In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df = pd.DataFrame({'key':['A','B','C','A','B','C','A','B','C'],'data':[0,5,10,5,10,15,10,15,20]})
df
```

Out[2]:

	key	data
0	A	0
1	B	5
2	C	10
3	A	5
4	B	10
5	C	15
6	A	10
7	B	15
8	C	20

In [3]:
```
groupdf = df['data'].groupby(df['key'])

for name, group in groupdf:
    print(name)
    print(group)
```

```
A
0     0
3     5
6    10
Name: data, dtype: int64
B
1     5
4    10
7    15
Name: data, dtype: int64
C
2    10
5    15
8    20
Name: data, dtype: int64
```

In [4]:
```
df = pd.DataFrame({'key1' : ['a', 'b', 'b', 'a', 'a'],
 'key2' : ['one', 'two', 'one', 'two', 'one'],
 'data1' : np.random.randn(5),
'data2' : np.random.randn(5)})
```

In [5]:
```
for (k1, k2), group in df.groupby(['key1', 'key2']):
    print((k1, k2))
    print(group)
```

```
('a', 'one')
  key1 key2     data1     data2
0    a  one  0.079666 -0.756283
4    a  one  1.720034 -1.369992
('a', 'two')
  key1 key2    data1     data2
3    a  two -1.18315 -1.587875
('b', 'one')
  key1 key2     data1     data2
2    b  one  1.566039  0.187998
('b', 'two')
  key1 key2     data1     data2
1    b  two  1.895124  0.719245
```

In [6]:
```
df['data2'].groupby(df['key1']).sum()
```

Out[6]:
```
key1
a   -3.714150
b    0.907243
Name: data2, dtype: float64
```

In [7]:
```
means = df['data1'].groupby([df['key1'], df['key2']]).mean()
means
```

Out[7]:
```
key1  key2
a     one     0.899850
      two    -1.183150
b     one     1.566039
      two     1.895124
Name: data1, dtype: float64
```

In [8]:
```
means.unstack()
```

Out[8]:

key2	one	two
key1		
a	0.899850	-1.183150
b	1.566039	1.895124

In [9]:
```
states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
```

In [10]:
```
years = np.array([2005, 2006, 2005, 2006, 2005])
df['data1'].groupby([states, years]).mean()
```

Out[10]:
```
California  2005    1.566039
            2006    1.895124
Ohio        2005    0.899850
            2006   -1.183150
Name: data1, dtype: float64
```

In [11]:
```
df['data1'].groupby([states, years]).mean()
```

Out[11]:
```
California  2005    1.566039
            2006    1.895124
Ohio        2005    0.899850
            2006   -1.183150
Name: data1, dtype: float64
```

In [12]:
```
df['states']=states
df['years']=years
```

In [13]:
```
df
```

Out[13]:

	key1	key2	data1	data2	states	years
0	a	one	0.079666	-0.756283	Ohio	2005
1	b	two	1.895124	0.719245	California	2006
2	b	one	1.566039	0.187998	California	2005
3	a	two	-1.183150	-1.587875	Ohio	2006
4	a	one	1.720034	-1.369992	Ohio	2005

In [14]:
```
df['data1'].groupby([df['states'],df['years']]).mean()
```

Out[14]:
```
states      years
California  2005     1.566039
            2006     1.895124
Ohio        2005     0.899850
            2006    -1.183150
Name: data1, dtype: float64
```

In [15]:
```
df['data1'].groupby([df['key1'], df['key2']]).mean()
```

Out[15]:
```
key1  key2
a     one     0.899850
      two    -1.183150
b     one     1.566039
      two     1.895124
Name: data1, dtype: float64
```

In [16]:
```
df.groupby(['key1', 'key2']).size()
```

Out[16]:
```
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

