

Data Science Probability

The textbook for the Data Science course series is [freely available online](#).

This course corresponds to the Probability section of textbook, starting [here](#).

This course assumes you are comfortable with basic math, algebra, and logical operations. HarvardX has partnered with DataCamp for all assignments in R that allow you to program directly in a browser-based interface. You will not need to download any special software.

Using a combination of a guided introduction through short video lectures and more independent in-depth exploration, you will get to practice your new R skills on real-life applications.

Probability theory is the mathematical foundation of statistical inference which is indispensable for analyzing data affected by chance, and thus essential for data scientists. In this course, you will learn important concepts in probability theory. The motivation for this course is the circumstances surrounding the financial crisis of 2007-2008. Part of what caused this financial crisis was that the risk of certain securities sold by financial institutions was underestimated. To begin to understand this very complicated event, we need to understand the basics of probability. We will introduce important concepts such as random variables, independence, Monte Carlo simulations, expected values, standard errors, and the Central Limit Theorem. These statistical concepts are fundamental to conducting statistical tests on data and understanding whether the data you are analyzing are likely occurring due to an experimental method or to chance. Statistical inference, covered in the next course in this series, builds upon probability theory.

Learning Objectives

- Important concepts in probability theory including random variables and independence
- How to perform a Monte Carlo simulation
- The meaning of expected values and standard errors and how to compute them in R
- The importance of the Central Limit Theorem

Course Overview

Section 1: Discrete Probability

You will learn about basic principles of probability related to categorical data using card games as examples.

Section 2: Continuous Probability

You will learn about basic principles of probability related to numeric and continuous data.

Section 3: Random Variables, Sampling Models, and The Central Limit Theorem

You will learn about random variables (numeric outcomes resulting from random processes), how to model data generation procedures as draws from an urn, and the Central Limit Theorem, which applies to large sample sizes.

Section 4: The Big Short

You will learn how interest rates are determined and how some bad assumptions led to the financial crisis of 2007-2008.

Section 1 Overview

Section 1 introduces you to Discrete Probability. Section 1 is divided into three parts:

- Introduction to Discrete Probability
- Combinations and Permutations
- Addition Rule and Monty Hall

After completing Section 1, you will be able to:

- Apply basic probability theory to categorical data.
- Perform a Monte Carlo simulation to approximate the results of repeating an experiment over and over, including simulating the outcomes in the Monty Hall problem.
- Distinguish between: sampling with and without replacement, events that are and are not independent, and combinations and permutations.
- Apply the multiplication and addition rules, as appropriate, to calculate the probability of multiple events occurring.
- Use **sapply** instead of a for loop to perform element-wise operations on a function.

Discrete Probability

The textbook for this section is available [here](#)

Key points

- The *probability of an event* is the proportion of times the event occurs when we repeat the experiment independently under the same conditions.

$Pr(A)$ = probability of event A

- An *event* is defined as an outcome that can occur when something happens by chance.
- We can determine probabilities related to discrete variables (picking a red bead, choosing 48 Democrats and 52 Republicans from 100 likely voters) and continuous variables (height over 6 feet).

Monte Carlo Simulations

The textbook for this section is available [here](#)

Key points

- Monte Carlo simulations model the probability of different outcomes by repeating a random process a large enough number of times that the results are similar to what would be observed if the process were repeated forever.
- The **sample** function draws random outcomes from a set of options.
- The **replicate** function repeats lines of code a set number of times. It is used with **sample** and similar functions to run Monte Carlo simulations.

Code: The rep function and the sample function

```
beads <- rep(c("red", "blue"), times = c(2,3))    # create an urn with 2 red, 3 blue
beads      # view beads object
```

```
## [1] "red" "red" "blue" "blue" "blue"
```

```
sample(beads, 1)    # sample 1 bead at random
```

```
## [1] "blue"
```

Code: Monte Carlo simulation

Note that your exact outcome values will differ because the sampling is random.

```
B <- 10000    # number of times to draw 1 bead
events <- replicate(B, sample(beads, 1))    # draw 1 bead, B times
tab <- table(events)    # make a table of outcome counts
tab      # view count table
```

```
## events
## blue red
## 5983 4017
```

```
prop.table(tab)    # view table of outcome proportions
```

```
## events
##   blue   red
## 0.5983 0.4017
```

Setting the Random Seed

The `set.seed` function

Before we continue, we will briefly explain the following important line of code:

```
set.seed(1986)
```

Throughout this book, we use random number generators. This implies that many of the results presented can actually change by chance, which then suggests that a frozen version of the book may show a different result than what you obtain when you try to code as shown in the book. This is actually fine since the results are random and change from time to time. However, if you want to ensure that results are exactly the same every time you run them, you can set R's random number generation seed to a specific number. Above we set it to 1986. We want to avoid using the same seed every time. A popular way to pick the seed is the year - month - day. For example, we picked 1986 on December 20, 2018: $2018 - 12 - 20 = 1986$.

You can learn more about setting the seed by looking at the documentation:

```
??set.seed
```

In the exercises, we may ask you to set the seed to assure that the results you obtain are exactly what we expect them to be.

Important note on seeds in R 3.5 and R 3.6

R was updated to version 3.6 in early 2019. In this update, the default method for setting the seed changed. This means that exercises, videos, textbook excerpts and other code you encounter online may yield a different result based on your version of R.

If you are running R 3.6, you can revert to the original seed setting behavior by adding the argument `sample.kind="Rounding"`. For example:

```
set.seed(1)
set.seed(1, sample.kind="Rounding")    # will make R 3.6 generate a seed as in R 3.5
```

Using the `sample.kind="Rounding"` argument will generate a message:

```
non-uniform 'Rounding' sampler used
```

This is not a warning or a cause for alarm - it is a confirmation that R is using the alternate seed generation method, and you should expect to receive this message in your console.

If you use R 3.6, you should always use the second form of `set.seed` in this course series (outside of DataCamp assignments). Failure to do so may result in an otherwise correct answer being rejected by the grader. In most cases where a seed is required, you will be reminded of this fact.

Using the mean Function for Probability

An important application of the mean function

In R, applying the `mean` function to a logical vector returns the proportion of elements that are TRUE. It is very common to use the `mean` function in this way to calculate probabilities and we will do so throughout the course.

Suppose you have the vector `beads`:

```
beads <- rep(c("red", "blue"), times = c(2,3))
beads
```

```
## [1] "red" "red" "blue" "blue" "blue"
```

```
# To find the probability of drawing a blue bead at random, you can run:
mean(beads == "blue")
```

```
## [1] 0.6
```

This code is broken down into steps inside R. First, R evaluates the logical statement `beads == "blue"`, which generates the vector:

```
FALSE FALSE TRUE TRUE TRUE
```

When the `mean` function is applied, R coerces the logical values to numeric values, changing TRUE to 1 and FALSE to 0:

```
0 0 1 1 1
```

The mean of the zeros and ones thus gives the proportion of TRUE values. As we have learned and will continue to see, probabilities are directly related to the proportion of events that satisfy a requirement.

Probability Distributions

The textbook for this section is available [here](#)

Key points

- The probability distribution for a variable describes the probability of observing each possible outcome.
- For discrete categorical variables, the probability distribution is defined by the proportions for each group.

Independence

The textbook section on independence, conditional probability and the multiplication rule is available [here](#)

Key points

- *Conditional probabilities* compute the probability that an event occurs given information about dependent events. For example, the probability of drawing a second king given that the first draw is a king is:

$$Pr(\text{Card 2 is a king} \mid \text{Card 1 is a king}) = 3/51$$

- If two events A and B are independent, $Pr(A \mid B) = Pr(A)$.
- To determine the probability of multiple events occurring, we use the *multiplication rule*.

Equations

The multiplication rule for independent events is:

$$Pr(A \text{ and } B \text{ and } C) = Pr(A) \times Pr(B) \times Pr(C)$$

The multiplication rule for dependent events considers the conditional probability of both events occurring:

$$Pr(A \text{ and } B) = Pr(A) \times Pr(B \mid A)$$

We can expand the multiplication rule for dependent events to more than 2 events:

$$Pr(A \text{ and } B \text{ and } C) = Pr(A) \times Pr(B \mid A) \times Pr(C \mid A \text{ and } B)$$

Assessment - Introduction to Discrete Probability (using R)

1. Probability of cyan

One ball will be drawn at random from a box containing: 3 cyan balls, 5 magenta balls, and 7 yellow balls.

What is the probability that the ball will be cyan?

```
cyan <- 3
magenta <- 5
yellow <- 7

# Assign a variable `p` as the probability of choosing a cyan ball from the box
p <- cyan/(cyan+magenta+yellow)

# Print the variable `p` to the console
p
```

```
## [1] 0.2
```

2. Probability of not cyan

One ball will be drawn at random from a box containing: 3 cyan balls, 5 magenta balls, and 7 yellow balls.

What is the probability that the ball will not be cyan?

```
# `p` is defined as the probability of choosing a cyan ball from a box containing: 3 cyan balls, 5 magenta balls, and 7 yellow balls.  
# Using variable `p`, calculate the probability of choosing any ball that is not cyan from the box  
p <- cyan/(cyan+magenta+yellow)  
prop <- 1 - p  
prop
```

```
## [1] 0.8
```

3. Sampling without replacement

Instead of taking just one draw, consider taking two draws. You take the second draw without returning the first draw to the box. We call this sampling without replacement.

What is the probability that the first draw is cyan and that the second draw is not cyan?

```
# The variable `p_1` is the probability of choosing a cyan ball from the box on the first draw.  
p_1 <- cyan / (cyan + magenta + yellow)  
  
# Assign a variable `p_2` as the probability of not choosing a cyan ball on the second draw without replacement.  
p_2 <- (magenta+yellow)/((cyan-1)+magenta+yellow)  
p_2
```

```
## [1] 0.8571429
```

```
# Calculate the probability that the first draw is cyan and the second draw is not cyan using `p_1` and `p_2`  
p_1 * p_2
```

```
## [1] 0.1714286
```

4. Sampling with replacement

Now repeat the experiment, but this time, after taking the first draw and recording the color, return it back to the box and shake the box. We call this sampling with replacement. What is the probability that the first draw is cyan and that the second draw is not cyan?

```
# The variable 'p_1' is the probability of choosing a cyan ball from the box on the first draw.  
p_1 <- cyan / (cyan + magenta + yellow)  
  
# Assign a variable 'p_2' as the probability of not choosing a cyan ball on the second draw with replacement.  
p_2 <- (magenta + yellow) / (cyan + magenta + yellow)  
  
# Calculate the probability that the first draw is cyan and the second draw is not cyan using `p_1` and `p_2`  
p_1 * p_2
```

```
## [1] 0.16
```

Combinations and Permutations

The textbook for this section is available [here](#)

Key points

- **paste** joins two strings and inserts a space in between.
- **expand.grid** gives the combinations of 2 vectors or lists.
- **permutations(n,r)** from the **gtools** package lists the different ways that **r** items can be selected from a set of **n** options when order matters.
- **combinations(n,r)** from the **gtools** package lists the different ways that **r** items can be selected from a set of **n** options when order does not matter.

Code: Introducing paste and expand.grid

```
# joining strings with paste
number <- "Three"
suit <- "Hearts"
paste(number, suit)
```

```
## [1] "Three Hearts"
```

```
# joining vectors element-wise with paste
paste(letters[1:5], as.character(1:5))
```

```
## [1] "a 1" "b 2" "c 3" "d 4" "e 5"
```

```
# generating combinations of 2 vectors with expand.grid
expand.grid(pants = c("blue", "black"), shirt = c("white", "grey", "plaid"))
```

```
##   pants shirt
## 1  blue white
## 2 black white
## 3  blue  grey
## 4 black  grey
## 5  blue plaid
## 6 black plaid
```

Code: Generating a deck of cards

```
suits <- c("Diamonds", "Clubs", "Hearts", "Spades")
numbers <- c("Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "King")
deck <- expand.grid(number = numbers, suit = suits)
deck <- paste(deck$number, deck$suit)
```

```
# probability of drawing a king
kings <- paste("King", suits)
mean(deck %in% kings)
```

```
## [1] 0.07692308
```

Code: Permutations and combinations

```
if(!require(gtools)) install.packages("gtools")
```

```
## Loading required package: gtools
```

```
library(gtools)
permutations(5,2)    # ways to choose 2 numbers in order from 1:5
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    1    3
## [3,]    1    4
## [4,]    1    5
## [5,]    2    1
## [6,]    2    3
## [7,]    2    4
## [8,]    2    5
## [9,]    3    1
## [10,]   3    2
## [11,]   3    4
## [12,]   3    5
## [13,]   4    1
## [14,]   4    2
## [15,]   4    3
## [16,]   4    5
## [17,]   5    1
## [18,]   5    2
## [19,]   5    3
## [20,]   5    4
```

```
all_phone_numbers <- permutations(10, 7, v = 0:9)
n <- nrow(all_phone_numbers)
index <- sample(n, 5)
all_phone_numbers[index,]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]    4    2    6    3    8    7    5
## [2,]    0    3    4    2    7    6    9
## [3,]    1    9    6    2    3    5    7
## [4,]    4    8    5    7    6    0    1
## [5,]    9    1    7    4    8    2    0
```

```
permutations(3,2)    # order matters
```

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



```
combinations(3,2)      # order does not matter
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    1    3
## [3,]    2    3
```

Code: Probability of drawing a second king given that one king is drawn

```
hands <- permutations(52,2, v = deck)
first_card <- hands[,1]
second_card <- hands[,2]
sum(first_card %in% kings)
```

```
## [1] 204
```

```
sum(first_card %in% kings & second_card %in% kings) / sum(first_card %in% kings)
```

```
## [1] 0.05882353
```

Code: Probability of a natural 21 in blackjack

```
aces <- paste("Ace", suits)
```

```
facecard <- c("King", "Queen", "Jack", "Ten")
facecard <- expand.grid(number = facecard, suit = suits)
facecard <- paste(facecard$number, facecard$suit)
```

```
hands <- combinations(52, 2, v=deck) # all possible hands
```

```
# probability of a natural 21 given that the ace is listed first in `combinations`
mean(hands[,1] %in% aces & hands[,2] %in% facecard)
```

```
## [1] 0.04826546
```

```
# probability of a natural 21 checking for both ace first and ace second
```

```
mean((hands[,1] %in% aces & hands[,2] %in% facecard) | (hands[,2] %in% aces & hands[,1] %in% facecard))
```

```
## [1] 0.04826546
```

Code: Monte Carlo simulation of natural 21 in blackjack

Note that your exact values will differ because the process is random and the seed is not set.

```
# code for one hand of blackjack
```

```
hand <- sample(deck, 2)
hand
```

```
## [1] "Six Diamonds" "Nine Diamonds"
```

```
# code for B=10,000 hands of blackjack
B <- 10000
results <- replicate(B, {
  hand <- sample(deck, 2)
  (hand[1] %in% aces & hand[2] %in% facecard) | (hand[2] %in% aces & hand[1] %in% facecard)
})
mean(results)
```

```
## [1] 0.046
```

The Birthday Problem

The textbook for this section is available [here](#)

Key points

- **duplicated** takes a vector and returns a vector of the same length with **TRUE** for any elements that have appeared previously in that vector.
- We can compute the probability of shared birthdays in a group of people by modeling birthdays as random draws from the numbers 1 through 365. We can then use this sampling model of birthdays to run a Monte Carlo simulation to estimate the probability of shared birthdays.

Code: The birthday problem

```
# checking for duplicated bdays in one 50 person group
n <- 50
bdays <- sample(1:365, n, replace = TRUE) # generate n random birthdays
any(duplicated(bdays)) # check if any birthdays are duplicated
```

```
## [1] FALSE
```

```
# Monte Carlo simulation with B=10000 replicates
B <- 10000
results <- replicate(B, { # returns vector of B logical values
  bdays <- sample(1:365, n, replace = TRUE)
  any(duplicated(bdays))
})
mean(results) # calculates proportion of groups with duplicated bdays
```

```
## [1] 0.9692
```

sapply

The textbook discussion of the basics of **sapply** can be found [in this textbook section](#).

The textbook discussion of **sapply** for the birthday problem can be found within [the birthday problem section](#).

Key points

- Some functions automatically apply element-wise to vectors, such as **sqrt** and ********.

- However, other functions do not operate element-wise by default. This includes functions we define ourselves.
- The function **sapply(x, f)** allows any other function **f** to be applied element-wise to the vector **x**.
- The probability of an event happening is 1 minus the probability of that event not happening:

$$Pr(event) = 1 - Pr(no\ event)$$

- We can compute the probability of shared birthdays mathematically:

$$Pr(shared\ birthdays) = 1 - Pr(no\ shared\ birthdays) = 1 - (1 \times \frac{364}{365} \times \frac{363}{365} \times \dots \times \frac{365-n+1}{365})$$

Code: Function for calculating birthday problem Monte Carlo simulations for any value of n

Note that the function body of **compute_prob** is the code that we wrote earlier. If we write this code as a function, we can use **sapply** to apply this function to several values of **n**.

```
# function to calculate probability of shared bdays across n people
compute_prob <- function(n, B = 10000) {
  same_day <- replicate(B, {
    bdays <- sample(1:365, n, replace = TRUE)
    any(duplicated(bdays))
  })
  mean(same_day)
}

n <- seq(1, 60)
```

Code: Element-wise operation over vectors and sapply

```
x <- 1:10
sqrt(x)      # sqrt operates on each element of the vector
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
## [9] 3.000000 3.162278
```

```
y <- 1:10
x*y      # * operates element-wise on both vectors
```

```
## [1] 1 4 9 16 25 36 49 64 81 100
```

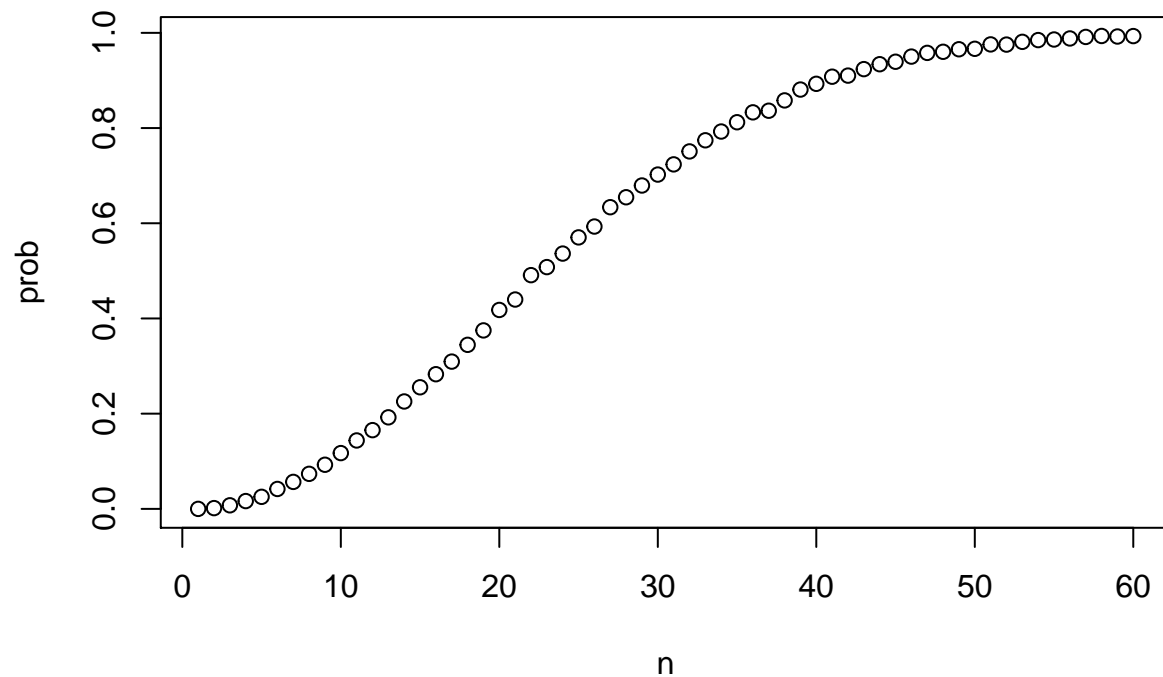
```
compute_prob(n)      # does not iterate over the vector n without sapply
```

```
## [1] 0
```

```
x <- 1:10
sapply(x, sqrt)      # this is equivalent to sqrt(x)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
## [9] 3.000000 3.162278
```

```
prob <- sapply(n, compute_prob)    # element-wise application of compute_prob to n
plot(n, prob)
```

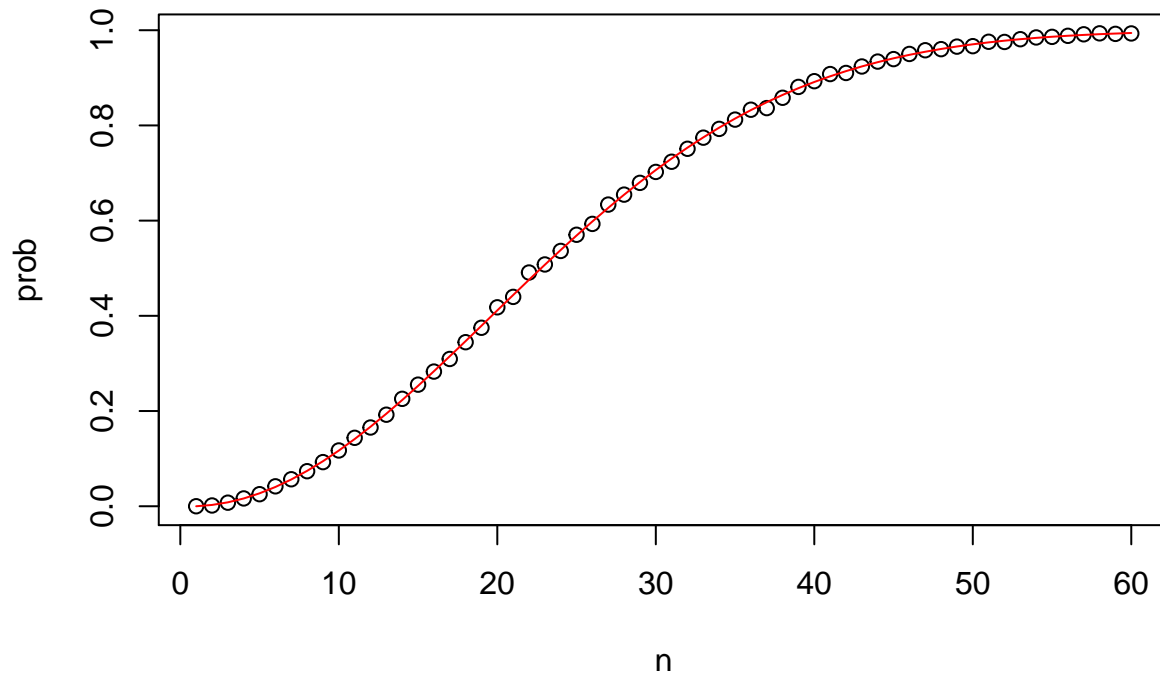


Computing birthday problem probabilities with sapply

```
# function for computing exact probability of shared birthdays for any n
exact_prob <- function(n){
  prob_unique <- seq(365, 365-n+1)/365    # vector of fractions for mult. rule
  1 - prod(prob_unique)    # calculate prob of no shared birthdays and subtract from 1
}

# applying function element-wise to vector of n values
eprob <- sapply(n, exact_prob)

# plotting Monte Carlo results and exact probabilities on same graph
plot(n, prob)    # plot Monte Carlo results
lines(n, eprob, col = "red")    # add line for exact prob
```



How many Monte Carlo Experiments are enough?

Here is a link to the [matching textbook section](#).

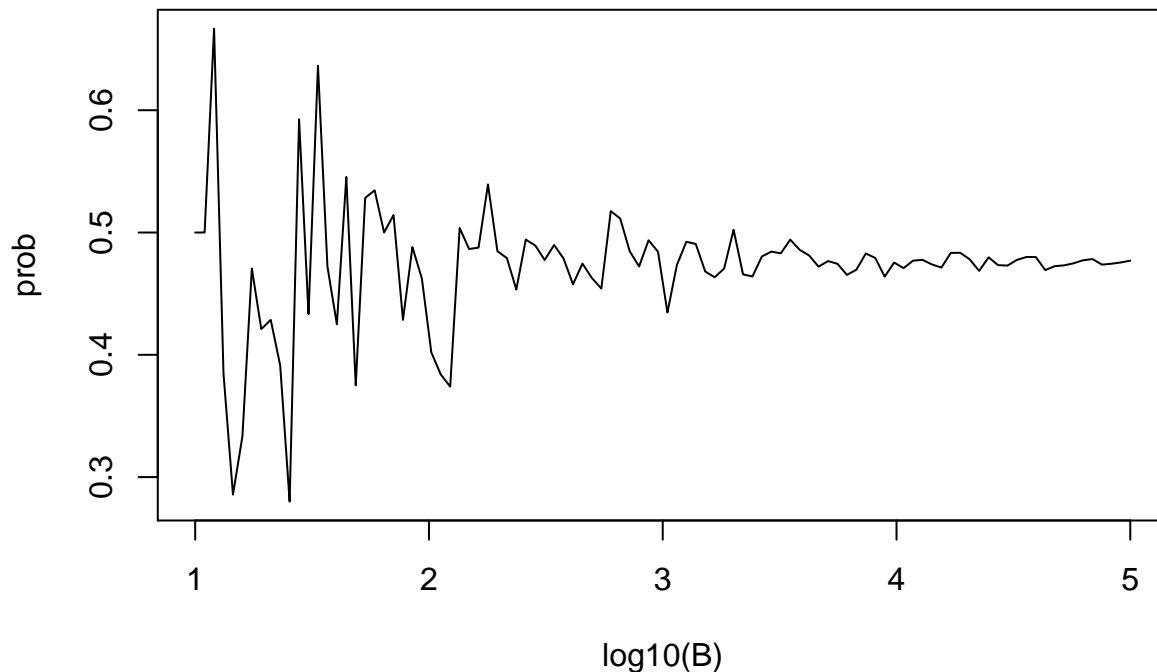
Key points

- The larger the number of Monte Carlo replicates B , the more accurate the estimate.
- Determining the appropriate size for B can require advanced statistics.
- One practical approach is to try many sizes for B and look for sizes that provide stable estimates.

Code: Estimating a practical value of B

This code runs Monte Carlo simulations to estimate the probability of shared birthdays using several B values and plots the results. When B is large enough that the estimated probability stays stable, then we have selected a useful value of B .

```
B <- 10^seq(1, 5, len = 100)      # defines vector of many B values
compute_prob <- function(B, n = 22){  # function to run Monte Carlo simulation with each B
  same_day <- replicate(B, {
    bdays <- sample(1:365, n, replace = TRUE)
    any(duplicated(bdays))
  })
  mean(same_day)
}
prob <- sapply(B, compute_prob)      # apply compute_prob to many values of B
plot(log10(B), prob, type = "l")    # plot a line graph of estimates
```



Assessment - Combinations and Permutations

1. Imagine you draw two balls from a box containing colored balls. You either replace the first ball before you draw the second or you leave the first ball out of the box when you draw the second ball.

Under which situation are the two draws independent of one another?

Remember that two events A and B are independent if:

$$Pr(A \text{ and } B) = Pr(A)Pr(B)$$

- ☐ A. You don't replace the first ball before drawing the next.
- ☒ B. You do replace the first ball before drawing the next.
- ☐ C. Neither situation describes independent events.
- ☐ D. Both situations describe independent events.

2. Say you've drawn 5 balls from a box that has 3 cyan balls, 5 magenta balls, and 7 yellow balls, with replacement, and all have been yellow.

What is the probability that the next one is yellow?

```
# Assign the variable 'p_yellow' as the probability that a yellow ball is drawn from the box.
p_yellow <- yellow / (cyan + magenta + yellow)
```

```
# Using the variable 'p_yellow', calculate the probability of drawing a yellow ball on the sixth draw.
p_yellow
```

```
## [1] 0.4666667
```

3. If you roll a 6-sided die once, what is the probability of not seeing a 6? If you roll a 6-sided die six times, what is the probability of not seeing a 6 on any roll?

```
# Assign the variable 'p_no6' as the probability of not seeing a 6 on a single roll.
p_no6 <- 5/6
```

```
# Calculate the probability of not seeing a 6 on six rolls using `p_no6`. Print your result to the console.
p_no6*p_no6*p_no6*p_no6*p_no6*p_no6
```

```
## [1] 0.334898
```

- Two teams, say the Celtics and the Cavs, are playing a seven game series. The Cavs are a better team and have a 60% chance of winning each game.

What is the probability that the Celtics win at least one game? Remember that the Celtics must win one of the first four games, or the series will be over!

```
# Assign the variable `p_cavs_win4` as the probability that the Cavs will win the first four games of the series.
p_cavs_win4 <- (3/5)*(3/5)*(3/5)*(3/5)
```

```
# Using the variable `p_cavs_win4`, calculate the probability that the Celtics win at least one game in the series.
1-p_cavs_win4
```

```
## [1] 0.8704
```

- Create a Monte Carlo simulation to confirm your answer to the previous problem by estimating how frequently the Celtics win at least 1 of 4 games. Use `B <- 10000` simulations.

The provided sample code simulates a single series of four random games, `simulated_games`

```
# This line of example code simulates four independent random games where the Celtics either lose or win.
simulated_games <- sample(c("lose","win"), 4, replace = TRUE, prob = c(0.6, 0.4))
```

```
# The variable 'B' specifies the number of times we want the simulation to run. Let's run the Monte Carlo simulation 10,000 times.
B <- 10000
```

```
# Use the `set.seed` function to make sure your answer matches the expected result after random sampling.
set.seed(1)
```

```
# Create an object called `celtic_wins` that replicates two steps for B iterations: (1) generating a random series of four games, and (2) checking if the Celtics won at least one game.
celtic_wins <- replicate(B, {
  simulated_games <- sample(c("lose","win"), 4, replace = TRUE, prob = c(0.6, 0.4))
  any(simulated_games=="win")}
)
```

```
# Calculate the frequency out of B iterations that the Celtics won at least one game. Print your answer to the console.
mean(celtic_wins)
```

```
## [1] 0.8757
```

The Addition rule

Here is a link to the textbook section on the [addition rule](#).

Key points

- The addition rule states that the probability of event A or event B happening is the probability of event A plus the probability of event B minus the probability of both events A and B happening together.

$$Pr(A \text{ or } B) = Pr(A) + Pr(B) - Pr(A \text{ and } B)$$

Note that $(A \text{ or } B)$ is equivalent to $(A \mid B)$.

Example: The addition rule for a natural 21 in blackjack

We apply the addition rule where A = drawing an ace then a facecard and B = drawing a facecard then an ace. Note that in this case, both events A and B cannot happen at the same time, so $Pr(A \text{ and } B) = 0$.

$$Pr(\text{ace then facecard}) = \frac{4}{52} \times \frac{16}{51}$$

$$Pr(\text{facecard then ace}) = \frac{16}{52} \times \frac{4}{51}$$

$$Pr(\text{ace then facecard} \mid \text{facecard then ace}) = \frac{4}{52} \times \frac{16}{51} + \frac{16}{52} \times \frac{4}{51} = 0.0483$$

The Monty Hall Problem

Here is the textbook section on the [Monty Hall Problem](#).

Key points

- Monte Carlo simulations can be used to simulate random outcomes, which makes them useful when exploring ambiguous or less intuitive problems like the Monty Hall problem.
- In the Monty Hall problem, contestants choose one of three doors that may contain a prize. Then, one of the doors that was not chosen by the contestant and does not contain a prize is revealed. The contestant can then choose whether to stick with the original choice or switch to the remaining unopened door.
- Although it may seem intuitively like the contestant has a 1 in 2 chance of winning regardless of whether they stick or switch, Monte Carlo simulations demonstrate that the actual probability of winning is 1 in 3 with the stick strategy and 2 in 3 with the switch strategy.
- For more on the Monty Hall problem, you can [watch a detailed explanation here](#) or [read an explanation here](#).

Code: Monte Carlo simulation of stick strategy

```
B <- 10000
stick <- replicate(B, {
  doors <- as.character(1:3)
  prize <- sample(c("car", "goat", "goat")) # puts prizes in random order
  prize_door <- doors[prize == "car"] # note which door has prize
  my_pick <- sample(doors, 1) # note which door is chosen
  show <- sample(doors[!doors %in% c(my_pick, prize_door)], 1) # open door with no prize that isn't c
  stick <- my_pick # stick with original door
  stick == prize_door # test whether the original door has the prize
})
mean(stick) # probability of choosing prize door when sticking
```

```
## [1] 0.3388
```

Code: Monte Carlo simulation of switch strategy


```

switch <- replicate(B, {
  doors <- as.character(1:3)
  prize <- sample(c("car", "goat", "goat")) # puts prizes in random order
  prize_door <- doors[prize == "car"] # note which door has prize
  my_pick <- sample(door, 1) # note which door is chosen first
  show <- sample(door[!door %in% c(my_pick, prize_door)], 1) # open door with no prize that isn't
  switch <- doors[!door %in% c(my_pick, show)] # switch to the door that wasn't chosen first or opened
  switch == prize_door # test whether the switched door has the prize
})
mean(switch) # probability of choosing prize door when switching

```

```
## [1] 0.6708
```

Assessment - The Addition Rule and Monty Hall

1. Two teams, say the Cavs and the Warriors, are playing a seven game championship series. The first to win four games wins the series. The teams are equally good, so they each have a 50-50 chance of winning each game.

If the Cavs lose the first game, what is the probability that they win the series?

```

# Assign a variable 'n' as the number of remaining games.
n <- 6

# Assign a variable 'outcomes' as a vector of possible game outcomes, where 0 indicates a loss and 1 indicates a win
outcomes <- (0:1)

# Assign a variable 'l' to a list of all possible outcomes in all remaining games. Use the 'rep' function
l <- rep(list(outcomes), n)

# Create a data frame named 'possibilities' that contains all combinations of possible outcomes for the remaining games
possibilities <- expand.grid(l)

# Create a vector named 'results' that indicates whether each row in the data frame 'possibilities' contains a win for the Cavs
results <- rowSums(possibilities) >= 4

# Calculate the proportion of 'results' in which the Cavs win the series. Print the outcome to the console
mean(results)

```

```
## [1] 0.34375
```

2. Confirm the results of the previous question with a Monte Carlo simulation to estimate the probability of the Cavs winning the series after losing the first game.

```

# The variable 'B' specifies the number of times we want the simulation to run. Let's run the Monte Carlo simulation 10,000 times
B <- 10000

# Use the 'set.seed' function to make sure your answer matches the expected result after random sampling
set.seed(1)

# Create an object called 'results' that replicates for 'B' iterations a simulated series and determines the winner

```

```

results <- replicate(B, {
  cavs_wins <- sample(c(0,1), 6, replace = TRUE)
  sum(cavs_wins)>=4 }
)

# Calculate the frequency out of `B` iterations that the Cavs won at least four games in the remainder
mean(results)

```

```
## [1] 0.3371
```

- Two teams, A and B , are playing a seven series game series. Team A is better than team B and has a $p > 0.5$ chance of winning each game.

```

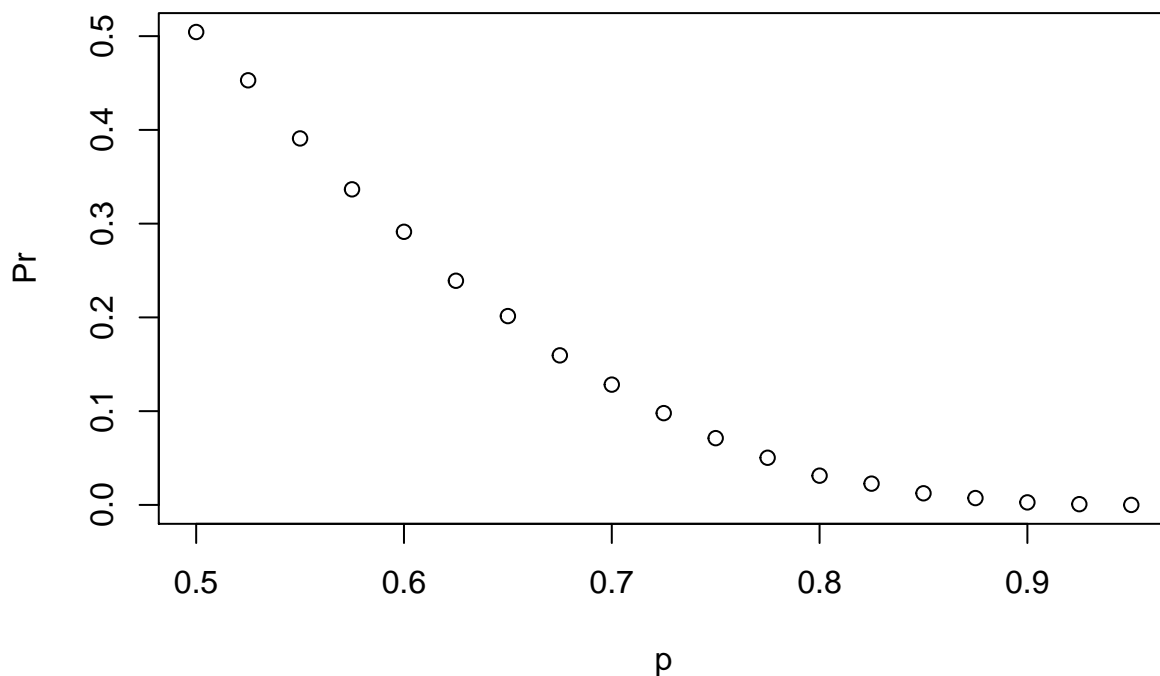
# Let's assign the variable 'p' as the vector of probabilities that team A will win.
p <- seq(0.5, 0.95, 0.025)

# Given a value 'p', the probability of winning the series for the underdog team B can be computed with
prob_win <- function(p){
  B <- 10000
  result <- replicate(B, {
    b_win <- sample(c(1,0), 7, replace = TRUE, prob = c(1-p, p))
    sum(b_win)>=4
  })
  mean(result)
}

# Apply the 'prob_win' function across the vector of probabilities that team A will win to determine th
Pr <- sapply(p, prob_win)

# Plot the probability 'p' on the x-axis and 'Pr' on the y-axis.
plot(p,Pr)

```



- Repeat the previous exercise, but now keep the probability that team A wins fixed at $p \leftarrow 0.75$ and compute the probability for different series lengths.

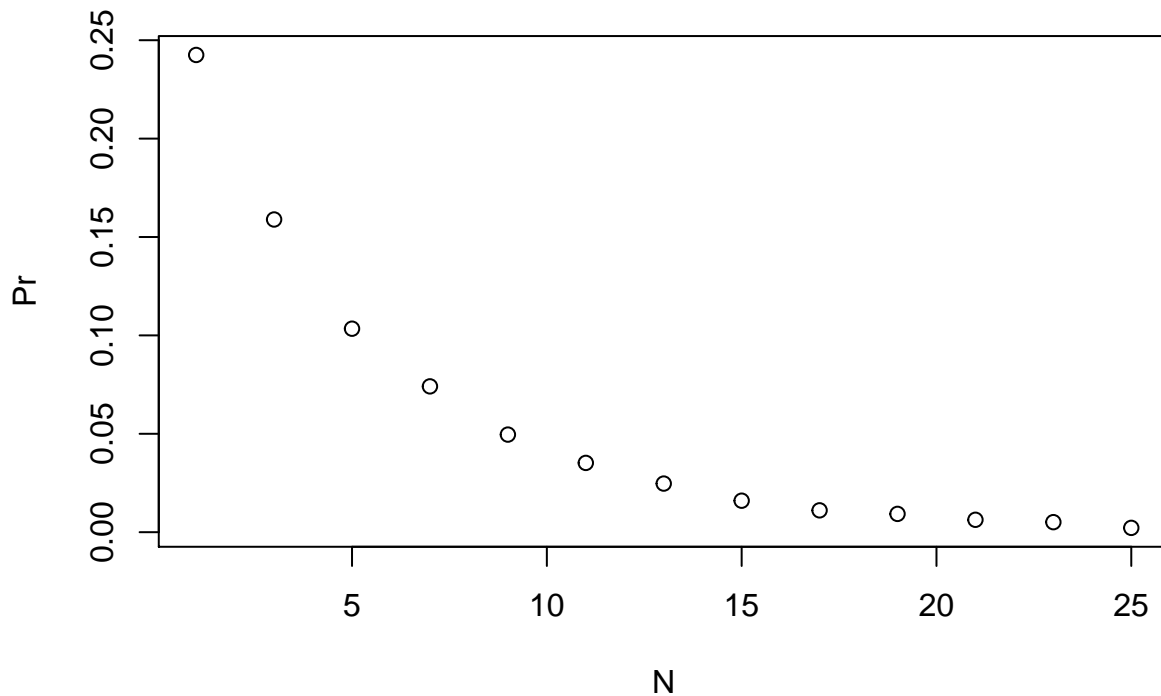
For example, wins in best of 1 game, 3 games, 5 games, and so on through a series that lasts 25 games.

```
# Given a value 'p', the probability of winning the series for the underdog team B can be computed with
prob_win <- function(N, p=0.75){
  B <- 10000
  result <- replicate(B, {
    b_win <- sample(c(1,0), N, replace = TRUE, prob = c(1-p, p))
    sum(b_win)>=(N+1)/2
  })
  mean(result)
}

# Assign the variable 'N' as the vector of series lengths. Use only odd numbers ranging from 1 to 25 games
N <- seq(1, 25, 2)

# Apply the 'prob_win' function across the vector of series lengths to determine the probability that t
Pr <- sapply(N, prob_win)

# Plot the number of games in the series 'N' on the x-axis and 'Pr' on the y-axis.
plot(N,Pr)
```



Assessment - Olympic Running

```
if(!require(tidyverse)) install.packages("tidyverse")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages -----

## v ggplot2 3.3.2    v purrr  0.3.4
## v tibble  3.0.3    v dplyr  1.0.1
## v tidyr   1.1.1    v stringr 1.4.0
## v readr   1.3.1    v forcats 0.5.0

## -- Conflicts -----
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidyverse)
options(digits = 3)    # report 3 significant digits
```

1. In the 200m dash finals in the Olympics, 8 runners compete for 3 medals (order matters). In the 2012 Olympics, 3 of the 8 runners were from Jamaica and the other 5 were from different countries. The three medals were all won by Jamaica (Usain Bolt, Yohan Blake, and Warren Weir).

Use the information above to help you answer the following four questions.

- 1a. How many different ways can the 3 medals be distributed across 8 runners?

```
medals <- permutations(8,3)
nrow(medals)
```

```
## [1] 336
```

- 1b. How many different ways can the three medals be distributed among the 3 runners from Jamaica?

```
jamaica <- permutations(3,3)
nrow(jamaica)
```

```
## [1] 6
```

- 1c. What is the probability that all 3 medals are won by Jamaica?

```
nrow(jamaica)/nrow(medals)
```

```
## [1] 0.0179
```

- 1d. Run a Monte Carlo simulation on this vector representing the countries of the 8 runners in this race:

```
runners <- c("Jamaica", "Jamaica", "Jamaica", "USA", "Ecuador", "Netherlands", "France", "South Africa")
```

For each iteration of the Monte Carlo simulation, within a `replicate` loop, select 3 runners representing the 3 medalists and check whether they are all from Jamaica. Repeat this simulation 10,000 times. Set the seed to 1 before running the loop.

Calculate the probability that all the runners are from Jamaica.

```

set.seed(1)
runners <- c("Jamaica", "Jamaica", "Jamaica", "USA", "Ecuador", "Netherlands", "France", "South Africa")
B <- 10000
all_jamaica <- replicate(B, {
  results <- sample(runners, 3)
  all(results == "Jamaica")
})
mean(all_jamaica)

```

```
## [1] 0.0174
```

Assessment - Restaurant Management

2: Use the information below to answer the following five questions.

A restaurant manager wants to advertise that his lunch special offers enough choices to eat different meals every day of the year. He doesn't think his current special actually allows that number of choices, but wants to change his special if needed to allow at least 365 choices.

A meal at the restaurant includes 1 entree, 2 sides, and 1 drink. He currently offers a choice of 1 entree from a list of 6 options, a choice of 2 different sides from a list of 6 options, and a choice of 1 drink from a list of 2 options.

2a. How many meal combinations are possible with the current menu?

```
6 * nrow(combinations(6,2)) * 2
```

```
## [1] 180
```

2b. The manager has one additional drink he could add to the special.

How many combinations are possible if he expands his original special to 3 drink options?

```
6 * nrow(combinations(6,2)) * 3
```

```
## [1] 270
```

2c. The manager decides to add the third drink but needs to expand the number of options. The manager would prefer not to change his menu further and wants to know if he can meet his goal by letting customers choose more sides.

How many meal combinations are there if customers can choose from 6 entrees, 3 drinks, and select 3 sides from the current 6 options?

```
6 * nrow(combinations(6,3)) * 3
```

```
## [1] 360
```

2d. The manager is concerned that customers may not want 3 sides with their meal. He is willing to increase the number of entree choices instead, but if he adds too many expensive options it could eat into profits. He wants to know how many entree choices he would have to offer in order to meet his goal.

- Write a function that takes a number of entree choices and returns the number of meal combinations possible given that number of entree options, 3 drink choices, and a selection of 2 sides from 6 options.
- Use `sapply` to apply the function to entree option counts ranging from 1 to 12.

What is the minimum number of entree options required in order to generate more than 365 combinations?

```
entree_choices <- function(x){
  x * nrow(combinations(6,2)) * 3
}

combos <- sapply(1:12, entree_choices)

data.frame(entrees = 1:12, combos = combos) %>%
  filter(combos > 365) %>%
  min(.$entrees)
```

```
## [1] 9
```

2e. The manager isn't sure he can afford to put that many entree choices on the lunch menu and thinks it would be cheaper for him to expand the number of sides. He wants to know how many sides he would have to offer to meet his goal of at least 365 combinations.

- Write a function that takes a number of side choices and returns the number of meal combinations possible given 6 entree choices, 3 drink choices, and a selection of 2 sides from the specified number of side choices.
- Use `sapply` to apply the function to side counts ranging from 2 to 12.

What is the minimum number of side options required in order to generate more than 365 combinations?

```
side_choices <- function(x){
  6 * nrow(combinations(x, 2)) * 3
}

combos <- sapply(2:12, side_choices)

data.frame(sides = 2:12, combos = combos) %>%
  filter(combos > 365) %>%
  min(.$sides)
```

```
## [1] 7
```

Assessment - Esophageal cancer and alcohol/tobacco use

3., 4., 5. and 6. Case-control studies help determine whether certain exposures are associated with outcomes such as developing cancer. The built-in dataset **esoph** contains data from a case-control study in France comparing people with esophageal cancer (cases, counted in **ncases**) to people without esophageal cancer (controls, counted in **ncontrols**) that are carefully matched on a variety of demographic and medical characteristics. The study compares alcohol intake in grams per day (**alcgp**) and tobacco intake in grams per day (**tobgp**) across cases and controls grouped by age range (**agegp**).

The dataset is available in base R and can be called with the variable name **esoph**:

```
head(esoph)
```

```
##   agegp   alcgp   tobgp ncases ncontrols
## 1 25-34 0-39g/day 0-9g/day     0         40
## 2 25-34 0-39g/day 10-19     0         10
## 3 25-34 0-39g/day 20-29     0          6
## 4 25-34 0-39g/day 30+       0          5
## 5 25-34 40-79 0-9g/day     0         27
## 6 25-34 40-79 10-19       0          7
```

You will be using this dataset to answer the following four multi-part questions (Questions 3-6).

You may wish to use the tidyverse package.

The following three parts have you explore some basic characteristics of the dataset.

Each row contains one group of the experiment. Each group has a different combination of age, alcohol consumption, and tobacco consumption. The number of cancer cases and number of controls (individuals without cancer) are reported for each group.

3a. How many groups are in the study?

```
nrow(esoph)
```

```
## [1] 88
```

3b. How many cases are there?

Save this value as `all_cases` for later problems.

```
all_cases <- sum(esoph$ncases)
all_cases
```

```
## [1] 200
```

3c. How many controls are there?

Save this value as `all_controls` for later problems. Remember from the instructions that controls are individuals without cancer.

```
all_controls <- sum(esoph$ncontrols)
all_controls
```

```
## [1] 975
```

4a. What is the probability that a subject in the highest alcohol consumption group is a cancer case?

Remember that the total number of individuals in the study includes people with cancer (cases) and people without cancer (controls), so you must add both values together to get the denominator for your probability.

```
esoph %>%
  filter(alcgp == "120+") %>%
  summarize(ncases = sum(ncases), ncontrols = sum(ncontrols)) %>%
  mutate(p_case = ncases / (ncases + ncontrols)) %>%
  pull(p_case)
```

```
## [1] 0.402
```

4b. What is the probability that a subject in the lowest alcohol consumption group is a cancer case?

```
esoph %>%  
  filter(alcgp == "0-39g/day") %>%  
  summarize(ncases = sum(ncases), ncontrols = sum(ncontrols)) %>%  
  mutate(p_case = ncases / (ncases + ncontrols)) %>%  
  pull(p_case)
```

```
## [1] 0.0653
```

4c. Given that a person is a case, what is the probability that they smoke 10g or more a day?

```
tob_cases <- esoph %>%  
  filter(tobgp != "0-9g/day") %>%  
  pull(ncases) %>%  
  sum()  
  
tob_cases/all_cases
```

```
## [1] 0.61
```

4d. Given that a person is a control, what is the probability that they smoke 10g or more a day?

```
tob_controls <- esoph %>%  
  filter(tobgp != "0-9g/day") %>%  
  pull(ncontrols) %>%  
  sum()  
  
tob_controls/all_controls
```

```
## [1] 0.462
```

5a. For cases, what is the probability of being in the highest alcohol group?

```
high_alc_cases <- esoph %>%  
  filter(alcgp == "120+") %>%  
  pull(ncases) %>%  
  sum()  
  
p_case_high_alc <- high_alc_cases/all_cases  
p_case_high_alc
```

```
## [1] 0.225
```

5b. For cases, what is the probability of being in the highest tobacco group?


```
high_tob_cases <- esoph %>%
  filter(tobgp == "30+") %>%
  pull(ncases) %>%
  sum()

p_case_high_tob <- high_tob_cases/all_cases
p_case_high_tob
```

```
## [1] 0.155
```

5c. For cases, what is the probability of being in the highest alcohol group **and** the highest tobacco group?

```
high_alc_tob_cases <- esoph %>%
  filter(alcgp == "120+" & tobgp == "30+") %>%
  pull(ncases) %>%
  sum()

p_case_high_alc_tob <- high_alc_tob_cases/all_cases
p_case_high_alc_tob
```

```
## [1] 0.05
```

5d. For cases, what is the probability of being in the highest alcohol group **or** the highest tobacco group?

```
p_case_either_highest <- p_case_high_alc + p_case_high_tob - p_case_high_alc_tob
p_case_either_highest
```

```
## [1] 0.33
```

6a. For controls, what is the probability of being in the highest alcohol group?

```
high_alc_controls <- esoph %>%
  filter(alcgp == "120+") %>%
  pull(ncontrols) %>%
  sum()

p_control_high_alc <- high_alc_controls/all_controls
p_control_high_alc
```

```
## [1] 0.0687
```

6b. How many times more likely are cases than controls to be in the highest alcohol group?

```
p_case_high_alc/p_control_high_alc
```

```
## [1] 3.27
```

6c. For controls, what is the probability of being in the highest tobacco group?

```
high_tob_controls <- esoph %>%
  filter(tobgp == "30+") %>%
  pull(ncontrols) %>%
  sum()

p_control_high_tob <- high_tob_controls/all_controls
p_control_high_tob
```

```
## [1] 0.0841
```

6d. For controls, what is the probability of being in the highest alcohol group **and** the highest tobacco group?

```
high_alc_tob_controls <- esoph %>%
  filter(alcgp == "120+" & tobgp == "30+") %>%
  pull(ncontrols) %>%
  sum()

p_control_high_alc_tob <- high_alc_tob_controls/all_controls
p_control_high_alc_tob
```

```
## [1] 0.0133
```

6e. For controls, what is the probability of being in the highest alcohol group **or** the highest tobacco group?

```
p_control_either_highest <- p_control_high_alc + p_control_high_tob - p_control_high_alc_tob
p_control_either_highest
```

```
## [1] 0.139
```

6f. How many times more likely are cases than controls to be in the highest alcohol group or the highest tobacco group?

```
p_case_either_highest/p_control_either_highest
```

```
## [1] 2.37
```

Section 2 Overview

Section 2 introduces you to Continuous Probability.

After completing Section 2, you will:

- understand the differences between calculating probabilities for discrete and continuous data.
- be able to use cumulative distribution functions to assign probabilities to intervals when dealing with continuous data.
- be able to use R to generate normally distributed outcomes for use in Monte Carlo simulations.
- know some of the useful theoretical continuous distributions in addition to the normal distribution, such as the student-t, chi-squared, exponential, gamma, beta, and beta-binomial distributions.

Continuous Probability

The textbook for this section is available [here](#)

The previous discussion of CDF is from the Data Visualization course. Here is the [textbook section on the CDF](#).

Key points

- The *cumulative distribution function (CDF)* is a distribution function for continuous data x that reports the proportion of the data below a for all values of a :

$$F(a) = Pr(a) = Pr(x \leq a)$$

- The CDF is the *probability distribution function* for continuous variables. For example, to determine the probability that a male student is taller than 70.5 inches given a vector of male heights x , we can use the CDF:

$$Pr(x > 70.5) = 1 - Pr(x \leq 70.5) = 1 - F(70.5)$$

- The probability that an observation is in between two values a, b is $F(b) - F(a)$.

Code: Cumulative distribution function

Define **x** as male heights from the **dslabs** dataset:

```
if(!require(dslabs)) install.packages("dslabs")

## Loading required package: dslabs

library(dslabs)
data(heights)
x <- heights %>% filter(sex=="Male") %>% pull(height)
```

Given a vector **x**, we can define a function for computing the CDF of **x** using:

```
F <- function(a) mean(x <= a)

1 - F(70)    # probability of male taller than 70 inches
```

```
## [1] 0.377
```

Theoretical Distribution

The textbook for this section is available [here](#)

Key points

- **pnorm(a, avg, s)** gives the value of the cumulative distribution function $F(a)$ for the normal distribution defined by average **avg** and standard deviation **s**.

- We say that a random quantity is normally distributed with average **avg** and standard deviation **s** if the approximation **pnorm(a, avg, s)** holds for all values of **a**.
- If we are willing to use the normal approximation for height, we can estimate the distribution simply from the mean and standard deviation of our values.
- If we treat the height data as discrete rather than categorical, we see that the data are not very useful because integer values are more common than expected due to rounding. This is called *discretization*.
- With rounded data, the normal approximation is particularly useful when computing probabilities of intervals of length 1 that include exactly one integer.

Code: Using **pnorm** to calculate probabilities

Given male heights **x**:

```
x <- heights %>% filter(sex=="Male") %>% pull(height)
```

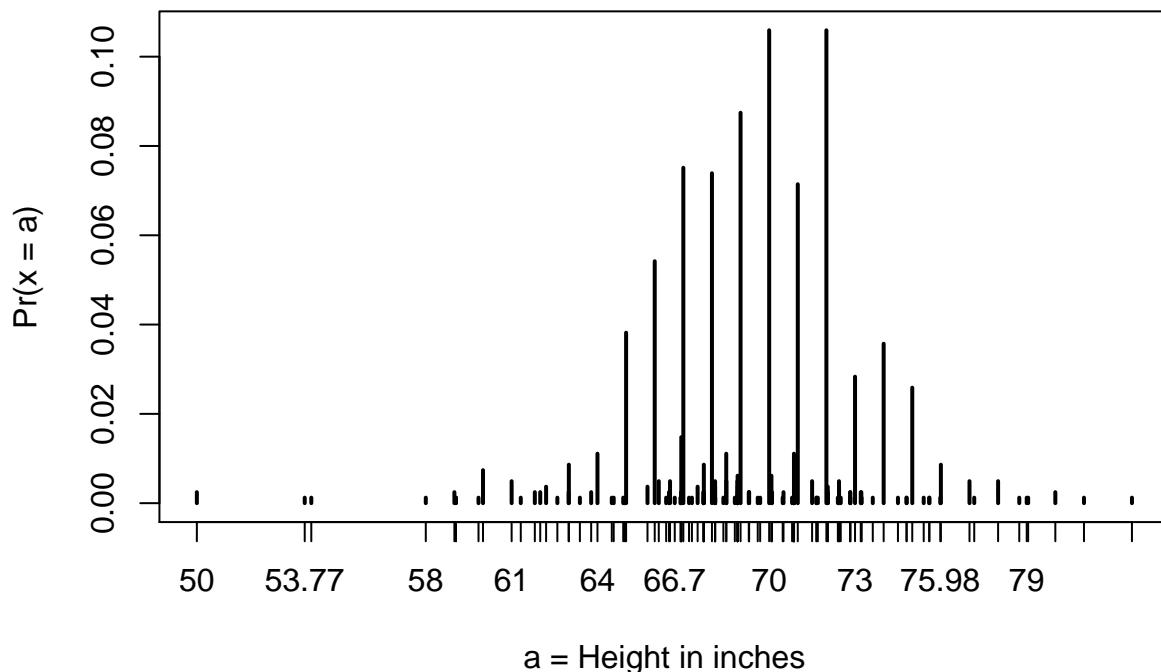
We can estimate the probability that a male is taller than 70.5 inches using:

```
1 - pnorm(70.5, mean(x), sd(x))
```

```
## [1] 0.371
```

Code: Discretization and the normal approximation

```
# plot distribution of exact heights in data
plot(prop.table(table(x)), xlab = "a = Height in inches", ylab = "Pr(x = a)")
```



```
# probabilities in actual data over length 1 ranges containing an integer
mean(x <= 68.5) - mean(x <= 67.5)
```

```
## [1] 0.115
```

```

mean(x <= 69.5) - mean(x <= 68.5)

## [1] 0.119

mean(x <= 70.5) - mean(x <= 69.5)

## [1] 0.122

# probabilities in normal approximation match well
pnorm(68.5, mean(x), sd(x)) - pnorm(67.5, mean(x), sd(x))

## [1] 0.103

pnorm(69.5, mean(x), sd(x)) - pnorm(68.5, mean(x), sd(x))

## [1] 0.11

pnorm(70.5, mean(x), sd(x)) - pnorm(69.5, mean(x), sd(x))

## [1] 0.108

# probabilities in actual data over other ranges don't match normal approx as well
mean(x <= 70.9) - mean(x <= 70.1)

## [1] 0.0222

pnorm(70.9, mean(x), sd(x)) - pnorm(70.1, mean(x), sd(x))

## [1] 0.0836

```

Probability Density

The textbook for this section is available [here](#)

Key points

- The probability of a single value is not defined for a continuous distribution.
- The quantity with the most similar interpretation to the probability of a single value is the probability density function $f(x)$.
- The probability density $f(x)$ is defined such that the integral of $f(x)$ over a range gives the CDF of that range.

$$F(a) = Pr(X \leq a) = \int_{-\infty}^a f(x)dx$$

- In R, the probability density function for the normal distribution is given by **dnorm**. We will see uses of **dnorm** in the future.
- Note that **dnorm** gives the density function and **pnorm** gives the distribution function, which is the integral of the density function.

Monte Carlo simulations

The textbook for this section is available [here](#)

Key points

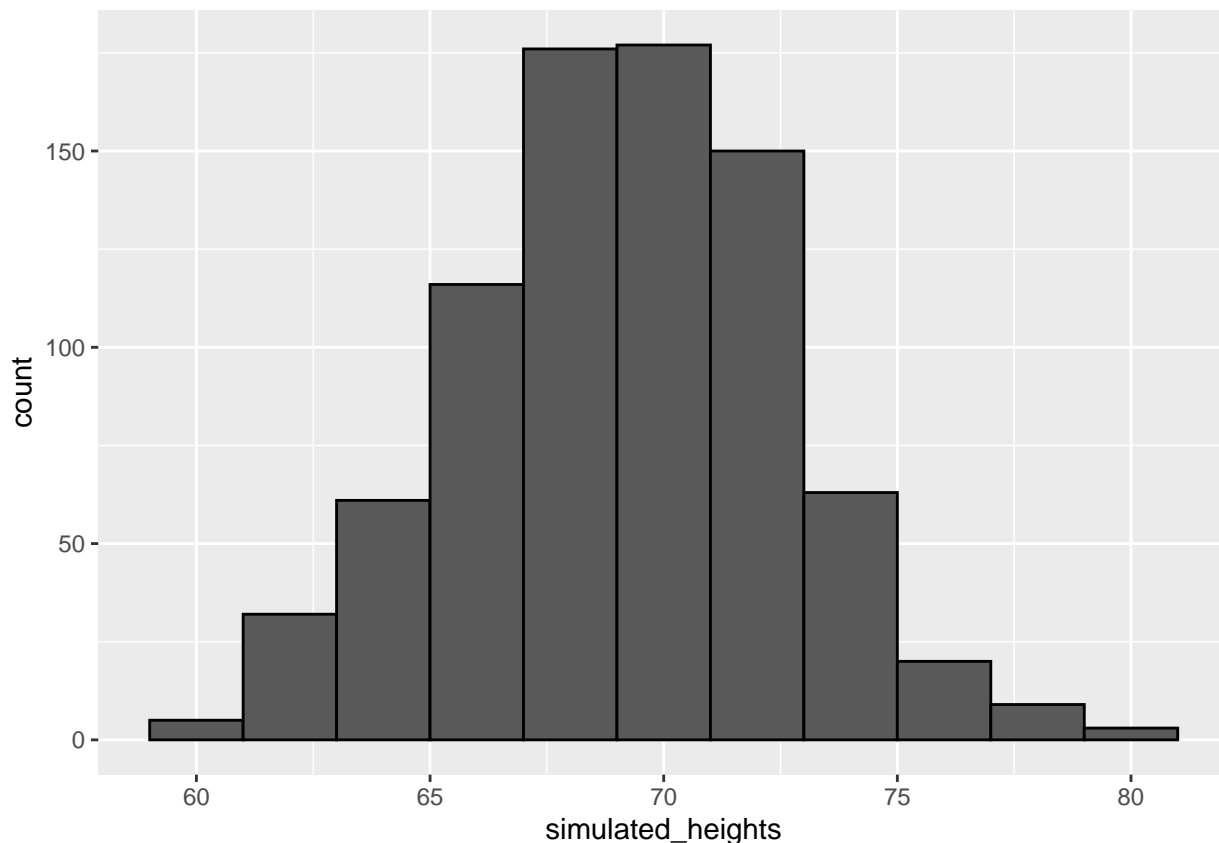
- `rnorm(n, avg, s)` generates **n** random numbers from the normal distribution with average **avg** and standard deviation **s**.
- By generating random numbers from the normal distribution, we can simulate height data with similar properties to our dataset. Here we generate simulated height data using the normal distribution.

Code: Generating normally distributed random numbers for Monte Carlo simulations

```
# define x as male heights from dslabs data
library(tidyverse)
library(dslabs)
data(heights)
x <- heights %>% filter(sex=="Male") %>% pull(height)

# generate simulated height data using normal distribution - both datasets should have n observations
n <- length(x)
avg <- mean(x)
s <- sd(x)
simulated_heights <- rnorm(n, avg, s)

# plot distribution of simulated_heights
data.frame(simulated_heights = simulated_heights) %>%
  ggplot(aes(simulated_heights)) +
  geom_histogram(color="black", binwidth = 2)
```



Code: Monte Carlo simulation of probability of tallest person being over 7 feet

```
B <- 10000
tallest <- replicate(B, {
  simulated_data <- rnorm(800, avg, s) # generate 800 normally distributed random heights
  max(simulated_data) # determine the tallest height
})
mean(tallest >= 7*12) # proportion of times that tallest person exceeded 7 feet (84 inches)
```

[1] 0.0209

Other Continuous Distributions

The textbook for this section is available [here](#)

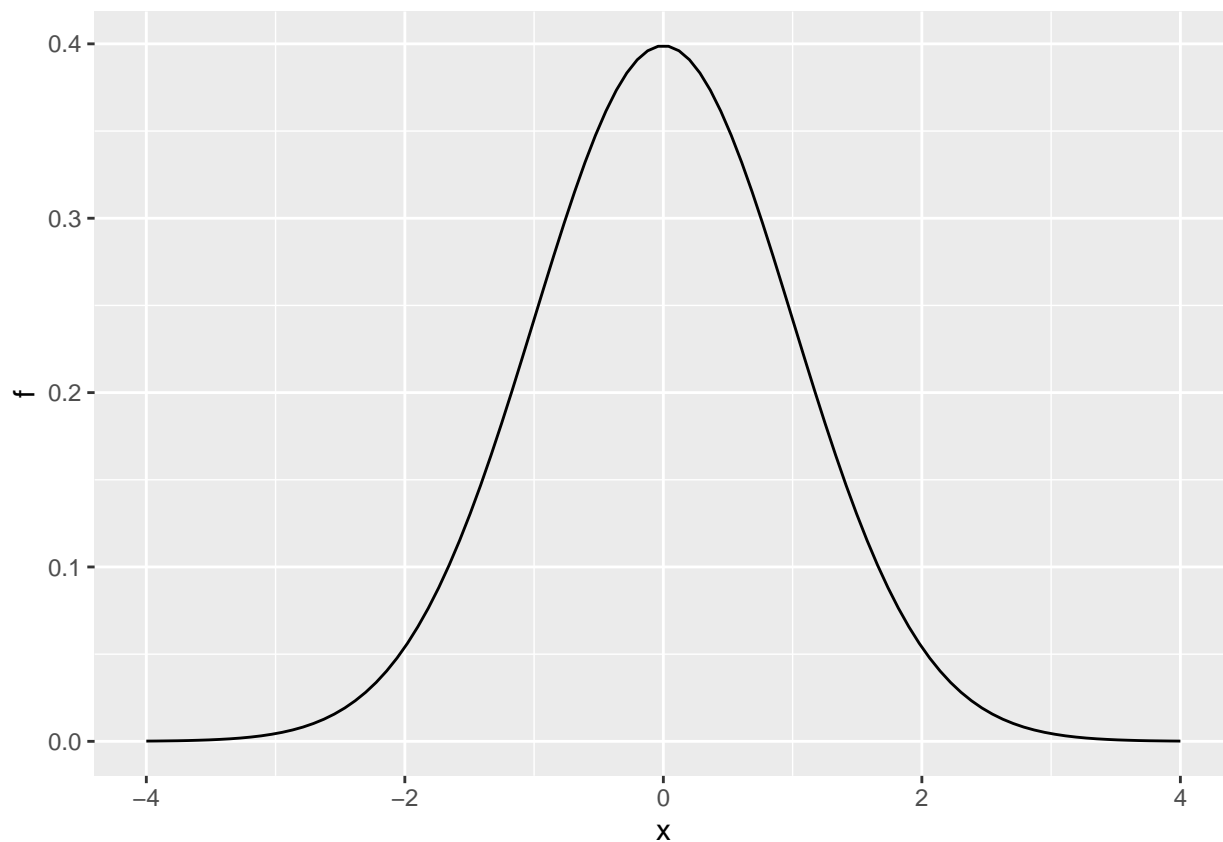
Key points

- You may encounter other continuous distributions (Student t, chi-squared, exponential, gamma, beta, etc.).
- R provides functions for density (**d**), quantile (**q**), probability distribution (**p**) and random number generation (**r**) for many of these distributions.
- Each distribution has a matching abbreviation (for example, **norm** or **t**) that is paired with the related function abbreviations (**d**, **p**, **q**, **r**) to create appropriate functions.
- For example, use **rt** to generate random numbers for a Monte Carlo simulation using the Student t distribution.

Code: Plotting the normal distribution with dnorm

Use `d` to plot the density function of a continuous distribution. Here is the density function for the normal distribution (abbreviation `norm`):

```
x <- seq(-4, 4, length.out = 100)
data.frame(x, f = dnorm(x)) %>%
  ggplot(aes(x,f)) +
  geom_line()
```



Assessment - Continuous Probability

1. Assume the distribution of female heights is approximated by a normal distribution with a mean of 64 inches and a standard deviation of 3 inches.

If we pick a female at random, what is the probability that she is 5 feet or shorter?

```
# Assign a variable 'female_avg' as the average female height.
female_avg <- 64
```

```
# Assign a variable 'female_sd' as the standard deviation for female heights.
female_sd <- 3
```

```
# Using variables 'female_avg' and 'female_sd', calculate the probability that a randomly selected female is 5 feet or shorter.
pnorm((5*12), female_avg, female_sd)
```



```
## [1] 0.0912
```

2. Assume the distribution of female heights is approximated by a normal distribution with a mean of 64 inches and a standard deviation of 3 inches.

If we pick a female at random, what is the probability that she is 6 feet or taller?

```
# Assign a variable 'female_avg' as the average female height.
female_avg <- 64

# Assign a variable 'female_sd' as the standard deviation for female heights.
female_sd <- 3

# Using variables 'female_avg' and 'female_sd', calculate the probability that a randomly selected female is 6 feet or taller.
1-pnorm((6*12), female_avg, female_sd)
```

```
## [1] 0.00383
```

3. Assume the distribution of female heights is approximated by a normal distribution with a mean of 64 inches and a standard deviation of 3 inches.

If we pick a female at random, what is the probability that she is between 61 and 67 inches?

```
# Assign a variable 'female_avg' as the average female height.
female_avg <- 64

# Assign a variable 'female_sd' as the standard deviation for female heights.
female_sd <- 3

# Using variables 'female_avg' and 'female_sd', calculate the probability that a randomly selected female is between 61 and 67 inches.
pnorm(67, female_avg, female_sd) - pnorm(61, female_avg, female_sd)
```

```
## [1] 0.683
```

4. Repeat the previous exercise, but convert everything to centimeters.

That is, multiply every height, including the standard deviation, by 2.54. What is the answer now?

```
# Assign a variable 'female_avg' as the average female height. Convert this value to centimeters.
female_avg <- 64*2.54

# Assign a variable 'female_sd' as the standard deviation for female heights. Convert this value to centimeters.
female_sd <- 3*2.54

# Using variables 'female_avg' and 'female_sd', calculate the probability that a randomly selected female is between 61 and 67 inches.
pnorm((67*2.54), female_avg, female_sd) - pnorm((61*2.54), female_avg, female_sd)
```

```
## [1] 0.683
```

5. Compute the probability that the height of a randomly chosen female is within 1 SD from the average height.

```

# Assign a variable 'female_avg' as the average female height.
female_avg <- 64

# Assign a variable 'female_sd' as the standard deviation for female heights.
female_sd <- 3

# To a variable named 'taller', assign the value of a height that is one SD taller than average.
taller <- female_avg + female_sd

# To a variable named 'shorter', assign the value of a height that is one SD shorter than average.
shorter <- female_avg - female_sd

# Calculate the probability that a randomly selected female is between the desired height range. Print
pnorm(taller, female_avg, female_sd) - pnorm(shorter, female_avg, female_sd)

## [1] 0.683

```

6. Imagine the distribution of male adults is approximately normal with an expected value of 69 inches and a standard deviation of 3 inches.

How tall is a male in the 99th percentile?

```

# Assign a variable 'male_avg' as the average male height.
male_avg <- 69

# Assign a variable 'male_sd' as the standard deviation for male heights.
male_sd <- 3

# Determine the height of a man in the 99th percentile of the distribution.
qnorm(0.99, male_avg, male_sd)

```

```
## [1] 76
```

7. The distribution of IQ scores is approximately normally distributed.

The average is 100 and the standard deviation is 15. Suppose you want to know the distribution of the person with the highest IQ in your school district, where 10,000 people are born each year.

Generate 10,000 IQ scores 1,000 times using a Monte Carlo simulation. Make a histogram of the highest IQ scores.

```

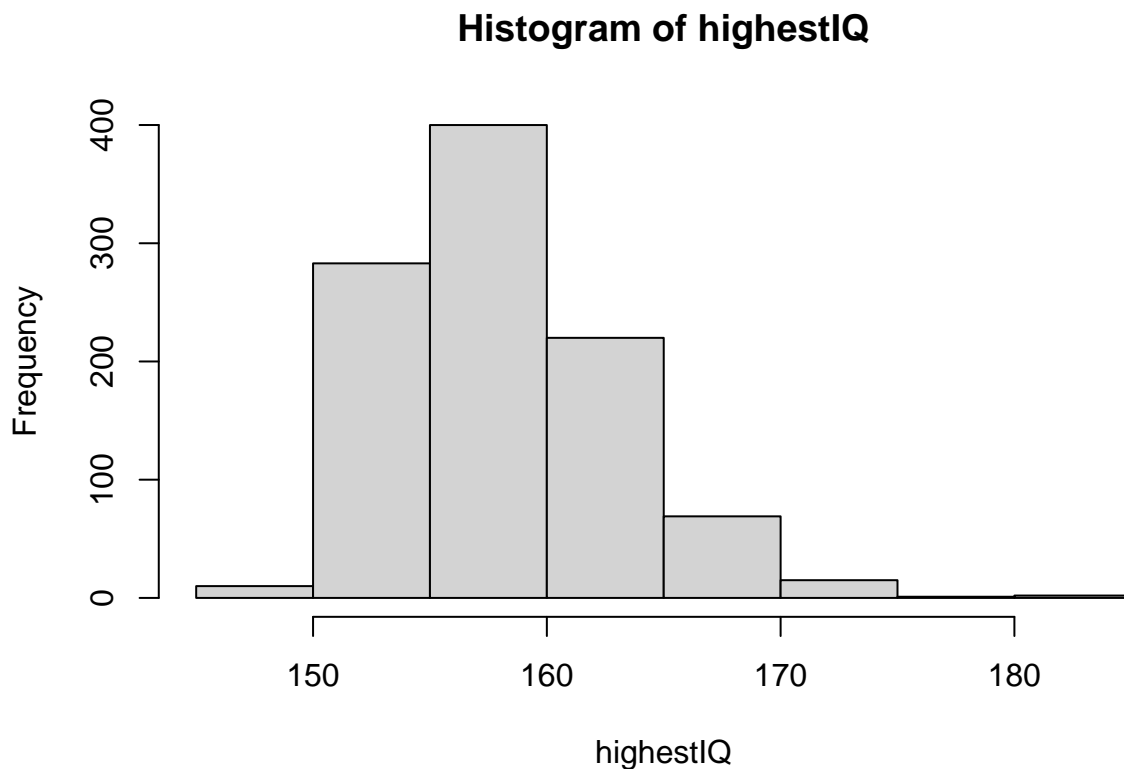
# The variable `B` specifies the number of times we want the simulation to run.
B <- 1000

# Use the `set.seed` function to make sure your answer matches the expected result after random number
set.seed(1)

# Create an object called `highestIQ` that contains the highest IQ score from each random distribution
highestIQ <- replicate(B, {
  IQ <- rnorm(10000, 100, 15)
  max(IQ)
})

```

```
# Make a histogram of the highest IQ scores.  
hist(highestIQ)
```



Assessment - ACT scores, part 1

1. and 2. The ACT is a standardized college admissions test used in the United States. The four multi-part questions in this assessment all involve simulating some ACT test scores and answering probability questions about them.

For the three year period 2016-2018, [ACT standardized test scores](#) were approximately normally distributed with a mean of 20.9 and standard deviation of 5.7. (Real ACT scores are integers between 1 and 36, but we will ignore this detail and use continuous values instead.)

First we'll simulate an ACT test score dataset and answer some questions about it.

Set the seed to 16, then use **rnorm** to generate a normal distribution of 10000 tests with a mean of 20.9 and standard deviation of 5.7. Save these values as **act_scores**. You'll be using this dataset throughout these four multi-part questions.

(IMPORTANT NOTE! If you use R 3.6 or later, you will need to use the command `set.seed(x, sample.kind = "Rounding")` instead of `set.seed(x)`. Your R version will be printed at the top of the Console window when you start RStudio.)

1a. What is the mean of **act_scores**?

```
set.seed(16, sample.kind = "Rounding")
```

```
## Warning in set.seed(16, sample.kind = "Rounding"): non-uniform 'Rounding'  
## sampler used
```

```
act_scores <- rnorm(10000, 20.9, 5.7)
mean(act_scores)
```

```
## [1] 20.8
```

1b. What is the standard deviation of `act_scores`?

```
sd(act_scores)
```

```
## [1] 5.68
```

1c. A perfect score is 36 or greater (the maximum reported score is 36).

In `act_scores`, how many perfect scores are there out of 10,000 simulated tests?

```
sum(act_scores >= 36)
```

```
## [1] 41
```

1d. In `act_scores`, what is the probability of an ACT score greater than 30?

```
mean(act_scores > 30)
```

```
## [1] 0.0527
```

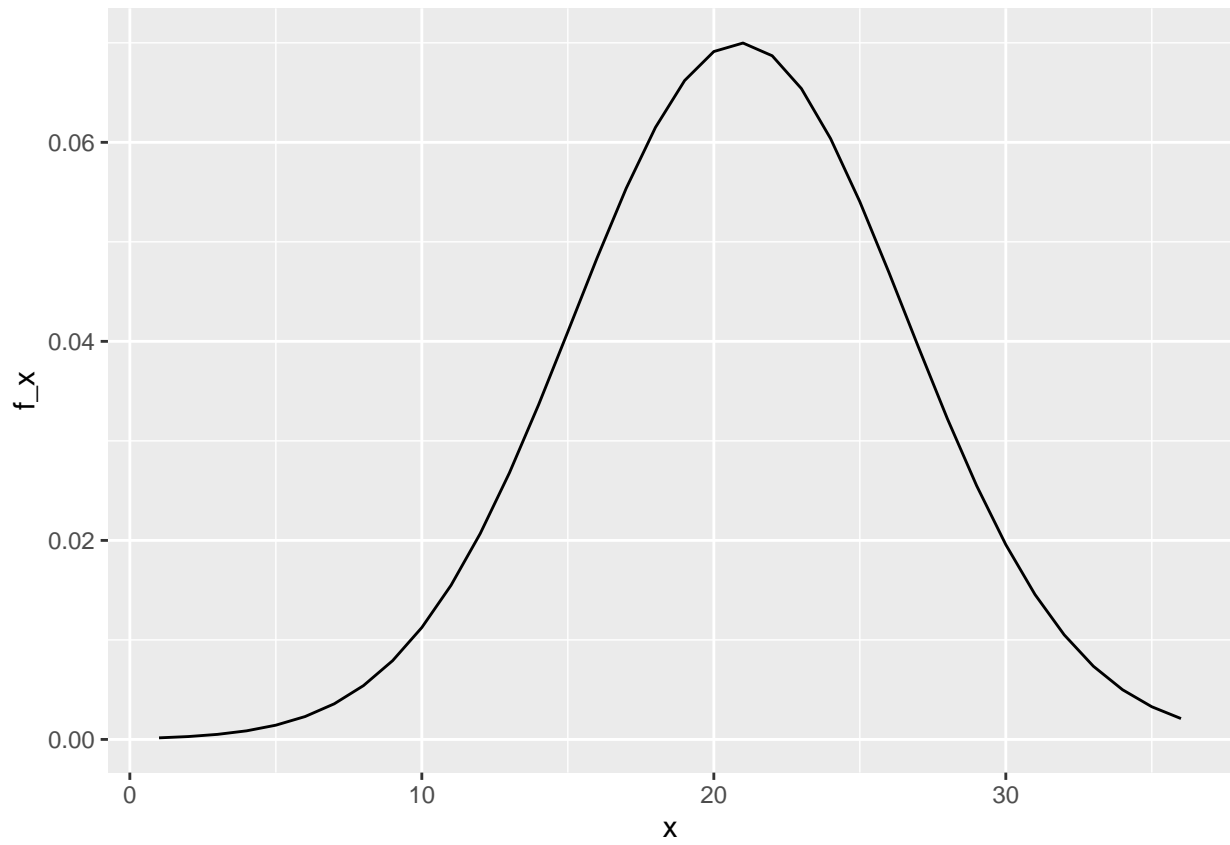
1e. In `act_scores`, what is the probability of an ACT score less than or equal to 10?

```
mean(act_scores <= 10)
```

```
## [1] 0.0282
```

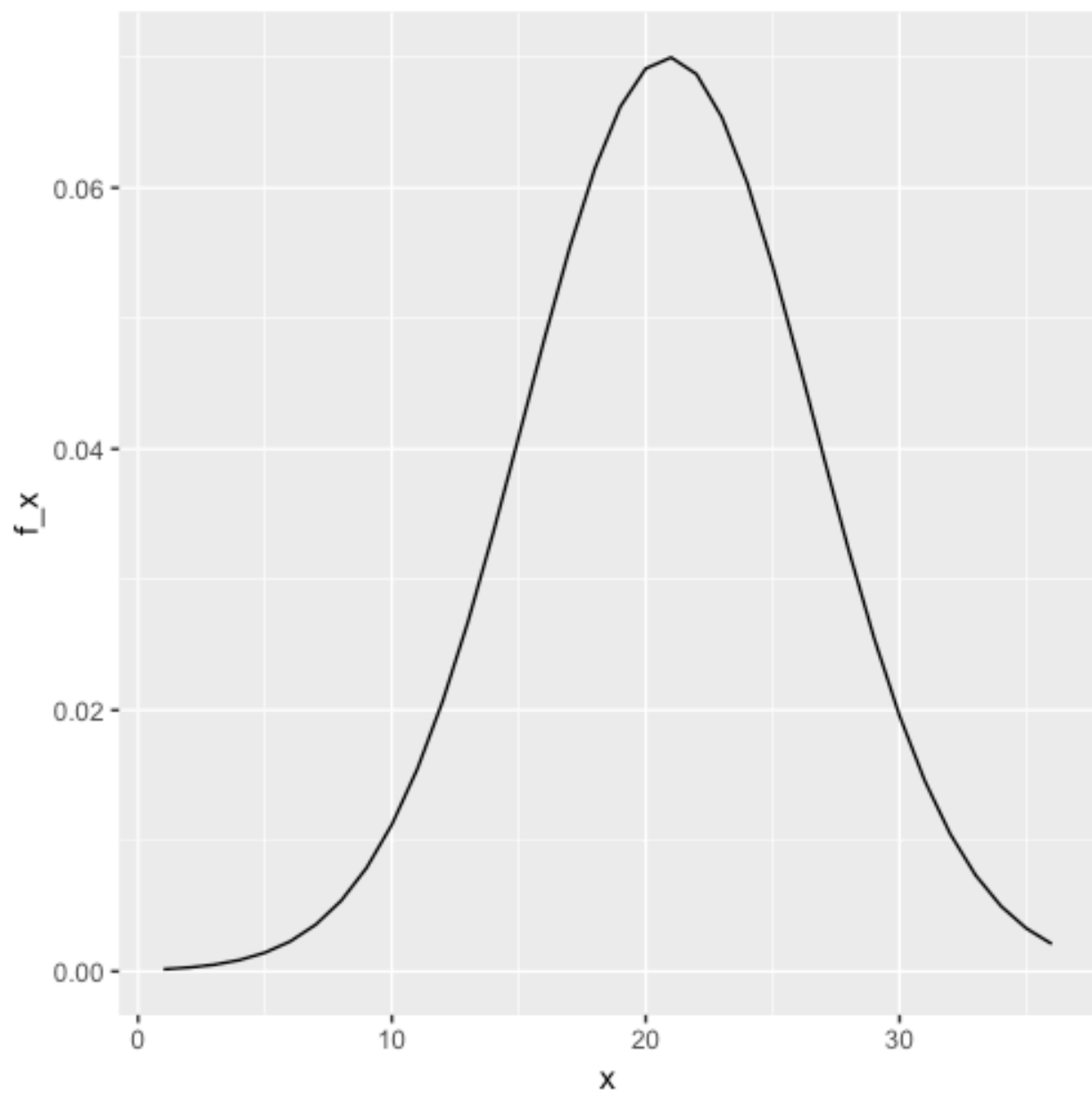
2. Set `x` equal to the sequence of integers 1 to 36. Use `dnorm` to determine the value of the probability density function over `x` given a mean of 20.9 and standard deviation of 5.7; save the result as `f_x`. Plot `x` against `f_x`.

```
x <- 1:36
f_x <- dnorm(x, 20.9, 5.7)
data.frame(x, f_x) %>%
  ggplot(aes(x, f_x)) +
  geom_line()
```



Which of the following plots is correct?

☒ B.



Assessment - ACT scores, part 2

3. In this 3-part question, you will convert raw ACT scores to Z-scores and answer some questions about them.

Convert `act_scores` to Z-scores. Recall from [Data Visualization](#) (the second course in this series) that to standardize values (convert values into Z-scores, that is, values distributed with a mean of 0 and standard deviation of 1), you must subtract the mean and then divide by the standard deviation. Use the mean and standard deviation of `act_scores`, not the original values used to generate random test scores.

- 3a. What is the probability of a Z-score greater than 2 (2 standard deviations above the mean)?

```
z_scores <- (act_scores - mean(act_scores))/sd(act_scores)
mean(z_scores > 2)
```

```
## [1] 0.0233
```

- 3b. What ACT score value corresponds to 2 standard deviations above the mean ($Z = 2$)?

```
2*sd(act_scores) + mean(act_scores)
```

```
## [1] 32.2
```

- 3c. A Z-score of 2 corresponds roughly to the 97.5th percentile.

Use `qnorm` to determine the 97.5th percentile of normally distributed data with the mean and standard deviation observed in `act_scores`.

What is the 97.5th percentile of `act_scores`?

```
qnorm(.975, mean(act_scores), sd(act_scores))
```

```
## [1] 32
```

4. In this 4-part question, you will write a function to create a CDF for ACT scores.

Write a function that takes a value and produces the probability of an ACT score less than or equal to that value (the CDF). Apply this function to the range 1 to 36.

- 4a. What is the minimum integer score such that the probability of that score or lower is at least .95?

Your answer should be an integer 1-36.

```
cdf <- sapply(1:36, function (x){
  mean(act_scores <= x)
})
min(which(cdf >= .95))
```

```
## [1] 31
```

- 4b. Use `qnorm` to determine the expected 95th percentile, the value for which the probability of receiving that score or lower is 0.95, given a mean score of 20.9 and standard deviation of 5.7.

What is the expected 95th percentile of ACT scores?

```
qnorm(.95, 20.9, 5.7)
```

```
## [1] 30.3
```

4c. As discussed in the Data Visualization course, we can use `quantile` to determine sample quantiles from the data.

Make a vector containing the quantiles for `p <- seq(0.01, 0.99, 0.01)`, the 1st through 99th percentiles of the `act_scores` data. Save these as `sample_quantiles`.

In what percentile is a score of 26?

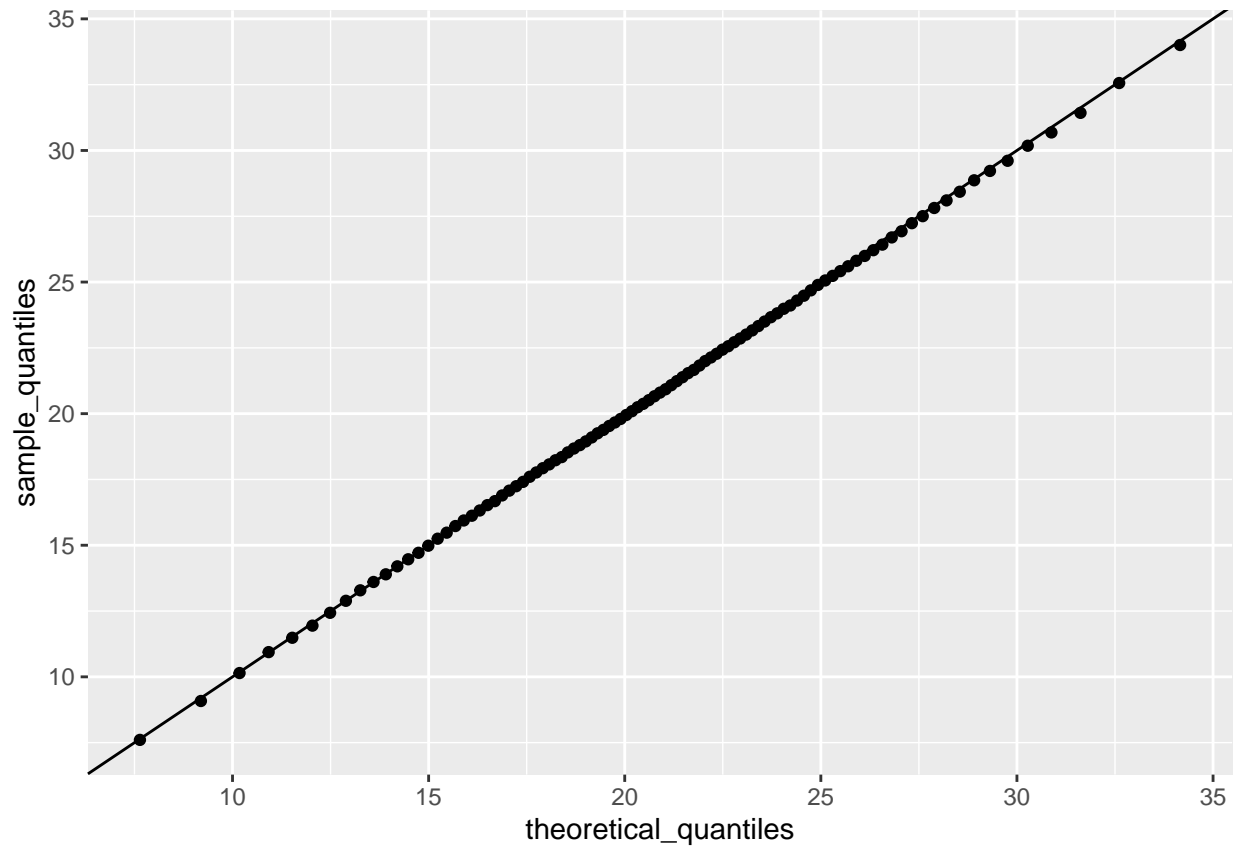
Note that a score between the 98th and 99th percentile should be considered the 98th percentile, for example, and that quantile numbers are used as names for the vector `sample_quantiles`.

```
p <- seq(0.01, 0.99, 0.01)
sample_quantiles <- quantile(act_scores, p)
names(sample_quantiles[max(which(sample_quantiles < 26))])
```

```
## [1] "82%"
```

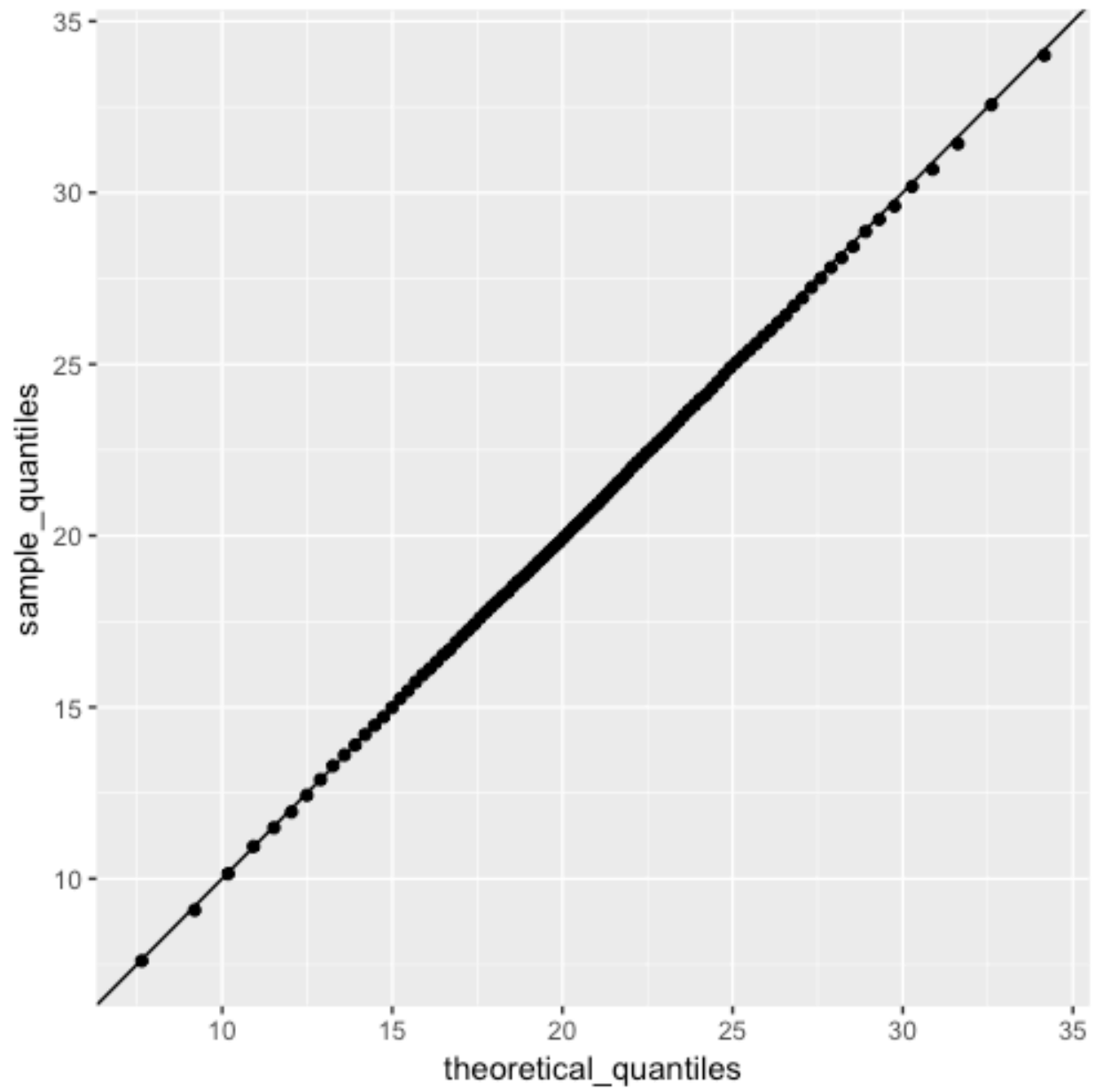
4d. Make a corresponding set of theoretical quantiles using `qnorm` over the interval `p <- seq(0.01, 0.99, 0.01)` with mean 20.9 and standard deviation 5.7. Save these as `theoretical_quantiles`. Make a QQ-plot graphing `sample_quantiles` on the y-axis versus `theoretical_quantiles` on the x-axis.

```
p <- seq(0.01, 0.99, 0.01)
sample_quantiles <- quantile(act_scores, p)
theoretical_quantiles <- qnorm(p, 20.9, 5.7)
qplot(theoretical_quantiles, sample_quantiles) + geom_abline()
```

Which of the following graphs is correct?

☒ D.



Section 3 Overview

Section 3 introduces you to Random Variables, Sampling Models, and the Central Limit Theorem.

Section 3 is divided into two parts: - Random Variables and Sampling Models - The Central Limit Theorem.

After completing Section 3, you will: - understand what random variables are, how to generate them, and the correct mathematical notation to use with them. - be able to use sampling models to estimate characteristics of a larger population. - be able to explain the difference between a distribution and a probability distribution. - understand the Central Limit Theorem and the law of large numbers.

The textbook for this section is available [here](#)

Random variables

Random variables are numeric outcomes resulting from random processes. We can easily generate random variables using some of the simple examples we have shown. For example, define X to be 1 if a bead is blue and red otherwise:

```
beads <- rep( c("red", "blue"), times = c(2,3))
X <- ifelse(sample(beads, 1) == "blue", 1, 0)

#Here X is a random variable: every time we select a new bead the outcome changes randomly. See below:

ifelse(sample(beads, 1) == "blue", 1, 0)

## [1] 1

ifelse(sample(beads, 1) == "blue", 1, 0)

## [1] 1

ifelse(sample(beads, 1) == "blue", 1, 0)

## [1] 1

#Sometimes it's 1 and sometimes it's 0.
```

Sampling Models

For example, we can model the process of polling likely voters as drawing 0's- Republicans- and 1's- Democrats- from an urn containing the 0 and 1 code for all likely voters.

In epidemiological studies, we often assume that the subjects in our study are a random sample from the population of interest. The data related to a specific outcome can be modeled as a random sample

Similarly, in experimental research, we often assume that the individual organisms we are studying- for example, worms, flies, or mice- are a random sample from a larger population.

Casino games offer a plethora of examples of real-world situations in which sampling models are used to answer specific questions.

Roulette wheel example