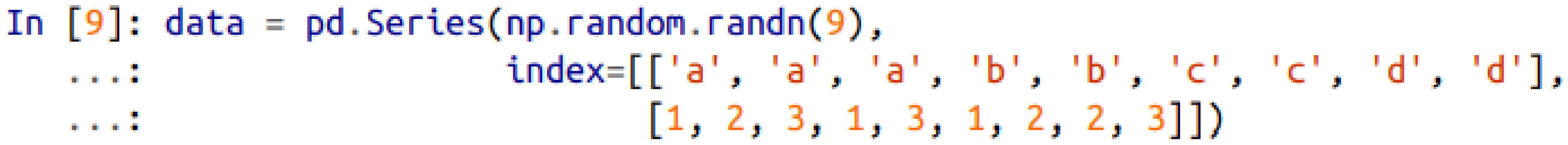
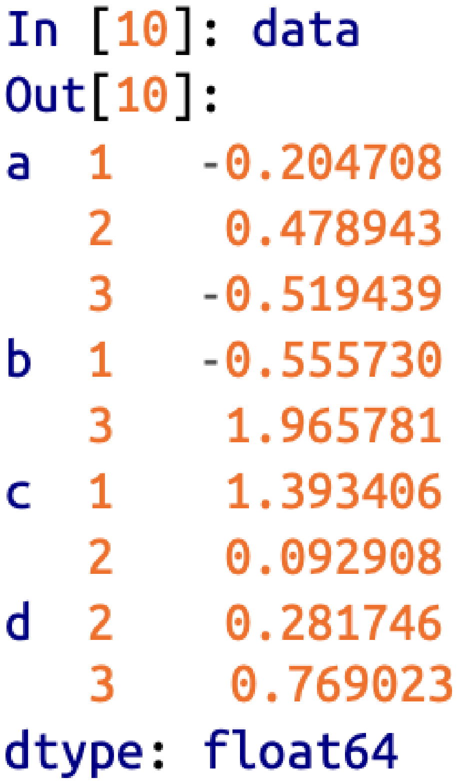
Lecture 7: Data Wrangling: Join, Combine and Reshape

Alpar Sultan, PhD, Associate professor

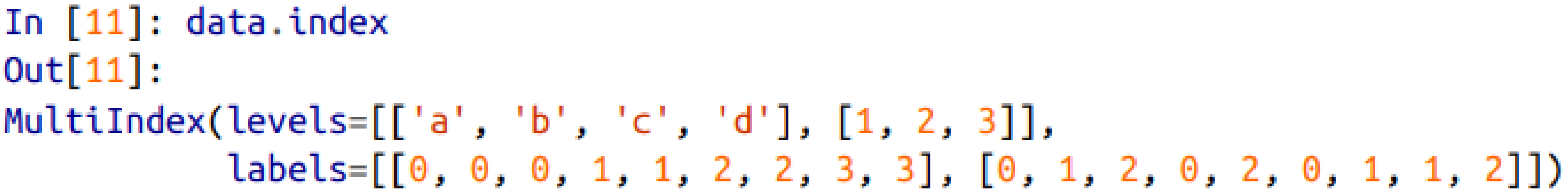
Hierarchical Indexing



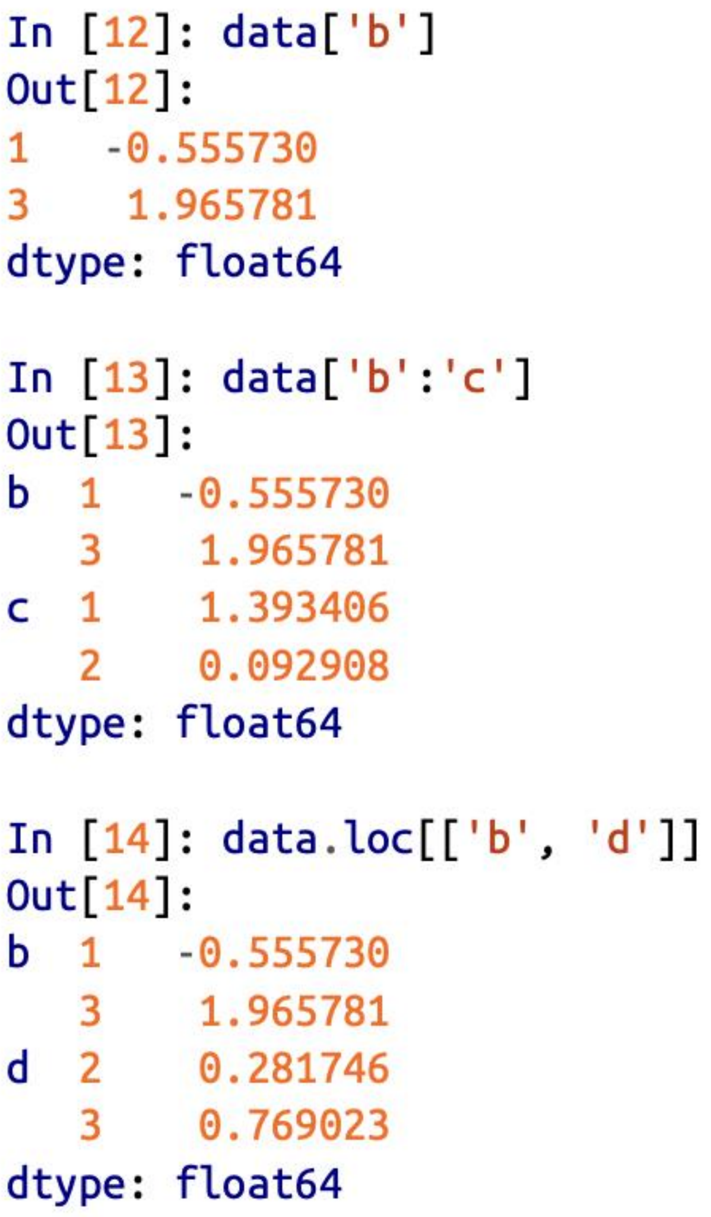
```
In [9]: data = pd.Series(np.random.randn(9),
   ...:                  index=[['a', 'a', 'a', 'b', 'b', 'c', 'c', 'd', 'd'],
   ...:                         [1, 2, 3, 1, 3, 1, 2, 2, 3]])
```



```
In [10]: data
Out[10]:
a  1   -0.204708
   2    0.478943
   3   -0.519439
b  1   -0.555730
   3    1.965781
c  1    1.393406
   2    0.092908
d  2    0.281746
   3    0.769023
dtype: float64
```

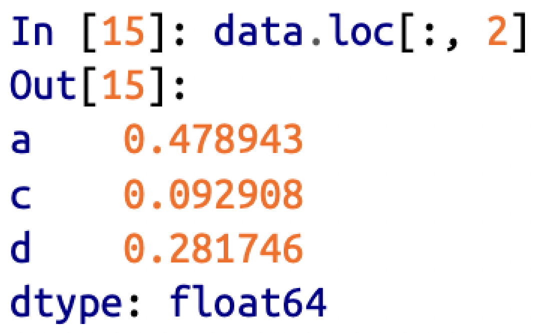


```
In [11]: data.index
Out[11]:
MultiIndex(levels=[['a', 'b', 'c', 'd'], [1, 2, 3]],
           labels=[[0, 0, 0, 1, 1, 2, 2, 3, 3], [0, 1, 2, 0, 2, 0, 1, 1, 2]])
```

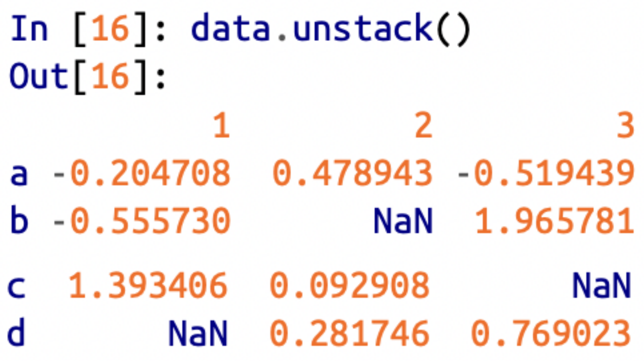


```
In [12]: data['b']
Out[12]:
1   -0.555730
3    1.965781
dtype: float64

In [13]: data['b':'c']
Out[13]:
b  1   -0.555730
   3    1.965781
c  1    1.393406
   2    0.092908
dtype: float64

In [14]: data.loc[['b', 'd']]
Out[14]:
b  1   -0.555730
   3    1.965781
d  2    0.281746
   3    0.769023
dtype: float64
```

Selection is even possible from an “inner” level:

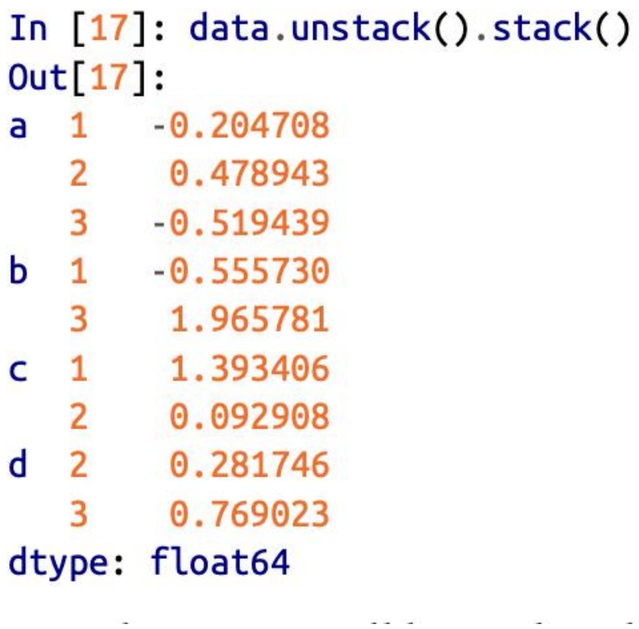


```
In [15]: data.loc[:, 2]
Out[15]:
a    0.478943
c    0.092908
d    0.281746
dtype: float64
```



```
In [16]: data.unstack()
Out[16]:
          1         2         3
a -0.204708  0.478943 -0.519439
b -0.555730       NaN  1.965781
c  1.393406  0.092908       NaN
d       NaN  0.281746  0.769023
```

The inverse operation of unstack is stack:



```
In [17]: data.unstack().stack()
Out[17]:
a  1   -0.204708
   2    0.478943
   3   -0.519439
b  1   -0.555730
   3    1.965781
c  1    1.393406
   2    0.092908
d  2    0.281746
   3    0.769023
dtype: float64
```

```
In [18]: frame = pd.DataFrame(np.arange(12).reshape((4, 3)),
   ....:                      index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
   ....:                      columns=[['Ohio', 'Ohio', 'Colorado'],
   ....:                               ['Green', 'Red', 'Green']])
```

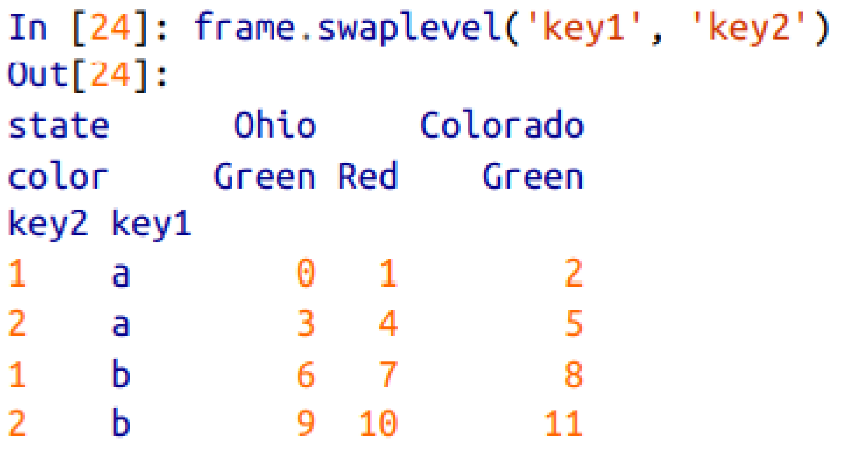
```
In [19]: frame
Out[19]:
     Ohio     Colorado
    Green Red    Green
a 1     0   1        2
  2     3   4        5
b 1     6   7        8
  2     9  10       11
```

```
In [20]: frame.index.names = ['key1', 'key2']

In [21]: frame.columns.names = ['state', 'color']

In [22]: frame
Out[22]:
state      Ohio     Colorado
color     Green Red    Green
key1 key2
a    1        0   1        2
     2        3   4        5
b    1        6   7        8
     2        9  10       11
```

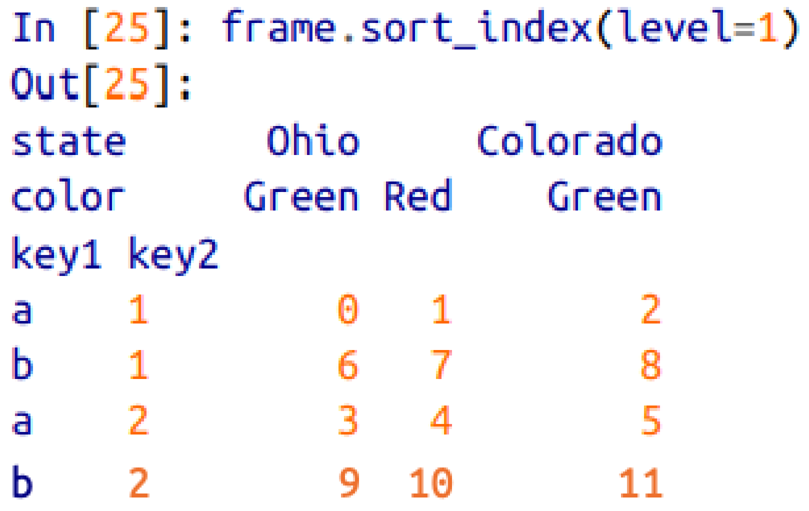
```
In [23]: frame['Ohio']
Out[23]:
color      Green  Red
key1 key2
a    1         0    1
     2         3    4
b    1         6    7
     2         9   10
```

```
MultiIndex.from_arrays([['Ohio', 'Ohio', 'Colorado'], ['Green', 'Red', 'Green']],
                       names=['state', 'color'])
```

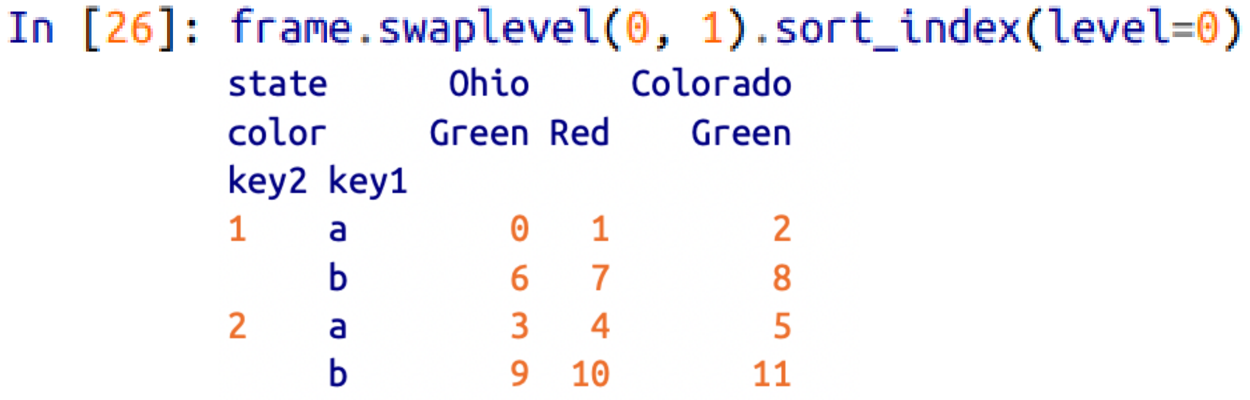
Reordering and Sorting Levels



```
In [24]: frame.swaplevel('key1', 'key2')
Out[24]:
state      Ohio     Colorado
color     Green Red    Green
key2 key1
1    a        0   1        2
2    a        3   4        5
1    b        6   7        8
2    b        9  10       11
```

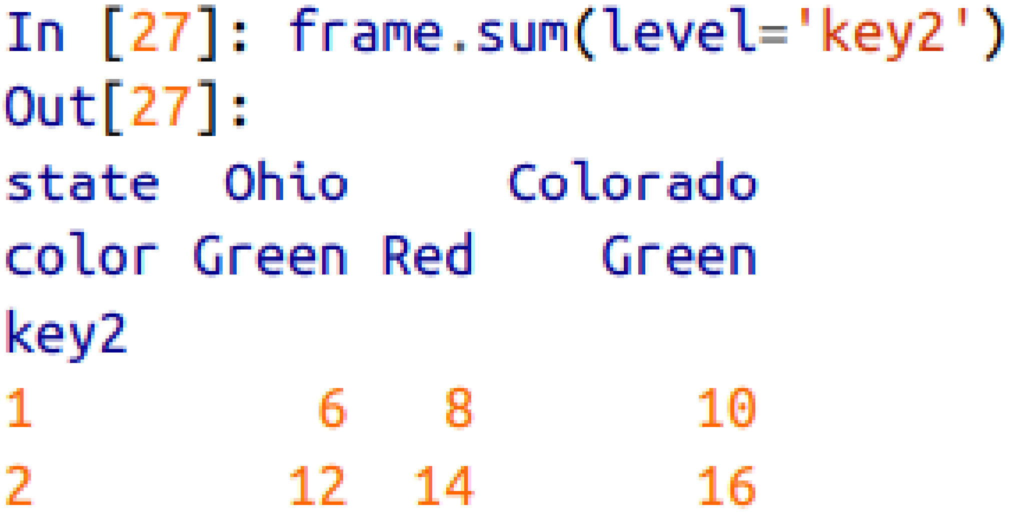


```
In [25]: frame.sort_index(level=1)
Out[25]:
state      Ohio     Colorado
color     Green Red    Green
key1 key2
a    1        0   1        2
b    1        6   7        8
a    2        3   4        5
b    2        9  10       11
```

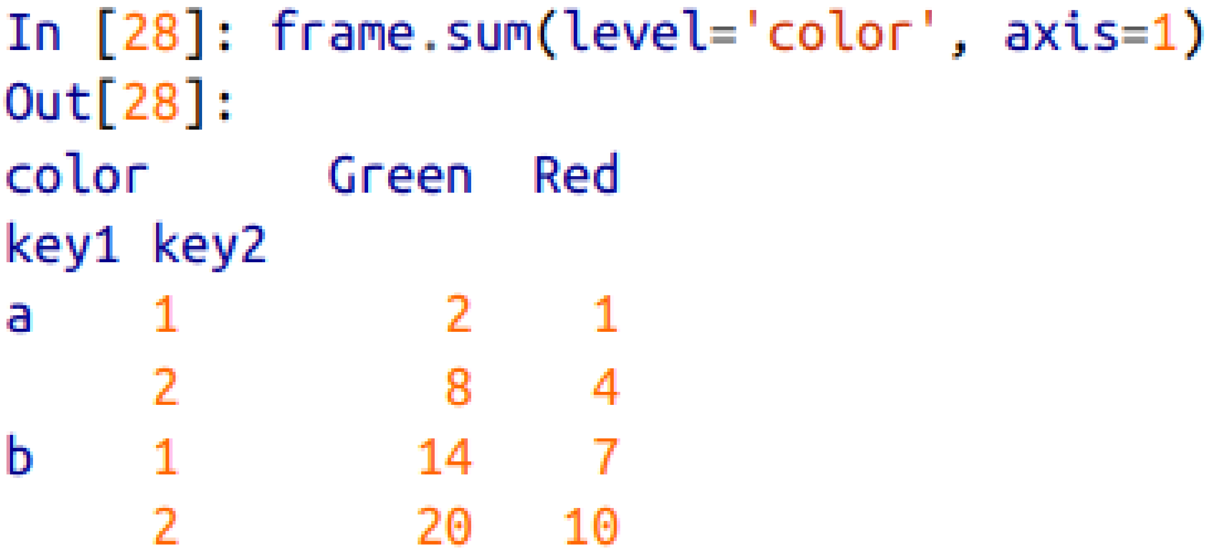


```
In [26]: frame.swaplevel(0, 1).sort_index(level=0)
         state      Ohio     Colorado
         color     Green Red    Green
         key2 key1
         1    a        0   1        2
              b        6   7        8
         2    a        3   4        5
              b        9  10       11
```

Summary Statistics by Level



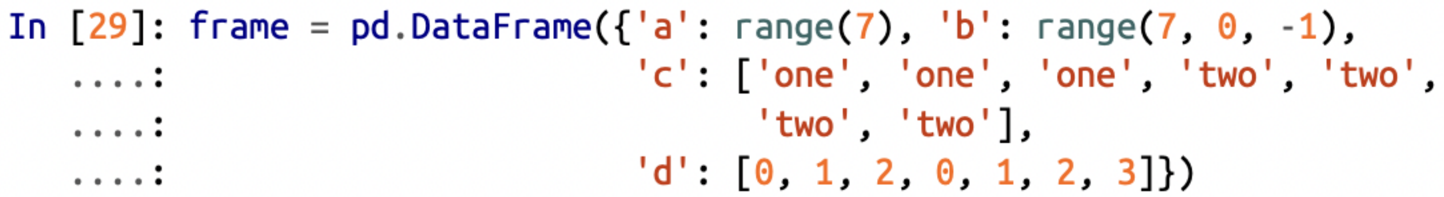
```
In [27]: frame.sum(level='key2')
Out[27]:
state  Ohio     Colorado
color Green Red    Green
key2
1         6   8       10
2        12  14       16
```



