The Monte Carlo process

MONTE CARLO SIMULATIONS IN PYTHON

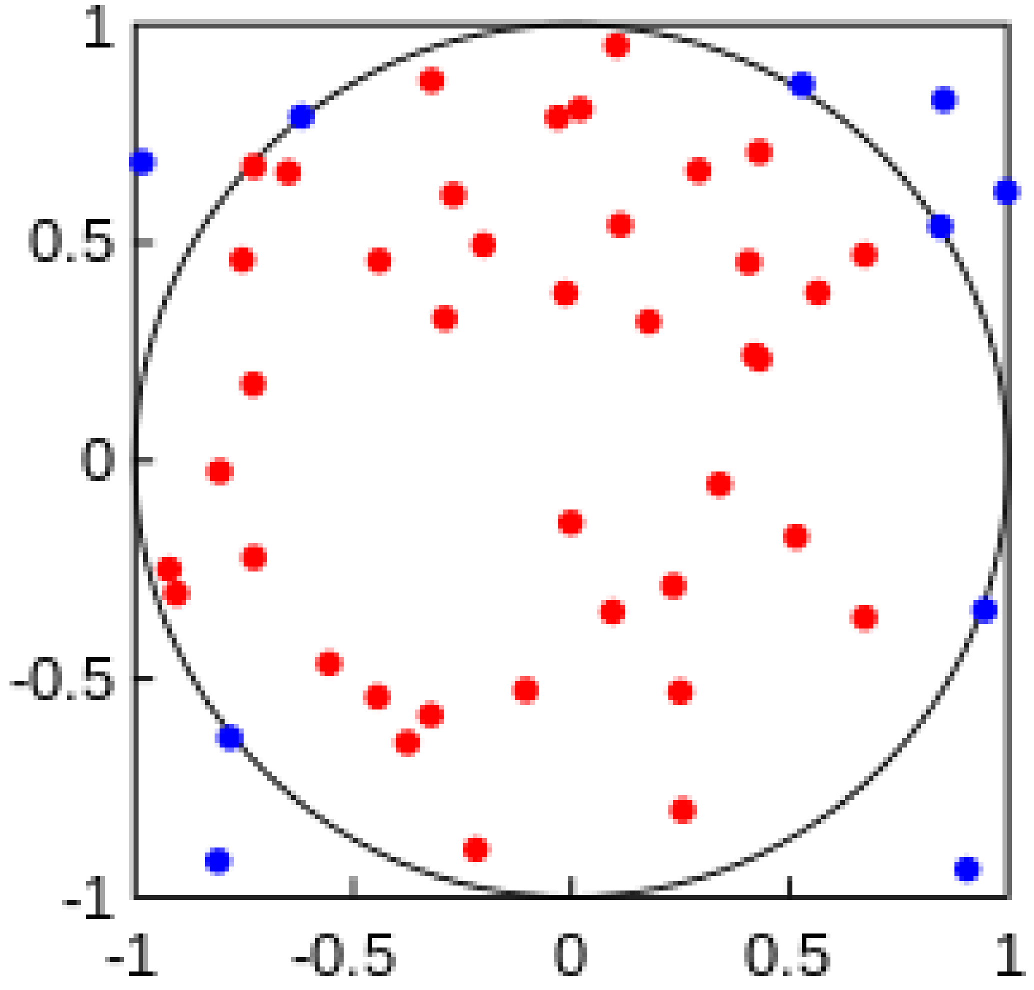
Izzy Weber
Curriculum Manager, DataCamp

Simulation steps

1. Define the input variables and pick probability distributions for them
2. Generate inputs by sampling from these distributions
3. Perform a deterministic calculation of the simulated inputs
4. Summarize results

Calculating the value of pi

Generate random points (x, y) *where* x *and* y *are in the interval from -1 to 1.*



$$Area_{circle} = \pi$$

$$Area_{square} = 2 \times 2 = 4$$

$$\frac{Area_{circle}}{Area_{square}} = \frac{\pi}{4}$$

$$\frac{n_{red}}{n_{all}} = \frac{\pi}{4}$$

$$\pi = 4 \times \frac{n_{red}}{n_{all}}$$

Step 1

Define the input variables and pick probability distributions for them

- **Inputs:** the individual points represented by (x, y) coordinates
- **Probability distributions:** x and y follow uniform distributions from negative one to one.

```
circle_points = 0
square_points = 0
```

Step 2

Generate inputs by sampling from these distributions

Sample random x and y coordinate values distributed uniformly between -1 and 1:

```
for i in range(n):
    x = random.uniform(-1, 1)
    y = random.uniform(-1, 1)
```

Step 3

Perform deterministic calculation of the simulated inputs

Check whether each point lies within the circle: deterministic for given x and y

```
dist_from_origin = x**2 + y**2
```

If yes, add the point to `circle_points` ; always add the point to `square_points`

```
if dist_from_origin <= 1:
    circle_points += 1
square_points += 1
```

Step 4

Summarize the results to answer questions of interest

After many rounds of simulations, calculate the value of pi!

```
pi = 4 * circle_points/ square_points
```

All together now

```
n = 4000000
circle_points = 0
square_points = 0
for i in range(n):
    x = random.uniform(-1, 1)
    y = random.uniform(-1, 1)
    dist_from_origin = x**2 + y**2
    if dist_from_origin <= 1:
        circle_points += 1
    square_point += 1
pi = 4 * circle_points / square_points
print(pi)
```

```
3.142518
```

Let's practice!

MONTE CARLO SIMULATIONS IN PYTHON

Generating discrete random variables

MONTE CARLO SIMULATIONS IN PYTHON

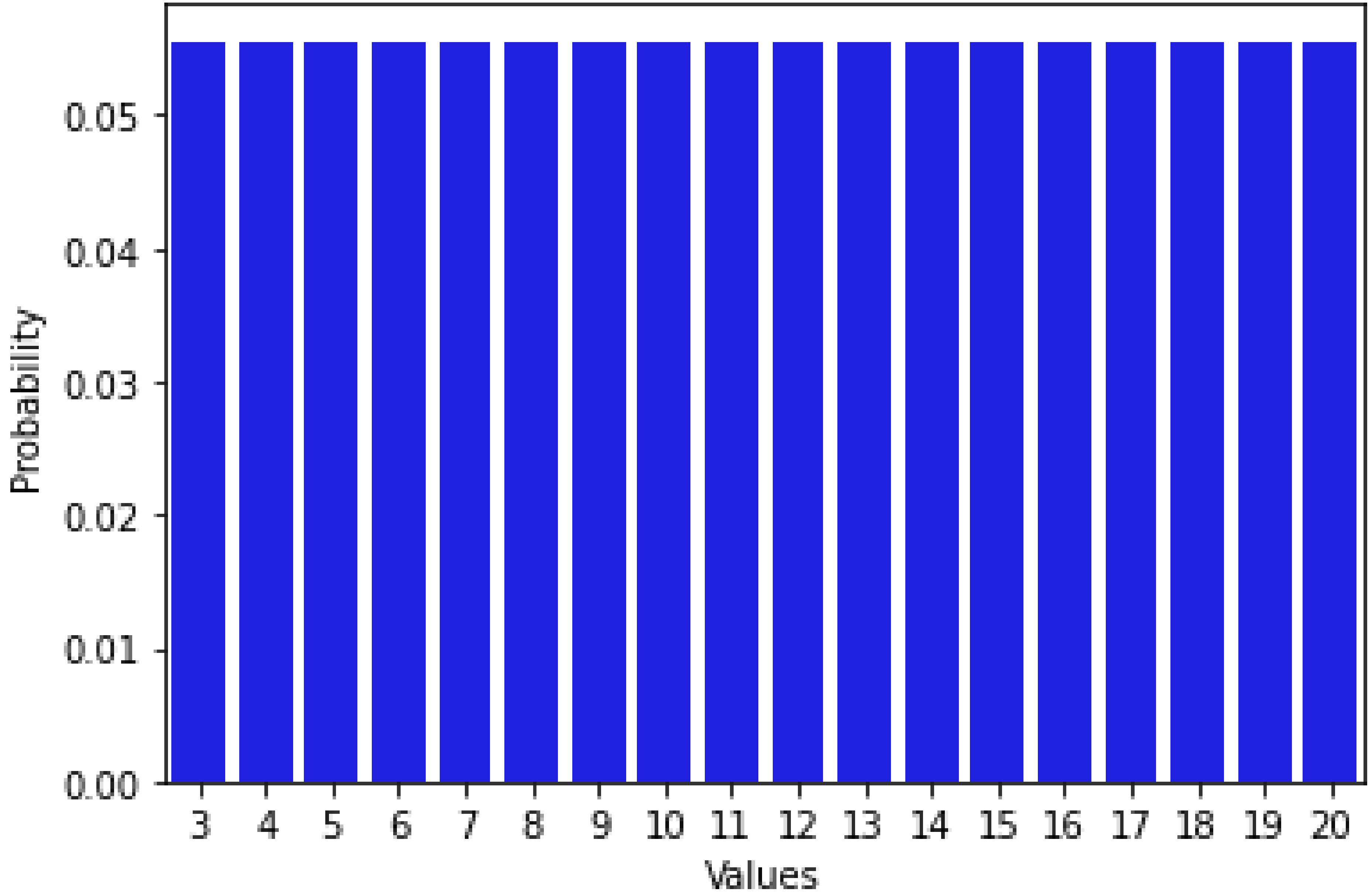
Izzy Weber
Curriculum Manager, DataCamp

Required imports

```
import scipy.stats as st
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

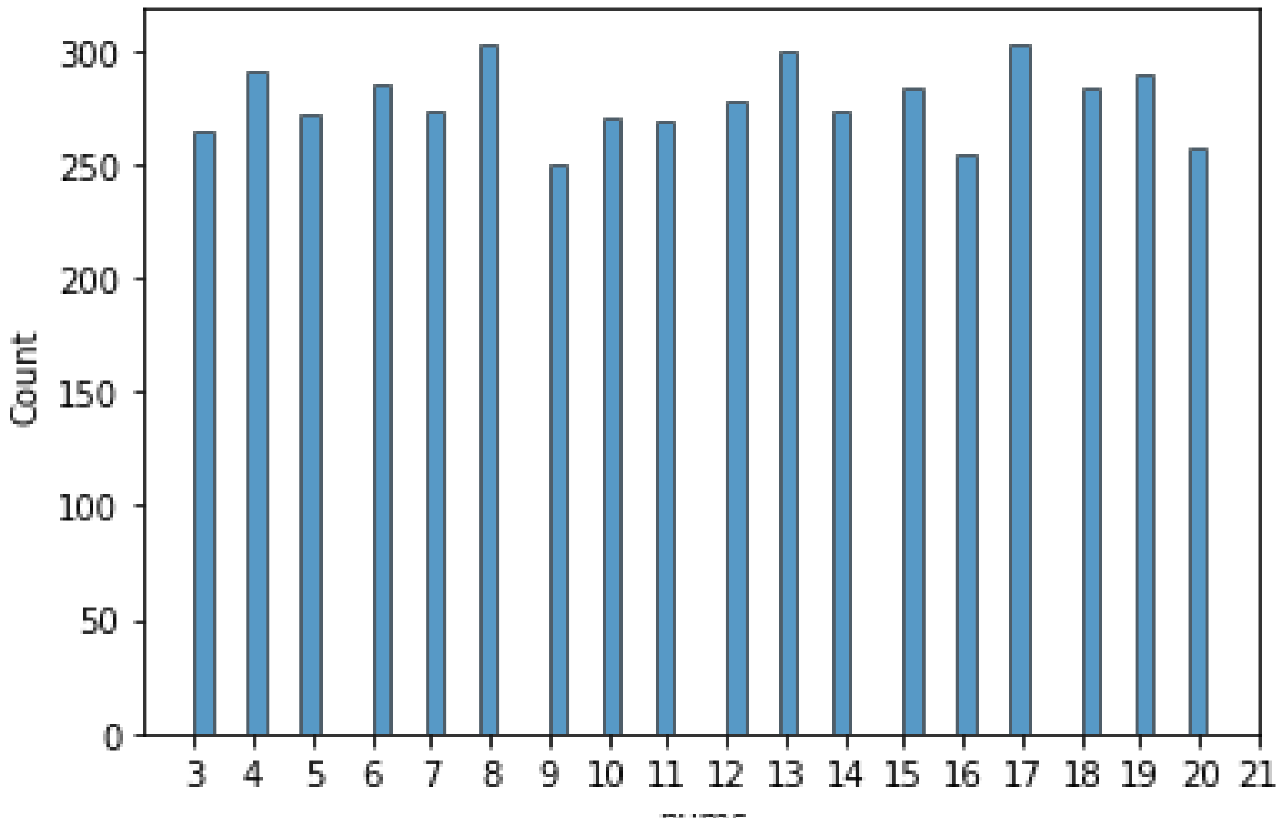
Discrete uniform distribution

Theoretical probability mass function (PMF):



Sampling from the discrete uniform distribution

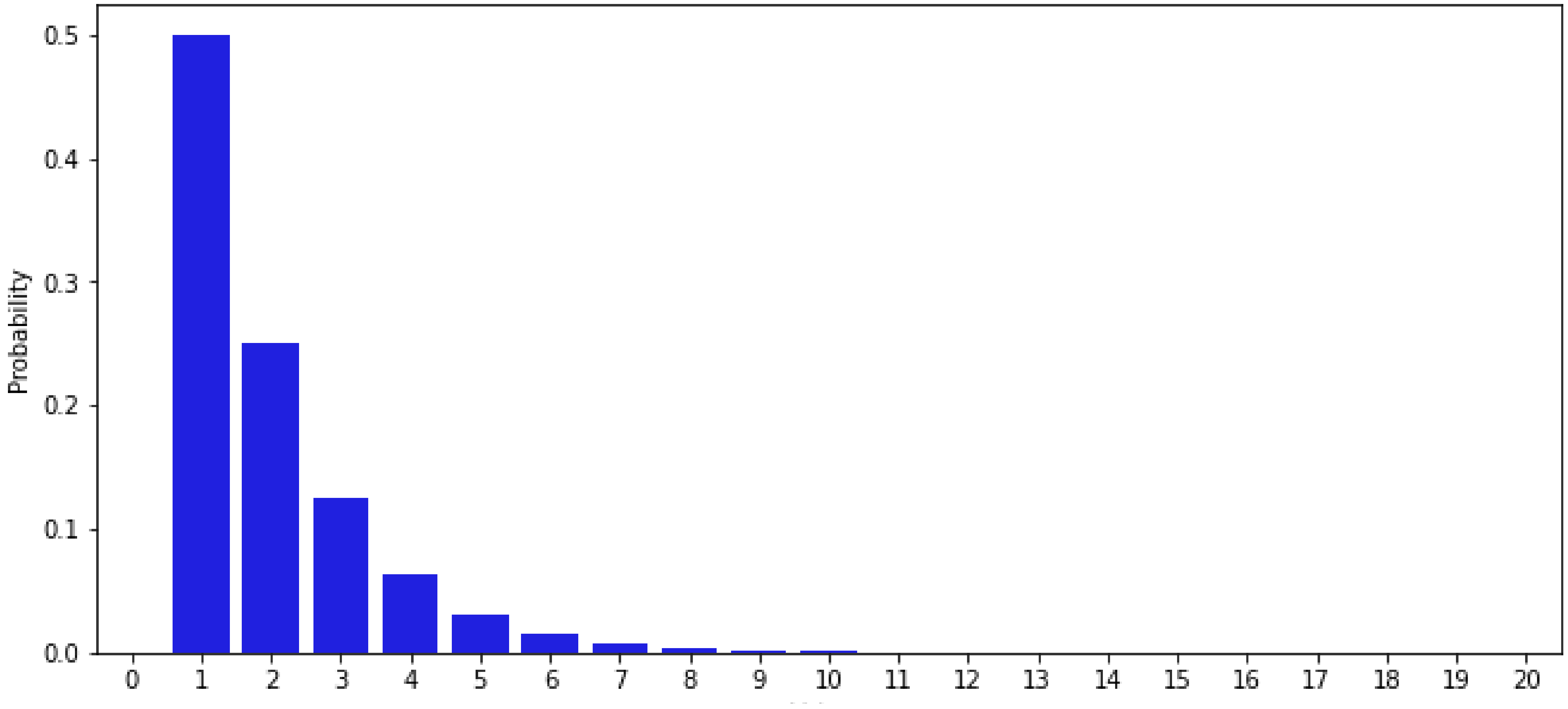
```
low = 3
high = 21
samples = st.randint.rvs(low, high, size=1000)
samples_dict = {"nums":samples}
sns.histplot(x="nums", data=samples_dict, bins=6, binwidth=0.3)
```



Geometric distribution

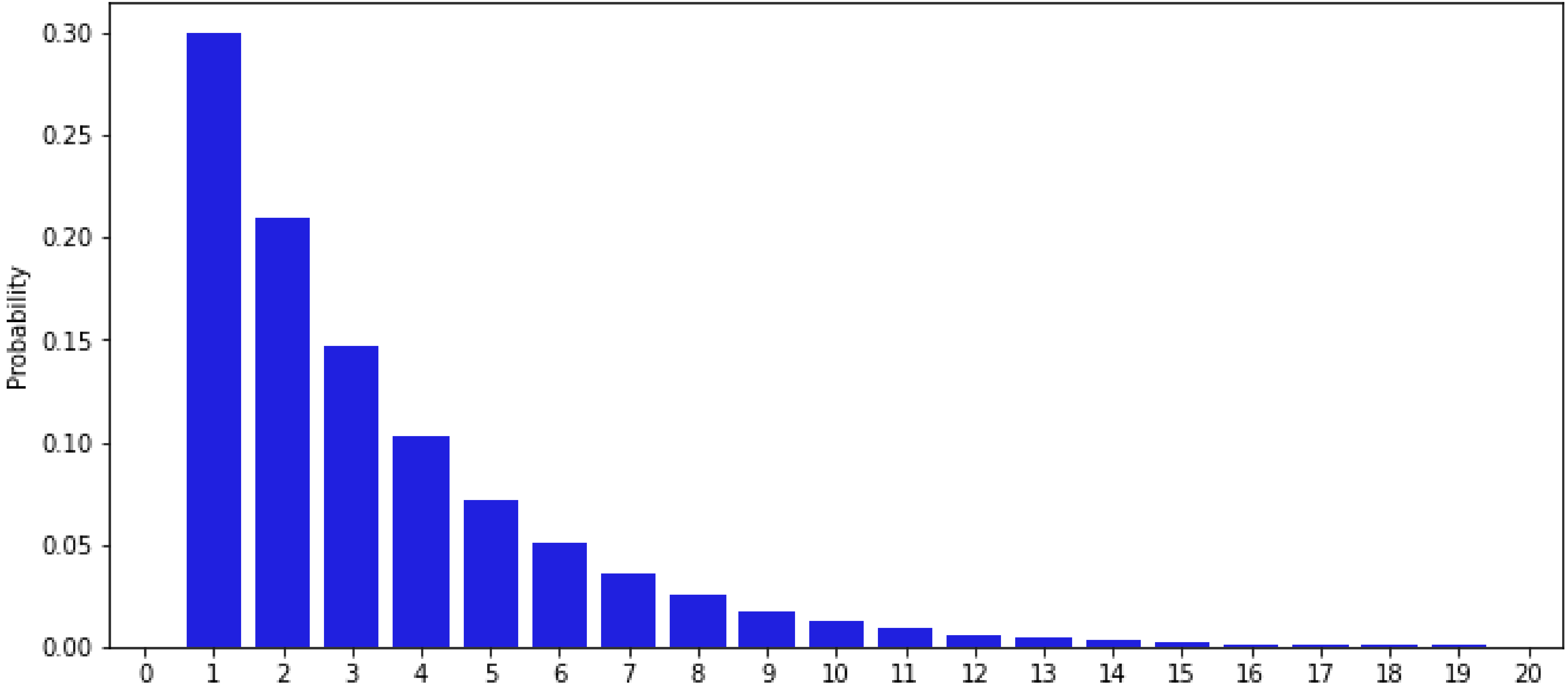
The probability distribution of the number of trials, X, needed to get one success, given the success probability, p.

Probability Mass Function, `p = 0.5`



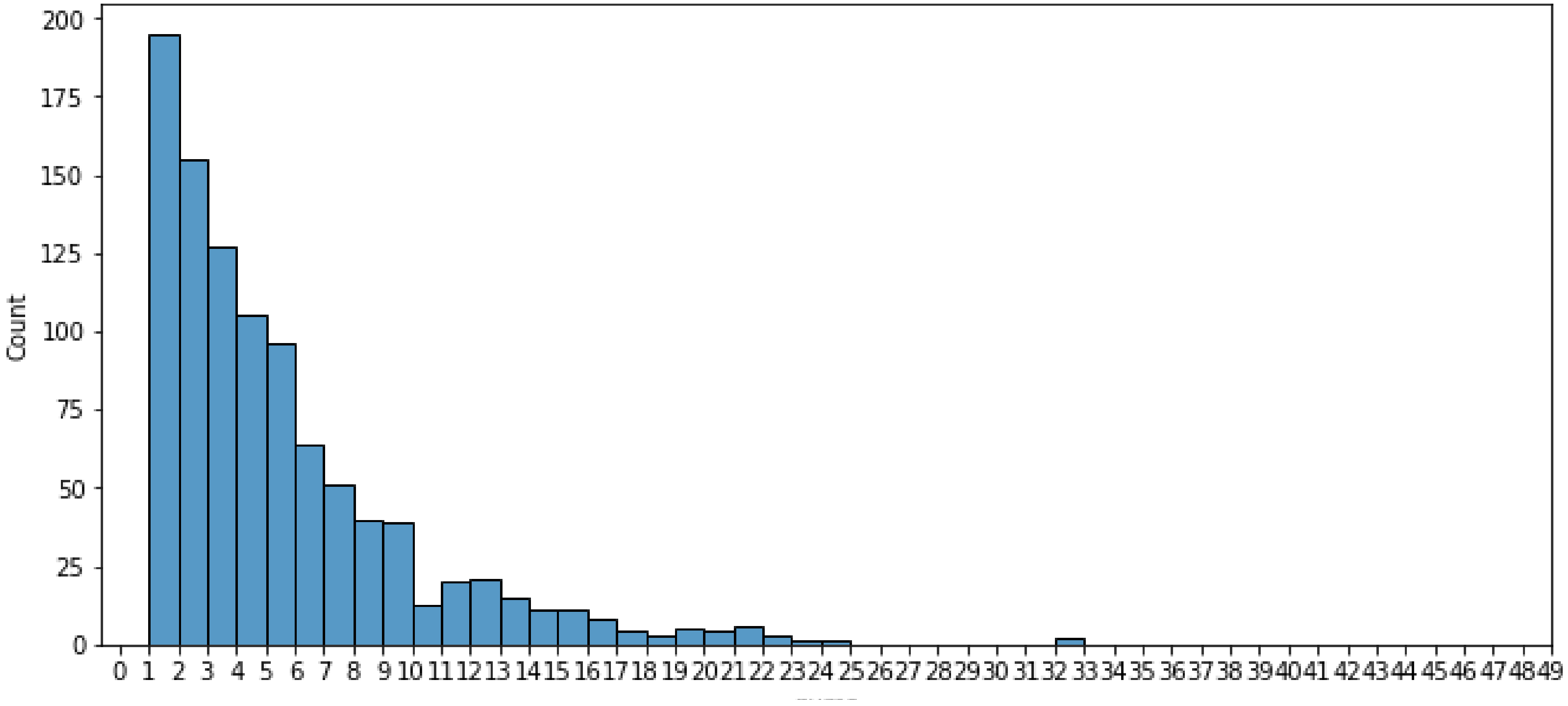
Geometric distribution

Probability Mass Function, `p = 0.3`



Sampling from geometric distribution

```
p = 0.2
samples = st.geom.rvs(p, size=1000)
samples_dict = {"nums":samples}
sns.histplot(x="nums", data=samples_dict)
```



More discrete probability distributions

- Poisson (`scipy.stats.poisson`)
  - Expresses the probability of a given number of events occurring in a fixed interval of time or space
- Binomial (`scipy.stats.binom`)
  - Expresses probability of the number of successes in a sequence of n independent experiments.
- And more!

[1] https://docs.scipy.org/doc/scipy/reference/stats.html#discrete-distributions

Let's practice!

MONTE CARLO SIMULATIONS IN PYTHON

Generating continuous random variables

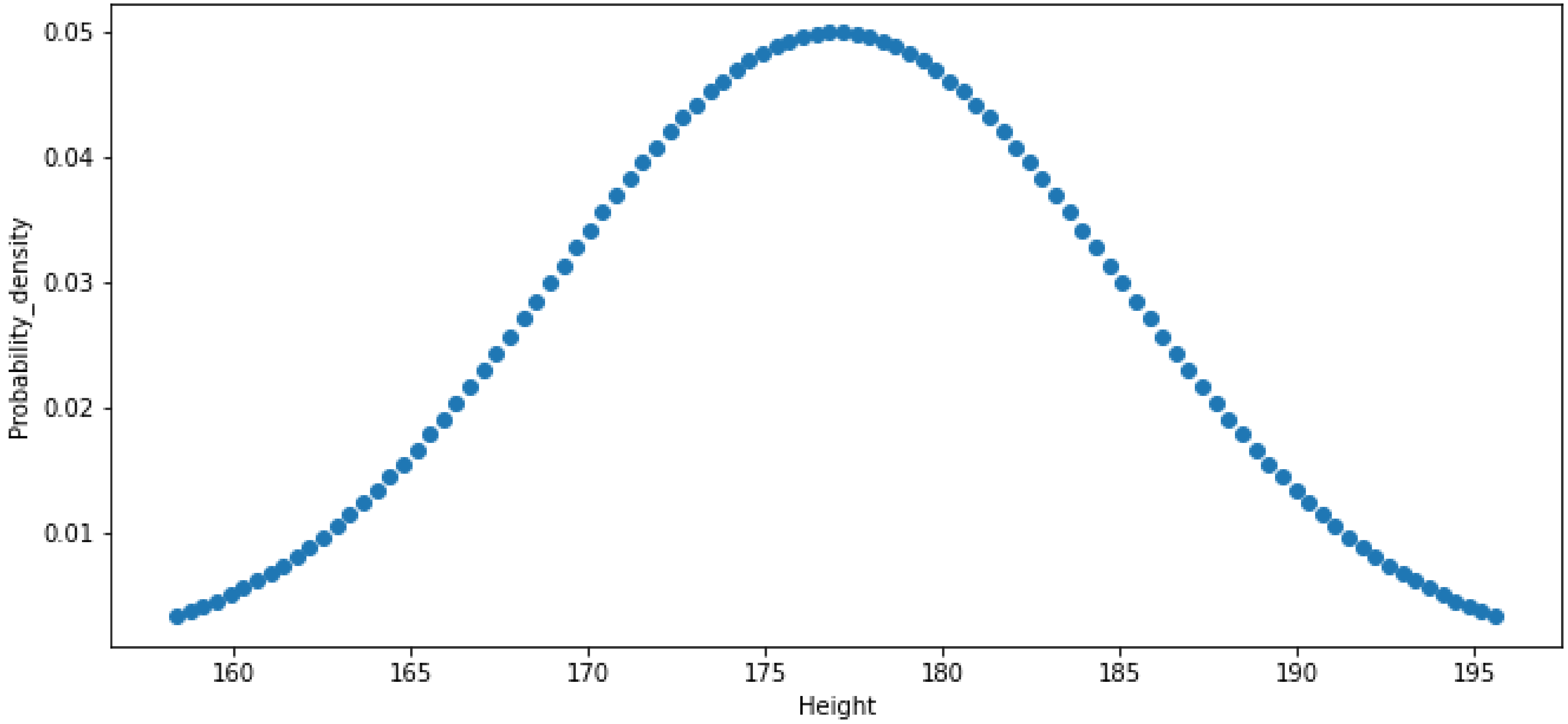
MONTE CARLO SIMULATIONS IN PYTHON

Izzy Weber
Curriculum Manager, DataCamp

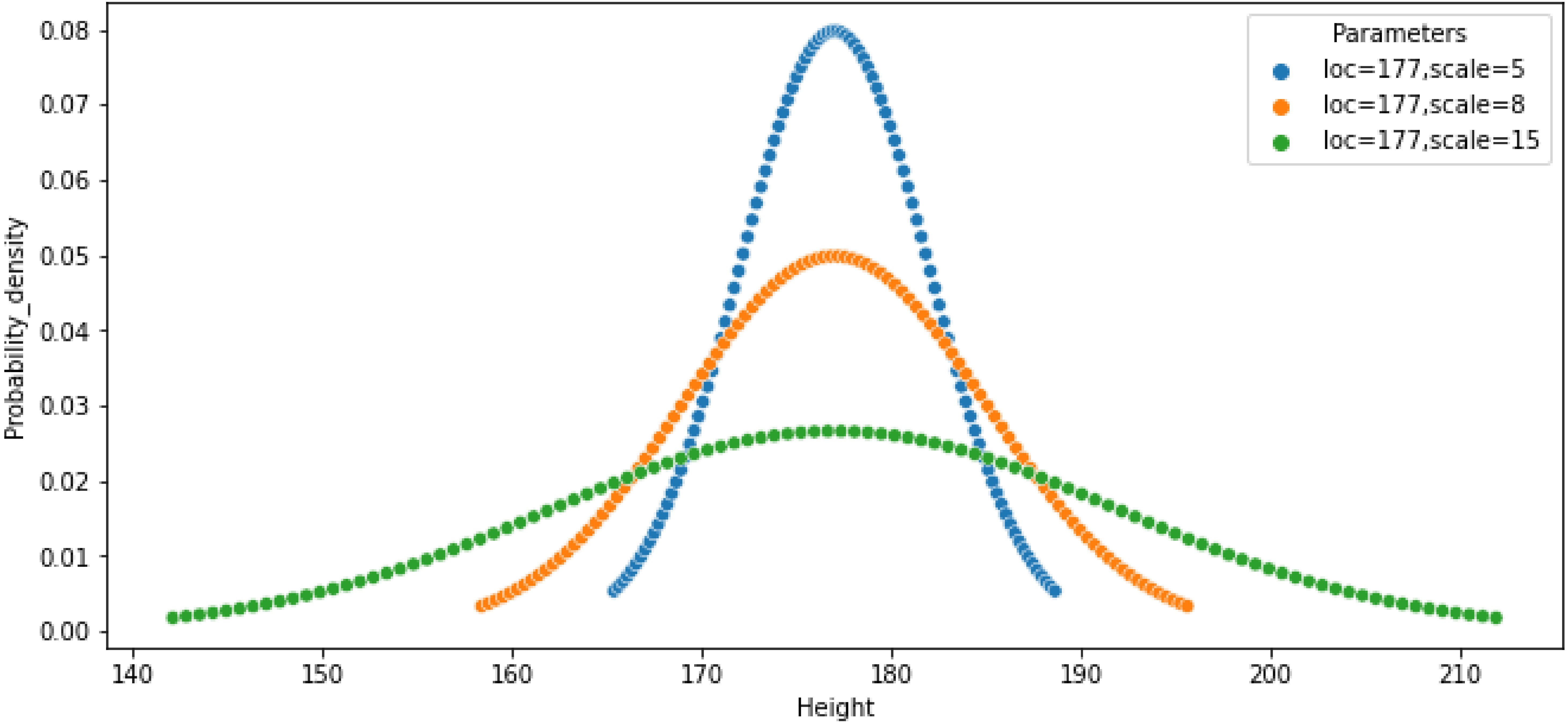
Normal distribution

Bell-shaped and centered at the mean (or loc); width defined by standard deviation (or scale)

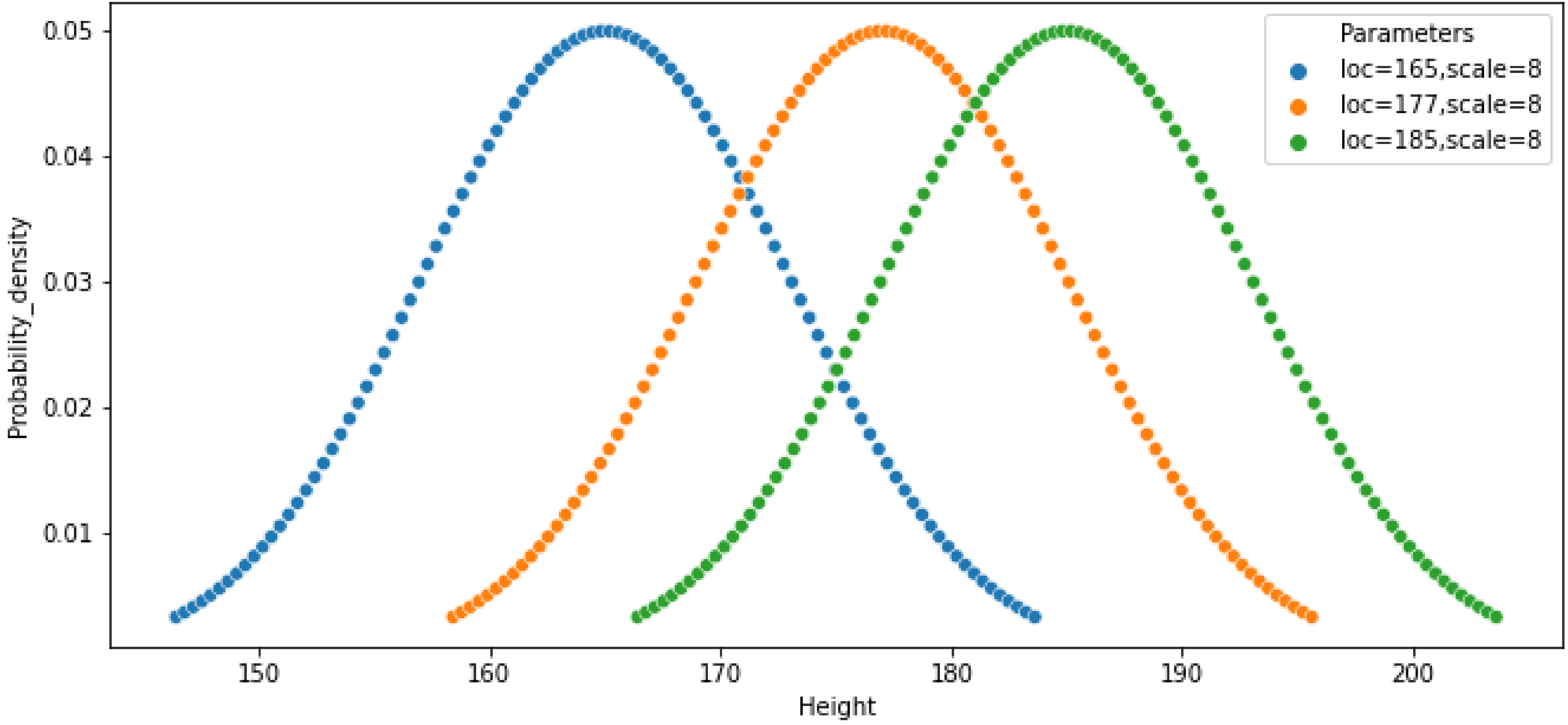
