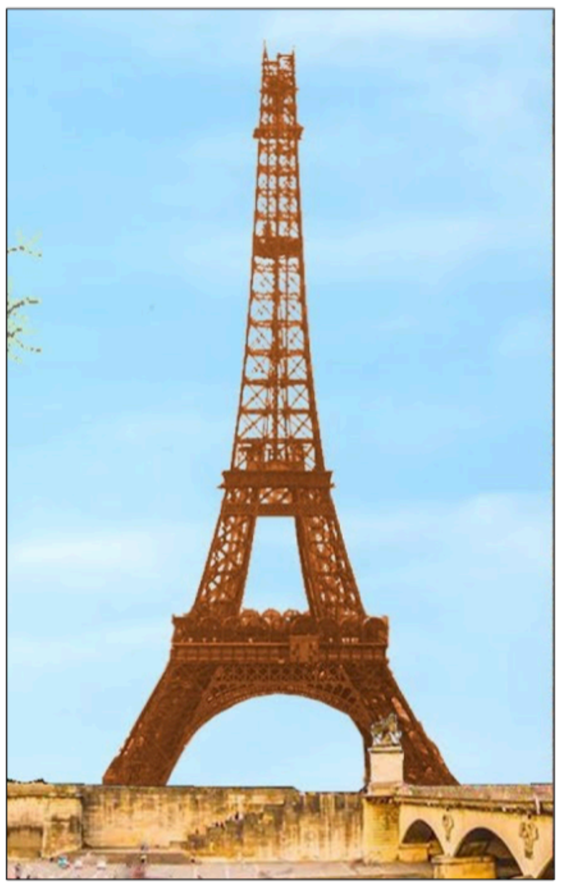
Random Sampling

Contents

Populations and Samples

- The art of data science

Population



Sample Data

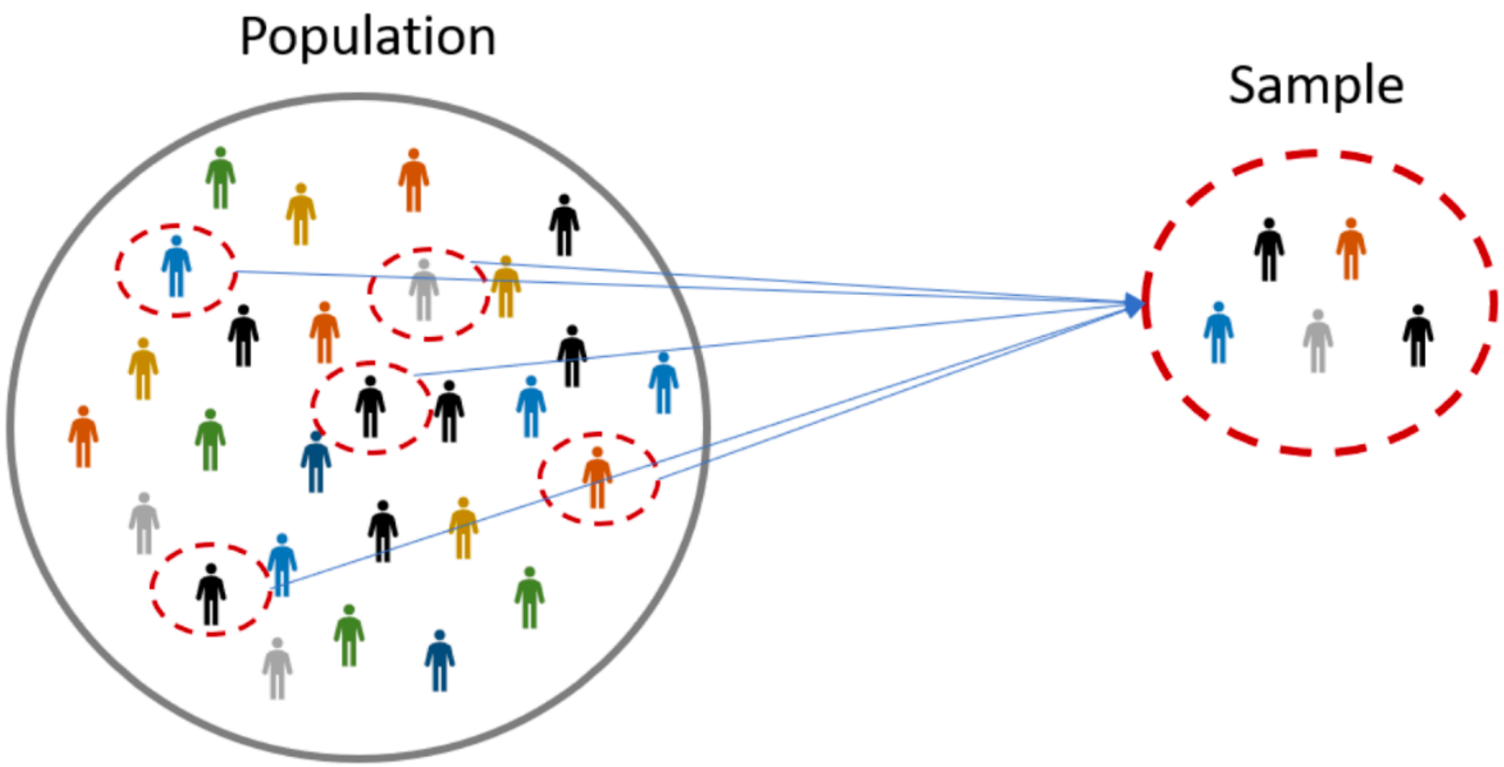
"Some say they see poetry in my paintings; I see only science."

–Georges Seurat

"La tour Eiffel" by Georges Seurat (1889)

Populations and Samples

- Population parameters and sample statistic

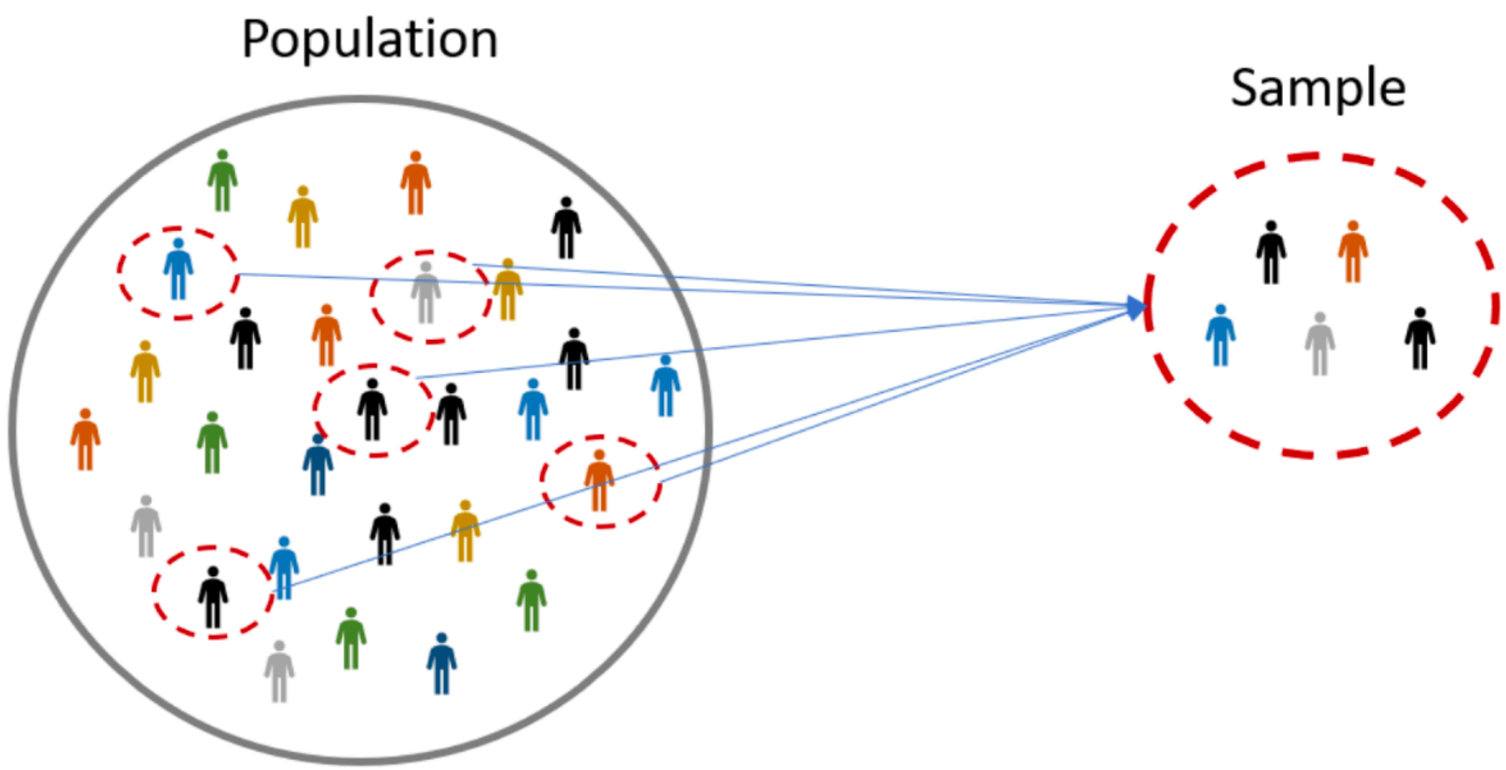


Notes:

- **Population:** the collection of all individuals or items under consideration in a statistical study.
- **Sample:** a part of the population from which information is obtained.

Populations and Samples

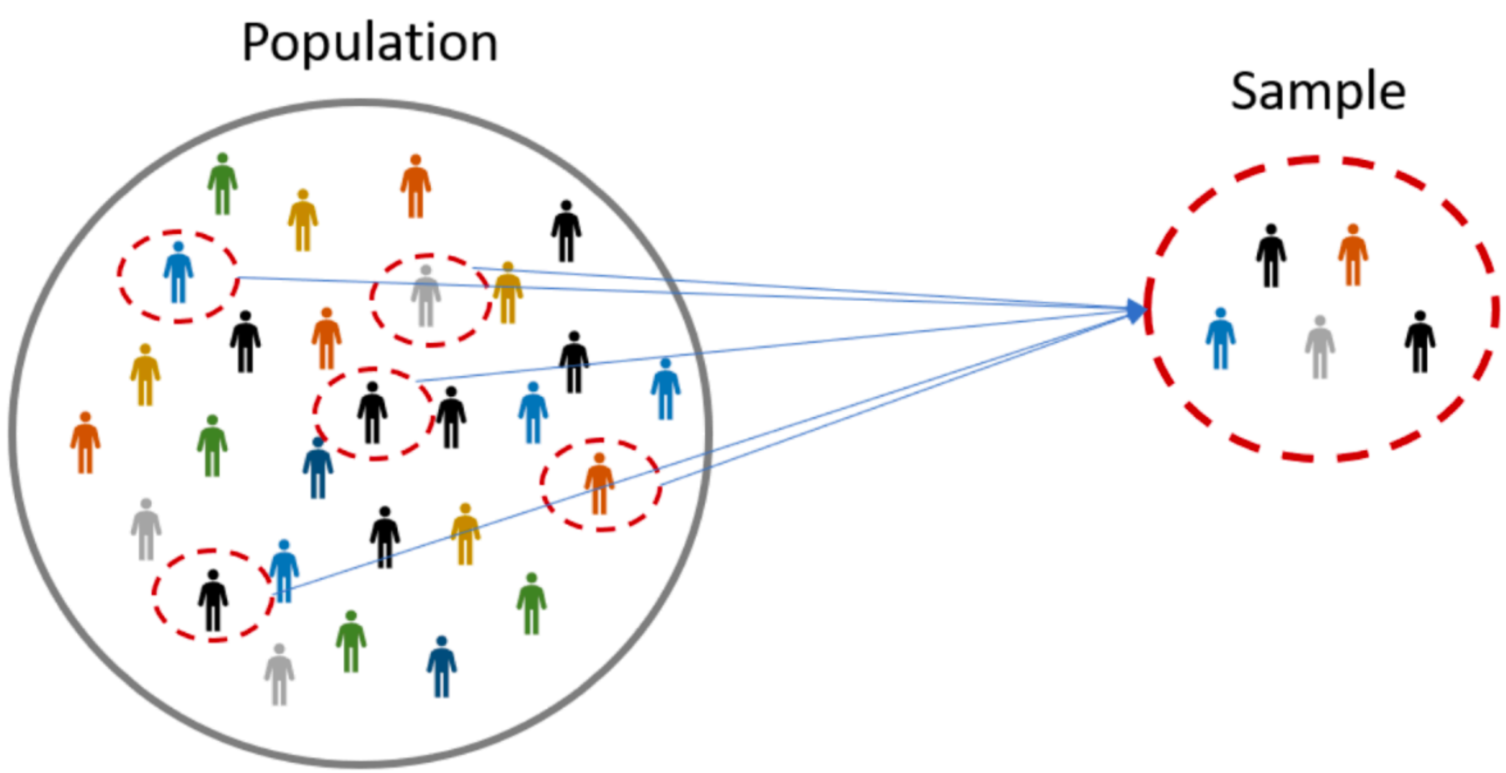
- Population parameters and sample statistic



- **Population**: all US voters in 2020
- **Parameter**: the proportion of all voters who support Trump
- **Sample**: a poll of 1000 voters
- **Statistic** the proportion of polled voters who support Trump

Populations and Samples

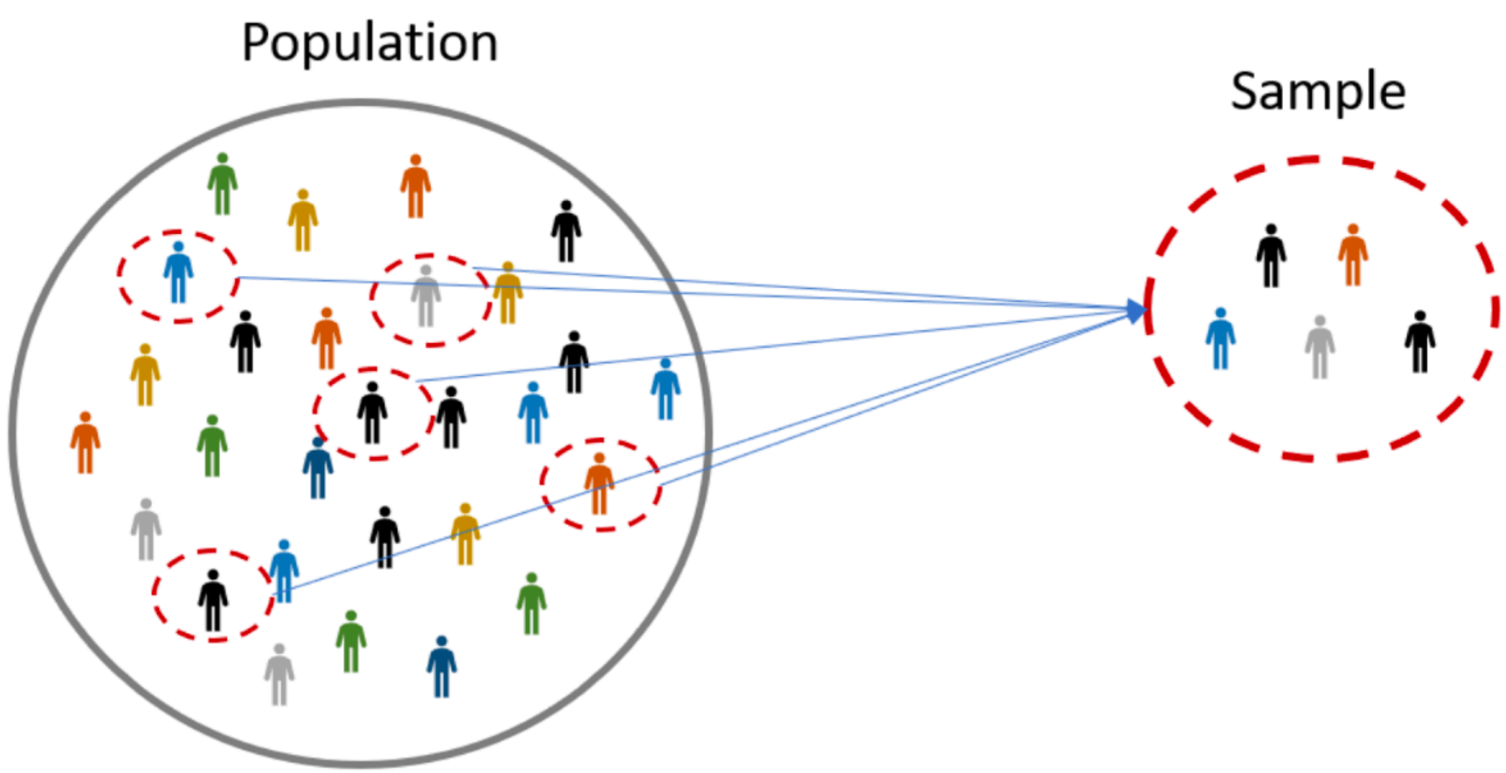
- Population parameters and sample statistic



- **Population**: all households in Singapore
- **Parameter**: the average income of all households in Singapore
- **Sample**: a sample of 100 randomly selected households in Singapore
- **Statistic** the average income of all selected households in Singapore

Populations and Samples

- Population parameters and sample statistic



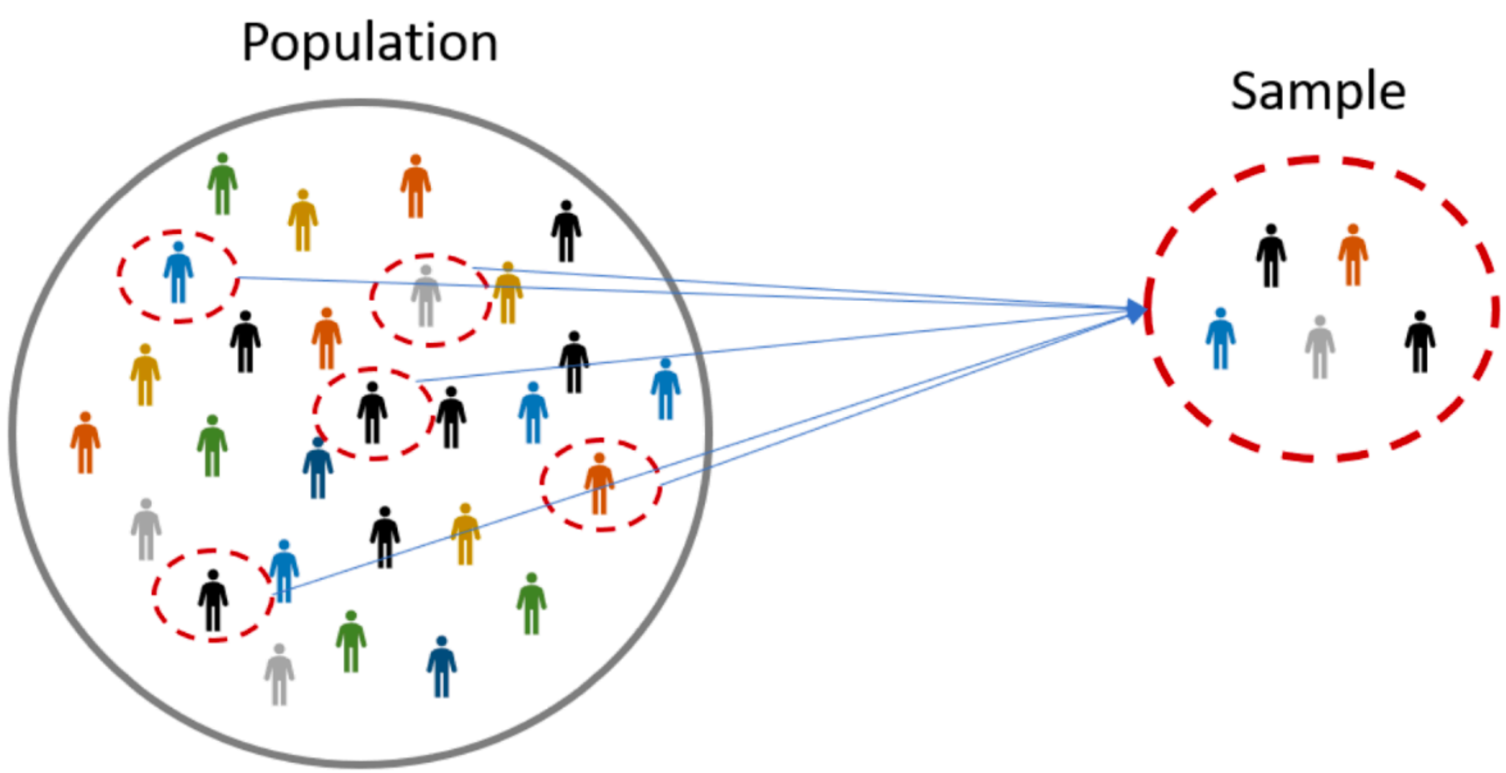
Mean value

$$\mu = \begin{cases} \sum_{i=1}^{k} x_i p_i \\ \int_{x \in \mathcal{X}} x f(x) dx \end{cases}$$

$$\bar{X} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

Populations and Samples

- Population parameters and sample statistic



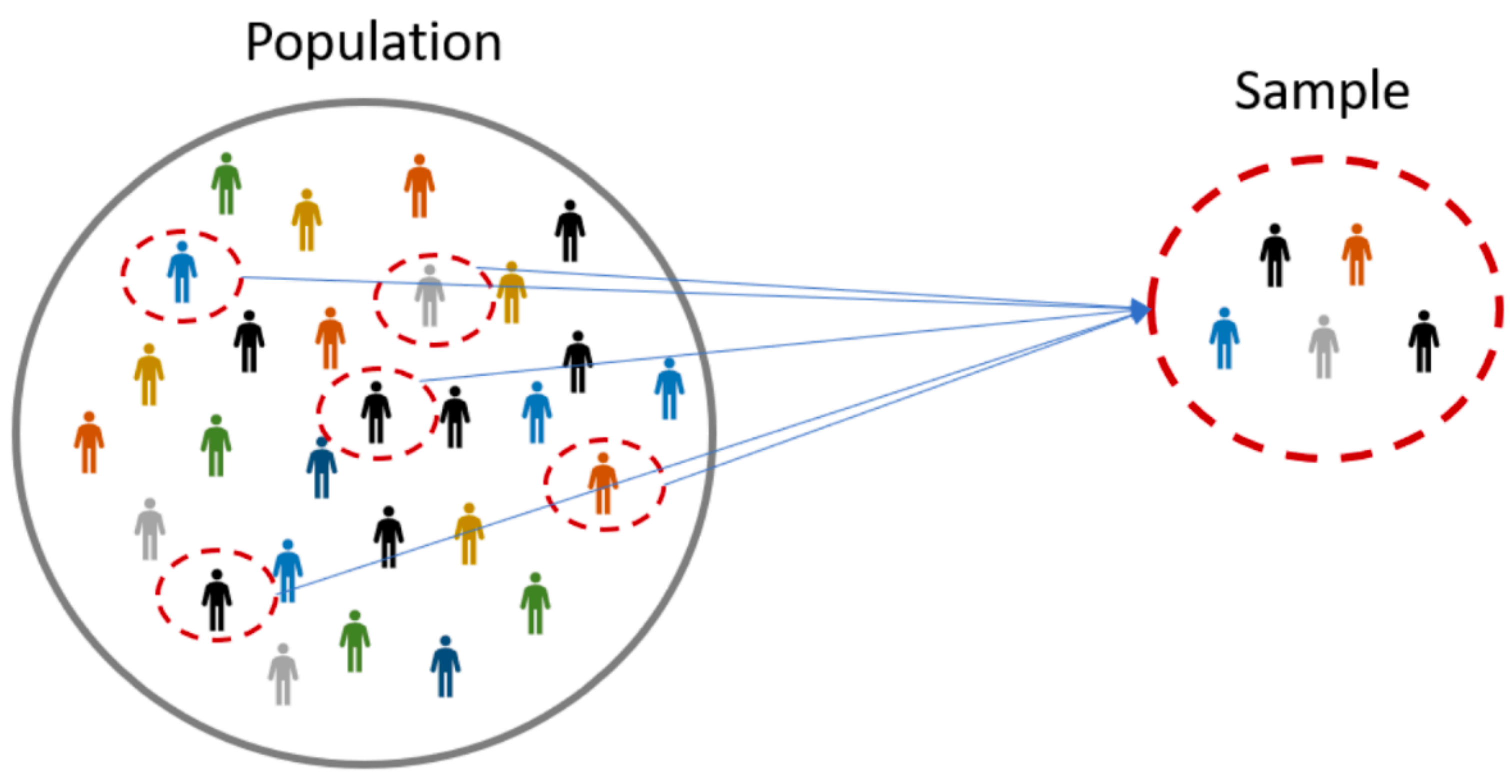
Variance

$$\sigma^2 = \begin{cases} \sum_{i=1}^{k} (x_i - \mathbb{E}(X))^2 p_i \\ \int_{x \in \mathcal{X}} (x - \mathbb{E}(X))^2 f(x) dx \end{cases}$$

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})^2$$

Populations and Samples

- Population parameters and sample statistic

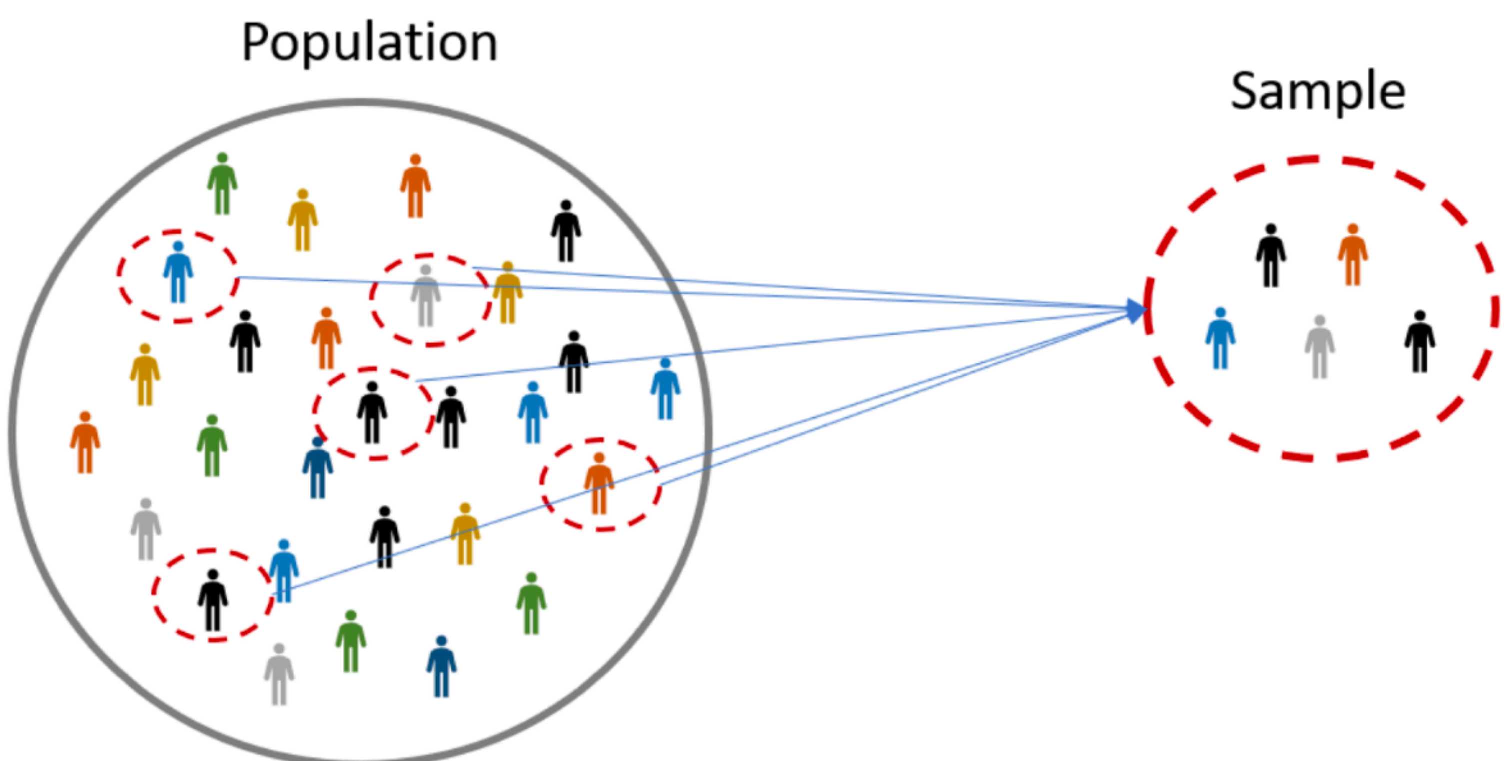


Probability $P(X \leq x) = F(x)$

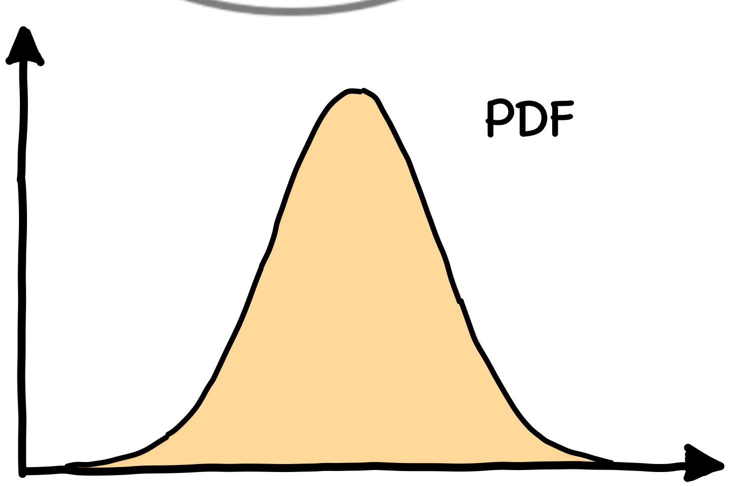
Proportion of $X_i \leq x$ in the sample