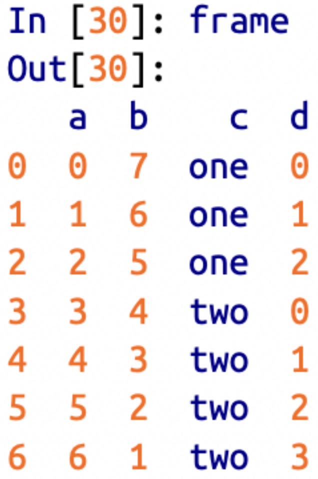
```
In [28]: frame.sum(level='color', axis=1)
Out[28]:
color      Green  Red
key1 key2
a    1         2    1
     2         8    4
b    1        14    7
     2        20   10
```

Indexing with a DataFrame’s columns

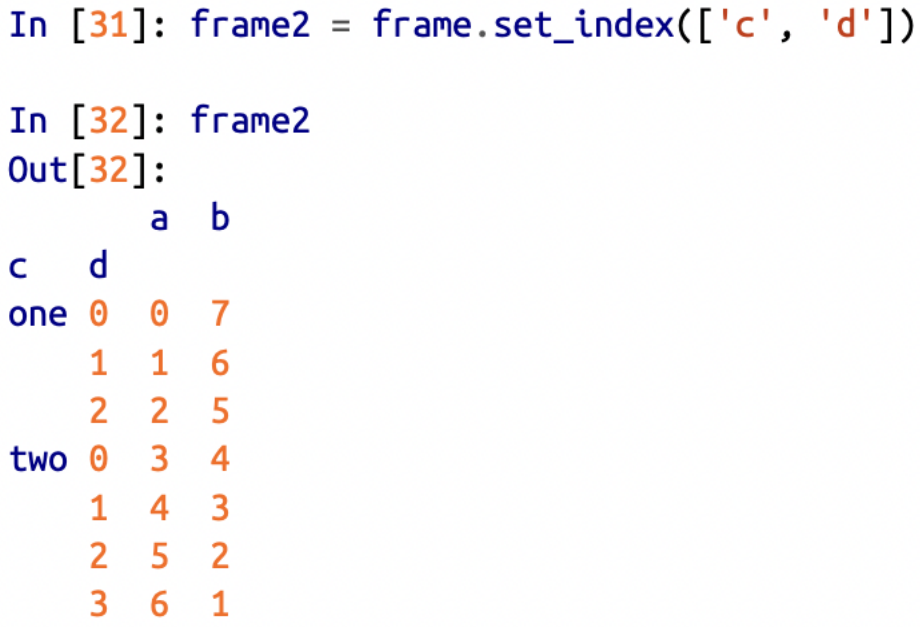
- DataFrame’s set_index function will create a new DataFrame using one or more of its columns as the index:



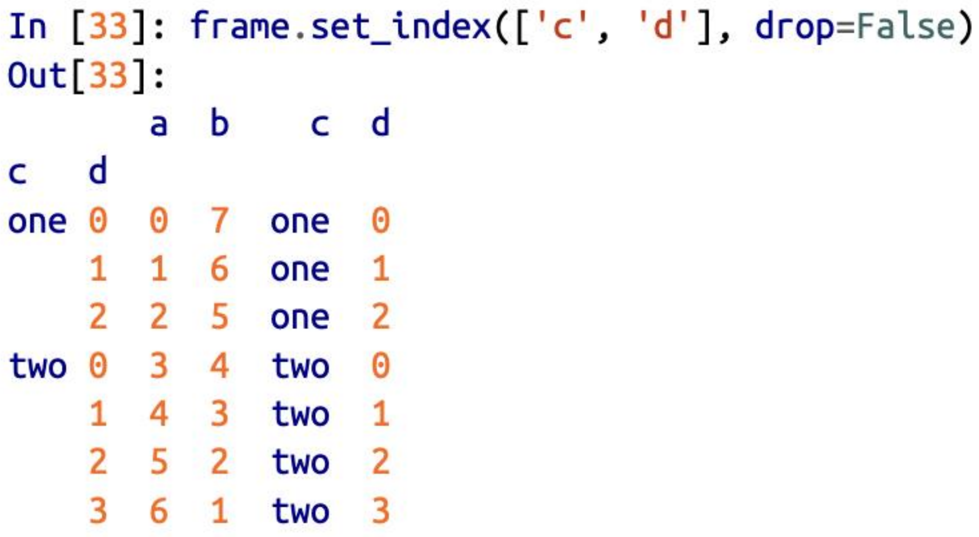
```
In [29]: frame = pd.DataFrame({'a': range(7), 'b': range(7, 0, -1),
   ....:                       'c': ['one', 'one', 'one', 'two', 'two',
   ....:                             'two', 'two'],
   ....:                       'd': [0, 1, 2, 0, 1, 2, 3]})
```



```
In [30]: frame
Out[30]:
   a  b    c  d
0  0  7  one  0
1  1  6  one  1
2  2  5  one  2
3  3  4  two  0
4  4  3  two  1
5  5  2  two  2
6  6  1  two  3
```

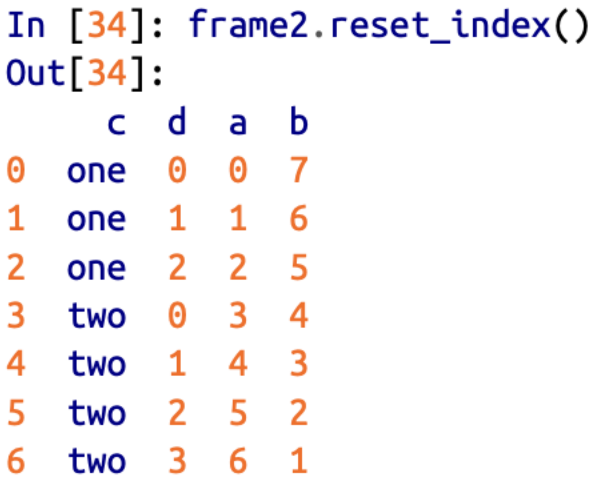


```
In [31]: frame2 = frame.set_index(['c', 'd'])

In [32]: frame2
Out[32]:
       a  b
c   d
one 0  0  7
    1  1  6
    2  2  5
two 0  3  4
    1  4  3
    2  5  2
    3  6  1
```



```
In [33]: frame.set_index(['c', 'd'], drop=False)
Out[33]:
       a  b    c  d
c   d
one 0  0  7  one  0
    1  1  6  one  1
    2  2  5  one  2
two 0  3  4  two  0
    1  4  3  two  1
    2  5  2  two  2
    3  6  1  two  3
```

reset_index, on the other hand, does the opposite of set_index; the hierarchical index levels are moved into the columns:

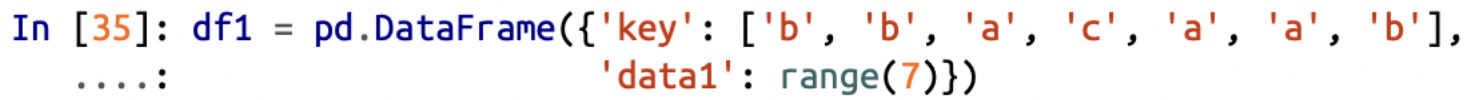


```
In [34]: frame2.reset_index()
Out[34]:
     c  d  a  b
0  one  0  0  7
1  one  1  1  6
2  one  2  2  5
3  two  0  3  4
4  two  1  4  3
5  two  2  5  2
6  two  3  6  1
```

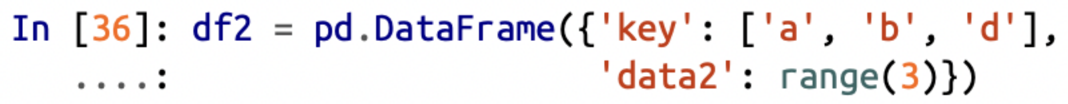
Combining and Merging Datasets

Data contained in pandas objects can be combined together in a number of ways:

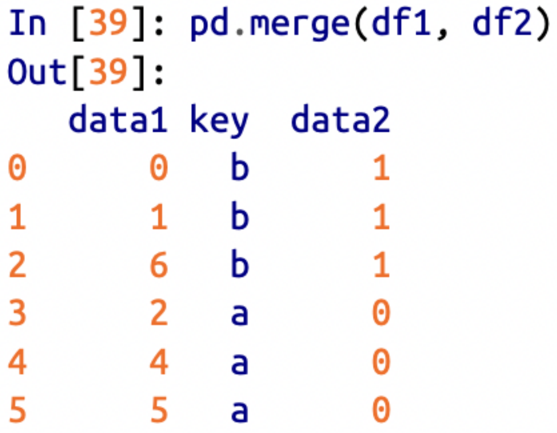
- pandas.merge connects rows in DataFrames based on one or more keys. This will be familiar to users of SQL or other relational databases, as it implements database join operations.
- pandas.concat concatenates or “stacks” together objects along an axis.
- The combine_first instance method enables splicing together overlapping data to fill in missing values in one object with values from another.



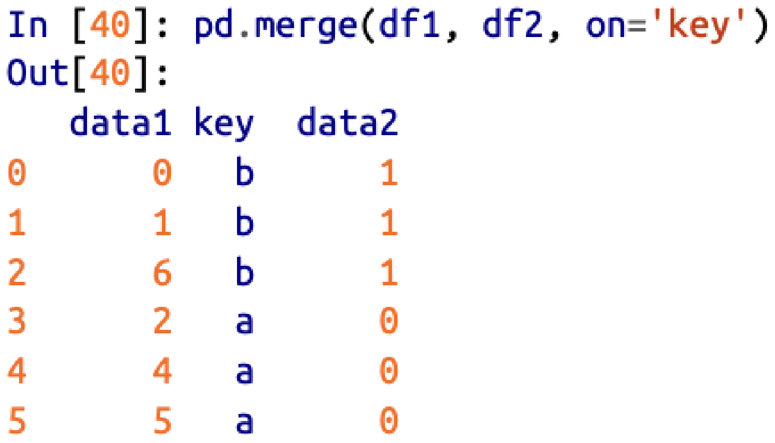
```
In [35]: df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
   ....:                     'data1': range(7)})
```



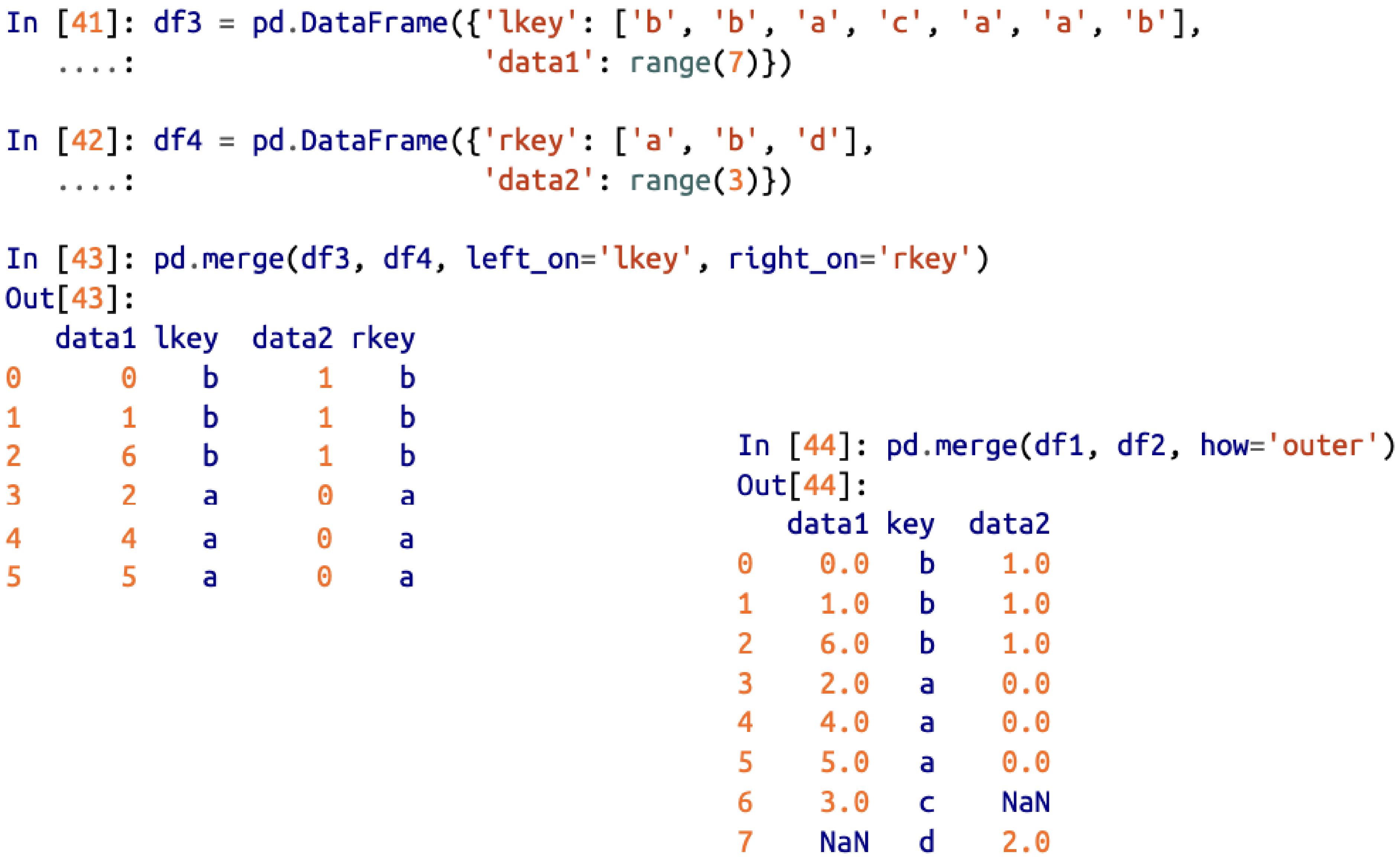
```
In [36]: df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
   ....:                     'data2': range(3)})
```



```
In [39]: pd.merge(df1, df2)
Out[39]:
   data1 key  data2
0      0   b      1
1      1   b      1
2      6   b      1
3      2   a      0
4      4   a      0
5      5   a      0
```



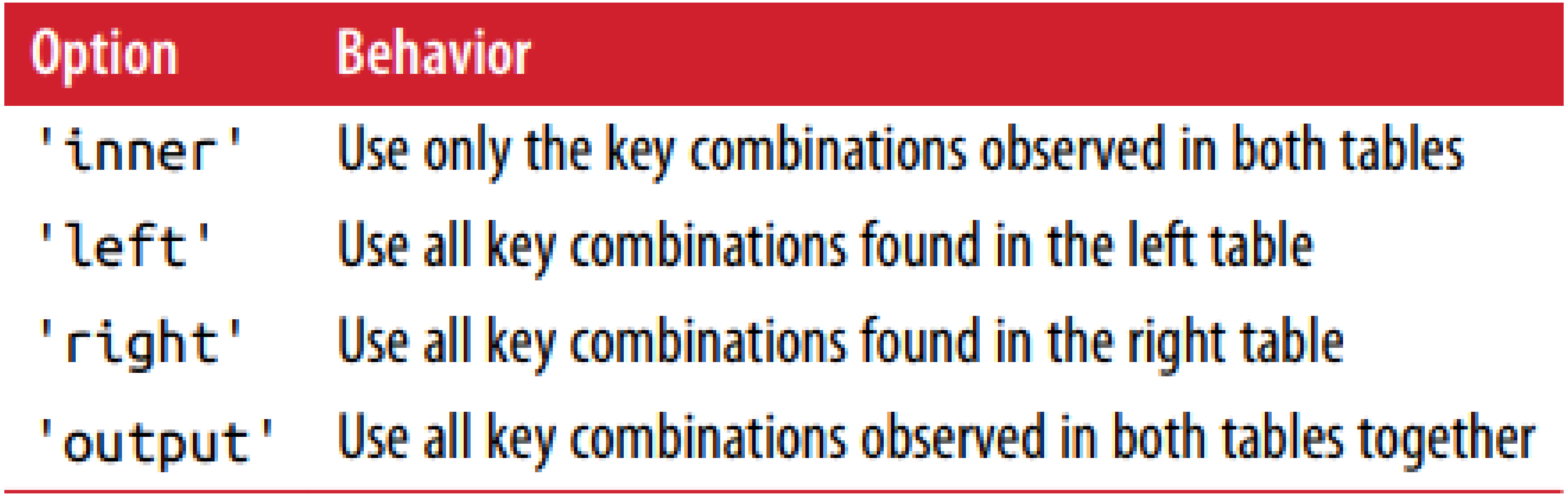
```
In [40]: pd.merge(df1, df2, on='key')
Out[40]:
   data1 key  data2
0      0   b      1
1      1   b      1
2      6   b      1
3      2   a      0
4      4   a      0
5      5   a      0
```



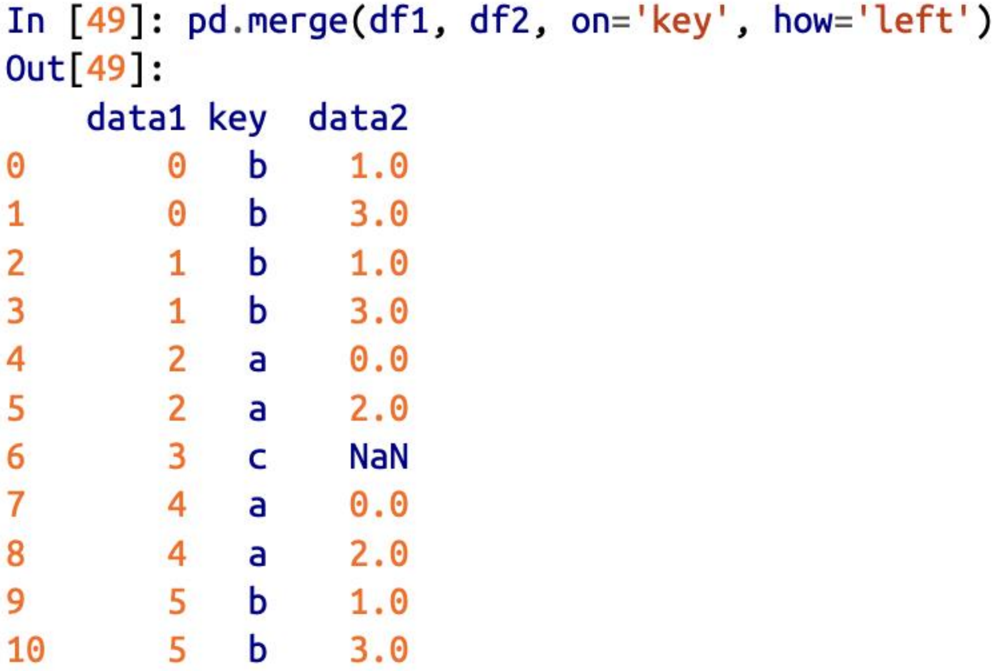
```
In [41]: df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
   ....:                     'data1': range(7)})

In [42]: df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
   ....:                     'data2': range(3)})

In [43]: pd.merge(df3, df4, left_on='lkey', right_on='rkey')
Out[43]:
   data1 lkey  data2 rkey
0      0    b      1    b
1      1    b      1    b
2      6    b      1    b
3      2    a      0    a
4      4    a      0    a
5      5    a      0    a
```

```
In [44]: pd.merge(df1, df2, how='outer')
Out[44]:
   data1 key  data2
0    0.0   b    1.0
1    1.0   b    1.0
2    6.0   b    1.0
3    2.0   a    0.0
4    4.0   a    0.0
5    5.0   a    0.0
6    3.0   c    NaN
7    NaN   d    2.0
```