The heights of American adult males are normally distributed:



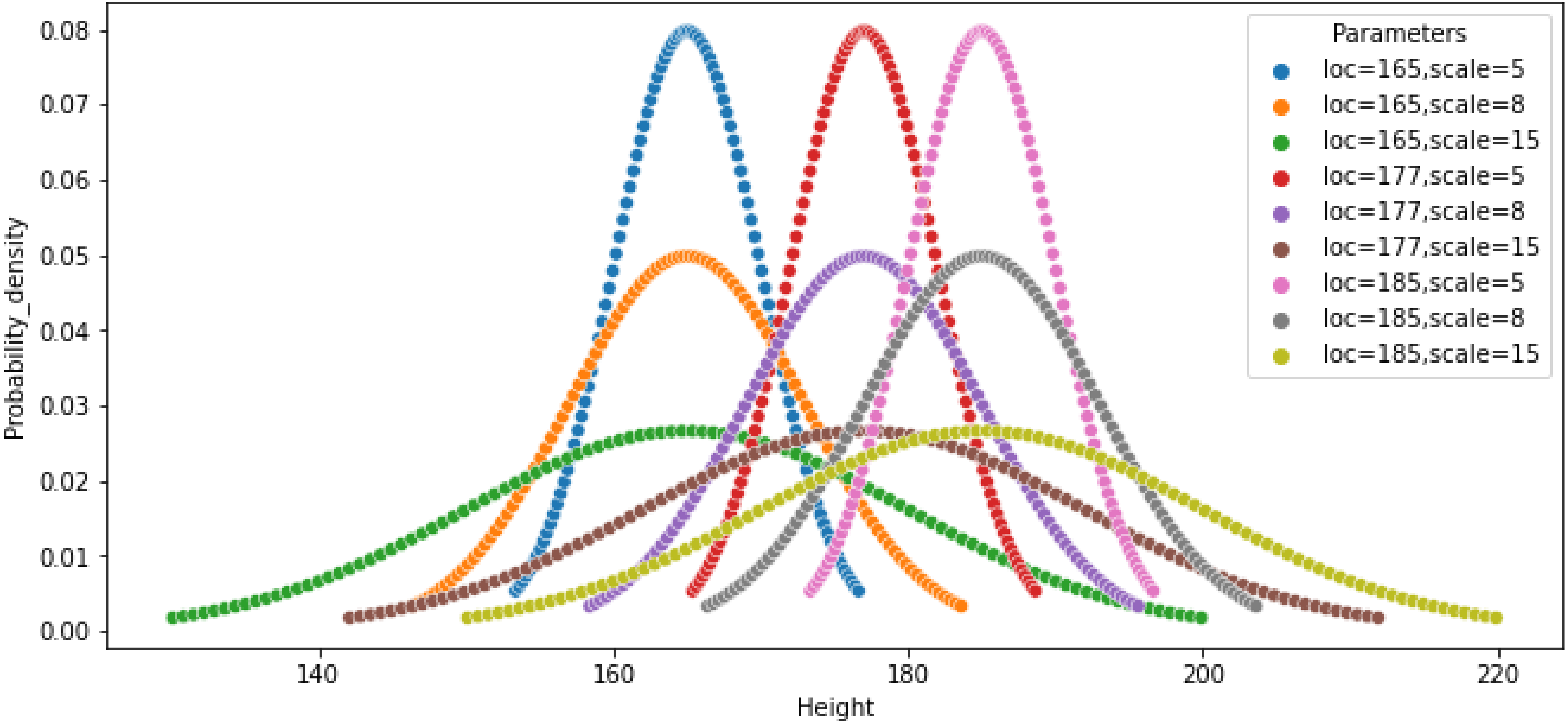
Changing the scale (standard deviation)



Changing the loc (mean)



Changing both scale and loc



Sampling from normal distributions

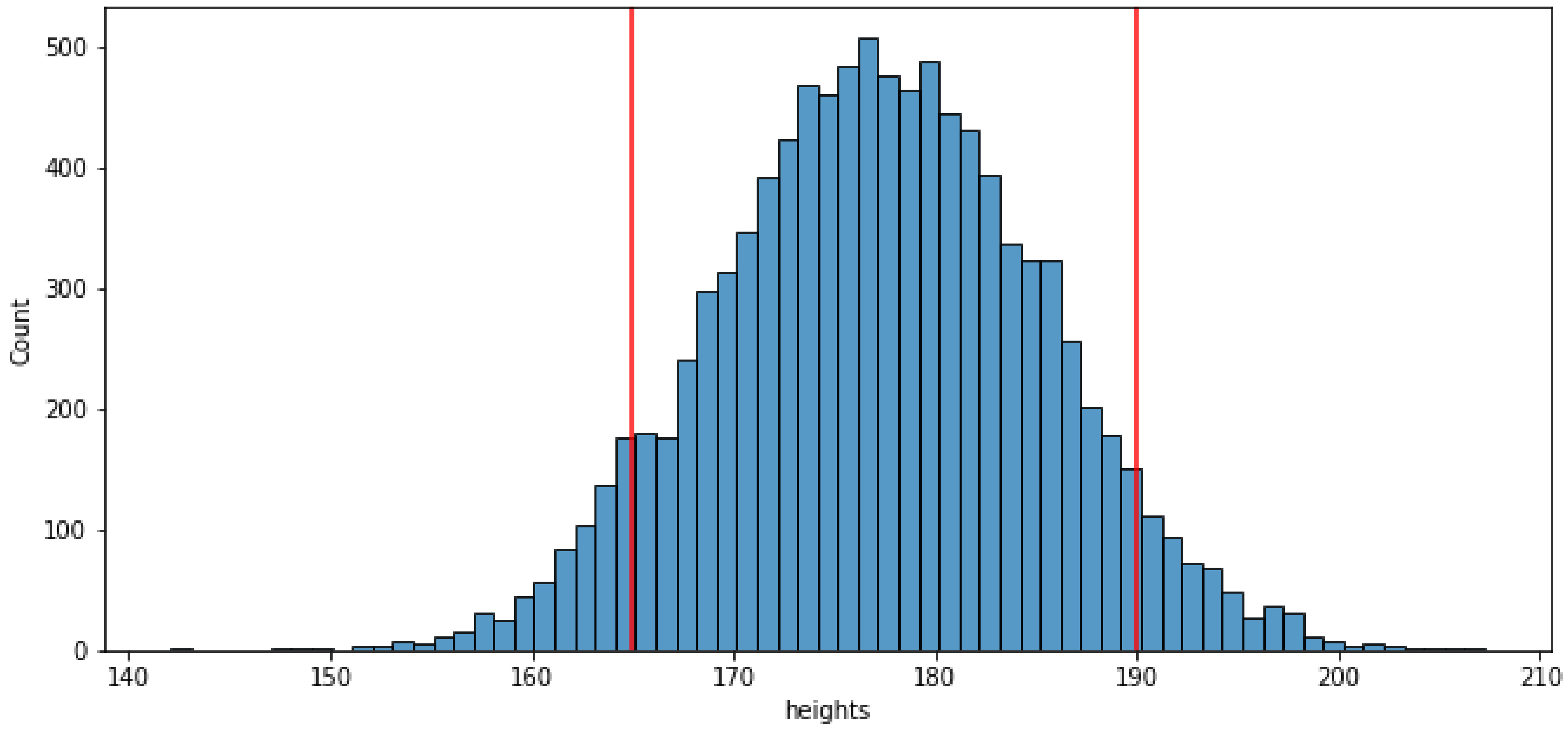
- Normally distributed US adult male heights; mean = 177 cm; standard deviation = 8 cm
- What's the percentage of people with a height either above 190 or below 165 cm?

