SSE 691: Intro to Data Science Project 4

Jonathan Alfred
10-3-2016

Contents

Table of Figures

Introduction

Chapter 6

A brief explanation of each section will be given, along with code from the chapter, code changes and code outputs for any code within that section, with explanations given as necessary.

Dependence and Independence

This section talked about the difference of independence and dependence of variables within probabilities. When two items are independent, that means the probabilities of both events or variables occurring is the probability of each event multiplied together. For example, the probability of rolling a one on a fair die is 1/6. The probability of doing this twice in a row is the probability of rolling a one the first time multiplied by the probability of rolling a one the second time, coming to be 1/36. This is defined in the following equation:

$$P(E, F) = P(E)P(F)$$

There was no code in this section to change.

Conditional Probability

Conditional probability refers to times when events may not be separate and independent of each other. A way to define this is the probability that one event occurs, given that we know another event has occurred. This is defined in the following equation:

$$P(E|F) = P(E, F)/P(F)$$

To explain conditional probability, the book used the following code, creating families of two children and then checking the probability that both are girls given that either are girls or that the older is a girl. The code is seen below:

```
def random_kid():
    return random.choice(["boy", "girl"])

both_girls = 0
either_girl = 0
older_girl = 0

random.seed(0)
for _ in range(10000):
    younger = random_kid()
    older = random_kid()
    if older == "girl":
        older_girl += 1
    if older == "girl" and younger == "girl":
        both_girls += 1
    if older == "girl" or younger == "girl":
        either_girl += 1

print("P(both | older)", both_girls/older_girl)
print("P(both | either)", both_girls/either_girl)
```

```
P(both | older) 0.5007089325501317
P(both | either) 0.3311897106109325
```

SciPy.stats does not have a built in function for conditional probability. While it does have common variable distributions, such as the normal distribution and uniform distribution (which random.random() uses), it also does not have the functionality to make a random choice within a list. NumPy, another SciPy module, has this functionality. The code, thus, was not changed for this section.

Baye's Theorem

Baye's Theorem is an important methodology for finding the probability of an event occurring. Suppose we only have the probability of event one occurring conditional event two occurring. Baye's theorem allows us to find the opposite: the probability of event two occurring conditional event one occurring.

This section did not have any code to be changed.

Random Variables

Random variables are variables whose positive values follow a probability distribution. For Python, the random.random() function uses the uniform probability distribution to generate its random numbers.

SciPy has a number of built in distributions. The following sections will cover creating these distributions within Python and converting the code into SciPy.

Continuous Distributions

As mentioned before, Python uses the uniform distribution. This is an example of continuous distribution. Continuous distributions have an infinite number of points and are infinitely connected along their distributions, allowing for an infinite number of outcomes.

Continuous distributions make use of a probability density function. Because of the infinite number of points with continuous distributions, all weights on an individual number must be 0. To find the probability of seeing a value over a certain interval, the integral must be taken over that interval. Figure 1 shows the PDF of a uniform distribution, generated using SciPy.

The cumulative distribution function is more commonly used over the PDF. This gives the probability of x being less than a certain value over the PDF of a function. The CDF of a uniform distribution is seen in Figure 2, generated using SciPy.

