**Enough talk - just show me how to use merge!**

Setup & Basics

np.random.seed(0)

left = pd.DataFrame({'key': ['A', 'B', 'C', 'D'], 'value': np.random.randn(4)})

right = pd.DataFrame({'key': ['B', 'D', 'E', 'F'], 'value': np.random.randn(4)})

left

key value

0 A 1.764052

1 B 0.400157

2 C 0.978738

3 D 2.240893

right

key value

0 B 1.867558

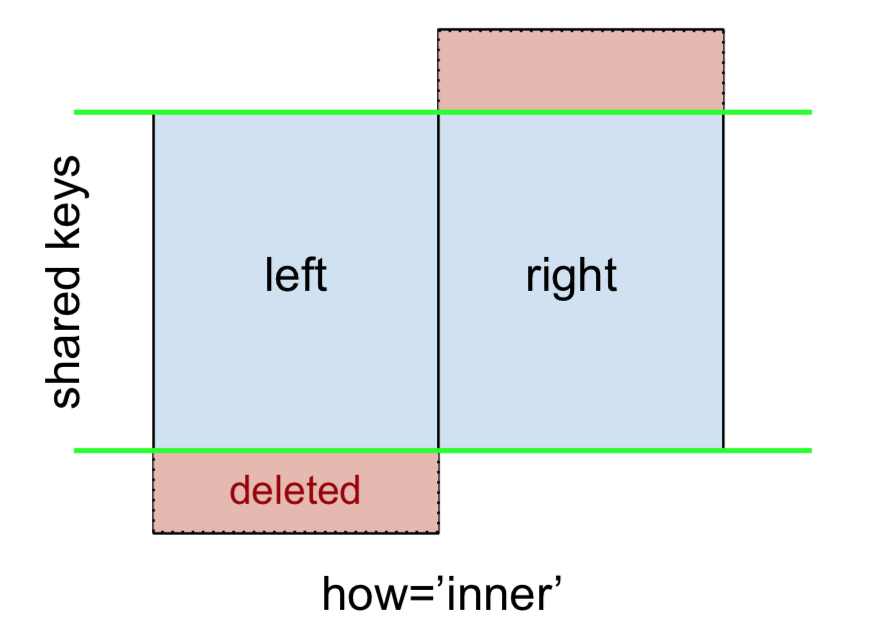
1 D -0.977278

2 E 0.950088

3 F -0.151357

For the sake of simplicity, the key column has the same name (for now).

An **INNER JOIN** is represented by



**Note** This, along with the forthcoming figures all follow this convention:

* **blue** indicates rows that are present in the merge result
* **red** indicates rows that are excluded from the result (i.e., removed)
* **green** indicates missing values that are replaced with NaNs in the result

To perform an INNER JOIN, call [merge](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html) on the left DataFrame, specifying the right DataFrame and the join key (at the very least) as arguments.

left.merge(right, on='key')

# Or, if you want to be explicit

# left.merge(right, on='key', how='inner')

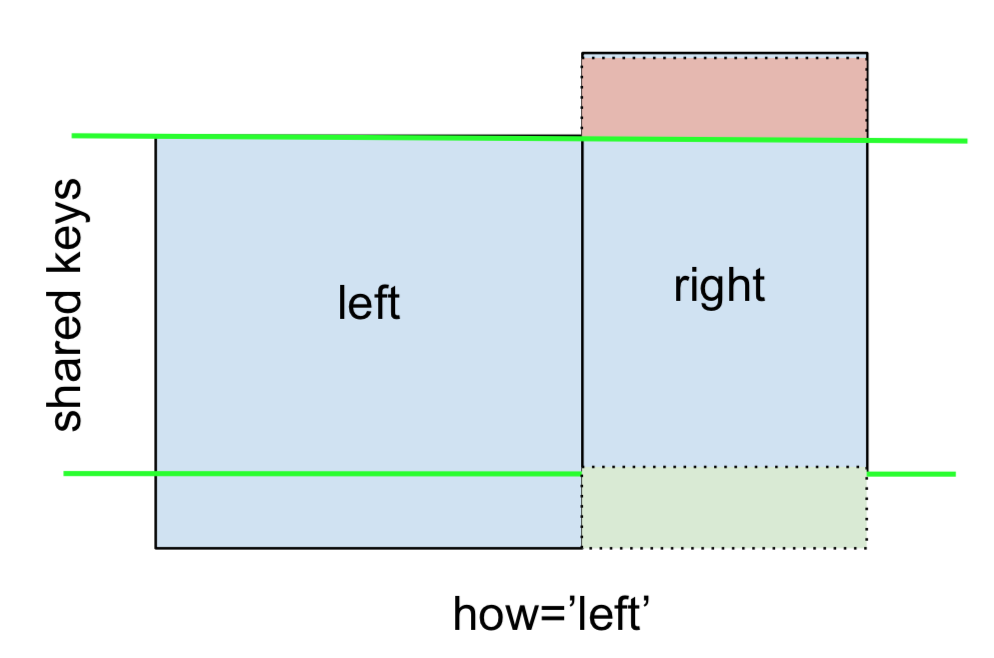
key value\_x value\_y

0 B 0.400157 1.867558

1 D 2.240893 -0.977278

This returns only rows from left and right which share a common key (in this example, "B" and "D).

