

# Pandas

# **Getting Started with pandas**

# Series

```
1 obj = pd.Series([4, 7, -5, 3])  
2 obj
```

```
0    4  
1    7  
2   -5  
3    3  
dtype: int64
```

```
1 obj.values  
2 obj.index # like range(4)
```

RangeIndex(start=0, stop=4, step=1)

# Series

```
1 obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])  
2 obj2  
3 obj2.index
```

```
Index(['d', 'b', 'a', 'c'], dtype='object')
```

```
1 obj2['a']  
2 obj2['d'] = 6  
3 obj2[['c', 'a', 'd']]
```

```
c    3  
a   -5  
d    6  
dtype: int64
```

# Series

```
1 obj2[obj2 > 0]
2 obj2 * 2
3 np.exp(obj2)
```

```
d    403.428793
b    1096.633158
a     0.006738
c    20.085537
dtype: float64
```

```
1 'b' in obj2
2 'e' in obj2
```

False

# Series

```
1 sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000,  
2             'Utah': 5000}  
3 obj3 = pd.Series(sdata)  
4 obj3
```

```
Ohio      35000  
Texas     71000  
Oregon    16000  
Utah      5000  
dtype: int64
```

# Series

```
1 states = ['California', 'Ohio', 'Oregon', 'Texas']  
2 obj4 = pd.Series(sdata, index=states)  
3 obj4
```

```
California      NaN  
Ohio          35000.0  
Oregon        16000.0  
Texas         71000.0  
dtype: float64
```

# Series

```
1 pd.isnull(obj4)  
2 pd.notnull(obj4)
```

```
California    False  
Ohio          True  
Oregon         True  
Texas          True  
dtype: bool
```

```
1 obj4.isnull()
```

```
California    True  
Ohio          False  
Oregon         False  
Texas          False  
dtype: bool
```

# Series

```
1 obj3  
2 obj4  
3 obj3 + obj4
```

```
California      NaN  
Ohio          70000.0  
Oregon        32000.0  
Texas         142000.0  
Utah           NaN  
dtype: float64
```

```
1 obj4.name = 'population'  
2 obj4.index.name = 'state'  
3 obj4
```

```
state  
California      NaN  
Ohio          35000.0  
Oregon        16000.0  
Texas         71000.0  
Name: population, dtype: float64
```

# Series

```
1 obj
2 obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']
3 obj
```

```
Bob      4
Steve    7
Jeff     -5
Ryan     3
dtype: int64
```

# DataFrame

```
1 data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
2        'year': [2000, 2001, 2002, 2001, 2002, 2003],
3        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
4 frame = pd.DataFrame(data)
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4

# DataFrame

```
1 frame.head()
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9

# DataFrame

```
1 pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9
5	2003	Nevada	3.2

# DataFrame

```
1 frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],  
2                         index=['one', 'two', 'three', 'four',  
3                           'five', 'six'])  
4 frame2  
5 frame2.columns
```

```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

# DataFrame

```
1 frame2['state']
2 frame2.year
```

```
one      2000
two      2001
three    2002
four     2001
five     2002
six      2003
Name: year , dtype: int64
```

# DataFrame

```
1 frame2.loc['three']
```

```
year      2002
state    Ohio
pop       3.6
debt      NaN
Name: three, dtype: object
```

# DataFrame

```
1 frame2['debt'] = 16.5
2 frame2
3 frame2['debt'] = np.arange(6.)
4 frame2
```

	year	state	pop	debt
<b>one</b>	2000	Ohio	1.5	0.0
<b>two</b>	2001	Ohio	1.7	1.0
<b>three</b>	2002	Ohio	3.6	2.0
<b>four</b>	2001	Nevada	2.4	3.0

# DataFrame

```
1 val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
2 frame2['debt'] = val
3 frame2
```

	year	state	pop	debt
<b>one</b>	2000	Ohio	1.5	NaN
<b>two</b>	2001	Ohio	1.7	-1.2
<b>three</b>	2002	Ohio	3.6	NaN
<b>four</b>	2001	Nevada	2.4	-1.5

# DataFrame

```
1 frame2['eastern'] = frame2.state == 'Ohio'  
2 frame2
```

	year	state	pop	debt	eastern
<b>one</b>	2000	Ohio	1.5	NaN	True
<b>two</b>	2001	Ohio	1.7	-1.2	True
<b>three</b>	2002	Ohio	3.6	NaN	True
<b>four</b>	2001	Nevada	2.4	-1.5	False

# DataFrame

```
1 del frame2['eastern']
2 frame2.columns
```

```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

```
1 pop = {'Nevada': {2001: 2.4, 2002: 2.9},
2      'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

# DataFrame

```
1 frame3 = pd.DataFrame(pop)  
2 frame3
```

	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

```
1 frame3.T
```

	2000	2001	2002
Nevada	NaN	2.4	2.9
Ohio	1.5	1.7	3.6

# DataFrame

```
1 pd.DataFrame(pop, index=[2001, 2002, 2003])
```

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2003	NaN	NaN

# DataFrame

```
1 pdata = {'Ohio': frame3['Ohio'][:-1],  
2         'Nevada': frame3['Nevada'][:2]}  
3 pd.DataFrame(pdata)
```

	<b>Ohio</b>	<b>Nevada</b>
<b>2000</b>	1.5	NaN
<b>2001</b>	1.7	2.4

# DataFrame

```
1 frame3.index.name = 'year'; frame3.columns.name = 'state'  
2 frame3
```

state	Nevada	Ohio
year		
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

# DataFrame

```
1 | frame3.values
```

```
array([[nan, 1.5],  
       [2.4, 1.7],  
       [2.9, 3.6]])
```

```
1 | frame2.values
```

```
array([[2000, 'Ohio', 1.5, nan],  
       [2001, 'Ohio', 1.7, -1.2],  
       [2002, 'Ohio', 3.6, nan],  
       [2001, 'Nevada', 2.4, -1.5],  
       [2002, 'Nevada', 2.9, -1.7],  
       [2003, 'Nevada', 3.2, nan]], dtype=object)
```

# Index Objects

```
1 obj = pd.Series(range(3), index=['a', 'b', 'c'])
2 index = obj.index
3 index
4 index[1:]
```

```
Index(['b', 'c'], dtype='object')
```

```
index[1] = 'd' # TypeError
```

# Index Objects

```
1 labels = pd.Index(np.arange(3))
2 labels
3 obj2 = pd.Series([1.5, -2.5, 0], index=labels)
4 obj2
5 obj2.index is labels
```

True

```
1 frame3
2 frame3.columns
3 'Ohio' in frame3.columns
4 2003 in frame3.index
```

False

# Index Objects

```
1 dup_labels = pd.Index(['foo', 'foo', 'bar', 'bar'])  
2 dup_labels
```

```
Index(['foo', 'foo', 'bar', 'bar'], dtype='object')
```

# Reindexing

```
1 | obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])  
2 | obj
```

```
d    4.5  
b    7.2  
a   -5.3  
c    3.6  
dtype: float64
```

# Reindexing

```
1 obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])  
2 obj2
```

```
a    -5.3  
b     7.2  
c     3.6  
d     4.5  
e      NaN  
dtype: float64
```

# Reindexing

```
1 obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])  
2 obj3  
3 obj3.reindex(range(6), method='ffill')
```

```
0    blue  
1    blue  
2  purple  
3  purple  
4  yellow  
5  yellow  
dtype: object
```

reindex method (interpolation) options

Argument	Description
<code>ffill</code> or <code>pad</code>	Fill (or carry) values forward
<code>bfill</code> or <code>backfill</code>	Fill (or carry) values backward

# Reindexing

```
1 frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
2                      index=['a', 'c', 'd'],
3                      columns=['Ohio', 'Texas', 'California'])
4 frame
5 frame2 = frame.reindex(['a', 'b', 'c', 'd'])
6 frame2
```

	Ohio	Texas	California
--	------	-------	------------

a	0.0	1.0	2.0
---	-----	-----	-----

b	NaN	NaN	NaN
---	-----	-----	-----

c	3.0	4.0	5.0
---	-----	-----	-----

d	6.0	7.0	8.0
---	-----	-----	-----

# Reindexing

```
1 states = ['Texas', 'Utah', 'California']
2 frame.reindex(columns=states)
```

Texas Utah California

a 1 NaN 2

c 4 NaN 5

d 7 NaN 8

# Dropping Entries from an Axis

```
1 obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
2 obj
3 new_obj = obj.drop('c')
4 new_obj
5 obj.drop(['d', 'c'])
```

```
a    0.0
b    1.0
e    4.0
dtype: float64
```

# Dropping Entries from an Axis

```
1 data = pd.DataFrame(np.arange(16).reshape((4, 4)),  
2                      index=['Ohio', 'Colorado', 'Utah', 'New York'],  
3                      columns=['one', 'two', 'three', 'four'])  
4 data
```

	one	two	three	four
--	-----	-----	-------	------

<b>Ohio</b>	0	1	2	3
-------------	---	---	---	---

<b>Colorado</b>	4	5	6	7
-----------------	---	---	---	---

<b>Utah</b>	8	9	10	11
-------------	---	---	----	----

<b>New York</b>	12	13	14	15
-----------------	----	----	----	----

# Dropping Entries from an Axis

```
1 data.drop(['Colorado', 'Ohio'])
```

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

# Dropping Entries from an Axis

```
1 data.drop('two', axis=1)
2 data.drop(['two', 'four'], axis='columns')
```

	one	three
<b>Ohio</b>	0	2
<b>Colorado</b>	4	6
<b>Utah</b>	8	10
<b>New York</b>	12	14

# Dropping Entries from an Axis

```
1 obj.drop('c', inplace=True)
2 obj
```

```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

# Indexing, Selection, and Filtering

```
1 obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
2 obj
3 obj['b']
4 obj[1]
5 obj[2:4]
6 obj[['b', 'a', 'd']]
7 obj[[1, 3]]
8 obj[obj < 2]
```

```
a    0.0
b    1.0
dtype: float64
```

# Indexing, Selection, and Filtering

```
1 obj['b':'c']
```

```
b    1.0  
c    2.0  
dtype: float64
```

```
1 obj['b':'c'] = 5  
2 obj
```

```
a    0.0  
b    5.0  
c    5.0  
d    3.0  
dtype: float64
```

# Indexing, Selection, and Filtering

```
1 data = pd.DataFrame(np.arange(16).reshape((4, 4)),  
2                      index=['Ohio', 'Colorado', 'Utah', 'New York'],  
3                      columns=['one', 'two', 'three', 'four'])  
4 data  
5 data['two']  
6 data[['three', 'one']]
```

	three	one
Ohio	2	0
Colorado	6	4
Utah	10	8

# Indexing, Selection, and Filtering

```
1 data[:2]
2 data[data['three'] > 5]
```

	one	two	three	four
<b>Colorado</b>	4	5	6	7
<b>Utah</b>	8	9	10	11
<b>New York</b>	12	13	14	15

# Indexing, Selection, and Filtering

```
1 data < 5  
2 data[data < 5] = 0  
3 data
```

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

# Selection with loc and iloc

```
1 data.loc['Colorado', ['two', 'three']]
```

```
two      5  
three    6  
Name: Colorado, dtype: int32
```

```
1 data.iloc[2, [3, 0, 1]]  
2 data.iloc[2]  
3 data.iloc[[1, 2], [3, 0, 1]]
```

	four	one	two
<b>Colorado</b>	7	0	5
<b>Utah</b>	11	8	9

# Selection with loc and iloc

