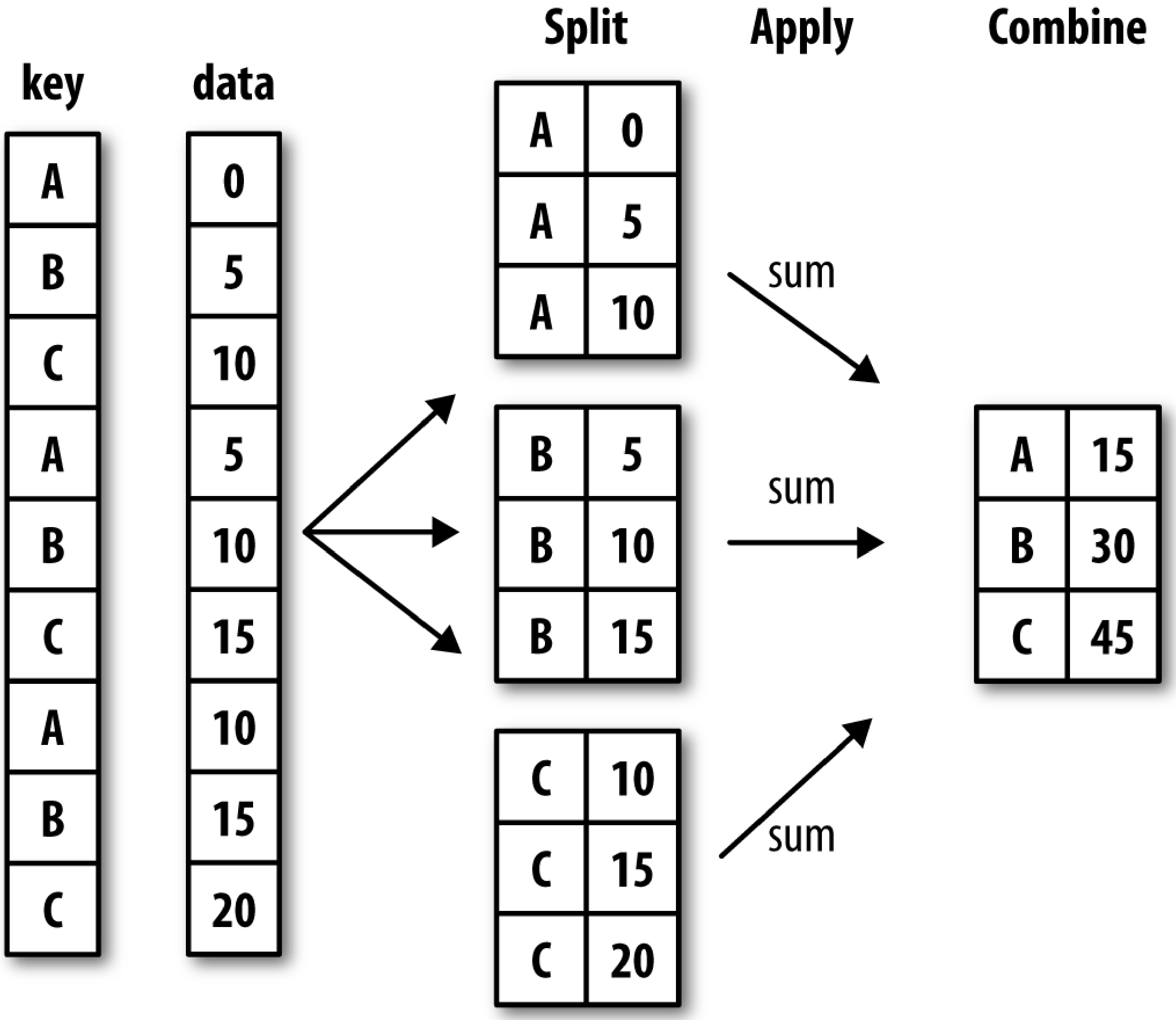
Data Aggregation and Group Operations

Part 1

GroupBy Mechanics

Part 1

- Hadley Wickham, an author of many popular packages for the R programming language, coined the term *split-apply-combine* for describing group operations.
- In the first stage of the process, data contained in a pandas object, whether a Series, DataFrame, or otherwise, is *split* into groups based on one or more *keys* that you provide.
- The splitting is performed on a particular axis of an object.
    - For example, a DataFrame can be grouped on its rows (`axis=0`) or its columns (`axis=1`).
- Once this is done, a function is *applied* to each group, producing a new value.
- Finally, the results of all those function applications are *combined* into a result object.