```
heights = st.norm.rvs(loc=177, scale=8, size=10000)
qualified = (heights < 165) | (heights > 190)
print(np.sum(qualified) * 100/10000)
```

```
12.28
```

Plotting simulation results

```
heights_dict = {"heights":heights}
sns.histplot(x="heights", data=heights_dict)
plt.axvline(x=165, color="red")
plt.axvline(x=190, color="red")
```



More continuous probability distributions

- Continuous Uniform distribution (`st.uniform`)
  - The continuous analog of the discrete uniform distribution
- Exponential distribution (`st.expon`)
  - The continuous analog of the geometric distribution

[1] https://docs.scipy.org/doc/scipy/tutorial/stats/continuous.html

Let's practice!

MONTE CARLO SIMULATIONS IN PYTHON

Generating multivariate random variables

MONTE CARLO SIMULATIONS IN PYTHON

Izzy Weber
Curriculum Manager, DataCamp

Sampling from multivariate distributions

Multinomial distribution

- Variables each follow binomial distribution
- Probabilities of these variables sum to one

Example: simulating the results of flipping a biased coin

```
scipy.stats.multinomial.rvs()
```

Sampling from multivariate distributions

Multivariate normal distribution

- Variables each follow normal distribution
- Variables can be correlated with each other or not

Example: simulating price and demand

```
scipy.stats.multivariate_normal.rvs()
```

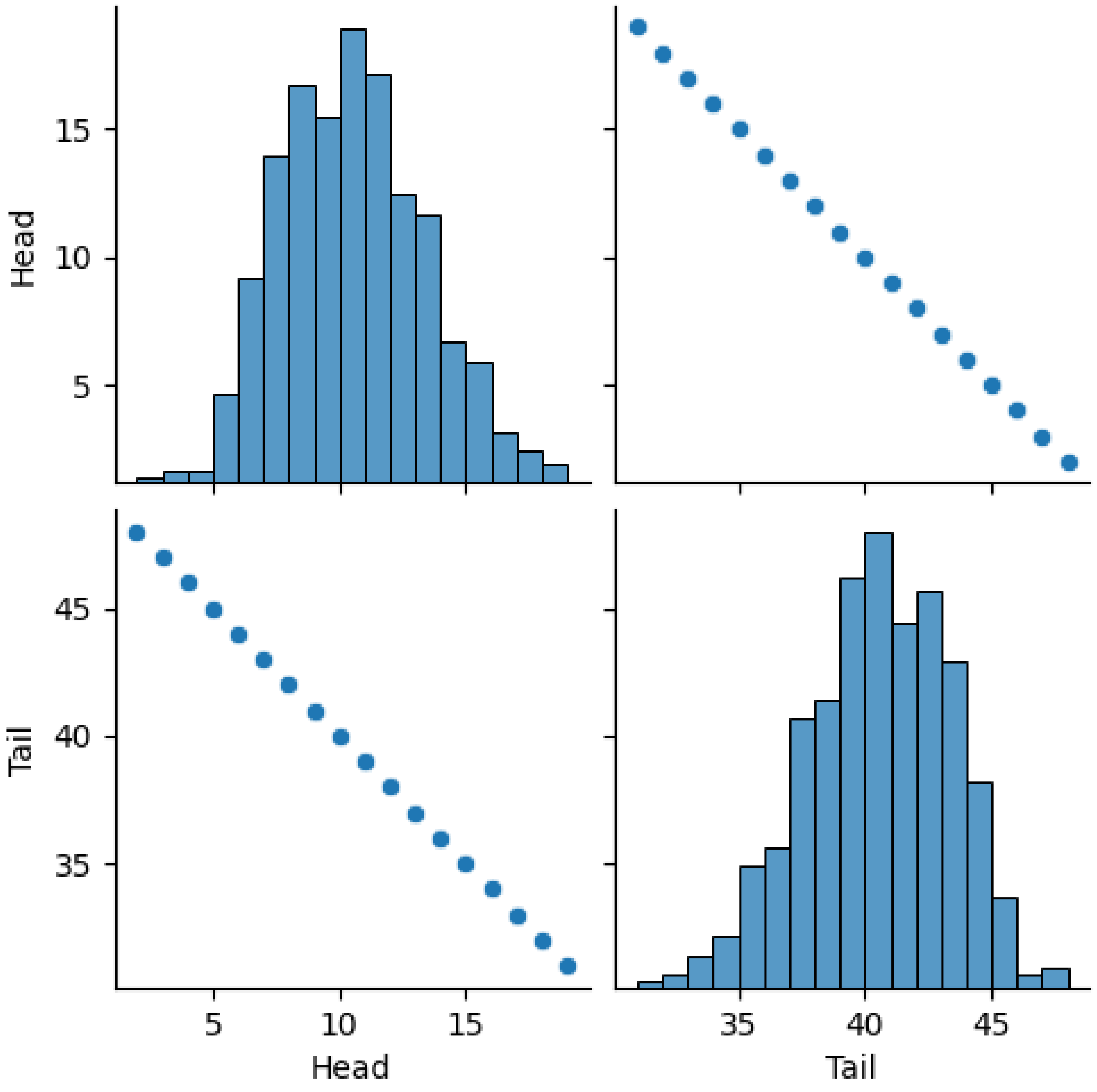
Sampling from multinomial distributions

Simulation: `.rvs(n, p, size)`

- n: `50`
- p: `[0.2, 0.8]`
- size: `500`

```
results = st.multinomial.rvs(50,
    [0.2, 0.8], size=500)

df_results=pd.DataFrame(
    {"Head":results[:, 0],
     "Tail":results[:, 1]})
sns.pairplot(df_results)
```



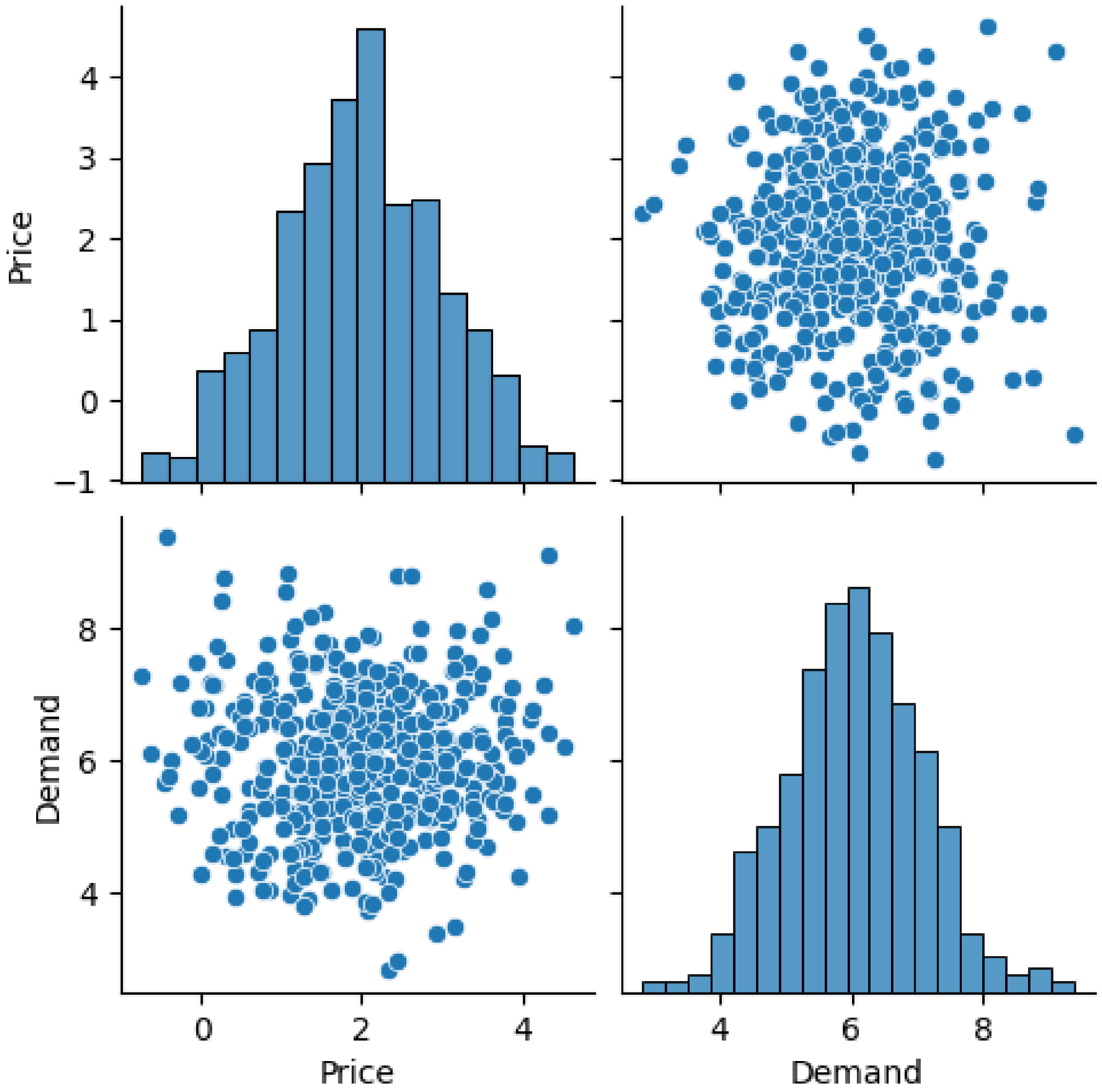
Sampling from multivariate normal distributions

Simulation: `.rvs(mean, size)`

- mean: `[2, 6]`
- size: `500`

```
results=st.multivariate_normal.rvs(
    mean=[2, 6], size=500)

df_results=pd.DataFrame(
    {"Price":results[:, 0],
     "Demand":results[:, 1]})
sns.pairplot(df_results)
```



Covariance matrix

- Captures the variance and covariances of variables
- Definition using two random variables x and y:

$$
\begin{array}{ccc}
 & x & y \\
x & \multicolumn{2}{c}{\multirow{2}{*}{$\begin{bmatrix} var(x) & cov(x,y) \\ cov(x,y) & var(y) \end{bmatrix}$}} \\
y & &
\end{array}
$$

Example:

```
df_historical.cov()
```

```
|        | Price     | Demand   |
|--------|-----------|----------|
| Price  | 0.920545  | -0.85578 |
| Demand | -0.855780 | 0.98417  |
```

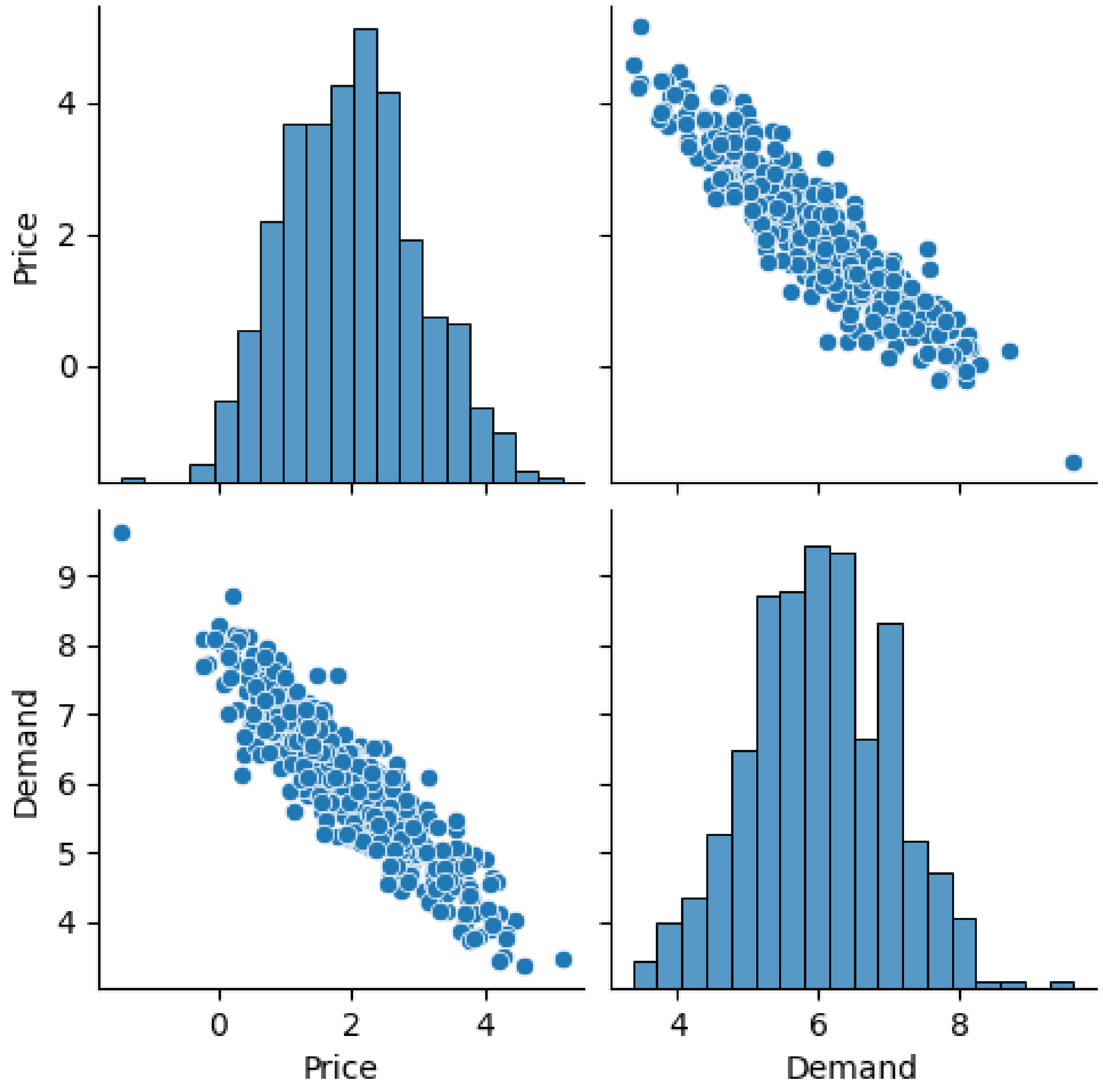
Multivariate normal sampling with defined covariance

Simulation: `.rvs(mean, size)`

- mean: `[2, 6]`
- size: `500`
- cov: `np.array([[1, -0.9], [-0.9, 1]])`

```
cov_mat = np.array([[1,-0.9], [-0.9,1]])
results = st.multivariate_normal.rvs(
    mean=[2,6], size=500, cov=cov_mat)

df_results = pd.DataFrame(
    {"Price":results[:,0],
     "Demand":results[:,1]})
sns.pairplot(df_results)
```



Let's practice!

MONTE CARLO SIMULATIONS IN PYTHON