In [1]:
```
import pandas as pd
import numpy as np
df = pd.DataFrame({'key1':['a', 'a', 'b', 'b', 'a'],'key2':['one', 'two', 'one', '
df
```

Out[1]:

	key1	key2	data1	data2
0	a	one	-0.215747	-0.148936
1	a	two	-0.480787	0.889284
2	b	one	0.917291	-0.159969
3	b	two	-0.836880	-0.648607
4	a	one	-0.353333	0.283690

Data Aggregation and Group Operations A list or array of values that is the same length as the axis being grouped A value indicating a column name in a DataFrame A dict or Series giving a correspondence between the values on the axis being grouped and the group names A function to be invoked on the axis index or the individual labels in the index

In [6]:
```
group_df = df['data1'].groupby(df['key1'])
group_df
group_df.mean()
```

Out[6]:
```
key1
a   -0.349956
b    0.040206
Name: data1, dtype: float64
```

In [ ]:
```

```

In [8]:
```
means = df['data1'].groupby([df['key1'], df['key2']]).mean()
means
```

Out[8]:
```
key1  key2
a     one     0.042803
      two    -1.198112
b     one     0.144663
      two     0.635151
Name: data1, dtype: float64
```

In [9]:
```
means.unstack()
```

Out[9]:

key2	**one**	**two**
key1		
a	0.042803	-1.198112
b	0.144663	0.635151

In [12]:
```
states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
years = np.array([2005, 2005, 2006, 2005, 2006])
df['data1'].groupby([states, years]).mean()
```

Out[12]:
```
California  2005   -1.198112
            2006    0.144663
Ohio        2005    0.401797
            2006   -0.082838
Name: data1, dtype: float64
```

In [11]:
```
df.groupby('key1').mean()
```

Out[11]:

key1	data1	data2
a	-0.370835	0.777503
b	0.389907	-0.251653

In [13]:
```
df.groupby(['key1','key2']).mean()
```

Out[13]:

key1	key2	data1	data2
a	one	0.042803	0.768411
	two	-1.198112	0.795687
b	one	0.144663	0.421021
	two	0.635151	-0.924328

In [14]:
```
df.groupby(['key1','key2']).size()
```

Out[14]:
```
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

In [17]:
```
for name,group in df.groupby(['key1','key2']):
    print(name)
    print( group)
```

```
('a', 'one')
  key1 key2     data1     data2
0    a  one  0.168443  0.374125
4    a  one -0.082838  1.162696
('a', 'two')
  key1 key2     data1     data2
1    a  two -1.198112  0.795687
('b', 'one')
  key1 key2     data1     data2
2    b  one  0.144663  0.421021
('b', 'two')
  key1 key2     data1     data2
3    b  two  0.635151 -0.924328
```

In [18]:
```
chunk=dict(list(df.groupby('key1')))
chunk
```

Out[18]:
```
{'a':   key1 key2     data1     data2
 0    a  one   0.168443  0.374125
 1    a  two  -1.198112  0.795687
 4    a  one  -0.082838  1.162696,
 'b':   key1 key2     data1     data2
 2    b  one   0.144663  0.421021
 3    b  two   0.635151 -0.924328}
```

In [19]:
```
chunk['b']
```

Out[19]:

	key1	key2	data1	data2
2	b	one	0.144663	0.421021
3	b	two	0.635151	-0.924328

In [21]:
```
df.dtypes
```

Out[21]:
```
key1       object
key2       object
data1     float64
data2     float64
dtype: object
```

In [27]:
```
group_df1=df.groupby(df.dtypes)
```

In [28]:
```
group_df1
```

Out[28]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001B1FA58D1F0>
```

In [29]:
```
dict(list(group_df1))
```

Out[29]:
```
{}
```

In [31]:
```
df.groupby('key1')['data1'].mean()
```

Out[31]:
```
key1
a   -0.370835
b    0.389907
Name: data1, dtype: float64
```

In [9]:
```
p1=pd.DataFrame(np.random.randn(5,5),columns=['a','b','c','d','e'],index=['p','q',
```

In [10]:
```
p1
```

Out[10]:

	a	b	c	d	e
p	1.260514	0.056524	-0.158196	0.238014	0.152677
q	-0.327649	1.018723	0.825544	-0.081136	1.350238
r	-0.583995	-0.046794	-1.033420	-0.280814	0.579508
s	1.461915	-0.841728	0.335503	-0.453683	-0.477569
t	-0.360498	0.297372	-0.034374	-1.539890	1.266437

In [34]:
```
mapp={'a':'red','b': 'red','c':'blue','d':'blue','e':'red','f':'orange'}
by_col=p1.groupby(mapp,axis=1)
```

In [35]:
```
by_col.sum()
```

Out[35]:

	blue	red
p	0.663607	-1.730694
q	2.332881	3.403698
r	-0.412175	1.916062
s	-1.054664	-0.162910
t	1.366158	1.638960

In [36]:
```
by_col.count()
```

Out[36]:

	blue	red
p	2	3
q	2	3
r	2	3
s	2	3
t	2	3

In [11]:
```
p1.groupby(len).sum()
```

Out[11]:

	a	b	c	d	e
1	1.450287	0.484096	-0.064942	-2.11751	2.871292

In [ ]:
```
lk=['one','one','one','two','two']
p1.groupby([len,lk]).min()
```

In [14]:
```
cols=pd.MultiIndex.from_arrays([['US','US','US','JP','JP'],[1,3,5,1,3]],names=['ci
```

In [15]:
```
cols
```

Out[15]:
```
MultiIndex([('US', 1),
            ('US', 3),
            ('US', 5),
            ('JP', 1),
            ('JP', 3)],
           names=['city', 'tenor'])
```

In [16]:
```
hier_df=pd.DataFrame(np.random.randn(4,5),columns=cols)
```

In [43]:
```
hier_df
```

Out[43]:

city			US		JP
tenor	**1**	**3**	**5**	**1**	**3**
0	-0.797834	0.388431	-0.045545	0.445960	-1.232713
1	-0.148043	-0.550704	1.131381	0.382454	-0.793190
2	-0.232824	-1.697088	1.214961	-0.671577	0.668943
3	0.636425	-0.825803	0.873724	1.529052	0.723373

In [17]:
```
hier_df.groupby(level='city',axis=1).count()
```

Out[17]:

