on the lkey and rkey columns. The value columns have the default suffixes, _x and _y, appended.

```
>>> df1.merge(df2, left_on='lkey', right_on='rkey')
   lkey  value_x rkey  value_y
0   foo        1  foo        5
1   foo        1  foo        8
2   bar        2  bar        6
3   baz        3  baz        7
4   foo        5  foo        5
5   foo        5  foo        8
```

Merge DataFrames df1 and df2 with specified left and right suffixes appended to any overlapping columns.

```
>>> df1.merge(df2, left_on='lkey', right_on='rkey',
...           suffixes=('_left', '_right'))
   lkey  value_left rkey  value_right
0   foo           1  foo            5
1   foo           1  foo            8
2   bar           2  bar            6
3   baz           3  baz            7
4   foo           5  foo            5
5   foo           5  foo            8
```

Merge DataFrames df1 and df2, but raise an exception if the DataFrames have any overlapping

Skip to main content

```
>>> df1.merge(df2, left_on='lkey', right_on='rkey', suffixes=(False, False))
Traceback (most recent call last):
...
ValueError: columns overlap but no suffix specified:
    Index(['value'], dtype='object')
```

```
>>> df1 = pd.DataFrame({'a': ['foo', 'bar'], 'b': [1, 2]})
>>> df2 = pd.DataFrame({'a': ['foo', 'baz'], 'c': [3, 4]})
>>> df1
     a  b
0  foo  1
1  bar  2
>>> df2
     a  c
0  foo  3
1  baz  4
```

```
>>> df1.merge(df2, how='inner', on='a')
     a  b  c
0  foo  1  3
```

```
>>> df1.merge(df2, how='left', on='a')
     a  b    c
0  foo  1  3.0
1  bar  2  NaN
```

```
>>> df1 = pd.DataFrame({'left': ['foo', 'bar']})
>>> df2 = pd.DataFrame({'right': [7, 8]})
>>> df1
   left
0   foo
1   bar
>>> df2
   right
0      7
1      8
```

```
>>> df1.merge(df2, how='cross')
   left  right
0   foo      7
1   foo      8
2   bar      7
3   bar      8
```

Skip to main content

Built with the PyData Sphinx Theme 0.14.4.