```
In [10]: 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)})
In [11]: df
```

	key1	key2	data1	data2
0	a	one	0.022212	0.555785
1	a	two	0.645846	2.091152
2	b	one	0.692301	0.252874
3	b	two	0.261721	-0.505822
4	a	one	-0.377385	-0.041419

```
import pandas as pd
import numpy as np
```

New Section

```
In [12]: grouped = df['data1'].groupby(df['key1'])
In [13]: grouped

     <pandas.core.groupby.generic.SeriesGroupBy object at 0x7f9931d1b6d8>


In [14]: grouped.mean()

     key1
     a   -0.376656
     b    0.617990
     Name: data1, dtype: float64


In [15]: means = df['data2'].groupby([df['key1'], df['key2']]).mean()
In [16]: means

     key1  key2
     a     one     0.858315
           two     0.475349
     b     one    -0.149166
           two     0.945977
     Name: data2, dtype: float64


In [17]: means.unstack()
```

key1	key2	one	two
a		0.858315	0.475349
b		-0.149166	0.945977

```
In [18]: states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
In [19]: years = np.array([2005, 2005, 2006, 2005, 2006])
In [20]: df['data1'].groupby([states]).mean()
```

```
California    0.178570
Ohio         -0.083709
Name: data1, dtype: float64
```

```
In [21]: df.groupby('key1').mean()
```

key1	data1	data2
a	-0.376656	0.730660
b	0.617990	0.398405

```
In [23]: df.groupby(['key1', 'key2']).size()
```

```
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

Iterating Over Groups

```
In [24]: for name, group in df.groupby('key1'):
             print(name)
             print(group)
```

```
a
  key1 key2     data1     data2
0    a  one -0.968738  0.993311
1    a  two  0.016645  0.475349
4    a  one -0.177876  0.723319
b
  key1 key2     data1     data2
```

```
2    b  one  0.340494 -0.149166
3    b  two  0.895487  0.945977
```

```
In [25]: for (k1, k2), group in df.groupby(['key1', 'key2']):
             print((k1, k2))
             print(group)
```

```
('a', 'one')
  key1 key2     data1     data2
0    a  one -0.968738  0.993311
4    a  one -0.177876  0.723319
('a', 'two')
  key1 key2     data1     data2
1    a  two  0.016645  0.475349
('b', 'one')
  key1 key2     data1     data2
2    b  one  0.340494 -0.149166
('b', 'two')
  key1 key2     data1     data2
3    b  two  0.895487  0.945977
```

```
In [26]: pieces = dict(list(df.groupby('key1')))
In [27]: pieces['b']
```

	key1	key2	data1	data2
2	b	one	0.340494	-0.149166
3	b	two	0.895487	0.945977

```
In [28]: df.dtypes
```

```
key1      object
key2      object
data1    float64
data2    float64
dtype: object
```

```
In [29]: grouped = df.groupby(df.dtypes, axis=1)
grouped
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f9931b78b00>
```

```
In [30]: for dtype, group in grouped:
....:      print(dtype)
....:      print(group)
....:
```

```
float64
      data1     data2
0 -0.968738  0.993311
1  0.016645  0.475349
```

```
2  0.340494 -0.149166
3  0.895487  0.945977
4 -0.177876  0.723319
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

```
In [35]: people = pd.DataFrame(np.random.randn(5, 5),
....:    columns=['a', 'b', 'c', 'd', 'e'],
....:    index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
In [36]: people.iloc[2:3, [1, 2]] = np.nan # Add a few NA values
In [37]: people
```

	a	b	c	d	e
Joe	0.777215	-1.026809	0.424430	-0.754204	-0.827794
Steve	1.730370	-0.313633	0.792021	-0.304843	-0.394003
Wes	-0.286294	NaN	NaN	1.512097	0.538304
Jim	-0.349600	-0.216453	-0.821539	-0.148407	-0.691319
Travis	-2.857388	0.980405	0.683356	0.743843	0.779444

```
In [38]: mapping = {'a': 'red', 'b': 'red', 'c': 'blue', 'd': 'blue', 'e': 'red', 'f' : 'oran
```

```
In [39]: by_column = people.groupby(mapping, axis=1)
In [40]: by_column.sum()
```

	blue	red
Joe	-0.329773	-1.077387
Steve	0.487178	1.022734
Wes	1.512097	0.252010
Jim	-0.969946	-1.257372
Travis	1.427198	-1.097540

```
In [41]: map_series = pd.Series(mapping)
In [42]: map_series
```

```
a       red
b       red
c      blue
d      blue
e       red
f    orange
dtype: object
```

```
In [43]: people.groupby(map_series, axis=1).count()
```

	blue	red
Joe	2	3
Steve	2	3
Wes	1	2
Jim	2	3
Travis	2	3

Grouping with Functions

```
In [44]: people.groupby(len).sum()
```

	a	b	c	d	e
3	0.141322	-1.243262	-0.397109	0.609486	-0.980809
5	1.730370	-0.313633	0.792021	-0.304843	-0.394003
6	-2.857388	0.980405	0.683356	0.743843	0.779444

```
In [45]: key_list = ['one', 'one', 'one', 'two', 'two']
In [46]: people.groupby([len, key_list]).min()
```

		a	b	c	d	e
3	**one**	-0.286294	-1.026809	0.424430	-0.754204	-0.827794
	two	-0.349600	-0.216453	-0.821539	-0.148407	-0.691319
5	**one**	1.730370	-0.313633	0.792021	-0.304843	-0.394003
6	**two**	-2.857388	0.980405	0.683356	0.743843	0.779444

Grouping by Index Levels

```
In [47]: columns = pd.MultiIndex.from_arrays([['US', 'US', 'US', 'JP', 'JP'], [1, 3, 5, 1, 3]
```

```
In [47]: columns = pd.MultiIndex.from_arrays([['US', 'US', 'US', 'JP', 'JP'], [1, 3, 5, 1, 3]
In [48]: hier_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)
In [49]: hier_df
```

cty	US			JP	
tenor	1	3	5	1	3
0	1.479513	-1.316179	0.038301	1.372888	1.162746
1	-0.255851	-0.076422	0.157403	0.752023	0.020871
2	-0.217075	-0.373315	-0.225237	1.736880	-0.430735
3	-0.485730	0.926083	-0.748173	0.593308	-0.186148

```
In [50]: hier_df.groupby(level='cty', axis=1).count()
```

cty	JP	US
0	2	3
1	2	3
2	2	3
3	2	3

Data Aggregation

```
In [51]: df
```

	key1	key2	data1	data2
0	a	one	0.022212	0.555785
1	a	two	0.645846	2.091152
2	b	one	0.692301	0.252874
3	b	two	0.261721	-0.505822
4	a	one	-0.377385	-0.041419

```
In [52]: grouped = df.groupby('key1')
In [53]: grouped['data1'].quantile(0.9)
```

```
key1
a    0.521120
b    0.649243
Name: data1, dtype: float64
```

```
In [54]: def peak_to_peak(arr):
....:   return arr.max() - arr.min()
In [55]: grouped.agg(peak_to_peak)
```

key1	data1	data2
a	1.023231	2.132571
b	0.430580	0.758697

Column-Wise and Multiple Function Application

```
In [57]: tips = pd.read_csv('tips.csv')
# Add tip percentage of total bill
In [58]: tips['tip_pct'] = tips['tip'] / tips['total_bill']
In [59]: tips[:6]
```

