Data Aggregation and Group Operations

In [2]:

```
import numpy as np
import pandas as pd
PREVIOUS_MAX_ROWS = pd.options.display.max_rows
pd.options.display.max_rows = 20
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
np.set_printoptions(precision=4, suppress=True)
```

GroupBy Mechanics

In [3]:

```
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[3]:

	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

In [4]:

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

Out[4]:

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x0000021DC3683CA0>
```

In [5]:

```
grouped.mean()
```

Out[5]:

```
key1
a    0.746672
b   -0.537585
Name: data1, dtype: float64
```

In [6]:

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

Out[6]:

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

In [7]:

```
means.unstack()
```

Out[7]:

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

In [8]:

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

Out[8]:

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

In [9]:

```
df.groupby('key1').mean()
```

Out[9]:

	data1	data2
key1		
a	0.746672	0.910916
b	-0.537585	0.525384

In [11]:

```
df.groupby(['key1', 'key2']).mean()
```

Out[11]:

		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

In [12]:

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

Out[12]:

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

Iterating Over Groups

In [13]:

```
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 [14]:

```python
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
```

In [15]:

```python
pieces = dict(list(df.groupby('key1')))
pieces['b']
```

Out[15]:

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

In [16]:

```python
df.dtypes
grouped = df.groupby(df.dtypes, axis=1)
```

In [17]:

```python
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

In [18]:

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

Out[18]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000021DC365DCD0>
```

In [19]:

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

Out[19]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000021DC05E9940>
```

In [20]:

```
df.groupby(['key1', 'key2'])[['data2']].mean()
```

Out[20]:

		data2
key1	**key2**	
a	**one**	1.319920
	two	0.092908
b	**one**	0.281746
	two	0.769023

In [21]:

```
s_grouped = df.groupby(['key1', 'key2'])['data2']
s_grouped
s_grouped.mean()
```

Out[21]:

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

Grouping with Dicts and Series

In [22]:

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

Out[22]:

	a	**b**	**c**	**d**	**e**
Joe	1.007189	-1.296221	0.274992	0.228913	1.352917
Steve	0.886429	-2.001637	-0.371843	1.669025	-0.438570
Wes	-0.539741	NaN	NaN	-1.021228	-0.577087
Jim	0.124121	0.302614	0.523772	0.000940	1.343810
Travis	-0.713544	-0.831154	-2.370232	-1.860761	-0.860757

In [23]:

```
mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
           'd': 'blue', 'e': 'red', 'f' : 'orange'}
```

In [24]:

```
by_column = people.groupby(mapping, axis=1)
by_column.sum()
```

Out[24]:

	blue	red
Joe	0.503905	1.063885
Steve	1.297183	-1.553778
Wes	-1.021228	-1.116829
Jim	0.524712	1.770545
Travis	-4.230992	-2.405455

In [25]:

```
map_series = pd.Series(mapping)
map_series
```

Out[25]:

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

In [26]:

```
people.groupby(map_series, axis=1).count()
```

Out[26]:

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

In [28]:

```
people.groupby(map_series, axis=1).describe()
```

Out[28]:

	count	mean	std	min	25%	50%	75%	max
a	5.0	0.152891	0.790440	-0.713544	-0.539741	0.124121	0.886429	1.007189
b	4.0	-0.956600	0.967609	-2.001637	-1.472575	-1.063687	-0.547712	0.302614
c	4.0	-0.485828	1.311756	-2.370232	-0.871440	-0.048425	0.337187	0.523772
d	5.0	-0.196622	1.336982	-1.860761	-1.021228	0.000940	0.228913	1.669025
e	5.0	0.164062	1.091776	-0.860757	-0.577087	-0.438570	1.343810	1.352917

Grouping with Functions

In [29]:

```
people.groupby(len).sum()
```

Out[29]:

	a	b	c	d	e
3	0.591569	-0.993608	0.798764	-0.791374	2.119639
5	0.886429	-2.001637	-0.371843	1.669025	-0.438570
6	-0.713544	-0.831154	-2.370232	-1.860761	-0.860757

