Notebook

March 15, 2019

Local date & time is : 03/15/2019 20:18:08 PDT

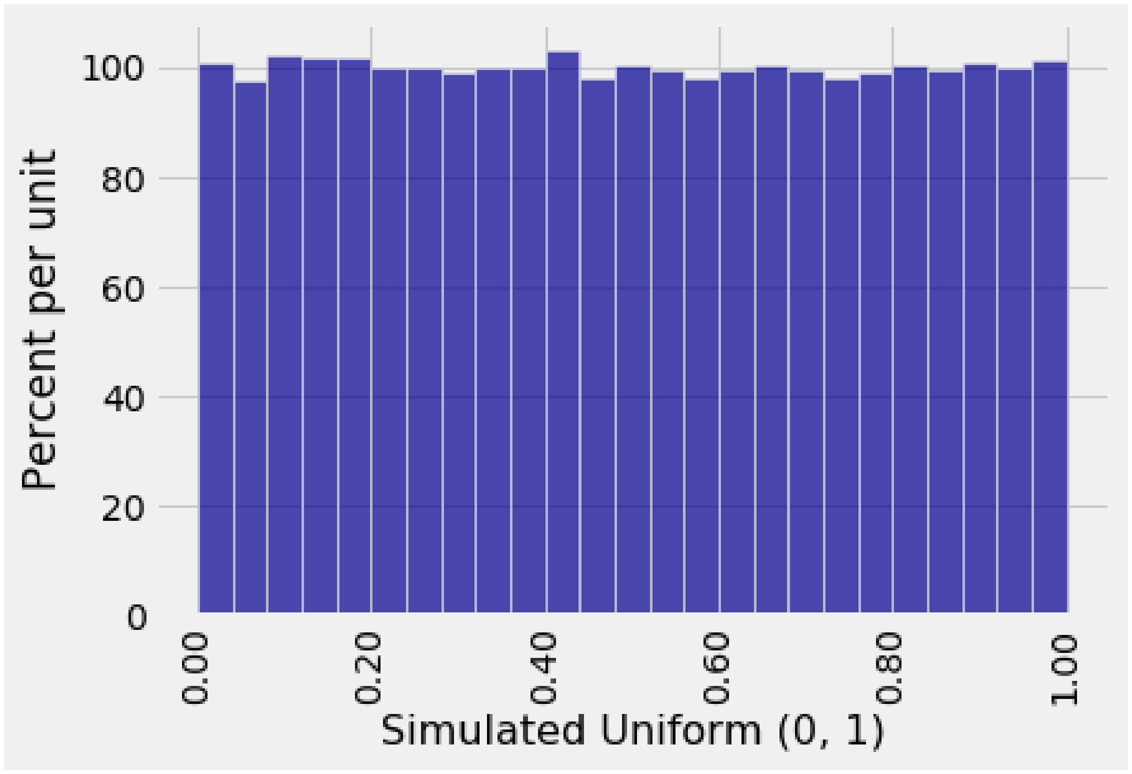
1 newpage

1.1 Part 1: Simulation in SciPy

The `stats` module of `SciPy` is familiar to you by now. For any of the well known distributions, you can use `stats` to simulate values of a random variable with that distribution. The general call is `stats.distribution_name.rvs(size = n)` where `rvs` stands for "random variates" and `n` is the number of independent replications you want.

