This is a collection of various statements, features, etc. of the Python language.

```
In [1]: a = 10
```

```
In [2]: print(a)
        10
```

```
In [3]: import math
In [4]: x = math.cos(2 * math.pi)
        print(x)
        1.0
```

Import the whole module into the current namespace instead.

```
In [5]:from math import *
       x = cos(2 * pi)
       print(x)
       1.0
```

Several ways to look at documentation for a module.

```
In [6]: print(dir(math))
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan', '
atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc',
'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'h
ypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'modf', 'pi',
'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

```
In [7]: help(math.cos)
        Help on built-in function cos in module math:
        cos(...)
               cos(x)
    Return the cosine of x (measured in radians).
```

Variables

```
In [8]: x = 1.0
        type(x)
Out[8]: float
```

```
In [9]: # dynamically typed
        x = 1
        type(x)
Out[9]: int
```

Operators

```
In [10]: 1 + 2, 1 - 2, 1 * 2, 1 / 2
Out[10]: (3, -1, 2, 0)
```

```
In [11]: # integer division of float numbers
```

```
3.0 // 2.0
```

Out[11]: 1.0

```
In [12]: # power operator
         2 ** 2
```

Out[12]: 4

```
In [13]: True and False
```

Out[13]: False

```
In [14]: not False
```

Out[14]: True

```
In [15]: True or False
```

Out[15]: True

```
In [16]: 2 > 1, 2 < 1, 2 > 2, 2 < 2, 2 >= 2, 2 <= 2
```

Out[16]: (True, False, False, False, True, True)

```
In [17]: # equality
         [1,2] == [1,2]
```

Out[17]: True

Strings

```
In [18]: s = "Hello world"
         type(s)
```

Out[18]: str

```
In [19]: len(s)
```

Out[19]: 11

```
In [20]: s2 = s.replace("world", "test")
         print(s2)
```

```
Hello test
```

```
In [21]: s[0]
```

Out[21]: 'H'

```
In [22]: s[0:5]
```

Out[22]: 'Hello'

```
In [23]: s[6:]
```

Out[23]: 'world'

```
In [24]: s[:]
```

Out[24]: 'Hello world'

```
In [25]: # define step size of 2
         s[::2]
```

```
Out[25]: 'Hlowrd'
```

```
In [26]: # automatically adds a space
         print("str1", "str2", "str3")

         ('str1', 'str2', 'str3')
```

```
In [27]: # C-style formatting
         print("value = %f" % 1.0)

         value = 1.000000
```

```
In [28]: # alternative, more intuitive way of formatting a string
         s3 = 'value1 = {0}, value2 = {1}'.format(3.1415, 1.5)
         print(s3)

         value1 = 3.1415, value2 = 1.5
```

Lists

```
In [29]: l = [1,2,3,4]
         print(type(l))
         print(l)

         <type 'list'>
         [1, 2, 3, 4]
In [30]: print(l[1:3])
         print(l[::2])

        [2, 3]
        [1, 3]
```

```
In [31]: l[0]
Out[31]: 1
```

```
In [32]: # don't have to be the same type
         l = [1, 'a', 1.0, 1-1j]
         print(l)

         [1, 'a', 1.0, (1-1j)]
```

```
In [33]: start = 10
         stop = 30
         step = 2
         range(start, stop, step)

         # consume the iterator created by range
         list(range(start, stop, step))
Out[33]: [10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

```
In [34]: # create a new empty list
         l = []
         # add an elements using `append`
         l.append("A")
         l.append("d")
         l.append("d")
```

```
print(l)

['A', 'd', 'd']
```

```
In [35]: l[1:3] = ["b", "c"]
         print(l)

['A', 'b', 'c']
```

```
In [36]: l.insert(0, "i")
         l.insert(1, "n")
         l.insert(2, "s")
         l.insert(3, "e")
         l.insert(4, "r")
         l.insert(5, "t")
         print(l)

['i', 'n', 's', 'e', 'r', 't', 'A', 'b', 'c']
```

```
In [37]: l.remove("A")
         print(l)

['i', 'n', 's', 'e', 'r', 't', 'b', 'c']
```

```
In [38]: del l[7]
         del l[6]
         print(l)

['i', 'n', 's', 'e', 'r', 't']
```

Tuples

```
In [39]: point = (10, 20)
         print(point, type(point))

((10, 20), <type 'tuple'>)
```

```
In [40]: # unpacking
         x, y = point
         print("x =", x)
         print("y =", y)

('x =', 10)
('y =', 20)
```

Dictionaries

```
In [41]: params = {"parameter1" : 1.0,
                   "parameter2" : 2.0,
                   "parameter3" : 3.0,}
         print(type(params))
         print(params)

<type 'dict'>
{'parameter1': 1.0, 'parameter3': 3.0, 'parameter2': 2.0}
```

```
In [42]: params["parameter1"] = "A"
```

```
params["parameter2"] = "B"
# add a new entry
params["parameter4"] = "D"
print("parameter1 = " + str(params["parameter1"]))
print("parameter2 = " + str(params["parameter2"]))
print("parameter3 = " + str(params["parameter3"]))
print("parameter4 = " + str(params["parameter4"]))
```

```
parameter1 = A
parameter2 = B
parameter3 = 3.0
parameter4 = D
```

Control Flow

In [43]:
```
statement1 = False
statement2 = False

if statement1:
    print("statement1 is True")
elif statement2:
    print("statement2 is True")
else:
    print("statement1 and statement2 are False")
```

```
statement1 and statement2 are False
```

Loops

In [44]:
```
for x in range(4):
  print(x)
```

```
0
1
2
3
```

In [45]:
```
for word in ["scientific", "computing", "with", "python"]:
      print(word)
```

```
scientific
computing
with
python
```

In [46]:
```
for key, value in params.items():
      print(key + " = " + str(value))
```

```
parameter4 = D
parameter1 = A
parameter3 = 3.0
parameter2 = B
```

In [47]:
```
for idx, x in enumerate(range(-3,3)):
      print(idx, x)
```

```
(0, -3)
```

```
(1, -2)
(2, -1)
(3, 0)
(4, 1)
(5, 2)
```

```
In [48]: l1 = [x**2 for x in range(0,5)]
         print(l1)

         [0, 1, 4, 9, 16]
```

```
In [49]: i = 0
         while i < 5:
             print(i)
             i = i + 1
         print("done")

         0
         1
         2
         3
         4
         done
```

Functions

```
In [50]: # include a docstring
         def func(s):
             """
             Print a string 's' and tell how many characters it has
             """
             print(s + " has " + str(len(s)) + " characters")
```

```
In [51]: help(func)
         Help on function func in module __main__:

         func(s)
             Print a string 's' and tell how many characters it has
```

```
In [52]: func("test")
         test has 4 characters
```

```
In [53]: def square(x):
             return x ** 2
```

```
In [54]: square(5)
Out[54]: 25
```

```
In [55]: # multiple return values
         def powers(x):
             return x ** 2, x ** 3, x ** 4
```

```
In [56]: powers(5)
Out[56]: (25, 125, 625)
```

```
In [57]: x2, x3, x4 = powers(5)
         print(x3)

         125
```

```
In [58]: f1 = lambda x: x**2
         f1(5)

Out[58]: 25
```

```
In [59]: map(lambda x: x**2, range(-3,4))
Out[59]: [9, 4, 1, 0, 1, 4, 9]
```

```
In [60]: # convert iterator to list
         list(map(lambda x: x**2, range(-3,4)))

Out[60]: [9, 4, 1, 0, 1, 4, 9]
```

Classes

```
In [61]: class Point:
        def __init__(self, x, y):
            self.x = x
            self.y = y

        def translate(self, dx, dy):
            self.x += dx
            self.y += dy

        def __str__(self):
            return("Point at [%f, %f]" % (self.x, self.y))
```

```
In [62]: p1 = Point(0, 0)
               print(p1)

         Point at [0.000000, 0.000000]
```

```
In [63]: p2 = Point(1, 1)
               p1.translate(0.25, 1.5)
         print(p1)
         print(p2)

         Point at [0.250000, 1.500000]
         Point at [1.000000, 1.000000]
```

Exceptions

```
In [64]: try:
            print(test)
         except:
            print("Caught an expection")

         Caught an expection
```

```
In [65]: try:
              print(test)
```

```python
except Exception as e:
    print("Caught an exception: " + str(e))
```

```
Caught an exception: name 'test' is not defined
```

NumPy

NumPy is a library of Python, and it is a shorthand form of Numerical Python. NumPy, along with other python packages SciPy and Matplotlib, aims is aiming to replace Matlab, another popular development environment, for implementing scientific data science applications.

NumPy provides an array of data structure and helps in numerical analysis. NumPy is used to manipulate arrays. The manipulation includes mathematical and logical operations. It can be used for variety of tasks like shape manipulation such as Fourier analysis, and linear algebra operations.

NumPy Data Structures

The important characteristics of defining a NumPy array are listed below:

- Data type
- Item size
- Shape – dimensions
- Data

Data type:

Data types are integers, int, float, complex other data types are Boolean, string, datatime and Python objects.

Item size is the memory requirement of data elements in bytes.

Shape is the dimension of the array.

Data are the elements of a NumPy array.

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined. This allows NumPy to integrate with a wide variety of databases seamlessly and speedily.

