# ***An Interactive ‘Co-Laboratory’ Manual for Undergraduate Statistics Using R***

**W. Spencer Murch, Ph.D.**

Version 1.0 (*Beta*)

2023

Lesson 1 - Welcome!

Last edited on: 2023-12-07

# Foreword: What is this thing?

Existing statistics textbooks are very good at two broad tasks: (1) teaching the precise details of statistical theory in a pencil-and-paper kind of way, and (2) teaching the foundations of statistical computing in a precise and detailed kind of way. This is great news if you are either a first- or second-year undergraduate student trying to get a firm grasp on the basics of statistical theory, or a first- or second-year graduate student trying to get a firm grasp on the practical use of statistical software like R or IBM SPSS.

But in-between these two stages of most researchers’ lives is a senior undergraduate research project that presents one’s first real encounter with practical data analysis. These projects are usually undertaken as part of voluntary research assistantships in addition to a full-time course load. Compounding all this work is the very real pressure to perform exceptionally in both arenas in order to secure one of the few coveted spots in the graduate programs of one’s field.

I believe that senior undergraduates try to succeed in both their course work and their voluntary research, not just for the prestige that can come with admission to graduate school, but because they genuinely want to contribute to science and the discovery of new knowledge. This brief manual is for those students who have conquered introductory statistics, but who do not have the bandwidth to tackle 1,000 page statistical computing texts on a self-directed basis. It strives to provide a lightspeed crash course in statistical computing that prepares undergraduate psychology students to run their first real analyses under the supervision of a course instructor or university researcher. It provides lots of practical examples with regular opportunities to reflect critically on the procedures being taught. To this end, this manual does not cover every detail of statistical testing in psychological science or any other field. Instead, it presents a handful of practical methods and core competencies for getting started with statistical computing in R. It is best described as a limited introduction to computing classical statistics; a bridge between undergraduate-level theory and graduate-level practice which trusts that readers will seek out [much](https://www.discoveringstatistics.com/books/discovering-statistics-using-r/) [better](https://learningstatisticswithr.com/) [books](https://www.statlearning.com/) as they continue to develop their scientific acumen.

# Table of Contents

1. Lesson 1 - Welcome!
2. Lesson 2 - Introduction to R
3. Lesson 3 - Data Cleaning and Visualization Using *dplyr* and *ggplot2*
4. Lesson 4 - Frequency tables, chi square test and the odds ratio
5. Lesson 5 - *t*-Tests
6. Lesson 6 - Select topics in correlation and regression
7. Lesson 7 - One-way ANOVA
8. Lesson 8 - Factorial ANOVA
9. Lesson 9 - Bonus! An Introduction to RMarkdown for Psychological Science

### Credits and License

All codes and graphics presented in this manual were created by W. Spencer Murch (2023), and are licensed under Creative Commons CC BY-NC 4.0. All functions, packages, and data used here are licensed by their respective creators. Special thanks are owed to Zakary Draper whose example datasets appear throughout, and who provided revisions on several chapters.

# Lesson 1 - Welcome!

This manual is the main companion to the laboratory work we will do this semester. Its purpose is to be a living notebook, and that means it will do three important jobs:

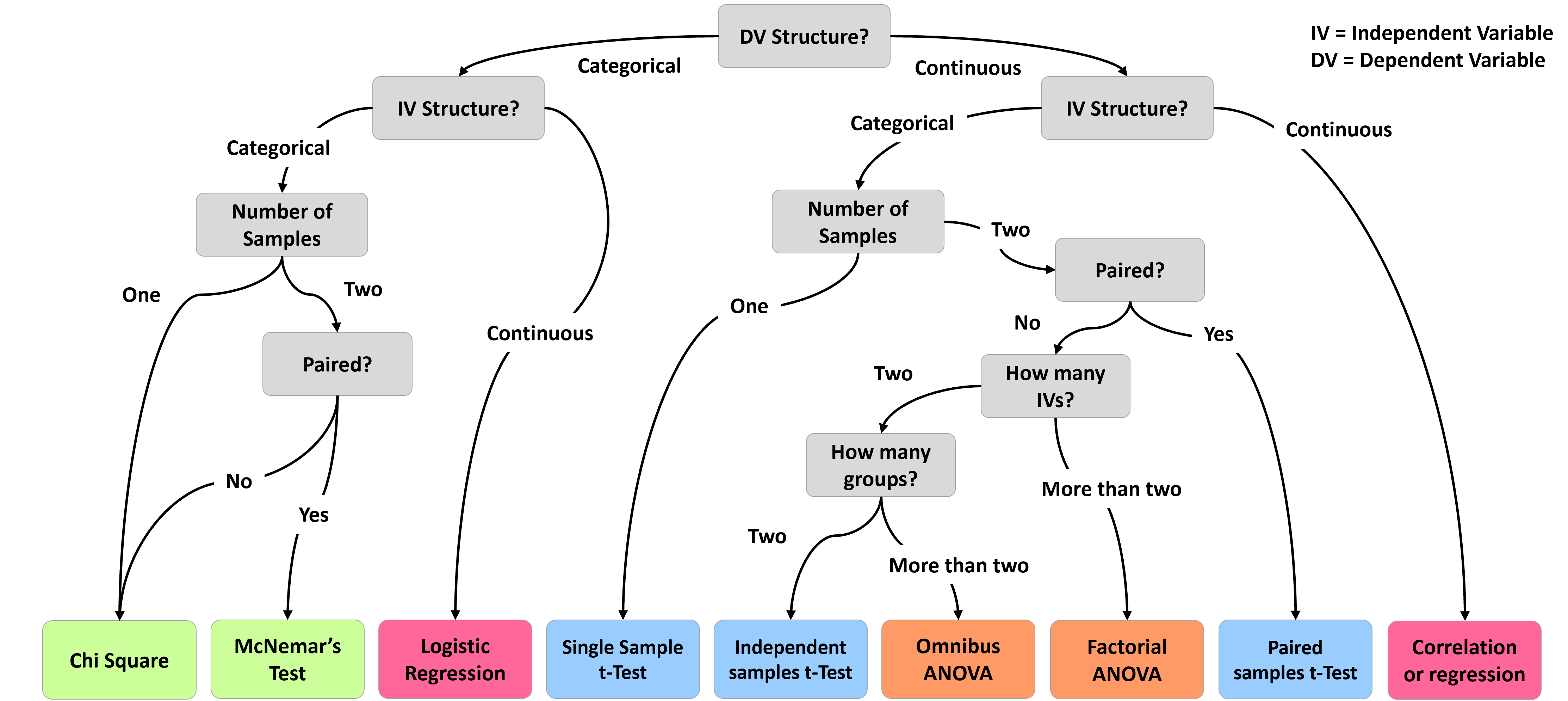
1. This manual exists in two parts: a Microsoft Word document that is easy to read, and an RMarkdown script that is easy to edit. The Word document is where you will do your studying, and the Markdown script is the place where you can enter chunks of code that we will create together. These *code chunks* will give us the ability to work with data, run statistical tests, and make visualizations. Code chunks will look like this…

print("Hello world!")

## [1] "Hello world!"

1. Using both halves of this manual, you can learn new concepts in R and Rstudio, and then practice them by writing down your thoughts, reflections, and key takeaways from each new skill you learn. Any time you see a *blank space* made out of ten underscores (“\_\_\_\_\_\_\_\_\_\_”), this is a spot where you can reflect on what you learned, review the principles that were taught, or describe the code you’ve just written using your own words. This is important because…
2. When each lesson is done, you can use the *Knit* button at the top of RStudio to turn this RMarkdown script into an elegant, functional laboratory manual that contains both the material you studied, and your own reflections. In other words, it will contain the whole semester’s material *in your own words*! That way, you can look back to the material you learned this semester as you carry on to your next statistics course, your Honours Thesis project, and even Graduate School. This is why I call it a “*Collaboratory Manual*”; it’s a laboratory manual that we are collaborating to make.

Each Lesson below covers one important topic in statistical programming. In each Lesson, you will find *code chunks* that we will work together to populate with working R code, and some blank spaces that will prompt you to explain, report, or reflect on the code you wrote. Each Lesson begins by stating our objectives for that Lesson, and ends with blank spaces that you and your classmates will fill in as you reflect on the codes you created. As you progress through the Lessons, it is my hope that you will begin to see how each new statistical approach tries to address a specific arrangement of experiment data that you will often encounter in the world of psychological science. Indeed, we could fit each test we will encounter into one big flow chart like this:



**Fig 1.1. A flowchart of statistical tests we will encounter.**

At this point, I anticipate that you might have some questions, so I will try to provide some answers below. If your question does not appear here, ask your instructor.

### FAQ

**Do I need to know R to use this manual?**  
No, in fact this manual is meant to introduce you to R and RStudio for the first time.

**Do I need to know statistics to use this manual?**  
Yes, to some extent. This manual assumes you have previously completed one introductory statistics class. It assumes you are familiar with basic descriptive statistics (means, standard deviations, etc.), basic inferential statistics (*t*-tests, bivariate correlation, etc.), and foundational principles of their computation (normal distributions, standard (“z”) scores, etc.). If you are looking for a great introductory statistics textbook for psychology students, I recommend “Understanding Statistics in the Behavioral Sciences” by Robert Pagano. For a much more comprehensive introduction to programming in R for psychological experiments, check out Andy Field’s excellent textbook “Discovering Statistics Using R”.

**What is Markdown and why does it look different from other R scripts?**  
RMarkdown is a tool for creating openly-reproducible research reports and webpages using R. The author used it to put together everything you see in this manual. Markdown scripts allow you to run code, explain the choices you made when you created the code, and share those working codes and explanations with other people. It works alongside the base programming language R, and inside of the software RStudio.

**Will I have to hand this in for my lab? Will it be graded?**  
No, and no. This manual exists to help you organize the semester’s lessons into a more digestible format. It will not be graded, or even seen by your instructors (unless you ask for help with something).

**Does that mean I can choose to *not* fill in this manual?**  
Yes, but know that your instructor wouldn’t give you materials unless I thought they were useful. All homework assignments, both quizzes, and lots of material next semester will rely on code we will learn in class. The homework and quizzes will also be open book, so it is a very good idea to have everything you learn written down somewhere. If you strongly prefer another approach, but you are concerned that it might hurt your grade, ask your instructor about it.

**Can I ditch RMarkdown and just use regular R scripts?**  
Absolutely. But, this will not result in a finished Collaboratory Manual at the end of the semester unless you paste your working code snippets into the appropriate boxes of the manual document.

**What happens if my code does not run, or something looks wrong in my manual?**  
Your instructor can help fix whatever problem comes up, just ask!

**Uh oh! I broke something and now all the formatting looks wrong! What now?**  
Ask your instructor. They will help you find and fix the problem. If all else fails, re-download a copy of the manual and begin adding your work back into this new version until something breaks again. This will not be the last time you find yourself re-coding something you wrote earlier.

**Can we please learn some programming already?**  
Yes! Let’s get started!

Lesson 2 - Introduction to R

Last edited on: 2023-12-07

# Lesson 2 - Introduction to R

Welcome! In the last Lesson, we introduced the concept of report writing using RMarkdown and your Collaboratory Manual for this semester. In this Lesson, we will take our first steps into statistical programming using R and RStudio.

### Our objectives this lesson:

* Explore the RStudio interface
* Learn about objects and their classes in the R environment
* Learn to distinguish functions(), arguments, and packages
* Familiarize ourselves with the working directory
* Create #Comments in our codes
* Learn where to look for “? help”

As we do, make sure to look out for…

print("Code chunks where you will tell R to do something.")

## [1] "Code chunks where you will tell R to do something."

…as well as blank spaces (“\_\_\_\_\_\_\_\_\_\_”) where you will explain, report, or reflect on the code you wrote. Using code chunks and blank spaces together will help you to create working codes, and remember how they work later on.

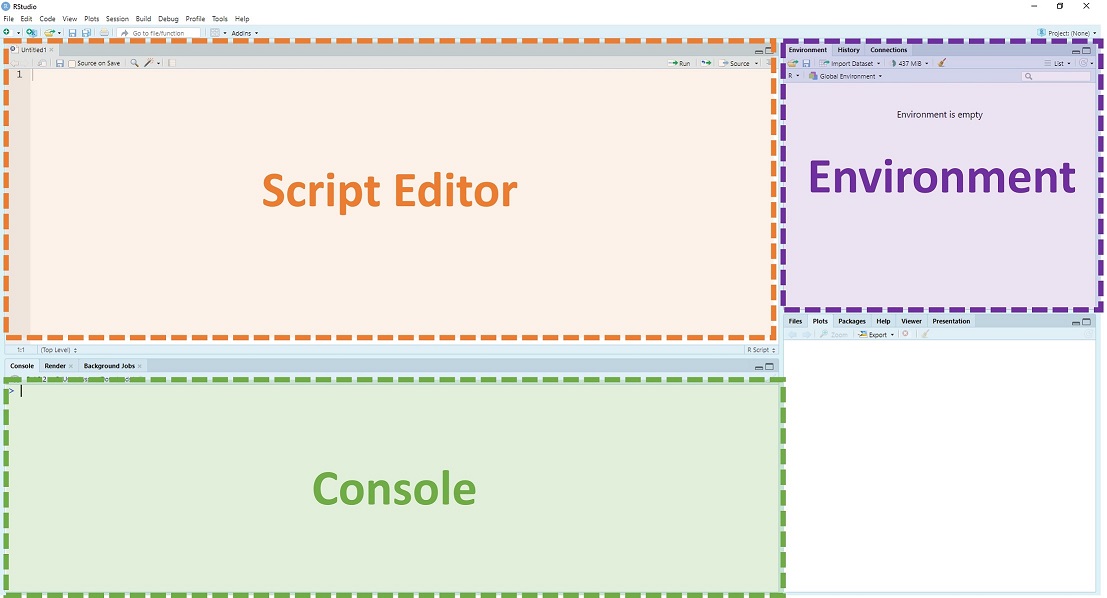
Without further ado, let’s get to our first objective.

## The RStudio interface

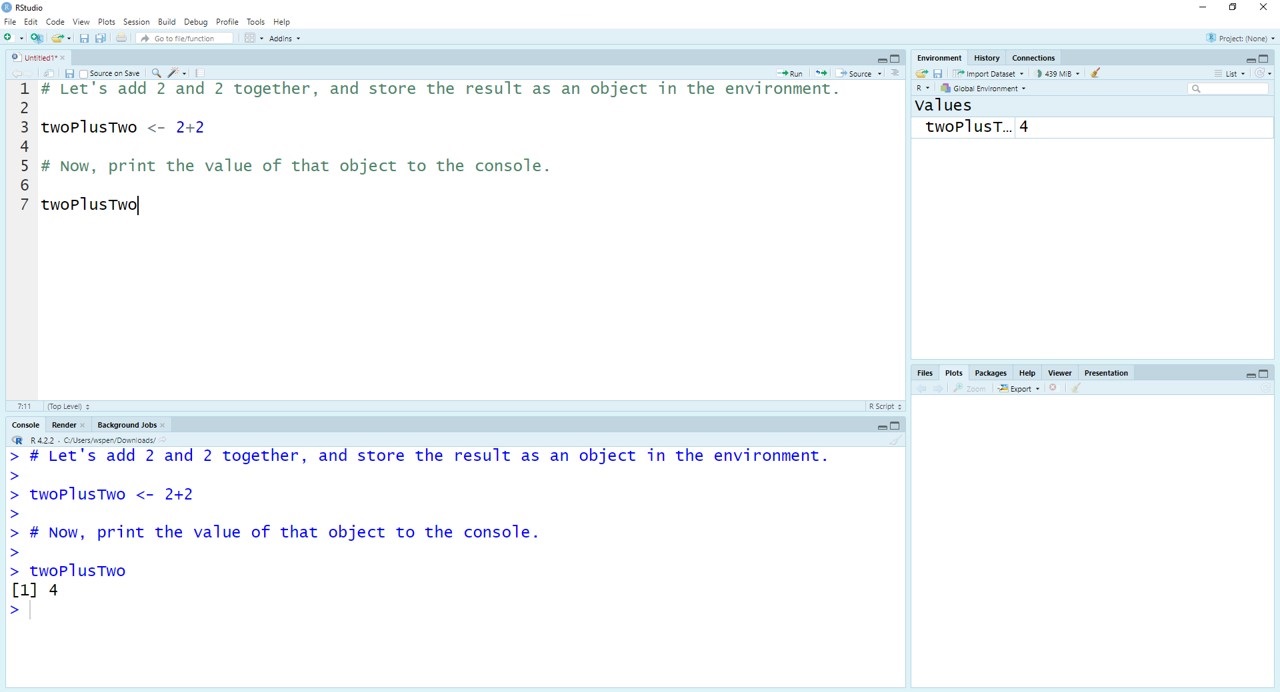
In the world of RStudio, we will spend most of our time looking in one of 3 places:

1. the script editor, where we will write our code,
2. the environment, where any objects we create will be stored, or
3. the console, where any code we run will be shown as output.

Here is a labeled diagram of the RStudio interface, and a simple example of what a workflow in R might look like if you wanted to compute 2+2:



**Fig 2.1. A labeled R interface**



**Fig 2.2. A workflow using the R interface**

**Activity**  
Take some time to play around in RStudio. Create a new script by clicking **File > New File > R Script**. Write something in the console and press enter. Does anything change in the script editor or environment? Write something in the script editor and click the button that says “Run”. Does anything change in the console? Finally, clean up your environment and console windows by finding and clicking the “broom” icon to remove everything you might have created and printed.

**Explain**  
When I want to run a line of code, I will…  
(“\_\_\_\_\_\_\_\_\_\_”)  
When I want to change my code, I will make those edits inside the…  
(“\_\_\_\_\_\_\_\_\_\_”)  
If I wanted to delete everything from the environment or console, I would click…  
(“\_\_\_\_\_\_\_\_\_\_”)

This activity was intended to familiarize you with RStudio, but you probably still have a lot of questions, and much of the time you probably felt like you were just clicking and typing randomly. In the next topic, we will start to do things intentionally. Let’s go!

## Objects and their classes in the R environment.

The most fundamental thing R does to enable statistical programming is the ability to create **objects** and store them in the R **environment**. The way you create an object is by using the operator <-, which looks like a little arrow pointing left. This symbol is called an assignment operator, because it assigns an identity to some object. Every object you define will be given some name that you will decide, and some kind of data that it represents. If you don’t assign an object when you write a piece of code, then the results of that code will just print to the console without being saved. You may have experienced this first hand during the activity in the previous lesson.

There are some important rules when it comes to naming objects:

1. object identities are case sensitive. This means that Thing1 and thing1 refer to two different objects.
2. objects generally should not have names that start with numbers, but can have numbers inside them (i.e., Thing1 is a valid object name, but trying to name an object 1Thing will be a real hassle).
3. object names can have periods or underscores (e.g., Object\_21), but other special characters are usually not allowed.

An object can be many different things, including a single number, a word, a graph, or a whole database. In order to print the identity of an object to our R console, we just have to run a line of code that calls its name.

Let’s look at one example of some objects, and what happens when I call their names. Notice that R does not print anything to the console when we assign an object; that only happens when we execute a different line of code that contains only the object’s name. This is because assigning objects and viewing them are two different operations. It is important to notice this difference because sometimes it might seem like your code was broken because nothing happened. In all likelihood, your code ran successfully but simply wasn’t printed to the console. You can check whether the command you executed did what you wanted by printing the object in the console (like I did for each of the commands in the code above), or by clicking on the name of the object in the environment pane.

First, an object called theMeaningOfLife, which I have assigned the number value 41.

Next, Let’s use some simple computation to make theMeaningOfLife equal to 42 instead of 41. Notice how we must use the assignment operator to *re-define* this object as something new. If we simply run the code theMeaningOfLife + 1, then R will print 42 to the console, but will not re-define the object theMeaningOfLife inside our Environment.

Next, let’s create an object called mostImportantObject, which I have assigned the text value "A towel.". Notice the use of quotation marks when assigning *strings* of text or characters.

Finally, a simple data frame (which will be the basis of all our analyses this semester), which I have called countToTen, and given one variable which simply counts to ten. Don’t worry yet about the code I used here, it will all be explained in the next section. For now, just notice that objects can be different things.

# Assign the object  
theMeaningOfLife <- 41  
  
# Call its name to print it to the console  
theMeaningOfLife

## [1] 41

# Redefine the object  
theMeaningOfLife <- theMeaningOfLife + 1  
  
# Call its name to print it to the console  
theMeaningOfLife

## [1] 42

# Assign the object  
mostImportantObject <- "A towel."  
  
# Call its name to print it to the console  
mostImportantObject

## [1] "A towel."

# Assign the object  
countToTen <- data.frame(seq(1:10))  
  
# Call its name to print it to the console  
countToTen

## seq.1.10.  
## 1 1  
## 2 2  
## 3 3  
## 4 4  
## 5 5  
## 6 6  
## 7 7  
## 8 8  
## 9 9  
## 10 10

Run the above code chunk in RStudio, and then take a moment to look at the Environment window. If you hover your mouse cursor over the name of each object we created, you will notice that RStudio tells you the object’s *class*. You can also use the class() function (see our next topic on functions) to print a specific object’s class to the console. There are several different classes of data that objects can exist as, including:

* Numeric - Can be any real number, or set of real numbers
* Character - Any kind of text, or set of texts, existing between “quotation marks”
* Factor - A categorical variable that has predefined levels and may have labels for those levels.
* Logical - Can have only two values, either “TRUE” or “FALSE”.
* Date - Indicate a specific time in a particular format (i.e., YYYY-MM-DD)
* Data frame - Holds multiple kinds of data in different variables. Each variable gets its own column, and each case (a research participant, laboratory animal, etc.) gets its own row. We will use many data frames this semester.

**Activity**  
Use the assignment operator (<-) to define R objects with the names and values described in the code chunk below.

# A numeric object called "X" with the value 10.  
  
  
# A numeric object called "Y" with the value pi.  
  
  
# A character object called "haiku" that is a haiku you wrote about R.  
  
  
# A logical object called "excited" that indicates whether the statement "I love learning R!" is TRUE or FALSE.

**Reflect**  
What is the class of our object “excited”, and how did you find that information?  
(“\_\_\_\_\_\_\_\_\_\_”)  
What happens if you click on the object “countToTen” in the Environment window? Can you imagine a situation where this might be useful?  
(“\_\_\_\_\_\_\_\_\_\_”)

## Functions(), arguments, and packages.

Many of the most common and complicated operations we will perform in R make use of **functions**. A function is just a chunk of code that has been written to perform a specific operation. You will recognize functions becuase their names are always followed by open and closed parentheses without spaces. For example: class(), mean(), and sd() are functions that return an object’s class identity, compute an object’s mean value, and compute an object’s standard deviation.

The specific operation that a function performs can often be modified by supplying the function with one or more **arguments**. Arguments are bits of information that go inside a function in order to tell it how to behave. For example, the argument “x = …” lives inside the mean() function, and allows you to dictate the values whose mean will be computed. Here is an example using a vector of ten numbers that includes [0,0,0,0,10,10,50,50,50,50]. Notice how I use the c() function to *combine* all 10 numbers into a single object.

# Define an object containing the ten numbers  
exampleNumbers <- c(0,0,0,0,10,10,50,50,50,50)  
  
# This first code prints the mean of the full object  
mean(x=exampleNumbers)

## [1] 22

# mean() also has an interesting argument that lets us 'trim off' some proportion of scores from   
# either end of our object before computing the mean. Let's remove 40% (or 0.4/1.0) of the scores from   
# the start and end of our vector, so that only the middle 20% of scores (10, and 10) remain.  
mean(x=exampleNumbers,trim=0.4)

## [1] 10

**Packages** are, in the simplest sense, just an assortment of functions that have been written by a person (or people), and then bundled together so they can be shared over the internet and downloaded by you. This can be extremely useful if you need to perform a specific set of statistical tests, and you do not want to program all the code by hand. By downloading a package, and loading its functions into your R Library, you can perform complicated statistical tests very quickly and easily.

A brief word of caution: R is an open-source software, and the packages you can load into R will not always be completely free from other people’s coding errors or mathematical issues. Sometimes, it will be up to you to check that a package or function is providing sensible outputs. This is especially important if you are using an R package that is recently-released or uncommonly-used, or if you are using a popular package in an unusual way. Popular packages (though likely not entirely error-free) are generally trustworthy. The more esoteric your analyses become, the more careful you will need to be.

In order to download packages, you can use the install.packages() function, with the name of the package entered as an argument in quotation marks. To then use those packages in an R code, you will need to use the library() function to load them from your library of downloaded R packages. You only need to download packages once, but you need to load necessary packages every time you start RStudio. R does not load all your packages by default, because this could very quickly take up many computers’ memory capacity, slowing the whole system down[[1]](#footnote-1). It is best to load only the packages you need for a particular task or analysis.

As an example, let’s download and load two packages we will use in Lesson 3: “ggplot2”, which we will use for making plots, and “dplyr”, which we will use for data cleaning. First, paste this code into your console and press enter:

**install.packages(“ggplot2”)**  
**install.packages(“dplyr”)**

*Note: if you receive an error regarding the rlang package, you may need to first run:*  
*remove.packages(“rlang”)*  
*This is a known issue with installing ggplot2. Sorry!*

Next, load your installed packages by running this code chunk. Notice that the package names no-longer need to be surrounded by quotation marks. This is a silly difference between the designs of the two functions. Since R functions are written by different people all over the world, you will find that their conventions are not always consistent.

library(ggplot2)  
library(dplyr)

*Note: You will sometimes notice some warnings appear when you run the library() function for a particular package. Most often, this occurs because two or more packages contain functions with the same name. Make note of these and consider explicitly specifying which package a function comes from if you notice that a function is behaving oddly. You can specify functions inside of packages using two colons between the package name and function name (for example, dplyr::filter())*

**Reflect**  
“Packages go inside arguments, which go inside functions.” Is this statement true or false? Why or why not?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Can you think of a situation when you might want to ‘trim’ some data before calculating their mean?  
(“\_\_\_\_\_\_\_\_\_\_”)

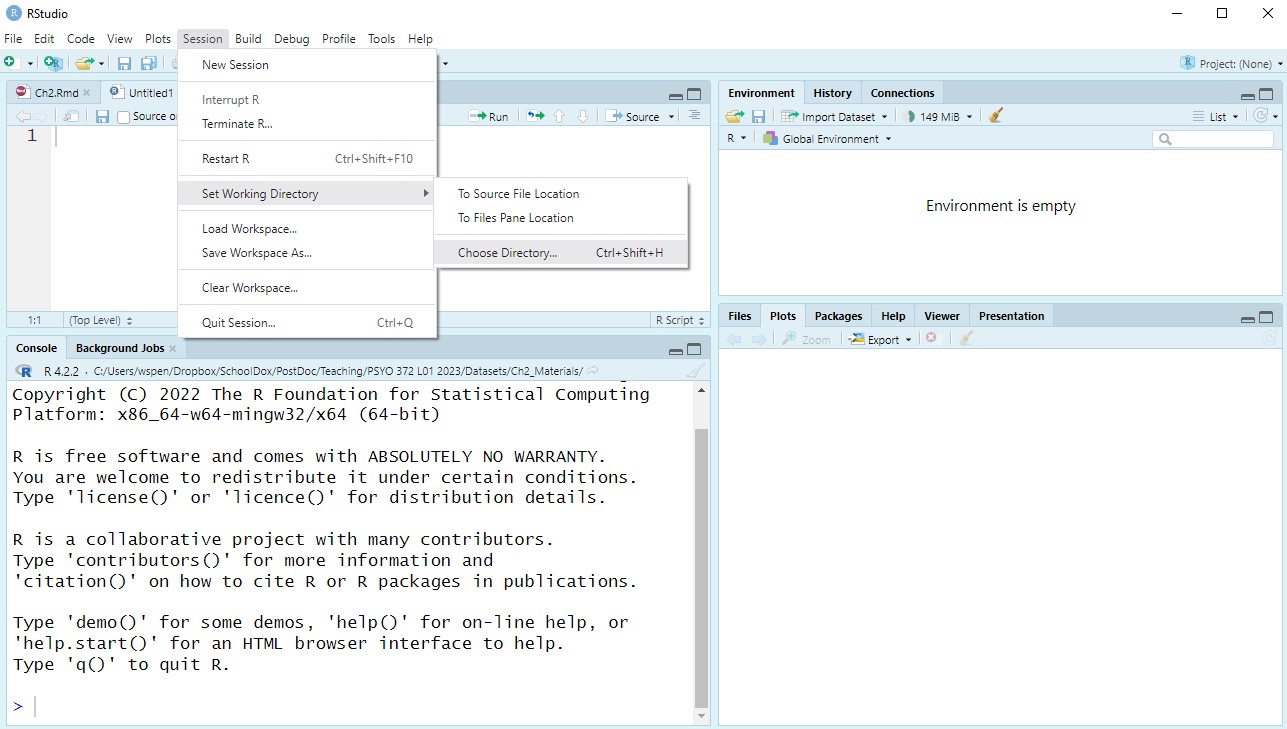
## The working directory

Every time you want to load a dataset using R, and every time you want to save the results of your statistical analyses to your computer, you need to tell R what file folder to look inside. R always has one file folder open, and this is called the “working directory”.

You can find the current working directory by running the code chunk below.

currentDirectory <- getwd()

Now, you could try to run every R analysis inside that same location, but we will often want to save the results of our analysis inside a different file folder. So, we will want to *set* the working directory to be someplace else. To make R look inside a particular folder, we need to tell it where to look. The easiest way to do this in RStudio is to use the “Set Working Directory” option under the “Session” tab. Simply navigate to the folder you wish to set as the working directory for this session, and select it. Your choice will be reset after you close RStudio.



**Fig 2.3. Setting the working directory**

For scripts that you plan to run often, it can be a good idea to use a function() called **setwd()**, and tell it where to set the working directory. Work with your instructor and classmates to edit the code below so that it changes the working directory to your computer’s desktop. Presently, this code only works on my computer. You will need to find the unique **path** that leads to your desktop using your computer’s file manager. The example below shows the path to my personal computer’s Desktop. This code will not work for you because your computer has different paths than mine. You will need to edit this code to identify an existing location on your own computer. Pay close attention to the direction of the slashes in your path name!

setwd("C:/Users/wspen/Desktop")

**Reflect**  
What file path did R print to the console when you ran the first code chunk?  
(“\_\_\_\_\_\_\_\_\_\_”)

Why do you think the first *function()* we used was called “getwd()”? Why was the second one “setwd()”?  
(“\_\_\_\_\_\_\_\_\_\_”)

## #Commenting your codes.

Anyone who has done coding work in a team environment will tell you that it is sometimes near-impossible to understand what your teammates’ code is doing unless they provide a written explanation. Well, way back in the *bad-old-days* of computational statistics (just a few years ago, really), R users didn’t have Markdown to act as a living laboratory notebook, and so any time they wanted to explain the purpose of a particular line of code, they would have to create a **#comment** about it.

This creates an unpleasant tension inside your code: on one hand, you want to have code that is easy to follow inside a concise computer script, but on the other hand, you want to have comments that are detailed enough for other people to understand what you’ve done and why.

All of this is to say that, sometimes, you may be writing an R script (or a ‘code chunk’ if you are working in RMarkdown), and you may feel the urge to explain what you are doing (in fact, commenting your code as you write it can often help you to keep track of where you’re trying to end up). When this happens, you will want to use the “#” symbol, so that R knows not to try and run the comment as if it were code. Try to make a habit of commenting or explaining each chunk of code you write. It will save you many headaches during your career in psychological science!  
Here is an example of a comment inside a code chunk:

# This comment explains that my code will compute the mean of 10, 20, and 40. It will then print the result to the console  
# The code below will actually carry out that operation.  
mean(x=c(10,20,40))

## [1] 23.33333

**Activity**  
Write an informative comment inside this code chunk to explain what it is doing.

# (YOUR COMMENT GOES HERE)  
myGrade <- 76.5  
classAverage <- 72.0  
classStandardDeviation <- 8.5  
  
myZscore <- (myGrade - classAverage) / classStandardDeviation  
myZscore

## [1] 0.5294118

**Reflect**  
What is the symbol to use when you want to create a comment?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Would you ever consider writing a comment to your future self? Why or why not?  
(“\_\_\_\_\_\_\_\_\_\_”)

Great work! Now, it’s time for our final topic in this Lesson.

## Where to look for ? help

If you ever get stuck trying to figure out something about a function or its arguments, **“?”** followed by the function’s name can be run in RStudio to find the documents that go with any function you are using. This is also a good way to find our if there is a function that already does a complicated procedure you would rather not program by hand. Try entering “?” followed by a word that describes the procedure you are trying to do. But make sure to read the documentation carefully to see how it’s used and what it does. It might not be exactly what you want!

Paste the following code into the console and press enter to learn more about the mean() function:

**? mean**

**Other places to find help when you get stuck:**

* Google is a wonderful resource. If you get an error code printed out in your console, something about your code is probably incorrect! Try copying the exact error code, and entering it into google (make sure to remove any sensitive data). Almost every possible R problem you will encounter has been asked and answered on websites like StackOverflow.
* ChatGPT can provide excellent summaries of general functionality in R. Try asking chatGPT “Explain data classes in the R programming language” (Reminder: never paste sensitive data into ChatGPT).
* Your instructor and classmates will also be able to help you problem solve. Remember, we work as a team and there is no shame in asking for assistance!

We have reached the end of another challenging lesson in R statistics! Before you go, make sure to write down your three major takeaways and review their key concepts when we come back next week.

**Key Takeaways**  
The most important lessons from this Lesson were:  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)

Lesson 3 - Data Cleaning and Visualization Using *dplyr* and *ggplot2*

Last edited on: 2023-12-07

# Lesson 3 - Data Cleaning and Visualization Using *dplyr* and *ggplot2*

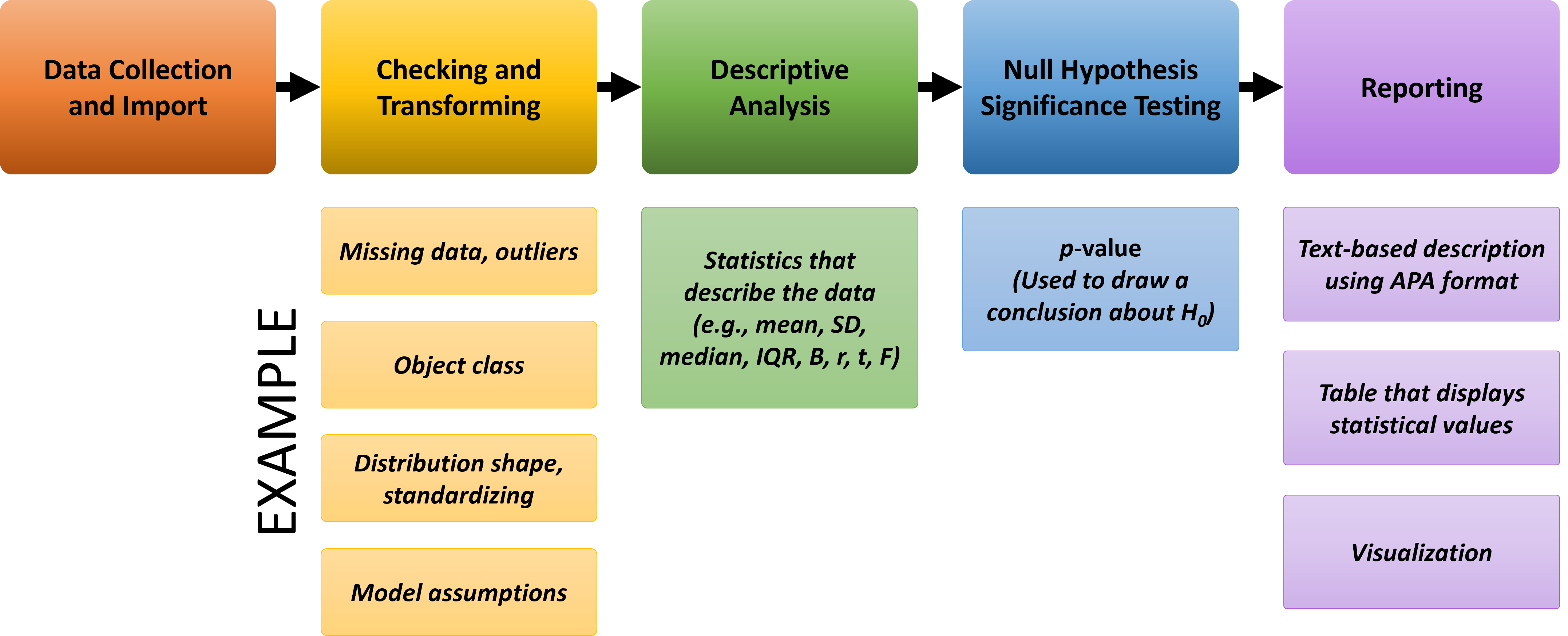
Welcome! Now that we have introduced the core concepts of working with R, we will turn our attention to the main purpose of this laboratory: data analysis and reporting in the psychological sciences.

### Our objectives this lesson:

* Consider the data analysis pipeline
* Learn to do manual data entry
* Learn to import data from a saved file (.csv)
* Try several tools for data cleaning
* Compute new columns (also known as variables) in a data.frame
* Extract some descriptive statistics
* Create visualizations using ggplot2
* Make accessible graphics for diverse audiences

## The data analysis pipeline

Starting now, and continuing to the end of the semester, we will structure every lesson around some aspect of **The Data Analysis Pipeline**. You can think of this pipeline as a kind of mental shortcut that explains how to logically approach data cleaning, visualization, descriptive analysis, hypothesis testing, and reporting. The figure below depicts the pipeline, and provides some concrete procedures you might perform during a particular analysis.



**Figure 3.1. The Data Analysis Pipeline.**

**Reflect**  
Consider Figure 3.1 above. What might happen if a psychologist decided to skip the “Checking and Transforming” step?  
(“\_\_\_\_\_\_\_\_\_\_”)  
What might happen if they skipped the “Descriptive Analysis” step?  
(“\_\_\_\_\_\_\_\_\_\_”)

## Manual data entry

As you can see, the first task we will need to accomplish for any analysis is to import whatever data we have collected from our experiment. The most basic (and worst) way to do this is by manually entering all your data into R. For example:

# Create a new data frame with some scores  
manualData <- data.frame(c(1,2,3))  
  
# Use the colnames function to give the first variable a name.  
colnames(manualData) <- "Variable1"  
  
# Add a new variable and give it the name "Variable2"   
# (see the topic on this in "Computing new columns" below)  
manualData$Variable2 <- c(4,5,6)  
  
# Notice that data frames can have different variables, each with a different class.  
# Here's a new variable that has the class "character" because it contains text.  
manualData$TextVariable <- c('your','text','here')  
  
# Now notice that data frames cannot handle missing data unless it is noted with "NA"  
manualData$IncompleteVariable <- c(1975,1986, NA)

Phew! Even with just 3 rows of data, manually entering these scores is *tedious*. There must be a better way (*the writer exclaimed, smugly setting up the next topic*)!

## Importing data from a saved file (.csv)

Most people working with R will prefer to load their already-entered dataset in the form of a Comma Separated Values file (.csv). This format is easy for R (and other statistical programs) to read, and can be easily entered and saved using a spreadsheet program like Microsoft Excel or Google Sheets (reminder not to save sensitive participant data on any Google server, as this may violate participant confidentiality and FOIPPA).

Let’s import a simple dataset containing 29 research participants who completed a Stroop Task. The idea with the Stroop Task is that people will be shown colour words (“red”, “yellow”, etc.) on a screen, and each word will have a colour that either matches that word, or does not match that word. People need to correctly identify the colour of the word, and generally most people can identify word colours more quickly when the word is the same colour as (or “congruent to”) the colour that it is painted.

For each participant, we have an average response time in milliseconds when the words and colours match (“ResponseTimeMSCongruent”), and we have their average response time in milliseconds when the words and colours do not match (“ResponseTimeMSIncongruent”).

Run the code chunk below to load the data.

stroopDataset <- read.csv("StroopTaskData.csv")

**Reflect**  
What is one thing you notice about this data frame?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Do you see any issues with this data frame that might cause problems for our analysis?  
(“\_\_\_\_\_\_\_\_\_\_”)

*Note: If you wanted to save a dataset from R instead of loading a dataset into R, you could use the function write.csv() with the arguments X = to indicate the object you want to save, and file = “” to indicate the file name you will give the new csv. file when it is saved in your working directory. For example, this code saves the dataset we just loaded under a new file name.*

write.csv(x=stroopDataset, file = “stroopDatasetNewSave.csv”)

## Data cleaning

A (now-pretty-old) 2016 survey conducted by the data science firm CrowdFlower found that data scientists report spending nearly 80% of their time on curating and cleaning datasets to get them ready for analysis. Although this single survey was conducted several years ago, I can attest to the fact that this reality has not changed: the most time-consuming and often challenging portion of the data analysis pipeline is data cleaning and checking.

There are several reasons for this fact:  
1. You absolutely must know what your raw data look like if you are to have any confidence in your analyses.  
2. Instrument malfunctions, disingenuous research participants, and other random happenings can create severe outliers that need to be found and resolved before analyses are performed. If not, the analysis can be dead wrong.  
3. Many statistical tests only work as-intended if the variables they consider approximate a particular distribution of scores. If not, the analysis can be dead wrong.

Although data cleaning and checking can feel extremely tedious, you can take some solace in the fact that it is definitely necessary. As far as R is concerned, there are a few core skills that will be needed for handling problematic scores in any given data frame. That is the purpose of this topic.

### Indexing specific rows and columns

Take another look at the data frame we loaded, stroopDataset. I will use the head() function to show the first 10 rows of data below. Those following along inside RStudio may have noticed that there are two big problems with the data that were entered:

# head() returns the first (or last) n parts of an object, x.  
head(x = stroopDataset, n = 10)

## Student ResponseTimeMSCongruent ResponseTimeMSIncongruent  
## 1 1 757.728 NA  
## 2 2 754.855 969.368  
## 3 3 810.734 1062.622  
## 4 4 808.872 1021.044  
## 5 5 829.647 1022.208  
## 6 6 778.325 1096.442  
## 7 7 820.743 1048.550  
## 8 8 810.131 1071.957  
## 9 9 822.638 1092.611  
## 10 10 NA NA

1. There are missing scores on one or both variables for Students 1 and 10 (and 27, although it is not displayed above).
2. There are some surprisingly extreme outlying scores on one or both variables for Students 16 and 22.

We can deal with both of these problems relatively easily using Base R (that is, the packages that came bundled with our original download of the R software).

First, we will deal with the outlying scores that we identified. If we cross-reference the row and column number of each outlying score, we can see that Student 16’s outlier is in the **cell** located on Row 16 and column 3. In R, we could get this value by adding square brackets with the row and column brackets to our data frame, like this:

stroopDataset[16,3]

## [1] 10044.91

Similarly, Student 22’s outlier is located in the cell on row 22 and column 2. We can locate it with the code:

stroopDataset[22,2]

## [1] 82.188

But knowing the location of an outlier is hardly useful unless we do something with that knowledge. So, let’s replace both outliers with the value NA to indicate that these cells should be empty.

stroopDataset[16,3] <- NA  
stroopDataset[22,2] <- NA

Now, in some cases we could carry on to our analysis without doing anything else to the Stroop Task dataset. But, it is important to know that the mere presence of NA scores in a dataframe can create serious headaches for various functions later on. For example, if we try to run the mean() function on one of our variables, by default the function will return NA. This is because the mean of the scores is unknown when missing values are present. The NA tells R that a number should be there, but it is missing—we do not know what it is. The average of the scores is unknown because some of the scores are unknown.

Some functions, including mean(), contain the argument na.rm, which tell R whether missing values should be removed prior to computation (the rm is short for remove). Typically, the default value to na.rm will be na.rm = FALSE. Changing this to na.rm = TRUE indicates it is TRUE that NA scores should be removed from analysis. Try using mean() to calculate average congruent response time with missing values removed.

The benefit of na.rm argument is that it allows you to remove missing values for one computation without permanently removing cases from your data frame. But not all functions are convenient enough to include na.rm. For this reason, and depending on the analysis you are planning to run, it may be wise to exclude rows of data that contain one or more NA. For example, let’s create a new data frame called stroopExcludeNA, which uses the na.omit() function to remove rows with NA values. We will give this function an argument that identifies the object we want to run na.omit() on.

stroopExcludeNA <- na.omit(stroopDataset)

**Reflect**  
What is one thing you notice about the rows in this new data frame?  
(“\_\_\_\_\_\_\_\_\_\_”)  
What kind of t-test would you run on paired data like these? Based on your answer, is it ok that we removed rows where only one score was missing?  
(“\_\_\_\_\_\_\_\_\_\_”)

Recall that when we replaced the outlier scores with NA, we identified the cells by their row and column numbers. This is called **indexing** and it is just one of many ways to identify specific cells. If we had a very large dataset for which it would take a very long time to identify outliers by hand, we may need to use a more sophisticated approach. That’s where the **dplyr** package comes in.

### Data cleaning tools from the *dplyr* package

In the last lesson, we installed the dplyr (pronounced “DEE-PLY-ER”) package. Let’s make sure that it is loaded by running this code chunk. Recall that we use the library() function to load packages, and give the package’s name as the function’s sole argument.

library(dplyr)

Now, let’s talk about what dplyr does that can help us with data cleaning. Dplyr is going to be most helpful when we are working with experiment data stored in a data frame. There are three very helpful functions inside the dplyr package that I want to highlight at this stage in the course:

1. arrange() - This function will allow us to sort a data frame based on the values in one of its variables. This will make it very easy to see if you have a few extreme outliers inside a specific variable.
2. select() - This function will allow us to reduce cluttered data frames by removing variables from them.
3. filter() - This function will seriously improve our ability to remove bad rows of data from a data frame.

The argument order for these functions was designed to work a slightly special way. As we will see, dplyr functions are first expecting to see the name of our data frame, then a comma, then the name of whichever variable (or variables, again separated by commas) we are trying to modify. This will sometimes (but not always) be followed by a logical operation, which we will explore in detail when we look at filter().

Let’s start with arrange(). Looking at our data frame called *stroopDataset*, we can ‘arrange’ participants’ scores on the variable *ResponseTimeMSIncongruent* from lowest to highest by giving arrange() two important arguments: first the name of the data frame, then a comma, then the name of the variable inside that data frame that will be arranged. [[2]](#footnote-2) In order to save our results to the data frame itself, we will redefine our data frame as the arranged version of itself. Run the code chunk below, and then write down some thoughts or reactions in the reflection questions beneath it.

# Notice also that argument's names (e.g., "X = ") do not need to be given  
# if they are entered in the correct order.  
stroopDataset <- arrange(stroopDataset, ResponseTimeMSIncongruent)  
  
# Use the head() function to show how it changed.  
head(x = stroopDataset, n = 10)

## Student ResponseTimeMSCongruent ResponseTimeMSIncongruent  
## 1 15 845.618 950.339  
## 2 20 759.582 952.305  
## 3 12 843.985 952.867  
## 4 2 754.855 969.368  
## 5 23 845.840 987.450  
## 6 14 800.588 1017.724  
## 7 4 808.872 1021.044  
## 8 5 829.647 1022.208  
## 9 22 NA 1025.477  
## 10 18 831.121 1040.775

**Reflect**  
What did you notice happened to the variable “Student” when you arranged the data frame by ResponseTimeMSIncongruent?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Is this a good thing, or a bad thing? Explain.  
(“\_\_\_\_\_\_\_\_\_\_”)  
How might you use the arrange function to return the stroopDataset data frame back to the order it previously had?  
(“\_\_\_\_\_\_\_\_\_\_”)

Now, let’s look at the select() function. Since we have two different measurements for the participants in this experiment, we will probably want to run some kind of paired-samples t-test in order to determine if the variables *ResponseTimeMSCongruent* and *ResponseTimeMSIncongruent* differ significantly from one another. You will remember from your introductory statistics course that this is computationally identical to running a single-sample t-test on a set of *difference* scores where *ResponseTimeMSCongruent* has been subtracted from *ResponseTimeMSIncongruent*.

For the sake of having a clean data frame, let’s go through the process of computing a new variable of difference scores called “DifferenceScore”, and then creating a new data frame object called “stroopDatasetSingleSample” that is made up of the Student and DifferenceScore variables.

Just like arrange(), select() wants to see at least two arguments: the name of the data frame you will be selecting variables from, and the name(s) of any variable(s) you will be selecting from it (separated by commas)[[3]](#footnote-3).

# First calculate our new variable made of difference scores inside the existing data frame stroopDataset.  
stroopDataset$DifferenceScore <- stroopDataset$ResponseTimeMSIncongruent - stroopDataset$ResponseTimeMSCongruent  
# Note: I have used the dollar sign '$' here to indicate specific variables inside a data frame.  
# see the topic called "Computing new columns (also known as variables) in a data.frame" below for a full explanation.  
  
# Next, create the new data frame object as a selection of the variables Student and DifferenceScore.  
stroopDataSetSingleSample <- select(stroopDataset,Student,DifferenceScore)  
  
# Use the head() function to show how it changed.  
head(x = stroopDataSetSingleSample, n = 10)

## Student DifferenceScore  
## 1 15 104.721  
## 2 20 192.723  
## 3 12 108.882  
## 4 2 214.513  
## 5 23 141.610  
## 6 14 217.136  
## 7 4 212.172  
## 8 5 192.561  
## 9 22 NA  
## 10 18 209.654

**Reflect**  
What might be one benefit of being able to remove certain variables from a data frame?  
(“\_\_\_\_\_\_\_\_\_\_”)  
What happens if you run the code “select(stroopDataset,Student,DifferenceScore)” without creating a new object that the code is assigned to?  
(“\_\_\_\_\_\_\_\_\_\_”)  
What happened in cases where NA was subtracted from a scrore that was not NA?  
(“\_\_\_\_\_\_\_\_\_\_”)

Now let’s look at the filter() function. The simplest way to understand this function is to notice that, while select() worked on columns in a data frame, filter() works on rows. There will be situations when it is necessary to remove certain rows of data from a data frame. We saw one such case above, when we learned about the na.omit() function from base R.

Just like select(), filter() needs at least two arguments in order to work properly. The first argument is the name of the data frame you will be filtering, and the second (and any subsequent) argument will be the name of a variable involved in the filtering. Unlike select(), you will also need to explain *how* a particular variable should be filtered using logical rules. This is our first brush with **logical operators** in R. This is a lengthy topic in the world of computer science, but instead of spending a whole lesson on it, I want to highlight just a few logical operators that we will be using often. In each of the examples below, run the code chunk and then write down the rule that has been used to filter out certain cases.

First, this new object called “filterOne” is the same as our stroopDataset data frame, except that some Students have been removed.

filterOne <- filter(stroopDataset, Student < 15)

What logical operator was used, and what does it seem to do?  
(“\_\_\_\_\_\_\_\_\_\_”)

This new object called “filterTwo” is the same as our stroopDataset data frame, except that other Students have been removed.

filterTwo <- filter(stroopDataset, Student >= 15)

What logical operator was used, and what does it seem to do?  
(“\_\_\_\_\_\_\_\_\_\_”)

This new object called “filterThree” contains just one participant.

filterThree <- filter(stroopDataset, Student == 15)

This logical operator tests for equality. It looks a bit like the equals sign = we are used to seeing for functions’ arguments, but it only works in cases where we are *testing equality* (i.e., we want to know if it is TRUE or FALSE that two things are exactly equal). It will not work in cases where we want to assert, for example, that x = 3.

This new object called “filterFour” contains all but one participant.

filterFour <- filter(stroopDataset, Student != 15)

What logical operator was used, and what does it seem to do?  
(“\_\_\_\_\_\_\_\_\_\_”)

Finally, it is crucial to note that functions can sometimes be used inside other functions. For example, we could use the function is.na() on our filtering variable in order to keep only the participants’ whose score on that variable is “NA”. Alternatively, we could add in the logical operator “!” to keep only the participants’ whose scores are *not* “NA”. In effect, we are recreating the functionality of the na.omit() function by combining the different functions of filter() and is.na(). This is crucial when you only want to remove rows where NA values are present for a specific variable that you care about.

filterFive <- filter(stroopDataset, !is.na(ResponseTimeMSCongruent) & !is.na(ResponseTimeMSIncongruent))

**Explain**  
When I want to remove a column or variable, I will use [arrange()/select()/filter()]  
(“\_\_\_\_\_\_\_\_\_\_”)  
When I want to remove some rows, I will use [arrange()/select()/filter()]  
(“\_\_\_\_\_\_\_\_\_\_”)  
When I want to visually inspect my data for outliers, I will use [arrange()/select()/filter()]  
(“\_\_\_\_\_\_\_\_\_\_”)

**Reflect**  
What do you think the “&” logical operator does?  
(“\_\_\_\_\_\_\_\_\_\_”)  
What might be one situation where you want to filter rows based on NA values in just one variable?  
(“\_\_\_\_\_\_\_\_\_\_”)  
To summarize our work on this topic, write a short note explaining the purpose of the logical operators “>”, “<”, “>=”, “<=”, “==”, “!=”, and “&”.  
(“\_\_\_\_\_\_\_\_\_\_”)

Now it’s on to the next topic!

## Computing new columns (also known as variables) in a data.frame

This is one of the most important topics in your introduction to R. It concerns mathematical computation and the way R data frames use the dollar sign “$” to indicate the identity of a variable. We used both these principles in the last lesson when we computed a new variable made of difference scores. That code looked like this:

stroopDataset$DifferenceScore <- stroopDataset$ResponseTimeMSIncongruent - stroopDataset$ResponseTimeMSCongruent

If we wanted to describe this code in words, we might say that we followed these steps:

1. We defined a new variable inside the existing stroopDataset data frame, and said that this variable should be named “DifferenceScore”.
2. We assigned this new variable to have an identity equal to each score on ResponseTimeMSIncongruent minus each score on ResponseTimeMSCongruent.
3. In making reference to these two existing variables, we had to tell R that they are found inside of the existing data frame object called stroopDataset.

From this example, you can see a couple important things. First, the “$” is used by R to both create, and refer to, variables inside of data frame objects. Second, we can use mathematical symbols to easily compute new variables based on information that we already have inside our data frame. Common mathematical operators you will use include:

* +, indicating addition.
* -, indicating subtraction.
* \*, indicating multiplication.
* /, indicating division.
* ^2, indicating that whatever value comes before will be squared (or cubed if you use ^3, etc.)
* sqrt(), a function used to find the square root of whatever number appears inside it’s parentheses.

**Activity**  
Create two new variables called “CongruentSeconds” and “IncongruentSeconds” in the code chunk below. Make sure they appear inside the data frame stroopDataset. Make their values equal to the values of “ResponseTimeMSCongruent” and “ResponseTimeMSIncongruent”, but set the unit of measurement to seconds, rather than milliseconds.  
Hint: There are 1,000 milliseconds in 1 second.

# Create the variable CongruentSeconds  
stroopDataset$CongruentSeconds <- stroopDataset$ResponseTimeMSCongruent/1000   
  
# Create the variable IncongruentSeconds  
stroopDataset$IncongruentSeconds <- stroopDataset$ResponseTimeMSIncongruent/1000

## Extracting descriptive statistics

So far in this lesson, we have looked at important tools and techniques for data checking and transforming. There are many other tools we will use for this purpose, including specific functions for plotting and transforming the distributions of different variables. We will return to these other data transformation procedures during specific lessons on inferential analysis using t-tests and regression. But first, we are going to take a brief step forward into the next section of the data analysis pipeline: descriptive analysis.

R is extraordinarily helpful when it comes to extracting many of the basic descriptive statistics we will need to understand our variables. Most often, these statistics can be easily computed using a specific function whose primary argument, “x =”, identifies the variable or set of scores about whom the descriptive statistic will be computed. Let’s take a quick look at some of the ones you will use most often:

* mean() computes the arithmetic mean.
* min() gives you the smallest (‘minimum’) score.
* max() gives you the larges (‘maximum’) score.
* sd() gives you the standard deviation, and sd()^2 gives you the variance (remember that a variance is just a squared standard deviation).
* median() gives you the median of your data.
* quantile() is a special case that can give you the 0th, 25th, 50th, 75th and 100th percentile of your data all at once. You can also add a second argument called “probs =” to ask for a specific percentile score. Try this one with the variable ‘stroopExcludeNA$ResponseTimeMSIncongruent’ from our earlier topic, and see what happens!
* For categorical data, table() will give you a frequency table that tells you how many cases are found in each category. We will use table() many times over the course of this semester.

**Activity**  
Working inside the data frame ‘stroopExcludeNA’, which we created earlier in this lesson, manually enter a new variable called “earlytParticipant” that uses the words “Yes” and “No” to distinguish participants with scores less than 10 on the ‘Student’ variable.  
Then, use the table function to find out how many participants are in each group on the earlyParticipant variable.  
Next, use the filter() function to create two new data frames that separate the people whose earlyParticipant score is “Yes”, from those whose score is “No” (remember to include the quotation marks).  
Finally, compute a mean and standard deviation for the variable “ResponseTimeMSIncongruent” in each of these new data frames (remember to use the data frame name and “$” symbol).

# I have done the first step for you, because it is tedious.  
stroopExcludeNA$earlyParticipant <- c("Yes","Yes","Yes","Yes","Yes","Yes","Yes","Yes","No","No","No","No", "No","No","No","No","No","No","No","No","No","No","No","No")  
  
# Use table() to find out how many participants are in each group on earlyParticipant  
  
# Use the `filter()` function to create two new data frames that separate Yes and No  
  
# Compute a mean and standard deviation for "ResponseTimeMSIncongruent" in each new data frame

That last activity was the most challenging we’ve seen so far! If you got stuck, remember to ask your classmates and instructor for help. These kinds of questions will keep coming up as you continue performing statistical analyses using R.

## Creating visualizations using ggplot2

For the remainder of this lesson, we will dive into the world of data visualization using the most popular R package for that purpose, *ggplot2*. Over the course of this topic, we will create bar charts, scatterplots, histograms and boxplots. To do this, we will import two different datasets that exist by default in Base R and ggplot2. The first, which we will call “carsData” contains information about 32 different models of car. This includes things like the number of cylinders in a given car’s engine, and its fuel efficiency in terms of miles per gallon, mpg.

The second dataset, which we will call ‘diamondsDataset’, contains almost 54 thousand (!) records of different diamonds. It includes measurements of each stone’s cut, carat, color (note the American spelling on this variable name), and clarity. This is the first, but not last ‘big’ dataset we will be handling during the course.

Run the code chunk below to load the “cars” dataset. Notice how we will assign the dataset we are loading to an object we will choose to call “carsDataset”. Ditto for the “diamonds” dataset. Notice once again how the function that loads csv files is called read.csv(), and it needs one argument to be provided which is the full file name (including “.csv” and all in quotations).

We will also make sure to load ggplot2 using the library() function.

# Things got a little messy in the last lesson. Let's run this code to empty the environment:  
# Note: this is the same as using the 'broom' icon in your RStudio environment.  
# If you're curious how this code works, try asking R with the code   
# ? rm()  
# ? ls()  
rm(list=ls())  
  
# Now that that's done, let's start fresh by loading two new datasets.  
carsDataset <- read.csv("mtcars.csv")  
diamondsDataset <- read.csv("diamonds.csv")   
  
# Print the first few rows of the mtcars dataset  
head(carsDataset)

## X mpg cyl disp hp drat wt qsec vs am gear carb  
## 1 Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4  
## 2 Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4  
## 3 Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1  
## 4 Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1  
## 5 Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2  
## 6 Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1

# Print the first few rows of the diamonds dataset  
head(diamondsDataset)

## X carat cut color clarity depth table price x y z  
## 1 1 0.23 Ideal E SI2 61.5 55 326 3.95 3.98 2.43  
## 2 2 0.21 Premium E SI1 59.8 61 326 3.89 3.84 2.31  
## 3 3 0.23 Good E VS1 56.9 65 327 4.05 4.07 2.31  
## 4 4 0.29 Premium I VS2 62.4 58 334 4.20 4.23 2.63  
## 5 5 0.31 Good J SI2 63.3 58 335 4.34 4.35 2.75  
## 6 6 0.24 Very Good J VVS2 62.8 57 336 3.94 3.96 2.48

# load ggplot2  
library(ggplot2)

Before we start plotting, let’s take a moment to talk about the ggplot2 package. the ggplot2 package has one core function that will be involved every time we make a plot. It is called ggplot(). The ggplot() function, like dplyr() always requires that its first argument identify the data frame we are using to make the plot. So, a plot made using the carsDataset data frame will create an object that is defined as a ggplot with carsDataset as its first argument, like this:

myFirstPlot <- ggplot(carsDataset)  
myFirstPlot

A white rectangular object with a black border

Description automatically generated

*Oh boy, an empty white square! ggplot is quite impressive!*

In all seriousness, you are likely wondering why our plot object (myFirstPlot) just looks like an empty white box. This is because ggplot() works to construct visualizations in **layers** using different functions that are connected by the symbol “+”. To make things even more complicated, these layers can be affected in two different ways by adding either **geoms** (or ‘geometric objects’) or **aesthetics**. Geoms are functions that tell ggplot what kind of visualization to make, and aesthetics tell those geoms what data from our data frame to use and how to use it. On top of all that, ggplot() also works alongside extra functions that add different layers to visualizations to change how they look (for example, the labs() function allows us to modify the titles and labels that appear on our plot).

This is a lot to grapple with, but I have found that the best way to learn is by creating an empty plot, and then adding things until it looks right. So, let’s first try to work towards a scatterplot using the carsDataset data frame and its variable “cyl”, the number of cylinders, and “mpg”, the fuel efficiency in miles per gallon.

In the next step, we will take our empty plot and add some labels.

myFirstPlot <- ggplot(carsDataset)+  
 labs(x = "Number of Cylinders", y = "Miles Per Gallon", title = "Miles per Gallon by Number of Cylinders")   
myFirstPlot

A white rectangular object with black text

Description automatically generated

Now, let’s add our first aesthetics. Aesthetics will be added to our plots by putting the aes() function inside the ggplot() function, and defining the arguments “x =” and “y =” to specify that the cyl and mpg variables go on the x and y axis of the plot. But, the plot is not done yet because we have not added a geom to explain what kind of plot to make.

myFirstPlot <- ggplot(carsDataset,aes(x=cyl,y=mpg))+  
 labs(x = "Number of Cylinders", y = "Miles Per Gallon", title = "Miles per Gallon by Number of Cylinders")   
myFirstPlot

A graph of miles per gallon

Description automatically generated

Finally, we are ready to add our first geom. In order to make a scatterplot, we will use the function geom\_point(), which tells ggplot to make a plot where each data ‘point’ gets its own point on the graph.

myFirstPlot <- ggplot(carsDataset,aes(x=cyl,y=mpg))+  
 labs(x = "Number of Cylinders", y = "Miles Per Gallon", title = "Miles per Gallon by Number of Cylinders")+  
 geom\_point()  
myFirstPlot

A graph of miles per gallon by number of cylinders

Description automatically generated

At this stage of your training, you are probably familiar with seeing lines of best fit on most scatterplots. In ggplot, we can add another geom to our plot in order to fit the line of best fit. In this case, we will need to use the function geom\_smooth() and give it the arguments *method = “lm”*, and *se = FALSE*. geom\_smooth() fits a line across our plot, which in this case will be fit using the linear modelling (“lm”) method, and then surrounds that line with a smoothed, shaded region that indicates the standard error around the line. We do not need this information just yet, so we will make the shaded region invisible with the argument “se = FALSE”.[[4]](#footnote-4)

myFirstPlot <- ggplot(carsDataset,aes(x=cyl,y=mpg))+  
 labs(x = "Number of Cylinders", y = "Miles Per Gallon", title = "Miles per Gallon by Number of Cylinders")+  
 geom\_point()+  
 geom\_smooth(method="lm", se=FALSE)  
myFirstPlot

## `geom\_smooth()` using formula = 'y ~ x'

A graph of a number of cylinders

Description automatically generated

And that is the general grammar of ggplot2. Let’s apply these principles to three other kinds of plots. challenge yourself to go through the code line by line and explain what every object, function, and argument is doing.

First, let’s make a histogram of the weight (variable name “wt”) of different cars measured in tons. Notice how we can supply the “binwidth” argument to geom\_histogram() to stipulate how many bins should be put on the plot. How many cars have a weight of 5 tons or more?

histogramContinuous <- ggplot(carsDataset,aes(x=wt))+  
 labs(x = "Weight in Tons", y = "Frequency", title = "Weight of Cars")+  
 geom\_histogram(binwidth = 0.5)  
histogramContinuous

A graph of weight of cars

Description automatically generated

Next, a different histogram using the function geom\_histogram(). Let’s examine how often we find diamonds of different clarity ratings. Clarity is a categorical variable, but the car weight variable in our last example was continuous and numerical. Notice how we have to give geom\_histogram() the argument stat=“count” to tell it to count up all the different clarity ratings in the clarity variable. Looking at the plot below, approximately how many diamonds had a rating of VVS2?[[5]](#footnote-5)

histogramDiscrete <- ggplot(diamondsDataset, aes(x=clarity))+  
 labs(x = "Clarity", y = "Frequency", title = "Frequency of Clear Diamonds")+  
 geom\_histogram(stat="count")

## Warning in geom\_histogram(stat = "count"): Ignoring unknown parameters:  
## `binwidth`, `bins`, and `pad`

histogramDiscrete

A graph of a number of diamonds

Description automatically generated

Third, let’s make a simple boxplot. You likely remember from your previous statistics course that a boxplot shows the median of a continuous variable, as well as its 25th and 75th percentiles. It also usually has bars that represent 1.5\* the distance between the 25th and 75th percentiles (also called interquartile range, IQR). Values beyond these bars are often considered to be outliers, and usually appear as individual points on the boxplot.

This boxplot will show the cost of different diamonds, with different boxplots for the different cut ratings that these stones have. Which cut rating had the lowest median price?

boxplotExample <- ggplot(diamondsDataset, aes(x=cut,y=price))+  
 labs(x = "Cut Rating", y = "Price in Dollars", title = "Cut vs. Price for Diamonds")+  
 geom\_boxplot()  
boxplotExample

A graph of cut vs. price for diamonds

Description automatically generated

Finally, a bar plot that shows the average price in US dollars of diamonds with different color ratings. Which color rating appears to be the most valuable?  
*Note: ggplot handles bar plots in a way that I think is particularly sticky. By default, it wants to make them behave as histograms, and the only way to make the y axis show specific values is to override a default argument in geom\_bar(), and supply two different arguments that stipulate the y variable should be averaged (“fun=mean”) and used to make the binned summary of the x variable (“stat=”summary\_bin”“). I apologize for this frustrating bit of arcane code.*

barPlot <- ggplot(diamondsDataset, aes(x=color,y=price))+  
 labs(x="Color",y="Cost",title="Cost of Diamonds by Color Rating")+  
 geom\_bar(stat="summary\_bin",fun=mean)  
barPlot

A graph of a graph of diamonds

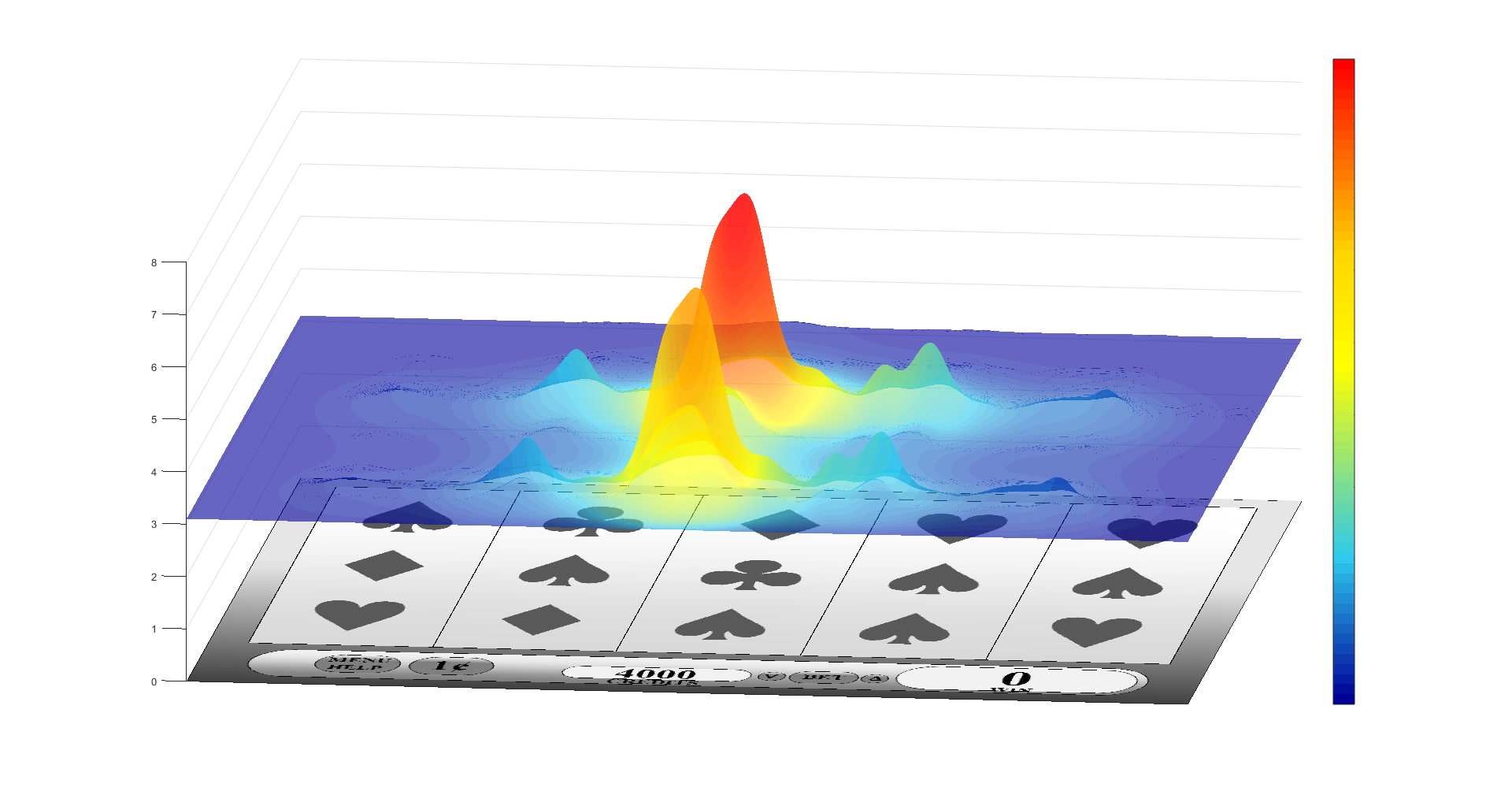
Description automatically generated with medium confidence

## Accessible graphics for diverse audiences

Looking at each of the plots we fit during the previous topic, you may have found yourself wondering why every plot looks so… *boring*! In no small way, this is due to the fact that I intentionally omitted *colour* from every single plot.

To say that colourful visualization is a fraught topic in the world of academic publishing is a serious understatement. Many papers you will encounter will intentionally choose to avoid colour for two specific reasons:

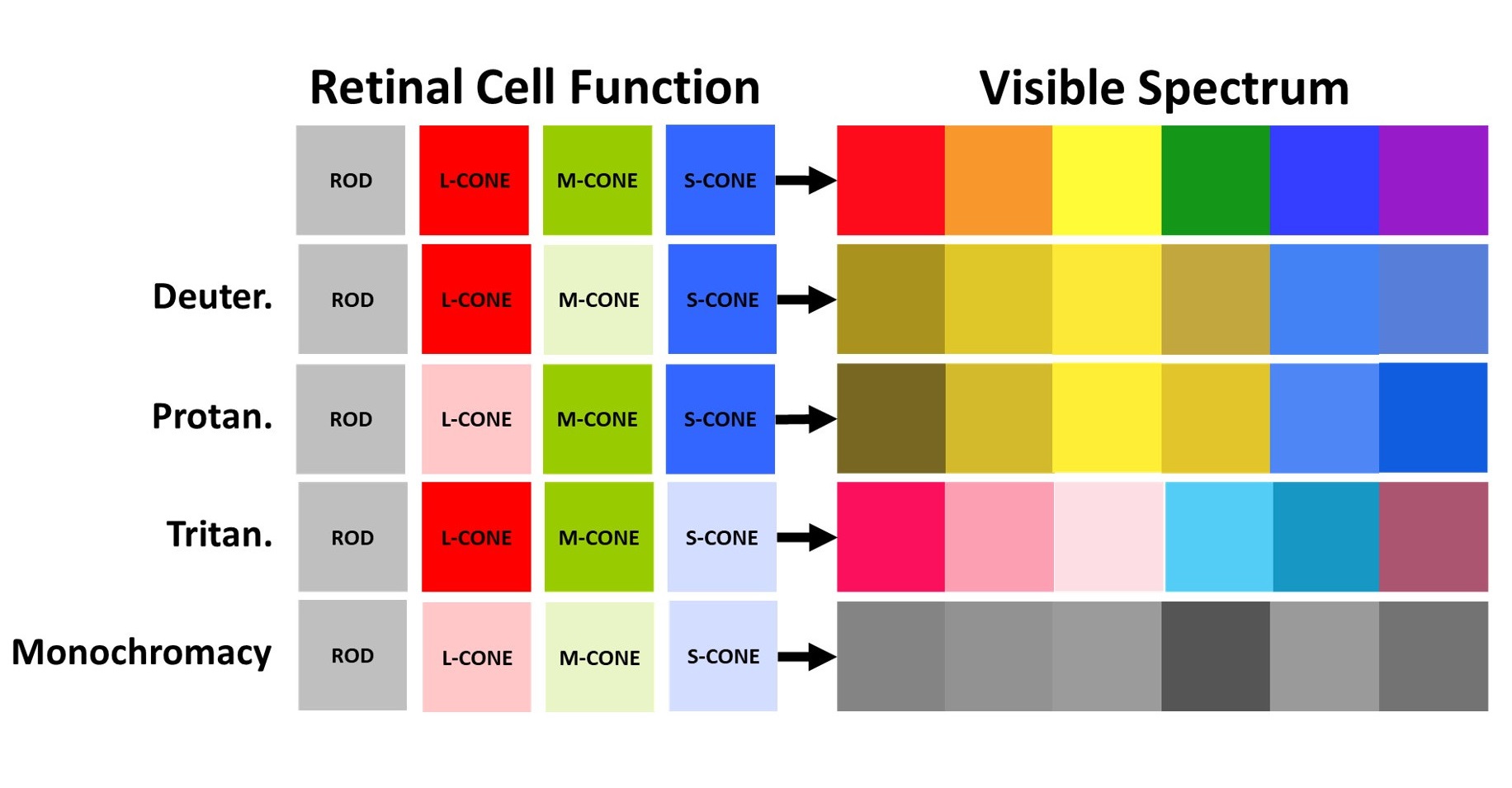
1. Back in the *bad-old-days* before every research paper was easily available online, researchers often had to read physical copies of academic journals that were printed in black-and-white. If you wanted your paper to have colour graphics, you often had to pay for a colour page to be inserted into every copy of that issue of the journal. The costs (as you might imagine) add up quickly. In order to save yourself from getting a pricey bill from the journal you are publishing in, it is often best to just make all the graphics black-and-white. At minimum, this allows you to clearly preview what the image will look like when it is printed (colours with similar lightness can become indistinguishable when a colourful image is converted to *greyscale*).
2. Colour can be distracting if it is not tastefully used. The statistician Edward Tufte famously coined the term ‘chartjunk’, which refers to any aspect of a graph or plot that is not 100% necessary to convey its meaning. His point was that visualizations *deserve* to be as simple as possible. I will put one example of a plot with lots of chartjunk below.



**Figure 3.2. The author of this plot (and this manual) thought he could include some chartjunk without making the plot impossible to interpret. He was wrong.**

But, what if you need to use colour? Heck, what if you just really, really *want* to use colour? Surely there’s a way to do it safely!

There is. But we will need to consider one more important fact: people do not all perceive colours the same way. In fact, the diversity of colour vision is substantial. In the chart below, I show how several colour vision diversities impact the colours that different people perceive, and what specific conditions (protanomaly, protanopia, deuteranomaly, etc.) may be responsible.



**Figure 3.3. Visual diversity is considerable in humans, and affects how we can communicate scientific findings using visualizations.**

Luckily, there is an R package, *RColorBrewer*, that can help us to find colour palettes that can be differentiated by people experiencing a wide range of colour vision conditions. Use the install.packages() function to install “RColorBrewer”, and then load it using the code chunk below.

library(RColorBrewer)

Now, let’s take a look at the colour palettes available through RColorBrewer. Pay special attention to “Set2” and “Dark2” in the graphic below; they often work quite well for making colour visualizations that can be understood by everyone.

display.brewer.all(colorblindFriendly = TRUE)

A chart of different colors

Description automatically generated

Now, how will we go about actually using rColorBrewer? Colour, whether in the form of a ‘fill’ or a ‘colour’ aesthetic (this varies depending on what geom you are using), is just one more thing that we have to add when we create a plot. Let’s make a minor modification to the code we assigned to the plot we made earlier. Specifically, we will add:

* A new aesthetic, **fill=** to define the variable that will be used to divide up the colours. In this case, every different level of the variable “color” from the diamonds dataset gets its own colour.
* a new function called scale\_fill\_brewer(), with the argument **palette=“Dark2”** to define exactly what colours will be given to the different levels of the variable we gave to the fill= aesthetic.

This is a lot to take in, so try playing around with this code chunk and see what happens.

barPlotColour <- ggplot(diamondsDataset, aes(x=color,y=price, fill=color))+  
 labs(x="Color",y="Cost",  
 title="Cost of Diamonds by Color Rating")+  
 geom\_bar(stat="summary\_bin",fun=mean)+  
 scale\_fill\_brewer(palette="Dark2")  
  
barPlotColour

A graph of different colored bars

Description automatically generated

Things can be a little different when we use other geoms. Notice where I used the **color=** aesthetic and scale\_color\_brewer() function instead of the **fill** aesthetic and scale\_fill\_brewer() function. What happens if you modify this code chunk to use fill instead?

boxplotExampleColour <- ggplot(diamondsDataset, aes(x=cut,y=price,color=cut))+  
 labs(x = "Cut Rating", y = "Price in Dollars",   
 title = "Cut vs. Price for Diamonds")+  
 geom\_boxplot()+  
 scale\_color\_brewer(palette="Dark2")  
  
boxplotExampleColour

A graph of cut vs. price for diamonds

Description automatically generated

**Activity**

Take some time to play around with the datasets and packages we used to make visualizations. Using different geoms, aesthetics, or functions, can you make a version of any of the plots that you like better?  
Next, visit the ggplot website (<https://ggplot2.tidyverse.org/>), and try to find a new kind of geom that we did not use in this lesson. Then, using the diamonds dataset, make a brand new visualization and add it to this manual. Make sure to give your visualization appropriate labels, and a bit of colour where it is appropriate to do so! Finally, add a comment explaining whether you think your new plot has any elements that could be considered ‘chartjunk’.

# My modification of one of the earlier plots.  
  
# Call the name of my modified plot to display it.  
  
# My brand new plot using a geom I discovered.  
  
# Call the name of my new plot to display it.  
  
# I believe this plot does / does not contain chartjunk because...  
#  
#  
#

We have reached the end of another challenging lesson in R statistics! Before you go, make sure to write down your three major takeaways and review their key concepts when we come back next week.

**Key Takeaways**  
The most important lessons from this Lesson were:  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)

Lesson 4 - Frequency tables, chi squared test, and the odds ratio

Last edited on: 2023-12-07

# Lesson 4 - Frequency tables, chi squared test, and the odds ratio

Welcome! In the last Lesson, we took a rather deep dive into the topics of data cleaning and visualization. You may recall from Figure 3.1 that these two sections of the data analysis pipeline can be found on either side of the activities we traditionally think of as constituting ‘statistical analysis’: the computation of descriptive and inferential statistics.

So… what gives? Why did the author of this book make you read a fancy pipeline analogy and then immediately do things out of order?

As you will see, beginning with this lesson and carrying on until the end of the manual, we will now take a topic-based approach to specific statistical analyses. With each new lesson, we will import and clean a *real dataset*. We will then go through the analysis pipeline in earnest: starting at the start, and ending at the end. Since these lessons are, after all, meant to teach you primarily about all the statistics that will appear in the middle, I thought it would be best if we sorted out the cleaning and visualization topics first so that we could devote more energy to classical statistics later on.

In this lesson, we deal with descriptive and inferential statistics for the first and simplest data structure that you will encounter as a psychologist. This is the situation where both the *independent* (also called ‘*input*’ or sometimes ‘*predictor*’) variable, and the *dependent* (also called ‘*output*’ or sometimes ‘*criterion*’) variable have a *categorical* (also called ‘*nominal*’) structure. When this happens, the most useful way to interpret our results usually involves:

1. describing the relationships between specific categories using *odds ratios*,
2. assessing statistical significance using a *chi square* statistic, and
3. displaying our data on a frequency table.

Bringing these three procedures together, people who read about your study will be able to clearly understand the practical significance (odds ratio), statistical significance (chi square and its p-value), and conceptual relatedness (frequency table) of your two categorical variables.

### Our objectives this lesson:

* Create frequency tables in R
* Briefly review the chi squared test
* Compute chi squared for two independent samples
* Compute chi squared for single samples
* Learn the assumptions of chi squared-type tests
* Compute an odds ratio as a measure of effect size
* Report chi squared tests with APA format

## Frequency tables in R

You probably recall from your introductory statistics course that frequency tables are typically composed of two related variables:

1. A categorical variable, and
2. A numeric variable that counts the number of cases in each category.

For example, we might create a simple table in R to count and compare how many car dealerships there are in Kelowna, Canada. In the code chunk below, I demonstrate how to manually create a data frame that looks like a frequency table. As you explore this code, notice how I introduce the function cbind(), which works to put objects with equal numbers of rows side by side in a matrix. I then use the function as.data.frame() to convert that matrix into a data frame. I then use colnames() to define new column names for my table (notice how the function is applied to the left side of “<-” because it is modifying a specific aspect of an existing object, the column names).

# First, clear out any stuff in the Environment leftover from the last lesson.   
rm(list=ls())  
  
# Manually enter the data  
carDealerships <- c("Ford","Honda","Chevrolet","Kia","Hyundai")  
dealershipFrequencies <- c(2,2,2,1,1)  
  
# Create a table of the data by binding the columns using cbind(). Notice how the class is "matrix" and not data frame.  
carDealershipTable <- cbind(carDealerships,dealershipFrequencies)  
  
# Convert the table to a data frame using as.data.frame()  
carDealershipTable <- as.data.frame(carDealershipTable)  
  
# Give the data frame some clearer column names.  
colnames(carDealershipTable) <- c("Manufacturer","Frequency")  
  
# Call the table's name so it prints to the console  
carDealershipTable

## Manufacturer Frequency  
## 1 Ford 2  
## 2 Honda 2  
## 3 Chevrolet 2  
## 4 Kia 1  
## 5 Hyundai 1

Readers looking at this object in RStudio may have noticed that the “Frequency” variable has the class <chr>, meaning “character”, even though we entered numbers in this column. This is a problem because R does not know to how to do calculations with numbers that it thinks are characters. It is an unfortunate side effect of our use of the cbind() function, but one that we can remedy quite easily by redefining this object using the as.numeric() function. This will convert our variable from character class to numeric class.

# Convert the Frequency column to numeric.  
carDealershipTable$Frequency <- as.numeric(carDealershipTable$Frequency)  
  
# Check that this procedure worked by calculating the total number of car dealerships  
totalDealerships <- sum(carDealershipTable$Frequency)  
  
# Call the object's name to print the number of dealerships to the console.  
totalDealerships

## [1] 8

This is a very common situation you will encounter many times during your statistical training. But, even more common is the situation where you already have a data frame that you need to boil down to a frequency table. Let’s take a look at how to do that using one of the most famous datasets in the world of data science, the Palmer Penguins!

These data were collected from 2007-2009 by Dr. Kristen Gorman from the University of Alaska, Fairbanks. The ‘penguins’ in this data were observed on three islands in Antarctica (Biscoe, Torgersen, and Dream islands), near the Palmer research outpost. Each one was classified as belonging to one of three distinct subspecies (Adelie, Gentoo, or Chinstrap penguins). They were then measured on several important characteristics (bill length [mm], bill depth [mm], flipper length [mm], body mass [g], and biological sex), so that the different species could be compared.

The Palmer Penguins dataset can be easily loaded into r with the package and function of the same name, but we will simply load it from our course files here. Each month, the palmerpenguins package in R is downloaded more than 40,000 times. Thousands of careers in data science have been launched by this dataset, and now it’s your turn. Let’s get started!

# Load the data.  
penguinsData <- read.csv("PalmerPenguins.csv")

One of the first questions we might ask is: how many penguins were recorded on each island? And how many of each type of penguin were recorded? Let’s use the table() function to quickly answer both questions!

First the islands:

# Create the table object  
islandTable <- table(penguinsData$island)  
  
# Call its name to print it to the console  
islandTable

##   
## Biscoe Dream Torgersen   
## 168 124 52

Now the penguin species:

# Create the table object  
speciesTable <- table(penguinsData$species)  
  
# Call its name to print it to the console  
speciesTable

##   
## Adelie Chinstrap Gentoo   
## 152 68 124

Notice with each of the objects that the class is listed as ‘table’. This is different from character, numeric, and data frame! You can do mathematical operations of these tables without naming a variable inside the table. For example, we can count the total number of penguins with the code:

sum(speciesTable)

## [1] 344

You might now find yourself wondering, how many of each penguin are found on each island? The table function is quite powerful for creating this kind of **cross-tabulation**. When we crosstabulate, we are taking two or more variables, and comparing their frequencies in two or more dimensions. If you crosstabulate two variables, you will get a two dimensional table with rows and columns. If you crosstabulate three variables, you will get several two dimensional tables (as many as the number of categories in your third variable). And so on.

To crosstabulate with table(), simply give the names of the variable to be cross-tabulated as arguments inside the table() function. But be careful, you can get some pretty wild tables if you give more than two variables’ names, or if your variables have a large number of distinct categories.

# Create the table  
penguinCrossTab <- table(penguinsData$island,penguinsData$species)  
  
# Call its name to print to console  
penguinCrossTab

##   
## Adelie Chinstrap Gentoo  
## Biscoe 44 0 124  
## Dream 56 68 0  
## Torgersen 52 0 0

How interesting! Some penguin species were found on only some of the islands (*We will return to this topic of empty table cells later on in our lesson on assumptions of the chi squared test*)! Like plots and graphs, data tables can be one simple way to boil down hundreds of data points into a small number of clearly visible trends.

**Reflect**  
Compare the number of rows and columns for the tables we created as data frame- and table-type objects. What is one thing you notice?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Add a third variable, “longbill”, to the code that created the table object “penguinCrossTab”. What do you notice? What would happen if we added a fourth variable?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Create a table for the variable “bill\_length\_mm”. What problem do you notice with this table, and why might this have occurred?  
(“\_\_\_\_\_\_\_\_\_\_”)

## Mini review - What is a chi squared test?

At its most basic level, the chi squared test asks:

“do the frequencies of the categories of this variable significantly differ from what we expected to find?”

To test this question, the chi squared formula compares two important quantities:

1. the frequencies actually found in each category, and
2. the frequencies that were expected in those categories, either because we have another sample with its own frequency distribution, or because we know what the frequencies theoretically should have been.

Taking the differences between these quantities, the chi squared formula calculates the total discrepancy between what we observed, and what would be expected by random chance alone. The chi squared statistic is then compared against the theoretical distribution of chi squared values with some number of degrees of freedom in order to calculate the relevant *p*-value. Here is the general formula to obtain chi squared:

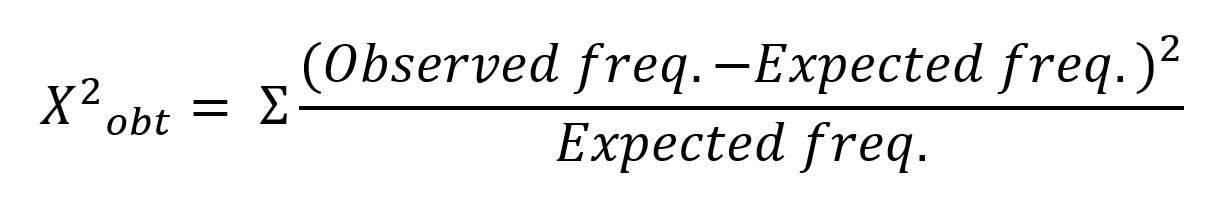


Figure 4.1. The chi squared formula.

There are several different cases for using chi squared that we will cover in this lesson. Chi squared tests can:

1. compare frequency distributions from two independent samples,
2. compare a single frequency distribution to a set of known probabilities,
3. compare specific portions of frequency distributions for paired samples (*McNemar’s test*, a procedure that is often relevant to clinical psychology, health psychology, and disciplines where treatments involving matched patient populations are compared)

## Chi squared test of independence

For this form of chi squared test, we are essentially asking:

“Do the frequencies observed for the categories of one variable differ based on the categories of another variable?”

Or, for a more concrete example using the Palmer Penguins dataset:

“Does the number of penguins of each biological sex differ based on the island where the penguins were recorded?”

In order to test this question, we may visualize the cells of this crosstabulated table like this…

table(penguinsData$island,penguinsData$sex)

##   
## female male  
## Biscoe 80 83  
## Dream 61 62  
## Torgersen 24 23

Mercifully, the code for our chi squared test is almost identical. Just substitute the table() function with the chisq.test() function!

chisq.test(penguinsData$island,penguinsData$sex)

##   
## Pearson's Chi-squared test  
##   
## data: penguinsData$island and penguinsData$sex  
## X-squared = 0.057599, df = 2, p-value = 0.9716

Notice in the test output that we are given three values: a chi squared value, a degrees of freedom value, and a *p*-value. If we wanted to report this result using APA format, we might say something like:

“The number of penguins categorized as male or female did not significantly differ across the three islands where data were collected (*X*2(2) = 0.06, *p* = .97).”

Notice how we had two degrees of freedom for this analysis. For chi squared, df is equal to (No. of rows - 1) \* (No. of columns - 1). In this case, since we had three islands and two penguin sexes, df = (3-1) \* (2-1) = 2. Notice also how we used a *leading zero* when we reported the chi square statistic, but not the *p*-value. This is because chi square can be greater than 1, but a *p*-value never can.

## Chi squared goodness of fit test

Perhaps we instead wanted to simply know if the number of male penguins in our sample was significantly greater than we would expect to find by random chance alone. This would suggest that the experimenters somehow inadvertently oversampled male penguins.

In this case, the chi squared test we will run compares our categories against *known probabilities* (we know that penguins are roughly 49% male and 51% female in the wild). To do this, we will give two arguments to the chisq.test() function:

* a table made of the variable to be tested (penguinsData$sex), and
* “p =”, an object that combines the probabilities for each category.[[6]](#footnote-6)

# Create the table for this test  
singleTable <- table(penguinsData$sex)  
  
# Call its name to print the table to the console. Notice that the female penguins are listed first.  
singleTable

##   
## female male   
## 165 168

# Run the chi squared test with probabilities specified for female followed by male penguins.  
chisq.test(singleTable,p=c(0.51,0.49))

##   
## Chi-squared test for given probabilities  
##   
## data: singleTable  
## X-squared = 0.28034, df = 1, p-value = 0.5965

Notice that we now have df = 1, because we have two sexes, and two probabilities, so df = (2-1) \* (2-1) = 1.

To report this result using APA style, we might say:

“The sexes of sampled penguins did not significantly differ from expected proportions (*X*2(1) = 0.28, *p* = .60).”

**Explain**  
What is the c() function doing in the code chunk above?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Why does the number of male and female penguins not add up to the total sample size?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Based on your answer to the previous question, is there a problem with these data that will need to be reported to readers of a scientific paper based on this sample?  
(“\_\_\_\_\_\_\_\_\_\_”)

## Assumptions for chi square-type tests

These assumptions justify the appropriateness of every chi squared test. If any of them is untenable, then a chi squared test is not suitable for your data and hypothesis. Under some circumstances, it may be appropriate to use Fisher’s exact test instead. We do not discuss that test in this lesson, but it is explained in detail in Andy Field’s excellent resource “*Discovering Statistics Using R*”.

1. Categorical variables  
   If your variables are not categorical, you must select a different test that takes into account the relationships between numeric entries in your variables. If one variable is categorical and the other is numeric, you will do a t-test if the independent variable is the categorical and a logistic regression if the dependent variable is categorical. We will see both in later lessons.
2. Independence of observations  
   The family of chi squared tests assumes that each observation in your categories is *independent*. That is, knowing the value of any one score gives us no information about any other score.
3. Categories must be mutually exclusive  
   Observations cannot exist in multiple categories at once. Each observation must exist in only one category.
4. Expected frequencies in each category should be greater than 5 for at least 80% of categories.  
   Chi squared tests may be inaccurate if an expected cell frequency is too small. The result of this is a loss of statistical power, which seriously limits your ability to accurately detect real effects that may be present in your data.

## The odds ratio as a measure of effect size

In each lesson from now until the end of this manual, we will use **effect sizes** to understand the *practical significance* of our findings. This is conceptually different from the *statistical significance* that we assess using *p*-values. Whereas a *p*-value indicates how likely we are to have obtained our observed results (or results more extreme than what we observed) by random chance alone, effect sizes tell us something tangible about how much change on the dependent variable we can expect to see as a result of someone being exposed to a particular experimental manipulation, reporting a high score on a continuous variable that is correlated with the dependent variable, or (in this case) belonging to a particular category on our independent variable.

For chi squared analyses, the most typical way we will understand effect sizes usually involves an **odds ratio**. Let’s break down that term. The **odds** of something occurring is simply the number of times we observed that thing to occur divided by the number of times we observed that thing to *not* occur. Any **ratio** is just a situation where we will be dividing something by something else. So, an odds ratio is just one odds divided by some other odds.

Imagine two categorical variables, one with the categories “Treatment” and “Control”, and the other with the categories “Yes” and “No”. An odds ratio could tell us whether the odds of being “Yes” given that one is in the “Treatment” category are greater than, less than, or equal to the odds of being “Yes” given that one is in the “Control” category. Odds ratios corresponding to 1 indicate equal odds between the two, while odds ratios between 1 and infinity indicate greater odds, and odds ratios between 0 and 1 indicate lesser odds.

That’s still too abstract to be useful. Let’s walk through an example.

In the penguinsData object, I created a variable called “longbill” that divides penguins into two groups: those who have “Long Bill” which is greater than 48mm, and those who have a “Shorter Bill” which is not greater than 48mm. We might hypothesize that male penguins in our sample are more likely to have longer bills (this is apparently true of many species of penguin). In order to check that hypothesis, we will want to create a table, then run a statistical test to evaluate statistical significance, then calculate and interpret an odds ratio to evaluate practical significance.

Create a table of penguin sex and status on “longbill”:

# Create the table object  
longbillTable <- table(penguinsData$sex,penguinsData$longbill)  
  
# Call its name to print the table to the console  
longbillTable

##   
## Long Bill Shorter Bill  
## female 14 151  
## male 80 88

Run a chi squared test to evaluate statistical significance:

chisq.test(penguinsData$sex,penguinsData$longbill)

##   
## Pearson's Chi-squared test with Yates' continuity correction  
##   
## data: penguinsData$sex and penguinsData$longbill  
## X-squared = 61.008, df = 1, p-value = 5.683e-15

The chi squared test was statistically significant at an alpha-level equal to .05 (two-tailed).  
Now calculate an odds ratio. But what goes into the odds ratio? In this case, we want to know if the odds of having a long bill given that a penguin is male are greater than the odds of having a long bill given that a penguin is female. In terms of a formula, this will look like:

(# of long bill males) / (# of shorter bill males)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
(# of long bill females) / (# of shorter bill females)

In numeric terms, this formula is:

(80) / (88)  
\_\_\_\_\_\_\_\_\_\_\_\_\_  
(14) / (151)

This is easiest to compute in R without using any functions.

OR <- (80/88)/(14/151)  
OR

## [1] 9.805195

So, the odds ratio (often abbreviated to “OR” by psychologists) is approximately 9.81. This tells us that a male penguin in this sample is approximately ten times more likely to have a long bill than a female penguin in this sample.

We could just as easily have asked whether the odds of having a long bill given that a penguin is female is greater than the odds of having a long bill given that a penguin is male. In this case, the numerator and denominator of the odds ratio is flipped such that:

(# of long bill females) / (# of shorter bill females)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
(# of long bill males) / (# of shorter bill males)

And computationally:

ORflipped <- (14/151)/(80/88)  
ORflipped

## [1] 0.1019868

Now our odds ratio is 0.10, indicating that female penguins were only about one tenth as likely to have long bills in this sample. By converting the frequencies in our table to odds values, we can easily compare the relative likelihood of different events occurring. These kinds of probabilities are very useful for clinicians, researchers, and psychologists of every flavour to contextualize the practical significance of their findings in an intuitive way that can be understood by other people. It is often a great idea to include odds ratios when reporting the results of any analysis where the dependent variable is categorical. This includes chi squared tests, and logistic regressions (which we will visit in a later lesson).

**Activity**  
Use the “longbill” and “year” variables to Evaluate the hypothesis that penguins in 2009 were less likely to have long bills than penguins in 2007. First, filter the data to remove penguins recorded in 2008. Then, create a 2x2 table of longbill and year. Next, compute a chi squared statistic and a sensible odds ratio. Finally, report your findings using proper APA style.

# Filter the data  
  
# Make a table  
  
# Run the chi squared test  
  
# Compute an odds ratio  
  
# Don't forget to report your results using APA style!

We have reached the end of another challenging lesson in R statistics! Before you go, make sure to write down your three major takeaways and review their key concepts when we come back next week.

**Key Takeaways**  
The most important lessons from this Lesson were:  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)

Lesson 5 - t-Tests

Last edited on: 2023-12-07

# Lesson 5 - t-Tests

Welcome! In the last Lesson, we took our first steps into null hypothesis significance testing using R. We looked at the unique and very common situation where the crosstabulation of categorical variables must be tested. In this lesson, we will come back to the more-familiar situation in which one numeric variable is being tested against (1) a known population mean, (2) an independent sample of scores on the same variable, or (3) a paired sample of scores on the same variable. When only one numeric variable is concerned, we will usually be performing some kind of *t*-test.

### Our objectives this lesson:

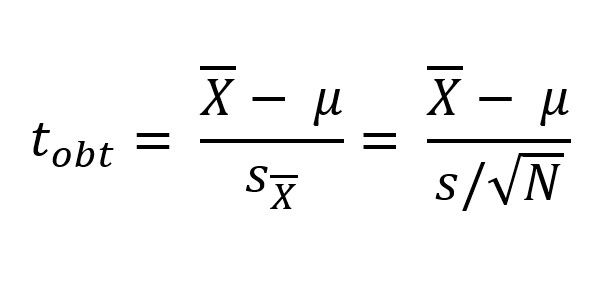
* Review the logic of t-tests
* Conduct *a priori* and *post hoc* power calculations for t-tests
* Consider different data frame structures for t-tests in R
* Compute each *t*-test
* Compute effect sizes for t-tests (Cohen’s d)
* Test assumptions: normality, independence, and homogeneity of variances
* Visualize numeric data using bar plots and line graphs

## The logic of t-tests

The conceptual core of the *t* statistic follows directly from the related concepts of standard scores (also known as Z-scores) and sampling distributions. The general idea of *Student’s t* is to compare an obtained *t* statistic (which is calculated as a kind of standard score), against its sampling distribution in order to determine how unlikely it would be to obtain your results (or results even more extreme) if the null hypothesis were true. This probability is reflected in the test’s *p*-value.

Every time you calculate a t-statistic, you will be:  
1. taking some relevant mean (usually this is the mean of your sample),  
2. subtracting an estimate of the null hypothesis population mean, and  
3. dividing the result of that subtraction by a standard error (an indicator of variability).

The generic formula for a t-test is thus:



**Figure 5.1. The general t-test formula.**

Where the obtained *t* statistic is equal to the sample mean (X-bar), minus the our estimate of the population mean (mu), divided by the standard error (computed as a standard deviation, *s*, over the square root of the sample size, N).

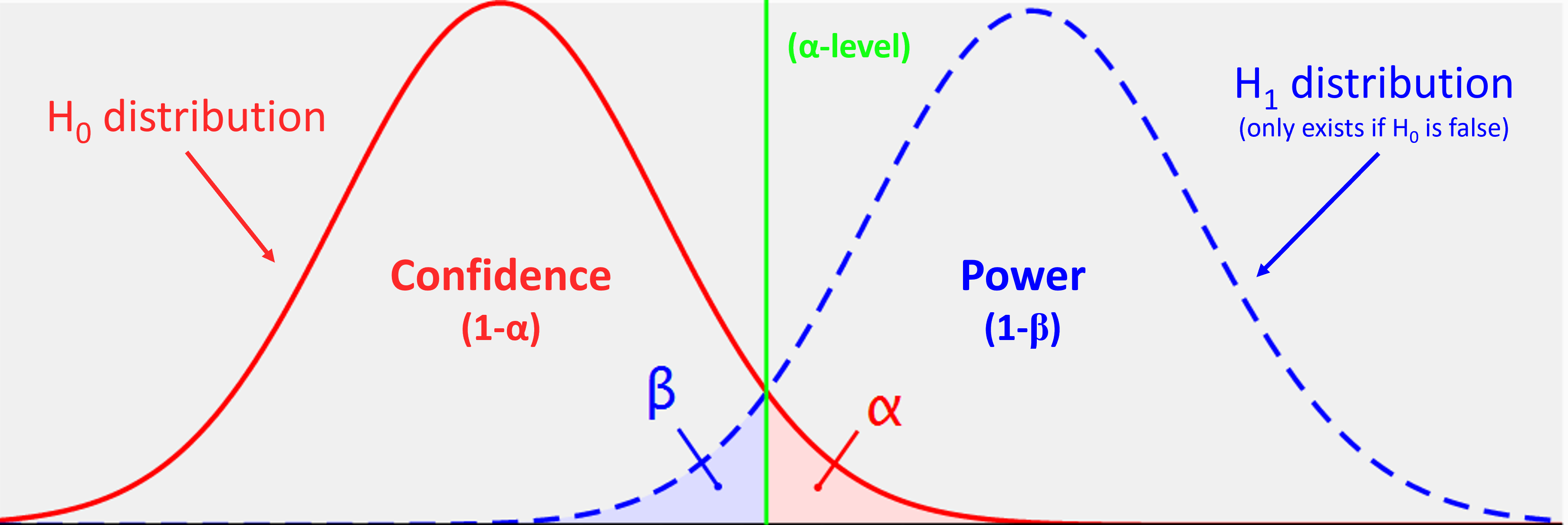
The exact permutation of this formula depends on the kind of t-test you select, but is not germane to our use of t-tests in R. If you want to dig deeper into the logic of t-tests, please check out “Understanding Statistics for the Behavioral Sciences” by Robert Pagano[[7]](#footnote-7).

Later in this lesson, we will compute the three different kinds of t-tests using real data. But first, I want to take this opportunity to check in on a concept that is central to every experiment you will conduct throughout your career in psychological science: *power*.

## *A priori* and *post hoc* power calculations for t-tests

In statistics, we spend a lot of time concerned with **confidence**, and its counterpart **alpha**. Together, they represent the probabilities of correctly retaining – or incorrectly rejecting – a true null hypothesis. We take great care to set and maintain our **alpha-levels**, so that we know the exact frequency we feel comfortable making **Type I errors** (that is, incorrectly rejecting the null hypothesis when it actually was true).

However, the closely related concepts of **power** and **beta** represent the second, often-overlooked side of that topic: the probabilities of correctly rejecting – or incorrectly retaining – a false null hypothesis.



**Figure 5.2. Power and Confidence fit together like this.**

When we consider ways to improve our experiment’s statistical power, there are several tools we could use:

1. use a larger alpha-level (but this increases Type I error rate)
2. use a one-tailed test (but the direction of effects can surprise us)
3. increase the potential mean difference by using a more intense manipulation (there are many ways this can backfire)
4. reduce sample variability by focusing the experiment on a relevant sub-population (again, many ways this can backfire)
5. increase sample size (this can be expensive)

Although there is no *perfect* way to easily increase statistical power, researchers generally agree that it is best to plan your study carefully to ensure that it will have an adequate sample size to achieve a particular power level given a particular alpha-level and effect size you are hoping to find. For example, before running their experiment, a researcher may ask:

“How many participants will I need to include in each of my independent samples so that my t-test has 80% power to detect real effects with an alpha-level equal to .05, and a Cohen’s d effect size equal to 0.50?”

When they do this, the researcher is conducting an **a priori** power calculation. This is a Latin term meaning “beforehand”. Whenever possible, you want to avoid **post hoc** (“after the fact”) power analysis because having too-little power is not something that can be remedied without repeating the entire experiment from the start[[8]](#footnote-8).

A priori power calculation using R is fairly easy via the **pwr** package. Using pwr, and its core function for t-tests **pwr.t.test()**, you can quickly find out how many participants should be sampled to achieve a particular power level. Let’s install pwr and look at a few examples.

First, run this code to install pwr:

install.packages("pwr")

Next, use the library function to load it into our R session.

# Clean out anything hanging around our environment from the last lesson  
rm(list=ls())  
  
# load the pwr package  
library(pwr)

The function pwr.t.test() has 6 key arguments that need to be specified in order to work correctly. They are:

* “n =” the number of observations per sample. Since this is the value we are trying to find, we will usually write “n = NULL”, thus emptying this argument out and signalling to the function that this is the thing to fill in.
* “d =” the effect size, Cohen’s d, that we are targeting for the power analysis. If you say you want “d = 0.50”, you are indicating that the power calculation will aim for the desired power to detect effects of a moderate size (see the topic on Cohen’s d below for a chart explaining effect size interpretation with Cohen’s d).
* “sig.level =” the alpha-level. This will usually be “sig.level = .05” because psychology studies typically select an alpha-level equal to 5%.
* “power =” the desired power level. If you say “power = .80”, you are asking the function to solve for an experiment with 80% power. This is typically understood to be the minimum acceptable power level in psychology studies.
* “type =” the test design. This can have one of three arguments, depending on if you are using a single sample (“one.sample”), paired samples (“paired”), or independent samples (“two.sample”) t-test.
* “alternative =” the hypothesis direction. Typically, you will want to specify a two-tailed (“two.sided”) hypothesis, but sometimes you may want to use a one-tailed hypothesis. In this case, you will have to indicate that you expect the mean will be either greater than the population mean estimate (“greater”), or less than the population mean estimate (“less”).

Here’s an example: imagine that you want to run an independent samples t-test. You know that you want to use a two-tailed hypothesis test with an alpha-level of .05. You know that you want to have 80% power to detect moderate effect sizes as measured by Cohen’s d (d = 0.50). How many people should be in each of the two groups?

pwr.t.test(n=NULL,  
 d=0.50,  
 sig.level=0.05,  
 power=0.80,  
 type="two.sample",  
 alternative="two.sided")

##   
## Two-sample t test power calculation   
##   
## n = 63.76561  
## d = 0.5  
## sig.level = 0.05  
## power = 0.8  
## alternative = two.sided  
##   
## NOTE: n is number in \*each\* group

**Reflect**  
If n was not equal to a perfectly round number, what should we do to determine how many people to put in each group? Can we round up, round down, or do nothing?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Based on your answer to the previous question, how many people should be in each group? How many people should be in the full experiment?  
(“\_\_\_\_\_\_\_\_\_\_”)

Let’s try another example, but this time let’s imagine that we are doing a post hoc power calculation. Suppose you are a peer-reviewer examining a study that claims to have achieved 80% power and independent samples t-test with an effect size of d = 0.70, alpha-level = .05, a one-tailed test, and n = 14.

pwr.t.test(n=14,  
 d=0.70,  
 sig.level=0.05,  
 power=NULL,  
 type="two.sample",  
 alternative="greater")

##   
## Two-sample t test power calculation   
##   
## n = 14  
## d = 0.7  
## sig.level = 0.05  
## power = 0.5630375  
## alternative = greater  
##   
## NOTE: n is number in \*each\* group

**Reflect**  
What was the actual statistical power of the study? How do you know?  
(“\_\_\_\_\_\_\_\_\_\_”)  
What should the authors do when you tell them that they did not achieve 80% power? Should they add more participants to the existing sample and resubmit their work to a different journal? Why or why not?  
(“\_\_\_\_\_\_\_\_\_\_”)

**Activity**  
For our last look at power, imagine you are the head of a psychology laboratory. Suppose that you know every participant you test will cost you 40.00 dollars, and your total budget for the experiment is 5,000.00 dollars. Ideally, you would like to detect small effect sizes (d = 0.20) with 80% power using an independent samples t-test with alpha-level = .05 (two-tailed).

Challenge yourself to design a study that your team can afford. What values will you choose to change? What effects might your choices have on the study you run?

# Fill in the code with your choices, and determine what the sample size should be  
pwr.t.test(n=NULL,  
 d=0.20,  
 sig.level=0.05,  
 power=0.80,  
 type="two.sample",  
 alternative="two.sided")

##   
## Two-sample t test power calculation   
##   
## n = 393.4057  
## d = 0.2  
## sig.level = 0.05  
## power = 0.8  
## alternative = two.sided  
##   
## NOTE: n is number in \*each\* group

In the next lesson, we will take a brief moment to consider the different ways one might choose to arrange their data frame depending on the analysis they hope to run.

## Data frame structures for *t*-tests

Before we look at the base R function that we will use to compute all forms of *t*-tests, let’s pause to talk about the data frame **structures** that different *t*-tests are expecting to see. The test we plan to run – of course – depends on the kind of data we collect. But before we run any test, the analysis pipeline (Figure 3.1) demonstrates how we need to import the collected data and prepare it to be analyzed. How will we build a usable data frame out of our data?

It turns out that the structure of our data frame will vary slightly based on whether we want to run a single sample, paired samples, or independent samples *t*-test. To make things even more complicated, the structure will also sometimes depend on whether or not we want to make a visualization of our results in ggplot.

To make sense of these differences, please consider the terms **cross-sectional data**, **panel data**, **grouping variable** and **paired variables**.

* Cross-sectional data frames are the ones you are likely most familiar with. In a cross-sectional data frame, each participant gets their own row of data, but no participant is allowed to have more than one row of data. This means that every measurement taken from that participant has to be coded in its own variable. For example, if you conducted a study of development in kindergarten-aged children, and you measured the height each participant at two different points in time, your cross-sectional data frame would have three variables with names like “ParticipantNumber”, “TimePoint1”, and “TimePoint2”. The two ‘TimePoint’ variables are **paired variables** because they form a sensible pair inside of each row. These two variables exist to separate the different conditions of the independent variable (time point) of our study, and the values recorded inside each one represent the scores on our dependent variable (height).
* Panel data (called ‘longitudinal data’ in some cases), allows each participant to have more than one row of data. When this happens, you might have a smaller number of variables, but you will always have a larger number of rows. Each participant gets the same number of rows as the number of conditions in the experiment. Using the same ‘two time points’ example, your cross-sectional data frame might have three variables with names like “ParticipantNumber”, “TimePoint1or2”, and “HeightInCM”. As you might imagine, the variable called HeightInCM records the measurements taken on the dependent variable (height), while the variable called TimePoint1or2 uses some kind of code (Like the numbers 1 and 2) to indicate whether each row of data was recorded for a given participant at the first or second time-point. This variable – TimePoint1or2 – is called a *grouping variable* because we need to use it to indicate which group (or condition, as the case may be) claims ownership of each measurement on our dependent variable.

Here are two data frames illustrating this hypothetical experiment:

# Variables for cross-sectional data  
ParticipantID <- c(1,2,3,4,5)  
TimePoint1 <- c(98,95,104,97,105)  
TimePoint2 <- c(100,99,106,98,107)  
  
# Variables for panel data  
ParticipantIDPanel <- c(1,1,2,2,3,3,4,4,5,5)  
TimePoint1or2 <- c(1,2,1,2,1,2,1,2,1,2)  
Height <- c(98,100,95,99,104,106,97,98,105,107)  
  
# Make a cross-sectional data frame  
crossSectionalData <- as.data.frame(cbind(ParticipantID, TimePoint1, TimePoint2))  
crossSectionalData

## ParticipantID TimePoint1 TimePoint2  
## 1 1 98 100  
## 2 2 95 99  
## 3 3 104 106  
## 4 4 97 98  
## 5 5 105 107

# Make a panel data frame  
panelData <- as.data.frame(cbind(ParticipantIDPanel, TimePoint1or2, Height))  
panelData

## ParticipantIDPanel TimePoint1or2 Height  
## 1 1 1 98  
## 2 1 2 100  
## 3 2 1 95  
## 4 2 2 99  
## 5 3 1 104  
## 6 3 2 106  
## 7 4 1 97  
## 8 4 2 98  
## 9 5 1 105  
## 10 5 2 107

When it comes to conducting *t*-tests and making visualizations in R, you will want to follow these conventions:

* Single sample *t*-tests want a cross-sectional data frame with one variable containing scores on the dependent variable. Visualizations of single sample tests want the same.
* Paired samples *t*-tests want a cross-sectional data frame with two paired variables that contain the paired scores of two conditions on the dependent variable. Visualizations of paired samples tests want a panel data frame with one variable containing the dependent variable scores, and a grouping variable identifying the experimental condition they belong to.
* Independent samples *t*-tests want a cross-sectional data frame with one variable containing scores on the dependent variable, and one grouping variable that identifies which group (or experimental condition) each participant was randomly assigned to. Visualizations of independent samples tests want the same.

**Reflect**  
What might go wrong if you ran a paired samples *t*-test on a panel data frame instead of a cross-sectional data frame by mistake? Would your statistical analysis be correct? *(Hint: Would R know that you are doing a paired test?)*  
(“\_\_\_\_\_\_\_\_\_\_”)  
Would there ever be a situation where you used a panel data frame for a single sample *t*-test? Why or why not?  
(“\_\_\_\_\_\_\_\_\_\_”)

In the next topic, we will dig right into computing each *t*-test. In the real world, there are some assumptions underlying each test that need to be checked before the test can be run, but for the sake of this tutorial, we will keep things simple at first and deal with assumption checking later on.

## Computing each *t*-test

For the rest of this chapter, we will consider a dataset based on a real psychological experiment. Here’s the scenario:

One factor that defines people’s social identities is perceived social group threat. Previous research has demonstrated that people strengthen ties to social groups when they perceive those groups as threatened in some way. One example is that, in the aftermath of the September 11th attacks on the United States, many US citizens felt a threat to their American social identity. This led to a dramatic outpouring of displays of patriotism, such as flying the US flag on one’s property.  
This study will investigate the effects of a perceived social group threat using a Canadian sample. Although several studies have demonstrated this effect, there is value in replicating these findings.  
Hypothesis  
It is expected that Canadians who are primed to perceive a foreign – rather than a domestic – threat to Canadian social identity, will place a different level of importance on national identity.

**Study Design**

This study will employ an experimental design adapted by Zakary Draper from experiment 3 of Davies et al. (2008). Although the data are laid out the same as a real experiment, they were created solely for demonstration purposes, and do not indicate any real scientific evidence.

In this task, participants will read a brief newspaper article describing either a foreign or domestic event that could be perceived as threatening to Canadian people. They will then complete a measure of national identity. Participants will be recruited at in-person public events in a Canadian city during the fall of 2022. Research assistants will operate tables at events throughout the summer. For completing the study, participants will be offered $5.00 food vouchers redeemable at participating vendors at the event. Recruitment will continue until sufficient data are obtained or until the summer’s end.

Individuals must be at least 18-years-old and Canadian citizens to be eligible to participate.

**Measures**  
Participants age and gender identity were recorded. The researchers noted whether each participant was in the ‘foreign’ or ‘domestic’ threat condition.

National identity will be measured with items adapted from Davies et al. (2008). They will be measured on a 7-point Likert-type scale ranging from 1 (not at all) to 7 (completely).

1. Do you identify with being Canadian?
2. Is being Canadian important to you?
3. Are you proud to be a Canadian?
4. Do you think of yourself as a Canadian?

Participant scores will be averaged across these four items to create a composite measure of national identity. This is our dependent variable.

Let’s get started by loading our data frame and assigning it to an object called ‘identityData’. Take a moment to look through the data frame that is loaded by this code chunk. Take note of each the name and class of each variable. Is this a panel dataset, or a cross-sectional dataset? Are there any pairs of variables that might go together? Are there any grouping variables?

# Load the dataset  
identityData <- read.csv("national-identity.csv")  
  
# Preview the first few rows of data.  
head(identityData)

## pid gender age condition nat\_id\_1 nat\_id\_2 nat\_id\_3 nat\_id\_4 nat\_id\_mean  
## 1 1 woman 26 domestic 5 5 5 5 5.00  
## 2 2 man 27 domestic 5 7 6 6 6.00  
## 3 3 man 28 domestic 5 4 5 4 4.50  
## 4 4 man 20 foreign 7 7 7 7 7.00  
## 5 5 woman 22 foreign 5 5 7 6 5.75  
## 6 6 <NA> 26 domestic 6 6 6 6 6.00

### Single sample *t*-test

Every *t*-test we examine in this lesson can be run using the same function called t.test(). This function belongs to the package ‘stats’, and should be loaded automatically into R. The function has several arguments that need to be handled with great care so that the correct analysis is performed.

* “x =” This argument identifies the first variable used in a *t*-test.
* “y =” This argument identifies the second variable used in a *t*-test. This will not be used in single sample tests.
* “formula =” This argument replaces “x =” and “y =” in some cases, and uses regression-style formula notation with the tilde (“~”). We will come back to this topic in much more detail in later lessons. For now, don’t worry about it.
* “mu =” This argument is used to specify the exact numeric value of the population mean (mu), in single sample *t*-tests where it is required.
* “paired =” This argument, either TRUE or FALSE, indicates whether scores on the x and y variables are paired.
* “alternative =” This argument must be one of “two.sided” (default), “greater” or “less”, depending on the one-tailed or two-tailed nature of your hypothesis. X always comes before Y. To keep these labels straight, read X and Y as “X is greater than Y” or “X is less than Y”.
* “var.equal =” This argument tells the function whether to assume that the variances of any groups are homogenous (see “Assumptions made by *t*-tests” below). If TRUE, you will run an ordinary *t*-test. If FALSE, you will run something called a “Welch-corrected *t*-test”, which we will discuss in depth later on.
* There are several additional arguments that can be useful, but we will not discuss them in this manual. You can read about them by running the code “? t.test” in your RStudio console.

With all that said, let’s first test the hypothesis that the mean national identity score in this dataset differs from the true national identity score across all Canadians. Assume that we somehow know the population mean is 5.15. Assume also that we choose to use a two-tailed hypothesis test with an alpha-level equal to .05.

# Create an object containing our test  
singleSampleTest <- t.test(x = identityData$nat\_id\_mean, mu = 5.15, paired = FALSE, alternative = "two.sided", var.equal = TRUE)  
  
# Call the object's name to print our test results to the console  
singleSampleTest

##   
## One Sample t-test  
##   
## data: identityData$nat\_id\_mean  
## t = -2.6188, df = 253, p-value = 0.009355  
## alternative hypothesis: true mean is not equal to 5.15  
## 95 percent confidence interval:  
## 4.744073 5.092542  
## sample estimates:  
## mean of x   
## 4.918307

First, take a look through the code that was written. Which arguments did we need to include, and why? Which arguments could we leave out, and why?

Next, look at the test results that were printed to the console. Compare them against the APA-formatted results statement provided below to find how each number should be reported.

“The mean national identity score in this study (4.92) significantly differed from the population mean value of 5.15 (*t*(253) = -2.62, *p* = .01, 95% CI [4.74, 5.09]).”

**Explain**  
The “mu =” argument is necessary in single sample t-tests because…  
(“\_\_\_\_\_\_\_\_\_\_”)  
The “mu =” argument will never be used in a paired or independent samples t-test because…  
(“\_\_\_\_\_\_\_\_\_\_”)  
The “y =” argument was not necessary in this example because…  
(“\_\_\_\_\_\_\_\_\_\_”)

### Paired samples *t*-test

Next, let’s imagine that we hypothesized there was a difference between two of the items on the national identity scale we created. We suspect that scores on item 3 are significantly greater than scores on item 4. We want to investigate this possibility using a paired samples t-test, and we will use a one-tailed hypothesis test with an alpha-level equal to .05.

# Create an object containing our test  
pairedSamplesTest <- t.test(x = identityData$nat\_id\_3, y = identityData$nat\_id\_4, paired = TRUE, alternative = "greater", var.equal = TRUE)  
  
# Call the object's name to print our test results to the console  
pairedSamplesTest

##   
## Paired t-test  
##   
## data: identityData$nat\_id\_3 and identityData$nat\_id\_4  
## t = -1.6531, df = 253, p-value = 0.9502  
## alternative hypothesis: true mean difference is greater than 0  
## 95 percent confidence interval:  
## -0.1495058 Inf  
## sample estimates:  
## mean difference   
## -0.07480315

Once again, take a look through the code that was written. Which arguments did we need to include, and why? Which arguments could we leave out, and why?

Next, look at the test results that were printed to the console. Uh oh! We have a problem here! The *t* statistic seems relatively far from zero (the value we would obtain if there was no difference at all between the means of our paired samples), but the p-value we obtained is more than .95! This happened because the difference between the group means turned out in the opposite direction from what we hypothesized. When this occurs, we cannot go back and re-run the test. We have to report the results of our hypothesis as we see them here. But notice, this kind of outcome could easily be printed by mistake if you are not careful to make sure your “alternative =” argument is correctly specified.

Let’s report this result using APA format.

Our hypothesis that the sample mean of scale item 3 would significantly exceed that of scale item 4 was not supported (*t*(253) = -1.65, *p*(one-tailed) = .95, 95% CI [-0.15, ∞]).

### Independent samples *t*-test

Finally, let’s test the main aim of this experiment: the hypothesis that Canadians who are primed to perceive a foreign – rather than a domestic – threat to Canadian social identity, will place a different level of importance on national identity.

To test this hypothesis, we will make use of a new argument called “formula” that uses a symbol called the tilde (“~”). This symbol will become very familiar as we begin regression modelling in our later lessons. The easiest way to think about the tilde is to imagine that it is short for the phrase “… will be predicted by …”, as in “national identity will be predicted by experimental condition”. With the tilde, you always want to put your dependent variable before the tilde, and your independent variable(s) after the tilde.

# Create an object containing our test  
independentSamplesTest <- t.test(formula = identityData$nat\_id\_mean ~ identityData$condition,  
 paired = FALSE, alternative = "two.sided", var.equal = TRUE)  
  
# Call the object's name to print our test results to the console  
independentSamplesTest

##   
## Two Sample t-test  
##   
## data: identityData$nat\_id\_mean by identityData$condition  
## t = -2.5607, df = 252, p-value = 0.01103  
## alternative hypothesis: true difference in means between group domestic and group foreign is not equal to 0  
## 95 percent confidence interval:  
## -0.7949186 -0.1037604  
## sample estimates:  
## mean in group domestic mean in group foreign   
## 4.709559 5.158898

Once again, take a look through the code that was written. Which arguments did we need to include, and why? Which arguments could we leave out, and why?

Next, look at the test results that were printed to the console. Consider each value, and try to find where they are reported in the APA-style statement below.

For APA-style reports, it is also typical to include a standard deviation wherever a mean is reported. However, the t.test() function does not give us this information by default. Here’s a bit of code using the sd() function to find the missing standard deviations.

# Standard deviation for people in the "foreign" condition  
# First, isolate the 'foreign' condition into a data frame by itself  
library(dplyr) # We will need to use the filter() function.

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

foreignData <- filter(identityData, condition == "foreign")  
paste("Foreign SD = ",sd(foreignData$nat\_id\_mean))

## [1] "Foreign SD = 1.50521360831736"

# Repeat the same process for the 'domestic' condition  
domesticData <- filter(identityData, condition == "domestic")  
paste("Domestic SD = ",sd(domesticData$nat\_id\_mean))

## [1] "Domestic SD = 1.29143208917798"

Participants who read the foreign threat prompt (mean = 5.16, sd = 1.51) reported significantly higher levels of national identity than participants who read the domestic threat (mean = 4.71, sd = 1.29) prompt, a mean difference of 0.45 scale points (*t*(252) = -2.56, *p* = .01, 95% CI [-0.79, -0.10]).

**One last thing…**  
Independent samples t-tests can be run several different ways in R. The most obvious method is to use the t.test() function as demonstrated above. However, we could also use a regression-based approach using the function **lm()**, which is one of the main functions of linear (regression) modelling in R. The formula we use will be very much the same, but we will have fewer options when it comes to the automatic integration of confidence intervals and directional hypothesis testing. I am including an example here to show you one simple but extremely powerful fact: the *t*-test is just a form of linear modelling. You will encounter this fact over and over as you continue your journey into the world of classical statistics.

# Make a regression model to run a t-test  
regressionStyletTest <- lm(identityData$nat\_id\_mean ~ identityData$condition)  
  
# Generate a summary of the regression model and print that summary to the console.  
summary(regressionStyletTest)

##   
## Call:  
## lm(formula = identityData$nat\_id\_mean ~ identityData$condition)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -3.7096 -0.9596 0.0911 1.0911 2.2904   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 4.7096 0.1196 39.377 <2e-16 \*\*\*  
## identityData$conditionforeign 0.4493 0.1755 2.561 0.011 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.395 on 252 degrees of freedom  
## Multiple R-squared: 0.02536, Adjusted R-squared: 0.02149   
## F-statistic: 6.557 on 1 and 252 DF, p-value: 0.01103

Notice how this test still has an Estimated difference between group means of 0.45 points on our identity scale. Notice how the Y-intercept of this regression model is 4.7096, the exact mean of “domestic” condition. Notice how we still have 252 degrees of freedom. Notice how the p-value associated with the “foreign” condition (p = .011) is the same as the p-value for our previous analysis using the same data. Notice how it does not tell you a confidence interval, but does give you a standard error that could be used to calculate a confidence interval. Finally, notice how it gives you a couple versions of an effect size in the form of R-squared. This could be useful if you prefer to report effect sizes using R squared and not Cohen’s d[[9]](#footnote-9).

**Reflect**  
Why did the independent samples test have 252 degrees of freedom, while the paired samples and single sample tests had 253?  
(“\_\_\_\_\_\_\_\_\_\_”)  
When might it be a good idea to use the *t*-test-style code for a *t*-test? When might it be a good idea to use the regression-style code instead?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Why can’t we just run a one-tailed test twice, and then pick the lower p-value that occurred when “alternative =”greater”” versus “alternative =”less”“?  
(”\_\_\_\_\_\_\_\_\_\_“)

We have now reached the end of another topic in this lesson! In the next topic, we will look at how to calculate an effect size, Cohen’s d, for any *t*-test.

## Effect size - Cohen’s *d*

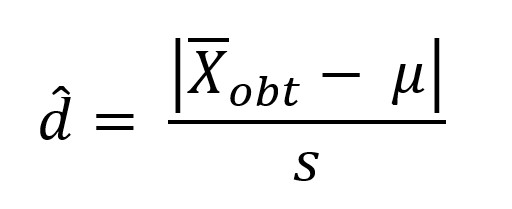
In earlier chapters, I hinted at the difference between statistical significance, and practical significance. Statistical significance is a very narrow conception of whether or not a particular result is meaningful, and it rests entirely on the logic of null hypothesis significance testing. We say that a finding is statistically significant if the likelihood of obtaining results as-extreme or more-extreme than those we actually observed (the *p*-value) is less than some critical threshold (the alpha level) we set when we first conceived of our hypothesis.

You can see that – when we say something is statistically significant – we are making a very specific statement about probability in a world where the null hypothesis is true. This has a couple of very clear problems: (1) statistical significance is closely related to sample size for many statistical procedures (in other words, findings may appear to be statistically significant when the true difference between groups is minuscule and the sample size is huge), and (2) as a result, this kind of probability statement has almost nothing useful to say about the practical impact of our findings.

For example, suppose that I conduct a large-scale randomized control trial for a drug that is intended to reduce dizziness among pregnant people (dizziness is a common side effect of pregnancy). Suppose that my trial, which had 10,000 pregnant people in each of the experimental and control groups finds that dizziness symptoms – which were rated on a 10-point scale – decreased by 0.05 points on average in the experimental group. I might find that this result is statistically significant in my study of 20,000 pregnant people, but it is clear that the drug has only a very small practical impact on dizziness ratings. In other words, my analysis was statistically significant, but not practically significant.

How can we indicate when a finding is *both* statistically significant and practically significant? We will need to calculate a new descriptive statistic for our sample data, and report it in addition to the *p*-value. We have seen one metric of this type in the previous chapter (the odds ratio, often reported as “OR =”). For *t*-tests, the most-frequently reported measure of practical significant (or “effect size”) is called **Cohen’s** ***d***.

Cohen’s *d* is computed in a mercifully simple manner: it is simply the absolute mean difference between samples (or between a sample and the population mean, in the case of single sample *t*-tests), and our estimate of the population standard deviation. This can be described for the single samples case using the formula:



**Figure 5.3. Generic formula for Cohen’s** **d.**

Where Xbar is the sample mean, mu is the population mean, and s is the sample standard deviation.

There are a couple of things to notice about this formula. First, *d* is wearing a hat “^”. This is because d is being estimated using sample data. Second, the numerator of this formula is written in **absolute** terms using vertical bars (“| |”). This means that, even when an effect goes in the negative direction, Cohen’s *d* will always be reported as a positive number. This fact is sometimes forgotten by statisticians writing R packages, and you should watch out for functions that erroneously return negative Cohen’s *d* values (the abs() function can be useful for making these negative values into positive values).

Let’s compute an effect size by hand. For the single sample case, this is often the easiest way to do it.

In our example of a single sample *t*-test, we used the variable “nat\_id\_mean” inside the data frame “identityData”. We found that the mean of this sample was significantly less than the previously-known population mean which we said was equal to 5.15.

To calculate Cohen’s *d*, let’s first recall the mean of the relevant variable.

# Create an object called 'groupMean' that will store the mean of the variable nat\_id\_mean.  
groupMean <- mean(identityData$nat\_id\_mean)  
# Call the object's name to print it to the console.   
groupMean

## [1] 4.918307

Next, compute the standard deviation that goes with that same variable, and store it as an object.

# Create an object called 'groupSD' that will store the standard deviation of the variable nat\_id\_mean.  
groupSD <- sd(identityData$nat\_id\_mean)  
# Call the object's name to print it to the console.   
groupSD

## [1] 1.410004

Now, using these new objects, and our prior knowledge that the population mean was equal to 5.15, calculate Cohen’s *d*. Notice how I will use the abs() function to make sure that my numerator is an absolute value.

# Create an object storing Cohen's d.  
cohensD <- abs(groupMean-5.15)/groupSD  
  
# Call its name to print the Cohen's d value to the console.  
cohensD

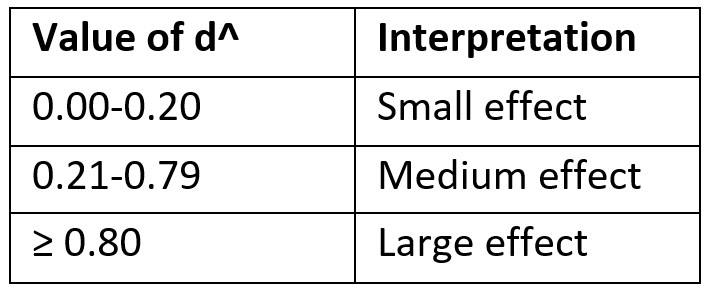
## [1] 0.1643207

*Ta da!* we have computed a Cohen’s d value and found that it was equal to 0.1643207.

Wait… what does that mean???

Interpreting Cohen’s *d* needs two pieces of knowledge. The first is the fact that Cohen’s d is computed *in units of standard deviations*. This is because the denominator of the formula is a standard deviation. So, when we think about our effect size that was approximately equal to 0.16, we can say that our sample was 0.16 standard deviations away from the population mean. This kind of interpretation is vitally important for meta analysis studies that combine and compare many different experiments and samples, but that is a topic for another manual.

The second vital piece of knowledge is that Cohen’s *d* is typically interpreted using three rules of thumb that decide what counts as a small, medium, or large effect size. Here are those rules:

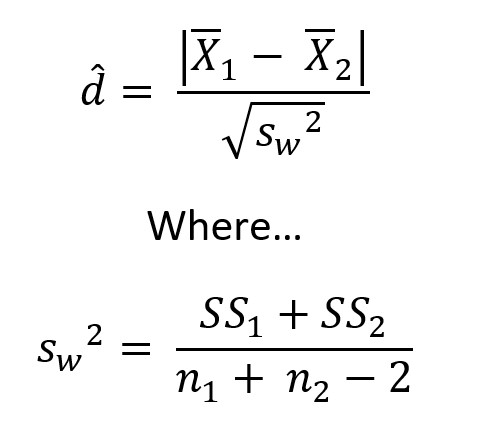


**Figure 5.4. How to interpret Cohen’s d.**

So, with this knowledge in hand, we can now say that our sample mean was 0.16 standard deviations away from the population mean, and that this would typically be interpreted as a small effect size.

### Cohen’s *d* for independent samples

Things get a little more complicated when we want to obtain Cohen’s *d* for analyses that use two independent samples. This is because: (1) we cannot just compute a new variable made of differences between paired scores and run a single sample *t*-test, and because the two different standard deviations from our independent samples need to be *pooled* in order to accurately estimate the standard deviation used in the denominator of Cohen’s formula.



**Figure 5.5. Cohen’s** **d** **for independent samples. Slightly annoying to compute by hand.**

In this case, it is much easier just to use an existing package. Run the code below to download and install the package ‘psych’, which contains various tools that are useful for psychological analyses.

install.packages("psych")

Now, load that package using the library function.

library(psych)

The main function we will use to calculate Cohen’s *d* for any *t*-test is called **t2d()**. It has four arguments that we may need to use:

* “t =” This argument specifies the *t* statistic that was found for a particular model. You can declare the exact number that you found, or you can save time by calling the part of a model where that is stored. For example, in our object called “independentSamplesTest”, which stores the *t*-test we ran earlier, the code **independentSamplesTest[[“statistic”]]** gives you the exact *t* statistic that was obtained.
* “n =” This argument specifies the sample size if you are doing a single or paired samples test. If you have no missing data, this will be equal to the number of rows in your dataset.
* “n1 =” and “n2 =” These arguments go together and are used whenever you are doing an independent samples test. You may want to use a table to find your group sizes. In the example below, I will use the table function to find how many people were in each group of my earlier independent samples test.

# First, find our how many people were in each group  
table(identityData$condition)

##   
## domestic foreign   
## 136 118

# We seem to have n1 = 136 and n2 = 118  
  
# Use the t2d() function to find Cohen's d from the t statistic and sample size.  
cohensDIndependent <- t2d(t = -2.560743, n1 = 136, n2 = 118)  
  
# Call the object's name to print Cohen's d to the console.  
cohensDIndependent

## [1] -0.3221606

Notice that the function t2d() makes the mistake I mentioned earlier: it does not output Cohen’s *d* values as absolute numbers. You will need to make sure you drop the “-” sign from any negative numbers it gives you.

**Reflect**  
Experiment 1 had a much larger variance than Experiment 2, but both experiments found d = 0.50. Was Experiment 1 more practically significant than Experiment 2? Why or why not?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Is it theoretically possible to obtain a Cohen’s *d* = 3.25? Explain your rationale.  
(“\_\_\_\_\_\_\_\_\_\_”)  
Why might it be useful for meta analysis studies to calculate *standardized mean differences* such as Cohen’s *d*?  
(“\_\_\_\_\_\_\_\_\_\_”)

You have now completed this brief introduction to Cohen’s *d* and effect size calculations. In the next topic, we will look at three assumptions underlying *t*-tests.

## Assumptions made by *t*-tests

Probably the biggest single issue with *parametric statistics* (that is to say, inferential tests that assume a population of scores with some fixed distribution) is that they can behave suboptimally or downright badly when one or more of their core assumptions are not met. In the case of *t*-tests, we will be dealing with two or three major assumptions: **independence**, **normality**, and the **homogeneity of variance**. Let’s look at them one by one.

### The assumption of independence

All *t*-tests assume that observations on the dependent variable are entirely unrelated to one another. For example, it cannot be the case that one participant appears in both groups in an independent samples test (the paired samples test gets around this possibility by turning paired observations into a single distribution of difference scores). It might also be considered a violation of the assumption of independence if something happened during data collection that calls into question the randomness of your sampling procedure (for example, if your experimental group was sampled using the snowball method wherein participants were encouraged to refer their friends and relatives to participate). In short, any event that results in some non-random relationship between two people in your study constitutes a violation of the assumption of independence. This is a major problem because the bedrock of inferential statistics relies on groups being random samples from the population prior to any experimental manipulation. If this is not true, then whatever results we obtain will be confounded by the dependencies that existed between and within our groups. This can seriously weaken, or even defeat the interpretability of any experiment.

There is no single test or visualization we can use to determine conclusively whether the assumption of independence is tenable. But, it is always a good idea to verify that each observation appears only once in your dataset, and to do everything you can to ensure that your sampling method and procedure for determining condition assignment was as random as possible.

### The assumption of normality

All *t*-tests assume that the frequency distributions of dependent variable scores for any sample(s) we collect follows the normal distribution. This is what enables us to generate meaningful *p*-values. When our data are normally distributed, most cases fall near the sample mean, and it makes sense to use that mean as a way of testing our data. If lots of cases do not fall near the mean, then the mean does not do a great job of describing our data, and it makes a lot less sense to use it as a way of conducting inferential statistics.

One fact that you might recall from earlier statistics courses: the central limit theorem holds that, as the sample size increases, the distribution of sample mean values becomes more normal. Many textbooks argue that, since this is true, the assumption of normality becomes unimportant when sample sizes are sufficiently large. This is only partly right because, although the sampling distribution of sample means approaches normality given adequate sample size, the actual distribution of scores in the population may not be normal at all. Since the whole point of a *t*-test is to draw probabilistic conclusions about a population, some researchers[[10]](#footnote-10) argue that it is a real problem if the population of scores is not normally distributed. On the other hand, considerable *monte carlo simulation* research has shown that *t*-tests are only slightly impaired by violations of the normality assumption when sample sizes are adequate[[11]](#footnote-11). So, while it is important to check this assumption each time you run a *t*-test, and to consider non-parametric alternatives (such as the Mann-Whitney *U*-test, a non-parametric alternative to the independent samples *t*-test) when it is not tenable, violations of normality are not as clearly disastrous as violations of independence.

There are several statistical tests (for example, the Shapiro–Wilks test and the Kolmogorov–Smirnov test) intended to determine if the distribution of a sample departs from normality in a statistically significant manner. These tests can be informative when sample sizes are small (i.e., below about N = 50), but they come with a number of caveats (mainly, they become less meaningful when sample sizes are very large), and should be handled in a full lesson by themselves.

For the purposes of this manual, we will adopt the much more common *visual inspection* approach to determining whether our sample data appear to be reasonably normally distributed. One easy way to do this is to examine a Quantile-Quantile (or QQ) plot using the plot() function along with our lm() regression-style *t*-test object. We will examine QQ plots, and other diagnostic plots, in more depth with our lesson on regression. For now, it is sufficient to know that normally-distributed dependent variables will have QQ plots with dots that line up along the diagonal line reaching from the bottom left to the top right corner of the plot. If the data are not normally distributed, these dots will veer away from the diagonal line at one or both ends.

# Use the plot function with our lm() t-test from earlier. Ask for QQ plot specifically using the argument "which = 2".  
plot(regressionStyletTest, which = 2)

A graph of a normal q-q

Description automatically generated

Looking at this plot, it seems that our variable of interest roughly follows the normal distribution. The key thing to notice here is that lots of cases are stacked up near the diagonal line, although there may be some concern about normality with higher-value scores bending away from the line.

### The homogeneity of variance (HOV) assumption

The independent samples *t*-test assumes that the two samples have roughly equal variances (if you prefer to think of this assumption in terms of standard deviation, that’s ok too). This mainly becomes a problem when our two samples do not have the same sample size. Since the independent samples *t*-test uses the variance of both samples in order to create a pooled estimate of the standard error, one variance can have a disproportionate impact on the test statistic denominator if sample sizes are unbalanced. This results in some amount of bias in our calculation of the *p*-value. When the larger variance is observed in the group with a larger sample size, then we find ourselves in a *conservative* condition wherein we reject the null hypothesis less often than we actually should. When the larger variance is observed in the group with a smaller sample size, we find ourselves in a *liberal* situation wherein we reject the null hypothesis more often than we actually should.

As you can see, it is important for the integrity of our hypothesis test that we assess whether HOV is tenable, and to select a more-robust statistical test if we find evidence that HOV has been violated.

Correcting for HOV violations is only necessary if group sizes differ. The *t*-test is robust against HOV violations when sample sizes are equal. We can assess the tenability of the HOV assumption using **Levene’s test**. This simple procedure will give us some indication of whether our two independent samples have significantly different variances. Since it is important to know if variances are even a little bit different, we will typically use an alpha level equal to .10 rather than .05.

We saw in our previous topic that the number of participants in each condition of the national identity experiment were n1 = 136 and n2 = 118. Since these group sizes are unequal, we should actually have run Levene’s test prior to conducting the regular *t*-test that I demonstrated in the previous section. In order to run Levene’s test, we will need the **car** package. CAR is an acronym that stands for “Companion to Applied Regression.” Run this code to install car:

install.packages(“car”)

Now let’s run a Levene’s test using the leveneTest() function. You will notice that this function needs two arguments: “y =” which represents our dependent variable scores, and “group =” which represents our grouping variable.

# Make sure to load the car package  
library(car)  
  
# Run a Levene's test  
Levene <- leveneTest(y = identityData$nat\_id\_mean, group = identityData$condition)  
  
# Call the object's name to print our results to the console  
Levene

## Levene's Test for Homogeneity of Variance (center = median)  
## Df F value Pr(>F)   
## group 1 3.8069 0.05215 .  
## 252   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Take a look at these results. The main thing to notice is that our *p*-value, .05215 is less than the alpha level we selected of .10. This means that we must reject the null hypothesis that the HOV assumption has not been violated and conclude that the HOV assumption has been violated. In order to make sure that the *p*-value we generate for a *t*-test using these variables is reliable, we must use Welch’s *t*-test, a variation on the traditional *t*-test procedure that does not rely on the HOV assumption. In exchange for not relying on the HOV assumption, Welch’s test is somewhat less powerful than the traditional *t*-test procedure. As you can imagine, it is a good idea to use the most powerful test available for any given analysis. This is why we do not simply use Welch’s test in every case.

Luckily for us, using Welch’s *t*-test in R is as simple as setting the argument “var.equal = FALSE” in our code. Nothing else needs to change.

# Create an object containing our test  
independentSamplesWelch <- t.test(formula = identityData$nat\_id\_mean ~ identityData$condition,  
 paired = FALSE, alternative = "two.sided", var.equal = FALSE)  
  
# Call the object's name to print our test results to the console  
independentSamplesWelch

##   
## Welch Two Sample t-test  
##   
## data: identityData$nat\_id\_mean by identityData$condition  
## t = -2.5332, df = 232.12, p-value = 0.01196  
## alternative hypothesis: true difference in means between group domestic and group foreign is not equal to 0  
## 95 percent confidence interval:  
## -0.79882075 -0.09985821  
## sample estimates:  
## mean in group domestic mean in group foreign   
## 4.709559 5.158898

Notice in this output how the top line now declares that we are conducting a Welch Two Sample t-Test. Notice also how the df used for this test are a *fraction* what they had been in the typical *t*-test we ran earlier. This is one of the corrections that the Welch test makes. Otherwise, this result will be reported in exactly the same manner.[[12]](#footnote-12)

**Reflect**  
Which *t*-test assumption seems to be the most-important? Why?  
(“\_\_\_\_\_\_\_\_\_\_”)  
What two scenarios could occur if I ran an uncorrected *t*-test with unequal group sizes, unequal variances, and obtained a statistically-significant *p*-value = .04? What information could clarify whether a the null hypothesis was incorrectly rejected?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Describe how random sampling relates to the assumptions discussed in this section. Is random sampling important for all *t*-tests? Why or why not?  
(“\_\_\_\_\_\_\_\_\_\_”)

In the final topic for this lesson, we will look at two ways of visualizing *t*-test effects. Let’s get to it!

## Visualizing *t*-tests

In general, *t*-tests are best visualized using bar plots or line graphs. Here are two examples based on the paired samples and independent samples cases we used above.

First, let’s make a plot of the independent samples t-test. Notice how we need to supply the arguments “stat =” and “fun =” to geom\_bar(). This is because we want to plot two summaries of our y variable using the groups of our x variable. In this case, the summary we want is the mean, so we have said we want to use the mean() function. We could just as easily have created a new object that contained group names and means we computed by hand, but doing the procedure this way allows ggplot2 to handle it for us. I will demonstrate a way to plot these means by hand in the next example.

# load the ggplot2 package  
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.2.3

##   
## Attaching package: 'ggplot2'

## The following objects are masked from 'package:psych':  
##   
## %+%, alpha

# load RColorBrewer to make things look nicer.  
library(RColorBrewer)  
  
# Fit a plot for the variable nat\_id\_mean  
indySamplePlot <- ggplot(identityData, aes(x=condition, y=nat\_id\_mean, fill = condition))+  
 geom\_bar(stat = "summary\_bin", fun = mean)+  
 labs(x = "Condition", y = "Mean National Identity")+  
 scale\_fill\_brewer(palette="Paired")  
  
# Call the plot's name to print it to RStudio's plot window.  
indySamplePlot

A graph of condition and condition

Description automatically generated

Next, let’s make a plot of paired samples. When we visualize paired cases, it is a good idea to connect paired cases using a line. This signals to viewers that these data are related in some way.

First, I will make a new data frame that has one variable identifying the two survey questions involved in our earlier paired samples analysis, and a separate variable showing the mean score on those variables. I will also calculate two variables that are equal to the mean of each group plus-or-minus one standard error. This will allow me to create upper and lower bounds for standard error bars using an additional geom.

# Make the data frame that will define the plot   
pairedPlotData <- as.data.frame(c("nat\_id\_3","nat\_id\_4"))  
colnames(pairedPlotData) <- "QuestionNumber"  
pairedPlotData$meanIdentityScore <- c(mean(identityData$nat\_id\_3), mean(identityData$nat\_id\_4))  
  
# Calculate standard errors and add them to the plot data  
SEquestion3 <- sd(identityData$nat\_id\_3)/sqrt(nrow(identityData)) # Calculate a standard error for question 3.  
SEquestion4 <- sd(identityData$nat\_id\_4)/sqrt(nrow(identityData)) # Calculate a standard error for question 4.  
pairedPlotData$errorBarUpper <- c(mean(identityData$nat\_id\_3)+SEquestion3, mean(identityData$nat\_id\_4)+SEquestion4)  
pairedPlotData$errorBarLower <- c(mean(identityData$nat\_id\_3)-SEquestion3, mean(identityData$nat\_id\_4)-SEquestion4)  
  
# Show the data frame we just made  
head(pairedPlotData)

## QuestionNumber meanIdentityScore errorBarUpper errorBarLower  
## 1 nat\_id\_3 4.881890 4.974777 4.789002  
## 2 nat\_id\_4 4.956693 5.049920 4.863465

# Make a ggplot using the geom\_point() geom to make individual points,   
# and the geom\_line() geom to draw a line that connects them.  
# Additionally, add a "size = " argument to each geom and give it a number that makes the point and line a bit larger.  
# Then, add error bars defined in the data with the geom\_errorbar() geom.  
# This new geom has three arguments that identify the minimum and maximum error bar values,   
# as well as the visual width of the bars themselves.  
pairedPlot <- ggplot(pairedPlotData, aes(x = QuestionNumber, y = meanIdentityScore, group=1))+  
 geom\_point(size = 4)+  
 geom\_line(linewidth = 2)+  
 geom\_errorbar(aes(ymin=errorBarLower, ymax=errorBarUpper), width = 0.2)  
  
# Call the plot's name to print it to the RStudio Plot window  
pairedPlot

A graph with a line and a question number

Description automatically generated

**Activity**  
Challenge yourself to improve the plots I created in the last example. Find a way to create informative titles. Try adjusting the names and labels displayed on the X axes. Find a way to capitalize group names that appear in the first plot’s legend. Google the **ylim()** function, and find a way to adjust the displayed Y axis limits so that they match the range of possible responses. Keep making changes as you see fit, and don’t stop until your plots look ‘publication-worthy’.

We have reached the end of another challenging lesson in R statistics! Before you go, make sure to write down your three major takeaways and review their key concepts when we come back next week.

**Key Takeaways**  
The most important lessons from this Lesson were:  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)

Lesson 6 - Correlation and Regression

Last edited on: 2023-12-07

# Lesson 6 - Select topics in correlation and regression

Welcome! In the last Lesson, we looked at hypothesis testing, power analysis, effect sizes, and visualization using the very common *t*-test as an example case. We looked at three cases of *t*-tests, and I even showed you how the *t*-test can be treated as a form of regression.

In this lesson, we will continue many of those same themes, but we will spend most of our time looking at regression itself.

You likely remember from your introductory statistics class that linear regressions generally follow some version of the formula:

**Y’ = BX + A**

where *Y’* is a prediction being made about the outcome variable *Y* using one or more predictor variable(s) *X*, which is/are multiplied by some slope coefficient *B*, and finally added to the point where the regression line intercepts the Y-axis, *A*.

Another valid way to think about regression is to know that we are seeking to find a *line of best fit* that minimizes the squared differences between itself and any observations on Y for all values of X. This line has several important properties that we will discuss in some detail when we come to the topic of model assumptions, but for now the most important thing to remember from introductory statistics is that the regression line minimizes the squared errors between itself and the actual values recorded for the outcome variable Y.

This lesson will also take a look at **correlation**. You will recall from introductory statistics that the Pearson correlation coefficient is bounded between -1 and +1 with scores near zero indicating a lack of either positive or negative correlation between our variables X and Y. You likely also recall seeing correlations visualized using a scatterplot with one line of best fit running through a cloud of data. This is your first hint that correlations, like *t*-tests, can be thought of as just another example of linear modelling. *It’s all just regression!*

The goal of this lesson is not to teach you everything about regression. It is a *massive* field that occupies a very large place in the world of classical statistics. Instead, I want to present you with some useful tools for thinking about, and getting started with, computing and interpreting regression analyses using psychological data. If you love this chapter, that’s great news because there is an ever-expanding universe of material still to learn about regression.

### But for now, let’s take a quick look at these objectives:

* Connect the pearson correlation coefficient to linear regression
* Compute single correlation coefficients and correlation matrices
* Consider corrections for multiple comparisons
* Compute and interpret bivariate linear regression
* Understand assumptions and diagnostic plots for linear regression
* Compute effect size for two continuous variables: R2 and R2adjusted.
* Visualize linear regression using scatterplots and correlation matrices
* Understand when to use logistic regression
* Compute and interpret odds ratios for logistic regression

## Correlation is like standardized regression

In our lesson on *t*-tests, I referred to **standard scores**, and said that you have likely seen these described called *z-scores*. A value on any of our variables can be said to have been “standardized” if we:

1. subtract from it the mean of the whole variable, and
2. divide the result of that subtraction by the standard deviation of the variable.

When we do this for all the scores in a variable, we are transforming the distribution of our scores so that this new, standardized variable will have a mean equal to zero and a standard deviation equal to 1. This is especially useful for comparing two different samples of scores on a particular variable. But, it is also very useful for thinking about how the Pearson correlation coefficient relates to linear regression.

Consider the following scenario From Zakary Draper’s “Statistics Labs for Psychology” (<https://rlabmanual.com/cor-study>):  
>**Psychopathy and Pain Tolerance** Psychopathy is a trait marked by reduced empathy, emotional response, and behavioural control. Past research has demonstrated a positive association between psychopathy and higher pain tolerance. Some researchers believe that a higher tolerance for physical pain may be linked to a lack of empathy and willingness to harm others, which are characteristic traits of people with high levels of psychopathy. At the same time, most research on psychopathy has used samples of incarcerated people, and there has been relatively little attention paid to psychopathy within the general (i.e., non-incarcerated) population. The goal of this study is to investigate the association between psychopathy and pain tolerance in a non-incarcerated sample.

Hypothesis  
Psychopathy will be associated with higher pain tolerance.

Procedure  
Participants will be 110 undergraduate students recruited through SONA. This study will be completed in-person under the supervision of a trained research assistant. After providing informed consent, participants will respond to a questionnaire which includes questions about demographic characteristics and a measure of psychopathy. They will then be fitted with a heart rate monitor and their baseline heart rate will be established. Participants will then complete a cold pressor test. They will be compensated for their time by receiving course credit in a participating course.

Measures  
Psychopathy will be measured using the Levenson Self-Report Psychopathy Scale (LSRP). The LSRP was designed to measure psychopathy in non-institutionalized settings (Levenson & Fitzpatrick, 1995). It is free to use and simple to administer. Participants report their agreement with 26 statements, using the response options: 1 (disagree strongly), 2 (disagree somewhat), 3 (agree somewhat), and 4 (agree strongly). Some items are reverse-coded, and then all items are summed to produce a composite measure of psychopathy.  
Pain tolerance will be measured using a cold pressor test. The cold pressor test is a standardized and widely used measure of pain tolerance. Participants are asked to submerse one hand in a cold water container (3–4 °C) for as long as they can. They are instructed to inform the research assistant as soon as they begin to feel pain and to remove their hands from the cold water when the pain becomes intolerable. Each participant’s pain tolerance is the amount of time between reporting first feeling pain and removing their hand from the water.

Analytic Strategy  
We will conduct a directional null hypothesis significance test of the simple bivariate correlation between LSRP total scores and pain tolerance at the traditional α = .05.

Sample Size and Power Analysis  
We plan to recruit 110 participants to complete this study. This is based on a power analysis that identified N = 101 is required to for 95% power if the population effect size is r2 ≥ .10, indicating that at least 10% of the variance in pain tolerance is explainable by psychopathy scores. Just to be safe, we recruit 110 participants to allow for the possibility that some participants may need to be excluded from analysis (e.g., if data are missing).

References  
Levenson, M., & Fitzpatrick, C. (1995). Assessing psychopathic attributes in a noninstitutionalized population. Journal of Personality and Social Psychology, 68, 151–158.

Let’s clean up our environment and load this dataset.

# First, clear out any stuff in the Environment leftover from the last lesson.   
rm(list=ls())  
  
# Load the data  
psychopathyData <- read.csv("psychopathy-pain-tolerance.csv")

In these data, we have 8 variables:

1. Participant - The participant number
2. Gender - Self-reported gender identity with the categories man, non-binary, or woman
3. Age - A grouping variable with the categories “Over25” and “Under25”
4. LSRP - Total score on the Levenson Self-report Psychopathy scale
5. Threshold - The time in seconds from submersion until the participant reports feeling pain in the cold pressor task
6. Tolerance - The total time submerged minus Threshold
7. HR\_B - Baseline heart rate measure before the cold pressor task
8. HR\_CP - Average heart rate during the cold pressor task

First, let’s consider the variable HR\_B. If we want to turn this variable into a set of standard scores, we can easily accomplish this task manually or using the scale() function. By default, whatever variable or matrix of scores you give to the argument “x =” in the scale() function will return a set of standard scores. Alternatively, we could just define a new variable as HR\_B, minus mean(HR\_B), divided by sd(HR\_B). Here is an example of both methods:

# create a new data frame to house our standardized variables.  
standardizedData <- as.data.frame(psychopathyData$HR\_B)  
colnames(standardizedData) <- "HR\_B"  
  
# create a standardized variable using HR\_B with the manual computation method  
standardizedData$HR\_B\_manual <- (standardizedData$HR\_B-mean(standardizedData$HR\_B))/sd(standardizedData$HR\_B)  
  
# now create a standardized variable using HR\_B with the scale() method  
standardizedData$HR\_B\_scale <- scale(standardizedData$HR\_B)  
  
# Use the head function to show that the first few rows of both new variables are identical  
head(standardizedData)

## HR\_B HR\_B\_manual HR\_B\_scale  
## 1 80 0.12999475 0.12999475  
## 2 58 -2.20028153 -2.20028153  
## 3 82 0.34183805 0.34183805  
## 4 78 -0.08184855 -0.08184855  
## 5 86 0.76552465 0.76552465  
## 6 89 1.08328960 1.08328960

# Use the mean() and sd() functions to show that the means of both variables are equal to zero (or extremely, extremely close to zero) and the standard deviations are equal to 1.   
mean(standardizedData$HR\_B\_manual)

## [1] 6.894523e-16

sd(standardizedData$HR\_B\_manual)

## [1] 1

mean(standardizedData$HR\_B\_scale)

## [1] 6.894523e-16

sd(standardizedData$HR\_B\_scale)

## [1] 1

As you can see, there are multiple working methods for standardizing variables. Now here’s the magic part: a linear regression using two standardized variables is identical to a pearson correlation using the unstandardized forms of those same two variables.

Let’s prove it. In the code below, I will use the cor() function (see the next topic for a detailed description) and the lm() function to perform a correlation on unstandardized variables, and a regression on standardized variables.

In this case, I will find the relationship between our variables HR\_B and HR\_CP.

# A correlation on unstandardized variables  
corEx1 <- cor(x = psychopathyData$HR\_B, y = psychopathyData$HR\_CP)  
#Print the results in a conveniently-readable manner (note: I used the 'paste()' function so I could clearly show you where the correlation coefficient value was located)  
paste("The correlation coefficient r =",corEx1)

## [1] "The correlation coefficient r = 0.829291435072104"

# A regression using standardized variables  
lmExample1 <- lm(scale(psychopathyData$HR\_CP)~scale(psychopathyData$HR\_B))  
summary(lmExample1)

##   
## Call:  
## lm(formula = scale(psychopathyData$HR\_CP) ~ scale(psychopathyData$HR\_B))  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.79102 -0.56660 0.03904 0.41426 2.07043   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -7.488e-16 5.353e-02 0.00 1   
## scale(psychopathyData$HR\_B) 8.293e-01 5.377e-02 15.42 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.5614 on 108 degrees of freedom  
## Multiple R-squared: 0.6877, Adjusted R-squared: 0.6848   
## F-statistic: 237.8 on 1 and 108 DF, p-value: < 2.2e-16

Notice at the very top of this output where it says: “The correlation coefficient r =…”. This is the Pearson correlation coefficient computed by the cor() function. If you then look at the regression output from the lm() function, in a the column called “Estimate” and the row for HR\_B, you will find the very same number (perhaps rounded to 4 decimal places, and reported using scientific notation). This is our proof that a standardized regression involving two variables is the same as a Pearson bivariate correlation of those two variables.

Now notice how the row called “Intercept” has an Estimate that is very, very, extremely close to zero. This makes sense because when we standardized our variables we centered them around a mean of zero. Any minute difference from zero here is attributable to the fact that computers must at some point choose to round non-terminating numbers.

Now consider all the other information that the lm() function gives you. Just by thinking about your correlation as if it were a regression, you now have easy access to all kinds of useful information like degrees of freedom, *t* statistics with associated *p*-values for both the independent variable and Y intercept, two measures of effect size (we will get to R2 and R2adjusted in a later topic), and an F statistic that tells you if the whole model is statistically significant.

There is a lot to unpack here, and we will come to each part of this output in good time. Let’s first take a deeper look at computing correlation coefficients.

## Computing correlation coefficients and matrices

As you begin conducting statistical analyses in the field of psychological science, you will be shocked how many correlation coefficients must be computed. Luckily, R has many tools that will help us to do this efficiently.

The simplest of these tools is the **cor()** function from base R. This function takes two arguments, “x =” and “y =”, which identify the independent and dependent variables, respectively. These arguments work the same way as those in the t.test() function worked for independent samples *t*-tests in the last Lesson. Keep in mind that, if you have some NA values in one or both variables, you will also want to include “na.rm = TRUE” to exclude rows with NA values.

*Note: In order to have sensible outputs, these should both be continuously scaled. It is not appropriate to compute a Pearson correlation coefficient for categorical variables (biserial and point-biserial correlation coefficients are a slight caveat to this idea involving dichotomous data, but these days they have mostly been replaced by chi squared tests and logistic regression). In some cases, correlation coefficients can be computed for two ordinal (i.e., rank-ordered) variables using a Spearman’s Rho analysis. If your data are ordinal, you will want to include the argument method = “spearman” in your cor() function.*

Anyway, back to the topic at hand. Once again, here’s our example of a Parson correlation using the HR\_B and HR\_CP variables:

# Compute the correlation  
HRCorrelation <- cor(x = psychopathyData$HR\_B, y = psychopathyData$HR\_CP)  
  
# Call the new object's name to print it to the console  
HRCorrelation

## [1] 0.8292914

Notice how we do not obtain a p-value by default? Let’s try a slight variation on the cor() function, cor.test() from the stats package.