A **LEFT OUTER JOIN**, or LEFT JOIN is represented by



This can be performed by specifying how='left'.

left.merge(right, on='key', how='left')

key value\_x value\_y

0 A 1.764052 NaN

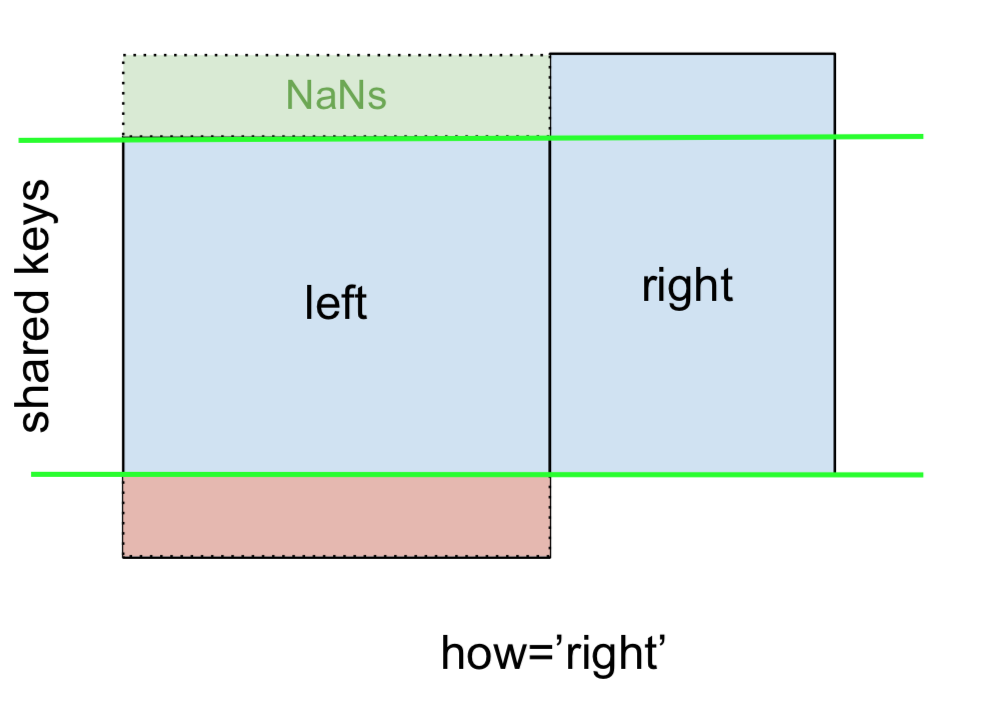
1 B 0.400157 1.867558

2 C 0.978738 NaN

3 D 2.240893 -0.977278

Carefully note the placement of NaNs here. If you specify how='left', then only keys from left are used, and missing data from right is replaced by NaN.

And similarly, for a **RIGHT OUTER JOIN**, or RIGHT JOIN which is...



...specify how='right':

left.merge(right, on='key', how='right')

key value\_x value\_y

0 B 0.400157 1.867558

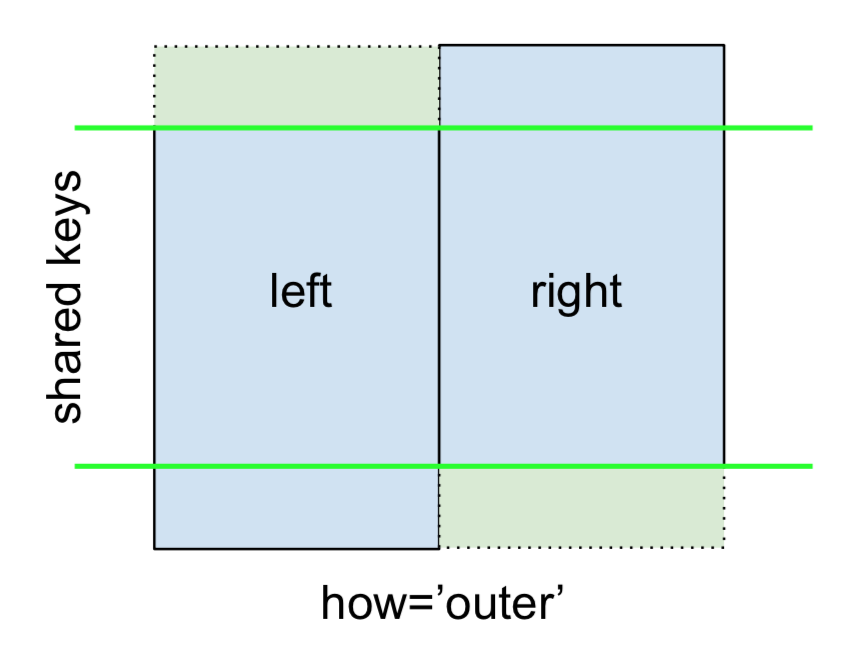
1 D 2.240893 -0.977278

2 E NaN 0.950088

3 F NaN -0.151357

Here, keys from right are used, and missing data from left is replaced by NaN.

Finally, for the **FULL OUTER JOIN**, given by



specify how='outer'.

left.merge(right, on='key', how='outer')

key value\_x value\_y

0 A 1.764052 NaN

1 B 0.400157 1.867558

2 C 0.978738 NaN

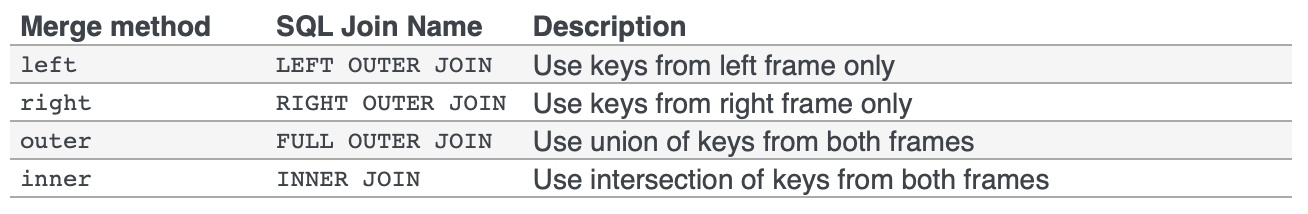
3 D 2.240893 -0.977278

4 E NaN 0.950088

5 F NaN -0.151357

This uses the keys from both frames, and NaNs are inserted for missing rows in both.

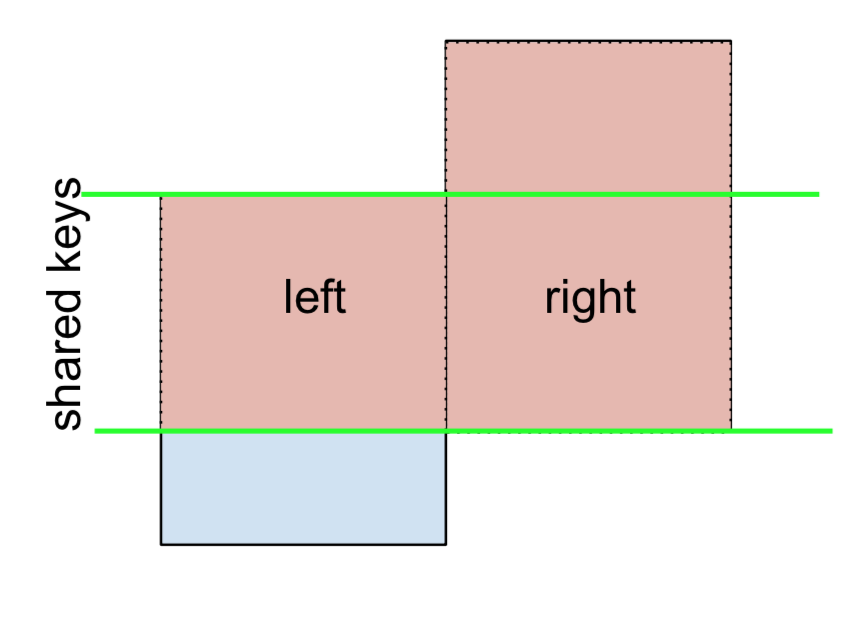
The documentation summarizes these various merges nicely:

[](https://i.stack.imgur.com/5qDIy.png)

**Other JOINs - LEFT-Excluding, RIGHT-Excluding, and FULL-Excluding/ANTI JOINs**

If you need **LEFT-Excluding JOINs** and **RIGHT-Excluding JOINs** in two steps.

For LEFT-Excluding JOIN, represented as



Start by performing a LEFT OUTER JOIN and then filtering to rows coming from left only (excluding everything from the right),

(left.merge(right, on='key', how='left', indicator=True)

.query('\_merge == "left\_only"')

.drop('\_merge', 1))

key value\_x value\_y

0 A 1.764052 NaN

2 C 0.978738 NaN

Where,

left.merge(right, on='key', how='left', **indicator=True**)