Library documentation: http://www.numpy.org/

```
In [1]: from numpy import *
In [2]: # declare a vector using a list as the argument
        v = array([1,2,3,4])
```

```
        v
Out[2]: array([1, 2, 3, 4])

In [3]: # declare a matrix using a nested list as the argument
        M = array([[1,2],[3,4]])
        M
Out[3]:array([[1, 2],
              [3, 4]])

In [4]: # still the same core type with different shapes
        type(v), type(M)
Out[4]: (numpy.ndarray, numpy.ndarray)

In [5]: M.size
Out[5]: 4

In [6]: # arguments: start, stop, step
        x = arange(0, 10, 1)
        x
Out[6]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [7]: linspace(0, 10, 25)
Out[7]: array([  0.        ,   0.41666667,   0.83333333,   1.25,
        1.66666667,   2.08333333,   2.5       ,   2.91666667,
        3.33333333,   3.75      ,   4.16666667,   4.58333333,
        5.        ,   5.41666667,   5.83333333,   6.25      ,
        6.66666667,   7.08333333,   7.5       ,   7.91666667,
        8.33333333,   8.75      ,   9.16666667,   9.58333333,  10.          ])

In [8]: logspace(0, 10, 10, base=e)
Out[8]:
array([  1.00000000e+00,   3.03773178e+00,   9.22781435e+00,
         2.80316249e+01,   8.51525577e+01,   2.58670631e+02,
         7.85771994e+02,   2.38696456e+03,   7.25095809e+03,
         2.20264658e+04])

In [9]: x, y = mgrid[0:5, 0:5]
        x
Out[9]: array([[0, 0, 0, 0, 0],
              [1, 1, 1, 1, 1],
              [2, 2, 2, 2, 2],
              [3, 3, 3, 3, 3],
              [4, 4, 4, 4, 4]])

In [10]: y
Out[10]:
          array([[0, 1, 2, 3, 4],
                 [0, 1, 2, 3, 4],
                 [0, 1, 2, 3, 4],
                 [0, 1, 2, 3, 4],
```

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

In [11]: from numpy import random
In [12]: random.rand(5,5)
Out[12]:
array([[ 0.88096372,  0.53238822,  0.17775764,  0.76591586,  0.6127709 ],
       [ 0.51258827,  0.05731522,  0.05610599,  0.36338405,  0.29548536],
       [ 0.54649788,  0.60544106,  0.38081415,  0.5717322 ,  0.2426889 ],
       [ 0.96448533,  0.22105112,  0.41292727,  0.40652867,  0.57179488],
       [ 0.55815745,  0.22049273,  0.30680923,  0.82881023,  0.36665264]])

In [13]: # normal distribution
         random.randn(5,5)
Out[13]:
array([[ 0.40801047, -0.36738023,  0.0654462 ,  0.16108406,  0.08391533],
       [-1.31495404, -1.31773965,  1.01225524,  0.28113264, -1.32523908],
       [ 1.09106398, -0.37571802,  2.01780085,  0.16072945,  1.0688331 ],
       [ 0.54306468,  0.9436181 , -2.60779314,  0.27348637,  0.60950091],
       [-1.0055051 ,  1.77771874,  0.33209667, -0.10772336, -0.66501805]])

In [14]: diag([1,2,3])
Out[14]:
         array([[1, 0, 0],
                [0, 2, 0],
                [0, 0, 3]])

In [15]: M.itemsize
Out[15]: 4

In [16]: M.nbytes
Out[16]: 16

In [17]: M.ndim
Out[17]: 2

In [18]: v[0], M[1,1]
Out[18]: (1, 4)

In [19]: M[1]
Out[19]: array([3, 4])

In [20]: # assign new value
        M[0,0] = 7
        M
Out[20]:array([[7, 2],
              [3, 4]])

In [21]: M[0,:] = 0
         M
Out[21]:array([[0, 0],
```

```
              [3, 4]])
```

```
In [22]: # slicing works just like with lists
         A = array([1,2,3,4,5])
         A[1:3]
Out[22]: array([2, 3])
```

```
In [23]: A = array([[n+m*10 for n in range(5)] for m in range(5)])
         A
Out[23]:
         array([[ 0,  1,  2,  3,  4],
                [10, 11, 12, 13, 14],
                [20, 21, 22, 23, 24],
                [30, 31, 32, 33, 34],
                [40, 41, 42, 43, 44]])
```

```
In [24]: row_indices = [1, 2, 3]
         A[row_indices]
Out[24]:
         array([[10, 11, 12, 13, 14],
                [20, 21, 22, 23, 24],
                [30, 31, 32, 33, 34]])
```

```
In [25]: # index masking
         B = array([n for n in range(5)])
         row_mask = array([True, False, True, False, False])
         B[row_mask]
Out[25]: array([0, 2])
```

Linear Algebra

```
In [26]: v1 = arange(0, 5)

In [27]: v1 + 2
Out[27]: array([2, 3, 4, 5, 6])

In [28]: v1 * 2
Out[28]: array([0, 2, 4, 6, 8])

In [29]: v1 * v1
Out[29]: array([ 0,  1,  4,  9, 16])

In [30]: dot(v1, v1)
Out[30]: 30

In [31]: dot(A, v1)
Out[31]: array([ 30, 130, 230, 330, 430])

In [32]: # cast changes behavior of + - * etc. to use matrix algebra
         M = matrix(A)
         M * M
```

```
Out[32]: matrix([[ 300,  310,  320,  330,  340],
                 [1300, 1360, 1420, 1480, 1540],
                 [2300, 2410, 2520, 2630, 2740],
                 [3300, 3460, 3620, 3780, 3940],
                 [4300, 4510, 4720, 4930, 5140]])

In [33]: # inner product
         v.T * v
Out[33]: array([ 1,  4,  9, 16])

In [34]: C = matrix([[1j, 2j], [3j, 4j]])
         C
Out[34]: matrix([[ 0.+1.j,  0.+2.j],
                [0.+3.j,  0.+4.j]])

In [35]: conjugate(C)
Out[35]: matrix([[ 0.-1.j,  0.-2.j],
               [0.-3.j,  0.-4.j]])

In [36]: # inverse
C.I
Out[36]:
matrix([[ 0.+2.j ,  0.-1.j ],
        [ 0.-1.5j,  0.+0.5j]])
```

Statistics

```
In [37]: mean(A[:,3])
Out[37]: 23.0

In [38]: std(A[:,3]), var(A[:,3])
Out[38]:(14.142135623730951, 200.0)

In [39]: A[:,3].min(), A[:,3].max()
Out[39]: (3, 43)

In [40]: d = arange(1, 10)
         sum(d), prod(d)
Out[40]:(45, 362880)

In [41]: cumsum(d)
Out[41]: array([ 1,  3,  6, 10, 15, 21, 28, 36, 45])

In [42]: cumprod(d)
Out[42]: array([     1,      2,      6,     24,    120,    720,   5040,
               40320,362880])

In [43]: # sum of diagonal
trace(A)
Out[43]: 110
```

```
In [44]:m = random.rand(3, 3)
        m
Out[44]:
        array([[ 0.37938474,  0.93337301,  0.10864521],
               [ 0.144712  ,  0.12270014,  0.622434  ],
               [ 0.16307745,  0.4850791 ,  0.59703797]])

In [45]: # use axis parameter to specify how function behaves
         m.max(), m.max(axis=0)
Out[45]:
(0.93337300979654614, array([ 0.37938474,  0.93337301,  0.622434 ]))

In [46]: A
Out[46]: array([[ 0,  1,  2,  3,  4],
                [10, 11, 12, 13, 14],
                [20, 21, 22, 23, 24],
                [30, 31, 32, 33, 34],
                [40, 41, 42, 43, 44]])

In [47]: # reshape without copying underlying data
         n, m = A.shape
         B = A.reshape((1,n*m))
         B
Out[47]:
array([[ 0,  1,  2,  3,  4, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24, 30, 31,
        32, 33, 34, 40, 41, 42, 43, 44]])

In [48]: # modify the array
         B[0,0:5] = 5
         B
Out[48]:
array([[ 5,  5,  5,  5,  5, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24, 30, 31,
        32, 33, 34, 40, 41, 42, 43, 44]])

In [49]: # also changed
         A
Out[49]:array([[ 5,  5,  5,  5,  5],
               [10, 11, 12, 13, 14],
               [20, 21, 22, 23, 24],
               [30, 31, 32, 33, 34],
               [40, 41, 42, 43, 44]])
In [50]:# creates a copy
         B = A.flatten()
         B
Out[50]:
array([ 5,  5,  5,  5,  5, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24, 30, 31,
       32, 33, 34, 40, 41, 42, 43, 44])

In [51]: # can insert a dimension in an array
         v = array([1,2,3])
```

```
        v[:, newaxis], v[:,newaxis].shape, v[newaxis,:].shape

Out[51]:(array([[1],
                [2],
                [3]]), (3L, 1L), (1L, 3L))

In [52]:repeat(v, 3)
Out[52]:array([1, 1, 1, 2, 2, 2, 3, 3, 3])

In [53]:tile(v, 3)
Out[53]:array([1, 2, 3, 1, 2, 3, 1, 2, 3])

In [54]:w = array([5, 6])

In [55]: concatenate((v, w), axis=0)
Out[55]: array([1, 2, 3, 5, 6])

In [56]: # deep copy
         B = copy(A)
```

Matplotlib

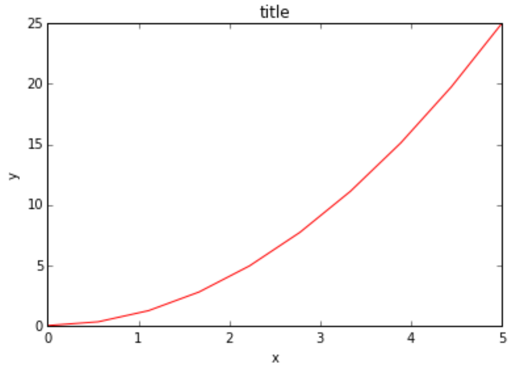
Matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell, web application servers, and six graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code.

Library documentation: http://matplotlib.org/

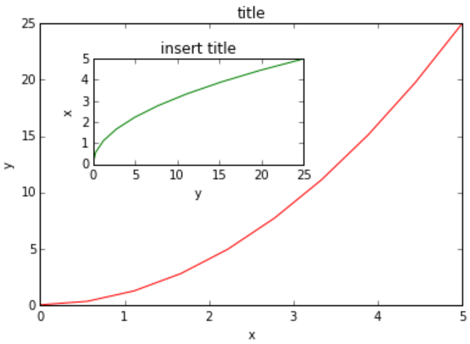
```
In [1]: # needed to display the graphs
        %matplotlib inline
        from pylab import *
```

```
In [2]:x = linspace(0, 5, 10)
y = x ** 2
fig = plt.figure()
# left, bottom, width, height (range 0 to 1)
axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```