```
1 data.loc[:'Utah', 'two']
2 data.iloc[:, :3][data.three > 5]
```

	one	two	three
<b>Colorado</b>	0	5	6
<b>Utah</b>	8	9	10
<b>New York</b>	12	13	14

# Integer Indexes

```
1 ser = pd.Series(np.arange(3.))
```

```
1 ser
```

```
0    0.0
1    1.0
2    2.0
dtype: float64
```

# Integer Indexes

```
1 ser2 = pd.Series(np.arange(3.), index=['a', 'b', 'c'])  
2 ser2[-1]
```

2.0

```
1 ser[:1]  
2 ser.loc[:1]  
3 ser.iloc[:1]
```

0 0.0  
dtype: float64

# Arithmetic and Data Alignment

```
1 s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])
2 s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
3                 index=['a', 'c', 'e', 'f', 'g'])
4 s1
5 s2
```

```
a    -2.1
c     3.6
e    -1.5
f     4.0
g     3.1
dtype: float64
```

# Arithmetic and Data Alignment

```
1 | s1 + s2
```

```
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

# Arithmetic and Data Alignment

```
1 df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),  
2                     index=['Ohio', 'Texas', 'Colorado'])  
3 df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),  
4                     index=['Utah', 'Ohio', 'Texas', 'Oregon'])  
5 df1  
6 df2
```

	b	d	e
Utah	0.0	1.0	2.0
Ohio	3.0	4.0	5.0
Texas	6.0	7.0	8.0
Oregon	9.0	10.0	11.0

# Arithmetic and Data Alignment

```
1 df1 + df2
```

	b	c	d	e
<b>Colorado</b>	NaN	NaN	NaN	NaN
<b>Ohio</b>	3.0	NaN	6.0	NaN
<b>Oregon</b>	NaN	NaN	NaN	NaN
<b>Texas</b>	9.0	NaN	12.0	NaN
<b>Utah</b>	NaN	NaN	NaN	NaN

# Arithmetic and Data Alignment

```
1 df1 = pd.DataFrame({'A': [1, 2]})  
2 df2 = pd.DataFrame({'B': [3, 4]})  
3 df1  
4 df2  
5 df1 - df2
```

	A	B
0	NaN	NaN
1	NaN	NaN

# Arithmetic methods with fill values

```
1 df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),  
2                      columns=list('abcd'))  
3 df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),  
4                      columns=list('abcde'))  
5 df2.loc[1, 'b'] = np.nan  
6 df1  
7 df2
```

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	4.0
1	5.0	NaN	7.0	8.0	9.0
2	10.0	11.0	12.0	13.0	14.0
3	15.0	16.0	17.0	18.0	19.0

# Arithmetic methods with fill values

```
1 df1 + df2
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	NaN
1	9.0	NaN	13.0	15.0	NaN
2	18.0	20.0	22.0	24.0	NaN
3	NaN	NaN	NaN	NaN	NaN

# Arithmetic methods with fill values

```
1 df1.add(df2, fill_value=0)
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	4.0
1	9.0	5.0	13.0	15.0	9.0
2	18.0	20.0	22.0	24.0	14.0
3	15.0	16.0	17.0	18.0	19.0

# Arithmetic methods with fill values

```
1 1 / df1  
2 df1.rdiv(1)
```

	a	b	c	d
0	inf	<u>1.000000</u>	0.500000	<u>0.333333</u>
1	0.250	<u>0.200000</u>	<u>0.166667</u>	0.142857
2	0.125	<u>0.111111</u>	<u>0.100000</u>	0.090909

# Arithmetic methods with fill values

```
1 df1.reindex(columns=df2.columns, fill_value=0)
```

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	0
1	4.0	5.0	6.0	7.0	0
2	8.0	9.0	10.0	11.0	0

# Operations between DataFrame and Series

```
1 arr = np.arange(12.).reshape((3, 4))
2 arr
3 arr[0]
4 arr - arr[0]
```

```
array([[0., 0., 0., 0.],
       [4., 4., 4., 4.],
       [8., 8., 8., 8.]])
```

# Operations between DataFrame and Series

```
1 frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
2                      columns=list('bde'),
3                      index=['Utah', 'Ohio', 'Texas', 'Oregon'])
4 series = frame.iloc[0]
5 frame
6 series
```

```
b    0.0
d    1.0
e    2.0
Name: Utah, dtype: float64
```

# Operations between DataFrame and Series

```
1 frame - series
```

	b	d	e
Utah	0.0	0.0	0.0
Ohio	3.0	3.0	3.0
Texas	6.0	6.0	6.0
Oregon	9.0	9.0	9.0

# Operations between DataFrame and Series

```
1 series2 = pd.Series(range(3), index=['b', 'e', 'f'])  
2 frame + series2
```

	b	d	e	f
<b>Utah</b>	0.0	NaN	3.0	NaN
<b>Ohio</b>	3.0	NaN	6.0	NaN
<b>Texas</b>	6.0	NaN	9.0	NaN
<b>Oregon</b>	9.0	NaN	12.0	NaN

# Operations between DataFrame and Series

```
1 series3 = frame['d']
2 frame
3 series3
4 frame.sub(series3, axis='index')
```

	<b>b</b>	<b>d</b>	<b>e</b>
<b>Utah</b>	-1.0	0.0	1.0
<b>Ohio</b>	-1.0	0.0	1.0
<b>Texas</b>	-1.0	0.0	1.0
<b>Oregon</b>	-1.0	0.0	1.0

# Function Application and Mapping

```
1 frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'),  
2 index=['Utah', 'Ohio', 'Texas', 'Oregon'])  
3 frame  
4 np.abs(frame)
```

	b	d	e
Utah	<a href="#">0.204708</a>	0.478943	<a href="#">0.519439</a>
Ohio	<a href="#">0.555730</a>	<a href="#">1.965781</a>	1.393406
Texas	0.092908	<a href="#">0.281746</a>	0.769023
Oregon	1.246435	<a href="#">1.007189</a>	1.296221

# Function Application and Mapping

```
1 f = lambda x: x.max() - x.min()  
2 frame.apply(f)
```

```
b    1.802165  
d    1.684034  
e    2.689627  
dtype: float64
```

```
1 frame.apply(f, axis='columns')
```

Utah	0.998382
Ohio	2.521511
Texas	0.676115
Oregon	2.542656

```
dtype: float64
```

# Function Application and Mapping

```
1 def f(x):  
2     return pd.Series([x.min(), x.max()], index=['min', 'max'])  
3 frame.apply(f)
```

	b	d	e
<b>min</b>	-0.555730	0.281746	-1.296221
<b>max</b>	1.246435	1.965781	1.393406

# Function Application and Mapping

```
1 format = lambda x: '%.2f' % x  
2 frame.applymap(format)
```

	b	d	e
<b>Utah</b>	-0.20	0.48	-0.52
<b>Ohio</b>	-0.56	1.97	1.39
<b>Texas</b>	0.09	0.28	0.77
<b>Oregon</b>	1.25	1.01	-1.30

# Function Application and Mapping

```
1 frame['e'].map(format)
```

```
Utah      -0.52
Ohio      1.39
Texas     0.77
Oregon    -1.30
Name: e, dtype: object
```

# Sorting and Ranking

```
1 obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])  
2 obj.sort_index()
```

```
a    1  
b    2  
c    3  
d    0  
dtype: int64
```

# Sorting and Ranking

```
1 frame = pd.DataFrame(np.arange(8).reshape((2, 4)),  
2                      index=['three', 'one'],  
3                      columns=['d', 'a', 'b', 'c'])  
4 frame.sort_index()  
5 frame.sort_index(axis=1)
```

	a	b	c	d
three	1	2	3	0
one	5	6	7	4

# Sorting and Ranking

```
1 frame.sort_index(axis=1, ascending=False)
```

	d	c	b	a
three	0	3	2	1
one	4	7	6	5

# Sorting and Ranking

```
1 obj = pd.Series([4, 7, -3, 2])  
2 obj.sort_values()
```

```
2 -3  
3 2  
0 4  
1 7  
dtype: int64
```

# Sorting and Ranking

```
1 obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])  
2 obj.sort_values()
```

```
4    -3.0  
5     2.0  
0     4.0  
2     7.0  
1     NaN  
3     NaN  
dtype: float64
```

# Sorting and Ranking

```
1 frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})  
2 frame  
3 frame.sort_values(by='b')
```

	b	a
2	-3	0
3	2	1
0	4	0
1	7	1

# Sorting and Ranking

```
1 frame.sort_values(by=['a', 'b'])
```

	b	a
2	-3	0
0	4	0
3	2	1
1	7	1

# Sorting and Ranking

```
1 obj = pd.Series([7, -5, 7, 4, 2, 0, 4])  
2 obj.rank()
```

```
0    6.5  
1    1.0  
2    6.5  
3    4.5  
4    3.0  
5    2.0  
6    4.5  
dtype: float64
```

# Sorting and Ranking

```
1 obj.rank(method='first')
```

```
0    6.0
1    1.0
2    7.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

# Sorting and Ranking

```
1 # Assign tie values the maximum rank in the group  
2 obj.rank(ascending=False, method='max')
```

```
0    2.0  
1    7.0  
2    2.0  
3    4.0  
4    5.0  
5    6.0  
6    4.0  
dtype: float64
```

# Sorting and Ranking

Tie-breaking methods with rank

Method	Description
'average'	Default: assign the average rank to each entry in the equal group.
'min'	Use the minimum rank for the whole group.
'max'	Use the maximum rank for the whole group.
'first'	Assign ranks in the order the values appear in the data.

# Sorting and Ranking

```
1 frame = pd.DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],  
2                           'c': [-2, 5, 8, -2.5]})  
3 frame  
4 frame.rank(axis='columns')
```

	b	a	c
0	3.0	2.0	1.0
1	3.0	1.0	2.0
2	1.0	2.0	3.0
3	3.0	2.0	1.0

# Axis Indexes with Duplicate Labels

```
1 obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])  
2 obj
```

```
a    0  
a    1  
b    2  
b    3  
c    4  
dtype: int64
```

```
1 obj.index.is_unique
```

```
False
```

# Axis Indexes with Duplicate Labels

```
1 obj['a']
2 obj['c']
```

4

```
1 df = pd.DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])
2 df
3 df.loc['b']
```

0 1 2

	0	1	2
<b>b</b>	<u>1.669025</u>	<u>-0.438570</u>	<u>-0.539741</u>
<b>b</b>	0.476985	<u>3.248944</u>	<u>-1.021228</u>

# Summarizing and Computing Descriptive Statistics

```
1 df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],  
2                 [np.nan, np.nan], [0.75, -1.3]],  
3                 index=['a', 'b', 'c', 'd'],  
4                 columns=['one', 'two'])  
5 df
```

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

# Summarizing and Computing Descriptive Statistics

```
1 df.sum()
```

```
one    9.25
two   -5.80
dtype: float64
```

```
1 df.sum(axis='columns')
```

```
a    1.40
b    2.60
c    0.00
d   -0.55
dtype: float64
```

# Summarizing and Computing Descriptive Statistics

```
1 df.mean(axis='columns', skipna=False)
```

```
a      NaN  
b    1.300  
c      NaN  
d   -0.275  
dtype: float64
```

Options for reduction methods

Method	Description
axis	Axis to reduce over. 0 for DataFrame's rows and 1 for columns.
skipna	Exclude missing values, True by default.
level	Reduce grouped by level if the axis is hierarchically-indexed (MultiIndex).

# Summarizing and Computing Descriptive Statistics