	Unnamed: 0	total_bill	tip	sex	smoker	day	time	size	tip_pct
0	1	16.99	1.01	Female	No	Sun	Dinner	2	0.059447
1	2	10.34	1.66	Male	No	Sun	Dinner	3	0.160542
2	3	21.01	3.50	Male	No	Sun	Dinner	3	0.166587
3	4	23.68	3.31	Male	No	Sun	Dinner	2	0.139780
4	5	24.59	3.61	Female	No	Sun	Dinner	4	0.146808
5	6	25.29	4.71	Male	No	Sun	Dinner	4	0.186240

```
In [60]: grouped = tips.groupby(['day', 'smoker'])
```

```
In [61]: grouped_pct = grouped['tip_pct']
In [62]: grouped_pct.agg('mean')
```

```
day   smoker
Fri   No        0.151650
      Yes       0.174783
Sat   No        0.158048
      Yes       0.147906
Sun   No        0.160113
      Yes       0.187250
Thur  No        0.160298
      Yes       0.163863
Name: tip_pct, dtype: float64
```

```
In [63]: grouped_pct.agg(['mean', 'std', peak_to_peak])
```

		mean	std	peak_to_peak
day	smoker			
Fri	No	0.151650	0.028123	0.067349
	Yes	0.174783	0.051293	0.159925
Sat	No	0.158048	0.039767	0.235193
	Yes	0.147906	0.061375	0.290095
Sun	No	0.160113	0.042347	0.193226
	Yes	0.187250	0.154134	0.644685
Thur	No	0.160298	0.038774	0.193350
	Yes	0.163863	0.039389	0.151240

```
In [64]: grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
```

		foo	bar
day	smoker		
Fri	No	0.151650	0.028123
	Yes	0.174783	0.051293
Sat	No	0.158048	0.039767
	Yes	0.147906	0.061375
Sun	No	0.160113	0.042347
	Yes	0.187250	0.154134
Thur	No	0.160298	0.038774
	Yes	0.163863	0.039389

```
In [65]: functions = ['count', 'mean', 'max']
In [66]: result = grouped['tip_pct', 'total_bill'].agg(functions)
In [67]: result
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: FutureWarning: Indexing
```

		tip_pct			total_bill		
		count	mean	max	count	mean	max
day	smoker						
Fri	No	4	0.151650	0.187735	4	18.420000	22.75
	Yes	15	0.174783	0.263480	15	16.813333	40.17
Sat	No	45	0.158048	0.291990	45	19.661778	48.33

```
In [68]: result['tip_pct']
```

		count	mean	max
day	smoker			
Fri	No	4	0.151650	0.187735
	Yes	15	0.174783	0.263480
Sat	No	45	0.158048	0.291990
	Yes	42	0.147906	0.325733
Sun	No	57	0.160113	0.252672
	Yes	19	0.187250	0.710345
Thur	No	45	0.160298	0.266312
	Yes	17	0.163863	0.241255

Returning Aggregated Data Without Row Indexes

```
In [73]: tips.groupby(['day', 'smoker'], as_index=False).mean()
```

	day	smoker	Unnamed: 0	total_bill	tip	size	tip_pct
1	Fri	Yes	147.666667	16.813333	2.714000	2.066667	0.174783
4	Sun	No	77.052632	20.506667	3.167895	2.929825	0.160113

Example: Filling Missing Values with Group-Specific Values

```
In [91]: s = pd.Series(np.random.randn(6))
In [92]: s[::2] = np.nan
In [93]: s
```

```
s.fillna(s.mean())
```

```
In [95]: states = ['Ohio', 'New York', 'Vermont', 'Florida',
....: 'Oregon', 'Nevada', 'California', 'Idaho']
In [96]: group_key = ['East'] * 4 + ['West'] * 4
In [97]: data = pd.Series(np.random.randn(8), index=states)
In [98]: data
```

```
In [99]: data[['Vermont', 'Nevada', 'Idaho']] = np.nan
In [100]: data
```

```
In [101]: data.groupby(group_key).mean()
```

```
In [102]: fill_mean = lambda g: g.fillna(g.mean())
In [103]: data.groupby(group_key).apply(fill_mean)
```