key value\_x value\_y \_merge

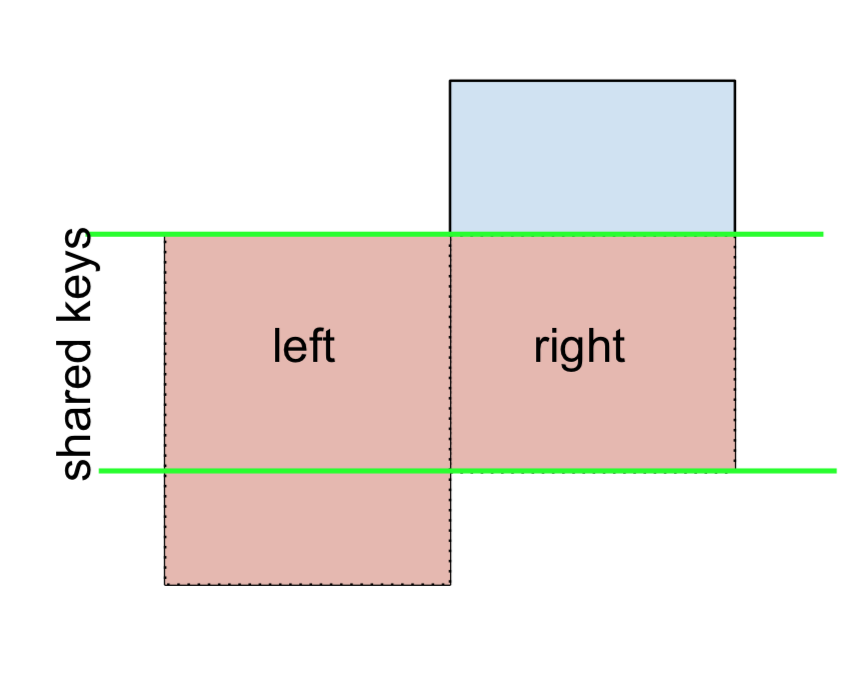
0 A 1.764052 NaN left\_only

1 B 0.400157 1.867558 both

2 C 0.978738 NaN left\_only

3 D 2.240893 -0.977278 both

And similarly, for a RIGHT-Excluding JOIN,



(left.merge(right, on='key', how='right', **indicator=True**)

.query('\_merge == "right\_only"')

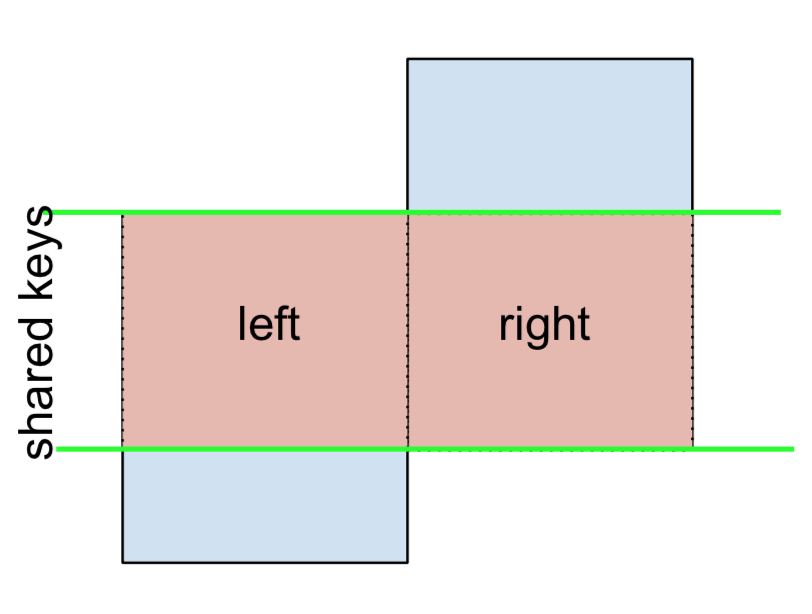
.drop('\_merge', 1))

key value\_x value\_y

2 E NaN 0.950088

3 F NaN -0.151357

Lastly, if you are required to do a merge that only retains keys from the left or right, but not both (IOW, performing an **ANTI-JOIN**),



You can do this in similar fashion—

(left.merge(right, on='key', how='outer', indicator=True)

.query('\_merge != "both"')

.drop('\_merge', 1))

key value\_x value\_y

0 A 1.764052 NaN

2 C 0.978738 NaN

4 E NaN 0.950088

5 F NaN -0.151357

**Different names for key columns**

If the key columns are named differently—for example, left has keyLeft, and right has keyRight instead of key—then you will have to specify left\_on and right\_on as arguments instead of on:

left2 = left.rename({'key':'keyLeft'}, axis=1)

right2 = right.rename({'key':'keyRight'}, axis=1)

left2

keyLeft value

0 A 1.764052

1 B 0.400157

2 C 0.978738

3 D 2.240893

right2

keyRight value

0 B 1.867558

1 D -0.977278

2 E 0.950088

3 F -0.151357

left2.merge(right2, left\_on='keyLeft', right\_on='keyRight', how='inner')

keyLeft value\_x keyRight value\_y

0 B 0.400157 B 1.867558

1 D 2.240893 D -0.977278

**Avoiding duplicate key column in output**

When merging on keyLeft from left and keyRight from right, if you only want either of the keyLeft or keyRight (but not both) in the output, you can start by setting the index as a preliminary step.

left3 = left2.set\_index('keyLeft')

left3.merge(right2, left\_index=True, right\_on='keyRight')

value\_x keyRight value\_y

0 0.400157 B 1.867558

1 2.240893 D -0.977278

Contrast this with the output of the command just before (that is, the output of left2.merge(right2, left\_on='keyLeft', right\_on='keyRight', how='inner')), you'll notice keyLeft is missing. You can figure out what column to keep based on which frame's index is set as the key. This may matter when, say, performing some OUTER JOIN operation.