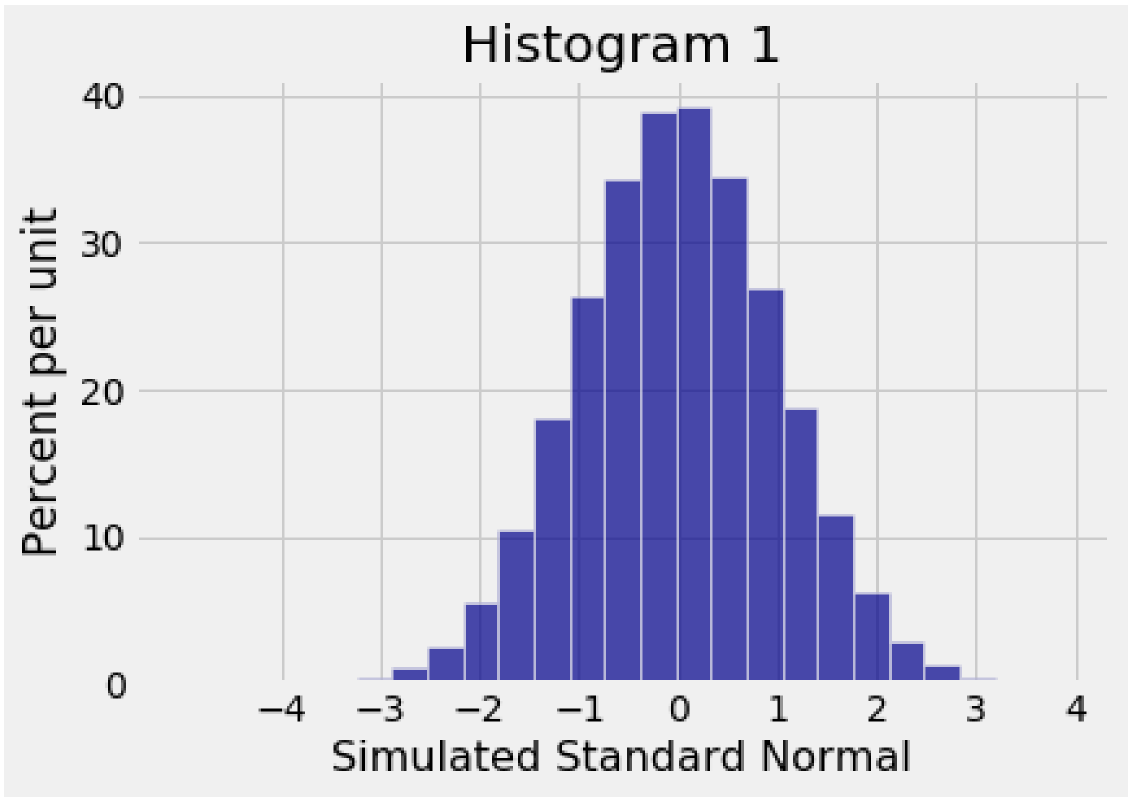
Every statistical system has conventions for how to specify the parameters of a distribution. In this lab we will tell you the specifications for a few distributions. Later you will be able to see a general pattern in the specifications.

```
In [4]: sim_uniform = stats.uniform.rvs(0,1, size = 100000)
        sim_uniform_tbl = Table().with_column(
            'Simulated Uniform (0, 1)', sim_uniform
        )
        sim_uniform_tbl.hist(bins=25)
```

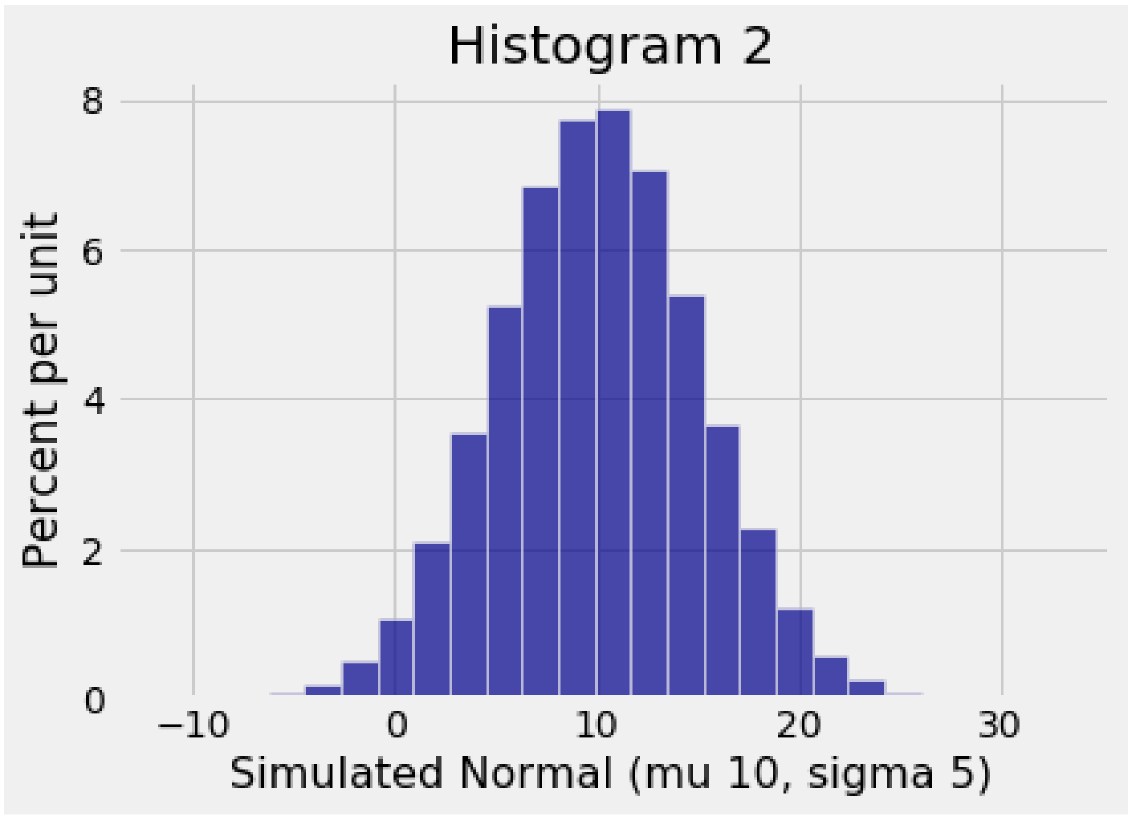


(i) 0.04
(ii) 100% per unit
(iii) area, 4%
(iv) 4%
(v) 5%

```
In [5]: stansimul = stats.norm.rvs(0 , 1, size = 100000)              # array of simulated values
        sim_std_norm_tbl = Table().with_column('Simulated Standard Normal', stansimul)
        sim_std_norm_tbl.hist(bins = 25)
        plt.xticks(np.arange(-4, 4.1))
        plt.title('Histogram 1');
```



```
In [6]: normsimul = stats.norm.rvs(10 , 5, size = 100000)
        sim_norm_tbl = Table().with_column('Simulated Normal (mu 10, sigma 5)', normsimul)
        sim_norm_tbl.hist(bins = 25)
        plt.title('Histogram 2');
```



The value -1 on the horizontal axis of Histogram 1 is the same as the value 5 on the horizontal axis of Histogram 2 expressed in standards units.

```
In [8]: # average value is 2 and rate of the simulated values is 0.498 which is consistent with the giv
        np.mean(sim_expon), 1/np.mean(sim_expon)

Out[8]: (1.9899269245553748, 0.5025310164208356)
```

2 newpage

2.1 Part 2. The Idea

How are all these random numbers generated? In the rest of the lab we will develop the method that underlies all the simulations above, by considering examples of increasing complexity.

Our starting point is a distribution on just four values.

Suppose$X$has the distribution `dist_X` below.

i. 0.3

ii. 0.1

iii. 0.2

iv. 0.4

for U in between 0(inclusive) and 0.3(inclusive), value is assigned to -2
for U in between 0.3(exclusive) and 0.4(inclusive), value is assigned to 1
for U in between 0.4(exclusive) and 0.6(inclusive), value is assigned to 4
for U in between 0.6(exclusive) and 1(inclusive), value is assigned to 7

3 newpage

3.1 Part 3. Visualizing the Idea

The method `plot_discrete_cdf` takes a distribution as its argument and plots a graph of the cdf.

Run the cell below to get a graph of the cdf of the random variable*X*in Part 1.

Fx(2.57) means sum of all the probability of getting X less than 2.57.

The graph shows the correct value of Fx(2.57) which is the prob of getting 1 plus prob of getting -2, which is 0.4

Whenever there is the next greater value of X, there is a jump. The size of the jump equals to the probability of that value X

For u in between 0 and 0.3, the returned value is -2

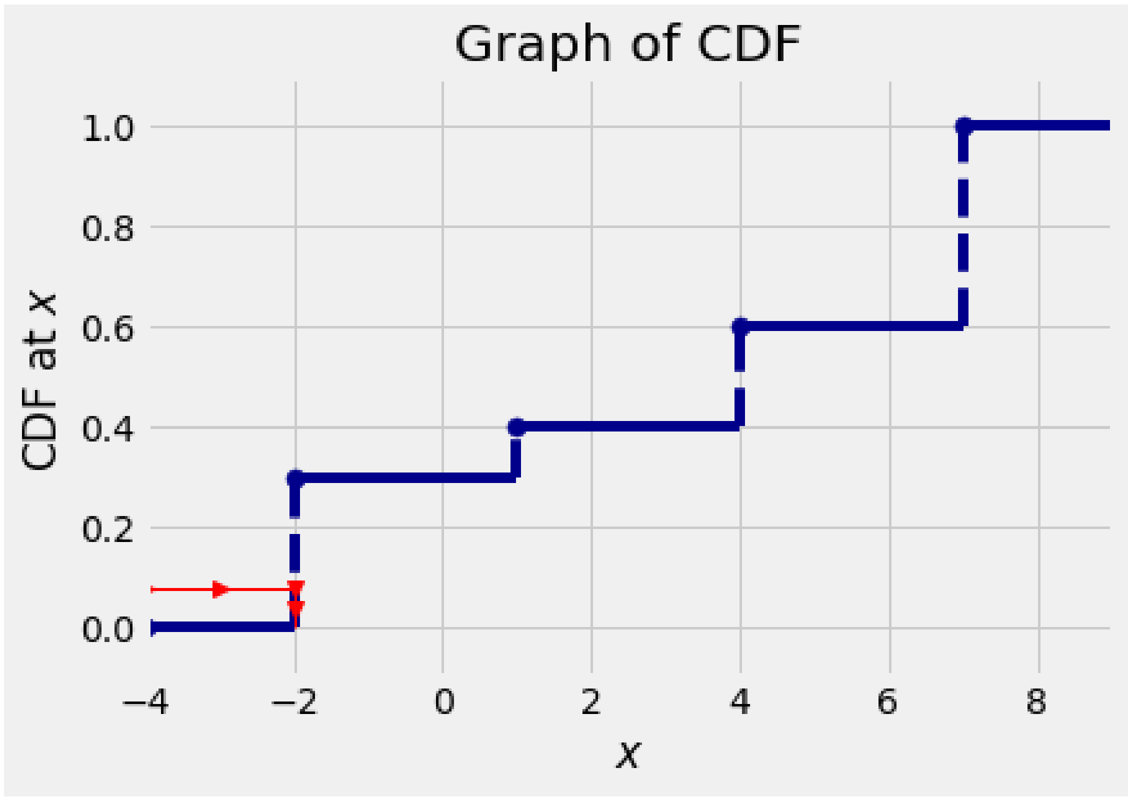
For u in between 0.3 and 0.4, the returned value is 1

for u in between 0.4 and 0.6, the returned value is 4

for u in between 0.6 and 1, the returned value is 7.

This is precisely how we set up the values corresponding to [0,1] interval in part 1

```
In [13]: plot_discrete_cdf(dist_X, stats.uniform.rvs(0,1, size = 1))
```



Just like part c, for the corresponding intervals for each value of X with respect to its probability, if our randomly generated falls in the interval designated for the specific value of X, that value is returned.

Intervals as follow, which is same is part c

For u in between 0 and 0.3, the returned value is -2

For u in between 0.3 and 0.4, the returned value is 1

for u in between 0.4 and 0.6, the returned value is 4

for u in between 0.6 and 1, the returned value is 7.

4 newpage

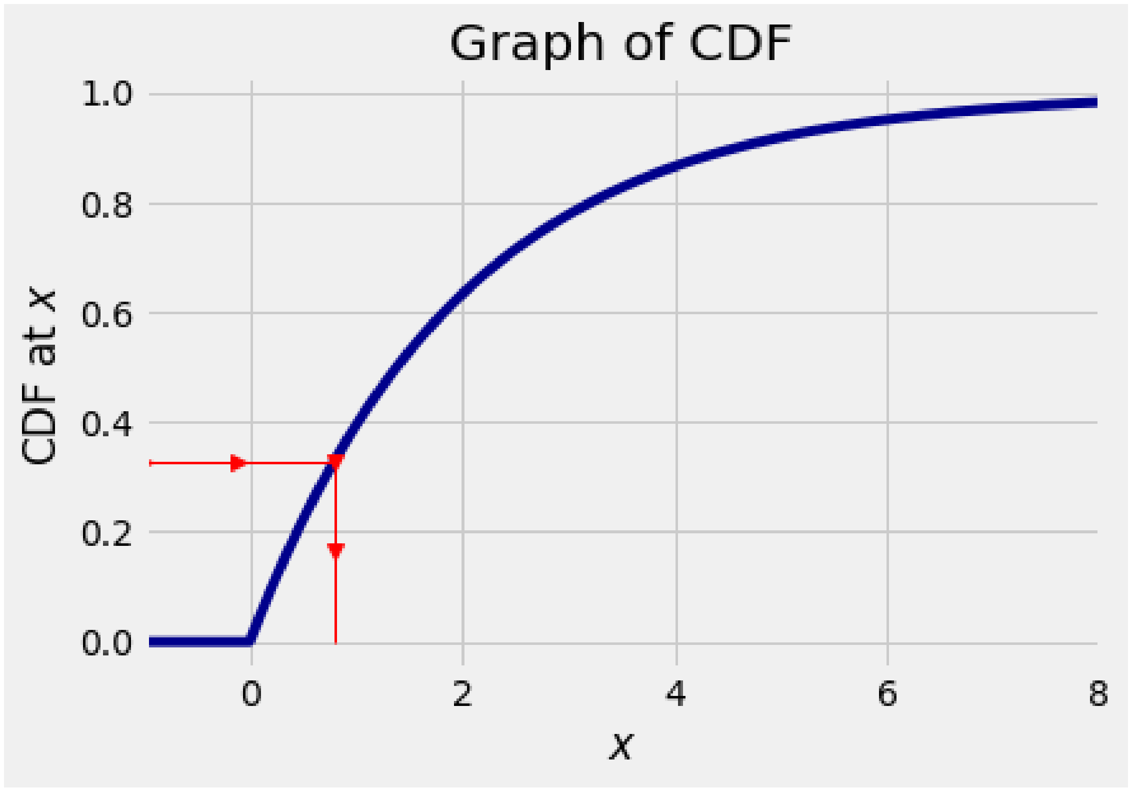
4.1 Part 4. Extension to Continuous Distributions

Now suppose you want to generate a random variable that has a specified continuous distribution. Let's start with the exponential(λ)distribution.

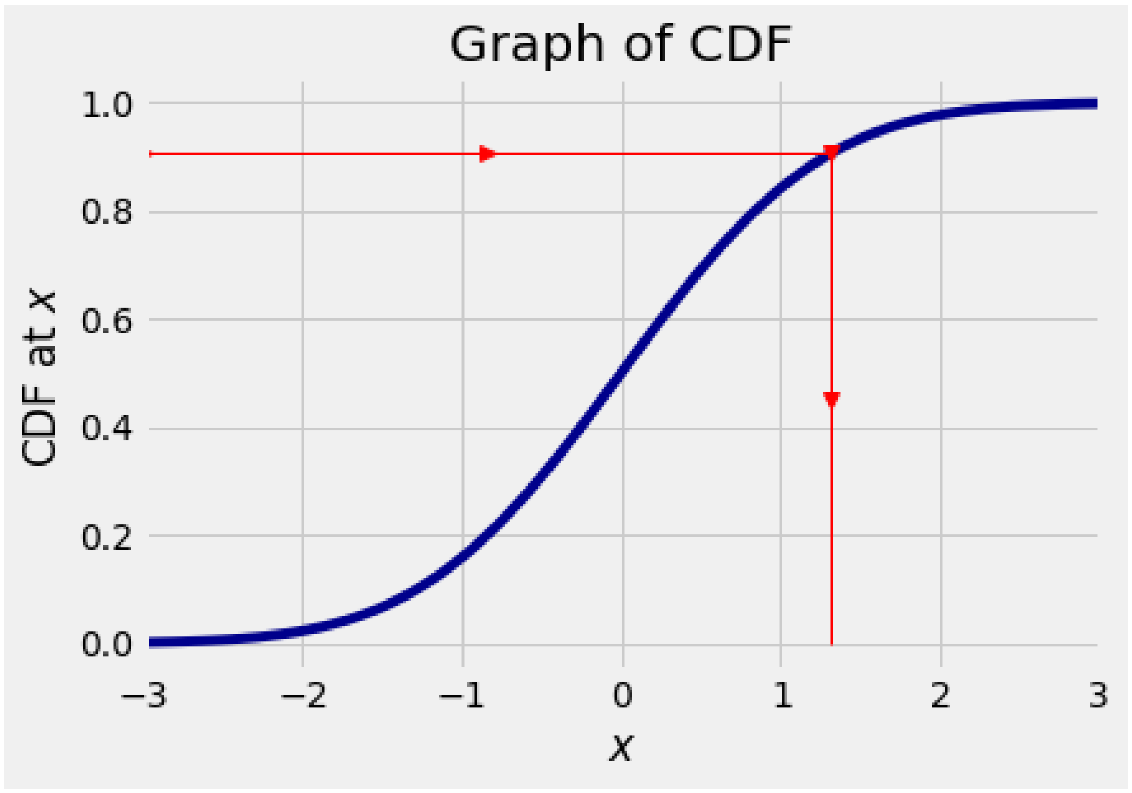
```
In [14]: # don't use "lambda" as that means something else in Python
         lamb = 0.5

         def expon_mean2_cdf(x):
             if x < 0:
                 return 0
             else:
                 return 1- np.exp(-lamb*x)
```

For the given random number between (0, 1), look at the corresponding value X at the graph of CDF.

```
In [17]: plot_continuous_cdf((-1, 8), expon_mean2_cdf,stats.uniform.rvs(0,1, size = 1))
```



```
In [18]: plot_continuous_cdf((-3, 3), stats.norm.cdf, stats.uniform.rvs(0,1, size = 1))
```



Graph of CDF
CDF at x
x
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3

5 newpage

5.1 Part 5. Empirical Verification that the Method Works

5.1.1 a) The Initial Values

Create a table that is called `sim` for simulation and consists of one column called `Uniform` that contains the values of 100,000 i.i.d. uniform$(0, 1)$random variables.

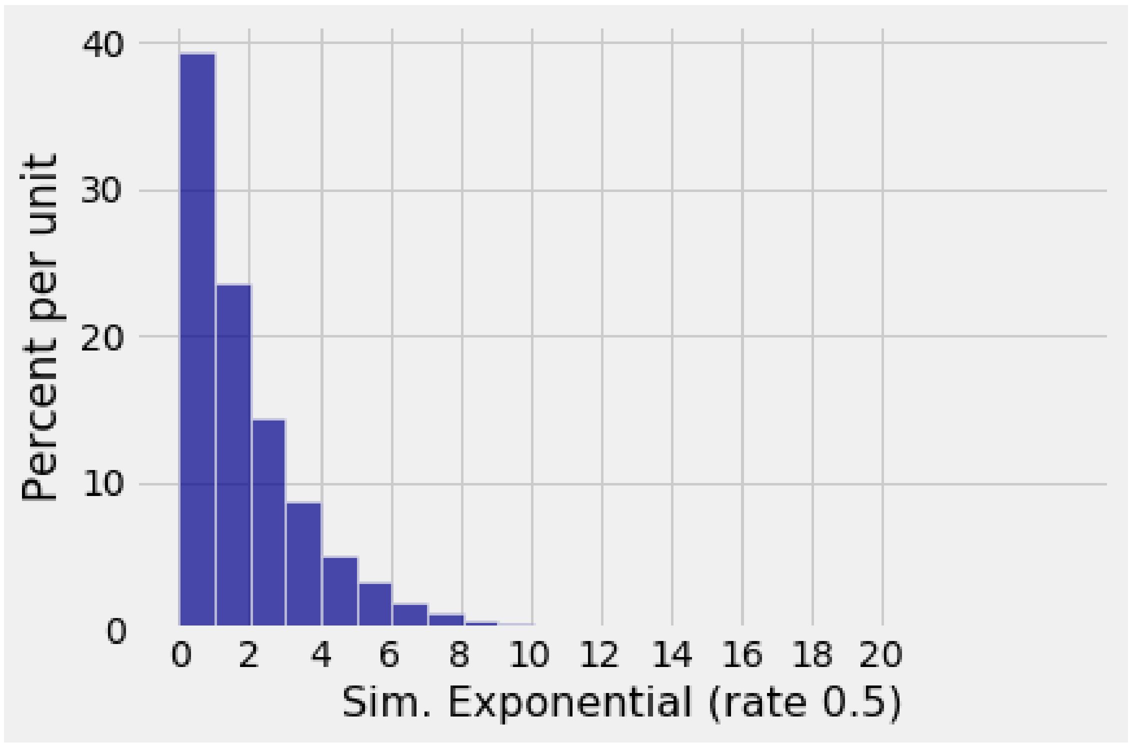
```
In [19]: N = 100000
         u = stats.uniform.rvs(0,1, size = 100000)
         sim = Table().with_column("Uniform", u)
         sim
```

```
Out[19]: Uniform
         0.572225
         0.263104
         ... Omitting 5 lines ...
         0.911534
         0.595276
         ... (99990 rows omitted)
```

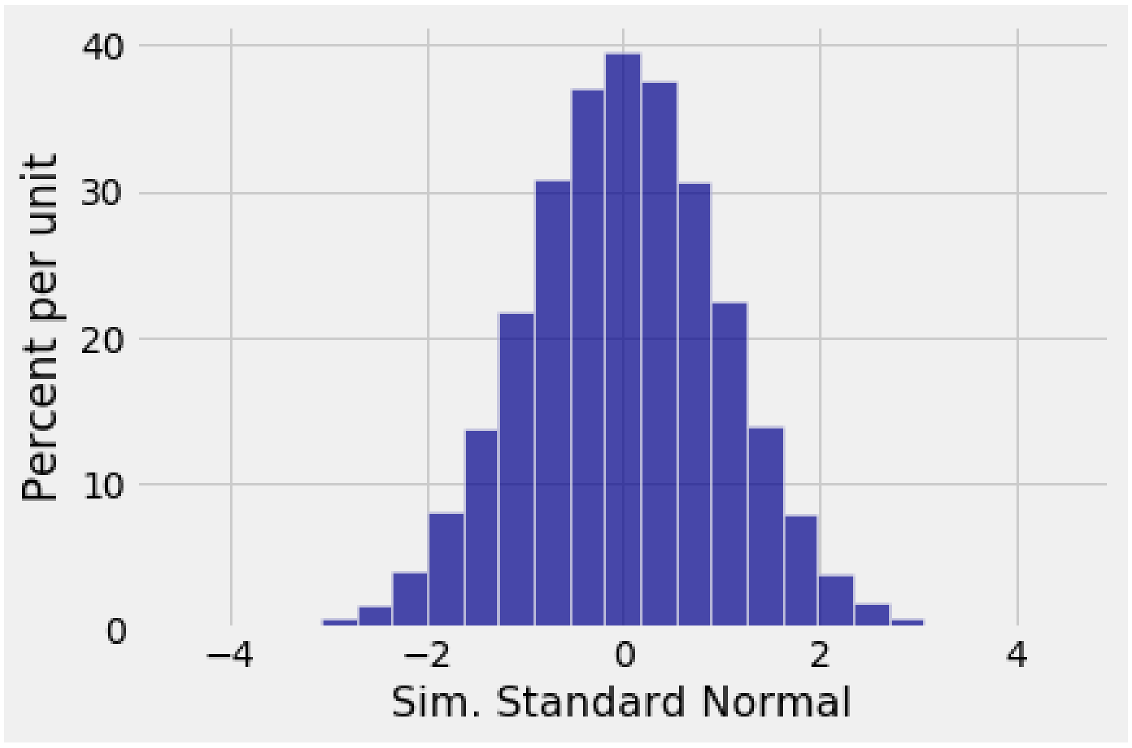
```
In [20]: def uniform_to_exponential_mean2(u):
             return np.log(1-u) / (-1*lamb)

         exponential_mean2 = sim.apply(uniform_to_exponential_mean2, 0)
         sim = sim.with_column('Sim. Exponential (rate 0.5)', exponential_mean2)
         sim
```

```
Out[20]: Uniform  | Sim. Exponential (rate 0.5)
         0.572225 | 1.69832
         0.263104 | 0.610618
         ... Omitting 5 lines ...
         0.911534 | 4.85027
         0.595276 | 1.8091
         ... (99990 rows omitted)
```

```
In [21]: sim.hist('Sim. Exponential (rate 0.5)', bins=25)
         plt.xticks(np.arange(0, 21, 2));
```



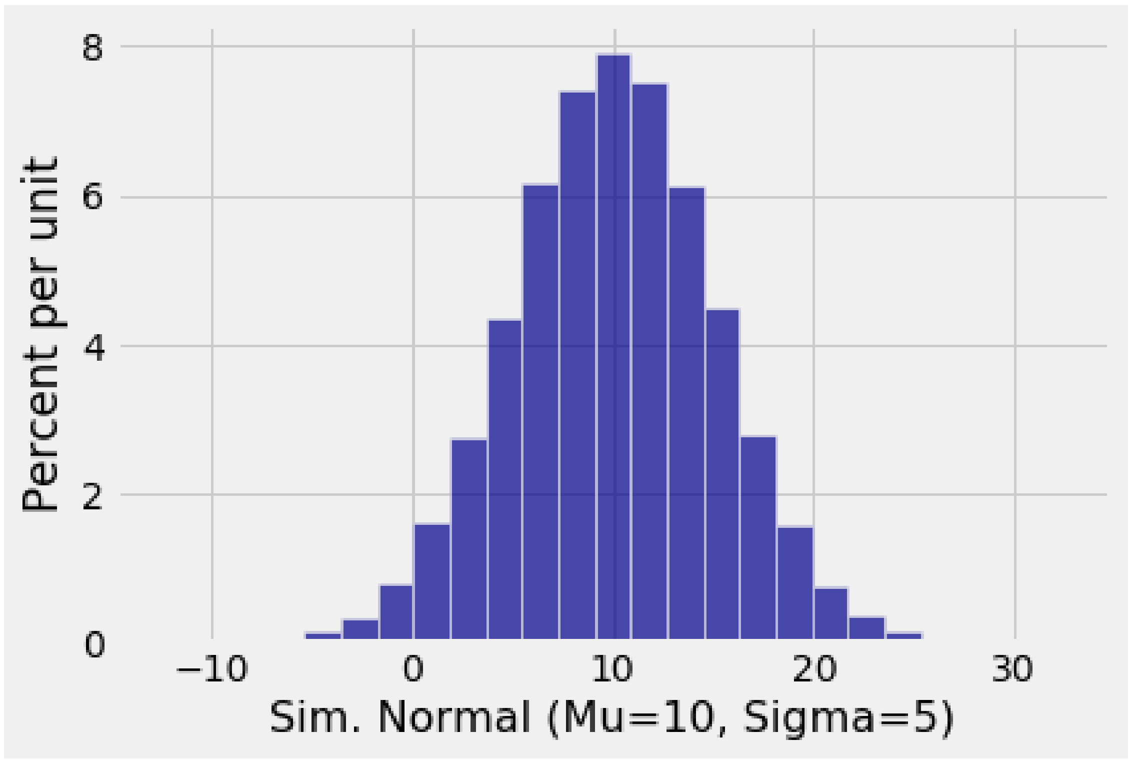
```
In [22]: standard_normal = sim.apply(stats.norm.ppf, 'Uniform')
         sim = sim.with_column('Sim. Standard Normal', standard_normal)
         sim

Out[22]: Uniform  | Sim. Exponential (rate 0.5) | Sim. Standard Normal
         0.572225 | 1.69832                     | 0.182042
         0.263104 | 0.610618                    | -0.633804
         ... Omitting 5 lines ...
         0.911534 | 4.85027                     | 1.35026
         0.595276 | 1.8091                      | 0.241139
         ... (99990 rows omitted)

In [23]: sim.hist('Sim. Standard Normal', bins=25)
```



```
In [24]: z = sim.column('Sim. Standard Normal')
         sim = sim.with_column('Sim. Normal (Mu=10, Sigma=5)', z*5+10)
         sim
```

```
Out[24]: Uniform   | Sim. Exponential (rate 0.5) | Sim. Standard Normal | Sim. Normal (Mu=10, Sigma=5)
         0.572225 | 1.69832                     | 0.182042             | 10.9102
         0.263104 | 0.610618                    | -0.633804            | 6.83098
         ... Omitting 5 lines ...
         0.911534 | 4.85027                     | 1.35026              | 16.7513
         0.595276 | 1.8091                      | 0.241139             | 11.2057
         ... (99990 rows omitted)
```

```
In [25]: sim.hist('Sim. Normal (Mu=10, Sigma=5)', bins=25)
```


Percent per unit
8
6
4
2
0
−10
0
10
20
30
Sim. Normal (Mu=10, Sigma=5)

6 newpage

6.1 Part 6. Radial Distance

You can apply the general method developed above to simulate values of any continuous random variable. Here is an example.

Consider a point(X, Y)picked uniformly on the unit disc$\{(x, y) : x^2 + y^2 \leq 1\}$. That's the disc with radius 1 centered at the origin (0, 0).

Let$R$be the distance between the point$(X, Y)$and the center$(0, 0)$.

The point$(X, Y)$is random, so the radial distance$R$is random as well and has a density.

6.1.1 6a) Visualization

Run the cell below. The figure on the left shows simulated i.i.d. copies of the point. On the right you have the empirical histogram of the simulated distances. Move the slider to increase the number of simulations.

```
In [31]: def radial(u):
             return (u/np.pi)**0.5

         radial_dist = sim.apply(radial, 'Uniform')
         sim = sim.with_column('Sim. Radial Distance', radial_dist)

         sim
```

```
Out[31]: Uniform  | Sim. Exponential (rate 0.5) | Sim. Standard Normal | Sim. Normal (Mu=10, Sigma=5) |
         0.572225 | 1.69832                     | 0.182042             | 10.9102                      |
         0.263104 | 0.610618                    | -0.633804            | 6.83098                      |
         ... Omitting 5 lines ...
         0.911534 | 4.85027                     | 1.35026              | 16.7513                      |
         0.595276 | 1.8091                      | 0.241139             | 11.2057                      |
         ... (99990 rows omitted)
```