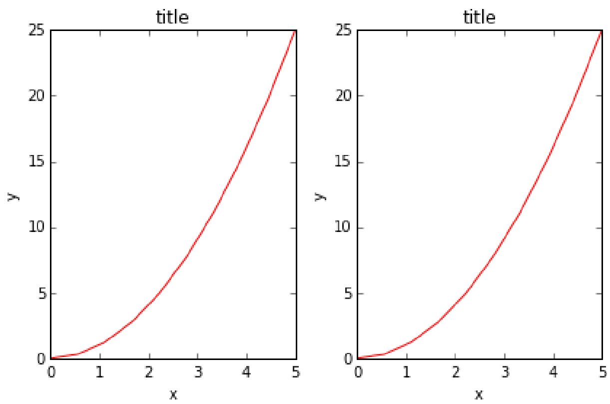



```
In [3]: fig = plt.figure()

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # inset axes

# main figure
axes1.plot(x, y, 'r')
axes1.set_xlabel('x')
axes1.set_ylabel('y')
axes1.set_title('title')

# insert
axes2.plot(y, x, 'g')
axes2.set_xlabel('y')
axes2.set_ylabel('x')
```

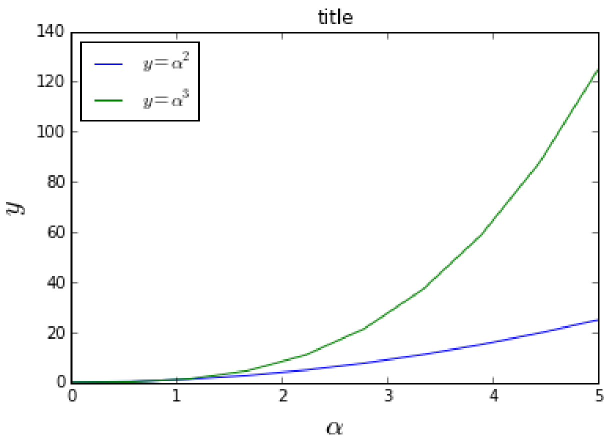
```
axes2.set_title('insert title');
```




```
In [4]: fig, axes = plt.subplots(nrows=1, ncols=2)

for ax in axes:
    ax.plot(x, y, 'r')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')

fig.tight_layout()
```

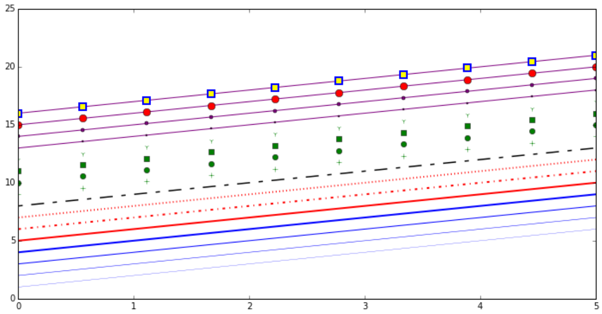



```
In [5]: # example with a legend and latex symbols
fig, ax = plt.subplots()

ax.plot(x, x**2, label=r"$y = \alpha^2$")
ax.plot(x, x**3, label=r"$y = \alpha^3$")
ax.legend(loc=2) # upper left corner
ax.set_xlabel(r'$\alpha$', fontsize=18)
ax.set_ylabel(r'$y$', fontsize=18)
ax.set_title('title');
```



In [6]:

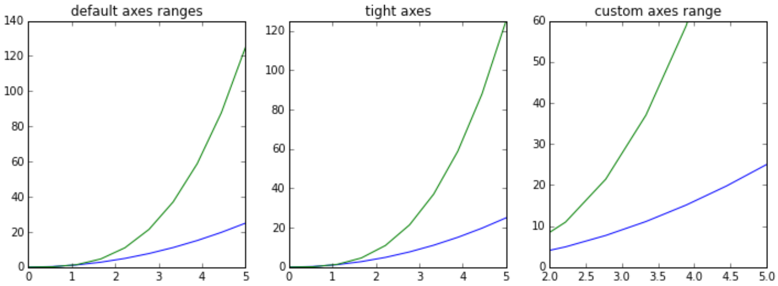
```
# line customization
fig, ax = plt.subplots(figsize=(12,6))

ax.plot(x, x+1, color="blue", linewidth=0.25)
ax.plot(x, x+2, color="blue", linewidth=0.50)
ax.plot(x, x+3, color="blue", linewidth=1.00)
ax.plot(x, x+4, color="blue", linewidth=2.00)

# possible linestype options '-', '–', '-.', ':', 'steps'
ax.plot(x, x+5, color="red", lw=2, linestyle='-')
ax.plot(x, x+6, color="red", lw=2, ls='-.')
ax.plot(x, x+7, color="red", lw=2, ls=':')

# custom dash
line, = ax.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10]) # format: line length, space length, ...

# possible marker symbols: marker = '+', 'o', '*', 's', ',', '.',
# '1', '2', '3', '4', ...
ax.plot(x, x+ 9, color="green", lw=2, ls='*', marker='+')
ax.plot(x, x+10, color="green", lw=2, ls='*', marker='o')
ax.plot(x, x+11, color="green", lw=2, ls='*', marker='s')
ax.plot(x, x+12, color="green", lw=2, ls='*', marker='1')

# marker size and color
ax.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8,
        markerfacecolor="red")
ax.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
        markerfacecolor="yellow", markeredgewidth=2, markeredgecolor="blue");
```



In [7]:

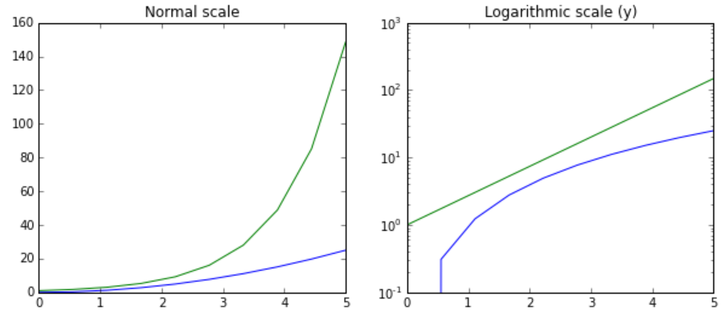
```
# axis controls
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

axes[0].plot(x, x**2, x, x**3)
axes[0].set_title("default axes ranges")

axes[1].plot(x, x**2, x, x**3)
axes[1].axis('tight')
```

```
axes[1].set_title("tight axes")

axes[2].plot(x, x**2, x, x**3)
axes[2].set_ylim([0, 60])
axes[2].set_xlim([2, 5])
axes[2].set_title("custom axes range");
```



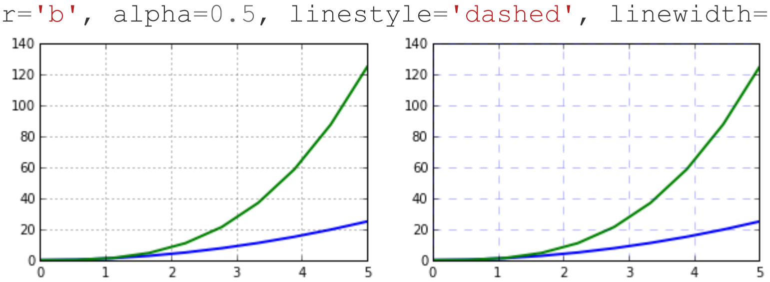
In [8]:

```
# scaling
fig, axes = plt.subplots(1, 2, figsize=(10,4))

axes[0].plot(x, x**2, x, exp(x))
axes[0].set_title("Normal scale")

axes[1].plot(x, x**2, x, exp(x))
axes[1].set_yscale("log")
axes[1].set_title("Logarithmic scale (y)");
```

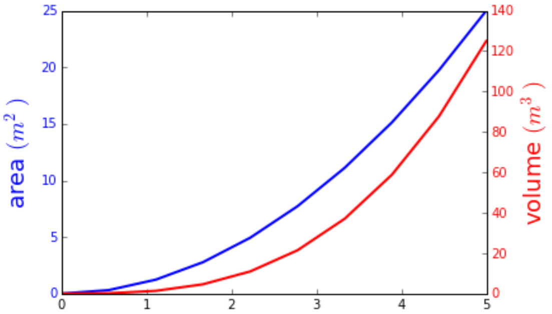


In [9]:

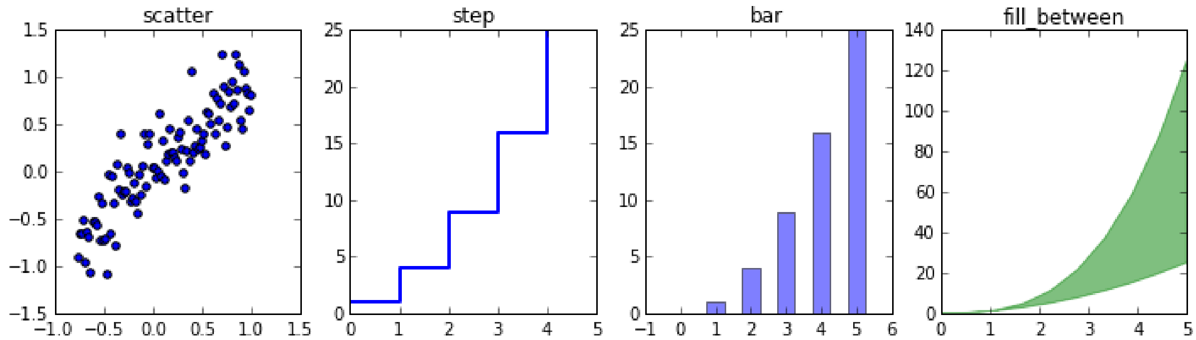
```
# axis grid
fig, axes = plt.subplots(1, 2, figsize=(10,3))

# default grid appearance
axes[0].plot(x, x**2, x, x**3, lw=2)
axes[0].grid(True)

# custom grid appearance
axes[1].plot(x, x**2, x, x**3, lw=2)
axes[1].grid(color='b', alpha=0.5, linestyle='dashed', linewidth=0.5)
```