**Merging only a single column from one of the DataFrames**

For example, consider

**Enough talk - just show me how to use merge!**

Setup & Basics

np.random.seed(0)

left = pd.DataFrame({'key': ['A', 'B', 'C', 'D'], 'value': np.random.randn(4)})

right = pd.DataFrame({'key': ['B', 'D', 'E', 'F'], 'value': np.random.randn(4)})

left

key value

0 A 1.764052

1 B 0.400157

2 C 0.978738

3 D 2.240893

right

key value

0 B 1.867558

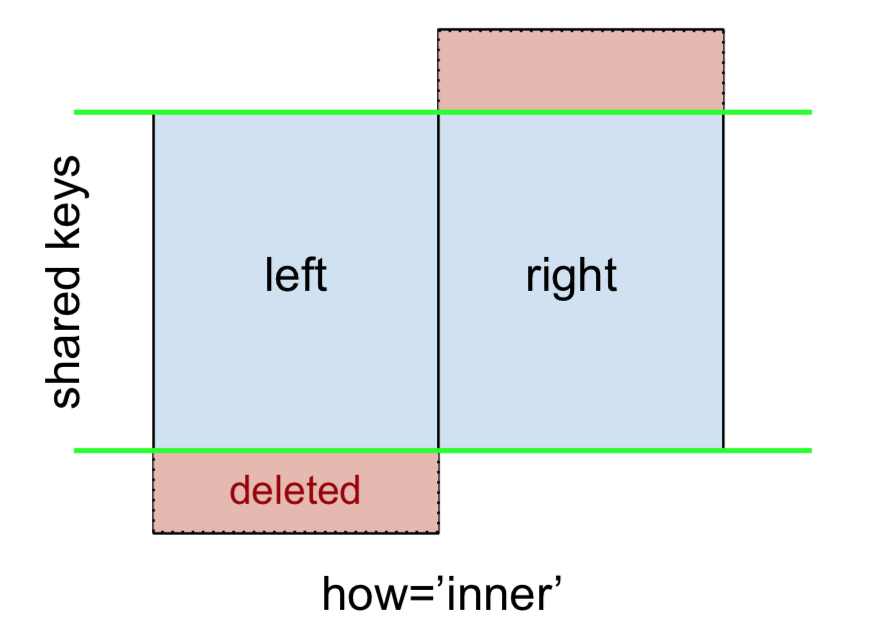
1 D -0.977278

2 E 0.950088

3 F -0.151357

For the sake of simplicity, the key column has the same name (for now).

An **INNER JOIN** is represented by



**Note** This, along with the forthcoming figures all follow this convention:

* **blue** indicates rows that are present in the merge result
* **red** indicates rows that are excluded from the result (i.e., removed)
* **green** indicates missing values that are replaced with NaNs in the result

To perform an INNER JOIN, call [merge](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html) on the left DataFrame, specifying the right DataFrame and the join key (at the very least) as arguments.

left.merge(right, on='key')

# Or, if you want to be explicit

# left.merge(right, on='key', how='inner')

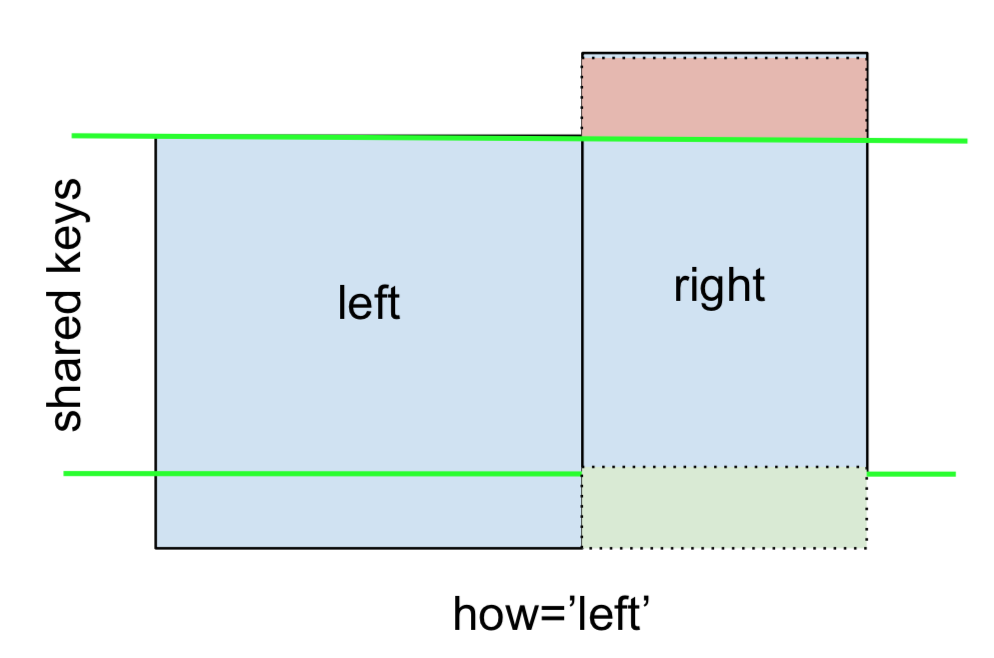
key value\_x value\_y

0 B 0.400157 1.867558

1 D 2.240893 -0.977278

This returns only rows from left and right which share a common key (in this example, "B" and "D).

A **LEFT OUTER JOIN**, or LEFT JOIN is represented by



This can be performed by specifying how='left'.

left.merge(right, on='key', how='left')

key value\_x value\_y

0 A 1.764052 NaN

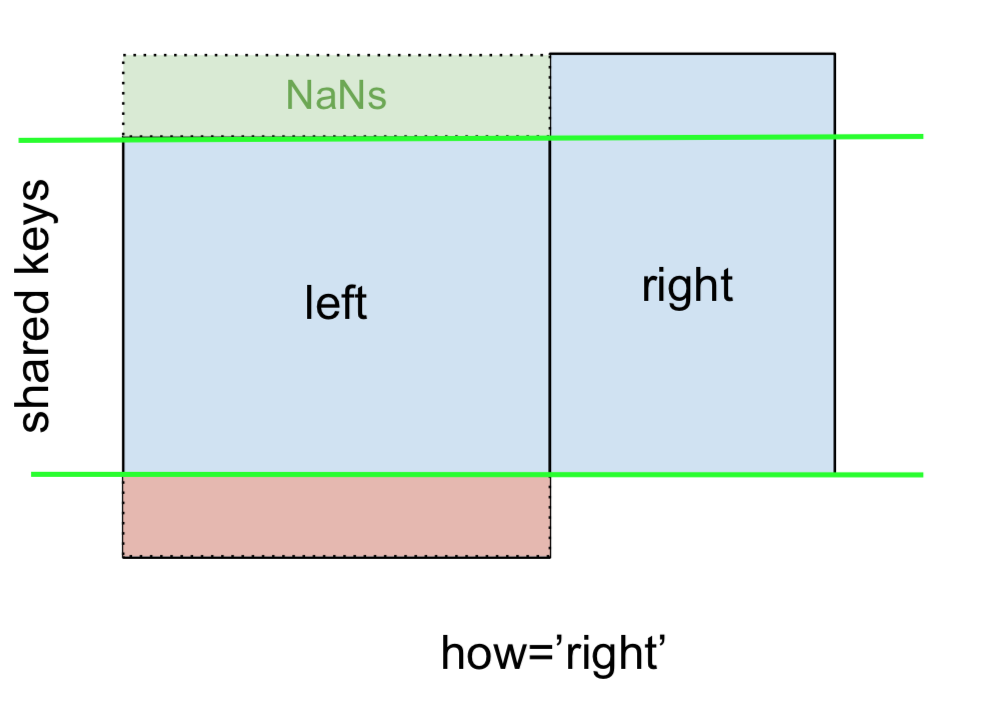
1 B 0.400157 1.867558

2 C 0.978738 NaN

3 D 2.240893 -0.977278

Carefully note the placement of NaNs here. If you specify how='left', then only keys from left are used, and missing data from right is replaced by NaN.

And similarly, for a **RIGHT OUTER JOIN**, or RIGHT JOIN which is...



...specify how='right':

left.merge(right, on='key', how='right')

key value\_x value\_y

0 B 0.400157 1.867558

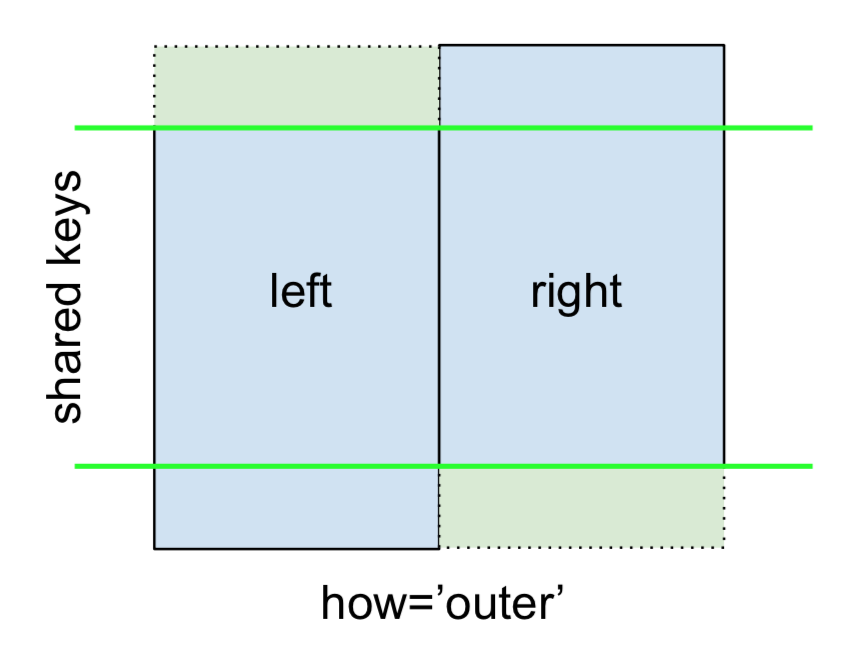
1 D 2.240893 -0.977278

2 E NaN 0.950088

3 F NaN -0.151357

Here, keys from right are used, and missing data from left is replaced by NaN.