In [17]:
```
df
```

Out[17]:

	key1	key2	data1	data2	states	years
0	a	one	0.079666	-0.756283	Ohio	2005
1	b	two	1.895124	0.719245	California	2006
2	b	one	1.566039	0.187998	California	2005
3	a	two	-1.183150	-1.587875	Ohio	2006
4	a	one	1.720034	-1.369992	Ohio	2005

In [18]:
```
df.groupby(['key1', 'key2']).size()
```

Out[18]:
```
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

In [19]:
```
pieces = dict(list(df.groupby('key1')))
```

In [20]:
```
pieces
```

Out[20]:
```
{'a':   key1 key2     data1     data2 states  years
 0    a  one  0.079666 -0.756283   Ohio   2005
 3    a  two -1.183150 -1.587875   Ohio   2006
 4    a  one  1.720034 -1.369992   Ohio   2005,
 'b':   key1 key2     data1     data2      states  years
 1    b  two  1.895124  0.719245  California   2006
 2    b  one  1.566039  0.187998  California   2005}
```

In [21]:
```
df.groupby('key1')['data1'].sum()
```

Out[21]:
```
key1
a    0.616550
b    3.461163
Name: data1, 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'])
```

In [23]:
```
# Add a few NA values
people.iloc[2:3, [1, 2]] = np.nan
```

In [24]:
```
people
```

Out[24]:

	a	b	c	d	e
Joe	1.208190	2.048987	0.358917	0.484018	0.202471
Steve	-1.786598	-0.518500	-0.603746	0.333413	-0.459614
Wes	1.400862	NaN	NaN	-0.955373	-1.991733
Jim	-0.838726	0.000937	-0.362894	0.324241	0.355562
Travis	2.148565	-0.360806	1.038020	0.659817	0.398897

Now, suppose I have a group correspondence for the columns and want to sum together the columns by group:

In [25]:
```
mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
'd': 'blue', 'e': 'red', 'f' : 'orange'}
```

In [26]:
```
by_column = people.groupby(mapping,axis=1)
by_column.count()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_7420\941954695.py:1: FutureWarning: DataFrame.groupby with axis=1 is deprecated. Do `frame.T.groupby(...)` without axis inst
ead.
  by_column = people.groupby(mapping,axis=1)
```

Out[26]:

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

In [27]:
```
map_series = pd.Series(mapping)
```

In [28]:
```
people.groupby(map_series, axis=1).count()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_7420\1833935060.py:1: FutureWarning: DataFrame.groupby with axis=1 is deprecated. Do `frame.T.groupby(...)` without axis ins
tead.
  people.groupby(map_series, axis=1).count()
```

Out[28]:

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

Grouping with Functions

In [29]:
```
people.groupby(len).sum()
```

Out[29]:

	a	b	c	d	e
3	1.770326	2.049924	-0.003977	-0.147114	-1.433701
5	-1.786598	-0.518500	-0.603746	0.333413	-0.459614
6	2.148565	-0.360806	1.038020	0.659817	0.398897

Grouping by Index Levels

In [30]:
```
columns = pd.MultiIndex.from_arrays([['US', 'US', 'US', 'JP', 'JP'],[1, 3, 5, 1, 3]], names=['city', 'tenor'])
hier_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)
```

In [31]:
```
hier_df
```

Out[31]:

city			US		JP
tenor	**1**	**3**	**5**	**1**	**3**
0	0.377038	1.781552	-0.996772	-0.099908	-0.379270
1	-1.052875	1.148036	3.133074	-0.584158	0.413407
2	-0.779934	0.170106	-1.161362	-0.298124	-0.419382
3	0.635020	-0.134928	-2.012240	0.203866	-1.015665

In [32]:
```
hier_df.groupby(level='city', axis=1).count()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_7420\419287342.py:1: FutureWarning: DataFrame.groupby with axis=1 is deprecated. Do `frame.T.groupby(...)` without axis inst
ead.
  hier_df.groupby(level='city', axis=1).count()