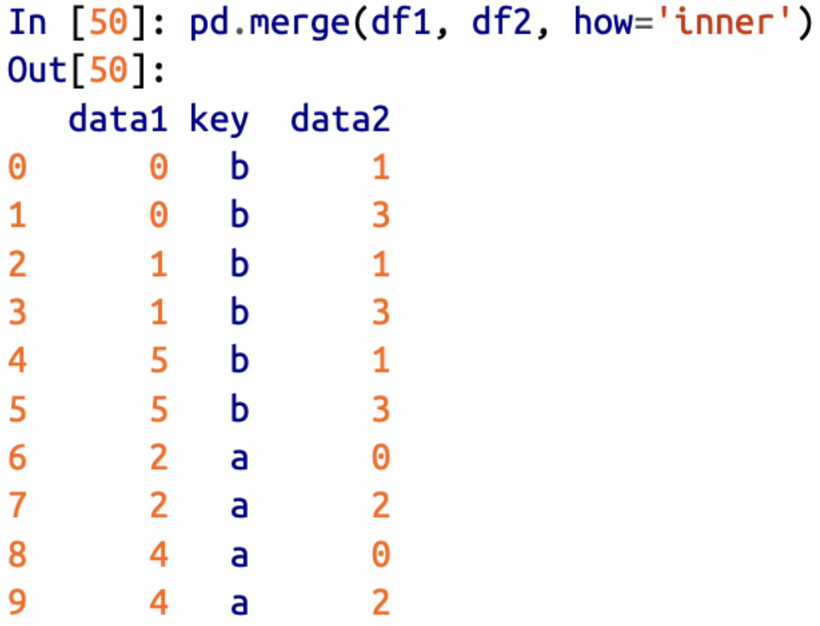
Database-Style DataFrame Joins



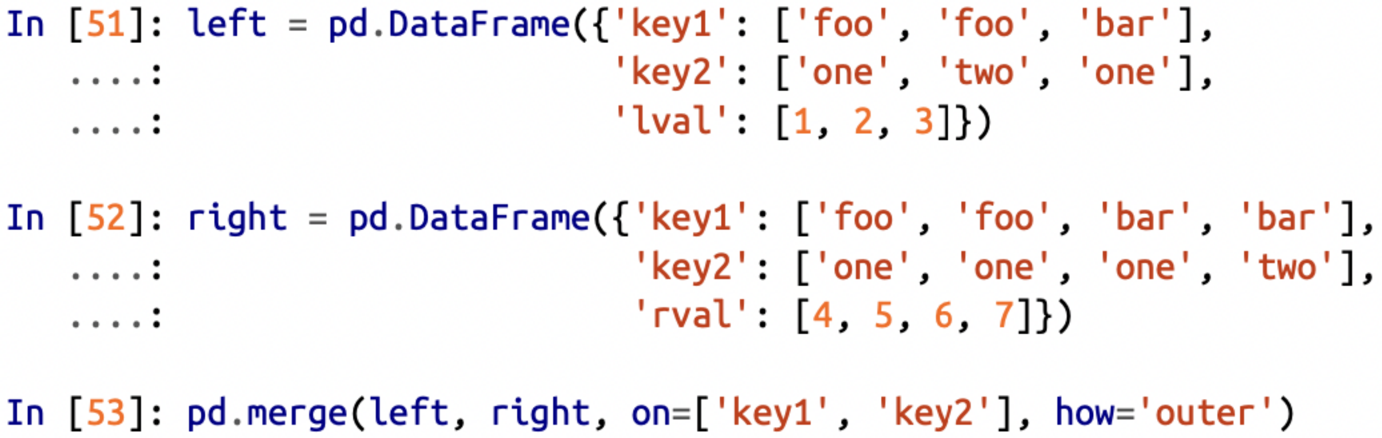
Option	Behavior
`'inner'`	Use only the key combinations observed in both tables
`'left'`	Use all key combinations found in the left table
`'right'`	Use all key combinations found in the right table
`'output'`	Use all key combinations observed in both tables together



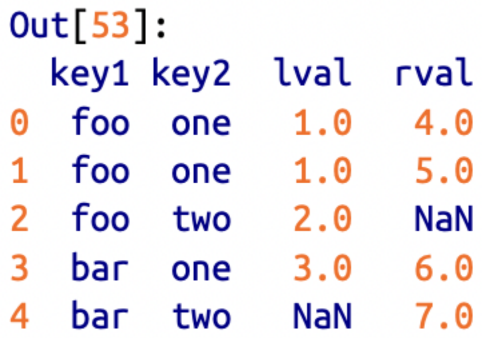
```
In [49]: pd.merge(df1, df2, on='key', how='left')
Out[49]:
    data1 key  data2
0       0   b    1.0
1       0   b    3.0
2       1   b    1.0
3       1   b    3.0
4       2   a    0.0
5       2   a    2.0
6       3   c    NaN
7       4   a    0.0
8       4   a    2.0
9       5   b    1.0
10      5   b    3.0
```



```
In [50]: pd.merge(df1, df2, how='inner')
Out[50]:
   data1 key  data2
0      0   b      1
1      0   b      3
2      1   b      1
3      1   b      3
4      5   b      1
5      5   b      3
6      2   a      0
7      2   a      2
8      4   a      0
9      4   a      2
```



```
In [51]: left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
   ....:                      'key2': ['one', 'two', 'one'],
   ....:                      'lval': [1, 2, 3]})

In [52]: right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
   ....:                       'key2': ['one', 'one', 'one', 'two'],
   ....:                       'rval': [4, 5, 6, 7]})

In [53]: pd.merge(left, right, on=['key1', 'key2'], how='outer')
```

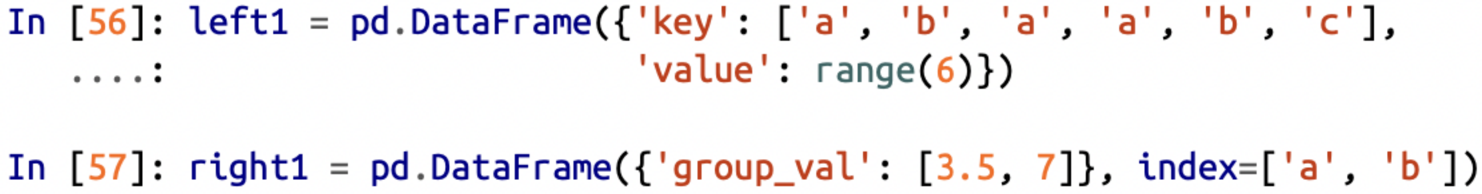


```
Out[53]:
  key1 key2  lval  rval
0  foo  one   1.0   4.0
1  foo  one   1.0   5.0
2  foo  two   2.0   NaN
3  bar  one   3.0   6.0
4  bar  two   NaN   7.0
```

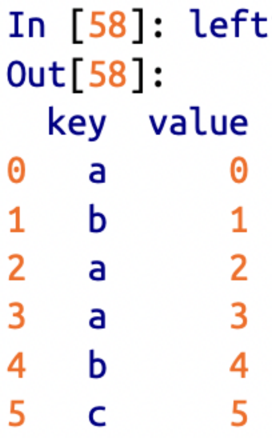
```
In [54]: pd.merge(left, right, on='key1')
Out[54]:
  key1 key2_x  lval key2_y  rval
0  foo    one     1    one     4
1  foo    one     1    one     5
2  foo    two     2    one     4
3  foo    two     2    one     5
4  bar    one     3    one     6
5  bar    one     3    two     7

In [55]: pd.merge(left, right, on='key1', suffixes=('_left', '_right'))
Out[55]:
  key1 key2_left  lval key2_right  rval
0  foo       one     1        one     4
1  foo       one     1        one     5
2  foo       two     2        one     4
3  foo       two     2        one     5
4  bar       one     3        one     6
5  bar       one     3        two     7
```

Argument	Description
`left`	DataFrame to be merged on the left side.
`right`	DataFrame to be merged on the right side.
`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 in `left` and `right` as 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`.
`sort`	Sort merged data lexicographically by join keys; `True` by default (disable to get better performance in some cases on large datasets).
`suffixes`	Tuple of string values to append to column names in case of overlap; defaults to `('_x', '_y')` (e.g., if `'data'` in both DataFrame objects, would appear as `'data_x'` and `'data_y'` in result).
`copy`	If `False`, avoid copying data into resulting data structure in some exceptional cases; by default always copies.
`indicator`	Adds a special column `_merge` that indicates the source of each row; values will be `'left_only'`, `'right_only'`, or `'both'` based on the origin of the joined data in each row.

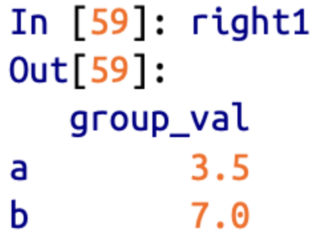
Merging on Index



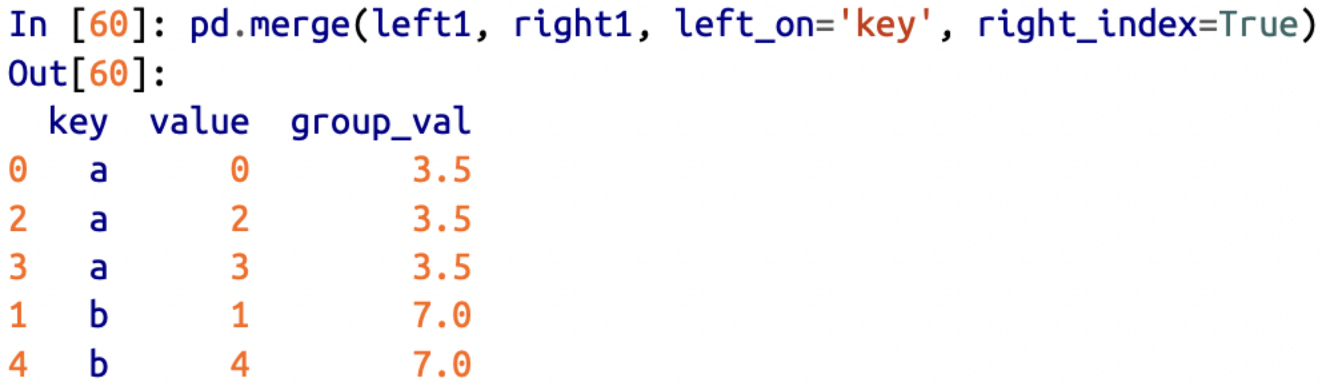
```
In [56]: left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
   ....:                        'value': range(6)})

In [57]: right1 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
```



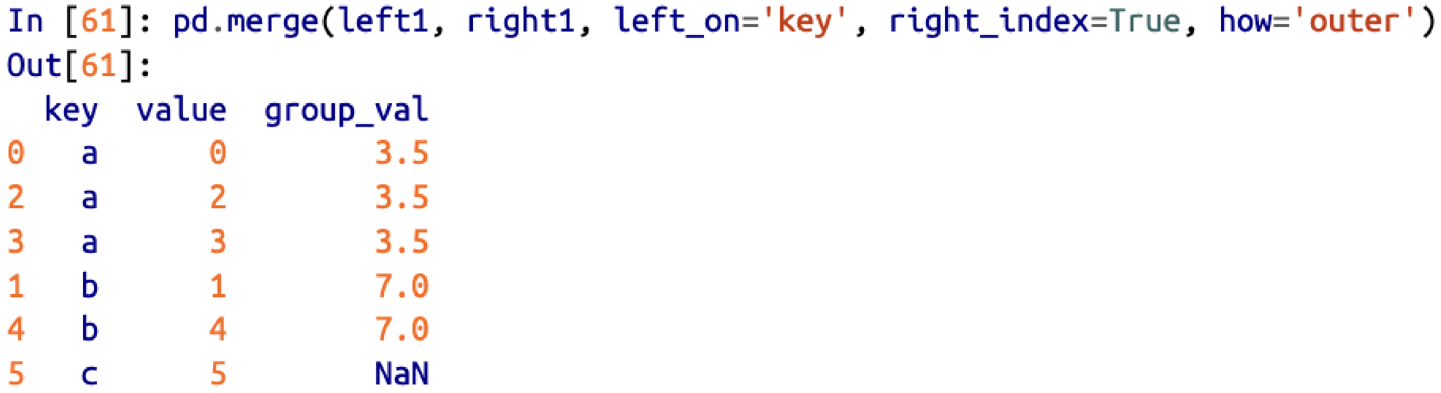
```
In [58]: left1
Out[58]:
  key  value
0   a      0
1   b      1
2   a      2
3   a      3
4   b      4
5   c      5
```



```
In [59]: right1
Out[59]:
   group_val
a        3.5
b        7.0
```



```
In [60]: pd.merge(left1, right1, left_on='key', right_index=True)
Out[60]:
  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
```



```
In [61]: pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
Out[61]:
  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
```