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

In [18]:
```
p1.describe()
```

Out[18]:

	a	b	c	d	e
count	5.000000	5.000000	5.000000	5.000000	5.000000
mean	0.290057	0.096819	-0.012988	-0.423502	0.574258
std	0.985366	0.669561	0.686915	0.674983	0.768986
min	-0.583995	-0.841728	-1.033420	-1.539890	-0.477569
25%	-0.360498	-0.046794	-0.158196	-0.453683	0.152677
50%	-0.327649	0.056524	-0.034374	-0.280814	0.579508
75%	1.260514	0.297372	0.335503	-0.081136	1.266437
max	1.461915	1.018723	0.825544	0.238014	1.350238

In [19]:
```
df
```

Out[19]:

	key1	key2	data1	data2
0	a	one	-0.215747	-0.148936
1	a	two	-0.480787	0.889284
2	b	one	0.917291	-0.159969
3	b	two	-0.836880	-0.648607
4	a	one	-0.353333	0.283690

In [20]:
```
group_df
```

Out[20]:
```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x000001F8FBB3CE50>
```

In [30]:
```
df['data1'].quantile(.7)
```

Out[30]:
```
-0.2432641628007493
```

In [29]:
```
df.describe()
```

Out[29]:

	data1	data2
count	5.000000	5.000000
mean	-0.193891	0.043092
std	0.662641	0.576733
min	-0.836880	-0.648607
25%	-0.480787	-0.159969
50%	-0.353333	-0.148936
75%	-0.215747	0.283690
max	0.917291	0.889284

In [55]:
```
np.arange(2,8,1)
```

Out[55]:
```
array([2, 3, 4, 5, 6, 7])
```

In [33]:
```
np.arange(12).reshape(3,4)
```

Out[33]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [31]:
```
np.matrix(4,3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-2c2d69ae43ce> in <module>
----> 1 np.matrix(4,3)

~\Documents\anaconda\lib\site-packages\numpy\matrixlib\defmatrix.py in __new__(subtype, data, dtype, copy)
    145
    146         # now convert data to an array
--> 147         arr = N.array(data, dtype=dtype, copy=copy)
    148         ndim = arr.ndim
    149         shape = arr.shape

TypeError: data type not understood
```

In [34]:
```
np.random.randn(12).reshape(3,4)
```

Out[34]:
```
array([[-0.40307808, -1.1418005 , -2.45671087, -0.53431459],
       [ 0.64376898, -0.83655377,  0.15400713,  1.15840427],
       [-0.90777839, -0.31423662, -0.93846373, -0.10318989]])
```

In [37]:
```
df.iloc[1]
```

Out[37]:
```
key1            a
key2          two
data1    -0.480787
data2     0.889284
Name: 1, dtype: object
```

In [63]:
```
i=0
def change(i):
   i=i+1
   return i
change(1)
print(i)
```

0

In [9]:
```
import seaborn as sns
import pandas as pd

tips['tip_pct'] = tips['tip'] / (tips['total_bill'] - tips['tip'])
tips.head()
```

Out[9]:

	total_bill	tip	sex	smoker	day	time	size	tip_pct
0	16.99	1.01	Female	No	Sun	Dinner	2	0.063204
1	10.34	1.66	Male	No	Sun	Dinner	3	0.191244
2	21.01	3.50	Male	No	Sun	Dinner	3	0.199886
3	23.68	3.31	Male	No	Sun	Dinner	2	0.162494
4	24.59	3.61	Female	No	Sun	Dinner	4	0.172069

In [7]:
```
tips=pd.read_csv("downloads/tips.csv")
tips
```

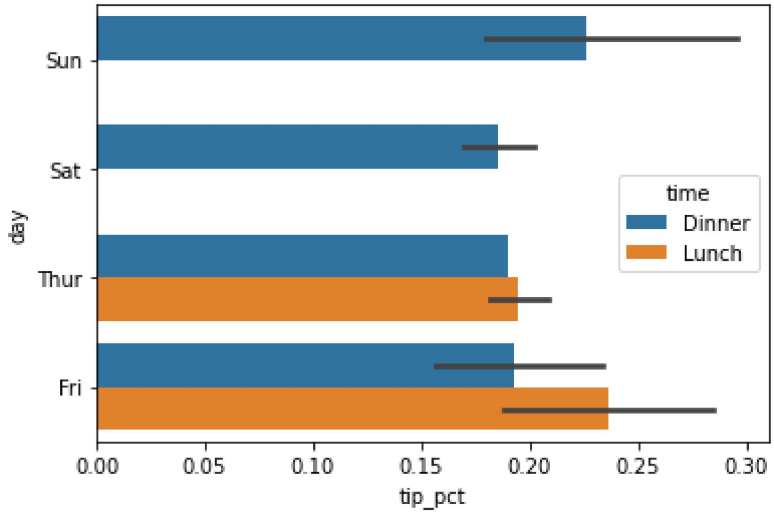
Out[7]:

	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 [10]:
```
sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h')
```

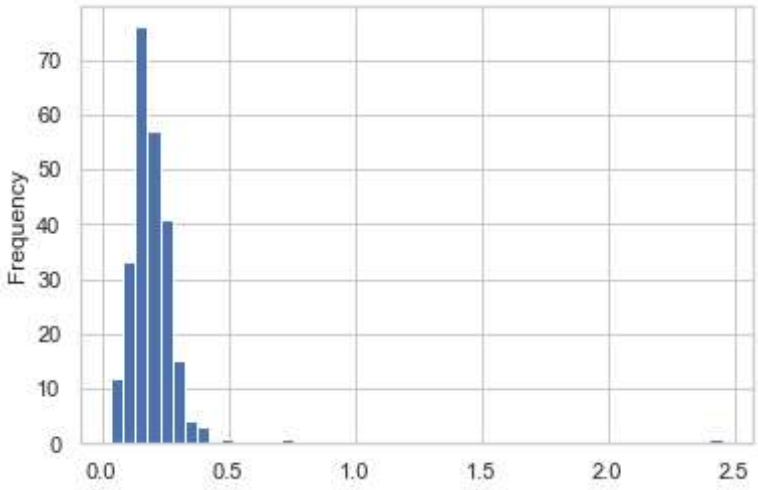
Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x274ff5d5610>
```



In [11]:
```
sns.set(style="whitegrid")
```

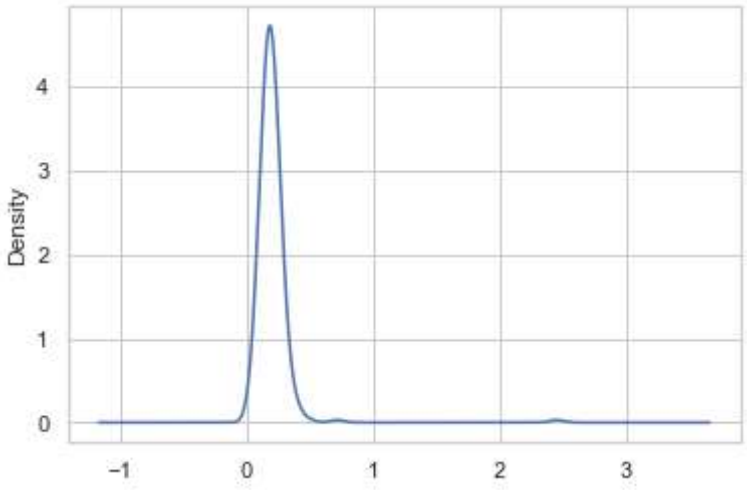
In [12]:
```
tips['tip_pct'].plot.hist(bins=50)
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x274ffda8c40>
```



In [13]:
```
tips['tip_pct'].plot.density()
```

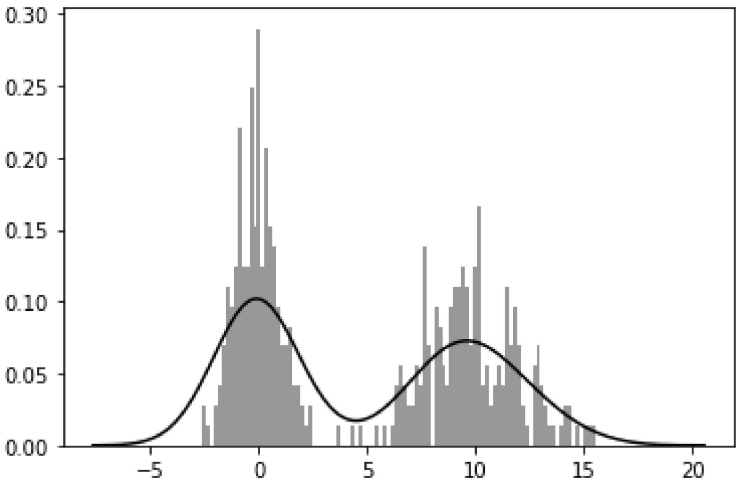
Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x274ffe8f310>
```



In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
comp1 = np.random.normal(0, 1, size=200)
```

```
comp2 = np.random.normal(10, 2, size=200)
values = pd.Series(np.concatenate([comp1, comp2]))
sns.distplot(values, bins=100, color='k')
```

Out[2]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2223ff245b0>
```



In [8]:
```
comp2
```

Out[8]:
```
array([ 8.7473117 , 11.13220889, 11.40786457, 11.80467278,  7.7558097 ,
        7.64109161, 11.29529845, 10.16241713,  8.61778926,  9.26558868,
        9.49031241,  8.92732675, 10.01818201, 10.58293488, 10.66785505,
        5.30063638,  8.83691078, 10.72406831,  8.74431021,  9.31134229,
        6.31069591,  8.5416601 , 12.10929238, 13.23708802,  7.46790596,
       10.4293611 ,  8.98209976, 11.55896486,  9.08665747, 10.68155155,
        9.39249634,  8.46401446, 11.04520645,  9.90023851, 13.09628207,
        8.39630547,  9.83028924,  8.43091102, 10.14483772, 10.27034895,
        6.77515614,  9.25462387, 10.49167982, 10.54679421, 10.88478053,
       11.8978995 ,  7.80254728, 10.9559278 ,  9.69587756,  9.61647952,
       11.17625119, 10.73227516,  7.79454529, 11.62256395, 12.91579495,
       12.08603303, 11.7056091 , 10.62805583,  9.98706398,  9.26483618,
       10.7139098 ,  8.80867026, 11.17146237, 10.33220013, 12.09216405,
        7.56096082,  8.4591409 ,  9.8557539 ,  9.37053753, 10.90982936,
       10.38242212,  6.50111356, 11.51692027, 11.05729619, 11.33592525,
        9.06386805, 10.14875279, 13.88625176,  9.70324403,  9.57577134,
       10.73657322, 11.14922247, 10.64058936, 10.05417778,  7.11414674,
        8.79873853, 10.53106866, 12.15568563, 14.73725377, 10.4677728 ,
       11.3381813 , 10.93815417,  6.36984126, 11.59672245,  8.36035359,
        7.17650357,  7.01187797,  7.83576183, 10.26683872, 13.03565983,
        9.82808508, 11.58233697,  9.96893248, 13.08069356,  9.0082224 ,
        5.56059198,  4.5538385 ,  7.4801978 ,  7.58478857, 12.76612942,
       11.44347736,  9.00393829, 11.86838214,  8.23714371,  6.13850954,
        8.14513016,  7.95021754, 11.02861511, 12.49574681,  6.87744304,
       10.02599155, 12.72948886,  7.82443768, 10.61069998, 11.87694174,
        9.50753557, 11.71489237, 11.85715748, 10.40138756,  9.86741632,
       11.34159445, 11.55168054, 12.10473518,  7.55497693,  7.72224676,
        9.42112717,  8.97684818, 11.36235945,  9.15185321, 11.6660796 ,
        7.05295923, 12.09171893, 10.37775301,  5.18613173, 13.74554017,
        9.39357233, 10.89387172,  7.61734182,  7.56980009, 12.89233218,
       13.95823326,  9.3076926 ,  9.12715854,  7.86410187, 11.46989646,
        9.46980782, 10.76791743,  7.64959886,  7.84876528,  6.8132351 ,
       12.03046547,  9.39635715,  9.86233115, 11.44152939,  9.65702191,
       12.92602493, 12.51231321,  9.5290725 , 10.05525405, 11.23532525,
       11.34176987,  8.49062323,  8.55423377, 13.3337241 ,  6.88892955,
        8.42306257,  9.1733929 ,  7.73100341, 10.3292513 , 10.7768093 ,
        6.90476554, 10.80536613,  8.55721253, 12.35557752,  8.17362026,
        9.9222233 , 10.17447989,  9.65853759,  9.10258682,  6.81186594,
       11.41636114,  8.22080459,  7.19722404,  8.37382443, 10.66612777,
       11.66895152,  8.08679863,  8.16335041,  9.96872857,  7.41910189])
```

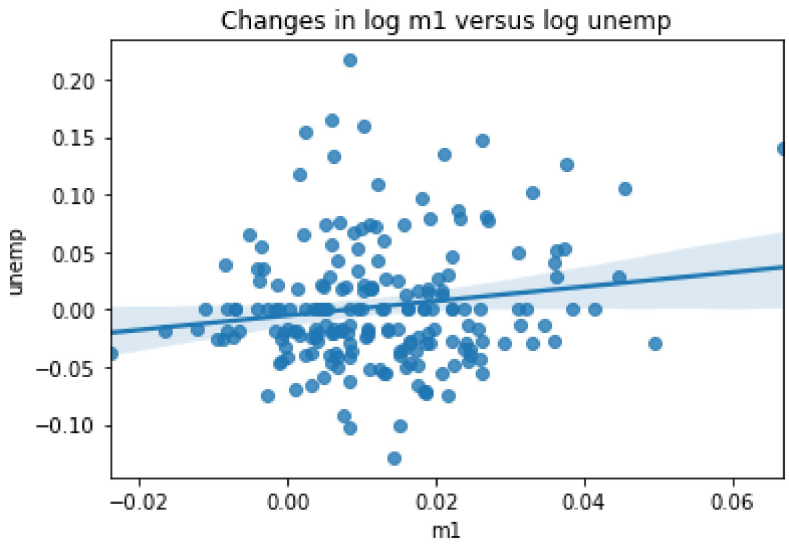
In [10]:
```
macro = pd.read_csv('documents/macrodata.csv')
data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
trans_data = np.log(data).diff().dropna()
trans_data[-5:]
```

Out[10]:

	cpi	m1	tbilrate	unemp
198	-0.007904	0.045361	-0.396881	0.105361
199	-0.021979	0.066753	-2.277267	0.139762
200	0.002340	0.010286	0.606136	0.160343
201	0.008419	0.037461	-0.200671	0.127339
202	0.008894	0.012202	-0.405465	0.042560

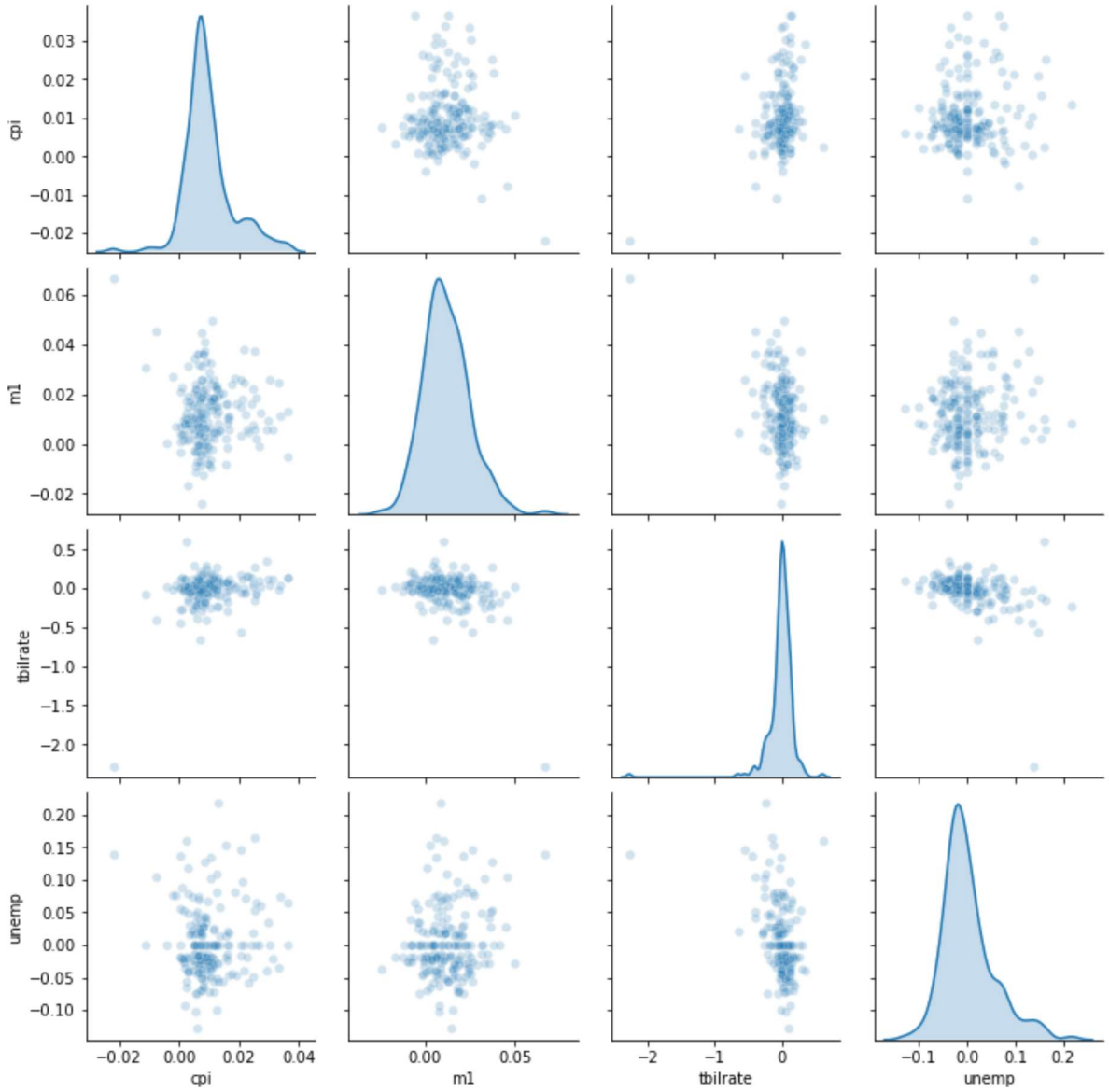
In [12]:
```
import matplotlib.pyplot as plt
sns.regplot('m1', 'unemp', data=trans_data)
plt.title('Changes in log %s versus log %s' % ('m1', 'unemp'))
```

Out[12]:
```
Text(0.5, 1.0, 'Changes in log m1 versus log unemp')
```



In [13]:
```
sns.pairplot(trans_data, diag_kind='kde', plot_kws={'alpha': 0.2})
```

Out[13]:
```
<seaborn.axisgrid.PairGrid at 0x23c1b6067f0>
```


cpi
m1
tbilrate
unemp
0.03
0.02
0.01
0.00
−0.01
−0.02
0.06
0.04
0.02
0.00
−0.02
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.02
0.00
0.02
0.04
0.00
0.05
−2
−1
0
−0.1
0.0
0.1
0.2

In [3]:
```
dir(np)
```

Out[3]:
```
['ALLOW_THREADS',
 'AxisError',
 'BUFSIZE',
 'CLIP',
 'ComplexWarning',
 'DataSource',
 'ERR_CALL',
 'ERR_DEFAULT',
 'ERR_IGNORE',
 'ERR_LOG',
 'ERR_PRINT',
 'ERR_RAISE',
 'ERR_WARN',
 'FLOATING_POINT_SUPPORT',
 'FPE_DIVIDEBYZERO',
 'FPE_INVALID',
 'FPE_OVERFLOW',
 'FPE_UNDERFLOW',
 'False_',
 'Inf',
 'Infinity',
 'MAXDIMS',
 'MAY_SHARE_BOUNDS',
 'MAY_SHARE_EXACT',
 'MachAr',
 'ModuleDeprecationWarning',
 'NAN',
 'NINF',
 'NZERO',
 'NaN',
 'PINF',
 'PZERO',
 'RAISE',
 'RankWarning',
 'SHIFT_DIVIDEBYZERO',
 'SHIFT_INVALID',
 'SHIFT_OVERFLOW',
 'SHIFT_UNDERFLOW',
 'ScalarType',
 'Tester',
 'TooHardError',
 'True_',
 'UFUNC_BUFSIZE_DEFAULT',
 'UFUNC_PYVALS_NAME',
 'VisibleDeprecationWarning',
 'WRAP',
 '_NoValue',
 '_UFUNC_API',
 '__NUMPY_SETUP__',
 '__all__',
 '__builtins__',
 '__cached__',
 '__config__',
 '__dir__',
 '__doc__',
 '__file__',
 '__getattr__',
 '__git_revision__',
 '__loader__',
 '__mkl_version__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
```

```
'__version__',
'_add_newdoc_ufunc',
'_distributor_init',
'_globals',
'_mat',
'_pytesttester',
'abs',
'absolute',
'absolute_import',
'add',
'add_docstring',
'add_newdoc',
'add_newdoc_ufunc',
'alen',
'all',
'allclose',
'alltrue',
'amax',
'amin',
'angle',
'any',
'append',
'apply_along_axis',
'apply_over_axes',
'arange',
'arccos',
'arccosh',
'arcsin',
'arcsinh',
'arctan',
'arctan2',
'arctanh',
'argmax',
'argmin',
'argpartition',
'argsort',
'argwhere',
'around',
'array',
'array2string',
'array_equal',
'array_equiv',
'array_repr',
'array_split',
'array_str',
'asanyarray',
'asarray',
'asarray_chkfinite',
'ascontiguousarray',
'asfarray',
'asfortranarray',
'asmatrix',
'asscalar',
'atleast_1d',
'atleast_2d',
'atleast_3d',
'average',
'bartlett',
'base_repr',
'binary_repr',
'bincount',
'bitwise_and',
'bitwise_not',
'bitwise_or',
```

```
'bitwise_xor',
'blackman',
'block',
'bmat',
'bool',
'bool8',
'bool_',
'broadcast',
'broadcast_arrays',
'broadcast_to',
'busday_count',
'busday_offset',
'busdaycalendar',
'byte',
'byte_bounds',
'bytes0',
'bytes_',
'c_',
'can_cast',
'cast',
'cbrt',
'cdouble',
'ceil',
'cfloat',
'char',
'character',
'chararray',
'choose',
'clip',
'clongdouble',
'clongfloat',
'column_stack',
'common_type',
'compare_chararrays',
'compat',
'complex',
'complex128',
'complex64',
'complex_',
'complexfloating',
'compress',
'concatenate',
'conj',
'conjugate',
'convolve',
'copy',
'copysign',
'copyto',
'core',
'corrcoef',
'correlate',
'cos',
'cosh',
'count_nonzero',
'cov',
'cross',
'csingle',
'ctypeslib',
'cumprod',
'cumproduct',
'cumsum',
'datetime64',
'datetime_as_string',
'datetime_data',
```

```
'deg2rad',
'degrees',
'delete',
'deprecate',
'deprecate_with_doc',
'diag',
'diag_indices',
'diag_indices_from',
'diagflat',
'diagonal',
'diff',
'digitize',
'disp',
'divide',
'division',
'divmod',
'dot',
'double',
'dsplit',
'dstack',
'dtype',
'e',
'ediff1d',
'einsum',
'einsum_path',
'emath',
'empty',
'empty_like',
'equal',
'errstate',
'euler_gamma',
'exp',
'exp2',
'expand_dims',
'expm1',
'extract',
'eye',
'fabs',
'fastCopyAndTranspose',
'fft',
'fill_diagonal',
'find_common_type',
'finfo',
'fix',
'flatiter',
'flatnonzero',
'flexible',
'flip',
'fliplr',
'flipud',
'float',
'float16',
'float32',
'float64',
'float_',
'float_power',
'floating',
'floor',
'floor_divide',
'fmax',
'fmin',
'fmod',
'format_float_positional',
'format_float_scientific',
```

```
'format_parser',
'frexp',
'frombuffer',
'fromfile',
'fromfunction',
'fromiter',
'frompyfunc',
'fromregex',
'fromstring',
'full',
'full_like',
'fv',
'gcd',
'generic',
'genfromtxt',
'geomspace',
'get_array_wrap',
'get_include',
'get_printoptions',
'getbufsize',
'geterr',
'geterrcall',
'geterrobj',
'gradient',
'greater',
'greater_equal',
'half',
'hamming',
'hanning',
'heaviside',
'histogram',
'histogram2d',
'histogram_bin_edges',
'histogramdd',
'hsplit',
'hstack',
'hypot',
'i0',
'identity',
'iinfo',
'imag',
'in1d',
'index_exp',
'indices',
'inexact',
'inf',
'info',
'infty',
'inner',
'insert',
'int',
'int0',
'int16',
'int32',
'int64',
'int8',
'int_',
'int_asbuffer',
'intc',
'integer',
'interp',
'intersect1d',
'intp',
'invert',
```

```
'ipmt',
'irr',
'is_busday',
'isclose',
'iscomplex',
'iscomplexobj',
'isfinite',
'isfortran',
'isin',
'isinf',
'isnan',
'isnat',
'isneginf',
'isposinf',
'isreal',
'isrealobj',
'isscalar',
'issctype',
'issubclass_',
'issubdtype',
'issubsctype',
'iterable',
'ix_',
'kaiser',
'kron',
'lcm',
'ldexp',
'left_shift',
'less',
'less_equal',
'lexsort',
'lib',
'linalg',
'linspace',
'little_endian',
'load',
'loads',
'loadtxt',
'log',
'log10',
'log1p',
'log2',
'logaddexp',
'logaddexp2',
'logical_and',
'logical_not',
'logical_or',
'logical_xor',
'logspace',
'long',
'longcomplex',
'longdouble',
'longfloat',
'longlong',
'lookfor',
'ma',
'mafromtxt',
'mask_indices',
'mat',
'math',
'matmul',
'matrix',
'matrixlib',
'max',
```

```
'maximum',
'maximum_sctype',
'may_share_memory',
'mean',
'median',
'memmap',
'meshgrid',
'mgrid',
'min',
'min_scalar_type',
'minimum',
'mintypecode',
'mirr',
'mkl',
'mod',
'modf',
'moveaxis',
'msort',
'multiply',
'nan',
'nan_to_num',
'nanargmax',
'nanargmin',
'nancumprod',
'nancumsum',
'nanmax',
'nanmean',
'nanmedian',
'nanmin',
'nanpercentile',
'nanprod',
'nanquantile',
'nanstd',
'nansum',
'nanvar',
'nbytes',
'ndarray',
'ndenumerate',
'ndfromtxt',
'ndim',
'ndindex',
'nditer',
'negative',
'nested_iters',
'newaxis',
'nextafter',
'nonzero',
'not_equal',
'nper',
'npv',
'numarray',
'number',
'obj2sctype',
'object',
'object0',
'object_',
'ogrid',
'oldnumeric',
'ones',
'ones_like',
'outer',
'packbits',
'pad',
'partition',
```

```
'percentile',
'pi',
'piecewise',
'place',
'pmt',
'poly',
'poly1d',
'polyadd',
'polyder',
'polydiv',
'polyfit',
'polyint',
'polymul',
'polynomial',
'polysub',
'polyval',
'positive',
'power',
'ppmt',
'print_function',
'printoptions',
'prod',
'product',
'promote_types',
'ptp',
'put',
'put_along_axis',
'putmask',
'pv',
'quantile',
'r_',
'rad2deg',
'radians',
'random',
'rate',
'ravel',
'ravel_multi_index',
'real',
'real_if_close',
'rec',
'recarray',
'recfromcsv',
'recfromtxt',
'reciprocal',
'record',
'remainder',
'repeat',
'require',
'reshape',
'resize',
'result_type',
'right_shift',
'rint',
'roll',
'rollaxis',
'roots',
'rot90',
'round',
'round_',
'row_stack',
's_',
'safe_eval',
'save',
'savetxt',
```

```
'savez',
'savez_compressed',
'sctype2char',
'sctypeDict',
'sctypeNA',
'sctypes',
'searchsorted',
'select',
'set_numeric_ops',
'set_printoptions',
'set_string_function',
'setbufsize',
'setdiff1d',
'seterr',
'seterrcall',
'seterrobj',
'setxor1d',
'shape',
'shares_memory',
'short',
'show_config',
'sign',
'signbit',
'signedinteger',
'sin',
'sinc',
'single',
'singlecomplex',
'sinh',
'size',
'sometrue',
'sort',
'sort_complex',
'source',
'spacing',
'split',
'sqrt',
'square',
'squeeze',
'stack',
'std',
'str',
'str0',
'str_',
'string_',
'subtract',
'sum',
'swapaxes',
'sys',
'take',
'take_along_axis',
'tan',
'tanh',
'tensordot',
'test',
'testing',
'testing',
'tile',
'timedelta64',
'trace',
'tracemalloc_domain',
'transpose',
'trapz',
'tri',
```

```
 'tril',
 'tril_indices',
 'tril_indices_from',
 'trim_zeros',
 'triu',
 'triu_indices',
 'triu_indices_from',
 'true_divide',
 'trunc',
 'typeDict',
 'typeNA',
 'typecodes',
 'typename',
 'ubyte',
 'ufunc',
 'uint',
 'uint0',
 'uint16',
 'uint32',
 'uint64',
 'uint8',
 'uintc',
 'uintp',
 'ulonglong',
 'unicode',
 'unicode_',
 'union1d',
 'unique',
 'unpackbits',
 'unravel_index',
 'unsignedinteger',
 'unwrap',
 'ushort',
 'vander',
 'var',
 'vdot',
 'vectorize',
 'version',
 'void',
 'void0',
 'vsplit',
 'vstack',
 'warnings',
 'where',
 'who',
 'zeros',
 'zeros_like']
```

In [4]:
```
help(np.random.normal)
```

```
Help on built-in function normal:

normal(...) method of numpy.random.mtrand.RandomState instance
    normal(loc=0.0, scale=1.0, size=None)

    Draw random samples from a normal (Gaussian) distribution.

    The probability density function of the normal distribution, first
    derived by De Moivre and 200 years later by both Gauss and Laplace
    independently [2]_, is often called the bell curve because of
    its characteristic shape (see the example below).

    The normal distributions occurs often in nature.  For example, it
    describes the commonly occurring distribution of samples influenced
    by a large number of tiny, random disturbances, each with its own
    unique distribution [2]_.

    .. note::
        New code should use the ``normal`` method of a ``default_rng()``
        instance instead; see `random-quick-start`.

    Parameters
    ----------
    loc : float or array_like of floats
        Mean ("centre") of the distribution.
    scale : float or array_like of floats
        Standard deviation (spread or "width") of the distribution. Must be
        non-negative.
    size : int or tuple of ints, optional
        Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
        ``m * n * k`` samples are drawn.  If size is ``None`` (default),
        a single value is returned if ``loc`` and ``scale`` are both scalars.
        Otherwise, ``np.broadcast(loc, scale).size`` samples are drawn.

    Returns
    -------
    out : ndarray or scalar
        Drawn samples from the parameterized normal distribution.

    See Also
    --------
    scipy.stats.norm : probability density function, distribution or
        cumulative density function, etc.
    Generator.normal: which should be used for new code.

    Notes
    -----
    The probability density for the Gaussian distribution is

    .. math:: p(x) = \frac{1}{\sqrt{ 2 \pi \sigma^2 }}
                     e^{ - \frac{ (x - \mu)^2 } {2 \sigma^2} },

    where :math:`\mu` is the mean and :math:`\sigma` the standard
    deviation. The square of the standard deviation, :math:`\sigma^2`,
    is called the variance.

    The function has its peak at the mean, and its "spread" increases with
    the standard deviation (the function reaches 0.607 times its maximum at
    :math:`x + \sigma` and :math:`x - \sigma` [2]_).  This implies that
    normal is more likely to return samples lying close to the mean, rather
    than those far away.

    References
    ----------
```

```
    .. [1] Wikipedia, "Normal distribution",
           https://en.wikipedia.org/wiki/Normal_distribution
    .. [2] P. R. Peebles Jr., "Central Limit Theorem" in "Probability,
           Random Variables and Random Signal Principles", 4th ed., 2001,
           pp. 51, 51, 125.

    Examples
    --------
    Draw samples from the distribution:

    >>> mu, sigma = 0, 0.1 # mean and standard deviation
    >>> s = np.random.normal(mu, sigma, 1000)

    Verify the mean and the variance:

    >>> abs(mu - np.mean(s))
    0.0  # may vary

    >>> abs(sigma - np.std(s, ddof=1))
    0.1  # may vary

    Display the histogram of the samples, along with
    the probability density function:

    >>> import matplotlib.pyplot as plt
    >>> count, bins, ignored = plt.hist(s, 30, density=True)
    >>> plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
    ...                np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
    ...          linewidth=2, color='r')
    >>> plt.show()

    Two-by-four array of samples from N(3, 6.25):

    >>> np.random.normal(3, 2.5, size=(2, 4))
    array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],   # random
           [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]])  # random
```

In [38]:
```
np.random.normal(3, 2.5, size=(2, 4))
```

Out[38]:
```
array([[5.91483128, 5.32599636, 5.63035748, 1.23429182],
       [7.33032086, 1.7341025 , 1.78613077, 2.75638412]])
```

Filling Missing Values with Group-Specific Values

In [79]:
```
tips = pd.read_csv('Downloads/tips.csv')
# Add tip percentage of total bill
tips['tip_pct'] = tips['tip'] / tips['total_bill']
tips[:6]
grouped = tips.groupby(['day', 'smoker'])
grouped_pct = grouped['tip_pct']
grouped_pct.agg('mean')
```

Out[79]:
```
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 [80]:
```
functions = ['count', 'mean', 'max']
result = grouped['tip_pct', 'total_bill'].agg(functions)
result
result['tip_pct']
```

```
<ipython-input-80-30de1f60e52c>:2: FutureWarning: Indexing with multiple keys (imp
licitly converted to a tuple of keys) will be deprecated, use a list instead.
  result = grouped['tip_pct', 'total_bill'].agg(functions)
```

Out[80]:

		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 [81]:
```
ftuples = [('Durchschnitt', 'mean'), ('Abweichung', np.var)]
grouped['tip_pct', 'total_bill'].agg(ftuples)
grouped.agg({'tip' : np.max, 'size' : 'sum'})
grouped.agg({'tip_pct' : ['min', 'max', 'mean', 'std'],
             'size' : 'sum'})
```

```
<ipython-input-81-a57737d781e6>:2: FutureWarning: Indexing with multiple keys (imp
licitly converted to a tuple of keys) will be deprecated, use a list instead.
  grouped['tip_pct', 'total_bill'].agg(ftuples)
```

Out[81]:

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

In [84]:
```
#Returning Aggregated Data Without Row Indexes
tips.groupby(['day', 'smoker'], as_index=False).mean()
#Apply: General split-apply-combine
def top(df, n=5, column='tip_pct'):
    return df.sort_values(by=column)[-n:]
top(tips, n=6)
tips.groupby('smoker').apply(top)
```

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

Out[84]:

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

In [ ]:
```
Cross-Tabulations: Crosstab
from io import StringIO
data = """\
Sample  Nationality  Handedness
1   USA  Right-handed
2   Japan    Left-handed
3   USA  Right-handed
4   Japan    Right-handed
5   Japan    Left-handed
6   Japan    Right-handed
7   USA  Right-handed
8   USA  Left-handed
9   Japan    Right-handed
10  USA  Right-handed"""
data = pd.read_table(StringIO(data), sep='\s+')
data
pd.crosstab(data.Nationality, data.Handedness, margins=True)
pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
pd.options.display.max_rows = PREVIOUS_MAX_ROWS
```

In [85]:
```
result.unstack('smoker')
```

Out[85]:
```
       smoker
count  No        151.000000
       Yes        93.000000
mean   No          0.159328
       Yes         0.163196
std    No          0.039910
       Yes         0.085119
min    No          0.056797
       Yes         0.035638
25%    No          0.136906
       Yes         0.106771
50%    No          0.155625
       Yes         0.153846
75%    No          0.185014
       Yes         0.195059
max    No          0.291990
       Yes         0.710345
dtype: float64
```

In [51]:
```
f = lambda x: x.describe()
grouped.apply(f)
#Suppressing the Group Keys
tips.groupby('smoker', group_keys=False).apply(top)
```

Out[51]:

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

In [52]:
```
#Quantile and Bucket Analysis
frame = pd.DataFrame({'data1': np.random.randn(1000),
                      'data2': np.random.randn(1000)})
quartiles = pd.cut(frame.data1, 4)
quartiles[:10]
```

Out[52]:
```
0     (0.123, 1.598]
1    (-1.352, 0.123]
2    (-1.352, 0.123]
3     (0.123, 1.598]
4     (1.598, 3.073]
5     (0.123, 1.598]
6     (0.123, 1.598]
7    (-1.352, 0.123]
8     (0.123, 1.598]
9     (0.123, 1.598]
Name: data1, dtype: category
Categories (4, interval[float64]): [(-2.833, -1.352] < (-1.352, 0.123] < (0.123,
1.598] < (1.598, 3.073]]
```

In [88]:
```
def get_stats(group):
    return {'min': group.min(), 'max': group.max(),
            'count': group.count(), 'mean': group.mean()}
grouped = frame.data2.groupby(quartiles)
grouped.apply(get_stats).unstack()
```

Out[88]:

	min	max	count	mean
data1				
(-2.833, -1.352]	-2.441812	2.276219	81.0	0.152779
(-1.352, 0.123]	-2.649437	3.566932	471.0	-0.014782
(0.123, 1.598]	-3.924078	2.856075	378.0	-0.005122
(1.598, 3.073]	-2.344985	2.449851	70.0	0.073952

In [54]:
```
# Return quantile numbers
grouping = pd.qcut(frame.data1, 10, labels=False)
grouped = frame.data2.groupby(grouping)
grouped.apply(get_stats).unstack()
```

Out[54]:

	min	max	count	mean
data1				
0	-2.441812	2.824646	100.0	0.139306
1	-2.153272	3.566932	100.0	0.084296
2	-2.045475	2.132152	100.0	0.063267
3	-2.319525	3.065723	100.0	-0.054366
4	-2.440137	1.755701	100.0	-0.112216
5	-2.649437	2.104650	100.0	-0.101220
6	-3.924078	2.856075	100.0	-0.084339
7	-2.508853	1.719684	100.0	-0.093591
8	-2.568530	2.401409	100.0	0.182234
9	-2.344985	2.558548	100.0	0.063164

In [90]:
```
#Example: Filling Missing Values with Group-Specific Values
s = pd.Series(np.random.randn(6))
s[::2] = np.nan
s
s.fillna(s.mean())
states = ['Ohio', 'New York', 'Vermont', 'Florida',
          'Oregon', 'Nevada', 'California', 'Idaho']
group_key = ['East'] * 4 + ['West'] * 4
data = pd.Series(np.random.randn(8), index=states)
data
data[['Vermont', 'Nevada', 'Idaho']] = np.nan
data
data.groupby(group_key).mean()
fill_mean = lambda g: g.fillna(g.mean())
data.groupby(group_key).apply(fill_mean)
```

Out[90]:
```
Ohio          0.358479
New York     -0.067875
Vermont       0.232800
Florida       0.407795
Oregon        0.922629
Nevada        1.421370
California    1.920111
Idaho         1.421370
dtype: float64
```

In [91]:
```
fill_values = {'East': 0.5, 'West': -1}
fill_func = lambda g: g.fillna(fill_values[g.name])
data.groupby(group_key).apply(fill_func)
```

Out[91]:
```
Ohio          0.358479
New York     -0.067875
Vermont       0.500000
Florida       0.407795
Oregon        0.922629
Nevada       -1.000000
California    1.920111
Idaho        -1.000000
dtype: float64
```

In [58]:
```
#Example: Random Sampling and Permutation
# Hearts, Spades, Clubs, Diamonds
```

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

deck = pd.Series(card_val, index=cards)
deck[:13]
```