```
1 df.idxmax()
```

```
one    b  
two    d  
dtype: object
```

```
1 df.cumsum()
```

	one	two
<b>a</b>	1.40	NaN
<b>b</b>	8.50	-4.5
<b>c</b>	NaN	NaN
<b>d</b>	9.25	-5.8

# Summarizing and Computing Descriptive Statistics

```
1 df.describe()
```

	one	two			
<b>count</b>	3.000000	<u>2.000000</u>	<b>25%</b>	<u>1.075000</u>	-3.700000
<b>mean</b>	3.083333	<u>-2.900000</u>	<b>50%</b>	1.400000	<u>-2.900000</u>
<b>std</b>	3.493685	<u>2.262742</u>	<b>75%</b>	<u>4.250000</u>	<u>-2.100000</u>
<b>min</b>	0.750000	-4.500000	<b>max</b>	7.100000	-1.300000

# Summarizing and Computing Descriptive Statistics

```
1 obj = pd.Series(['a', 'a', 'b', 'c'] * 4)  
2 obj.describe()
```

```
count      16  
unique      3  
top        a  
freq       8  
dtype: object
```

# Unique Values, Value Counts, and Membership

```
1 obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
```

```
1 uniques = obj.unique()
2 uniques
```

```
array(['c', 'a', 'd', 'b'], dtype=object)
```

# Unique Values, Value Counts, and Membership

```
1 obj.value_counts()
```

```
c    3  
a    3  
b    2  
d    1  
dtype: int64
```

# Unique Values, Value Counts, and Membership

```
1 pd.value_counts(obj.values, sort=False)
```

```
a    3  
b    2  
c    3  
d    1  
dtype: int64
```

# Unique Values, Value Counts, and Membership

```
1 obj
2 mask = obj.isin(['b', 'c'])
3 mask
4 obj[mask]
```

```
0    c
5    b
6    b
7    c
8    c
dtype: object
```

# Unique Values, Value Counts, and Membership

```
1 to_match = pd.Series(['c', 'a', 'b', 'b', 'c', 'a'])  
2 unique_vals = pd.Series(['c', 'b', 'a'])  
3 pd.Index(unique_vals).get_indexer(to_match)
```

```
array([0, 2, 1, 1, 0, 2], dtype=int64)
```

# Unique Values, Value Counts, and Membership

```
1 data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],  
2                      'Qu2': [2, 3, 1, 2, 3],  
3                      'Qu3': [1, 5, 2, 4, 4]})  
4 data
```

	Qu1	Qu2	Qu3
0	1	2	1
1	3	3	5
2	4	1	2
3	3	2	4
4	4	3	4

# Unique Values, Value Counts, and Membership

```
1 result = data.apply(pd.value_counts).fillna(0)
2 result
```

	Qu1	Qu2	Qu3
1	1.0	1.0	1.0
2	0.0	2.0	1.0
3	2.0	2.0	0.0
4	2.0	0.0	2.0
5	0.0	0.0	1.0

```
pd.options.display.max_rows = PREVIOUS_MAX_ROWS
```

# **Data Loading, Storage,**

# Reading and Writing Data in Text Format

```
1 !type ex1.csv
```

```
a,b,c,d,message  
1,2,3,4,hello  
5,6,7,8,world  
9,10,11,12,foo
```

```
1 df = pd.read_csv('ex1.csv')  
2 df
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

# Reading and Writing Data in Text Format

```
1 pd.read_csv('ex1.csv', sep=',')
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

# Reading and Writing Data in Text Format

```
1 !type ex2.csv
```

```
1,2,3,4,hello  
5,6,7,8,world  
9,10,11,12,foo
```

```
1 pd.read_csv('ex2.csv', header=None)
```

	0	1	2	3	4	
0	1	2	3	4	hello	
1	5	6	7	8	world	
2	9	10	11	12	foo	

# Reading and Writing Data in Text Format

```
1 pd.read_csv('ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

# Reading and Writing Data in Text Format

```
1 names = ['a', 'b', 'c', 'd', 'message']  
2 pd.read_csv('ex2.csv', names=names, index_col='message')
```

	a	b	c	d
message				
<b>hello</b>	1	2	3	4
<b>world</b>	5	6	7	8
<b>foo</b>	9	10	11	12

# Reading and Writing Data in Text Format

```
1 parsed = pd.read_csv('csv_mindex.csv',  
2                         index_col=['key1', 'key2'])  
3 parsed
```

		value1	value2
	key1	key2	
one	a	1	2
	b	3	4
	c	5	6
	d	7	8
two	a	9	10
	b	11	12
	c	13	14
	d	15	16

# Reading and Writing Data in Text Format

```
1 list(open('ex3.txt'))
```

```
[ '           A           B           C\n',  
  'aaa -0.264438 -1.026059 -0.619500\n',  
  'bbb  0.927272  0.302904 -0.032399\n',  
  'ccc -0.264273 -0.386314 -0.217601\n',  
  'ddd -0.871858 -0.348382  1.100491\n']
```

# Reading and Writing Data in Text Format

```
1 result = pd.read_csv('ex3.txt', sep='\\s+')
2 result
```

	A	B	C
<b>aaa</b>	<u>-0.264438</u>	<u>-1.026059</u>	<u>-0.619500</u>
<b>bbb</b>	0.927272	0.302904	-0.032399
<b>ccc</b>	<u>-0.264273</u>	-0.386314	<u>-0.217601</u>
<b>ddd</b>	-0.871858	-0.348382	<u>1.100491</u>

# Reading and Writing Data in Text Format

```
1 pd.read_csv('ex4.csv', skiprows=[0, 2, 3])
```

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

# Reading and Writing Data in Text Format

```
1 result = pd.read_csv('ex5.csv')
2 result
3 pd.isnull(result)
```

	<b>something</b>	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>message</b>
<b>0</b>		False	False	False	False	False
<b>1</b>		False	False	False	True	False
<b>2</b>		False	False	False	False	False

# Reading and Writing Data in Text Format

```
1 result = pd.read_csv('ex5.csv', na_values=['NULL'])  
2 result
```

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

# Reading and Writing Data in Text Format

```
1 sentinels = {'message': ['foo', 'NA'], 'something': ['two']}
```

```
2 pd.read_csv('ex5.csv', na_values=sentinels)
```

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	NaN	5	6	NaN	8	world
2	three	9	10	11.0	12	NaN

# Reading Text Files in Pieces

```
1 pd.options.display.max_rows = 6
```

```
1 result = pd.read_csv('ex6.csv')
2 result
```

	one	two	three	four	key
0	0.467976	-0.038649	<a href="#">0.295344</a>	<a href="#">-1.824726</a>	L
1	-0.358893	1.404453	0.704965	<a href="#">-0.200638</a>	B
2	-0.501840	0.659254	<a href="#">-0.421691</a>	-0.057688	G
...	...	...	...	...	...
9997	<a href="#">0.523331</a>	0.787112	0.486066	<a href="#">1.093156</a>	K
9998	-0.362559	0.598894	<a href="#">-1.843201</a>	0.887292	G
9999	-0.096376	<a href="#">-1.012999</a>	-0.657431	-0.573315	0

# Reading Text Files in Pieces

```
1 pd.read_csv('ex6.csv', nrows=5)
```

	one	two	three	four	key
0	0.467976	-0.038649	<u>-0.295344</u>	<u>-1.824726</u>	L
1	-0.358893	1.404453	0.704965	<u>-0.200638</u>	B
2	-0.501840	0.659254	-0.421691	-0.057688	G
3	0.204886	1.074134	1.388361	-0.982404	R
4	0.354628	-0.133116	0.283763	-0.837063	Q

# Reading Text Files in Pieces

```
1 chunker = pd.read_csv('ex6.csv', chunkszie=1000)  
2 chunker
```

<pandas.io.parsers.TextFileReader at 0x1a5d4c8f668>

```
1 chunker = pd.read_csv('ex6.csv', chunkszie=1000)  
2  
3 tot = pd.Series([])  
4 for piece in chunker:  
5     tot = tot.add(piece['key'].value_counts(), fill_value=0)  
6  
7 tot = tot.sort_values(ascending=False)
```

# Reading Text Files in Pieces

```
1 tot[:10]
```

E 368.0

X 364.0

L 346.0

...

F 335.0

K 334.0

H 330.0

Length: 10, dtype: float64

pd.options.display.max\_rows = n

# Writing Data to Text Format

```
1 data = pd.read_csv('ex5.csv')
2 data
```

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

# Writing Data to Text Format

```
1 data.to_csv('out.csv')  
2 !type out.csv
```

,something,a,b,c,d,message  
0,one,1,2,3.0,4,  
1,two,5,6,,8,wor ld  
2,three,9,10,11.0,12,foo

# Writing Data to Text Format

```
1 import sys  
2 data.to_csv(sys.stdout, sep='|')
```

|something|a|b|c|d|message  
0|one|1|2|3.0|4|  
1|two|5|6| |8|wor|d  
2|three|9|10|11.0|12|foo

# Writing Data to Text Format

```
1 | data.to_csv(sys.stdout, na_rep='NULL')
```

```
,something,a,b,c,d,message
0,one,1,2,3.0,4,NULL
1,two,5,6,NULL,8,wor ld
2,three,9,10,11.0,12,foo
```

# Writing Data to Text Format

```
1 data.to_csv(sys.stdout, index=False, header=False)
```

```
one,1,2,3.0,4,  
two,5,6,,8,world  
three,9,10,11.0,12,foo
```

# Writing Data to Text Format

```
1 data.to_csv(sys.stdout, index=False, columns=['a', 'b', 'c'])
```

```
a,b,c  
1,2,3.0  
5,6,  
9,10,11.0
```

# Writing Data to Text Format

```
1 dates = pd.date_range('2000/1/1', periods=7)
2 ts = pd.Series(np.arange(7), index=dates)
3 ts.to_csv('tseries.csv', header=False)
4 !type tseries.csv
```

2000-01-01,0  
2000-01-02,1  
2000-01-03,2  
2000-01-04,3  
2000-01-05,4  
2000-01-06,5  
2000-01-07,6

# Working with Delimited Formats

```
1 !type ex7.csv
```

```
"a","b","c"  
"1","2","3"  
"1","2","3"
```

```
1 import csv  
2 f = open('ex7.csv')  
3  
4 reader = csv.reader(f)
```

# Working with Delimited Formats

```
1 for line in reader:  
2     print(line)
```

```
['a', 'b', 'c']  
['1', '2', '3']  
['1', '2', '3']
```

# Working with Delimited Formats

```
1 with open('ex7.csv') as f:  
2     lines = list(csv.reader(f))
```

```
1 header, values = lines[0], lines[1:]
```

```
1 data_dict = {h: v for h, v in zip(header, zip(*values))}  
2 data_dict
```

```
{'a': ('1', '1'), 'b': ('2', '2'), 'c': ('3', '3')}
```