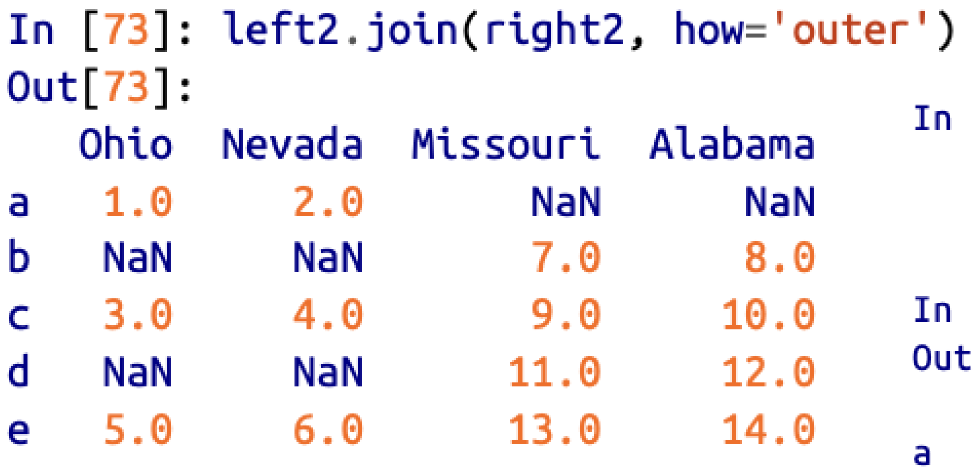
```
In [62]: lefth = pd.DataFrame({'key1': ['Ohio', 'Ohio', 'Ohio',
   ....:                               'Nevada', 'Nevada'],
   ....:                      'key2': [2000, 2001, 2002, 2001, 2002],
   ....:                      'data': np.arange(5.)})

In [63]: righth = pd.DataFrame(np.arange(12).reshape((6, 2)),
   ....:                       index=[['Nevada', 'Nevada', 'Ohio', 'Ohio',
   ....:                               'Ohio', 'Ohio'],
   ....:                              [2001, 2000, 2000, 2000, 2001, 2002]],
   ....:                       columns=['event1', 'event2'])
```

```
In [64]: lefth
Out[64]:
   data    key1  key2
0   0.0    Ohio  2000
1   1.0    Ohio  2001
2   2.0    Ohio  2002
3   3.0  Nevada  2001
4   4.0  Nevada  2002

In [65]: righth
Out[65]:
             event1  event2
Nevada 2001       0       1
       2000       2       3
Ohio   2000       4       5
       2000       6       7
       2001       8       9
       2002      10      11
```

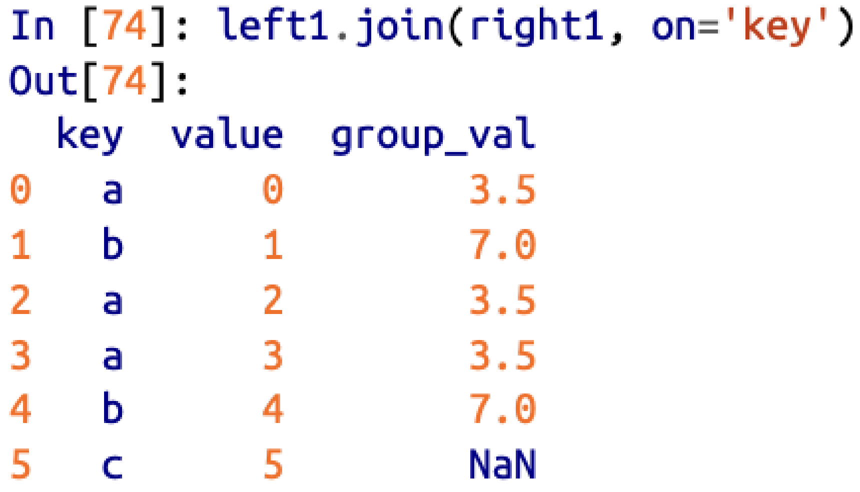
```
In [66]: pd.merge(lefth, righth, left_on=['key1', 'key2'], right_index=True)
Out[66]:
   data    key1  key2  event1  event2
0   0.0    Ohio  2000       4       5
0   0.0    Ohio  2000       6       7
1   1.0    Ohio  2001       8       9
2   2.0    Ohio  2002      10      11
3   3.0  Nevada  2001       0       1

In [67]: pd.merge(lefth, righth, left_on=['key1', 'key2'],
   ....:          right_index=True, how='outer')
Out[67]:
   data    key1  key2  event1  event2
0   0.0    Ohio  2000     4.0     5.0
0   0.0    Ohio  2000     6.0     7.0
1   1.0    Ohio  2001     8.0     9.0
2   2.0    Ohio  2002    10.0    11.0
3   3.0  Nevada  2001     0.0     1.0
4   4.0  Nevada  2002     NaN     NaN
4   NaN  Nevada  2000     2.0     3.0
```

```
In [68]: left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],
   ....:                      index=['a', 'c', 'e'],
   ....:                      columns=['Ohio', 'Nevada'])

In [69]: right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14]],
   ....:                       index=['b', 'c', 'd', 'e'],
   ....:                       columns=['Missouri', 'Alabama'])

In [70]: left2
Out[70]:
   Ohio  Nevada
a   1.0     2.0
c   3.0     4.0
e   5.0     6.0

In [71]: right2
Out[71]:
   Missouri  Alabama
b       7.0      8.0
c       9.0     10.0
d      11.0     12.0
e      13.0     14.0

In [72]: pd.merge(left2, right2, how='outer', left_index=True, right_index=True)
```

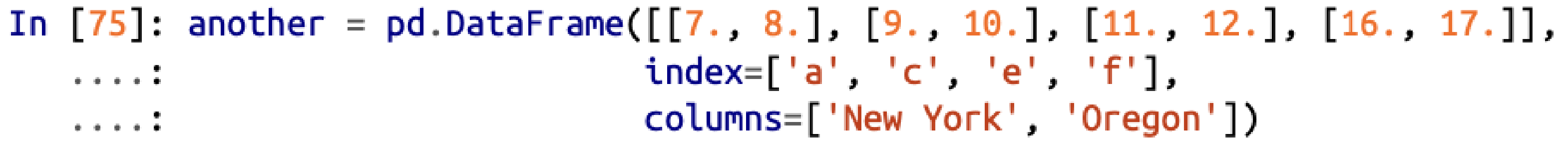
```
Out[72]:
   Ohio  Nevada  Missouri  Alabama
a   1.0     2.0       NaN      NaN
b   NaN     NaN       7.0      8.0
c   3.0     4.0       9.0     10.0
d   NaN     NaN      11.0     12.0
e   5.0     6.0      13.0     14.0
```



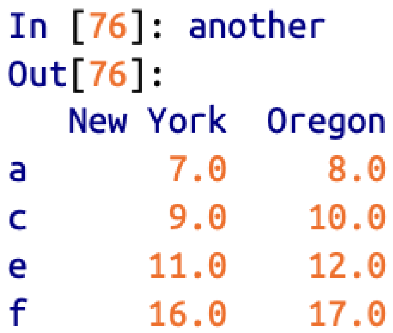
```
In [73]: left2.join(right2, how='outer')
Out[73]:
   Ohio  Nevada  Missouri  Alabama
a   1.0     2.0       NaN      NaN
b   NaN     NaN       7.0      8.0
c   3.0     4.0       9.0     10.0
d   NaN     NaN      11.0     12.0
e   5.0     6.0      13.0     14.0
```



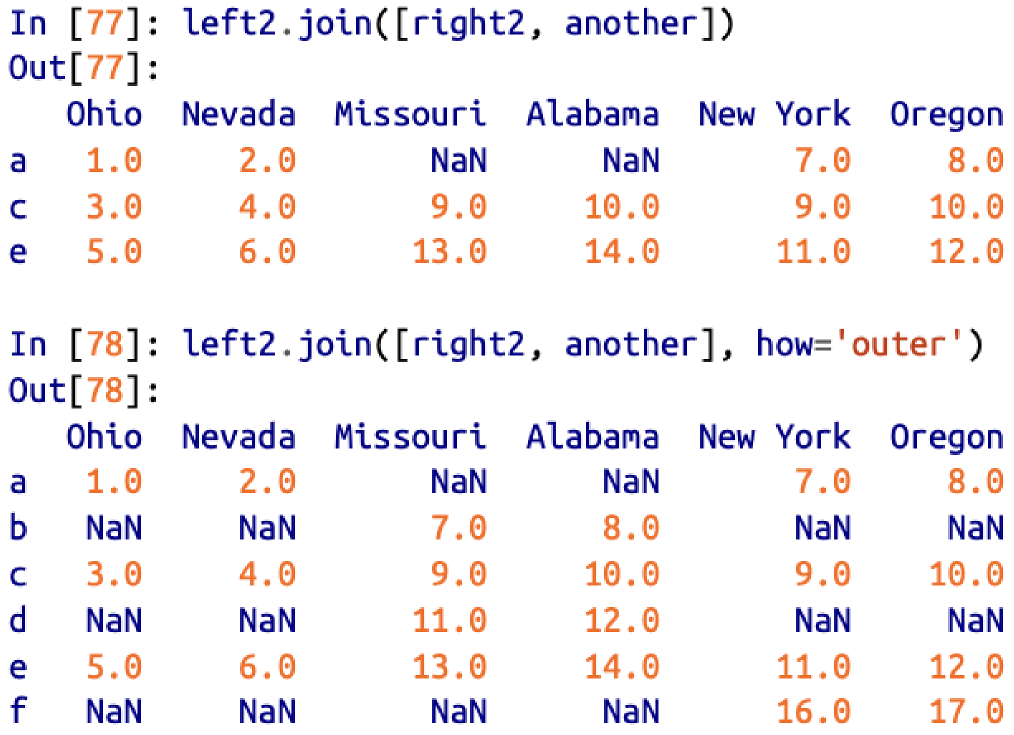
```
In [74]: left1.join(right1, on='key')
Out[74]:
  key  value  group_val
0   a      0        3.5
1   b      1        7.0
2   a      2        3.5
3   a      3        3.5
4   b      4        7.0
5   c      5        NaN
```



```
In [75]: another = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],
   ....:                        index=['a', 'c', 'e', 'f'],
   ....:                        columns=['New York', 'Oregon'])
```

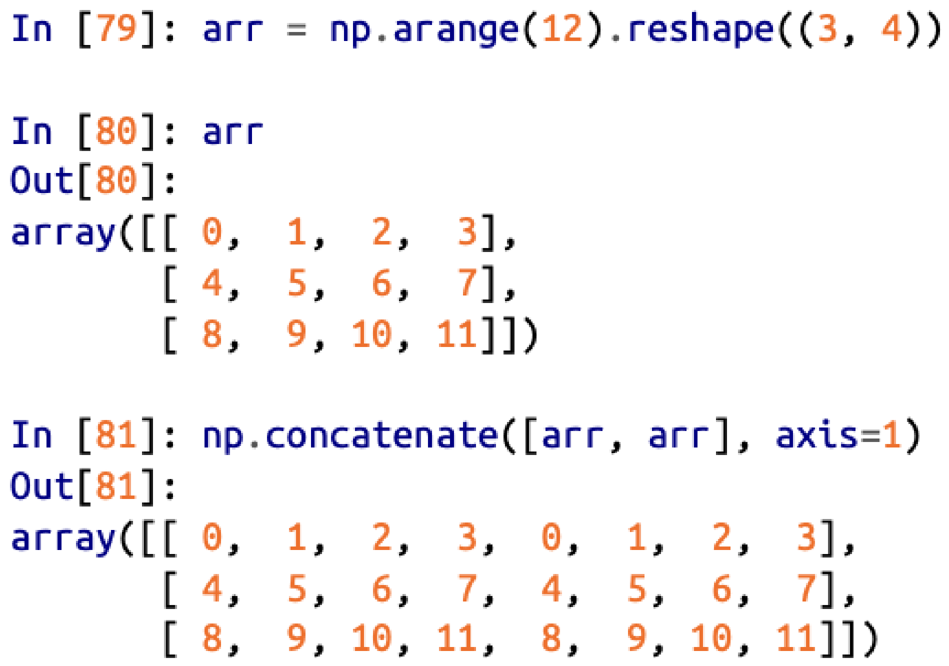


```
In [76]: another
Out[76]:
   New York  Oregon
a       7.0     8.0
c       9.0    10.0
e      11.0    12.0
f      16.0    17.0
```



```
In [77]: left2.join([right2, another])
Out[77]:
   Ohio  Nevada  Missouri  Alabama  New York  Oregon
a   1.0     2.0       NaN      NaN       7.0     8.0
c   3.0     4.0       9.0     10.0       9.0    10.0
e   5.0     6.0      13.0     14.0      11.0    12.0

In [78]: left2.join([right2, another], how='outer')
Out[78]:
   Ohio  Nevada  Missouri  Alabama  New York  Oregon
a   1.0     2.0       NaN      NaN       7.0     8.0
b   NaN     NaN       7.0      8.0       NaN     NaN
c   3.0     4.0       9.0     10.0       9.0    10.0
d   NaN     NaN      11.0     12.0       NaN     NaN
e   5.0     6.0      13.0     14.0      11.0    12.0
f   NaN     NaN       NaN      NaN      16.0    17.0
```

Concatenating Along an Axis

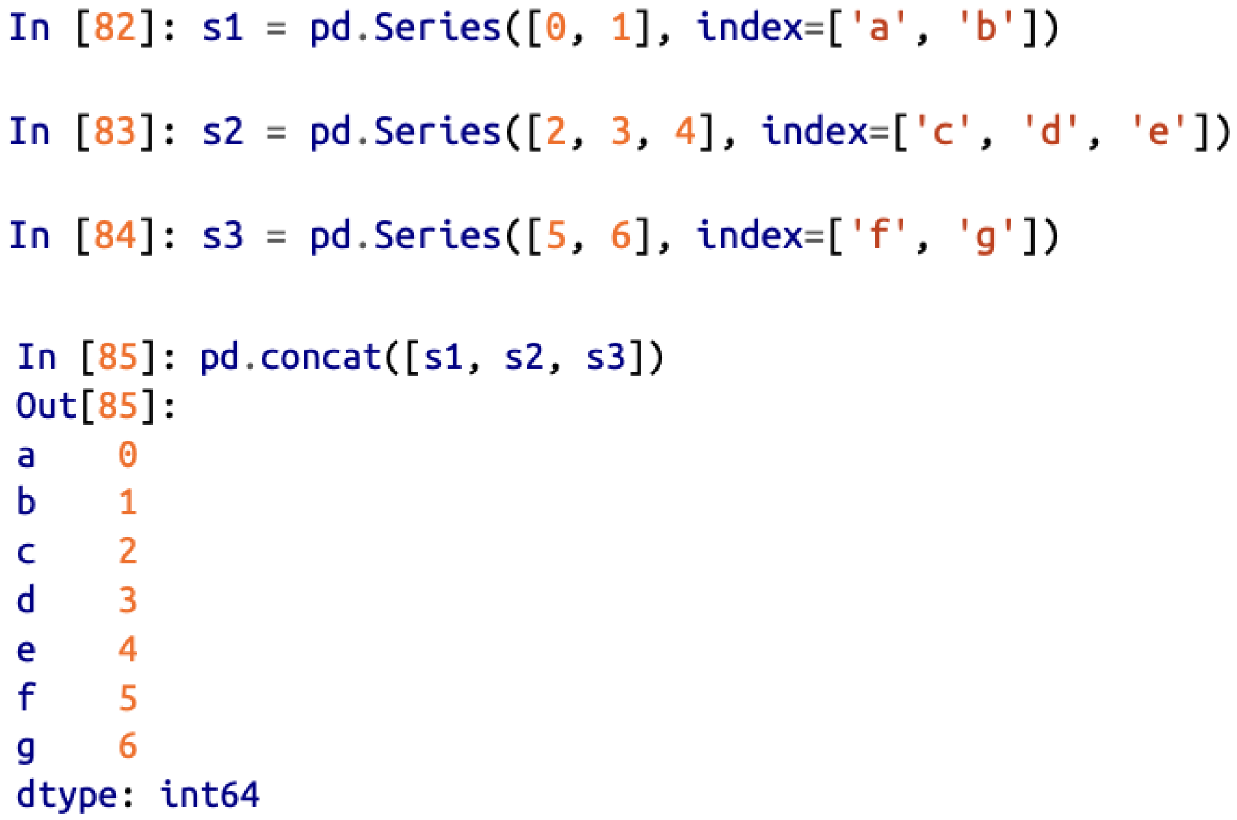


```
In [79]: arr = np.arange(12).reshape((3, 4))

In [80]: arr
Out[80]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])

In [81]: np.concatenate([arr, arr], axis=1)
Out[81]:
array([[ 0,  1,  2,  3,  0,  1,  2,  3],
       [ 4,  5,  6,  7,  4,  5,  6,  7],
       [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

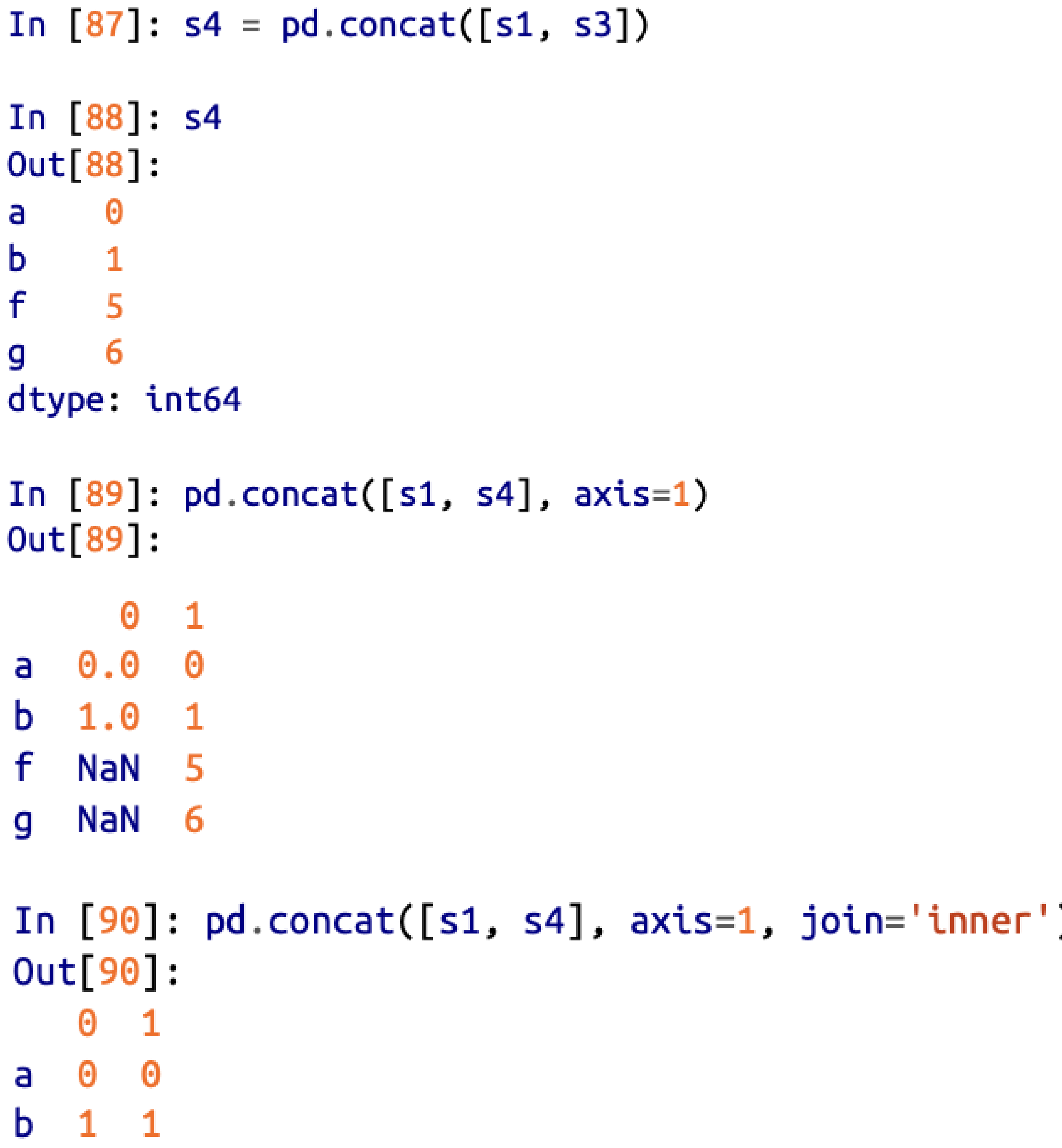
•If the objects are indexed differently on the other axes, should we combine the distinct elements in these axes or use only the shared values (the intersection)?

•Do the concatenated chunks of data need to be identifiable in the resulting object?

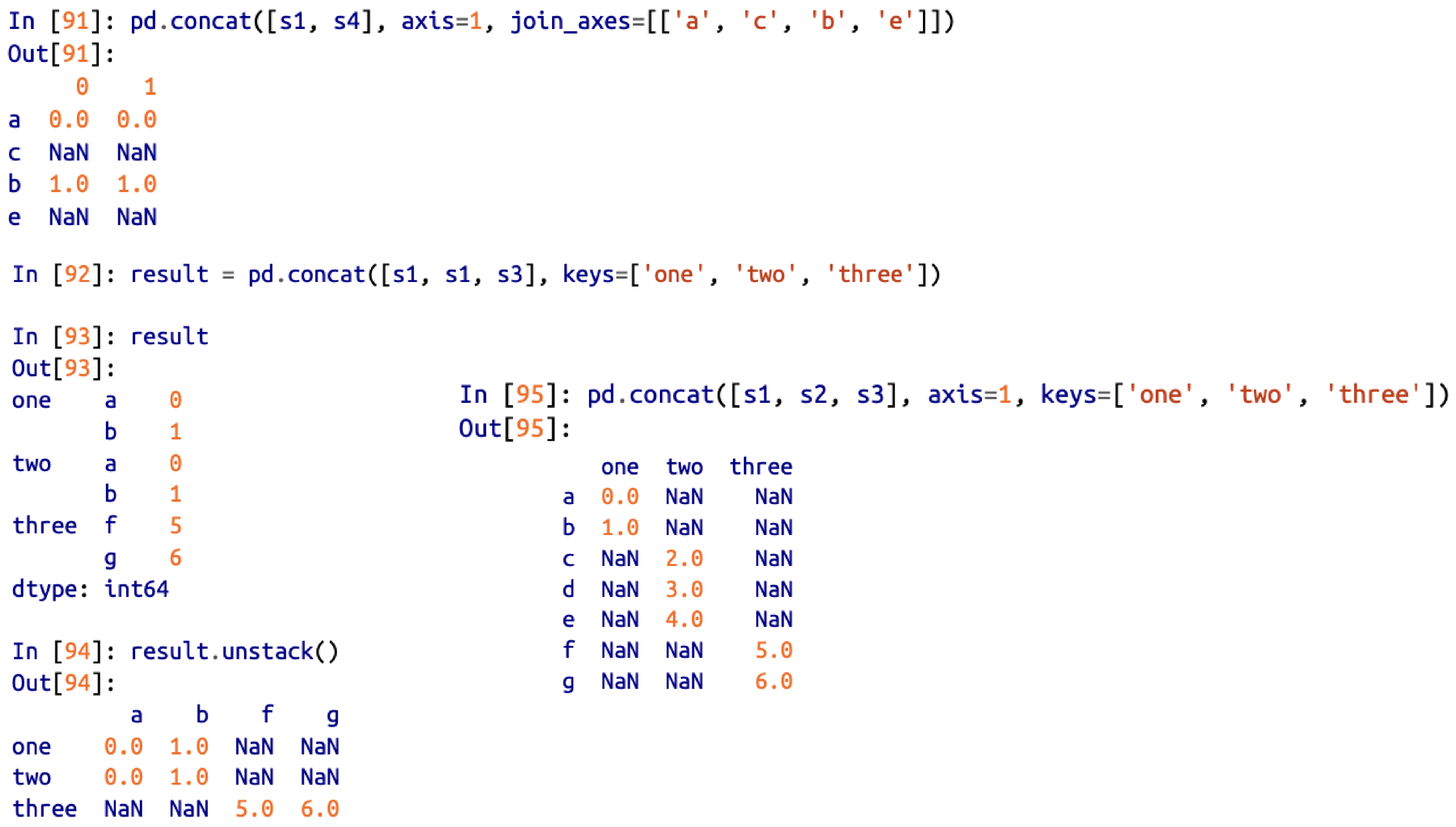
•Does the “concatenation axis” contain data that needs to be preserved? In many cases, the default integer labels in a DataFrame are best discarded during concatenation.



```
In [82]: s1 = pd.Series([0, 1], index=['a', 'b'])

In [83]: s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])

In [84]: s3 = pd.Series([5, 6], index=['f', 'g'])


In [85]: pd.concat([s1, s2, s3])
Out[85]:
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
```



```
In [87]: s4 = pd.concat([s1, s3])

In [88]: s4
Out[88]:
a    0
b    1
f    5
g    6
dtype: int64

In [89]: pd.concat([s1, s4], axis=1)
Out[89]:

     0  1
a  0.0  0
b  1.0  1
f  NaN  5
g  NaN  6

In [90]: pd.concat([s1, s4], axis=1, join='inner'
Out[90]:
   0  1
a  0  0
b  1  1
```



```
In [91]: pd.concat([s1, s4], axis=1, join_axes=[['a', 'c', 'b', 'e']])
Out[91]:
     0    1
a  0.0  0.0
c  NaN  NaN
b  1.0  1.0
e  NaN  NaN
```

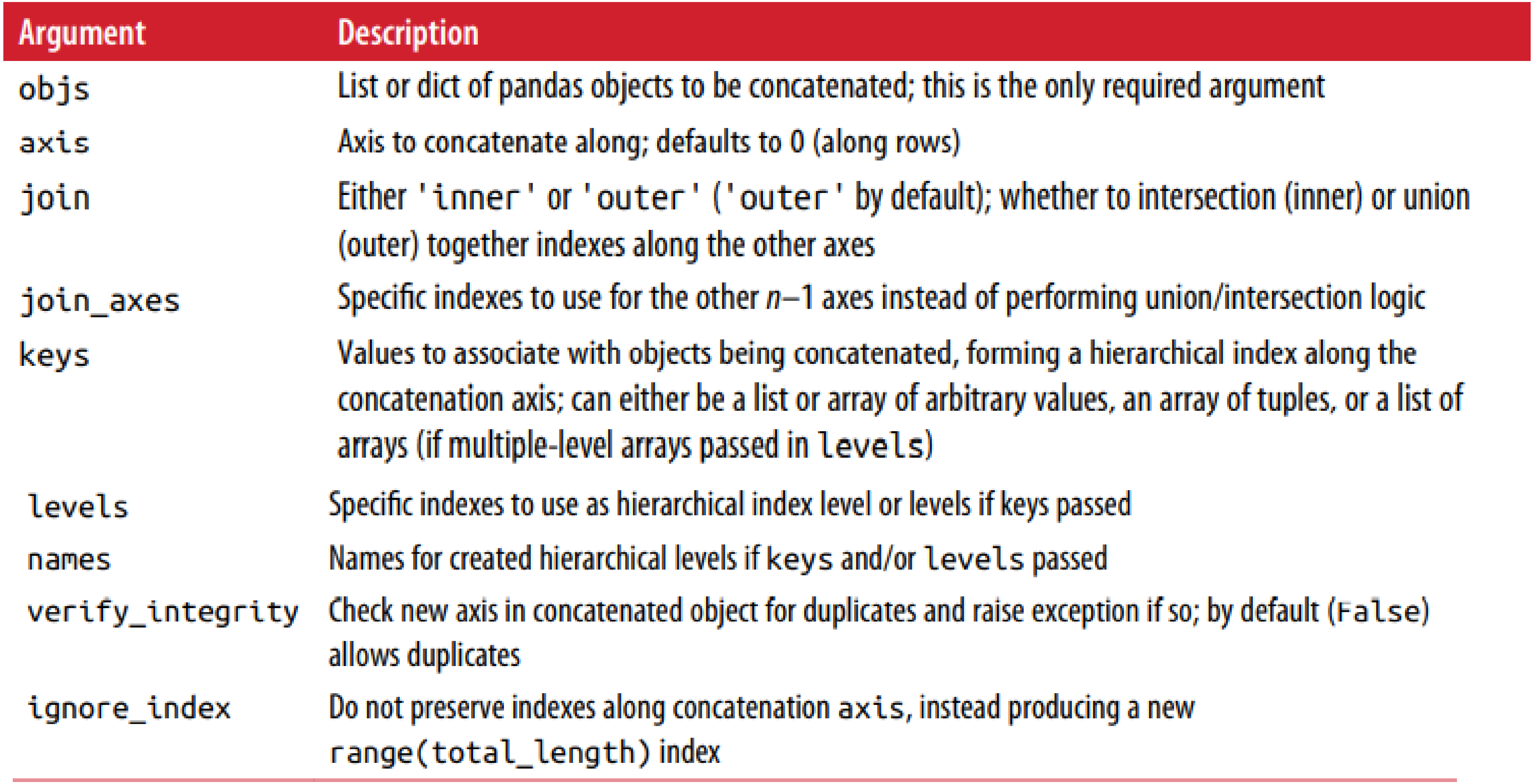
```
In [92]: result = pd.concat([s1, s1, s3], keys=['one', 'two', 'three'])
```

```
In [93]: result
Out[93]:
one    a    0
       b    1
two    a    0
       b    1
three  f    5
       g    6
dtype: int64
```

```
In [94]: result.unstack()
Out[94]:
         a    b    f    g
one    0.0  1.0  NaN  NaN
two    0.0  1.0  NaN  NaN
three  NaN  NaN  5.0  6.0
```

```
In [95]: pd.concat([s1, s2, s3], axis=1, keys=['one', 'two', 'three'])
Out[95]:
   one  two  three
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
```

Merging on Index



Argument	Description
`objs`	List or dict of pandas objects to be concatenated; this is the only required argument
`axis`	Axis to concatenate along; defaults to 0 (along rows)
`join`	Either `'inner'` or `'outer'` (`'outer'` by default); whether to intersection (inner) or union (outer) together indexes along the other axes
`join_axes`	Specific indexes to use for the other *n*–1 axes instead of performing union/intersection logic
`keys`	Values to associate with objects being concatenated, forming a hierarchical index along the concatenation axis; can either be a list or array of arbitrary values, an array of tuples, or a list of arrays (if multiple-level arrays passed in `levels`)
`levels`	Specific indexes to use as hierarchical index level or levels if keys passed
`names`	Names for created hierarchical levels if `keys` and/or `levels` passed
`verify_integrity`	Check new axis in concatenated object for duplicates and raise exception if so; by default (`False`) allows duplicates
`ignore_index`	Do not preserve indexes along concatenation `axis`, instead producing a new `range(total_length)` index

Reshaping and Pivoting

- Hierarchical indexing provides a consistent way to rearrange data in a DataFrame. There are two primary actions:

stack

This “rotates” or pivots from the columns in the data to the rows

unstack

This pivots from the rows into the columns