David J. Stanley

Welcome!

To my students,

from whom I've learn so much about teaching.

Contents

List of Tables

List of Figures

Course:

PSYC4780 Advanced Research Methods & Statistics

FALL 2020

These notes are a resource for students in my PSYC*4780 class. The notes are a subset of material that I am creating for a public domain statistics book to be released sometime in the distant future.

About the Author

David J. Stanley is an Associate Professor of Industrial and Organizational Psychology at the University of Guelph in Canada. He obtained his PhD from Western University in London, Ontario. David has published articles in Advances in Methods and Practices in Psychological Science, Organizational Research Methods, Journal of Applied Psychology, Perspectives in Psychological Science, Journal of Business and Psychology, Journal of Vocational Behaviour, Journal of Personality and Social Psychology, Behavior Research Methods, Industrial and Organizational Psychology, and Emotion among other journals. David also created the apaTables R package.

1

Introduction

Welcome! In this guide, we will teach you about statistics using the statistical software R with the interface provided by R Studio. The purpose of this chapter to is provide you with a set of activities that get you up-and-running in R quickly so get a sense of how it works. In later chatpers we will revisit these same topics in more detail.

1.1 A focus on workflow

An important part of this guide is training you in a workflow that will avoid many problems than can occur when using R.

1.2 R works with plug-ins

R is a statistical language with many plug-ins called **packages** that you will use for analyses. You can think of R as being like your smartphone. To do things with your phone you need **an App** (R equivalent: a *package*) from the App Store (R equivalent: *CRAN*). Apps need to be **downloaded** (R equivalent: *install.packages*) before you can use them. To use the app you need **Open** it (R equivalent: *library command*). These similarities are illustrated in Table **??** below.

TABLE 1.1: R packages are similar to smart phone apps (Kim, 2018)

Smart Phone Terminology	R Terminology
App	package
App Store	CRAN
Download App from App Store	install.packages("apaTables", dependencies = TRUE)
Open App	library("apaTables")

1.3 Create an account at R Studio Cloud

R Studio Cloud[1] accounts are free and required for this guide. Please go to the website and set up a new account.

1.4 Join the class workspace

To do the assignment required for this class you need to join the class workspace on R Studio Cloud. To do so:

1. Log into R Studio Cloud (if you haven't already done so).
2. Go to your university email account and find the message with the subject "R Studio Workspace Invitation". In this message there is a link to the class R Studio Cloud workspace.
3. Click on the workspace link in the email or paste it into your web browser. You should see a screen like the one below in Figure **??**. Click on the Join button.
4. Then you should see the welcome message illustrated in Figure **??**. Above this message is the Projects menu option. Click on the word Project.
5. You should now see the First Project displayed as in **??**. Click the Start button. You will then move to a view of R Studio.

[1] http://www.rstudio.cloud

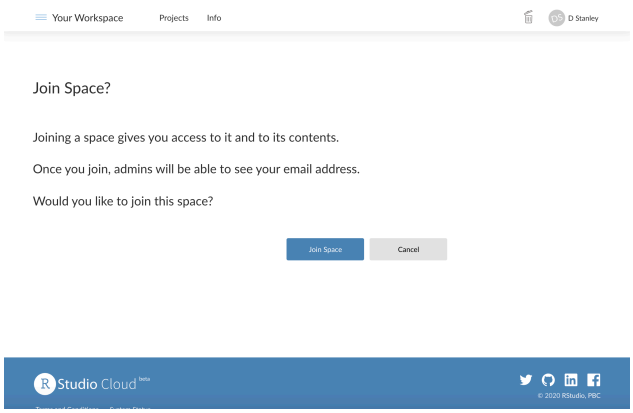


FIGURE 1.1: Screenshot of workspace join message

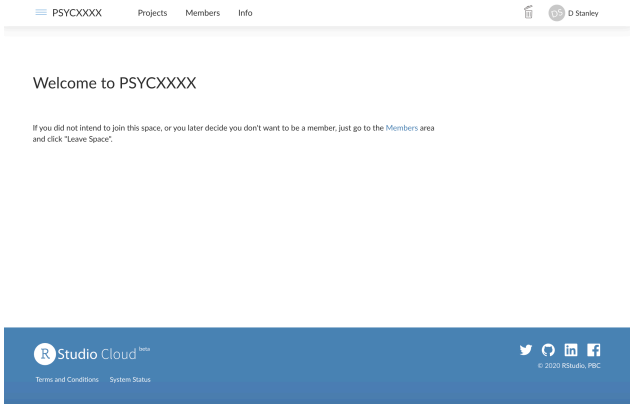


FIGURE 1.2: Screenshot of welcome message

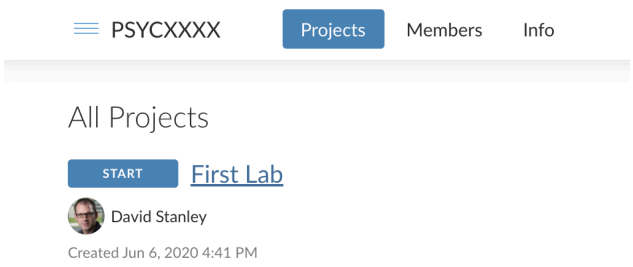


FIGURE 1.3: Screenshot of starting first assignment

5. In R Studio it is essential you use projects to keep your files organized and in the same spot. For this course, when your start an assignment on R Studio Cloud and the project will already have been made for you. Later you will learn to make your own R Studio Projects.

1.5 Exploring the R Studio Interface

Once you have opened (or created) a Project folder, you are presented with the R Studio interface. There are a few key elements to the user interface that are illustrated in Figure **??** In the lower right of the screen you can see the a panel with several tabs (i.e., Files, Plots, Packages, etc) that I will refer to as the Files pane. You look in this pane to see all the files associated with your project. On the left side of the screen is the Console which is an interactive pane where you type and obtain results in real time. I've placed two large grey blocks on the screen with text to more clearly identify the Console and Files panes. Not shown in this figure is the Script panel where we can store our commands for later reuse.

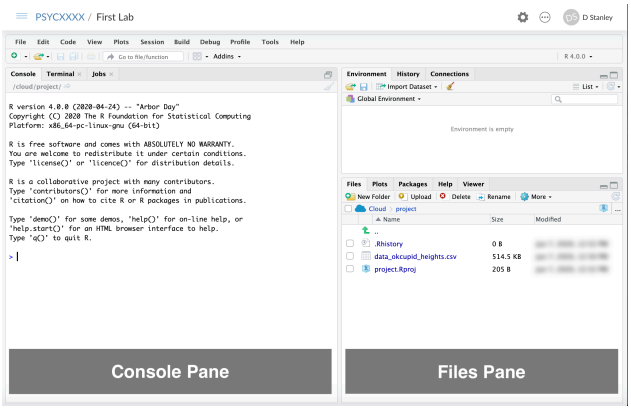


FIGURE 1.4: R Studio interface

1.5.1 Console panel

When you first start R, the Console panel is on the left side of the screen. Sometimes there are two panels on the left side (one above the other); if so, the Console panel is the lower one (and labeled accordingly). We can use R

a bit like a calculator. Try typing the following into the Console window: 8 + 6 + 7 + 5. You can see that R immediately produced the result on a line preceded by two hashtags (##).

```
8 + 6 + 7 + 5
```

```
## [1] 26
```

We can also put the result into a variable to store it. Later we can use the print command to see that result. In the example below we add the numbers 3, 0, and 9 and store the result in the variable my_sum. The text "<-" indicate you are putting what is on the right side of the arrow into the variable on the left side of the arrow. You can think of a variable as cup into which you can put different things. In this case, imagine a real-world cup with my_sum written on the outside and inside the cup we have stored the sum of 3, 0, and 9 (i.e., 12).

```
my_sum <- 3 + 0 + 9
```

We can inspect the contents of the my_sum variable (i.e., my_sum cup) with the print command:

```
print(my_sum)
```

```
## [1] 12
```

Variable are very useful in R. We will use them to store a single number, an entire data set, the results of an analysis, or anything else.

1.5.2 Script Panel

Although you can use R with just with the Console panel, it's a better idea to use scripts via the Script panel - not visible yet. Scripts are just text files with the commands you use stored in them. You can run a script (as you will see below) using the Run or Source buttons located in the top right of the Script panel.

Scripts are valuable because if you need to run an analysis a second time you don't have to type the command in a second time. You can run the script again and again without retyping your commands. More importantly though, the script provides a record of your analyses.

A common problem in science is that after an article is published, the authors can't reproduce the numbers in the paper. You can read more about the

important problem in a surprising article in the journal Molecular Brain[2]. In this article an editor reports how a request for the data underlying articles resulted in the wrong data for 40 out of 41 papers. Long story short – keep track of the data and scripts you use for your paper. In a later chapter, it's generally poor practice to manipulate or modify or analyze your data using any menu driven software because this approach does not provide a record of what you have done.

1.6 Writing your first script

1.6.1 Create the script file

Create a script in your R Studio project by using the menu File > New File > R Script.

Save the file with an appropriate name using the File menu. The file will be saved in your Project folder. A common, and good, convention for naming is to start all script names with the word "script" and separate words with an underscore. You might save this first script file with the name "script_my_first_one.R". The advantage of beginning all script files with the word script is that when you look at your list of files alphabetically, all the script files will cluster together. Likewise, it's a good idea to save all data files such that they begin with "data_". This way all the data files will cluster together in your directory view as well. You can see there is already a data file with this convention called "data_okcupid.csv".

You can see as discussed previously, we are trying to instill an effective workflow as you learn R. Using a good naming convention (that is consistent with what others use) is part of the workflow. When you write your scripts it's a good idea to follow the tidyverse style guide[3] for script names, variable name, file names, and more.

1.6.2 Add a comment to your script

In the previous section you created your first script. We begin by adding a comment to the script. A comment is something that will be read by humans rather than the computer/R. You make comments for other people that will read your code and need to understand what you have done. However, realize

[2] https://molecularbrain.biomedcentral.com/articles/10.1186/s13041-020-0552-2
[3] https://style.tidyverse.org

that you are also making comments for your future self as illustrated in an XKCD cartoon[4].

A good way to start every script is with a comment that includes the date of your script (or even better when you installed your packages, more on this later). Like smartphone apps, packages are updated regularly. Sometimes after a package is updated it will no longer work with an older script. Fortunately, the checkpoint package[5] lets users role back the clock and use older versions of packages. Adding a comment with the date of your script will help future users (including you) to use your script with the same version of the package used when you wrote the script. Dating your script is an important part of an effective and reproducible workflow.

```
# Code written on: YYYY/MM/DD
# By: John Smith
```

Moving forward, I suggest you use comments to make your own personal notes in your own code as your write it. Note that in the above comment I used the internationally accepted date format order Year/Month/Day created by the International Organization for Standardization[6] (ISO). Some people use the mnemonic *Your My Dream* to remember the **Y**ear **M**onth **D**ay order.

[4] https://xkcd.com/1421/
[5] https://cran.r-project.org/web/packages/checkpoint/index.html
[6] https://www.iso.org/home.html

Wikipedia provides more information about this International Date Format (ISO 8601)[7]. An XKCD[8] cartoon highlights the ISO date format:

1.6.3 Background about the tidyverse

There are generally two broad ways of using R, the older way and the newer way. Using R the older way is refered to as using base R. A more modern approach to using R is the tidyverse. The tidyverse represents a collection of packages the work together to give R a modern workflow. These packages do many things to help the data analyst (loading data, rearranging data, graphing, etc.). We will use the tidyverse approach to R in this guide.

A noted the tidyverse is a collection of packges. Each package adds new commands to R. The number of packges and correspondingly the number of new commands added to R by the tidyverse is large. Below is a list of the tidyverse packages:

```
##  [1] "broom"      "cli"        "crayon"     "dbplyr"
##  [5] "dplyr"      "forcats"    "ggplot2"    "haven"
##  [9] "hms"        "httr"       "jsonlite"   "lubridate"
## [13] "magrittr"   "modelr"     "pillar"     "purrr"
## [17] "readr"      "readxl"     "reprex"     "rlang"
## [21] "rstudioapi" "rvest"      "stringr"    "tibble"
## [25] "tidyr"      "xml2"       "tidyverse"
```

Before you can use a package it needs to be installed – this is the same as downloading an app from the App Store. Normally, you can install a **single** packages with the install.packages command. Previously, you needed run

[7] https://en.wikipedia.org/wiki/ISO_8601
[8] https://xkcd.com/1179/

an install.package command for every package in the tidyverse as illustrated below (though we no longer use this approach).

```
# The old way of installing the tidyverse packages
# Like downloading apps from the app store

install.packages("broom", dep = TRUE)
install.packages("cli", dep = TRUE)
install.packages("ggplot", dep = TRUE)
# etc
```

Fortunately, the tidyverse packages can now by installed with a single install.packages command. Specifically, the install.packages command below will install all of the packages listed above.

Class note: For the "First Lab", I've done the install.packages for you. So there is no need to use the install.packages command below in this first lab.

```
install.packages("tidyverse", dep = TRUE)
```

1.6.4 Add library(tidyverse) to your script

The tidyverse is now installed, so we need to activate it. We do that with the library command. Put the library line below at the top of your script file (below your comment):

```
# Code written on: YYYY/MM/DD
# By: John Smith
library(tidyverse)
```

1.6.5 Activate tidyverse auto-complete for your script

Select the library(tidyverse) text with your mouse/track-pad so that it is highlighted. Then click the Run button in the upper right of the Script panel. Doing this "runs" the selected text. After you click the Run button you should see text like the following the Console panel:

```
## -- Attaching packages ---------------------------------------------------------------------
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.2
```

```
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

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

When you use library(tidyverse) to activate the tidyverse you activate the most commonly used subset of the tidyverse packages. In the output you see checkmarks beside names of the tidyverse packages you have activated.

By activating these packages you have added new commands to R that you will use. Sometimes these packages replace older versions of commands in R. The "Conflicts" section in the output shows you where the packages you activated replaced older R commands with newer R commands. You can activate the other tidyverse package by running a library command for each package – if needed. No need to do so now.

Most importantly, running the library(tidyverse) prior to entering the rest of your script allows R Studio to present auto-complete options when typing your text. Remember to start each script with the library(tidyverse) command and then Run it so you get the autocomplete options for the rest of the commands your enter.

1.7 Loading your data

1.7.1 Use read_csv (not read.csv) to open files.

If you inspect the Files pane on the right of the screen you see the **data_okcupid.csv** data file in our project directory. We will load this data with the commands below. If you followed the steps above, you should have auto-complete for the tidyverse commands you type for now in – in the current R session. Enter the command below into your script. As your start to type read_csv you will likely be presented with an auto-complete option. You can use the arrow keys to move up and down the list of options to select the one you want - then press tab to select it.

Once your command looks like the one below select the text and click on the "Run" button.

Note: If you are not in the class, the data file is available from: https://github.com/dstanley4/psyc6060bookdown

```
okcupid_profiles <- read_csv(file = "data_okcupid.csv")
```

```
## Parsed with column specification:
## cols(
##   age = col_double(),
##   diet = col_character(),
##   height = col_double(),
##   pets = col_character(),
##   sex = col_character(),
##   status = col_character()
## )
```

The output indicates that you have loaded a data file and the type of data in each column. The sex column is of type col_character which indicates it contains text/letters. Most of the columns are of the type character. The age and height columns contain numbers are correspondingly indicated to be the type col_double. The label col_double indicates that a column of numbers represented in R with high precision[9]. There are other ways of representing numbers in R but this is the type we will see/use most often.

1.8 Checking out your data

There many ways of viewing the actual data you loaded. A few of these are illustrated now.

1.8.1 view(): See a spreadsheet view of your data

You can inspect your data in a spreadsheet view by using the view command. Do NOT add this command to your script file – EVER. Adding it to the script can cause substantial problems. Type this command in the Console.

```
view(okcupid_profiles)
```

[9] https://en.wikipedia.org/wiki/Double-precision_floating-point_format

1.8.2 print(): See you data in the Console

You can inspect the first few rows of your data with the print() command. It is OK to add a print command to your script. Try the print() command below in the Console:

```
print(okcupid_profiles)
```

```
## # A tibble: 59,946 x 6
##      age diet         height pets              sex   status
##    <dbl> <chr>         <dbl> <chr>             <chr> <chr>
##  1    22 strictly an~     75 likes dogs and l~ m     single
##  2    35 mostly other     70 likes dogs and l~ m     single
##  3    38 anything         68 has cats          m     availa~
##  4    23 vegetarian       71 likes cats        m     single
##  5    29 <NA>             66 likes dogs and l~ m     single
##  6    29 mostly anyt~     67 likes cats        m     single
##  7    32 strictly an~     65 likes dogs and l~ f     single
##  8    31 mostly anyt~     65 likes dogs and l~ f     single
##  9    24 strictly an~     67 likes dogs and l~ f     single
## 10    37 mostly anyt~     65 likes dogs and l~ m     single
## # ... with 59,936 more rows
```

1.8.3 head(): Check out the first few rows of data

You can inspect the first few rows of your data with the head() command. Try the command below in the Console:

```
head(okcupid_profiles)
```

```
## # A tibble: 6 x 6
##     age diet          height pets              sex   status
##   <dbl> <chr>          <dbl> <chr>             <chr> <chr>
## 1    22 strictly any~     75 likes dogs and l~ m     single
## 2    35 mostly other      70 likes dogs and l~ m     single
## 3    38 anything          68 has cats          m     availa~
## 4    23 vegetarian        71 likes cats        m     single
## 5    29 <NA>              66 likes dogs and l~ m     single
## 6    29 mostly anyth~     67 likes cats        m     single
```

You can be even more specific and indicate you only want the first three row of your data with the head() command. Try the command below in the Console:

```
head(okcupid_profiles, 3)
```

```
## # A tibble: 3 x 6
##     age diet          height pets              sex   status
##   <dbl> <chr>          <dbl> <chr>             <chr> <chr>
## 1    22 strictly any~     75 likes dogs and l~ m     single
## 2    35 mostly other      70 likes dogs and l~ m     single
## 3    38 anything          68 has cats          m     availa~
```

1.8.4 tail(): Check out the last few rows of data

You can inspect the last few rows of your data with the tail() command. Try the command below in the Console:

```
tail(okcupid_profiles)
```

```
## # A tibble: 6 x 6
##     age diet         height pets                sex   status
##   <dbl> <chr>         <dbl> <chr>               <chr> <chr>
## 1    31 <NA>             62 likes dogs          f     single
## 2    59 <NA>             62 has dogs            f     single
## 3    24 mostly anyt~     72 likes dogs and lik~ m     single
## 4    42 mostly anyt~     71 <NA>                m     single
## 5    27 mostly anyt~     73 likes dogs and lik~ m     single
## 6    39 <NA>             68 likes dogs and lik~ m     single
```

You can be even more specific and indicate you only want the last three row of your data with the tail() command. Try the command below in the Console:

```
tail(okcupid_profiles, 3)
```

```
## # A tibble: 3 x 6
##     age diet         height pets                sex   status
##   <dbl> <chr>         <dbl> <chr>               <chr> <chr>
## 1    42 mostly anyt~     71 <NA>                m     single
## 2    27 mostly anyt~     73 likes dogs and lik~ m     single
## 3    39 <NA>             68 likes dogs and lik~ m     single
```

1.8.5 summary(): Quick summaries

You can a short summary of your data with the summary() command. Note that we will use the summary() command in many places in the guide. The output of the summary() command changes depending on what you give it - that is put inside the brackets. You can give the summary() command many things such as data, the results of a regression analysis, etc.

Try the command below in the Console. You will see that summary() give the mean and median for each of the numeric variables (age and height).

```
summary(okcupid_profiles)
```

```
##       age            diet               height
##  Min.   : 18.0   Length:59946       Min.   : 1.0
##  1st Qu.: 26.0   Class :character   1st Qu.:66.0
##  Median : 30.0   Mode  :character   Median :68.0
##  Mean   : 32.3                      Mean   :68.3
##  3rd Qu.: 37.0                      3rd Qu.:71.0
##  Max.   :110.0                      Max.   :95.0
##                                     NA's   :3
##      pets               sex              status
##  Length:59946       Length:59946       Length:59946
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##
```

1.9 Run *vs.* Source with Echo *vs.* Source

There are different ways of running commands in R. So far you have used two of these. You can enter them into the Console as we have done already. Or you can put them in your script select the text and clickk the Run button. There are four ways of running commands in your script.

You can:

1. Console: Enter commands directly
2. Script: Select the command(s) and press the Run button.

3. Script: Source (Without Echo)
4. Script: Source With Echo

Two of these approaches involve using the Source button, see Figure **??**. You bring up the options for the Source button, illustrated in this figure, by clicking on the small arrow to the right of the word Source.

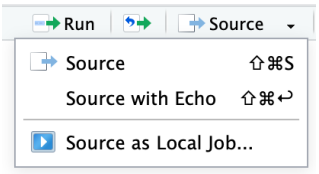


FIGURE 1.5: Source button options

1.9.1 Run select text

The Run button will run the text you highlight and present the relevant output. You have used this command a fair amount already.

I strongly suggest you ONLY use the Run button when testing a command to make sure it works or to debug a script. Or to run library(tidyverse) as you start working on your script so that you get the autocomplete options.

In general, you should always try to execute your R Scripts using the Source with Echo command (preceded by a Restart, see below). This ensures your script will work beginning to end for you in the future and for others that attempt to use it. Using the Run button in an ad lib basis can create output that is not reproducible.

1.9.2 Source (without Echo)

Source (without Echo) is not designed for the typical analysis workflow. It is mostly helpful when you run simulations. When you run Source (without Echo) much of the output you would wish to read is suppressed. In general, avoid this option. If you use it, you often won't see what you want to see in the output.

1.9.3 Source with Echo

The Source with Echo command runs all of the contents of a script and presents the output in the R console. This is the approach you should use to running your scripts in most cases.

Prior to running Source with Echo (or just Source), it's always a good idea to restart R. This makes sure you clear the computer memory of any errors from any previous runs.

So you should do the following EVERY time you run your script.

1. Use the menu item: **Session > Restart R**
2. Click the down arrow beside the Source button, and click on Source With Echo

This will clear potentially problematic previous stats, run the script commands, and display the output in the Console. Moving forward we will use this approach for running scripts. Once you have used Source wiht Echo once, you can just click the Source button and it will use Source with Echo automatically (without the need to use the pull down option for selecting Source with Echo).

Using Restart R before you run a script, or R code in general, is a critical workflow tip.

1.10 Trying Source with Echo

Put the head(), tail(), and summary() command we used previously into your script. Then save your script using using the File > Save menu. You script should appear as below.

```
# Code written on: YYYY/MM/DD
# By: John Smith
library(tidyverse)

okcupid_profiles <- read_csv(file = "data_okcupid.csv")

head(okcupid_profiles)
```

```
tail(okcupid_profiles)

summary(okcupid_profiles)
```

Now do the following:

1. Use the menu item: **Session > Restart R**
2. Click the down arrow beside the Source button, and click on Source With Echo

You should see the output below:

```
# Code written on: YYYY/MM/DD
# By: John Smith
library(tidyverse)

okcupid_profiles <- read_csv(file = "data_okcupid.csv")
```

```
## Parsed with column specification:
## cols(
##   age = col_double(),
##   diet = col_character(),
##   height = col_double(),
##   pets = col_character(),
##   sex = col_character(),
##   status = col_character()
## )
```

```
head(okcupid_profiles)
```

```
## # A tibble: 6 x 6
##     age diet          height pets              sex   status
##   <dbl> <chr>          <dbl> <chr>             <chr> <chr>
## 1    22 strictly any~     75 likes dogs and l~ m     single
## 2    35 mostly other      70 likes dogs and l~ m     single
## 3    38 anything          68 has cats          m     availa~
## 4    23 vegetarian        71 likes cats        m     single
## 5    29 <NA>              66 likes dogs and l~ m     single
## 6    29 mostly anyth~     67 likes cats        m     single
```

```
tail(okcupid_profiles)
```

```
## # A tibble: 6 x 6
##     age diet         height pets                sex   status
##   <dbl> <chr>         <dbl> <chr>               <chr> <chr>
## 1    31 <NA>             62 likes dogs          f     single
## 2    59 <NA>             62 has dogs            f     single
## 3    24 mostly anyt~     72 likes dogs and lik~ m     single
## 4    42 mostly anyt~     71 <NA>                m     single
## 5    27 mostly anyt~     73 likes dogs and lik~ m     single
## 6    39 <NA>             68 likes dogs and lik~ m     single
```

```
summary(okcupid_profiles)
```

```
##       age            diet               height
##  Min.   : 18.0   Length:59946       Min.   : 1.0
##  1st Qu.: 26.0   Class :character   1st Qu.:66.0
##  Median : 30.0   Mode  :character   Median :68.0
##  Mean   : 32.3                      Mean   :68.3
##  3rd Qu.: 37.0                      3rd Qu.:71.0
##  Max.   :110.0                      Max.   :95.0
##                                     NA's   :3
##      pets               sex              status
##  Length:59946       Length:59946       Length:59946
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##
##
##
##
```

Congratulations you just ran your first script!

1.11 A few key points about

Sometimes you will need to send a command additional information. Moreover, that information often needs to be grouped together into a vector or a list before you can send it to the command. We'll learn more about doing so in the future but here is a quick over view of vectors and lists to provide a foundation for future chapters.

1.11.0.1 Vector of numbers

We can create a vector of only numbers using the "c" function - which you can think of as being short for "combine" (or concatenate). In the commands below we create a vector of a few even numbers called "even_numbers".

```
even_numbers <- c(2, 4, 6, 8, 10)
```

```
print(even_numbers)
```

```
## [1]  2  4  6  8 10
```

We can obtain the second number in the vector using the following notation:

```
print(even_numbers[2])
```

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

1.11.0.2 Vector of characters

We can also create vectors using only characters. Note that I use **SHIFT RETURN** after each comma to move to the next line.

```
favourite_things <- c("copper kettles",
                      "woolen mittens",
                      "brown paper packages")
```

```
print(favourite_things)
```

```
## [1] "copper kettles"       "woolen mittens"
## [3] "brown paper packages"
```

As before, can obtain the second item in the vector using the following notation:

```
print(favourite_things[2])
```

```
## [1] "woolen mittens"
```

1.11.1 Lists

Lists are similar to vectors in that you can create them and access items by their numeric position. Vectors must be all characters or all numbers. Lists can be a mix of characters or numbers. Most importantly items in lists can be accessed by their label. Note that I use **SHIFT RETURN** after each comma to move to the next line in the code below.

```
my_list <- list(last_name = "Smith",
                first_name = "John",
                office_number = 1913)

print(my_list)
```

```
## $last_name
## [1] "Smith"
##
## $first_name
## [1] "John"
##
## $office_number
## [1] 1913
```

You can access an item in a list using double brackets:

```
print(my_list[2])
```

```
## $first_name
## [1] "John"
```

You can access an item in a list by its label/name using the dollar sign:

```
print(my_list$last_name)
```

```
## [1] "Smith"
```

```
print(my_list$office_number)
```

```
## [1] 1913
```

1.12 That's it!

Congratulations! You've reached the end of the introduction to R. Take a break, have a cookie, and read some more about R tomorrow!

2

Handling Data with the Tidyverse

2.1 Required

The data files below are used in this chapter. The files are available at: https://github.com/dstanley4/psyc6060bookdown

Required Data
data_okcupid.csv
data_experiment.csv

The following CRAN packages must be installed:

Required CRAN Packages
tidyverse

2.2 Objective

The objective of this chapter is to familiarize you with some key commands in the tidyverse. These commands are used in isolation of each other for the most part. In the next chapter we will use these commands in a more coordinated way as we load a data set and move it from raw data to data that is ready for analysis (i.e., analytic data). You can start this project by Starting the class assignment on R Studio Cloud that corresponds to the chapter name.

2.3 Using the Console

All of the commands in this chapter should be typed into the Console within R. If you see a command split over multiple lines, use SHIFT-RETURN (macOS) or SHIFT-ENTER (Windows) to move the next line that is part of the same command.

2.4 Basic tidyverse commands

If you inspect the Files tab on the lower-right panel in R Studio you will see the file data_okcupid.csv. The code below loads that file.

```
library(tidyverse)
okcupid_profiles <- read_csv("data_okcupid.csv")
```

You can see the first few rows of the data using the print() command. Each row presents a person whereas each column represents a variable. If you have a large number of columns you will only see the first several columns with this approach to viewing your data.

```
print(okcupid_profiles)
```

```
## # A tibble: 59,946 x 6
##      age diet         height pets              sex   status
##    <dbl> <chr>         <dbl> <chr>             <chr> <chr>
##  1    22 strictly an~     75 likes dogs and l~ m     single
##  2    35 mostly other     70 likes dogs and l~ m     single
##  3    38 anything         68 has cats          m     availa~
##  4    23 vegetarian       71 likes cats        m     single
##  5    29 <NA>             66 likes dogs and l~ m     single
##  6    29 mostly anyt~     67 likes cats        m     single
##  7    32 strictly an~     65 likes dogs and l~ f     single
##  8    31 mostly anyt~     65 likes dogs and l~ f     single
##  9    24 strictly an~     67 likes dogs and l~ f     single
## 10    37 mostly anyt~     65 likes dogs and l~ m     single
## # ... with 59,936 more rows
```

But it's also helpful just to see a list of the columns in the data with the glimpse() command:

```
glimpse(okcupid_profiles)
```

```
## Rows: 59,946
## Columns: 6
## $ age    <dbl> 22, 35, 38, 23, 29, 29, 32, 31, 24, 37, ...
## $ diet   <chr> "strictly anything", "mostly other", "an...
## $ height <dbl> 75, 70, 68, 71, 66, 67, 65, 65, 67, 65, ...
## $ pets   <chr> "likes dogs and likes cats", "likes dogs...
## $ sex    <chr> "m", "m", "m", "m", "m", "m", "f", "f", ...
## $ status <chr> "single", "single", "available", "single...
```

The glimpse() command is useful because it quickly allows you to see all of the columns. Moreover, it allows you to see the type for each column. Types were briefly discussed in the last chapter. Notice in the output beside each column name that some columns are labeled "dbl" which is short for double – a type of numeric column. Other columns are labeled "chr" which is short for character - meaning the columns contain characters. These designations will become important in the next chapter as we prepare data for analysis.

2.4.1 select()

The select() command allows you to obtain a subset of the columns in your data. The commands below can be used to obtain the age and height columns. You can read the command as: take the okcupid_profiles data and then select the age and height columns. The "%>%" symbol can be read as "and then". You can see that this code prints out the data with just the age and height columns. Remember, use SHIFT-ENTER or SHIFT-RETURN to move to the next line in the block of code.

```
okcupid_profiles %>%
  select(age, height)
```

```
## # A tibble: 59,946 x 2
##      age height
##    <dbl>  <dbl>
##  1    22     75
##  2    35     70
##  3    38     68
##  4    23     71
##  5    29     66
```

```
##  6    29     67
##  7    32     65
##  8    31     65
##  9    24     67
## 10    37     65
## # ... with 59,936 more rows
```

Of course, it's usually of little help to just print the subset of the data. It's better to store it in a new data. In the command below we store the resulting data in a new data set called new_data.

```
new_data <- okcupid_profiles %>%
  select(age, height)
```

The glimpse() command shows us that only the age and height columns are in new_data.

```
glimpse(new_data)
```

```
## Rows: 59,946
## Columns: 2
## $ age    <dbl> 22, 35, 38, 23, 29, 29, 32, 31, 24, 37, ...
## $ height <dbl> 75, 70, 68, 71, 66, 67, 65, 65, 67, 65, ...
```

In the above example we indicated the columns we wanted to retain from the okcupid_profiles data using the select() command. However, we can also indicate the columns we want to drop from okcupid_profiles using a minus sign (-) in front of the columns we specify in the select() command.

```
new_data <- okcupid_profiles %>% select(-age, -height)
```

The glimpse() command shows us that we kept all the columns except the age and height columns when we created new_data.

```
glimpse(new_data)
```

```
## Rows: 59,946
## Columns: 4
## $ diet   <chr> "strictly anything", "mostly other", "an...
## $ pets   <chr> "likes dogs and likes cats", "likes dogs...
## $ sex    <chr> "m", "m", "m", "m", "m", "m", "f", "f", ...
## $ status <chr> "single", "single", "available", "single...
```

2.4.2 summarise()

The summarise() command can be used to generate descriptive statistics for a specified column. You can easily calculate column descriptive statistics using the corresponding commands for mean(), sd(), min(), max(), among others. In the example below we calculate the mean for the age column.

In the code below, mean(age, na.rm = TRUE), indicates to R that it should calculate the mean of the age column. The na.rm indicates how missing values should be handled. The na stands for not available; in R missing values are classified as Not Available or NA. The rm stands for remove. Consequently, na.rm is asking: "Should we remove missing values when calculating the mean?" The TRUE indicates that yes, missing values should be removed when calculating the mean. The result of this calculation is placed into a variable labelled age_mean, though we could have used any label we wanted instead of age_mean. We see that the mean of the age column is, with rounding, 32.3.

```
okcupid_profiles %>%
  summarise(age_mean = mean(age, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   age_mean
##      <dbl>
## 1     32.3
```

More than one calculation can occur in the same summarise() command. You can easily add the calculation for the standared deviation with the sd() command.

```
okcupid_profiles %>%
  summarise(age_mean = mean(age, na.rm = TRUE),
            age_sd = sd(age, na.rm = TRUE))
```

```
## # A tibble: 1 x 2
##   age_mean age_sd
##      <dbl>  <dbl>
## 1     32.3   9.45
```

Often this process does too much rounding. We can get more exact results by adding an as.data.frame() to the end of the commands.

```
okcupid_profiles %>%
  summarise(age_mean = mean(age, na.rm = TRUE),
            age_sd = sd(age, na.rm = TRUE)) %>%
  as.data.frame()
```

```
##   age_mean age_sd
## 1    32.34  9.453
```

2.4.3 filter()

The filter() command allows you to obtain a subset of the rows in your data. In the example below we create a new data set with just the males from the original data.

Notice the structure of the original data below in the glimpse() output. There is a column called sex that uses m and f to indicate male and female, respectively. Also notice that there are 59946 rows in the okcupid_profiles data.

```
glimpse(okcupid_profiles)
```

```
## Rows: 59,946
## Columns: 6
## $ age    <dbl> 22, 35, 38, 23, 29, 29, 32, 31, 24, 37, ...
## $ diet   <chr> "strictly anything", "mostly other", "an...
## $ height <dbl> 75, 70, 68, 71, 66, 67, 65, 65, 67, 65, ...
## $ pets   <chr> "likes dogs and likes cats", "likes dogs...
## $ sex    <chr> "m", "m", "m", "m", "m", "m", "f", "f", ...
## $ status <chr> "single", "single", "available", "single...
```

We use the filter command to select a subset of the rows based on the content of any column. In this case the sex column is used to obtain a subset of the rows; the rows with the value "m" are obtained. Notice the double equals sign is used to indicate "equal to". The reason a double equals sign is used here (instead of a single equals sign) is to distinguish it from the use of the single equals sign in the summarise command above. In the summarise command above, the single equal sign was used to indicate "assign to". That is, assign to age_mean the mean of the column age after it is calculated. A single equals sign indicates "assign to" whereas a double equals sign indicates "is equal to".

```
okcupid_males <- okcupid_profiles %>%
  filter(sex == "m")
```

We use glimpse() to inspect these all male data. Notice that only the letter m is in the sex column - indicating only males are in the data set. Also notice that there are 35829 rows in the okcupid_males data - fewer people because males are a subset of the total number of rows.

```
glimpse(okcupid_males)
```

```
## Rows: 35,829
## Columns: 6
## $ age    <dbl> 22, 35, 38, 23, 29, 29, 37, 35, 28, 24, ...
## $ diet   <chr> "strictly anything", "mostly other", "an...
## $ height <dbl> 75, 70, 68, 71, 66, 67, 65, 70, 72, 72, ...
## $ pets   <chr> "likes dogs and likes cats", "likes dogs...
## $ sex    <chr> "m", "m", "m", "m", "m", "m", "m", "m", ...
## $ status <chr> "single", "single", "available", "single...
```

The filter command can be combined with the summarise command to get the descriptive statistics for males without the hassle of creating new data. This is again done using the %>% "and then" operator.

```
okcupid_profiles %>%
  filter(sex == "m") %>%
  summarise(age_mean = mean(age, na.rm = TRUE),
            age_sd = sd(age, na.rm = TRUE))
```

```
## # A tibble: 1 x 2
##   age_mean age_sd
##      <dbl>  <dbl>
## 1     32.0   9.03
```

We see that for the 35829 females the mean age is 32.0 and the standard deviation is 9.0.

Likewise, we can obtain the descriptive statistics for females with only a slight modification, changing m to f in the filter command:

```
okcupid_profiles %>%
  filter(sex == "f") %>%
  summarise(age_mean = mean(age, na.rm = TRUE),
            age_sd = sd(age, na.rm = TRUE))
```

```
## # A tibble: 1 x 2
##   age_mean age_sd
##      <dbl>  <dbl>
## 1     32.8   10.0
```

We see that for the 24117 females the mean age is 32.8 and the standard deviation is 10.0.

2.4.4 group_by()

The process we used with the filter command would quickly become onerous if we had many subgroups for a column. Consequently, it's often better to use the group() command to calculate descriptive statistics for the levels (e.g., male/female) of a variable. By telling the computer to group_by() sex the summarise command is run separately for every level of sex (i.e., m and f).

```
okcupid_profiles %>%
  group_by(sex) %>%
  summarise(age_mean = mean(age, na.rm = TRUE),
            age_sd = sd(age, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 3
##   sex   age_mean age_sd
##   <chr>    <dbl>  <dbl>
## 1 f         32.8  10.0
## 2 m         32.0   9.03
```

Fortunately, it's possible to use more than one grouping variable with the group_by() command. In the code below we group by sex and status (i.e., dating status).

```
okcupid_profiles %>%
  group_by(sex, status) %>%
  summarise(age_mean = mean(age, na.rm = TRUE),
            age_sd = sd(age, na.rm = TRUE))
```

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

```
## # A tibble: 10 x 4
## # Groups:   sex [2]
##    sex   status         age_mean age_sd
##    <chr> <chr>             <dbl>  <dbl>
##  1 f     available          32.2   8.54
##  2 f     married            33.7   8.13
##  3 f     seeing someone     28.1   6.44
##  4 f     single             33.0  10.2
##  5 f     unknown            27.8   5.91
##  6 m     available          34.8   9.40
##  7 m     married            38.7  10.1
##  8 m     seeing someone     30.8   7.06
##  9 m     single             31.9   9.04
## 10 m     unknown            40.7   8.87
```

The resulting output provide for age the mean and standard deviation for every combination of sex and dating status. The first five rows provide output for females at every level of dating status whereas the subsequent five rows provide output for males at every level of dating status.

2.4.5 mutate()

The mutate() command can be used to calculate a new column in a data. In the example below we calculate a new column called age_centered which is the new version of the age_column where the mean of the column has been removed from every value. This is merely an example of the many different types of calculations we can perform to create a new column using mutate().

```
okcupid_profiles <- okcupid_profiles %>%
  mutate(age_centered = age - mean(age, na.rm = TRUE))
```

Notice that the glimpse() command reveals that after we use the mutate() command there is a new column called age_centered.

```
glimpse(okcupid_profiles)
```

```
## Rows: 59,946
## Columns: 7
## $ age          <dbl> 22, 35, 38, 23, 29, 29, 32, 31, 24...
## $ diet         <chr> "strictly anything", "mostly other...
## $ height       <dbl> 75, 70, 68, 71, 66, 67, 65, 65, 67...
## $ pets         <chr> "likes dogs and likes cats", "like...
## $ sex          <chr> "m", "m", "m", "m", "m", "m", "f",...
## $ status       <chr> "single", "single", "available", "...
## $ age_centered <dbl> -10.3403, 2.6597, 5.6597, -9.3403,...
```

2.5 Advanced tidyverse commands

In this advanced selection we revisit the commands from the basic tidyverse section but use more complicated code to either select or apply an action to more than one column at a time. We will indicate the columns that we want to select or apply an action to using: starts_with(), ends_with(), contains(), matches(), or where(). The first four of these are used to indicate columns based on column names. In contrast, the last command, where(), is used to

indicate the columns based on the column type (numeric, character, factor, etc.).

We will review all five commands for indicating the columns we want in the select() selection below. Following that we will, for brevity, typically use only one of the five commands when illustrating how they work with summarise() and mutate().

We begin by loading a new data.

```
library(tidyverse)
data_exp <- read_csv("data_experiment.csv")
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   sex = col_character(),
##   t1_vomit = col_double(),
##   t1_aggression = col_double(),
##   t2_vomit = col_double(),
##   t2_aggression = col_double()
## )
```

The glimpse() command reveals that this is a small data set where every row represents one rat. The sex of the rat is recorded as well as, for each of two time points, a rating of vomiting and aggression.

```
glimpse(data_exp)
```

```
## Rows: 6
## Columns: 6
## $ id            <dbl> 1, 2, 3, 4, 5, 6
## $ sex           <chr> "male", "female", "male", "female...
## $ t1_vomit      <dbl> 3, 2, 0, 3, 2, 1
## $ t1_aggression <dbl> 5, 6, 4, 7, 3, 8
## $ t2_vomit      <dbl> 2, 1, 1, 2, 1, 2
## $ t2_aggression <dbl> 6, 7, 6, 7, 5, 8
```

2.5.1 select()

2.5.1.1 select() using column name

2.5.1.1.1 starts_with()

starts_with() allows us to select columns based on how the column name begins. Here we put the columns that begin with "t1" into a new data called data_time1.

```
data_time1 <- data_exp %>%
  select(starts_with("t1"))
```

The glimpse command shows us the new data only contains the columns that begin with "t1"

```
glimpse(data_time1)
```

```
## Rows: 6
## Columns: 2
## $ t1_vomit      <dbl> 3, 2, 0, 3, 2, 1
## $ t1_aggression <dbl> 5, 6, 4, 7, 3, 8
```

2.5.1.1.2 ends_with()

ends_with() allows us to select columns based on how the column name ends. Here we put the columns that end with "aggression" into a new data set called data_aggression.

```
data_aggression <- data_exp %>%
  select(ends_with("aggression"))
```

```
glimpse(data_aggression)
```

```
## Rows: 6
## Columns: 2
## $ t1_aggression <dbl> 5, 6, 4, 7, 3, 8
## $ t2_aggression <dbl> 6, 7, 6, 7, 5, 8
```

2.5.1.1.3 contains()

contains() allows us to select columns based on the contents of the column name. Here we put the columns that have "_" in the name into a new data set called new_data.

```
new_data <- data_exp %>%
  select(contains("_"))
```

```
glimpse(new_data)
```

```
## Rows: 6
## Columns: 4
## $ t1_vomit      <dbl> 3, 2, 0, 3, 2, 1
## $ t1_aggression <dbl> 5, 6, 4, 7, 3, 8
## $ t2_vomit      <dbl> 2, 1, 1, 2, 1, 2
## $ t2_aggression <dbl> 6, 7, 6, 7, 5, 8
```

2.5.1.1.4 matches()

It's also possible to use *regex* (regular expressiona) to select columns. Regex is a powerful way to specify search/matching requirements for text - in this case the text of column names. An explanation of regex is beyond the scope of this chapter. Nonetheless the example below uses regex to select any column with an underscore in the column name followed by any character. The result is the same as the above for the contains() command. However, the matches() command is more flexible than the contains() command and can take into account substantially more complicated situations.

```
data_matched<- data_exp %>%
  select(matches("(_.)"))
```

You can see the columns selected using regex:

```
glimpse(data_matched)
```

```
## Rows: 6
## Columns: 4
## $ t1_vomit      <dbl> 3, 2, 0, 3, 2, 1
## $ t1_aggression <dbl> 5, 6, 4, 7, 3, 8
## $ t2_vomit      <dbl> 2, 1, 1, 2, 1, 2
## $ t2_aggression <dbl> 6, 7, 6, 7, 5, 8
```

You can learn about regex at RegexOne[1] and test your regex specification at Regex101[2]. Ideally though, as we discuss in the next chapter, you can use naming conventions that are sufficiently thoughtful that you don't need regex, or only rarely. The reason for this is that regex can be challenging to use. As Twitter user @ThatJenPerson noted "Regex is like tequila: use it to try to solve a problem and now you have two problems." Nonetheless, at one or two points in the future we will use regex to solve a problem (but not tequila).

2.5.1.2 select() using column type

If many cases we will want to select or perform an action on a column based on whether the column is a numeric, character, or factor column (indicated in glimpse output as dbl, chr, and fct, respectively). We will learn more about factors later in this chapter. Each of these column types can be selected by using is.numeric, is.character, or is.factor, respectively, in combination with the where() command.

We can select numeric columns using where() and is.numeric:

```
data_numeric_columns <- data_exp %>%
  select(where(is.numeric))
```

You can see the new data contains only the numeric columns:

```
glimpse(data_numeric_columns)
```

```
## Rows: 6
## Columns: 5
## $ id            <dbl> 1, 2, 3, 4, 5, 6
## $ t1_vomit      <dbl> 3, 2, 0, 3, 2, 1
## $ t1_aggression <dbl> 5, 6, 4, 7, 3, 8
## $ t2_vomit      <dbl> 2, 1, 1, 2, 1, 2
## $ t2_aggression <dbl> 6, 7, 6, 7, 5, 8
```

We can select numeric columns using where() and is.character:

```
data_character_columns <- data_exp %>%
  select(where(is.character))
```

You can see the new data contains only the character columns:

[1] https://regexone.com
[2] https://regex101.com

```
glimpse(data_character_columns)
```

```
## Rows: 6
## Columns: 1
## $ sex <chr> "male", "female", "male", "female", "male",...
```

If a future chapter you will see how we can select factors using where(is.factor).

2.5.2 summarise()

The summarise() command can summarise multiple columns when combined with starts_with(), ends_with(), contains(), matches(), and where(). However, to use these powerful tools for indicating columns with the summarise command we need the help of the across() command (i.e., across multiple columns).

If we want to obtain the mean of all the columns that start with "t1" we use the commands below. The across command requires that we indicate the columns we want via the .cols argument and the command/function we want to run on those columns via the .fns argument. In the example below, we also add na.rm = TRUE at the end; this is something we send to the mean command to let it know how we want to handle missing data. We add as.data.frame() to get a larger number of decimals.

```
data_exp %>%
  summarise(across(.cols = starts_with("t1"),
                   .fns = mean,
                   na.rm = TRUE)) %>%
  as.data.frame()
```

```
##   t1_vomit t1_aggression
## 1    1.833           5.5
```

If you want to get more sophisticated, you can also add this .names argument below which tells R to call label each output mean by the column name followed by "_mean".

```
data_exp %>%
  summarise(across(.cols = starts_with("t1"),
                   .fns = mean,
                   na.rm = TRUE,
                   .names = "{col}_mean"))  %>%
  as.data.frame()
```

```
##   t1_vomit_mean t1_aggression_mean
## 1         1.833                5.5
```

Often you want to calculate more than one statistic for each column. For example, you might want the mean, standard deviation, min, and max. These statistics can be calculated via the mean, sd, min, and max commands, respectively. However, you need to create a list with the statistics you desire.

Below we create a list of the descriptive statistics we desire called desired_statistics, but you can use any name you want. This list only needs to be specified once, but we will repeat it in the examples below for clarity.

```
desired_descriptives <- list(
  mean = ~mean(.x, na.rm = TRUE),
  sd = ~sd(.x, na.rm = TRUE)
)
```

Once you have created the list of descriptive statistics you want you can run the command below to obtain those statistics. However, as you will see the output is too wide to be helpful.

```
data_exp %>%
  summarise(across(.cols = starts_with("t1"),
                   .fns = desired_descriptives)) %>%
  as.data.frame()
```

```
##   t1_vomit_mean t1_vomit_sd t1_aggression_mean
## 1         1.833       1.169                5.5
##   t1_aggression_sd
## 1            1.871
```

Consequently, we add the t() command (i.e., transpose command) to the end of the summarise request to get a more readable list of statistics:

```
desired_descriptives <- list(
  mean = ~mean(.x, na.rm = TRUE),
  sd = ~sd(.x, na.rm = TRUE)
)

data_exp %>%
  summarise(across(.cols = starts_with("t1"),
                   .fns = desired_descriptives)) %>%
  as.data.frame() %>%
  t()
```

```
##                      [,1]
## t1_vomit_mean      1.833
## t1_vomit_sd        1.169
## t1_aggression_mean 5.500
## t1_aggression_sd   1.871
```

Note that in the across command above we could also have used: ends_with(), contains(), matches(), or where().

2.5.3 mutate()

The mutate() command can also be applied to multiple columns using the across() command. However, sometimes we need to embed our calculation in a custom function. Below is a custom function called make_centered. This custom function takes the values in a column and subtracts the column mean from each value in the column. This is the same task we did previous using the mutate() command in the basic tidyverse section.

```
make_centered <- function(values) {
  values_out <- values - mean(values, na.rm = TRUE)
  return(values_out)
}
```

The glimpse() command shows us all the column names. Also notice the values in the agresssion columns are integers.

```
glimpse(data_exp)
```

```
## Rows: 6
## Columns: 6
## $ id            <dbl> 1, 2, 3, 4, 5, 6
## $ sex           <chr> "male", "female", "male", "female...
## $ t1_vomit      <dbl> 3, 2, 0, 3, 2, 1
## $ t1_aggression <dbl> 5, 6, 4, 7, 3, 8
## $ t2_vomit      <dbl> 2, 1, 1, 2, 1, 2
## $ t2_aggression <dbl> 6, 7, 6, 7, 5, 8
```

We combine the mutate() command, with the across() command, and our custom make_centered() command below. The command "centers" or subtracts the mean from any column that ends with "aggression".

```
data_exp <- data_exp %>%
```

```
  mutate(across(.cols = ends_with("aggression"),
                .fns = make_centered))
```

You can see via the glimpse() output that the contents of all the columns that end with "aggression" have changed. Every value in one these columns has had the column mean subtracted from it.

```
glimpse(data_exp)
```

```
## Rows: 6
## Columns: 6
## $ id            <dbl> 1, 2, 3, 4, 5, 6
## $ sex           <chr> "male", "female", "male", "female...
## $ t1_vomit      <dbl> 3, 2, 0, 3, 2, 1
## $ t1_aggression <dbl> -0.5, 0.5, -1.5, 1.5, -2.5, 2.5
## $ t2_vomit      <dbl> 2, 1, 1, 2, 1, 2
## $ t2_aggression <dbl> -0.5, 0.5, -0.5, 0.5, -1.5, 1.5
```

Note that in the across() command above, we could also have used: starts_with(), contains(), matches(), or where().

2.5.3.1 mutate() within rows

Researchers often want to average within rows and across columns to create a new column. That is, for each participant (i.e., rat in the current data) we might want to calculate a vomit score that is the average of the two time points (that we will call vomit_avg).

To average within rows (and across columns) we use the rowwise() command to inform R of our intent. However, after we do the necessary calculations we have to shut off the rowwise() calculation state by using the ungroup() command. As well, when we are averaging within rows we have to use c_across() instead of across(). The commands below create a new column called vomit_avg which is the average of the vomit ratings across both times. As before, we also include na.rm = TRUE so the computer drops missing values (if present) when calculating the mean.

```
data_exp <- data_exp %>%
  rowwise() %>%
  mutate(vomit_avg = mean( c_across(cols = ends_with("vomit")),
                           na.rm = TRUE)) %>%
  ungroup()
```

You can see the new column we created with the glimpse() command:

```
glimpse(data_exp)
```

```
## Rows: 6
## Columns: 7
## $ id            <dbl> 1, 2, 3, 4, 5, 6
## $ sex           <chr> "male", "female", "male", "female...
## $ t1_vomit      <dbl> 3, 2, 0, 3, 2, 1
## $ t1_aggression <dbl> -0.5, 0.5, -1.5, 1.5, -2.5, 2.5
## $ t2_vomit      <dbl> 2, 1, 1, 2, 1, 2
## $ t2_aggression <dbl> -0.5, 0.5, -0.5, 0.5, -1.5, 1.5
## $ vomit_avg     <dbl> 2.5, 1.5, 0.5, 2.5, 1.5, 1.5
```

The print() command could make it easier to see that the new column is the average of the other two, but if we use the print() command below it wouldn't work. Why? There are too many columns in the data set, so only the first few columns are shown.

```
print(data_exp)
```

To see that the new column, vomit_avg, is the average of the other vomit columns we use the select command before print(). This prints only the relvant columns. When this is done, it's easy to see how the values in the vomit_avg column are the mean of the other two columns.

```
data_exp %>%
  select(contains("vomit")) %>%
  print()
```

```
## # A tibble: 6 x 3
##   t1_vomit t2_vomit vomit_avg
##      <dbl>    <dbl>     <dbl>
## 1        3        2       2.5
## 2        2        1       1.5
## 3        0        1       0.5
## 4        3        2       2.5
## 5        2        1       1.5
## 6        1        2       1.5
```

2.5.3.2 mutate() within rows using column names

Sometimes it can be difficult to use one of the advanced select commands to obtain the columns you need to average across. The advanced commands

like ends_with() and starts_with() can sometimes include columns you don't want. The command below is equivalent the one above, however, we explicitly name the variables we want to average across.

```
data_exp <- data_exp %>%
  rowwise() %>%
  mutate(vomit_avg = mean( c_across(cols = c(t1_vomit, t2_vomit)),
                           na.rm = TRUE)) %>%
  ungroup()
```

You can use print() to confirm we get the same result:

```
data_exp %>%
  select(contains("vomit")) %>%
  print()
```

```
## # A tibble: 6 x 3
##   t1_vomit t2_vomit vomit_avg
##      <dbl>    <dbl>     <dbl>
## 1        3        2       2.5
## 2        2        1       1.5
## 3        0        1       0.5
## 4        3        2       2.5
## 5        2        1       1.5
## 6        1        2       1.5
```

2.5.3.3 mutate() for factors

It is critical that you indicate to R that categorical variables are in fact categorical variables. In R, categorical variables are referred to as factors. For humans, a factor like sex has three possible levels: female, male, intersex.

An inspection of the glimpse() command output reveals that the sex column has the type character - as indicated by "chr". Also notice, as you inspect this output, that we use words (e.g., female) to indicate the sex in the column rather than a number to represent a female participant (e.g., 2). This is the preferred, but less common, approach to entering data.

```
glimpse(data_exp)
```

```
## Rows: 6
## Columns: 7
## $ id            <dbl> 1, 2, 3, 4, 5, 6
```

```
## $ sex           <chr> "male", "female", "male", "female...
## $ t1_vomit      <dbl> 3, 2, 0, 3, 2, 1
## $ t1_aggression <dbl> -0.5, 0.5, -1.5, 1.5, -2.5, 2.5
## $ t2_vomit      <dbl> 2, 1, 1, 2, 1, 2
## $ t2_aggression <dbl> -0.5, 0.5, -0.5, 0.5, -1.5, 1.5
## $ vomit_avg     <dbl> 2.5, 1.5, 0.5, 2.5, 1.5, 1.5
```

We need to convert the sex column to a factor in order for R to handle it appropriately in analyses. Failure to indicate the column is a factor could result in R conducting all the analyses and presenting incorrect results. Consequently, it is critical that we covert the column to a factor. Fortunately, that is easily done using the as_factor() command (there is also an as.factor command if as_factor won't work for some reason).

We convert the sex column to a factor with this code:

```
data_exp <- data_exp %>%
  mutate(sex = as_factor(sex))
```

You can confirm this worked with the glimpse() command:

```
glimpse(data_exp)
```

```
## Rows: 6
## Columns: 7
## $ id            <dbl> 1, 2, 3, 4, 5, 6
## $ sex           <fct> male, female, male, female, male,...
## $ t1_vomit      <dbl> 3, 2, 0, 3, 2, 1
## $ t1_aggression <dbl> -0.5, 0.5, -1.5, 1.5, -2.5, 2.5
## $ t2_vomit      <dbl> 2, 1, 1, 2, 1, 2
## $ t2_aggression <dbl> -0.5, 0.5, -0.5, 0.5, -1.5, 1.5
## $ vomit_avg     <dbl> 2.5, 1.5, 0.5, 2.5, 1.5, 1.5
```

If you entered your data using words for each level of sex (e.g., male, female) you're done at this point. However, if you used numbers to represent each level of sex in your data, there is one more step. Imagine your data was entered in a poorly advised manner, such that 1 was used to indicate male, 2 was used to indicate female, and 3 was used to indicate intersex. If this was the case, you need to indicate to R what each of those values represent. We do that with the code below.

```
data_exp <- data_exp %>%
  mutate(sex = fct_recode(sex,
                          male = "1",
```

```
                              female = "2",
                              intersex = "3"))
```

2.6 Using help

In order to become efficient at analyzing data using R, you will need to become adapt at reading and understanding the help files associated with each command. After you have activated a package using the library command (e.g., library(tidyverse)) you can access the help page for every command in that package. To access the help page simply type a question mark followed by the command you want to know how to use (no space between them). The code below brings up the help page for the select() command. Notice that we put the library() command first - just a reminder that this needs to be done prior to using help for that package. Try the commands below in the Console:

```
library(tidyverse)
?select
```

Examine the page that appears on the Help tab in the panel in the lower right of your screen. Read through the help file comparing what you read there to what we have learned about the select command. Notice how the help file tells you about the argument that you sent into the select() command, and also what the select() command returns when it receives those commands. Pay particular attention to the examples near the bottom of the help page. At the very bottom of the help page you will see [Package dplyr version 1.0.0 Index]. This tells you the select() command is from the dplyr package (part of the tidyverse). Notice that the word "Index" is underlined. Click on the word Index. You will be presented with list of other commands in the dplyr package.

As you become more experienced with R help pages, this is how you will learn to use new commands. Examine the help pages for the commands below by typing a question mark into the Console followed by the command name. Note that for filter and starts_with you will be presented with a menu instead of help page. This typically occurs because the command is in more than one package. If this does occur, read through the options you are presented with to try and figure out which one you wanted. Typically, you want the first option. If you're not sure, try one. IF it's not what you want, use the back arrow in the Help panel to go back and pick another one.

- mutate
- filter
- starts_with

2.7 Base R vs tidyverse

All of the commands used to this point in the chapter have been the tidyverse approach to using R. That is the approach we will normally use. However, it's important to note that there is another way of using R, called base R.

Sometimes students have problems with their code when they mix and match these approaches using a bit of both. We will be using the tidyverse approach to using R but on the internet you will often see sample code that uses the older base R approach. A bit of background knowledge is helpful for understanding why we do things one way (e.g., read_csv with the tidyverse) instead of another (e.g., read.csv with base R).

2.7.1 Tibbles vs. data frames

When you load data into R, it is typically represented in one of two formats inside the computer - depending on the command you used. The original format for representing a data set in R is the data frame. You will see this term used frequently when you read about R. When you load data using read.csv(), your data is loaded into a data frame in the computer. That is, your data is represented in the memory of the computer in a particular format and structure called a data frame. This contrasts with the newer tidyverse approach to representing data in the computer called a tibble - which is just a newer more advanced version of the data frame.

2.7.2 read.csv() and data frames

When you read data into R using the command read.csv() (with a period) you load the data into a data frame (base R).

```
my_dataframe <- read.csv(file = "data_okcupid.csv")
```

Notice that when you print a data frame it does not show you the number of rows or columns above the data like our example did with the okcupid_profiles data. Likewise, it does not show you the type of data in each column (e.g.,

dbl, fct, chr). It also presents all of your data rather than just the first few rows (as the tibble does). As a result, in the output below, we show only the first 10 rows of the output - because all the rows are printed in your Console with a data frame (too much to show here).

```
print(my_dataframe)
```

```
##    age              diet height                      pets
## 1   22 strictly anything     75 likes dogs and likes cats
## 2   35      mostly other     70 likes dogs and likes cats
## 3   38          anything     68                   has cats
## 4   23        vegetarian     71                likes cats
## 5   29              <NA>     66 likes dogs and likes cats
## 6   29   mostly anything     67                likes cats
## 7   32 strictly anything     65 likes dogs and likes cats
## 8   31   mostly anything     65 likes dogs and likes cats
## 9   24 strictly anything     67 likes dogs and likes cats
## 10  37   mostly anything     65 likes dogs and likes cats
##    sex    status
## 1    m    single
## 2    m    single
## 3    m available
## 4    m    single
## 5    m    single
## 6    m    single
## 7    f    single
## 8    f    single
## 9    f    single
## 10   m    single
```

2.7.3 read_csv() and tibbles

When you read data into R using the command read_csv() (with an underscore) you load the data into a tibble (tidyverse).

```
my_tibble <- read_csv(file = "data_okcupid.csv")
```

```
## Parsed with column specification:
## cols(
##   age = col_double(),
##   diet = col_character(),
##   height = col_double(),
##   pets = col_character(),
```

```
##   sex = col_character(),
##   status = col_character()
## )
```

The tibble is modern version of the data frame. Notice that when you print a tibble it DOES show you the number of rows and columns. As well, it shows you the type of data in each column. Importantly, the tibble only provides the first few rows of output so it doesn't fill your screen.

```
print(my_tibble)
```

```
## # A tibble: 59,946 x 6
##      age diet         height pets              sex   status
##    <dbl> <chr>         <dbl> <chr>             <chr> <chr>
##  1    22 strictly an~     75 likes dogs and l~ m     single
##  2    35 mostly other     70 likes dogs and l~ m     single
##  3    38 anything         68 has cats          m     availa~
##  4    23 vegetarian       71 likes cats        m     single
##  5    29 <NA>             66 likes dogs and l~ m     single
##  6    29 mostly anyt~     67 likes cats        m     single
##  7    32 strictly an~     65 likes dogs and l~ f     single
##  8    31 mostly anyt~     65 likes dogs and l~ f     single
##  9    24 strictly an~     67 likes dogs and l~ f     single
## 10    37 mostly anyt~     65 likes dogs and l~ m     single
## # ... with 59,936 more rows
```

In short, you should always use tibbles (i.e., use read_csv() not read.csv()). The differences between data frames and tibbles run deeper than the superficial output provided here. On some rare occasions an old package or command may not work with a tibble so you need to make it a data frame. You can do so with the commands below. We will flag these rare occurances to you when they occur.

```
# Create a data frame from a tibble
new_dataframe <- as.data.frame(my_tibble)
```

3

Populations

3.1 Notation

In this chapter we will use summation notation. If you are not familiar with summation notation, we present a brief overview here.

Consider a scenario where we have the IQ data for three participants We use the N symbol to represent the number of participants. Because we have three participants N = 3. The data for these participants is illustrated in Figure **??**.

Notice how each person in the data set can be represented by the variable X: the first person by X_1, the second by X_2, and the third by X_3. Often we refer to individuals in a data set by using the variable X accompanied by a subscript (e.g., 1, 2, 3, etc.).

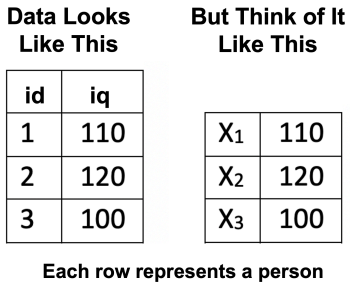


FIGURE 3.1: Data for understanding summation notation

Referring to participants using the variable X and subscript is valuable because it can be used in conjunction with the sigma (i.e., Σ) symbol for summation. Consider the example below in which we use the summation notation to indicate that we want to add all the X values (representing IQ) for the participants. We use a lower case i to represent all possible subscript values. The notation, $i = 1$, below the Σ symbol indicates that we should start with participant 1. The notation, N, above the Σ symbol indicates that we should iterate i up to the value indicated by N; in this case 3, because there are three participants.

$$\begin{aligned}\sum_{i=1}^{N} X_i &= X_1 + X_2 + X_3 \\ &= 110 + 120 + 100 \\ &= 330\end{aligned}$$

Sometimes, to simplify the notation, the numbers above and below the Σ symbol are omitted. Likewise, the i subscript is omitted. There is a general understanding that when these components of the notation are omitted the version of the notation above is implied.

$$\begin{aligned}\sum X &= X_1 + X_2 + X_3 \\ &= 110 + 120 + 100 \\ &= 330\end{aligned}$$

Calculating a mean. The full version of the notation can be used to indicate how an average/mean is calculated.

$$\begin{aligned}\bar{X} &= \frac{\sum_{i=1}^{N} X_i}{N} \\ &= \frac{X_1 + X_2 + X_3}{3} \\ &= \frac{110 + 120 + 100}{3} \\ &= \frac{330}{3} \\ &= 110\end{aligned}$$

Likewise, the concise version of the notation can be used to indicate how an average/mean is calculated.

$$\begin{aligned}\bar{X} &= \frac{\sum X}{N} \\ &= \frac{X_1 + X_2 + X_3}{3} \\ &= \frac{110 + 120 + 100}{3} \\ &= \frac{330}{3} \\ &= 110\end{aligned}$$

Calculating squared differences.A common task in statistics is to calculate 1) the squared difference between each person and the mean, and 2) add up

those squared differences. This calculation is easily expressed with the full version of the notation.

$$\begin{aligned}\sum_{i=1}^{N}(X_i - \bar{X})^2 &= (X_1 - \bar{X})^2 + (X_2 - \bar{X})^2 + (X_3 - \bar{X})^2 \\ &= (110 - 110)^2 + (120 - 110)^2 + (100 - 110)^2 \\ &= (0)^2 + (10)^2(-10)^2 \\ &= 0 + 100 + 100 \\ &= 200\end{aligned}$$

Likewise, the sum of the squared differences from the mean can be expressed using the concise version of the notation.

$$\begin{aligned}\sum(X - \bar{X})^2 &= (X_1 - \bar{X})^2 + (X_2 - \bar{X})^2 + (X_3 - \bar{X})^2 \\ &= (110 - 110)^2 + (120 - 110)^2 + (100 - 110)^2 \\ &= (0)^2 + (10)^2(-10)^2 \\ &= 0 + 100 + 100 \\ &= 200\end{aligned}$$

3.2 Population vs samples

As we move closer to conducting our own research it is critical to make a distinction between populations and samples. A population is the complete set of people/animals about which we want to make conclusions. A sample is a randomly selected subset of the population. In most scenarios it is impractical to work with an entire population and, for practical reasons, we study a subset of the population called a sample.

Researchers, and consumers of research, typically have little interest in making conclusions at the sample level. In general, we care about conclusions that generalize to the population but not conclusions that only apply to specific individuals in the sample. Consider the case of COVID-19. Imagine a research team creates a vaccine that they hope generates immunity to COVID-19. We care very little if the immunity only works for the specific individuals in the study. However, we care a great deal if the immunity works, or is likely to work, for all Canadians or all humans. We study samples but typically wish to make conclusions that apply to the population. Thus, even if you are an experimental researcher it's critical that you think in terms of populations and not samples. Indeed, statistical tests (such as the *t*-test) are a means of

helping researchers use sample data to make conclusions at the population level.

In this chapter, our focus is on describing populations. When we calculate a number that summarises an attribute of all of the people/animals in the population we refer to it as a **parameter**.

3.3 A small population

In this section we review how to calculate three commonly used population parameters (mean, variance, and standard deviation). Populations are typically quite large but for simplicity we focus on a population composed of the weights of just three chocolate chip cookies. We refer to the three cookies as X_1, X_2, and X_3. The cookies have the weights of 8, 10, and 12 grams, respectively.

3.3.1 Mean (μ)

It can be helpful to create a model that describes our data. Of course, the model won't describe every participant perfectly and each participant will differ to some extent from the model.

Model: To create a model we first need data, which in this example will be the weight of three different chocolate chip cookies. As mentioned previously, the weights of the three cookies are designated by X_1, X_2, and X_3. A simple model for our cookie weight data is the mean. At the population level the mean is represented by the symbol μ see Formula (**??**) below. At the sample level a different notation is used.

$$\mu = \frac{\sum X}{N} \tag{3.1}$$

Using that equation with values:

$$\begin{aligned}
\mu &= \frac{\sum X}{N} \\
&= \frac{X_1 + X_2 + X_3}{3} \\
&= \frac{8 + 10 + 12}{3} \\
&= \frac{30}{3} \\
&= 10
\end{aligned}$$

We can think of the "mean cookie" as our model for our cookie weight data, see Figure **??**. The "mean cookie" is represented by μ in equations.

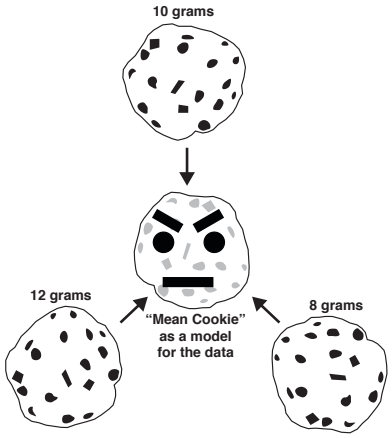


FIGURE 3.2: Variance as a fit index for the mean

Error: As mentioned previously, each participant (i.e., cookie) differs to some extent from our model ("mean cookie"). In general this can be conceptualized as:

$$X_i = model + error_i$$

More specifically, the difference between the weight of any individual cookie (X_i) and the model (μ) is indicated by $error_i$ as shown below.

$$X_i = \mu + error_i$$

The model, above is just a concise way of describing the following:

$$X_1 = \mu + error_1$$
$$X_2 = \mu + error_2$$
$$X_3 = \mu + error_3$$

That is the weights of the three cookies ($X_1 = 8$, $X_2 = 10$, and $X_3 = 12$) can be conceptualized as:

$$X_1 = 10 + (-2)$$
$$X_2 = 10 + 0$$
$$X_3 = 10 + 2$$

The mean/average of the population, $\mu = 10$, is a parameter that serves as a model for the cookie weight data. However, it's helpful to have an index, known as variance, that indicates the extent to which the data do not correspond to the model from the model.

3.3.2 Variance (σ^2)

Variance is a simple way of calculating a single number to represent how data differ from a model. It is represented, at the population level, by the symbol σ^2; a different notation is used at the sample level.

Previously, how we expressed the difference/deviation of cookie weights (data) from the model (i.e., mean) with an error term in the equation $X_i = \mu + error_i$, see see Figure **??**. The model for all the cookies is $\mu = 10$. If we consider a single cookie weight of 8 grams (a data point represented by X_1), the difference between the cookie from the model is -2 (i.e., error):

$$X_1 = 10 + (-2)$$

We want a number that indicates the quality of the cookie model. Specifically, we want a single number that indexes overall how the data (i.e., cookie weights) differ from the model (i.e., the mean cookie). We refer to that index as variance (σ^2).

Calculating Squared Errors. To calculate variance (σ^2), we use the errors for the cookies – how the cookies differ from the mean/model. The first step is to square the errors/differnces. Those squared numbers are referred to as the "squared differences" or "squared errors". The calculation of the squared error for each cookie weight is shown below. The squared errors (or squared differences) are 4, 0, and 4.

Cookie Weight	Model	Squared Difference
$X_1 = 8$	$\mu = 10$	$(X_1 - \mu)^2 = (8 - 10)^2 = 4$
$X_2 = 10$	$\mu = 10$	$(X_2 - \mu)^2 = (10 - 10)^2 = 0$
$X_3 = 12$	$\mu = 10$	$(X_3 - \mu)^2 = (12 - 10)^2 = 4$

Averaging Squared Errors. To obtain variance we calculate the average of the squared errors. At the population level the variance is represented by the symbol σ^2 see Formula (**??**) below. In this formula, N refers to the number of people in the population. At the sample level a different notation is used.

$$\sigma^2 = \frac{\sum (X - \mu)^2}{N} \tag{3.2}$$

Using that equation with values:

$$\begin{aligned}
\sigma^2 &= \frac{\sum (X - \mu)^2}{N} \\
&= \frac{(X_1 - \mu)^2 + (X_2 - \mu)^2 + (X_3 - \mu)^2}{N} \\
&= \frac{(8 - 10)^2 + (10 - 10)^2 + (12 - 10)^2}{3} \\
&= \frac{(-2)^2 + (0)^2(2)^2}{3} \\
&= \frac{4 + 0 + 4}{3} \\
&= \frac{8}{3} \\
&= 2.67 grams^2
\end{aligned}$$

The resulting variance is 2.67 grams^2. The cookie weights were measured in grams. The unit for variance, however, is grams^2 because we squared the errors as part of the calculation. Recall the formula for calculating an average (shown below) and compare it to the variance calculation (above). Notice that variance is just an average – an average of squared errors. Correspondingly, in some areas of statistics they don't use the term variance, they use a synonym - **mean squared error**.

$$\bar{X} = \frac{\sum X}{N}$$

It probably strikes you as an odd choice to square the difference between each data point and the model. Why not just use the difference (e.g., $(8 - 10) = -2$) when calculating variance? Why not use the absolute difference (e.g., $|8-10| = 2$) when calculating variance? The answer is somewhat complex, but it relates to the more general situation in statistics of trying to find models that best fit the data (which occurs by minimizing errors). When we use squared errors it is easier to apply calculus, via derivatives, to calculate a model that minimizes the errors (i.e., obtains the best fit). Long story short, for complex mathematical reasons, we use squared errors when calculating the fit (or lack of fit) of a model.

Interpretation. A variance of zero indicates that the model fits the data perfectly. In the cookie case, if the variance was zero, that would indicate that all the cookies had the same weight as the model, exactly 10 grams. To the extent that the variance is larger than zero it implies the data points (i.e., cookie weights) differ from the model (i.e., the mean cookie). By implication, a larger variance indicates larger differences among the observations (e.g., cookie weights). That is, when the variance is small, cookie weights tend to be similar to the model – and each other. In contrast, when the variance is large, cookie weights tend to be different from the model – and each other.

3.3.3 Standard Deviation (σ)

An alternative index for how data differ from the mean/model is the standard deviation. To understand standard deviation you have to understand varince. Variance is a single number that indexes how data differ from a model. The interpretation of variance is straight forward. It is the average of the squared differences between the data and the model.

Standard deviation is represented by the symbol σ and can be calculated as the square root of variance as in Formula (**??**) below.

$$\sigma = \sqrt{\frac{\sum (X - \mu)^2}{N}} \tag{3.3}$$

Using that equation with values:

$$\begin{aligned} \sigma &= \sqrt{\frac{\sum (X - \mu)^2}{N}} \\ &= \sqrt{\sigma^2} \\ &= \sqrt{2.67} \\ &= 1.63 grams \end{aligned}$$

One reason that people like standard deviation is presents the difference between the data and the model in the original units (e.g., grams). This is in contrast to variance which presents the difference betweeen the data and the model in squared units (e.g., 2.67 grams^2).

Interpretation. Unfortunately, although variance has a straight forward interpretation, standard deviation does not. Sometimes standard deviation is, incorrectly, described as how much data points differ on average from the mean. A quick calculation of the average difference reveals a number (1.33) that does not correspond to the standard deviation (1.63):

$$\begin{aligned}\overline{diff} &= \frac{\sum |X - \mu|}{N} \\ &= \frac{|8-10| + |10-10| + |12-10|}{3} \\ &= \frac{2+0+2}{3} \\ &= \frac{4}{3} \\ &= 1.33\end{aligned}$$

As illustrated above, standard deviation is not equal to the average of the deviations from the mean. Because standard deviation is not an average, it's much harder to describe how to interpret it. In our view, the best way to think of standard deviation is simply as the square root of variance; because variance has a straight forward interpretation.

Therefore, we encourage you to think primarily in terms of variance rather than standard deviation due to the fact the interpretation of variance is more straightforward. Additionally, variance is foundational in the language used to describe regression and analysis of variance. That said, standard deviation is used in the calculation of some standardized effect sizes - so it is important to know and understand both indices.

Overall, the rules for interpreting standard deviation are similar to those for variance; but the standard deviation values are smaller than variance values. In the cookie case, if the standard deviation was zero, that would indicate that all the cookies had the same weight as the model, exactly 10 grams. To the extent that the standard deviation is larger than zero it implies the data points (i.e., cookie weights) differ from the model (i.e., the mean cookie). By implication, a larger standard deviation indicates larger differences among the observations (e.g., cookie weights). That is, when the standard deviation is small, cookie weights tend to be similar to the model – and each other. In contrast, when the standard deviation is large, cookie weights tend to be different from the model – and each other.

3.4 Visualizing populations

Populations are typically quite large in nature and it's often imposssible to pratically list all of the members of the population. Consequently, it helps to have ways to visual the entire population. In Figure **??** we present three ways of visualizing a population. In all three graphs (A, B, C) in this figure

the x-axis represents heights in centimeter and the y-axis is used to indicate which values on the x-axis are more common. In Figure **??**A we use a large number of X's to indicate the members of the population. Because X's are also used in formulas to represent individual participants a strength of this graph is that it reminds you that it is a graph reflecting a large number of individuals. In Figure **??**B we present a standard histogram that illustrate the distribution of heights. In Figure **??**C we present a density curve that illustrate the distribution of heights. All three approaches are useful for illustrating that most people have heights around 170 cm.

3.5 Comparisons: Same σ

In this section we review a method of comparing two population means when the populations have the same standard deviation. To facilitate comparing two populations we use the heights of males and females, measured in centimeters, as an example. On average males are taller than females. There is, however, variability in the heights of both males and females. The variability in heights is the same for males and females even though the means differ. We illustrate these differences in a series of figures below. Each figure contains three scenarios (labeled A, B, C) in which we manipulate the mean height and variability of the populations. In each scenario the standard deviation of heights is the same for the male and female populations.

3.5.1 Standardized units

3.5.1.1 Individual scores

Often when we compare two means we use the original metric. In the case of the male and female heights that metric is centimeters. The original units are a useful way to convey information about the difference between two populations.

In addition to the original unit it is also possible, and sometimes desirable, to use the standardized mean difference. The word *standardized* is used to indicate that the comparison is relative to the standard deviation.

Imagine a population of male heights ($\mu = 170$, $\sigma = 10$) from which we have obtained a single individual, Ian, whose height is 185 cm.

$$X_{Ian} = 185cm$$

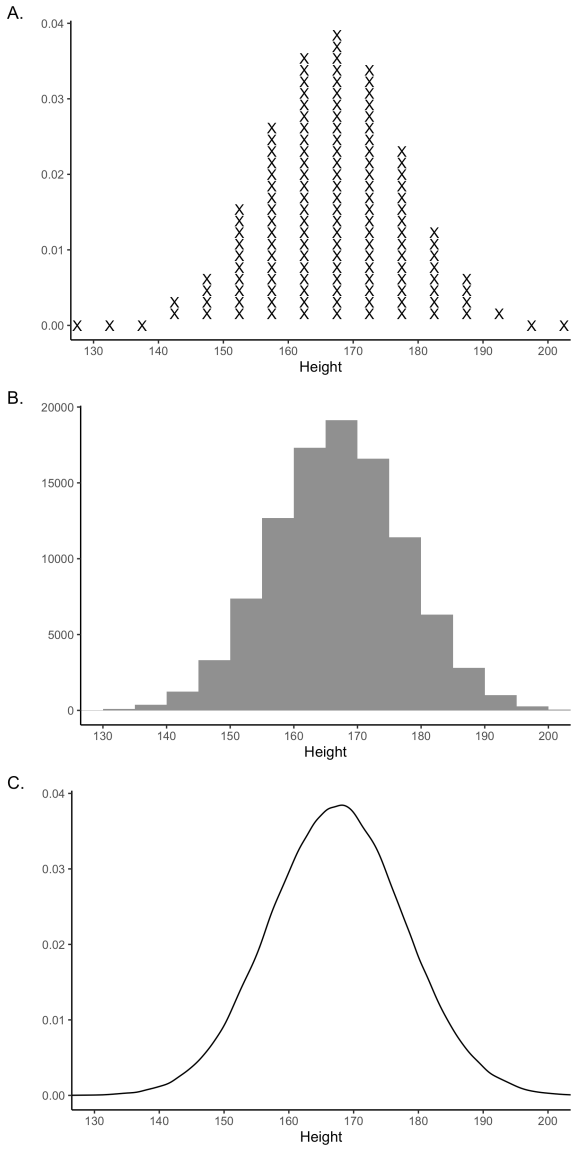


FIGURE 3.3: Three ways of visualizing a population distribution

The original units are useful for describing Ian's height but it doesn't tell us about his height relative to the other people in the population. We have to know the mean and standard deviation of the population to know if Ian is shorter or taller than the average height - and by how much. We use a z-score calculation for this purpose:

$$\begin{aligned} z_{Ian} &= \frac{X_{Ian} - \mu_{males}}{\sigma_{males}} \\ &= \frac{185 - 170}{10} \\ &= \frac{15}{10} \\ &= 1.50 \end{aligned}$$

The above calculation is a ratio. Ratios are used to compare two numbers. The numerator (number on the top) is compared to the denominator (number on bottom) through division. The resulting number tells you how much larger the numerator is than the denominator.

In this case, the numerator is the extent to which Ian is taller than the mean height for males ($X_{Ian} - \mu_{males}$). This numerator is compared to the denominator – which is the standard deviation for males (σ_{males}). The resulting number is 1.50 which indicates the numerator is 1.50 times larger than the denominator. In other words, Ian is 1.50 standard deviations taller than the average male. This is a standardized score for Ian's height - it expresses the difference between his height and the mean height in standard deviation units.

3.5.1.2 Independent Group: Population Means

The same approach to generating standardized scores can be applied to population means. Consider a situation where we have population of male heights ($\mu = 170$, $\sigma = 10$) and a population of female heights ($\mu = 165$, $\sigma = 10$). Notice that both populations have the same standard deviation.

We can calculate a standardized value to compare these heights. This standardized value is called the standardized mean difference (SMD). Alternatively, it is also known as Cohen's d which is represented at the population level with the symbol δ. Calculation of the standardized mean difference is based on the premise that both populations have the same standard deviation.

In this calculation the numerator represents the difference between the two population means. The denominator represents the population standard deviation - which is the same for both populations see Formula (**??**) below.

$$\delta = \frac{\mu_1 - \mu_2}{\sigma} \tag{3.4}$$

Using that equation with values:

$$\begin{aligned}\delta &= \frac{\mu_{males} - \mu_{females}}{\sigma} \\ &= \frac{170 - 165}{10} \\ &= \frac{5}{10} \\ &= 0.50\end{aligned}$$

The resulting division of this ratio reveals that the numerator is 0.50 times as large (i.e., half as large) as the denominator. That is, the difference between the populations is half as large as the standard deviation. Therefore, the population mean for males is 0.50 standard deviations larger than the population mean for female; $\delta = 0.50$:

3.5.2 Cohen's d units

Because the unit for the standardized mean difference is the standard deviation it can be easy to interpret if you are unfamiliar with the original units. It is usually a good idea to report the difference between populations in both the original units (cm) and standardized units (δ). Figure **??** illustrates three different population difference scenarios (A through C). The population standard deviation is held constant across the three scenarios. You can see that as the difference between the population means increases in raw units - it does the same in δ (i.e., Cohen's d) units. In raw units (i.e., cm), the difference between the population means for scenarios A through C are 5 cm, 10 cm, and 20 cm, respectively. In standardized units (i.e., standard deviations), the difference the between the population means for scenarios A through C are 0.50 standard deviations, 1.0 standard deviations, and 2.0 standard deviations, respectively. In other words, the population-level Cohen's d-values are 0.50, 1.0, and 2.0 for scenarios A through C.

3.5.3 Cohen's d advantages

The standardized mean difference takes into account the variability of heights around each population mean. This means that the same difference between two population means can produce different standardized mean difference values if the population standard deviation varies. In the scenarios depicted in Figure **??** the population standard deviation becomes increasing small - resulting in larger standardized mean difference values (i.e., δ). This larger δ value corresponds to progressively less overlap between the two populations.

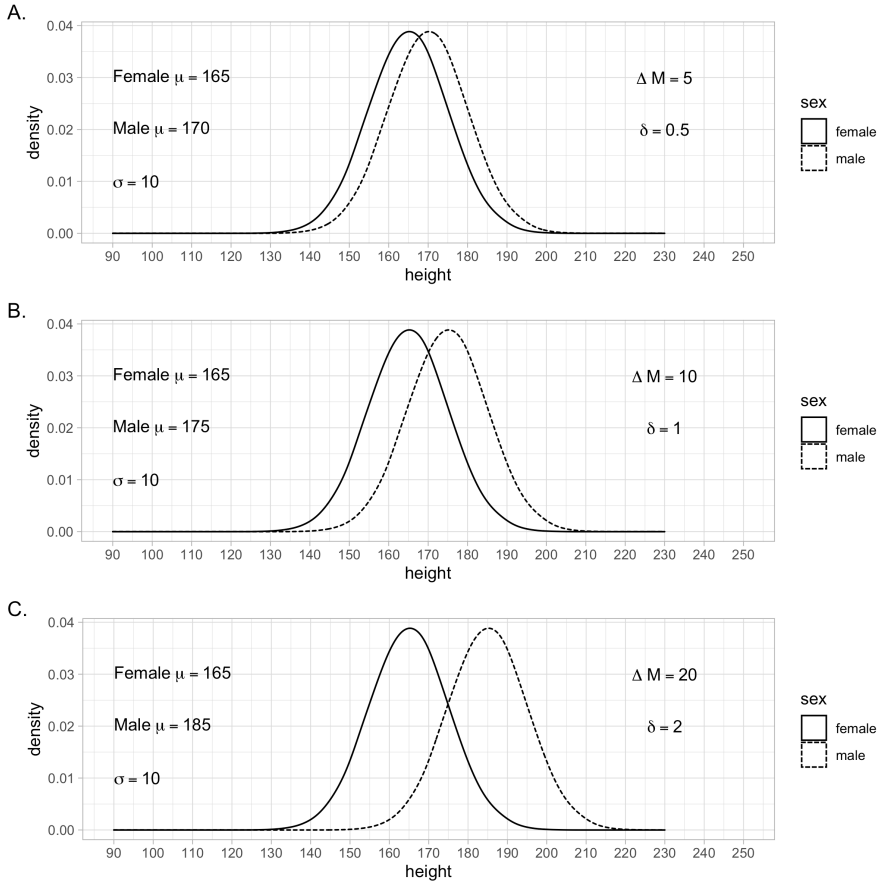


FIGURE 3.4: The difference between two population means can be expressed in the original units as indicated by ΔM. Alternatively, the difference can be expressed using a Standardized Mean Difference (SMD). The SMD index is also known as the population-level d-value and is represented by the symbol δ. The SMD is a way of expressing the difference between population means without using the original units.

Thus, taking into account the standard deviation of the populations can be viewed as a strength of using the standardized mean difference.

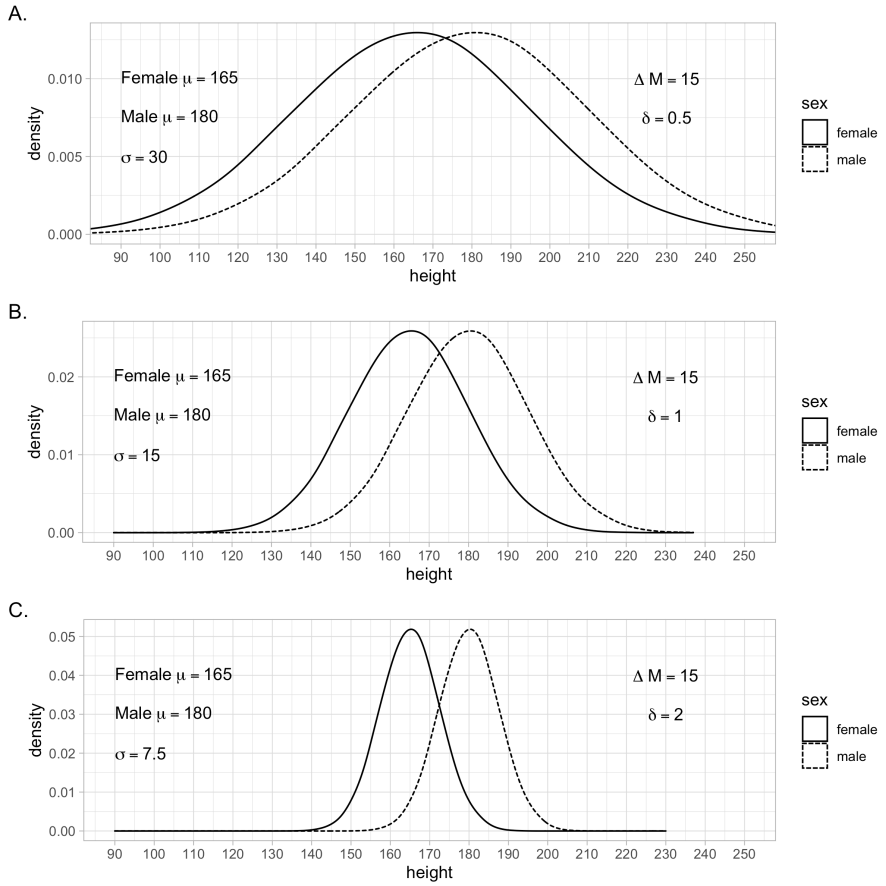


FIGURE 3.5: An advantage of using the Standardized Mean Difference (SMD) to index the difference between two population means (i.e., δ) is that it takes the population standard deviation into account. In these three examples, the difference between the populations means is the same using the original/raw units of centimeters. However, the standard deviation of the populations varies across scenarios A, B, and C. The SMD illustrates that these three scenarios are different. If you only examined the difference in the original units (i.e., ΔM) you would conclude the effect is the same across the three scenarios. However, by using SMD, indexed by δ - the population d-value, you see that the effect is progressively stronger from scenario A, to B, to C. This is illustrated by the fact that there is progressively less overlap between the distributions as you move from scenario A to C.

3.5.4 Cohen's d caveats

It is important to also look at the original units when interpreting results - not just the standardized mean difference. Examine the scenarios in Figure **??**. Notice how the δ value stays constant across scenarios - as does the overlap of the two distributions. However, inspect the shape of the curves and the original units to see how the scenarios vary. Both the original units and the standardized mean difference (i.e., Cohen's d) provide important interpretational information - don't rely on just one of them.

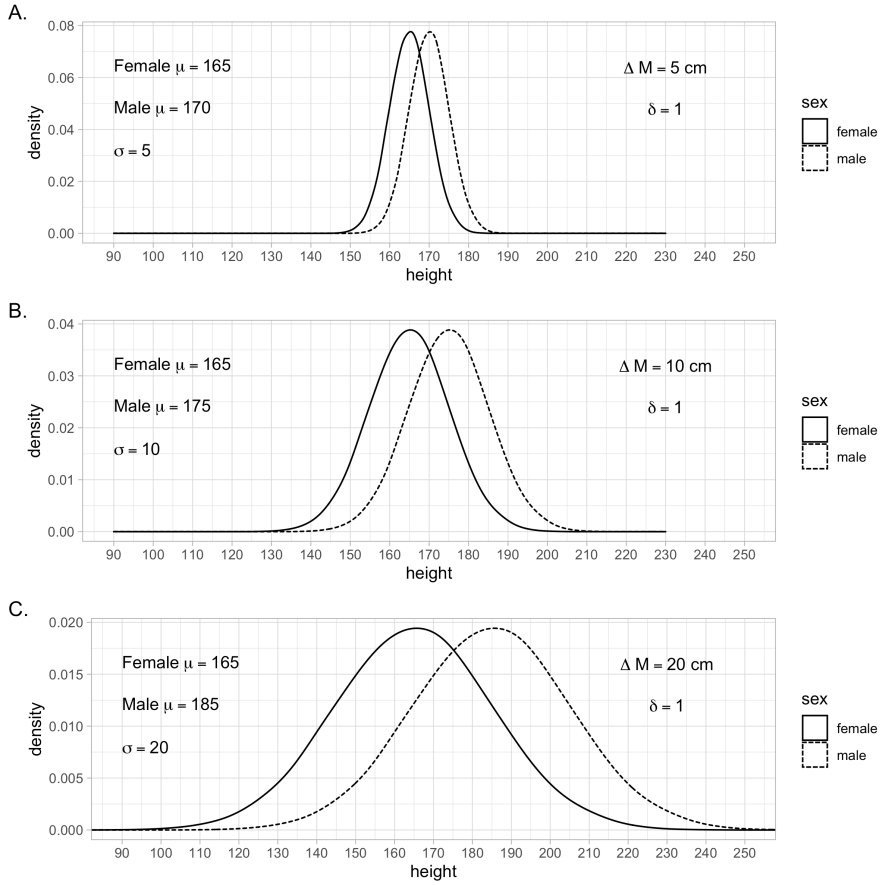


FIGURE 3.6: The three scenarios in this figure illustrate that a Standardized Mean Difference (i.e., population d-value or δ) can remain constant across scenarios when there is a change in the raw difference (i.e., ΔM) between the population means. This SMD is consistent across the three scenario despite a change in the mean difference using original units; this occurs because the standard deviations also changes across the three scenarios.

3.6 Comparisons: Different σ

The calculation of the standardized mean difference values (δ-values) above assumed that the two populations had identical standard deviations. In some scenarios, you might reasonably expect the standard deviations to be different for two populations. When the **populations** being compared have different standard deviations, we can still calculate a standardized mean difference (i.e, δ-value) but we need to pick one of the standard deviations to use in the formula. In many cases, it makes sense to think of one population as the frame of reference for the comparison; a "Control" population of sorts. In this case, you use the standard deviation of the control population as the reference/denominator when calculating the δ-value as illustrated in Formula (**??**) below. Make sure you recognize we are talking about a scenario with different population standard deviations (not sample standard deviations) as the appropriate situation for this type of calculation.

$$\delta = \frac{\mu_1 - \mu_2}{\sigma_{control}} \tag{3.5}$$

3.7 Comparisons: Repeated Measures

Sometimes we measures the members of a single population twice and are interested in the change across occasions. Consider a scenario where everyone in a large population ($N = 1000000$) attempts to loose weight over a given period of time. We weigh everyone in the population at time 1 before the weight loss attempt. Then we weigh everyone in the population at time 2 after the weight loss attempt.

Think of this scenario concretely with respect to how we would record this information. Imagine a large spreadsheet with 1000000 rows - each representing a person. There are two columns. The first column contains time 1 weights. The second column contains time 2 weights. We are interested in how weights changed across the times. So we create a third column, called diff, by subtracting time 1 weight from time 2 weights. That is, diff = time 2 weight - time 1 weight. The new diff column indicates how the weights for each person have changed over the diet. We can think of this single column of differences as being a population. This column is used to calculate the repeated-measures δ-value. Specifically, the mean of the diff column ($barx_{diff}$) and the standard deviation of the diff column (s_{diff}) are used in Formula (**??**) below:

$$\delta = \frac{\mu_{diff}}{\sigma_{diff}} \tag{3.6}$$

3.8 Comparison Benchmarks

Regardless of how the standardized mean difference (i.e., δ-value) is calculated, Cohen suggested that the values of 0.20, 0.50, and 0.80 correspond to the effect size labels of small, medium, and large, respectively (**?**). These effect sizes are illustrated in Figure **??**. As described in an interesting blog post[1], these benchmark values came from reviewing a single issue of the *Journal of Abnormal and Social Psychology*. Basing benchmarks on such a small number of studies is potentially problematic - as is the fact that the that time all the studies were prone to publication bias. A recent investigation (see **?**) of effect sizes in pre-registered studies, with no publication bias, suggests substantially lower benchmark values.

You can visualize any δ value (i.e., population d-value) using the rpsychologist website[2]. This website also provides a number of interesting statistics such as the percentage of overlapping values in two populations for a given δ/d value. Take a minute to use this website now.

Cohen's benchmarks for standardized mean differences are displayed in the table below. These effect size labels should be interpreted with caution. The magnitude of an effect is best considered in the context of the field of research and the consequences of an effect on individuals in both the short and long run.

Cohen (1988) Label	Value
Small	$\delta = .20$
Medium	$\delta = .50$
Large	$\delta = .80$

[1] https://replicationindex.com/2015/09/22/the-statistical-power-of-abnormal-social-psychological-research-a-revew-by-jacob-cohen/

[2] https://rpsychologist.com/d3/cohend/

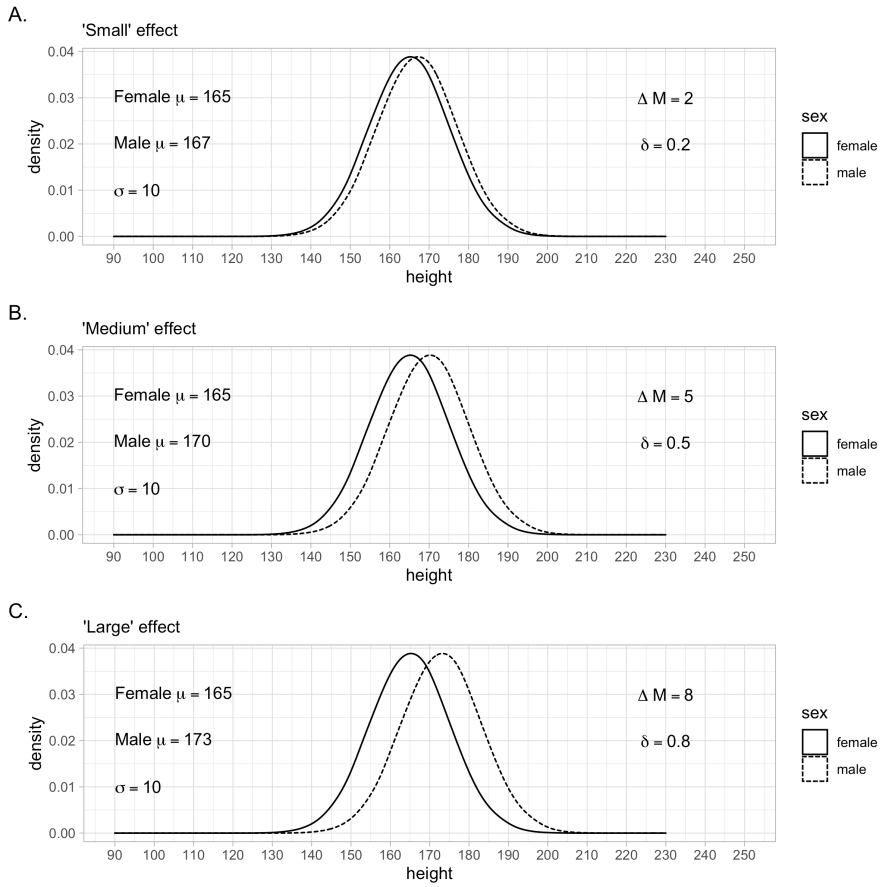


FIGURE 3.7: Cohen's (1988) effect size benchmarks

3.9 Population relations

Often researchers are interested in the extent to which one variable is related to another variable. For example, to what extent does variability in weight relate to variability in height? In other words, is there a relation between weight and height? One approach to this question to calculate a regression equation relating weight to height - this provides an index of the relation in the original units of the variables. Here we focus on a second approach to describing the relation between variable, namely, the correlation. The correlation is standardized effect size for the *linear* relation between two variables.

A population correlation is represented by the symbol ρ (pronounced rho) and calculated using Formula (**??**) below. The correlation may range from -1.00 to 1.00. A strong negative correlation indicates that as one variable increases the other decreases. A positive correlation indicates that as one variable increases the other increases. There are at least thirteen ways to conceptualize a correlations (see **?**) but it's easiest to think of it as an index of the extent to which the variables covary in a linear way.

$$\rho = \frac{\Sigma(X - \mu_X)(Y - \mu_Y)}{\sqrt{\Sigma(X - \mu_X)^2 \Sigma(Y - \mu_Y)^2}} \tag{3.7}$$

Because a correlation only provides an index of a linear relation - it is important to plot the data. Weak correlations (close to zero) may indicate there is not a linear relation. But there may still be a relation between the variables - just not one that follows a straight line. Indeed, the same correlation may take many different forms. Consider the data sets from the datasauRus package presented in Figure **??**. The graphs for each data set appear quite different. Yet, the following is true for all 12 data sets:

- The mean of X is 54.3 and the standard deviation is 16.8
- The mean of Y is 47.8 and the standard deviation is 26.9
- The correlation between X and Y is ρ = -.06

Therefore, make sure you ALWAYS graph your data. The numbers only tell part of the story.

3.10 Relation benchmarks

Plots of linear relations are presented in Figure **??**. The three graphs in this figure corresponds to Cohen's benchmarks for correlations (i.e., ρ), displayed in the table below.

Cohen (1988) Label	Value
Small	$\rho = .10$
Medium	$\rho = .30$
Large	$\rho = .50$

These effect size labels should be interpreted with caution. The magnitude of an effect is best considered in the context of the field of research and the consequences of an effect on individuals in both the short and long run.

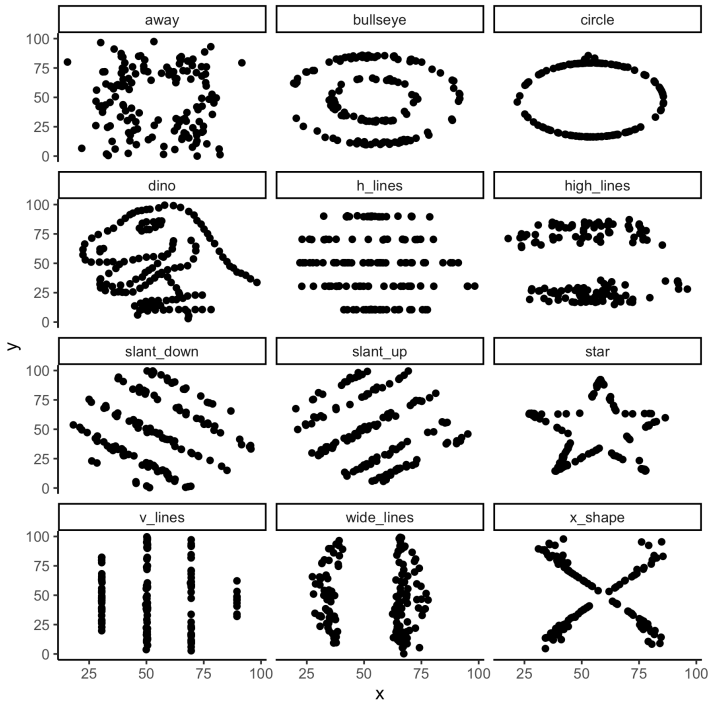


FIGURE 3.8: Various data sets with the same correlation, same means, and same standard deviations

3.11 Key points

1. Populations are described using numbers called parameters.
2. Population-level parameters are often represented using Greek letter.
3. Commonly used parameters include mean (μ or $\bar{X}$), variance (σ^2), or standard deviation (σ).
4. Individual scores can be expressed in standardized units.
5. Population differences can be described in original raw units or standardized units called the standardized mean difference (SMD).
6. SMD is based on the premise that the two population being compared have the same standard deviation.

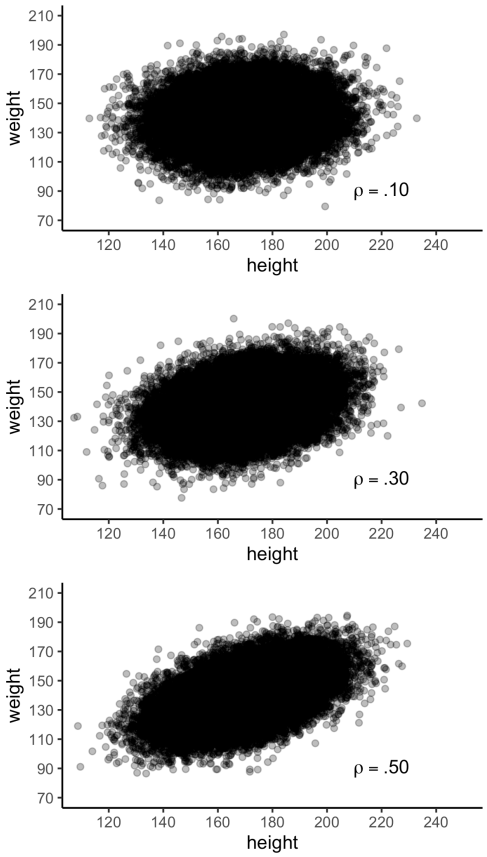


FIGURE 3.9: Populations of various strengths.

7. SMD is a ratio that compares two numbers (the numerator and the denominator).
8. Make sure you understand what is represented by both the numerator and denominator in the SMD ratio.
9. SMD (i.e., Cohen's d) represents the number of standard deviations between two population means. Recall both populations have the same standard deviation.
10. SMD is indicated at the population level using the Greek letter delta (δ). At the sample level we tend to use the term "d-value" or "Cohen's d".

3.12 Minor Points

1. Standardized mean differences are reported with a leading zero: $\delta = 0.50$.
2. Correlations are reported *without* a leading zero: $\rho = .30$.
3. The difference in reporting is an APA-style issue. Values that are bounded between 0 and 1 (or between -1 and +1) are reported without a leading zero (e.g., ρ and r). Values that are not bounded are reported with the leading zero (e.g., δ and d).

3.13 Self Assessment

1. What is the difference between parameters and statistics?
2. Are the formulas (excluding notation differences) always the same for parameters and statistics? If not, explain why not?
3. Grade 4 students in Ontario are taller than the Grade 3 students. Specifically, $\delta = 0.30$. How would you describe what this value means to an audience of experts at a conference? How would you describe what this values means to community members during a talk at the local public library?
4. We reviewed three different ways of calculating a standardized mean different (δ). What is the appropriate circumstance to use each formula? Describe a concrete scenario for each one.
5. There is a strong relation between two variables that follows and upside-down U-shape. Would you expect there to be a strong correlation between these two variables? Why or why not?

4

Graphing

4.1 Required

The data files below are used in this chapter. The files are available at: https://github.com/dstanley4/psyc6060bookdown

Required Data
data_movies.csv

The following packages CRAN must be installed:

Required CRAN Packages
tidyverse
RColorBrewer
remotes

The following GitHub packages must be installed:

Required GitHub Packages
dill/emoGG

After the remotes package is installed, it can be used to install a package from GitHub:

```
remotes::install_github("dill/emoGG")
```

4.2 Data

To learn about making graphs using the tidyverse we use movie ratings and box office data obtained at the time of writing. Movie ratings were obtained from the IMDB[1] and RottenTomatoes[2]. Box office data (in millions of dollars) was obtained from Box Office Mojo[3]. If you enjoy learning about movies these are all excellent sites.

We begin by loading data_movies.csv using read_csv(), not read.csv():

```
movie_data <- read_csv("data_movies.csv")
```

Next we inspect movie_data using the print() command. We see that each row of the data set corresponds to a superhero movie.

```
print(movie_data)
```

```
## # A tibble: 8 x 7
##   title short_title  year  imdb tomatoes_aud boxoffice
##   <chr> <chr>       <dbl> <dbl>        <dbl>     <dbl>
## 1 Iron~ Iron         2008   7.9           96       585
## 2 Thor~ Thor         2017   7.9           93       854
## 3 Aven~ AV3          2018   8.5           91      2048
## 4 Aven~ AV4          2019   8.7           91      2744
## 5 Man ~ Sup          2013   7.1           75       668
## 6 Batm~ BvS          2015   6.5           63       873
## 7 Just~ JL           2017   6.5           72       657
## 8 Wond~ WW           2017   7.5           88       821
## # ... with 1 more variable: studio <chr>
```

Next we use glimpse() to see the columns.

```
glimpse(movie_data)
```

```
## Rows: 8
## Columns: 7
## $ title        <chr> "Iron Man", "Thor Ragnarok", "Aven...
## $ short_title  <chr> "Iron", "Thor", "AV3", "AV4", "Sup...
## $ year         <dbl> 2008, 2017, 2018, 2019, 2013, 2015...
```

[1] https://www.imdb.com
[2] https://www.rottentomatoes.com
[3] https://www.boxofficemojo.com

```
## $ imdb         <dbl> 7.9, 7.9, 8.5, 8.7, 7.1, 6.5, 6.5,...
## $ tomatoes_aud <dbl> 96, 93, 91, 91, 75, 63, 72, 88
## $ boxoffice    <dbl> 585, 854, 2048, 2744, 668, 873, 65...
## $ studio       <chr> "Marvel", "Marvel", "Marvel", "Mar...
```

The title and short_title columns provide the full title and short title for each movie. Additionally, the IMDB rating, the Rotten Tomatoes Audience rating, and the Box Office Mojo revenue numbers are provided in the imdb, tomatoes_aud, and boxoffice columns, respectively. Finally, the last column, studio, indicates the studio that made the movie (Marvel or DC).

It is extremely important for graphing and analyses that you tell R which columns are composed of categorical variables. We do that using the as_factor command. The as_factor command turns a column into a categorical column. We use the mutate command to replace the original column with the column that has been defined as a categorical variables using as_factor.

```
movie_data <- movie_data %>%
  mutate(across(.cols = where(is.character),
                .fns = as_factor))
```

We can confirm the column type has changed by using the glimpse() command again and examing the column types:

```
glimpse(movie_data)
```

```
## Rows: 8
## Columns: 7
## $ title        <fct> Iron Man, Thor Ragnarok, Avengers ...
## $ short_title  <fct> Iron, Thor, AV3, AV4, Sup, BvS, JL...
## $ year         <dbl> 2008, 2017, 2018, 2019, 2013, 2015...
## $ imdb         <dbl> 7.9, 7.9, 8.5, 8.7, 7.1, 6.5, 6.5,...
## $ tomatoes_aud <dbl> 96, 93, 91, 91, 75, 63, 72, 88
## $ boxoffice    <dbl> 585, 854, 2048, 2744, 668, 873, 65...
## $ studio       <fct> Marvel, Marvel, Marvel, Marvel, DC...
```

4.3 Graph basics

In this section we teach you how to make a graph from first principles to form a foundation for understanding how the tidyverse graphing command ggplot() works. Note, however, that the approach used for creating a graph in this

section is for teaching purposes only. Later we will make graphs in a typical, and more efficient, manner.

We start a graph using the ggplot() command. The ggplot() command creates an empty template for the graph. After creating the template we have to add content (like bars) to the graph using the geom_col() command. We can also add text using the geom_text() command.

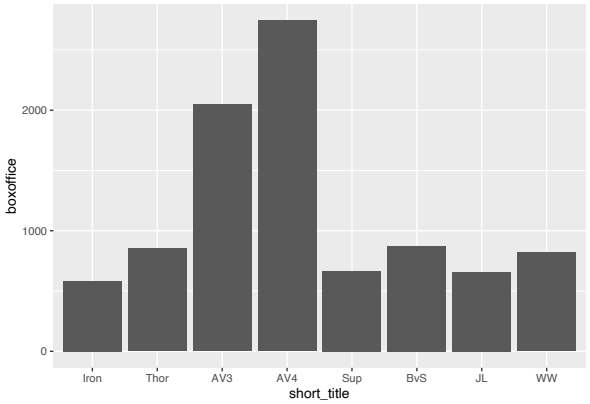
Commands that plot information on the graph, such as geom_col(), need to know what data set to use to create the graph. We specify the data set to use via the **data** argument. For example, we use "data = movie_data" to tell a command which data set to use.

Additionally, graphing commands, such as geom_col(), must know the columns/variables to use within that data set when plotting the graph. Specifically, commands need to know which variable/column will vary over the x- and y-axes. We can indicate these columns via the **mapping** argument. For example, we use "mapping = aes(x = short_title, y = boxoffice)" to tell ggplot() that we should use the column short_title along the x-axis and the column boxoffice when determining heights on the y-axis. This information is nested within the aes() command which is short for aesthetic. You are telling ggplot() about the aesthetics for the graph (i.e., which columns to use for the x- and y-axes) using the aes() command. There are a larger number of aesthetics that you can specify within the aes() command (e.g., color, fill, linetype, etc.).

In the examples that follow we tell each command (geom_col, geom_text) which data set and columns to use via the **data** and **mapping** arguments.

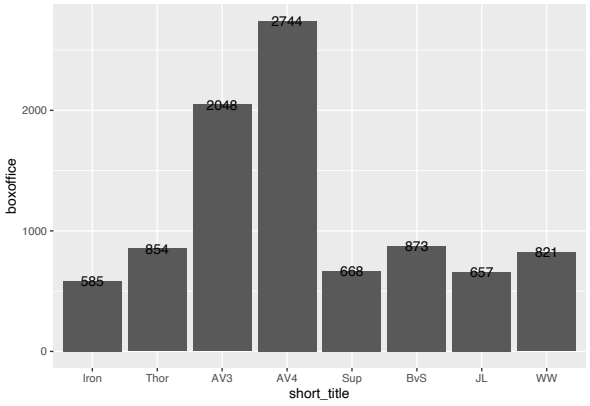
Use geom_col() to put each boxoffice column value into a bar.

```
my_graph <- ggplot() +
  geom_col(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice))

print(my_graph)
```



Next, we want to put the boxoffice revenue above each bar so it easier to interpret. In R terms, we are putting a label above each bar. We want the contents for the labels to come from the boxoffice column. Therefore, we add the geom_text() command below:

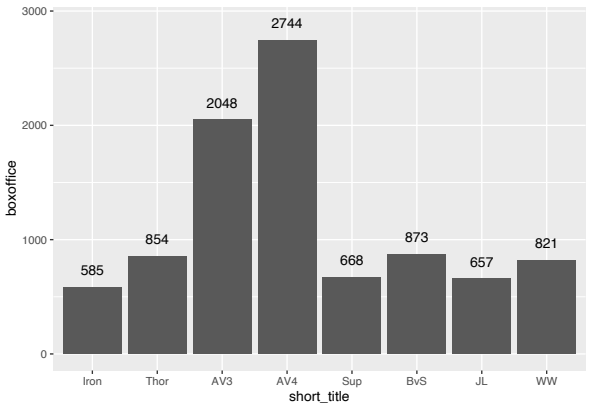
```
my_graph <- ggplot() +
  geom_col(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice)) +
  geom_text(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice,
                         label = boxoffice))
print(my_graph)
```



Unfortunately, when we position the text at the exact height of each column it overlaps with the column making it difficult to read. We fix this on the next page using nudge_y.

We can nudge each label higher on the y-axis using the *nudge_y* command. In the above code, we nudge it up 150 units. Since nudge_y uses the values on the y-axis we are nudging the labels up by 150 million on the y-axis.

```
my_graph <- ggplot() +
  geom_col(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice)) +
  geom_text(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice,
                         label = boxoffice),
           nudge_y = 150)

print(my_graph)
```



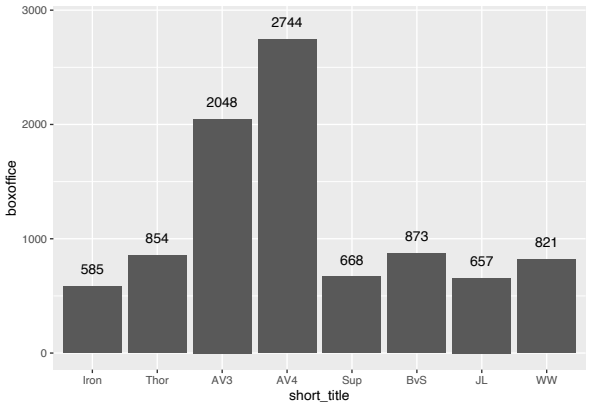
4.4 Graphing efficiently

You may have noticed that creating the graphs the way we did above required repeating the data and mapping assignments within each command (e.g., geom_text, geom_col). Fortunately, we can use a shortcut and specify the data and mapping only once in the ggplot() command. Once we do that, the contents of the mapping argument are invisibly copied into each subsequent command (e.g., geom_col, geom_text). In this way, we only have to specify the data and the mapping once.

Examine the code below and compare it to the code above. Notice how in

the geom_col() command we don't have anything specified – the data and mapping from the ggplot command are used. Likewise, notice how in the geom_text() command we only specify the arguments we need that are different from those in the ggplot command. In this case, that means simply adding the nudge_y = 150 to the geom_text command.

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice,
                         label = boxoffice)) +
  geom_col() +
  geom_text(nudge_y = 150)

print(my_graph)
```



4.5 Aesthetics

Exactly how does that aesthetic, aes(), command work? What happens when we put the data and mapping in the ggplot() command instead of the specific commands such as geom_col()? When we put data and the mapping arguments in the ggplot() command we set those attributes for the entire graph. To understand this, you need to know that ggplot uses an internal data set that we will call the "black box" data set (i.e., inside the black box[4] of ggplot). To create a graph ggplot maps/copies columns from your data set (e.g., movie_data) to an internal data set. This internal data set is then used to

[4] https://en.wikipedia.org/wiki/Black_box

create the graph. Figure **??** below illustrates what is happening "inside the black box" when you create the graph using the code above.

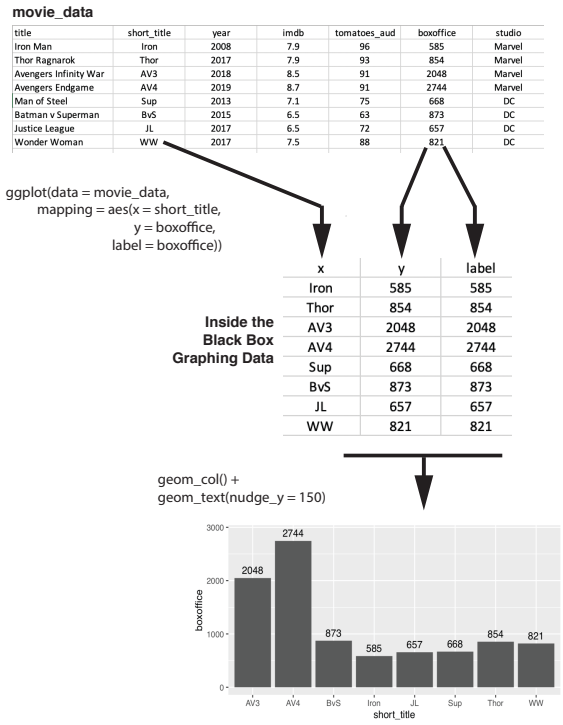


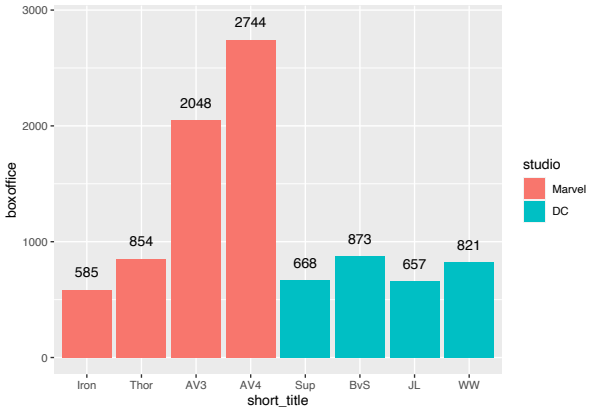
FIGURE 4.1: Internal data structure for ggplot

4.5.1 Fill color

You might want to influence the color of the bars in the graph such that the bars for Marvel and DC movies have different colors. That's easy to do with the aes() command. We simply tell aes() that the **fill** color of any object in the graph should be determined by the studio column. Simply adding "fill = studio" to the aes() command changes the colors of the bars – and any other object on the graph for which fill would be relevant. The internal workings are illustrated in Figure **??**.

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice,
```

```
                        label = boxoffice,
                        fill = studio )) +
  geom_col() +
  geom_text(nudge_y = 150)

print(my_graph)
```



4.5.2 Overriding aes()

Just because you specify something in the ggplot() command doesn't mean that you are "stuck with it" for all your subsequent commands. Recall how at the start of this exercise we specified the data and the mapping for each geom_col() and geom_text() individually. We can still do that.

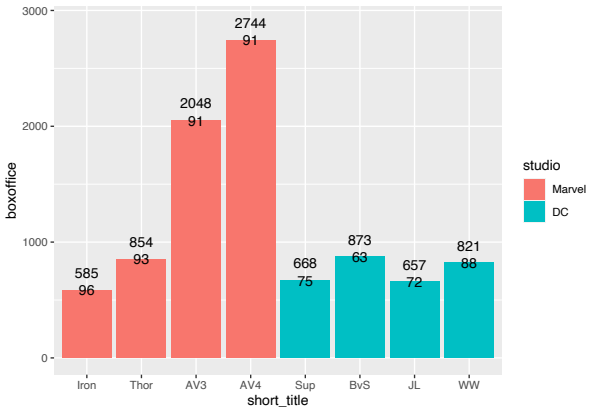
Now we want to add the Rotten Tomatoes Audience score for each movie above the box office revenue. But if we use geom_text(), like we did before, it will plot the same boxoffice information because of "label = boxoffice" in the aes() specification within ggplot(). We want the new geom_text() command to plot different text; that is, we want it to use "label = tomatoes_aud".

Fortunately, if we simply put "mapping = aes(label = tomatoes_aud)" within the new geom_text() command we get the desired information on the graph. The mapping/aes arguments within geom_text() override the mapping/aes arguments within ggplot(). Or more accurately, the new geom_text() command creates its own version of the internal data set in which the label column is filled with information from tomatoes_aud.

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = short_title,
```

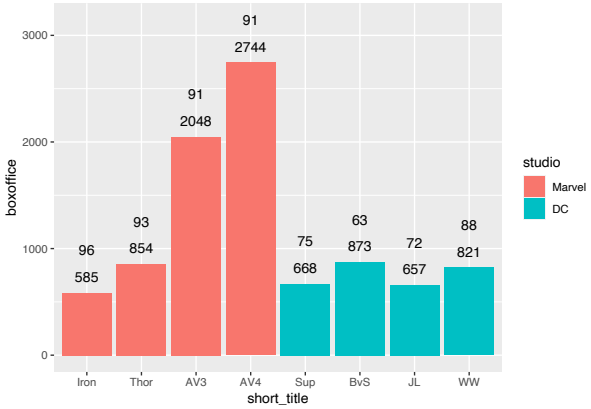
FIGURE 4.2: Adding a fill column to the internal data

```
                        y = boxoffice,
                        label = boxoffice,
                        fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud))
print(my_graph)
```



Notice that we have the same problem as before with the text being difficult to read because it overlaps with the bar. We add "nudge_y = 400" to move the text higher than the boxoffice text/label. Don't forget the units used by nudge_y are the units on the y-axis.

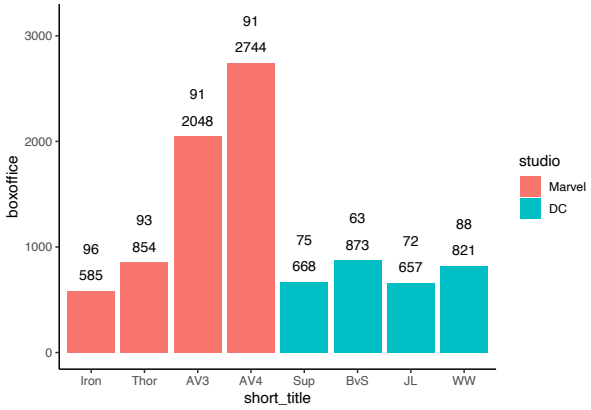
```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400)
print(my_graph)
```



4.6 APA style

Use theme_classic() to make the graph apear in APA style. We use theme_classic(12) to make the graph APA style with a 12-point font:

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  theme_classic(12)

print(my_graph)
```

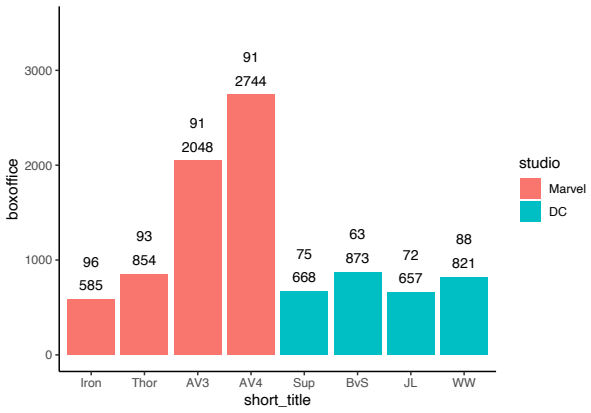


4.7 Axes

4.7.1 Range

We use the coord_cartesian() command to adjust range of axes. In the code below we use coord_cartesian() to make the y-axis range from 0 to 3500.

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  theme_classic(12)


print(my_graph)
```



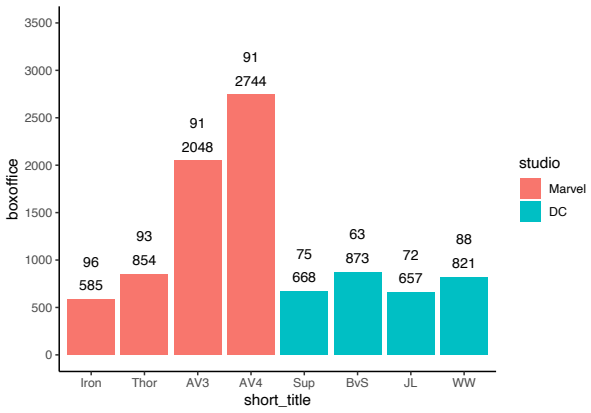
Note that if you had a continuous variable on the x-axis (we do not in this example), you could set the range of both the x- and y-axes like this:

```
coord_cartesian(ylim = c(0, 3500),
                xlim = c(0, 3500))
```

4.7.2 Ticks

We use the scale_y_continuous() command to adjust the ticks on the y-axis. We set the ticks on the y-axis to range from 0 to 3500 in 500 tick increments using the scale_y_continuous command below via the breaks argument:

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  theme_classic(12)

print(my_graph)
```



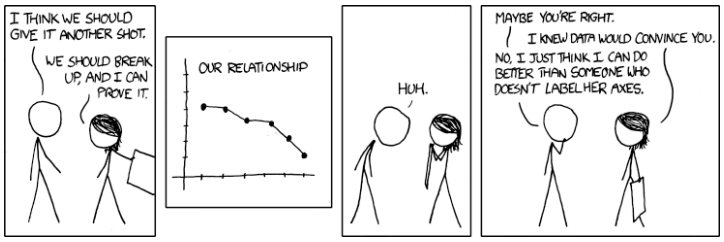
Note that if you had a continuous variable on the x-axis (we do not in this example), you could set the ticks of the x-axis like the code below using scale_x_continuous:

```
scale_x_continuous(breaks = seq(0, 3500, by = 500))
```

4.7.3 Labels

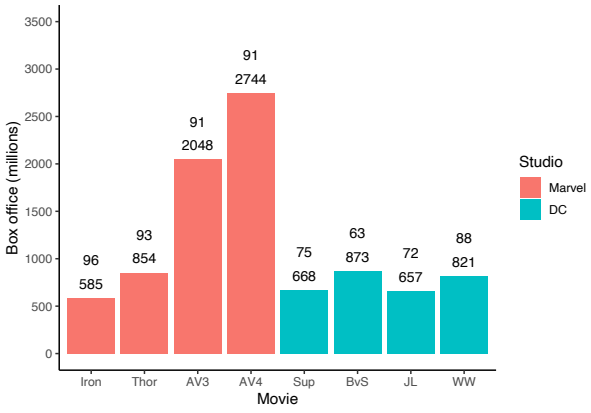
Labels are an extremely important part of any graph. This fact is the focus of the xkcd[5] cartoon below:

[5] https://xkcd.com/833/



We use the labs() command to set the labels for the x- and y-axes:

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12)

print(my_graph)
```



4.8 Axis values

4.8.1 Text

What if we want to use the full movie title rather than the short version on the x-axis of the graph? That is, you want to change the values along the x-axis. Two methods are presented below.

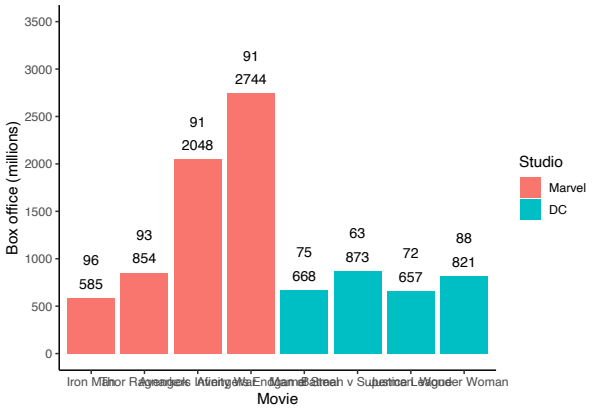
4.8.1.1 Method 1: Recoding axis values

Our data file contains a column with the long/full version of the movie names. But many times you won't have the additional/longer labels available in this manner. In this situation, you use the scale_x_discrete() command to change the values along the x-axis.

The values along the x-axis come from the short_title column which is a factor. The levels of that factor correspond to the short titles for the movies (Iron, Thor, etc.). We need to relabel the levels of the short_title factor to get a graph with full titles. We relabel the levels of the short_title factor using the scale_x_discrete() command. The graph code with scale_x_discrete() command is below - notice the problem we have with the labels overlapping though. On the next page, we'll use an easier approach though - since we have an extra column with the full titles.

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = short_title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  scale_x_discrete(labels=c("Iron" = "Iron Man",
                            "Thor" = "Thor Ragnarkok",
                            "AV3" = "Avengers Infinity War",
                            "AV4" = "Avengers Endgame",
                            "Sup" = "Man of Steel",
                            "BvS" = "Batman v Superman",
                            "JL" = "Justice League",
```

```
                              "WW" = "Wonder Woman")) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12)


print(my_graph)
```

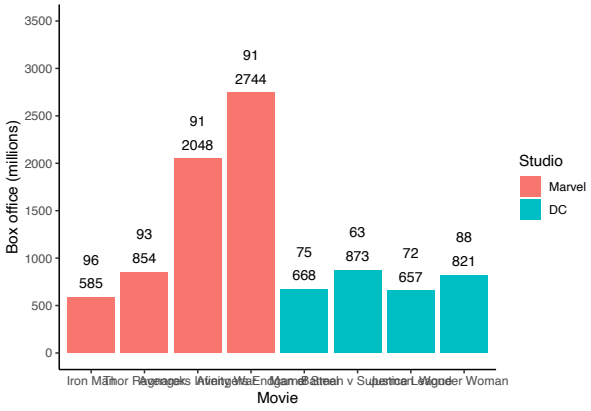


4.8.1.2 Method 2: Longer label columns

With our movie data we don't need to use scale_x_discrete() because we have a column in the data with the full titles. Consequently, to use full length titles we just have to change the *mapping*for x from short_title to title. Notice that we still have the problem with overlapping labels on the x-axis!

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
```

```
       fill = "Studio") +
  theme_classic(12)


print(my_graph)
```



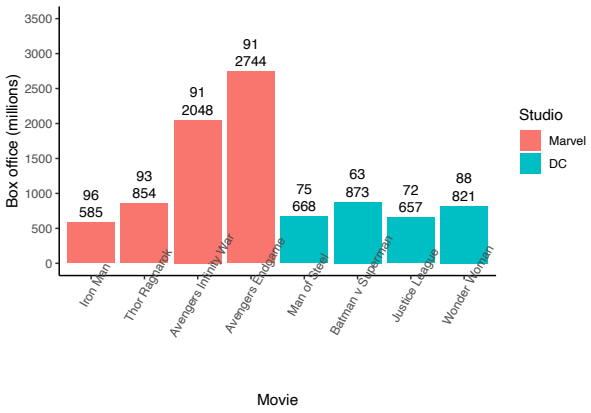
4.8.2 Angle

Use theme() to adjust the angle of x-axis labels. Notice, however, that the longer titles are vertically centered on each point on the x-axis. In the next section we fix this problem.

Important: The theme command must come after the theme_classic command. Otherwise, theme_classic will undo the work done by the theme_command if it appears after the theme command.

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
```

```
       fill = "Studio") +
  theme_classic(12) +
  theme(axis.text.x = element_text(angle = 60))

print(my_graph)
```

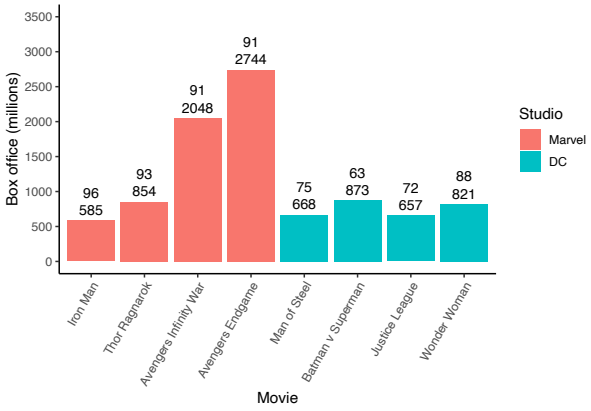


4.8.3 Alignment

Use theme() and the hjust argument to adjust the alignment of the x-axis labels. To make the titles look correct on the x-axis we need them at an angle, but we also need them right justified against the x-axis. We do that with the the hjust argument (1 means right justify). See the code below:

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
```

```
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1))

print(my_graph)
```



4.8.4 Order

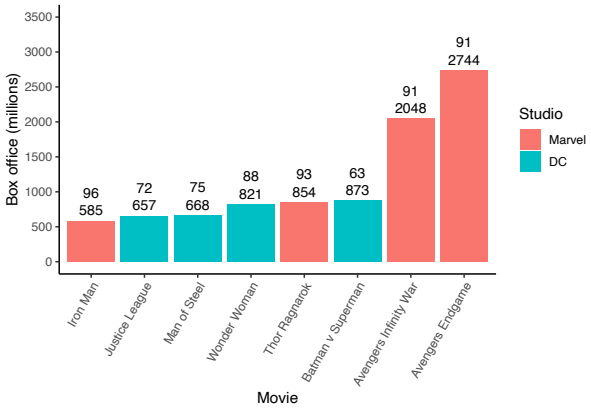
4.8.4.1 Increasing order

We can make the movie bars go left to right in lowest to highest box office receipt order by changing the factor order prior to creating the graph. We do so with the mutate() and fct_reorder() commands. The default order is ascending values even though we don't specify it.

```
movie_data <- movie_data %>%
  mutate(title = fct_reorder(title,
                             boxoffice))
```

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
```

```
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1))

print(my_graph)
```



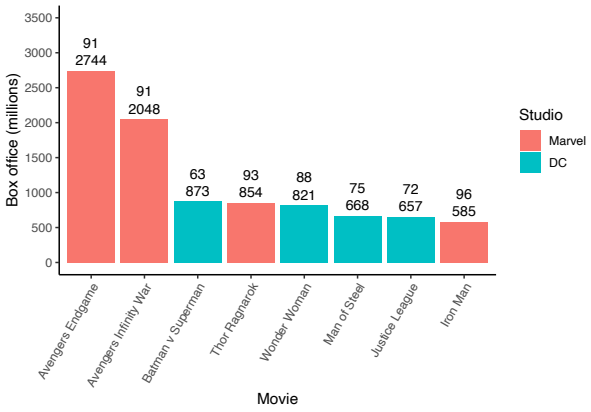
4.8.4.2 Decreasing order

We can make the movie bars go left to right in highest to lowest box office receipt order by changing the factor order prior to creating the graph. We use the same code as before but add the desc() command (i.e.,descending) around boxoffice in the fct_reorder() call:

```
movie_data <- movie_data %>% mutate(title = fct_reorder(title,
                                    desc(boxoffice)))
```

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
```

```
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1))

print(my_graph)
```



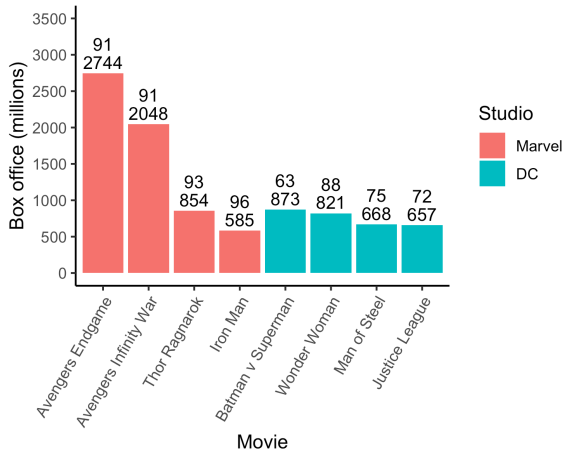
4.8.4.3 Custom order

We can make the movie bars go left to right in a custom order by changing the factor order prior to creating the graph. Because we want a custom order of the factor levels we use fct_relevel(), instead of the the fct_reorder() command from the previous two examples. Below I use this approach to manually order movies highest to lowest boxoffice within movie studio (Marvel or DC).

```
movie_data <- movie_data %>%
  mutate(title = fct_relevel(title,
                             "Avengers Endgame",
                             "Avengers Infinity War",
                             "Thor Ragnarok",
                             "Iron Man",
                             "Batman v Superman",
                             "Wonder Woman",
```

```
                                "Man of Steel",
                                "Justice League"))
```

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1))
```

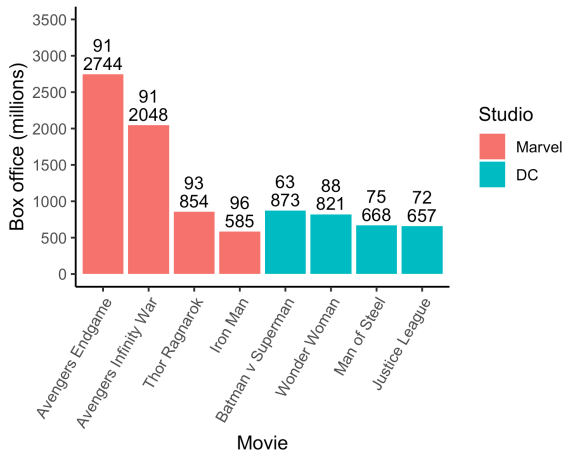


4.8.5 Legend order

After inspecting the graph on the previous page, you might think that Marvel should be above DC in the legend. You can do that by reordering the studio factor:

```
movie_data <- movie_data %>%
  mutate(studio = fct_relevel(studio,
                              "Marvel",
                              "DC"))
```

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1))
```



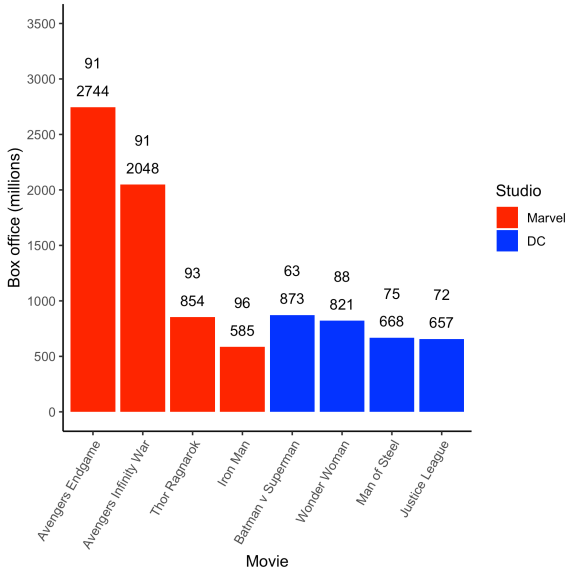
4.9 Custom colours

4.9.1 R palette

You might look at the previous graph and think "Marvel should be red and DC should be blue since those are the colours of their respective logos". You can do that with the code below. Note that you specify the colours in the order the names appear in the legend (top to bottom).

R colour names/pictures can be found here: http://sape.inf.usi.ch/quick-reference/ggplot2/colour

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1)) +
  scale_fill_manual(values = c("red", "blue"))
```

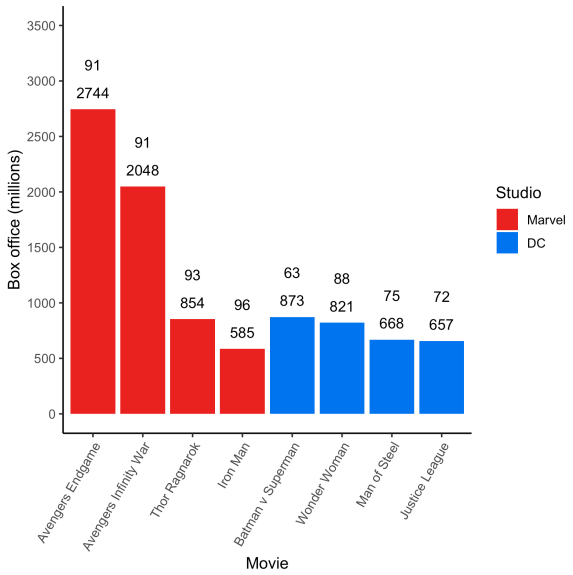


4.9.2 Hex colours

If you are a really big geek (like me) you might look at the previous graph and think "Those aren't the proper colours for the Marvel and DC - lame!" So... you do some internet research and determine that you can specify colours using hexidecimal numbers. More specifically, you find Marvel red is #ed1d24 and DC blue is #0476F2 using hex colour codes. You can use those precise colours via the scale_fill_manual() command below.

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 400) +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
```

```
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1)) +
  scale_fill_manual(values = c("#ed1d24", "#0476F2"))
```



4.10 Emoji

Make the graph more fun with the emoGG package. You might like to make the graph more fun by putting tomatoes on the graph to indicate what the extra numbers mean. We can do that with the emoGG package. The installation instructions for this package are at the start of this chapter; note, that it is installed via GitHub rather than the CRAN. Course R Studio Cloud users - the installation has already been done.

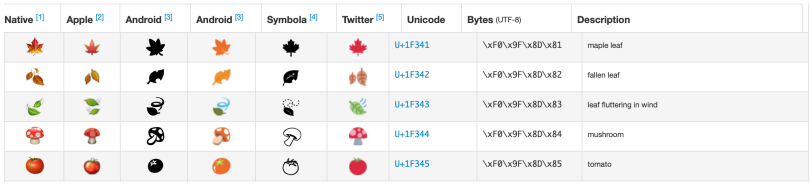
After installation you need to activate the emoGG package:

```
library(emoGG)
```

Visit this link to check out the codes for emoji: https://apps.timwhitlock.info/emoji/tables/unicode

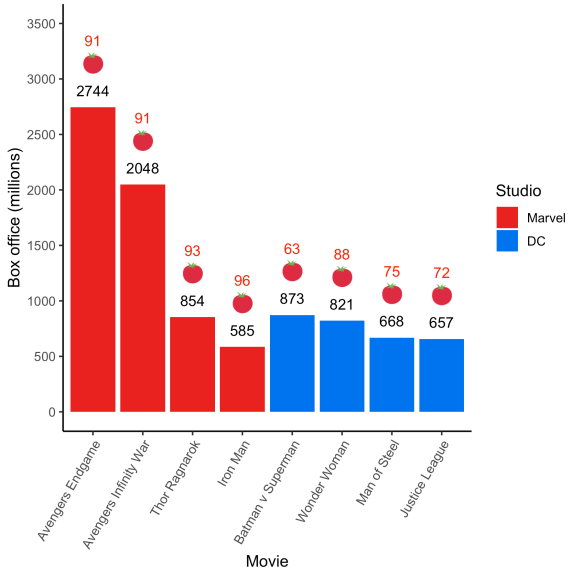
If you scroll down to section 5 Uncategorized on this page you will find the

code for a tomato is 1f345. Note that the code below will only work with an internet connection. The command geom_emoji() needs internet access to retrieve the emoji graphic requested.



Native [1]	Apple [2]	Android [3]	Android [3]	Symbola [4]	Twitter [5]	Unicode	Bytes (UTF-8)	Description
						U+1F341	\xF0\x9F\x8D\x81	maple leaf
						U+1F342	\xF0\x9F\x8D\x82	fallen leaf
						U+1F343	\xF0\x9F\x8D\x83	leaf fluttering in wind
						U+1F344	\xF0\x9F\x8D\x84	mushroom
						U+1F345	\xF0\x9F\x8D\x85	tomato

```
my_graph <- ggplot(data = movie_data,
            mapping = aes(x = title,
                          y = boxoffice,
                          label = boxoffice,
                          fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 600,
            colour = "red") +
  geom_emoji(mapping = aes(y = boxoffice + 400),
             emoji="1f345") +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1)) +
  scale_fill_manual(values = c("#ed1d24", "#0476F2"))
```



4.11 Accessible Colors

The current version of the graph is much improved from where we started at the begining of the chapter. One notable improvement was the use of the "proper" colors for the Marvel and DC studios. Although using these colors was aethetically pleasing, a major consideration is ensuring your graphs are accessible to a wide audience.

Color blindness is an issue that affects approximately five percent of the population. A nuanced discussion of the different types of color blindness is beyond the scope of this chapter. We can, however, take a momement to think about every graph as being composed of two parts that work together to create the colors overall image. We can think of there being a lightness (i.e., light vs. dark) component and a hue component (e.g., magenta, yellow, etc.). Together these two components work together to create colors that we see. This distinction between lightness and hue is relevant to all images not just graphs (see **?**).

Figure **??**A presents our current graph whereas Figure **??**B presents the same graph with the hue component removed. You can see that when the hue component is removed that the distinction between the Marvel and DC bars is also removed. The particular colors used to represent Marvel/DC differ in terms of hue but not lightness. Consequently, when we remove the hue component we

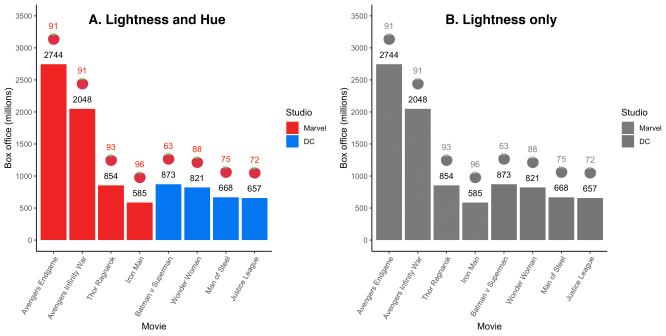


FIGURE 4.3: Hue removed from current graph

are left with a graph, **??**B, that does not differentiate between the two studios. In order for a color graph to be accessible to people with color blindness we need to pick colors that vary in terms of lightness as well as hue. We can do that with the help of the RColorBrewer package.

4.11.1 RColorBrewer

4.11.1.1 Picking a palette

The RColorBrewer package can be used to generate color palettes for graphs that are accessible to people with color blindness. That is, it creates sets of colors, called palettes, for which the colors vary in terms of both lightness and hue. You can see the color-blind accessible palettes, along with their respective names, by using the command below. This code produces a wide range of color-blind accessible palettes with a large number of colors in each palette - as illustrated in Figure **??**.

```
library(RColorBrewer)
display.brewer.all(n = NULL, colorblindFriendly = TRUE)
```

FIGURE 4.4: Wide range color palette (color-blind accessible)

In Figure **??**, above, we showed palettes with a large number of colors. But if you have a smaller number of possible colors in your graph - you want a palette with fewer colors (to ensure maximum contrast between those colors). You can, for example, obtain palettes with only three colors using the code below.

```
library(RColorBrewer)
display.brewer.all(n = 3, colorblindFriendly = TRUE)
```

The above code generates the three-color palettes presented in Figure **??**.

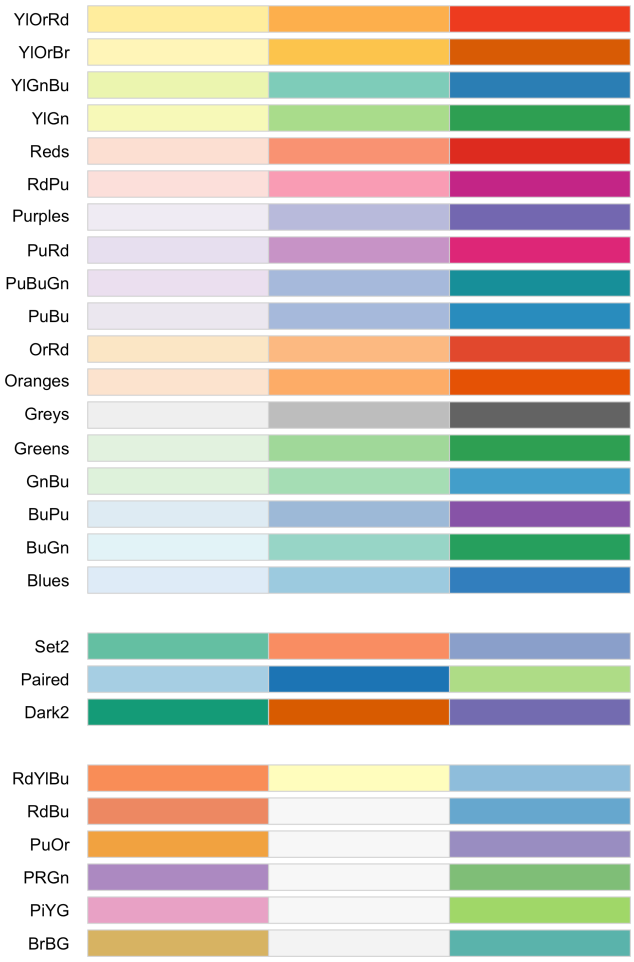


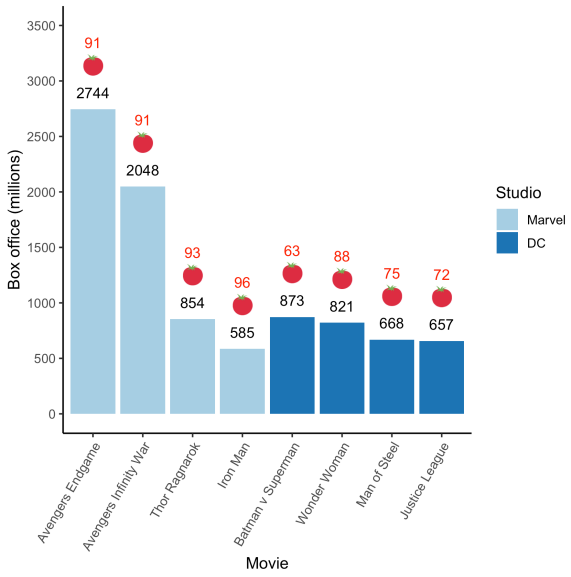
FIGURE 4.5: Narrow range color palette (color-blind accessible)

4.11.1.2 Using a palette

We can use a color palette by specifying its name within scale_fill_brewer(). We want to use the "Paired" palette in Figure **??** so we use the code: scale_fill_brewer(palette = "Paired"), as illustrated below:

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
```

```
geom_col() +
geom_text(nudge_y = 150)  +
geom_text(mapping = aes(label = tomatoes_aud),
          nudge_y = 600,
          colour = "red") +
geom_emoji(mapping = aes(y = boxoffice + 400),
           emoji="1f345") +
coord_cartesian(ylim = c(0, 3500)) +
scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
labs(x = "Movie",
     y = "Box office (millions)",
     fill = "Studio") +
theme_classic(12) +
theme(axis.text.x = element_text(angle = 60,
                                 hjust = 1)) +
scale_fill_brewer(palette = "Paired")
```



4.11.1.3 Palette subsets

Our graph had only two colors but the "Paired" palette had three colors. When ggplot made the graph it automatically used the first two colors of the three color palette. What if you wanted the second two colors in the palette? We can do that but we have to revert to a manual color process like we used before via the scale_fill_manual(). But we want to do so using the colors from

the "Paired" palette. We can obtain the color codes for the "Paired" palette with the code below:

```
brewer.pal(n = 3, name = "Paired")
```

```
## [1] "#A6CEE3" "#1F78B4" "#B2DF8A"
```

The numbers in the output correspond to values representing the three colors of the "Paired" palette, see Figure **??**. If we want the last two colors in the palette for our graph that means we want the colors: #1F78B4 and #B2DF8A.

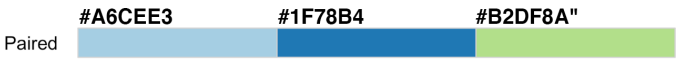
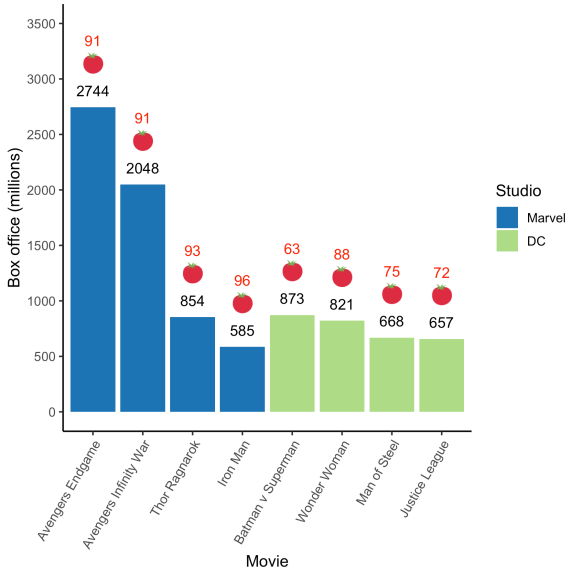


FIGURE 4.6: Paired palette colors with hex numbers

We can put the two colors we want (#1F78B4 and #B2DF8A) on the graph by using the scale_fill_manual() command - instead of the scale_fill_brewer() command. We can see this in the code below:

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 600,
            colour = "red") +
  geom_emoji(mapping = aes(y = boxoffice + 400),
             emoji="1f345") +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1)) +
  scale_fill_manual(values = c("#1F78B4", "#B2DF8A"))
```



Now that we have used a color-blind accessible palette in the graph we can look at the graph without the hue component. In Figure **??**B, below, notice that the bars representing the two studios are easily distinguishable based on lightness alone. That is, the lightness graph in Figure **??**B demonstrates the new color-blind accessible colors vary in term of both hue and lightness - not just hue. Therefore, when hue is removed the two colors are distinguishable. This makes the graph accessible to color-blind individuals.

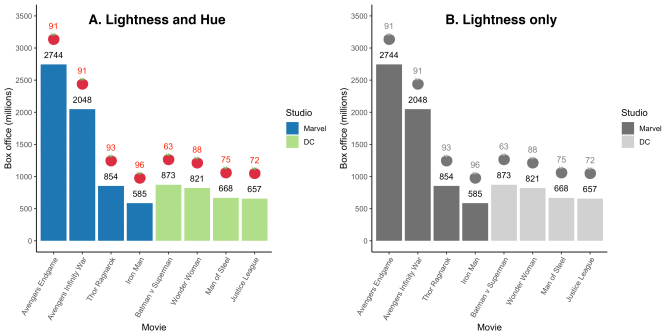
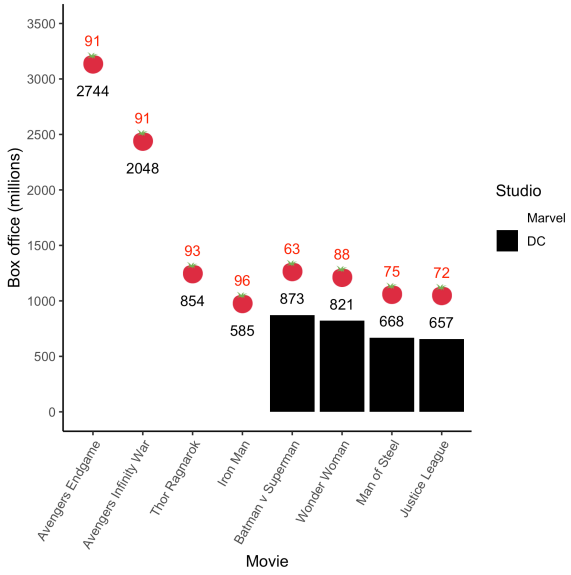


FIGURE 4.7: Color information (i.e., hue) removed from new color-blind accessible graph

4.11.2 Avoid color

When you only have two colors on a graph another option for creating an accessible graph is to remove the colors entirely and make a black and white graph. In the code below we use scale_fill_manual() with the values "#ffffff" (white) and ""#000000" (black). The intent was to create bars that are white for Marvel and black for DC. Unfortunately, because the graph has a white background you can see this resulted in the Marvel bars disappearing.

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col() +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 600,
            colour = "red") +
  geom_emoji(mapping = aes(y = boxoffice + 400),
             emoji="1f345") +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1)) +
  scale_fill_manual(values = c("#ffffff", "#000000"))
```

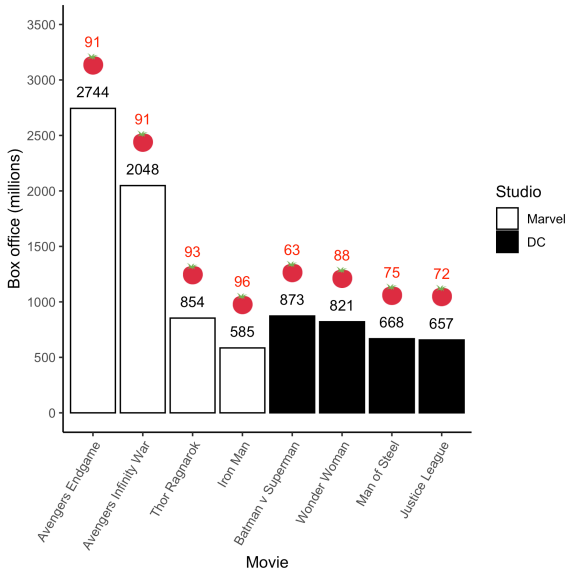


How do we get around the Marvel bars disappearing when we use white?

Color vs. fill. The ggplot packages makes a distinction between **fill** and **color**. The term **fill** is used to refer to the inside color of bars. In contrast, **color** is used to the refer to the lines that outline the shape of each bar. In the previous code we used fill to change the colors of the bars but we did not use the "color" argument in a command. In the code below, we modify the geom_col() command to use the "color" argument. Specifically, we change geom_col() to geom_col(color = "black"). This change adds a black outline around all the bars – including the ones with a white fill. You can see the result is a much improved graph. This black/white graph is accessible to everyone.

```
my_graph <- ggplot(data = movie_data,
           mapping = aes(x = title,
                         y = boxoffice,
                         label = boxoffice,
                         fill = studio)) +
  geom_col(color = "black") +
  geom_text(nudge_y = 150)  +
  geom_text(mapping = aes(label = tomatoes_aud),
            nudge_y = 600,
            colour = "red") +
  geom_emoji(mapping = aes(y = boxoffice + 400),
             emoji="1f345") +
  coord_cartesian(ylim = c(0, 3500)) +
  scale_y_continuous(breaks = seq(0, 3500, by = 500)) +
```

```
  labs(x = "Movie",
       y = "Box office (millions)",
       fill = "Studio") +
  theme_classic(12) +
  theme(axis.text.x = element_text(angle = 60,
                                   hjust = 1)) +
  scale_fill_manual(values = c("#ffffff", "#000000"))
```



4.12 Saving

If you have a Mac it is easy to drag and drop a PDF file into MS Word - so making a PDF file is the best bet for saving your graph. You can do so with the code below which creates a 6 inch by 6 inch graph.

4.12.1 MAC

If you are able to use PDFs in your workflow that's often the best option for saving. PDFs are mathematical in nature and therefore can be printed at any size at high quality. With a MAC you can just drag and drop the PDF file into your MSWord document.

```
ggsave(plot = my_graph,
       filename = "emoji_graph.pdf",
       width = 6,
       height = 6)
```

4.12.2 PC or MAC

If you have a PC it's hard to put a PDF into MSWord. Therefore, save the graph as a .jpg file. You do so with the code below. This creates a picture type file at a resolution (dpi = dots per inch) that is sufficiently high for quality printing.

With a PC you need to use the INSERT menu and insert the graph as a picture in MSWord. With a MAC you can just drag and drop the .jpg file into your MSWord document.

```
ggsave(plot = my_graph,
       filename = "emoji_graph.jpg",
       width = 6,
       height = 6,
       dpi = "print")
```

5

Sampling Accuracy

The following CRAN packages must be installed:

Required CRAN Packages
tidyverse
remotes

The following GitHub packages must be installed:

Required GitHub Packages
dstanley4/learnSampling

A GitHub package can be installed using the code below:

```
remotes::install_github("dstanley4/learnsampling")
```

5.1 Overview

Researchers are usually interested in describing the attributes of a population; numbers that describe the population are called parameters. Two parameters that are frequently of interest are the mean and variance of the population. Unfortunately, it's rarely possible to obtain information from every member of a population to calculate a parameter. Consequently, researchers use subsets of the population called samples to estimate parameters. Numbers calculated from sample data are called statistics. Typically, sample statistics are used to estimate population parameters.

Sample statistics, however, often differ from population parameters. The difference between a sample statistic and the population parameter occurs because

the sample data is random subset of the population data — with correspondingly fewer observations. This difference is typically refered to as **sampling error**.

Sometimes a sample statistic will be higher than the population parameter; other times the sample statistic will be lower than the population parameter. Because random sampling is used to select the sample data the direction and magnitude of the difference between the sample statistic and the population parameter will vary randomly.

Sample accuracy refers to the extent to which sample statistics correctly estimate the population parameter. We typically used the terms biased and unbiased to describe the accuracy of sample statistics. Consider a scenario where we take many thousands of samples from the same population. For each sample, we calculate a statistic (e.g., the mean). If the average of the sample statistics (e.g., sample means) equals the population parameter (e.g., population mean) then we refer to the statistic as being unbiased. In contrast, if the average of the sample statistics (e.g., sample means) does not equal the population parameter (e.g., population mean) then we refer to the statistic as being biased.

Further complicating matters is the fact that the formula used for a sample statistic may, or may not, be the same as the formula used for the corresponding population parameter. This occurs because the purpose of the sample statistic is typically not to describe the sample. Rather the purpose of the sample statistic is to estimate the population parameter. Depending on the parameter, you may or may not be able to use the same formula with sample data as you would with population data.

Also keep in mind that, even if you conduct experiments, the distinction between samples and populations is relevant to you. Consider a scenario where you run an experiment to test the effectiveness of a particular drug. Half the rats are assigned to a placebo condition (e.g., saline injection) whereas the other half of the rats are assigned to the drug condition (e.g., drug injection). Recognize that the placebo-condition rats are considered a sample from a much larger population of all rats who could have received the placebo. Likewise, the drug-condition rats are considered a sample of a much large population of all rats who could have received the drug. Indeed, when you conduct your analyses on this experiment the results do not tell you about the rats in your study - rather they tell you about rats in general (i.e., the larger populations of rats). Therefore, when we discuss the importance of estimating a population parameter from a sample realize that it applies to both experimental and survey research.

5.2 Data for the chapter

In this chapter we will use a population of heights to learn about random sampling. To engage in the learning activities you need to activate the required packages:

```
library(tidyverse)
library(learnSampling)
```

Next, we create a large population with 100,000 people using the get_height_population() command:

```
pop_data <- get_height_population()
```

The print() command can be used to confirm that the population contains 100,000 people. We see that each row in pop_data represents a single person. There is a column called height that contains the heights for everyone in the population.

```
print(pop_data)
```

```
## # A tibble: 100,000 x 3
##       id sex    height
##    <int> <chr>   <dbl>
##  1     1 male      177
##  2     2 female    150
##  3     3 female    171
##  4     4 male      157
##  5     5 male      169
##  6     6 male      187
##  7     7 female    163
##  8     8 male      173
##  9     9 female    172
## 10    10 male      193
## # ... with 99,990 more rows
```

5.3 Notation

In the formulas below, when we refer to the population, we use uppercase letters to indicate members (X) or the size (N). The population mean is indicated by the symbol μ. In contrast, when we refer to the sample, we use lowercase letters to indicate members (x) or the size (n). The sample mean is indicated by the symbol $\bar{x}$. A single bar above a letter indicates a mean. If we calculate the average of several sample means we indicate this with the symbol $\bar{\bar{x}}$. A double bar above a letter indicates a mean of means. Make sure you notice the similarities between subsequent population and sample formulas even though the notation often differs.

5.4 Estimating μ

We are interested in the sample mean ($\bar{x}$) to the extent that it provides an accurate estimate of the population mean (μ). The population mean is calculated using Formula (**??**). In this formula, the letter N corresponds to the number of people in the population.

$$\mu = \frac{\sum X}{N} \tag{5.1}$$

We can calculate the population mean for the height column of pop_data using the summarise() and mean() commands. The mean() command uses Formula (**??**). We see in the output that the population mean is 172.48 ($\mu = 172.48$).

```
pop_data %>%
  summarise(pop_mean = mean(height)) %>%
  as.data.frame()
```

```
##   pop_mean
## 1    172.5
```

As noted previously, we rarely have access to data from an entire population. Consequently, we use the sample mean as an estimate of the population mean. The sample mean, $\bar{x}$, is a statistic calculated using the using Formula (**??**) below. The bar above the x, indicates that it is a mean. Notice that Formula (**??**) and Formula (**??**) are the same - even though they use different notation. In this formula, the letter n corresponds to the number of people in the sample.

$$\bar{x} = \frac{\sum x}{n} \tag{5.2}$$

Because a sample mean (a statistic) is calculated using a random subset of the population it is likely to differ from the population mean (a parameter). If you, inaccurately, believe that you can learn something meaningful from a single study, this fact may be disconcerting. Statisticians know, however, that rarely can you learn anything from a single study, or even a small set of studies. Consequently, they are more interested in the extent to which sample means are right, on average. That is, they are interested in the extent to which the mean of many sample means ($\bar{\bar{x}}$) corresponds to the population mean (μ). The mean of many sample means can be calculated using Formula (**??**) below. In this formula, the letter k corresponds to the number of sample means.

$$\bar{\bar{x}} = \frac{\sum \bar{x}}{k} \tag{5.3}$$

If the mean of the sample means, $\bar{\bar{x}}$, equals the population mean, μ, then the sample mean is an unbiased (or accurate) estimate of the population mean. Figure **??** illustrates the concept of accuracy/bias with a distribution of sample means (i.e., $\bar{x}$). Accuracy/bias is an index of the extent to which the mean of many sample means, $\bar{\bar{x}}$, deviates from the population mean, μ.

We can assess bias, as illustrated in the above figure by drawing a large number of samples from a population with the code below. Our goal is calculate a mean for each sample so that we have a sampling distribution of means. In theory, we should take an infinite number of samples, however, to be practical we will take 50000 samples to create an approximate sampling distribution of means. We use the code below to do so:

```
many_samples <- get_M_samples(pop.data = pop_data,
                              pop.column.name = height,
                              n = 10,
                              number.of.samples = 50000)
```

```
many_samples <- readRDS("ch_samples/many_samples_n10.RDS")
```

We use the print() command to see the first few rows of the 50000 samples:

```
print(many_samples)
```

```
## # A tibble: 50,000 x 5
##    study     n sample_mean sample_var_n sample_var_n_1
##    <int> <dbl>       <dbl>        <dbl>          <dbl>
```

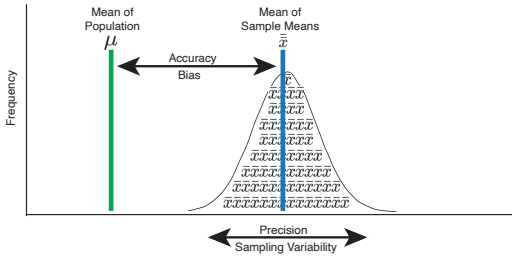


FIGURE 5.1: Sampling accuracy and precision

```
##  1     1    10        174.         27.8          30.9
##  2     2    10        162.         83.4          92.6
##  3     3    10        173.        139.          155.
##  4     4    10        180         322.          358.
##  5     5    10        173.        161.          179.
##  6     6    10        172         160.          177.
##  7     7    10        166.         86.5          96.1
##  8     8    10        174.        139.          155.
##  9     9    10        178.        167.          186.
## 10    10    10        168.        263.          292.
## # ... with 49,990 more rows
```

Each row of many_samples represents a sample of 10 people. Each column

of many_samples indicates a sample statistic. You can see that for each sample/row we indicate "n" (the sample size) and "sample_mean" (the mean of the population), and a few other statistics. Even though all the samples came from the same population you can see how the values in the sample_mean column vary across samples/rows.

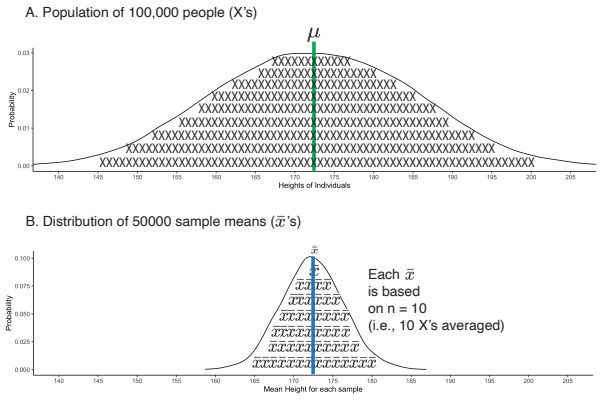


FIGURE 5.2: Sampling distribution of the mean. The population of individuals is presented at the top and filled with X's to remind you they are individuals. There are more individuals than X's. The sampling distribution of means is presented in the bottom part of the graph. The sampling distribution of means is filled with $\bar{x}$'s to remind you that it is sample means being graphed. There are more means than $\bar{x}$'s. The population mean (green line) and the mean of sample means (blue line) are in the same spot, indicating high accuracy (i.e., no bias).

Above, in Figure **??**, we present a graph comparing the distribution of peoples heights (i.e., the population) to the distribution of sample means based on those heights (i.e., the sampling distribution). The sample means plotted are the 50000 sample means, from the sample_mean column. Recall the population mean for heights is $\mu = 172.48$ cm. Notice that most of the sample means cluster around this value. Also notice that there is considerable variability about this value. Any given sample mean ($\bar{x}$) may differ substantially from the population mean ($\mu = 172.48$). This variability illustrates the challenges with learning something from a single study - particularly a study with a small sample size. Many of the sample means fall quite far from the population mean.

5.4.1 Assessing bias

Statisticians, recognizing the limitations of a single study, are not particularly concerned if a single sample mean deviates from the population mean. That said, statisticians are very concerned as to whether or not the results of a large number of studies are correct – on average. That is, does the average of many sample means correspond to the population mean? If, on average, the sample mean does corresponds to the population mean, it is accurate and we refer to it as an unbiased estimator. Visually, this appears to be the case. But in the code below we confirm it numerically.

```
many_samples %>%
  summarise(mean_of_sample_mean = mean(sample_mean)) %>%
  as.data.frame()
```

```
##   mean_of_sample_mean
## 1               172.5
```

We find that the average of the 50000 sample means is 172.47 which is very close to the population mean of 172.48. Note that when we did this, we used the same formula to calculate the sample mean (Formula (**??**)) as we did the population mean (Formula (**??**)), although the notations differed. The average of the sample means was not identical to the population mean but it was very close - it would have been exactly the same with many more samples (i.e., an infinite number of samples). Therefore, we conclude the sample mean provides an unbiased estimate of the population mean. In other words, it makes sense to use the sample mean as an estimate of the population mean. If we try to estimate the population mean with a sample mean we will, on average, be correct; although any given sample/study mean might be "wrong".

5.5 Estimating σ^2

We are interested in the sample variance (s^2) to the extent that it provides an estimate of the population variance (σ^2). We begin by reviewing population variance. The population variance is calculated using Formula (**??**):

$$\sigma^2 = \frac{\sum (X - \mu)^2}{N} \tag{5.4}$$

We can calculate the population variance for the height column of pop_data using the summarise() and var.pop() commands. The var.pop() command uses Formula (**??**). We see in the output that the population variance is 157.5 ($\sigma^2 = 157.5$).

```
pop_data %>%
  summarise(pop_var = var.pop(height)) %>%
  as.data.frame()
```

```
##   pop_var
## 1   157.5
```

5.5.1 Assessing bias

The formula for sample variance with an n in the denominator is, unfortunately, a biased estimator of population variance (formula below).

$$s^2 = \frac{\sum (x - \bar{x})^2}{n}$$

Estimates of the population variance are systematically too low when you use a sample variance formula with an n in the denominator. We can see that this is true by examining the many_samples data. In these data, the column sample_var_n contains the variance for the sample calculated with the above formula. Below we use code to obtain the average of the sample_var_n column over the 50000 samples. If this average equals the population variance of 157.5 then variance, using n in the denominator, is an unbiased estimator of the population variance.

```
many_samples %>%
  summarise(mean_of_var_n = mean(sample_var_n)) %>%
  as.data.frame()
```

```
##   mean_of_var_n
## 1         141.5
```

You can see the average of sample_var_n column (141.54) is much smaller than the population variance (157.5). That is, the average of the sample variances, using n in the denominator, was smaller than the population variance. Consequently, sample variance (using n in the denominator) provides a biased estimate of the population variance. If we try to estimate the population variance with sample variance (using n in the denominator) we will, on average, be wrong.

Fortunately, there is a sample-level formula that estimates the population variance without bias (see Hayes). An unbiased estimate of the population variance can be obtained if we calculate the sample variance but divide by $n-1$ instead of n. The unbiased estimate is calculated using Formula (**??**).

$$s^2 = \frac{\sum (x - \bar{x})^2}{n-1} \tag{5.5}$$

In the many_samples data, the column sample_var_n_1 was generated using Formula (**??**). We can evaluate the quality of Formula (**??**), using $n-1$, by averaging the values in the sample_var_n_1 column.

```
many_samples %>%
  summarise(mean_of_var_n_1 = mean(sample_var_n_1)) %>%
  as.data.frame()
```

```
##   mean_of_var_n_1
## 1           157.3
```

We see that the average of the 50000 values using $n-1$ in the denominator is 157.27 which is very close to the population variance of 157.46. These numbers would have been identical with an infinite number of samples. Consequently, when we use $n-1$ in the denominator we have an unbiased estimate of the population variance. If we try to estimate the population variance with a sample variance, using $n-1$ in the denominator, we will, on average, be right.

You may wonder at this point, when we use $n-1$ in the denominator of the sample variance, can we still think of it as the average of the squared differences from the mean? The short answer is yes. When you use $n-1$ in the denominator of the sample variance you are not calculating the variance for the group people in the sample. Rather, you are estimating the variance for the much larger group of people in the population. Consequently, it makes sense to think of sample variance, using $n-1$, as an estimate of the average of the squared differences/errors *in the population*. That is, it makes sense to think of sample variance, using $n-1$, as an estimate of the average of the

squared differences between each person in the population and the population mean.

5.6 Estimating σ

The population standard deviation is calculated using Formula (**??**) below.

$$\sigma = \sqrt{\frac{\sum (X - \mu)^2}{N}} \tag{5.6}$$

Due to the above findings for variance, we estimate the population standard deviation using Formula (**??**) below.

$$s = \sqrt{\frac{\sum (x - \bar{x})^2}{n - 1}} \tag{5.7}$$

5.7 Estimating δ

We are interested in the sample standardized mean difference (d) to the extent that it provides an estimate of the population standardized mean difference (δ). The population standardized mean difference is calculated using Formula (**??**) when we work for the assumption that the two population have the same variance / standard deviation:

$$\delta = \frac{\mu_1 - \mu_2}{\sigma} \tag{5.8}$$

We can calculate the population standardized mean difference for men and women once we have the respective population means and standard deviations. Recall the initial data mixed males and females. We begin by creating separate data sets for males and females:

```
male_population_heights <- pop_data %>%
  filter(sex == "male")

female_population_heights <- pop_data %>%
  filter(sex == "female")
```

Next, we calculate the mean and standard deviation of each population:

```
male_population_heights %>%
  summarise(mean = mean(height),
            sd = sd.pop(height))
```

```
## # A tibble: 1 x 2
##    mean    sd
##   <dbl> <dbl>
## 1  180.  10.1
```

```
female_population_heights %>%
  summarise(mean = mean(height),
            sd = sd.pop(height))
```

```
## # A tibble: 1 x 2
##    mean    sd
##   <dbl> <dbl>
## 1  165.  10.1
```

This reveals the population parameters are:

$$\begin{aligned} \mu_{female} &= 165 \\ \mu_{male} &= 180 \\ \sigma = \sigma_{female} = \sigma_{male} &= 10.1 \end{aligned}$$

Likewise, as calculated below, the population-level standardized mean difference (δ) is 1.49. We can see this population-level difference illustrated in Figure **??**.

$$\begin{aligned} \delta &= \frac{\mu_{male} - \mu_{female}}{\sigma} \\ &= \frac{180 - 165}{10.1} \\ &= \frac{15}{10.1} \\ &= 1.49 \end{aligned}$$

We typically need to estimate the population-level standardized mean difference from sample data because we rarely have access to data for an entire population. Many researchers estimate the population standardized mean difference from sample data using the Formula (**??**) below – when we assume the populations have equal variances. This value is known by many other names: d, Cohen's d, and Hedges' g. Notice that the sample-level formula, Formula

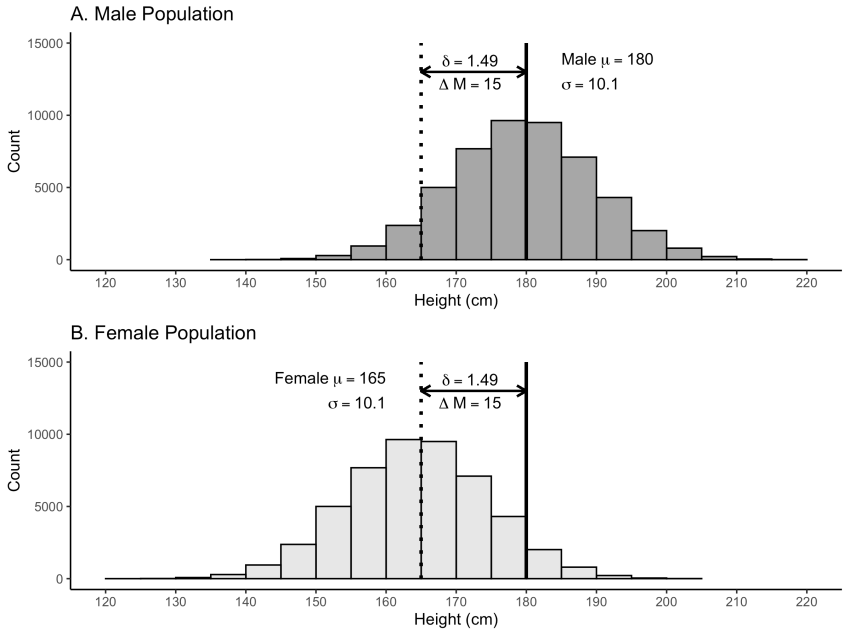


FIGURE 5.3: Illustration of the standardized mean difference of 1.49 for male and female heights. The solid black vertical line indicates the mean for males; whereas the dotted vertical line indicates the mean for females.

(**??**), below, is the same as the population-level formula, Formula (**??**), above, only the notation differs.

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s_{pooled}} \tag{5.9}$$

Unfortunately, Formula (**??**) provides a biased estimate of the population standardized mean difference for small sample sizes. That is, on average, Formula (**??**), provides d-values that overestimate the size of the population standardize mean difference (δ). Fortunately, we can obtain an unbiased estimate of the population-level standardized mean difference from sample data using Formula (**??**). This is one approach to calculating $d_{unbiased}$ – there are others.

$$d_{unbiased} = \frac{\bar{x}_1 - \bar{x}_2}{s_{pooled}} \times [1 - \frac{3}{4(n_1 + n_2) - 9}] \tag{5.10}$$

If we try to apply either d-value formula ((**??**) or (**??**)) to real data we quickly encounter a problem. We don't have the pooled standard deviation, s_{pooled}

5.7.1 Pooled standard deviation

When we calculated the population-level standardized mean difference we knew the population variances where the same. Consequently, there was only one standard deviation (i.e., only one variance). More specifically, the male and female populations both had a standard deviation but it was the same for both populations. The population-level formula for the standardized mean difference, Formula (**??**), has only one standard deviation in it. This is because calculation of the standardized mean difference explicitly depends on the fact that both populations have the same standard deviation.

Let's consider hypothetical sample data to make the situation clear. More specifically, we will examine the sample-level statistics below for males and females. Notice that we have two standard deviations – one for males and one for females. Moreover, these two sample-level standard deviations (using n-1) are not the same - they are different from each other. This initially seems problematic - calculation of standardized mean difference requires that population standard deviations are identical.

$$\bar{x}_{males} = 187.2$$
$$s^2_{males} = 92.2$$
$$s_{males} = 9.6$$

And females:

$$\bar{x}_{females} = 160.1$$
$$s^2_{females} = 66.8$$
$$s_{females} = 8.2$$

Fortunately, this is sample-level data and not population-level data. Sample-level standard deviations may differ even when the population-level standard deviations are the same. In fact, sample-level standard deviations are *likely* to differ from the population-level standard deviation due to sampling error. Consequently, we are *likely* to get two different sample-level standard deviations even if the population-level standard deviations are identical for males and females.

How do we resolve this situation of having two sample-level standard deviations? The first step is to switch to thinking in terms of variance rather than standard deviation. Due to the way the math works, life becomes very complicated, very quickly, if we continue to think in terms of standard deviations. Therefore, we reframe the problem into a variance problem. Variances are preferable to standard deviations because we can add and subtract variances - but not standard deviations.

We have a sample variance for males (92.2) and a sample variance for females

(66.8). We view each of these sample variances as an estimate of the respective population variances (see Figure **??**). That is, the male sample variance is an estimate of the male population variance. Likewise, the female sample variance is an estimate of the female population variance. However, we also **assume** that the population variances for males and females are the same. Consequently, the male sample variance and the female sample variance are both estimates of the same value (see Figure **??**). Because the two sample variances are estimates of the same population variance, we can (when the sample sizes are equal) calculate a new variance by averaging them together. This new variance, the average of the sample variances, provides us with a better estimate of the single population variance. The logic behind this approach is similar to averaging two measurements of the same distance to reduce error. We call this new variance pooled variance; and represent it with the symbol s^2_{pooled}.

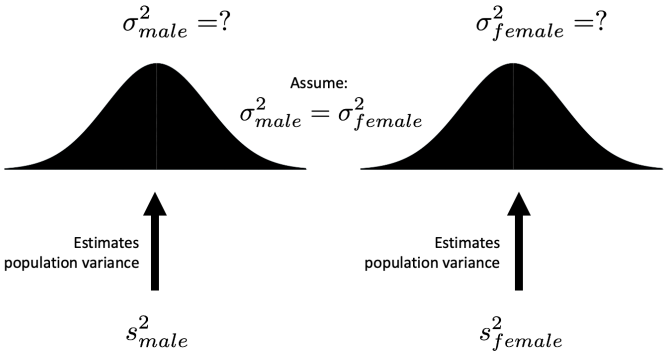


FIGURE 5.4: Estimating population variances with sample variances. The male sample variance (n-1) is an estimate of male population variance. Likewise, the female sample variance (n-1) is an estimate of the female population variance.

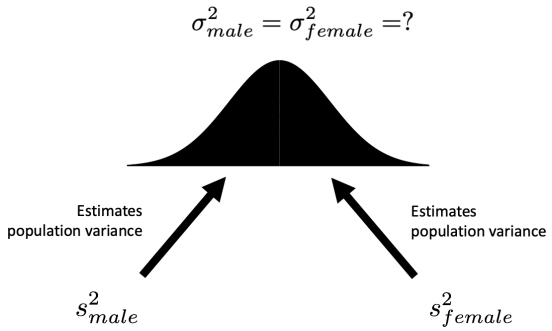


FIGURE 5.5: Two estimates of a single population variance. We assume the population variances are the same. Therefore, the male and female sample variances are both estimates of the same population variance.

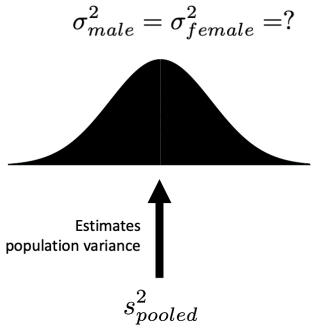


FIGURE 5.6: Pooled variance from the samples estimates population variance. The population variance is estimated by averaging two sample variances into a single estimate called pooled variance (s^2_{pooled}). When sample sizes are equal, the pooled variance is just the regular/simple average of the two sample variances (both using n-1 in the denominator). When the sample sizes are unequal (i.e., different numbers of males and females), however, we need to use a more sophisticated averaging formula to obtain the pooled variance.

When the sample sizes for males and females are the same (i.e., $n_{males} = n_{females}$) we can use the Formula (**??**) below to calculate the pooled variance.

$$s^2_{pooled} = \frac{s^2_1 + s^2_2}{2} \tag{5.11}$$

When the sample sizes for males and females are the different (i.e., $n_{males} \neq n_{females}$) we can use the Formula (**??**) below to calculate the pooled variance. This formula can be used all of the time. We only show Formula (**??**), above, to make it clear that Formula (**??**) below is basically just averaging the variances in a way that takes sample size into account.

$$s^2_{pooled} = \frac{(n_1 - 1)s^2_1 + (n_2 - 1)s^2_2}{n_1 + n_2 - 2} \tag{5.12}$$

We get the single standard deviation, s_{pooled}, by taking the square root of the variance, s^2_{pooled}.

$$s_{pooled} = \sqrt{s^2_{pooled}}$$

We apply the pooled standard variance, Formula (**??**), to the sample data:

$$\begin{aligned} s^2_{pooled} &= \frac{(n_{male} - 1)s^2_{male} + (n_{female} - 1)s^2_{female}}{n_{male} + n_{female} - 2} \\ &= \frac{(10 - 1)92.2 + (10 - 1)66.8}{10 + 10 - 2} \\ &= 79.5 \end{aligned}$$

Then we obtain the pooled standard deviation, below, for the standardized mean difference formula.

$$\begin{aligned} s_{pooled} &= \sqrt{79.5} \\ &= 8.9 \end{aligned}$$

5.7.2 Calculating d

Now that we have the pooled standard deviation, s_{pooled}, we can calculate the standardized mean difference. We do so below using unbiased formula, Formula (**??**) below.

$$\begin{aligned} d_{unbiased} &= d \times [1 - \frac{3}{4(n_{males} + n_{females}) - 9}] \\ &= \frac{\bar{x}_{males} - \bar{x}_{females}}{s_{pooled}} \times [1 - \frac{3}{4(n_{males} + n_{females}) - 9}] \\ &= \frac{187.2 - 160.1}{8.9} \times [1 - \frac{3}{4(10 + 10) - 9}] \\ &= 3.0 \times 0.96 \\ &= 2.9 \end{aligned}$$

5.7.3 Assessing bias

Sample-level $d_{unbiased}$-values, calculated above, often differ from the population-level standardized mean difference (i.e., δ) due to sampling error. We can confirm that $d_{unbiased}$-values are actually unbiased with a simulation. That is, we can confirm that the average of many $d_{unbiased}$-values equals the population standardized mean difference (i.e., δ) using a simulation.

First, we obtain the heights from the male and female populations and place them into male_heights and female_heights, respectively.

```
male_heights <- male_population_heights %>%
  pull(height)

female_heights <- female_population_heights %>%
  pull(height)
```

Next, we obtain a large number of samples from each population and place them in many_samples.

```
many_samples<-  get_d_samples_from_population_data(pop1 = male_heights,
                                    pop2 = female_heights,
                                    cell.n = 10,
                                    number.of.samples = 50000)
```

We can examine the contents of many_samples using the print() command. Each row of many_samples represents a single study. Each study has two samples: 10 males and 10 females. For both males and females we calculate the mean and variance. As well, we calculate the d and $d_{unbiased}$ for each row. If you examine the first row carefully you see that the data in this row corresponds to the hand calculation example.

```
print(many_samples)
```

```
## # A tibble: 50,000 x 7
##    n_per_cell mean1 var1_n_1 mean2 var2_n_1     d d_unbiased
##         <dbl> <dbl>    <dbl> <dbl>    <dbl> <dbl>      <dbl>
##  1         10  187.     92.2  160.     66.8  3.04       2.91
##  2         10  183     107.   166      77.8  1.77       1.69
##  3         10  181      39.8  169.    144.   1.27       1.22
##  4         10  175.     80.3  162.     89.4  1.4        1.34
##  5         10  184.     91.0  164.     28.9  2.65       2.54
##  6         10  182.     96.4  166.    106.   1.61       1.54
##  7         10  177.     97.2  167     162.   0.84       0.81
##  8         10  188     212.   161.    205.   1.89       1.81
##  9         10  185.     54.7  166.     91.5  2.28       2.18
## 10         10  179.     69.3  162.     55.4  2.2        2.11
## # ... with 49,990 more rows
```

Recall the population-level standardized mean difference, δ, was 1.49. We can see the extent to which the average of the sample-level d and $d_{unbiased}$ values compare to this population-level value.

```
many_samples %>%
  summarise(mean_d = mean(d),
            mean_d_unbiased = mean(d_unbiased))
```

```
## # A tibble: 1 x 2
##   mean_d mean_d_unbiased
##    <dbl>           <dbl>
## 1   1.56            1.49
```

You can see that the mean of the sample-level d values is 1.56 which is higher than the population-level standardized mean difference ($\delta = 1.49$). In contrast,

you can see that the mean of the sample-level $d_{unbiased}$ values is 1.49 which corresponds to the population-level standardized mean difference ($\delta = 1.49$).

5.7.4 Illustrating variability

An inspection of the first few rows of the many_samples data, above, illustrates that many of the $d_{unbiased}$ values differed from the population-level standardized mean difference of $\delta = 1.49$. We can see the variability in sample-level $d_{unbiased}$ values in the histogram below.

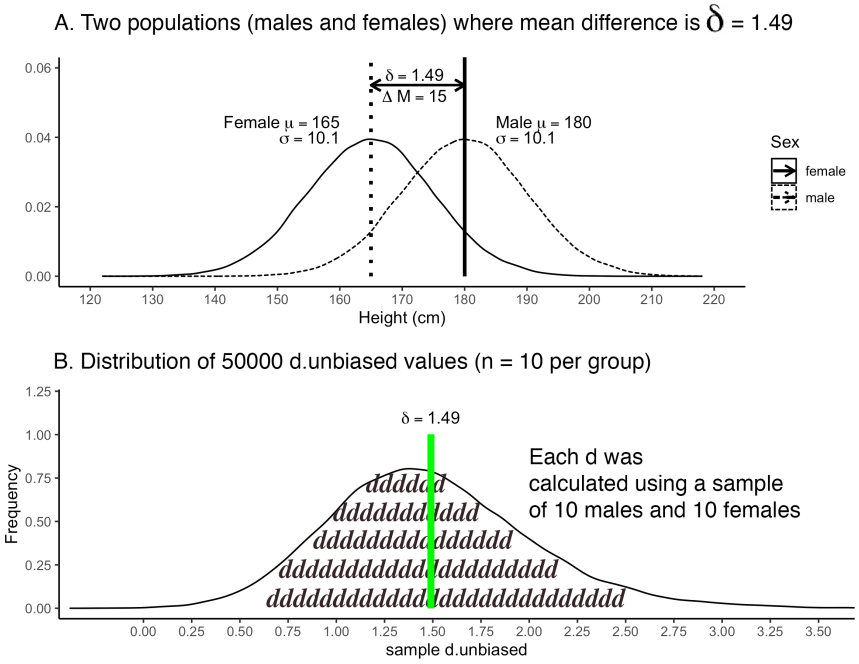


FIGURE 5.7: Histogram of $d_{unbiased}$ when $\delta = 1.49$

We can calculate the full range of sample-level $d_{unbiased}$ values with the commands below:

```
many_samples %>%
  summarise(d_min = min(d_unbiased),
            d_max = max(d_unbiased))
```

```
## # A tibble: 1 x 2
##   d_min d_max
##   <dbl> <dbl>
```

```
## 1 -0.38  5.15
```

We see from the output that $d_{unbiased}$ values were as small as -0.38 and as large as 5.15. All of these values are estimates of the population-level standardized mean difference of $\delta = 1.49$. You can see that many of the sample-level estimates differed considerably from the population-level value. The negative d-value (i.e., the minimum) is a case where the study (i.e, sample-level result) would have found that women are taller than men - a reversal of what is actually true at the population-level. This range of results illustrates the extent to which the findings for a single sample/study may deviate from the underlying truth for the entire population. We see that when the sample size is small (n = 10 per group) that the study results are likely to differ extraordinarily from what is true at the population level. This suggests that the "old school" suggestion of 10 participants per group when conducting a study leads to findings with little informational value.

5.8 Estimating ρ

The population-level correlation, ρ, is estimated by the sample-level correlation, r. The value for r can be calculated using Formula (**??**) below.

$$r = \frac{\Sigma(x - \bar{x})(y - \bar{y})}{\sqrt{\Sigma(x - \bar{x})^2 \Sigma(y - \bar{y})^2}} \tag{5.13}$$

Sample-level correlations, r, often differ from the population-level correlation (ρ) due to sampling error. We can confirm that sample correlations are not substantially biased with a simulation. That is, we can confirm that the average of many sample correlations (r) roughly equals the population correlation (ρ) using a simulation. We say "roughly equal" because r is technically a biased estimator of ρ but the bias is sufficiently small that it can be ignored (**?**).

We have the height and weight for 300000 people that comprise our population (fictious data). This data can be loaded with the command below.

```
library(tidyverse)
pop_data <- read_csv(file = "data_cor_pop.csv")
```

```
## Parsed with column specification:
## cols(
##   weight = col_double(),
##   height = col_double()
```

```
## )
```

The print() command reveals there are 300000 rows and two columns (weight and height). Each row represents a different person in the population.

```
print(pop_data)
```

```
## # A tibble: 300,000 x 2
##    weight height
##     <dbl>  <dbl>
##  1   143.   163.
##  2   156.   171.
##  3   144.   160.
##  4   152    190.
##  5   155.   173.
##  6   147.   160.
##  7   145.   177.
##  8   153.   179.
##  9   151.   177.
## 10   146.   167.
## # ... with 299,990 more rows
```

We can obtain the population correlation, ρ, by correlating the weight and height columns:

```
cor(pop_data)
```

```
##        weight height
## weight    1.0    0.5
## height    0.5    1.0
```

We see from this matrix that the population correlation ($N = 300000$) between weight and height is $\rho = .50$ (with rounding), see Figure **??**A.

To examine the extent to which this population correlation, $\rho = .50$, is estimated by the sample statistic, r, we need to take a large number of samples. Therefore, we take 50000 samples (each $n = 75$) and calculate the correlation for each:

```
set.seed(1)
many_samples <- get_r_samples_from_population_data(data = pop_data,
                                                   n = 75,
                                                   number.of.samples = 50000)
```

We can examine the first few rows of many_samples using the print() command. There are 50000 rows and each row represent a different sample of 75

people. The correlation between height and weight for each sample is presented in the r column.

```
print(many_samples)
```

```
## # A tibble: 50,000 x 3
##    sample.number     n     r
##            <int> <dbl> <dbl>
##  1             1    75 0.38
##  2             2    75 0.5
##  3             3    75 0.56
##  4             4    75 0.52
##  5             5    75 0.49
##  6             6    75 0.37
##  7             7    75 0.570
##  8             8    75 0.4
##  9             9    75 0.59
## 10            10    75 0.5
## # ... with 49,990 more rows
```

If you examine the first row carefully you see a sample correlation of $r = .38$ based on $n = 75$. This sample correlation is illustrated in Figure **??**B and it is just one of the 50000 sample correlations. The distribution of the 50000 sample correlations is illustrated in in Figure **??**C.

Even though the population correlation is $\rho = .50$ there is considerable variability in the sample correlations, r. Each sample correlation is based on a subset of the population data (i.e., 75 of the 300000 rows). Consequently, the sample correlations differ from the population correlation due to sampling error. The differences among the sample correlations can be quite large. Indeed, as the code below reveals, some sample correlation were as low as $r = .06$ and as high as $r = .77$ – even though the population correlation was $\rho = .50$.

```
many_samples %>%
  summarise(min_r = min(r),
            max_r = max(r))
```

```
## # A tibble: 1 x 2
##   min_r max_r
##   <dbl> <dbl>
## 1  0.06  0.77
```

5.8.1 Assessing bias

Even though the sample correlations usually differed from the population correlation we are not concerned. We recognize that there is little to be learned from a single study. We are more concerned as to whether the average of a large number of sample correlations is correct. We assess this with the code below.

```
many_samples %>%
  summarise(mean_r = mean(r))
```

```
## # A tibble: 1 x 1
##   mean_r
##    <dbl>
## 1  0.497
```

You can see that the mean of the sample-level correlations is $\bar{r} = .497$ which is very close to the population-level correlation $\rho = .50$ (.4999951 without rounding). Consequently, for practical purposes, we don't worry about the sample correlation being a biased estimator of the population correlation. On average, sample correlation are correct - even though any single sample correlation is likely incorrect due to sampling error (i.e., the fact it is based on a small subset of the population).

5.9 Overview

In this chapter we have illustrated how population parameters can be estimated by sample statistics; these are summarized below:

	Parameter	Estimated by this statistic
Mean	$\mu = \frac{\sum X}{N}$	$\bar{x} = \frac{\sum x}{n}$
Variance	$\sigma^2 = \frac{\sum (X-\mu)^2}{N}$	$s^2 = \frac{\sum (x-\bar{x})^2}{n-1}$
		$s^2_{pooled} = \frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{n_1+n_2-2}$
Standard deviation	$\sigma = \sqrt{\frac{\sum (X-\mu)^2}{N}}$	$s = \sqrt{\frac{\sum (x-\bar{x})^2}{n-1}}$

	Parameter	Estimated by this statistic
		$s_{pooled} = \sqrt{\frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{n_1+n_2-2}}$
Cohen's d or SMD	$\delta = \frac{\mu_1-\mu_2}{\sigma}$	$d = \frac{\bar{x}_1-\bar{x}_2}{s_{pooled}}$
		$d_{unbiased} = \frac{\bar{x}_1-\bar{x}_2}{s_{pooled}} \times [1-\frac{3}{4(n_1+n_2)-9}]$
Correlation	$\rho = \frac{\Sigma(X-\mu_X)(Y-\mu_Y)}{\sqrt{\Sigma(X-\mu_X)^2\Sigma(Y-\mu_Y)^2}}$	$r = \frac{\Sigma(x-\bar{x})(y-\bar{y})}{\sqrt{\Sigma(x-\bar{x})^2\Sigma(y-\bar{y})^2}}$

5.10 Meta-analysis

It may seem odd that we used so many simulations to investigate the properties of statistics. Surely, researchers don't do that "in the real world". In fact, researchers who are aware of the enormous impact of sampling error know that single studies have little informational value. They recognize that any single study has a high probability of being misleading. Consequently, these individuals survey the literature and find all the studies on a single topic (possibly thousands of studies). An average of the results of all of the thousands of studies can then be calculated and reported. This process is referred to as conducting a meta-analysis; and it perfectly corresponds to the process we used in the simulations. A meta-analysis finds "the truth" of what is happening at the population level by averaging all of the studies on that topic.

5.11 A joke

Now that you understand the logic for assessing bias, we present an old statistics joke.

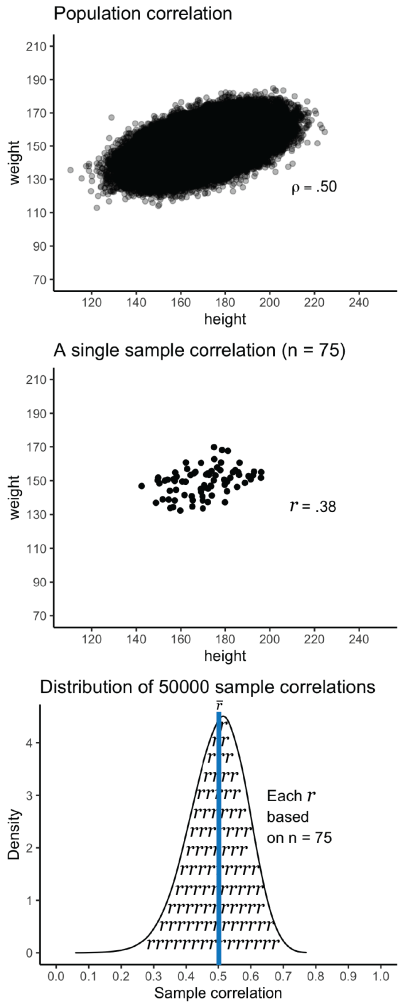


FIGURE 5.8: Correlation sampling distribution

"A physicist, a chemist, and a statistician go hunting. The physicist shoots at a deer and misses by 2 meters to the left. The chemist shoots and misses by 2 meters to the right. The statistician immediately yells"We got it!"

5.12 Key Points

1. Samples are of interest because they help us estimate attributes of a population.
2. Sample *statistics* estimate population *parameters*.
3. Due to the fact that sample statistics are based on a random subset of the population (i.e., a sample) they often differ substantially from the population parameter. This illustrates that informational value of a single study is typically quite low.
4. A statistic is unbiased if the average of the sample statistics, over many thousand of samples, equals the population parameter.
5. To avoid bias, sometimes the formula for a sample statistics differs from the formula for the population parameter.
6. Meta-analyses are used in the "real world" the way we used simulations in this chapter.

6

Sampling Precision

6.1 Overview

In the previous chapter we focused on examining the extent to which we could estimate a population parameter from a statistic in an unbiased manner. That is, we focused on the fact that, due to sampling error, a sample estimate (i.e., a statistic) of a population parameter will likely be wrong. Sample statistics may overestimate, or underestimate, a population parameter - often substantially. Consequently, we learned to think about the accuracy of sample statistics. More specifically, we learned to think about accuracy not in terms of a single study, but rather in terms of whether sample statistics correctly estimate the population parameter on average over many studies (as illustrated in Figure **??**). When the average of many sample statistics (e.g., sample means) equals the population parameter (e.g., population mean) there is no bias and the sample statistic can be considered accurate (on average).

In this chapter, we assume there is no bias and that sample means are, on average, accurate. In doing so, we focus now on the precision of sample means as estimates of the population mean, see Figure **??**. That is, assuming no bias, when you conduct a single study you know the sample mean will differ from the population mean due to sampling error. You might wonder, how much does my sample mean differ from the population mean? That's the question we address in this chapter. When we conduct a single study we can't know how much that particular sample mean differs from the population mean, but we can estimate the extent to which sample means differ from the population mean in general. We focus on two indices of precision – variance of sample mean and standard error of sample means. By the end of the chapter you will understand how to a) conceptualize these indices with respect to a large number of samples/studies and b) estimate them for a single study.

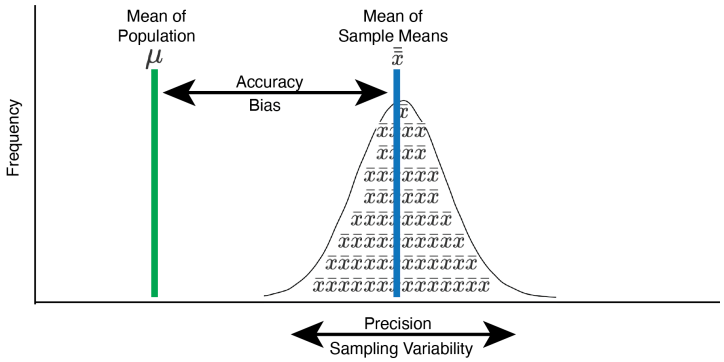


FIGURE 6.1: Sampling accuracy and precision

6.2 Population / Individuals

We use a population of heights to learn about random sampling and the precision of sample estimates. We can obtain that population with the code below:

```
library(tidyverse)
library(learnSampling)

pop_data <- get_height_population()
```

The glimpse() command can be used to confirm that the population contains 100,000 people.

```
glimpse(pop_data)
```

```
## Rows: 100,000
## Columns: 3
## $ id     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1...
## $ sex    <chr> "male", "female", "female", "male", "mal...
## $ height <dbl> 177, 150, 171, 157, 169, 187, 163, 173, ...
```

We can use the head() command to see the first 10 rows of the 100,00 rows. We see that each row in pop_data represents a single person. There is a column called height that contains the height for everyone in the population.

```
head(pop_data, 10)
```

```
## # A tibble: 10 x 3
##       id sex    height
##    <int> <chr>   <dbl>
##  1     1 male      177
##  2     2 female    150
##  3     3 female    171
##  4     4 male      157
##  5     5 male      169
##  6     6 male      187
##  7     7 female    163
##  8     8 male      173
##  9     9 female    172
## 10    10 male      193
```

We can calculate the descriptive parameters for the population with the code below:

```
pop_data %>%
  summarise(pop_mean = mean(height),
            pop_var = var.pop(height),
            pop_sd = sd.pop(height)) %>%
  as.data.frame()
```

```
##   pop_mean pop_var pop_sd
## 1    172.5   157.5  12.55
```

We see the mean of the population of heights is 172.5 ($\mu = 172.5$) and the variance is 157.5 ($\sigma^2 = 157.5$). Correspondingly, the standard deviation of the population of heights is 12.5 ($\sigma = 12.5$); simply the square root of the variance.

$$\mu = 172.5$$
$$\sigma^2 = 157.5$$
$$\sigma = 12.5$$

6.3 Sampling distribution

A sampling distribution is composed of an infinite number of samples. For pedagogical purposes, we will use a sampling distribution of just 50000 samples. This large number of samples will lead us to roughly the same conclusions

as using an infinite, but impractical, number of samples. We obtain 50000 samples with code below:

```
many_samples <- get_M_samples(pop.data = pop_data,
                              pop.column.name = height,
                              n = 50,
                              number.of.samples = 50000)
```

We can use the head() command to see the first few rows of many_samples. Every row represents a sample of 10 people. That is, each row represents a study with 10 participants for which we measured their heights. For each row/study we have the sample mean and variance (using n-1) for the 10 heights.

```
head(many_samples)
```

```
## # A tibble: 6 x 5
##   study     n sample_mean sample_var_n sample_var_n_1
##   <int> <dbl>       <dbl>        <dbl>          <dbl>
## 1     1    10        174.         27.8           30.9
## 2     2    10        162.         83.4           92.6
## 3     3    10        173.        139.           155.
## 4     4    10        180         322.           358.
## 5     5    10        173.        161.           179.
## 6     6    10        172         160.           177.
```

We will study the precision of sample means as estimates of the population mean using the values in the sample_mean column. Each of the 50000 values in the sample_mean column is a sample mean ($\bar{x}$), based on $n = 10$. Each sample mean provides an estimate of the population mean ($\mu = 172.5$). Sample means often overestimate, or underestimate, the population mean. We can see this using the code below which reveals that the smallest sample mean is 155.6 ($\bar{x} = 155.6$) and the largest sample mean is 188.8 ($\bar{x} = 188.8$). Both are estimates of the population mean ($\mu = 172.5$) but differ from that value due to sampling error (i.e., the fact that sample means are based on a small subset of the population). You can see that the conclusions of any single study may differ rather substantially from what is true for the overall population.

```
many_samples %>%
   summarise(min_mean = min(sample_mean),
             max_mean = max(sample_mean))
```

```
## # A tibble: 1 x 2
##   min_mean max_mean
##      <dbl>    <dbl>
```

```
## 1     156.     189.
```

The 50000 means in the sample_mean column form a **sampling distribution**; specifically, the sampling distribution of the mean. It's critical to distinguish between a) the distribution of the sample means and b) the distribution of population heights. Examine Figure **??** below. The top part of this figure illustrates the variability in the heights for the 100,000 people (represented by X's). There are more people than X's; the X's are a reminder it is a distribution of people's heights. The bottom part of this figure illustrates the variability in sample means - each based on 10 people. Each sample mean is an estimate of the mean height of the population (172.5 cm). There are more sample means than $\bar{x}$'s; the $\bar{x}$'s are a reminder it is a distribution of sample means. When we describe how precisely sample means estimate the mean height for the population we are referring to the width of this distribution of means.

6.4 Precision indices

The most common way of referring to the width of this distribution of sample means is with the term standard error. Standard error is simply the standard deviation of the sample means. We calculate the standard error of the mean using Formula (**??**) below. We use k in the formula to refer to the number of sample means in the distribution.

$$\sigma_{\bar{x}} = \sqrt{\frac{\sum (\bar{x} - \bar{\bar{x}})^2}{k}} \tag{6.1}$$

Notice that the formula for standard error is actually just the formula for standard deviation but with notation adapted to reflect the fact we are examining sample means (i.e., $\bar{x}$'s) instead of people (i.e., X's).

Often we will not talk about standard error but rather the variance of the distribution of sample means. When we do so, we are talking about the same thing - the variability in sample means - but using different units. You can see that the formula for the variance of sample means, see Formula (**??**) below, is just a squared version of the standard error equation, Formula (**??**) above.

$$\sigma^2_{\bar{x}} = \frac{\sum (\bar{x} - \bar{\bar{x}})^2}{k} \tag{6.2}$$

How do we interpret variance of sample means? Assuming the sample mean provides an unbiased estimate of the population mean (i.e., $\bar{\bar{x}} = \mu$), then the

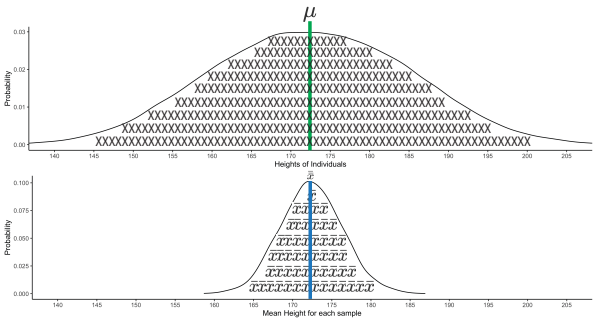


FIGURE 6.2: Sampling distribution of the mean. The population of individual height's is presented at the top and filled with X's to remind you they are individuals. There are more individuals than X's. The sampling distribution of means is presented in the bottom part of the graph. The sampling distribution of means is filled with $\bar{x}$'s to remind you that sample means are being graphed. There are more means than $\bar{x}$'s. The population mean (green line) and the mean of sample means (blue line) are in the same spot, indicating high accuracy (i.e., no bias).

variance of the sample means is simply the average of the squared differences between sample means and the population mean. Standard error is simply the square root of the variance of sample means. Both are different ways of describing the same thing - the precision with which sample means estimate the population mean.

If the variance of sample means is large, it indicates sample means differ considerably (on average) from the population mean. If the variance of sample means is small, it indicates sample means differ only slightly (on average) from the population mean. If the variance of sample means is zero, all the sample means are exactly the same as the population mean.

We can calculate the variance of sample means below:

```
many_samples %>%
   summarise(var_of_means = var.pop(sample_mean)) %>%
   as.data.frame()
```

```
##   var_of_means
## 1        15.73
```

The standard error of sample mean is just the square root of this value:

```
many_samples %>%
   summarise(sd_of_means = sqrt(var.pop(sample_mean))) %>%
   as.data.frame()
```

```
##   sd_of_means
## 1       3.967
```

Thus, we can indicate the precision with which sample means ($\bar{x}$) estimate the population mean (μ) using the notation below. We use $\sigma^2_{\bar{x}}$ to refer to the variance of sample means and $\sigma_{\bar{x}}$ to refer to the standard error of sample means. Again notice in the notation that for both of these symbols there is a subscript with an $\bar{x}$ to remind us we are talking about the variability of sample means (and not people).

$$\sigma^2_{\bar{x}} = 15.73$$
$$\sigma_{\bar{x}} = 3.97$$

6.5 A short cut

In the previous section we used a computer simulation to calculate the variance of sample means. This involved several steps:

1. Obtaining the entire population
2. Creating 50000 samples (each n = 10) from the population
3. Calculating the mean for each of the 50000 samples
4. Calculating the variance of the 50000 sample means as per Formula (**??**)

All those steps are a lot of work - and a lot of computer time. Interestingly, we don't need to go to all that work to determine the variance of sample means. There is a quicker way to obtain the variance of sample means. That is, there is a quicker way to determine the precision with which sample means estimate the population mean when we use a sample size of $n = 10$.

Statisticians have created a short-cut formula for obtaining the variance of sample means. The equation by itself, see Formula (**??**) below, is extraordinarily simple to use. The formula, however, is not easy to understand. Over many years of teaching, we've seen students struggle to understand why the formula is structured in the way that it is - looking for a straightforward logic as to why the short-cut formula works. We encourage you to avoid going down that road. Understanding Formula (**??**) below is beyond the scope of this chapter and, likely, your mathematical background at this point. It was derived via a complex mathematical proof discussed in **?** if you want more information. Simply accept that there is a short-cut formula and do not try to understand the logic of the formula.

Variance sample means:

$$\sigma_{\bar{x}}^2 = \frac{\sigma^2}{n} \tag{6.3}$$

You can see that with this formula the variance of sample means, Formula (**??**), is simply the variance of heights of the individuals in the population divided by the sample size. Oddly, this short cut works and provides the variance of sample means. Notice that we don't need to have the entire population, create thousands of samples, or go through any of the calculation work. All we need to know is the variance of the population and the sample size. In the context of the current example we can compute the variance of the sample means using this information. Recall the variance of people's heights in the population is $\sigma^2 = 157.5$ and our sample size is $n = 10$.

$$\begin{aligned}\sigma^2_{\bar{x}} &= \frac{\sigma^2}{n} \\ &= \frac{157.5}{10} \\ &= \frac{157.5}{10} \\ &= 15.75\end{aligned}$$

We see that the variance of the sample means from this short-cut approach is 15.75 (i.e., $\sigma^2_{\bar{x}} = 15.75$). Likewise, the variance of the sample means from our simulation was 15.73; incredibly close. If we had used substantially more than 50000 samples in the simulation the 15.73 value would have been 15.75 – making the two results identical. Thus, the short-cut formula gives us the variance of sample means without having to do all the simulation work.

Of course, the variance of sample means is only one of the two ways we can describe the precision of sample estimates of the population mean. Many people prefer to use standard error (i.e. standard deviation of sample means) instead. The calculation for standard error is below:

$$\begin{aligned}\sigma_{\bar{x}} &= \sqrt{\frac{\sigma^2}{n}} \\ &= \sqrt{\frac{157.5}{10}} \\ &= \sqrt{\frac{157.5}{10}} \\ &= \sqrt{15.75} \\ &= 3.97\end{aligned}$$

6.6 Estimates of precision

As wonderful as it is to have a short-cut formula for obtaining the variance of sample means (or standard error) there is a catch. The formula requires knowledge of the population variance (or standard deviation). In a research scenario we don't know the mean or standard deviation of the population - that's why we're conducting research. Consequently, we can't directly calculate the variance of sample means (or standard error) using Formula (**??**) above.

Fortunately, if we have only a single study, we can use the sample variance (with n-1 in the denominator) to estimate the population variance. This es-

timate of the population variance may be higher or lower than the actual population variance - but it will, on average, be correct across many studies. With this estimate of the population variance in hand we can calculate an estimate of the variance of sample means (i.e, the precision of sample means) using Formula (**??**) below.

Estimated variance sample means:

$$s^2_{\bar{x}} = \frac{s^2}{n} \tag{6.4}$$

Estimated standard error for the mean:

$$s_{\bar{x}} = \sqrt{\frac{s^2}{n}} = \frac{s}{\sqrt{n}} \tag{6.5}$$

6.6.1 A worked example

Consider a worked example below. Imagine a scenario where we do not know anything about the population of heights. Therefore we conduct a study by taking a random sample of 10 people to estimate the average height of people in the population. The sample mean is 174.3cm and the sample variance is 30.90. Therefore, our best guess of the unknown population mean is 174.3cm. Because we used a sample (a small subset of the population) we know that 174.3cm is unlikely to be the actual mean height for the population. The mean height for the population might be higher or lower than 174.3cm. Nonetheless, at this point our best guess is that the mean of the population is 174.3cm; but we recognize that this is just an estimate of the population mean and is likely off by some amount.

Wouldn't it be great to know the extent to which the sample mean in our study might differ from the population mean? As budding statisticians, we know we can't know that for this particular sample/study, but we can try to figure out how much sample means differ from the population mean on average (when using $n = 10$) to help us understand the precision of our current sample/study estimate (174.3cm) of the population mean. This precision information is exactly what is conveyed by the variance of sample means, $\sigma^2_{\bar{x}}$, or standard error of sample means, $\sigma_{\bar{x}}$. We can't obtain these precision indices but we can estimate them (via $s^2_{\bar{x}}$ and $s_{\bar{x}}$).

We want to know the precision with which sample means estimate the population mean. Said another way, we want to know the variance of sample means (or standard error of sample means) but we only have a single study. Therefore, we have to rely on the short-cut formula for determining the variability in sample means. Unfortunately, the short-cut formula for the variance of

sample means formula requires the population variance which we don't know. We do, however, have an estimate of the population variance from our sample. The sample variance is 30.9cm^2. Therefore, our best guess of the unknown population variance is 30.9cm^2. Because we used a sample (a small subset of the population) we know that 30.9cm^2 is likely to be different than the actual variance for the population. The variance for the population might be higher or lower than 30.9cm^2. So even though our estimate of the population variance may be "off" in this single sample we know it will be accurate in the long run averaging over many studies (see previous chapter).

Nonetheless, at this point our best guess is that the variance of the population is 30.9cm^2. We use this information to estimate the variance of sample means:

$$\begin{aligned} s^2_{\bar{x}} &= \frac{s^2}{n} \\ &= \frac{30.90}{10} \\ &= 3.09 \end{aligned}$$

Or alternatively, estimate standard error:

$$\begin{aligned} s_{\bar{x}} &= \sqrt{\frac{s^2}{n}} \\ &= \sqrt{\frac{30.90}{10}} \\ &= \sqrt{3.09} \\ &= 1.76 \end{aligned}$$

At this point we don't know the precision of sample means when $n = 10$; that is, we don't know $\sigma^2_{\bar{x}}$ or $\sigma_{\bar{x}}$. We do have an estimate of the precision of sample means (i.e., with via $s^2_{\bar{x}}$ and $s_{\bar{x}}$). Whenever you calculate the variance of sample means, or standard error, based on sample data it is ALWAYS just an estimate of the actual variance of sample means or the actual standard error.

That fact that an estimate of the variance of sample means is the best you can do with sample data is evident when we calculate the estimate for the many samples. We do that with the R code below:

```
many_samples <- many_samples %>%
   mutate(est_se2 = sample_var_n_1/n)
```

The command above calculated the estimate variance of sample means

(est_se2 or $s^2_{\bar{x}}$) for each of the 50000 samples. We can see the first few samples with the head() command.

```
head(many_samples)
```

```
## # A tibble: 6 x 6
##   study     n sample_mean sample_var_n sample_var_n_1
##   <int> <dbl>       <dbl>        <dbl>          <dbl>
## 1     1    10        174.         27.8           30.9
## 2     2    10        162.         83.4           92.6
## 3     3    10        173.        139.           155.
## 4     4    10        180         322.           358.
## 5     5    10        173.        161.           179.
## 6     6    10        172         160.           177.
## # ... with 1 more variable: est_se2 <dbl>
```

Inspect the est_se2 column. Notice how each of these values is an $s^2_{\bar{x}}$ number that is an estimate of $\sigma^2_{\bar{x}} = 15.75$. The first estimate, $s^2_{\bar{x}} = 3.09$, is the one we did by hand above. This estimate of the variance of sample means ($s^2_{\bar{x}} = 3.09$) is much lower than the actual variance of sample means ($\sigma^2_{\bar{x}} = 15.75$). Notice how the other values in the est_se2 column tend to either overestimate, or underestimate, the variance of sample means. That is, the other studies also overestimate, or underestimate, the precision with which sample means ($n = 10$) estimate the population mean.

6.7 Bias of precision estimates

At this point, you might be a bit concerned about our ability to estimate the variance of sample means ($\sigma^2_{\bar{x}} = 15.75$). In the simulation above, we calculated an estimate of the variance of sample means ($s^2_{\bar{x}}$) and placed those values in the est_se2 column. We saw that the values in this column mostly differed from the actual variance of sample means (15.75). You might wonder if these estimates of the variance of sample means are themselves biased? The answer is no - they are not biased. Our estimate of the population variance (n-1 in the denominator) was not biased so there is no reason to suspect that the estimate of the variance of sample means is biased. But we can confirm this by averaging over the 50000 samples. If the average of the est_est2 column is 15.75 (or very close) than $s^2_{\bar{x}}$ is not a biased estimate of $\sigma^2_{\bar{x}} = 15.75$.

```
many_samples %>%
   summarise(avg.of.est_se2 = mean(est_se2))
```

```
## # A tibble: 1 x 1
##   avg.of.est_se2
##            <dbl>
## 1           15.7
```

We see that the average of the 50000 estimates of the variance of sample means ($s^2_{\bar{x}}$) is 15.73 which is very close to the actual value of $\sigma^2_{\bar{x}} = 15.75$. Any sample estimate of the variance of sample means will be off somewhat due to sampling error. But, on average, estimates of the variance of sample means will be unbiased/accurate.

6.8 Where are we?

So where are we? We've learned that sample means may overestimate, or underestimate, the population mean. Over many studies, on average, sample means will approximate the population mean. In other words, on average, sample means are accurate. As well, we've learned that the variability in estimates of the population mean is referred to as **precision**. We can index the precision of sample means, for a given sample size, using two indices: variance of sample means and standard error of sample means (i.e., standard deviation of sample means). Both of these formulas require knowledge of the population variance - something we never have.

When we conduct a single study, we get a sample mean. We cannot estimate how precise that sample mean is; we can however, estimate how precise sample means are in general – via the variance of sample means (or standard error). There is a formula for calculating the variance of sample means, Formula (**??**), but it requires knowledge of the (unknown) population variance. Consequently, we use an estimate of the population variance in these formulas rather than the actual population variance, see Formula (**??**). As a result, when we calculate variance of sample means (or standard error) using an estimate of the population variance (from our sample) we only obtain an estimates of the variance of sample means. Moreover, this estimate of the variance of sample means is itself influenced by sampling error - but accurate when averaged over many studies.

6.9 Precision for means: Causes

The formula for variance of the sampling distribution for the mean reveals that it is influenced only by the population variance and sample size. In a research scenario we don't have any control over the population variance (it is what it is), but we do have control over sample size. Increasing the sample size decreases the variability in sample means - as illustrated in the formulas below.

$$\sigma^2 = 157.5$$
$$\sigma^2_{\bar{x}} = \frac{\sigma^2}{n} = \frac{157.5}{10} = 15.75$$
$$\sigma^2_{\bar{x}} = \frac{\sigma^2}{n} = \frac{157.5}{50} = 3.15$$

We also show in Figure **??** the difference that increasing the sample size makes. The larger the sample size the less sample means vary due to sampling error. A larger sample size means a more precise estimate of the population parameter.

Additionally, a larger sample size makes the sample estimate (s^2) of the population variance (σ^2) more precise. Consequently, it makes the sample estimate ($s^2_{\bar{x}}$ or $s_{\bar{x}}$) of the variance of sample means ($\sigma^2_{\bar{x}}$ or $\sigma_{\bar{x}}$) more precise. Therefore, increasing the sample size increases the precision of a) sample means ($\bar{x}$) and b) precision estimates ($s^2_{\bar{x}}$ or $s_{\bar{x}}$).

The precision of sample means is influenced by:

- population variance/standard deviation (σ^2 / σ^2)
- sample size (%n%)

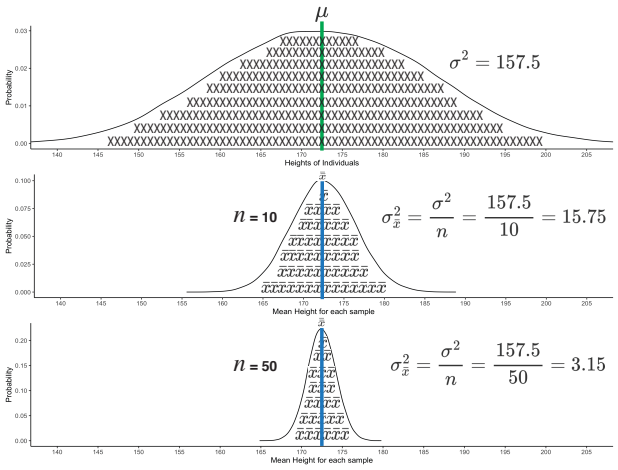


FIGURE 6.3: Two sampling distributions illustrating variability in sample means based on sample size. The top graph show the variance of individual's heights in the population. The graph is filled with X's to remind you it is a graph of individuals. There are more individuals than X's. The bottom two graphs show variability in sample means due to random sampling. Each of the lower two graphs is a sampling distribution for a given sample size. Each sampling distribution contains 50000 sample means. Each sample mean is the average of the heights of individuals in a sample. The sampling distributions are filled with $\bar{x}$'s to remind you that it is sample means being graphed. There are more means than $\bar{x}$'s. You can see that by using a larger sample size ($n = 50$, for 50000 samples) there is less variability in sample means than when using a smaller sample size ($n = 10$), for 50000 samples.

6.10 Precision for d-values: Causes

Whenever we use a statistic (e.g., $\bar{x}$) to estimate a population-level parameter (e.g., μ) the statistic will differ from the parameter due to sampling error. This is true of all statistics. Therefore, it is possible to generate a sampling distribution for all statistics. In this section will focus on the factors that influence the precision with which d-value estimate the population-level standardized mean difference (δ). More specifically, we focus on how the sample size and the size of the population-level effect influence width of the sampling distribution.

6.10.1 Sample size

Consider the case of estimating the standardized mean difference (i.e., δ) between the heights of males and females, illustrated in Figure **??**. The population-level height information is presented in Figure **??**A. The sampling distribution for sample-level $d_{unbiased}$ values when you use 10 people per group (10 males, 10 females) is presented in Figure **??**B. We could calculate the variance of $d_{unbiased}$ values or the standard error (i.e., standard deviation) of $d_{unbiased}$ values; just like we did for sample mean.

You can see in Figure **??**B that even though the population-level standardized mean difference is 1.49 ($\delta = 1.49$) there was considerably variability in sample-level $d_{unbiased}$ values. Indeed, inspecting the sample-level data reveals when the population difference is $\delta = 1.49$, the sample level differences range from $d_{unbiased}$ = -0.38 to $d_{unbiased} = 5.50$. That's a considerable range. When $d_{unbiased}$ = -0.38 you would conclude, in your study, that the mean female height is .38 standard deviations higher than than the mean male height. Yet a colleague, conducting exactly the same study, could obtain a sample that leads him to observe $d_{unbiased} = 5.50$. This would indicate to him that the mean male height is 5.50 standard deviations higher than the mean female height; an opposite conclusion. The results of both your study and the colleague's study differ considerably from the truth that $\delta = 1.49$ – which indicates the mean male height is 1.49 standard deviations higher than the mean female height.

We investigated the extent to which study-level $d_{unbiased}$ values differ from the population $\delta = 1.49$ when using 100 people per group for the comparison (i.e., 100 males, 100 females). You can see in Figure **??**C that in this scenario the precision of the sample estimates ($d_{unbiased}$) of the population parameter (δ) still vary considerably - but substantially less than they did when it was 10 per group. With a 100 people per group, the $d_{unbiased}$ values range from 0.91 to 2.21. For the lower end of this range, 0.91, a researcher would conclude

that the mean for males is .91 standard deviations higher than the mean for females. At the upper end of this range, 2.21, a researcher would conclude that the mean for males is 2.21 standard deviations higher than the mean for females. The variability in $d_{unbiased}$ values is still large - but at least the conclusions are a) all in the same direction, and b) would all be considered large (i.e, above 0.80) according to Cohen's standards.

These simulations illustrate a few important points. First, drawing any conclusion from a single study is problematic. The findings for a single study may differ substantially from the population parameter. Second, increasing sample size increases the precision with which a sample statistic estimates a population parameter. Third, drawing conclusions from small sample size studies (e.g., $n = 10$ per group) is extraordinarily problematic. Finally, we note that single studies (even with small sample size) are useful because they serve as data points for a future meta-analysis.

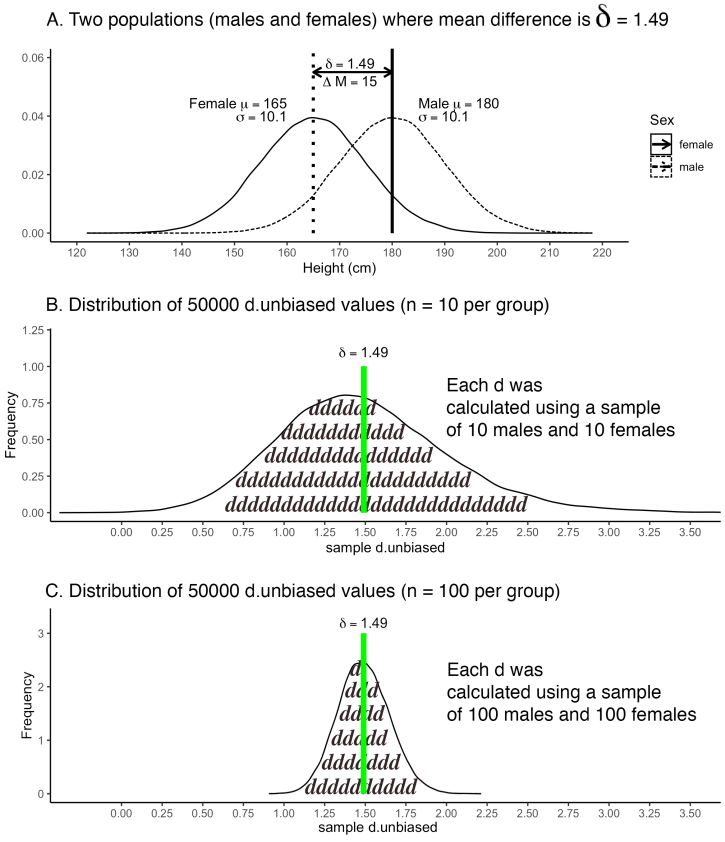


FIGURE 6.4: Sampling distribution for $d_{unbiased}$

6.10.2 Population effect size (δ)

The precision with which sample d-values estimate the population standardized mean difference is influenced by the magnitude of the population standardized mean difference. This situation differs sharply from that of sample means. The magnitude of the population mean does not influence the shape of the distribution of sample means. The distribution of sample means is normal - regardless of whether the population mean is high or low.

In contrast, the shape of the the sampling distribution of d-values changes depending on the magnitude of the population effect (i.e., δ). In Figure **??** we illustrate two scenarios. Both scenarios are based on a repeated measures design. For each scenario we created a population-level difference and then obtained 50000 sample d-values. The sample size, $n = 10$ for each d-value, was held constant across the two scenarios. In contrast, the population standardized mean difference was different across the two scenarios. In Figure **??**A the population standardized mean difference was $\delta = 0.50$. In contrast, in Figure **??**B the population standardized mean difference was $\delta = 2.00$. Compare the shape of the distributions of d-values in Figure **??**A and Figure **??**B. You can see how the shape changes when the population level effect size changes. Consequently, the size of the population standardized mean difference influences the precision with which sample d-values estimate the population level effect (δ).

Standard Error

There is a standard error for d-values (i.e., SE_d or $SE_{d_{unbiased}}$). It is simply the standard deviation of the thousands of d-values in the simulation. More generally, it is an index of much d-values vary from a population standardized mean difference (δ) due to sampling error for a given sample size. In geometric terms, the standard error is an index of the width of a sampling distribution. The standard error is larger in Figure **??**B than in Figure **??**C.

The precision of sample d-values is influenced by:

- population standardized mean difference (δ) - influences shape of the distribution
- sample size (n)

6.11 Precision for sample correlations (r): Causes

Sample correlations (r) are likely to differ from the population correlation (ρ) due to sampling error. The precision with which sample correlations estimate

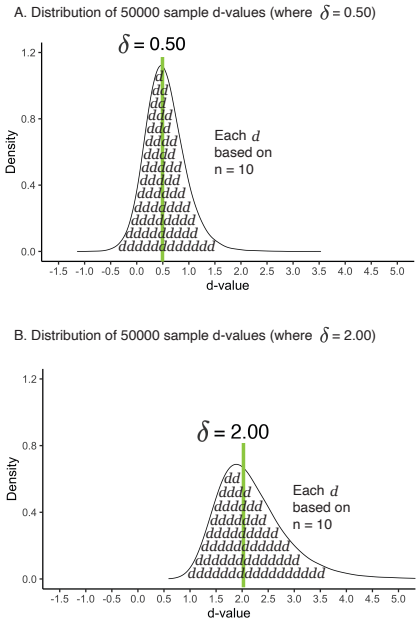


FIGURE 6.5: d-value skew graphs

the population correlation is influenced by sample size and the magnitude of the population correlation. These are reviewed below.

6.11.1 Sample size

Consider a situation where the population-level relation between two variables is $\rho = .30$. For example, when looking at 300000 people the correlation between weight and height is .30. Then consider two scenarios where we sample from the population. In Scenario 1, we use a sample size of 50 people. We take 50000 samples (each comprised of 50 people) and calculate for each sample the correlation (r). The distribution of these sample correlations for Scenario 1

are presented in Figure **??**A. In Scenario 2, we use a sample size of 500 people (ten times more per sample). We take 50000 samples (each comprised of 500 people) and calculate for each sample the correlation (r). The distribution of these sample correlations for Scenario 2 are presented in Figure **??**B. If you contrast these two graphs you see the sampling distribution is narrower in Scenario 2 where the sample size is larger, see Figure **??**B. This means that sampling error is less when the sample size is larger. In other words, a larger sample size results in more precise sample correlations.

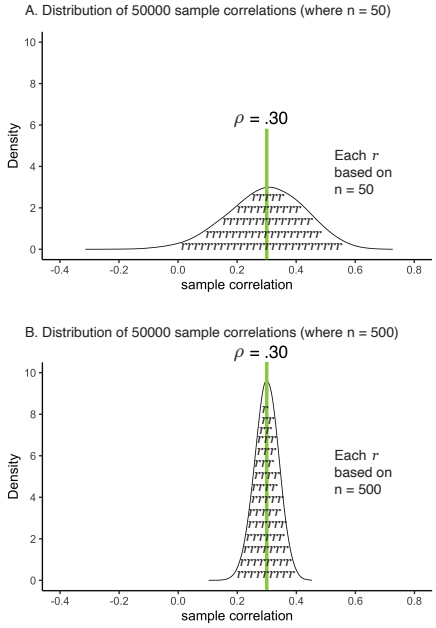


FIGURE 6.6: Correlation precision and sample size

6.11.2 Population effect size (ρ)

Consider a situation where we look at the correlation between height and weight in two different cities. Each city has 300000 people and we consider each city a population. In City 1 the population correlation is $\rho = .30$ whereas in City 2 the population correlation is $\rho = .70$. For City 1, we take 50000 samples (each comprised of 50 people) and calculate for each sample the correlation (r). The distribution of these sample correlations for City 1 are presented in Figure **??**A. We repeat the process and obtain 50000 sample correlations (each comprised of 50 people) for City 2; the distribution of these sample correlations is presented in Figure **??**B. If you contrast these two graphs you see the sampling distribution is narrower for City 2 where the population correlation is stronger, see Figure **??**B. This means that sampling error is less when the population correlation (ρ) is stronger. In other words, a larger population correlation results in more precise sample correlations. Importantly, also notice how the shape of the sampling distribution varies across the two effect-size scenarios (Figure **??** A vs B).

Standard Error

As with the other statistics reviewed, for sample correlations (r) you can calculate a standard error (SE_r). It is simply the standard deviation of the thousands of sample correlations (r's) in the simulation. More generally, it is an index of much sample correlations vary from a population correlation (ρ) due to sampling error for a given sample size. In geometric terms, the standard error is an index of the width of a sampling distribution. The standard error is larger in Figure **??**A than in Figure **??**B.

The precision of sample correlations (r) is influenced by:

- population correlation (ρ) - influences shape of the distribution
- sample size (n)

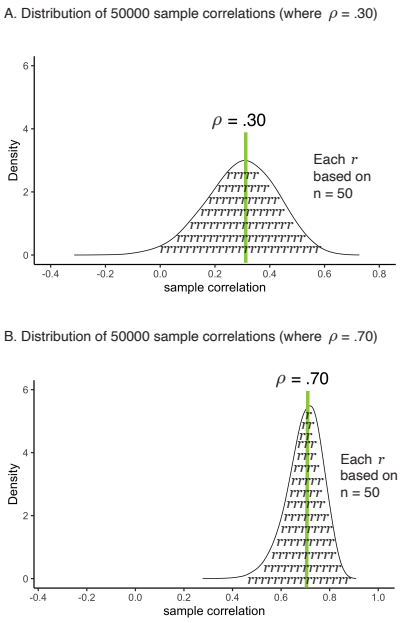


FIGURE 6.7: Correlation precision and effect size

7

Effect sizes from publications

7.1 Required

The following CRAN packages must be installed:

Required CRAN Packages
MBESS
cocor

7.2 Objective

In this chapter we focus on obtaining effect size estimates (e.g., correlation or standardized mean difference) from published articles. Often studies do not report effect sizes (e.g., d-values) or if they do, they don't report the confidence interval for that effect. We will learn more about confidence intervals in a future chapter, but for now, think of a confidence interval for a study/sample effect (e.g., d-value) as providing a plausible range of values for the population parameter (e.g., δ). Below we learn the R commands needed to obtain effect sizes with confidence intervals from published articles.

The ability to obtain effect size estimate from articles will be come extraordinarily important later in the course. Specifically, when it is time to engage in sample size planning (e.g., for your thesis) you will need to obtain estimates of the effects you are interested in from past research. The commands in this chapter will help you to do so.

7.3 Estimating δ

7.3.1 Independent Groups

When conducting an independent-groups t-test there are a few different ways to calculate d-values. We focus on two approaches in this chapter. First, researchers may assume that any intervention (e.g., control group vs. experimental group) will only affect group means (and not group standard deviations / variances). This is the most common scenario. When researchers have this belief they use a *pooled variance* approach to calculating the d-value. That is, the variances of the two groups are pooled (i.e., averaged) to create the denominator for the d-value. Second, researchers may assume that the intervention (e.g., control group vs. experimental group) will affect both group means and standard deviations/variances. This is a less common situation. When researchers have this belief they use one group (e.g., a control group) as the frame of reference for the comparison. That is, the standard deviation for a single group (e.g., the control group) is used as the denominator for the d-value.

7.3.1.1 d denominator: Pooled variance

7.3.1.1.1 Reported d-value no confidence interval

Scenario: A researcher reports a d-value of 1.67 in their article and that each condition has 50 people. The researcher believed that the intervention would only influence means and not standard deviations and used a pooled-variance d-value. They did not report a confidence interval. We can calculate a confidence interval for the independent groups d-value (pooled variance) with the command below:

```
library(MBESS)
ci.smd(sm = 1.67, n.1 = 50, n.2 = 50)
```

```
## $Lower.Conf.Limit.smd
## [1] 1.211
##
## $smd
## [1] 1.67
##
## $Upper.Conf.Limit.smd
## [1] 2.123
```

The population effect (δ) is unknown but our sample estimate is $d = 1.67$, 95% CI [1.21, 2.12]. This d-value will differ from the population effect (δ) due to sampling error. The confidence interval indicates that a plausible range for δ is 1.21 to 2.12.

7.3.1.1.2 Using cell means only

Scenario: A researcher does not report a d-value but indicates the descriptive statistics for the groups. For the first group $M = 5.00$, $SD = 1.20$, and $n = 50$. For the second group, $M = 3.00$, $SD = 1.00$, and $n = 50$. We can calculate the standardized mean difference (i.e., independent groups d-value) using the command below:

```
library(MBESS)
smd(Mean.1 = 5,
    s.1 = 1.2,
    n.1 = 50,
    Mean.2 = 3,
    s.2 = 1.0,
    n.2 = 50)
```

```
## [1] 1.811
```

Our sample estimate of δ is $d = 1.81$. The 95% confidence interval is obtained below:

```
library(MBESS)
ci.smd(sm = 1.81, n.1 = 50, n.2 = 50)
```

```
## $Lower.Conf.Limit.smd
## [1] 1.34
##
## $smd
## [1] 1.81
##
## $Upper.Conf.Limit.smd
## [1] 2.273
```

The population effect (δ) is unknown but our sample estimate is $d = 1.81$, 95% CI [1.34, 2.27]. This d-value will differ from the population effect (δ) due to sampling error. The confidence interval indicates that a plausible range for δ is 1.34 to 2.27.

7.3.1.1.3 t: Equal group sizes

Scenario: A researcher does not report a d-value, descriptive statistics, or n per group. Fortunately, the researcher did report the t-value, $t(98) = 8.30$, and the fact that the groups were the same size. After reading the article you believe the intervention is only likely to affect the means of the groups and not the variances - so a pooled variance term for the denominator of the d-value is appropriate. We can use the information provided to calculate a d-value and confidence interval.

To calculate the d-value we need to know the number of people per group. We can obtain that information from the degrees of freedom (98) using the formula below.

$$df = n_1 + n_2 - 2$$

Because the two groups were the same size we just use n instead of n_1 and n_2. We know $df = 98$:

$$\begin{aligned} 98 &= n + n - 2 \\ 98 &= 2n - 2 \\ 98 + 2 &= 2n \\ 100 &= 2n \\ \frac{100}{2} &= n \\ 50 &= n \end{aligned}$$

```
library(MBESS)
ci.smd(ncp = 8.30, n.1 = 50, n.2 = 50)
```

```
## $Lower.Conf.Limit.smd
## [1] 1.201
##
## $smd
## [1] 1.66
##
## $Upper.Conf.Limit.smd
## [1] 2.112
```

The population effect (δ) is unknown but our sample estimate is $d = 1.66$, 95% CI [1.20, 2.11]. This d-value value will differ from the population effect (δ) due to sampling error. The confidence interval indicates that a plausible range for δ is 1.20 to 2.11.

7.3.1.1.4 t: Unequal group sizes

Scenario: A researcher does not report a d-value or descriptive statistics. Fortunately, the researcher did report the t-value, $t(98) = 9.36$, and the fact that the groups had 65 and 35 people. After reading the article you believe the intervention is only likely to affect the means of the groups and not the variances - so a pooled variance term for the denominator of the d-value is appropriate. We can use the information provided to calculate a d-value and confidence interval.

```
library(MBESS)
ci.smd(ncp = 9.36, n.1 = 65, n.2 = 35)
```

```
## $Lower.Conf.Limit.smd
## [1] 1.465
##
## $smd
## [1] 1.962
##
## $Upper.Conf.Limit.smd
## [1] 2.453
```

The population effect (δ) is unknown but our sample estimate is $d = 1.96$. This value will differ from the population effect (δ) due to sampling error. The confidence interval, 95% CI [1.47, 2.45] indicates that a plausible range for δ is 1.47 to 2.45.

7.3.1.2 d denominator: Control group variance

In this section we focus on calculating the d-value when the researcher of the published article has used the standard deviation of a single group as the denominator for the d-value. This is usually done when the researcher believes an experimental intervention will influence both group means and group standard deviations. Researcher who use this approach will hopefully have made it clear what they have done. If you're not sure what a researcher has done I suggest using the pooled variance approach for the denominator above.

7.3.1.2.1 Reported d-value no confidence interval

Scenario: A researcher reports a d-value of 0.85 in their article and that each condition has 50 people. The researcher believed that the intervention would influence both means and standard deviations. Therefore, when they calculated the d-value they used the standard deviation of a single group (the

control group). We can calculate a confidence interval for the independent groups d-value (control group variance) with the command below:

```
library(MBESS)
ci.smd.c(smd.c = 0.85, n.C = 50, n.E = 50)
```

```
## $Lower.Conf.Limit.smd.c
## [1] 0.4199
##
## $smd.c
## [1] 0.85
##
## $Upper.Conf.Limit.smd.c
## [1] 1.273
```

The population effect (δ) is unknown but our sample estimate is $d = 0.85$, 95% CI [0.42, 1.27]. This d-value will differ from the population effect (δ) due to sampling error. The confidence interval indicates that a plausible range for δ is 0.42 to 1.27.

7.3.1.2.2 Using cell means

Scenario: A researcher does not report a d-value but indicates the descriptive statistics for the groups. For the experimental group $M = 5.00$, $SD = 1.20$, and $n = 50$. For the control group, $M = 3.00$, $SD = 1.40$, and $n = 50$. After reading the article you believe that intervention might well influence both means and standard deviations. Therefore, you think a control group standard deviation should be the frame of reference for the d-value. You can calculate the standardized mean difference (i.e., independent groups d-value) using the command below:

```
library(MBESS)
smd.c(Mean.T = 5,
    Mean.C = 3,
    s.C = 1.4,
    n.C = 50)
```

```
## [1] 1.429
```

The confidence interval can be obtained with the command:

```
library(MBESS)
ci.smd.c(smd.c = 1.43, n.E = 50, n.C = 50)
```

```
## $Lower.Conf.Limit.smd.c
## [1] 0.942
##
## $smd.c
## [1] 1.43
##
## $Upper.Conf.Limit.smd.c
## [1] 1.908
```

The population effect (δ) is unknown but our sample estimate is $d = 1.43$, 95% CI [0.94, 1.91]. This d-value will differ from the population effect (δ) due to sampling error. The confidence interval indicates that a plausible range for δ is 0.94 to 1.91.

7.3.2 Repeated Measures

For a repeated measures design, different d-values can be reported. We focus on the formula below for repeated measures d-values. In this formula the symbol, $\bar{x}_{diff}$, indicates the mean of the column of differences between the two times. Likewise, s_{diff}, is used to indicate the standard deviation of the column of differences.

$$d = \frac{\bar{x}_{diff}}{s_{diff}}$$

7.3.2.1 Reported repeated d-value, no confidence interval

Scenario: A researcher reports a repeated-measures $d = .50$ and $n = 150$. The 95% confidence interval for this d-value (i.e., standardized mean difference) can be obtained with the command below.

```
library(MBESS)
ci.sm(sm = .50, N = 150)
```

```
## [1] "The 0.95 confidence limits for the standardized mean are given as:"
## $Lower.Conf.Limit.Standardized.Mean
## [1] 0.3295
##
## $Standardized.Mean
## [1] 0.5
##
## $Upper.Conf.Limit.Standardized.Mean
## [1] 0.669
```

In this repeated measures design the population effect (δ) is unknown but our sample estimate of weight loss is $d = 0.50$, 95% CI [0.33, 0.67]. This d-value will differ from the population effect (δ) due to sampling error. The confidence interval indicates that a plausible range for δ is 0.33 to 0.67.

7.3.2.2 Using repeated measures mean difference

Scenario: A researcher reports a repeated-measures design on weight loss but does not report the d-value or confidence interval. He does report, however, that weights decreased on average by 3.00 lbs, SD = 6.00.

In order to understand what the M = 3.00 lbs, SD = 6.00 refers to, consider the following situation. The participants "weigh in" at time 1. They diet for three months. Then they "weight out" at time 2. There are now two columns (time1, time2) with weight information - one participant per row. We can create a third column, called diff, by subtracting time 1 weight from time 2 weights. That is, diff = time 2 weight - time 1 weight. The new diff column indicates how the weights for each person have changed over the diet. The mean of the diff column is $\bar{x}_{diff} = 3.00$ and the standard deviation of the diff column is $s_{diff} = 6.00$.

This information can be used in the formula below:

$$\begin{aligned} d &= \frac{\bar{x}_{diff}}{s_{diff}} \\ &= \frac{-3.00}{6.00} \\ &= -.50 \end{aligned}$$

We tend to always report d-values as positive values so $d = .50$. We can get a confidence interval around this value with the code below.

```
library(MBESS)
ci.sm(sm = .5, N =150)
```

```
## [1] "The 0.95 confidence limits for the standardized mean are given as:"
## $Lower.Conf.Limit.Standardized.Mean
## [1] 0.3295
##
## $Standardized.Mean
## [1] 0.5
##
## $Upper.Conf.Limit.Standardized.Mean
## [1] 0.669
```

In this repeated measures design the population effect (δ) is unknown but our sample estimate of weight loss is $d = 0.50$, 95% CI [0.33, 0.67]. This d-value will differ from the population effect (δ) due to sampling error. The confidence interval indicates that a plausible range for δ is 0.33 to 0.67.

7.3.2.3 Using repeated measures cell information

Scenario: A researcher reports a repeated-measures design on weight loss but does not report the d-value, mean difference, or standard deviation for the differences. He does, however, report mean and standard deviation for before and after the weight loss program. As well, he reports the correlation between before diet weights and after diet weights. The mean weight before the diet was M = 140 lbs, SD = 5 whereas after the diet the mean weight was 132 lbs, SD = 6. The correlation between time 1 and time 2 weights was r =.80.

In order to understand what the M = 140 lbs, SD = 5 refers to, consider the following situation. The participants "weigh in" at time 1. They diet for three months. Then they "weight out" at time 2. There are now two columns (time1, time2) with weight information. We can create a third column, called diff, by subtracting time 1 weight from time 2 weights. That is, diff = time 2 weight - time 1 weight. The new diff column indicates how the weights for each person have changed over the diet.

The researcher reports descriptive statistics (M, SD) for the time 1 and time 2 columns but nothing about the diff column. We can, however, figure the the mean of the diff column, $barx_{diff}$, and the standard deviation of the diff column, s_{diff}, from the information provided.

Recall the formula for repeated measures d-value:

$$d = \frac{\bar{x}_{diff}}{s_{diff}}$$

We can obtain the numerator easily:

$$\begin{aligned} \bar{x}_{diff} &= \bar{x}_1 - \bar{x}_2 \\ &= 140 - 132 \\ &= 8 \end{aligned}$$

Obtaining the values for the denominator is a bit more complicated. We can calculate s_{diff} but it requires the standard deviations of the before weights ($s_1 = 5$) and the after weights ($s_1 = 6$) - as well as the correlation between the two times ($r = .80$).

We begin by calculating s^2_{diff} which is variance of the column of differences differences:

$$\begin{aligned} s^2_{diff} &= s^2_1 + s^2_2 + 2(s_1)(s_2)r_{12} \\ &= 5^2 + 6^2 + 2(5)(6)(.80) \\ &= 25 + 36 + 48 \\ &= 109 \end{aligned}$$

We obtain s_{diff} by taking the square root of s^2_{diff}:

$$\begin{aligned} s_{diff} &= \sqrt{s^2_{diff}} \\ &= \sqrt{109} \\ &= 10.44031 \end{aligned}$$

Then we calculate the d-value using the numerator and denominator we calculated:

$$\begin{aligned} d &= \frac{\bar{x}_{diff}}{s_{diff}} \\ &= \frac{8}{10.44031} \\ &= 0.7662608 \\ &= 0.77 \end{aligned}$$

The d-value is 0.77 but we need a confidence interval:

```
library(MBESS)
ci.sm(sm = 0.77, N =100)
```

```
## [1] "The 0.95 confidence limits for the standardized mean are given as:"
## $Lower.Conf.Limit.Standardized.Mean
## [1] 0.5451
##
## $Standardized.Mean
## [1] 0.77
##
## $Upper.Conf.Limit.Standardized.Mean
## [1] 0.9918
```

In this repeated measures design the population effect (δ) is unknown but our sample estimate of weight loss is $d = 0.77$, 95% CI [0.55, 0.99]. This d-value will differ from the population effect (δ) due to sampling error. The confidence interval indicates that a plausible range for δ is 0.55 to 0.99.

7.3.2.4 Using repeated measures t-value

Scenario: A researcher reports a repeated-measures but does not report a d-value, mean difference with SD, or even the sample size. He does, however, report that t(40) = 6.23. The 95% confidence interval for this d-value (i.e., standardized mean difference) can be obtained with the command below.

$$\begin{aligned} df &= N - 1 \\ 40 &= N - 1 \\ 40 + 1 &= N \\ 41 &= N \end{aligned}$$

We can use the N and t-value to calculate the d-value and CI using the command:

```
# t-value for repeated measures if
library(MBESS)
ci.sm(ncp = 6.23, N = 41)
```

```
## [1] "The 0.95 confidence limits for the standardized mean are given as:"
## $Lower.Conf.Limit.Standardized.Mean
## [1] 0.5962
##
## $Standardized.Mean
## [1] 0.973
##
## $Upper.Conf.Limit.Standardized.Mean
## [1] 1.341
```

In this repeated measures design the population effect (δ) is unknown but our sample estimate of weight loss is $d = 0.97$, 95% CI [0.60, 1.34]. This d-value will differ from the population effect (δ) due to sampling error. The confidence interval indicates that a plausible range for δ is 0.60 to 1.34.

7.4 Estimating ρ

7.4.1 A single correlation

Correlations are frequently reported in the literature. Less common, however, is the reporting of confidence intervals for those correlations. A confidence

interval for a reported correlations (e.g., $r = .40$) can be obtained if you also have the sample size ($n = 200$) with the command below:

```
library(MBESS)
ci.cc(r = .40, n = 200)
```

```
## $Lower.Limit
## [1] 0.2766
##
## $Estimated.Correlation
## [1] 0.4
##
## $Upper.Limit
## [1] 0.5104
```

From this output you see that $r = .40$, 95% CI [.28, .51]. The population effect (ρ) is unknown but our sample estimate of the population correlation is $r = .30$, 95% CI [.28, .51]. This sample correlation will differ from the population correlation (ρ) due to sampling error. The confidence interval indicates that a plausible range for ρ is .28 to .51.

7.4.2 Difference between two correlations

Sometimes a researcher will conduct two studies in the same article and compare the correlations.

Scenario: A researcher reports in Study 1 he found that $r = .32$, N = 300. But in Study 2 the he found a correlation of $r = .51$, N = 400. You want to know the difference between the correlations and a confidence interval for that difference.

$$\begin{aligned}\Delta r &= r_2 - r_1 \\ &= .51 - .32 \\ &= .19\end{aligned}$$

You can obtain a confidence interval for this difference between correlations using the code below. Note, however, that this code applies only when the two correlations are from different samples. If the correlations are from the same sample the code is more complex but possible - see the cocor package documentation for details. The code for obtaining the confidence interval for the difference between these two correlations from different samples is below. Examine the last part of the output labeled zou2007.

```
library(cocor)
cocor.indep.groups(r1.jk = .51, r2.hm = .32, n1 = 300, n2 = 400)
```

```
##
##   Results of a comparison of two correlations based on independent groups
##
## Comparison between r1.jk = 0.51 and r2.hm = 0.32
## Difference: r1.jk - r2.hm = 0.19
## Group sizes: n1 = 300, n2 = 400
## Null hypothesis: r1.jk is equal to r2.hm
## Alternative hypothesis: r1.jk is not equal to r2.hm (two-sided)
## Alpha: 0.05
##
## fisher1925: Fisher's z (1925)
##   z = 3.0120, p-value = 0.0026
##   Null hypothesis rejected
##
## zou2007: Zou's (2007) confidence interval
##   95% confidence interval for r1.jk - r2.hm: 0.0668 0.3105
##   Null hypothesis rejected (Interval does not include 0)
```

From this output you see that in Study 1 $r = .32$ and in Study 2 $r = .51$. The difference between the correlations $\Delta r = .19$, 95% CI [.07, .31]. Both population correlations are unknown as is the difference between the population correlations. We have a sample estimate of the difference between the two population correlations, $\Delta r = .19$, and the confidence interval indicates that difference between the population correlations may plausibly be as small as .07 or as large as .31.

8

NHST and sample size

8.1 Required

The following CRAN packages must be installed:

Required CRAN Packages
MBESS
pwr

8.2 Overview

In this chapter we focus on statistical power – which is the probability of finding an effect if it exists. For example, if the power for your study is .90 that means you have a 90% chance of finding an effect if it exists. On the other hand, if your statistical power is below .50 that means you have less than a 50% chance of finding an effect if it exists. If your power is .50 you might well question whether it is even worth conducting your study - because the odds of finding an effect (if there is one) are so incredibly low. Low statistical power means you have a ***low*** *chance of concluding there is an effect* when an effect **is** present.

Unfortunately, many researchers only focus on statistical power with respect to failing to find an effect when it is present. In fact, low statistical power is directly related to false positive findings. That is, when statistical power is low and you obtain a significant p-value, it's likely that this is false positive finding (i.e., a falsely significant p-value). Low statistical power means you have a ***high chance*** *of concluding there is an effect* when an effect **is not** present.

The two points above indicate that low statistical power is associated with

untrustworthy research findings. When statistical power is low findings are not credible - regardless of whether they are significant or not. Unfortunately, power levels are typically quite low in psychology and related disciplines. Indeed, (**?**) noted in their *Nature Reviews Neuroscience* article that the "median statistical power of studies in the neuroscience field is optimistically estimated to be between ~8% and ~31%" (p. 8). This may seem surprising given the large effects sometimes observed in neuroscience, however, "the larger reported effect sizes in cognitive neuroscience may well be the consequence of effect size exaggeration due to having smaller sample sizes (as shown above) and consequential low power" (p. 8, **?**). Biomedical research, more generally, appears to have a low power problem (**?**). That said, the power levels in the rest of psychology are only marginally higher - so the entire field has substantial problem - it is not a problem limited to neuroscience or biomedical research.

Interesting, it is possible to calculate the average statistical power for a journal. Doing so allows us to examine many journals and, in particular, the relation between journal impact factor (how often they are cited) and journal statistical power. It turns out there is a negative relation between journal statistical power and journal impact factor (**?**). This indicates that as the journal impact factor increases there is a corresponding decrease in statistical power – which means means that journals with high impact factors (those that are most cited) tend to have the most untrustworthy findings. This conclusion is perhaps not as counter intuitive as it might seem. High impact journals often have policies that require published findings to be surprising. The most surprising finding is one that is wrong and inconsistent with previous research – the exact type of finding you will get with low statistical power.

8.3 Goals

In the rest of the chapter we focus on obtaining the desired power for our study by conducting a sample size analysis based on Null Hypothesis Significance Testing (NHST) logic. We caveat all of the subsequent advise is based on the goal of obtaining research findings that are trustworthy. Many times, however, research may be conducted with a different goal. For example, an Honours Thesis in psychology may be conducted with the primary goal being a learning experience for the student. When this is the case (as it often is in training scenarios) then the sample sizes used for the project may fall substantially short of the sample sizes suggested by power analyses below. This occurs frequently due to the fact that student theses have limited time and financial resources because the focus is on learning the research process rather than producing robust findings.

Conducting a sample size analysis requires a few pieces of information before you start.

1. Desired power. Power refers to probability of obtaining a significant result ($p < .05$) if the effect/difference exists. Often a power level of .80 or .85 is suggested when planning studies. However, a power-level of .90 is probability more advisable given how easy it is to obtain a p-value less than .05. Indeed, Dr. Daniel Lakens notes[1] that only individuals with power .90 or higher are eligible for funding in his Department at the Eindhoven University of Technology.
2. Population-level effect size. To conduct a sample-size analysis you need an estimate of the population effect size you are trying to find. Most commonly, for theses, that means you needs a estimate of the population-level d-value (i.e., δ) or population-level correlation (ρ) that you are trying to detect.

Below we present the process for conducting a sample size analysis for the independent groups t-test, repeated measures t-test, andcorrelations. For each of these analyses we go through three steps:

1. Estimating the population effect size
2. Determining the desired sample size for your study
3. Determining what sample-level effect sizes will be significant in your study with the desired sample size.

As we review these steps for the analyses below we omit using subjective judgment as the basis for estimating the population effect size. Sometimes researchers are tempted to use a subjective judgment as to whether the population effect is small, medium, or large based on Cohen's (1988) benchmarks. Unfortunately, using benchmarks of this sort as the basis for determine the smallest effect size of interest (**SESOI**) at the population level is problematic because it is subjective. Many researchers might be tempted to take a middle of the road approach and use a medium effect size. Yet a review of meta-analytic effect sizes (i.e., population estimates) by (**?**) indicated that 2/3 of the population effect sizes in psychology are lower than a medium effect. Consequently, taking a middle of the road approach and assuming a medium effect size is likely to result in low statistical power. Indeed, "[r]elying on a benchmark is the weakest possible justification of a SESOI and should be avoided." (p. 262, **?**)

[1] http://daniellakens.blogspot.com/2017/05/how-power-analysis-implicitly-reveals.html

Using a sample size of previous study. One approach to sample size analysis is simply to use the sample size from a previous study conducted in that research area. This is the worst possible strategy for choosing a sample size. Richard Morey and Daniel Lakens discuss[2] how most study results in psychology are statistically unfalsifiable because the sample sizes (and corresponding power) are so low. Consequently, using a the sample size for your study based on past research is not advised. Doing so will most likely to result in a sample size that is so small any resulting findings will be untrustworthy and unfalsifiable. Conduct a proper sample size analysis.

8.4 Independent groups t-test

8.4.1 Population effect size

8.4.1.1 Safeguard approach

The d-value obtained from a previous independent groups t-test study is highly influenced by sampling error and therefore a relatively poor estimate of the population d-value (δ). The population d-value may be substantially higher or lower than the previous study's sample d-value. If the population d-value is higher than the previous study's d-value this does not create a problem for detecting the effect being studied. On the other hand, if the population d-value is substantially lower than the previous study's d-value then the situation is problematic. Using the previous study's effect size in your sample size/power analysis will produce a desired sample size for your study that is insufficient to detect the effect of interest. Consequently one approach for determining the smallest effect size of interest in your power analysis is a safeguard approach (**?**).

With this approach you use the lower bound of the 95% confidence interval as the population effect size in your power / sample size analysis. Let's assume the effect size in the previous study was $d = 0.45$ and there were 50 people in each group. If the previous study did not report an effect size - you can use the approaches outlined in the "Effect sizes from publications" chapter to obtain one. The confidence interval is obtained by:

```
library(MBESS)
ci.smd(smd = .45 , n.1 = 50, n.2 = 50)
```

```
## $Lower.Conf.Limit.smd
## [1] 0.05187
##
## $smd
## [1] 0.45
##
## $Upper.Conf.Limit.smd
## [1] 0.8459
```

Thus, $d = 0.45$, 95% CI [0.05, 0.85]. Therefore, with a sample d-value of 0.45 ($n_1 = n_2 = 50$) a plausible range for the population d-value (δ) is 0.05 to 0.85. That is, the population d-value (δ) could be as low as 0.05. Therefore, we would use $\delta = 0.05$ as the population effect size in next stage of the power/sample size analysis: Determining Sample Size.

8.4.1.2 Small telescope

The small telescope approach to determining the effect size to use in your independent groups t-test power/sample size analysis is based on the analogy of a telescope (**?**). Imagine you are an astronomer. A previous study used a small telescope and claimed to see a new asteroid. You are the proud owner of a large telescope and decide to look for the asteroid. If the asteroid exists then you should be able to find it easily with your larger telescope. Conversely, if the you are unable to find the asteroid with your larger telescope it draws into question the original study. In this example, the size of the telescope is a proxy for statistical power. Use of this approach is based on the premise the previous study had a significant finding.

We begin by assuming the previous study is a small telescope and has low statistical power. We use the cell sizes for the two groups from the previous study and assume a power of .33. Then we calculate the population effect that would produce a power of .33 using the code below:

```
library(pwr)

# Based on the previous study modify the settings below.
# For alternative: use "greater" for one-sided
# and "two.sided" for two-sided test
# For n1 and n2 indicate the two group sizes
alternative <- "greater"
n1 <- 50
n2 <- 50

# do not modify this line
pwr.t2n.test(power = .33,
```

```
            n1 = n1,
            n2 = n2,
            alternative = alternative)
```

```
##
##      t test power calculation
##
##              n1 = 50
##              n2 = 50
##               d = 0.2427
##       sig.level = 0.05
##           power = 0.33
##     alternative = greater
```

The indicates when $\delta = .24$ the previous study (with 50 per group) would have statistical power of .33. Therefore, we would use $\delta = .24$ as the population effect size in next stage of the power/sample size analysis: Determining Sample Size.

8.4.1.3 Smallest sig. effect

Another approach to determine the effect size using in your independent groups *t*-test power / sample size analysis is to determine the smallest effect that would have been significant in the previous study (**?**). We do that with the code below:

```
# Based on the previous study modify the settings below.
# For n1 and n2 indicate the two group sizes
n1 <- 50
n2 <- 50

# Did the previous study use a direction (one-tail test)
# Set p_critical accordingly
# use .975 if original study was two-tail test;
# use .95 if original study was one-tail test
p_critical <- .95 # indicates one-tail test

# do not modify the lines below
df <- n1 + n2 - 2
t_critical <- qt(p = p_critical, df = df)
d_critical <- t_critical * sqrt(1/n1 + 1/n2)
print(d_critical)
```

```
## [1] 0.3321
```

The indicates that the smallest sample d-value that would have been significant in the previous study is $d = 0.33$. Therefore, we would use $\delta = .33$ as the population effect size in next stage of the power/sample size analysis: Determining Sample Size.

8.4.2 Determining sample size

To determine a sample size for your independent groups t-test you need to:

- specify an estimate of the population effect size (see the various methods reviewed above)
- specify desired power (probability of a significant effect if it exists)
- specify if you are conducting a one-tailed or two-tailed test
- examine a graph to see how a greater/fewer number of participants influences power
- examine a graph to see how a higher/lower population effect power

In the example below we use:

- Effect size estimate from the small telescope approach ($\delta = .24$)
- Desired power of .90 - a 90% chance of finding our effect if it exists
- The fact the we wish to conduct a one-tail test (i.e., in our study we will have a directional alternative hypothesis)

```
library(pwr)
# Based on the previous study modify the settings below.
# For alternative: use "greater" for one-sided
# and "two.sided" for two-sided test
alternative <- "greater"
pop_d <- .24
power <- .90

pwr_out <- pwr.t.test(d = pop_d,
                      power = power,
                      type = "two.sample",
                      alternative = alternative)
```

Then we need to print our power / sample size analysis:

```
print(pwr_out)
```

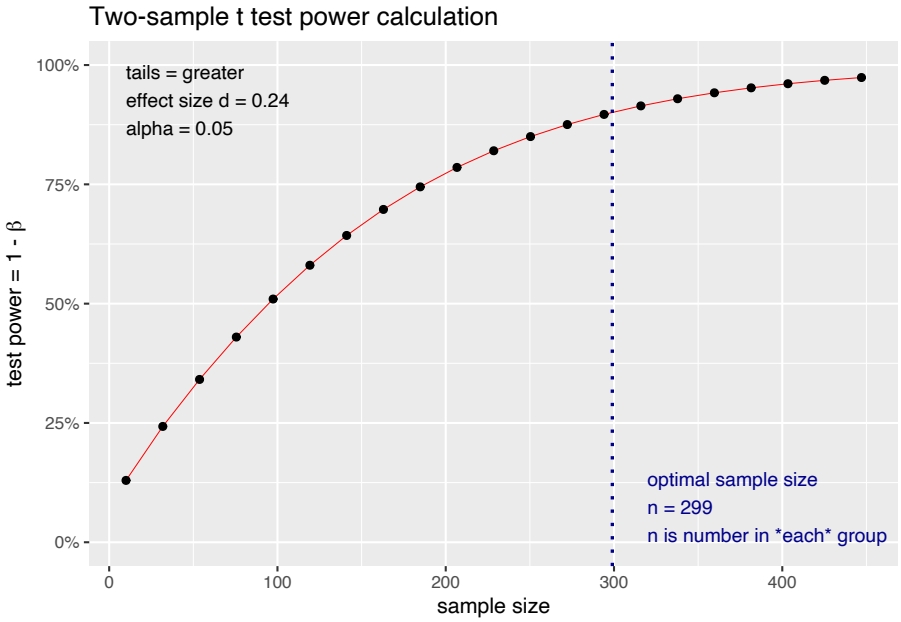
```
##
##      Two-sample t test power calculation
##
##               n = 298
##               d = 0.24
##       sig.level = 0.05
##           power = 0.9
##     alternative = greater
##
## NOTE: n is number in *each* group
```

This analysis reveals that a) if we assume the population effect size is .24 ($\delta = .24$) and b) desire a 90% chance of finding our effect if it exits - we need 298 people/animals per group. So a total of 596 participants for these two cells.

8.4.2.0.1 A different number of participants?

But what happens to our power if we get a greater/fewer number of participants? We don't always have control over the exact number due to non-response rates etc. We can use the graph below to see how power changes as the number of participants changes. Based on an examination of this graph, we may want to adjust the number of participants.

```
plot(pwr_out)
```



We can see the information conveyed in the graph above for an independent groups *t*-test in table form using the code below.

```
# Specify the population effect size from your power analysis
pop_d <- .24

# Do not change code below
pwr_50 <- round(pwr.t.test(d = pop_d, power = .50)$n)
pwr_80 <- round(pwr.t.test(d = pop_d, power = .80)$n)
pwr_95 <- round(pwr.t.test(d = pop_d, power = .95)$n)

power_labels <- c("dangerously low power (0 to .50)",
                  "low power (.50 to .80)",
                  "adequate power (.80 to .95)",
                  "almost certain power (> .95)")

power_table <- data.frame(power = power_labels,
                          n_start = round(c(1, pwr_50, pwr_80, pwr_95),2),
                          n_end = round(c(pwr_50, pwr_80, pwr_95, NA),2))

print(power_table)
```

```
##                              power n_start n_end
## 1 dangerously low power (0 to .50)       1   134
## 2           low power (.50 to .80)     134   273
```

```
## 3      adequate power (.80 to .95)     273   452
## 4     almost certain power (> .95)     452    NA
```

This table indicates that assuming a population effect size of $\delta = .24$ that your statistical power will be:

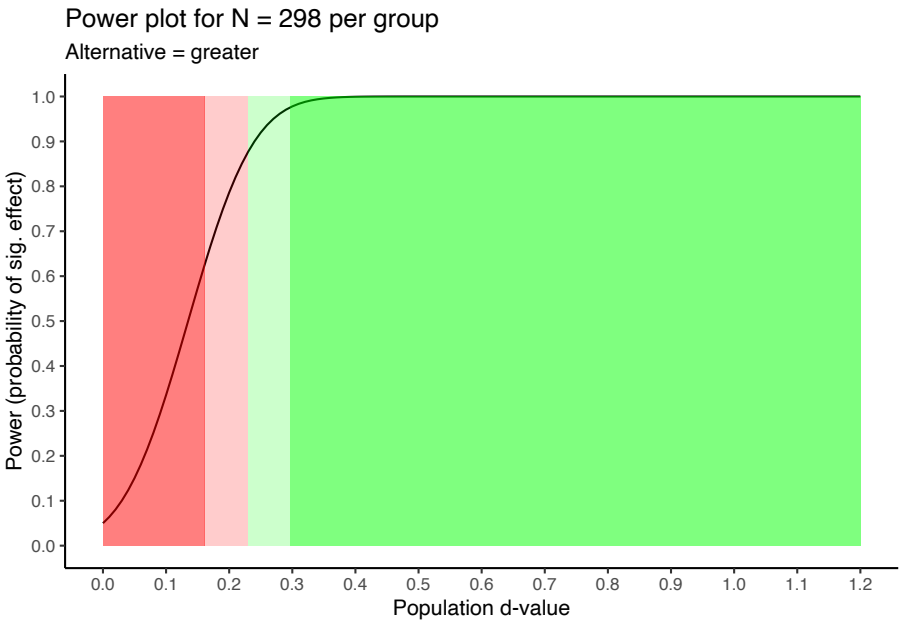
- dangerously low if you have between 1 and 134 participants **per group**
- low if you have between 134 and 273 participants **per group**
- adequate if you have between 273 and 452 participants **per group**
- excellent if you have 452 or more participants **per group**

8.4.2.0.2 A different effect size?

Sample size calculations use an estimate of the unknown population effect size. What happens to power if the population effect size is different than what we estimated. That information is conveyed in the graph below. An examination of this output might cause you to adjust your sample size.

```
library(pwr)
library(tidyverse)

# Indicate the type of test for your study.
# For alternative: use "greater" for one-sided
# and "two.sided" for two-sided test
alternative <- "greater"
n <- 298  # N per group for two.sample test
max_pop_d = 1.2  #max d value on x-axis. Must be high (e.g., 3 or so) when n is low.


# Do not modify code below
type = "two.sample" # Use "two.sample" or "paired"
pop_d <- seq(0, max_pop_d, by = 0.01)
pop_d_axis_values <- seq(0, max_pop_d, by = 0.10)
power_values <- data.frame(pop_d = pop_d, n = n, power = NA)
for (i in 1:nrow(power_values)){
  pwr_analysis <- pwr.t.test(d = power_values$pop_d[i],
                             n = n,
                             type = type,
                             alternative = alternative)
  power_values$power[i] <- pwr_analysis$power
}
```

```
pwr_50 <- pwr.t.test(n = n, power = .50, type = type)$d
pwr_80 <- pwr.t.test(n = n, power = .80, type = type)$d
pwr_95 <- pwr.t.test(n = n, power = .95, type = type)$d

power_plot <- ggplot(data = power_values,
                     mapping = aes(x = pop_d, y = power)) +
  geom_line() +
  scale_x_continuous(breaks = pop_d_axis_values) +
  scale_y_continuous(breaks = seq(0, 1, by = .10)) +
  labs(title = sprintf("Power plot for N = %g per group", n),
       subtitle = sprintf("Alternative = %s", alternative),
       x = "Population d-value",
       y = "Power (probability of sig. effect)") +
  annotate("rect", xmin = 0, xmax = pwr_50,
           ymin = 0, ymax = 1,
           fill = "red", alpha = .5) +
  annotate("rect", xmin = pwr_50, xmax = pwr_80,
           ymin = 0, ymax = 1,
           fill = "red", alpha = .2) +
  annotate("rect", xmin = pwr_80, xmax = pwr_95,
           ymin = 0, ymax = 1,
           fill = "green", alpha = .2) +
  annotate("rect", xmin = pwr_95, xmax = max_pop_d,
           ymin = 0, ymax = 1,
           fill = "green", alpha = .5) +
  theme_classic()

print(power_plot)
```



We can see the information conveyed in the graph above for an independent groups t-test in table form using the code below.

```
# Specify the sample size per group from your power analysis
n = 298 # N per group

# Do not modify the code below
pwr_50 <- pwr.t.test(n = n, power = .50, type = "two.sample")$d
pwr_80 <- pwr.t.test(n = n, power = .80, type = "two.sample")$d
pwr_95 <- pwr.t.test(n = n, power = .95, type = "two.sample")$d

power_labels <- c("dangerously low power (0 to .50)",
                  "low power (.50 to .80)",
                  "adequate power (.80 to .95)",
                  "almost certain power (> .95)")

power_table <- data.frame(power = power_labels,
                          delta_start = round(c(0, pwr_50, pwr_80, pwr_95),2),
                          delta_end = round(c(pwr_50, pwr_80, pwr_95, NA),2))

print(power_table)
```

```
##                              power delta_start delta_end
## 1 dangerously low power (0 to .50)        0.00      0.16
## 2           low power (.50 to .80)        0.16      0.23
```

```
## 3      adequate power (.80 to .95)        0.23      0.30
## 4     almost certain power (> .95)        0.30        NA
```

This table indicates that using 298 people **per group** your statistical power will be:

- dangerously low if the actual population effect is higher than 0 and lower than $\delta = 0.16$
- low if the actual population effect is between $\delta = 0.16$ and $\delta = 0.23$
- adequate if the actual population effect is between $\delta = 0.23$ and $\delta = 0.30$
- excellent if the actual population effect is higher than $\delta = 0.30$

8.4.3 What will be signficant?

Let's assume that after looking at all the graphs and tables you end up sticking with your initial sample size of 298 per group. What sample-level effect sizes will be significant when you do so?

```
# Specify the sample size per group from your power / sample size analysis
n1 <- 298
n2 <- 298

# Use .95 for one-sided and .975 for two-sided tests
percentile = .95

# Do not modify the code below
t_critical <- qt(percentile, df = (n1 + n2 - 2))
d_critical = t_critical * sqrt((1/n1) + (1/n2))
print(d_critical)
```

```
## [1] 0.135
```

This indicates that with a sample size of 298 per group that only independent groups d-values greater than $d = 0.135$ will be significant.

8.5 Repeated measures t-test

8.5.1 Population effect size

8.5.1.1 Safeguard approach

The d-value obtained from a previous repeated measures t-test study is highly influenced by sampling error and therefore a relatively poor estimate of the population d-value (δ). The population d-value may be substantially higher or lower than the previous study's sample d-value. If the population d-value is higher than the previous study's d-value this does not create a problem for detecting the effect being studied. On the other hand, if the population d-value is substantially lower than the previous study's d-value then the situation is problematic. Using the previous study's effect size in your sample size/power analysis will produce a desired sample size for your study that is insufficient to detect the effect of interest. Consequently one approach for determining the smallest effect size of interest in your power analysis is a safeguard approach (**?**).

With this approach you use the lower bound of the 95% confidence interval as the population effect size in your power / sample size analysis. Let's assume the effect size in the previous study was $d = 0.45$ and there were 50 participants. If the previous study did not report an effect size - you can use the approaches outlined in the "Effect sizes from publications" chapter to obtain one. The confidence interval is obtained by:

```
library(MBESS)
ci.sm(sm = .45 , N = 50)
```

```
## [1] "The 0.95 confidence limits for the standardized mean are given as:"
## $Lower.Conf.Limit.Standardized.Mean
## [1] 0.1568
##
## $Standardized.Mean
## [1] 0.45
##
## $Upper.Conf.Limit.Standardized.Mean
## [1] 0.739
```

Thus, $d = 0.45$, 95% CI [0.16, 0.74]. Therefore, with a sample d-value of 0.45 ($N = 50$) a plausible range for the population d-value (δ) is 0.16 to 0.74. That is, the population d-value (δ) could be as low as 0.16. Therefore, we would use

$\delta = 0.16$ as the population effect size in next stage of the power/sample size analysis: Determining Sample Size.

8.5.1.2 Small telescope

The small telescope approach to determining the effect size to use in your repeated measures *t*-test power/sample size analysis is based on the analogy of a telescope (**?**). Imagine you are an astronomer. A previous study used a small telescope and claimed to see a new asteroid. You are the proud owner of a large telescope and decide to look for the asteroid. If the asteroid exists then you should be able to find it easily with your larger telescope. Conversely, if the you are unable to find the asteroid with your larger telescope it draws into question the original study. In this example, the size of the telescope is a proxy for statistical power. Use of this approach is based on the premise the previous study had a significant finding.

We begin by assuming the previous study is a small telescope and has low statistical power. We use the csample size from the previous study and assume a power of .33. Then we calculate the population effect that would produce a power of .33 using the code below:

```
library(pwr)

# Based on the previous study modify the settings below.
# For alternative: use "greater" for one-sided
# and "two.sided" for two-sided test
# For n indicate the sample size
alternative <- "greater"
n <- 50

# do not modify this line
pwr.t.test(power = .33,
           n = n,
           type = "paired",
           alternative = alternative)
```

```
##
##      Paired t test power calculation
##
##               n = 50
##               d = 0.1728
##       sig.level = 0.05
##           power = 0.33
##     alternative = greater
##
```

```
## NOTE: n is number of *pairs*
```

The indicates when $\delta = 0.17$ the previous study (with 50 participants) would have statistical power of .33. Therefore, we would use $\delta = 0.17$ as the population effect size in next stage of the power/sample size analysis: Determining Sample Size.

8.5.1.3 Smallest sig. effect

Another approach to determine the effect size using in your repeated measures t-test power / sample size analysis is to determine the smallest effect that would have been significant in the previous study (**?**). We do that with the code below:

```
# Based on the previous study modify the settings below.
# For n indicate the sample size
n <- 50

# Did the previous study use a direction (one-tail test)
# Set p_critical accordingly
# use .975 if original study was two-tail test;
# use .95 if original study was one-tail test
p_critical <- .95 # indicates one-tail test

# do not modify the lines below
df <- n - 1
t_critical <- qt(p = p_critical, df = df)
d_critical <- t_critical * sqrt(1/n)
print(d_critical)
```

```
## [1] 0.2371
```

The indicates that the smallest sample d-value that would have been significant in the previous study is $d = 0.24$. Therefore, we would use $\delta = .24$ as the population effect size in next stage of the power/sample size analysis: Determining Sample Size.

8.5.2 Determining sample size

To determine a sample size for your repeated measures t-test you need to:

- specify an estimate of the population effect size (see the various methods reviewed above)
- specify desired power (probability of a significant effect if it exists)

- specify if you are conducting a one-tailed or two-tailed test
- examine a graph to see how a greater/fewer number of participants influences power
- examine a graph to see how a higher/lower population effect power

In the example below we use:

- Effect size estimate from the small telescope approach ($\delta = .17$)
- Desired power of .90 - a 90% chance of finding our effect if it exists
- The fact the we wish to conduct a one-tail test (i.e., in our study we will have a directional alternative hypothesis)

```
library(pwr)
# Based on the previous study modify the settings below.
# For alternative: use "greater" for one-sided
# and "two.sided" for two-sided test
alternative <- "greater"
pop_d <- .17
power <- .90

pwr_out <- pwr.t.test(d = pop_d,
                      power = power,
                      type = "paired",
                      alternative = alternative)
```

Then we need to print our power / sample size analysis:

```
print(pwr_out)
```

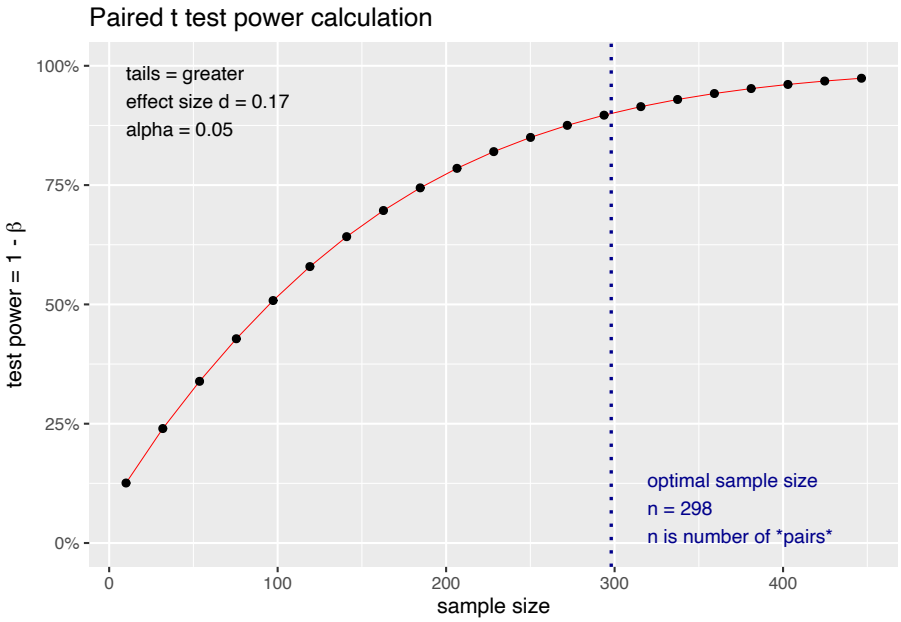
```
##
##      Paired t test power calculation
##
##               n = 297.7
##               d = 0.17
##       sig.level = 0.05
##           power = 0.9
##     alternative = greater
##
## NOTE: n is number of *pairs*
```

This analysis reveals that a) if we assume the population effect size for our repeated measures t-test is 0.17 ($\delta = 0.17$) and b) desire a 90% chance of finding our effect if it exits - we need 298 participants.

8.5.2.0.1 A different number of participants?

But what happens to our power if we get a greater/fewer number of participants? We don't always have control over the exact number due to non-response rates etc. We can use the graph below to see how power changes as the number of participants changes. Based on an examination of this graph, we may want to adjust the number of participants.

```
plot(pwr_out)
```



We can see the information conveyed in the graph above for an repeated measures *t*-test in table form using the code below.

```
# Specify the population effect size from your power analysis
pop_d <- .17

# Do not change code below
pwr_50 <- round(pwr.t.test(d = pop_d, power = .50, type = "paired")$n)
pwr_80 <- round(pwr.t.test(d = pop_d, power = .80, type = "paired")$n)
pwr_95 <- round(pwr.t.test(d = pop_d, power = .95, type = "paired")$n)

power_labels <- c("dangerously low power (0 to .50)",
                  "low power (.50 to .80)",
                  "adequate power (.80 to .95)",
```

```
                  "almost certain power (> .95)")

power_table <- data.frame(power = power_labels,
                          n_start = round(c(1, pwr_50, pwr_80, pwr_95),2),
                          n_end = round(c(pwr_50, pwr_80, pwr_95, NA),2))

print(power_table)
```

```
##                               power n_start n_end
## 1 dangerously low power (0 to .50)       1   135
## 2           low power (.50 to .80)     135   274
## 3      adequate power (.80 to .95)     274   452
## 4     almost certain power (> .95)     452    NA
```

This table indicates that assuming a population effect size of $\delta = .24$ that your statistical power will be:

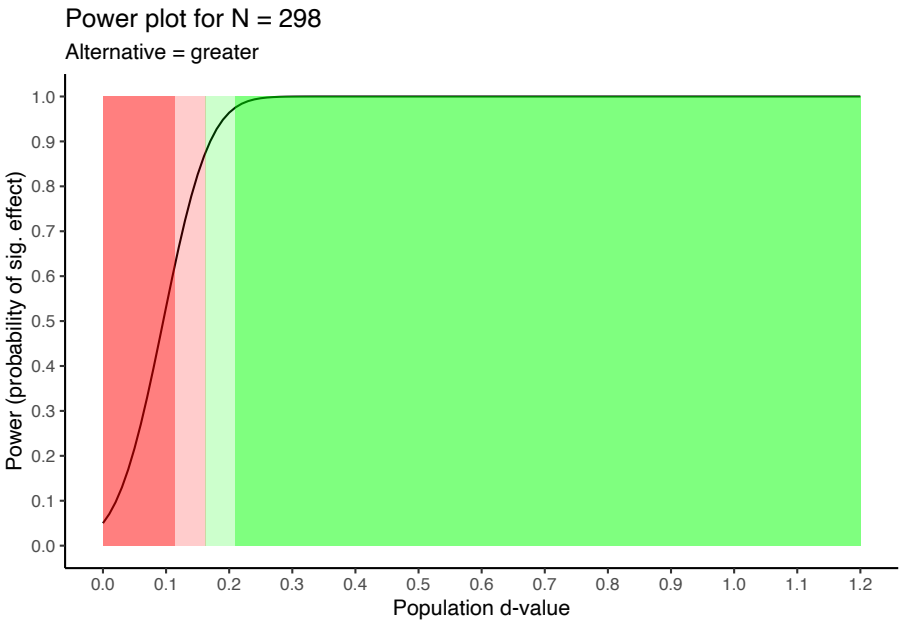
- dangerously low if you have between 1 and 135 participants
- low if you have between 135 and 274 participants
- adequate if you have between 274 and 452 participants
- excellent if you have 452 or more participants

8.5.2.0.2 A different effect size?

Sample size calculations use an estimate of the unknown population effect size. What happens to power if the population effect size is different than what we estimated. That information is conveyed in the graph below. An examination of this output might cause you to adjust your sample size.

```
library(pwr)
library(tidyverse)

# Indicate the type of test for your study.
# For alternative: use "greater" for one-sided
# and "two.sided" for two-sided test
alternative <- "greater"
n <- 298  # sample size
max_pop_d = 1.2  #max d value on x-axis. Must be high (e.g., 3 or so) when n is low.


# Do not modify code below
type = "paired" # Use "two.sample" or "paired"
```

```
pop_d <- seq(0, max_pop_d, by = 0.01)
pop_d_axis_values <- seq(0, max_pop_d, by = 0.10)
power_values <- data.frame(pop_d = pop_d, n = n, power = NA)
for (i in 1:nrow(power_values)){
  pwr_analysis <- pwr.t.test(d = power_values$pop_d[i],
                             n = n,
                             type = type,
                             alternative = alternative)
  power_values$power[i] <- pwr_analysis$power
}


pwr_50 <- pwr.t.test(n = n, power = .50, type = type)$d
pwr_80 <- pwr.t.test(n = n, power = .80, type = type)$d
pwr_95 <- pwr.t.test(n = n, power = .95, type = type)$d

power_plot <- ggplot(data = power_values,
                     mapping = aes(x = pop_d, y = power)) +
  geom_line() +
  scale_x_continuous(breaks = pop_d_axis_values) +
  scale_y_continuous(breaks = seq(0, 1, by = .10)) +
  labs(title = sprintf("Power plot for N = %g", n),
       subtitle = sprintf("Alternative = %s", alternative),
       x = "Population d-value",
       y = "Power (probability of sig. effect)") +
  annotate("rect", xmin = 0, xmax = pwr_50,
           ymin = 0, ymax = 1,
           fill = "red", alpha = .5) +
  annotate("rect", xmin = pwr_50, xmax = pwr_80,
           ymin = 0, ymax = 1,
           fill = "red", alpha = .2) +
  annotate("rect", xmin = pwr_80, xmax = pwr_95,
           ymin = 0, ymax = 1,
           fill = "green", alpha = .2) +
  annotate("rect", xmin = pwr_95, xmax = max_pop_d,
           ymin = 0, ymax = 1,
           fill = "green", alpha = .5) +
  theme_classic()

print(power_plot)
```



We can see the information conveyed in the graph above for an repeated measures t-test in table form using the code below.

```
# Specify the sample size from your power analysis
n = 298

# Do not modify the code below
pwr_50 <- pwr.t.test(n = n, power = .50, type = "paired")$d
pwr_80 <- pwr.t.test(n = n, power = .80, type = "paired")$d
pwr_95 <- pwr.t.test(n = n, power = .95, type = "paired")$d

power_labels <- c("dangerously low power (0 to .50)",
                  "low power (.50 to .80)",
                  "adequate power (.80 to .95)",
                  "almost certain power (> .95)")

power_table <- data.frame(power = power_labels,
                          delta_start = round(c(0, pwr_50, pwr_80, pwr_95),2),
                          delta_end = round(c(pwr_50, pwr_80, pwr_95, NA),2))

print(power_table)
```

```
##                              power delta_start delta_end
## 1 dangerously low power (0 to .50)        0.00      0.11
## 2           low power (.50 to .80)        0.11      0.16
```

```
## 3      adequate power (.80 to .95)        0.16      0.21
## 4     almost certain power (> .95)        0.21        NA
```

This table indicates that using 298 participants your statistical power will be:

- dangerously low if the actual population effect is higher than 0 and lower than $\delta = 0.11$
- low if the actual population effect is between $\delta = 0.11$ and $\delta = 0.16$
- adequate if the actual population effect is between $\delta = 0.16$ and $\delta = 0.21$
- excellent if the actual population effect is higher than $\delta = 0.21$

8.5.3 What will be signficant?

Let's assume that after looking at all the graphs and tables you end up sticking with your initial sample size of 298. What sample-level effect sizes will be significant when you do so?

```
# Specify the sample size from your power / sample size analysis
n <- 298

# Use .95 for one-sided and .975 for two-sided tests
percentile = .95

# Do not modify the code below
t_critical <- qt(percentile, df = (n - 1))
d_critical <- t_critical * sqrt(1/n)
print(d_critical)
```

```
## [1] 0.09558
```

This indicates that with a sample size of 298 that only repeated measures d-values greater than $d = 0.10$ (rounded) will be significant.

8.6 Correlations

8.6.1 Population effect size

8.6.1.1 Safeguard approach

The sample correlation obtained from a previous study is highly influenced by sampling error and therefore a relatively poor estimate of the population correlation (ρ). The population correlation may be substantially higher or lower than the previous study's sample correlation. If the population correlation is higher than the previous study's sample correlation this does not create a problem for detecting the effect being studied. On the other hand, if the population correlation is substantially lower than the previous study's sample correlation then the situation is problematic. Using the previous study's effect size in your sample size/power analysis will produce a desired sample size for your study that is insufficient to detect the effect of interest. Consequently one approach for determining the smallest effect size of interest in your power analysis is a safeguard approach (**?**).

With this approach you use the lower bound of the 95% confidence interval as the population effect size in your power / sample size analysis. Let's assume the effect size in the previous study was $r = .35$ and there were 75 people. If the previous study did not report an effect size - you can use the approaches outlined in the "Effect sizes from publications" chapter to obtain one. The confidence interval is obtained by:

```
library(MBESS)
ci.cc(r = .35 , n = 75)
```

```
## $Lower.Limit
## [1] 0.1337
##
## $Estimated.Correlation
## [1] 0.35
##
## $Upper.Limit
## [1] 0.5345
```

Thus, $r = .35$, 95% CI [.13, .53]. Therefore, with a sample correlation of .35 ($N = 75$) a plausible range for the population correlation (ρ) is .13 to .53. That is, the population correlation (ρ) could be as low as .13. Therefore, we would use $\rho = .13$ as the population effect size in next stage of the power/sample size analysis: Determining Sample Size.

8.6.1.2 Small telescope

The small telescope approach to determining the effect size to use in your correlation power/sample size analysis is based on the analogy of a telescope (**?**). Imagine you are an astronomer. A previous study used a small telescope and claimed to see a new asteroid. You are the proud owner of a large telescope and decide to look for the asteroid. If the asteroid exists then you should be able to find it easily with your larger telescope. Conversely, if the you are unable to find the asteroid with your larger telescope it draws into question the original study. In this example, the size of the telescope is a proxy for statistical power. Use of this approach is based on the premise the previous study had a significant finding.

We begin by assuming the previous study is a small telescope and has low statistical power. We use the sample size from the previous study and assume a power of .33. Then we calculate the population effect that would produce a power of .33 using the code below:

```
library(pwr)

# Based on the previous study modify the settings below.
# For alternative: use "greater" for one-sided
# and "two.sided" for two-sided test
# For n indicate the sample size
alternative <- "two.sided"
n <- 75

# do not modify this line
pwr.r.test(n = n,
           power = .33,
           alternative = alternative)
```

```
##
##      approximate correlation power calculation (arctangh transformation)
##
##               n = 75
##               r = 0.1762
##       sig.level = 0.05
##           power = 0.33
##     alternative = two.sided
```

The indicates when the population correlation is .18 (rounded), $\rho = .18$, the previous study (with 75 participants) would have statistical power of .33. Therefore, we would use $\rho = .18$ as the population effect size in next stage of the power/sample size analysis: Determining Sample Size.

8.6.1.3 Smallest sig. effect

Another approach to determine the effect size using in your correlation power / sample size analysis is to determine the smallest effect that would have been significant in the previous study (**?**). We do that with the code below:

```
# Based on the previous study modify the settings below.
# For n indicate the sample size
n <- 75

# Did the previous study use a direction (one-tail test)
# Set p_critical accordingly
# use .975 if original study was two-tail test;
# use .95 if original study was one-tail test
p_critical <- .975 # indicates two-tail test

# do not modify the lines below
df <- n - 2
t_critical <- qt(p = p_critical, df = df)
r_critical <- sqrt((t_critical^2)/(df + t_critical^2))
print(r_critical)
```

```
## [1] 0.2272
```

The indicates that the smallest sample correlation that would have been significant in the previous study is $r = 0.23$. Therefore, we would use $\rho = .23$ as the population effect size in next stage of the power/sample size analysis: Determining Sample Size.

8.6.2 Determining sample size

To determine a sample size for your correlation study you need to:

- specify an estimate of the population effect size (see the various methods reviewed above)
- specify desired power (probability of a significant effect if it exists)
- specify if you are conducting a one-tailed or two-tailed test
- examine a graph to see how a greater/fewer number of participants influences power
- examine a graph to see how a higher/lower population effect power

In the example below we use:

- Effect size estimate from the small telescope approach ($\rho = .18$)
- Desired power of .90 - a 90% chance of finding our effect if it exists
- The fact the we wish to conduct a one-tail test (i.e., in our study we will have a directional alternative hypothesis)

```
library(pwr)
# Based on the previous study modify the settings below.
# For alternative: use "greater" for one-sided
# and "two.sided" for two-sided test
alternative <- "greater"
pop_r <- .18
power <- .90

pwr_out <- pwr.r.test(r = pop_r,
                      power = power,
                      alternative = alternative)
```

Then we need to print our power / sample size analysis:

```
print(pwr_out)
```

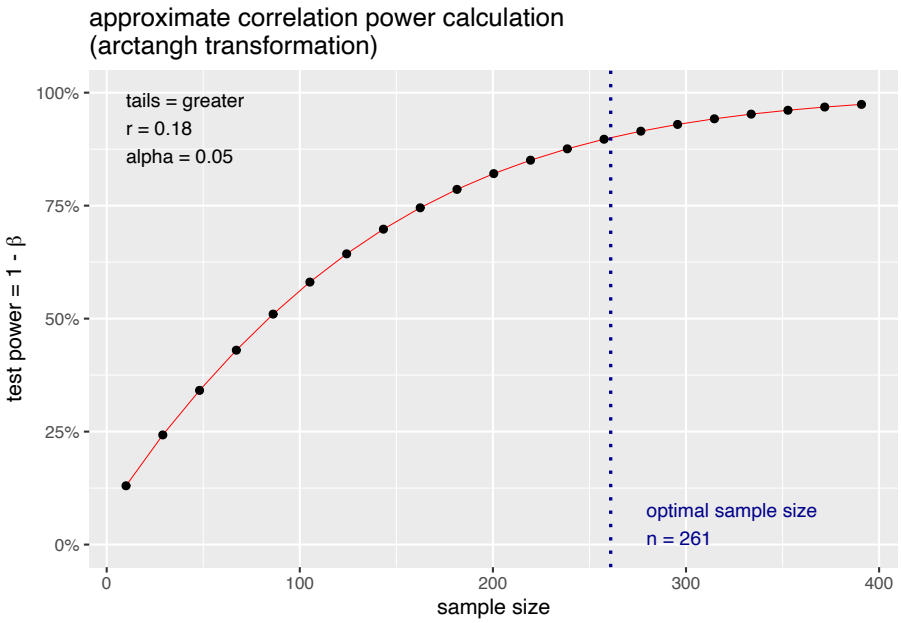
```
##
##      approximate correlation power calculation (arctangh transformation)
##
##               n = 260.6
##               r = 0.18
##       sig.level = 0.05
##           power = 0.9
##     alternative = greater
```

This analysis reveals that a) if we assume the population correlation is .18 ($\rho = .18$) and b) desire a 90% chance of finding our effect if it exits - we need 261 participants.

8.6.2.0.1 A different number of participants?

But what happens to our power if we get a greater/fewer number of participants? We don't always have control over the exact number due to non-response rates etc. We can use the graph below to see how power changes as the number of participants changes. Based on an examination of this graph, we may want to adjust the number of participants.

```
plot(pwr_out)
```



We can see the information conveyed in the graph above for our proposed correlation study in table form using the code below.

```
# Specify the population effect size from your power analysis
pop_r <- .18

# Do not change code below
pwr_50 <- round(pwr.r.test(r = pop_r, power = .50)$n)
pwr_80 <- round(pwr.r.test(r = pop_r, power = .80)$n)
pwr_95 <- round(pwr.r.test(r = pop_r, power = .95)$n)

power_labels <- c("dangerously low power (0 to .50)",
                  "low power (.50 to .80)",
                  "adequate power (.80 to .95)",
                  "almost certain power (> .95)")

power_table <- data.frame(power = power_labels,
                          n_start = round(c(1, pwr_50, pwr_80, pwr_95),2),
                          n_end = round(c(pwr_50, pwr_80, pwr_95, NA),2))

print(power_table)
```

```
##                                power n_start n_end
## 1 dangerously low power (0 to .50)       1   118
## 2           low power (.50 to .80)     118   239
## 3      adequate power (.80 to .95)     239   394
## 4     almost certain power (> .95)     394    NA
```

This table indicates that assuming a population effect size of $\rho = .18$ that your statistical power will be:

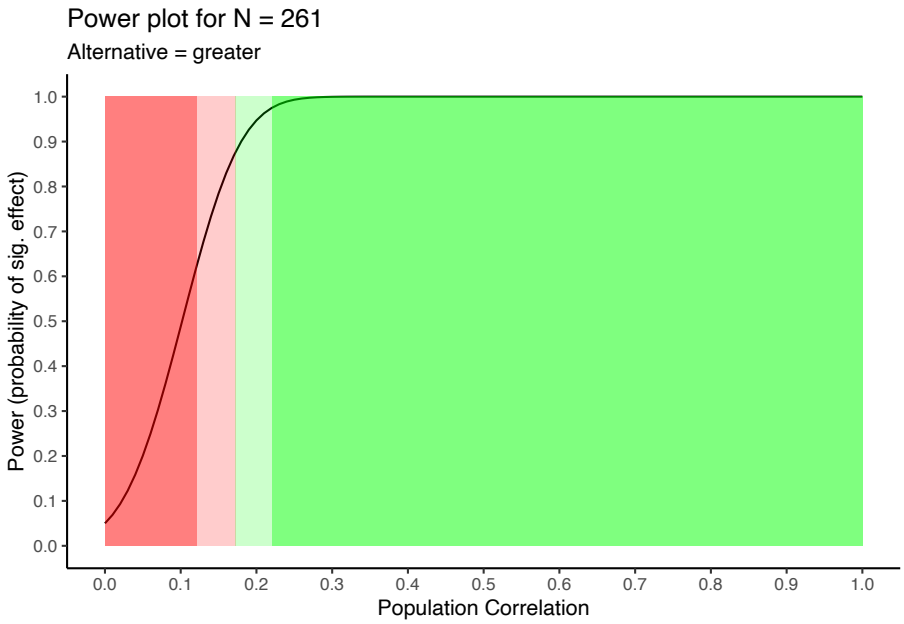
- dangerously low if you have between 1 and 118 participants
- low if you have between 118 and 239 participants
- adequate if you have between 239 and 394 participants
- excellent if you have 394 or more participants

8.6.2.0.2 A different effect size?

Sample size calculations use an estimate of the unknown population effect size. What happens to power if the population effect size is different than what we estimated. That information is conveyed in the graph below. An examination of this output might cause you to adjust your sample size.

```
library(pwr)
library(tidyverse)

# Indicate the type of test for your study.
# For alternative: use "greater" for one-sided
# and "two.sided" for two-sided test
alternative <- "greater"
n <- 261  # sample size


# Do not modify code below
rho <- seq(0, 1, by = 0.01)
rho_axis_values <- seq(0, 1, by = 0.10)
power_values <- data.frame(rho = rho, n = n, power = NA)
for (i in 1:nrow(power_values)){
  pwr_analysis <- pwr.r.test(r = power_values$rho[i],
                             n = n,
                             alternative = alternative)
  power_values$power[i] <- pwr_analysis$power
}

pwr_50 <- pwr.r.test(n = n, power = .50)$r
```

```
pwr_80 <- pwr.r.test(n = n, power = .80)$r
pwr_95 <- pwr.r.test(n = n, power = .95)$r
power_plot <- ggplot(data = power_values,
                     mapping = aes(x = rho, y = power)) +
  geom_line() +
  scale_x_continuous(breaks = rho_axis_values) +
  scale_y_continuous(breaks = seq(0, 1, by = .10)) +
  labs(title = sprintf("Power plot for N = %g", n),
       subtitle = sprintf("Alternative = %s", alternative),
       x = "Population Correlation",
       y = "Power (probability of sig. effect)") +
  annotate("rect", xmin = 0, xmax = pwr_50, ymin = 0, ymax = 1, fill = "red", alpha = .5)
  annotate("rect", xmin = pwr_50, xmax = pwr_80, ymin = 0, ymax = 1, fill = "red", alpha =
  annotate("rect", xmin = pwr_80, xmax = pwr_95, ymin = 0, ymax = 1, fill = "green", alpha
  annotate("rect", xmin = pwr_95, xmax = 1, ymin = 0, ymax = 1, fill = "green", alpha = .5
  theme_classic()
print(power_plot)
```



We can see the information conveyed in the graph above for a correlation study in table form using the code below.

```
# Specify the sample size from your power analysis
n = 261
```

```
# Do not modify the code below
pwr_50 <- pwr.r.test(n = n, power = .50)$r
pwr_80 <- pwr.r.test(n = n, power = .80)$r
pwr_95 <- pwr.r.test(n = n, power = .95)$r

power_labels <- c("dangerously low power (0 to .50)",
                  "low power (.50 to .80)",
                  "adequate power (.80 to .95)",
                  "almost certain power (> .95)")

power_table <- data.frame(power = power_labels,
                          rho_start = round(c(0, pwr_50, pwr_80, pwr_95),2),
                          rho_end = round(c(pwr_50, pwr_80, pwr_95, NA),2))

print(power_table)
```

```
##                              power rho_start rho_end
## 1 dangerously low power (0 to .50)      0.00    0.12
## 2           low power (.50 to .80)      0.12    0.17
## 3      adequate power (.80 to .95)      0.17    0.22
## 4     almost certain power (> .95)      0.22      NA
```

This table indicates that using 261 participants your statistical power will be:

- dangerously low if the actual population effect is higher than 0 and lower than $\rho = .12$
- low if the actual population effect is between $\rho = .12$ and $\rho = .17$
- adequate if the actual population effect is between $\rho = 0.16$ and $\rho = 0.22$
- excellent if the actual population effect is higher than $\rho = 0.22$

8.6.3 What will be signficant?

Let's assume that after looking at all the graphs and tables you end up sticking with your initial sample size of 261. What sample-level effect sizes will be significant when you do so?

```
# Specify the sample size from your power / sample size analysis
n <- 261

# Use .95 for one-sided and .975 for two-sided tests
percentile = .95
```

```
# Do not modify the code below
t_critical <- qt(percentile, df = (n - 2))
t2 <- t_critical^2
r_critical = sqrt(t2 / (t2 + (n-2)))
print(r_critical)
```

```
## [1] 0.102
```

This indicates that with a sample size of 261 that only sample correlations greater than $r = .10$ (rounded) will be significant.

9

Equivalence tests and sample size

9.1 Required

The following CRAN packages must be installed:

Required CRAN Packages
MBESS
TOSTER

9.2 Interpreting non-significant findings

A common problem in the psychological literature is the interpretation of non-significant effects. As (Kirk **?**) notes "some researchers mistakenly interpret a failure to reject the null hypothesis as evidence for accepting it". Indeed, it is unfortunately common to see an incorrect sentences like the following examples based on fictitious data: "The difference between the mean IQ's for males and females was non-significant, $t(98) = 12.247$, $p = .458$, indicating that males and females have, on average, the same intelligence." Or alternatively an incorrect sentence like: "The correlation between height and IQ was non-significant, $r = .15$, $p = .854$, indicating that height was not related to IQ." Both of these sentences are statistically incorrect because the conclusions do not follow from the reported statistics.

The tendency for researchers to incorrectly conclude that there is no effect, when $p > .05$, is particularly troubling (and ubiquitous) when interpreting the interaction in an ANOVA. For example, consider a 2 (sex) by 2 (occasion) between/within ANOVA where the dependent variable is reaction time. Imagine there is a significant sex x occasion interaction. The significant interaction indicates the relation between occasion and response time depends on the level of sex. The researcher describes this significant interaction by comparing the

reaction times of males and females at occasion 1 and then again at occasion 2: "A comparison of the mean reaction of times of males and females at occasion 1, $t(28) = 1.06$, $p = .300$, revealed no difference in reaction time. In contrast, at occasion 2, there was a difference in the mean reaction times of males and females, $t(28) = 2.15$, $p = .040$. Thus, reaction time were the same, on average, for males and females at occasion 1 but not occasion 2." In this example the researcher erred in the their interpretation of the results at occasion 1. Specifically, the researcher incorrectly concluded at occasion 1 that the reaction time for males and females was the same because $p > .05$. That type of conclusion is not possible for $p > .05$ in a standard paired comparison / t-test.

The calculation of a p-value begins by assuming the null hypothesis is true; consequently, you cannot use a p-value as evidence the null hypothesis is true. That is, when a p-value exceeds the threshold for significance (.05) you cannot conclude there is no effect. This fact is discussed at the 10 minute mark in an excellent video[1] by Daniel Lakens.

You might well wonder what to do if you do want to make the conclusion there this no effect/relation. This type of conclusion is possible but you need to use the right tool to do so. One tool for concluding there is no effect is the Bayes Factor BF[2] - but that is beyond the scope of this course. An easily accessible alternative for concluding there is no effect or relation is the equivalence test (**?**).

9.3 When would I use an equivalence test?

You can use an equivalent test when you want to conclude there is not effect or relation. This situation might be more common that you might think. A few of the scenarios where you would like to use an equivalence test are outlined below:

9.3.1 *t*-test

- You calculate a t-test and expect to find a difference between the two conditions. Unexpectedly, the hypothesized difference is non-significant. At this point you can't draw much of a conclusion. You can say the groups were not statistically different but you cannot say the groups were statistically

[1] https://youtu.be/RVxHlsIw_Do
[2] https://en.wikipedia.org/wiki/Bayes_factor

the same. An equivalence test could allow you to conclude the groups are statistically equivalent.

- For theoretical reasons your study may begin with the primary purpose being determine if two groups are the same (e.g., two treatments for the same disease that are believed to be equally effective).

9.3.2 Correlation

- You calculate a correlation and expect to find a relation between the two variables. Unexpectedly, the hypothesized relation is non-significant. At this point you can't draw much of a conclusion. You can say you didn't find evidence for a relation but you cannot say you found evidence of no relation. An equivalence test could allow you to conclude there is no relation between the two variables.
- For theoretical reasons the purpose of your study may be to provide evidence that there is no relation between two variables (e.g., video game use and violent behaviors).

9.4 Possible Outcomes

(**?**) review a variety of outcomes from an equivalence test. These are easiest to understand in the context of a *t*-test with two groups. You could find the two groups are:

1. Not statistically equivalent and not statistically different
2. Statistically equivalent and not statistically different
3. Statistically equivalent and statistically different
4. Not statistically equivalent and not statistically different

You can see from the possible outcomes above it is still possible to obtain an outcome that is difficult to interpret. The outcomes that are challenging to interpret are most likely to occur when you have small sample sizes and low statistical power for the equivalence test.

To avoid an ambiguous outcome from an equivalence test make sure you conduct a sample size analysis for an equivalence test prior to running your study. The sample size demands for an equivalence test may be **substantially** greater than for a traditional analysis. Therefore we encourage you to

conduct both a traditional sample size analysis and a sample size analysis for an equivalence test before you start collecting data.

9.5 What is an equivalence test

An equivalence test is just a pair of one-sided *t*-tests that are used to establish if an effect falls within a specified range of effect sizes bounding zero. That is, an equivalence test is used to indicate if an effect/relation is close enough to zero to be considered zero for practical purposes.

9.6 Defining a zero effect

How do you decide how close to zero is close enough to be practically zero? You already did so in the the previous chapter on "NHST and sample size". In that chapter you reviewed various ways of determining the smallest effect size of interest (SESOI). Any effect below the SESOI is logically close enough to zero to, for practical purposes, be zero. However, I encourage you to review (**?**) to see the full discussion on this issue.

For now, the most important aspect of conducting an equivalence test is the fact that you need to determine the smallest effect size of interest **prior** to data collection - to avoid equivalence testing being a fancy form of intentional or unintentional ***p*-hacking**[3]. Take note of this point – a recent article[4] indicated that approximately 25% of researchers have engaged in at least one form of *p*-hacking in the just last 12 months.

[3] https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1002106

[4] https://psyarxiv.com/3v7hx

9.7 Equivalence - repeated measures

Consider the following scenario where you begin a study with the intent to prove there is no effect. Many years ago the cereal Shreddies[5] engaged in an interested marketing strategy. They decided to market new Diamond Shreddies as illustrated below. Imagine that you are a researcher tasked to compare the taste of the two types of Shreddies. Participants are given a bowl of the old Shreddies and then asked to rate it on a 1 to 15 point scale where higher ratings indicate a better taste. Following this they are given a bowl of the new Diamond Shreddies and asked rate the taste. How do you go about comparing the taste ratings if your goal is to establish the taste of the old Shreddies is the same as new Diamond Shreddies?

An incorrect approach to determining if the two types of cereal taste the same would be to **just** conduct a repeated measures t-test and look for a non-significant difference. A non-significant repeated measures t-test would leave you with no conclusion. The appropriate approach in this circumstance is to use an equivalence test. Prior to collecting data you set your smallest effect size of interest. Specifically, you imagine getting a mean for each group and calculating a difference using the original numbers on the 15-point rating scale. You decide that if that difference is between -1 and +1 (the smallest jump on the rating scale) then you will consider the two types of Shreddies to have equivalent taste. Notably, as per (**?**), you set this smallest effect size of interest before you examine your data - to avoid being a p-hacker.

9.7.1 Raw units

After you collect your data ($N = 50$) you have ratings for old Shreddies ($M = 12.1$, $SD = 2.50$) and new Diamond Shreddies ($M = 11.9$, $SD = 2.50$). Because the same people taste both cereals you also have a correlation between the two

[5] https://fameable.com/diamond-shreddies-rebranding-case-study/144/

taste ratings of $r = .80$. You run the R-code below to conduct the equivalence test.

```
library(TOSTER)

TOSTpaired.raw(m1 = 12.1,
        sd1 = 2.5,
        m2 = 11.9,
        sd2 = 2.5,
        r12 = .8,
        n = 50,
        low_eqbound = -1,
        high_eqbound = 1,
        plot = FALSE)
```

```
Equivalence Test Result:
The equivalence test was significant, t(49) = -3.578, p = 0.000396,
given equivalence bounds of -1.000 and 1.000 (on a raw scale) and an alpha of 0.05.

Null Hypothesis Test Result:
The null hypothesis test was non-significant,
t(49) = 0.894, p = 0.375, given an alpha of 0.05.

Based on the equivalence test and the null-hypothesis test combined,
we can conclude that the observed effect is statistically not different
from zero and statistically equivalent to zero.
```

We can then report that:

A repeated measures *t*-test indicated that the mean taste ratings for old Shreddies ($M = 12.1$, $SD = 2.50$) and new Diamond Shreddies ($M = 11.9$, $SD = 2.50$) were not significantly different, $d = 0.13$, 95% CI [-0.15, 0.40], $t(49) = 0.984$, $p = 0.375$. Prior to conducting analyses we established that a raw difference in the -1 to +1 range (the smallest possible change on the scale) would, for practical purposes, be considered equivalent to zero. The equivalence test was significant $t(49) = -3.578$, $p < .001$ indicating the means for the two conditions were equivalent. Thus, we can conclude that the taste ratings of the two types of Shreddies are not statistically different and that they are statistically equivalent.

Note that the *d*-value with 95% CI was obtained with the MBESS command: ci.sm(ncp = 0.894, N = 50)

9.7.2 Standardized units

You could also have run this test by indicating the smallest effect size of interest using standardized effect size (i.e., a repeated measures d-value). The code below produces a result identical to the above code. We simply indicate the range of values that count as practically equivalent to zero using d-values.

```
library(TOSTER)

TOSTpaired(m1 = 12.1,
        sd1 = 2.5,
        m2 = 11.9,
        sd2 = 2.5,
        r12 = .8,
        n = 50,
        low_eqbound_dz = -0.6325,
        high_eqbound_dz = 0.6325,
        plot = FALSE)
```

9.8 Equivalence - Independent groups

9.8.1 Raw units

You could also have run this study as an independent groups t-test where different people received each type of cereal. In this case, the R-code would be as below. Note the output in this case provides an ambiguous outcome: "the observed effect is statistically not different from zero and statistically not equivalent to zero" due to our small sample size. Thus, the primary finding from this study is that we should have used a larger number of participants.

```
library(TOSTER)


TOSTtwo.raw(m1 = 12.1,
        sd1 = 2.5,
        m2 = 11.9,
        sd2 = 2.5,
        n1 = 50,
        n2 = 50,
```

```
        low_eqbound = -1,
        high_eqbound = 1,
        plot = FALSE)
```

```
Equivalence Test Result:
The equivalence test was non-significant, t(98) = -1.600, p = 0.0564,
given equivalence bounds of -1.000 and 1.000 (on a raw scale) and an alpha of 0.05.

Null Hypothesis Test Result:
The null hypothesis test was non-significant,
t(98) = 0.400, p = 0.690, given an alpha of 0.05.

Based on the equivalence test and the null-hypothesis test combined,
we can conclude that the observed effect is statistically not different
from zero and statistically not equivalent to zero.
```

9.8.2 Standardized units

The R-code again for using standardized effect sizes:

```
library(TOSTER)

TOSTtwo(m1 = 12.1,
        sd1 = 2.5,
        m2 = 11.9,
        sd2 = 2.5,
        n1 = 50,
        n2 = 50,
        low_eqbound_d = -0.4,
        high_eqbound_d = 0.4,
        plot = FALSE)
```

9.9 Equivalence Correlation

Imagine we are interested in conducting a study to prove our theory that there is a strong positive relation between academic performance and self-esteem. Prior to conducing the study we determined our smallest effect size

of interest was .20. A traditional sample size analysis (with a desire for 90% power) indicated we should use a sample size of 210 (for a one-sided test). However, we also conducted a sample size analysis for an equivalence test, in case the correlation was non-significant, which suggested a larger sample size of 267. Consequently, we collected data from 267 students.

Our study found $r = .09$, $p = 0.071$ (one-tailed). This non-significant difference means there is little to conclude from this study. We can't say there is a relation due to the non-significance. But, we also can't say there is no relation. Fortunately, because we established our smallest effect size of interest prior to looking at our data we can run an equivalence test with the code below.

```
library(TOSTER)
TOSTr(n = 267,
      r = .09,
      low_eqbound_r = -.20,
      high_eqbound_r = .20,
      plot = FALSE)
```

```
Equivalence Test Result:
The equivalence test was significant, p = 0.0338,
given equivalence bounds of -0.200 and 0.200 and an alpha of 0.05.

Null Hypothesis Test Result:
The null hypothesis test was non-significant, p = 0.142, given an alpha of 0.05.

Based on the equivalence test and the null-hypothesis test combined,
we can conclude that the observed effect is statistically not different
from zero and statistically equivalent to zero.
```

Because we conducted the equivalence test we can write a clear results section.

The relation between academic performance and self-esteem was non-significant, $r = .09$, 95% CI [-.03, .21], $p = 0.071$ (one-sided). A follow up equivalence test (based on our apriori smallest effect size of interest, $\rho = .20$) was significant, $p = 0.034$, which indicates that for practical purposes the relation was equivalent to zero. That is, the observed correlation, $r = .09$, was not statistically different from zero and it was statistically equivalent to zero.

Notice how much clearer and stronger this results section is with the inclusion of an equivalence test. Note that the 95% CI for the correlation was obtained with the MBESS command: ci.cc(r = .09, n = 267)

9.10 Sample sizes for equivalence testing

9.10.1 Independent *t*-test

9.10.1.1 Standardized units

```
library(TOSTER)

powerTOSTtwo(alpha = 0.05,
             statistical_power = 0.90,
             low_eqbound_d = -0.40,
             high_eqbound_d = 0.40)
```

```
## The required sample size to achieve 90 % power with equivalence bounds of -0.4 and 0.4
##
## [1] 135.3
```

9.10.1.2 Raw units

```
library(TOSTER)

powerTOSTtwo.raw(alpha = 0.05,
                 statistical_power = 0.90,
                 sdpooled = 2.50,
                 low_eqbound = -1,
                 high_eqbound = 1)
```

```
## The required sample size to achieve 90 % power with equivalence bounds of -1 and 1 is 1
##
## [1] 135.3
```

9.10.2 Repeated *t*-test

9.10.2.1 Standardized units

```
library(TOSTER)

powerTOSTpaired(alpha = 0.05,
                statistical_power = 0.90,
                low_eqbound_dz = -0.6325,
                high_eqbound_dz = 0.6325)
```

```
## The required sample size to achieve 90 % power with equivalence bounds of -0.6325 and 0
##
## [1] 27.05
```

9.10.2.2 Raw units

```
library(TOSTER)

powerTOSTpaired.raw(alpha = 0.05,
                    statistical_power = 0.90,
                    sdif = 1.581,
                    low_eqbound = -1,
                    high_eqbound = 1)
```

```
## The required sample size to achieve 90 % power with equivalence bounds of -1 and 1 is 2
##
## [1] 27.05
```

9.10.3 Correlation

```
library(TOSTER)

powerTOSTr(alpha = .05,
           statistical_power = .90,
           low_eqbound_r = -.20,
           high_eqbound_r = .20)
```

```
## The required sample size to achieve 90 % power with equivalence bounds of -0.2 and 0.2
##
## [1] 266.3
```

10

Precision and sample size

10.1 Required

The following CRAN packages must be installed:

Required CRAN Packages
MBESS
TOSTER

10.2 Overview

In the previous two chapter, we reviewed the sample size planning using the traditional NHST approach as well as equivalence testing. In this chapter, we focus on determining the appropriate sample size for your study using the precision approach. Specifically, we review determining the desired sample size based on the effect size and desired confidence interval width.

10.3 Precision

A confidence interval provides an interval estimate of the population parameter. If you took multiple samples from a population and constructed a confidence interval for each sample the confidence intervals would likely all be different. More specifically, each confidence interval is likely to have a different range and width. Over a large number of samples 95% of the confidence intervals would overlap with the population parameter.

Interpretation of confidence intervals can be challenging. **Prior** to collecting your data we say there is a 95% chance a constructed confidence interval will overlap with the population parameter. Once you have collected your data and obtained the end points for your sample confidence interval - interpretation is more difficult. Indeed, a specific interval will either overlap with the population parameter or not; unfortunately, you don't know which is the case. Consequently, once you have a specific confidence interval with end points, based on your data, you should only interpret it as plausible range of values for the population parameter - don't associate the 95% probability with interpretting the confidence interval once you have constructed it.

Imagine you are about to conduct a correlation study ($N = 100$) and you plan to construct a confidence interval around that correlation. At this point, prior to data collection, there is a 95% chance that the confidence interval you construct around the sample correlation will overlap with the population correlation (i.e., the population parameter). You collect your data and find $r = .35$, 95% CI [.16, .51]. Now that you have a specific confidence interval how do you interpret it? You can say the confidence interval is plausible range of the values for the population correlation. You should **not** say there is a 95% chance that population correlation is in the .16 to .51 range. Why not? Once the data is collected and you have a specific interval with end points (e.g., .16 to .51) statisticians think of the interval as overlapping with the population correlation (a probability of 1.00) or not (a probability of 0). Thus, once you have a specific interval, after data collection, the probability that it contains the population parmater is 1.00 or 0 - you just don't know which. So a convienient way to think about the range is by using the word **plausible** which is not associated with a probability. We simply say the confidence interval uses sample data to provide a plausible range of values for the population parameter (e.g., population correlation).

Using confidence intervals for research can, from one perspective, be a fundamental shift in your thinking about how to interpret results. Null hypothesis significance testing allows for a simplistic conclusion - the population effect size may not be exactly zero. This is a form fo categorical decision making: effect / no effect. In contrast, the using confidence intervals to interpret your data is an **estimation** approach: How large is the effect and how uncertain is my estimate of the effect size? Moreover, using the estimation approach makes it a small jump to drawing scientific conclusions using meta-analysis (see **?**). You can learn more about confidence interval / estimation approach by checking out this video workshop[1]. Alternatively, you can check out these general articles **?**, **?**, **?**. **?**.

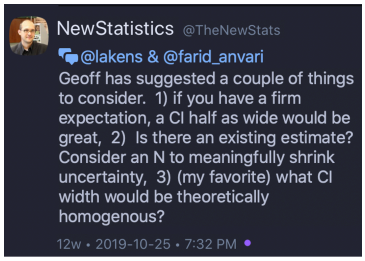
There are two parts to planning using precision:

[1] https://www.psychologicalscience.org/members/new-statistics

1. Determining the effect size of interest (reviewed in previous chapters on smallest effect size of interest).
2. Indicating the desired width of the confidence interval.

Unfortunately, there is little consensus of how to pick the desired width of the confidence interval. Our sense is that the width of the confidence interval should not be larger than the effect size. That, is your uncertainty should not be larger than the effect itself. For example, if you specified a smallest size of interest as $\delta = 0.40$ then you would specify a width of 0.40 as well. Likewise, with correlations, if you specified a smallest effect size of interest as $r = .30$ then you would indicate a desired with of .30 as well.

More stringent advise was offered, indirectly by Geoff Cumming, on social media:



Below I present a series of examples for determining the sample size for your study using the less stringent approach of setting the confidence interval width to be the same as the effect size.

You can read more about planning for precision in **?**.

10.3.1 Precision Independent *t*-test

```
library(MBESS)

ss.aipe.smd(delta = 0.40,
            width = 0.40)
```

```
## [1] 196
```

```
# Returns sample size per group
```

10.3.2 Precision Repeated *t*-test

```
library(MBESS)

ss.aipe.sm(sm = 0.40,
           width = 0.40)
```

```
## [1] 104
```

10.3.3 Precision Correlation

```
library(MBESS)
rho <- .30
rho2 <- rho * rho

ss.aipe.R2(Population.R2 = rho2,
           width = rho2,
           p = 1)
```

```
## [1] "The approximate sample size is given below; you should consider using the addition
## [1] "argument 'verify.ss=TRUE' to ensure the exact sample size value is obtained."
```

```
## $Required.Sample.Size
## [1] 565
```

11

Starting your research

11.1 Required Packages

The data files below are used in this chapter. The files are available at: https://github.com/dstanley4/psyc6060bookdown

Required Data
data_ex_between.csv
data_ex_within.csv
data_food.csv
data_item_scoring.csv
data_item_time.csv

The following CRAN packages must be installed:

Required CRAN Packages
apaTables
Hmisc
janitor
psych
skimr
tidyverse

Important Note: You should NOT use library(psych) at any point! There are major conflicts between the psych package and the tidyverse. We will access the psych package commands by preceding each command with psych:: instead of using library(psych).

11.2 Objective

Due to a number of high profile failures to replicate study results (**?**) it's become increasingly clear that there is a general crisis of confidence in many areas of science (**?**). Statistical (and other) explanations have been offered (**?**) for why it's hard to replicate results across different sets of data. However, scientists are also finding it challenging to recreate the numbers in their own papers using their own data. Indeed, the editor of Molecular Brain asked authors to submit the data used to create the numbers in published papers and found that the wrong data was submitted for 40 out of 41 papers (**?**).

Consequently, some researchers have suggested that it is critical to distinguish between replication and reproducibility (**?**). Replication refers to trying to obtain the same result from a different data sets. Reproducibility refers to trying to obtain the same results from the same data set. Unfortunately, some authors use these two terms interchangeably and fail to make any distinction between them. I encourage you to make the distinction and the use the terms consist with use suggested by (**?**).

It may seem that reproducibility should be a given - but it's not. Correspondingly, there is a trend for journals and authors to adopt Transparency and Openness Promotion (TOP) guidelines[1]. These guidelines involve such things as making your materials, data, code, and analysis scripts available on public repositories so anyone can check your data. A new open science journal rating system has even emerged called the TOP Factor[2].

The idea is not that open science articles are more trustworthy than other types of articles – the idea is that trust doesn't play a role. Anyone can inspect the data using the scripts and data provided by authors. It's really just the same as making your science available for auditing the way financial records can be audited. But just like in the world of business, some people don't like the idea of making it possible for others to audit their work. The problems reported in Molecular Brain (doubtless common to many journals) are likely avoided with open science - because the data and scripts needed to reproduce the statistics in the articles are uploaded prior to publication.

The TOP open science guidelines have made an impact and some newer journals, such as Meta Psychology, have fully embraced open science. Figure **??** shows the header from an article[3] in Meta Psychology that clearly delineates the open science attributes of the article that used computer simulations (in-

[1] https://www.cos.io/our-services/top-guidelines
[2] https://topfactor.org
[3] https://open.lnu.se/index.php/metapsychology/article/view/1630/2266

stead of participant data). Take note that the header even indicates who verified that the analyses in the article were reproducible.

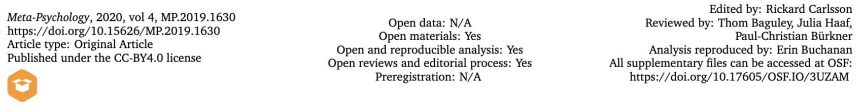


FIGURE 11.1: Open science in an article header

In Canada, the majority of university research is funded by the Federal Government's Tri-Agency (i.e., NSERC, SSHRC, CIHR). The agency has a new draft Data Management Policy[4] in which they state that "*The agencies believe that research data collected with the use of public funds belong, to the fullest extent possible, in the public domain and available for reuse by others.*" The perspective of the funding agency on data ownership differs substantially from that of some researchers who incorrectly believe "they own their data". In Canada at least, the government makes it clear that when tax payers fund research (through the Tri-Agency) the research data is public property. Additionally the Tri-Agency Data Management policy clearly indicates the responsibilities of funded researchers:

"Responsibilities of researchers include:

- incorporating data management best practices into their research;
- developing data management plans to guide the responsible collection, formatting, preservation and sharing of their data throughout the entire life cycle of a research project and beyond;
- following the requirements of applicable institutional and/or funding agency policies and professional or disciplinary standards;
- acknowledging and citing data sets that contribute to their research; and
- staying abreast of standards and expectations of their disciplinary community."

As a result of this perspective on data, it's important that you think about structuring your data for reuse by yourself and others before you collect it. Toward this end, properly documenting your data file and analysis scripts is critical.

[4] https://www.ic.gc.ca/eic/site/063.nsf/eng/h_83F7624E.html

11.3 Begin with the end in mind

In this chapter we will walk you though the steps from data collection, data entry, loading raw data, and the creation of data you will analyze (analytic data) via pre-processing scripts. These steps are outlined in Figure **??**. This figure makes a clear distinction between raw data and analytic data. Raw data refers to the data as you entered it into a spreadsheet or received it from survey software. Analytic data is the data that has been structured and processed so that it is ready for analysis. This pre-processing could include such things as identifying categorical variables to the computer, averaging multiple items into a scale score, and other tasks.

It's critical that you don't think of the analysis of your data as being completely removed from the data collection and data entry choices you make. Poor choices at the data collection and data entry stage can make your life substantially more complicated when it comes time to write the pre-processing script that will convert your raw data to analytic data. The mantra of this chapter is *begin with the end in mind*.

It's difficult to being with the end in mind when you haven't read later chapters. So, here we will be provide you with some general thoughts around different approaches to structuring data files and the naming conventions you can use when creating those data files.

Indeed, in this chapter we strongly advocate that you use a naming convention for file, variable, and column names. This convention will save you hours of hassles and permit easy application of certain tidyverse commands. However, we must stress that although the naming convention we advocate is based on the tidyverse style guide, it is not "right" or "correct" - there are other naming conventions you can use. Any naming convention is better than no naming convention. The naming convention we advocate here will solve many problems. We encourage to use this system for for weeks or months over many projects - until you see the benefits of this system, and correspondingly its shortcomings. After you are well versed in the strengths/weaknesses of the naming conventions used here you may choose to create your own naming convention system.

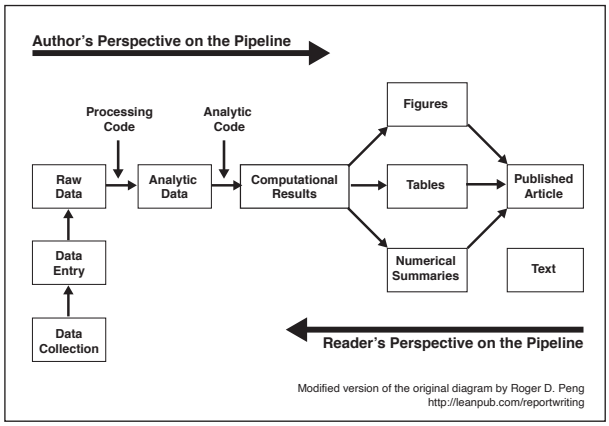


FIGURE 11.2: Data science pipeline by Roger Peng

11.3.1 Structuring data: Obtaining tidy data

When conducting analyses in R it is typically necessary to have data in a format called tidy data (**?**). Tidy data[5], as defined by Hadley, involves (among other requirements) that:

1. Each variable forms a column.
2. Each observation forms a row.

[5] https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

TABLE 11.3: Between participant data entered one row per participant

id	sex	elapsed_time
1	male	40
2	female	35
3	male	38
4	female	33
5	male	42
6	female	36

The tidy data format can be initially challenging for some researchers to understand because it is based on thinking about, and structuring data, in terms of observations/measurements instead of participants. In this section we will describe common approaches to entering animal and human participant data and how they can be done keeping the tidy data requirement in mind. It's not essential that data be entered in a tidy data format but it is essential that you enter data in a manner that makes it easy to later convert data to a tidy data format. When dealing with animal or human participant data it's common to enter data into a spreadsheet. Each row of the spreadsheet is typically used to represent a single participant and each column of the spreadsheet is used to represent a variable.

Between participant data. Consider Table **??** which illustrates between participant data for six human participants running 5 kilometers. The first column is id, which indicates there are six unique participants and provides and identification number for each of them. The second column is sex, which is a variable, and there is one observation per for row, so sex also conforms to the tidy data specification. Finally, there is a last column five_km_time which is a variable with one observation per row – also conforming to tidy data specification. Thus, single occasion between subject data like this conforms to the tidy data specification. There is usually nothing you need to do to convert between participant data (or cross-sectional data) to be in a tidy data format.

Within participant data. Consider Table **??** which illustrates within participant data for six human participants running 5 kilometers - but on three different occasions. The first column is id, which indicates there are six unique participants and provides and identification number for each of them. The second column is sex, which is a variable, and there is one observation per for row, so sex also conforms to the tidy data specification. Next, there are three different columns (march, may, july) each of which contains elapsed time (in minutes) for the runner in a different month. Elapsed run times are spread out over three columns so elapse_time is not in a tidy data format. Moreover, it's not clear from the data file that march, may, and july are levels of a variable called occasion. Nor is it clear that elapsed_times are recorded in

TABLE 11.4: Within participant data entered one row per participant

id	sex	march	may	july
1	male	40	37	35
2	female	35	32	30
3	male	38	35	33
4	female	33	30	28
5	male	42	39	37
6	female	36	33	31

TABLE 11.5: A tidy data version of the within participant data

id	sex	occasion	elapsed_time
1	male	march	40
1	male	may	37
1	male	july	35
2	female	march	35
2	female	may	32
2	female	july	30
3	male	march	38
3	male	may	35
3	male	july	33
4	female	march	33
4	female	may	30
4	female	july	28
5	male	march	42
5	male	may	39
5	male	july	37
6	female	march	36
6	female	may	33
6	female	july	31

each of those columns (i.e., the dependent variable is unknown/not labeled). Although this format is fine as a data entry format it clearly has problems associated with it when it comes time to analyze your data.

Thus, a major problem with entering repeated measures data in the one row per person format is that there are hidden variables in the data and you need insider knowledge to know what the columns represent. That said, this is not necessarily a terrible way to enter your data as long as you have all of this missing information documented in a data code book.

Disadvantages one row per participant	Advantages one row per participant
1) Predictor variable (*occasion*) is hidden and spread over multiple columns 2) Unclear that each month is a level of the predictor variable *occasion* 3) Dependent variable (*elapsed_time*) is not indicated 4) Unclear that *elapsed_time* is the measurement in each month column	1) Easy to enter this way

Fortunately, the problems with Table **??** can be largely resolved by converting the data to the a tidy data format. This can be done with the pivot_long() command that we will learn about later in this chapter. Thus, we can enter the data in the format of Table **??** and later convert it to a tidy data format. After this conversion the data will be appear as in Table **??**. For elapsed_time variable this data is now in the tidy data format. Each row corresponds to a single elapsed_time observed. Each column corresponds to a single variable. Somewhat problematically, however, sex is repeated three times for each person (i.e., over the three rows) - and this can be confusing. However, if the focus in on analyzing elapsed time this tidy data format makes sense. Importantly, there is an id column for each participant so R knows that this information is repeated for each participant and is not confused by repeating the sex designation over three rows. Indirectly, this illustrates the importance of having an id column to indicate each unique participant.

Why did we walk you through this technical treatment of structuring data at this point in time - so that you pay attention to the advice that follows. You can see at this point that you may well need to restructure your data for certain analyses. The ability to do so quickly and easily depends upon following the advice in this chapter around naming conventions for variables and other aspects of your analyses. You can imagine the challenges for converting the data in Figure **??** to the data in Figure **??** by hand. You want to be able to automate that process and others - which is made substantially easier if you follow the forthcoming advice about naming conventions in the tidyverse.

11.4 Data collection considerations

Data can be collected in a wide variety of ways. Regardless of the method of data collection researchers typically come to data in one of two ways: 1) a research assistant enters the data into a spreadsheet type interface, or 2) the data is obtained as the output from computer software (e.g., Qualtrics, SurveyMonkey, Noldus, etc.).

Regardless of the approach, it is critical to name your variables appropriately. For those using software, such as Qualtrics, this means setting up the software to use appropriate variable names PRIOR to data collection - so the exported file has desirable column names. For spreadsheet users, this means setting up the spreadsheet in which the data will be recorded with column names that are amenable to the future analyses you want to conduct.

Although failure to take this thoughtful approach at the data collection stage can be overcome - it is only overcome with substantial manual effort. There-

fore, as noted previously, we strongly encourage you to follow the naming conventions we espouse here when you set up your data recording regime. Additionally, we encourage you to give careful thought in advance to the codes you will use to record missing data.

11.4.1 Naming conventions

To make your life easier down the road, it is critical you set up your spreadsheet or online survey such that it uses a naming convention prior to data collection. The naming conventions suggested here are adapted from the tidyverse style guide[6].

- Lowercase letters only
- If two word column names are necessary, only use the underscore ("_") character to separate words in the name.
- Avoid short decontextualized variable names like q1, q2, q3, etc.
- Do use moderate length column names. Aim to achieve a unique prefix for related columns so that those columns can be selected using the starts_with() command discussed in the previous chapter. Be sure to avoid short two or three letter prefixes for item names. Instead, use unique moderate length item prefixes so that it will easy to select those columns using start_with() such that you don't accidentally get additionally columns you don't want - that have a similar prefix. See the Likert-type item section below for additional details.
- If you have a column name that represents the levels of two repeated measures variables only use the underscore character to separate the levels of the different variables. See within-participant ANOVA section below for details.

11.4.2 Likert-type items

A Likert-type item is typically composed of a statement with which participants are asked to agree or disagree. For example, participants could be asked to indicate the extent to which they agree with a number of statements such as "I like my job". Then tey would be presented with response scale such as: 1 - Strongly Disagree, 2 - Moderately Disagree, 3 - Neutral, 4, Moderately Agree, 5 - Strongly Agree. A common question is, how should I enter the data?

- **Enter numeric responses not labels**. You should enter the numeric value for each item response (e.g., 5) into your data - not the label (e.g., Strongly

[6] https://style.tidyverse.org

Agree). The labels associated with each value can be applied later in a script, if needed.

- **High numbers should be associated with more of the construct being measured.** When designing your survey or data collection tools, it is important that you set the response options appropriately. If your scale measures job satisfaction, it is important that you collect data in a manner that ensures high numbers on the job satisfaction scale indicate high levels of job satisfaction. Therefore, assigning numbers makes sense using the 5-point scale: 1 - Strongly Disagree, 2 - Moderately Disagree, 3 - Neutral, 4, Moderately Agree, 5 - Strongly Agree. With this approach high response numbers indicate more job satisfaction. However, using the opposite scale would not make sense: 1 - Strongly Agree, 2 - Moderately Agree, 3 - Neutral, 4, Moderately Disagree, 5 - Strongly Disagree. With this opposite scale high numbers on a job satisfaction scale would indicate lower levels of job satisfaction - a very confusing situation. Avoid this situation, assign numbers so that higher numbers are associated with more of the construct being measured.
- **Use appropriate item names.** As described in the naming convention section, use moderate length names with different labels for each subscale.
- **Use moderate length column names unique to each subscale**. Imagine you have a survey with an 18-item commitment scale (**?**) composed of three 6-item subscales: affective, normative, and continuance commitment. It would be a poor choice to prefix the labels of all 18 columns in your data with "commit" such that the names would be commit1, commit2, commit3, etc. The problem with this approach is that it fails to distinguish among the three subscales in naming convention; making it impossible to select the items for a single subscale using starts_with(). A better, but still poor choice for a naming convention would be use use a two letter prefix for the three scale such ac, nc, and cc. This would result in names for the columns like ac1, ac2, ac3, etc. This is an improvement because you could apparently (but likely not) select the columns using starts_with("ac"). The problem with these short names is that there could be many columns in data set that start with "ac" beside the affective commitment items. You might want to select the affective commitment items using starts_with("ac"); but you would get all the affective commitment item columns; but also all the columns measuring other variables that also start with "ac". Therefore, it's a good idea to use a moderate length unique prefix for column names. For example, you might use prefixes like affectcom, normcom, and contincom for the three subscales. This would create column names like affectcom1, affectcom2, affectcom3, etc. These column prefixes are unlikely to be duplicated in other places in your column name conventions making it easy to select those columns using a command like starts_with("affectcom").

Indicate in the item name if the item is reversed keyed. Sometimes

with Likert-type items, an item is reverse keyed. For example, on a job satisfaction scale, participants will typically respond to items that reflect job satisfaction using the scale: 1 - Strongly Disagree, 2 - Moderately Disagree, 3 - Neutral, 4, Moderately Agree, 5 - Strongly Agree. Higher numbers indicate more job satisfaction. Sometimes, however, some items will use the same 1 to 5 response scale but be worded in the opposite manner such as "I hate my job". Responding with a 5 to this item would indicate high job *dissatisfaction*. But the columns for job satisfaction items should have high values indicate job satisfaction not job dissatifaction. Consequently, we flag the names of columns with reversed responses (i.e., reverse-key items) so that we know to treat those column differently later. Columns with reverse-keyed items need to be processed by a script so that the values are flipped and scored in the right direction. The procedure for doing so is outlined in the next point.

Indicate in the item name the range for reverse-key items. If an item is reverse keyed, the process for the flipping the scores depends upon the range of a scale. Although 5-point scales are common, any number of points are possible. The process for correcting a reverse-key item depends upon: 1) the number of points on the scale, and 2) the range of the points on the scale. The reverse-key item correction process is different for an item that uses a 5-point scale ranging from 1 to 5 versus from 0 to 4. Both are 5-point scales but your correction process will be different. Therefore, for reverse-key items add a suffix at the end of each item name that indicates an item is reverse keyed and the range of the item. For example, if the third job satisfaction item was reversed keyed on scale using a 1 to 5 response format you might name the item: jobsat3_rev15. The suffix "_rev15" indicates the item is reverse keyed and the range of responses used on the item is 1 to 5. Be sure to set up your survey with this naming convention when you collect your data.

- If you collect items over multiple time points use a prefix with a short code to indicate the time followed by an underscore. For example, if you had a multi-item self-esteem scale you might call the column for the first time "t1_esteem1_rev15". This indicate that you have for time 1 (t1), the first self-esteem item (esteem1) and that item is reverse keyed on a 1 to 5 scale.

11.4.3 ANOVA between

Avoid numerical representation of categorical variables. Don't use 1 or 2 to represent a variable like sex. Use male and female in your spreadsheet - likewise in your survey program. Similarly, for between participant variables like drug_condition don't use 1 or 2 use "drug" and "placebo" but the actually drug name would be even better than the word "drug."

11.4.4 ANOVA within

If you have a study that involves within-participant predictors naming conventions can become examples. When you have a single repeated measures predictor like occasion in the previous running example, it is often necessary to spread the level of that variable over multiple columns (e.g., march, may, july).

When you have multiple repeated-measures predictors the situation is even more complicated. In this case, each column name needs to represent the levels of multiple repeated measures predictors at the time of data entry. For example, imagine you are a food researcher interested in taste ratings (the dependent variable) for various foods and contexts. You have a predictor, food type (i.e., food_type), with three levels (pizza, steak, burger). You have a second predictor, temperature, with two levels (hot, cold). All participants taste all foods at all temperatures. Thus, six columns are required to record taste rating for each participant: pizza_hot, pizza_cold, steak_hot, steak_cold, burger_hot, and burger_cold. Notice how each name contains one level of each predictor variable. The levels by the two predictor variables are separated by a single underscore. This should be the only underscore in the variable name because that underscore will be used by the computer when changing the data to the tidy format. If you had two underscores in a name like "italian_pizza_hot" you would confuse the pivot_longer() command when it attempts to create a tidy version of the data. The computer would think there were three repeated-measures predictors instead of two. Thus, when dealing with repeated measures predictors, only use underscores to separate levels of predictor variables in column names.

11.5 Following the examples

Below we present example scripts transforming raw data to analytic data for various study designs (experimental and survey). These examples illustrate the value of using the naming conventions outlined previously. Don't just read the example - follow along with the projects by creating a separate script for each example. Resist the urge to cut and paste from this document - type the script yourself.

When first learning iPhone/Mac software development, I did so by taking a course at Big Nerd Ranch[7] - yes, that's a real place. They advised in their material (and now book) the following: "We have learned that "going through

[7] https://www.bignerdranch.com

the motions" is much more important than it sounds. Many times we will ask you to start typing in code before you understand it. We realize that you may feel like a trained monkey typing in a bunch of code that you do not fully grasp. But the best way to learn coding is to find and fix your typos. Far from being a drag, this basic debugging is where you really learn the ins and outs of the code. That is why we encourage you to type in the code yourself. You could just download it, but copying and pasting is not programming. We want better for you and your skills.", p. xiv, (**?**). This is excellent advice for a beginning statistician or data scientist as well. And as an aside: if you want to learn iPhone programming you can't go wrong with the Big Nerd Ranch guide!

As you work through this chapter, create your own new script for each example. In light of the above advice, avoid copying and pasting code - type it out; you will be the better for it.

Getting started:

The Class: R Studio in the Cloud Assignment

1. The data should be in the assignment project automatically. Just start the assignment.

For everyone in the class, that's it.

For those of you not in the class, and reading this work, see the two options below:

R Studio Cloud, custom project

1. Create a new Project using the web interface
2. Upload all the example data files into the project. The data files needed are listed at the beginning of this chapter. The upload button can be found on the Files tab.

R Studio Computer, custom project

1. Create a folder on your computer for the example
2. Place all the example data files in that folder. The data files needed are listed at the beginning of this chapter.
3. Use the menu item File > New Project… to start the project
4. On the window that appears select "Existing Directory"
5. On the next screen, press the "Browse" button and find/select the folder with your data

6. Press the Create Project Button

Regardless of whether your are working from the cloud, or locally, you should now have an R Studio project with your data files in it.

We anticipate that many people will doubtless want to refer back to an encapsulated set of instructions for each design. Therefore the example for each design is written in a way that it stands alone. A consequence of this approach is that there is some redundancy in the code across examples. We see this a strength - because readers will see the commonalities across differ types of designs.

As you make a script for each example:

- Recall the instruction from Chapter 1 about putting the date and your name in the script via comments.
- Recall the instruction from Chapter 1 about running library(tidyverse) before you type the rest of each script - this provides you with tidyverse autocomplete for the script.
- After you type each new block of code in an example, save your script.
- After you type each new block of code in an example, do two additional things: 1) Session Restart R, 2) Run your script using Source with Echo.

11.6 Entering data into spreadsheets

The first example uses a data file data_ex_between.csv that corresponds to a fictitious example where we recorded the run times for a number of male and female participants. How did we create this data file? We used a spreadsheet to enter the data, as illustrated in Figure **??**. Programs like Microsoft Excel and Google Sheets are good options for entering data.

id	sex	elapsed_time
1	male	40
2	female	35
3	male	38
4	female	33
5	male	42
6	female	36

FIGURE 11.3: Spreadsheet entry of running data

The key to using these types of programs is to save the data as a .csv file when

you are done. CSV is short for Comma Separated Values. After entering the data in Figure **??** we saved it as data_ex_between.csv. There is no need to do so, but if you were to open this file in a text editor (such as TextEdit on a Mac or Notepad on Windows) you would see the information displayed in Figure **??**. You can see there is one row per person and the columns are created by separating each values by a comma; hence, comma separated values.

```
id,sex,elapsed_time
1,male,40
2,female,35
3,male,38
4,female,33
5,male,42
6,female,36
```

FIGURE 11.4: Text view of CSV data

There are many ways to save data, but the CSV data is one of the better ones because it is a non-proprietary format. Some software, such as SPSS, uses a proprietary format (e.g., .sav for SPSS) this makes it challenging to access that data if you don't have that (often expensive) software. One of our goals as scientists is to make it easy for others to audit our work - that allows science to be self-correcting. Therefore, choose an open format for your data like .csv.

11.7 Experiment: Between

This section outlines a workflow appropriate for when you plan to a conduct independent-groups *t*-test or a between-participants ANOVA.

To Begin:

- Use the Files tab to confirm you have the data: data_ex_between.csv
- Start a new script for this example. Don't forget to start the script name with "script_".

As noted previously, these data correspond to a design where the researcher is interested in comparing run times (elapsed_time) based on sex (male/female).

```
# Date: YYYY-MM-DD
# Name: your name here
# Example: Between-participant experiment

# Load data
```

```
library(tidyverse)

my_missing_value_codes <- c("-999", "", "NA")

raw_data_between <- read_csv(file = "data_ex_between.csv",
                     na = my_missing_value_codes)
```

We load the initial data into a raw_data_between data set but immediately make a copy that we will work with called analytic_data_between. It's good to keep a copy of the raw data for reference in the event that you encounter problems.

```
analytic_data_between <- raw_data_between
```

After loading the data we do initial cleaning to remove empty row/columns and ensure proper naming for columns:

```
library(janitor)

# Initial cleaning
analytic_data_between <- analytic_data_between %>%
  remove_empty("rows") %>%
  remove_empty("cols") %>%
  clean_names()
```

You can confirm the column names follow our naming convention with the glimpse() command.

```
glimpse(analytic_data_between)
```

```
## Rows: 6
## Columns: 3
## $ id           <dbl> 1, 2, 3, 4, 5, 6
## $ sex          <chr> "male", "female", "male", "female"...
## $ elapsed_time <dbl> 40, 35, 38, 33, 42, 36
```

11.7.1 Creating factors

Following initial cleaning, we identify categorical variables as factors. If you plan to conduct an ANOVA - it's critical that all predictor variables are converted to factors. Inspect the glimpse() output - if you followed our data entry

naming conventions, categorical variables should be of the type character. We have one variable, sex, that is a categorical variable of type character (i.e., chr). The participant id column is categorical as well, but of type double (i.e., dbl) which is a numeric column.

```
glimpse(analytic_data_between)
```

```
## Rows: 6
## Columns: 3
## $ id           <dbl> 1, 2, 3, 4, 5, 6
## $ sex          <chr> "male", "female", "male", "female"...
## $ elapsed_time <dbl> 40, 35, 38, 33, 42, 36
```

You can quickly convert all character columns to factors using the code below. In this case, the code just converts the sex column to a factor. Because there is only one column (sex) being converted to a factor, we could have treated it the same way as the id column below. However, we use this code because of its broad applicability to many scripts.

```
analytic_data_between <- analytic_data_between %>%
  mutate(across(.cols = where(is.character),
                .fns = as_factor))
```

The participant identification number in the id column is a numeric column, so it was not converted by the above code. The id column is converted to a factor with the code below.

```
analytic_data_between <- analytic_data_between %>%
  mutate(id = as_factor(id))
```

You can ensure both the sex and id columns are now factors using the glimpse() command.

```
glimpse(analytic_data_between)
```

```
## Rows: 6
## Columns: 3
## $ id           <fct> 1, 2, 3, 4, 5, 6
## $ sex          <fct> male, female, male, female, male, ...
## $ elapsed_time <dbl> 40, 35, 38, 33, 42, 36
```

This example is so small it's clear you didn't miss converting any columns to factors. In general, however, at this point you should inspect the output

of the glimpse() command and make sure you have converted all categorical variables to factors - especially those you will use as predictors.

11.7.2 Factor screening

Inspect the levels of each factor carefully. Make sure the factor levels of each variable are correct. Examine spelling and look for additional unwanted levels. For example, you wouldn't want to have the following levels for sex: male, mmale, female. Obviously, mmale is an incorrectly typed version of male. Scan all the factors in your data for erroneous factor levels. The code below displays the factor levels:

```
analytic_data_between %>%
  select(where(is.factor)) %>%
  summary()
```

```
##  id        sex
##  1:1   male  :3
##  2:1   female:3
##  3:1
##  4:1
##  5:1
##  6:1
```

The order of the levels influences how graphs are generated. In these data, the sex column has two levels: male and female in that order. The code below adjusts the order of the sex variable because we want the x-axis of a future graph to display columns in the left to right order: female, male.

```
analytic_data_between <- analytic_data_between %>%
  mutate(sex = fct_relevel(sex,
                           "female",
                           "male"))
```

You can see the new order of the factor levels with summary():

```
analytic_data_between %>%
  select(where(is.factor)) %>%
  summary()
```

```
##  id        sex
##  1:1   female:3
##  2:1   male  :3
```

```
##  3:1
##  4:1
##  5:1
##  6:1
```

11.7.3 Numeric screening

For numeric variables, you should search for impossible values. For example, in the context of this example you want to ensure that none of the elapsed_times are impossible, or so large they appear to be data entry errors. One option for doing so is the summary command again. This time, however, we use "is.numeric" in the where() command.

```
analytic_data_between %>%
  select(where(is.numeric)) %>%
  summary()
```

```
##   elapsed_time
##  Min.   :33.0
##  1st Qu.:35.2
##  Median :37.0
##  Mean   :37.3
##  3rd Qu.:39.5
##  Max.   :42.0
```

Scan the min and max values to ensure there are not any impossible values. If necessary, go back to the original data source and fix these impossible values. Alternatively, you might need to change them to missing values (i.e., NA values).

In this example all the values are reasonable values. However, if we discovered an out of range value (or values) for elapsed time, we could convert those values to missing values with the code below. This code changes (i.e., mutates) a value in the elapsed_time column to become NA (not available or missing) if that value is less than zero. If the value is greater than, or equal to zero, it stays the same. Note that when using this command we have to be very specific in terms of specifying our missing value. It usually needs to be one of NA_real_ or NA_character_. For numeric columns use NA_real_ and for character columns use NA_character_.

```
analytic_data_between <- analytic_data_between  %>%
    mutate(elapsed_time = case_when(
      elapsed_time < 0 ~ NA_real_,
      elapsed_time >= 0 ~ elapsed_time))
```

Once you are done numeric screening, the data is ready for analysis.

11.8 Experiment: Within one-way

This section outlines a workflow appropriate for when you have a repeated measures design with a single repeated measures predictor. The data corresponds to a design where the researcher is interested in comparing run times (elapsed_time) across three different occasions (march/may/july).

To Begin:

- Use the Files tab to confirm you have the data: data_ex_within.csv
- Start a new script for this example. Don't forget to start the script name with "script_".

```
# Date: YYYY-MM-DD
# Name: your name here
# Example: Within-participant experiment

# Load data
library(tidyverse)

my_missing_value_codes <- c("-999", "", "NA")

raw_data_within <- read_csv(file = "data_ex_within.csv",
                     na = my_missing_value_codes)
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   sex = col_character(),
##   march = col_double(),
##   may = col_double(),
##   july = col_double()
## )
```

We load the initial data into raw_data_within but immediately make a copy that we will work with called analytic_data_within. It's good to keep a copy of the raw data for reference if you encounter problems.

```
analytic_data_within <- raw_data_within
```

After loading the data we do initial cleaning to remove empty row/columns and ensure proper naming for columns:

```
library(janitor)

# Initial cleaning
analytic_data_within <- analytic_data_within %>%
  remove_empty("rows") %>%
  remove_empty("cols") %>%
  clean_names()
```

You can confirm the column names following our naming convention with the glimpse command - and see the data type for each column.

```
glimpse(analytic_data_within)
```

```
## Rows: 6
## Columns: 5
## $ id    <dbl> 1, 2, 3, 4, 5, 6
## $ sex   <chr> "male", "female", "male", "female", "male...
## $ march <dbl> 40, 35, 38, 33, 42, 36
## $ may   <dbl> 37, 32, 35, 30, 39, 33
## $ july  <dbl> 35, 30, 33, 28, 37, 31
```

11.8.1 Creating factors

Following initial cleaning, we identify categorical variables as factors. If you plan to conduct an ANOVA - it's critical that all predictor variables are converted to factors. Inspect the glimpse() output - if you followed our data entry naming conventions, categorical variables should be of the type character.

```
glimpse(analytic_data_within)
```

```
## Rows: 6
## Columns: 5
## $ id    <dbl> 1, 2, 3, 4, 5, 6
## $ sex   <chr> "male", "female", "male", "female", "male...
## $ march <dbl> 40, 35, 38, 33, 42, 36
## $ may   <dbl> 37, 32, 35, 30, 39, 33
```

```
## $ july  <dbl> 35, 30, 33, 28, 37, 31
```

We have one variable, sex, that is a categorical variable of type character (i.e., chr). The participant id column is categorical as well, but of type double (i.e., dbl) which is a numeric column.

You can quickly convert all character columns to factors using the code below. In this case, this just converts the sex column to a factor. Because there is only one column (sex) being converted to a factor, we could have treated it the same was as the id column below. However, we use this code because of its broad applicability to many scripts.

```
analytic_data_within <- analytic_data_within %>%
  mutate(across(.cols = where(is.character),
                .fns = as_factor))
```

The participant identification number in the id column is a numeric column, so it was not converted by the above code. The id column is converted to a factor with the code below.

```
analytic_data_within <- analytic_data_within %>%
  mutate(id = as_factor(id))
```

You can ensure both the sex and id columns are now factors using the glimpse() command.

```
glimpse(analytic_data_within)
```

```
## Rows: 6
## Columns: 5
## $ id    <fct> 1, 2, 3, 4, 5, 6
## $ sex   <fct> male, female, male, female, male, female
## $ march <dbl> 40, 35, 38, 33, 42, 36
## $ may   <dbl> 37, 32, 35, 30, 39, 33
## $ july  <dbl> 35, 30, 33, 28, 37, 31
```

This example is so small it's clear you didn't miss converting any columns to factors. In general, however, at this point you should inspect the output of the glimpse() command and make sure you have converted all categorical variables to factors - especially those you will use as predictors.

11.8.2 Factor screening

Inspect the levels of each factor carefully. Make sure the factor levels of each variable are correct. Examine spelling and look for additional unwanted levels. For example, you wouldn't want to have the following levels for sex: male, mmale, female. Obviously, mmale is an incorrectly typed version of male. Scan all the factors in your data for erroneous factor levels. The code below displays the factor levels:

```
analytic_data_within %>%
  select(where(is.factor)) %>%
  summary()
```

```
##  id        sex
##  1:1   male  :3
##  2:1   female:3
##  3:1
##  4:1
##  5:1
##  6:1
```

The order of the levels influences how graphs are generated. In these data, the sex column has two levels: male and female in that order. The code below adjusts the order of the sex variable because we want the x-axis of a future graph to display columns in the left to right order: female, male.

```
analytic_data_within <- analytic_data_within %>%
  mutate(sex = fct_relevel(sex,
                           "female",
                           "male"))
```

You can see the new order of the factor levels with summary():

```
analytic_data_within %>%
  select(where(is.factor)) %>%
  summary()
```

```
##  id        sex
##  1:1   female:3
##  2:1   male  :3
##  3:1
##  4:1
##  5:1
##  6:1
```

11.8.3 Numeric screening

For numeric variables, it is important to find and remove impossible values. For example, in the context of this example you want to ensure none of the elapsed_times are impossible (i.e., negative) or clearly data entry errors.

Because we have several numeric columns that we are screening, we use the skim() command from the skimr package. The skim() command quickly provides basic descriptive statistics. In the output for this command there are also several columns that begin with p: p0, p25, p50, p75, and p100 (p25 and p75 omitted in output due to space). These columns correspond to the 0th, 25th, 50th, 75th, and 100th percentiles, respectively. The minimum and maximum values for the data column are indicated under the p0 and p100 labels. The median is the 50th percentile (p50). The interquartile range is the range between p25 and p75.

```
library(skimr)

analytic_data_within %>%
  select(where(is.numeric)) %>%
  skim()
```

```
##   skim_variable n_missing  mean   sd p0 p50 p100
## 1         march         0 37.33 3.33 33  37   42
## 2           may         0 34.33 3.33 30  34   39
## 3          july         0 32.33 3.33 28  32   37
```

Scan the minimum and maximum values (p0 and p100) to ensure there are not any impossible values. If necessary, go back to the original data source and fix these impossible values. Alternatively, you might need to change them to missing values (i.e., NA values).

In this example all the values are reasonable values. However, if we discovered an out of range value (or values) for elapsed time we could convert those values to missing values with the code below. This code changes (i.e., mutates) a value in the march column to become NA (not available or missing) if that value is less than zero. If the value is greater than or equal to zero, it stays the same. Note that when using this command we have to be very specific in terms of specifying our missing value. It usually needs to be one of NA_real_ or NA_character_. For numeric columns use NA_real_ and for character columns use NA_character_.

```
analytic_data_within <- analytic_data_within  %>%
    mutate(march = case_when(
```

```
      march < 0 ~ NA_real_,
      march >= 0 ~ march))
```

11.8.4 Pivot to tidy data

The analytic data in it's current form does not conform to the tidy data specification. Inspect the data with the print() command. Notice thatthere is not a column for occasion (with levels march/may/july). Instead, there are three columns each of which represents a level of occasion. The levels of occasion are spread across three columns called march, may, and july. Each of these columns contains elapsed time for participants in that month.

```
print(analytic_data_within)
```

```
## # A tibble: 6 x 5
##   id    sex    march   may  july
##   <fct> <fct>  <dbl> <dbl> <dbl>
## 1 1     male      40    37    35
## 2 2     female    35    32    30
## 3 3     male      38    35    33
## 4 4     female    33    30    28
## 5 5     male      42    39    37
## 6 6     female    36    33    31
```

The pivot_longer() command below coverts our data to the tidy data format. In this command we specify the columns march, may, and july are all levels of a single variable called occasion. We specify the columns involved with the cols argument. The code march:july after the cols argument selects the march column, the july column, and all the columns in-between. Each column contains elapsed times at level of the variable occasion. The names_to argument is used to indicate that a new column called occasion should be created to hold the different months. The value_to argument is used to indicate that a new column called elapsed_time should created to hold all the values from the march, may, and july columns.

```
analytic_data_within_tidy <- analytic_data_within %>%
  pivot_longer(cols = march:july,
               names_to = "occasion",
               values_to = "elapsed_time"
  )
```

You can see the data in the new format below.

```
print(analytic_data_within_tidy)
```

```
## # A tibble: 18 x 4
##    id    sex    occasion elapsed_time
##    <fct> <fct>  <chr>           <dbl>
##  1 1     male   march              40
##  2 1     male   may                37
##  3 1     male   july               35
##  4 2     female march              35
##  5 2     female may                32
##  6 2     female july               30
##  7 3     male   march              38
##  8 3     male   may                35
##  9 3     male   july               33
## 10 4     female march              33
## 11 4     female may                30
## 12 4     female july               28
## 13 5     male   march              42
## 14 5     male   may                39
## 15 5     male   july               37
## 16 6     female march              36
## 17 6     female may                33
## 18 6     female july               31
```

Notice that the new column occasion is of the type character. We need it to be a factor. So use the code below to do so:

```
analytic_data_within_tidy <- analytic_data_within_tidy %>%
  mutate(occasion = as_factor(occasion))
```

You can confirm that occasion is now a factor with the glimpse() command. Once this is complete, you are done preparing your one-way within participant analytic data.

```
glimpse(analytic_data_within_tidy)
```

```
## Rows: 18
## Columns: 4
## $ id           <fct> 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4...
## $ sex          <fct> male, male, male, female, female, ...
## $ occasion     <fct> march, may, july, march, may, july...
## $ elapsed_time <dbl> 40, 37, 35, 35, 32, 30, 38, 35, 33...
```

You now have two data sets analytic_data_within and ana-

lytic_data_within_tidy. You can calculate descriptive statistics, correlations and general cross-sectional analyses using the analytic_data_within data set. If you want to conduct a repeated measures ANOVA you use the analytic_data_within_tidy data set. Both data sets are now ready for analysis.

11.9 Experiment: Within N-way

This section outlines a workflow appropriate for when you have a repeated measures design with multiple repeated measures predictors. The data corresponds to a design where the researcher is interested in assessing the taste of food as a function of food type (pizza/steak/burger) and temperature (hot/cold).

To Begin:

- Use the Files tab to confirm you have the data: data_food.csv
- Start a new script for this example. Don't forget to start the script name with "script_".

```
# Date: YYYY-MM-DD
# Name: your name here
# Example: 2-way within-participant experiment

# Load data
library(tidyverse)

my_missing_value_codes <- c("-999", "", "NA")

raw_data_within_nway <- read_csv(file = "data_food.csv",
                     na = my_missing_value_codes)
```

We load the initial data into raw_data_within_nway but immediately make a copy that we will work with called analytic_data_within_nway. It's good to keep a copy of the raw data for reference if you encounter problems.

```
analytic_data_within_nway <- raw_data_within_nway
```

After loading the data we do initial cleaning to remove empty row/columns and ensure proper naming for columns:

```
library(janitor)

# Initial cleaning
analytic_data_within_nway <- analytic_data_within_nway %>%
  remove_empty("rows") %>%
  remove_empty("cols") %>%
  clean_names()
```

You can confirm the column names following our naming convention with the glimpse command - and see the data type for each column.

```
glimpse(analytic_data_within_nway)
```

```
## Rows: 6
## Columns: 8
## $ id          <dbl> 1, 2, 3, 4, 5, 6
## $ sex         <dbl> 1, 2, 1, 2, 1, 2
## $ pizza_hot   <dbl> 7, 8, 7, 8, 7, 9
## $ pizza_cold  <dbl> 6, 7, 5, 7, 6, 7
## $ steak_hot   <dbl> 6, 6, 7, 7, 8, 7
## $ steak_cold  <dbl> 3, 3, 4, 5, 7, 8
## $ burger_hot  <dbl> 7, 8, 7, 8, 7, 8
## $ burger_cold <dbl> 4, 3, 3, 3, 2, 5
```

11.9.1 Creating factors

Following initial cleaning, we identify categorical variables as factors. If you plan to conduct an ANOVA - it's critical that all predictor variables are converted to factors. In this example, there are two categorical variables id and sex, but both are represented numerically. As revealed by the glimpse() output.

```
glimpse(analytic_data_within_nway)
```

```
## Rows: 6
## Columns: 8
## $ id          <dbl> 1, 2, 3, 4, 5, 6
## $ sex         <dbl> 1, 2, 1, 2, 1, 2
## $ pizza_hot   <dbl> 7, 8, 7, 8, 7, 9
## $ pizza_cold  <dbl> 6, 7, 5, 7, 6, 7
## $ steak_hot   <dbl> 6, 6, 7, 7, 8, 7
## $ steak_cold  <dbl> 3, 3, 4, 5, 7, 8
```

```
## $ burger_hot  <dbl> 7, 8, 7, 8, 7, 8
## $ burger_cold <dbl> 4, 3, 3, 3, 2, 5
```

We convert both the sex and id columns to factors with the mutate() command below:

```
analytic_data_within_nway <- analytic_data_within_nway %>%
  mutate(id = as_factor(id),
         sex = as_factor(sex))
```

The sex column is a factor but we have to tell the computer that 1 indicates male and 2 indicates female.

```
analytic_data_within_nway <- analytic_data_within_nway %>%
  mutate(sex = fct_recode(sex,
                          male = "1",
                          female = "2"))
```

You can ensure all of these columns are now factors using the glimpse() command.

```
glimpse(analytic_data_within_nway)
```

```
## Rows: 6
## Columns: 8
## $ id          <fct> 1, 2, 3, 4, 5, 6
## $ sex         <fct> male, female, male, female, male, f...
## $ pizza_hot   <dbl> 7, 8, 7, 8, 7, 9
## $ pizza_cold  <dbl> 6, 7, 5, 7, 6, 7
## $ steak_hot   <dbl> 6, 6, 7, 7, 8, 7
## $ steak_cold  <dbl> 3, 3, 4, 5, 7, 8
## $ burger_hot  <dbl> 7, 8, 7, 8, 7, 8
## $ burger_cold <dbl> 4, 3, 3, 3, 2, 5
```

Inspect the output of the glimpse() command and make sure you have converted all categorical variables to factors - especially those you will use as predictors. As noted in the previous examples, its common to have additional columns that are categorical predictors but appear in the glimpse() output as being of the type character. That is not the case in these data, but if it were the command below would turn them into factors:

```
analytic_data_within_nway <- analytic_data_within_nway %>%
  mutate(across(.cols = where(is.character),
                .fns = as_factor))
```

11.9.2 Factor screening

Inspect the levels of each factor carefully. Make sure the factor levels of each variable are correct. Examine spelling and look for additional unwanted levels. For example, you wouldn't want to have the following levels for sex: male, mmale, female. Obviously, mmale is an incorrectly typed version of male. Scan all the factors in your data for erroneous factor levels. The code below displays the factor levels:

```
analytic_data_within_nway %>%
  select(where(is.factor)) %>%
  summary()
```

```
##  id        sex
##  1:1   male  :3
##  2:1   female:3
##  3:1
##  4:1
##  5:1
##  6:1
```

The order of the levels influences how graphs are generated. In these data, the sex column has two levels: male and female in that order. The code below adjusts the order of the sex variable because we want the x-axis of a future graph to display columns in the left to right order: female, male.

```
analytic_data_within_nway <- analytic_data_within_nway %>%
  mutate(sex = fct_relevel(sex,
                           "female",
                           "male"))
```

You can see the new order of the factor levels with summary():

```
analytic_data_within_nway %>%
  select(where(is.factor)) %>%
  summary()
```

```
##  id        sex
##  1:1   female:3
##  2:1   male  :3
##  3:1
##  4:1
##  5:1
##  6:1
```

11.9.3 Numeric screening

For numeric variables, it's important find and remove impossible values. For example, in the context of this example you want to ensure none of the taste ratings the six columns (pizza_hot, pizza_cold, steak_hot, steak_cold, burger_hot, and burger_cold) are outside the range of the 1 to 10 rating scale.

Because we have several numeric columns that we are screening, we use the skim() command from the skimr package. The skim() command quickly provides basic descriptive statistics. In the output for this command there are also several columns that begin with p: p0, p25, p50, p75, and p100 (p25 and p75 omitted in output due to space). These columns correspond to the 0th, 25th, 50th, 75th, and 100th percentiles, respectively. The minimum and maximum values for the data column are indicated under the p0 and p100 labels. The median is the 50th percentile (p50). The interquartile range is the range between p25 and p75.

```
library(skimr)

analytic_data_within_nway %>%
  select(where(is.numeric)) %>%
  skim()
```

```
##   skim_variable n_missing mean   sd p0 p50 p100
## 1     pizza_hot         0 7.67 0.82  7 7.5    9
## 2    pizza_cold         0 6.33 0.82  5 6.5    7
## 3     steak_hot         0 6.83 0.75  6 7.0    8
## 4    steak_cold         0 5.00 2.10  3 4.5    8
## 5    burger_hot         0 7.50 0.55  7 7.5    8
## 6   burger_cold         0 3.33 1.03  2 3.0    5
```

Scan the minimum and maximum values (p0 and p100) to ensure there are not any impossible values. That is, ensure all values are inside the 1 to 10 range for the dependent variable. If necessary, get back to the original data source and fix these impossible values. Alternatively, you might need to change them to missing values (i.e., NA values).

In this example all the values are reasonable values. However, if we discovered an out-of-range value (or values) for elapsed time we could convert those values to missing values with the code below. This code changes (i.e., mutates) a value in the pizza_hot column to become NA (not available or missing) if that value is outside the 1 to 10 range of the rating scale. Note that when using this command we have to be very specific in terms of specifying our missing value. It usually needs to be one of NA_real_ or NA_character_. For numeric columns use NA_real_ and for character columns use NA_character_.

```
# Values lower than 1 are converted to missing values
analytic_data_within_nway <- analytic_data_within_nway  %>%
    mutate(pizza_hot = case_when(
      pizza_hot < 1  ~ NA_real_,
      pizza_hot >= 1 ~ pizza_hot))

# Values greater than 10 are converted to missing values
analytic_data_within_nway <- analytic_data_within_nway  %>%
    mutate(pizza_hot = case_when(
      pizza_hot > 10  ~ NA_real_,
      pizza_hot <= 10 ~ pizza_hot))
```

11.9.4 Pivot to tidy data

The analytic data in its current form does not conform to the tidy data specification. Inspect the data with the print() command. Notice that columns do not exist for temperature (with levels hot/cold) or food type (with levels pizza/steak/burger). Instead, there are six columns that are combinations of the levels of these variables (i.e., pizza_hot, pizza_cold, steak_hot, steak_cold, burger_hot, and burger_cold). Each of these columns contains taste ratings on a 1 to 10 point scale.

```
print(analytic_data_within)
```

```
## # A tibble: 6 x 5
##   id    sex    march   may  july
##   <fct> <fct>  <dbl> <dbl> <dbl>
## 1 1     male      40    37    35
## 2 2     female    35    32    30
## 3 3     male      38    35    33
## 4 4     female    33    30    28
## 5 5     male      42    39    37
## 6 6     female    36    33    31
```

We need to restructure the data into the tidy data format so that we have a food_type column and a temperature column to properly represent these predictors. As well, we need a column that contains all of the taste ratings that is clearly labeled taste. Doing all of these things will ensure we have one variable per column and one observation per row - consistent with the requirements of tidy data. The pivot_longer() command below coverts our data to the tidy data format.

In this command we specify the columns march, may, and july are all levels

of a single variable called occasion. We specify the columns involved with the cols argument. The code pizza_hot:burger_cold after the cols argument selects the pizza_hot column, the burger_cold column, and all the columns in between. Each column contains taste ratings at a combination of the levels for the variables food_type and temperature. The names_to argument is used to indicate that two new columns should be created to represent food and temperature. Notice the order food then temperature. This is consistent with our naming convention; in the column name pizza_hot the food_type is specified before temperature. The value_to argument is used to indicate that a new column called taste should created to hold all the values from the pizza_hot, pizza_cold, steak_hot, steak_cold, burger_hot, and burger_cold columns.

```
analytic_data_nway_tidy <- analytic_data_within_nway %>%
  pivot_longer(cols = pizza_hot:burger_cold,
               names_to = c("food_type", "temperature"),
               names_sep = "_",
               values_to = "taste"
  )
```

You can see the data in the new format below.

```
print(analytic_data_nway_tidy)
```

```
## # A tibble: 36 x 5
##    id    sex    food_type temperature taste
##    <fct> <fct>  <chr>     <chr>       <dbl>
##  1 1     male   pizza     hot             7
##  2 1     male   pizza     cold            6
##  3 1     male   steak     hot             6
##  4 1     male   steak     cold            3
##  5 1     male   burger    hot             7
##  6 1     male   burger    cold            4
##  7 2     female pizza     hot             8
##  8 2     female pizza     cold            7
##  9 2     female steak     hot             6
## 10 2     female steak     cold            3
## # ... with 26 more rows
```

Notice that the new column occasion is of the type character. We need it to be a factor. Use the code below to do so:

```
analytic_data_nway_tidy <- analytic_data_nway_tidy %>%
  mutate(food = as_factor(food_type),
         temperature = as_factor(temperature))
```

You can confirm that occasion is now a factor with the glimpse() command. Once this is complete, you are done preparing your one-way within participant analytic data.

```
glimpse(analytic_data_nway_tidy)
```

```
## Rows: 36
## Columns: 6
## $ id          <fct> 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,...
## $ sex         <fct> male, male, male, male, male, male,...
## $ food_type   <chr> "pizza", "pizza", "steak", "steak",...
## $ temperature <fct> hot, cold, hot, cold, hot, cold, ho...
## $ taste       <dbl> 7, 6, 6, 3, 7, 4, 8, 7, 6, 3, 8, 3,...
## $ food        <fct> pizza, pizza, steak, steak, burger,...
```

You now have two data sets analytic_data_within_nway and analytic_data_nway_tidy. You can calculate descriptive statistics, correlations and general cross-sectional analyses using the analytic_data_within_nway data set. If you want to conduct a repeated measures ANOVA you use the analytic_data_nway_tidy data set. Both data sets are now ready for analysis.

11.10 Surveys: Single Occassion

This section outlines a workflow appropriate for when you have cross-sectional single occasion survey data. The data corresponds to a design where the researcher has measured, age, sex, eye color, self-esteem, and job satisfaction. Two of these, self-esteem and job satisfaction, were measured with multi-item scales with reverse-keyed items.

To Begin:

- Use the Files tab to confirm you have the data: data_item_scoring.csv
- Start a new script for this example. Don't forget to start the script name with "script_".

```
# Date: YYYY-MM-DD
# Name: your name here
# Example: Single occasion survey

# Load data
library(tidyverse)
```

```
my_missing_value_codes <- c("-999", "", "NA")

raw_data_survey <- read_csv(file = "data_item_scoring.csv",
                     na = my_missing_value_codes)
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   age = col_double(),
##   sex = col_character(),
##   eye_color = col_character(),
##   esteem1 = col_double(),
##   esteem2 = col_double(),
##   esteem3 = col_double(),
##   esteem4 = col_double(),
##   esteem5_rev15 = col_double(),
##   jobsat1 = col_double(),
##   jobsat2_rev15 = col_double(),
##   jobsat3 = col_double(),
##   jobsat4 = col_double(),
##   jobsat5 = col_double()
## )
```

We load the initial data into a raw_data_survey but immediately make a copy we will work with called analytic_data_survey. It's good to keep a copy of the raw data for reference if you encounter problems.

```
analytic_data_survey <- raw_data_survey
```

Remove empty row and columns from your data using the remove_empty_cols() and remove_empty_rows(), respectively. As well, clean the names of your columns to ensure they conform to tidyverse naming conventions.

```
library(janitor)

# Initial cleaning
analytic_data_survey <- analytic_data_survey %>%
  remove_empty("rows") %>%
  remove_empty("cols") %>%
  clean_names()
```

You can confirm the column names following our naming convention with the glimpse command - and see the data type for each column.

```
glimpse(analytic_data_survey)
```

```
## Rows: 300
## Columns: 14
## $ id            <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...
## $ age           <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 1...
## $ sex           <chr> "male", "female", "male", "female...
## $ eye_color     <chr> "blue", "brown", "hazel", "blue",...
## $ esteem1       <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, ...
## $ esteem2       <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, ...
## $ esteem3       <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4,...
## $ esteem4       <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA,...
## $ esteem5_rev15 <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2, 3...
## $ jobsat1       <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, ...
## $ jobsat2_rev15 <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3,...
## $ jobsat3       <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3,...
## $ jobsat4       <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, N...
## $ jobsat5       <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4,...
```

11.10.1 Creating factors

Following initial cleaning, we identify categorical variables as factors. If you plan to conduct an ANOVA - it's critical that all predictor variables are converted to factors. Inspect the glimpse() output - if you followed our data entry naming conventions, categorical variables should be of the type character.

```
glimpse(analytic_data_survey)
```

```
## Rows: 300
## Columns: 14
## $ id            <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...
## $ age           <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 1...
## $ sex           <chr> "male", "female", "male", "female...
## $ eye_color     <chr> "blue", "brown", "hazel", "blue",...
## $ esteem1       <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, ...
## $ esteem2       <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, ...
## $ esteem3       <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4,...
## $ esteem4       <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA,...
## $ esteem5_rev15 <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2, 3...
## $ jobsat1       <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, ...
```

```
## $ jobsat2_rev15 <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3,...
## $ jobsat3       <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3,...
## $ jobsat4       <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, N...
## $ jobsat5       <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4,...
```

We have two variables, sex and eye_color, that are categorical variable of type character (i.e., chr). The participant id column is categorical as well, but of type double (i.e., dbl) which is a numeric column. You can quickly convert all character columns to factors using the code below:

```
analytic_data_survey <- analytic_data_survey %>%
  mutate(across(.cols = where(is.character),
                .fns = as_factor))
```

The participant identification number in the id column is a numeric column, so we have to handle that column on its own.

```
analytic_data_survey <- analytic_data_survey %>%
  mutate(id = as_factor(id))
```

You can ensure all of these columns are now factors using the glimpse() command.

```
glimpse(analytic_data_survey)
```

```
## Rows: 300
## Columns: 14
## $ id            <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...
## $ age           <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 1...
## $ sex           <fct> male, female, male, female, male,...
## $ eye_color     <fct> blue, brown, hazel, blue, NA, haz...
## $ esteem1       <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, ...
## $ esteem2       <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, ...
## $ esteem3       <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4,...
## $ esteem4       <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA,...
## $ esteem5_rev15 <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2, 3...
## $ jobsat1       <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, ...
## $ jobsat2_rev15 <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3,...
## $ jobsat3       <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3,...
## $ jobsat4       <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, N...
## $ jobsat5       <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4,...
```

Inspect the output of the glimpse() command and make sure you have converted all categorical variables to factors - especially those you will use as predictors.

Note: f you have factors like sex that have numeric data in the column (e.g, 1 and 2) instead of male/female you need to handle the situation differently. The preceding section, Experiment: Within N-way, illustrates how to handle this scenario.

11.10.2 Factor screening

Inspect the levels of each factor carefully. Make sure the factor levels of each variable are correct. Examine spelling and look for additional unwanted levels. For example, you wouldn't want to have the following levels for sex: male, mmale, female. Obviously, mmale is an incorrectly typed version of male. Scan all the factors in your data for erroneous factor levels. The code below displays the factor levels:

```
analytic_data_survey %>%
  select(where(is.factor)) %>%
  summary()
```

```
##        id            sex      eye_color
##  1      :  1   male    :147   blue : 99
##  2      :  1   female  :149   brown: 98
##  3      :  1   intersex:  2   hazel:100
##  4      :  1   NA's    :  2   NA's :  3
##  5      :  1
##  6      :  1
##  (Other):294
```

Also inspect the output of the above summary() command paying attention to the order of the levels in the factors. The order influences how text output and graphs are generated. In these data, the sex column has two levels: male and female in that order. Below we adjust the order of the sex variable because we want the x-axis of a future graph to display columns in the left to right order: female, male.

```
analytic_data_survey <- analytic_data_survey %>%
  mutate(sex = fct_relevel(sex,
                           "intersex",
                           "female",
                           "male"))
```

For eye color, we want a future graph to have the most common eye colors on the left so we reorder the factor levels:

```
analytic_data_survey <- analytic_data_survey %>%
  mutate(eye_color = fct_infreq(eye_color))
```

You can see the new order of the factor levels with summary():

```
analytic_data_survey %>%
  select(where(is.factor)) %>%
  summary()
```

```
##        id            sex      eye_color
##  1      :  1   intersex:  2   hazel:100
##  2      :  1   female  :149   blue : 99
##  3      :  1   male    :147   brown: 98
##  4      :  1   NA's    :  2   NA's :  3
##  5      :  1
##  6      :  1
##  (Other):294
```

11.10.3 Numeric screening

For numeric variables, it's important to find and remove impossible values. For example, in the context of this example you want to ensure none of the Likert responses are impossible (e.g., outside the 1- to 5-point rating scale) or clearly data entry errors.

Because we have several numeric columns that we are screening, we use the skim() command from the skimr package. The skim() command quickly provides basic descriptive statistics. In the output for this command there are also several columns that begin with p: p0, p25, p50, p75, and p100 (p25 and p75 omitted in output due to space). These columns correspond to the 0th, 25th, 50th, 75th, and 100th percentiles, respectively. The minimum and maximum values for the data column are indicated under the p0 and p100 labels. The median is the 50th percentile (p50). The interquartile range is the range between p25 and p75.

Start by examining the range of non-scale items. In this case it's only age. Examine the output to see if any of the age values are unreasonable. As noted, in the output p0 and p100 indicate the 0th percentile and the 100th percentile; that is the minimum and maximum values for the variable. Check to make sure none of the age values are unreasonably low or high. If they are, you may need to check the original data source or replace them with missing values.

```
library(skimr)
analytic_data_survey %>%
  select(age) %>%
  skim()
```

```
##   skim_variable n_missing  mean   sd p0 p50 p100
## 1           age         3 20.52 2.05 17  20   24
```

With respect to the multi-item scales, it makes sense to look at sets of items rather than all of the items at once. This is because sometimes items from different scales use different response ranges. For example, one measure might use a response scale with a range from 1 to 5; whereas another measure might use a response scale with a range from 1 to 7. This is undesirable from a psychometric point of view, as discussed previously, but if it happens in your data - look at the scale items separately to make it easy to see out of range values.

We begin by looking at the items in the first scale, self-esteem. Possible items responses for this scale range from 1 to 5; make sure all responses are in this range. If any values fall outside this range, you may need to check the original data source or replace them with missing values - as described previously.

```
analytic_data_survey %>%
  select(starts_with("esteem")) %>%
  skim()
```

```
##   skim_variable n_missing mean   sd p0 p50 p100
## 1       esteem1        24 3.39 0.54  3   3    5
## 2       esteem2        28 2.35 0.48  2   2    3
## 3       esteem3        31 3.96 0.37  3   4    5
## 4       esteem4        15 3.54 0.50  3   4    4
## 5 esteem5_rev15        35 2.22 0.47  1   2    3
```

Follow the same process for the job satisfaction items. Write that code on your own now.

Possible item responses for the job satisfaction scale range from 1 to 5, make sure all responses are in this range. If any values fall outside this range, you may need to check the original data source or replace them with missing values - as described previously.

```
analytic_data_survey %>%
  select(starts_with("jobsat")) %>%
  skim()
```

```
##   skim_variable n_missing mean   sd p0 p50 p100
## 1       jobsat1        25 3.34 0.51  3   3    5
## 2 jobsat2_rev15        27 1.51 0.61  1   1    3
## 3       jobsat3        28 2.84 0.37  2   3    3
## 4       jobsat4        35 4.29 0.70  3   4    5
## 5       jobsat5        24 4.57 0.61  3   5    5
```

11.10.4 Scale scores

For each person, scale scores involve averaging scores from several items to create an overall score. The first step in the creation of scales is correcting the values of any reverse-keyed items.

11.10.4.1 Reverse-key items

The way you deal with reverse-keyed items depends on how you scored them. Imagine you had a 5-point scale. You could have scored the scale with the values 1, 2, 3, 4, and 5. Alternatively, you could have scored the scale with the values 0, 1, 2, 3, and 4. The mathematical approach you use to correcting reverse-keyed items depends upon whether the scale starts with 1 or 0.

In this example, we scored the data using the value 1 to 5; so that is the approach illustrated here. See the extra information box for details on how to fixed reverse-keyed items when the scale begins with zero.

In this data file all the reverse-keyed items were identified with the suffix "_rev15" in the column names. This suffix indicates the item was reverse keyed and that the original scale used the response points 1 to 5. We can see those items with the glimpse() command below. Notice that there are two reverse-keyed items - each on difference scales.

```
analytic_data_survey %>%
  select(ends_with("_rev15")) %>%
  glimpse()
```

```
## Rows: 300
## Columns: 2
## $ esteem5_rev15 <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2, 3...
## $ jobsat2_rev15 <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3,...
```

To correct a reverse-keyed item where the lowest possible rating is 1 (i.e, 1 on a 1 to 5 scale), we simply subtract all the scores from a value one more than the highest point possible on the scale (i.e., one more than 5). For example, if

a 1 to 5 response scale was used we subtract each response from 6 to obtain the recoded value.

Original value	Math	Recoded value
1	6 - 1	5
2	6 - 2	4
3	6 - 3	3
4	6 - 4	2
5	6 - 5	1

The code below:

- selects columns that end with "_rev15" (i.e., both esteem and jobsat scales)
- subtracts the values in those columns from 6
- renames the columns by removing "_rev15" from the name because the reverse coding is complete

```
analytic_data_survey <- analytic_data_survey %>%
  mutate(6 - across(.cols = ends_with("_rev15")) ) %>%
  rename_with(.fn = str_remove,
              .cols = ends_with("_rev15"),
              pattern = "_rev15")
```

You can use the glimpse() command to see the result of your work. If you compare these new values to those obtained from the previous glimpse() command you can see they have changed. Also notice the column names no longer indicate the items are reverse keyed.

```
glimpse(analytic_data_survey)
```

```
## Rows: 300
## Columns: 14
## $ id        <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
## $ age       <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, N...
## $ sex       <fct> male, female, male, female, male, fem...
## $ eye_color <fct> blue, brown, hazel, blue, NA, hazel, ...
## $ esteem1   <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, N...
## $ esteem2   <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, N...
## $ esteem3   <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, ...
## $ esteem4   <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, ...
## $ esteem5   <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4, 3, 4,...
## $ jobsat1   <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4...
## $ jobsat2   <dbl> 5, 5, 5, NA, 5, 5, 4, 5, 4, 4, 3, 5, ...
```

```
## $ jobsat3   <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
## $ jobsat4   <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5...
## $ jobsat5   <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA,...
```

If your scale had used response options numbered 0 to 4 the math is different. For each item you would use subtract values from the highest possible point (i.e, 4) instead of one larger than the highest possible point.

Original value	Math	Recoded value
0	4 - 0	4
1	4 - 1	3
2	4 - 2	2
3	4 - 3	1
4	4 - 4	0

Thus, the mutate command would instead be:

mutate(4 - across(.cols = ends_with("_rev15")))

11.10.4.2 Creating scores

The process we use for creating scale scores deletes item-level data from analytic_data_survey. This is a desirable aspect of the process because it removes information that we are no longer interested in from our analytic data. That said, before we create scale score, we create a backup on the item-level data called analytic_data_survey_items. We will need to use this backup later to compute the reliability of the scales we are creating.

```
analytic_data_survey_items <- analytic_data_survey
```

We want to make a self_esteem scale and plan to select items using starts_with("esteem"). But prior to doing this we make sure the start_with() command only gives us the items we want - and not additional unwanted items. The output below confirms there are not problems associated with using starts_with("esteem").

```
analytic_data_survey %>%
  select(starts_with("esteem")) %>%
  glimpse()
```

```
## Rows: 300
```

```
## Columns: 5
## $ esteem1 <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, NA,...
## $ esteem2 <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, NA,...
## $ esteem3 <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, 4,...
## $ esteem4 <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, 3,...
## $ esteem5 <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4, 3, 4, 4...
```

Likewise, we want to make a job_sat scale and plan to select items using starts_with("jobsat"). The code and output below using starts_with("jobsat") only returns the items we are interested in.

```
analytic_data_survey %>%
  select(starts_with("jobsat")) %>%
  glimpse()
```

```
## Rows: 300
## Columns: 5
## $ jobsat1 <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4, ...
## $ jobsat2 <dbl> 5, 5, 5, NA, 5, 5, 4, 5, 4, 4, 3, 5, 3,...
## $ jobsat3 <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,...
## $ jobsat4 <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5, ...
## $ jobsat5 <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA, 4...
```

We calculate the scale scores using the rowwise() command. The mean() command provides the mean of columns by default - not people. We use the rowwise() command in the code below to make the mean() command work across columns (within participants) rather than within columns. The mutate command calculates the scale score for each person. The c_across() command combined with the starts_with() command ensures the items we want averaged together are the items that are averaged together. Notice there is a separate mutate line for each scale. The ungroup() command turns off the rowwise() command. We end the code block by removing the item-level data from the data set.

Important: Take note of how we name the scale variables (e.g., self_esteem, job_sat). We use a slightly different convention than our items. That is, these scale labels were picked so that they would *not* be selected by a starts_with("esteem") or starts_with("jobsat"). Why - because we later use those commands to remove the item-level data. We would want the command designed to remove the item-level data to also remove the scale we just calculated! This example illustrates how carefully you need to think about your naming conventions.

```
analytic_data_survey <- analytic_data_survey %>%
  rowwise() %>%
```

```
  mutate(self_esteem = mean(c_across(starts_with("esteem")),
                            na.rm = TRUE)) %>%
  mutate(job_sat = mean(c_across(starts_with("jobsat")),
                            na.rm = TRUE)) %>%
  ungroup() %>%
  select(-starts_with("esteem")) %>%
  select(-starts_with("jobsat"))
```

We can see our data now has the self_esteem column, a job_sat column, and that all of the item-level data has been removed.

```
glimpse(analytic_data_survey)
```

```
## Rows: 300
## Columns: 6
## $ id          <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...
## $ age         <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17,...
## $ sex         <fct> male, female, male, female, male, f...
## $ eye_color   <fct> blue, brown, hazel, blue, NA, hazel...
## $ self_esteem <dbl> 3.200, 3.800, 3.800, 3.000, 3.400, ...
## $ job_sat     <dbl> 4.00, 5.00, 4.40, 3.50, 4.00, 3.80,...
```

You now have two data sets analytic_data_survey and analytic_data_survey_items. You can calculate descriptive statistics, correlations and most analyses using the analytic_data_survey. To obtain the reliability of the scales you just created though you will need to use the analytic_data_survey_items. Both sets of data are ready for analysis.

11.11 Surveys: Multiple Occasions

This section outlines a workflow appropriate for when you have multiple occasion survey data. The data corresponds to a design where the researcher has measured, age, sex, eye color, self-esteem, and job satisfaction at each of two times points. Self-esteem and job satisfaction were measured with multi-item scales with reverse-keyed items.

To Begin:

- Use the Files tab to confirm you have the data: data_item_time.csv

- Start a new script for this example. Don't forget to start the script name with "script_".

```
# Date: YYYY-MM-DD
# Name: your name here
# Example: Multiple occasion survey

# Load data
library(tidyverse)

my_missing_value_codes <- c("-999", "", "NA")

raw_data_occasions <- read_csv(file = "data_item_time.csv",
                               na = my_missing_value_codes)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   sex = col_character(),
##   eye_color = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

We load the initial data into a raw_data_occasions but immediately make a copy we will work with called analytic_data_occasions. It's good to keep a copy of the raw data for reference if you encounter problems.

```
analytic_data_occasions <- raw_data_occasions
```

Remove empty rows and columns from your data using the remove_empty("rows") and remove_empty("cols"), respectively. As well, clean the names of your columns to ensure they conform to tidyverse naming conventions.

```
library(janitor)

# Initial cleaning
analytic_data_occasions <- analytic_data_occasions %>%
  remove_empty("rows") %>%
  remove_empty("cols") %>%
  clean_names()
```

You can confirm the column names following our naming convention with the glimpse command - and see the data type for each column.

```
glimpse(analytic_data_occasions)
```

```
## Rows: 300
## Columns: 24
## $ id                <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,...
## $ age               <dbl> 23, 22, 18, 23, 22, 17, 23, 22...
## $ sex               <chr> "male", "female", "male", "fem...
## $ eye_color         <chr> "blue", "brown", "hazel", "blu...
## $ t1_esteem1        <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, ...
## $ t1_esteem2        <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, ...
## $ t1_esteem3        <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3,...
## $ t1_esteem4        <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, ...
## $ t1_esteem5_rev15 <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2...
## $ t1_jobsat1        <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, ...
## $ t1_jobsat2_rev15 <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2,...
## $ t1_jobsat3        <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3,...
## $ t1_jobsat4        <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4...
## $ t1_jobsat5        <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5,...
## $ t2_esteem1        <dbl> 4, 5, 5, 4, NA, 4, 4, 5, 5, 5,...
## $ t2_esteem2        <dbl> 3, 4, 4, 3, 3, 4, 3, 4, 4, 4, ...
## $ t2_esteem3        <dbl> 5, 5, 5, 4, 5, 5, 3, 5, 5, 4, ...
## $ t2_esteem4        <dbl> 4, 5, 5, 4, 5, 5, 5, 5, 4, 5, ...
## $ t2_esteem5_rev15 <dbl> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
## $ t2_jobsat1        <dbl> 4, 6, 5, 4, 4, 4, 4, 6, 4, NA,...
## $ t2_jobsat2_rev15 <dbl> 2, 2, 2, 3, 2, 2, 3, 2, 3, 3, ...
## $ t2_jobsat3        <dbl> 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, ...
## $ t2_jobsat4        <dbl> 3, 6, 6, 5, 5, 5, 5, 6, 3, 5, ...
## $ t2_jobsat5        <dbl> 6, 3, 6, 5, NA, 5, 5, 6, 6, 6,...
```

11.11.1 Creating factors

Following initial cleaning, we identify categorical variables as factors. If you plan to conduct an ANOVA - it's critical that all predictor variables are converted to factors. Inspect the glimpse() output - if you followed our data entry naming conventions, categorical variables should be of the type character.

```
glimpse(analytic_data_occasions)
```

```
## Rows: 300
## Columns: 24
## $ id                <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,...
## $ age               <dbl> 23, 22, 18, 23, 22, 17, 23, 22...
```

```
## $ sex              <chr> "male", "female", "male", "fem...
## $ eye_color        <chr> "blue", "brown", "hazel", "blu...
## $ t1_esteem1       <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, ...
## $ t1_esteem2       <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, ...
## $ t1_esteem3       <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3,...
## $ t1_esteem4       <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, ...
## $ t1_esteem5_rev15 <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2...
## $ t1_jobsat1       <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, ...
## $ t1_jobsat2_rev15 <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2,...
## $ t1_jobsat3       <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3,...
## $ t1_jobsat4       <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4...
## $ t1_jobsat5       <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5,...
## $ t2_esteem1       <dbl> 4, 5, 5, 4, NA, 4, 4, 5, 5, 5,...
## $ t2_esteem2       <dbl> 3, 4, 4, 3, 3, 4, 3, 4, 4, 4, ...
## $ t2_esteem3       <dbl> 5, 5, 5, 4, 5, 5, 3, 5, 5, 4, ...
## $ t2_esteem4       <dbl> 4, 5, 5, 4, 5, 5, 5, 5, 4, 5, ...
## $ t2_esteem5_rev15 <dbl> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
## $ t2_jobsat1       <dbl> 4, 6, 5, 4, 4, 4, 4, 6, 4, NA,...
## $ t2_jobsat2_rev15 <dbl> 2, 2, 2, 3, 2, 2, 3, 2, 3, 3, ...
## $ t2_jobsat3       <dbl> 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, ...
## $ t2_jobsat4       <dbl> 3, 6, 6, 5, 5, 5, 5, 6, 3, 5, ...
## $ t2_jobsat5       <dbl> 6, 3, 6, 5, NA, 5, 5, 6, 6, 6,...
```

We have two variables, sex and eye_color, that are categorical variable of type character (i.e., chr). The participant id column is categorical as well, but of type double (i.e., dbl) which is a numeric column. You can quickly convert all character columns to factors using the code below:

```
analytic_data_occasions <- analytic_data_occasions %>%
  mutate(across(.cols = where(is.character),
                .fns = as_factor))
```

The participant identification number in the id column is numeric, so we have to handle that column on its own.

```
analytic_data_occasions <- analytic_data_occasions %>%
  mutate(id = as_factor(id))
```

You can ensure all of these columns are now factors using the glimpse() command.

```
glimpse(analytic_data_occasions)
```

```
## Rows: 300
```

```
## Columns: 24
## $ id               <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,...
## $ age              <dbl> 23, 22, 18, 23, 22, 17, 23, 22...
## $ sex              <fct> male, female, male, female, ma...
## $ eye_color        <fct> blue, brown, hazel, blue, NA, ...
## $ t1_esteem1       <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, ...
## $ t1_esteem2       <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, ...
## $ t1_esteem3       <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3,...
## $ t1_esteem4       <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, ...
## $ t1_esteem5_rev15 <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2...
## $ t1_jobsat1       <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, ...
## $ t1_jobsat2_rev15 <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2,...
## $ t1_jobsat3       <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3,...
## $ t1_jobsat4       <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4...
## $ t1_jobsat5       <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5,...
## $ t2_esteem1       <dbl> 4, 5, 5, 4, NA, 4, 4, 5, 5, 5,...
## $ t2_esteem2       <dbl> 3, 4, 4, 3, 3, 4, 3, 4, 4, 4, ...
## $ t2_esteem3       <dbl> 5, 5, 5, 4, 5, 5, 3, 5, 5, 4, ...
## $ t2_esteem4       <dbl> 4, 5, 5, 4, 5, 5, 5, 5, 4, 5, ...
## $ t2_esteem5_rev15 <dbl> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
## $ t2_jobsat1       <dbl> 4, 6, 5, 4, 4, 4, 4, 6, 4, NA,...
## $ t2_jobsat2_rev15 <dbl> 2, 2, 2, 3, 2, 2, 3, 2, 3, 3, ...
## $ t2_jobsat3       <dbl> 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, ...
## $ t2_jobsat4       <dbl> 3, 6, 6, 5, 5, 5, 5, 6, 3, 5, ...
## $ t2_jobsat5       <dbl> 6, 3, 6, 5, NA, 5, 5, 6, 6, 6,...
```

Inspect the output of the glimpse() command and make sure you have converted all categorical variables to factors - especially those you will use as predictors.

Note: If you have factors like sex that have numeric data in the column (e.g, 1 and 2) instead of male/female you need to handle the situation differently. The preceding section, Experiment: Within N-way, illustrates how to handle this scenario.

11.11.2 Factor screening

Inspect the levels of each factor carefully. Make sure the factor levels of each variable are correct. Examine spelling and look for additional unwanted levels. For example, you wouldn't want to have the following levels for sex: male, mmale, female. Obviously, mmale is an incorrectly typed version of male. Scan all the factors in your data for erroneous factor levels. The code below displays the factor levels:

```
analytic_data_occasions %>%
  select(where(is.factor)) %>%
  summary()
```

```
##        id            sex      eye_color
##  1      :  1   male    :147   blue : 99
##  2      :  1   female  :149   brown: 98
##  3      :  1   intersex:  2   hazel:100
##  4      :  1   NA's    :  2   NA's :  3
##  5      :  1
##  6      :  1
##  (Other):294
```

Also inspect the output of the above summary() command paying attention to the order of the levels in the factors. The order influences how text output and graphs are generated. In these data, the sex column has two levels: male and female in that order. Below we adjust the order of the sex variable because we want the x-axis of a future graph to display columns in the left to right order: female, male.

```
analytic_data_occasions <- analytic_data_occasions %>%
  mutate(sex = fct_relevel(sex,
                           "intersex",
                           "female",
                           "male"))
```

For eye color, we want a future graph to have the most common eye colors on the left so we reorder the factor levels:

```
analytic_data_occasions <- analytic_data_occasions %>%
  mutate(eye_color = fct_infreq(eye_color))
```

You can see the new order of the factor levels with summary():

```
analytic_data_occasions %>%
  select(where(is.factor)) %>%
  summary()
```

```
##        id            sex      eye_color
##  1      :  1   intersex:  2   hazel:100
##  2      :  1   female  :149   blue : 99
##  3      :  1   male    :147   brown: 98
##  4      :  1   NA's    :  2   NA's :  3
```

```
##  5       :  1
##  6       :  1
##  (Other):294
```

11.11.3 Numeric screening

For numeric variables, it's important to find and remove impossible values. For example, in the context of this example you want to ensure none of the Likert responses are impossible (e.g., outside the 1- to 5-point rating scale) or clearly data entry errors.

Because we have several numeric columns that we are screening, we use the skim() command from the skimr package. The skim() command quickly provides basic descriptive statistics. In the output for this command there are also several columns that begin with p: p0, p25, p50, p75, and p100 (p25 and p75 are omitted in output due to space). These columns correspond to the 0th, 25th, 50th, 75th, and 100th percentiles, respectively. The minimum and maximum values for the data column are indicated under the p0 and p100 labels. The median is the 50th percentile (p50). The interquartile range is the range between p25 and p75.

Start by examining the range of non-scale items. In this case it's only age. Examine the output to see if any of the age values are unreasonable. As noted, p0 and p100 in the output indicate the 0th percentile and the 100th percentile; that is the minimum and maximum values for the variable. Check to make sure none of the age values are unreasonably low or high. If they are, you may need to check the original data source or replace them with missing values.

```
library(skimr)
analytic_data_occasions %>%
  select(age) %>%
  skim()
```

```
##   skim_variable n_missing  mean   sd p0 p50 p100
## 1           age         3 20.52 2.05 17  20   24
```

With respect to the multi-item scales, it makes sense to look at sets of items rather than all of the items at once. This is because sometimes items from different scales use different response ranges. For example, one measure might use a response scale with a range from 1 to 5; whereas another measure might use a response scale with a range from 1 to 7. This is undesirable from a psychometric point of view, as discussed previously, but if it happens in your data - look at the scale items separately to make it easy to see out of range values.

We begin by looking at the items in the first scale, self-esteem. Possible item

responses for this scale range from 1 to 5. Make sure all responses are in this range. If any values fall outside this range, you may need to check the original data source or replace them with missing values - as described previously.

Because we want to select the self-esteem items from both time 1 and time 2 we cannot use the starts_with() command. Instead we use the contains() command in the code below.

```
analytic_data_occasions %>%
  select(contains("esteem")) %>%
  skim()
```

```
##       skim_variable n_missing mean   sd p0 p50 p100
## 1        t1_esteem1        24 3.39 0.54  3   3    5
## 2        t1_esteem2        28 2.35 0.48  2   2    3
## 3        t1_esteem3        31 3.96 0.37  3   4    5
## 4        t1_esteem4        15 3.54 0.50  3   4    4
## 5  t1_esteem5_rev15        35 2.22 0.47  1   2    3
## 6        t2_esteem1         5 4.27 0.64  3   4    6
## 7        t2_esteem2         5 3.33 0.47  3   3    4
## 8        t2_esteem3         6 4.77 0.69  3   5    6
## 9        t2_esteem4         3 4.46 0.59  3   5    5
## 10 t2_esteem5_rev15         4 3.19 0.45  2   3    4
```

Follow the same process for the job satisfaction items. Possible item responses for the job satisfaction scale range from 1 to 5, make sure all responses are in this range. If any values fall outside this range, you may need to check the original data source or replace them with missing values - as described previously.

```
analytic_data_occasions %>%
  select(contains("jobsat")) %>%
  skim()
```

```
##       skim_variable n_missing mean   sd p0 p50 p100
## 1        t1_jobsat1        25 3.34 0.51  3   3    5
## 2  t1_jobsat2_rev15        27 1.51 0.61  1   1    3
## 3        t1_jobsat3        28 2.84 0.37  2   3    3
## 4        t1_jobsat4        35 4.29 0.70  3   4    5
## 5        t1_jobsat5        24 4.57 0.61  3   5    5
## 6        t2_jobsat1         2 4.23 0.62  3   4    6
## 7  t2_jobsat2_rev15         3 2.54 0.59  2   2    4
## 8        t2_jobsat3         5 3.76 0.43  3   4    4
## 9        t2_jobsat4         3 5.03 0.99  3   5    6
## 10       t2_jobsat5         3 5.36 0.92  3   6    6
```

11.11.4 Scale scores

For each person, scale scores involve averaging scores from several items to create an overall score. The first step in the creation of scales is correcting the values of any reverse-keyed items.

11.11.4.1 Reverse-key items

The way you deal with reverse-keyed items depends on how you scored them. Imagine you had a 5-point scale. You could have scored the scale with the values 1, 2, 3, 4, and 5. Alternatively, you could have scored the scale with the values 0, 1, 2, 3, and 4. The mathematical approach you use to correcting reverse-keyed items depends upon whether the scale starts with 1 or 0.

In this example, we scored the Likert items using the values 1 to 5. Therefore, we use the reverse keying approach for scales that being with 1. The preceding section, "Surveys: Single occasion", describes how the math differs when the response scale starts with 0. We encourage you to read that section before going further if you have not done so already.

In this data file all the reverse-keyed items were identified with the suffix "_rev15" in the column names. This suffix indicates the item was reverse keyed and that the original scale used the response points 1 to 5. We can see those items with the glimpse() command below. Notice that there are two reverse-keyed items - each on difference scales.

```
analytic_data_survey %>%
  select(ends_with("_rev15")) %>%
  glimpse()
```

```
## Rows: 300
## Columns: 0
```

To correct reverse-keyed items where the lowest possible rating is 1 (i.e, 1 on a 1 to 5 scale), we simply subtract all the scores from a value one more than the highest possible rating (i.e., 6).

The code below:

- selects columns that end with "_rev15" (i.e., both esteem and jobsat scales)
- subtracts the values in those columns from 6
- renames the columns by removing "_rev15" from the name because the reverse coding is complete

```
analytic_data_occasions <- analytic_data_occasions %>%
  mutate(6 - across(.cols = ends_with("_rev15")) ) %>%
```

```
  rename_with(.fn = str_remove,
              .cols = ends_with("_rev15"),
              pattern = "_rev15")
```

You can use the glimpse() command to see the result of your work. If you compare these new values to those obtained from the previous glimpse() command you can see they have changed. Also notice the column names no longer indicate the items are reverse keyed.

11.11.4.2 Creating scores

The process we use for creating scale scores deletes item-level data from analytic_data_survey. This is a desirable aspect of the process because it removes information we no longer need. That said, before we create scale scores, we create a backup on the item-level data in analytic_data_survey called analytic_data_survey_items. We will need to use this backup later to compute the reliability of the scales we are creating.

```
analytic_data_occasions_items <- analytic_data_occasions
```

We want to make a self_esteem scale and plan to select items using starts_with("t1_esteem"). Prior to doing this we make sure the start_with() command only gives us the items we want - and not additional unwanted items. The output below confirms there are not problems associated with using starts_with("t1_esteem").

```
analytic_data_occasions %>%
  select(starts_with("t1_esteem")) %>%
  glimpse()
```

```
## Rows: 300
## Columns: 5
## $ t1_esteem1 <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, ...
## $ t1_esteem2 <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, ...
## $ t1_esteem3 <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4,...
## $ t1_esteem4 <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4,...
## $ t1_esteem5 <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4, 3, 4...
```

Repeat this set of commands using t2_esteem, t1_jobsat, and t2_jobsat. Make sure that those start_with() terms select only the relevant items and not others.

We calculate the scale scores using the rowwise() command. The mean() command provides the mean of columns by default - not people. We use the

rowwise() command in the code below to make the mean() command work across columns (within participants) rather than within columns. The mutate command calculates the scale score for each person. The c_across() command combined with the starts_with() command ensures the items we want averaged together are the items that are averaged together. Notice that there is a separate mutate line for each scale. The ungroup() command turns off the rowwise() command. We end the code block by removing the item-level data from the data set.

Important: Take note of how the names of the scale variables (e.g., esteem_t1, jobsat_t1) use a slightly different convention than our items. That is, these scale labels were picked so that they would *not* be selected by a starts_with("t1_esteem") or starts_with("t1_jobsat"). Why - because we later use those commands to remove the item-level data. We would want the command designed to remove the item-level data to also remove the scale we just calculated! This example illustrates how carefully you need to think about your naming conventions.

```
analytic_data_occasions <- analytic_data_occasions %>%
  rowwise() %>%
  mutate(esteem_t1 = mean(c_across(starts_with("t1_esteem")),
                                na.rm = TRUE)) %>%
  mutate(esteem_t2 = mean(c_across(starts_with("t2_esteem")),
                                na.rm = TRUE)) %>%
  mutate(jobsat_t1 = mean(c_across(starts_with("t1_jobsat")),
                                na.rm = TRUE)) %>%
  mutate(jobsat_t2 = mean(c_across(starts_with("t2_jobsat")),
                                na.rm = TRUE)) %>%
  ungroup() %>%
  select(-starts_with("t1_esteem")) %>%
  select(-starts_with("t2_esteem")) %>%
  select(-starts_with("t1_jobsat")) %>%
  select(-starts_with("t2_jobsat"))
```

We can see our data now has the columns t1_esteem, t2_esteem, t1_jobsat, and t2_jobsat. As well, we can see that all of the item-level data has been removed from the data set.

```
glimpse(analytic_data_occasions)
```

```
## Rows: 300
## Columns: 8
## $ id        <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
## $ age       <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, N...
## $ sex       <fct> male, female, male, female, male, fem...
```

```
## $ eye_color <fct> blue, brown, hazel, blue, NA, hazel, ...
## $ esteem_t1 <dbl> 3.200, 3.800, 3.800, 3.000, 3.400, 3....
## $ esteem_t2 <dbl> 3.8, 4.4, 4.4, 3.6, 4.0, 4.2, 3.6, 4....
## $ jobsat_t1 <dbl> 4.00, 5.00, 4.40, 3.50, 4.00, 3.80, 3...
## $ jobsat_t2 <dbl> 4.20, 4.40, 5.00, 4.20, 4.25, 4.40, 4...
```

11.11.5 Pivot to tidy data

The analytic data in its current form does not conform to the tidy data specification. Inspect the data with the print() command. Notice that a single column is not used to represent esteem, jobsat, or time. Rather, there are four columns that are a mix of these variables. The consequence of this is that there are two esteem ratings/observations on each row and two jobsat ratings/observations on each row. Tidy data is structured so that each variable is represented in a single column and each observation has its own row.

```
print(analytic_data_occasions)
```

```
## # A tibble: 300 x 8
##    id      age sex   eye_color esteem_t1 esteem_t2 jobsat_t1
##    <fct> <dbl> <fct> <fct>         <dbl>     <dbl>     <dbl>
##  1 1        23 male  blue            3.2       3.8       4
##  2 2        22 fema~ brown           3.8       4.4       5
##  3 3        18 male  hazel           3.8       4.4       4.4
##  4 4        23 fema~ blue            3         3.6       3.5
##  5 5        22 male  <NA>            3.4       4         4
##  6 6        17 fema~ hazel           3.5       4.2       3.8
##  7 7        23 male  blue            3         3.6       3.6
##  8 8        22 fema~ brown           3.8       4.4       4.6
##  9 9        17 male  hazel           3.6       4.2       3.75
## 10 10       NA fema~ blue            3.6       4.2       3.8
## # ... with 290 more rows, and 1 more variable:
## #   jobsat_t2 <dbl>
```

The pivot_longer() command below coverts our data to the tidy data format. In this command we specify the columns with data by using esteem_t1:jobsat_t2; this selects these two columns and all of the columns between them. Each of these columns represents a dependent variable at a particular time in the format "variable_time" (e.g., esteem_t1). The code names_to = c(".value", "time") explains this format to R. It indicates that the first part of the column name (e.g., esteem) contains the name of the variable (expressed in the code as ".value"). It also indicates that the second part of the column name represents time. The line names_sep = "_" tells the R that the underscore character is used to separate the first part of the name

from the second part of the name. When this code is executed it creates a tidy version of data set stored in analytic_survey_tidy.

```
analytic_occasion_tidy <- analytic_data_occasions %>%
  pivot_longer(esteem_t1:jobsat_t2,
               names_to = c(".value", "time"),
               names_sep = "_")
```

You can see the new data with the print() command. Notice that each participant has multiple rows associated with them.

```
print(analytic_occasion_tidy)
```

```
## # A tibble: 600 x 7
##    id      age sex    eye_color time  esteem jobsat
##    <fct> <dbl> <fct>  <fct>     <chr>  <dbl>  <dbl>
##  1 1        23 male   blue      t1       3.2   4
##  2 1        23 male   blue      t2       3.8   4.2
##  3 2        22 female brown     t1       3.8   5
##  4 2        22 female brown     t2       4.4   4.4
##  5 3        18 male   hazel     t1       3.8   4.4
##  6 3        18 male   hazel     t2       4.4   5
##  7 4        23 female blue      t1       3     3.5
##  8 4        23 female blue      t2       3.6   4.2
##  9 5        22 male   <NA>      t1       3.4   4
## 10 5        22 male   <NA>      t2       4     4.25
## # ... with 590 more rows
```

You now have three data sets The data analytic_occasion_tidy is appropriate for conducting a repeated measures ANOVA or more complicated analyses. The data analytic_data_occasions is appropriate for calculating descriptive statistics and correlations. The data analytic_occasions_items is appropriate for calculating the reliability of the scales you constructed. These data are ready for analysis.

11.12 Basic descriptive statistics

Regardless of the design of the study, most researchers want to see descriptive statistics for the variables in their study. We offer three approaches for obtaining descriptive statistics below. For convenience we use the recent data

set analytic_data_occasions. But recognize the commands below can be used with all the analytic data sets we created for the various designs.

11.12.1 skim()

One approach is the skim() command from the skimr package. The skim() command quickly provides the basic descriptive statistics. In the output for this command there are also several columns that begin with p: p0, p25, p50, p75, and p100 (p25 and p75 are omitted in output due to space). These columns correspond to the 0th, 25th, 50th, 75th, and 100th percentiles, respectively. The minimum and maximum values for the data column are indicated under the p0 and p100 labels. The median is the 50th percentile (p50). The interquartile range is the range between p25 and p75. Notice that we run this command on the "wide" version of the data (analytic_data_occasions) rather than tidy version of the data (analytic_occasion_tidy).

```
library(skimr)
skim(analytic_data_occasions)
```

```
##   skim_variable n_missing  mean   sd   p0  p50  p100
## 1           age         3 20.52 2.05 17.0 20.0 24.00
## 2     esteem_t1         0  3.40 0.32  2.5  3.4  4.25
## 3     esteem_t2         0  3.93 0.34  3.2  4.0  4.80
## 4     jobsat_t1         0  3.91 0.43  2.0  4.0  5.00
## 5     jobsat_t2         0  4.37 0.42  3.0  4.4  5.25
```

11.12.2 apa.cor.table()

Another approach is the apa.cor.table() command from the apaTables package. This quickly provides the basic descriptive statistics as well as correlations among variable. As well, it will even create a Word document with this information, see Figure **??**. Notice that we run this command on the "wide" version of the data (analytic_data_occasions) rather than tidy version of the data (analytic_occasion_tidy).

```
library(apaTables)
analytic_data_survey %>%
  select(where(is.numeric)) %>%
  apa.cor.table(filename = "apa_descriptives.doc")
```

Means, standard deviations, and correlations with confidence intervals

Variable	*M*	*SD*	1	2	3	4
1. age	20.52	2.05				
2. esteem_t1	3.40	0.32	-.04 [-.15, .08]			
3. esteem_t2	3.93	0.34	.01 [-.10, .13]	.84** [.80, .87]		
4. jobsat_t1	3.91	0.43	-.00 [-.12, .11]	.64** [.56, .70]	.56** [.48, .63]	
5. jobsat_t2	4.37	0.42	-.02 [-.13, .10]	.58** [.50, .65]	.52** [.43, .60]	.82** [.77, .85]

Note. M and *SD* are used to represent mean and standard deviation, respectively. Values in square brackets indicate the 95% confidence interval for each correlation. The confidence interval is a plausible range of population correlations that could have caused the sample correlation (Cumming, 2014). * indicates $p < .05$. ** indicates $p < .01$.

FIGURE 11.5: Word document created by apa.cor.table

11.12.3 tidyverse

A final approach uses tidyverse commands. This approach is oddly long - and we won't describe how it works in detail. But, based on the information in the previous chapter you should be able to work out how this code works. Even though this code is long - it provide the ultimate in flexibility. If a new statistic is developed that you want to use, you can simply include the command for it in the desired_descriptives list and it will be included in your table. Notice that we run this command on the "wide" version of the data (analytic_data_occasions) rather than tidy version of the data (analytic_occasion_tidy).

```
library(tidyverse)
# HMisc package must be installed.
# Library command not needed for HMisc package.

desired_descriptives <- list(
  mean = ~mean(.x, na.rm = TRUE),
  CI95_LL = ~Hmisc::smean.cl.normal(.x)[2],
  CI95_UL = ~Hmisc::smean.cl.normal(.x)[3],
  sd = ~sd(.x, na.rm = TRUE),
  min = ~min(.x, na.rm = TRUE),
  max = ~max(.x, na.rm = TRUE),
  n = ~sum(!is.na(.x))
)

row_sum <- analytic_data_occasions %>%
```

```
  summarise(across(.cols = where(is.numeric),
                   .fns =  desired_descriptives,
                   .names = "{col}___{fn}"))

long_summary <- row_sum %>%
  pivot_longer(cols = everything(),
               names_to = c("var", "stat"),
               names_sep = c("___"),
               values_to = "value")

summary_table <- long_summary %>%
  pivot_wider(names_from = stat,
              values_from = value)

# round to 3 decimals
summary_table_rounded <- summary_table %>%
  mutate(across(.cols = where(is.numeric),
                .fns= round,
                digits = 3)) %>%
  as.data.frame()

print(summary_table_rounded)
```

```
##         var   mean CI95_LL CI95_UL    sd  min   max   n
## 1       age 20.522  20.288  20.756 2.048 17.0 24.00 297
## 2 esteem_t1  3.403   3.366   3.440 0.324  2.5  4.25 300
## 3 esteem_t2  3.927   3.889   3.966 0.337  3.2  4.80 300
## 4 jobsat_t1  3.905   3.856   3.955 0.435  2.0  5.00 300
## 5 jobsat_t2  4.368   4.320   4.416 0.425  3.0  5.25 300
```

11.12.4 Cronbach's alpha

If you want Cronbach's alpha to estimate the reliability of the scale, you can use the alpha command from the psych package with the code below. Note we have to use the item-level data we previously created a copy of called analytic_data_survey_items. The glimpse() command illustrates this data set has all the original items (after reverse-key coding has been fixed).

```
analytic_data_survey_items %>%
  glimpse()
```

```
## Rows: 300
```

```
## Columns: 14
## $ id        <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
## $ age       <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, N...
## $ sex       <fct> male, female, male, female, male, fem...
## $ eye_color <fct> blue, brown, hazel, blue, NA, hazel, ...
## $ esteem1   <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, N...
## $ esteem2   <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, N...
## $ esteem3   <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, ...
## $ esteem4   <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, ...
## $ esteem5   <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4, 3, 4,...
## $ jobsat1   <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4...
## $ jobsat2   <dbl> 5, 5, 5, NA, 5, 5, 4, 5, 4, 4, 3, 5, ...
## $ jobsat3   <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
## $ jobsat4   <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5...
## $ jobsat5   <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA,...
```

We calculated reliability using psych::alpha() command. Cronbach's alpha is labeled "raw alpha" in the output. Cronbach's alpha is an estimate of the proportion of variability in observed scores that is due to actual differences among participants (rather than measurement error). Remember, never use library(psych), it will break the tidyverse packages. Instead, precede all psych package commands with psych:: as we do below with psych::alpha().

```
rxx_alpha <- analytic_data_occasions_items %>%
  select(starts_with("t1_esteem")) %>%
  psych::alpha()

print(rxx_alpha$total)
```

```
##  raw_alpha std.alpha G6(smc) average_r  S/N     ase  mean
##     0.6622    0.6634  0.6173    0.2827 1.97 0.03035 3.403
##       sd median_r
##  0.3239   0.2927
```

12

Positive Predictive Value

12.1 Required

The following CRAN packages must be installed:

Required CRAN Packages
pwr
TOSTER

12.2 Overview

Typically when students learn about p-values and Null Hypothesis Significance testing then tend to pair a two ideas. Specifically, they tend to pair a significant p-value with the idea there is an effect. The purpose of this section is to unpair those two ideas. Specifically, a significant p-value only indicates that there **may** be an effect.

At this point you've learned to be cautious when it comes to interpreting p-values. But so far, that's been more of a qualitative form of caution. In this section we attempt to make it a more quantitative form of caution. Specifically, we introduce you to the concept of positive predictive values (PPV) in the context of interpreting significant p-values. The PPV statistic provides us with an estimate of how likely it is there really is an effect when a p-value is significant. The approach for this chapter was inspired by a blog[1] post. I strongly encourage you to watch this video[2] by the Center for Open Science on the consequences of low statistical power.

[1] https://dirnagl.com/2014/09/22/p-value-vs-positive-predictive-value/
[2] https://youtu.be/7daQRvRO-NE

12.3 Informal explanation

12.3.1 Probability of an effect

When you conduct an experiment to determine if an effect exists you start off fairly confident that the effect is there - otherwise you would not invest the time and money to run the study. But some research hypotheses will be true and others will not be true. One way to think about this is by imagining 1000 hypothesis as per Figure **??**. In this figure there are 1000 squares. Each square represents an hypothesis worth testing. Just like the real world some hypotheses are true and some are false. We've shaded the squares to indicate which hypotheses are true or false. More specifically, we shaded 40% of the 1000 hypotheses (i.e., 600) green to indicate they are true (i.e., there is an effect). In contrast, we shaded 60% of the 1000 hypotheses (i.e., 600) yellow to indicate they are false (i.e., there is no effect).

Why did we pick the percentage of hypotheses that are true in Figure **??** to be 40%? We did so because 40% could well correspond to the number of hypotheses that are true in psychology research overall. It's difficult to assess the number of hypotheses that are true in psychology given that there is a tendency for only significant effects to be published. However, we can examine pre-registered studies to obtain an estimate of the percentage of hypotheses that are true. With pre-registration scientists register their hypotheses before they collect their data. Consequently looking at the percentage of pre-registered hypotheses that are true may be a reasonable estimate of the percentage of hypotheses that are true in general. Fortunately, a recent study examined a large set of pre-registered studies and determined that just over 40% of research hypotheses are true (**?**). This finding correspondingly implies that approximately 60% of research hypotheses are false. That is, approximately 60% of studies fail to find the desired effect. We used these percentages to shade Figure **??**. More specifically, we used these numbers to indicate P(effect) = .400 and the P(no effect) = .600.

12.3.2 Type I Error

Even when an hypothesis is false there still a chance we will obtain a significant result (i.e., $p < .05$). More specifically, we will obtain a significant result 5% of the time when the null hypothesis is true and there is no effect (i.e., $\alpha = .05$). In terms of our example, this means that of the 600 false hypotheses, we would obtain a significant result for 30 of them (i.e., .05 * 600 = 30). These 30 false positive significance tests are illustrated in Figure **??** below.

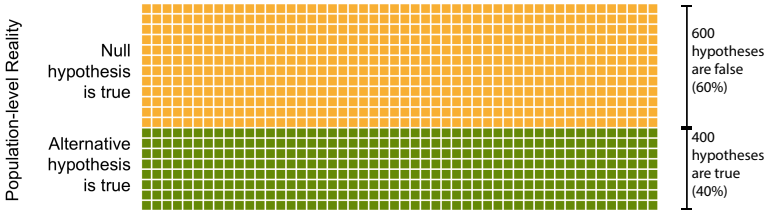


FIGURE 12.1: One-thousand hypotheses represented as squares. True hypotheses are indicated by the color green. False hypotheses are indicated by the color yellow.

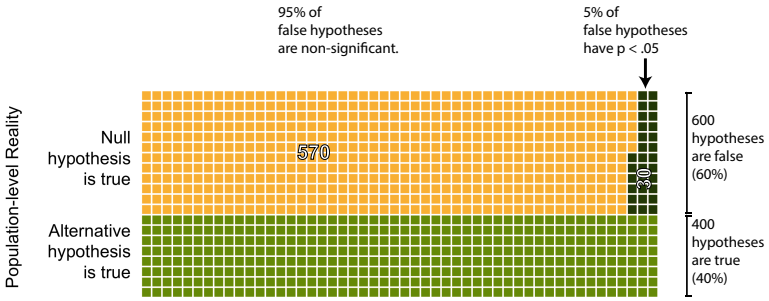


FIGURE 12.2: False positive significance tests. False hypotheses for which the researcher obtained a significant result are indicated by dark green squares.

12.3.3 Power

Unfortunately, even when an hypothesis is true we won't always obtain a significant p-value. Low sample sizes in psychology are common. As a result most studies in psychology only have a 30% chance of finding an effect if there is one. That is, the typical value for statistical power in psychology is around .30. This typical statistical power level is illustrated in the context of our example in Figure **??** below. In this figure, hypotheses that were true but obtained non-significant p-values are indicated by red squares. In contrast, the hypotheses that were true and obtained significant p-values are indicated by green squares; there are 120 green squares.

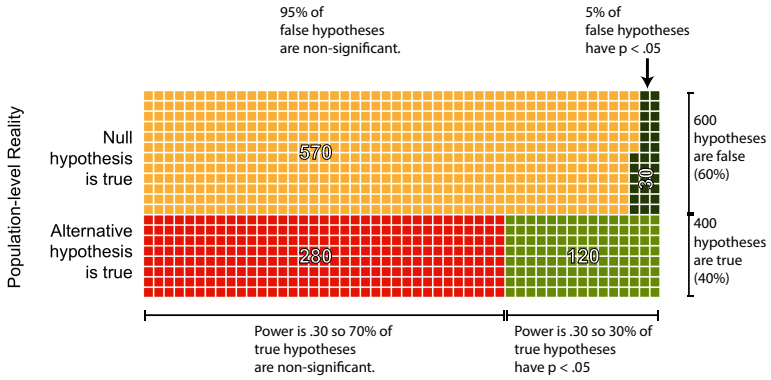


FIGURE 12.3: True positive signficance tests. True hypotheses for which the researcher obtained a significant result are indicated by green squares. True hypotheses for which the researcher obtained a non-significant result are indicated by red squares.

12.3.4 Calculating PPV

You can see from the blue outline in Figure **??**, below, that there two ways to obtain a significant p-value. Specifically, you can obtain a significant p-value when the null hypothesis is true (a false positive p-value). As well, you can obtain a significant p-value when the null hypothesis is false (a true positive p-value). As a researcher, when we obtain a significant p-value we don't know if it's a true-positive or a false-positive. We calculate PPV to determine how likely it is there is an effect when the p-value is significant.

As illustrated in Figure **??**, below, the positive predictive value (PPV) is