# Binary Data Formats

```
1 frame = pd.read_csv('ex1.csv')
2 frame
3 frame.to_pickle('frame_pickle')
```

```
1 pd.read_pickle('frame_pickle')
```

a	b	c	d	message
---	---	---	---	---------

0	1	2	3	4	hello
---	---	---	---	---	-------

1	5	6	7	8	world
---	---	---	---	---	-------

2	9	10	11	12	foo
---	---	----	----	----	-----

# Reading Microsoft Excel Files

```
1 xlsx = pd.ExcelFile('ex1.xlsx')
```

```
1 pd.read_excel(xlsx, 'Sheet1')
```

	Unnamed: 0	a	b	c	d	message
0		0	1	2	3	4
1		1	5	6	7	8
2		2	9	10	11	12

conda install xlrd

# Reading Microsoft Excel Files

```
1 frame = pd.read_excel('ex1.xlsx', 'Sheet1')  
2 frame
```

Unnamed: 0	a	b	c	d	message
------------	---	---	---	---	---------

0	0	1	2	3	4	hello
---	---	---	---	---	---	-------

1	1	5	6	7	8	world
---	---	---	---	---	---	-------

2	2	9	10	11	12	foo
---	---	---	----	----	----	-----

# Reading Microsoft Excel Files

```
1 writer = pd.ExcelWriter('ex2.xlsx')
2 frame.to_excel(writer, 'Sheet1')
3 writer.save()
```

```
1 frame.to_excel('ex2.xlsx')
```

conda install openpyxl

# **Data Cleaning and Preparation**

# Handling Missing Data

```
1 string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])  
2 string_data  
3 string_data.isnull()
```

```
0    False  
1    False  
2     True  
3    False  
dtype: bool
```

# Handling Missing Data

```
1 string_data[0] = None  
2 string_data.isnull()
```

```
0    True  
1   False  
2    True  
3   False  
dtype: bool
```

# Filtering Out Missing Data

```
1 from numpy import nan as NA  
2 data = pd.Series([1, NA, 3.5, NA, 7])  
3 data.dropna()
```

```
0    1.0  
2    3.5  
4    7.0  
dtype: float64
```

```
1 data[data.notnull()]  
  
0    1.0  
2    3.5  
4    7.0  
dtype: float64
```

# Filtering Out Missing Data

```
1 data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],  
2                         [NA, NA, NA], [NA, 6.5, 3.]])  
3 cleaned = data.dropna()  
4 data  
5 cleaned
```

	0	1	2
--	---	---	---

<b>0</b>	1.0	6.5	3.0
----------	-----	-----	-----

# Filtering Out Missing Data

```
1 data.dropna(how='all')
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

# Filtering Out Missing Data

```
1 | data[4] = NA  
2 | data  
3 | data.dropna(axis=1, how='all')
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

# Filtering Out Missing Data

```
1 df = pd.DataFrame(np.random.randn(7, 3))
2 df.iloc[:4, 1] = NA
3 df.iloc[:2, 2] = NA
4 df
5 df.dropna()
6 df.dropna(thresh=2)
```

	0	1	2
2	0.092908	NaN	0.769023
3	1.246435	NaN	-1.296221
4	<a href="#">0.274992</a>	<a href="#">0.228913</a>	1.352917
5	0.886429	<a href="#">-2.001637</a>	-0.371843
6	<a href="#">1.669025</a>	<a href="#">-0.438570</a>	<a href="#">-0.539741</a>

# Filling In Missing Data

```
1 df.fillna(0)
```

	0	1	2
0	-0.204708	0.000000	0.000000
1	-0.555730	0.000000	0.000000
2	0.092908	0.000000	0.769023
3	1.246435	0.000000	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

# Filling In Missing Data

```
1 df.fillna({1: 0.5, 2: 0})
```

	0	1	2
0	-0.204708	0.500000	0.000000
1	-0.555730	0.500000	0.000000
2	0.092908	0.500000	0.769023
3	1.246435	0.500000	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

# Filling In Missing Data

```
1 _ = df.fillna(0, inplace=True)
2 df
```

	0	1	2
0	-0.204708	0.000000	0.000000
1	-0.555730	0.000000	0.000000
2	0.092908	0.000000	0.769023
3	1.246435	0.000000	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

# Filling In Missing Data

```
1 df = pd.DataFrame(np.random.randn(6, 3))
2 df.iloc[2:, 1] = NA
3 df.iloc[4:, 2] = NA
4 df
5 df.fillna(method='ffill')
6 df.fillna(method='ffill', limit=2)
```

	0	1	2
0	0.476985	<a href="#">3.248944</a>	<a href="#">-1.021228</a>
1	-0.577087	0.124121	0.302614
2	<a href="#">0.523772</a>	0.124121	1.343810
3	-0.713544	0.124121	<a href="#">-2.370232</a>
4	<a href="#">-1.860761</a>	NaN	<a href="#">-2.370232</a>
5	-1.265934	NaN	<a href="#">-2.370232</a>

# Filling In Missing Data

```
1 data = pd.Series([1., NA, 3.5, NA, 7])  
2 data.fillna(data.mean())
```

```
0    1.000000  
1    3.833333  
2    3.500000  
3    3.833333  
4    7.000000  
dtype: float64
```

# Removing Duplicates

```
1 data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
2                      'k2': [1, 1, 2, 3, 3, 4, 4]})
3 data
```

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

# Removing Duplicates

```
1 data.duplicated()
```

```
0    False  
1    False  
2    False  
3    False  
4    False  
5    False  
6    True  
dtype: bool
```

```
1 data.drop_duplicates()
```

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

# Removing Duplicates

```
1 data['v1'] = range(7)
2 data.drop_duplicates(['k1'])
```

	k1	k2	v1
0	one	1	0
1	two	1	1

# Removing Duplicates

```
1 data.drop_duplicates(['k1', 'k2'], keep='last')
```

	k1	k2	v1
<b>0</b>	one	1	0
<b>1</b>	two	1	1
<b>2</b>	one	2	2
<b>3</b>	two	3	3
<b>4</b>	one	3	4
<b>6</b>	two	4	6

# Transforming Data Using a Function or Mapping

```
1 data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',
2                               'Pastrami', 'corned beef', 'Bacon',
3                               'pastrami', 'honey ham', 'nova lox'],
4                               'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})  
5 data
```

	food	ounces					
0	bacon	4.0	4	corned beef	7.5	8	nova lox
1	pulled pork	3.0	5	Bacon	8.0		
2	bacon	12.0	6	pastrami	3.0		
3	Pastrami	6.0	7	honey ham	5.0		

# Transforming Data Using a Function or Mapping

```
1 meat_to_animal = {  
2     'bacon': 'pig',  
3     'pulled pork': 'pig',  
4     'pastrami': 'cow',  
5     'corned beef': 'cow',  
6     'honey ham': 'pig',  
7     'nova lox': 'salmon'  
8 }
```

```
1 lowercased = data['food'].str.lower()  
lowercased  
3 data['animal'] = lowercased.map(meat_to_animal)  
4 data
```

# Transforming Data Using a Function or Mapping

	food	ounces	animal				
0	bacon	4.0	pig	5	Bacon	8.0	pig
1	pulled pork	3.0	pig	6	pastrami	3.0	cow
2	bacon	12.0	pig	7	honey ham	5.0	pig
3	Pastrami	6.0	cow	8	nova lox	6.0	salmon
4	corned beef	7.5	cow				

# Transforming Data Using a Function or Mapping

```
1 data['food'].map(lambda x: meat_to_animal[x.lower()])
```

```
0    pig
1    pig
2    pig
3    cow
4    cow
5    pig
6    cow
7    pig
8  salmon
Name: food, dtype: object
```

# Replacing Values

```
1 data = pd.Series([1., -999., 2., -999., -1000., 3.])  
2 data
```

```
0      1.0  
1    -999.0  
2      2.0  
3    -999.0  
4   -1000.0  
5      3.0  
dtype: float64
```

# Replacing Values

```
1 | data.replace(-999, np.nan)
```

```
0      1.0
1      NaN
2      2.0
3      NaN
4    -1000.0
5      3.0
dtype: float64
```

# Replacing Values

```
1 data.replace([-999, -1000], np.nan)
```

```
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

# Replacing Values

```
1 data.replace([-999, -1000], [np.nan, 0])
```

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

# Replacing Values

```
1 data.replace({-999: np.nan, -1000: 0})
```

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

# Renaming Axis Indexes

```
1 data = pd.DataFrame(np.arange(12).reshape((3, 4)),  
2                     index=['Ohio', 'Colorado', 'New York'],  
3                     columns=['one', 'two', 'three', 'four'])
```

```
1 transform = lambda x: x[:4].upper()  
2 data.index.map(transform)
```

```
Index(['OHIO', 'COLO', 'NEW '], dtype='object')
```

# Renaming Axis Indexes

```
1 data.index = data.index.map(transform)  
2 data
```

	one	two	three	four
OHIO	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

# Renaming Axis Indexes

```
1 data.rename(index=str.title, columns=str.upper)
```

	ONE	TWO	THREE	FOUR
Ohio	0	1	2	3
Colo	4	5	6	7
New	8	9	10	11

# Renaming Axis Indexes

```
1 data.rename(index={'OHIO': 'INDIANA'},  
2             columns={'three': 'peekaboo'})
```

	one	two	peekaboo	four
INDIANA	0	1		3
COLO	4	5		7
NEW	8	9		11

# Renaming Axis Indexes

```
1 data.rename(index={'OHIO': 'INDIANA'}, inplace=True)
2 data
```

	one	two	three	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

# Discretization and Binning

```
1 ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

```
1 bins = [18, 25, 35, 60, 100]
2 cats = pd.cut(ages, bins)
3 cats
```

```
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60], (35, 60], (25, 35])
```

Length: 12

Categories (4, interval[int64]): [(18, 25] < (25, 35] < (35, 60] < (60, 100])

# Discretization and Binning

```
1 cats.codes  
2 cats.categories  
3 pd.value_counts(cats)
```

```
(18, 25]      5  
(35, 60]      3  
(25, 35]      3  
(60, 100]     1  
dtype: int64
```

# Discretization and Binning

```
1 pd.cut(ages, [18, 26, 36, 61, 100], right=False)
```

```
[[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ..., [26, 36), [61, 100), [36, 61), [36, 61), [26, 36)]
```

```
Length: 12
```

```
Categories (4, interval[int64]): [[18, 26) < [26, 36) < [36, 61) < [61, 100]]
```

# Discretization and Binning

```
1 group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']  
2 pd.cut(ages, bins, labels=group_names)
```

```
[Youth, Youth, Youth, YoungAdult, Youth, ..., YoungAdult, Senior, MiddleAged, MiddleAged, YoungAdult]
```

```
Length: 12
```

```
Categories (4, object): [Youth < YoungAdult < MiddleAged < Senior]
```

# Discretization and Binning

```
1 data = np.random.rand(20)
2 pd.cut(data, 4, precision=2)
```

```
[(0.34, 0.55], (0.34, 0.55], (0.76, 0.97], (0.76, 0.97], (0.34, 0.55], ..., (0.34, 0.55], (0.34, 0.55], (0.55, 0.76], (0.34, 0.55], (0.12, 0.34)]
```

Length: 20

Categories (4, interval[float64]): [(0.12, 0.34] < (0.34, 0.55] < (0.55, 0.76] < (0.76, 0.97)]

# Discretization and Binning

```
1 data = np.random.randn(1000) # Normally distributed  
2 cats = pd.qcut(data, 4) # Cut into quartiles  
3 cats  
4 pd.value_counts(cats)
```

```
(0.62, 3.928]                250  
(-0.0265, 0.62]               250  
(-0.68, -0.0265]              250  
(-2.949999999999997, -0.68]  250  
dtype: int64
```

# Discretization and Binning

```
1 pd.qcut(data, [0, 0.1, 0.5, 0.9, 1.])
```

```
[(-0.0265, 1.286], (-0.0265, 1.286], (-1.187, -0.0265], (-0.0265, 1.286], (-0.0265,  
1.286], ..., (-1.187, -0.0265], (-1.187, -0.0265], (-2.9499999999999997, -1.187], (-  
0.0265, 1.286], (-1.187, -0.0265])
```

Length: 1000

Categories (4, interval[float64]): [(-2.9499999999999997, -1.187] < (-1.187, -0.0265]  
< (-0.0265, 1.286] < (1.286, 3.928)]

# Detecting and Filtering Outliers

```
1 data = pd.DataFrame(np.random.randn(1000, 4))  
2 data.describe()
```

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.049091	0.026112	-0.002544	-0.051827
std	0.996947	1.007458	0.995232	0.998311
min	-3.645860	-3.184377	-3.745356	-3.428254
25%	-0.599807	-0.612162	-0.687373	-0.747478
50%	0.047101	-0.013609	-0.022158	-0.088274
75%	0.756646	0.695298	0.699046	0.623331
max	2.653656	3.525865	2.735527	3.366626

# Detecting and Filtering Outliers

```
1 col = data[2]  
2 col[np.abs(col) > 3]
```

```
41    -3.399312  
136    -3.745356  
Name: 2, dtype: float64
```

# Detecting and Filtering Outliers

```
1 data[(np.abs(data) > 3).any(1)]
```

	0	1	2	3					
41	0.457246	-0.025907	<u>-3.399312</u>	-0.974657	635	-0.578093	<u>0.193299</u>	1.397822	<u>3.366626</u>
60	<u>1.951312</u>	<u>3.260383</u>	0.963301	1.201206	782	<u>-0.207434</u>	3.525865	<u>0.283070</u>	<u>0.544635</u>
136	0.508391	<u>-0.196713</u>	-3.745356	-1.520113	803	-3.645860	<u>0.255475</u>	<u>-0.549574</u>	<u>-1.907459</u>
235	<u>-0.242459</u>	-3.056990	<u>1.918403</u>	-0.578828					
258	0.682841	<u>0.326045</u>	<u>0.425384</u>	-3.428254					
322	<u>1.179227</u>	<u>-3.184377</u>	1.369891	<u>-1.074833</u>					
544	-3.548824	1.553205	<u>-2.186301</u>	1.277104					

# Detecting and Filtering Outliers

```
1 data[np.abs(data) > 3] = np.sign(data) * 3  
2 data.describe()
```

	0	1	2	3
count	<a href="#">1000.000000</a>	<a href="#">1000.000000</a>	<a href="#">1000.000000</a>	<a href="#">1000.000000</a>
mean	0.050286	0.025567	-0.001399	-0.051765
std	0.992920	<a href="#">1.004214</a>	0.991414	0.995761
min	-3.000000	-3.000000	-3.000000	-3.000000
25%	-0.599807	<a href="#">-0.612162</a>	-0.687373	-0.747478
50%	0.047101	-0.013609	-0.022158	-0.088274
75%	0.756646	0.695298	0.699046	<a href="#">0.623331</a>
max	<a href="#">2.653656</a>	3.000000	<a href="#">2.735527</a>	3.000000

# Detecting and Filtering Outliers

```
1 np.sign(data).head()
```

	0	1	2	3
<b>0</b>	-1.0	1.0	-1.0	1.0
<b>1</b>	1.0	-1.0	1.0	-1.0
<b>2</b>	1.0	1.0	1.0	-1.0
<b>3</b>	-1.0	-1.0	1.0	-1.0
<b>4</b>	-1.0	1.0	-1.0	-1.0

# Permutation and Random Sampling

```
1 df = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)))
2 sampler = np.random.permutation(5)
3 sampler
```

```
array([3, 1, 4, 2, 0])
```

```
1 df
2 df.take(sampler)
```

	0	1	2	3
3	12	13	14	15
1	4	5	6	7
4	16	17	18	19
2	8	9	10	11
0	0	1	2	3

# Permutation and Random Sampling

```
1 df.sample(n=3)
```

	0	1	2	3		
<b>3</b>	12	13	14	15		
<b>4</b>	16	17	18	19	4	4
<b>2</b>	8	9	10	11	1	7
					4	4
					2	-1
					0	5
					3	6
					1	7
					4	4
					0	5
					4	4
					dtype: int64	

```
1 choices = pd.Series([5, 7, -1, 6, 4])
2 draws = choices.sample(n=10, replace=True)
3 draws
```

# Computing Indicator/Dummy Variables

```
1 df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
2                     'data1': range(6)})
3 pd.get_dummies(df['key'])
```

	a	b	c
0	0	1	0
1	0	1	0
2	1	0	0
3	0	0	1
4	1	0	0
5	0	1	0

# Computing Indicator/Dummy Variables

```
1 dummies = pd.get_dummies(df['key'], prefix='key')
2 df_with_dummy = df[['data1']].join(dummies)
3 df_with_dummy
```

	data1	key_a	key_b	key_c
0	0	0	1	0
1	1	0	1	0
2	2	1	0	0
3	3	0	0	1
4	4	1	0	0
5	5	0	1	0

# Computing Indicator/Dummy Variables

```
1 mnames = ['movie_id', 'title', 'genres']
2 movies = pd.read_csv('movies.dat', sep='::',
3                      header=None, names=mnames, engine="python")
4 movies[:10]
```

	<b>movie_id</b>	<b>title</b>	<b>genres</b>
<b>0</b>	1	Toy Story (1995)	Animation Children's Comedy
<b>1</b>	2	Jumanji (1995)	Adventure Children's Fantasy
<b>2</b>	3	Grumpier Old Men (1995)	Comedy Romance
<b>3</b>	4	Waiting to Exhale (1995)	Comedy Drama

# Computing Indicator/Dummy Variables

```
1 all_genres = []
2 for x in movies.genres:
3     all_genres.extend(x.split('|'))
4 genres = pd.unique(all_genres)
```

```
1 genres
```

```
array(['Animation', "Children's", 'Comedy', 'Adventure', 'Fantasy',
       'Romance', 'Drama', 'Action', 'Crime', 'Thriller', 'Horror',
       'Sci-Fi', 'Documentary', 'War', 'Musical', 'Mystery', 'Film-Noir',
       'Western'], dtype=object)
```

# Computing Indicator/Dummy Variables

```
1 zero_matrix = np.zeros((len(movies), len(genres)))
2 dummies = pd.DataFrame(zero_matrix, columns=genres)
```

```
1 gen = movies.genres[0]
2 gen.split('||')
3 dummies.columns.get_indexer(gen.split('||'))
```

```
array([0, 1, 2], dtype=int64)
```

# Computing Indicator/Dummy Variables

```
1 for i, gen in enumerate(movies.genres):  
2     indices = dummies.columns.get_indexer(gen.split('|'))  
3     dummies.iloc[i, indices] = 1
```

```
1 movies_windic = movies.join(dummies.add_prefix('Genre_'))  
2 movies_windic.iloc[0]
```

# Computing Indicator/Dummy Variables

# Computing Indicator/Dummy Variables

```
1 np.random.seed(12345)
2 values = np.random.rand(10)
3 values
4 bins = [0, 0.2, 0.4, 0.6, 0.8, 1]
5 pd.get_dummies(pd.cut(values, bins))
```

# Computing Indicator/Dummy Variables

	(0.0, 0.2]	(0.2, 0.4]	(0.4, 0.6]	(0.6, 0.8]	(0.8, 1.0]
0	0	0	0	0	1
1	0	1	0	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	0	1	0	0
6	0	0	0	0	1
7	0	0	0	1	0
8	0	0	0	1	0
9	0	0	0	1	0

# String Object Methods

```
1 val = 'a,b, guido'  
2 val.split(,',')
```

```
['a', 'b', ' guido']
```

```
1 pieces = [x.strip() for x in val.split(,',')]  
2 pieces
```

```
['a', 'b', 'guido']
```

# String Object Methods

```
1 first, second, third = pieces  
2 first + '::' + second + '::' + third
```

'a::b::guido'

```
1 '::'.join(pieces)
```

'a::b::guido'

# String Object Methods

```
1 'guido' in val  
2 val.index(',', )  
3 val.find('::')
```

-1

```
1 val.index('::')
```

---

ValueError

Traceback (most recent call last)

```
<ipython-input-69-2c016e7367ac> in <module>  
----> 1 val.index('::')
```

ValueError: substring not found

# String Object Methods

```
1 val.count( , )
```

2

```
1 val.replace( , , ... )
2 val.replace( , , '' )
```

'ab guido'

# Regular Expressions

```
1 import re  
2 text = "foo      bar\t baz \tqux"  
3 re.split('\W+', text)
```

```
['foo', 'bar', 'baz', 'qux']
```

```
1 regex = re.compile('\W+')  
2 regex.split(text)
```

```
['foo', 'bar', 'baz', 'qux']
```

```
1 regex.findall(text)  
[ ' ', '\t', ' ', '\t' ]
```

# Regular Expressions

```
1 text = """Dave dave@google.com
2 Steve steve@gmail.com
3 Rob rob@gmail.com
4 Ryan ryan@yahoo.com
5 """
6 pattern = r'[A-Z0-9._%+-]+@[A-Z0-9.-]+\w.[A-Z]{2,4}'
7
8 # re.IGNORECASE makes the regex case-insensitive
9 regex = re.compile(pattern, flags=re.IGNORECASE)
```

```
1 regex.findall(text)
```

```
['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.com']
```

# Regular Expressions

```
1 m = regex.search(text)  
2 m  
3 text[m.start():m.end()]
```

'dave@google.com'

```
1 print(regex.match(text))
```

None

```
1 print(regex.sub('REDACTED', text))
```

Dave REDACTED  
Steve REDACTED  
Rob REDACTED  
Ryan REDACTED

# Regular Expressions

```
1 pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'
2 regex = re.compile(pattern, flags=re.IGNORECASE)
```

```
1 m = regex.match('wesm@bright.net')
2 m.groups()
```

```
('wesm', 'bright', 'net')
```

# Regular Expressions

```
1 | regex.findall(text)
```

```
[('dave', 'google', 'com'),  
 ('steve', 'gmail', 'com'),  
 ('rob', 'gmail', 'com'),  
 ('ryan', 'yahoo', 'com')]
```

```
1 | print(regex.sub(r'Username: \w1, Domain: \w2, Suffix: \w3', text))
```

Dave Username: dave, Domain: google, Suffix: com

Steve Username: steve, Domain: gmail, Suffix: com

Rob Username: rob, Domain: gmail, Suffix: com

Ryan Username: ryan, Domain: yahoo, Suffix: com

# Vectorized String Functions in pandas

```
1 data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',  
2        'Rob': 'rob@gmail.com', 'Wes': np.nan}  
3 data = pd.Series(data)  
4 data  
5 data.isnull()
```

```
Dave      False  
Steve     False  
Rob      False  
Wes       True  
dtype: bool
```

# Vectorized String Functions in pandas

```
1 data.str.contains('gmail')
```

```
Dave      False
Steve     True
Rob       True
Wes       NaN
dtype: object
```

# Vectorized String Functions in pandas

```
1 pattern
2 data.str.findall(pattern, flags=re.IGNORECASE)
```

```
Dave      [(dave, google, com)]
Steve     [(steve, gmail, com)]
Rob       [(rob, gmail, com)]
Wes          NaN
dtype: object
```

# Vectorized String Functions in pandas

```
1 matches = data.str.match(pattern, flags=re.IGNORECASE)
2 matches
```

```
Dave      True
Steve     True
Rob       True
Wes       NaN
dtype: object
```

# Vectorized String Functions in pandas

```
1 matches.str.get(1)
2 matches.str[0]
```

```
Dave      NaN
Steve     NaN
Rob       NaN
Wes       NaN
dtype: float64
```

```
1 data.str[:5]
```

```
Dave      dave@
Steve    steve
Rob      rob@g
Wes       NaN
dtype: object
```

# Data Wrangling

# Hierarchical Indexing

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

```
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
```

# Hierarchical Indexing

```
1 | data.index
```

```
MultiIndex(levels=[['a', 'b', 'c', 'd'], [1, 2, 3]],  
          codes=[[0, 0, 0, 1, 1, 2, 2, 3, 3], [0, 1, 2, 0, 2, 0, 1, 1, 2]])
```

```
1 | data['b']
```

```
2 | data['b':'c']
```

```
3 | data.loc[['b', 'd']]
```

```
b 1 -0.555730
```

```
3 1 1.965781
```

```
d 2 0.281746
```

```
3 2 0.769023
```

```
dtype: float64
```

# Hierarchical Indexing

```
1 data.loc[:, 2]
```

```
a    0.478943  
c    0.092908  
d    0.281746  
dtype: float64
```

```
1 data.unstack()
```

	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

# Hierarchical Indexing

```
1 data.unstack().stack()
```

```
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
```

# Hierarchical Indexing

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

Ohio            Colorado

  Green    Red      Green

	a	1	0	1	2
	b	1	2	3	4
		2	1	6	7
			2	9	10
					11

# Hierarchical Indexing

```
1 frame.index.names = ['key1', 'key2']
2 frame.columns.names = ['state', 'color']
3 frame
```

	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

# Hierarchical Indexing

```
1 frame['Ohio']
```

		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

```
1 frame.swaplevel('key1', 'key2')
```

	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

# Reordering and Sorting Levels

```
1 frame.sort_index(level=1)
2 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

```
1 frame.sum(level='key2')
```

state Ohio Colorado

color Green Red Green

key2

---

1	6	8	10
---	---	---	----

2	12	14	16
---	----	----	----

# Summary Statistics by Level

```
1 frame.sum(level='color', axis=1)
```

	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

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

	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

# Indexing with a DataFrame's columns

```
1 frame2 = frame.set_index(['c', 'd'])  
2 frame2
```

	a	b
c	d	
one	0	0
1	1	6
2	2	5
two	0	3
1	4	3
2	5	2
3	6	1

# Indexing with a DataFrame's columns

```
1 frame.set_index(['c', 'd'], drop=False)
```

	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	

# Indexing with a DataFrame's columns

```
1 frame2.reset_index()
```

	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

# Database-Style DataFrame Joins

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

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

# Database-Style DataFrame Joins

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

	key	data2
0	a	0
1	b	1
2	d	2

# Database-Style DataFrame Joins

```
1 pd.merge(df1, df2)
```

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

# Database-Style DataFrame Joins

```
1 pd.merge(df1, df2, on='key')
```

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

# Database-Style DataFrame Joins

```
1 df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
2                     'data1': range(7)})
3 df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
4                     'data2': range(3)})
5 pd.merge(df3, df4, left_on='lkey', right_on='rkey')
```

	lkey	data1	rkey	data2
0	b	0	b	1
1	b	1	b	1
2	b	6	b	1
3	a	2	a	0
4	a	4	a	0
5	a	5	a	0

# Database-Style DataFrame Joins

```
1 pd.merge(df1, df2, how='outer')
```

	key	data1	data2
0	b	0.0	1.0
1	b	1.0	1.0
2	b	6.0	1.0
3	a	2.0	0.0
4	a	4.0	0.0
5	a	5.0	0.0
6	c	3.0	NaN
7	d	NaN	2.0

# Database-Style DataFrame Joins

```
1 df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
2                     'data1': range(6)})
3 df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
4                     'data2': range(5)})
5 pd.merge(df1, df2, on='key', how='left')
```

	key	data1	data2
0	b	0	1.0
1	b	0	3.0
2	b	1	1.0
3	b	1	3.0
4	a	2	0.0
5	a	2	2.0
6	c	3	NaN
7	a	4	0.0
8	a	4	2.0
9	b	5	1.0
10	b	5	3.0

# Database-Style DataFrame Joins

```
1 pd.merge(df1, df2, how='inner')
```

	key	data1	data2
0	b	0	1
1	b	0	3
2	b	1	1
3	b	1	3
4	b	5	1
5	b	5	3
6	a	2	0
7	a	2	2
8	a	4	0
9	a	4	2

# Database-Style DataFrame Joins

```
1 left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
2                      'key2': ['one', 'two', 'one'],
3                      'lval': [1, 2, 3]})
4 right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
5                      'key2': ['one', 'one', 'one', 'two'],
6                      'rval': [4, 5, 6, 7]})
7 pd.merge(left, right, on=['key1', 'key2'], how='outer')
```

	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

# Database-Style DataFrame Joins

```
1 pd.merge(left, right, on='key1')
2 pd.merge(left, right, on='key1', suffixes=('_left', '_right'))
```

	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

# Merging on Index

```
1 left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
2                         'value': range(6)})
3 right1 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
4 pd.merge(left1, right1, left_on='key', right_index=True)
```

	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

# Merging on Index

```
1 pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
```

	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

# Merging on Index

```
1 left = pd.DataFrame({'key1': ['Ohio', 'Ohio', 'Ohio',
2                               'Nevada', 'Nevada'],
3                         'key2': [2000, 2001, 2002, 2001, 2002],
4                         'data': np.arange(5.)})
5 left
```

	key1	key2	data
0	Ohio	2000	0.0
1	Ohio	2001	1.0
2	Ohio	2002	2.0
3	Nevada	2001	3.0
4	Nevada	2002	4.0

# Merging on Index

```
1 righth = pd.DataFrame(np.arange(12).reshape((6, 2)),  
2                         index=[['Nevada', 'Nevada', 'Ohio', 'Ohio',  
3                           'Ohio', 'Ohio'],  
4                           [2001, 2000, 2000, 2000, 2001, 2002]],  
5                         columns=['event1', 'event2'])  
6 righth
```

		event1	event2
<b>Nevada</b>	<b>2001</b>	0	1
	<b>2000</b>	2	3
<b>Ohio</b>	<b>2000</b>	4	5
	<b>2000</b>	6	7
	<b>2001</b>	8	9
	<b>2002</b>	10	11

# Merging on Index

```
1 pd.merge(left, right, left_on=['key1', 'key2'], right_index=True)
```

	key1	key2	data	event1	event2
0	Ohio	2000	0.0	4	5
0	Ohio	2000	0.0	6	7
1	Ohio	2001	1.0	8	9
2	Ohio	2002	2.0	10	11
3	Nevada	2001	3.0	0	1

# Merging on Index

```
1 pd.merge(left, right, left_on=['key1', 'key2'],
2           right_index=True, how='outer')
```

	key1	key2	data	event1	event2
0	Ohio	2000	0.0	4.0	5.0
0	Ohio	2000	0.0	6.0	7.0
1	Ohio	2001	1.0	8.0	9.0
2	Ohio	2002	2.0	10.0	11.0
3	Nevada	2001	3.0	0.0	1.0
4	Nevada	2002	4.0	NaN	NaN
4	Nevada	2000	NaN	2.0	3.0

# Merging on Index

```
1 left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],  
2                         index=['a', 'c', 'e'],  
3                         columns=['Ohio', 'Nevada'])  
4 right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14.]],  
5                         index=['b', 'c', 'd', 'e'],  
6                         columns=['Missouri', 'Alabama'])  
7 pd.merge(left2, right2, how='outer', left_index=True, right_index=True)
```

	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

# Merging on Index

```
1 left2.join(right2, how='outer')
```

	<b>Ohio</b>	<b>Nevada</b>	<b>Missouri</b>	<b>Alabama</b>
<b>a</b>	1.0	2.0	NaN	NaN
<b>b</b>	NaN	NaN	7.0	8.0
<b>c</b>	3.0	4.0	9.0	10.0
<b>d</b>	NaN	NaN	11.0	12.0
<b>e</b>	5.0	6.0	13.0	14.0

# Merging on Index

```
1 left1.join(right1, on='key')
```

	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

# Merging on Index

```
1 another = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],  
2                               index=['a', 'c', 'e', 'f'],  
3                               columns=['New York', 'Oregon'])  
4 another  
5 left2.join([right2, another])
```

	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

# Merging on Index

```
1 left2.join([right2, another], how='outer')
```

	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

```
1 arr = np.arange(12).reshape((3, 4))  
2 arr  
3 np.concatenate([arr, arr], axis=1)
```

```
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]])
```

```
1 s1 = pd.Series([0, 1], index=['a', 'b'])  
2 s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])  
3 s3 = pd.Series([5, 6], index=['f', 'g'])
```

# Concatenating Along an Axis

```
1 pd.concat([s1, s2, s3])
```

```
a    0  
b    1  
c    2  
d    3  
e    4  
f    5  
g    6  
dtype: int64
```

# Concatenating Along an Axis

```
1 s4 = pd.concat([s1, s3])
2 s4
3 pd.concat([s1, s4], axis=1, sort=True)
```

	0	1
<b>a</b>	0.0	0
<b>b</b>	1.0	1
<b>f</b>	NaN	5
<b>g</b>	NaN	6

# Concatenating Along an Axis

```
1 pd.concat([s1, s4], axis=1, join='inner')
```

	0	1
<hr/>		
a	0	0
b	1	1

# Concatenating Along an Axis

```
1 pd.concat([s1, s4], axis=1, join_axes=[['a', 'c', 'b', 'e']])
```

	0	1
<b>a</b>	0.0	0.0
<b>c</b>	NaN	NaN
<b>b</b>	1.0	1.0
<b>e</b>	NaN	NaN

# Concatenating Along an Axis

```
1 result = pd.concat([s1, s1, s3], keys=['one', 'two', 'three'])  
2 result
```

```
one    a    0  
      b    1  
two    a    0  
      b    1  
three   f    5  
        g    6  
dtype: int64
```

# Concatenating Along an Axis

```
1 result.unstack()
```

	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

# Concatenating Along an Axis

```
1 pd.concat([s1, s2, s3], axis=1, keys=['one', 'two', 'three'], sort=True)
```

	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

# Concatenating Along an Axis

```
1 df1 = pd.DataFrame(np.arange(6).reshape(3, 2), index=['a', 'b', 'c'],  
2                      columns=['one', 'two'])  
3 df1
```

	one	two
a	0	1
b	2	3
c	4	5

# Concatenating Along an Axis

```
1 df2 = pd.DataFrame(5 + np.arange(4).reshape(2, 2), index=['a', 'c'],
2                     columns=['three', 'four'])
3 df2
```

	three	four
a	5	6
c	7	8

# Concatenating Along an Axis

```
1 pd.concat([df1, df2], axis=1, keys=['level1', 'level2'], sort=True)
```

	level1	level2		
	one	two	three	four
<b>a</b>	0	1	5.0	6.0
<b>b</b>	2	3	NaN	NaN
<b>c</b>	4	5	7.0	8.0

# Concatenating Along an Axis

```
1 pd.concat({'level1': df1, 'level2': df2}, axis=1, sort=True)
```

	level1	level2		
	one	two	three	four
<b>a</b>	0	1	5.0	6.0
<b>b</b>	2	3	NaN	NaN
<b>c</b>	4	5	7.0	8.0

# Concatenating Along an Axis

```
1 pd.concat([df1, df2], axis=1, keys=['level1', 'level2'],
2             names=['upper', 'lower'], sort=True)
```

	upper	level1	level2	
	lower	one	two	three four
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

# Concatenating Along an Axis

```
1 df1 = pd.DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c', 'd'])  
2 df1
```

	a	b	c	d
0	0.820211	-0.247423	0.302271	0.543980
1	-0.942369	-1.266383	0.937250	-0.720102
2	-1.593952	-0.375498	-0.958704	0.794336

# Concatenating Along an Axis

```
1 df2 = pd.DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])  
2 df2
```

	b	d	a
0	-1.605108	0.543710	0.925166
1	-1.469629	-0.399592	1.417343

# Concatenating Along an Axis

```
1 pd.concat([df1, df2], ignore_index=True, sort=True)
```

	a	b	c	d
0	0.820211	-0.247423	0.302271	0.543980
1	-0.942369	-1.266383	0.937250	-0.720102
2	-1.593952	-0.375498	-0.958704	0.794336
3	0.925166	-1.605108	NaN	0.543710
4	1.417343	-1.469629	NaN	-0.399592

# Combining Data with Overlap

```
1 a = pd.Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan],  
2                 index=['f', 'e', 'd', 'c', 'b', 'a'])  
3 a
```

```
f      NaN  
e      2.5  
d      NaN  
c      3.5  
b      4.5  
a      NaN  
dtype: float64
```

# Combining Data with Overlap

```
1 b = pd.Series(np.arange(len(a), dtype=np.float64),  
2                 index=['f', 'e', 'd', 'c', 'b', 'a'])  
3 b[-1] = np.nan  
4 b
```

```
f    0.0  
e    1.0  
d    2.0  
c    3.0  
b    4.0  
a    NaN  
dtype: float64
```

# Combining Data with Overlap

```
1 np.where(pd.isnull(a), b, a)
```

```
array([0. , 2.5, 2. , 3.5, 4.5, nan])
```

```
1 b[:-2].combine_first(a[2:])
```

```
a    NaN
b    4.5
c    3.0
d    2.0
e    1.0
f    0.0
dtype: float64
```

# Combining Data with Overlap

```
1 df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],  
2                      'b': [np.nan, 2., np.nan, 6.],  
3                      'c': range(2, 18, 4)})  
4 df1
```

	a	b	c
0	1.0	NaN	2
1	NaN	2.0	6
2	5.0	NaN	10
3	NaN	6.0	14

# Combining Data with Overlap

```
1 df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],  
2                      'b': [np.nan, 3., 4., 6., 8.]})  
3 df2
```

	a	b
0	5.0	NaN
1	4.0	3.0
2	NaN	4.0
3	3.0	6.0
4	7.0	8.0

# Combining Data with Overlap

```
1 df1.combine_first(df2)
```

	a	b	c
0	1.0	NaN	2.0
1	4.0	2.0	6.0
2	5.0	4.0	10.0
3	3.0	6.0	14.0
4	7.0	8.0	NaN

# Reshaping with Hierarchical Indexing

```
1 data = pd.DataFrame(np.arange(6).reshape((2, 3)),  
2                      index=pd.Index(['Ohio', 'Colorado'], name='state'),  
3                      columns=pd.Index(['one', 'two', 'three'],  
4                                         name='number'))  
5 data
```

number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

# Reshaping with Hierarchical Indexing

```
1 result = data.stack()  
2 result
```

```
state      number  
Ohio       one        0  
           two        1  
           three       2  
Colorado   one        3  
           two        4  
           three       5  
dtype: int32
```

```
1 result.unstack()
```

	number	one	two	three
state				
Ohio	0	1	2	
Colorado	3	4	5	

# Reshaping with Hierarchical Indexing

```
1 result.unstack(0)
2 result.unstack('state')
```

**state**      Ohio   Colorado

**number**

<b>one</b>	0	3
<b>two</b>	1	4
<b>three</b>	2	5

# Reshaping with Hierarchical Indexing

```
1 s1 = pd.Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])
2 s2 = pd.Series([4, 5, 6], index=['c', 'd', 'e'])
3 data2 = pd.concat([s1, s2], keys=['one', 'two'])
4 data2
```

```
one   a    0
      b    1
      c    2
      d    3
two   c    4
      d    5
      e    6
dtype: int64
```

# Reshaping with Hierarchical Indexing

```
1 data2.unstack()
```

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

```
1 data2.unstack().stack()
```

```
one    a    0.0
      b    1.0
      c    2.0
      d    3.0
two    c    4.0
      d    5.0
      e    6.0
dtype: float64
```

# Reshaping with Hierarchical Indexing

```
1 data2.unstack().stack(dropna=False)
```

```
one   a    0.0
      b    1.0
      c    2.0
      d    3.0
      e    NaN
two   a    NaN
      b    NaN
      c    4.0
      d    5.0
      e    6.0
dtype: float64
```

# Reshaping with Hierarchical Indexing

```
1 df = pd.DataFrame({'left': result, 'right': result + 5},  
2                 columns=pd.Index(['left', 'right'], name='side'))  
3 df
```

	side	left	right
state	number		
Ohio	one	0	5
	two	1	6
	three	2	7
Colorado	one	3	8
	two	4	9
	three	5	10

# Reshaping with Hierarchical Indexing

```
1 df.unstack('state')
```

side	left		right		
state	Ohio	Colorado	Ohio	Colorado	
number					
one	0		3	5	8
two	1		4	6	9
three	2		5	7	10

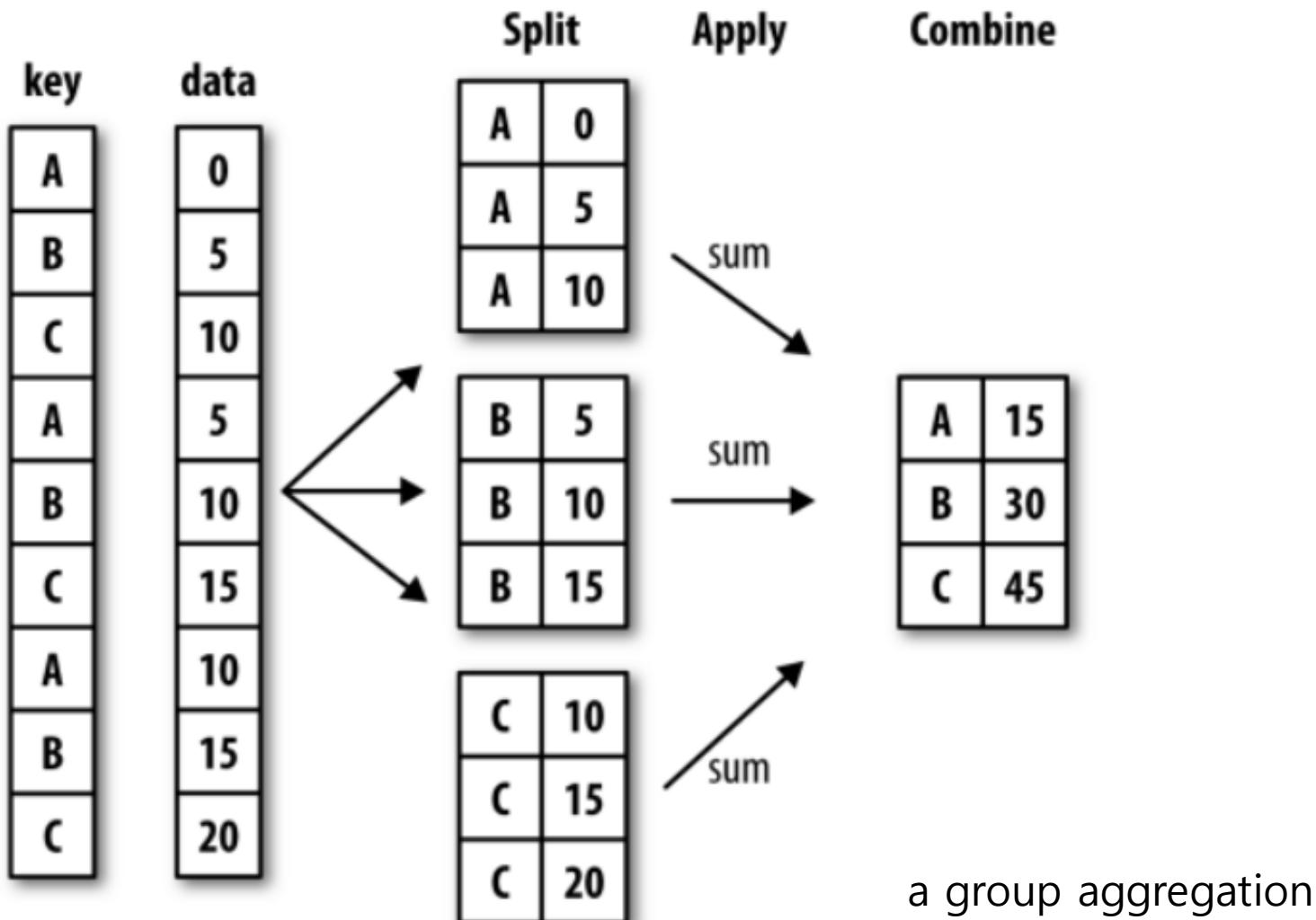
# Reshaping with Hierarchical Indexing

```
1 df.unstack('state').stack('side')
```

	state	Colorado	Ohio
number	side		
one	left	3	0
	right	8	5
two	left	4	1
	right	9	6
three	left	5	2
	right	10	7

# Data Aggregation and Group Operations

# GroupBy Mechanics



# GroupBy Mechanics

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

	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

# GroupBy Mechanics

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

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x0000012B1B3EE390>
```

```
1 grouped.mean()
```

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

# GroupBy Mechanics

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

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

# GroupBy Mechanics

```
1 means.unstack()
```

key2	one	two
key1		
a	0.880536	0.478943
b	<u>-0.519439</u>	<u>-0.555730</u>

# GroupBy Mechanics

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

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

# GroupBy Mechanics

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

		data1	data2
key1	key2		
a	one	0.880536	1.319920
	two	0.478943	0.092908
b	one	<a href="#">-0.519439</a>	<a href="#">0.281746</a>
	two	<a href="#">-0.555730</a>	0.769023

# GroupBy Mechanics

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

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

# Iterating Over Groups

```
1 for name, group in df.groupby('key1'):
2     print(name)
3     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

# Iterating Over Groups

```
1 for (k1, k2), group in df.groupby(['key1', 'key2']):  
2     print((k1, k2))  
3     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
```

# Iterating Over Groups

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

	<b>key1</b>	<b>key2</b>	<b>data1</b>	<b>data2</b>
<b>2</b>	b	one	<u>-0.519439</u>	<u>0.281746</u>
<b>3</b>	b	two	<u>-0.555730</u>	0.769023

# Iterating Over Groups

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

```
1 for dtype, group in grouped:  
2     print(dtype)  
3     print(group)
```

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

# Selecting a Column or Subset of Columns

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

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

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

		data2
	key1	key2
a	one	1.319920
	two	0.092908
b	one	<u>0.281746</u>
	two	0.769023

# Selecting a Column or Subset of Columns

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

```
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

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

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

# Grouping with Dicts and Series

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

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

	blue	red
<b>Joe</b>	0.503905	<a href="#">1.063885</a>
<b>Steve</b>	1.297183	-1.553778
<b>Wes</b>	<a href="#">-1.021228</a>	<a href="#">-1.116829</a>
<b>Jim</b>	<a href="#">0.524712</a>	<a href="#">1.770545</a>
<b>Travis</b>	<a href="#">-4.230992</a>	<a href="#">-2.405455</a>

# Grouping with Dicts and Series

```
1 map_series = pd.Series(mapping)
2 map_series
3 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

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

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

# Grouping with Functions

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

		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
	two	-0.713544	-0.831154	-2.370232	-1.860761	-0.860757

# Grouping by Index Levels

```
1 columns = pd.MultiIndex.from_arrays([[ 'US', 'US', 'US', 'KR', 'KR'],
2                                     [1, 3, 5, 1, 3]],
3                                     names=['cty', 'tenor'])
4 hier_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)
5 hier_df
```

cty	US			KR		
	tenor	1	3	5	1	3
0	0.560145	-1.265934	<a href="#">0.119827</a>	<a href="#">-1.063512</a>	<a href="#">0.332883</a>	
1	<a href="#">-2.359419</a>	<a href="#">-0.199543</a>	-1.541996	-0.970736	-1.307030	
2	<a href="#">0.286350</a>	0.377984	-0.753887	<a href="#">0.331286</a>	1.349742	
3	0.069877	<a href="#">0.246674</a>	-0.011862	<a href="#">1.004812</a>	1.327195	

# Grouping by Index Levels

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

cty	KR	US
0	2	3
1	2	3
2	2	3
3	2	3

# Data Aggregation

```
1 df
2 grouped = df.groupby('key1')
3 grouped['data1'].quantile(0.9)
```

```
key1
a    1.668413
b   -0.523068
Name: data1, dtype: float64
```

# Data Aggregation

```
1 def peak_to_peak(arr):  
2     return arr.max() - arr.min()  
3 grouped.agg(peak_to_peak)
```

	data1	data2
--	-------	-------

<b>key1</b>		
-------------	--	--

<b>a</b>	<a href="#">2.170488</a>	1.300498
----------	--------------------------	----------

<b>b</b>	0.036292	0.487276
----------	----------	----------

# Data Aggregation

```
1 grouped.describe()
```

key1	data1					data2			
	count	mean	std	min	25%	50%	75%	max	count
a	3.0	0.746672	<a href="#">1.109736</a>	<a href="#">-0.204708</a>	0.137118	0.478943	1.222362	<a href="#">1.965781</a>	3.0
b	2.0	<a href="#">-0.537585</a>	0.025662	<a href="#">-0.555730</a>	<a href="#">-0.546657</a>	<a href="#">-0.537585</a>	<a href="#">-0.528512</a>	<a href="#">-0.519439</a>	2.0

<

# Column-Wise and Multiple Function Application

```
1 tips = pd.read_csv('tips.csv')
2 # Add tip percentage of total bill
3 tips['tip_pct'] = tips['tip'] / tips['total_bill']
4 tips[:6]
```

	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	<u>0.160542</u>
2	21.01	3.50	No	Sun	Dinner	3	<u>0.166587</u>
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	<u>0.186240</u>

# Column-Wise and Multiple Function Application

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

```
1 grouped_pct = grouped['tip_pct']
2 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
```

# Column-Wise and Multiple Function Application

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

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

# Column-Wise and Multiple Function Application

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

		foo	bar
	day	smoker	
Fri		No	0.151650 0.028123
		Yes	<a href="#">0.174783</a> 0.051293
Sat		No	0.158048 0.039767
		Yes	0.147906 0.061375
Sun		No	<a href="#">0.160113</a> 0.042347
		Yes	<a href="#">0.187250</a> 0.154134
Thur		No	<a href="#">0.160298</a> 0.038774
		Yes	<a href="#">0.163863</a> 0.039389

# Column-Wise and Multiple Function Application

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

# Column-Wise and Multiple Function Application

	day	smoker	tip_pct			total_bill		
			count	mean	max	count	mean	max
Fri	No	4	0.151650	<a href="#">0.187735</a>	4	<a href="#">18.420000</a>	22.75	
	Yes	15	<a href="#">0.174783</a>	<a href="#">0.263480</a>	15	<a href="#">16.813333</a>	40.17	
Sat	No	45	0.158048	<a href="#">0.291990</a>	45	<a href="#">19.661778</a>	48.33	
	Yes	42	0.147906	<a href="#">0.325733</a>	42	<a href="#">21.276667</a>	50.81	
Sun	No	57	<a href="#">0.160113</a>	<a href="#">0.252672</a>	57	<a href="#">20.506667</a>	48.17	
	Yes	19	<a href="#">0.187250</a>	0.710345	19	<a href="#">24.120000</a>	45.35	
Thur	No	45	<a href="#">0.160298</a>	<a href="#">0.266312</a>	45	<a href="#">17.113111</a>	41.19	
	Yes	17	0.163863	0.241255	17	19.190588	43.11	

# Column-Wise and Multiple Function Application

```
1 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

# Column-Wise and Multiple Function Application

```
1 ftuples = [('Average', 'mean'), ('Deviation', np.var)]  
2 grouped['tip_pct', 'total_bill'].agg(ftuples)
```

# Column-Wise and Multiple Function Application

	day	smoker	tip_pct		total_bill	
			Average	Deviation	Average	Deviation
	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

# Column-Wise and Multiple Function Application

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

# Column-Wise and Multiple Function Application

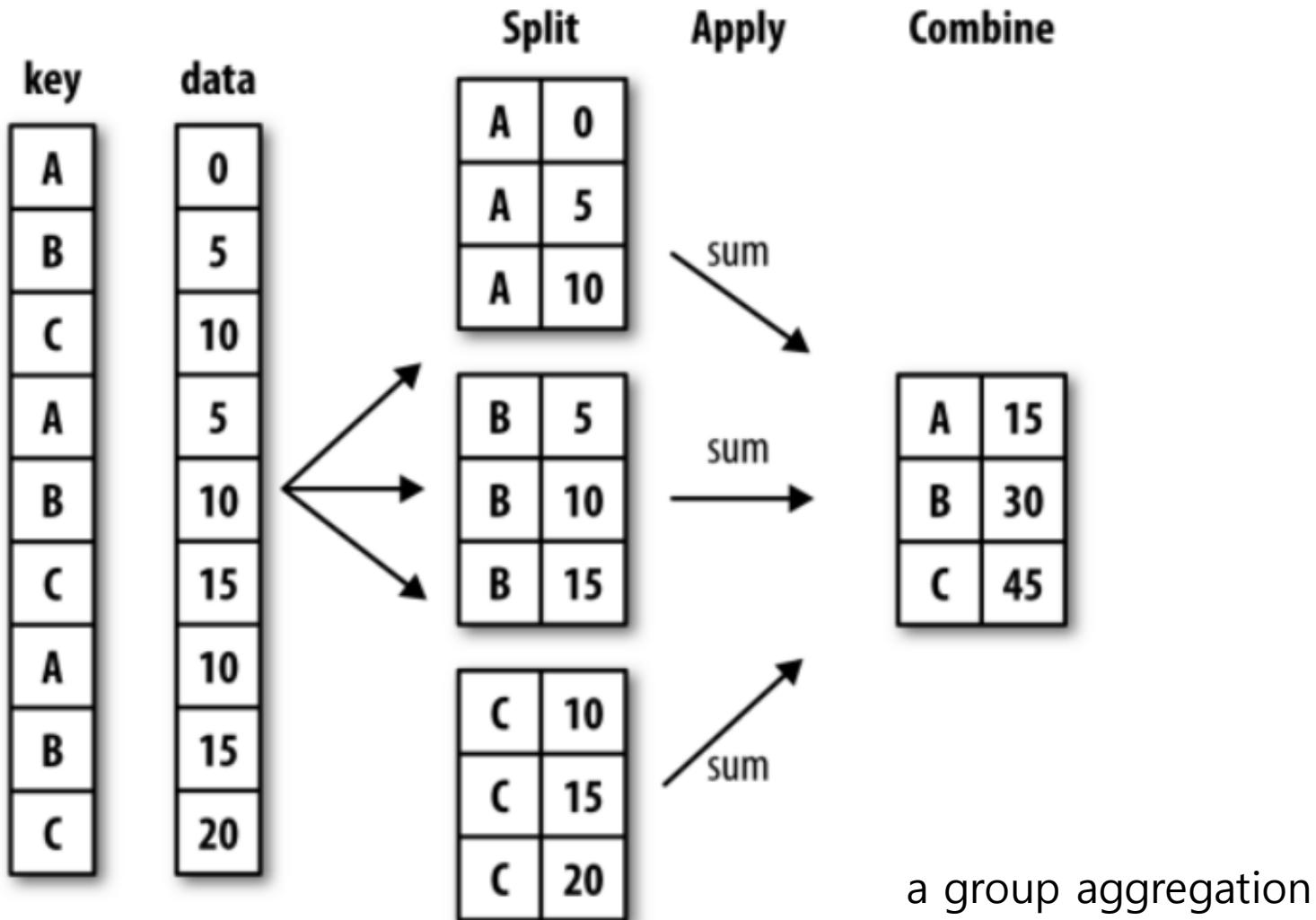
	day	smoker	tip_pct				size
			min	max	mean	std	
Fri	No	0.120385	0.187735	0.151650	0.028123	9	
	Yes	0.103555	0.263480	0.174783	0.051293	31	
Sat	No	0.056797	0.291990	0.158048	0.039767	115	
	Yes	0.035638	0.325733	0.147906	0.061375	104	
Sun	No	0.059447	0.252672	0.160113	0.042347	167	
	Yes	0.065660	0.710345	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

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

	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

# General split-apply-combine



# General split-apply-combine

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

	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

# General split-apply-combine

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

		total_bill	tip	smoker	day	time	size	tip_pct
	smoker							
1	No	88	24.71	5.85	No	Thur	Lunch	2 0.236746
2		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

# General split-apply-combine

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

			total_bill	tip	smoker	day	time	size	tip_pct	
smoker	day									
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	

# General split-apply-combine

```
1 result = tips.groupby('smoker')['tip_pct'].describe()  
2 result  
3 result.unstack('smoker')
```

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

dtype: float64

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

# Suppressing the Group Keys

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

	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

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

# Quantile and Bucket Analysis

```
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]): [(-2.956, -1.23] < (-1.23, 0.489] < (0.489, 2.208] < (2.208, 3.928)]

# Quantile and Bucket Analysis

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

	count	max	mean	min
--	-------	-----	------	-----

	count	max	mean	min
data1				

(-2.956, -1.23]	95.0	1.670835	-0.039521	-3.399312
-----------------	------	----------	-----------	-----------

(-1.23, 0.489]	598.0	3.260383	-0.002051	-2.989741
----------------	-------	----------	-----------	-----------

(0.489, 2.208]	297.0	2.954439	0.081822	-3.745356
----------------	-------	----------	----------	-----------

(2.208, 3.928]	10.0	1.765640	0.024750	-1.929776
----------------	------	----------	----------	-----------

# Quantile and Bucket Analysis

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

	count	max	mean	min
--	-------	-----	------	-----

**data1**

0	100.0	1.670835	-0.049902	-3.399312
---	-------	----------	-----------	-----------

1	100.0	2.628441	0.030989	-1.950098
---	-------	----------	----------	-----------

2	100.0	2.527939	-0.067179	-2.925113
---	-------	----------	-----------	-----------

3	100.0	3.260383	0.065713	-2.315555
---	-------	----------	----------	-----------

4	100.0	2.074345	-0.111653	-2.047939
---	-------	----------	-----------	-----------

5	100.0	2.184810	0.052130	-2.989741
---	-------	----------	----------	-----------

6	100.0	2.458842	-0.021489	-2.223506
---	-------	----------	-----------	-----------

7	100.0	2.954439	-0.026459	-3.056990
---	-------	----------	-----------	-----------

8	100.0	2.735527	0.103406	-3.745356
---	-------	----------	----------	-----------

9	100.0	2.377020	0.220122	-2.064111
---	-------	----------	----------	-----------