key	data
A	0
B	5
C	10
A	5
B	10
C	15
A	10
B	15
C	20

Split

A	0
A	5
A	10

B	5
B	10
B	15

C	10
C	15
C	20

Apply

sum

sum

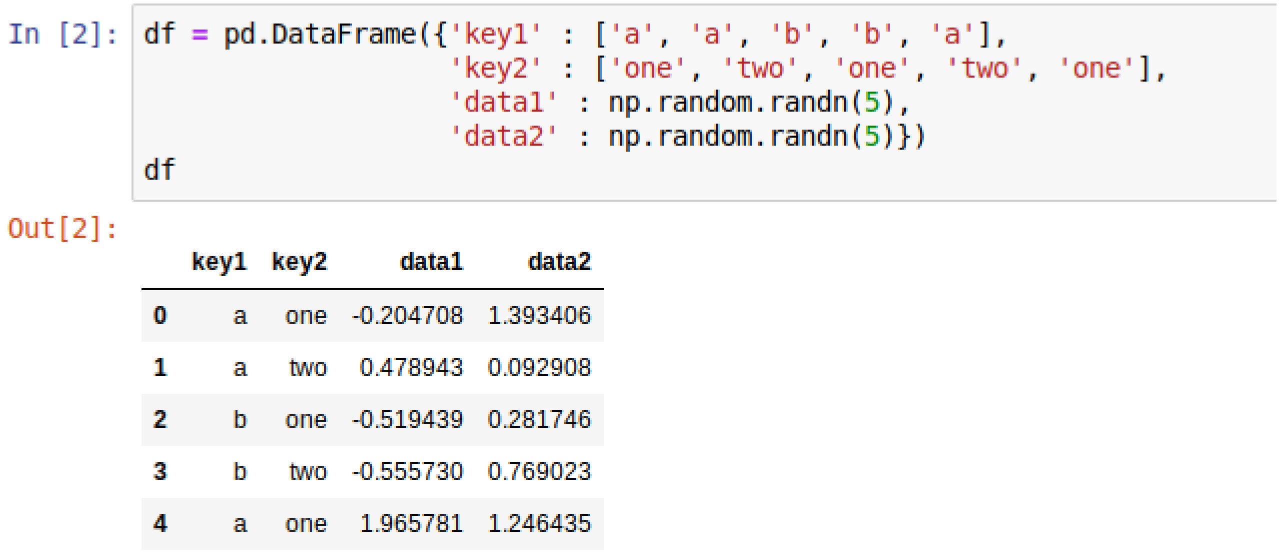
sum

Combine

A	15
B	30
C	45

- Each grouping key can take many forms, and the keys do not have to be all of the same type:
    - A list or array of values that is the same length as the axis being grouped
    - A value indicating a column name in a DataFrame
    - A dict or Series giving a correspondence between the values on the axis being grouped and the group names
    - A function to be invoked on the axis index or the individual labels in the index

- To get started, here is a small tabular dataset as a DataFrame:

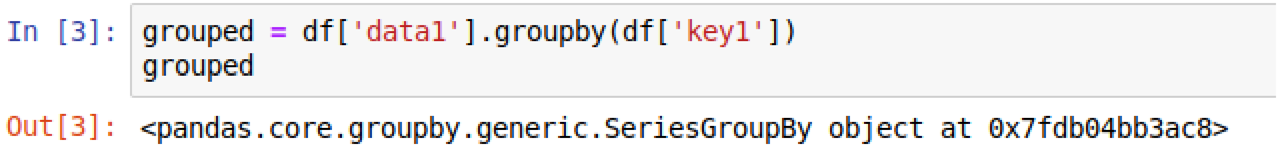


```
In [2]: df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                           'key2' : ['one', 'two', 'one', 'two', 'one'],
                           'data1' : np.random.randn(5),
                           'data2' : np.random.randn(5)})
        df
```

Out[2]:

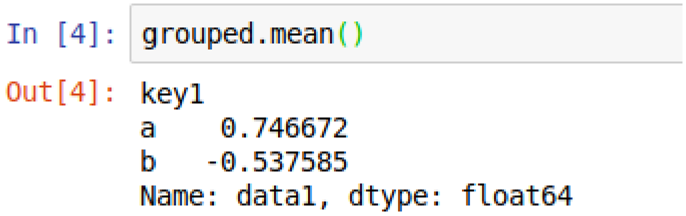
	key1	key2	data1	data2
0	a	one	-0.204708	1.393406
1	a	two	0.478943	0.092908
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023
4	a	one	1.965781	1.246435

- Suppose you wanted to compute the mean of the `data1` column using the labels from `key1`.
- There are a number of ways to do this.
- One is to access `data1` and call `groupby` with the column (a Series) at `key1`:



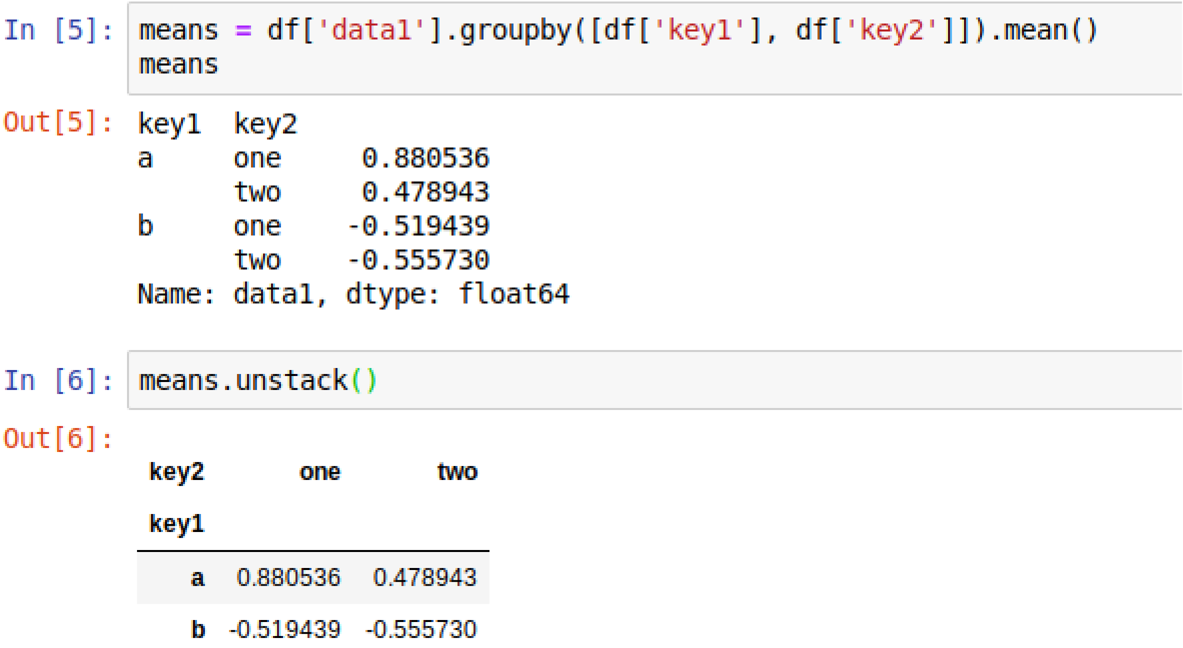
```
In [3]: grouped = df['data1'].groupby(df['key1'])
        grouped

Out[3]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x7fdb04bb3ac8>
```

- This `grouped` variable is now a *GroupBy* object.
- It has not actually computed anything yet except for some intermediate data about the group key `df['key1’]`.
- The idea is that this object has all of the information needed to then apply some operation to each of the groups.
- For example, to compute group means we can call the GroupBy’s `mean` method:



```
In [4]: grouped.mean()

Out[4]: key1
        a    0.746672
        b   -0.537585
        Name: data1, dtype: float64
```

- If instead we had passed multiple arrays as a list, we’d get something different:

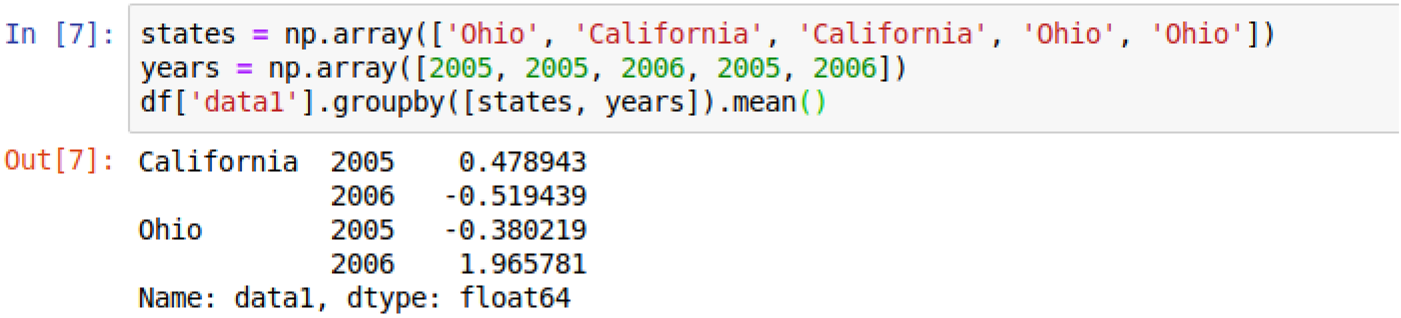


```
In [5]: means = df['data1'].groupby([df['key1'], df['key2']]).mean()
        means

Out[5]: key1  key2
        a     one      0.880536
              two      0.478943
        b     one     -0.519439
              two     -0.555730
        Name: data1, dtype: float64
```

```
In [6]: means.unstack()

Out[6]:
```

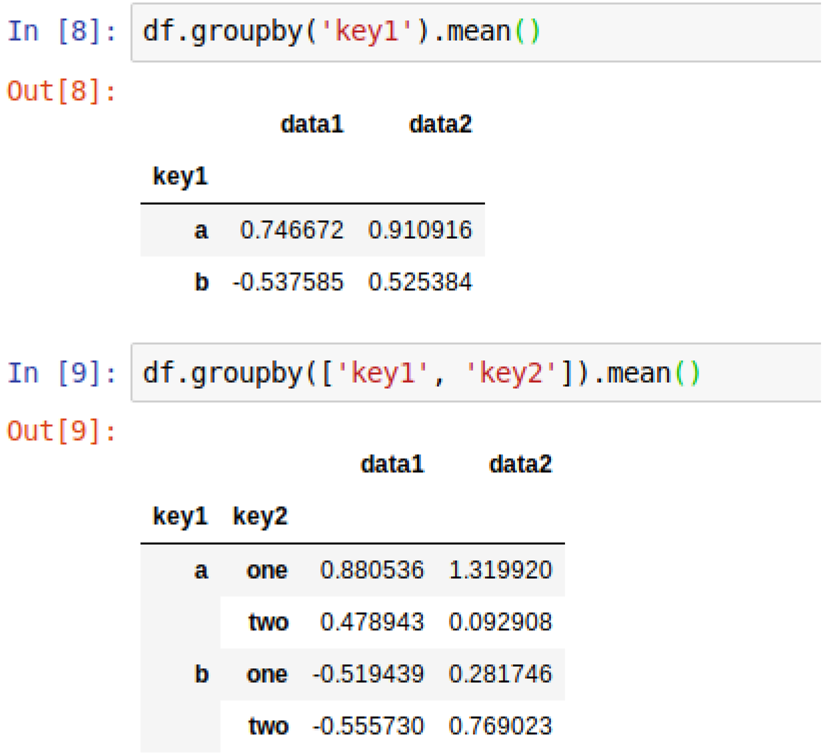
key2	one	two
key1		
a	0.880536	0.478943
b	-0.519439	-0.555730

- In this example, the group keys are all Series, though they could be any arrays of the right length:



```
In [7]: states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
        years = np.array([2005, 2005, 2006, 2005, 2006])
        df['data1'].groupby([states, years]).mean()

Out[7]: California  2005    0.478943
                    2006   -0.519439
        Ohio        2005   -0.380219
                    2006    1.965781
        Name: data1, dtype: float64
```