```

Out[32]:

city	JP	US
0	2	3
1	2	3
2	2	3
3	2	3

Data Aggregation

In [33]:
```
df
```

Out[33]:

	key1	key2	data1	data2	states	years
0	a	one	0.079666	-0.756283	Ohio	2005
1	b	two	1.895124	0.719245	California	2006
2	b	one	1.566039	0.187998	California	2005
3	a	two	-1.183150	-1.587875	Ohio	2006
4	a	one	1.720034	-1.369992	Ohio	2005

In [34]:
```
grouped = df.groupby(['key1','key2'])
grouped['data1'].sum()
```

Out[34]:
```
key1  key2
a     one     1.799700
      two    -1.183150
b     one     1.566039
      two     1.895124
Name: data1, dtype: float64
```

In [35]:
```
grouped['data1'].quantile(0.9)
```

Out[35]:
```
key1  key2
a     one     1.555997
      two    -1.183150
b     one     1.566039
      two     1.895124
Name: data1, dtype: float64
```

In [36]:
```
df
```

Out[36]:

	key1	key2	data1	data2	states	years
0	a	one	0.079666	-0.756283	Ohio	2005
1	b	two	1.895124	0.719245	California	2006
2	b	one	1.566039	0.187998	California	2005
3	a	two	-1.183150	-1.587875	Ohio	2006
4	a	one	1.720034	-1.369992	Ohio	2005

In [37]:
```
def peak_to_peak(arr):
    return arr.max()-arr.min()
grouped['data1'].agg(peak_to_peak)
```

Out[37]:
```
key1  key2
a     one     1.640368
      two     0.000000
b     one     0.000000
      two     0.000000
Name: data1, dtype: float64
```

In [38]:
```
grouped.describe()
```

Out[38]:

		data1								data2								
		count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std	min	25%	50%
key1	key2																		
a	one	2.0	0.899850	1.159915	0.079666	0.489758	0.899850	1.309942	1.720034	2.0	-1.063138	...	-0.909710	-0.756283	2.0	2005.0	0.0	2005.0	2005.0	2005.0
	two	1.0	-1.183150	NaN	-1.183150	-1.183150	-1.183150	-1.183150	-1.183150	1.0	-1.587875	...	-1.587875	-1.587875	1.0	2006.0	NaN	2006.0	2006.0	2006.0
b	one	1.0	1.566039	NaN	1.566039	1.566039	1.566039	1.566039	1.566039	1.0	0.187998	...	0.187998	0.187998	1.0	2005.0	NaN	2005.0	2005.0	2005.0
	two	1.0	1.895124	NaN	1.895124	1.895124	1.895124	1.895124	1.895124	1.0	0.719245	...	0.719245	0.719245	1.0	2006.0	NaN	2006.0	2006.0	2006.0

4 rows × 24 columns

Column-Wise and Multiple Function Application

In [39]:
```
tips = pd.read_csv('tips.csv')
tips
```

Out[39]:

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

244 rows × 7 columns

In [40]:
```
# Add tip percentage of total bill
tips['tip_pct'] = tips['tip'] / tips['total_bill']
```

In [41]:
```
tips[:6]
```

Out[41]:

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

In [42]:
```
grouped = tips.groupby(['day', 'smoker'])

grouped_pct = grouped['tip_pct']

grouped_pct.agg('mean')
```

Out[42]:
```
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 [43]:
```
grouped_pct.agg(['mean', 'std', peak_to_peak])
```

Out[43]:

		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 [44]:
```
grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_7420\345979284.py:1: FutureWarning: The provided callable <function std at 0x000001BDA2309940> is currently using SeriesGrou
pBy.std. In a future version of pandas, the provided callable will be used directly. To keep current behavior pass the string "std" instead.
  grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
```

Out[44]:

		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 [45]:
```
functions = ['count', 'mean', 'max']
result = grouped['tip_pct'].agg(functions)
result
```

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 [ ]: