Data Science for Public Policy Part II

Aaron R. Williams

2023-08-10

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

```
1 + 1
```

```
[1] 2
```

1 Simulation and Sampling

1.1 Review

1.2 Introduction

Simulation and sampling are important tools for statistics and data science. After reviewing/introducing basic concepts about probability theory and probability distributions, we will discuss two important applications of simulation and sampling.

1. **Monte Carlo simulation:** A class of methods where values are repeatedly sampled/simulated from theoretical distributions that model a data generation process. Theoretical distributions, like the normal distribution, have closed-from representations and a finite number of parameters like mean and variance.
2. **Resampling methods:** A class of methods where values are repeatedly sampled from observed data to approximate repeatedly sampling from a population. Bootstrapping is a common resampling method.

Monte Carlo methods and resampling methods have a wide range of applications. Monte Carlo simulation is used by election forecasters to predict electoral outcomes and econometricians to understand the properties of estimators. Resampling methods are used in machine learning and causal inference. Both ar fundamental methods for agent-based models including microsimulation.

1.2.1 Fundamentals of Probability Theory

> Random Variable
>
> X is a random variable if its value is unknown and/or could change.

X could be the outcome from the flip of a coin or the roll of a die. X could also be the amount of rain next July 4th.

Experiment

An **experiment** is a process that results in a fixed set of possible outcomes.

Set

A **set** is a collection of objects.

Sample Space

A **sample space** is the set of all possible outcomes for an experiment. We will denote a sample space with Ω.

Discrete Random Variable

A set is countable if there is a one-to-one correspondence from the elements of the set to some (finite) or all (countably infinite) positive integers (i.e. 1 = heads and 2 = tails). A random variable is discrete if its sample space is countable (finite or countably infinite).

Continuous Random Variable

A random variable is continuous if its sample space is any value in a $\mathbb{R}$ (real) interval. There are infinite possible values in a real interval so the sample space is uncountable.

1.3 Discrete Random Variables

Probability Mass Function

A **probability mass function (PMF)** computes the probability of all events in the sample space of a discrete random variable.

$$p(x) = P(X = x)$$

where $0 \leq p(x) \leq 1$ and $\sum_{x \in \Omega} p(x) = 1$

```
tibble(
  a = factor(1:6),
  `P(X = a)` = rep(1 / 6, 6)
) |>
```

```r
ggplot(aes(a, `P(X = a)`)) +
geom_col() +
labs(title = "PMF for rolling a fair die")
```

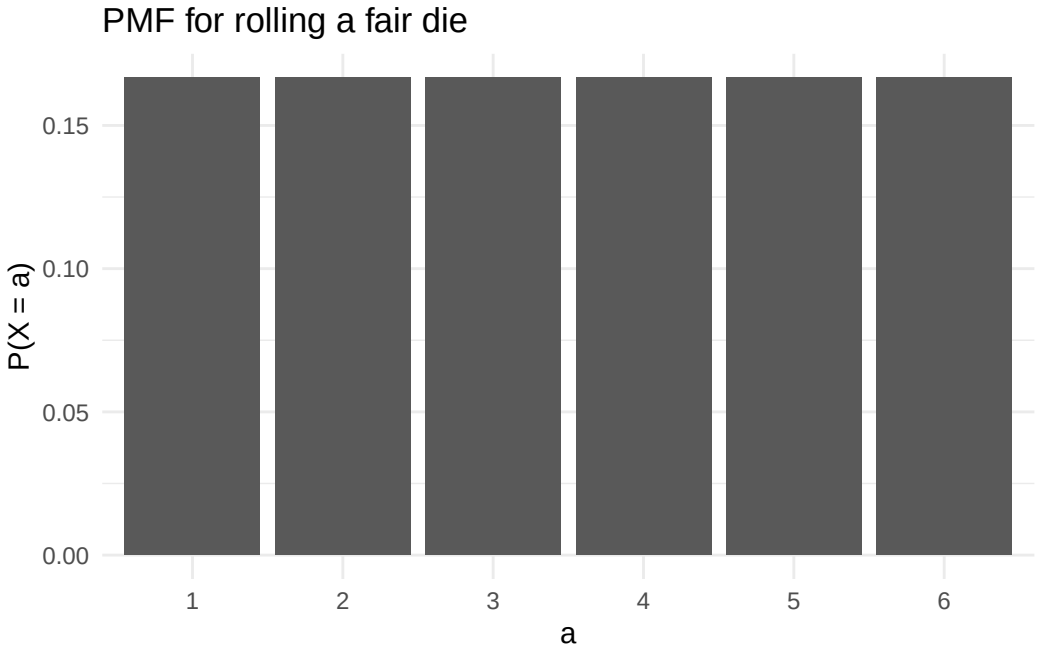


Figure 1.1: PMF for rolling a fair die

Now we can make statements like $P(X = a)$. For example, $P(X = 3) = \frac{1}{6}$.

1.3.1 Bernoulli Distribution

A Bernoulli random variable takes on the value 1 with probability p and 0 with probability 1 - p. It is often used to represent coins. When $p = \frac{1}{2}$ we refer to the coin as "fair".

We show that a random variable is Bernoulli distributed with

$$X \sim Ber(p)$$

The PMF of a Bernoulli random variable is

$$p(x) = \begin{cases} 1-p & \text{if } x = 0 \\ p & \text{if } x = 1 \end{cases} = p^x(1-p)^{1-x}$$

1.3.2 Binomial Distribution

A binomial random variable is the number of ones observed in n repeated Bernoulli trials.

We show that a random variable is Bernoulli distributed with

$$X \sim Bin(n, p)$$

The PMF of a Bernoulli random variable is

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

Figure 1.2 shows 1,000 random draws from a binomial distribution with 10 trials and `p = 0.5`. The theoretical distribution is overlayed in red.

```
tibble(
  x = rbinom(n = 1000, size = 10, prob = 0.5)
) |>
  ggplot(aes(x)) +
  geom_histogram(aes(y = after_stat(count / sum(count)))) +
  scale_x_continuous(breaks = 0:10) +
  geom_point(data = tibble(x = 0:10, y = map_dbl(0:10, dbinom, size = 10, prob = 0.5)),
             aes(x, y),
             color = "red") +
  labs(
    title = "1,000 samples of a binomial RV",
    subtitle = "Size = 10; prob = 0.5",
    y = NULL
  )
```

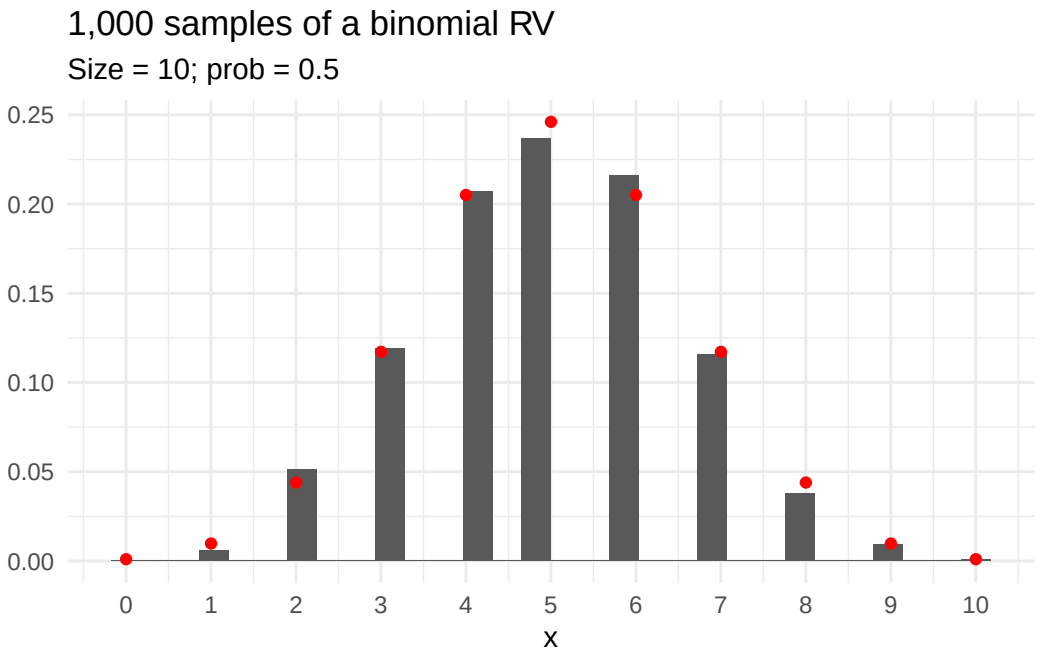


Figure 1.2: 1,000 random draws from a binomial distribution with 10 trials and p = 0.5

Sampling Error

Sampling error is the difference between sample statistics (estimates of population parameters) and population parameters.

The difference between the red dots and black bars in Figure 1.2 is caused by sampling error.