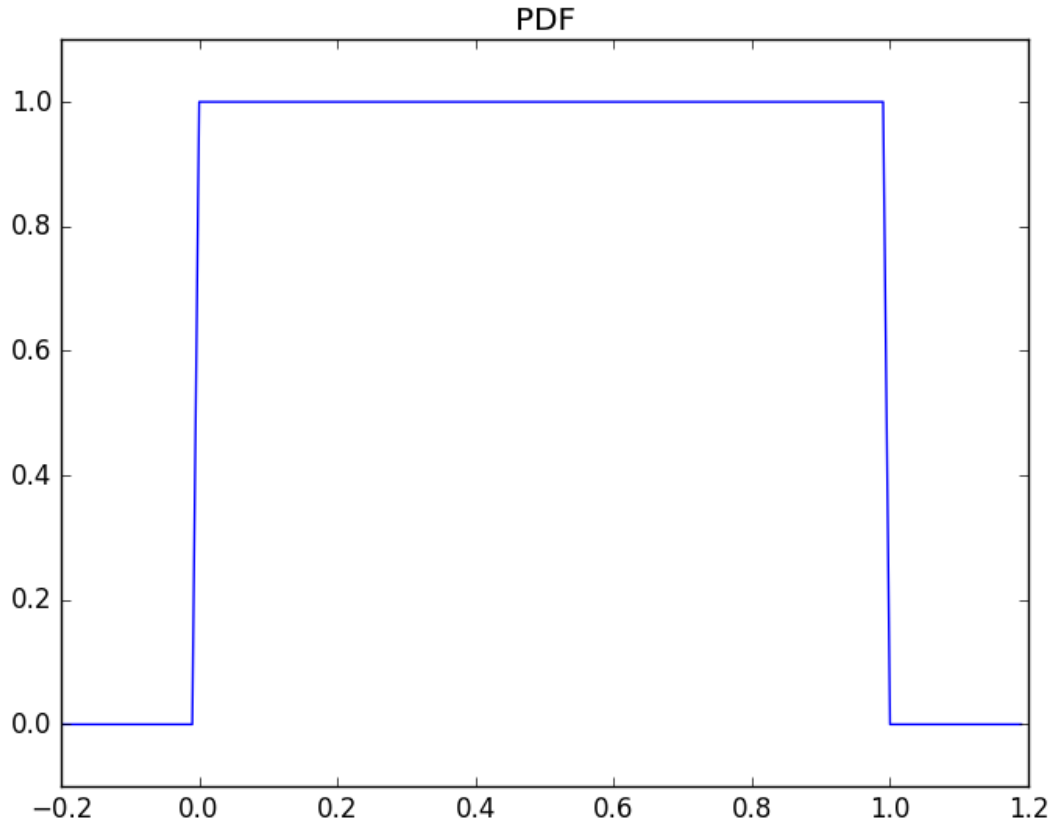

Figure 1 - PDF of a Uniform Distribution

```
fig= plt.figure()
rv = sp.uniform()
x = list(np.arange(-.2,1.2, .01))
#plt.axes([-.5, 1.5, 0, 1.5])
plt.plot(x, rv.pdf(x))
plt.title("PDF")
plt.ylim(ymax=1.1, ymin=-.1)
plt.show()
```

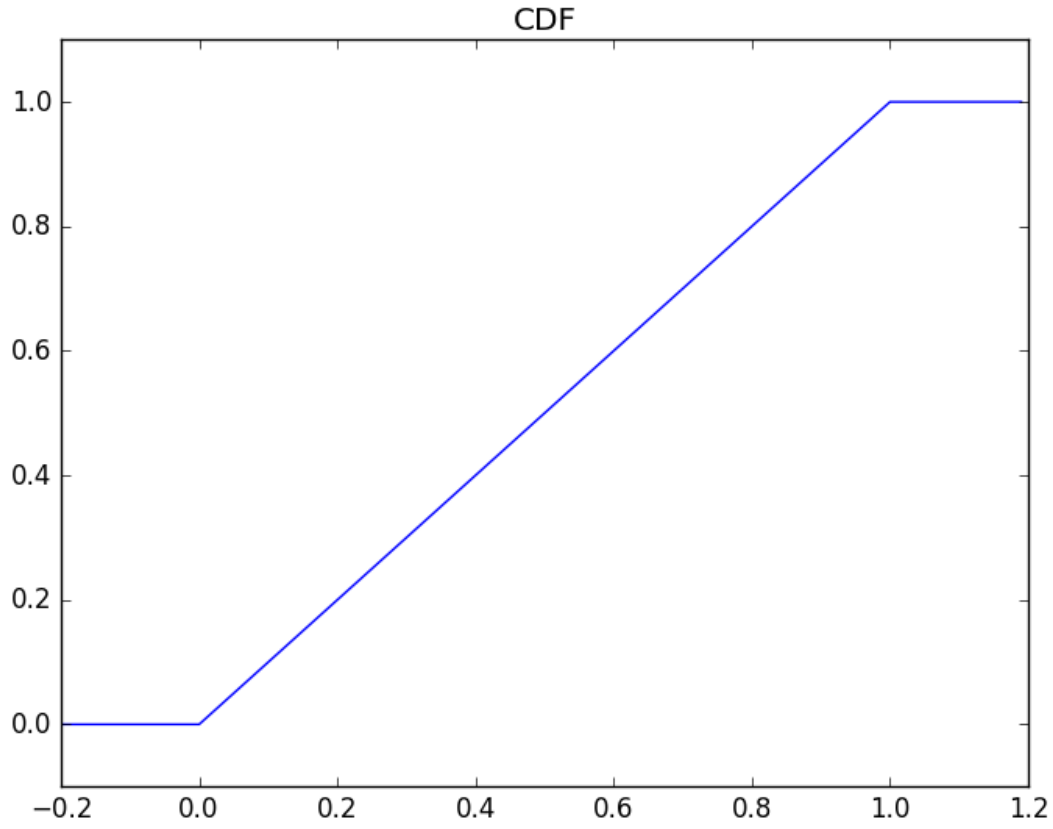

Figure 2 - CDF of a Uniform Distribution

```
plt.title("CDF")
plt.ylim(ymax=1.1, ymin=-.1)
plt.plot(x, rv.cdf(x))
plt.show()
```

As seen in the code above, SciPy has functionality for the uniform distribution built in, being able to check the PDF and CDF of the function very easily. The code for generating a uniform distribution is longer utilizing vanilla python and is seen below:

```
def uniform_pdf(x):
    return 1 if x >= 0 and x < 1 else 0

def uniform_cdf(x):
    if x < 0: return 0
    elif x < 1: return x
    else: return 1
```

The Normal Distribution

The normal distribution is one of the most commonly used distributions for generating random variables. This is the common bell-curve often used with data. It is determined very easily, using mu, the mean of the function, and sigma, the standard deviation or how wide the bell-curve is. Mu will shift the bell-curve along the x-axis, while sigma will flatten or sharpen the bell-curve. The function for a normal distribution is seen below:

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Creating this function in vanilla python requires the math library for both exponential and the square root function. The code for this is seen below, along with the code for generating Figure 3.

```
def normal_pdf(x, mu=0, sigma=1):
    sqrt_two_pi = math.sqrt(2 * math.pi)
    return(math.exp(-(x-mu) ** 2/2/sigma**2)/(sqrt_two_pi * sigma))

xs = [x /10.0 for x in range(-50, 50)]
plt.plot(xs, [normal_pdf(x, sigma=1) for x in xs], '-', label="mu=0,sigma=1")
plt.plot(xs, [normal_pdf(x, sigma=2) for x in xs], '-', label="mu=0,sigma=2")
plt.plot(xs, [normal_pdf(x, sigma=0.5) for x in xs], '-', label="mu=0,sigma=0.5")
plt.plot(xs, [normal_pdf(x, mu=-1) for x in xs], '-', label="mu=-1,sigma=1")
plt.legend()
plt.title("Various Normal PDFs without scipy")
plt.show()
```

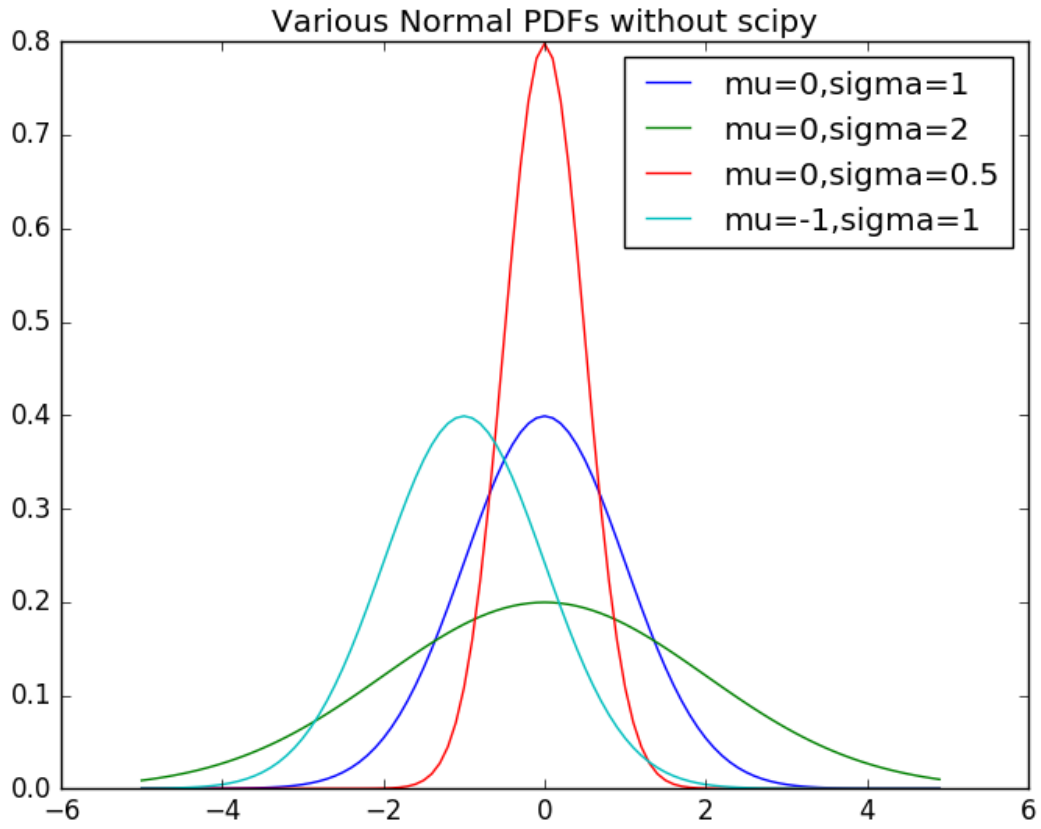

Figure 3 - PDFs of various normal distributions

Figure 4 was generated using SciPy and matplotlib. SciPy has a built in normal distribution, allowing for ease of use. The parameters, however, are not called mu and sigma, but loc and scale. This was confusing to deal with first, but after noting what mu and sigma do, this was no longer a difficult problem. Code for this is seen below:

```
plt.plot(xs, sp.norm().pdf(xs),'-', label="mu=0,sigma=1")
plt.plot(xs, sp.norm(0,2).pdf(xs),'-', label="mu=0,sigma=2")
plt.plot(xs, sp.norm(0, .5).pdf(xs),'-', label="mu=0,sigma=.5")
plt.plot(xs, sp.norm(-1, 1).pdf(xs),'-', label="mu=-1,sigma=1")
plt.title("Norm PDF with SciPy")
plt.legend()
plt.show()
```