In [30]:

```python
people.groupby(len, axis = 1).sum()
```

Out[30]:

	1
Joe	1.567790
Steve	-0.256595
Wes	-2.138056
Jim	2.295257
Travis	-6.636447

In [31]:

```python
key_list = ['one', 'one', 'one', 'two', 'two']
people.groupby([len, key_list]).min()
```

Out[31]:

		a	b	c	d	e
3	**one**	-0.539741	-1.296221	0.274992	-1.021228	-0.577087
	two	0.124121	0.302614	0.523772	0.000940	1.343810
5	**one**	0.886429	-2.001637	-0.371843	1.669025	-0.438570
6	**two**	-0.713544	-0.831154	-2.370232	-1.860761	-0.860757

Grouping by Index Levels

In [32]:

```python
columns = pd.MultiIndex.from_arrays([['US', 'US', 'US', 'JP', 'JP'],
                                    [1, 3, 5, 1, 3]],
                                    names=['cty', 'tenor'])
hier_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)
hier_df
```

Out[32]:

cty			US		JP
tenor	**1**	**3**	**5**	**1**	**3**
0	0.560145	-1.265934	0.119827	-1.063512	0.332883
1	-2.359419	-0.199543	-1.541996	-0.970736	-1.307030
2	0.286350	0.377984	-0.753887	0.331286	1.349742
3	0.069877	0.246674	-0.011862	1.004812	1.327195

In [33]:

```python
hier_df.groupby(level='cty', axis=1).count()
```

Out[33]:

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

Data Aggregation

In [34]:
```
df
```

Out[34]:

	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

In [35]:
```
grouped = df.groupby('key1')
grouped['data1'].quantile(0.9)
```

Out[35]:
```
key1
a    1.668413
b   -0.523068
Name: data1, dtype: float64
```

In [36]:
```
def peak_to_peak(arr):
    return arr.max() - arr.min()
grouped.agg(peak_to_peak)
```

```
C:\Users\Usuário\AppData\Local\Temp\ipykernel_6776\663192242.py:3: FutureWarning:
['key2'] did not aggregate successfully. If any error is raised this will raise in
a future version of pandas. Drop these columns/ops to avoid this warning.
  grouped.agg(peak_to_peak)
```

Out[36]:

	data1	data2
key1		
a	2.170488	1.300498
b	0.036292	0.487276

In [37]:
```
grouped.describe()
```

Out[37]:

								data1	
	count	mean	std	min	25%	50%	75%	max	count
key1									
a	3.0	0.746672	1.109736	-0.204708	0.137118	0.478943	1.222362	1.965781	3.0 0.9
b	2.0	-0.537585	0.025662	-0.555730	-0.546657	-0.537585	-0.528512	-0.519439	2.0 0.5

Column-Wise and Multiple Function Application

In [38]:
```
tips = pd.read_csv('examples/tips.csv')
tips
```

Out[38]:

	total_bill	tip	smoker	day	time	size
0	16.99	1.01	No	Sun	Dinner	2
1	10.34	1.66	No	Sun	Dinner	3
2	21.01	3.50	No	Sun	Dinner	3
3	23.68	3.31	No	Sun	Dinner	2
4	24.59	3.61	No	Sun	Dinner	4
...	...	...	...	...	...	...
239	29.03	5.92	No	Sat	Dinner	3
240	27.18	2.00	Yes	Sat	Dinner	2
241	22.67	2.00	Yes	Sat	Dinner	2
242	17.82	1.75	No	Sat	Dinner	2
243	18.78	3.00	No	Thur	Dinner	2

244 rows × 6 columns

In [39]:

```
# Add tip percentage of total bill
tips['tip_pct'] = tips['tip'] / tips['total_bill']
tips[:6]
```

