

# Pandas – merge()

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

**pandas.merge(left, right, how='inner', on=None, left\_on=None, right\_on=None, left\_index=False, right\_index=False, sort=False, suffixes=('\_x', '\_y'), copy=True, indicator=False, validate=None)**

- **left:** A DataFrame or named Series object.
- **right:** Another DataFrame or named Series object.
- **on:** Column or index level names to join on. Must be found in both the left and right DataFrame and/or Series objects. If not passed and left\_index and right\_index are False, the intersection of the columns in the DataFrames and/or Series will be inferred to be the join keys.
- **left\_on:** Columns or index levels from the left DataFrame or Series to use as keys. Can either be column names, index level names, or arrays with length equal to the length of the DataFrame or Series.
- **right\_on:** Columns or index levels from the right DataFrame or Series to use as keys. Can either be column names, index level names, or arrays with length equal to the length of the DataFrame or Series.
- **left\_index:** If True, use the index (row labels) from the left DataFrame or Series as its join key(s). In the case of a DataFrame or Series with a MultiIndex (hierarchical), the number of levels must match the number of join keys from the right DataFrame or Series.
- **right\_index:** Same usage as left\_index for the right DataFrame or Series
- **how:** One of 'left', 'right', 'outer', 'inner', 'cross'. Defaults to inner. See below for more detailed description of each method.

# Pandas – merge()

**pandas.merge(left, right, how='inner', on=None, left\_on=None, right\_on=None, left\_index=False, right\_index=False, sort=False, suffixes=('\_x', '\_y'), copy=True, indicator=False, validate=None)**

**sort:** Sort the result DataFrame by the join keys in lexicographical order. Defaults to True, setting to False will improve performance substantially in many cases.

**suffixes:** A tuple of string suffixes to apply to overlapping columns. Defaults to ('\_x', '\_y').

**copy:** Always copy data (default True) from the passed DataFrame or named Series objects, even when reindexing is not necessary. Cannot be avoided in many cases but may improve performance / memory usage. The cases where copying can be avoided are somewhat pathological but this option is provided nonetheless.

**indicator:** Add a column to the output DataFrame called `_merge` with information on the source of each row. `_merge` is Categorical-type and takes on a value of `left_only` for observations whose merge key only appears in 'left' DataFrame or Series, `right_only` for observations whose merge key only appears in 'right' DataFrame or Series, and `both` if the observation's merge key is found in both.

**validate :** string, default None. If specified, checks if merge is of specified type.

“one\_to\_one” or “1:1”: checks if merge keys are unique in both left and right datasets.

“one\_to\_many” or “1:m”: checks if merge keys are unique in left dataset.

“many\_to\_one” or “m:1”: checks if merge keys are unique in right dataset.

“many\_to\_many” or “m:m”: allowed, but does not result in checks.

# Pandas – merge()

how options and SQL equivalent names

Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames
cross	CROSS JOIN	Create the cartesian product of rows of both frames

## Pandas – merge()

```
result = pd.merge(left, right, how="left", on=["key1", "key2"])
```

left					right					Result						
	key1	key2	A	B		key1	key2	C	D		key1	key2	A	B	C	D
0	K0	K0	A0	B0	0	K0	K0	C0	D0	0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	1	K1	K0	C1	D1	1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	2	K1	K0	C2	D2	2	K1	K0	A2	B2	C1	D1
3	K2	K1	A3	B3	3	K2	K0	C3	D3	3	K1	K0	A2	B2	C2	D2
										4	K2	K1	A3	B3	NaN	NaN

```
result = pd.merge(left, right, how="right", on=["key1", "key2"])
```

left					right					Result						
	key1	key2	A	B		key1	key2	C	D		key1	key2	A	B	C	D
0	K0	K0	A0	B0	0	K0	K0	C0	D0	0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	1	K1	K0	C1	D1	1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	2	K1	K0	C2	D2	2	K1	K0	A2	B2	C2	D2
3	K2	K1	A3	B3	3	K2	K0	C3	D3	3	K2	K0	NaN	NaN	C3	D3

# Pandas – merge() – example: left / right

## Merge Dataframes

```
In [9]: df_merge_left=pd.merge(df1,df3, how='left', on='id')
df_merge_left
```

Out[9]:

	name_x	date	height	weight	age	name_y	hours	status	children	sex
id										
1373913	Marisa Martins	2013-02-05	155	48	45	Marisa Martins	3.0	married	2.0	F
1109818	Rita Fonseca	2018-08-28	166	54	45	NaN	NaN	NaN	NaN	NaN

```
In [10]: df_merge_right=pd.merge(df1,df3, how='right', on='id')
df_merge_right
```

Out[10]:

	name_x	date	height	weight	age	name_y	hours	status	children	sex
id										
1767703	NaN	NaT	NaN	NaN	NaN	Manuel Martins	6	single	0	M
1373913	Marisa Martins	2013-02-05	155.0	48.0	45.0	Marisa Martins	3	married	2	F
1158813	NaN	NaT	NaN	NaN	NaN	Joana Freitas	3	widow	1	F

## Pandas – merge()

```
result = pd.merge(left, right, how="outer", on=["key1", "key2"])
```

left					right					Result						
	key1	key2	A	B		key1	key2	C	D		key1	key2	A	B	C	D
0	K0	K0	A0	B0	0	K0	K0	C0	D0	0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	1	K1	K0	C1	D1	1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	2	K1	K0	C2	D2	2	K1	K0	A2	B2	C1	D1
3	K2	K1	A3	B3	3	K2	K0	C3	D3	3	K1	K0	A2	B2	C2	D2
										4	K2	K1	A3	B3	NaN	NaN
										5	K2	K0	NaN	NaN	C3	D3

```
result = pd.merge(left, right, how="inner", on=["key1", "key2"])
```

left					right					Result						
	key1	key2	A	B		key1	key2	C	D		key1	key2	A	B	C	D
0	K0	K0	A0	B0	0	K0	K0	C0	D0	0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	1	K1	K0	C1	D1	1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	2	K1	K0	C2	D2	2	K1	K0	A2	B2	C2	D2
3	K2	K1	A3	B3	3	K2	K0	C3	D3							

## Pandas – merge() – example: inner / outer

```
In [11]: df_merge_inner=pd.merge(df1,df3, how='inner',on='id')
df_merge_inner
```

```
Out[11]:
```

	name_x	date	height	weight	age	name_y	hours	status	children	sex
id										
1373913	Marisa Martins	2013-02-05	155	48	45	Marisa Martins	3	married	2	F

```
In [12]: df_merge_outer=pd.merge(df1,df3, how='outer', on='id')
df_merge_outer
```

```
Out[12]:
```

	name_x	date	height	weight	age	name_y	hours	status	children	sex
id										
1373913	Marisa Martins	2013-02-05	155.0	48.0	45.0	Marisa Martins	3.0	married	2.0	F
1109818	Rita Fonseca	2018-08-28	166.0	54.0	45.0	NaN	NaN	NaN	NaN	NaN
1767703	NaN	NaT	NaN	NaN	NaN	Manuel Martins	6.0	single	0.0	M
1158813	NaN	NaT	NaN	NaN	NaN	Joana Freitas	3.0	widow	1.0	F

# Pandas – merge()

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

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	key1_x	key2_x	A	B	key1_y	key2_y	C	D
0	K0	K0	A0	B0	K0	K0	C0	D0
1	K0	K0	A0	B0	K1	K0	C1	D1
2	K0	K0	A0	B0	K1	K0	C2	D2
3	K0	K0	A0	B0	K2	K0	C3	D3
4	K0	K1	A1	B1	K0	K0	C0	D0
5	K0	K1	A1	B1	K1	K0	C1	D1
6	K0	K1	A1	B1	K1	K0	C2	D2
7	K0	K1	A1	B1	K2	K0	C3	D3
8	K1	K0	A2	B2	K0	K0	C0	D0
9	K1	K0	A2	B2	K1	K0	C1	D1
10	K1	K0	A2	B2	K1	K0	C2	D2
11	K1	K0	A2	B2	K2	K0	C3	D3
12	K2	K1	A3	B3	K0	K0	C0	D0
13	K2	K1	A3	B3	K1	K0	C1	D1
14	K2	K1	A3	B3	K1	K0	C2	D2
15	K2	K1	A3	B3	K2	K0	C3	D3



## Pandas – merge() – example cross

```
In [13]: df_merge_cross = pd.merge(df1, df3, how='cross')
df_merge_cross
```

Out[13]:

	name_x	date	height	weight	age	name_y	hours	status	children	sex
0	Marisa Martins	2013-02-05	155	48	45	Manuel Martins	6	single	0	M
1	Marisa Martins	2013-02-05	155	48	45	Marisa Martins	3	married	2	F
2	Marisa Martins	2013-02-05	155	48	45	Joana Freitas	3	widow	1	F
3	Rita Fonseca	2018-08-28	166	54	45	Manuel Martins	6	single	0	M
4	Rita Fonseca	2018-08-28	166	54	45	Marisa Martins	3	married	2	F
5	Rita Fonseca	2018-08-28	166	54	45	Joana Freitas	3	widow	1	F

# Pandas – join()

Join columns with **other** DataFrame either on index or on a key column

**DataFrame.join(other, on=None, how='left', lsuffix="", rsuffix="", sort=False)**

**other** : DataFrame, Series, or list of DataFrame

Index should be similar to one of the columns in this one. If a Series is passed, its name attribute must be set, and that will be used as the column name in the resulting joined DataFrame.

**on** : str, list of str, or array-like, optional

Column or index level name(s) in the caller to join on the index in other, otherwise joins index-on-index. If multiple values given, the other DataFrame must have a MultiIndex. Can pass an array as the join key if it is not already contained in the calling DataFrame. Like an Excel VLOOKUP operation.

**how** : {'left', 'right', 'outer', 'inner'}, default 'left'

How to handle the operation of the two objects.

- left: use calling frame's index (or column if on is specified)
- right: use other's index.
- outer: form union of calling frame's index (or column if on is specified) with other's index, and sort it. lexicographically.
- inner: form intersection of calling frame's index (or column if on is specified) with other's index, preserving the order of the calling's one.

**lsuffix** : str, default ''

Suffix to use from left frame's overlapping columns.

**rsuffix** : str, default ''

Suffix to use from right frame's overlapping columns.

**sort** : bool, default False

Order result DataFrame lexicographically by the join key. If False, the order of the join key depends on the join type (how keyword).

## Pandas – join()

```
result = left.join(right)
```

left			right			Result				
	A	B		C	D		A	B	C	D
K0	A0	B0	K0	C0	D0	K0	A0	B0	C0	D0
K1	A1	B1	K2	C2	D2	K1	A1	B1	NaN	NaN
K2	A2	B2	K3	C3	D3	K2	A2	B2	C2	D2

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

left			right			Result				
	A	B		C	D		A	B	C	D
K0	A0	B0	K0	C0	D0	K0	A0	B0	C0	D0
K1	A1	B1	K2	C2	D2	K1	A1	B1	NaN	NaN
K2	A2	B2	K3	C3	D3	K2	A2	B2	C2	D2
						K3	NaN	NaN	C3	D3

```
result = left.join(right, how="inner")
```

left			right			Result				
	A	B		C	D		A	B	C	D
K0	A0	B0	K0	C0	D0	K0	A0	B0	C0	D0
K1	A1	B1	K2	C2	D2					
K2	A2	B2	K3	C3	D3	K2	A2	B2	C2	D2

# Pandas – join() – example: left / right

## Join DataFrames

```
In [14]: df_join_left = df1.join(df3, lsuffix='df1')
df_join_left
```

Out[14]:

	namedf1	date	height	weight	age	name	hours	status	children	sex
id										
1373913	Marisa Martins	2013-02-05	155	48	45	Marisa Martins	3.0	married	2.0	F
1109818	Rita Fonseca	2018-08-28	166	54	45	NaN	NaN	NaN	NaN	NaN

```
In [15]: df_join_right = df1.join(df3, how = 'right', lsuffix='df1')
df_join_right
```

Out[15]:

	namedf1	date	height	weight	age	name	hours	status	children	sex
id										
1767703	NaN	NaT	NaN	NaN	NaN	Manuel Martins	6	single	0	M
1373913	Marisa Martins	2013-02-05	155.0	48.0	45.0	Marisa Martins	3	married	2	F
1158813	NaN	NaT	NaN	NaN	NaN	Joana Freitas	3	widow	1	F

## Pandas – join() – example: inner / outer

```
In [16]: df_join_inner = df1.join(df3, how = 'inner', lsuffix='df1')
df_join_inner
```

```
Out[16]:
```

	namedf1	date	height	weight	age	name	hours	status	children	sex
id										
1373913	Marisa Martins	2013-02-05	155	48	45	Marisa Martins	3	married	2	F

```
In [17]: df_join_outer = df1.join(df3, how = 'outer', lsuffix='df1')
df_join_outer
```

```
Out[17]:
```

	namedf1	date	height	weight	age	name	hours	status	children	sex
id										
1109818	Rita Fonseca	2018-08-28	166.0	54.0	45.0	NaN	NaN	NaN	NaN	NaN
1158813	NaN	NaT	NaN	NaN	NaN	Joana Freitas	3.0	widow	1.0	F
1373913	Marisa Martins	2013-02-05	155.0	48.0	45.0	Marisa Martins	3.0	married	2.0	F
1767703	NaN	NaT	NaN	NaN	NaN	Manuel Martins	6.0	single	0.0	M