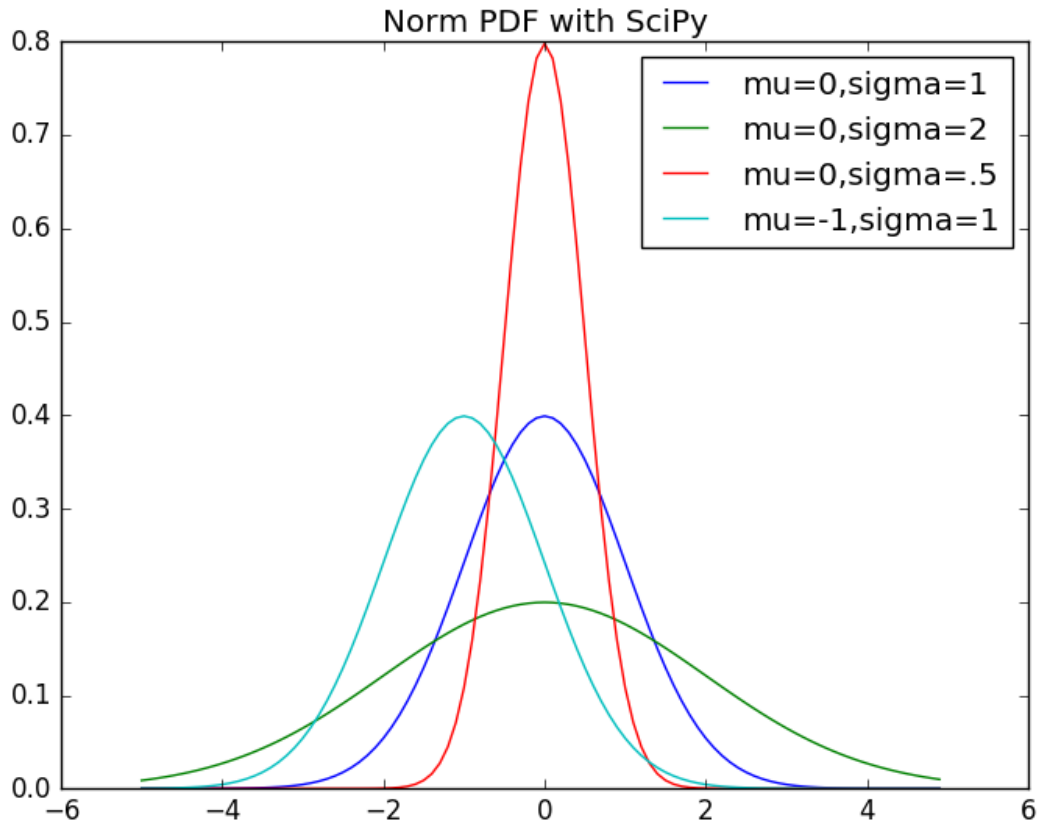

Figure 4 - SciPy Version of Normal Distributions

As noted before, the CDF is also important. Doing this was easier than creating the normal distribution within vanilla python. The code for this and Figure 5's generation is seen below.

```
def normal_cdf(x, mu=0, sigma=1):
    return(1 + math.erf((x-mu)/math.sqrt(2)/sigma))/2

xs = [x / 10.0 for x in range(-50, 50)]
plt.plot(xs, [normal_cdf(x, sigma=1) for x in xs],'-', label='mu=0, sigma=1')
plt.plot(xs, [normal_cdf(x, sigma=2) for x in xs],'-', label='mu=0, sigma=2')
plt.plot(xs, [normal_cdf(x, sigma=0.5) for x in xs],'-', label='mu=0, sigma=0.5')
plt.plot(xs, [normal_cdf(x, mu=-1) for x in xs],'-', label='mu=-1, sigma=1')
plt.legend(loc=4)
plt.title("Various Normal cdfs without SciPy")
plt.show()
```