Out[39]:

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

In [40]:

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

In [41]:

```
grouped_pct = grouped['tip_pct']
grouped_pct.agg('mean')
```

Out[41]:

```
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 [42]:

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

Out[42]:

		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 [43]:

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

Out[43]:

		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 [44]:

```
functions = ['count', 'mean', 'max']
result = grouped['tip_pct', 'total_bill'].agg(functions)
result
```

```
C:\Users\Usuário\AppData\Local\Temp\ipykernel_6776\576261660.py:2: FutureWarning:
Indexing with multiple keys (implicitly converted to a tuple of keys) will be depr
ecated, use a list instead.
  result = grouped['tip_pct', 'total_bill'].agg(functions)
```

Out[44]:

		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
	Yes	42	0.147906	0.325733	42	21.276667	50.81
Sun	No	57	0.160113	0.252672	57	20.506667	48.17
	Yes	19	0.187250	0.710345	19	24.120000	45.35
Thur	No	45	0.160298	0.266312	45	17.113111	41.19
	Yes	17	0.163863	0.241255	17	19.190588	43.11

In [45]:

```
result['tip_pct']
```

Out[45]:

		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

In [46]:

```
ftuples = [('Durchschnitt', 'mean'), ('Abweichung', np.var)]
grouped['tip_pct', 'total_bill'].agg(ftuples)
```

```
C:\Users\Usuário\AppData\Local\Temp\ipykernel_6776\3909980601.py:2: FutureWarning:
Indexing with multiple keys (implicitly converted to a tuple of keys) will be depr
ecated, use a list instead.
  grouped['tip_pct', 'total_bill'].agg(ftuples)
```

Out[46]:

		tip_pct		total_bill	
		Durchschnitt	Abweichung	Durchschnitt	Abweichung
day	smoker				
Fri	No	0.151650	0.000791	18.420000	25.596333
	Yes	0.174783	0.002631	16.813333	82.562438
Sat	No	0.158048	0.001581	19.661778	79.908965
	Yes	0.147906	0.003767	21.276667	101.387535
Sun	No	0.160113	0.001793	20.506667	66.099980
	Yes	0.187250	0.023757	24.120000	109.046044
Thur	No	0.160298	0.001503	17.113111	59.625081
	Yes	0.163863	0.001551	19.190588	69.808518

In [47]:

```
grouped.agg({'tip' : np.max, 'size' : 'sum'})
```

Out[47]:

		tip	size
day	smoker		
Fri	No	3.50	9
	Yes	4.73	31
Sat	No	9.00	115
	Yes	10.00	104
Sun	No	6.00	167
	Yes	6.50	49
Thur	No	6.70	112
	Yes	5.00	40

In [48]:

```
grouped.agg({'tip_pct' : ['min', 'max', 'mean', 'std'],
             'size' : 'sum'})
```

Out[48]:

		tip_pct				size
		min	max	mean	std	sum
day	smoker					
Fri	No	0.120385	0.187735	0.151650	0.028123	9
	Yes	0.103555	0.263480	0.174783	0.051293	31
Sat	No	0.056797	0.291990	0.158048	0.039767	115
	Yes	0.035638	0.325733	0.147906	0.061375	104
Sun	No	0.059447	0.252672	0.160113	0.042347	167
	Yes	0.065660	0.710345	0.187250	0.154134	49
Thur	No	0.072961	0.266312	0.160298	0.038774	112
	Yes	0.090014	0.241255	0.163863	0.039389	40

Returning Aggregated Data Without Row Indexes

In [49]:

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

Out[49]:

	day	smoker	total_bill	tip	size	tip_pct
0	Fri	No	18.420000	2.812500	2.250000	0.151650
1	Fri	Yes	16.813333	2.714000	2.066667	0.174783
2	Sat	No	19.661778	3.102889	2.555556	0.158048
3	Sat	Yes	21.276667	2.875476	2.476190	0.147906
4	Sun	No	20.506667	3.167895	2.929825	0.160113
5	Sun	Yes	24.120000	3.516842	2.578947	0.187250
6	Thur	No	17.113111	2.673778	2.488889	0.160298
7	Thur	Yes	19.190588	3.030000	2.352941	0.163863

Apply: General split-apply-combine

In [50]:

```
def top(df, n=5, column='tip_pct'):
    return df.sort_values(by=column)[-n:]
top(tips, n=6)
```

Out[50]:

	total_bill	tip	smoker	day	time	size	tip_pct
109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
232	11.61	3.39	No	Sat	Dinner	2	0.291990
67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

In [51]:

```
tips.groupby('smoker').apply(top)
```

Out[51]:

smoker		total_bill	tip	smoker	day	time	size	tip_pct
No	**88**	24.71	5.85	No	Thur	Lunch	2	0.236746
	185	20.69	5.00	No	Sun	Dinner	5	0.241663
	51	10.29	2.60	No	Sun	Dinner	2	0.252672
	149	7.51	2.00	No	Thur	Lunch	2	0.266312
	232	11.61	3.39	No	Sat	Dinner	2	0.291990
Yes	**109**	14.31	4.00	Yes	Sat	Dinner	2	0.279525
	183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
	67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
	178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
	172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

In [52]:

```
tips.groupby(['smoker', 'day']).apply(top, n=1, column='total_bill')
```

Out[52]:

smoker	day		total_bill	tip	smoker	day	time	size	tip_pct
No	**Fri**	**94**	22.75	3.25	No	Fri	Dinner	2	0.142857
	Sat	**212**	48.33	9.00	No	Sat	Dinner	4	0.186220
	Sun	**156**	48.17	5.00	No	Sun	Dinner	6	0.103799
	Thur	**142**	41.19	5.00	No	Thur	Lunch	5	0.121389
Yes	**Fri**	**95**	40.17	4.73	Yes	Fri	Dinner	4	0.117750
	Sat	**170**	50.81	10.00	Yes	Sat	Dinner	3	0.196812
	Sun	**182**	45.35	3.50	Yes	Sun	Dinner	3	0.077178
	Thur	**197**	43.11	5.00	Yes	Thur	Lunch	4	0.115982

In [53]:

```
result = tips.groupby('smoker')['tip_pct'].describe()
result
```

Out[53]:

smoker	count	mean	std	min	25%	50%	75%	max
No	151.0	0.159328	0.039910	0.056797	0.136906	0.155625	0.185014	0.291990
Yes	93.0	0.163196	0.085119	0.035638	0.106771	0.153846	0.195059	0.710345

In [54]:

```
result.unstack('smoker')
```

Out[54]:

```
       smoker
count  No        151.000000
       Yes        93.000000
mean   No          0.159328
       Yes         0.163196
std    No          0.039910
       Yes         0.085119
min    No          0.056797
       Yes         0.035638
25%    No          0.136906
       Yes         0.106771
50%    No          0.155625
       Yes         0.153846
75%    No          0.185014
       Yes         0.195059
max    No          0.291990
       Yes         0.710345
dtype: float64
```

In [55]:

```
f = lambda x: x.describe()
grouped.apply(f)
```

Out[55]:

			total_bill	**tip**	**size**	**tip_pct**
day	**smoker**					
Fri	**No**	**count**	4.000000	4.000000	4.00	4.000000
		mean	18.420000	2.812500	2.25	0.151650
		std	5.059282	0.898494	0.50	0.028123
		min	12.460000	1.500000	2.00	0.120385
		25%	15.100000	2.625000	2.00	0.137239
...	**...**	**...**	...	...	...	...
Thur	**Yes**	**min**	10.340000	2.000000	2.00	0.090014
		25%	13.510000	2.000000	2.00	0.148038
		50%	16.470000	2.560000	2.00	0.153846
		75%	19.810000	4.000000	2.00	0.194837
		max	43.110000	5.000000	4.00	0.241255

64 rows × 4 columns

Suppressing the Group Keys

In [56]:

```
tips.groupby('smoker', group_keys=False).apply(top)
```

Out[56]:

	total_bill	tip	smoker	day	time	size	tip_pct
88	24.71	5.85	No	Thur	Lunch	2	0.236746
185	20.69	5.00	No	Sun	Dinner	5	0.241663
51	10.29	2.60	No	Sun	Dinner	2	0.252672
149	7.51	2.00	No	Thur	Lunch	2	0.266312
232	11.61	3.39	No	Sat	Dinner	2	0.291990
109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

Quantile and Bucket Analysis

In [57]:

```
frame = pd.DataFrame({'data1': np.random.randn(1000),
                      'data2': np.random.randn(1000)})
quartiles = pd.cut(frame.data1, 4)
quartiles[:10]
```

Out[57]:

```
0     (-1.23, 0.489]
1    (-2.956, -1.23]
2     (-1.23, 0.489]
3     (0.489, 2.208]
4     (-1.23, 0.489]
5     (0.489, 2.208]
6     (-1.23, 0.489]
7     (-1.23, 0.489]
8     (0.489, 2.208]
9     (0.489, 2.208]
Name: data1, dtype: category
Categories (4, interval[float64, right]): [(-2.956, -1.23] < (-1.23, 0.489] < (0.4
89, 2.208] < (2.208, 3.928]]
```

In [58]:

```
def get_stats(group):
    return {'min': group.min(), 'max': group.max(),
            'count': group.count(), 'mean': group.mean()}
grouped = frame.data2.groupby(quartiles)
grouped.apply(get_stats).unstack()
```

Out[58]:

	min	max	count	mean
data1				
(-2.956, -1.23]	-3.399312	1.670835	95.0	-0.039521
(-1.23, 0.489]	-2.989741	3.260383	598.0	-0.002051
(0.489, 2.208]	-3.745356	2.954439	297.0	0.081822
(2.208, 3.928]	-1.929776	1.765640	10.0	0.024750

In [59]:

```
# Return quantile numbers
grouping = pd.qcut(frame.data1, 10, labels=False)
grouped = frame.data2.groupby(grouping)
grouped.apply(get_stats).unstack()
```

Out[59]:

data1	min	max	count	mean
0	-3.399312	1.670835	100.0	-0.049902
1	-1.950098	2.628441	100.0	0.030989
2	-2.925113	2.527939	100.0	-0.067179
3	-2.315555	3.260383	100.0	0.065713
4	-2.047939	2.074345	100.0	-0.111653
5	-2.989741	2.184810	100.0	0.052130
6	-2.223506	2.458842	100.0	-0.021489
7	-3.056990	2.954439	100.0	-0.026459
8	-3.745356	2.735527	100.0	0.103406
9	-2.064111	2.377020	100.0	0.220122

Example: Filling Missing Values with Group-Specific Values

In [60]:

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

Out[60]:

```
0         NaN
1   -0.125921
2         NaN
3   -0.884475
4         NaN
5    0.227290
dtype: float64
```

In [61]:

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

Out[61]:

```
0   -0.261035
1   -0.125921
2   -0.261035
3   -0.884475
4   -0.261035
5    0.227290
dtype: float64
```

In [62]:

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

Out[62]:

```
Ohio          0.922264
New York     -2.153545
Vermont      -0.365757
Florida      -0.375842
Oregon        0.329939
Nevada        0.981994
California    1.105913
Idaho        -1.613716
dtype: float64
```

In [63]:

```
data[['Vermont', 'Nevada', 'Idaho']] = np.nan
data
```

Out[63]:

```
Ohio          0.922264
New York     -2.153545
Vermont            NaN
Florida      -0.375842
Oregon        0.329939
Nevada             NaN
California    1.105913
Idaho              NaN
dtype: float64
```

In [64]:

```
data.groupby(group_key).mean()
```

Out[64]:

```
East   -0.535707
West    0.717926
dtype: float64
```

In [65]:

```
fill_mean = lambda g: g.fillna(g.mean())
data.groupby(group_key).apply(fill_mean)
```

Out[65]:

```
Ohio          0.922264
New York     -2.153545
Vermont      -0.535707
Florida      -0.375842
Oregon        0.329939
Nevada        0.717926
California    1.105913
Idaho         0.717926
dtype: float64
```

In [66]:

```
fill_values = {'East': 0.5, 'West': -1}
fill_func = lambda g: g.fillna(fill_values[g.name])
data.groupby(group_key).apply(fill_func)
```

Out[66]:

```
Ohio          0.922264
New York     -2.153545
Vermont       0.500000
Florida      -0.375842
Oregon        0.329939
Nevada       -1.000000
California    1.105913
Idaho        -1.000000
dtype: float64
```

Example: Random Sampling and Permutation

In [68]:

```
# Hearts, Spades, Clubs, Diamonds
suits = ['H', 'S', 'C', 'D']
card_val = (list(range(1, 11)) + [10] * 3) * 4
base_names = ['A'] + list(range(2, 11)) + ['J', 'K', 'Q']
cards = []
for suit in ['H', 'S', 'C', 'D']:
    cards.extend(str(num) + suit for num in base_names)

deck = pd.Series(card_val, index=cards)
```

In [69]:

```
deck[:13]
```

Out[69]:

```
AH      1
2H      2
3H      3
4H      4
5H      5
6H      6
7H      7
8H      8
9H      9
10H    10
JH     10
KH     10
QH     10
dtype: int64
```

In [70]:

```
def draw(deck, n=5):
    return deck.sample(n)
draw(deck)
```

Out[70]:

```
AD     1
8C     8
5H     5
KC    10
2C     2
dtype: int64
```

In [71]:

```
get_suit = lambda card: card[-1] # last letter is suit
deck.groupby(get_suit).apply(draw, n=2)
```

Out[71]:

```
C  2C     2
   3C     3
D  KD    10
   8D     8
H  KH    10
   3H     3
S  2S     2
   4S     4
dtype: int64
```

In [72]:

```
deck.groupby(get_suit, group_keys=False).apply(draw, n=2)
```

Out[72]:

```
KC    10
JC    10
AD     1
5D     5
5H     5
6H     6
7S     7
KS    10
dtype: int64
```

Example: Group Weighted Average and Correlation

In [73]:

```
df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',
                                'b', 'b', 'b', 'b'],
                   'data': np.random.randn(8),
                   'weights': np.random.rand(8)})
df
```

Out[73]:

	category	data	weights
0	a	1.561587	0.957515
1	a	1.219984	0.347267
2	a	-0.482239	0.581362
3	a	0.315667	0.217091
4	b	-0.047852	0.894406
5	b	-0.454145	0.918564
6	b	-0.556774	0.277825
7	b	0.253321	0.955905

In [74]:

```
grouped = df.groupby('category')
get_wavg = lambda g: np.average(g['data'], weights=g['weights'])
grouped.apply(get_wavg)
```

Out[74]:

```
category
a    0.811643
b   -0.122262
dtype: float64
```

In [75]:

```
close_px = pd.read_csv('examples/stock_px_2.csv', parse_dates=True,
                       index_col=0)
close_px.info()
close_px[-4:]
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2214 entries, 2003-01-02 to 2011-10-14
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   AAPL    2214 non-null   float64
 1   MSFT    2214 non-null   float64
 2   XOM     2214 non-null   float64
 3   SPX     2214 non-null   float64
dtypes: float64(4)
memory usage: 86.5 KB
```

Out[75]:

	AAPL	MSFT	XOM	SPX
2011-10-11	400.29	27.00	76.27	1195.54
2011-10-12	402.19	26.96	77.16	1207.25
2011-10-13	408.43	27.18	76.37	1203.66
2011-10-14	422.00	27.27	78.11	1224.58

In [76]:

```
spx_corr = lambda x: x.corrwith(x['SPX'])
```

In [77]:

```
rets = close_px.pct_change().dropna()
```

In [78]:

```
get_year = lambda x: x.year
by_year = rets.groupby(get_year)
by_year.apply(spx_corr)
```

Out[78]:

	AAPL	MSFT	XOM	SPX
2003	0.541124	0.745174	0.661265	1.0
2004	0.374283	0.588531	0.557742	1.0
2005	0.467540	0.562374	0.631010	1.0
2006	0.428267	0.406126	0.518514	1.0
2007	0.508118	0.658770	0.786264	1.0
2008	0.681434	0.804626	0.828303	1.0
2009	0.707103	0.654902	0.797921	1.0
2010	0.710105	0.730118	0.839057	1.0
2011	0.691931	0.800996	0.859975	1.0

In [79]:

```
by_year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
```

Out[79]:

```
2003    0.480868
2004    0.259024
2005    0.300093
2006    0.161735
2007    0.417738
2008    0.611901
2009    0.432738
2010    0.571946
2011    0.581987
dtype: float64
```

Example: Group-Wise Linear Regression

In [80]:

```
import statsmodels.api as sm
def regress(data, yvar, xvars):
    Y = data[yvar]
    X = data[xvars]
    X['intercept'] = 1.
    result = sm.OLS(Y, X).fit()
    return result.params
```

In [81]:

```
by_year.apply(regress, 'AAPL', ['SPX'])
```

Out[81]:

	SPX	intercept
2003	1.195406	0.000710
2004	1.363463	0.004201
2005	1.766415	0.003246
2006	1.645496	0.000080
2007	1.198761	0.003438
2008	0.968016	-0.001110
2009	0.879103	0.002954
2010	1.052608	0.001261
2011	0.806605	0.001514

Pivot Tables and Cross-Tabulation

In [82]:

```
tips.pivot_table(index=['day', 'smoker'])
```

Out[82]:

		size	tip	tip_pct	total_bill
day	**smoker**				
Fri	**No**	2.250000	2.812500	0.151650	18.420000
	Yes	2.066667	2.714000	0.174783	16.813333
Sat	**No**	2.555556	3.102889	0.158048	19.661778
	Yes	2.476190	2.875476	0.147906	21.276667
Sun	**No**	2.929825	3.167895	0.160113	20.506667
	Yes	2.578947	3.516842	0.187250	24.120000
Thur	**No**	2.488889	2.673778	0.160298	17.113111
	Yes	2.352941	3.030000	0.163863	19.190588

In [83]:

```
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                 columns='smoker')
```

Out[83]:

		size		tip_pct	
	smoker	**No**	**Yes**	**No**	**Yes**
time	**day**				
Dinner	**Fri**	2.000000	2.222222	0.139622	0.165347
	Sat	2.555556	2.476190	0.158048	0.147906
	Sun	2.929825	2.578947	0.160113	0.187250
	Thur	2.000000	NaN	0.159744	NaN
Lunch	**Fri**	3.000000	1.833333	0.187735	0.188937
	Thur	2.500000	2.352941	0.160311	0.163863

In [84]:

```
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                 columns='smoker', margins=True)
```

Out[84]:

		size			tip_pct		
	smoker	No	Yes	All	No	Yes	All
time	day						
Dinner	Fri	2.000000	2.222222	2.166667	0.139622	0.165347	0.158916
	Sat	2.555556	2.476190	2.517241	0.158048	0.147906	0.153152
	Sun	2.929825	2.578947	2.842105	0.160113	0.187250	0.166897
	Thur	2.000000	NaN	2.000000	0.159744	NaN	0.159744
Lunch	Fri	3.000000	1.833333	2.000000	0.187735	0.188937	0.188765
	Thur	2.500000	2.352941	2.459016	0.160311	0.163863	0.161301
All		2.668874	2.408602	2.569672	0.159328	0.163196	0.160803

In [85]:

```
tips.pivot_table('tip_pct', index=['time', 'smoker'], columns='day',
                 aggfunc=len, margins=True)
```

Out[85]:

	day	Fri	Sat	Sun	Thur	All
time	smoker					
Dinner	No	3.0	45.0	57.0	1.0	106
	Yes	9.0	42.0	19.0	NaN	70
Lunch	No	1.0	NaN	NaN	44.0	45
	Yes	6.0	NaN	NaN	17.0	23
All		19.0	87.0	76.0	62.0	244

In [87]:

```
tips.pivot_table('tip_pct', index=['time', 'smoker'], columns='day',
                 aggfunc='count', margins=True)
```

Out[87]:

	day	Fri	Sat	Sun	Thur	All
time	smoker					
Dinner	No	3.0	45.0	57.0	1.0	106
	Yes	9.0	42.0	19.0	NaN	70
Lunch	No	1.0	NaN	NaN	44.0	45
	Yes	6.0	NaN	NaN	17.0	23
All		19.0	87.0	76.0	62.0	244

In [88]:

```
tips.pivot_table('tip_pct', index=['time', 'size', 'smoker'],
                 columns='day', aggfunc='mean', fill_value=0)
```

Out[88]:

		day	Fri	Sat	Sun	Thur
time	**size**	**smoker**				
Dinner	**1**	**No**	0.000000	0.137931	0.000000	0.000000
		Yes	0.000000	0.325733	0.000000	0.000000
	2	**No**	0.139622	0.162705	0.168859	0.159744
		Yes	0.171297	0.148668	0.207893	0.000000
	3	**No**	0.000000	0.154661	0.152663	0.000000
...	**...**	**...**	...	...	...	...
Lunch	**3**	**Yes**	0.000000	0.000000	0.000000	0.204952
	4	**No**	0.000000	0.000000	0.000000	0.138919
		Yes	0.000000	0.000000	0.000000	0.155410
	5	**No**	0.000000	0.000000	0.000000	0.121389
	6	**No**	0.000000	0.000000	0.000000	0.173706

21 rows × 4 columns

Cross-Tabulations: Crosstab

In [89]:

```
from io import StringIO
data = """\
Sample  Nationality  Handedness
1   USA  Right-handed
2   Japan    Left-handed
3   USA  Right-handed
4   Japan    Right-handed
5   Japan    Left-handed
6   Japan    Right-handed
7   USA  Right-handed
8   USA  Left-handed
9   Japan    Right-handed
10  USA  Right-handed"""
data = pd.read_table(StringIO(data), sep='\s+')
```

In [90]:

```
data
```

Out[90]:

	Sample	Nationality	Handedness
0	1	USA	Right-handed
1	2	Japan	Left-handed
2	3	USA	Right-handed
3	4	Japan	Right-handed
4	5	Japan	Left-handed
5	6	Japan	Right-handed
6	7	USA	Right-handed
7	8	USA	Left-handed
8	9	Japan	Right-handed
9	10	USA	Right-handed

```
In [91]: pd.crosstab(data.Nationality, data.Handedness, margins=True)
```

Out[91]:

Handedness	Left-handed	Right-handed	All
Nationality			
Japan	2	3	5
USA	1	4	5
All	3	7	10

```
In [92]: pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
```

Out[92]:

	smoker	No	Yes	All
time	**day**			
Dinner	**Fri**	3	9	12
	Sat	45	42	87
	Sun	57	19	76
	Thur	1	0	1
Lunch	**Fri**	1	6	7
	Thur	44	17	61
All		151	93	244

```
In [93]: pd.options.display.max_rows = PREVIOUS_MAX_ROWS
```

Conclusion