Out[58]:
```
AH      1
2H      2
3H      3
4H      4
5H      5
6H      6
7H      7
8H      8
9H      9
10H    10
JH     10
KH     10
QH     10
dtype: int64
```

In [59]:
```
def draw(deck, n=5):
    return deck.sample(n)
draw(deck)
get_suit = lambda card: card[-1] # last letter is suit
deck.groupby(get_suit).apply(draw, n=2)
deck.groupby(get_suit, group_keys=False).apply(draw, n=2)
```

Out[59]:
```
3C      3
10C    10
8D      8
JD     10
7H      7
4H      4
6S      6
8S      8
dtype: int64
```

In [60]:
```
#Example: Group Weighted Average and Correlation
df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',
                                'b', 'b', 'b', 'b'],
                   'data': np.random.randn(8),
                   'weights': np.random.rand(8)})
df
grouped = df.groupby('category')
get_wavg = lambda g: np.average(g['data'], weights=g['weights'])
grouped.apply(get_wavg)
```

Out[60]:
```
category
a    0.389817
b   -0.136580
dtype: float64
```

In [107…
```
close_px = pd.read_csv('Documents/stock_px_2.csv', parse_dates=True,
                       index_col=0)
spx_corr = lambda x: x.corrwith(x['SPX'])
rets = close_px.pct_change().dropna()
rets
```

Out[107]:

	AAPL	MSFT	XOM	SPX
2003-03-01	0.006757	0.001421	0.000684	-0.000484
2003-06-01	0.000000	0.017975	0.024624	0.022474
2003-07-01	-0.002685	0.019052	-0.033712	-0.006545
2003-08-01	-0.020188	-0.028272	-0.004145	-0.014086
2003-09-01	0.008242	0.029094	0.021159	0.019386
...	...	...	...	...
2011-10-10	0.051406	0.026286	0.036977	0.034125
2011-11-10	0.029526	0.002227	-0.000131	0.000544
2011-12-10	0.004747	-0.001481	0.011669	0.009795
2011-10-13	0.015515	0.008160	-0.010238	-0.002974
2011-10-14	0.033225	0.003311	0.022784	0.017380

2213 rows × 4 columns

In [ ]:

In [63]:
```
get_year = lambda x: x.year
by_year = rets.groupby(get_year)
by_year.apply(spx_corr)
by_year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
```

Out[63]:
```
2003    0.480868
2004    0.259024
2005    0.300093
2006    0.161735
2007    0.417738
2008    0.611901
2009    0.432738
2010    0.571946
2011    0.581987
dtype: float64
```

In [64]:
```
#Example: Group-Wise Linear Regression
import statsmodels.api as sm
def regress(data, yvar, xvars):
    Y = data[yvar]
    X = data[xvars]
    X['intercept'] = 1.
    result = sm.OLS(Y, X).fit()
    return result.params
by_year.apply(regress, 'AAPL', ['SPX'])
```

Out[64]:

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

In [115…

```
##Pivot Tables and Cross-Tabulation
tips.pivot_table(index=['day', 'smoker'])
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                 columns='smoker')
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                 columns='smoker', margins=True)
tips.pivot_table('tip_pct', index=['time', 'smoker'], columns='day',
                 aggfunc=len, margins=True)
tips.pivot_table('tip_pct', index=['time', 'size', 'smoker'],
                 columns='day', aggfunc='mean', fill_value=0)
```

Out[115]:

		day	Fri	Sat	Sun	Thur
time	size	smoker				
Dinner	1	No	0.000000	0.137931	0.000000	0.000000
		Yes	0.000000	0.325733	0.000000	0.000000
	2	No	0.139622	0.162705	0.168859	0.159744
		Yes	0.171297	0.148668	0.207893	0.000000
	3	No	0.000000	0.154661	0.152663	0.000000
		Yes	0.000000	0.144995	0.152660	0.000000
	4	No	0.000000	0.150096	0.148143	0.000000
		Yes	0.117750	0.124515	0.193370	0.000000
	5	No	0.000000	0.000000	0.206928	0.000000
		Yes	0.000000	0.106572	0.065660	0.000000
	6	No	0.000000	0.000000	0.103799	0.000000
Lunch	1	No	0.000000	0.000000	0.000000	0.181728
		Yes	0.223776	0.000000	0.000000	0.000000
	2	No	0.000000	0.000000	0.000000	0.166005
		Yes	0.181969	0.000000	0.000000	0.158843
	3	No	0.187735	0.000000	0.000000	0.084246
		Yes	0.000000	0.000000	0.000000	0.204952
	4	No	0.000000	0.000000	0.000000	0.138919
		Yes	0.000000	0.000000	0.000000	0.155410
	5	No	0.000000	0.000000	0.000000	0.121389
	6	No	0.000000	0.000000	0.000000	0.173706

In [118…

```
#Cross-Tabulations: Crosstab
from io import StringIO
data = """\
Sample  Nationality  Handedness
1   USA  Right-handed
2   Japan    Left-handed
3   USA  Right-handed
4   Japan    Right-handed
5   Japan    Left-handed
6   Japan    Right-handed
7   USA  Right-handed
8   USA  Left-handed
9   Japan    Right-handed
10  USA  Right-handed"""
data = pd.read_table(StringIO(data), sep='\s+')
data
pd.crosstab(data.Nationality, data.Handedness, margins=True)
pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
#pd.options.display.max_rows = PREVIOUS_MAX_ROWS
```

Out[118]:

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

In [ ]:

In [ ]: