Notebook

October 19, 2018

Local date & time is : 10/19/2018 23:30:35 PDT

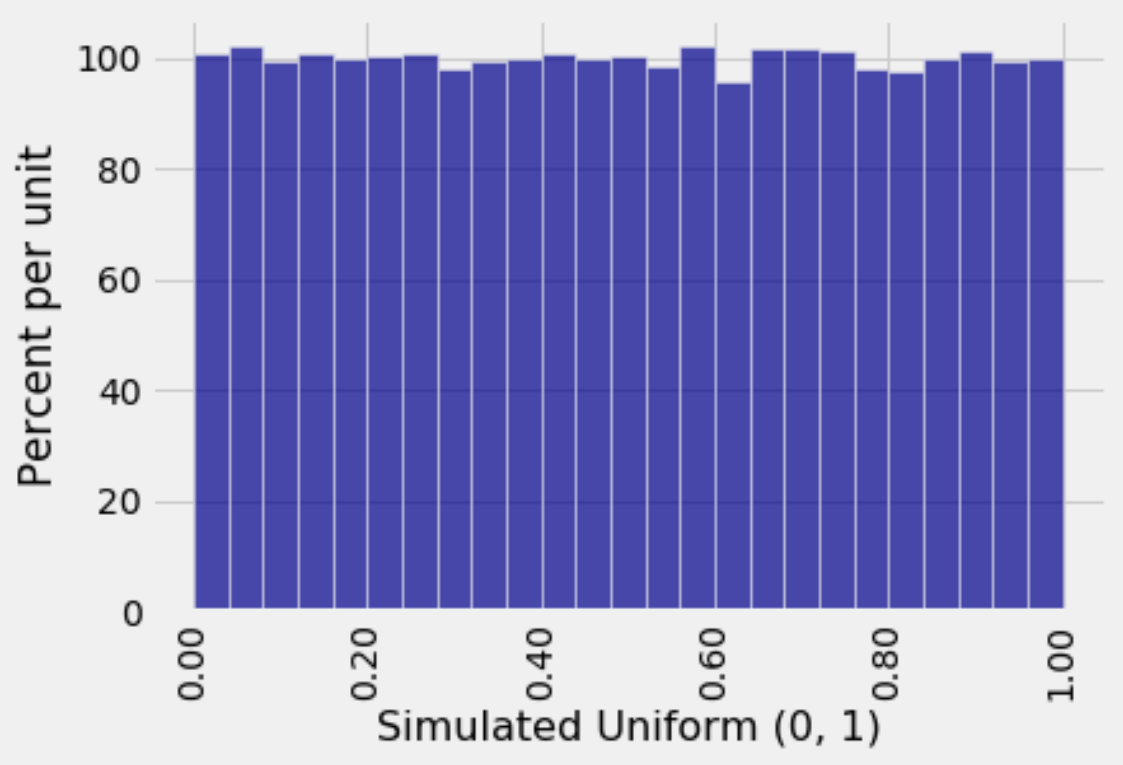
1 newpage

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.

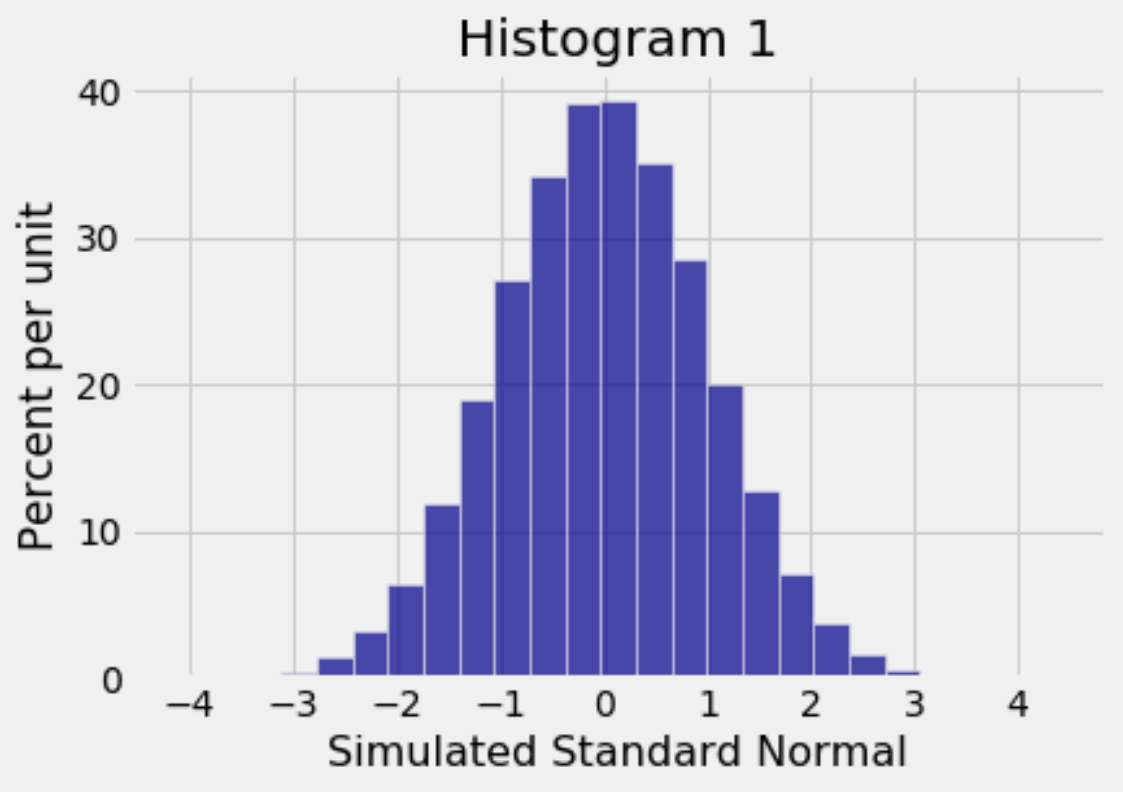
Every statistical system has conventions for how to specify the parameters of a distribution. In this lab we will tell you the specifications for a few distributions. Later you will be able to see a general pattern in the specifications.

```
In [94]: 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)
```

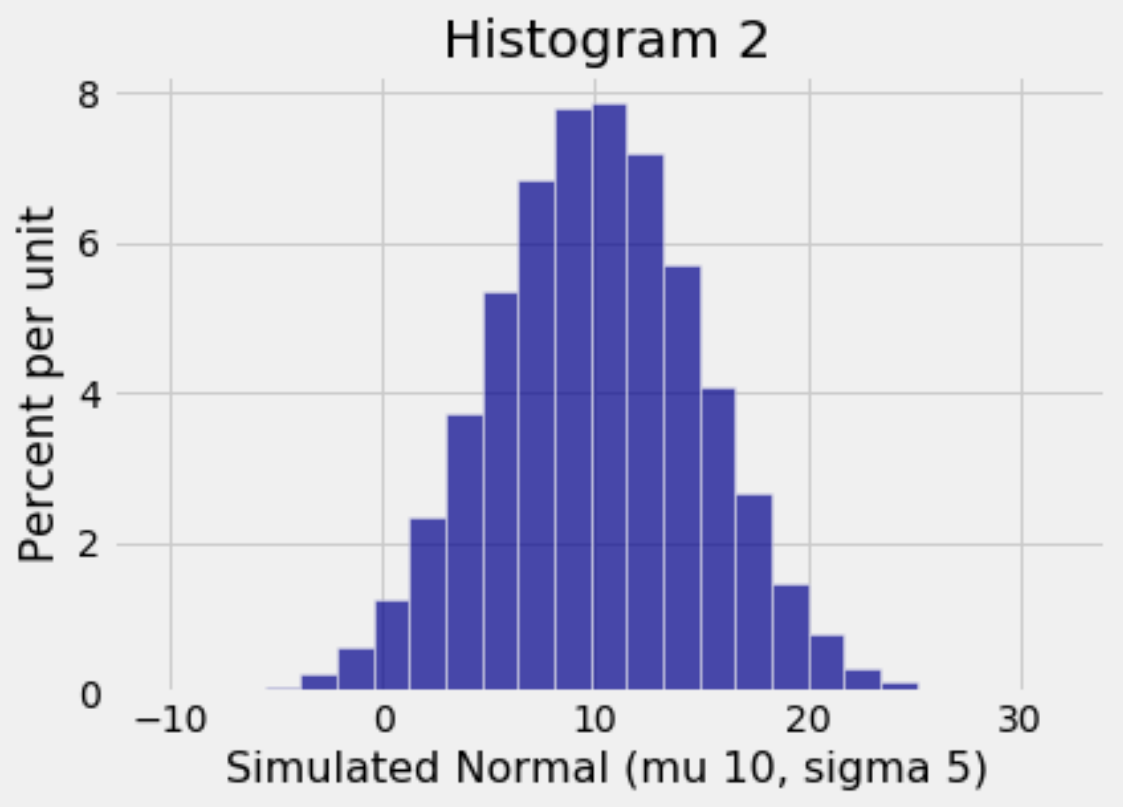


(i) 0.04
(ii) 100 percent per unit
(iii) area, 4
(iv) 4
(v) 5

```
In [95]: sim_std_norm = stats.norm.rvs(0, 1, size = 100000)
         sim_std_norm_tbl = Table().with_column('Simulated Standard Normal', sim_std_norm)
         sim_std_norm_tbl.hist(bins = 25)
         plt.xticks(np.arange(-4, 4.1))
         plt.title('Histogram 1');
```




```
In [96]: sim_norm = stats.norm.rvs(10, 5, size = 100000)
         sim_norm_tbl = Table().with_column('Simulated Normal (mu 10, sigma 5)', sim_norm)
         sim_norm_tbl.hist(bins = 25)
         plt.title('Histogram 2');
```



Compare the numbers on the horizontal axes of the two histograms, and fill in the blank.

The value **-1** on the horizontal axis of Histogram 1 is the same as the value5on the horizontal axis of Histogram 2 expressed in **standard** units.

```
In [98]: np.mean(sim_expon) # Yes it is consistent with 1/0.5 = 2
```

```
Out[98]: 1.9971716336478547
```

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

Generate a value of U. Find out which of the four intervals does it belong to :$(U \leq 0.3)$,$(0.3 < U \leq 0.4)$,$(0.4 < U \leq 0.6)$or$(0.6 < U \leq 1)$. Based on that X will take its value. So for example if we find U falls in the interval$(0.3 < U \leq 0.4)$then we will generate X as 1 with probability 0.1

3 newpage

3.1 Part 3. Visualizing the Idea

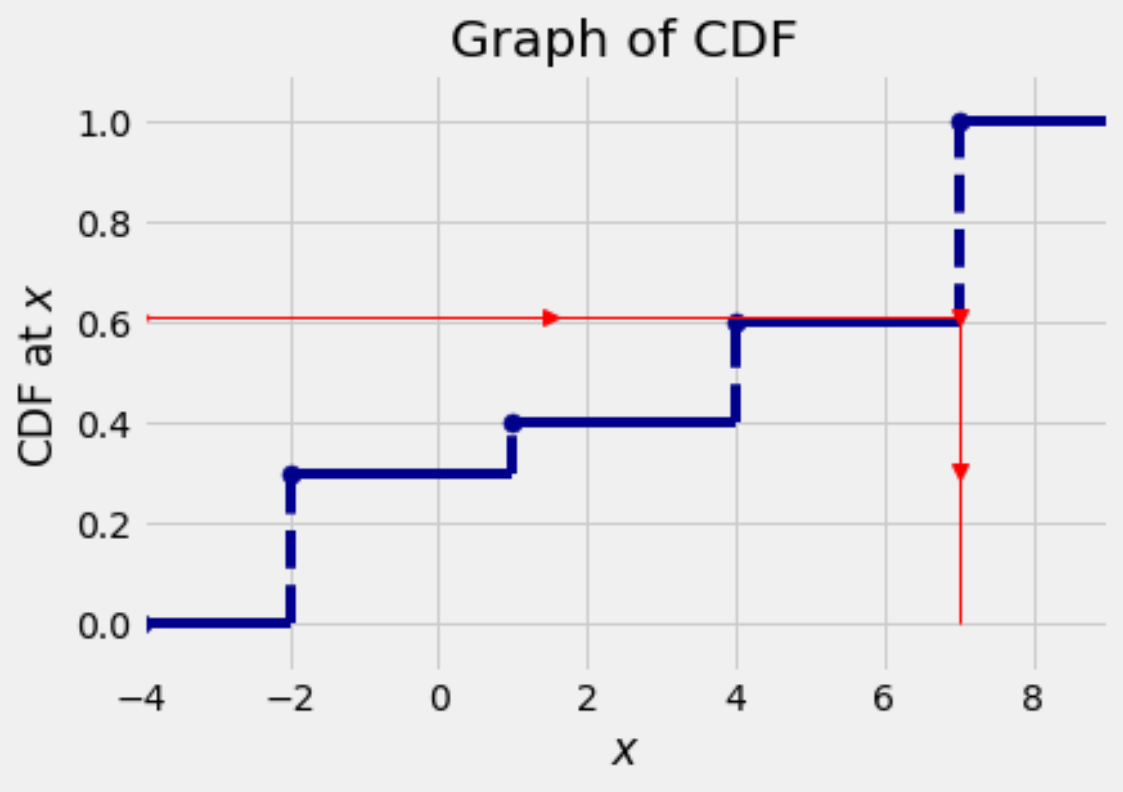
The method `plot_discrete_cdf` takes a distribution as its argument and plots a graph of the cdf.

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

$F_X(2.57) = P(X \leq 2.57)$. Yes the graph shows the correct value of $F_X(2.57)$ which is 0.4

At -2, 1, 4, and 7 the graph has jumps. Size of the jump at $x = -2$ is 0.3 which is equal to $P(X = -2)$. Size of the jump at $x = 1$ is 0.1 which is equal to $P(X = 1)$. Size of the jump at $x = 4$ is 0.2 which is equal to $P(X = 4)$. Size of the jump at $x = 7$ is 0.4 which is equal to $P(X = 7)$

```
In [103]: plot_discrete_cdf(dist_X, stats.uniform.rvs(0, 1, size = 1))
```



It's very closely related to the method proposed in Part 1. In Part 1, we decided the value of X based on whichever interval the value of U was in. Here, similar to part 1, we apply a function to uniformly randomly generated number (between 0 and 1) so that it maps to a value of X.

4 newpage

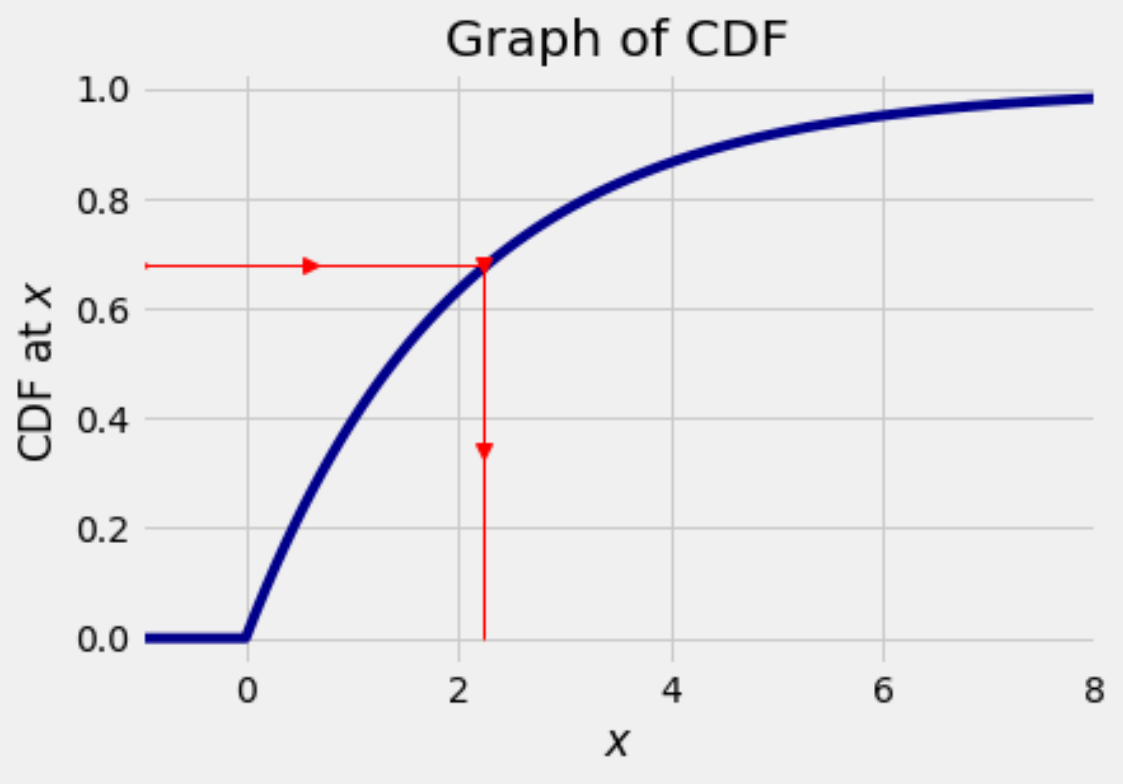
4.1 Part 4. Extension to Continuous Distributions

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

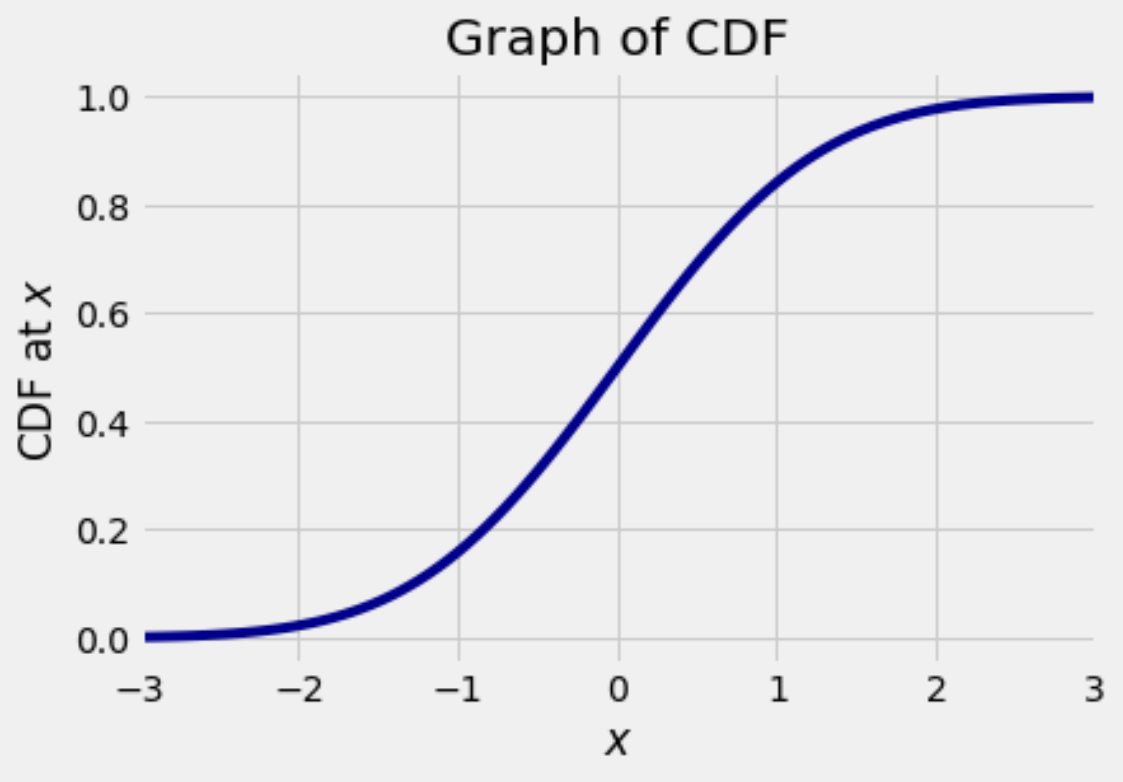
In [104]:
```
# don't use "lambda" as that means something else in Python
lamb = 0.5

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

We apply a function that is inverse of cdf to the U to get value of X

In [107]:
```
plot_continuous_cdf((-1, 8), expon_mean2_cdf, stats.uniform.rvs(0, 1, size = 1))
```



In [108]:
```
plot_continuous_cdf((-3, 3), stats.norm.cdf)
```


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

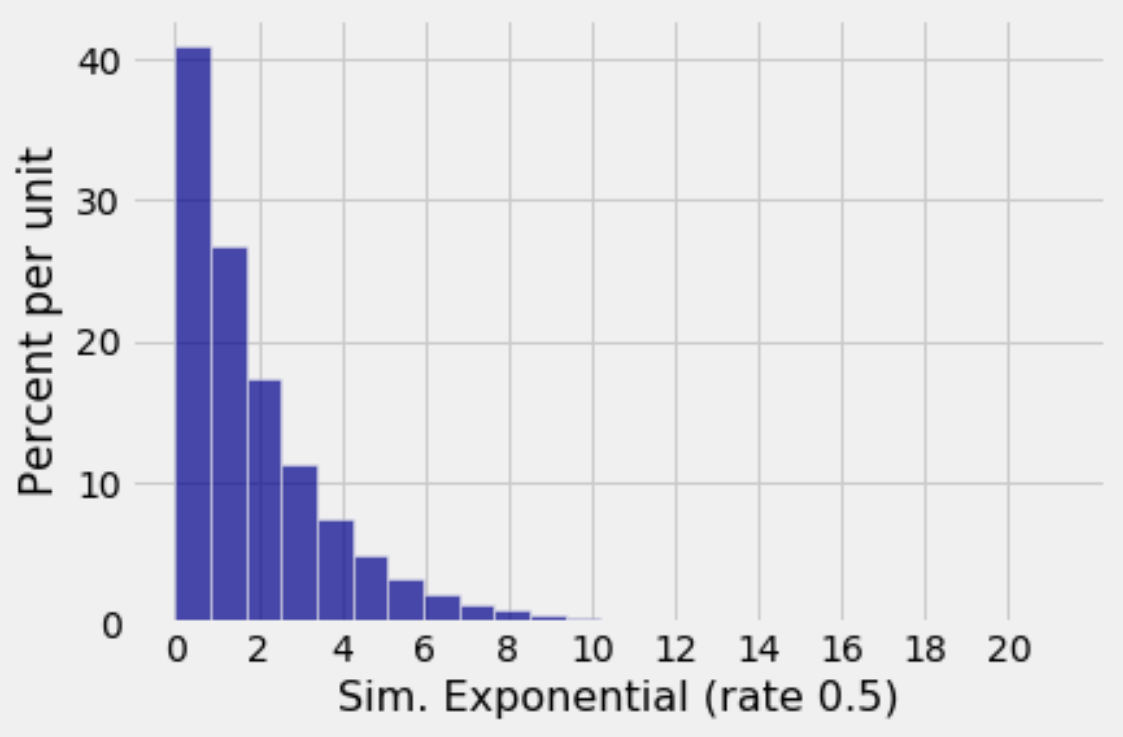
5 newpage

5.1 Part 5. Empirical Verification that the Method Works

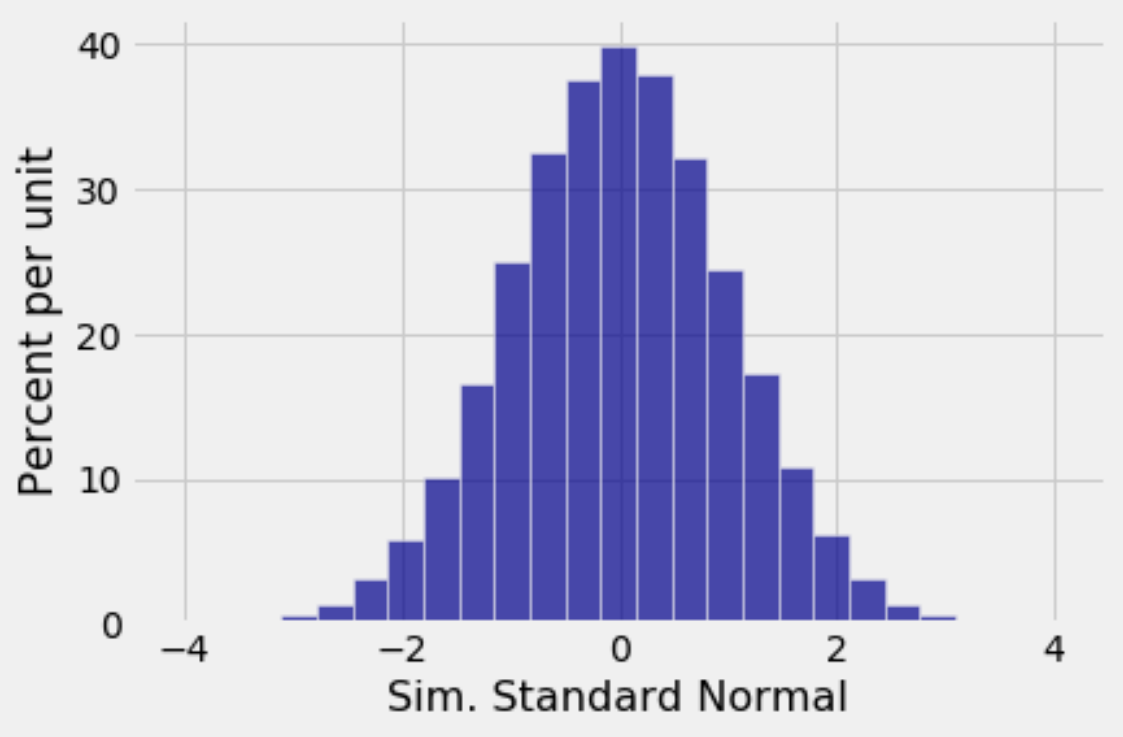
5.1.1 a) The Initial Values

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.

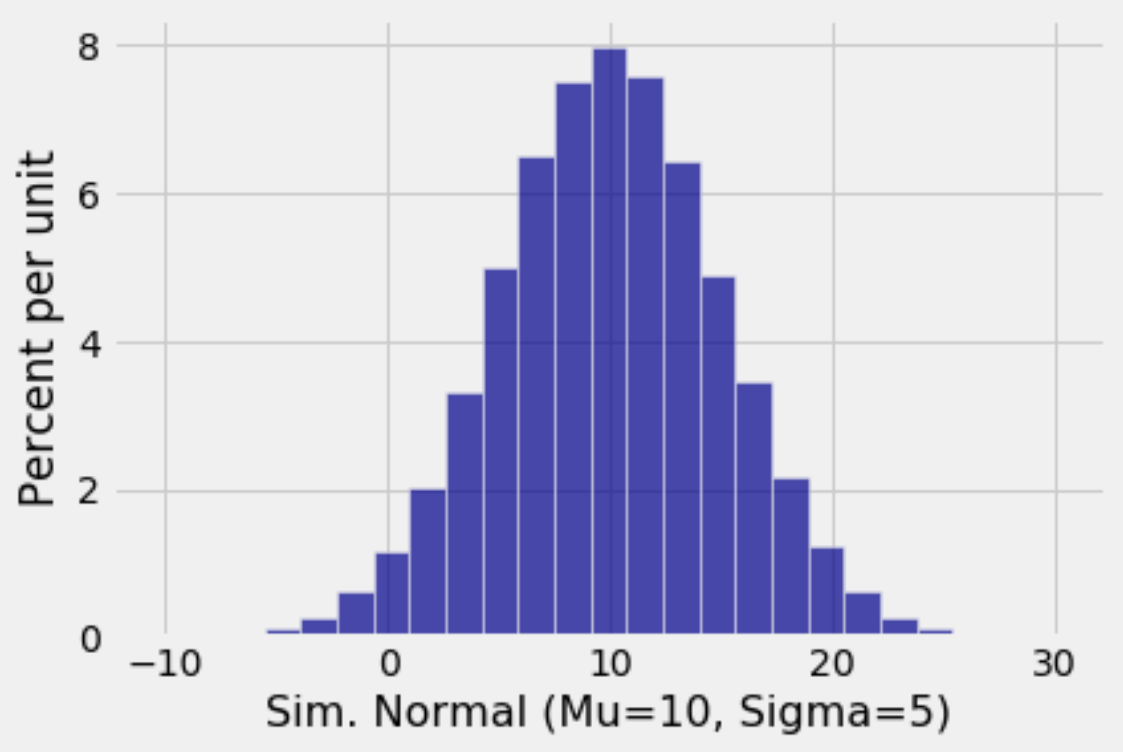
```
In [109]: N = 100000
          u = stats.uniform.rvs(0, 1, size = 100000)
          sim = Table().with_column("Uniform", u)
          sim

Out[109]: Uniform
          0.721201
          0.956558
          ... Omitting 5 lines ...
          0.278105
          0.9805
          ... (99990 rows omitted)

In [110]: def uniform_to_exponential_mean2(u):
              return -(2 * np.log(1-u))

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

Out[110]: Uniform   | Sim. Exponential (rate 0.5)
          0.721201  | 2.55453
          0.956558  | 6.27267
          ... Omitting 5 lines ...
          0.278105  | 0.651751
          0.9805    | 7.87471
          ... (99990 rows omitted)

In [111]: sim.hist('Sim. Exponential (rate 0.5)', bins=25)
          plt.xticks(np.arange(0, 21, 2));
```




```
In [112]: def norm_ppf(k):
              """Optional helper function."""
              return 0

          standard_normal = sim.apply(stats.norm.ppf, 'Uniform')
          sim = sim.with_column('Sim. Standard Normal', standard_normal)
          sim

Out[112]: Uniform   | Sim. Exponential (rate 0.5) | Sim. Standard Normal
          0.721201  | 2.55453                     | 0.586413
          0.956558  | 6.27267                     | 1.71207
          ... Omitting 5 lines ...
          0.278105  | 0.651751                    | -0.58848
          0.9805    | 7.87471                     | 2.06419
          ... (99990 rows omitted)

In [113]: sim.hist('Sim. Standard Normal', bins=25)
```




```
In [114]: z = sim.column('Sim. Standard Normal')
          sim = sim.with_column('Sim. Normal (Mu=10, Sigma=5)', (z * 5) + 10)
          sim

Out[114]: Uniform   | Sim. Exponential (rate 0.5) | Sim. Standard Normal | Sim. Normal (Mu=10, Sigma=5)
          0.721201  | 2.55453                     | 0.586413             | 12.9321
          0.956558  | 6.27267                     | 1.71207              | 18.5604
          ... Omitting 5 lines ...
          0.278105  | 0.651751                    | -0.58848             | 7.0576
          0.9805    | 7.87471                     | 2.06419              | 20.321
          ... (99990 rows omitted)

In [115]: 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 [117]: sim = sim.drop('Sim. Radial Distance', 'Sim. Radial Distance ')

In [118]: r = sim.column('Uniform')
          sim = sim.with_column('Sim. Radial Distance', np.sqrt(r))

          sim

Out[118]: Uniform   | Sim. Exponential (rate 0.5) | Sim. Standard Normal | Sim. Normal (Mu=10, Sigma=5)
          0.721201  | 2.55453                     | 0.586413             | 12.9321
          0.956558  | 6.27267                     | 1.71207              | 18.5604
          ... Omitting 5 lines ...
          0.278105  | 0.651751                    | -0.58848             | 7.0576
          0.9805    | 7.87471                     | 2.06419              | 20.321
          ... (99990 rows omitted)
```