1.3.3 Distributions Using R

Most common distributions have R functions to

- calculate the density of the pdf/pmf for a specific value
- calculate the probability of observing a value less than a
- calculate the value associated with specific quantiles of the pdf/pmf
- randomly sample from the probability distribution

Let's consider a few examples:

The following answers the question: "What is the probability of observing 10 events in 10 trials when `p = 0.5`?"

```r
dbinom(x = 10, size = 10, prob = 0.5)
```

```
[1] 0.0009765625
```

The following answers the question: "What's the probability of observing 3 or fewer events in 10 trials when `p = 0.5`"

```r
pbinom(q = 3, size = 10, prob = 0.5)
```

```
[1] 0.171875
```

The following answers the question: "What is a 10th percentile number of events to see in 10 trials when `p = 0.5`?

```r
qbinom(p = 0.1, size = 10, prob = 0.5)
```

```
[1] 3
```

The following randomly draw ten different binomially distributed random variables.

```r
rbinom(n = 10, size = 10, prob = 0.5)
```

```
 [1] 5 5 5 6 4 2 3 3 5 5
```

Exercise 1

1. Add `dbinom()` for 0, 1, and 2 when `size = 10` and `prob = 0.5`.
2. Calculate `pbinom()` with `size = 10` and `prob = 0.5`. Why do you get the same answer?
3. Plug the result from step 1/step 2 into `qbinom()` with `size = 10` and `prob = 0.5`. Why do you get the answer 2?
4. Use `rbinom()` with `size = 10` and `prob = 0.5` to sample 10,000 binomially distributed random variables and assign the output to `x`. Calculate `mean(x <= 2)`. How does the answer compare to step 1/step 2?

Pseudo-random numbers

Computers use pseudo-random numbers to generate samples from probability distributions. Modern pseudo-random samplers are very random.
Use `set.seed()` to make pseudo-random sampling reproducible.

1.3.4 Poisson Random Variable

A poisson random variable is the number of events that occur in a fixed period of time. For example, a poisson distribution can be used to model the number of visits in an emergency room between 1AM and 2AM.

We show that a random variable is poisson-distributed with

$$X \sim Pois(\lambda)$$

The parameter λ is both the mean and variance of the poisson distribution. The PMF of a poisson random variable is

$$p(x) = \frac{\lambda^x \exp(-\lambda)}{x!}$$

Figure 1.3 shows 1,000 draws from a poisson distribution with $\lambda = 10$.

```
set.seed(20200905)

tibble(
  x = rpois(1000, lambda = 10)
) |>
  ggplot(aes(x)) +
  geom_histogram(aes(y = after_stat(count / sum(count)))) +
  scale_x_continuous(limits = c(0, 29)) +
  stat_function(fun = dpois, n = 30, color = "red", args = list(lambda = 10)) +
  labs(
    title = "1,000 samples of a Poisson RV",
    y = NULL
  )
```

```
Warning: Removed 2 rows containing missing values (`geom_bar()`).
```

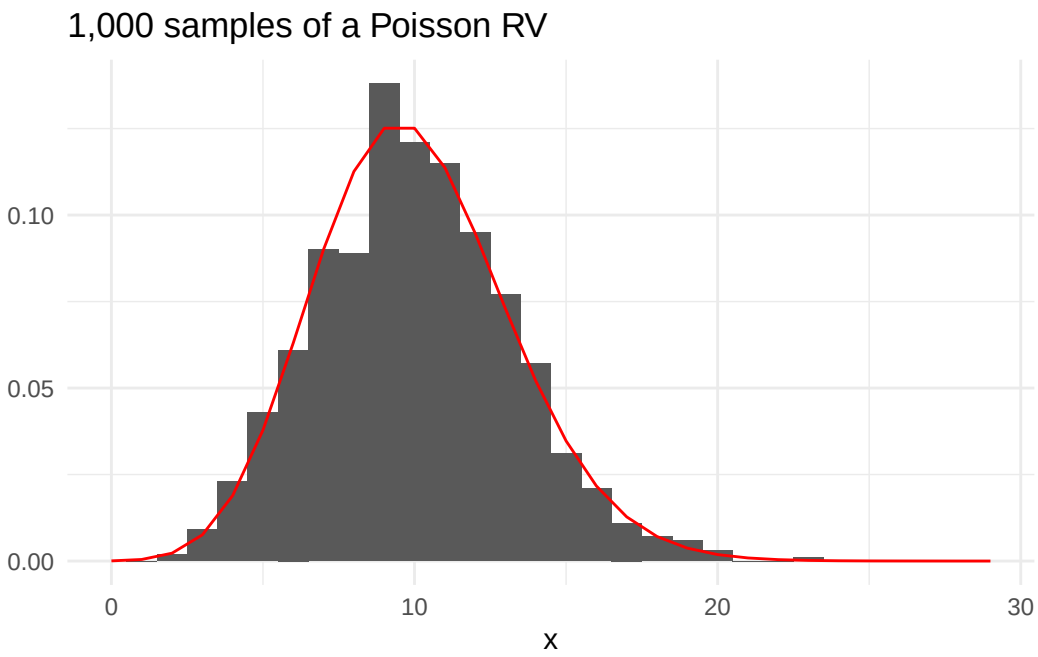


Figure 1.3: 1,000 samples of a Poisson RV

1.3.5 Categorical Random Variable

We can create a custom discrete probability distribution by enumerating the probability of each event in the sample space. For example, the PMF for the roll of a fair die is

$$p(x) = \begin{cases} \frac{1}{6} & \text{if } x = 1 \\ \frac{1}{6} & \text{if } x = 2 \\ \frac{1}{6} & \text{if } x = 3 \\ \frac{1}{6} & \text{if } x = 4 \\ \frac{1}{6} & \text{if } x = 5 \\ \frac{1}{6} & \text{if } x = 6 \end{cases}$$

This PMF is visualized in Figure 1.1. We can sample from this PDF with

```
sample(x = 1:6, size = 1)
```

```
[1] 3
```

We can also sample with probabilities that differ for each event:

```
sample(
  x = c("rain", "sunshine"),
  size = 1,
  prob = c(0.1, 0.9)
)
```

```
[1] "sunshine"
```

Exercise 2

1. Sample 1,000 observations from a Poisson distribution with $\lambda = 20$.
2. Sample 1,000 observations from a normal distribution with $\mu = 20$ and $\sigma = \sqrt{20}$.
3. Visualize and compare both distribution.

When λ is sufficiently large, the normal distribution is a reasonable approximation of the poisson distribution.

1.4 Continuous Random Variables

Probability Density Function (PDF)

A **probability density function** is a non-negative, integrable function for each real value x that shows the relative probability of values of x for an absolutely continuous random variable X.

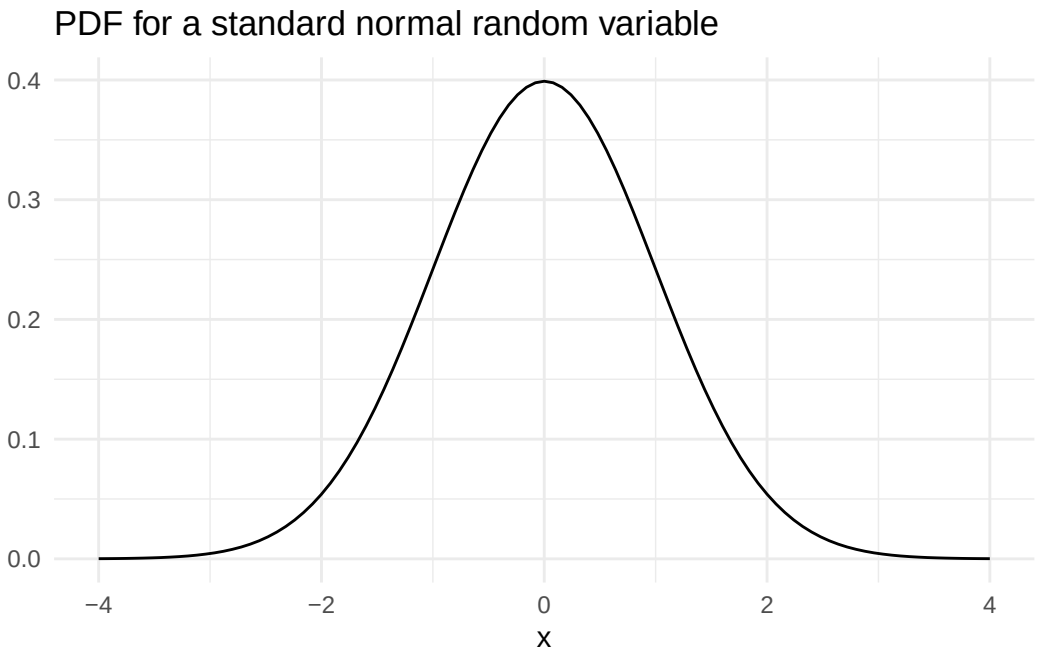
Cumulative Distribution Function (CDF)

A **cumulative distribution function (cdf)** shows the probability of a random variable X taking on any value less than or equal to x.
We note CDF with $F_X(x) = P(X \leq x)$