In [10]:
```
# twin axes example
fig, ax1 = plt.subplots()

ax1.plot(x, x**2, lw=2, color="blue")
ax1.set_ylabel(r"area $(m^2)$", fontsize=18, color="blue")
for label in ax1.get_yticklabels():
    label.set_color("blue")

ax2 = ax1.twinx()
ax2.plot(x, x**3, lw=2, color="red")
ax2.set_ylabel(r"volume $(m^3)$", fontsize=18, color="red")
for label in ax2.get_yticklabels():
    label.set_color("red")
```

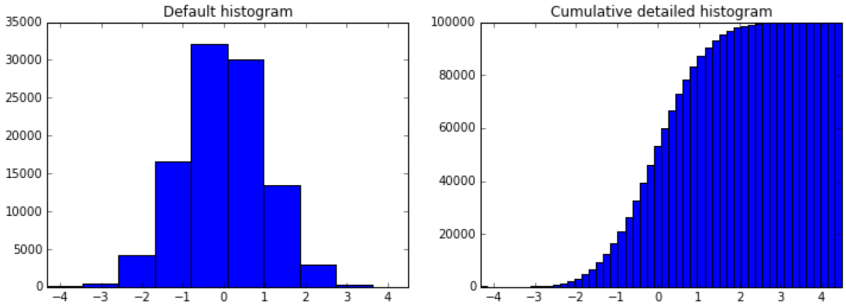


In [11]:
```
# other plot styles
xx = np.linspace(-0.75, 1., 100)
n = array([0,1,2,3,4,5])

fig, axes = plt.subplots(1, 4, figsize=(12,3))

axes[0].scatter(xx, xx + 0.25*randn(len(xx)))
axes[0].set_title("scatter")

axes[1].step(n, n**2, lw=2)
axes[1].set_title("step")

axes[2].bar(n, n**2, align="center", width=0.5, alpha=0.5)
axes[2].set_title("bar")

axes[3].fill_between(x, x**2, x**3, color="green", alpha=0.5);
axes[3].set_title("fill_between");
```



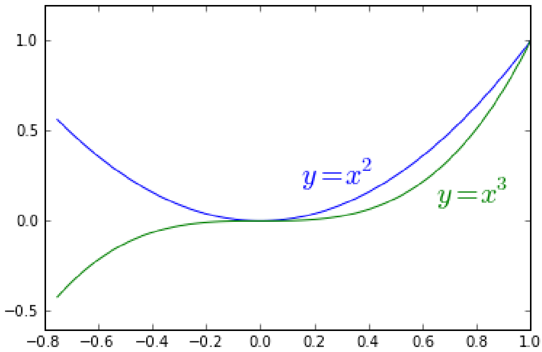
In [12]:

```
# histograms
n = np.random.randn(100000)
fig, axes = plt.subplots(1, 2, figsize=(12,4))

axes[0].hist(n)
axes[0].set_title("Default histogram")
axes[0].set_xlim((min(n), max(n)))

axes[1].hist(n, cumulative=True, bins=50)
axes[1].set_title("Cumulative detailed histogram")
axes[1].set_xlim((min(n), max(n)));
```



In [13]:

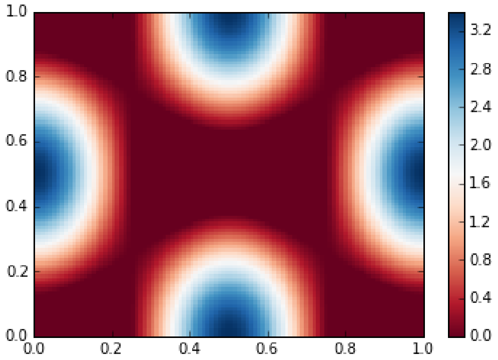
```
# annotations
fig, ax = plt.subplots()

ax.plot(xx, xx**2, xx, xx**3)

ax.text(0.15, 0.2, r"$y=x^2$", fontsize=20, color="blue")
ax.text(0.65, 0.1, r"$y=x^3$", fontsize=20, color="green");
```



In [14]:

```
# color map
alpha = 0.7
phi_ext = 2 * pi * 0.5

def flux_qubit_potential(phi_m, phi_p):
    return ( + alpha - 2 * cos(phi_p)*cos(phi_m) -
        alpha * cos(phi_ext - 2*phi_p))

phi_m = linspace(0, 2*pi, 100)
phi_p = linspace(0, 2*pi, 100)
```

```
X,Y = meshgrid(phi_p, phi_m)
Z = flux_qubit_potential(X, Y).T

fig, ax = plt.subplots()

p = ax.pcolor(X/(2*pi), Y/(2*pi), Z,
              cmap=cm.RdBu, vmin=abs(Z).min(), vmax=abs(Z).max())
cb = fig.colorbar(p, ax=ax)
```

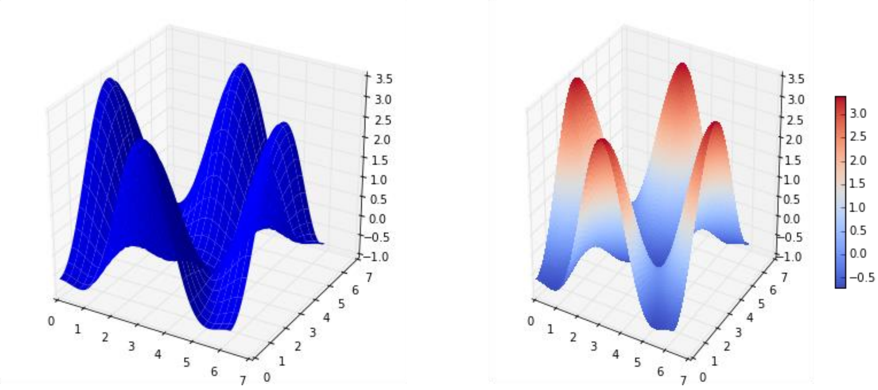


In [15]:

```
from mpl_toolkits.mplot3d.axes3d import Axes3D
```

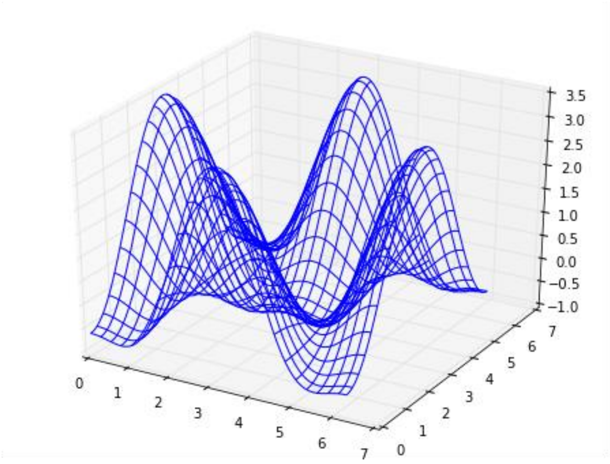
In [16]:

```
# surface plots
fig = plt.figure(figsize=(14,6))

# `ax` is a 3D-aware axis instance because of the projection='3d'
# keyword argument to add_subplot
ax = fig.add_subplot(1, 2, 1, projection='3d')

p = ax.plot_surface(X, Y, Z, rstride=4, cstride=4, linewidth=0)

# surface_plot with color grading and color bar
ax = fig.add_subplot(1, 2, 2, projection='3d')
p = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                    cmap=cm.coolwarm, linewidth=0, antialiased=False)
cb = fig.colorbar(p, shrink=0.5)
```



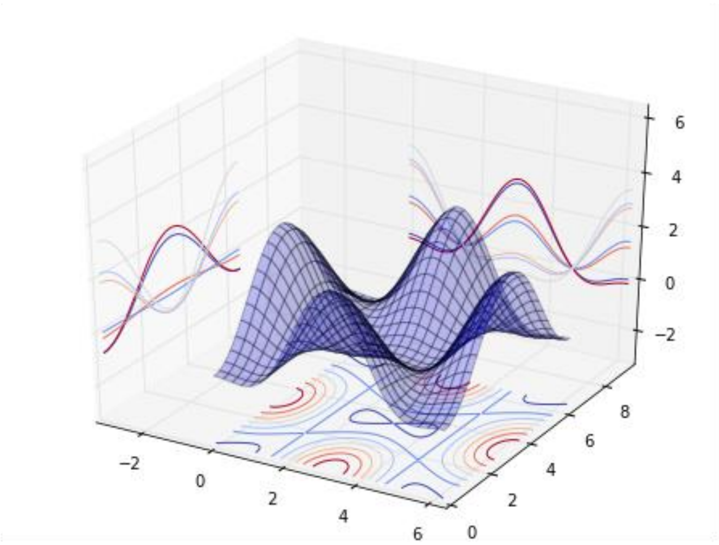
In [17]:

```
# wire frame
fig = plt.figure(figsize=(8,6))

ax = fig.add_subplot(1, 1, 1, projection='3d')

p = ax.plot_wireframe(X, Y, Z, rstride=4, cstride=4)
```



In [18]:

```
# contour plot with projections
fig = plt.figure(figsize=(8,6))

ax = fig.add_subplot(1,1,1, projection='3d')

ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
cset = ax.contour(X, Y, Z, zdir='z', offset=-pi, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='x', offset=-pi, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='y', offset=3*pi, cmap=cm.coolwarm)

ax.set_xlim3d(-pi, 2*pi);
ax.set_ylim3d(0, 3*pi);
ax.set_zlim3d(-pi, 2*pi);
```



Seaborn

Seaborn is a library for making attractive and informative statistical graphics in Python. It is built on top of matplotlib and tightly integrated with the PyData stack, including support for numpy and pandas data structures and statistical routines from scipy and statsmodels.