- Frequently the grouping information is found in the same DataFrame as the data you want to work on.
- In that case, you can pass column names (whether those are strings, numbers, or other Python objects) as the group keys:

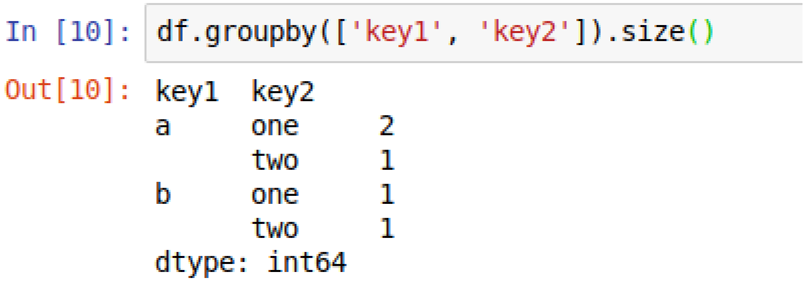


```
In [8]: df.groupby('key1').mean()
Out[8]:
            data1     data2
key1
   a     0.746672  0.910916
   b    -0.537585  0.525384
```

```
In [9]: df.groupby(['key1', 'key2']).mean()
Out[9]:
                  data1     data2
key1  key2
   a   one     0.880536  1.319920
       two     0.478943  0.092908
   b   one    -0.519439  0.281746
       two    -0.555730  0.769023
```

- You may have noticed in the first case `df.groupby('key1').mean()` that there is no `key2` column in the result.
- Because `df['key2']` is not numeric data, it is said to be a *nuisance* column, which is therefore excluded from the result.
- By default, all of the numeric columns are aggregated, though it is possible to filter down to a subset.

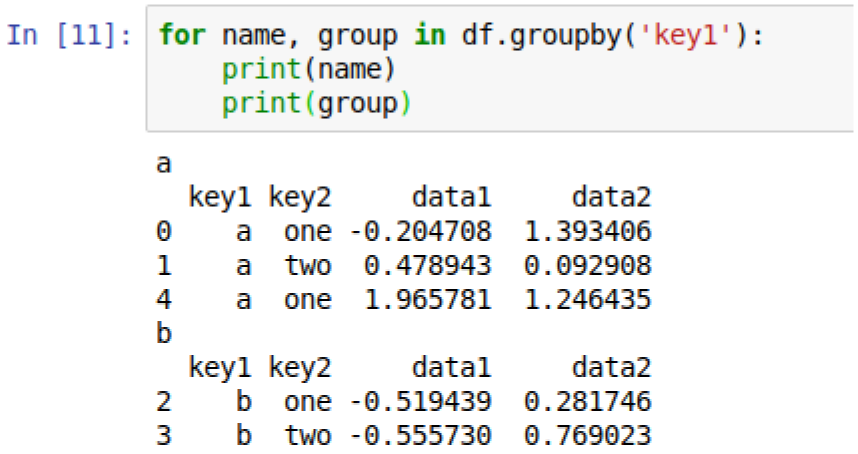
- Regardless of the objective in using `groupby`, a generally useful GroupBy method is `size`, which returns a Series containing group sizes:



```
In [10]: df.groupby(['key1', 'key2']).size()

Out[10]: key1  key2
         a     one     2
               two     1
         b     one     1
               two     1
         dtype: int64
```

Iterating Over Groups

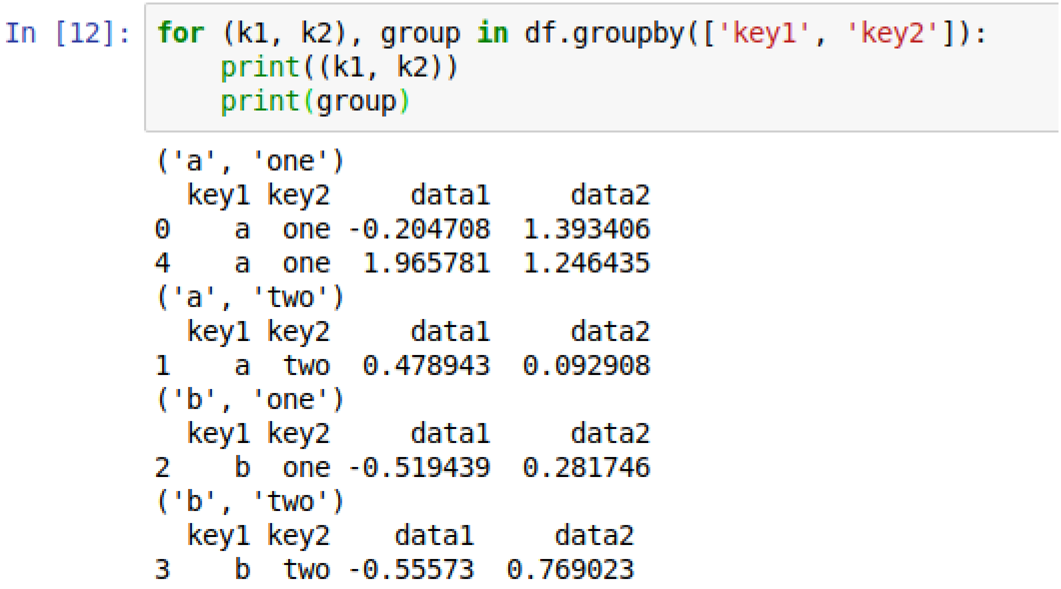
- The GroupBy object supports iteration, generating a sequence of 2-tuples containing the group name along with the chunk of data.



```
In [11]: for name, group in df.groupby('key1'):
             print(name)
             print(group)
```

```
a
  key1 key2     data1     data2
0    a  one -0.204708  1.393406
1    a  two  0.478943  0.092908
4    a  one  1.965781  1.246435
b
  key1 key2     data1     data2
2    b  one -0.519439  0.281746
3    b  two -0.555730  0.769023
```

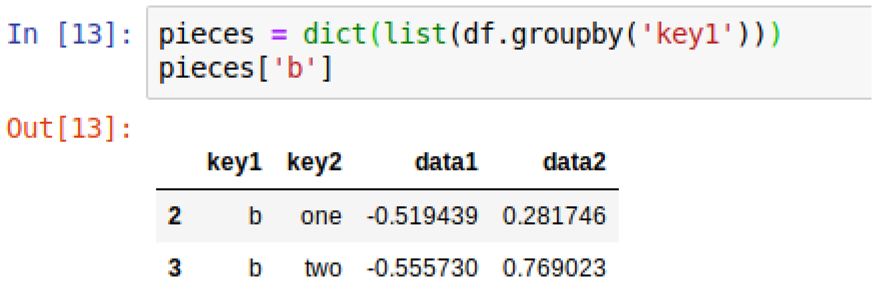
- In the case of multiple keys, the first element in the tuple will be a tuple of key values:



```
In [12]: for (k1, k2), group in df.groupby(['key1', 'key2']):
             print((k1, k2))
             print(group)
```

```
('a', 'one')
  key1 key2     data1     data2
0    a  one -0.204708  1.393406
4    a  one  1.965781  1.246435
('a', 'two')
  key1 key2     data1     data2
1    a  two  0.478943  0.092908
('b', 'one')
  key1 key2     data1     data2
2    b  one -0.519439  0.281746
('b', 'two')
  key1 key2    data1     data2
3    b  two -0.55573  0.769023
```

- Of course, you can choose to do whatever you want with the pieces of data.
- A recipe you may find useful is computing a dict of the data pieces as a one-liner:

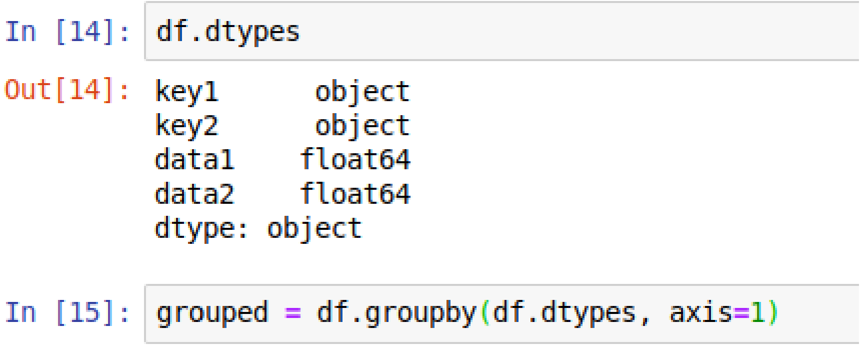


```
In [13]: pieces = dict(list(df.groupby('key1')))
         pieces['b']
```

Out[13]:

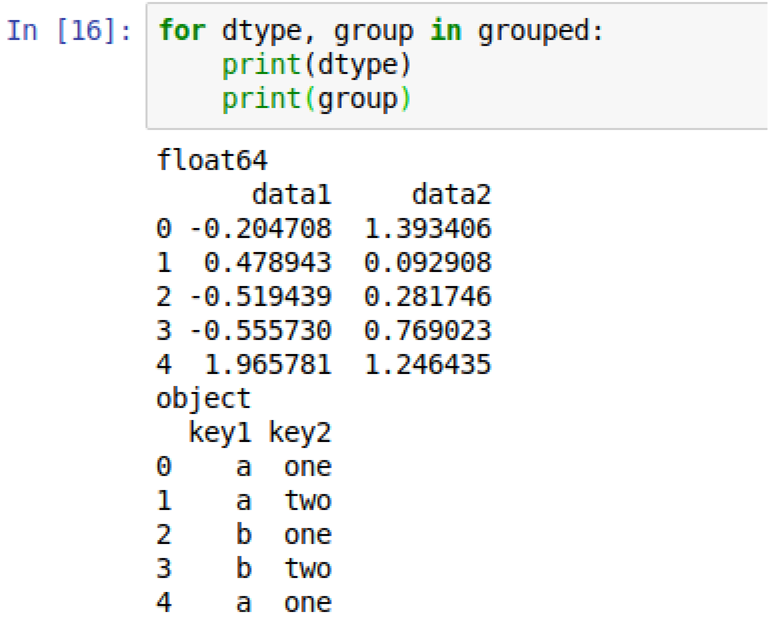
	key1	key2	data1	data2
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023

- By default `groupby` groups on `axis=0`, but you can group on any of the other axes.
- For example, we could group the columns of our example `df` here by `dtype` like so:



```
In [14]: df.dtypes

Out[14]: key1      object
         key2      object
         data1    float64
         data2    float64
         dtype: object

In [15]: grouped = df.groupby(df.dtypes, axis=1)
```

- We can print out the groups like so:

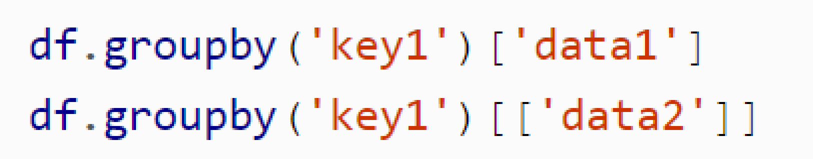


```
In [16]: for dtype, group in grouped:
             print(dtype)
             print(group)
```

```
float64
      data1     data2
0 -0.204708  1.393406
1  0.478943  0.092908
2 -0.519439  0.281746
3 -0.555730  0.769023
4  1.965781  1.246435
object
  key1 key2
0    a  one
1    a  two
2    b  one
3    b  two
4    a  one
```

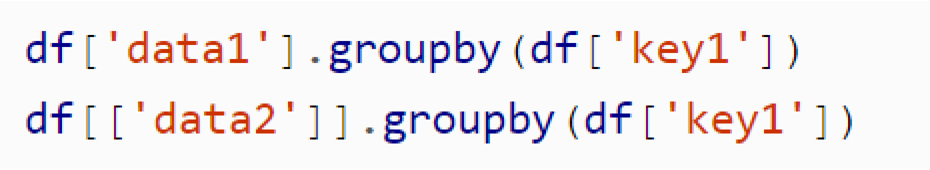
Selecting a Column or Subset of Columns

- Indexing a GroupBy object created from a DataFrame with a column name or array of column names has the effect of column subsetting for aggregation.
- This means that:



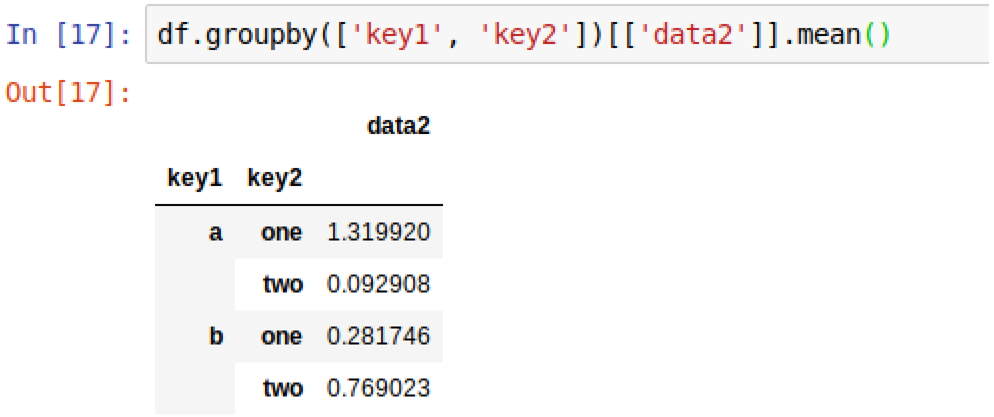
```
df.groupby('key1')['data1']
df.groupby('key1')[['data2']]
```

are syntactic sugar for:



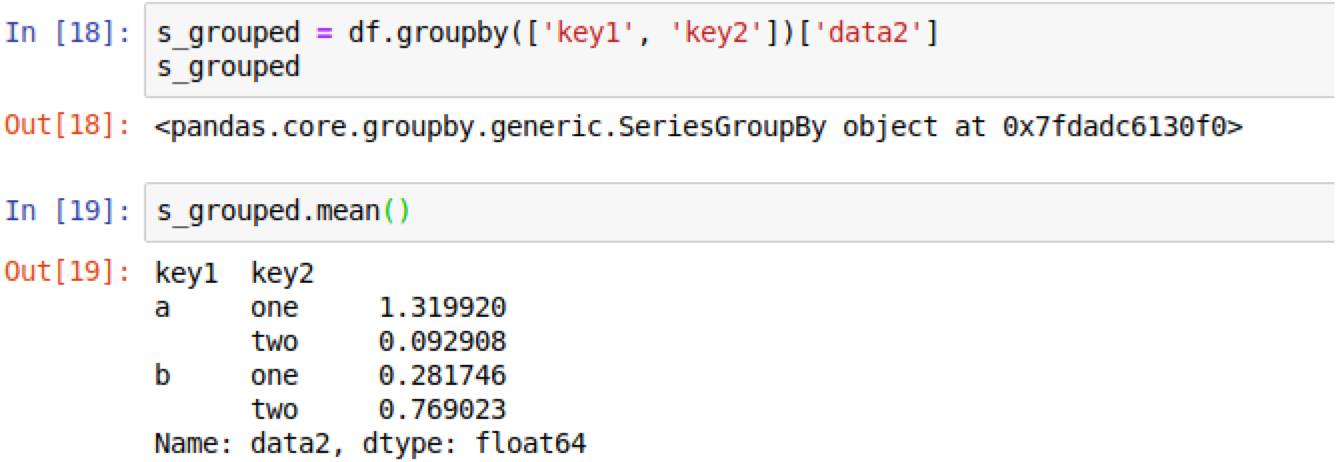
```
df['data1'].groupby(df['key1'])
df[['data2']].groupby(df['key1'])
```

- Especially for large datasets, it may be desirable to aggregate only a few columns.
- For example, in the preceding dataset, to compute means for just the `data2` column and get the result as a DataFrame, we could write:



```
In [17]: df.groupby(['key1', 'key2'])[['data2']].mean()
Out[17]:
                data2
key1  key2
   a   one   1.319920
       two   0.092908
   b   one   0.281746
       two   0.769023
```

- The object returned by this indexing operation is a grouped DataFrame if a list or array is passed or a grouped Series if only a single column name is passed as a scalar:



```
In [18]: s_grouped = df.groupby(['key1', 'key2'])['data2']
         s_grouped

Out[18]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x7fdadc6130f0>

In [19]: s_grouped.mean()

Out[19]: key1  key2
         a     one     1.319920
               two     0.092908
         b     one     0.281746
               two     0.769023
         Name: data2, dtype: float64
```