Here is the PDF for a **standard normal random variable:**

```
tibble(x = c(-4, 4)) |>
  ggplot(aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = 0, sd = 1)) +
  labs(
```

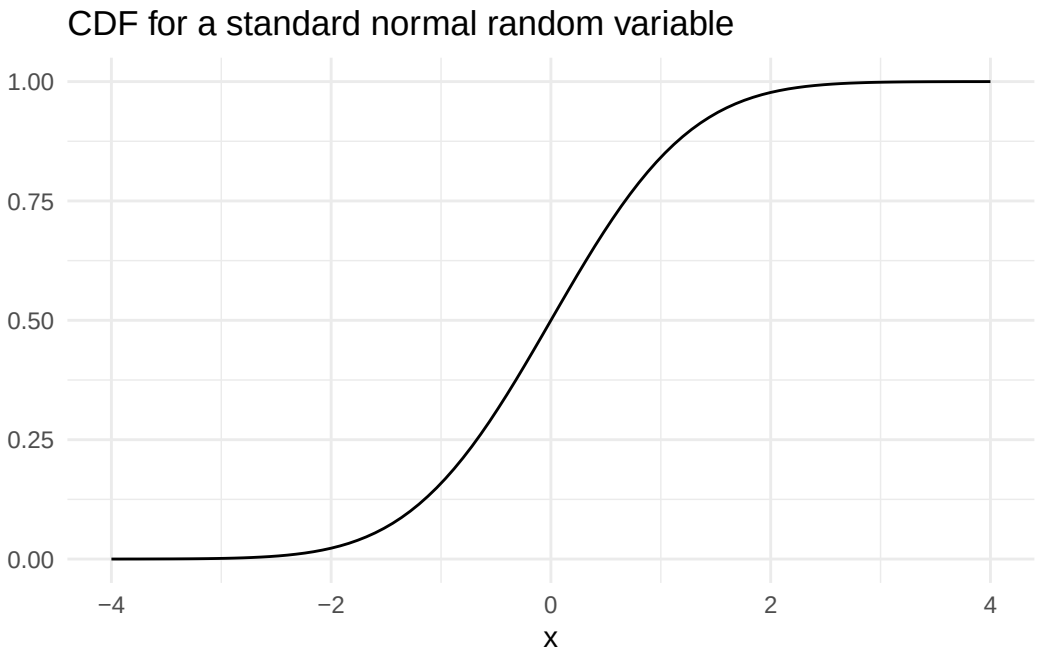
```
    title = "PDF for a standard normal random variable",
    y = NULL
  )
```



If we integrate the entire function we get the CDF.

Cumulative Density Function (CDF): A function of a random variable X that returns the probability that the value $X < x$.

```
tibble(x = c(-4, 4)) |>
  ggplot(aes(x)) +
  stat_function(fun = pnorm, n = 101, args = list(mean = 0, sd = 1)) +
  labs(
    title = "CDF for a standard normal random variable",
    y = NULL
  ) +
  theme_minimal()
```



1.4.1 Uniform Distribution

Uniform random variables have equal probability for every value in the sample space. The distribution has two parameters: minimum and maximum. A standard uniform random has minimum = 0 and maximum = 1.

We show that a random variable is uniform distributed with

$$X \sim U(a, b)$$

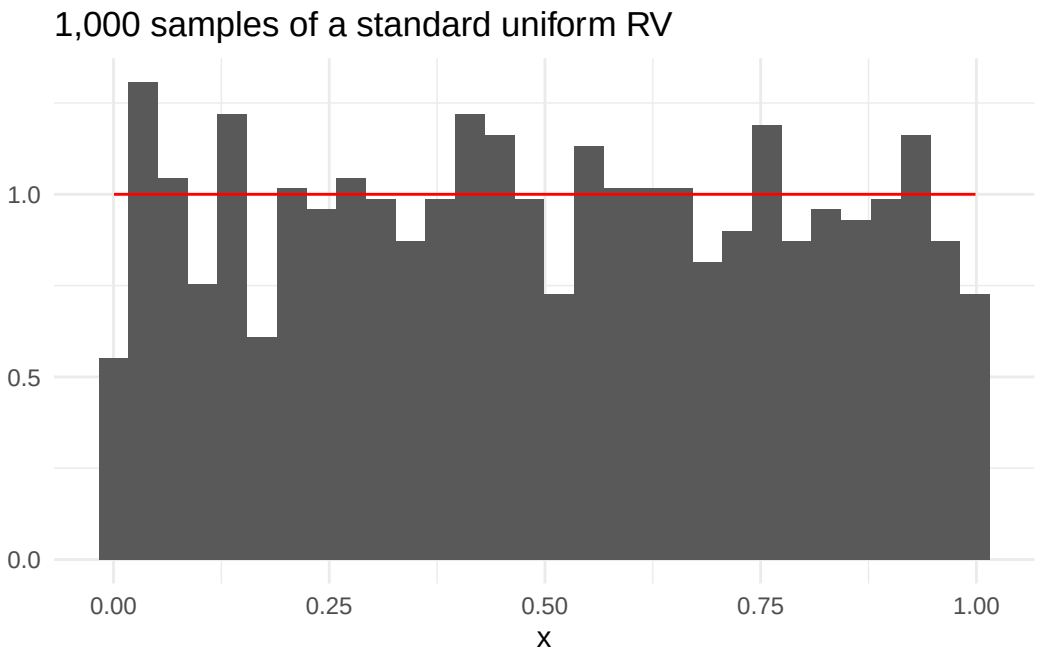
The PDF of a uniform random variable is

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{otherwise} \end{cases}$$

Standard uniform random variables are useful for generating other random processes and imputation.

```
set.seed(20200904)

tibble(
```

```
  x = runif(1000)
) |>
  ggplot(aes(x)) +
  geom_histogram(aes(y = after_stat(density))) +
  stat_function(fun = dunif, n = 101, color = "red") +
  labs(
    title = "1,000 samples of a standard uniform RV",
    y = NULL
  )
```



1.4.2 Normal Distribution

The normal distribution is the backbone of statistical inference because of the central limit theorem.

We show that a random variable is normally distributed with

$$X \sim N(\mu, \sigma)$$

The PDF of a normally distributed random variable is

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

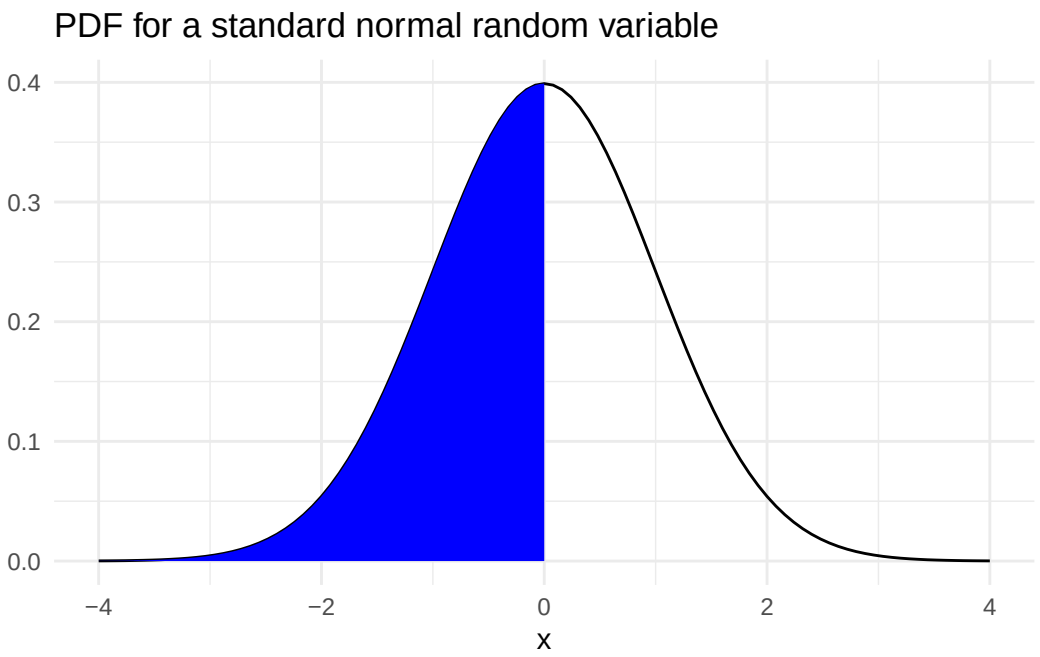
Fundamental Probability Formula for Intervals

The probability that an absolutely continuous random variable takes on any specific value is always zero because the sample space is uncountable. Accordingly, we express the probability of observing events within a region for absolutely continuous random variables.
If X has a PDF and $a < b$, then

$$P(a \leq X \leq b) = P(a \leq X < b) = P(a < X \leq b) = P(a < X < b) = \int_a^b f(x)dx = F_X(b) - F_X(a)$$

The last portion of this inequality is fundamental to working with continuous probability distributions and is the backbone of much of any intro to statistics course. For example, the probability, $P(X < 0)$ is represented by the blue region below.

```
tibble(x = c(-4, 4)) |>
  ggplot(aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = 0, sd = 1)) +
  geom_area(stat = "function", fun = dnorm, fill = "blue", xlim = c(-4, 0)) +
  labs(
    title = "PDF for a standard normal random variable",
    y = NULL
  )
```



Exercise 3

Student's t-distribution and the normal distribution are closely related.

