Prob_and_RV_Demo

August 21, 2018

Contents


```
In [1]: %pylab
        %matplotlib inline
        import scipy.stats as stats
```

```
Using matplotlib backend: MacOSX
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: pylab.rcParams['savefig.dpi'] = 100 # default 72
        #pylab.rcParams['figure.figsize'] = (6.0, 4.0) # default (6,4)
        #%config InlineBackend.figure_formats=['png'] # default for inline viewing
        %config InlineBackend.figure_formats=['svg'] # SVG inline viewing
        #%config InlineBackend.figure_formats=['pdf'] # render pdf figs for LaTeX
```

Probability and Random Variables

In this notebook you will learn about some of the capabilities of the Scipy stats package. This package shows off some of the great object oriented capabilities of Python.

```
In [5]: #dir(stats)
```

Bernoulli Trials

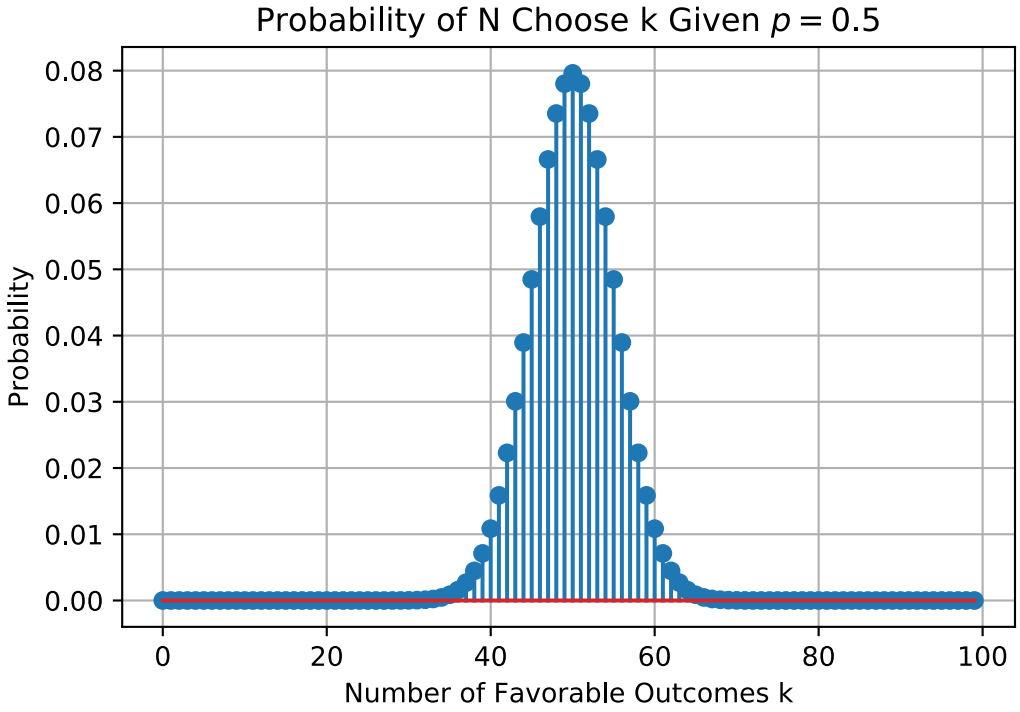
The Scipy package `stats` contains a wealth of probability and random variables functionality. In the Chapter 3 notes Bernoulli trials was discussed. Using the `stats` package you can play with the `binom` class whose instances allow calculation and simulation binomial random variables, which are closely related to the Bernoulli trials probability

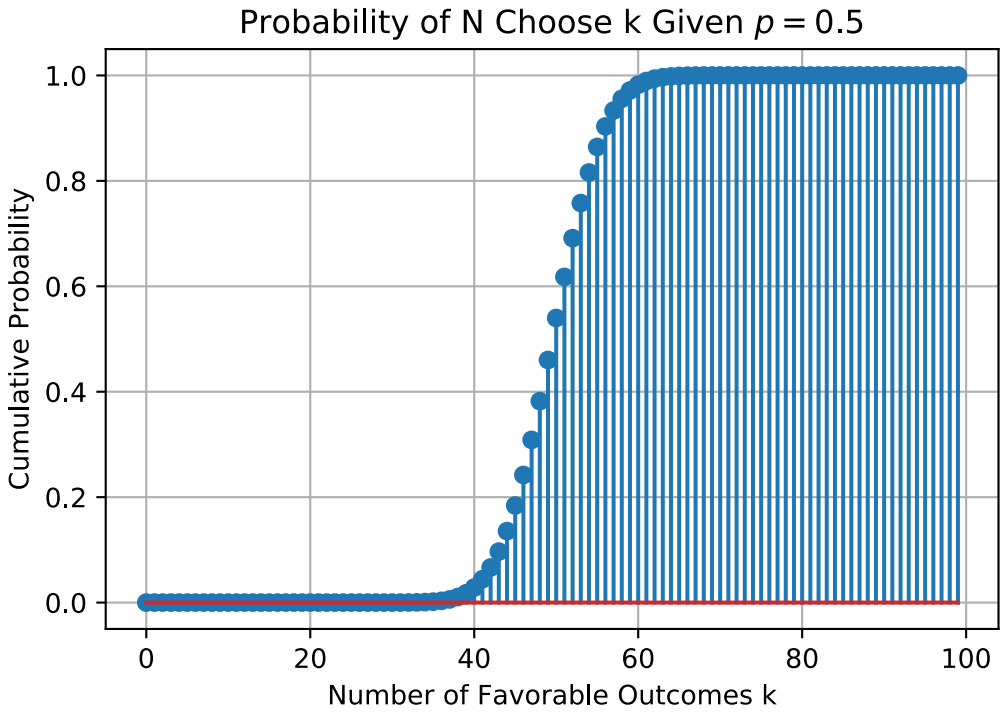
$$p_n(k) = \binom{n}{k} p^k q^{n-k}. \tag{1}$$

With `stats` the object is created (*instantiated*) using `x_rv = stats.binom(n,p)`. You can then use the the many functions (*methods*) available for the object instance, for example `pdf`, `cdf`, `mean`, `var`, or `rvs` to create random variates.

```
In [6]: x_binom = stats.binom(100,.5)
        k = arange(0,100)
        stem(k,x_binom.pmf(k))
        grid()
        xlabel('Number of Favorable Outcomes k')
        ylabel('Probability')
        title('Probability of N Choose k Given $p = 0.5$')
        figure()
        stem(k,x_binom.cdf(k))
        grid()
        xlabel('Number of Favorable Outcomes k')
        ylabel('Cumulative Probability')
        title('Probability of N Choose k Given $p = 0.5$')
```

```
Out[6]: Text(0.5,1,'Probability of N Choose k Given $p = 0.5$')
```






```
In [7]: (x_binom.mean(),x_binom.var())
```

```
Out[7]: (50.0, 25.0)
```

```
In [8]: x_rvs= x_binom.rvs(10000)
        x_rvs
```

```
Out[8]: array([47, 41, 50, ..., 53, 50, 54])
```

```
In [9]: (mean(x_rvs), var(x_rvs))
```

```
Out[9]: (50.038200000000003, 24.865340759999999)
```

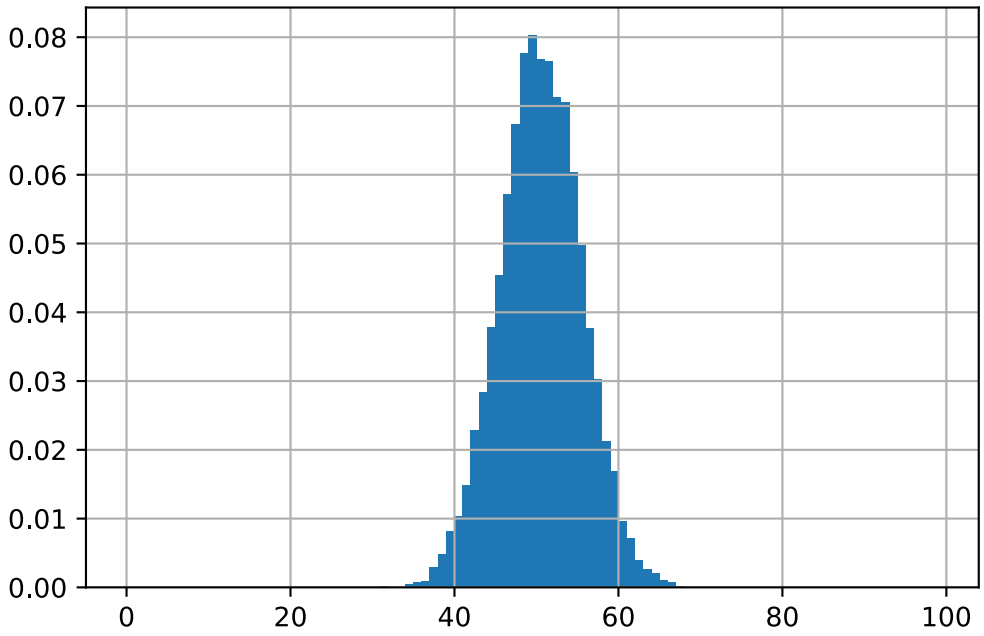
```
In [10]: var(x_rvs)
```

```
Out[10]: 24.865340759999999
```

Histogram of the Random Variates

It is easy to display a PDF normized histogram of the `x_rvs` trials:

```
In [16]: hist(x_rvs,bins=arange(0,100,1),density=True);
         grid()
```



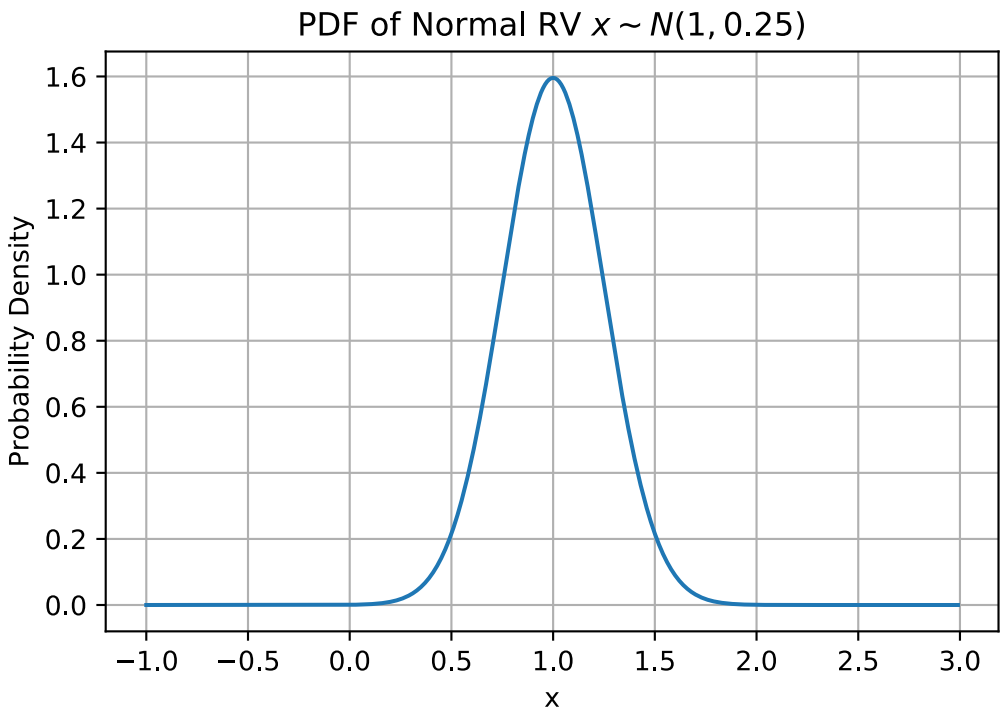
Normal Random Variable

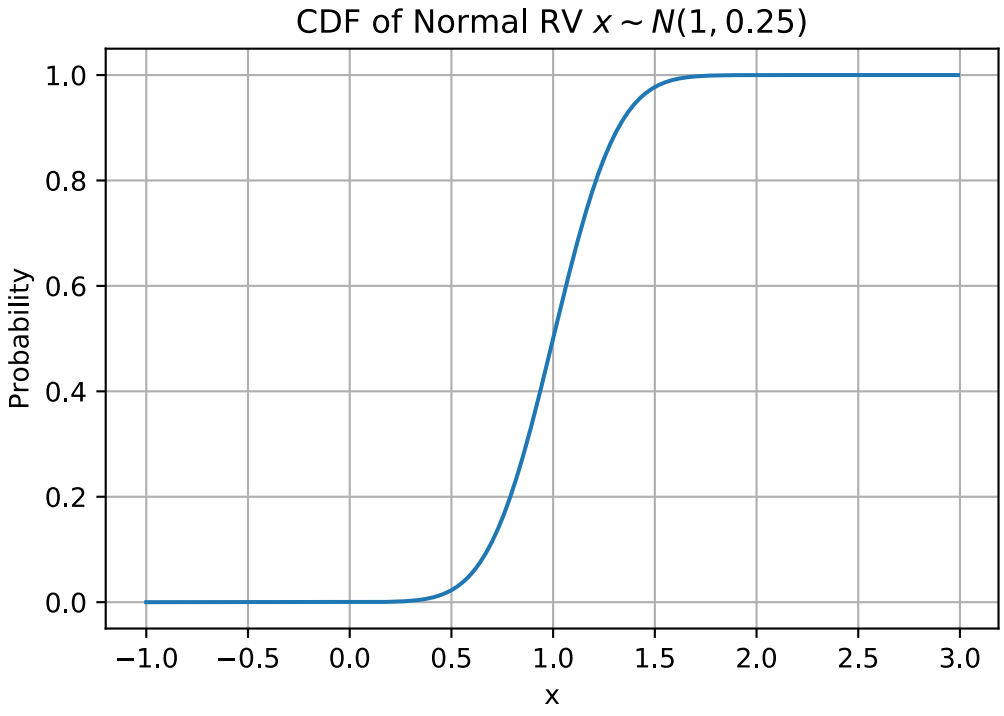
A normal random variable object can be created by supplying its mean and variance, i.e., $x_{nrv} \sim N(m, \sigma^2)$. Using the `stats` package is similar:

```
In [18]: x_grv = stats.norm(1.0,0.25)
```

Plot the PDF and CDF over the interval $-1 \leq x \leq 2$:

```
In [19]: x = arange(-1,3,.01)
         plot(x,x_grv.pdf(x));
         grid()
         xlabel('x')
         ylabel('Probability Density')
         title('PDF of Normal RV $x\sim N(1,0.25)$');
         figure()
         plot(x,x_grv.cdf(x));
         grid()
         xlabel('x')
         ylabel('Probability')
         title('CDF of Normal RV $x\sim N(1,0.25)$');
```


PDF of Normal RV $x \sim N(1, 0.25)$
Probability Density
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x


CDF of Normal RV $x \sim N(1, 0.25)$
Probability
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
x

A Simple Calculation Find the probability that rv $x \in [1.5, 2]$ using the CDF:

```
In [20]: x_grv.cdf(2) - x_grv.cdf(1.5)
```

```
Out[20]: 0.022718460706346089
```

Q-Function and Related Calculations We frequently need to calculate

$$P_a = Q(x) \tag{2}$$

$$\gamma = Q^{-1}(P_b) \tag{3}$$

```
In [21]: # The Q-function using stats.norm.sf() = survival function
         stats.norm.sf(0)
```

```
Out[21]: 0.5
```

```
In [22]: # The Q_inv-function using the stats.norm.isf() = inverse suvival function
         stats.norm.isf(1e-6)
```

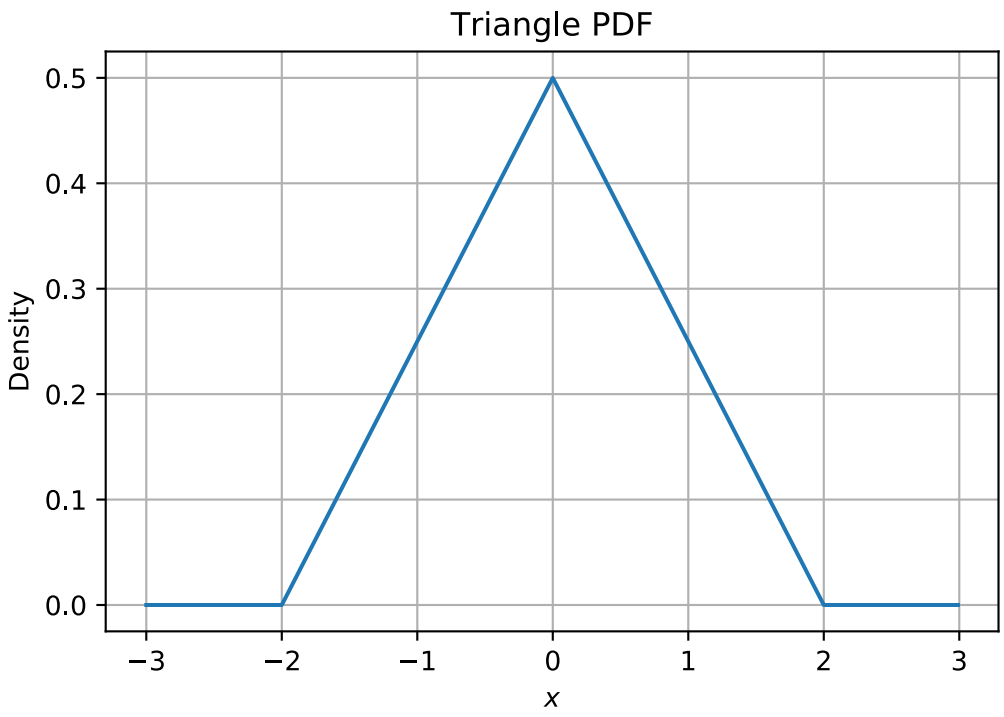
```
Out[22]: 4.7534243088228987
```

Trying Out Sympy for Some PDF Calculations

Consider a simple triangle shaped PDF, symmetrical about $x = 0$:

```
In [47]: import sk_dsp_comm.sigsys as ss
```

```
In [52]: x = arange(-3,3,.01)
         f_x = 1/2*ss.tri(x,2)
         plot(x,f_x)
         xlabel(r'$x$')
         ylabel(r'Density')
         title(r'Triangle PDF')
         grid();
```



The mathematical description of the PDF is $1/2 - x/4$ for $0 \leq x \leq 2$ and $1/2 + x/4$ for $-2 \leq x \leq 0$, or simply

$$f_x(x) = \begin{cases} \frac{1}{2} - \frac{|x|}{4}, & |x| \leq 2 \\ 0, & \text{otherwise} \end{cases}$$

To make things more interesting, consider a generalized triangle PDF with support over $-a \leq x \leq a$

$$f_x(x) = \begin{cases} \frac{1}{a}\left(1 - \frac{|x|}{a}\right), & |x| \leq a \\ 0, & \text{otherwise} \end{cases}$$

```
In [54]: # Use this cell if you are going to do symbolic calcs with sympy
         from IPython.display import display
         from sympy.interactive import printing
         printing.init_printing(use_latex='mathjax')
         import sympy as sym
```

```
In [61]: # Define a few symbolic variables
         mx,varx,a = sym.symbols("m_x,var_x,a", positive=True)
         x = sym.symbols("x",real=True)
```

Check that PDF Integrates to Unity

Piecewise integrate the PDF on the intervals $-a \leq x \leq 0$ and $0 \leq x \leq a$:

```
In [65]: area = sym.integrate(1*(1/a+x/a**2),(x,-a,0)) + sym.integrate(1*(1/a-x/a**2),(x,0,a))
         area
```

Out[65]:

$$1$$

Find the Mean

Symmtery about zero makes the mean zero:

```
In [66]: mx = sym.integrate(x*(1/a+x/a**2),(x,-a,0)) + sym.integrate(x*(1/a-x/a**2),(x,0,a))
         mx
```

Out[66]:

$$0$$

Find the Variance

Find the mean square, $E\{x^2\}$, and since $m_x = 0$ this will also be the variance:

```
In [67]: varx = sym.integrate(x**2*(1/a+x/a**2),(x,-a,0)) + sym.integrate(x**2*(1/a-x/a**2),(x
         varx
```

Out[67]:

$$\frac{a^2}{6}$$

- Consider the special case of $a = 2$, as shown in the plot above, in the variance calculation

```
In [68]: varx.subs({a:2})
```

Out[68]:

$$\frac{2}{3}$$

Bit Patterns Example from Notes Chapter 3

To do a more involved calculation with Python consider the *Bit Pattern* example (Example 3.3) from notes Chapter 3, 3-12. The probability of stopping at two ones was given by

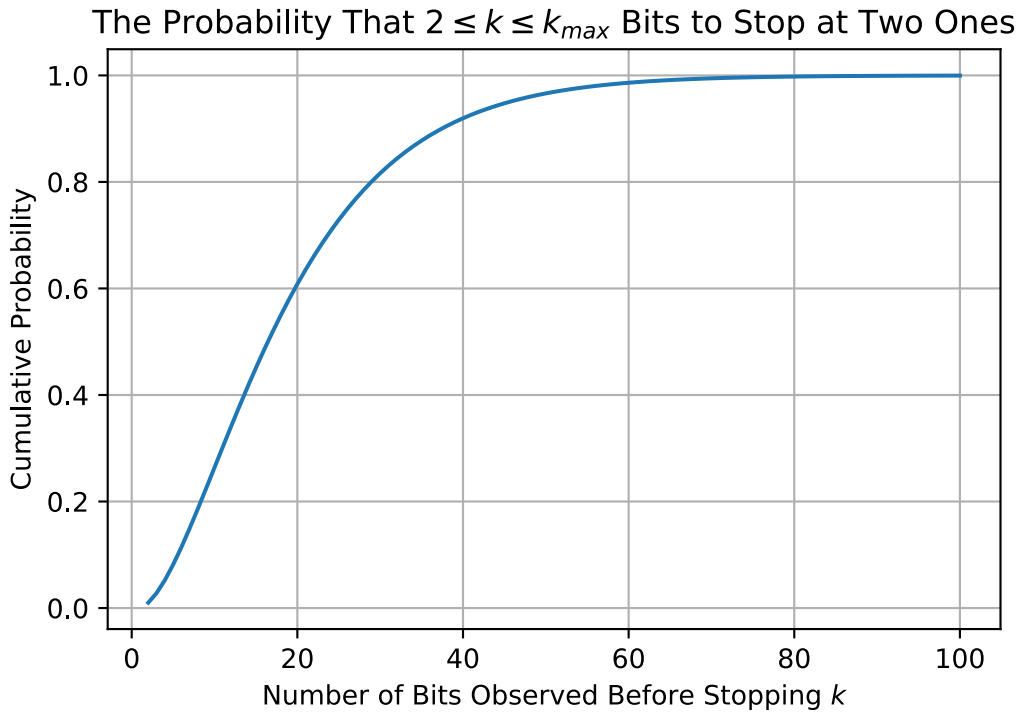
$$p(k|\text{stop at two 1s}) = (k-1)(1-p)^{k-2}p^2 \tag{4}$$

We observed that the probability that two or more bits must be observed until two ones occurs should be one, that is

$$\sum_{k=2}^{\infty} p(k|\text{stop at two 1s}) = \sum_{k=2}^{\infty}(k-1)(1-p)^{k-2}p^2 \overset{\text{must}}{=} 1 \tag{5}$$

Check this out numerically by writing the function `prob_k_stop2ones()`.

```
In [23]: def prob_k_stop2ones(k_max,n,p):
             k = arange(2,k_max+1) # note a code module with import numpy as np, need np.arang
             cum_prob = zeros(len(k)) # similarly here, np.zeros
             temp_prob = 0
             for k_index,k_value in enumerate(k): # enumerate is very handy when you write loo
                 temp_prob += (k_value-1)*(1-p)**(k_value-2)*p**2
                 cum_prob[k_index] = temp_prob
             return k, cum_prob

In [24]: k, cum_prob = prob_k_stop2ones(100,10,.1)

In [25]: plot(k,cum_prob)
         grid()
         xlabel('Number of Bits Observed Before Stopping $k$')
         ylabel('Cumulative Probability')
         title('The Probability That $2\leq k \leq k_{max}$ Bits to Stop at Two Ones');
```



Coding Observations With Python indentation is very very important. You do not ever want to use tabs. The rule is four spaces per indent. Most editors allow you to customize to this. For example *notepad++* and *Sublime Text*, and of course the editor in Canopy itself.

The `def` block allows you to define a custom function right inside a notebook. You can also write a stand alone text file (code module) with extension `*.py` to hold one or more `def`s. More on this later.

Explain the use of enumerate in a loop:

```
for k_index,k_value in enumerate(k): # enumerate is very handy when you write loops
        temp_prob += (k_value-1)*(1-p)**(k_value-2)*p**2
        cum_prob[k_index] = temp_prob
```

The enuerate function returns a two element *tuple* which contains the value pair index of its argument and the value contained at that index. The use of a formatted `print` is used below to verify what enumerate is returning in the `for` loop.

```
In [16]: u1 = randn(5)
         for k_idx, u1_val in enumerate(u1):
             print('k_idx = %d and u1_val = %2.4f' % (k_idx, u1_val))
```

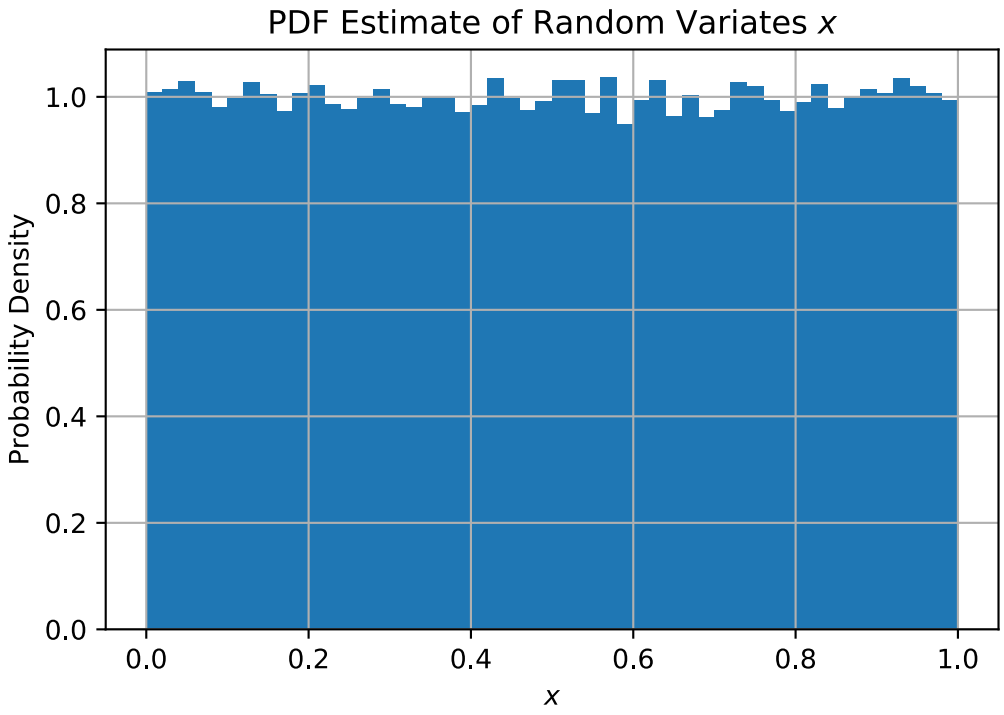
```
k_idx = 0 and u1_val = 0.6865
k_idx = 1 and u1_val = -0.2127
k_idx = 2 and u1_val = 0.0984
k_idx = 3 and u1_val = -1.0291
k_idx = 4 and u1_val = 0.8527
```

Functions of a Random Variable

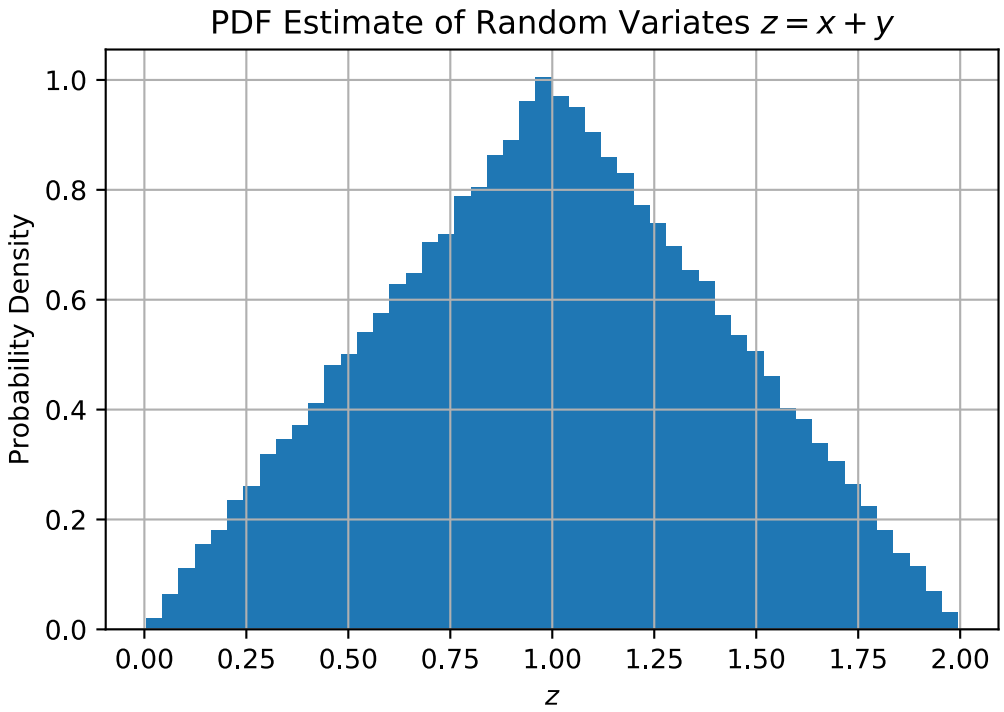
To get a feel for functions of random variable it is pretty easy to create RV realizations (trails), operate arrays (`ndarrays`) of trials, then histogram the results.

```
In [42]: x = rand(100000)
         y = rand(100000)
         z = x + y
```

```
In [45]: hist(x,50,density=True);
         grid()
         title(r'PDF Estimate of Random Variates $x$')
         ylabel(r'Probability Density')
         xlabel(r'$x$');
```




```
In [46]: hist(z,50,density=True);
         grid()
         title(r'PDF Estimate of Random Variates $z = x + y$')
         ylabel(r'Probability Density')
         xlabel(r'$z$')

Out[46]: Text(0.5,0,'$z$')
```



Correlated Random Variates

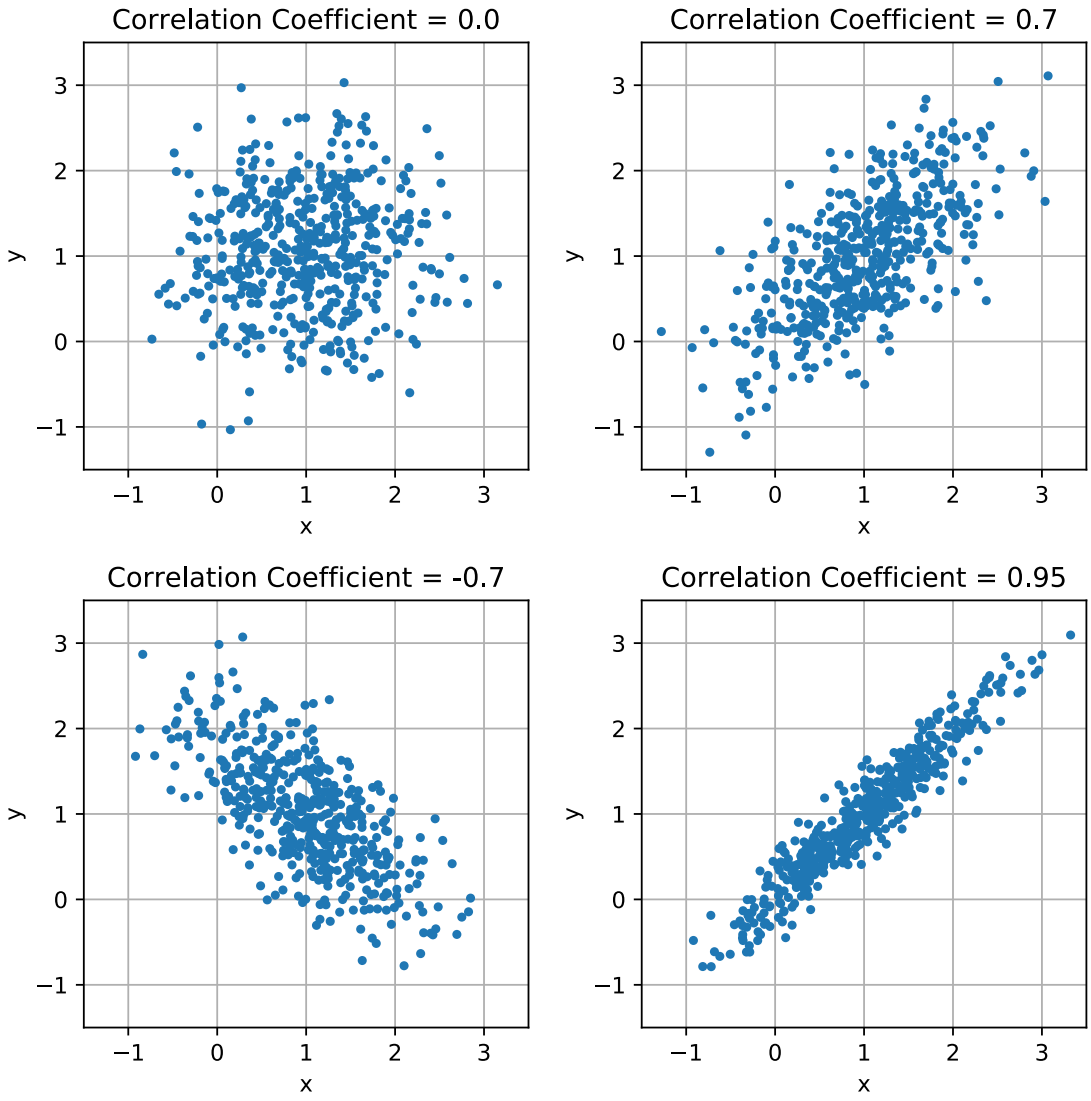
Generate 2D correlated Gaussian (normal) random variates using:

```
In [32]: def corrgauss(mean,var,p,N):
             """
             y = corrgauss(mean,var,p,N)
             Generate Correlated Gaussian RV's
             mean = mean vector (m1, m2)
             var = variance vector (var1, var2)
             p = correlation coefficient
             N = number of random pairs to generate
             """
             x = randn(2,N)
             y = zeros_like(x)
             D,P = eig([[var[0],p*sqrt(var[0]*var[1])],[p*sqrt(var[0]*var[1]),var[1]]])
             M = dot(P,diag(sqrt(D)))
             for k in range(N):
                 y[:,k] = dot(M,array([x[:,k]]).T).T
             y[0,:] += mean[0]
             y[1,:] += mean[1]
             return y
```

Example Consider four cases of 500 random pairs each. In all four cases the mean vector is $\mathbf{m}_x = [m_x\ m_y] = [1.0\ 1.0]$. The variance of each coordinate is given by $\sigma_x^2 = \sigma_y^2 = 0.5$ and the correlation coefficient varies over $\rho \in \{0.0, 0.7, -0.7, 0.95\}$. I first generate the samples as a 2×500 `ndarray`. Next check that the sample values have mean, convariance, and correlation coefficient as expected. The `mean()` function, which is part of Numpy, needs to know which axis is perform the mean along. Axis 0 is by row and axis 1 is by column. Both the `cov()` and `corrcoef()`, which are matrices (`ndarrays`) get this right automatically. Note also that the matrix returned by `corrcoef()` has ρ on the off diagonal.

```
In [33]: y1 = corrgauss([1.0,1.0],[0.5,0.5],0.0,500)
         (mean(y1,axis=1), cov(y1), corrcoef(y1))

Out[33]: (array([ 0.98941022,  1.06203749]), array([[ 0.49074792,  0.02743399],
                 [ 0.02743399,  0.48789009]]), array([[ 1.       ,  0.0560659],
                 [ 0.0560659,  1.       ]]))

In [34]: y2 = corrgauss([1.0,1.0],[0.5,0.5],0.7,500)
         (mean(y2,axis=1), cov(y2), corrcoef(y2))

Out[34]: (array([ 0.9896518 ,  0.98279781]), array([[ 0.50833203,  0.3527965 ],
                 [ 0.3527965 ,  0.54311693]]), array([[ 1.        ,  0.67143483],
                 [ 0.67143483,  1.        ]]))

In [35]: y3 = corrgauss([1.0,1.0],[0.5,0.5],-0.7,500)
         (mean(y3,axis=1), cov(y3), corrcoef(y3))

Out[35]: (array([ 0.99291309,  1.00952994]), array([[ 0.46459991, -0.31396529],
                 [-0.31396529,  0.45470455]]), array([[ 1.        , -0.68308922],
                 [-0.68308922,  1.        ]]))

In [36]: y4 = corrgauss([1.0,1.0],[0.5,0.5],0.95,500)
         (mean(y4,axis=1), cov(y4), corrcoef(y4))

Out[36]: (array([ 0.99120846,  0.98277702]), array([[ 0.56101091,  0.52690901],
                 [ 0.52690901,  0.54597374]]), array([[ 1.        ,  0.95205951],
                 [ 0.95205951,  1.        ]]))

In [37]: figure(figsize=(7,7))
         subplot(221)
         plot(y1[0,:],y1[1,:],'.')
         axis([-1.5,3.5,-1.5,3.5])
         grid()
         xlabel('x')
         ylabel('y')
         title('Correlation Coefficient = 0.0')
         subplot(222)
         plot(y2[0,:],y2[1,:],'.')
         axis([-1.5,3.5,-1.5,3.5])
         grid()
```

```
xlabel('x')
ylabel('y')
title('Correlation Coefficient = 0.7')
subplot(223)
plot(y3[0,:],y3[1,:],'.')
axis([-1.5,3.5,-1.5,3.5])
grid()
xlabel('x')
ylabel('y')
title('Correlation Coefficient = -0.7')
subplot(224)
plot(y4[0,:],y4[1,:],'.')
axis([-1.5,3.5,-1.5,3.5])
grid()
xlabel('x')
ylabel('y')
title('Correlation Coefficient = 0.95');
tight_layout()
```



In all cases above you see that the scatter points are centered about $(x, y) = (1, 1)$.

Correlated Gaussian Using `scipy.stats.multivariate_normal`

You can also generate correlated Gaussian random variates using the `rvs()` method of `scipy.stats.multivariate_normal`. This ,`multivariate_normal` class

```
In [26]: joint_norm = stats.multivariate_normal.rvs([1,1],[[1,.9],[0.9,1]],1000)

In [27]: plot(joint_norm[:,0],joint_norm[:,1],'.')
         axis([-1.5,3.5,-1.5,3.5])
         axis('equal')
         grid()
         xlabel('x')
```

```
ylabel('y')
title('Correlation Coefficient = 0.0')
```

```
Out[27]: Text(0.5,1,'Correlation Coefficient = 0.0')
```