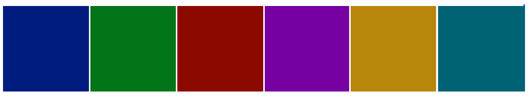
Library documentation: http://stanford.edu/~mwaskom/software/seaborn/

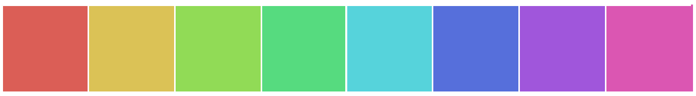
In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
%matplotlib inline
```

Themes

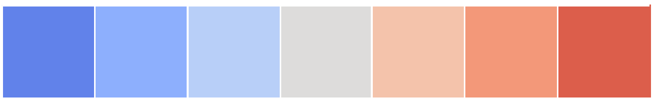
In [2]:
```
# global config settings to control things like style, font size, color palette etc.
sb.set(context="notebook", style="darkgrid", palette="dark")
```

In [3]:
```
# seaborn has some nice built-in color palette features
sb.palplot(sb.color_palette())
sb.palplot(sb.color_palette("husl", 8))
sb.palplot(sb.color_palette("hls", 8))
```

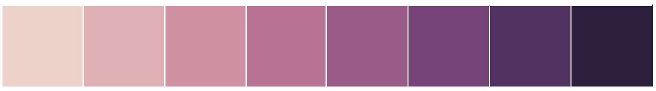




In [4]:
```
# matplotlib colormap of evenly spaced colors
sb.palplot(sb.color_palette("coolwarm", 7))
```



In [5]:
```
# sequential palette with linear increase in brightness
sb.palplot(sb.cubehelix_palette(8))
```

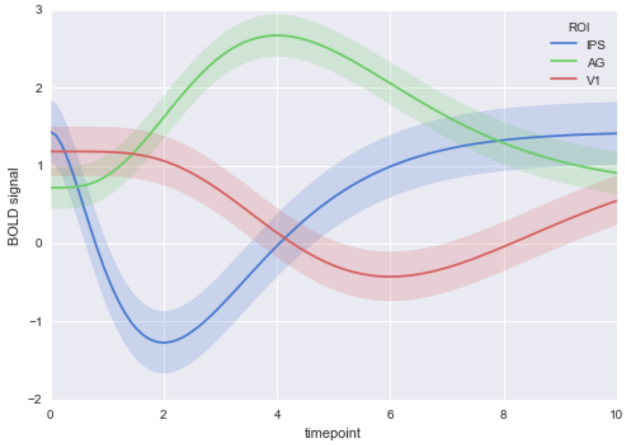


In [6]:

```
# palettes are used in a plot via the color paramter
gammas = sb.load_dataset("gammas")
sb.tsplot(gammas, "timepoint", "subject", "ROI", "BOLD signal", color="muted")
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1815b5f8>
```



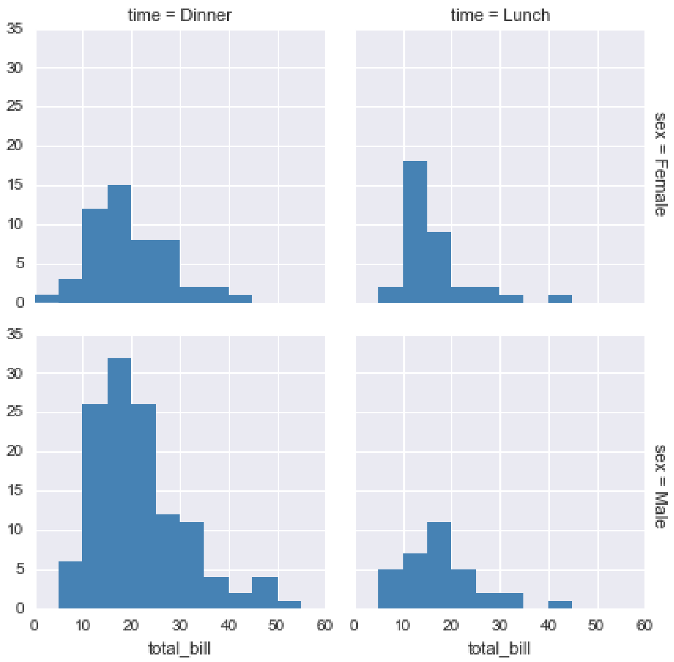
Advanced Plots

In [7]:

```
# facetting histograms by subsets of data
sb.set(style="darkgrid")

tips = sb.load_dataset("tips")
g = sb.FacetGrid(tips, row="sex", col="time", margin_titles=True)
bins = np.linspace(0, 60, 13)
g.map(plt.hist, "total_bill", color="steelblue", bins=bins, lw=0)
```

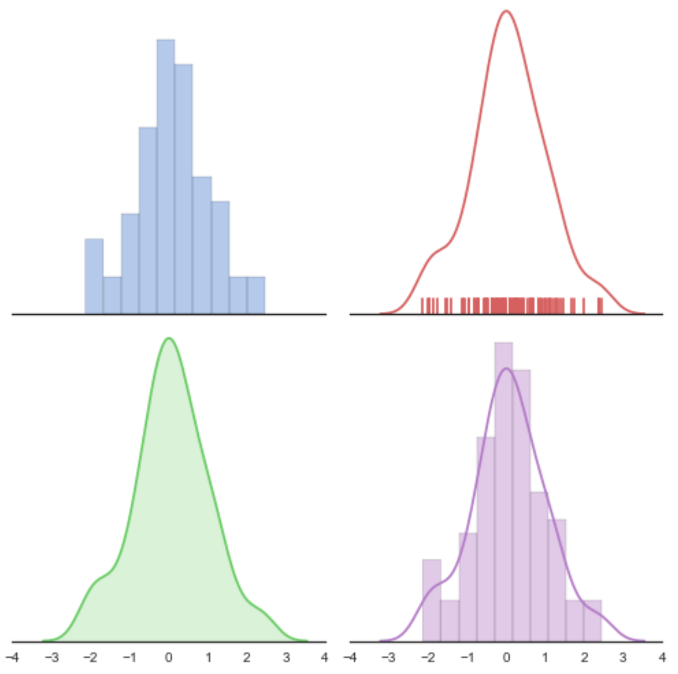
Out[7]:

```
<seaborn.axisgrid.FacetGrid at 0x1816a2e8>
```



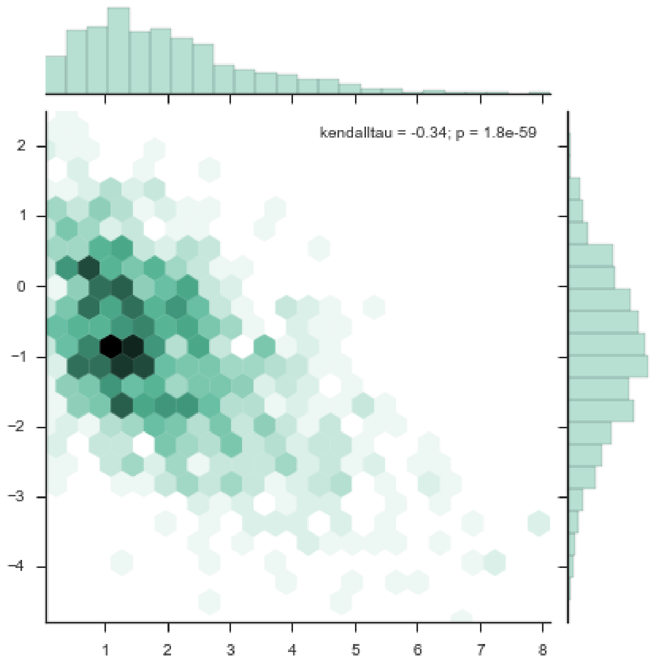
In [8]:

```
# several distribution plot examples
sb.set(style="white", palette="muted")
f, axes = plt.subplots(2, 2, figsize=(7, 7), sharex=True)
sb.despine(left=True)

rs = np.random.RandomState(10)

b, g, r, p = sb.color_palette("muted", 4)

d = rs.normal(size=100)

sb.distplot(d, kde=False, color=b, ax=axes[0, 0])
sb.distplot(d, hist=False, rug=True, color=r, ax=axes[0, 1])
sb.distplot(d, hist=False, color=g, kde_kws={"shade": True}, ax=axes[1, 0])
sb.distplot(d, color=p, ax=axes[1, 1])

plt.setp(axes, yticks=[])
plt.tight_layout()
```



```
In [9]:
# hexbin plot with marginal distributions
from scipy.stats import kendalltau
sb.set(style="ticks")

rs = np.random.RandomState(11)
x = rs.gamma(2, size=1000)
y = -.5 * x + rs.normal(size=1000)
sb.jointplot(x, y, kind="hex", stat_func=kendalltau, color="#4CB391")
```

```
Out[9]:
<seaborn.axisgrid.JointGrid at 0x19267550>
```

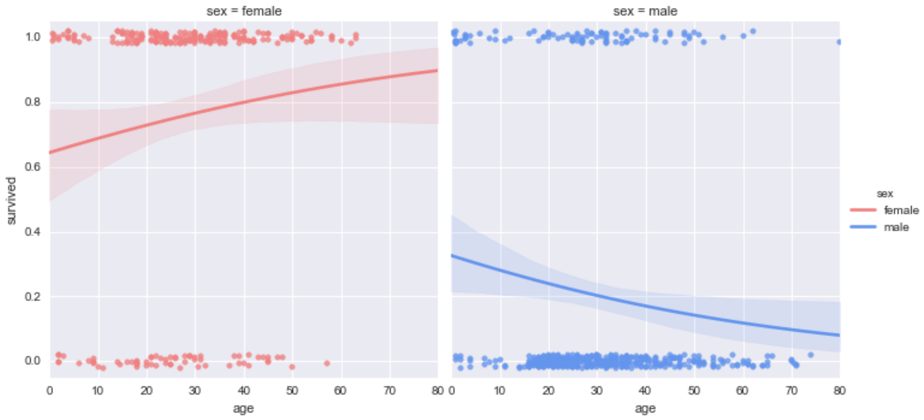