Finally, for the **FULL OUTER JOIN**, given by



specify how='outer'.

left.merge(right, on='key', how='outer')

key value\_x value\_y

0 A 1.764052 NaN

1 B 0.400157 1.867558

2 C 0.978738 NaN

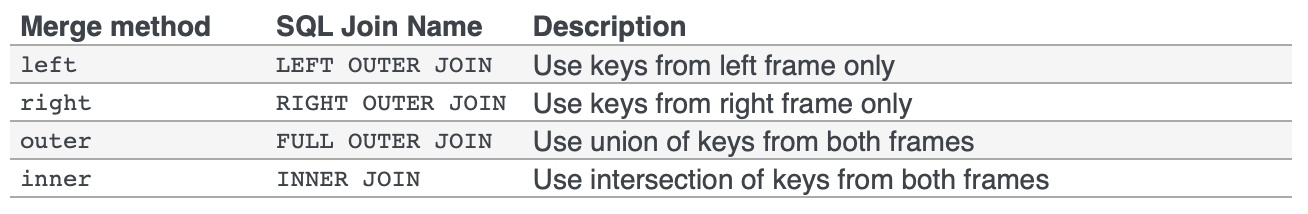
3 D 2.240893 -0.977278

4 E NaN 0.950088

5 F NaN -0.151357

This uses the keys from both frames, and NaNs are inserted for missing rows in both.

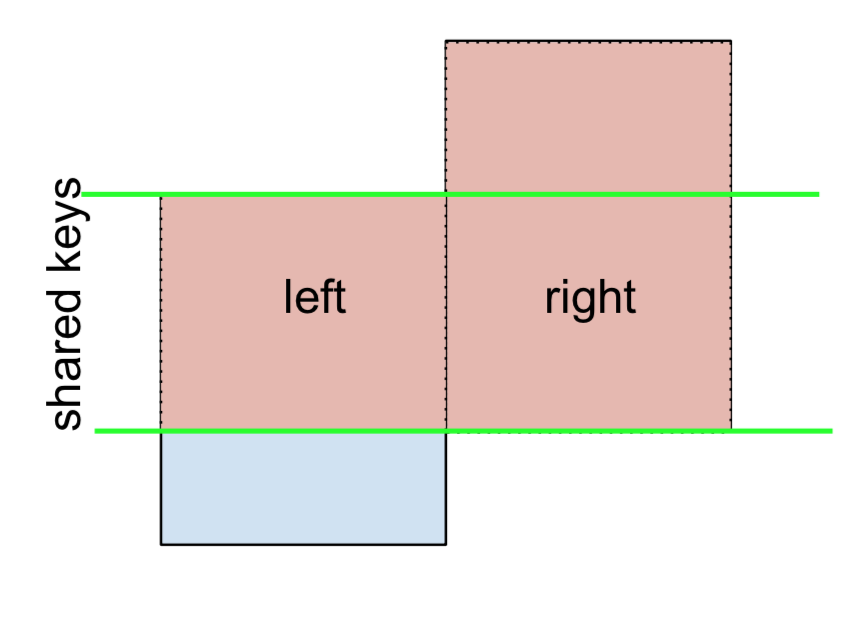
The documentation summarizes these various merges nicely:



**Other JOINs - LEFT-Excluding, RIGHT-Excluding, and FULL-Excluding/ANTI JOINs**

If you need **LEFT-Excluding JOINs** and **RIGHT-Excluding JOINs** in two steps.

For LEFT-Excluding JOIN, represented as



Start by performing a LEFT OUTER JOIN and then filtering to rows coming from left only (excluding everything from the right),

(left.merge(right, on='key', how='left', indicator=True)

.query('\_merge == "left\_only"')

.drop('\_merge', 1))

key value\_x value\_y

0 A 1.764052 NaN

2 C 0.978738 NaN

Where,

left.merge(right, on='key', how='left', **indicator=True**)