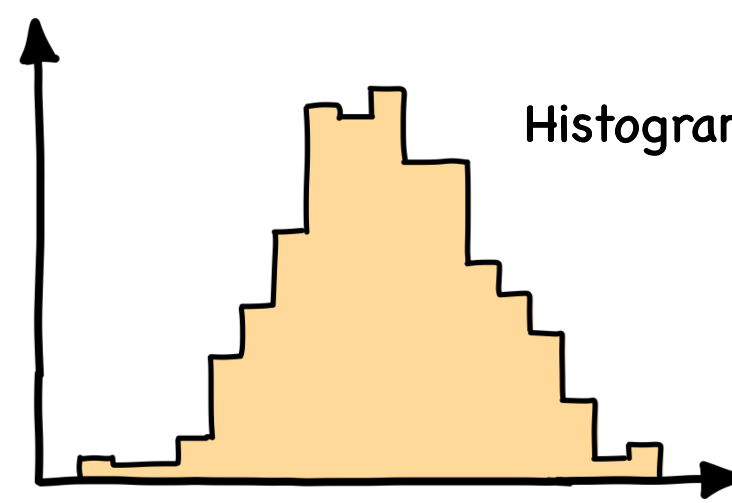
Populations and Samples

- Population parameters and sample statistic



Distribution

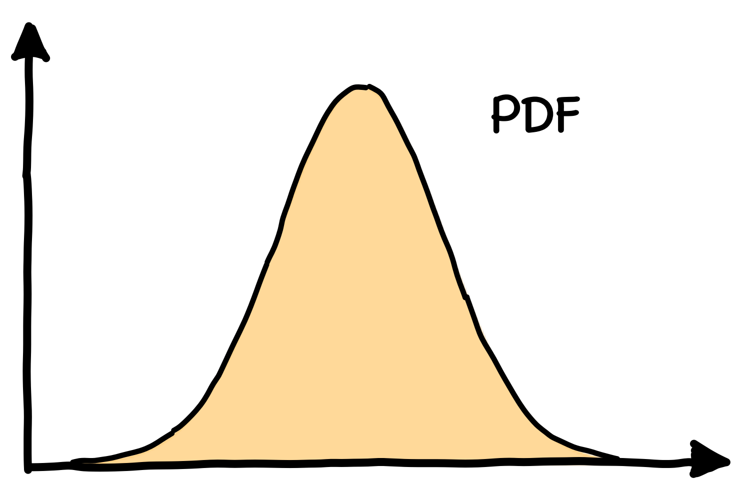


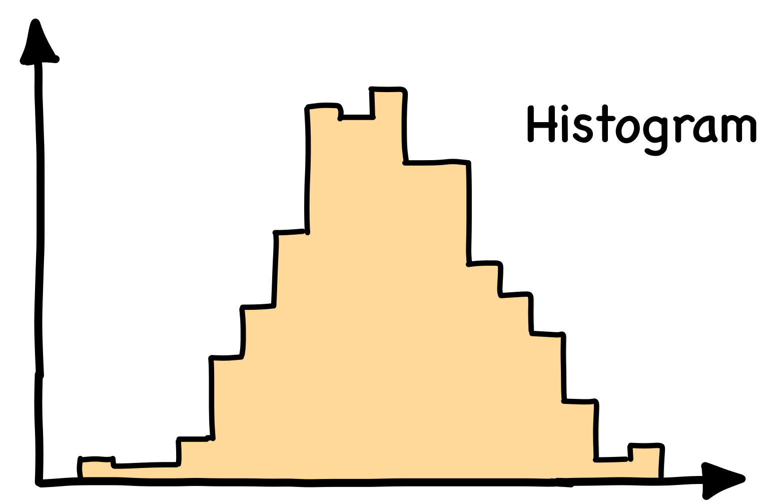


Populations and Samples

- Population parameters and sample statistic

	Population Parameters	Sample Statistics
mean value	population mean μ	sample average $\bar{X}$
standard deviation	population standard deviation σ	sample standard deviation s
Probability	population probability p	sample proportion $\hat{p}$
Distribution	PDF or PMF	histogram or density plot





Monte Carlo Simulations

- Random number generation
  - Syntax of calling random number generation functions

```
import numpy as np
```

```
np.random.<distr>(shape_param1, shape_param2, ..., size)
```

✓ <distr> is the name of the function representing a distribution

✓ shape_param1, shape_param2, … are the distribution parameters

✓ size is the shape of the returned array of random numbers.

Monte Carlo Simulations

- Random number generation

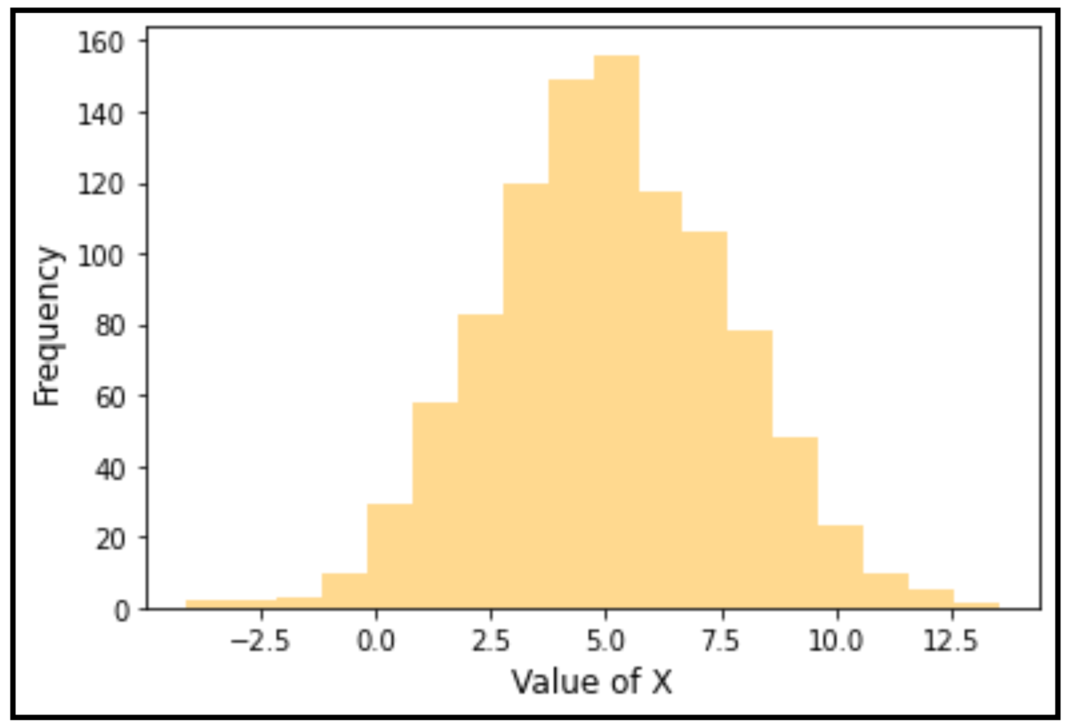
Distribution	Parameters	Random functions	Remarks
Binomial	`n` as a positive integer `p` as a probability	`binom`	-
Poisson	`lam` as the mean value	`poisson`	-
Uniform	`low` as the lower bound `high` as the upper bound	`uniform`	`low=0` and `high=1`, by default
Normal	`loc` as the mean value `scale` as the standard deviation	`normal`	`loc=0` and `scale=1`, by default

Monte Carlo Simulations

- Random number generation
  - An example of a sample following the normal distribution

```
sample = np.random.normal(5, 2.5, size=1000)
pd.Series(sample).describe()
```

```
count    1000.000000
mean        5.068111
std         2.577410
min        -4.079599
25%         3.390445
50%         4.972064
75%         6.841915
max        13.529048
dtype: float64
```

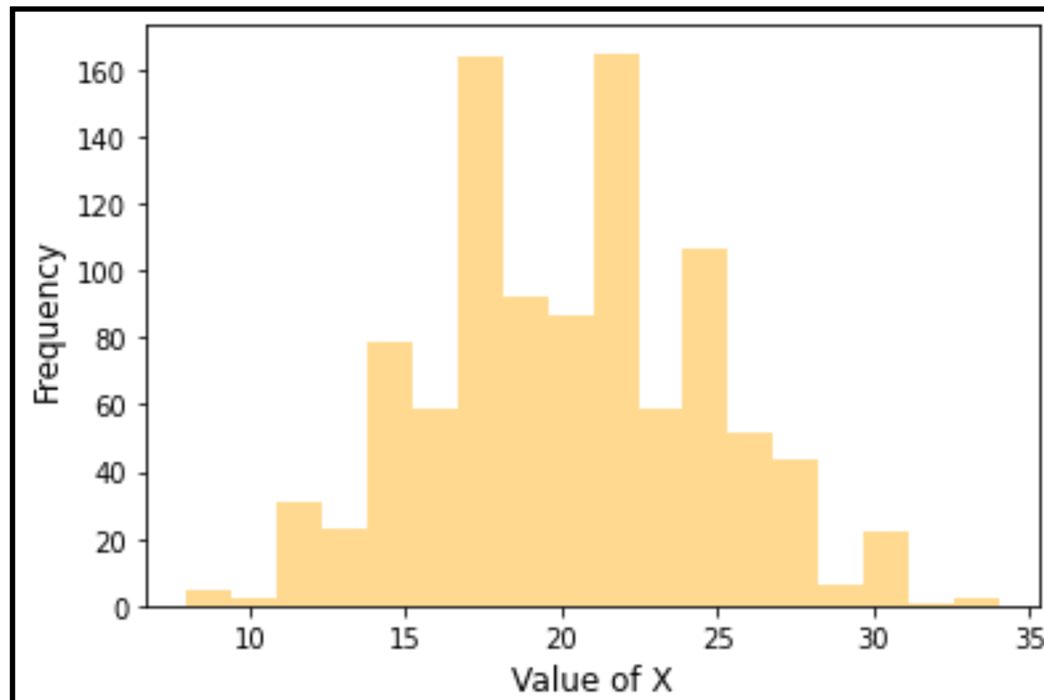


Monte Carlo Simulations

- Random number generation
  - An example of a sample following the Poisson distribution

```
sample = np.random.poisson(20, size=1000)
pd.Series(sample).describe()
```

```
count    1000.000000
mean       20.201000
std         4.405225
min         8.000000
25%        17.000000
50%        20.000000
75%        23.000000
max        34.000000
dtype: float64
```



Monte Carlo Simulations

- Simulations with continuous random variables

Example 1: Suppose that the daily log returns on a stock are independent and normally distributed with mean 0.001 and standard deviation 0.015. If you buy $\$1000$ worth of this stock at time $t = 1$, use Monte Carlo simulations to answer the following questions:

- What is the probability that after one trading day, i.e. at time $t = 2$, your investment is worth less than $\$990$?

$$r_2 = \log\left(\frac{Q_2}{Q_1}\right) \Rightarrow \exp(r_2) = \frac{Q_2}{Q_1} \Rightarrow Q_2 = Q_1 \exp(r_2)$$

Monte Carlo Simulations

- Simulations with continuous random variables

Example 1: Suppose that the daily log returns on a stock are independent and normally distributed with mean 0.001 and standard deviation 0.015. If you buy $\$1000$ worth of this stock at time $t = 1$, use Monte Carlo simulations to answer the following questions:

- What is the probability that after one trading day, i.e. at time $t = 2$, your investment is worth less than $\$990$?

```
mean = 0.001
std = 0.015
smp_size = 100000

log_returns = np.random.normal(mean, std, size=smp_size)
log_returns
```

```
array([ 0.00458223, -0.00576369, -0.00294265, ..., -0.0127849 ,
        0.03607525,  0.01488179])
```

Monte Carlo Simulations

- Simulations with continuous random variables

Example 1: Suppose that the daily log returns on a stock are independent and normally distributed with mean 0.001 and standard deviation 0.015. If you buy $\$1000$ worth of this stock at time $t = 1$, use Monte Carlo simulations to answer the following questions:

- What is the probability that after one trading day, i.e. at time $t = 2$, your investment is worth less than $\$990$?

```
q2 = 1000 * np.exp(log_returns)
prob = (q2 < 990).mean()

print(f'The probability is {prob}')
```

$$Q_2 = Q_1 \exp(r_2)$$

```
The probability is 0.23098
```

Monte Carlo Simulations

- Simulations with continuous random variables

Example 1: Suppose that the daily log returns on a stock are independent and normally distributed with mean 0.001 and standard deviation 0.015. If you buy $\$1000$ worth of this stock at time $t = 1$, use Monte Carlo simulations to answer the following questions:

- What is the probability that after one trading day, i.e. at time $t = 2$, your investment is worth less than $\$990$?

```
q2 = 1000 * np.exp(log_returns)
prob = (q2 < 990).mean()

print(f'The probability is {prob}')
```

$P(Q_2 \leq 990)$

```
The probability is 0.23098
```

Monte Carlo Simulations

- Simulations with continuous random variables

Example 1: Suppose that the daily log returns on a stock are independent and normally distributed with mean 0.001 and standard deviation 0.015. If you buy $\$1000$ worth of this stock at time $t = 1$, use Monte Carlo simulations to answer the following questions:

- What is the probability that after five trading day, i.e. at time $t = 6$, your investment is worth less than $\$990$?
- What is the expected value of the investment after five trading day, i.e. at time $t = 6$?
- What is the variance of the investment after five trading day, i.e. at time $t = 6$?

$$r_2 + r_3 + \ldots + r_6 = \log\left(\frac{Q_6}{Q_1}\right) \Rightarrow Q_6 = Q_1 \exp(r_2 + \ldots + r_6)$$

Monte Carlo Simulations

- Simulations with continuous random variables

```
days = 5
log_returns = np.random.normal(mean, std, size=(smp_size, days))
log_returns
```

```
array([[-0.01732922,  0.00666259, -0.01289302,  0.01409328, -0.01089381],
       [-0.00096519, -0.01771915, -0.0046359 , -0.00865865,  0.00638157],
       ...,
       [ 0.00894766, -0.0041353 ,  0.00994872,  0.01351102, -0.01767525]
       [-0.00416869, -0.00075058,  0.02998843, -0.00664735, -0.01876622]])
```

axis 0

axis 1

axis 0: each record in the sample

axis 1: log returns for each day

Monte Carlo Simulations

- Simulations with continuous random variables

```
days = 5
log_returns = np.random.normal(mean, std, size=(smp_size, days))
log_returns
```

```
array([[-0.01732922,  0.00666259, -0.01289302,  0.01409328, -0.01089381],
       [-0.00096519, -0.01771915, -0.0046359 , -0.00865865,  0.00638157],
       ...,
       [ 0.00894766, -0.0041353 ,  0.00994872,  0.01351102, -0.01767525]
       [-0.00416869, -0.00075058,  0.02998843, -0.00664735, -0.01876622]])
```

```
q6 = 1000 * np.exp(log_returns.sum(axis=1))
prob = (q6 < 990).mean()
print(f'The probability is {prob}')
print(f'The expected worth is {q6.mean()}')
print(f'The variance is {q6.var(ddof=1)}'
```

$$Q_6 = Q_1 \exp(r_2 + \ldots + r_6)$$

```
The probability is 0.32819
The expected worth is 1005.4708518041298
The variance is 1129.741563557124
```

Monte Carlo Simulations

- Simulations with continuous random variables

```
days = 5
log_returns = np.random.normal(mean, std, size=(smp_size, days))
log_returns
```

```
array([[-0.01732922,  0.00666259, -0.01289302,  0.01409328, -0.01089381],
       [-0.00096519, -0.01771915, -0.0046359 , -0.00865865,  0.00638157],
       ...,
       [ 0.00894766, -0.0041353 ,  0.00994872,  0.01351102, -0.01767525]
       [-0.00416869, -0.00075058,  0.02998843, -0.00664735, -0.01876622]])
```

```
q6 = 1000 * np.exp(log_returns.sum(axis=1))
prob = (q6 < 990).mean()
print(f'The probability is {prob}')
print(f'The expected worth is {q6.mean()}')
print(f'The variance is {q6.var(ddof=1)}'
```

$$P(Q_6 \leq 990)$$

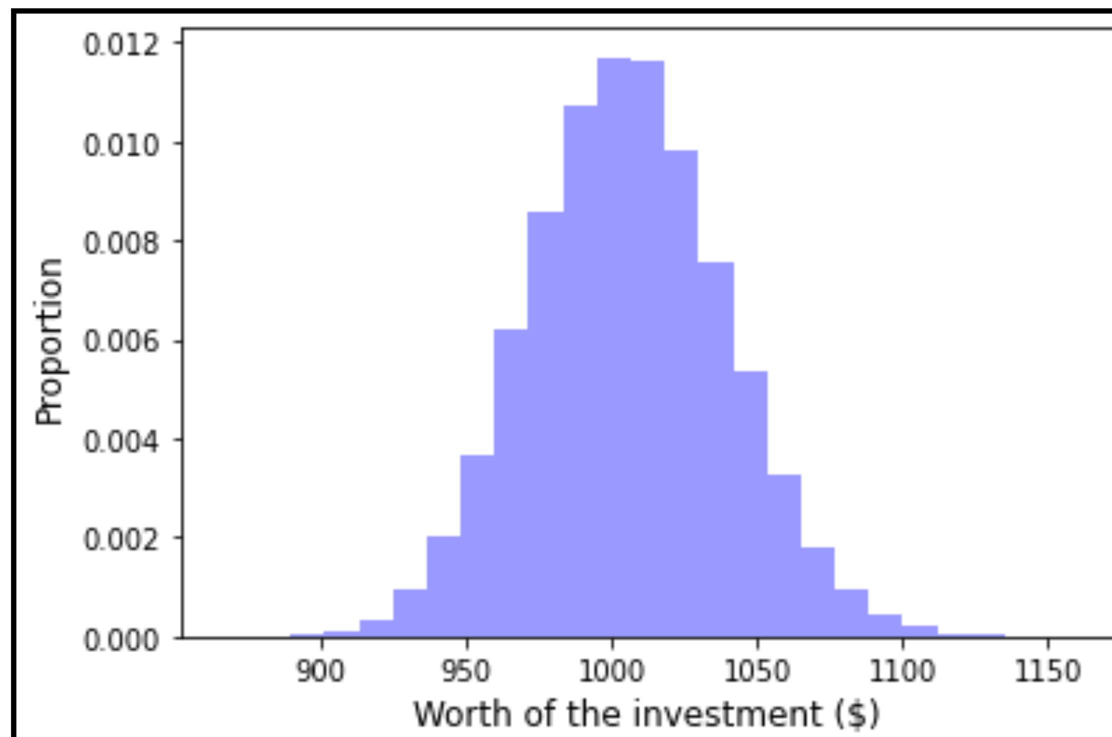
$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

```
The probability is 0.32819
The expected worth is 1005.4708518041298
The variance is 1129.741563557124
```

Monte Carlo Simulations

- Simulations with continuous random variables

```
plt.hist(q6, bins=25, density=True, color='b', alpha=0.4)
plt.xlabel('Worth of the investment ($)', fontsize=12)
plt.ylabel('Proportion', fontsize=12)
plt.show()
```



Monte Carlo Simulations

- Simulations with discrete random variables

Example 2: We have a few identical dice, and the probabilities of the rolled numbers of each die are provided in the series **`distr`**. Let X be a random variable representing the summation of numbers of five dice.

- What is the expected value of X?
- What is the variance of X
- What is the probability that X is larger than 20?

```
distr = pd.Series([0.15, 0.24, 0.18, 0.1, 0.21, 0.12],
                  index=range(1, 7))
print(distr)
```

```
1    0.15
2    0.24
3    0.18
4    0.10
5    0.21
6    0.12
dtype: float64
```

Monte Carlo Simulations

- Simulations with discrete random variables

Example 2: We have a few identical dice, and the probabilities of the rolled numbers of each die are provided in the series **`distr`**. Let X be a random variable representing the summation of numbers of five dice.

- What is the expected value of X?
- What is the variance of X
- What is the probability that X is larger than 20?

```
outcomes = np.arange(1, 7)
probs = distr.values
n = 5

exp_one = (outcomes * probs).sum()
print(f'The expected value is {exp_one * n}')
```

The expected value of rolling one die

```
The expected value is 16.7
```

Monte Carlo Simulations

- Simulations with discrete random variables

Example 2: We have a few identical dice, and the probabilities of the rolled numbers of each die are provided in the series **`distr`**. Let X be a random variable representing the summation of numbers of five dice.

- What is the expected value of X?
- What is the variance of X
- What is the probability that X is larger than 20?

```
var_one = ((outcomes - exp_one)**2 * probs).sum()
print(f'The variance is {var_one * n}')
```

```
The variance is 13.721999999999998
```

The variance of rolling one die

Monte Carlo Simulations

- Simulations with discrete random variables

```
rolls = np.random.choice(outcomes, p=probs, size=(smp_size, n))
rolls
```

Probabilities of all outcomes

Values of possible outcomes

```
array([[1, 6, 5, 1, 3],
       [2, 5, 3, 5, 2],
       ...,
       [6, 3, 5, 3, 1],
       [3, 5, 1, 3, 2]])
```

Monte Carlo Simulations

- Simulations with discrete random variables

```
rolls = np.random.choice(outcomes, p=probs, size=(smp_size, n))
rolls
```

```
array([[1, 6, 5, 1, 3],
       [2, 5, 3, 5, 2],
       ...,
       [6, 3, 5, 3, 1],
       [3, 5, 1, 3, 2]])
```

axis 0

axis 1

axis 0: each record in the sample

axis 1: rolled number of each die

Monte Carlo Simulations

- Simulations with discrete random variables

```
rolls = np.random.choice(outcomes, p=probs, size=(smp_size, n))
rolls
```

```
array([[1, 6, 5, 1, 3],
       [2, 5, 3, 5, 2],
       ...,
       [6, 3, 5, 3, 1],
       [3, 5, 1, 3, 2]])
```

```
xs = rolls.sum(axis=1)
print(f'The expected value from simulation is {xs.mean()}')
print(f'The variance from simulation is {xs.var(ddof=1)}')
print(f'P(X > 20) is {(xs > 20).mean()}')
```

```
The expected value from simulation is 16.6794
The variance from simulation is 13.693072570725706
P(X > 20) is 0.15521
```

Monte Carlo Simulations

- Simulations with discrete random variables

```
probs_sim = pd.Series(xs).value_counts(normalize=True)
probs_sim
```

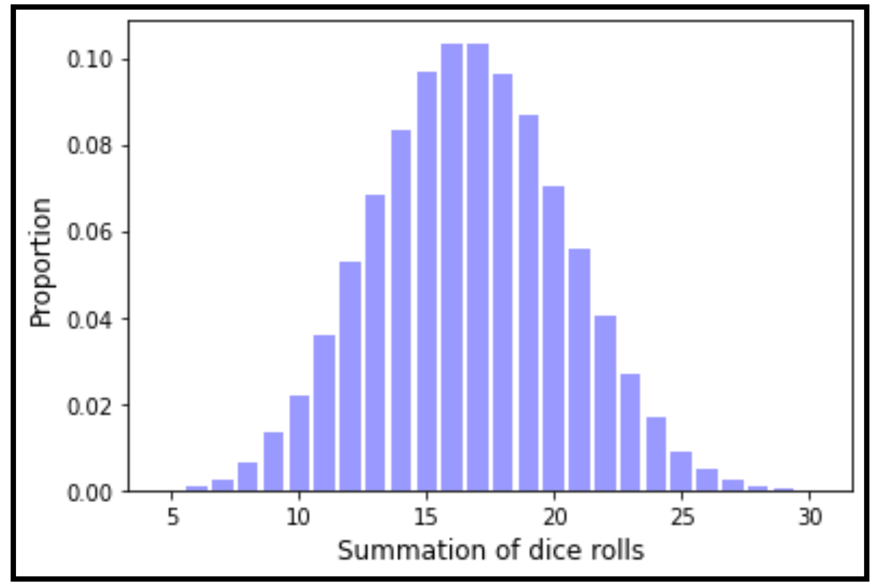
```
16    0.10341
17    0.10313
15    0.09664
...
29    0.00030
5     0.00010
30    0.00002
dtype: float64
```

Monte Carlo Simulations

- Simulations with discrete random variables

```
probs_sim = pd.Series(xs).value_counts(normalize=True)
probs_sim
```

```
16     0.10341
17     0.10313
15     0.09664
...
29     0.00030
5      0.00010
30     0.00002
dtype: float64
```




```
plt.bar(probs_sim.index, probs_sim, color='b', alpha=0.4)
plt.xlabel('Summation of dice rolls', fontsize=12)
plt.ylabel('Proportion', fontsize=12)
plt.show()
```

Monte Carlo Simulations

- Simulations with discrete random variables

Example 2: We have a few identical dice, and the probabilities of the rolled numbers of each die are provided in the series **`distr`**. Let X be a random variable representing the summation of five dice rolls.

- What is the expected difference between the maximum and the minimum rolled numbers of the five dice?

```
diff = rolls.max(axis=1) - rolls.min(axis=1)

print(f'The expected difference is {diff.mean()}')
```

```
The expected difference is 3.72109
```

axis 1

rolls

```
array([[1, 6, 5, 1, 3],
       [2, 5, 3, 5, 2],
       ...,
       [6, 3, 5, 3, 1],
       [3, 5, 1, 3, 2]])
```

Monte Carlo Simulations

- Simulations with discrete random variables

```
probs_diff = pd.Series(diff).value_counts(normalize=True)
probs_diff
```

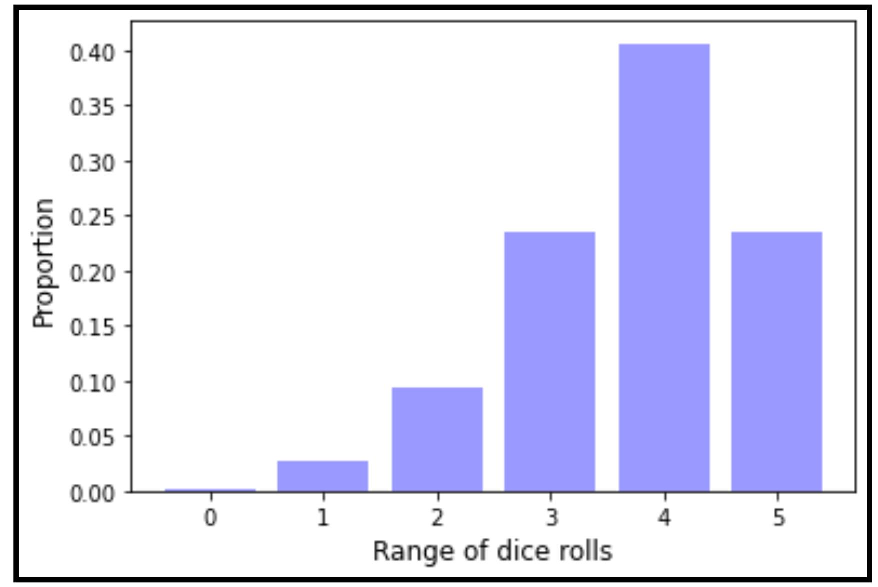
```
4    0.40596
3    0.23571
5    0.23486
2    0.09398
1    0.02786
0    0.00163
dtype: float64
```

Monte Carlo Simulations

- Simulations with discrete random variables

```
probs_diff = pd.Series(diff).value_counts(normalize=True)
probs_diff
```

```
4     0.40596
3     0.23571
5     0.23486
2     0.09398
1     0.02786
0     0.00163
dtype: float64
```




```
plt.bar(probs_diff.index, probs_diff, color='b', alpha=0.4)
plt.xlabel('Range of dice rolls', fontsize=12)
plt.ylabel('Proportion', fontsize=12)
plt.show()
```

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

Example 3: A bakery is making croissants every day. The cost of every croissant is $\$0.6$ and is sold at a price of $\$2.5$. Suppose that the daily demand of croissants follows a Poisson distribution with a mean value $\mu = 38.6$, and any unsold croissants will be disposed by the end of the day. What is the optimal number of croissants baked every day, so that the profit of the bakery is maximized?

```
cost = 0.6
price = 2.5
mu = 38.6

smp_size = 100000
demand = np.random.poisson(mu, size=smp_size)
```

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
profits = []
decisions = np.arange(20, 61)
for decision in decisions:
    sold = np.minimum(demand, decision)
    exp_profit = (price*sold - cost*decision).mean()
    profits.append(exp_profit)

np.array(profits)
```

The sold quantity is no larger than the demand and the number of baked croissants

```
array([37.997725, 39.89555 , 41.792   , 43.685875, 45.5744  , 47.454875,
       49.32315 , 51.17245 , 52.9948  , 54.7809  , 56.518025, 58.187975,
       59.77935 , 61.272575, 62.651275, 63.90205 , 65.010075, 65.9664  ,
       66.765125, 67.40315 , 67.884025, 68.2114  , 68.394025, 68.444825,
       68.374325, 68.19775 , 67.93295 , 67.5922  , 67.189775, 66.738625,
       66.247325, 65.725125, 65.1791  , 64.617425, 64.044425, 63.463075,
       62.875325, 62.283225, 61.6885  , 61.091825, 60.493875])
```

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
profits = []
decisions = np.arange(20, 61)
for decision in decisions:
    sold = np.minimum(demand, decision)
    exp_profit = (price*sold - cost*decision).mean()
    profits.append(exp_profit)

np.array(profits)
```

Vectorized operation for calculating the profits

```
array([37.997725, 39.89555 , 41.792   , 43.685875, 45.5744  , 47.454875,
       49.32315 , 51.17245 , 52.9948  , 54.7809  , 56.518025, 58.187975,
       59.77935 , 61.272575, 62.651275, 63.90205 , 65.010075, 65.9664  ,
       66.765125, 67.40315 , 67.884025, 68.2114  , 68.394025, 68.444825,
       68.374325, 68.19775 , 67.93295 , 67.5922  , 67.189775, 66.738625,
       66.247325, 65.725125, 65.1791  , 64.617425, 64.044425, 63.463075,
       62.875325, 62.283225, 61.6885  , 61.091825, 60.493875])
```

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
profits = []
decisions = np.arange(20, 61)
for decision in decisions:
    sold = np.minimum(demand, decision)
    exp_profit = (price*sold - cost*decision).mean()
    profits.append(exp_profit)

np.array(profits)
```

Append the expected profit to a list

```
array([37.997725, 39.89555 , 41.792    , 43.685875, 45.5744   , 47.454875,
       49.32315 , 51.17245 , 52.9948   , 54.7809   , 56.518025, 58.187975,
       59.77935 , 61.272575, 62.651275, 63.90205 , 65.010075, 65.9664   ,
       66.765125, 67.40315 , 67.884025, 68.2114   , 68.394025, 68.444825,
       68.374325, 68.19775 , 67.93295 , 67.5922   , 67.189775, 66.738625,
       66.247325, 65.725125, 65.1791   , 64.617425, 64.044425, 63.463075,
       62.875325, 62.283225, 61.6885   , 61.091825, 60.493875])
```

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```python
solds = np.minimum(demand.reshape((smp_size, 1)), decisions)
solds
```

48
33
...
37
44

20	21	...	59	60

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
solds = np.minimum(demand.reshape((smp_size, 1)), decisions)
solds
```

Broadcasting

48	48	...	48	48
33	33	...	33	33
...	...	...	...	...
37	37	...	37	37
44	44	...	44	44

Broadcasting

20	21	...	59	60
20	21	...	59	60
...	...	...	...	...
20	21	...	59	60
20	21	...	59	60

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
solds = np.minimum(demand.reshape((smp_size, 1)), decisions)
solds
```

```
array([[20, 21, 22, ..., 48, 48, 48],
       [20, 21, 22, ..., 33, 33, 33],
       ...,
       [20, 21, 22, ..., 37, 37, 37],
       [20, 21, 22, ..., 44, 44, 44]])
```

axis 0

axis 1

axis 0: each record in the sample

axis 1: each candidate decision

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
solds = np.minimum(demand.reshape((smp_size, 1)), decisions)
solds
```

```
array([[20, 21, 22, ..., 48, 48, 48],
       [20, 21, 22, ..., 33, 33, 33],
       ...,
       [20, 21, 22, ..., 37, 37, 37],
       [20, 21, 22, ..., 44, 44, 44]])
```

```
profits = (price*solds - cost*decisions).mean(axis=0)
profits
```

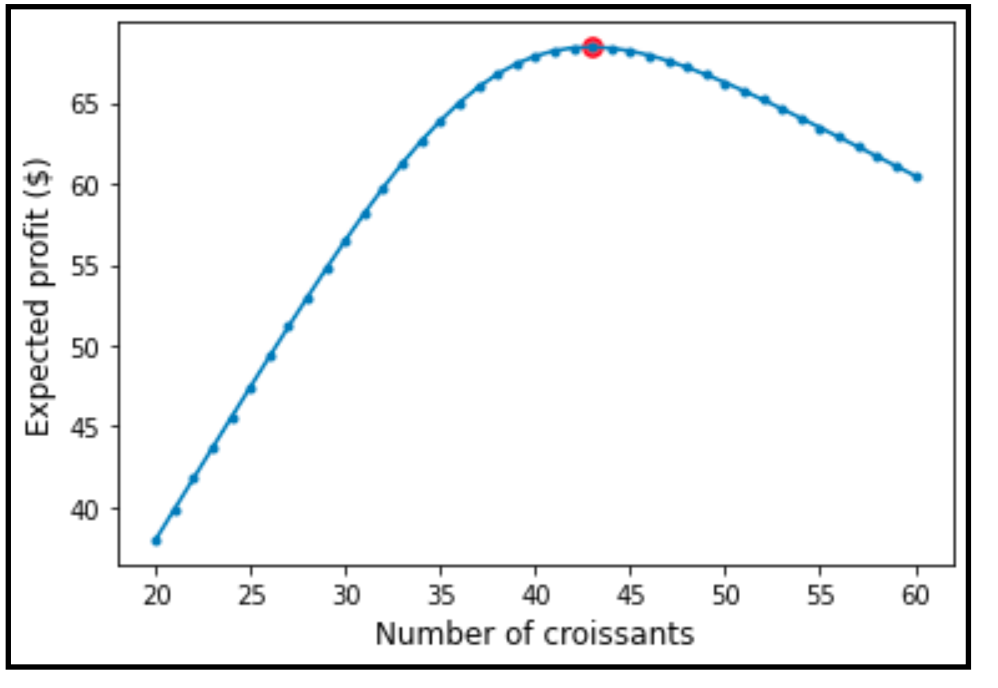
```
array([37.997725, 39.89555 , 41.792   , 43.685875, 45.5744  , 47.454875,
       49.32315 , 51.17245 , 52.9948  , 54.7809  , 56.518025, 58.187975,
       59.77935 , 61.272575, 62.651275, 63.90205 , 65.010075, 65.9664  ,
       66.765125, 67.40315 , 67.884025, 68.2114  , 68.394025, 68.444825,
       68.374325, 68.19775 , 67.93295 , 67.5922  , 67.189775, 66.738625,
       66.247325, 65.725125, 65.1791  , 64.617425, 64.044425, 63.463075,
       62.875325, 62.283225, 61.6885  , 61.091825, 60.493875])
```

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
index_max = np.argmax(profits)
optimal = decisions[index_max]

print(f'The optimal number is {optimal}')
print(f'The maximum profit is {np.max(profits)}')
```

```
The optimal number is 43
The maximum profit is 68.44625000011904
```



Monte Carlo Simulations

- Monte Carlo simulations for decision-making

Question 1: The bakery in **Example 3** is planning to sell croissants at a discount price of $\$1.5$ in the last two hours each day. Suppose that the demand before the last two hours follows a Poisson distribution with a mean value $\mu_1 = 32.6$, and the demand during the last three hours follows a Poisson distribution with a mean value $\mu_2 = 16.5$. What is the optimal number of croissants baked every day, so that the profit of the bakery is maximized?

```
cost = 0.6
price1 = 2.5
price2 = 1.5
mu1 = 32.6
mu2 = 16.5

smp_size = 100000
demand1 = np.random.poisson(mu1, size=smp_size)
demand2 = np.random.poisson(mu2, size=smp_size)
```

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
decisions = np.arange(20, 61)

solds1 = np.minimum(demand1.reshape((smp_size, 1)), decisions)
left = decisions - solds1
left
```

20	21	...	59	60

20	21	...	30	30
20	21	...	34	34
...	...	...	...	...
20	21	...	28	28
20	21	...	31	31

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
decisions = np.arange(20, 61)

solds1 = np.minimum(demand1.reshape((smp_size, 1)), decisions)
left = decisions - solds1
left
```

Broadcasting

20	21	...	59	60
20	21	...	59	60
...	...	...	...	...
20	21	...	59	60
20	21	...	59	60

20	21	...	30	30
20	21	...	34	34
...	...	...	...	...
20	21	...	28	28
20	21	...	31	31

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
decisions = np.arange(20, 61)

solds1 = np.minimum(demand1.reshape((smp_size, 1)), decisions)
left = decisions - solds1
left
```

```
array([[ 0,  0,  0, ..., 28, 29, 30],
       [ 0,  0,  0, ..., 24, 25, 26],
       ...,
       [ 0,  0,  0, ..., 30, 31, 32],
       [ 0,  0,  0, ..., 27, 28, 29]])
```

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
solds2 = np.minimum(demand2.reshape((smp_size, 1)), left)
solds2
```

19
19
...
19
24

20	21	...	30	30
20	21	...	34	34
...	...	...	...	...
20	21	...	28	28
20	21	...	31	31

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
solds2 = np.minimum(demand2.reshape((smp_size, 1)), left)
solds2
```

Broadcasting

19	19	...	19	19
19	19	...	19	19
...	...	...	...	...
19	19	...	19	19
24	24	...	24	24

20	21	...	30	30
20	21	...	34	34
...	...	...	...	...
20	21	...	28	28
20	21	...	31	31

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
solds2 = np.minimum(demand2.reshape((smp_size, 1)), left)
solds2
```

```
array([[ 0,  0,  0, ..., 19, 19, 19],
       [ 0,  0,  0, ..., 19, 19, 19],
       ...,
       [ 0,  0,  0, ..., 19, 19, 19],
       [ 0,  0,  0, ..., 24, 24, 24]])
```

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
revenue = price1*solds1 + price2*solds2
new_profits = (revenue - cost*decisions).mean(axis=0)
new_profits
```

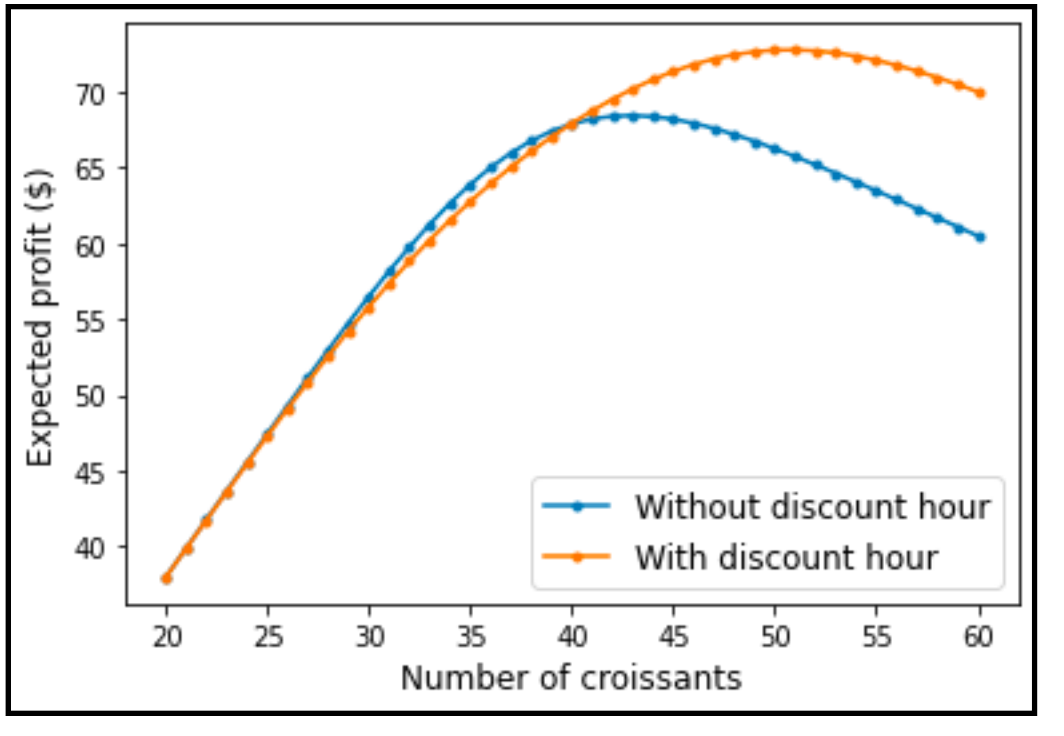
```
array([37.985625, 39.87354 , 41.75341 , 43.62136 , 45.472875, 47.300945,
       49.099095, 50.8592  , 52.573645, 54.233395, 55.83294 , 57.365515,
       58.827545, 60.216125, 61.52967 , 62.76949 , 63.93703 , 65.033985,
       66.06191 , 67.02365 , 67.91905 , 68.748475, 69.50833 , 70.19782 ,
       70.811925, 71.348605, 71.80355 , 72.17303 , 72.45561 , 72.64822 ,
       72.7544  , 72.77341 , 72.710555, 72.57173 , 72.36188 , 72.089325,
       71.75988 , 71.37913 , 70.95489 , 70.494775, 70.00417 ])
```

Monte Carlo Simulations

- Monte Carlo simulations for decision-making

```
index_max = np.argmax(new_profits)
optimal = decisions[index_max]

print(f'The optimal number is {optimal}')
print(f'The maximum profit is {new_profits.max()}')
```