1. Use `pnorm()` to calculate $P(X < -1)$ for a standard normal distribution.
2. Use `pt()` to calculate $P(X < -1)$ for Student's t-distribution with `df = 10`.
3. Use `pt()` to calculate $P(X < -1)$ for Student's t-distribution with `df = 100`.

Observe how the normal distribution is a better approximation for Student's t-distribution when the degrees of freedom is large.

1.4.3 Exponential Distribution

An exponential random variable is the wait time between events for a poisson random variable. It is useful for modeling wait time. For example, an exponential distribution can be used to model the wait time between arrivals in an emergency room between 1AM and 2AM. It has one parameter: rate (λ).

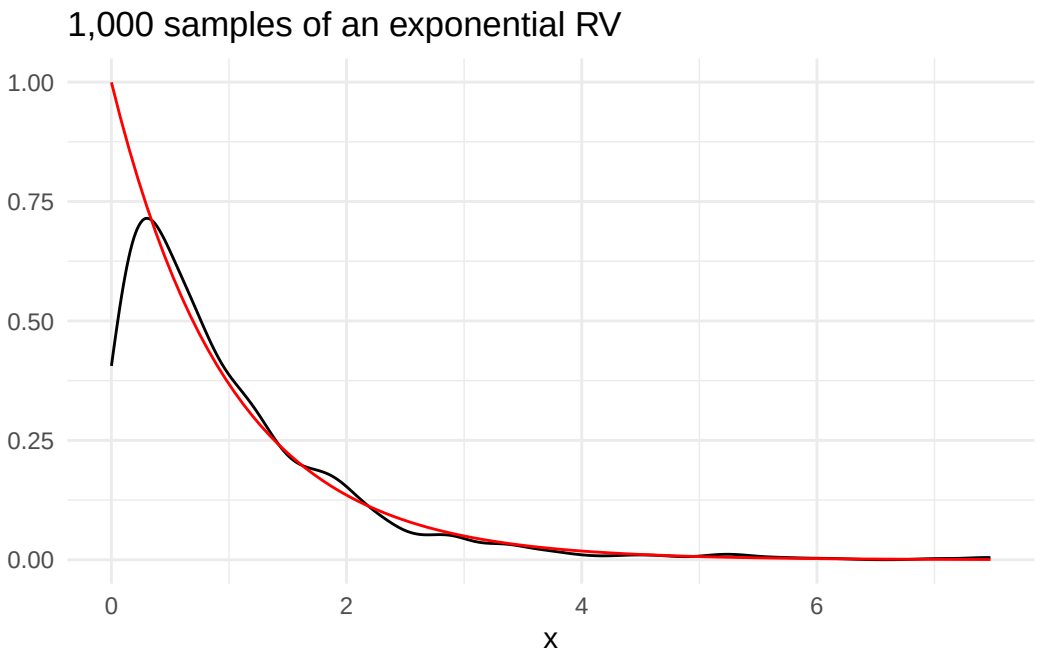
We show that a random variable is exponentially distributed with

$$X \sim Exp(\lambda)$$

The PDF of an exponential random variable is

$$f(x) = \lambda \exp(-\lambda x)$$

```
tibble(
  x = rexp(n = 1000, rate = 1)
) |>
  ggplot(aes(x)) +
  geom_density() +
  stat_function(fun = dexp, n = 101, args = list(rate = 1), color = "red") +
  labs(
    title = "1,000 samples of an exponential RV",
    y = NULL
  )
```



1.4.4 Other Distributions

- **Geometric RV:** Number of Bernoulli trials up to and including the 1^{st} success
- **Negative Binomial RV:** Number of Bernoulli trials up to and including the r^{th} success
- **Gamma RV:** Time until the α person arrives

1.5 Parametric Density Estimation

A key exercise in statistics is selecting a probability distribution to represent data and then learning the parameters of probability distributions from the data. The process is often called **model fitting**.

We are focused on parametric density estimation. Later, we will focus on nonparameteric density estimation. This section will focus on frequentist inference of population parameters from observed data. Later, we will adopt a Bayesian approach to inference.

1.5.1 Maximum Likelihood Estimation

All of the probability distributions we have observed have a finite number of parameters. Maximum likelihood estimation is a common method for estimating these parameters.

The general process is

1. Pick the probability distribution that fits the observed data.
2. Identify the finite number of parameters associated with the probability distribtuion.
3. Calculate the parameters that maximize the probability of the observed data.

Likelihood

Let $\vec{x}$ be observed data and θ be a parameter or parameters from a chosen probability distribution. The **likelihood** is the probability of the observed data conditional on values of the parameters.

$$L(\theta) = p(\vec{x}|\theta) = \Pi_{i=1}^{n} f(x_i|\theta)$$

Maximum Likelihood Estimation

Maximum likelihood estimation is a process for estimating parameters for a given distribution that maximizes the log likelihood.
In other words, MLEs find the estimated parameters that maximize the probability of observing the observed set of data.

We won't unpack how to derive the maximum likelihood estimators[1] but it is easy to look up most MLEs.

[1] (Casella and Berger 2002) offers a robust introduction to deriving maximum likelihood estimators.

1.5.1.1 Binomial distribution MLEs

Suppose we have a sample of data $x_1, ..., x_m$. If the number of trials n is already known, then p is the only parameter for the binomial distribution that needs to be estimated. The MLE for p is $\hat{p} = \frac{\sum_{i=1}^{n} x_i}{mn}$.

Suppose we observe the following vector of observed data. Next, we calculate $\hat{p}$.

```
x <- rbinom(n = 10, size = 10, prob = 0.3)

x
```

```
 [1] 0 3 5 0 4 2 3 4 1 5
```

```
mle_binom <- sum(x) / (10 * 10)
```

1.5.1.2 Normal distribution MLEs

μ and σ are the parameters of a normal distribution. The MLEs for a normal distribution are $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}$ and $s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$.[2]

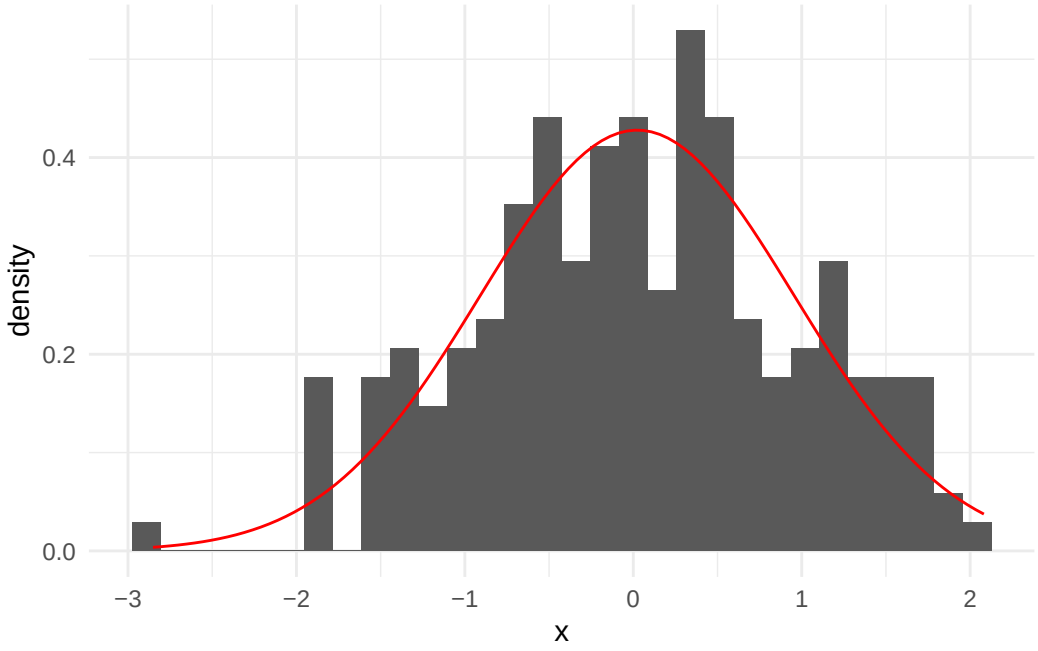
Suppose we observe the following vector of observed data. Next, we calculate $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$.

```
x <- rnorm(n = 200, mean = 0, sd = 1)
```

```
mean_hat <- mean(x)

sigma2_hat <- mean((x - mean(x)) ^ 2)


tibble(x = x) |>
  ggplot() +
  geom_histogram(aes(x, y = after_stat(density))) +
  stat_function(fun = dnorm, color = "red", args = list(mean = mean_hat, sd = sqrt(sigma2_
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

[2] Note that the MLE for variance is biased.



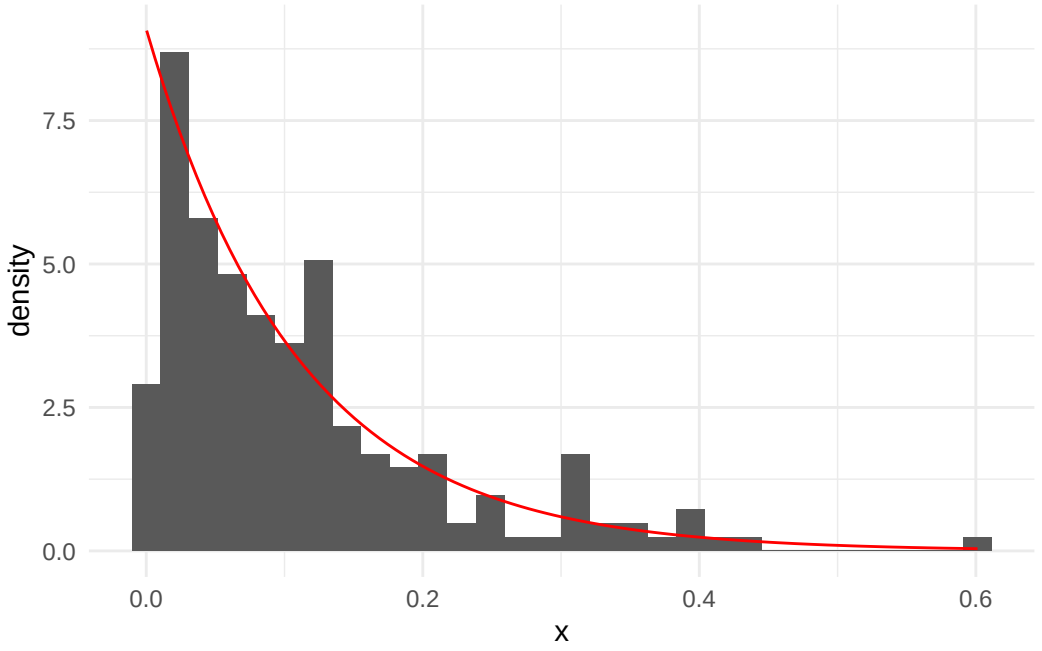
1.5.1.3 Exponential distribution MLEs

λ is the only parameter of an exponential distribution. The MLE for an exponential distribution is $\widehat{\lambda} = \frac{1}{\bar{x}}$.

Suppose we observe the following vector of observed data. Next, we calculate $\frac{1}{\bar{x}}$.

```
x <- rexp(n = 200, rate = 10)

mle_exp <- 1 / mean(x)

tibble(x = x) |>
  ggplot() +
  geom_histogram(aes(x, y = after_stat(density))) +
  stat_function(fun = dexp, color = "red", args = list(rate = mle_exp))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Exercise 4

1. Create the following vector.

```r
x <- c(
  30970.787, 10901.544, 15070.015, 10445.772, 8972.258,
  15759.614, 13341.328, 18498.858, 134462.066, 17498.930,
  7112.306, 27336.795, 75526.381, 110123.606, 32910.618,
  16764.452, 21244.380, 18952.455, 954373.470, 4219.635,
  7078.766, 27657.996, 18337.097, 14566.525, 14220.000,
  21457.202, 9322.311, 26018.018, 96325.728, 26780.329,
  25833.356, 10719.360, 8642.935, 29302.623, 10517.174,
  33831.547, 339077.456, 5805.707, 141505.710, 28168.790,
  10446.378, 4993.349, 27502.949, 35519.162, 45761.505,
  26163.096, 72163.668, 15515.435, 69396.895, 84972.590,
  67248.460, 26966.374, 24624.339, 4779.110, 23330.279,
  196311.913, 20517.739, 80257.587, 32108.466, 9735.061,
  20502.579, 2544.004, 165909.040, 20949.512, 16643.695,
  30267.741, 8359.024, 13355.154, 8425.988, 4491.550,
  32071.872, 61648.149, 75074.135, 62842.985, 26040.648,
  68733.979, 63368.710, 11157.211, 5782.610, 3629.674,
  44399.230, 2852.381, 8200.453, 41249.003, 15006.791,
  808974.653, 30705.915, 6341.954, 28208.144, 5409.821,
  54566.805, 10894.864, 4583.550, 31110.875, 43474.872,
  69059.161, 33054.574, 8789.910, 218887.477, 11051.292,
  3366.743, 63853.329, 68756.561, 48031.259, 11707.191,
  26593.634, 8868.942, 19225.309, 27704.670, 10666.549,
  47151.963, 20343.604, 123932.502, 33030.986, 5412.023,
  23540.382, 9894.513, 52742.541, 21397.990, 25100.143,
  23757.882, 48347.300, 4325.134, 23816.776, 11907.656,
  24179.849, 25967.574, 7531.294, 15131.240, 21595.781,
  40473.936, 35390.849, 4060.563, 55334.157, 37058.771,
  34050.456, 17351.500, 7453.829, 48131.565, 10576.746,
  26450.754, 33592.986, 21425.018, 34729.337, 77370.078,
  5819.325, 9067.356, 19829.998, 20120.706, 3637.042,
  44812.638, 22930.229, 29683.776, 76366.822, 15464.594,
  1273.101, 53036.266, 2846.294, 114076.200, 14492.680,
  55071.554, 31597.849, 199724.125, 52332.510, 98411.129,
  43108.506, 6580.620, 12833.836, 8846.348, 7599.796,
  6952.447, 30022.143, 24829.739, 40784.581, 8997.219,
  3786.354, 11515.298, 116515.617, 137873.967, 3282.185,
  107886.676, 13184.850, 51083.235, 2907.886, 51827.538,
  37564.196, 23196.399, 20169.037, 9020.364, 11118.250,
  56930.060, 11657.302, 84642.584, 44948.450, 16610.166,
  5509.231, 4770.262, 15614.233, 5993.999, 22628.114
)
```

2. Visualize the data with a relative frequency histogram.
3. Calculate the MLEs for a normal distribution and add a normal distribution to the visualization in red.
4. Calculate the MLEs for a log-normal distribution and add a log-normal distribution to the visualization in blue.

1.6 Multivariate Random Variables

We've explored univariate or marginal distributions thus far. Next, we will focus on multivariate distributions.

Multivariate Distribution

A **multivariate distribution** is a probability distribution that shows the probability (discrete) or relative probability (continuous) of more than one random variable. Multivariate distributions require describing characteristics of random variables and the relationships between random variables.

1.6.1 Multivariate Normal Distribution

The multivariate normal distribution is a higher-dimensional version of the normal distribution.

Instead of a single mean and a single variance, the k-dimensional multivariate normal distribution has a vector of means of length k and a k-by-k variance-covariance matrix. The vector describes the central tendencies of each dimension of the multivariate distribution and the matrix desrcibe the variance of the distributions and relationships between the distributions.

We show that a random vector is multivariate normally distributed with

$$\vec{X} \sim \mathcal{N}(\vec{\mu}, \Sigma)$$

The PDF of a multivariate normally distributed random variable is

$$f(x) = (2\pi)^{-k/2} det(\Sigma)^{-1/2} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})\right)$$

Functions for working with multi-variate normal distributions from `library(MASS)`. Figure 1.4 shows three different random samples from 2-dimensional multivariate normal distributions.

```r
Sigma1 <- matrix(c(1, 0.8,
                   0.8, 1),
                 nrow = 2,
                 byrow = TRUE)

mvrnorm(n = 1000, mu = c(0, 0), Sigma = Sigma1) |>
  as_tibble() |>
  ggplot(aes(V1, V2)) +
  geom_point()
```

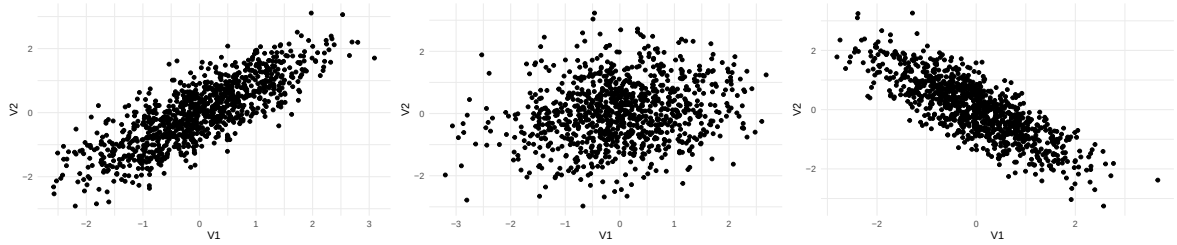
```
Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if
`.name_repair` is omitted as of tibble 2.0.0.
i Using compatibility `.name_repair`.
```

```r
Sigma2 <- matrix(c(1, 0.2,
                   0.2, 1),
                 nrow = 2,
                 byrow = TRUE)

mvrnorm(n = 1000, mu = c(0, 0), Sigma = Sigma2) |>
  as_tibble() |>
  ggplot(aes(V1, V2)) +
  geom_point()

Sigma3 <- matrix(c(1, -0.8,
                   -0.8, 1),
                 nrow = 2,
                 byrow = TRUE)

mvrnorm(n = 1000, mu = c(0, 0), Sigma = Sigma3) |>
  as_tibble() |>
  ggplot(aes(V1, V2)) +
  geom_point()
```

1.7 Monte Carlo Methods

Simulation methods, including Monte Carlo simulation, are used for policy analysis:

- FiveThirtyEight and the New York Times use Monte Carlo simulation to predict the outcomes of elections.



(a) Strong Positive Covariance (b) Weak Covariance (c) Strong Negative Covariance

Figure 1.4: Samples from Multivariate Normal Distributions

- The Social Security Administration uses microsimulation to evaluate the distributional impact of Social Security reforms.
- The Census Bureau uses simulation to understand the impact of statistical disclosure control on released data.
- Econometricians and statisticians use Monte Carlo simulation to demonstrate the properties of estimators.

We can make probabilistic statements about common continuous random variables because their PDFs are integrable or at least easy enough to approximate with lookup tables. We can make probabilistic statements about common discrete random variables with summation.

But we often want to make probabilistic statements about uncommon or complex probability distributions. Maybe the probability distribution of the random variable doesn't have a tractable integral (i.e. the area under the curve can't practically be computed). Or maybe there are too many potential outcomes (e.g. rays of light emitting from a lightbulb in the Marble Science video linked at the top).

Monte Carlo: A Monte Carlo method estimates a deterministic quantity using stochastic (random) sampling.

Monte Carlo but easier this time: A Monte Carlo method takes hundreds or thousands of independent samples from a random variable or variables and then approximates fixed population quantities with summaries of those draws. The quantities could be population parameters like a population mean or probabilities.

Monte Carlo methods have three major applications:

1. **Sampling** – Monte Carlo simulation allows for sampling from complex probability distributions. The samples can be used to model real-world events (queues), to model outcomes with uncertain model inputs (election modeling), to generate fake data with known parameters to evaluate statistical methods (model selection when assumptions fail), and to draw from the posteriors of Bayesian models.

2. **Numerical integration** – Integration, as noted above, is important to calculating probabilities and ultimately calculating quantities like expected value or the intervals. Monte Carlo methods can approximate multidimensional integrals that will never be directly solved by computers or simplify estimating probabilities when there are uncountably many potential outcomes (Solitaire).
3. **Optimization** – Monte Carlo methods can be used for complex optimization. We will not focus on optimization.

Let's explore some examples:

1.7.1 Example 1: Coin Tossing

We can calculate the proportion of tosses of a fair coin that we expect to turn up heads by finding the expected value of the binomial distribution and dividing by the number of tosses. But suppose we can't… Or maybe we wish to confirm our calculations with simulations…

Let's try repeated sampling from a binomial distribution to approximate this process:

```
#' Count the proportion of n tosses that turn up heads
#'
#' @param n An integer for the number of tosses
#'
#' @return The proportion of n tosses that turn up heads
#'
count_heads <- function(n) {

  # toss the fair coin n times
  coin_tosses <- rbinom(n = n, size = 1, prob = 0.5)

  coin_tosses <- if_else(coin_tosses == 1, "heads", "tails")

  # calculate the proportion of heads
  prop_heads <- mean(coin_tosses == "heads")

  return(prop_heads)

}
```

Let's toss the coin ten times.

```
set.seed(11)
count_heads(n = 10)
```

```
[1] 0.3
```

Ok, we got 0.3, which we know isn't close to the expected proportion of 0.5. What if we toss the coin 1 million times.

```
set.seed(20)
count_heads(n = 1000000)
```

```
[1] 0.499872
```

Ok, that's more like it.

. . .

Monte Carlo simulation works because of the **law of large numbers**. The law of large numbers states that the probability that the average of trials differs from the expected value converges to zero as the number of trials goes to infinity.

Monte Carlo simulation basically repeats the ideas behind frequentist inferential statistics. If we can't measure every unit in a population then we can sample a representative population and estimate parameters about that population.

The keys to Monte Carlo simulation are **randomness** and independent and identically distributed sampling (**i.i.d.**).

1.7.2 Example 2: Bootstrap Sampling

People claim that a bootstrap sample includes about 0.63 of the values from the source of the sample. We can evaluate this claim with repeated samples from a categorical distribution. We will use `sample()`.

So if we bootstrap a vector of length 100, then $\frac{63}{100}$ values will end up in the bootstrap sample on average and $\frac{37}{100}$ of the values will be repeats on average.

Let's demonstrate this with Monte Carlo simulation:

```
#' Calculate the proportion of unique values from a vector of integers included
#' in a bootstrap sample
#'
#' @param integers A vector of integers
#'
#' @return The proportion of integers included in the bootstrap sample
```

```r
#'
count_uniques <- function(integers) {

  # generate a bootstrap sample
  samples <- sample(integers, size = length(integers), replace = TRUE)

  # calculate the proportion of unique values from the original vector
  prop_unique <- length(unique(samples)) / length(integers)

  return(prop_unique)

}
```

Let's bootstrap sample 100,000 times.

```r
prop_unique <- vector(mode = "numeric", length = 100000)
for (i in seq_along(prop_unique)) {

  prop_unique[i] <- count_uniques(integers = 1:100)

}
```

Finally, calculate the mean proportion and estimate the expected value.

```r
mean(prop_unique)
```

```
[1] 0.6337935
```

We can also calculate a 95% confidence interval using the bootstrap samples.

```r
quantile(prop_unique, probs = c(0.025, 0.975))
```

```
 2.5% 97.5%
 0.57  0.69
```

1.7.3 Example 3: π

Consider one of the examples from Marble Science: Monte Carlo Simulation. Imagine we don't know π but we know that the equation for the area of a square is r^2 and the equation for the

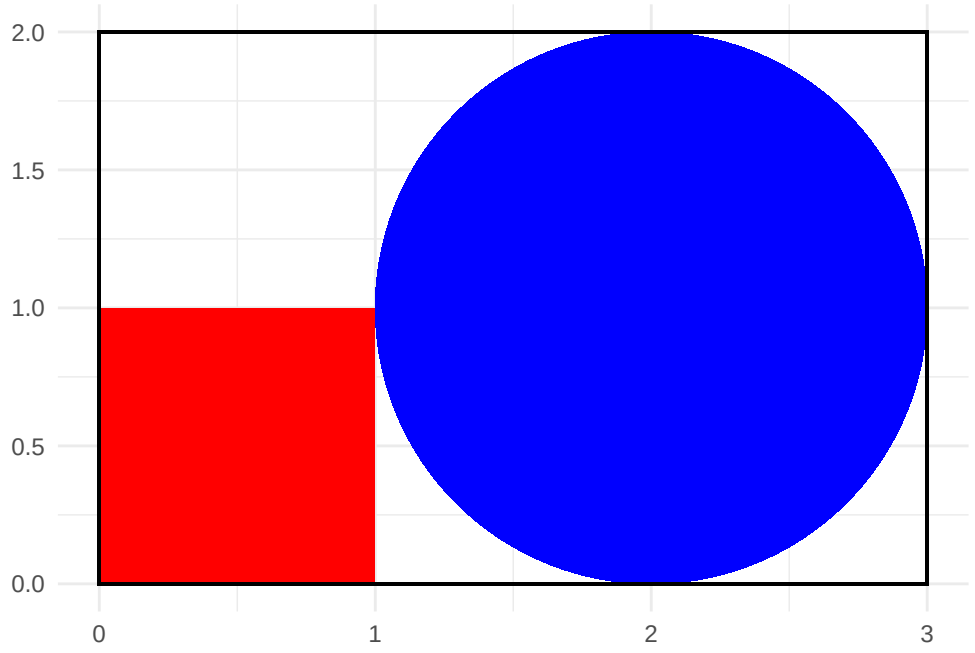
area of a circle is πr^2. If we know the ratio of the areas of the circle and the square, then we can solve for π.

$$\pi = \frac{\pi r^2}{r^2}$$

This is simply solved with Monte Carlo simulation. Randomly sample a bivariate uniform random variables and count how frequently the values are inside of the square or inside the circle.

```
expand_grid(
  x = seq(0, 4, 0.1),
  y = seq(0, 2, 0.1)
) |>
ggplot() +
  ggforce::geom_circle(aes(x0 = 2, y0 = 1, r = 1), fill = "blue", color = NA) +
  geom_rect(aes(xmin = 0, xmax = 1, ymin = 0, ymax = 1), fill = "red", color = NA) +
  geom_rect(aes(xmin = 0, xmax = 3, ymin = 0, ymax = 2), fill = NA, color = "black") +
  coord_fixed()
```

```
Warning: Using the `size` aesthetic in this geom was deprecated in ggplot2 3.4.0.
i Please use `linewidth` in the `default_aes` field and elsewhere instead.
```



```r
number_of_samples <- 2000000

# sample points in a rectangle with x in [0, 3] and y in [0, 2]
set.seed(20210907)
samples <- tibble(
  x = runif(number_of_samples, min = 0, max = 3),
  y = runif(number_of_samples, min = 0, max = 2)
)

# calculate if (x, y) is in the circle, the square, or neither
samples <- samples |>
  mutate(
    in_square = between(x, 0, 1) & between(y, 0, 1),
    in_circle = (x - 2) ^ 2 + (y - 1) ^ 2 < 1
  )

# calculate the proportion of samples in each shape
prop_in_shapes <- samples |>
  summarize(
    prop_in_square = mean(in_square),
    prop_in_circle = mean(in_circle)
  )

# calculate the ratio
prop_in_shapes |>
  mutate(prop_in_circle / prop_in_square) |>
  print(digits = 3)
```

```
# A tibble: 1 x 3
  prop_in_square prop_in_circle `prop_in_circle/prop_in_square`
           <dbl>          <dbl>                           <dbl>
1          0.166          0.524                            3.15
```

The answer approximates π!

1.7.4 Example 4: Simple Linear Regression

The goal of statistical inference is to use data, statistics, and assumptions to infer parameters and probabilities about a population. Typically we engage in point estimation and interval estimation.

Sometimes it is useful to reverse this process to understand and confirm the properties of estimators. That means starting with known population parameters, simulating hundreds or thousands of samples from that population, and then observing point estimates and interval estimates over those samples.

1.7.4.1 Linear Regression Assumptions

1. The population model is of the linear form $y = \beta_0 + \beta_1 x + \epsilon$
2. The estimation data come from a random sample or experiment
3. $\epsilon_i \sim N(0, \sigma^2)$ independently and identically distributed (i.i.d.)
4. x has variance and there is no perfect collinearity in x

1.7.4.2 Statistics

If we have one sample of data, we can estimate points and intervals with the following estimators:

The residual standard error is

$$\hat{\sigma} = \frac{\sum e_i^2}{(n-2)}$$

The estimate of the slope is

$$\hat{\beta}_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

The standard error of the estimate of the slope, which can be used to calculate t-statistics and confidence intervals, is

$$\hat{SE}(\hat{\beta}_1) = \sqrt{\frac{\hat{\sigma}^2}{\sum (x_i - \bar{x})^2}}$$

The estimate of the intercept term is

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

The standard error of the intercept is

$$\hat{SE}(\hat{\beta}_0) = \sqrt{\hat{\sigma}^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum (x_i - \bar{x})^2} \right]}$$

1.7.4.3 Monte Carlo Simulation

Consider a simple linear regression model with the following population model:

$$y = 5 + 15x + \epsilon$$

We can calculate the above statistics over repeated sampling and confirm their asymptotic properties with Monte Carlo simulation.

First, create 1,000 random samples from the population.

```
set.seed(20210906)

data <- map(
  .x = 1:1000,
  .f = ~ tibble(
    x = rnorm(n = 10000, mean = 0, sd = 2),
    epsilon = rnorm(n = 10000, mean = 0, sd = 10),
    y = 5 + 15 * x + epsilon
  )
)
```

Next, estimate a simple linear regression model for each draw of the population. This step includes calculating $\hat{\sigma}$, $\hat{\beta}_1$, $\hat{\beta}_0$, $\widehat{SE}(\hat{\beta}_1)$, and $\widehat{SE}(\hat{\beta}_0)$.

```
estimated_models <- map(
  .x = data,
  .f = ~ lm(y ~ x, data = .x)
)
```

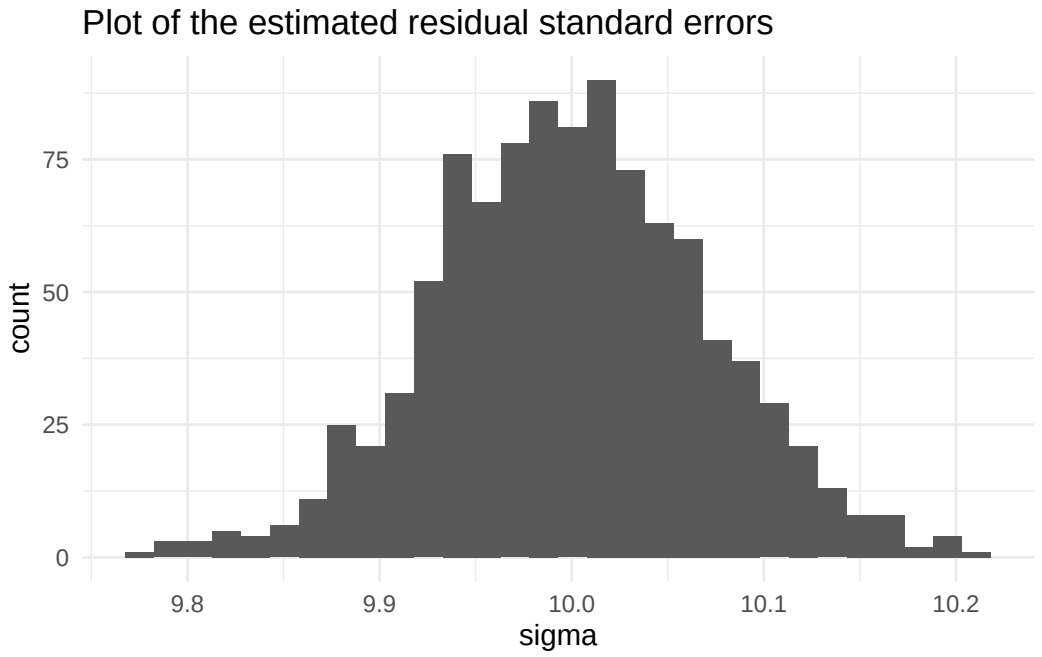
Next, we extract the coefficients and confidence intervals.

```
coefficients <- map_df(
  .x = estimated_models,
  .f = tidy,
  conf.int = TRUE
)
```

Let's look at estimates of the residual standard error. The center of the distribution closely matches the population standard deviation of the error term.

```r
model_metrics <- map_df(
  .x = estimated_models,
  .f = glance
)

model_metrics |>
  ggplot(aes(sigma)) +
  geom_histogram() +
  labs(title = "Plot of the estimated residual standard errors")
```

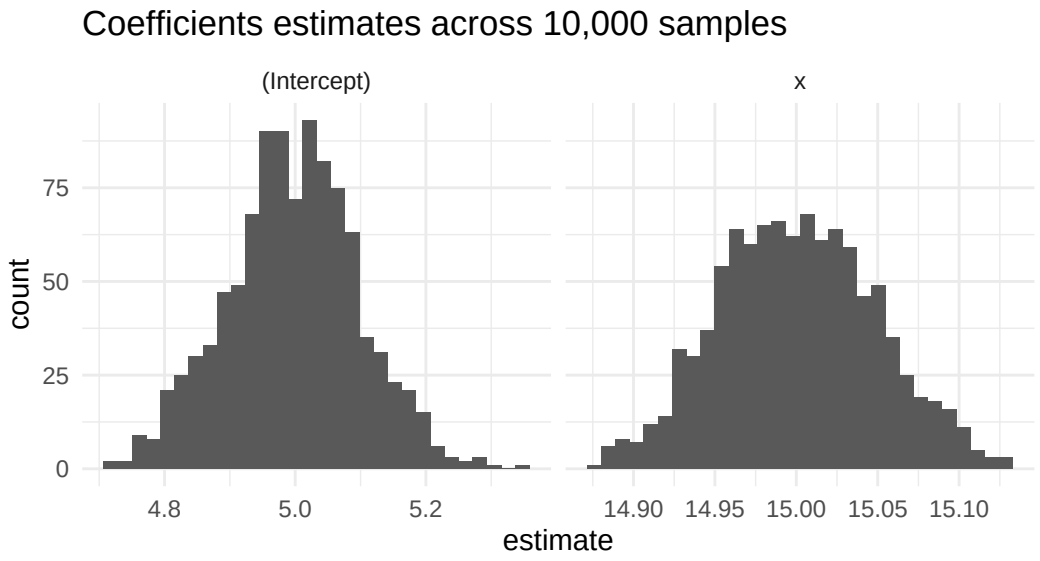
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Let's plot the coefficients. The centers approximately match the population intercept of 5 and slope of 15.

```r
coefficients |>
  ggplot(aes(estimate)) +
  geom_histogram() +
  facet_wrap(~term, scales = "free_x") +
  labs(title = "Coefficients estimates across 10,000 samples")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The standard deviation of the coefficients also matches the standard errors.

$$\hat{SE}(\hat{\beta}_1) = \sqrt{\frac{\hat{\sigma}^2}{\sum(x_i - \bar{x})^2}} = \sqrt{\frac{10^2}{40,000}} = 0.05$$

$$\hat{SE}(\hat{\beta}_0) = \sqrt{\hat{\sigma}^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum(x_i - \bar{x})^2}\right]} = \sqrt{10^2 \left[\frac{1}{10,000} + 0\right]} = 0.1$$

```
coefficients |>
  group_by(term) |>
  summarize(
    mean(estimate),
    sd(estimate)
  )
```

```
# A tibble: 2 x 3
  term        `mean(estimate)` `sd(estimate)`
  <chr>                  <dbl>          <dbl>
1 (Intercept)             5.00         0.100
2 x                      15.0          0.0482
```

Let's look at how often the true parameter is inside the 95% confidence interval. It's close although not exactly 95%.

```
coefficients |>
  filter(term == "x") |>
  summarize(ci_contain_beta = mean(conf.low <= 15 & conf.high >= 15))
```

```
# A tibble: 1 x 1
  ci_contain_beta
            <dbl>
1           0.959
```

```
coefficients |>
  filter(term == "(Intercept)") |>
  summarize(ci_contain_beta = mean(conf.low <= 5 & conf.high >= 5))
```

```
# A tibble: 1 x 1
  ci_contain_beta
            <dbl>
1            0.95
```

1.7.5 Example 5: Queuing Example

Suppose we have a queue at a Social Security field office. Let t be time. When the office opens, $t = 0$ and the queue is empty.

Let, T_i be the **inter**arrival time and $T_i \sim exp(\lambda_1)$

Let, S_i be the service time time and $S_I \sim exp(\lambda_2)$

From these two random variables, we can calculate the arrival times, departure times, and wait times for each customer.

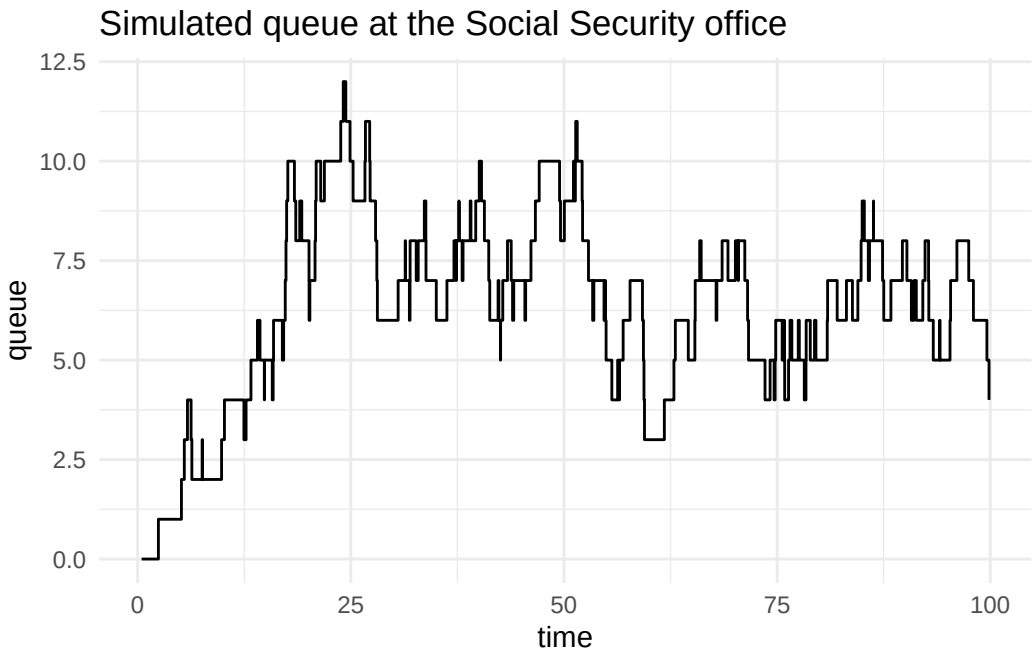
- The arrival times are the cumulative sum of the interarrival times.
- The wait times are zero if a person arrives after the person before them and the difference between the prior person's departure and the current person's arrival otherwise.
- The departure time is arrival time plus the wait time plus the service time.

```
set.seed(19920401)
queue <- generate_queue(t = 100, lambda = 1, mu = 1)
```

```
queue
```

```
# A tibble: 100 x 5
   interarrival_time arrival_time service_time wait_time departure_time
               <dbl>        <dbl>        <dbl>     <dbl>          <dbl>
 1             0.467        0.467       5.80        0              6.27
 2             1.97         2.43        0.0892      3.83           6.36
 3             2.70         5.13        1.26        1.22           7.61
 4             0.335        5.47        4.85        2.14          12.5
 5             0.372        5.84        1.89        6.62          14.3
 6             1.72         7.56        0.507       6.78          14.9
 7             2.28         9.84        0.932       5.01          15.8
 8             0.339       10.2         1.18        5.61          17.0
 9             2.54        12.7         1.42        4.25          18.4
10             0.572       13.3         0.157       5.10          18.5
# i 90 more rows
```

```
flow <- tibble::tibble(
  time = c(queue$arrival_time, queue$departure_time),
  type = c(rep("arrival", length(queue$arrival_time)),
           rep("departure", length(queue$departure_time))),
  change = c(rep(1, length(queue$arrival_time)), rep(-1, length(queue$departure_time))),
) |>
  arrange(time) |>
  filter(time < 100) |>
  mutate(queue = cumsum(change) - 1)

flow |>
  ggplot(aes(time, queue)) +
  geom_step() +
  labs(title = "Simulated queue at the Social Security office")
```



This is interesting, but it's still only one draw from a Monte Carlo simulation. What if we are interested in the distribution of wait times for the fifth customer?

```
#' Generate wait times at the queue
#'
#' @param person_number An integer for the person of interest
#' @param iterations An integer for the number of Monte Carlo iterations
#' @param t A t for the maximum time
#'
#' @return A vector of wait times
#'
generate_waits <- function(person_number, iterations, t) {

  wait_time <- vector(mode = "numeric", length = iterations)
  for (i in seq_along(wait_time)) {

    wait_time[i] <- generate_queue(t = t, lambda = 1, mu = 1)$wait_time[person_number]

  }

  return(wait_time)
```

```
}

set.seed(20200908)
wait_time <- generate_waits(person_number = 5, iterations = 10000, t = 50)

mean(wait_time)
```

```
[1] 1.464371
```

```
quantile(wait_time, probs = c(0.025, 0.5, 0.975))
```

```
     2.5%       50%     97.5%
0.0000000 0.9222193 5.8742015
```

Exercise 5

1. Create a Monte Carlo simulation of an unfair coin toss where `p = 0.6`.

Exercise 6

Suppose we have three independent normally-dstributed random variables.

$$X_1 \sim N(\mu = 0, \sigma = 1)$$

$$X_2 \sim N(\mu = 1, \sigma = 1)$$

$$X_3 \sim N(\mu = 2, \sigma = 1)$$

2. Use Monte Carlo simulation with 10,000 repetitions to estimate how often $X_{i1} < X_{i2} < X_{i3}$.

1.7.6 More examples of Monte Carlo simulation

- fivethirtyeight 2020 election forecast use
- U.S. Census Bureau simulation of data collection operations

1.7.6.1 Markov Chain Monte Carlo

Bayesian statisticians estimate posterior distributions of parameters that are combinations of prior distributions and sampling distributions. Outside of special cases, posterior distributions are difficult to identify. Accordingly, most Bayesian estimation uses an extension of Monte Carlo simulation called Markov Chain Monte Carlo or MCMC.

1.7.7 One Final Note

Monte Carlo simulations likely underestimate uncertainty. Monte Carlo simulations only capture aleatoric uncertainty and they don't capture epistemic uncertainty.

Aleatoric uncertainty: Uncertainty due to probabilistic variety

Epistemic uncertainty: Uncertainty due to a lack of knowledge

In other words, Monte Carlo simulations estimates assume the model is correct, which is almost certainly never fully true. Be transparent. Be humble.

1.8 Sampling from Observed Data

Until now, we've only discussed sampling from closed-form theoretical distributions. We also called this process simulation. There are many applications where we may want to sample from observed data.

We can break these methods into two general approaches:

1. Sampling
2. Resampling

1.8.1 Sampling

Sampling

Sampling is the process of selecting a subset of data. **Probability sampling** is the process of selecting a sample when the selection uses randomization.

Sampling has many applications:

- Reducing costs for the collection of data
- Implementing machine learning algorithms
- Resampling

1.8.2 Resampling

> Resampling
>
> **Resampling** is the process of repeatedly sampling from observed data to approximate the generation of new data.

There are at least three popular resmpling methods:

1. **Cross Validation:** Partitioning the data and shuffling the partitions to understand the accuracy of predictive models.
2. **Bootstrap sampling:** Repeated sampling with replacement to estimate sampling distributions from observed data.
3. **Jackknife:** Leave-one-out sampling to estimate the bias and standard error of a statistic.

We focused on cross-validation for machine learning and predictive modeling in data science for public policy. We will use this approach again for predictive modeling.

We will also learned about bootstrap sampling when we discuss nonparametric statistics.

References

Casella, George, and Roger L. Berger. 2002. *Statistical Inference*. 2nd ed. Australia ; Pacific Grove, CA: Thomson Learning.