key value\_x value\_y \_merge

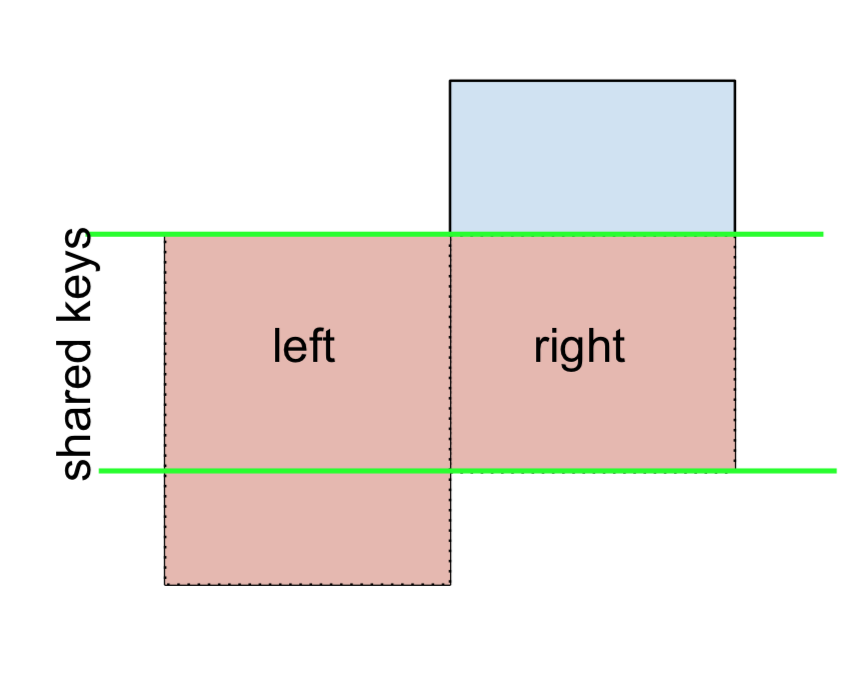
0 A 1.764052 NaN left\_only

1 B 0.400157 1.867558 both

2 C 0.978738 NaN left\_only

3 D 2.240893 -0.977278 both

And similarly, for a RIGHT-Excluding JOIN,



(left.merge(right, on='key', how='right', **indicator=True**)

.query('\_merge == "right\_only"')

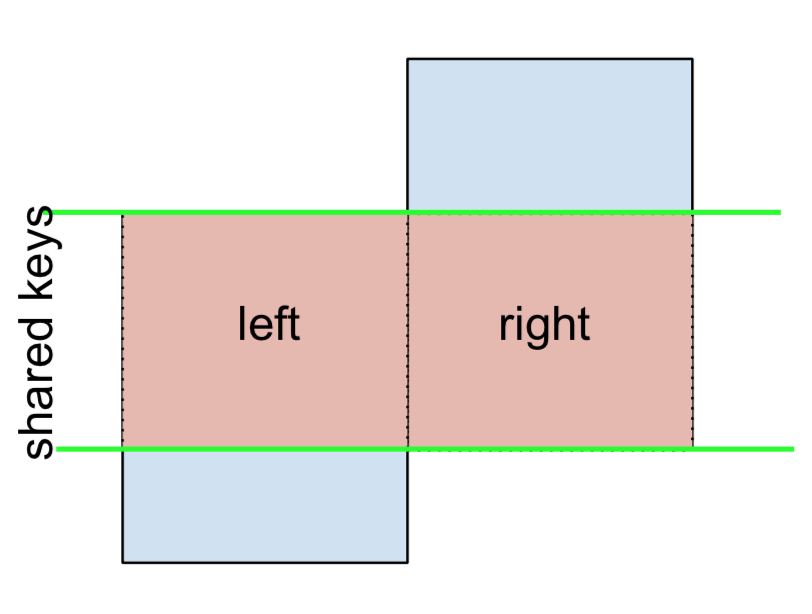
.drop('\_merge', 1))

key value\_x value\_y

2 E NaN 0.950088

3 F NaN -0.151357

Lastly, if you are required to do a merge that only retains keys from the left or right, but not both (IOW, performing an **ANTI-JOIN**),



You can do this in similar fashion—

(left.merge(right, on='key', how='outer', indicator=True)

.query('\_merge != "both"')

.drop('\_merge', 1))

key value\_x value\_y

0 A 1.764052 NaN

2 C 0.978738 NaN

4 E NaN 0.950088

5 F NaN -0.151357

**Different names for key columns**

If the key columns are named differently—for example, left has keyLeft, and right has keyRight instead of key—then you will have to specify left\_on and right\_on as arguments instead of on:

left2 = left.rename({'key':'keyLeft'}, axis=1)

right2 = right.rename({'key':'keyRight'}, axis=1)

left2

keyLeft value

0 A 1.764052

1 B 0.400157

2 C 0.978738

3 D 2.240893

right2

keyRight value

0 B 1.867558

1 D -0.977278

2 E 0.950088

3 F -0.151357

left2.merge(right2, left\_on='keyLeft', right\_on='keyRight', how='inner')

keyLeft value\_x keyRight value\_y

0 B 0.400157 B 1.867558

1 D 2.240893 D -0.977278

**Avoiding duplicate key column in output**

When merging on keyLeft from left and keyRight from right, if you only want either of the keyLeft or keyRight (but not both) in the output, you can start by setting the index as a preliminary step.

left3 = left2.set\_index('keyLeft')

left3.merge(right2, left\_index=True, right\_on='keyRight')

value\_x keyRight value\_y

0 0.400157 B 1.867558

1 2.240893 D -0.977278

Contrast this with the output of the command just before (that is, the output of left2.merge(right2, left\_on='keyLeft', right\_on='keyRight', how='inner')), you'll notice keyLeft is missing. You can figure out what column to keep based on which frame's index is set as the key. This may matter when, say, performing some OUTER JOIN operation.

**Merging only a single column from one of the DataFrames**

For example, consider