```
The optimal number is 51
The maximum profit is 72.77341000012011
```



Sampling Distributions

- Estimate the population mean using the sample average

Example 4: The lifespans of all bulbs in a batch are recorded in a file called "bulb.csv". Consider this batch bulbs as the population, estimate the population mean using a randomly selected a sample with $n = 25$ observations. Repeat the sampling experiment 1000 times to find the mean and standard deviation of the sample means.

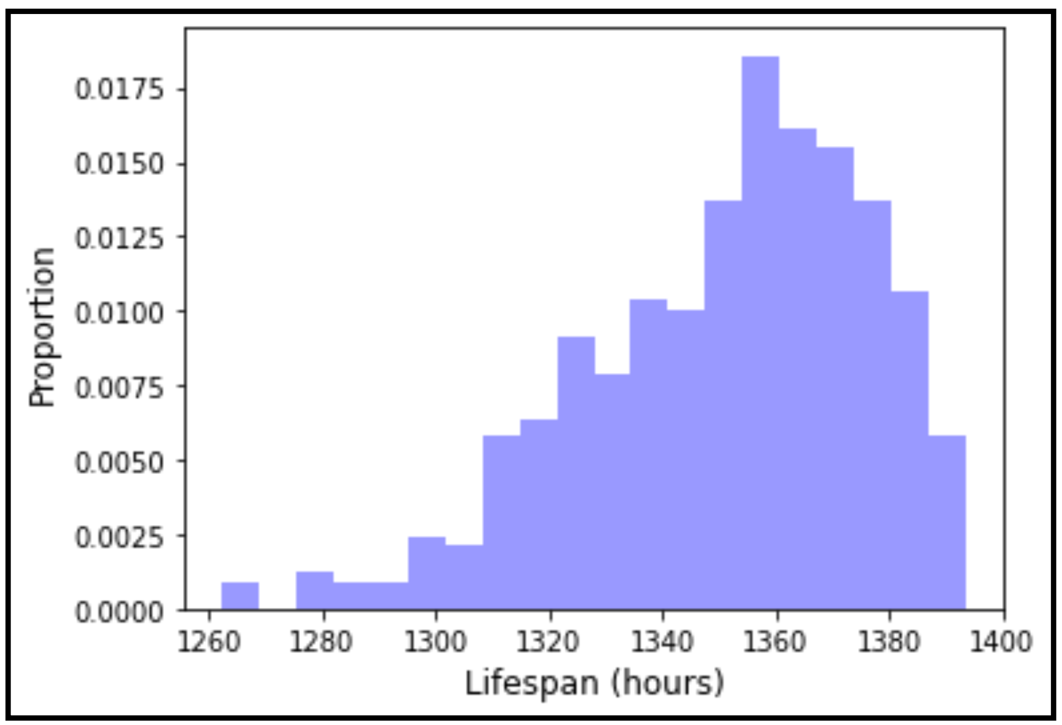
```
bulb = pd.read_csv('bulb.csv')
print(f'Population mean: {bulb.values.mean()}')
print(f'Population standard deviation: {bulb.values.std()}')
```

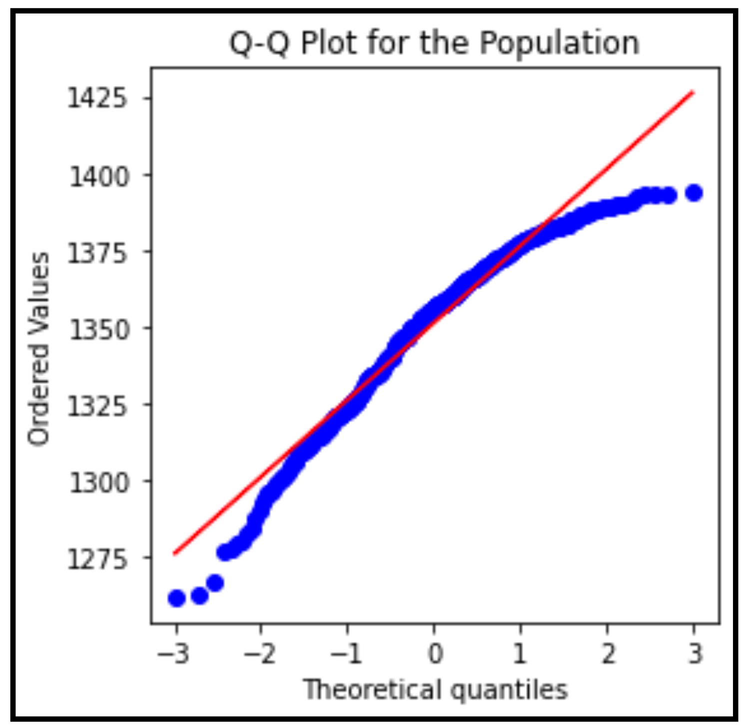
```
Population mean: 1351.245
Population standard deviation: 25.437524255752564
```

Sampling Distributions

- Estimate the population mean using the sample average

Example 4: The lifespans of all bulbs in a batch are recorded in a file called "bulb.csv". Consider this batch bulbs as the population, estimate the population mean using a randomly selected a sample with $n = 25$ observations. Repeat the sampling experiment 1000 times to find the mean and standard deviation of the sample means.





Sampling Distributions

- Estimate the population mean using the sample average

Example 4: The lifespans of all bulbs in a batch are recorded in a file called "bulb.csv". Consider this batch bulbs as the population, estimate the population mean using a randomly selected a sample with $n = 25$ observations. Repeat the sampling experiment 1000 times to find the mean and standard deviation of the sample means.

```
bulb['Lifespan'].sample(5, replace=True)
```

```
316    1371.146243
498    1319.989609
427    1306.717736
217    1328.840897
110    1347.284550
Name: Lifespan, dtype: float64
```

Sampling Distributions

- Estimate the population mean using the sample average

Example 4: The lifespans of all bulbs in a batch are recorded in a file called "bulb.csv". Consider this batch bulbs as the population, estimate the population mean using a randomly selected a sample with $n = 25$ observations. Repeat the sampling experiment 1000 times to find the mean and standard deviation of the sample means.

```
sample = bulb['Lifespan'].sample(25, replace=True)
sample.mean()
```

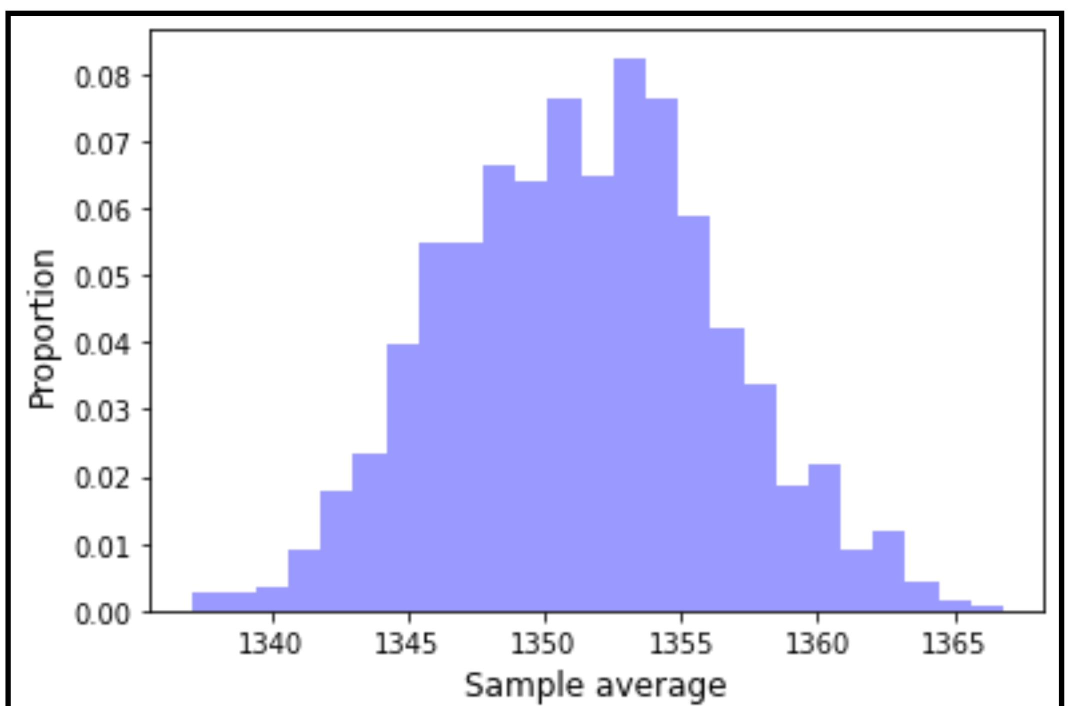
```
1351.9196336858292
```

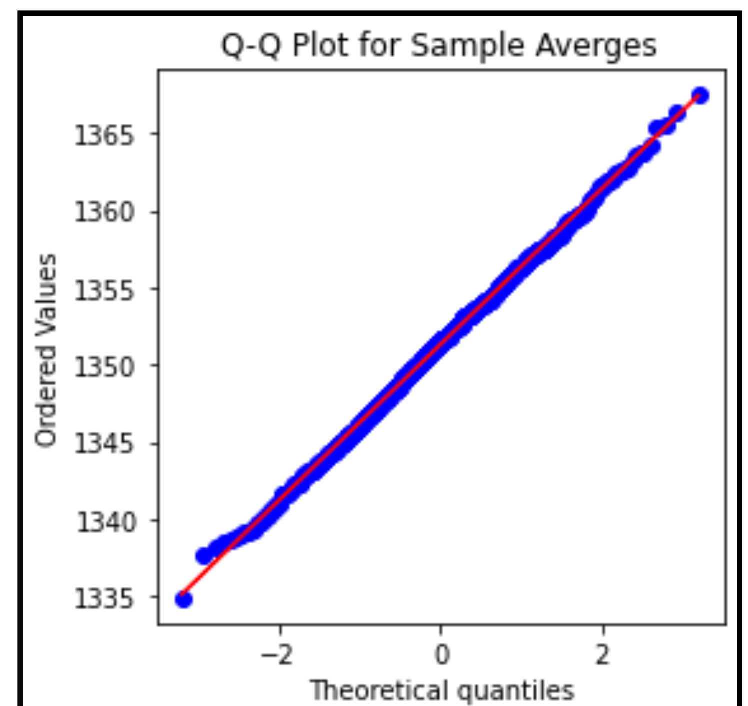
```
smp_size = 25
repeats = 1000

averages = []
for i in range(repeats):
    sample = bulb['Lifespan'].sample(smp_size, replace=True)
    averages.append(sample.mean())
```

Sampling Distributions

- Estimate the population mean using the sample average

Example 4: The lifespans of all bulbs in a batch are recorded in a file called "bulb.csv". Consider this batch bulbs as the population, estimate the population mean using a randomly selected a sample with $n = 25$ observations. Repeat the sampling experiment 1000 times to find the mean and standard deviation of the sample means.





Sampling Distributions

- Estimate the population mean using the sample average

Example 4: The lifespans of all bulbs in a batch are recorded in a file called "bulb.csv". Consider this batch bulbs as the population, estimate the population mean using a randomly selected a sample with $n = 25$ observations. Repeat the sampling experiment 1000 times to find the mean and standard deviation of the sample means.

```
pd.Series(averages).describe()
```

```
count    1000.000000
mean     1351.272444
std         5.039918
min      1334.876460
25%      1347.829726
50%      1351.493829
75%      1354.567983
max      1367.470897
dtype: float64
```

$$\mathbb{E}(\bar{X}) = \mathbb{E}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}(X_i) = \frac{1}{n}\sum_{i=1}^{n}\mu = \mu$$

Sampling Distributions

- Estimate the population mean using the sample average

Example 4: The lifespans of all bulbs in a batch are recorded in a file called "bulb.csv". Consider this batch bulbs as the population, estimate the population mean using a randomly selected a sample with $n = 25$ observations. Repeat the sampling experiment 1000 times to find the mean and standard deviation of the sample means.

```
pd.Series(averages).describe()
```

```
count    1000.000000
mean     1351.272444
std         5.039918
min      1334.876460
25%      1347.829726
50%      1351.493829
75%      1354.567983
max      1367.470897
dtype: float64
```

$$\text{Var}(\bar{X}) = \text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}(X_i) = \frac{1}{n^2}n\sigma^2 = \frac{\sigma^2}{n}$$

Sampling Distributions

- Central limit theorem
  - Sampling distribution of the sample mean

Notes: Central Limit Theorem (CLT): For a relatively large sample size, the random variable $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ is approximately normally distributed, regardless of the distribution of the population. The approximation becomes better with increased sample size.

  - Programming for Business Analytics (Topic: Sampling Distributions)