Técnicas cuantitativas

Enric Aguilar y Benito Zaragozí

19-Mar-2021

Índice general

Capítulo 1

Antes de empezar

En este breve capítulo explicamos los aspectos básicos para que podáis reproducir los ejercicios y entregar las actividades utilizando Markdown y Rstudio. Aquí planteamos los conceptos básicos para preparar un documento que incluye explicaciónes, código y los resultados de análisis R. Se trata de una estratégia de **programación literaria**.

1.1 Prerrequisitos

Para preparar cualquier tarea en este formato es necesario instalar primero R y RStudio con los paquetes `knitr` y `rmarkdown`.

1.2 Tipos de ficheros

- Los archivos para producir documentos de RMarkdown tienen la extensión `.Rmd`.
- Los archivos deben abrirse con RStudio y se compilan haciendo clic en el botón knitr.
- El resultado es un documento en formato `.pdf`, `.html` o `.doc`.

1.3 Un ejemplo muy sencillo

Cread un fichero con el siguiente contenido

```
Hola, soy **R Markdown**

Aprende más sobre mi [aquí](http://rmarkdown.rstudio.com/).
```

al hacer clic en el botón `knitr a HTML` de Rstudio se crea un archivo `.html` con este contenido.

1.4 Estructura básica de un documento Markdown

Revisemos las partes más importantes del documento `Rmd`.

1.4.1 Cabecera

La cabecera está en la parte superior del documento dentro de estas dos líneas
`---`

```
---
title: "Write your title here"
author: "Write your name here"
date: "Write the date here"
output:
  pdf_document: default
  html_document: default
  word_document: default
---
```

En el encabezado del archivo debe escribir el título del documento, su nombre y la fecha. La declaración de salida se utiliza para la clase del documento final, `pdf`, `htmlo` `doc`. Para producir un PDF es necesario tener instalado un motor de LaTeX.

1.4.2 Insertar código R

A continuación, configuramos las opciones necesarias para imprimir el código R y la salida R en el documento final.

La primera línea es ocultar este fragmento de código en el documento final.

La segunda línea es para imprimir el código R y la salida R en el documento final.

1.4.3 Formatos de texto

El texto sin formato se escribe como en cualquier otro documento, como un documento de Word. Debe tener cuidado con las letras en cursiva o negrita y algunos caracteres especiales. Por ejemplo

- **Negrita**: escriba el texto entre **Negrita** o ___Negrita___
- *Cursiva*: escriba su texto entre*cursiva* o _Italica_
- Encabezados de sección

```
# Título 1
## Título 2
### Título 3
```

Cuantos más símbolos # escriba antes de su texto, menor será el tamaño de su título

Puede encontrar más información sobre cómo escribir en el siguiente archivo.

1.4.4 Generando el documento

Para compilar el archivo `.Rmd` y obtener su documento final, simplemente haga clic en el botón `Knit` y seleccione `Knit a PDF` para producir un archivo `.pdf`.

Capítulo 2

Introducción a R

2.1 R como lenguaje de programación

R fue creado en 1992 por Ross Ihaka y Robert Gentleman en la Universidad de Auckland, Nueva Zelanda. R es una implementación gratuita de código abierto del lenguaje de programación estadística **S** creado inicialmente en Bell Labs. En esencia, R es un lenguaje de programación funcional (sus principales funcionalidades giran en torno a la definición y ejecución de funciones). Sin embargo, ahora es compatible, y se usa comúnmente como un lenguaje de programación imperativo (enfocado en instrucciones sobre variables y estructuras de control de programación) y orientado a objetos (que involucra estructuras de objetos complejas).

En términos simples, hoy en día, la programación en R se enfoca principalmente en diseñar una serie de instrucciones para ejecutar una tarea, más comúnmente, cargar y analizar un conjunto de datos.

Como tal, R se puede usar para programar creando secuencias de **instrucciones** que involucren **variables**, que son entidades con nombre que pueden almacenar valores. Ese será el tema principal de esta sesión práctica. Las instrucciones pueden incluir estructuras de flujo de control, como puntos de decisión (*if/else*) y bucles, que serán el tema de la próxima sesión práctica. Las instrucciones también se pueden agrupar en **funciones**, que también veremos en la próxima sesión práctica.

R es **interpretado**, no compilado. Lo que significa que un intérprete de R (si está utilizando R Studio, el intérprete de R simplemente está oculto en el backend y R Studio es el frontend que le permite interactuar con el intérprete) recibe una instrucción que escribe en R, la interpreta y la ejecuta . Otros lenguajes de programación requieren que su código sea compilado en un ejecutable para ser ejecutado en un ordenador.

2.2 Utilizando RStudio

La interfaz de RStudio se divide en dos secciones principales. En el lado izquierdo, encontrará la *Consola*, así como el editor de secuencias de comandos R, cuando se está editando una secuencia de comandos. La *Consola* en una ventana de entrada/salida en el intérprete de R, donde se pueden escribir instrucciones y se muestra la salida calculada.

Por ejemplo, si escribís en la *Consola*

```
1 + 1
```

el intérprete de R entiende eso como una instrucción para sumar uno más uno, y produce el resultado (dado que los materiales para este módulo se crean en RMarkdown, la salida del cálculo siempre está precedida por '##').

```
## [1] 2
```

Fijaos cómo el valor de salida `2` está precedido por `[1]`, lo que indica que la salida está constituida por un solo elemento. Si la salida está constituida por más de un elemento, como la lista de números a continuación, cada fila de la salida está precedida por el índice del primer elemento de la salida.

```
##  [1]   1   4   9  16  25  36  49  64  81 100 121 144 169 196 225 256 289 324 361
## [20] 400
```

On the right side, you find two groups of panels. On the top-right, the main element is the *Environment* panel, which is a representation of the current state of the interpreter's memory, and as such, it shows all the stored variables, datasets, and functions. On the bottom-right, you find the *Files* panel, which shows file system (file and folders on your computer or the server), as well as the *Help* panel, which shows you the help pages when required. We will discuss the other panels later on in the practical sessions.

2.3 Interpreting values

When a value is typed in the *Console*, the interpreter simply returns the same value. In the examples below, `2` is a simple numeric value, while `"String value"` is a textual value, which in R is referred to as a *character* value and in programming is also commonly referred to as a *string* (short for *a string of characters*).

Numeric example:

```
2
```

```
## [1] 2
```

Character example:

```
"String value"
```

```
## [1] "String value"
```

Note how character values need to start and end with a single or double quote (`'` or `"`), which are not part of the information themselves. The Tidyverse Style Guide suggests always to use the double quote (`"`), so we will use those in this module.

Anything that follows a `#` symbol is considered a *comment* and the interpreter ignores it.

```
# hi, I am a comment, please ignore me
```

As mentioned above, the interpreter understands simple operations on numeric values.

```
1 + 1
```

```
## [1] 2
```

There are also a large number of pre-defined functions, e.g., square-root: `sqrt`.

```
sqrt(2)
```

```
## [1] 1.414214
```

Functions are collected and stored in *libraries* (sometimes referred to as *packages*), which contains related functions. Libraries can range anywhere from the `base` library, which includes the `sqrt` function above, to the `rgdal` library, which contains implementations of the GDAL (Geospatial Data Abstraction Library) functionalities for R.

2.4 Variables

A variable can be defined using an **identifier** (e.g., `a_variable`) on the left of an **assignment operator** `<-`, followed by the object to be linked to the identifier, such as a **value** (e.g., `1`) to be assigned on the right. The value of the variable can be tested/invoked by simply specifying the **identifier**.

```
a_variable <- 1
a_variable
```

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

If you type `a_variable <- 1` in the *Console* in RStudio, a new element appears in the *Environment* panel, representing the new variable in the memory. The left part of the entry contains the identifier `a_variable`, and the right part contains the value assigned to the variable `a_variable`, that is `1`.

It is not necessary to provide a value directly. The right part of the assignment can be a **call to a function**. In that case, the function is **executed** on the provided input and **the result is assigned to the variable**.

```
a_variable <- sqrt(4)
a_variable
```

```
## [1] 2
```

Note how if you type `a_variable <- sqrt(4)` in the *Console* in RStudio, the element in the *Environment* panel changes to reflect the new value assigned to the variable `a_variable`, which is now the result of `sqrt(4)`, that is 2.

In the example below, another variable named `another_variable` is created and summed to `a_variable`, saving the result in `sum_of_two_variables`. The square root of that sum is then stored in the variable `square_root_of_sum`.

```
another_variable <- 4
another_variable
```

```
## [1] 4
```

```
sum_of_two_variables <- a_variable + another_variable

square_root_of_sum <- sqrt(sum_of_two_variables)
square_root_of_sum
```

```
## [1] 2.44949
```

2.5 Basic types

2.5.1 Numeric

The *numeric* type represents numbers (both integers and reals).

```
a_number <- 1.41
is.numeric(a_number)
```

```
## [1] TRUE
```

```
is.integer(a_number)
```

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

```
is.double(a_number) # i.e., is real
```

```
## [1] TRUE
```

Base numeric operators.

Operator	Meaning	Example	Output
+	Plus	`5+2`	7
-	Minus	`5-2`	3
*	Product	`5*2`	10
/	Division	`5/2`	2.5
%/%	Integer division	`5%/%2`	2
%%	Module	`5%%2`	1
^	Power	`5^2`	25

Some pre-defined functions in R:

```
abs(-2) # Absolute value
```

```
## [1] 2
```

```
ceiling(3.475) # Upper round
```

```
## [1] 4
```

```
floor(3.475) # Lower round
```

```
## [1] 3
```

```
trunc(5.99) # Truncate
```

```
## [1] 5
```

```
log10(100) # Logarithm 10
```

```
## [1] 2
```

```
log(exp(2)) # Natural logarithm and e
```

```
## [1] 2
```

Use simple brackets to specify the order of execution. If not specified the default order is: rise to power first, then multiplication and division, sum and subtraction last.

```
a_number <- 1
(a_number + 2) * 3
```

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

```
a_number + (2 * 3)
```

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

```
a_number + 2 * 3
```

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

The object `NaN` (*Not a Number*) is returned by R when the result of an operation is not a number.

```
0 / 0
```

```
## [1] NaN
```

```
is.nan(0 / 0)
```

```
## [1] TRUE
```

That is not to be confused with the object `NA` (*Not Available*), which is returned for missing data.

2.5.2 Logical

The *logical* type encodes two truth values: True and False.

```
logical_var <- TRUE
is.logical(logical_var)
```

```
## [1] TRUE
```

```
isTRUE(logical_var)
```

```
## [1] TRUE
```

```
as.logical(0) # TRUE if not zero
```

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

Basic logic operators

Operator	Meaning	Example	Output
==	Equal	`5==2`	FALSE
!=	Not equal	`5!=2`	TRUE
>	Greater than	`5>2`	TRUE
<	Less than	`5<2`	FALSE
>=	Greater or equal	`5>=2`	TRUE
<=	Less or equal	`5<=2`	FALSE
!	Not	`!TRUE`	FALSE
&	And	`TRUE & FALSE`	FALSE
|	Or	`TRUE | FALSE`	TRUE

2.5.3 Character

The *character* type represents text objects, including single characters and character strings (that is text objects longer than one character, commonly referred to simply as *strings* in computer science).

```r
a_string <- "Hello world!"
is.character(a_string)
```

```
## [1] TRUE
```

```r
is.numeric(a_string)
```

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

```r
as.character(2) # type conversion  (a.k.a. casting)
```

```
## [1] "2"
```

```r
as.numeric("2")
```

```
## [1] 2
```

```r
as.numeric("Ciao")
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA
```

2.6 Tidyverse

As mentioned in the lecture, libraries are collections of functions and/or datasets. Libraries can be installed in R using the function `install.packages` or using `Tool > Install Packages...` in RStudio.

The meta-library Tidyverse contains the following libraries:

- `ggplot2` is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.
- `dplyr` provides a grammar of data manipulation, providing a consistent set of verbs that solve the most common data manipulation challenges.
- `tidyr` provides a set of functions that help you get to tidy data. Tidy data is data with a consistent form: in brief, every variable goes in a column, and every column is a variable.
- `readr` provides a fast and friendly way to read rectangular data (like csv, tsv, and fwf). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes.
- `purrr` enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors. Once you master the basic concepts, purrr allows you to replace many for loops with code that is easier to write and more expressive.
- `tibble` is a modern re-imagining of the data frame, keeping what time has proven to be effective, and throwing out what it has not. Tibbles

are data.frames that are lazy and surly: they do less and complain more forcing you to confront problems earlier, typically leading to cleaner, more expressive code.
- `stringr` provides a cohesive set of functions designed to make working with strings as easy as possible. It is built on top of stringi, which uses the ICU C library to provide fast, correct implementations of common string manipulations.
- `forcats` provides a suite of useful tools that solve common problems with factors. R uses factors to handle categorical variables, variables that have a fixed and known set of possible values.

A library can be loaded using the function `library`, as shown below (note the name of the library is not quoted). Once a library is installed on a computer, you don't need to install it again, but every script needs to load all the library that it uses. Once a library is loaded, all its functions can be used.

Important: it is always necessary to load the `tidyverse` meta-library if you want to use the `stringr` functions or the pipe operator `%>%`.

```
library(tidyverse)
```

2.6.1 stringr

The code below presents the same examples used in the lecture session to demonstrate the use of `stringr` functions.

```
str_length("Leicester")
```

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

```
str_detect("Leicester", "e")
```

```
## [1] TRUE
```

```
str_replace_all("Leicester", "e", "x")
```

```
## [1] "Lxicxstxr"
```

2.6.2 The pipe operator

The pipe operator is useful to outline more complex operations, step by step (see also R for Data Science, Chapter 18). The pipe operator `%>%`

- takes the result from one function
- and passes it to the next function
- as the **first argument**
- that doesn't need to be included in the code anymore

The code below shows a simple example. The number `2` is taken as input for the first pipe that passes it on as the first argument to the function `sqrt`. The

output value `1.41` is then taken as input for the second pipe, that passes it on as the first argument to the function `trunc`. The final output `1` is finally returned.

```
2 %>%
  sqrt() %>%
  trunc()
```

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

```
sqrt(2) %>%
 round(digits = 2)
```

The first step of a sequence of pipes can be a value, a variable, or a function including arguments. The code below shows a series of examples of different ways of achieving the same result. The examples use the function `round`, which also allows for a second argument `digits = 2`. Note that, when using the pipe operator, only the nominally second argument is provided to the function `round` – that is `round(digits = 2)`

```
# No pipe, using variables
tmp_variable_A <- 2
tmp_variable_B <- sqrt(tmp_variable_A)
round(tmp_variable_B, digits = 2)

# No pipe, using functions only
round(sqrt(2), digits = 2)

# Pipe starting from a value
2 %>%
  sqrt() %>%
  round(digits = 2)

# Pipe starting from a variable
the_value_two <- 2
the_value_two %>%
  sqrt() %>%
  round(digits = 2)

# Pipe starting from a function
sqrt(2) %>%
 round(digits = 2)
```

A complex operation created through the use of `%>%` can be used on the right side of `<-`, to assign the outcome of the operation to a variable.

```
sqrt_of_two <- 2 %>%
  sqrt() %>%
  round(digits = 2)
```

2.7 Coding style

Study the Tidyverse Style Guide (style.tidyverse.org) and use it consistently!

2.8 Exercise 104.1

Question 104.1.1: Write a piece of code using the pipe operator that takes as input the number `1632`, calculates the logarithm to the base 10, takes the highest integer number lower than the calculated value (lower round), and verifies whether it is an integer.

Question 104.1.2: Write a piece of code using the pipe operator that takes as input the number `1632`, calculates the square root, takes the lowest integer number higher than the calculated value (higher round), and verifies whether it is an integer.

Question 104.1.3: Write a piece of code using the pipe operator that takes as input the string `"1632"`, transforms it into a number, and checks whether the result is *Not a Number*.

Question 104.1.4: Write a piece of code using the pipe operator that takes as input the string `"-16.32"`, transforms it into a number, takes the absolute value and truncates it, and finally checks whether the result is *Not Available*.

2.9 Exercise 104.2

Answer the question below, consulting the `stringr` library reference (stringr.tidyverse.org/reference) as necessary

Question 104.2.1: Write a piece of code using the pipe operator and the `stringr` library that takes as input the string `"I like programming in R"`, and transforms it all in uppercase.

Question 104.2.2: Write a piece of code using the pipe operator and the `stringr` library that takes as input the string `"I like programming in R"`, and truncates it, leaving only 10 characters.

Question 104.2.3: Write a piece of code using the pipe operator and the `stringr` library that takes as input the string `"I like programming in R"`, and truncates it, leaving only 10 characters and using no ellipsis.

Question 104.2.4: Write a piece of code using the pipe operator and the `stringr` library that takes as input the string `"I like programming in R"`, and manipulates to leave only the string `"I like R"`.

Capítulo 3

Estadística descriptiva

Here is a review of existing methods.

Capítulo 4

Estadística inferencial

Let's load all the packages needed for this chapter (this assumes you've already installed them). Recall from our discussion in Section **??** that loading the `tidyverse` package by running `library(tidyverse)` loads the following commonly used data science packages all at once:

- `ggplot2` for data visualization
- `dplyr` for data wrangling
- `tidyr` for converting data to "tidy" format
- `readr` for importing spreadsheet data into R
- As well as the more advanced `purrr`, `tibble`, `stringr`, and `forcats` packages

If needed, read Section **??** for information on how to install and load R packages.

```
library(tidyverse)
library(readr)
library(infer)
library(moderndive)
library(nycflights13)
library(ggplot2movies)
```

4.1 Promotions activity

Let's start with an activity studying the effect of gender on promotions at a bank.

4.1.1 Does gender affect promotions at a bank?

Say you are working at a bank in the 1970s and you are submitting your résumé to apply for a promotion. Will your gender affect your chances of getting

promoted? To answer this question, we'll focus on data from a study published in the *Journal of Applied Psychology* in 1974. This data is also used in the *OpenIntro* series of statistics textbooks.

To begin the study, 48 bank supervisors were asked to assume the role of a hypothetical director of a bank with multiple branches. Every one of the bank supervisors was given a résumé and asked whether or not the candidate on the résumé was fit to be promoted to a new position in one of their branches.

However, each of these 48 résumés were identical in all respects except one: the name of the applicant at the top of the résumé. Of the supervisors, 24 were randomly given résumés with stereotypically "male" names, while 24 of the supervisors were randomly given résumés with stereotypically "female" names. Since only (binary) gender varied from résumé to résumé, researchers could isolate the effect of this variable in promotion rates.

While many people today (including us, the authors) disagree with such binary views of gender, it is important to remember that this study was conducted at a time where more nuanced views of gender were not as prevalent. Despite this imperfection, we decided to still use this example as we feel it presents ideas still relevant today about how we could study discrimination in the workplace.

The `moderndive` package contains the data on the 48 applicants in the `promotions` data frame. Let's explore this data by looking at six randomly selected rows:

```
promotions %>%
  sample_n(size = 6) %>%
  arrange(id)
```

```
## # A tibble: 6 x 3
##      id decision gender
##   <int> <fct>    <fct>
## 1    11 promoted male
## 2    26 promoted female
## 3    28 promoted female
## 4    36 not      male
## 5    37 not      male
## 6    46 not      female
```

The variable `id` acts as an identification variable for all 48 rows, the `decision` variable indicates whether the applicant was selected for promotion or not, while the `gender` variable indicates the gender of the name used on the résumé. Recall that this data does not pertain to 24 actual men and 24 actual women, but rather 48 identical résumés of which 24 were assigned stereotypically "male" names and 24 were assigned stereotypically "female" names.

Let's perform an exploratory data analysis of the relationship between the two categorical variables `decision` and `gender`. Recall that we saw in Subsection **??**

that one way we can visualize such a relationship is by using a stacked barplot.

```
ggplot(promotions, aes(x = gender, fill = decision)) +
  geom_bar() +
  labs(x = "Gender of name on résumé")
```

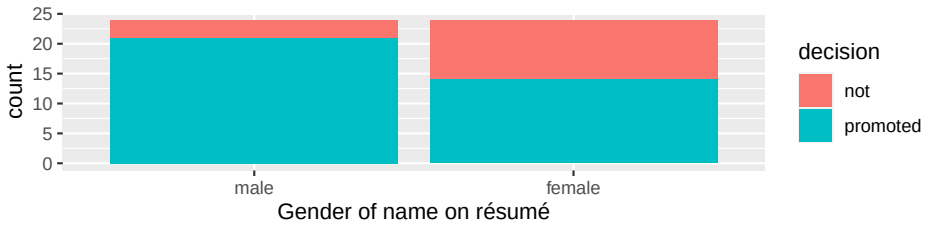

Figura 4.1: Barplot relating gender to promotion decision.

Observe in Figure 4.1 that it appears that résumés with female names were much less likely to be accepted for promotion. Let's quantify these promotion rates by computing the proportion of résumés accepted for promotion for each group using the `dplyr` package for data wrangling. Note the use of the `tally()` function here which is a shortcut for `summarize(n = n())` to get counts.

```
promotions %>%
  group_by(gender, decision) %>%
  tally()
```

```
## # A tibble: 4 x 3
## # Groups:   gender [2]
##   gender decision     n
##   <fct>  <fct>    <int>
## 1 male   not          3
## 2 male   promoted    21
## 3 female not         10
## 4 female promoted    14
```

So of the 24 résumés with male names, 21 were selected for promotion, for a proportion of $21/24 = 0.875 = 87.5\%$. On the other hand, of the 24 résumés with female names, 14 were selected for promotion, for a proportion of $14/24 = 0.583 = 58.3\%$. Comparing these two rates of promotion, it appears that résumés with male names were selected for promotion at a rate 0.875 - 0.583 = 0.292 = 29.2% higher than résumés with female names. This is suggestive of an advantage for résumés with a male name on it.

The question is, however, does this provide *conclusive* evidence that there is gender discrimination in promotions at banks? Could a difference in promotion rates of 29.2% still occur by chance, even in a hypothetical world where no gender-based discrimination existed? In other words, what is the role of *sampling variation* in this hypothesized world? To answer this question, we'll again rely

on a computer to run *simulations*.

4.1.2 Shuffling once

First, try to imagine a hypothetical universe where no gender discrimination in promotions existed. In such a hypothetical universe, the gender of an applicant would have no bearing on their chances of promotion. Bringing things back to our `promotions` data frame, the `gender` variable would thus be an irrelevant label. If these `gender` labels were irrelevant, then we could randomly reassign them by "shuffling" them to no consequence!

To illustrate this idea, let's narrow our focus to 6 arbitrarily chosen résumés of the 48 in Table 4.1.2. The `decision` column shows that 3 résumés resulted in promotion while 3 didn't. The `gender` column shows what the original gender of the résumé name was.

However, in our hypothesized universe of no gender discrimination, gender is irrelevant and thus it is of no consequence to randomly "shuffle" the values of `gender`. The `shuffled_gender` column shows one such possible random shuffling. Observe in the fourth column how the number of male and female names remains the same at 3 each, but they are now listed in a different order.

One example of shuffling gender variable

résumé number

decision

gender

shuffled gender

1

not

male

male

2

not

female

male

3

not

female

female

4

promoted

male

female

5

promoted

male

female

6

promoted

female

male

Again, such random shuffling of the gender label only makes sense in our hypothesized universe of no gender discrimination. How could we extend this shuffling of the gender variable to all 48 résumés by hand? One way would be by using standard deck of 52 playing cards, which we display in Figure **??**.

Since half the cards are red (diamonds and hearts) and the other half are black (spades and clubs), by removing two red cards and two black cards, we would end up with 24 red cards and 24 black cards. After shuffling these 48 cards as seen in Figure **??**, we can flip the cards over one-by-one, assigning "male" for each red card and "female" for each black card.

We've saved one such shuffling in the `promotions_shuffled` data frame of the `moderndive` package. If you compare the original `promotions` and the shuffled `promotions_shuffled` data frames, you'll see that while the `decision` variable is identical, the `gender` variable has changed.

Let's repeat the same exploratory data analysis we did for the original `promotions` data on our `promotions_shuffled` data frame. Let's create a barplot visualizing the relationship between `decision` and the new shuffled `gender` variable and compare this to the original unshuffled version in Figure 4.2.

```
ggplot(promotions_shuffled,
       aes(x = gender, fill = decision)) +
  geom_bar() +
  labs(x = "Gender of résumé name")
```

```
## `summarise()` regrouping output by 'gender' (override with `.groups` argument)
## `summarise()` regrouping output by 'gender' (override with `.groups` argument)
```

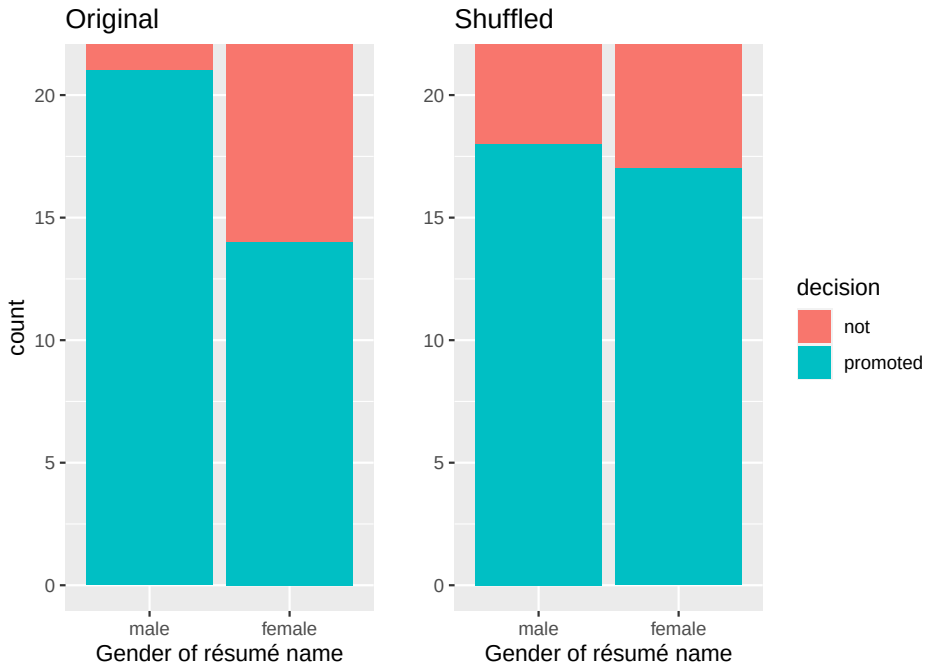

Figura 4.2: Barplots of relationship of promotion with gender (left) and shuffled gender (right).

It appears the difference in "male names" versus "female names" promotion rates is now different. Compared to the original data in the left barplot, the new "shuffled" data in the right barplot has promotion rates that are much more similar.

Let's also compute the proportion of résumés accepted for promotion for each group:

```
promotions_shuffled %>%
  group_by(gender, decision) %>%
  tally() # Same as summarize(n = n())
```

```
## # A tibble: 4 x 3
## # Groups:   gender [2]
##   gender decision     n
##   <fct>  <fct>    <int>
## 1 male   not          6
## 2 male   promoted    18
## 3 female not          7
## 4 female promoted    17
```

So in this hypothetical universe of no discrimination, $18/24 = 0.75 = 75\%$ of "male" résumés were selected for promotion. On the other hand, $17/24 = 0.708 = 70.8\%$ of "female" résumés were selected for promotion.

Let's next compare these two values. It appears that résumés with stereotypically male names were selected for promotion at a rate that was $0.75 - 0.708 = 0.042 = 4.2\%$ different than résumés with stereotypically female names.

Observe how this difference in rates is not the same as the difference in rates of $0.292 = 29.2\%$ we originally observed. This is once again due to *sampling variation*. How can we better understand the effect of this sampling variation? By repeating this shuffling several times!

4.1.3 Shuffling 16 times

We recruited 16 groups of our friends to repeat this shuffling exercise. They recorded these values in a shared spreadsheet; we display a snapshot of the first 10 rows and 5 columns in Figure **??**.

For each of these 16 columns of *shuffles*, we computed the difference in promotion rates, and in Figure 4.3 we display their distribution in a histogram. We also mark the observed difference in promotion rate that occurred in real life of $0.292 = 29.2\%$ with a dark line.

Before we discuss the distribution of the histogram, we emphasize the key thing to remember: this histogram represents differences in promotion rates that one would observe in our *hypothesized universe* of no gender discrimination.

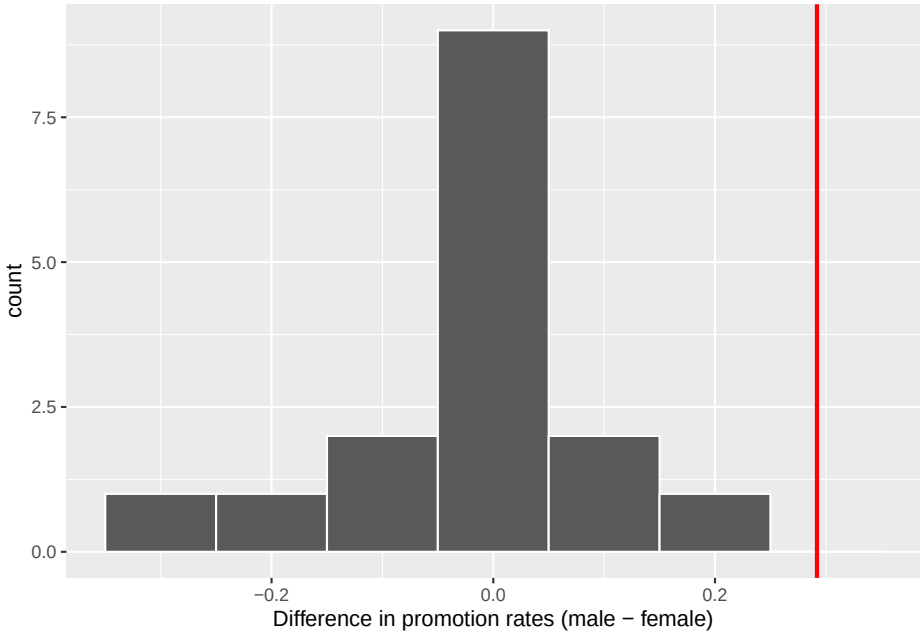

Figura 4.3: Distribution of shuffled differences in promotions.

Observe first that the histogram is roughly centered at 0. Saying that the difference in promotion rates is 0 is equivalent to saying that both genders had the same promotion rate. In other words, the center of these 16 values is consistent with what we would expect in our hypothesized universe of no gender discrimination.

However, while the values are centered at 0, there is variation about 0. This is because even in a hypothesized universe of no gender discrimination, you will still likely observe small differences in promotion rates because of chance *sampling variation*. Looking at the histogram in Figure 4.3, such differences could even be as extreme as -0.292 or 0.208.

Turning our attention to what we observed in real life: the difference of 0.292 = 29.2% is marked with a vertical dark line. Ask yourself: in a hypothesized world of no gender discrimination, how likely would it be that we observe this difference? While opinions here may differ, in our opinion not often! Now ask yourself: what do these results say about our hypothesized universe of no gender discrimination?

4.1.4 What did we just do?

What we just demonstrated in this activity is the statistical procedure known as *hypothesis testing* using a *permutation test*. The term “permutation” is the

mathematical term for "shuffling": taking a series of values and reordering them randomly, as you did with the playing cards.

In fact, permutations are another form of *resampling*, like the bootstrap method you performed in Chapter **??**. While the bootstrap method involves resampling *with* replacement, permutation methods involve resampling *without* replacement.

Think of our exercise involving the slips of paper representing pennies and the hat in Section **??**: after sampling a penny, you put it back in the hat. Now think of our deck of cards. After drawing a card, you laid it out in front of you, recorded the color, and then you *did not* put it back in the deck.

In our previous example, we tested the validity of the hypothesized universe of no gender discrimination. The evidence contained in our observed sample of 48 résumés was somewhat inconsistent with our hypothesized universe. Thus, we would be inclined to *reject* this hypothesized universe and declare that the evidence suggests there is gender discrimination.

Recall our case study on whether yawning is contagious from Section **??**. The previous example involves inference about an unknown difference of population proportions as well. This time, it will be $p_m - p_f$, where p_m is the population proportion of résumés with male names being recommended for promotion and p_f is the equivalent for résumés with female names. Recall that this is one of the scenarios for inference we've seen so far in Table 4.1.4.

Scenarios of sampling for inference

Scenario

Population parameter

Notation

Point estimate

Symbol(s)

1

Population proportion

p

Sample proportion

$\hat{p}$

2

Population mean

μ

Sample mean

$\overline{x}$ or $\widehat{\mu}$

3

Difference in population proportions

$p_1 - p_2$

Difference in sample proportions

$\widehat{p}_1 - \widehat{p}_2$

So, based on our sample of $n_m = 24$ "male" applicants and $n_w = 24$ "female" applicants, the *point estimate* for $p_m - p_f$ is the *difference in sample proportions* $\widehat{p}_m - \widehat{p}_f = 0.875 - 0.583 = 0.292 = 29.2\%$. This difference in favor of "male" résumés of 0.292 is greater than 0, suggesting discrimination in favor of men.

However, the question we asked ourselves was "is this difference meaningfully greater than 0?". In other words, is that difference indicative of true discrimination, or can we just attribute it to *sampling variation*? Hypothesis testing allows us to make such distinctions.

4.2 Understanding hypothesis tests

Much like the terminology, notation, and definitions relating to sampling you saw in Section **??**, there are a lot of terminology, notation, and definitions related to hypothesis testing as well. Learning these may seem like a very daunting task at first. However, with practice, practice, and more practice, anyone can master them.

First, a **hypothesis** is a statement about the value of an unknown population parameter. In our résumé activity, our population parameter of interest is the difference in population proportions $p_m - p_f$. Hypothesis tests can involve any of the population parameters in Table **??** of the five inference scenarios we'll cover in this book and also more advanced types we won't cover here.

Second, a **hypothesis test** consists of a test between two competing hypotheses: (1) a **null hypothesis** H_0 (pronounced "H-naught") versus (2) an **alternative hypothesis** H_A (also denoted H_1).

Generally the null hypothesis is a claim that there is "no effect" or "no difference of interest." In many cases, the null hypothesis represents the status quo or a situation that nothing interesting is happening. Furthermore, generally the alternative hypothesis is the claim the experimenter or researcher wants to establish or find evidence to support. It is viewed as a "challenger" hypothesis to the null hypothesis H_0. In our résumé activity, an appropriate hypothesis test would be:

$$\begin{aligned} H_0 &: \text{men and women are promoted at the same rate} \\ \text{vs } H_A &: \text{men are promoted at a higher rate than women} \end{aligned}$$

Note some of the choices we have made. First, we set the null hypothesis H_0 to be that there is no difference in promotion rate and the "challenger" alternative hypothesis H_A to be that there is a difference. While it would not be wrong in principle to reverse the two, it is a convention in statistical inference that the null hypothesis is set to reflect a "null" situation where "nothing is going on." As we discussed earlier, in this case, H_0 corresponds to there being no difference in promotion rates. Furthermore, we set H_A to be that men are promoted at a *higher* rate, a subjective choice reflecting a prior suspicion we have that this is the case. We call such alternative hypotheses *one-sided alternatives*. If someone else however does not share such suspicions and only wants to investigate that there is a difference, whether higher or lower, they would set what is known as a *two-sided alternative*.

We can re-express the formulation of our hypothesis test using the mathematical notation for our population parameter of interest, the difference in population proportions $p_m - p_f$:

$$\begin{aligned} H_0 &: p_m - p_f = 0 \\ \text{vs } H_A &: p_m - p_f > 0 \end{aligned}$$

Observe how the alternative hypothesis H_A is one-sided with $p_m - p_f > 0$. Had we opted for a two-sided alternative, we would have set $p_m - p_f \neq 0$. To keep things simple for now, we'll stick with the simpler one-sided alternative. We'll present an example of a two-sided alternative in Section **??**.

Third, a **test statistic** is a *point estimate/sample statistic* formula used for hypothesis testing. Note that a sample statistic is merely a summary statistic based on a sample of observations. Recall we saw in Section **??** that a summary statistic takes in many values and returns only one. Here, the samples would be the $n_m = 24$ résumés with male names and the $n_f = 24$ résumés with female names. Hence, the point estimate of interest is the difference in sample proportions $\hat{p}_m - \hat{p}_f$.

Fourth, the **observed test statistic** is the value of the test statistic that we observed in real life. In our case, we computed this value using the data saved in the `promotions` data frame. It was the observed difference of $\hat{p}_m - \hat{p}_f = 0.875 - 0.583 = 0.292 = 29.2\%$ in favor of résumés with male names.

Fifth, the **null distribution** is the sampling distribution of the test statistic *assuming the null hypothesis H_0 is true*. Ooof! That's a long one! Let's unpack it slowly. The key to understanding the null distribution is that the null hypothesis H_0 is *assumed* to be true. We're not saying that H_0 is true at this point, we're only assuming it to be true for hypothesis testing purposes. In our case, this corresponds to our hypothesized universe of no gender discrimination in promotion rates. Assuming the null hypothesis H_0, also stated as "Under H_0," how does the test statistic vary due to sampling variation? In our case, how will the difference in sample proportions $\hat{p}_m - \hat{p}_f$ vary due to sampling under H_0?

Recall from Subsection **??** that distributions displaying how point estimates vary due to sampling variation are called *sampling distributions*. The only additional thing to keep in mind about null distributions is that they are sampling distributions *assuming the null hypothesis H_0 is true*.

In our case, we previously visualized a null distribution in Figure 4.3, which we re-display in Figure 4.4 using our new notation and terminology. It is the distribution of the 16 differences in sample proportions our friends computed *assuming* a hypothetical universe of no gender discrimination. We also mark the value of the observed test statistic of 0.292 with a vertical line.

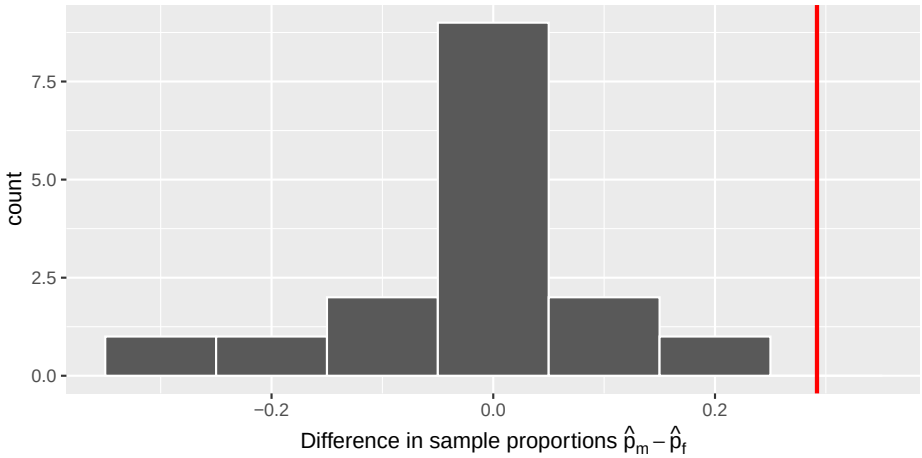

Figura 4.4: Null distribution and observed test statistic.

Sixth, the ***p*-value** is the probability of obtaining a test statistic just as extreme or more extreme than the observed test statistic *assuming the null hypothesis H_0 is true*. Double ooof! Let's unpack this slowly as well. You can think of the p-value as a quantification of "surprise": assuming H_0 is true, how surprised are we with what we observed? Or in our case, in our hypothesized universe of no gender discrimination, how surprised are we that we observed a difference in promotion rates of 0.292 from our collected samples assuming H_0 is true? Very surprised? Somewhat surprised?

The p-value quantifies this probability, or in the case of our 16 differences in sample proportions in Figure 4.4, what proportion had a more "extreme" result? Here, extreme is defined in terms of the alternative hypothesis H_A that "male" applicants are promoted at a higher rate than "female" applicants. In other words, how often was the discrimination in favor of men *even more* pronounced than $0.875 - 0.583 = 0.292 = 29.2\%$?

In this case, 0 times out of 16, we obtained a difference in proportion greater than or equal to the observed difference of $0.292 = 29.2\%$. A very rare (in fact, not occurring) outcome! Given the rarity of such a pronounced difference in

promotion rates in our hypothesized universe of no gender discrimination, we're inclined to *reject* our hypothesized universe. Instead, we favor the hypothesis stating there is discrimination in favor of the "male" applicants. In other words, we reject H_0 in favor of H_A.

Seventh and lastly, in many hypothesis testing procedures, it is commonly recommended to set the **significance level** of the test beforehand. It is denoted by the Greek letter α (pronounced "alpha"). This value acts as a cutoff on the p-value, where if the p-value falls below α, we would "reject the null hypothesis H_0."

Alternatively, if the p-value does not fall below α, we would "fail to reject H_0." Note the latter statement is not quite the same as saying we "accept H_0." This distinction is rather subtle and not immediately obvious. So we'll revisit it later in Section 4.4.

While different fields tend to use different values of α, some commonly used values for α are 0.1, 0.01, and 0.05; with 0.05 being the choice people often make without putting much thought into it. We'll talk more about α significance levels in Section 4.4, but first let's fully conduct the hypothesis test corresponding to our promotions activity using the `infer` package.

4.3 Conducting hypothesis tests

In Section **??**, we showed you how to construct confidence intervals. We first illustrated how to do this using `dplyr` data wrangling verbs and the `rep_sample_n()` function from Subsection **??** which we used as a virtual shovel. In particular, we constructed confidence intervals by resampling with replacement by setting the `replace = TRUE` argument to the `rep_sample_n()` function.

We then showed you how to perform the same task using the `infer` package workflow. While both workflows resulted in the same bootstrap distribution from which we can construct confidence intervals, the `infer` package workflow emphasizes each of the steps in the overall process in Figure **??**. It does so using function names that are intuitively named with verbs:

1. `specify()` the variables of interest in your data frame.
2. `generate()` replicates of bootstrap resamples with replacement.
3. `calculate()` the summary statistic of interest.
4. `visualize()` the resulting bootstrap distribution and confidence interval.

In this section, we'll now show you how to seamlessly modify the previously seen `infer` code for constructing confidence intervals to conduct hypothesis tests. You'll notice that the basic outline of the workflow is almost identical, except for an additional `hypothesize()` step between the `specify()` and `generate()` steps, as can be seen in Figure **??**.

Furthermore, we'll use a pre-specified significance level $\alpha = 0.05$ for this hypothesis test. Let's leave discussion on the choice of this α value until later on in Section 4.4.

4.3.1 `infer` package workflow

1. `specify` variables

Recall that we use the `specify()` verb to specify the response variable and, if needed, any explanatory variables for our study. In this case, since we are interested in any potential effects of gender on promotion decisions, we set `decision` as the response variable and `gender` as the explanatory variable. We do so using `formula = response ~ explanatory` where `response` is the name of the response variable in the data frame and `explanatory` is the name of the explanatory variable. So in our case it is `decision ~ gender`.

Furthermore, since we are interested in the proportion of résumés `"promoted"`, and not the proportion of résumés `not` promoted, we set the argument `success` to `"promoted"`.

```
promotions %>%
  specify(formula = decision ~ gender, success = "promoted")
```

```
## Response: decision (factor)
## Explanatory: gender (factor)
## # A tibble: 48 x 2
##    decision gender
##    <fct>    <fct>
##  1 promoted male
##  2 promoted male
##  3 promoted male
##  4 promoted male
##  5 promoted male
##  6 promoted male
##  7 promoted male
##  8 promoted male
##  9 promoted male
## 10 promoted male
## # ... with 38 more rows
```

Again, notice how the `promotions` data itself doesn't change, but the `Response: decision (factor)` and `Explanatory: gender (factor)` *meta-data* do. This is similar to how the `group_by()` verb from `dplyr` doesn't change the data, but only adds "grouping" meta-data, as we saw in Section **??**.

2. `hypothesize` the null

In order to conduct hypothesis tests using the `infer` workflow, we need a new step not present for confidence intervals: `hypothesize()`. Recall from Section 4.2 that our hypothesis test was

$$H_0 : p_m - p_f = 0$$
$$\text{vs. } H_A : p_m - p_f > 0$$

In other words, the null hypothesis H_0 corresponding to our "hypothesized universe" stated that there was no difference in gender-based discrimination rates. We set this null hypothesis H_0 in our `infer` workflow using the `null` argument of the `hypothesize()` function to either:

- `"point"` for hypotheses involving a single sample or
- `"independence"` for hypotheses involving two samples.

In our case, since we have two samples (the résumés with "male" and "female" names), we set `null = "independence"`.

```
promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  hypothesize(null = "independence")
```

```
## Response: decision (factor)
## Explanatory: gender (factor)
## Null Hypothesis: independence
## # A tibble: 48 x 2
##    decision gender
##    <fct>    <fct>
##  1 promoted male
##  2 promoted male
##  3 promoted male
##  4 promoted male
##  5 promoted male
##  6 promoted male
##  7 promoted male
##  8 promoted male
##  9 promoted male
## 10 promoted male
## # ... with 38 more rows
```

Again, the data has not changed yet. This will occur at the upcoming `generate()` step; we're merely setting meta-data for now.

Where do the terms `"point"` and `"independence"` come from? These are two technical statistical terms. The term "point" relates from the fact that for a single group of observations, you will test the value of a single point. Going

back to the pennies example from Chapter **??**, say we wanted to test if the mean year of all US pennies was equal to 1993 or not. We would be testing the value of a "point" μ, the mean year of *all* US pennies, as follows

$$H_0 : \mu = 1993$$
$$\text{vs } H_A : \mu \neq 1993$$

The term "independence" relates to the fact that for two groups of observations, you are testing whether or not the response variable is *independent* of the explanatory variable that assigns the groups. In our case, we are testing whether the `decision` response variable is "independent" of the explanatory variable `gender` that assigns each résumé to either of the two groups.

3. `generate` replicates

After we `hypothesize()` the null hypothesis, we `generate()` replicates of "shuffled" datasets assuming the null hypothesis is true. We do this by repeating the shuffling exercise you performed in Section 4.1 several times. Instead of merely doing it 16 times as our groups of friends did, let's use the computer to repeat this 1000 times by setting `reps = 1000` in the `generate()` function. However, unlike for confidence intervals where we generated replicates using `type = "bootstrap"` resampling with replacement, we'll now perform shuffles/permutations by setting `type = "permute"`. Recall that shuffles/permutations are a kind of resampling, but unlike the bootstrap method, they involve resampling *without* replacement.

```
promotions_generate <- promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute")
nrow(promotions_generate)
```

```
## [1] 48000
```

Observe that the resulting data frame has 48,000 rows. This is because we performed shuffles/permutations for each of the 48 rows 1000 times and $48,000 = 1000 \cdot 48$. If you explore the `promotions_generate` data frame with `View()`, you'll notice that the variable `replicate` indicates which resample each row belongs to. So it has the value `1` 48 times, the value `2` 48 times, all the way through to the value `1000` 48 times.

4. `calculate` summary statistics

Now that we have generated 1000 replicates of "shuffles" assuming the null hypothesis is true, let's `calculate()` the appropriate summary statistic for each of our 1000 shuffles. From Section 4.2, point estimates related to hypothesis testing have a specific name: *test statistics*. Since the unknown population

parameter of interest is the difference in population proportions $p_m - p_f$, the test statistic here is the difference in sample proportions $\hat{p}_m - \hat{p}_f$.

For each of our 1000 shuffles, we can calculate this test statistic by setting `stat = "diff in props"`. Furthermore, since we are interested in $\hat{p}_m - \hat{p}_f$ we set `order = c("male", "female")`. As we stated earlier, the order of the subtraction does not matter, so long as you stay consistent throughout your analysis and tailor your interpretations accordingly.

Let's save the result in a data frame called `null_distribution`:

```
null_distribution <- promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute") %>%
  calculate(stat = "diff in props", order = c("male", "female"))
null_distribution
```

```
## # A tibble: 1,000 x 2
##    replicate    stat
##        <int>   <dbl>
##  1         1 -0.0417
##  2         2 -0.125
##  3         3 -0.125
##  4         4 -0.0417
##  5         5 -0.0417
##  6         6 -0.125
##  7         7 -0.125
##  8         8 -0.125
##  9         9 -0.0417
## 10        10 -0.0417
## # ... with 990 more rows
```

Observe that we have 1000 values of `stat`, each representing one instance of $\hat{p}_m - \hat{p}_f$ in a hypothesized world of no gender discrimination. Observe as well that we chose the name of this data frame carefully: `null_distribution`. Recall once again from Section 4.2 that sampling distributions when the null hypothesis H_0 is assumed to be true have a special name: the *null distribution*.

What was the *observed* difference in promotion rates? In other words, what was the *observed test statistic* $\hat{p}_m - \hat{p}_f$? Recall from Section 4.1 that we computed this observed difference by hand to be 0.875 - 0.583 = 0.292 = 29.2%. We can also compute this value using the previous `infer` code but with the `hypothesize()` and `generate()` steps removed. Let's save this in `obs_diff_prop`:

```
obs_diff_prop <- promotions %>%
  specify(decision ~ gender, success = "promoted") %>%
  calculate(stat = "diff in props", order = c("male", "female"))
obs_diff_prop
```

```
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1 0.292
```

5. `visualize` the p-value

The final step is to measure how surprised we are by a promotion difference of 29.2% in a hypothesized universe of no gender discrimination. If the observed difference of 0.292 is highly unlikely, then we would be inclined to reject the validity of our hypothesized universe.

We start by visualizing the *null distribution* of our 1000 values of $\hat{p}_m - \hat{p}_f$ using `visualize()` in Figure 4.5. Recall that these are values of the difference in promotion rates assuming H_0 is true. This corresponds to being in our hypothesized universe of no gender discrimination.

```
visualize(null_distribution, bins = 10)
```

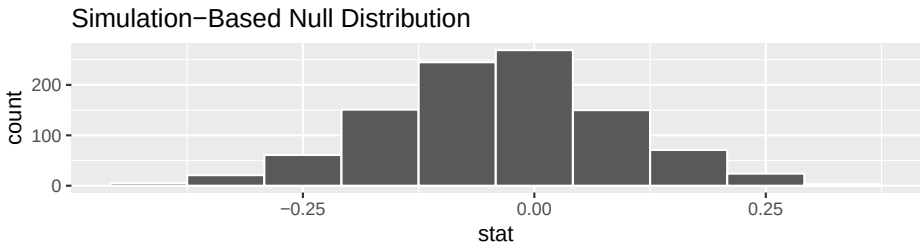

Figura 4.5: Null distribution.

Let's now add what happened in real life to Figure 4.5, the observed difference in promotion rates of 0.875 - 0.583 = 0.292 = 29.2%. However, instead of merely adding a vertical line using `geom_vline()`, let's use the `shade_p_value()` function with `obs_stat` set to the observed test statistic value we saved in `obs_diff_prop`.

Furthermore, we'll set the `direction = "right"` reflecting our alternative hypothesis $H_A : p_m - p_f > 0$. Recall our alternative hypothesis H_A is that $p_m - p_f > 0$, stating that there is a difference in promotion rates in favor of résumés with male names. "More extreme" here corresponds to differences that are "bigger" or "more positive" or "more to the right." Hence we set the `direction` argument of `shade_p_value()` to be `"right"`.

On the other hand, had our alternative hypothesis H_A been the other possible one-sided alternative $p_m - p_f < 0$, suggesting discrimination in favor of résumés with female names, we would've set `direction = "left"`. Had our alternative hypothesis H_A been two-sided $p_m - p_f \neq 0$, suggesting discrimination in either direction, we would've set `direction = "both"`.

```
visualize(null_distribution, bins = 10) +
  shade_p_value(obs_stat = obs_diff_prop, direction = "right")
```

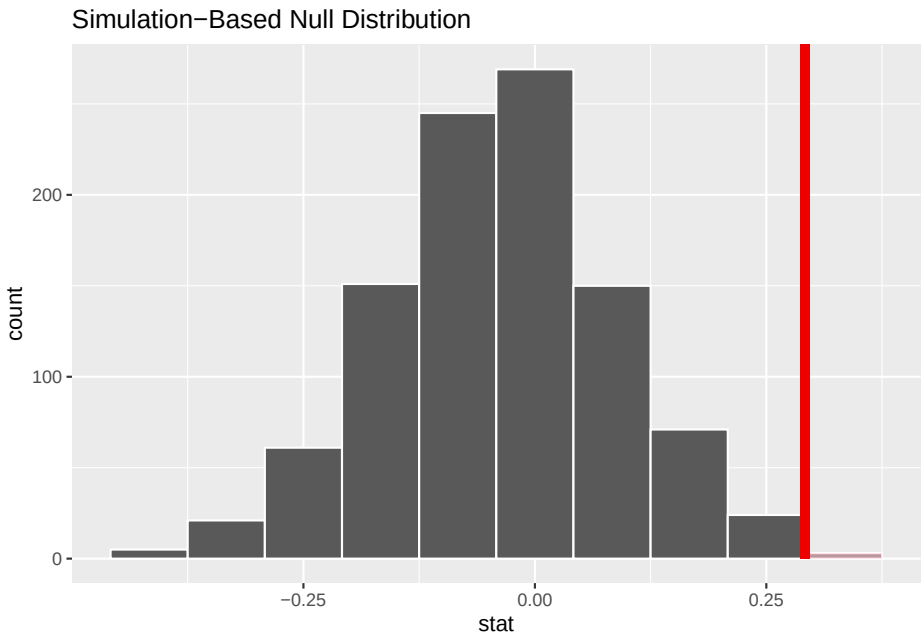

Figura 4.6: Shaded histogram to show p-value.

In the resulting Figure 4.6, the solid dark line marks $0.292 = 29.2\%$. However, what does the shaded-region correspond to? This is the *p-value*. Recall the definition of the p-value from Section 4.2:

> A p-value is the probability of obtaining a test statistic just as or more extreme than the observed test statistic *assuming the null hypothesis H_0 is true*. So judging by the shaded region in Figure 4.6, it seems we would somewhat rarely observe differences in promotion rates of $0.292 = 29.2\%$ or more in a hypothesized universe of no gender discrimination. In other words, the p-value is somewhat small. Hence, we would be inclined to reject this hypothesized universe, or using statistical language we would "reject H_0."

What fraction of the null distribution is shaded? In other words, what is the exact value of the p-value? We can compute it using the `get_p_value()` function with the same arguments as the previous `shade_p_value()` code:

```
null_distribution %>%
  get_p_value(obs_stat = obs_diff_prop, direction = "right")
```

```
## # A tibble: 1 x 1
```

```
##   p_value
##     <dbl>
## 1   0.027
```

Keeping the definition of a p-value in mind, the probability of observing a difference in promotion rates as large as $0.292 = 29.2\%$ due to sampling variation alone in the null distribution is $0.027 = 2.7\%$. Since this p-value is smaller than our pre-specified significance level $\alpha = 0.05$, we reject the null hypothesis $H_0 : p_m - p_f = 0$. In other words, this p-value is sufficiently small to reject our hypothesized universe of no gender discrimination. We instead have enough evidence to change our mind in favor of gender discrimination being a likely culprit here. Observe that whether we reject the null hypothesis H_0 or not depends in large part on our choice of significance level α. We'll discuss this more in Subsection 4.4.3.

4.3.2 Comparison with confidence intervals

One of the great things about the `infer` package is that we can jump seamlessly between conducting hypothesis tests and constructing confidence intervals with minimal changes! Recall the code from the previous section that creates the null distribution, which in turn is needed to compute the p-value:

```
null_distribution <- promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  hypothesize(null = "independence") %>%
  generate(reps = 1000, type = "permute") %>%
  calculate(stat = "diff in props", order = c("male", "female"))
```

To create the corresponding bootstrap distribution needed to construct a 95% confidence interval for $p_m - p_f$, we only need to make two changes. First, we remove the `hypothesize()` step since we are no longer assuming a null hypothesis H_0 is true. We can do this by deleting or commenting out the `hypothesize()` line of code. Second, we switch the `type` of resampling in the `generate()` step to be `"bootstrap"` instead of `"permute"`.

```
bootstrap_distribution <- promotions %>%
  specify(formula = decision ~ gender, success = "promoted") %>%
  # Change 1 - Remove hypothesize():
  # hypothesize(null = "independence") %>%
  # Change 2 - Switch type from "permute" to "bootstrap":
  generate(reps = 1000, type = "bootstrap") %>%
  calculate(stat = "diff in props", order = c("male", "female"))
```

Using this `bootstrap_distribution`, let's first compute the percentile-based confidence intervals, as we did in Section **??**:

```
percentile_ci <- bootstrap_distribution %>%
  get_confidence_interval(level = 0.95, type = "percentile")
```

```
percentile_ci
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1   0.0635    0.525
```

Using our shorthand interpretation for 95% confidence intervals from Subsection **??**, we are 95% "confident" that the true difference in population proportions $p_m - p_f$ is between (0.063, 0.525). Let's visualize `bootstrap_distribution` and this percentile-based 95% confidence interval for $p_m - p_f$ in Figure 4.7.

```
visualize(bootstrap_distribution) +
  shade_confidence_interval(endpoints = percentile_ci)
```

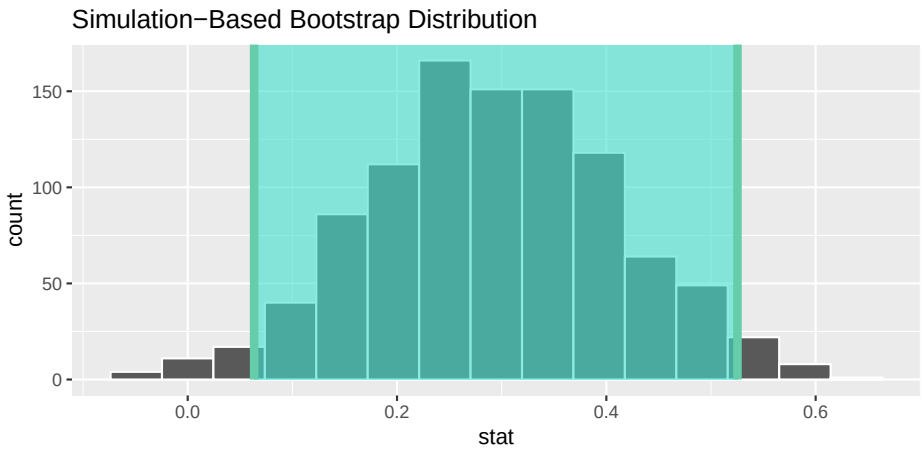

Figura 4.7: Percentile-based 95% confidence interval.

Notice a key value that is not included in the 95% confidence interval for $p_m - p_f$: the value 0. In other words, a difference of 0 is not included in our net, suggesting that p_m and p_f are truly different! Furthermore, observe how the entirety of the 95% confidence interval for $p_m - p_f$ lies above 0, suggesting that this difference is in favor of men.

Since the bootstrap distribution appears to be roughly normally shaped, we can also use the standard error method as we did in Section **??**. In this case, we must specify the `point_estimate` argument as the observed difference in promotion rates $0.292 = 29.2\%$ saved in `obs_diff_prop`. This value acts as the center of the confidence interval.

```
se_ci <- bootstrap_distribution %>%
  get_confidence_interval(level = 0.95, type = "se",
                          point_estimate = obs_diff_prop)
```

```
se_ci
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1   0.0587    0.525
```

Let's visualize `bootstrap_distribution` again, but now the standard error based 95% confidence interval for $p_m - p_f$ in Figure 4.8. Again, notice how the value 0 is not included in our confidence interval, again suggesting that p_m and p_f are truly different!

```
visualize(bootstrap_distribution) +
  shade_confidence_interval(endpoints = se_ci)
```

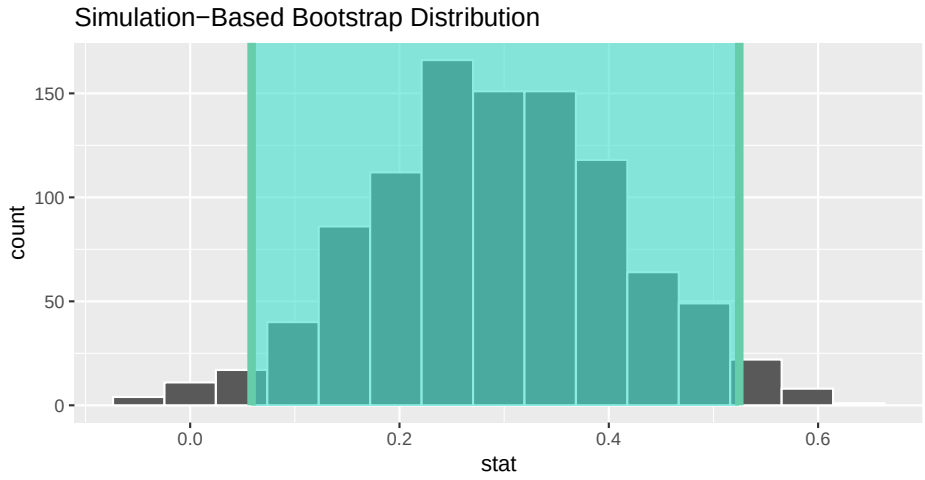

Figura 4.8: Standard error-based 95% confidence interval.

4.3.3 "There is only one test"

Let's recap the steps necessary to conduct a hypothesis test using the terminology, notation, and definitions related to sampling you saw in Section 4.2 and the `infer` workflow from Subsection 4.3.1:

1. `specify()` the variables of interest in your data frame.
2. `hypothesize()` the null hypothesis H_0. In other words, set a "model for the universe" assuming H_0 is true.
3. `generate()` shuffles assuming H_0 is true. In other words, *simulate* data assuming H_0 is true.
4. `calculate()` the *test statistic* of interest, both for the observed data and your *simulated* data.

5. `visualize()` the resulting *null distribution* and compute the *p-value* by comparing the null distribution to the observed test statistic.

While this is a lot to digest, especially the first time you encounter hypothesis testing, the nice thing is that once you understand this general framework, then you can understand *any* hypothesis test. In a famous blog post, computer scientist Allen Downey called this the "There is only one test" framework, for which he created the flowchart displayed in Figure **??**.

Notice its similarity with the "hypothesis testing with `infer`" diagram you saw in Figure **??**. That's because the `infer` package was explicitly designed to match the "There is only one test" framework. So if you can understand the framework, you can easily generalize these ideas for all hypothesis testing scenarios. Whether for population proportions p, population means μ, differences in population proportions $p_1 - p_2$, differences in population means $\mu_1 - \mu_2$, and as you'll see in Chapter **??** on inference for regression, population regression slopes β_1 as well. In fact, it applies more generally even than just these examples to more complicated hypothesis tests and test statistics as well.

4.4 Interpreting hypothesis tests

Interpreting the results of hypothesis tests is one of the more challenging aspects of this method for statistical inference. In this section, we'll focus on ways to help with deciphering the process and address some common misconceptions.

4.4.1 Two possible outcomes

In Section 4.2, we mentioned that given a pre-specified significance level α there are two possible outcomes of a hypothesis test:

- If the p-value is less than α, then we *reject* the null hypothesis H_0 in favor of H_A.
- If the p-value is greater than or equal to α, we *fail to reject* the null hypothesis H_0.

Unfortunately, the latter result is often misinterpreted as "accepting the null hypothesis H_0." While at first glance it may seem that the statements "failing to reject H_0" and "accepting H_0" are equivalent, there actually is a subtle difference. Saying that we "accept the null hypothesis H_0" is equivalent to stating that "we think the null hypothesis H_0 is true." However, saying that we "fail to reject the null hypothesis H_0" is saying something else: "While H_0 might still be false, we don't have enough evidence to say so." In other words, there is an absence of enough proof. However, the absence of proof is not proof of absence.

To further shed light on this distinction, let's use the United States criminal justice system as an analogy. A criminal trial in the United States is a similar

situation to hypothesis tests whereby a choice between two contradictory claims must be made about a defendant who is on trial:

1. The defendant is truly either "innocent" or "guilty."
2. The defendant is presumed "innocent until proven guilty."
3. The defendant is found guilty only if there is *strong evidence* that the defendant is guilty. The phrase "beyond a reasonable doubt" is often used as a guideline for determining a cutoff for when enough evidence exists to find the defendant guilty.
4. The defendant is found to be either "not guilty" or "guilty" in the ultimate verdict.

In other words, *not guilty* verdicts are not suggesting the defendant is *innocent*, but instead that "while the defendant may still actually be guilty, there wasn't enough evidence to prove this fact." Now let's make the connection with hypothesis tests:

1. Either the null hypothesis H_0 or the alternative hypothesis H_A is true.
2. Hypothesis tests are conducted assuming the null hypothesis H_0 is true.
3. We reject the null hypothesis H_0 in favor of H_A only if the evidence found in the sample suggests that H_A is true. The significance level α is used as a guideline to set the threshold on just how strong of evidence we require.
4. We ultimately decide to either "fail to reject H_0" or "reject H_0."

So while gut instinct may suggest "failing to reject H_0" and "accepting H_0" are equivalent statements, they are not. "Accepting H_0" is equivalent to finding a defendant innocent. However, courts do not find defendants "innocent," but rather they find them "not guilty." Putting things differently, defense attorneys do not need to prove that their clients are innocent, rather they only need to prove that clients are not "guilty beyond a reasonable doubt".

So going back to our résumés activity in Section 4.3, recall that our hypothesis test was $H_0 : p_m - p_f = 0$ versus $H_A : p_m - p_f > 0$ and that we used a pre-specified significance level of $\alpha = 0.05$. We found a p-value of 0.027. Since the p-value was smaller than $\alpha = 0.05$, we rejected H_0. In other words, we found needed levels of evidence in this particular sample to say that H_0 is false at the $\alpha = 0.05$ significance level. We also state this conclusion using non-statistical language: we found enough evidence in this data to suggest that there was gender discrimination at play.

4.4.2 Types of errors

Unfortunately, there is some chance a jury or a judge can make an incorrect decision in a criminal trial by reaching the wrong verdict. For example, finding a truly innocent defendant "guilty". Or on the other hand, finding a truly guilty defendant "not guilty." This can often stem from the fact that prosecutors don't have access to all the relevant evidence, but instead are limited to whatever evidence the police can find.

The same holds for hypothesis tests. We can make incorrect decisions about a population parameter because we only have a sample of data from the population and thus sampling variation can lead us to incorrect conclusions.

There are two possible erroneous conclusions in a criminal trial: either (1) a truly innocent person is found guilty or (2) a truly guilty person is found not guilty. Similarly, there are two possible errors in a hypothesis test: either (1) rejecting H_0 when in fact H_0 is true, called a **Type I error** or (2) failing to reject H_0 when in fact H_0 is false, called a **Type II error**. Another term used for "Type I error" is "false positive," while another term for "Type II error" is "false negative."

This risk of error is the price researchers pay for basing inference on a sample instead of performing a census on the entire population. But as we've seen in our numerous examples and activities so far, censuses are often very expensive and other times impossible, and thus researchers have no choice but to use a sample. Thus in any hypothesis test based on a sample, we have no choice but to tolerate some chance that a Type I error will be made and some chance that a Type II error will occur.

To help understand the concepts of Type I error and Type II errors, we apply these terms to our criminal justice analogy in Figure **??**.

Thus a Type I error corresponds to incorrectly putting a truly innocent person in jail, whereas a Type II error corresponds to letting a truly guilty person go free. Let's show the corresponding table in Figure **??** for hypothesis tests.

4.4.3 How do we choose alpha?

If we are using a sample to make inferences about a population, we run the risk of making errors. For confidence intervals, a corresponding "error" would be constructing a confidence interval that does not contain the true value of the population parameter. For hypothesis tests, this would be making either a Type I or Type II error. Obviously, we want to minimize the probability of either error; we want a small probability of making an incorrect conclusion:

- The probability of a Type I Error occurring is denoted by α. The value of α is called the *significance level* of the hypothesis test, which we defined in Section 4.2.
- The probability of a Type II Error is denoted by β. The value of $1 - \beta$ is known as the *power* of the hypothesis test.

In other words, α corresponds to the probability of incorrectly rejecting H_0 when in fact H_0 is true. On the other hand, β corresponds to the probability of incorrectly failing to reject H_0 when in fact H_0 is false.

Ideally, we want $\alpha = 0$ and $\beta = 0$, meaning that the chance of making either error is 0. However, this can never be the case in any situation where we are sampling for inference. There will always be the possibility of making either

error when we use sample data. Furthermore, these two error probabilities are inversely related. As the probability of a Type I error goes down, the probability of a Type II error goes up.

What is typically done in practice is to fix the probability of a Type I error by pre-specifying a significance level α and then try to minimize β. In other words, we will tolerate a certain fraction of incorrect rejections of the null hypothesis H_0, and then try to minimize the fraction of incorrect non-rejections of H_0.

So for example if we used $\alpha = 0.01$, we would be using a hypothesis testing procedure that in the long run would incorrectly reject the null hypothesis H_0 one percent of the time. This is analogous to setting the confidence level of a confidence interval.

So what value should you use for α? Different fields have different conventions, but some commonly used values include 0.10, 0.05, 0.01, and 0.001. However, it is important to keep in mind that if you use a relatively small value of α, then all things being equal, p-values will have a harder time being less than α. Thus we would reject the null hypothesis less often. In other words, we would reject the null hypothesis H_0 only if we have *very strong* evidence to do so. This is known as a "conservative" test.

On the other hand, if we used a relatively large value of α, then all things being equal, p-values will have an easier time being less than α. Thus we would reject the null hypothesis more often. In other words, we would reject the null hypothesis H_0 even if we only have *mild* evidence to do so. This is known as a "liberal" test.

4.5 Conclusions

4.5.1 When inference is not needed

We've now walked through several different examples of how to use the `infer` package to perform statistical inference: constructing confidence intervals and conducting hypothesis tests. For each of these examples, we made it a point to always perform an exploratory data analysis (EDA) first; specifically, by looking at the raw data values, by using data visualization with `ggplot2`, and by data wrangling with `dplyr` beforehand. We *highly* encourage you to always do the same. As a beginner to statistics, EDA helps you develop intuition as to what statistical methods like confidence intervals and hypothesis tests can tell us. Even as a seasoned practitioner of statistics, EDA helps guide your statistical investigations. In particular, is statistical inference even needed?

Let's consider an example. Say we're interested in the following question: Of *all* flights leaving a New York City airport, are Hawaiian Airlines flights in the air for longer than Alaska Airlines flights? Furthermore, let's assume that 2013 flights are a representative sample of all such flights. Then we can use the `flights` data frame in the `nycflights13` package we introduced in Section **??**

to answer our question. Let's filter this data frame to only include Hawaiian and Alaska Airlines using their `carrier` codes `HA` and `AS`:

```
flights_sample <- flights %>%
  filter(carrier %in% c("HA", "AS"))
```

There are two possible statistical inference methods we could use to answer such questions. First, we could construct a 95% confidence interval for the difference in population means $\mu_{HA} - \mu_{AS}$, where μ_{HA} is the mean air time of all Hawaiian Airlines flights and μ_{AS} is the mean air time of all Alaska Airlines flights. We could then check if the entirety of the interval is greater than 0, suggesting that $\mu_{HA} - \mu_{AS} > 0$, or, in other words suggesting that $\mu_{HA} > \mu_{AS}$. Second, we could perform a hypothesis test of the null hypothesis $H_0 : \mu_{HA} - \mu_{AS} = 0$ versus the alternative hypothesis $H_A : \mu_{HA} - \mu_{AS} > 0$.

However, let's first construct an exploratory visualization as we suggested earlier. Since `air_time` is numerical and `carrier` is categorical, a boxplot can display the relationship between these two variables, which we display in Figure 4.9.

```
ggplot(data = flights_sample, mapping = aes(x = carrier, y = air_time)) +
  geom_boxplot() +
  labs(x = "Carrier", y = "Air Time")
```

```
## Warning: Removed 5 rows containing non-finite values (stat_boxplot).
```

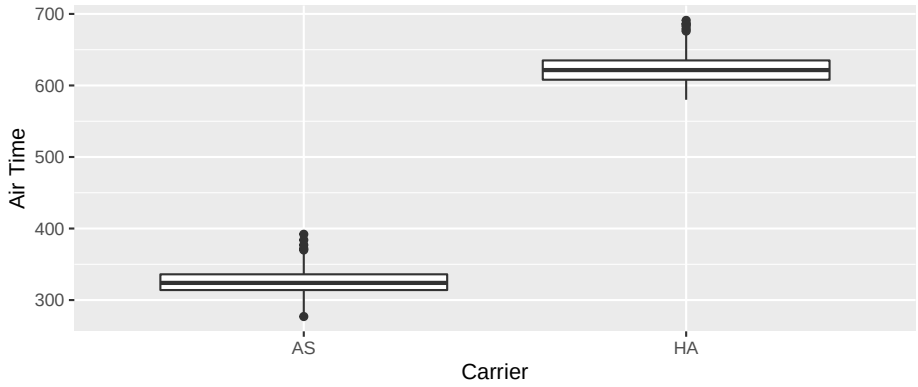

Figura 4.9: Air time for Hawaiian and Alaska Airlines flights departing NYC in 2013.

This is what we like to call "no PhD in Statistics needed" moments. You don't have to be an expert in statistics to know that Alaska Airlines and Hawaiian Airlines have *significantly* different air times. The two boxplots don't even overlap! Constructing a confidence interval or conducting a hypothesis test would frankly not provide much more insight than Figure 4.9.

Let's investigate why we observe such a clear cut difference between these two

airlines using data wrangling. Let's first group by the rows of `flights_sample` not only by `carrier` but also by destination `dest`. Subsequently, we'll compute two summary statistics: the number of observations using `n()` and the mean airtime:

```
flights_sample %>%
  group_by(carrier, dest) %>%
  summarize(n = n(), mean_time = mean(air_time, na.rm = TRUE))
```

```
## `summarise()` regrouping output by 'carrier' (override with `.groups` argument)
```

```
## # A tibble: 2 x 4
## # Groups:   carrier [2]
##   carrier dest      n mean_time
##   <chr>   <chr> <int>     <dbl>
## 1 AS      SEA     714      326.
## 2 HA      HNL     342      623.
```

It turns out that from New York City in 2013, Alaska only flew to `SEA` (Seattle) from New York City (NYC) while Hawaiian only flew to `HNL` (Honolulu) from NYC. Given the clear difference in distance from New York City to Seattle versus New York City to Honolulu, it is not surprising that we observe such different (*statistically significantly different*, in fact) air times in flights.

This is a clear example of not needing to do anything more than a simple exploratory data analysis using data visualization and descriptive statistics to get an appropriate conclusion. This is why we highly recommend you perform an EDA of any sample data before running statistical inference methods like confidence intervals and hypothesis tests.

4.5.2 Problems with p-values

On top of the many common misunderstandings about hypothesis testing and p-values we listed in Section 4.4, another unfortunate consequence of the expanded use of p-values and hypothesis testing is a phenomenon known as "p-hacking." p-hacking is the act of "cherry-picking" only results that are "statistically significant" while dismissing those that aren't, even if at the expense of the scientific ideas. There are lots of articles written recently about misunderstandings and the problems with p-values. We encourage you to check some of them out:

1. Misunderstandings of p-values
2. What a nerdy debate about p-values shows about science - and how to fix it
3. Statisticians issue warning over misuse of P values
4. You Can't Trust What You Read About Nutrition
5. A Litany of Problems with p-values

Such issues were getting so problematic that the American Statistical

Association (ASA) put out a statement in 2016 titled, "The ASA Statement on Statistical Significance and *P*-Values," with six principles underlying the proper use and interpretation of *p*-values. The ASA released this guidance on *p*-values to improve the conduct and interpretation of quantitative science and to inform the growing emphasis on reproducibility of science research.

We as authors much prefer the use of confidence intervals for statistical inference, since in our opinion they are much less prone to large misinterpretation. However, many fields still exclusively use *p*-values for statistical inference and this is one reason for including them in this text. We encourage you to learn more about "p-hacking" as well and its implication for science.

Capítulo 5

El muestreo estadístico

Some *significant* applications are demonstrated in this chapter.

5.1 Example one

5.2 Example two

Capítulo 6

Regresiones

We have finished a nice book.

Capítulo 7

Estadística multivariante

We have finished a nice book.