```
In [10]:
# faceted logistic regression
```

```
sb.set(style="darkgrid")
df = sb.load_dataset("titanic")

pal = dict(male="#6495ED", female="#F08080")
g = sb.lmplot("age", "survived", col="sex", hue="sex", data=df,
               palette=pal, y_jitter=.02, logistic=True)
g.set(xlim=(0, 80), ylim=(-.05, 1.05))
```

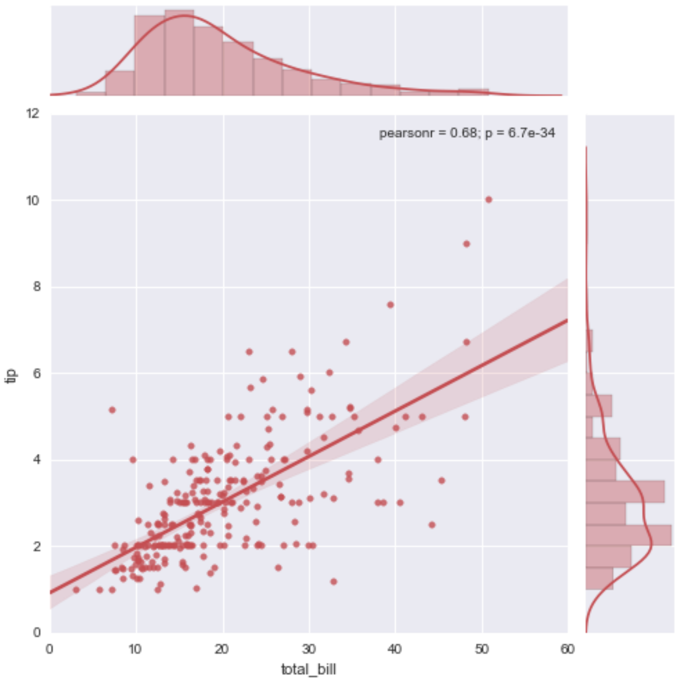
Out[10]:

```
<seaborn.axisgrid.FacetGrid at 0x1a053f98>
```



In [11]:

```
# linear regression with marginal distributions
sb.set(style="darkgrid")
tips = sb.load_dataset("tips")
color = sb.color_palette()[2]
g = sb.jointplot("total_bill", "tip", data=tips, kind="reg",
                  xlim=(0, 60), ylim=(0, 12), color=color, size=7)
```



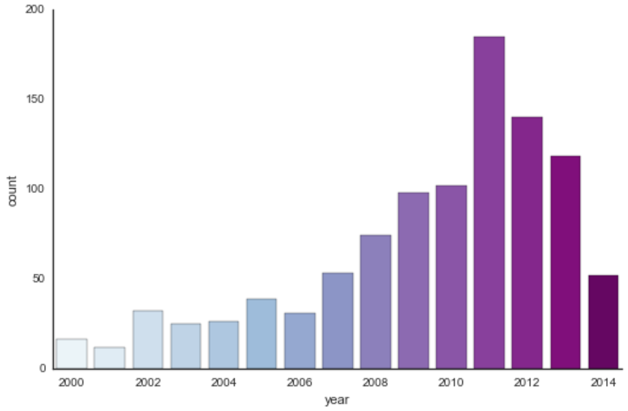
In [12]:

```
# time series factor plot
sb.set(style="white")
planets = sb.load_dataset("planets")
```

```
years = np.arange(2000, 2015)
g = sb.factorplot("year", data=planets, palette="BuPu",
                  aspect=1.5, x_order=years)
g.set_xticklabels(step=2)
```

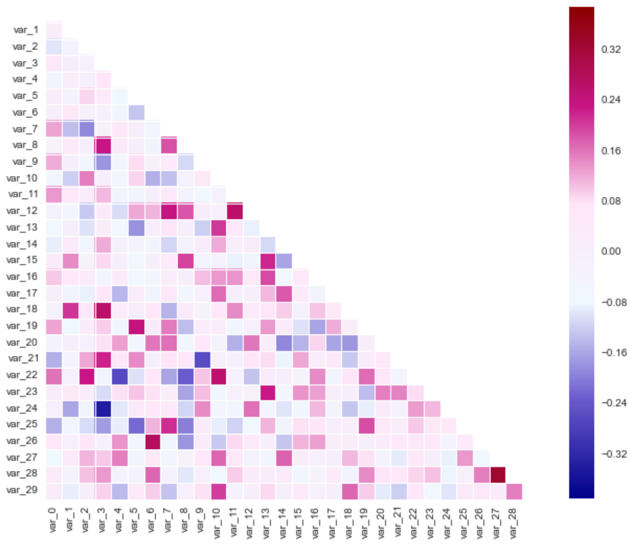
Out[12]:

```
<seaborn.axisgrid.FacetGrid at 0x1ab42518>
```



In [13]:

```
# correlation matrix
sb.set(style="darkgrid")

rs = np.random.RandomState(33)
d = rs.normal(size=(100, 30))

f, ax = plt.subplots(figsize=(9, 9))
cmap = sb.blend_palette(["#00008B", "#6A5ACD", "#F0F8FF",
                          "#FFE6F8", "#C71585", "#8B0000"], as_cmap=True)
sb.corrplot(d, annot=False, sig_stars=False,
             diag_names=False, cmap=cmap, ax=ax)
f.tight_layout()
```

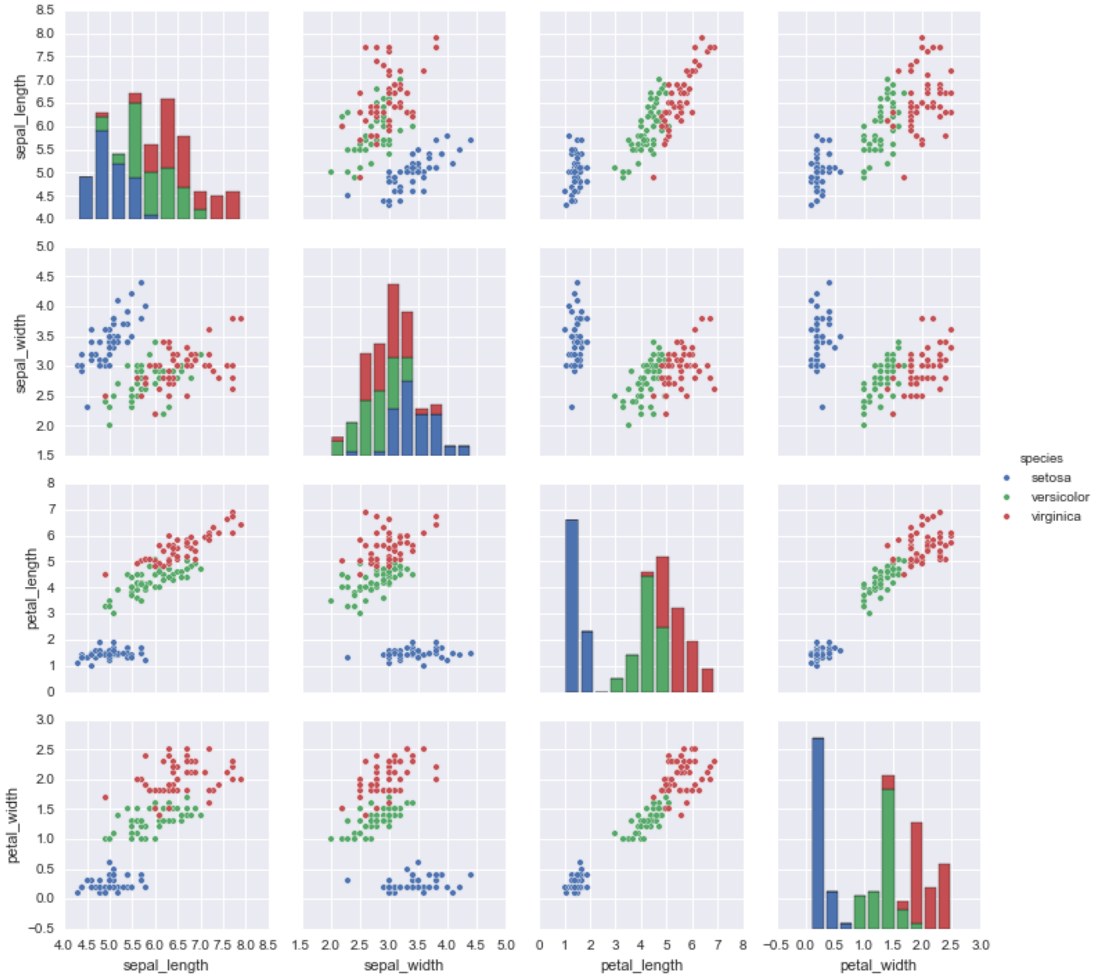


In [14]:

```
# pair plot example
sb.set(style="darkgrid")
df = sb.load_dataset("iris")
sb.pairplot(df, hue="species", size=2.5)
```

Out[14]:

<seaborn.axisgrid.PairGrid at 0x1a7cd4e0>



Pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Library documentation: http://pandas.pydata.org/

General

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
# create a series
s = pd.Series([1,3,5,np.nan,6,8])
s
```

Out[2]:

```
0      1
1      3
2      5
3    NaN
4      6
5      8
dtype: float64
```

In [3]:

```
# create a data frame
dates = pd.date_range('20130101',periods=6)
df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('ABCD'))
df
```

Out[3]:

	A	B	C	D
2013-01-01	0.205240	0.527603	0.610052	0.469292
2013-01-02	0.818113	-0.894390	-1.602831	0.862170
2013-01-03	-1.462109	0.483201	-1.044973	-0.534227
2013-01-04	0.719197	-0.499809	1.145788	-0.809526
2013-01-05	-1.161051	-0.115774	-0.624413	0.474422
2013-01-06	0.000782	0.146544	0.033628	-0.419772

In [4]:

```
# another way to create a data frame
df2 = pd.DataFrame(
    { 'A' : 1.,
      'B' : pd.Timestamp('20130102'),
      'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
      'D' : np.array([3] * 4,dtype='int32'),
      'E' : 'foo' })
df2
```

Out[4]:

	A	B	C	D	E
0	1	2013-01-02	1	3	foo
1	1	2013-01-02	1	3	foo
2	1	2013-01-02	1	3	foo
3	1	2013-01-02	1	3	foo

In [5]:

```
df2.dtypes
```

Out[5]:

```
A           float64
B    datetime64[ns]
C           float32
D             int32
E            object
dtype: object
```

In [6]:

```
df.head()
```

Out[6]:

	A	B	C	D
2013-01-01	0.205240	0.527603	0.610052	0.469292
2013-01-02	0.818113	-0.894390	-1.602831	0.862170
2013-01-03	-1.462109	0.483201	-1.044973	-0.534227
2013-01-04	0.719197	-0.499809	1.145788	-0.809526
2013-01-05	-1.161051	-0.115774	-0.624413	0.474422

In [7]:

```
df.index
```

Out[7]:

```
<class 'pandas.tseries.index.DatetimeIndex'>
[2013-01-01, ..., 2013-01-06]
Length: 6, Freq: D, Timezone: None
```

In [8]:

```
df.columns
```

Out[8]:

```
Index([u'A', u'B', u'C', u'D'], dtype='object')
```

In [9]:

```
df.values
```

Out[9]:

```
array([[  2.05240362e-01,   5.27602841e-01,   6.10052272e-01,
          4.69292270e-01],
       [  8.18112883e-01,  -8.94389618e-01,  -1.60283098e+00,
          8.62169894e-01],
       [ -1.46210940e+00,   4.83201108e-01,  -1.04497297e+00,
         -5.34226832e-01],
       [  7.19196807e-01,  -4.99809344e-01,   1.14578824e+00,
         -8.09525609e-01],
       [ -1.16105080e+00,  -1.15774007e-01,  -6.24412514e-01,
          4.74421893e-01],
       [  7.82298420e-04,   1.46543576e-01,   3.36282758e-02,
         -4.19771560e-01]])
```

In [10]:

```
# quick data summary
df.describe()
```

Out[10]:

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.146638	-0.058771	-0.247125	0.007060
std	0.957650	0.561381	1.036400	0.679012
min	-1.462109	-0.894390	-1.602831	-0.809526
25%	-0.870593	-0.403801	-0.939833	-0.505613
50%	0.103011	0.015385	-0.295392	0.024760
75%	0.590708	0.399037	0.465946	0.473139
max	0.818113	0.527603	1.145788	0.862170

In [11]:

```
df.T
```

Out[11]:

	2013-01-01 00:00:00	2013-01-02 00:00:00	2013-01-03 00:00:00	2013-01-04 00:00:00	2013-01-05 00:00:00	2013-01-06 00:00:00
A	0.205240	0.818113	-1.462109	0.719197	-1.161051	0.000782
B	0.527603	-0.894390	0.483201	-0.499809	-0.115774	0.146544

	2013-01-01 00:00:00	2013-01-02 00:00:00	2013-01-03 00:00:00	2013-01-04 00:00:00	2013-01-05 00:00:00	2013-01-06 00:00:00
C	0.610052	-1.602831	-1.044973	1.145788	-0.624413	0.033628
D	0.469292	0.862170	-0.534227	-0.809526	0.474422	-0.419772

In [12]:
```
# axis 0 is index, axis 1 is columns
df.sort_index(axis=1, ascending=False)
```
Out[12]:

	D	C	B	A
2013-01-01	0.469292	0.610052	0.527603	0.205240
2013-01-02	0.862170	-1.602831	-0.894390	0.818113
2013-01-03	-0.534227	-1.044973	0.483201	-1.462109
2013-01-04	-0.809526	1.145788	-0.499809	0.719197
2013-01-05	0.474422	-0.624413	-0.115774	-1.161051
2013-01-06	-0.419772	0.033628	0.146544	0.000782

In [13]:
```
# can sort by values too
df.sort(columns='B')
```
Out[13]:

	A	B	C	D
2013-01-02	0.818113	-0.894390	-1.602831	0.862170
2013-01-04	0.719197	-0.499809	1.145788	-0.809526
2013-01-05	-1.161051	-0.115774	-0.624413	0.474422
2013-01-06	0.000782	0.146544	0.033628	-0.419772
2013-01-03	-1.462109	0.483201	-1.044973	-0.534227
2013-01-01	0.205240	0.527603	0.610052	0.469292

Selection

In [14]:
```
# select a column (yields a series)
df['A']
```
Out[14]:
```
2013-01-01    0.205240
2013-01-02    0.818113
2013-01-03   -1.462109
2013-01-04    0.719197
2013-01-05   -1.161051
2013-01-06    0.000782
```

```
Freq: D, Name: A, dtype: float64
```

In [15]:

```
# column names also attached to the object
df.A
```

Out[15]:

```
2013-01-01    0.205240
2013-01-02    0.818113
2013-01-03   -1.462109
2013-01-04    0.719197
2013-01-05   -1.161051
2013-01-06    0.000782
Freq: D, Name: A, dtype: float64
```

In [16]:

```
# slicing works
df[0:3]
```

Out[16]:

	A	B	C	D
2013-01-01	0.205240	0.527603	0.610052	0.469292
2013-01-02	0.818113	-0.894390	-1.602831	0.862170
2013-01-03	-1.462109	0.483201	-1.044973	-0.534227

In [17]:

```
df['20130102':'20130104']
```

Out[17]:

	A	B	C	D
2013-01-02	0.818113	-0.894390	-1.602831	0.862170
2013-01-03	-1.462109	0.483201	-1.044973	-0.534227
2013-01-04	0.719197	-0.499809	1.145788	-0.809526

In [18]:

```
# cross-section using a label
df.loc[dates[0]]
```

Out[18]:

```
A    0.205240
B    0.527603
C    0.610052
D    0.469292
Name: 2013-01-01 00:00:00, dtype: float64
```

In [19]:
```
# getting a scalar value
df.loc[dates[0], 'A']
```
Out[19]:
```
0.20524036189008577
```

In [20]:
```
# select via position
df.iloc[3]
```
Out[20]:
```
A    0.719197
B   -0.499809
C    1.145788
D   -0.809526
Name: 2013-01-04 00:00:00, dtype: float64
```

In [21]:
```
df.iloc[3:5,0:2]
```
Out[21]:

	A	B
2013-01-04	0.719197	-0.499809
2013-01-05	-1.161051	-0.115774

In [22]:
```
# column slicing
df.iloc[:,1:3]
```
Out[22]:

	B	C
2013-01-01	0.527603	0.610052
2013-01-02	-0.894390	-1.602831
2013-01-03	0.483201	-1.044973
2013-01-04	-0.499809	1.145788
2013-01-05	-0.115774	-0.624413
2013-01-06	0.146544	0.033628

In [23]:
```
# get a value by index
df.iloc[1,1]
```
Out[23]:
```
-0.89438961765370562
```

In [24]:
```
# boolean indexing
df[df.A > 0]
```

Out[24]:

	A	B	C	D
2013-01-01	0.205240	0.527603	0.610052	0.469292
2013-01-02	0.818113	-0.894390	-1.602831	0.862170
2013-01-04	0.719197	-0.499809	1.145788	-0.809526
2013-01-06	0.000782	0.146544	0.033628	-0.419772

In [25]:
```
df[df > 0]
```
Out[25]:

	A	B	C	D
2013-01-01	0.205240	0.527603	0.610052	0.469292
2013-01-02	0.818113	NaN	NaN	0.862170
2013-01-03	NaN	0.483201	NaN	NaN
2013-01-04	0.719197	NaN	1.145788	NaN
2013-01-05	NaN	NaN	NaN	0.474422
2013-01-06	0.000782	0.146544	0.033628	NaN

In [26]:
```
# filtering
df3 = df.copy()
df3['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
df3[df3['E'].isin(['two', 'four'])]
```
Out[26]:

	A	B	C	D	E
2013-01-03	-1.462109	0.483201	-1.044973	-0.534227	two
2013-01-05	-1.161051	-0.115774	-0.624413	0.474422	four

In [27]:
```
# setting examples
df.at[dates[0],'A'] = 0
df.iat[0,1] = 0
df.loc[:, 'D'] = np.array([5] * len(df))
df
```
Out[27]:

	A	B	C	D
2013-01-01	0.000000	0.000000	0.610052	5
2013-01-02	0.818113	-0.894390	-1.602831	5

	A	B	C	D
2013-01-03	-1.462109	0.483201	-1.044973	5
2013-01-04	0.719197	-0.499809	1.145788	5
2013-01-05	-1.161051	-0.115774	-0.624413	5
2013-01-06	0.000782	0.146544	0.033628	5

In [28]:
```
# dealing with missing data
df4 = df.reindex(index=dates[0:4],columns=list(df.columns) + ['E'])
df4.loc[dates[0]:dates[1],'E'] = 1
df4
```
Out[28]:

	A	B	C	D	E
2013-01-01	0.000000	0.000000	0.610052	5	1
2013-01-02	0.818113	-0.894390	-1.602831	5	1
2013-01-03	-1.462109	0.483201	-1.044973	5	NaN
2013-01-04	0.719197	-0.499809	1.145788	5	NaN

In [29]:
```
# drop rows with missing data
df4.dropna(how='any')
```
Out[29]:

	A	B	C	D	E
2013-01-01	0.000000	0.00000	0.610052	5	1
2013-01-02	0.818113	-0.89439	-1.602831	5	1

In [30]:
```
# fill missing data
df4.fillna(value=5)
```
Out[30]:

	A	B	C	D	E
2013-01-01	0.000000	0.000000	0.610052	5	1
2013-01-02	0.818113	-0.894390	-1.602831	5	1
2013-01-03	-1.462109	0.483201	-1.044973	5	5
2013-01-04	0.719197	-0.499809	1.145788	5	5

In [31]:
```
# boolean mask for nan values
pd.isnull(df4)
```
Out[31]:

	A	B	C	D	E
2013-01-01	False	False	False	False	False
2013-01-02	False	False	False	False	False
2013-01-03	False	False	False	False	True
2013-01-04	False	False	False	False	True

Operations

In [32]:
```
df.mean()
```
Out[32]:
```
A   -0.180845
B   -0.146705
C   -0.247125
D    5.000000
dtype: float64
```

In [33]:
```
# pivot the mean calculation
df.mean(1)
```
Out[33]:
```
2013-01-01    1.402513
2013-01-02    0.830223
2013-01-03    0.744030
2013-01-04    1.591294
2013-01-05    0.774691
2013-01-06    1.295239
Freq: D, dtype: float64
```

In [34]:
```
# aligning objects with different dimensions
s = pd.Series([1,3,5,np.nan,6,8],index=dates).shift(2)
df.sub(s,axis='index')
```
Out[34]:

	A	B	C	D
2013-01-01	NaN	NaN	NaN	NaN
2013-01-02	NaN	NaN	NaN	NaN
2013-01-03	-2.462109	-0.516799	-2.044973	4
2013-01-04	-2.280803	-3.499809	-1.854212	2
2013-01-05	-6.161051	-5.115774	-5.624413	0

	A	B	C	D
2013-01-06	NaN	NaN	NaN	NaN

In [35]:

```
# applying functions
df.apply(np.cumsum)
```

Out[35]:

	A	B	C	D
2013-01-01	0.000000	0.000000	0.610052	5
2013-01-02	0.818113	-0.894390	-0.992779	10
2013-01-03	-0.643997	-0.411189	-2.037752	15
2013-01-04	0.075200	-0.910998	-0.891963	20
2013-01-05	-1.085851	-1.026772	-1.516376	25
2013-01-06	-1.085068	-0.880228	-1.482748	30

In [36]:

```
df.apply(lambda x: x.max() - x.min())
```

Out[36]:

```
A     2.280222
B     1.377591
C     2.748619
D     0.000000
dtype: float64
```

In [37]:

```
# simple count aggregation
s = pd.Series(np.random.randint(0,7,size=10))
s.value_counts()
```

Out[37]:

```
4     3
6     2
1     2
0     2
5     1
dtype: int64
```

Merging / Grouping / Shaping

In [38]:

```
# concatenation
df = pd.DataFrame(np.random.randn(10, 4))
pieces = [df[:3], df[3:7], df[7:]]
pd.concat(pieces)
```

Out[38]:

	0	1	2	3
0	-0.006589	-1.232048	-0.147323	0.709050
1	-1.201048	0.675688	1.110037	0.553489
2	-0.159224	-1.226735	-0.141689	-1.450920
3	-0.049450	-0.438565	0.670832	1.089032
4	-0.105969	-0.891644	0.626482	0.416679
5	-1.103222	-1.983806	0.282366	0.031730
6	0.380308	-0.397791	-0.322955	0.074480
7	-0.623134	-0.205967	-0.367622	1.437279
8	-0.481202	1.242607	-2.107715	1.020051
9	-0.345859	-0.759047	-0.927940	1.487916

In [39]:

```
# SQL-style join
left = pd.DataFrame({'key': ['foo', 'foo'], 'lval': [1, 2]})
right = pd.DataFrame({'key': ['foo', 'foo'], 'rval': [4, 5]})
pd.merge(left, right, on='key')
```

Out[39]:

	key	lval	rval
0	foo	1	4
1	foo	1	5
2	foo	2	4
3	foo	2	5

In [40]:

```
# append
df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
s = df.iloc[3]
df.append(s, ignore_index=True)
```

Out[40]:

	A	B	C	D
0	-0.992219	1.298979	0.998799	-0.164381
1	0.902147	1.118289	-0.169358	0.117833
2	1.201061	-1.699020	-2.112810	-1.412482
3	1.084910	1.171135	0.384876	0.535239

	A	B	C	D
4	-0.922543	-0.018670	-1.506012	0.293739
5	0.481017	0.639182	-0.090676	0.951261
6	1.201241	2.528836	-0.530795	0.901950
7	0.899290	0.562738	1.566468	-0.846827
8	1.084910	1.171135	0.384876	0.535239

In [41]:
```
df = pd.DataFrame(
    { 'A' : ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'foo'],
      'B' : ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
      'C' : np.random.randn(8),
      'D' : np.random.randn(8) })
df
```
Out[41]:

	A	B	C	D
0	foo	one	0.193948	-1.385614
1	bar	one	-0.257859	2.127808
2	foo	two	-0.944848	-0.760487
3	bar	three	-0.872161	-1.707254
4	foo	two	-0.658552	0.175699
5	bar	two	-1.887614	0.627801
6	foo	one	0.439001	-2.264125
7	foo	three	-0.829368	-1.229315

In [42]:
```
# group by
df.groupby('A').sum()
```
Out[42]:

	C	D
A		
bar	-3.017634	1.048355
foo	-1.799818	-5.463842

In [43]:

```
# group by multiple columns
df.groupby(['A','B']).sum()
```

Out[43]:

A	B	C	D
bar	one	-0.257859	2.127808
	three	-0.872161	-1.707254
	two	-1.887614	0.627801
foo	one	0.632949	-3.649739
	three	-0.829368	-1.229315
	two	-1.603400	-0.584788

In [44]:

```
df = pd.DataFrame(
    { 'A' : ['one', 'one', 'two', 'three'] * 3,
      'B' : ['A', 'B', 'C'] * 4,
      'C' : ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 2,
      'D' : np.random.randn(12),
      'E' : np.random.randn(12)} )
df
```

Out[44]:

	A	B	C	D	E
0	one	A	foo	-0.853288	2.549878
1	one	B	foo	0.552557	0.865465
2	two	C	foo	0.700943	0.800563
3	three	A	bar	-0.466072	0.011508
4	one	B	bar	0.465724	1.087874
5	one	C	bar	1.105949	-0.118134
6	two	A	foo	-0.666630	-0.143474
7	three	B	foo	0.644902	1.731818
8	one	C	foo	0.819170	-1.153036
9	one	A	bar	-1.849893	0.733137
10	two	B	bar	0.684170	-0.276237
11	three	C	bar	0.592939	-0.830433

In [45]:
```
# pivot table
pd.pivot_table(df, values='D', rows=['A', 'B'], columns=['C'])
```
```
C:\Program Files\Anaconda\lib\site-packages\pandas\util\decorators.py:53: FutureWa
rning: rows is deprecated, use index instead
  warnings.warn(msg, FutureWarning)
```
Out[45]:

	C	bar	foo
A	**B**		
one	**A**	-1.849893	-0.853288
	B	0.465724	0.552557
	C	1.105949	0.819170
three	**A**	-0.466072	NaN
	B	NaN	0.644902
	C	0.592939	NaN
two	**A**	NaN	-0.666630
	B	0.684170	NaN
	C	NaN	0.700943

Time Series

In [46]:
```
# time period resampling
rng = pd.date_range('1/1/2012', periods=100, freq='S')
ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)
ts.resample('5Min', how='sum')
```
Out[46]:
```
2012-01-01    24406
Freq: 5T, dtype: int32
```

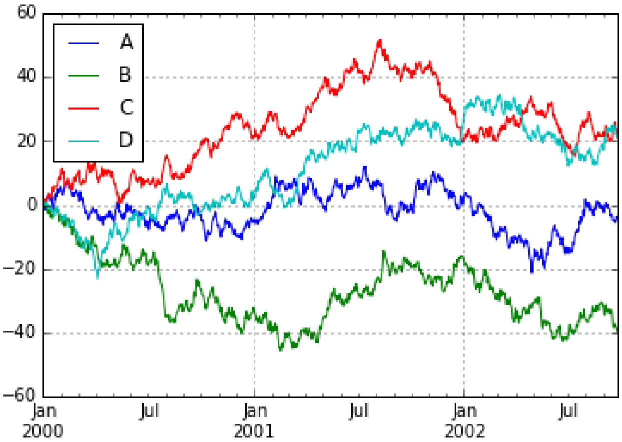
In [47]:
```
rng = pd.date_range('1/1/2012', periods=5, freq='M')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```
Out[47]:
```
2012-01-31   -0.624893
2012-02-29   -0.176292
2012-03-31    1.673556
2012-04-30    0.707903
2012-05-31    0.533647
Freq: M, dtype: float64
```

```
In [48]:
ps = ts.to_period()
ps.to_timestamp()
Out[48]:
2012-01-01   -0.624893
2012-02-01   -0.176292
2012-03-01    1.673556
2012-04-01    0.707903
2012-05-01    0.533647
Freq: MS, dtype: float64
```

Plotting

```
In [49]:
# time series plot
ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000
))
ts = ts.cumsum()
ts.plot()
Out[49]:
<matplotlib.axes._subplots.AxesSubplot at 0xd180438>
```


```
In [50]:
# plot with a data frame
df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=['A', 'B', 'C'
, 'D'])
df = df.cumsum()
plt.figure(); df.plot(); plt.legend(loc='best')
Out[50]:
<matplotlib.legend.Legend at 0xd541fd0>
<matplotlib.figure.Figure at 0xd554550>
```



Input / Output

```
In [51]:
# write to a csv file
df.to_csv('foo.csv', index=False)
```

```
In [52]:
# read file back in
path = r'C:\Users\John\Documents\IPython Notebooks\foo.csv'
newDf = pd.read_csv(path)
newDf.head()
```

Out[52]:

	A	B	C	D
0	-0.914956	0.294759	0.143332	0.174706
1	-0.297442	1.640208	0.425301	-0.075666
2	-0.762292	0.741179	0.505002	-0.128560
3	-1.577471	-0.495294	1.803332	0.188178
4	-0.137486	-0.676985	1.435308	0.181047

```
In [53]:
# remove the file
import os
os.remove(path)
```

```
In [54]:
# can also do Excel
df.to_excel('foo.xlsx', sheet_name='Sheet1')
```

```
In [55]:
newDf2 = pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
newDf2.head()
```

Out[55]:

	A	B	C	D
2000-01-01	-0.914956	0.294759	0.143332	0.174706
2000-01-02	-0.297442	1.640208	0.425301	-0.075666
2000-01-03	-0.762292	0.741179	0.505002	-0.128560
2000-01-04	-1.577471	-0.495294	1.803332	0.188178
2000-01-05	-0.137486	-0.676985	1.435308	0.181047

```
In [56]:
os.remove('foo.xlsx')
```