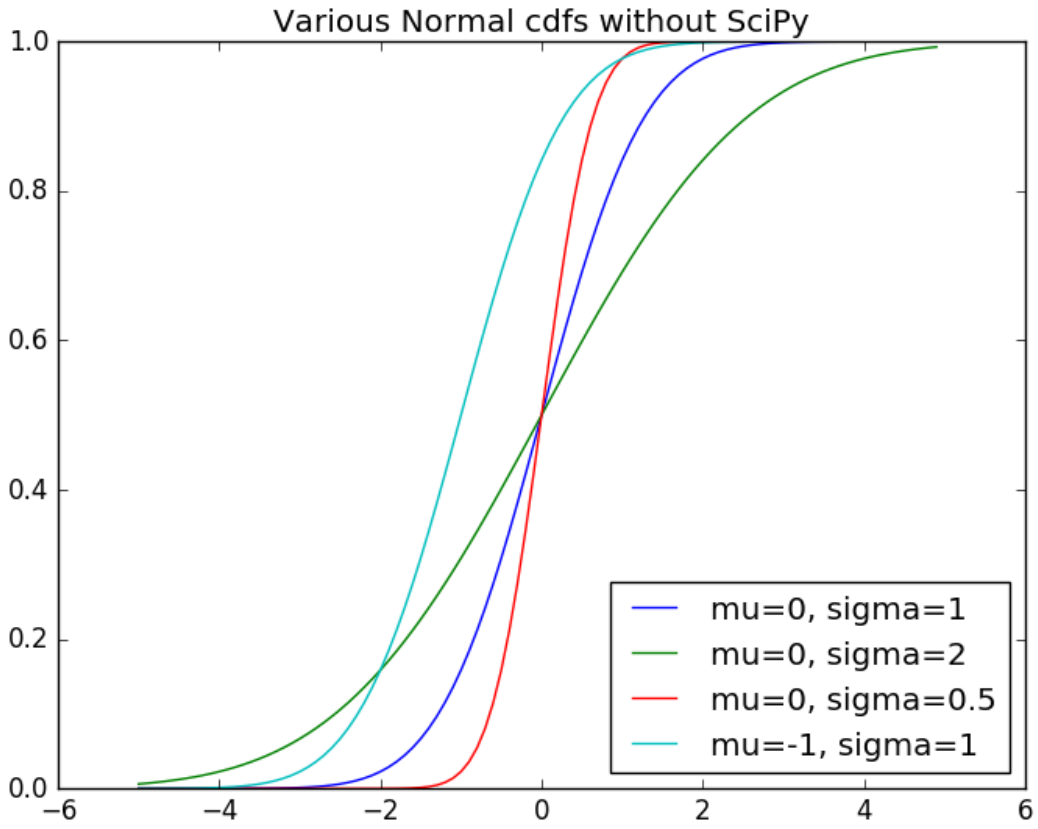

Figure 5 - Various Normal Distribution CDFs

Again, built into the normal function of SciPy is the CDF function. The code for generating Figure 6 is seen below:

```
plt.plot(xs, sp.norm().cdf(xs),'-', label="mu=0,sigma=1")
plt.plot(xs, sp.norm(0,2).cdf(xs),'-', label="mu=0,sigma=2")
plt.plot(xs, sp.norm(0, .5).cdf(xs),'-', label="mu=0,sigma=.5")
plt.plot(xs, sp.norm(-1, 1).cdf(xs),'-', label="mu=-1,sigma=1")
plt.title("Norm CDF with SciPy")
plt.legend()
plt.show()
```

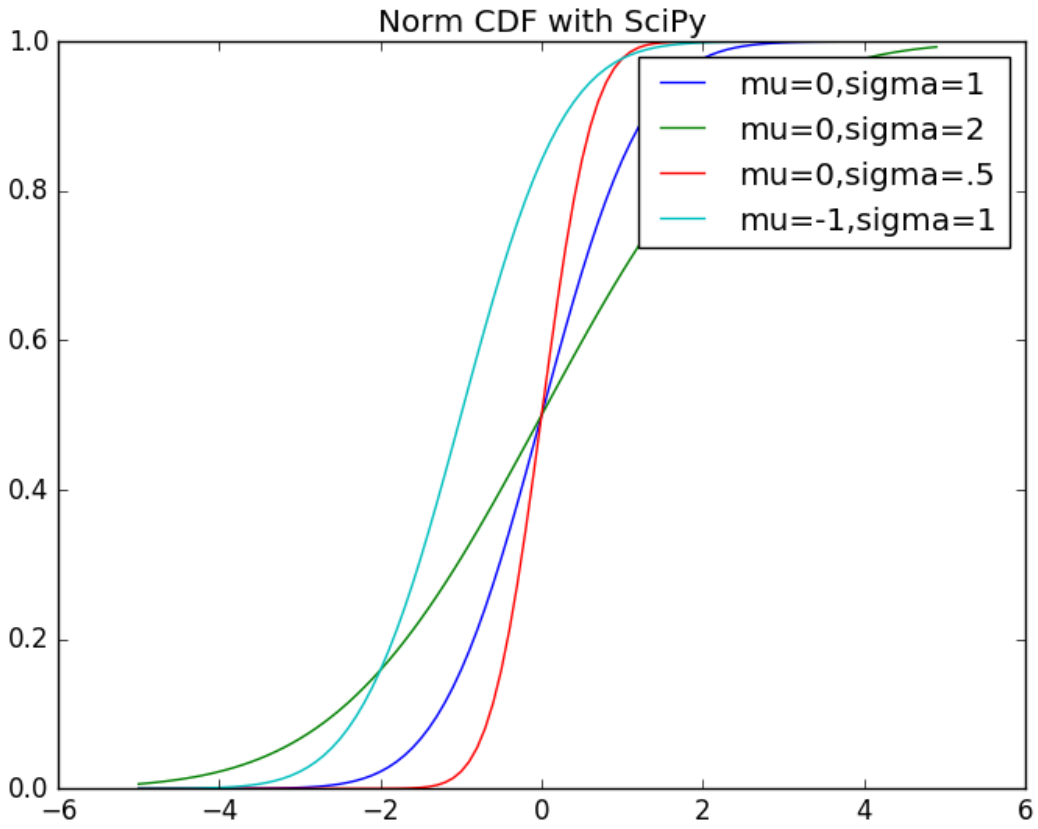

Figure 6 - Normal Distribution of CDFs using SciPy

Another important note is using a probability to find the specific normal distribution value tied to that probability. This would make use of the inverse CDF of a function. Since this is a continuous distribution, finding an exact value may not be possible. However, finding a value that lies within a very small tolerance will be. A tolerance of .00001 was used as the default, and a binary search was implemented to quickly find a value for a specific probability. The code for this is seen below:

```
def inverse_normal_cdf(p, mu=0, sigma=1, tolerance=0.00001):
    if mu != 0 or sigma != 1:
        return mu + sigma*inverse_normal_cdf(p, tolerance=tolerance)

    low_z, low_p = -10.0, 0
    hi_z, hi_p = 10.0, 1
    while hi_z - low_z > tolerance:
        mid_z = (low_z + hi_z) / 2
        mid_p = normal_cdf(mid_z)
```

```
        if mid_p < p:
            low_z, low_p = mid_z, mid_p
        elif mid_p > p:
            hi_z, hi_p = mid_z, mid_p
        else:
            break

    return mid_z

print(inverse_normal_cdf(.7))
# 0.5243968963623047
```

The normal distribution within SciPy has this built in in the ppf() function. It arguably has a more accurate answer as well, after checking with other source as well. The code and output is seen below.

```
print(sp.norm().ppf(.7))
# 0.524400512708
```

Central Limit Theorum

The central limit theorem states that average of a large number of independent and identically distributed randomly generated variables will be approximately normally generated. To show this, the book displayed code for creating a histogram using two discrete distributions, the Bernoulli distribution and binomial distribution. The code for this and Figure 7 is seen below:

```
def bernoulli_trial(p):
    return 1 if random.random() < p  else 0

def binomial(n, p):
    return sum(bernoulli_trial(p) for _ in range(n))

def make_hist(p, n, num_points):

    data = [binomial(n, p) for _ in range(num_points)]

    histogram = Counter(data)
    plt.bar([x - 0.4 for x in histogram.keys()],
            [v / num_points for v in histogram.values()],
            0.8,
            color='0.75')

    mu = p * n
    sigma = math.sqrt(n * p * (1 - p))

    xs = range(min(data), max(data) + 1)
    ys = [normal_cdf(i + 0.5, mu, sigma) - normal_cdf(i - 0.5, mu, sigma) for i in
xs]
```

```
    plt.plot(xs,ys)
    plt.title("Binom hist without SciPy")
    plt.show()

make_hist(0.75, 100, 10000)
```

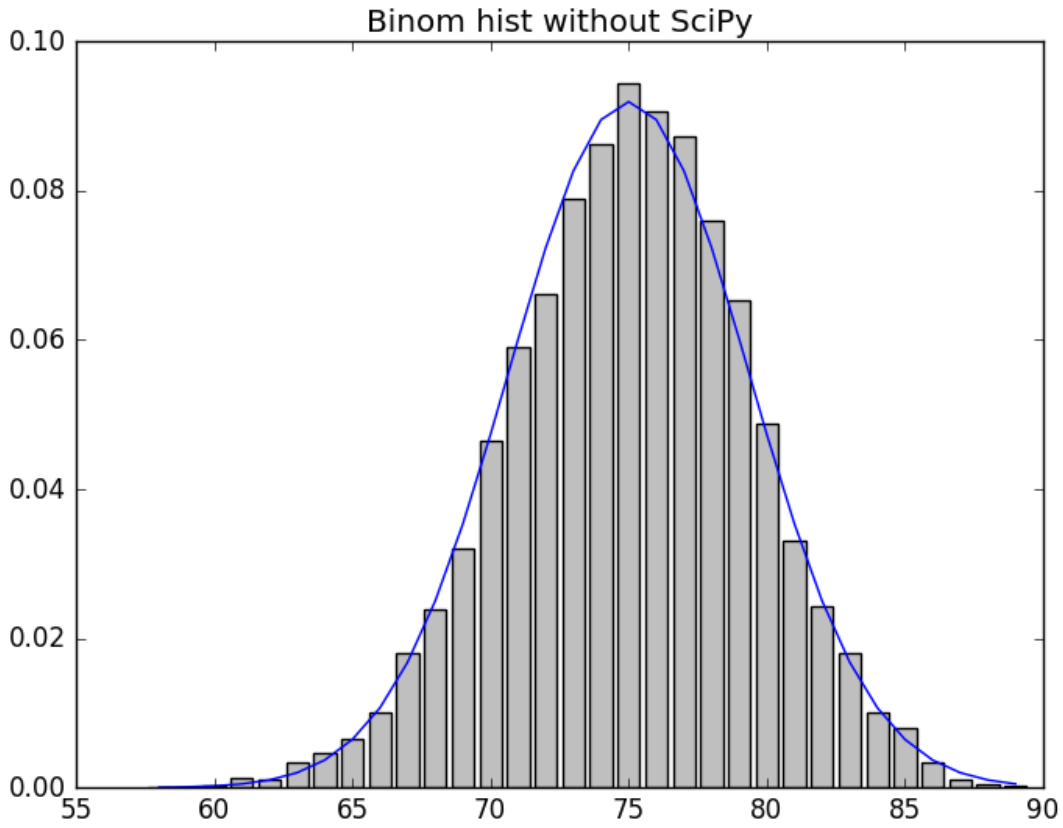

Figure 7 - Binomial Distribution with the normal distribution overlaid over it w/o SciPy

As seen in the figure, the large number of variables gravitated towards the probability .75, used for the binomial distribution. Outside of this, the values slowly began to fade off in a normal distribution. The average was .75. This code can be recreated using SciPy as well, seen below:

```
def make_hist_scipy(p, n, num_points):

    data = sp.binom(n, p).rvs(num_points)

    histogram = Counter(data)
    plt.bar([x - 0.4 for x in histogram.keys()],
            [v / num_points for v in histogram.values()],
            0.8,
            color='0.75')
```

```
    mu = p * n
    sigma = math.sqrt(n * p * (1 - p))

    xs = range(min(data), max(data) + 1)
    ys = [sp.norm(mu,sigma).cdf(i+.5) - sp.norm(mu,sigma).cdf(i -.5) for i in xs]
    plt.plot(xs,ys)
    plt.title("Binom Hist with SciPy")
    plt.show()

make_hist_scipy(.75, 100, 10000)
```

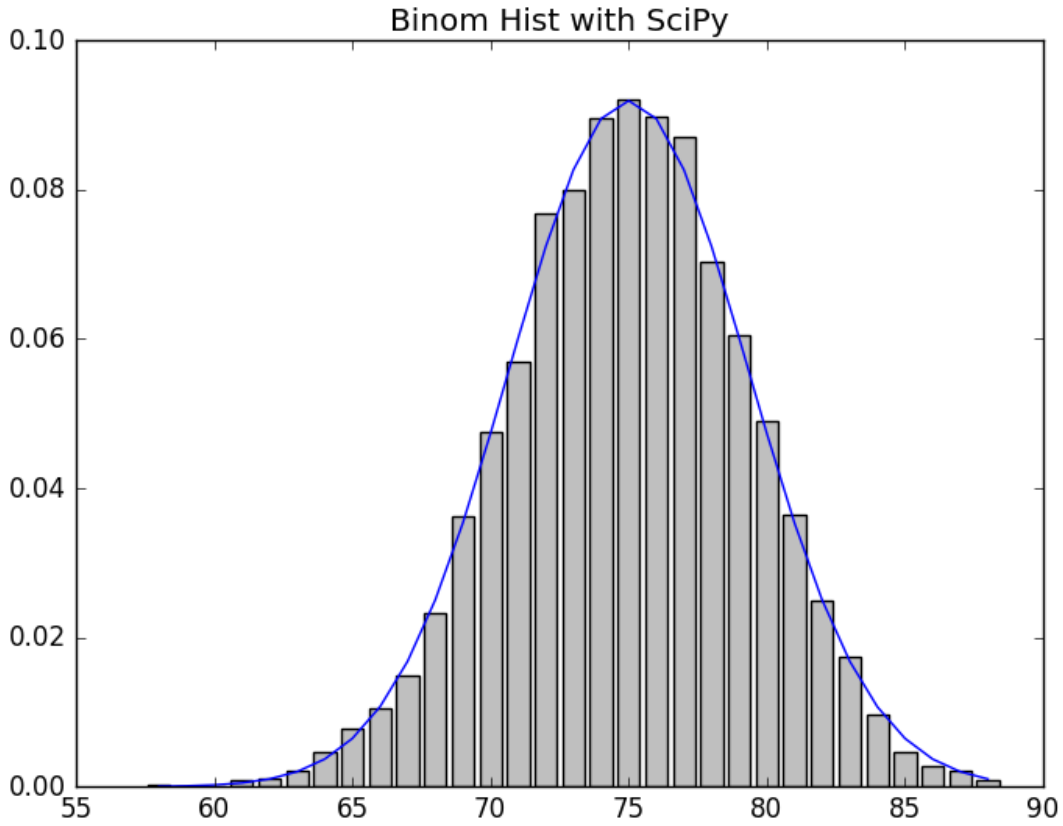

Figure 8 - Binomial Distribution with Normal Distribution over it w/ SciPy

SciPy Stats Extended Work

SciPy has a huge number of distributions that can be used. This was rather overwhelming, so I kept on using the normal distribution, looking into the functions that it had. I used a standard normal distribution with a mu of 0 and a sigma of 1.

Upon looking more into the documentation of the function, I noted the rvs function that most of the distributions had. This is the random variable generator that each distribution has. I tested it out a few times with the normal distribution. I had also used it with the Bernoulli and Binomial distributions, just to ensure that the distributions worked as I expected.

SciPy.Stats also has some statistical functionality as well. The stats function can gives the mean, variance, skew, and kurtosis of the distribution. These can all be done indivitually as well. There are things that Pandas did in the prior chapter that SciPy.Stats can do as well. This may come in handy later on within the chapter.

One thing that I did not get to test was the fit function. This could be useful for fitting data so some sort of curve. It would be much easier to display whether or not a set of data is a random distribution or not.

Discussion

SciPy.Stats seems to be more for statistical analysis using the distributions. NumPy seemed to have most of the functionality that was used in this chapter. However, SciPy.stats seems to have more distributions, and rather than just sampling to create an array, can be used different than NumPy. SciPy stats might have more ability than I could find.

I was particularly interested in how the fit() function worked within the various distributions of SciPy stats. However, as I did not have a dataset, I could not test this out on anything. Overall, this should be helpful for later ventures into data science.

Indirect Logs

Data	Time	Activity
10/1/16	120	Read through chapter 6
10/2/2016	240	Begin coding chapter 6.
10/3/2016	300	Finished coding chapter 6. Began and finished document.
Total	660	

Code

```
import random
import scipy.stats as sp
import matplotlib.pyplot as plt
import numpy as np
import math
from collections import Counter

def random_kid():
    return random.choice(["boy", "girl"])

both_girls = 0
either_girl = 0
older_girl = 0

random.seed(0)
for _ in range(10000):
    younger = random_kid()
    older = random_kid()
    if older == "girl":
        older_girl += 1
    if older == "girl" and younger == "girl":
        both_girls += 1
    if older == "girl" or younger == "girl":
        either_girl += 1
```

```
print("P(both | older)", both_girls/older_girl)
print("P(both | either)", both_girls/either_girl)

def uniform_pdf(x):
    return 1 if x >= 0 and x < 1 else 0

def uniform_cdf(x):
    if x < 0: return 0
    elif x < 1: return x
    else: return 1

fig= plt.figure()
rv = sp.uniform()
x = list(np.arange(-.2,1.2, .01))
#plt.axes([-.5, 1.5, 0, 1.5])
plt.plot(x, rv.pdf(x))
plt.title("PDF")
plt.ylim(ymax=1.1, ymin=-.1)
plt.show()
plt.title("CDF")
plt.ylim(ymax=1.1, ymin=-.1)
plt.plot(x, rv.cdf(x))
plt.show()

def normal_pdf(x, mu=0, sigma=1):
    sqrt_two_pi = math.sqrt(2 * math.pi)
    return(math.exp(-(x-mu) ** 2/2/sigma**2)/(sqrt_two_pi * sigma))

xs = [x /10.0 for x in range(-50, 50)]
plt.plot(xs, [normal_pdf(x, sigma=1) for x in xs], '-', label="mu=0,sigma=1")
plt.plot(xs, [normal_pdf(x, sigma=2) for x in xs], '-', label="mu=0,sigma=2")
plt.plot(xs, [normal_pdf(x, sigma=0.5) for x in xs], '-', label="mu=0,sigma=0.5")
plt.plot(xs, [normal_pdf(x, mu=-1) for x in xs], '-', label="mu=-1,sigma=1")
plt.legend()
plt.title("Various Normal PDFs without scipy")
plt.show()


plt.plot(xs, sp.norm().pdf(xs),'-', label="mu=0,sigma=1")
plt.plot(xs, sp.norm(0,2).pdf(xs),'-', label="mu=0,sigma=2")
plt.plot(xs, sp.norm(0, .5).pdf(xs),'-', label="mu=0,sigma=.5")
plt.plot(xs, sp.norm(-1, 1).pdf(xs),'-', label="mu=-1,sigma=1")
plt.title("Norm PDF with SciPy")
plt.legend()
plt.show()

def normal_cdf(x, mu=0, sigma=1):
    return(1 + math.erf((x-mu)/math.sqrt(2)/sigma))/2

xs = [x / 10.0 for x in range(-50, 50)]
plt.plot(xs, [normal_cdf(x, sigma=1) for x in xs],'-', label='mu=0, sigma=1')
plt.plot(xs, [normal_cdf(x, sigma=2) for x in xs],'-', label='mu=0, sigma=2')
plt.plot(xs, [normal_cdf(x, sigma=0.5) for x in xs],'-', label='mu=0, sigma=0.5')
plt.plot(xs, [normal_cdf(x, mu=-1) for x in xs],'-', label='mu=-1, sigma=1')
plt.legend(loc=4)
```

```
plt.title("Various Normal cdfs without SciPy")
plt.show()

plt.plot(xs, sp.norm().cdf(xs),'-', label="mu=0,sigma=1")
plt.plot(xs, sp.norm(0,2).cdf(xs),'-', label="mu=0,sigma=2")
plt.plot(xs, sp.norm(0, .5).cdf(xs),'-', label="mu=0,sigma=.5")
plt.plot(xs, sp.norm(-1, 1).cdf(xs),'-', label="mu=-1,sigma=1")
plt.title("Norm CDF with SciPy")
plt.legend()
plt.show()

def inverse_normal_cdf(p, mu=0, sigma=1, tolerance=0.00001):
    if mu != 0 or sigma != 1:
        return mu + sigma*inverse_normal_cdf(p, tolerance=tolerance)

    low_z, low_p = -10.0, 0
    hi_z, hi_p = 10.0, 1
    while hi_z - low_z > tolerance:
        mid_z = (low_z + hi_z) / 2
        mid_p = normal_cdf(mid_z)
        if mid_p < p:
            low_z, low_p = mid_z, mid_p
        elif mid_p > p:
            hi_z, hi_p = mid_z, mid_p
        else:
            break

    return mid_z

print(inverse_normal_cdf(.7))

print(sp.norm().ppf(.7))

def bernoulli_trial(p):
    return 1 if random.random() < p  else 0

def binomial(n, p):
    return sum(bernoulli_trial(p) for _ in range(n))

print(bernoulli_trial(.4))
print(binomial(20, .4))

print(sp.bernoulli(.4).rvs())
print(sp.binom(20, .4).rvs())

def make_hist(p, n, num_points):

    data = [binomial(n, p) for _ in range(num_points)]

    histogram = Counter(data)
    plt.bar([x - 0.4 for x in histogram.keys()],
            [v / num_points for v in histogram.values()],
            0.8,
            color='0.75')
```

```
    mu = p * n
    sigma = math.sqrt(n * p * (1 - p))

    xs = range(min(data), max(data) + 1)
    ys = [normal_cdf(i + 0.5, mu, sigma) - normal_cdf(i - 0.5, mu, sigma) for i in
xs]
    plt.plot(xs,ys)
    plt.title("Binom hist without SciPy")
    plt.show()

make_hist(0.75, 100, 10000)

def make_hist_scipy(p, n, num_points):

    data = sp.binom(n, p).rvs(num_points)

    histogram = Counter(data)
    plt.bar([x - 0.4 for x in histogram.keys()],
            [v / num_points for v in histogram.values()],
            0.8,
            color='0.75')

    mu = p * n
    sigma = math.sqrt(n * p * (1 - p))

    xs = range(min(data), max(data) + 1)
    ys = [sp.norm(mu,sigma).cdf(i+.5) - sp.norm(mu,sigma).cdf(i -.5) for i in xs]
    plt.plot(xs,ys)
    plt.title("Binom Hist with SciPy")
    plt.show()

make_hist_scipy(.75, 100, 10000)
```