# Compute the correlation  
HRCorrelation <- cor.test(x = psychopathyData$HR\_B, y = psychopathyData$HR\_CP)  
  
# Call the new object's name to print it to the console  
HRCorrelation

##   
## Pearson's product-moment correlation  
##   
## data: psychopathyData$HR\_B and psychopathyData$HR\_CP  
## t = 15.422, df = 108, p-value < 2.2e-16  
## alternative hypothesis: true correlation is not equal to 0  
## 95 percent confidence interval:  
## 0.7600723 0.8799035  
## sample estimates:  
## cor   
## 0.8292914

Splendid! This function gives us lots of useful information. If we wanted to write an APA-style conclusion about this test, we could now say something like:

A statistically-significant positive correlation was observed between participants’ heart rates during the resting baseline and cold pressor task (*r*(108) = .83, *p* < .001).

The previous example was great for situations where you need to compute a single correlation coefficient. But, sometimes you may want to quickly understand the relationships between a large group of continuous variables. In this case, it can be a great idea to create a **correlation matrix**. You can think of a *matrix* as just an arrangement of numbers into columns and rows. As far as R is concerned, a matrix is quite similar to a data frame but has some key differences. For our purposes, it is important to know that a matrix cannot have columns with different classes in the way that a data frame can.

So, a correlation matrix is a numeric arrangement of correlation coefficients into columns and rows. Having all these values in one place makes it very easy to see what variables are most strongly related to each other.

For this example, we will need to load the “psych” package.

library(psych)

The main function we care about is called cor.plot(), and its job is to create plots based on correlations. It has a large number of arguments that you can use, but the main one is “r =” which is looking for a data frame containing only the variables we want to include in the plot we are creating. Here’s an example using three of the variables from this Lesson’s dataset.

# Select the variables we might like to use: LSRP, Threshold, and Tolerance.  
library(dplyr)

## Warning: package 'dplyr' was built under R version 4.2.3

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

selectedVariables <- select(psychopathyData, LSRP, Threshold, Tolerance)  
  
# Create a simple plot of these 3 variables' correlations  
cor.plot(selectedVariables)

A screenshot of a graph

Description automatically generated

The resulting plot is not very pretty, but it is extremely informative. We can easily see the magnitude and direction of the correlations between the three variables we selected. cor.plot() has many, many arguments that let you change how the output looks, and you can learn a bit about them in the activity below.

**Activity**  
Take a look at the correlation plot and code below. Write down a brief explanation of what you think each argument in the code is doing, and then run the code “? cor.plot” to check your answers in the R documentation (Here’s one answer for free: the colorRampPalette() function creates a new colour scale for the legend of our plot).

gradient <- colorRampPalette(c("red", "white", "cyan"))  
  
cor.plot(selectedVariables, main = "Pain Tolerance",   
 scale=FALSE, cex = 1, alpha = 0.75, xlas = 2, diag = FALSE,   
 gr = gradient)

A diagram of a pain tolerance

Description automatically generated with medium confidence

## Correcting for multiple comparisons

In the last lesson, I provided you with a very powerful, but very **dangerous** tool: the correlation matrix. Being able to glance at dozens or even hundreds of correlation coefficients at one time is a great way to sift through a large dataset to find strong positive and negative relationships between variables, but it comes with a heavy cost: inflated Type I error rate. In fact, by the time you test the statistical significance of 14 different correlation coefficients, there is a greater than 50% chance that at least one of them incorrectly appears to be statistically significant because of a type one error.

To illustrate this effect, imagine that you want to test several correlations related to the same general hypothesis. Imagine also that you will use an alpha-level equal to .05 for each test, and that the null hypothesis is in fact true. In this case, the probability of finding at least one statistically significant effect (a Type I error, because H0 is true) is given by this formula:

Family-wise Error Rate (FWER) = 1 - (1 - alpha level)Ntests

We can demonstrate this effect computationally. With just one test, the error rate is right where you would expect it to be – the alpha level.

1 - (1-.05)^1

## [1] 0.05

But, as you add more significance tests to the hypothesis, the ‘family-wise error rate’ – the likelihood that at least one test results in a Type I error – begins to rise. When there are three tests in the family, the FWER inflates to 0.143, or 14.3%.

1 - (1-.05)^3

## [1] 0.142625

After 5 tests, the FWER is 0.226, or 22.6%.

1 - (1-.05)^5

## [1] 0.2262191

And after just 14 tests in a family, the FWER passes 50%, and it becomes more likely than not that you will make at least one Type I error when the null hypothesis is true.

1 - (1-.05)^14

## [1] 0.512325

Fourteen comparisons seems like an unreasonable number of tests to run for one hypothesis, but it’s entirely too easy to blunder into this situation if you dump all your study variables into a correlation matrix and just look for the ones that are significant. In fact, you will conduct 15 tests in a correlation matrix made out of just 6 variables! This is no way to think clearly about psychological experiments.

Correcting for multiple comparisons is an easy process, but many methods become increasingly problematic as the family of tests increases. The general ideas of the **Bonferroni** and **Dunn-Sidak** corrections for multiple comparisons are quite similar: for however many tests are involved in a single hypothesis (i.e., the ‘family’ of tests), we will reduce the alpha level of each test so that their aggregate, *family-wise* error rate matches our alpha level.

The Bonferroni procedure is very straightforward, and simply divides the alpha level used for each test in the family by the number of tests in the family. The Dunn-Sidak method does a similar thing and is very slightly more powerful than the Bonferroni method. This difference becomes more meaningful when a large number of tests are being corrected.

Here is a calculator that gives the Bonferroni and Dunn-Sidak alpha levels for individual tests based on a desired family-wise alpha level. You can change the value of nTests to see how the two techniques diverge.

# Set the desired alpha level for the whole family of tests  
alphaLevel <- .05  
  
# Enter the number of tests to be run (I will put 10 here as a default)  
# Change this number to see how the resulting alpha levels for individuals tests are impacted.  
nTests <- 10  
  
# Conduct a Bonferroni correction  
Bonferroni <- alphaLevel / nTests  
# Print the results to the console  
paste("The Bonferroni method suggests using a per-test alpha-level =", Bonferroni)

## [1] "The Bonferroni method suggests using a per-test alpha-level = 0.005"

# Conduct a Dunn-Sidak correction  
Sidak <- 1-(1-alphaLevel)^(1/nTests)  
# Print the results to the console  
paste("The Dunn-Sidak method suggests using a per-test alpha-level =", Sidak)

## [1] "The Dunn-Sidak method suggests using a per-test alpha-level = 0.00511619689182374"

As you can see, when the size of the family increases, the per-test alpha-level decreases. At some point, it becomes so small that you are exceedingly likely to make a Type II error when the null hypothesis is false. So, it is usually not advisable to just compare every pair of variables in a dataset and report the significant findings; this approach will result in either an inflated Type I error rate if we do not correct for our multiple comparisons, or an inflated Type II error rate if we do correct for our multiple comparisons.

**Reflect**  
Qasim is planning an experiment with one hypothesis that could be supported by any one of 5 different tests. What Is his most-appropriate course of action for testing these hypotheses?  
(“\_\_\_\_\_\_\_\_\_\_”)  
True or False: the Bonferroni and Dunn-Sidak methods are most different when the family of tests is small. Explain your answer.  
(“\_\_\_\_\_\_\_\_\_\_”)

## Computing linear regression

Next, let’s turn our attention to linear regression. In general, we will use linear regression any time you have a continuously-scaled dependent variable, and either A) a single continuously-scaled dependent variable, or B) more than one dependent variable which can be either continuous, or dichotomous. In the last Lesson on t-tests, we saw that an independent samples *t*-test can be treated either as a *t*-test or a linear regression. We will revisit that rationale somewhat in this topic, and expand our understanding to include analytical situations that involve a single, continuously-scaled variable.

There are many different packages that compute some version of linear regression, but for this topics, we will only worry about the most basic one: lm(). This function’s job is to fit a *linear model*, which is just a different way of saying it does linear regression. The package has several important arguments that you will want to know about.

* “formula =” is a mandatory argument where you can define your regression formula. This will involve using the tilde (“~”) symbol, which we previously said is synonymous with the phrase “will be predicted by”. Your variable names go on either side of the tilde. You always want to enter your dependent variable first (i.e., “Y will be predicted by X”).
* “data =” is an optional argument where you can specify a data frame object inside which your variables are found. If you give the name of your data frame to the “data =” argument, then you do not need to repeat it inside the “formula =” argument. Remember, one way or another, you always have to tell R where to find the variables you are testing.
* “na.action =” is a useful option if you have NA values in one or more variables. By setting “na.action = na.fail”, you can tell your model not to run if there is missing data.

Let’s test the hypothesis that the amount of change participants show in heart rate between the baseline and cold pressor task conditions is significantly predicted by their psychopathy score measured using the LSRP. In other words, our independent variable will be the continuously-scaled variable “LSRP” and the dependent variable will be a new variable computed as the difference between HR\_CP and HR\_B. We will compute that variable, run a linear regression model that tests the hypothesis, and then report our results.

# First, compute our dependent variable, which we will call HR\_Change  
psychopathyData$HR\_Change <- psychopathyData$HR\_CP - psychopathyData$HR\_B  
  
# Next, create a new object called "LSRPmodel" that stores our regression model  
LSRPmodel <- lm(formula = HR\_Change ~ LSRP, data = psychopathyData)  
  
# Now, view our results using the summary() function.  
summary(LSRPmodel)

##   
## Call:  
## lm(formula = HR\_Change ~ LSRP, data = psychopathyData)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -8.941 -4.779 -1.304 4.419 32.577   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 16.46927 1.94341 8.474 1.32e-13 \*\*\*  
## LSRP -0.18134 0.03797 -4.776 5.66e-06 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 6.008 on 108 degrees of freedom  
## Multiple R-squared: 0.1744, Adjusted R-squared: 0.1667   
## F-statistic: 22.81 on 1 and 108 DF, p-value: 5.658e-06

Let’s walk through this output line-by-line.

* “Call” reminds us of the exact formula we used to get these results. If you save this model to a file that you export from R, this is extremely useful for keeping track of where results came from.
* “Residuals” refer to the differences between the HR\_Change scores predicted by our linear regression model, and the actual HR\_Change scores that occurred at each value of our X variable. The reported values here show the biggest differences in the positive (“Max”) and negative (“Min”) directions, as well as the middle-most residual (“Median”). Finally, R shows us the residual values below which 25% (“1Q” for first quartile) and 75% (“3Q” for third quartile) fall. These scores are most useful for beginning to understand if the predictions a model makes are biased in either direction, or if there are extreme outliers that could not be captured well by the regression model. Since we have a Maximum residual that is 32 points (or ‘heartbeats per minute’) above what was predicted by our regression model, we might want to investigate this data point later to see if we need to find a better strategy for modelling our data.
* “Coefficients” is the area where you will spend most of your attention in any regression output. Notice the column called “Estimate”. This is where you will find the slope for each independent variable (in the row corresponding to our independent variable “LSRP”) as well as the value of the Y-intercept (in the row called “Intercept”). The coefficient for our independent variable can be interpreted by saying “For every 1 unit that LSRP increases, we predict that HR\_Change will decrease by -0.1813419 heartbeats per minute. This is an intuitive way to understand the practical significance of any effect.

Next, notice how both the Y intercept and independent variable get a *t* value with a corresponding *p* value. This tells us the statistical significance of both the slope of the independent variable LSRP compared to a flat horizontal line, and the Y-intercept compared to zero. In this case, the Y intercept is significantly greater than zero, and the LSRP variable has a negative slope that is statistically significant.

Next, notice how a standard error is given for each estimated effect. This is an estimate of uncertainty for each coefficient in the “Estimate” column. This can be useful for several reasons, like calculating a confidence interval around an Estimate (but we won’t worry too much about it here).

* “Signif. codes:” tells you how to interpret R’s use of asterisks next to each row of coefficients. 3 Asterisks indicates that a *p*-value is less than .001, 2 asterisks indicates *p* < .01, 1 asterisk indicates that *p* < .05. R also gives a ‘.’ to indicate *p* < .10, but this is rarely useful.
* “Residual standard error” tells us the average distance that the observed values fall from the regression line. Smaller values indicate that our independent variable(s) are doing a good job explaining everything going on with our dependent variable. This row also tells you how many degrees of freedom were used to test the statistical significance of the model. The formula for degrees of freedom in linear regression is DF = N-K-1 where N is the number of participants, and K is the number of independent variables.
* “Multiple R-squared” and “Adjusted R-squared” are measures of effect size. To learn the difference between them, see the topic “R2 and R2adjusted” below.
* “F-statistic” provides a significance test that indicates whether the independent variable(s) in the model explain significantly more variability in the dependent variable than a model that has no predictors at all. In essence, it indicates if the model is worth using in the first place. It is computed and reported in the typical way that ANOVA models are handled [[13]](#footnote-13).

With all that said, let’s look at our specific results and try to understand what is going on in our model. Here’s what I notice:

1. The F-statistic is significant. This tells us that the independent variable is better than nothing.
2. The model intercept is statistically significant and has a value of 16.4692694. This means that a participant with an LSRP score of zero is predicted to increase their heart rate by that many beats per minute when they go from the resting baseline recording to the cold pressor task condition.
3. The independent variable LSRP is statistically significant, indicating that the amount of HR\_Change we predict to occur depends on what a given participants’ LSRP score is. I see that the Estimate, -0.1813419, is a negative number. This means that – as LSRP increases, HR\_Change decreases. In other words, people with higher psychopathy scores show less change in heart rate when they do a cold pressor task. Finally, I see three asterisks indicating that the *p*-value for this significance test is less than .001.
4. I notice that this model used 108 degrees of freedom. This makes sense because I had N = 110, and K = 1 independent variables, and 110 - 1 - 1 = 108.
5. I notice that R-squared is equal to .17, and understand that about 17% of the variability in HR\_Change is attributable to variability in LSRP.
6. Finally, I notice that the median residual is quite close to zero (this is good because it does not show a strong bias in either direction), but the maxmimum residual is very large. I should investigate this value to see if a mistake was made when the data were recorded.

Suppose that I wanted to tell someone about this regression model. What kinds of details would I need to include? I’d certainly want them to know that the model was significant overall, but I’d also want them to understand the specific effect of my independent variable (conventional reporting uses the letter *B* for unstandardized regression coefficients, and the Greek capital letter Beta for standardized regression coefficients). So, I might say something like:

“A significant relationsip was observed between LSRP scores and HR\_Change (*R*2 = .17, *F*(1, 108) = 22.81, *p* < .001). For every five units increase in self-reported LSRP, HR\_Change was predicted to decrease by approximately one beat per minute (*B* = -0.18, *p* < .001).

**Explain**  
How many degrees of freedom would an experiment with 25 participants and 4 independent variables have? (“\_\_\_\_\_\_\_\_\_\_”)  
When might a Y-intercept be very close to – or even equal to – zero? Would this create any problems for the rest of the analysis?  
(“\_\_\_\_\_\_\_\_\_\_”)  
What might be one reason that a coefficient estimate and its statistical significance would both be very small numbers?  
(“\_\_\_\_\_\_\_\_\_\_”)

## Assumptions and diagnostic plots for linear regression

Just like *t*-tests, there are several important assumptions underlying every regression (and by extension, every correlation) analysis. If we are carefully following the analysis pipeline (see Figure 3.1), we understand that we should check these assumptions *before* we create any regression model and *especially before* we calculate any *p*-values. For some assumptions, this is always feasible. But other assumptions will need to be checked once the model has been fit and, unfortunately, R will usually give you a statistical significance test at the same time as a model is fit. My recommendation would be to use the functions we are going to discuss in this topic *after* you have run the lm() function to create a model, but *before* you use the summary() function to report the significance of the model.

The assumptions underlying linear regression models are:

1. Independence. In order for *p*-values associated with regression or correlation to be meaningful, participants in a study should be considered to have been randomly sampled from the population of interest. If there is any non-random relationship between two or more participants, then the assumption of independence has not been met. In practice, this most-often occurs when the sampling practices for one condition differ from other conditions. For example, if your experimental group is drawn from a clinical population while your control group is sampled from a University research pool. This kind of difference can render any experimental manipulation meaningless because numerous confounding variables could be present.
2. Normality means that all continuously-scaled variables should each approximate the normal distribution. If one variable in a correlation analysis is not normal, the absolute maximum possible value of the correlation coefficient actually decreases by some amount.
3. Homoscedascity comes from the Greek prefix hom, along with the Greek word skedastikos, which means ‘able to disperse’. Homoscedascity means that the variability of errors around the line of best fit is the same across the full range of the independent variable. If the variance of errors is smaller for a particular range of values of independent variable and larger for another range of values, then there is a violation of homoscedascity. It is quite easy to check for homoscedascity visually, by looking at a scatter plot or scale-location plot (see below). If the points lie equally on both sides of the line of best fit, then the data is homoscedastic. We will look at one such example later in this topic.
4. Linearity means that the data follows a linear relationship. If the relationship between X and Y hypothetically formed a “U” shape, then linear regression would not be appropriate. Fitting a straight line through a curving trend would require you to ignore much of the relationship between the two variables.
5. Paired observations: every data point must be in pairs. For every observation of the independent variable, there must be a corresponding observation of the dependent variable. We cannot run a regression on a full dataset if one variable has 12 observations and the other has only 10 observations. If R sees missing data, it will either fail to run the code, or ignore any rows containing missing data (depends on what you tell it to do).
6. No extreme outliers should be present in the data. Outliers can significantly skew the correlation and regression analyses and make them misleading. The Residuals vs. Leverage plot is particularly good for uncovering violations of this assumption.

### Diagnostic plots for regression models

One of the best ways to get a handle on what’s going on in your regression model is to use the plot() function on your regression model. This will give you four specialized plots that are extremely useful for assessing the tenability of various assumptions. Let’s go through them one at a time.

plot(LSRPmodel)

A graph with red line and black dots

Description automatically generatedA graph of a normal q-q

Description automatically generatedA graph of a line graph

Description automatically generated with medium confidenceA graph of a number of lines

Description automatically generated with medium confidence

1. Residuals Vs. Fitted  
   “Residuals” are the error values between the scores that we predict on our Y variable and the actual values that were observed on the Y variable. “Fitted” values are the predicted scores we created for the Y variable based on the scores we have on the X variable.

**What this shows:** if residuals have non-linear patterns. There could be a non-linear relationship between predictor variables and an outcome variable and the pattern could show up in this plot if the model doesn’t capture the non-linear relationship.

**What we use it for:** Assessing the assumption of linearity.

**What to look for:** If you find equally spread residuals around a horizontal line without distinct patterns, that is a good indication you don’t have non-linear relationships. I consider the plot you see above to be reasonably linear.

1. Normal Q-Q plot

**What this shows:** Whether our data are skewed.

**What we use it for:** Assessing the assumption of normality.

**What to look for:** A good QQ plot looks like a dots falling along diagonal line. If a variable is badly skewed, or non-normal, the data points will curve away from the diagonal. This model is reasonably linear, but the data point in row 82 of our data frame looks like it might be problematic. Let’s keep an eye on that one.

1. Scale-Location plot

**What this shows:** whether or not Y variable residuals fall equally far from the line of best fit across the whole range of the X variable

**What we use it for:** Assessing homoscedasticity

**What to look for:** Your data are homoscedastic if you have a roughly horizontal line with data points spread roughly equally around it. The plot above is a marginal case. There might be a slight problem with homoscedasticity here because Y-axis values seem to vary more as X increases.

1. Residuals vs. leverage

“Leverage” is a way of measuring both: (1) the extremeness of scores, and (2) the impact they have on model and significance tests. Not all outliers are going to adversely affect our estimates. Sometimes outliers fall very close the trend line that would have been fit if the outlier had been excluded. At that point, there’s little harm to leaving them in the data.

**What this shows:** Cases where leverage is high enough to cause problems for our estimated correlation coefficient.

**What we use it for:** Identifying high-leverage outliers in the data.

**What to look for:** If a case is beyond the dashed lines (“Cook’s distance”), then they have too much leverage on our model and need to be excluded. The data in line 82 of our data frame is clearly problematic for our estimate of effects. This score should be excluded.

**Reflect**  
Why is it a problem to look at a model’s *p*-values before looking at its Residuals vs. Leverage plot?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Since every correlation analysis depends on these assumptions, what is the most efficient way of running correlations in R without ignoring any diagnostic plots?  
(“\_\_\_\_\_\_\_\_\_\_”)  
It is very common for psychology researchers to report a large number of correlation coefficients in research papers without describing in detail the modelling procedures that were used to create them. What might be one way to make this information available to other researchers without creating a lot of clutter in the research paper itself?  
(“\_\_\_\_\_\_\_\_\_\_”)

## R2 and R2adjusted

Let’s take a moment to think about effect sizes in the context of correlation and linear regression. Probably the easiest way to extract an effect size measurement from a correlation coefficient is to compute **R2**, the **Coefficient of Determination**. This is probably the most-easily-computed statistic you will encounter, because it simply requires that you square a correlation coefficient.

For example,  
a value of r = -.50  
corresponds to R2 = .25  
because -0.50 \* -0.50 = 0.25

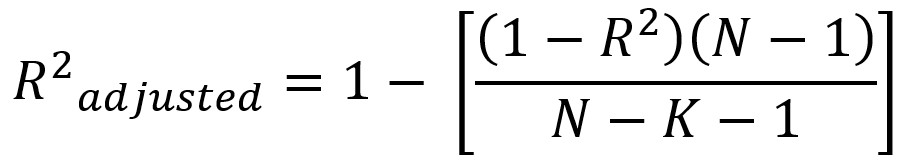
Notice how negative correlation coefficients correspond to positive coefficients of determination. R2 has a minimum value equal to zero and a maximum value equal to one. So, regardless of the direction of your effect, R2 indicates *the proportion of variability in your dependent (Y) variable that is attributable to variability in your independent (X) variable(s)*.

This is the point where things get a little more complicated. You see, correlation and linear regression have a very big problem with *capitalizing on random measurement error*. It is rare – verging on impossible – to find a correlation or regression coefficient that is exactly equal to zero when the null hypothesis is true. Although there is no real effect of our X variable on our Y variable, random measurement errors will – by random chance alone – usually result in some small correlation between the two variables. The model, in turn, uses this random measurement error when estimating the nature of the relationship between X and Y. This is usually not much of a problem when you only have one independent variable, but it becomes a huge problem when you start conducting larger **multiple regression** models using several, dozens, or even hundreds of independent variables.

This is where **R2adjusted** (or you might see the notation R2adj) comes in. This statistic tries to account for the number of independent variables (K in the formula below) when it estimates what proportion of variability in Y is accounted for by variability in some number (K) of variables X.

Why am I talking about multiple regression and R2adjusted in a lab manual that barely touches multivariate statistics? Because you will see the phrase “Adjusted R-Squared” right next to the phrase “R-Squared” in the output for every regression model you create using the lm() function. These numbers will never been exactly the same, and you ought to know which one is most appropriate to report in a particular analysis. As a general rule, correlations and linear regression analyses can report a regular R2 statistic, while multiple regression models with lots of independent variables should use R2adjusted instead.

Here’s the formula for R2adjusted. Notice how the K term appears in the denominator, so increasing the number of independent variables will always reduce R2adjusted UNLESS those variables happen to be strongly related to the dependent variable Y.



**Figure 6.1. The formula for R2adjusted.**

Here’s a bit of R code that computes R2adjusted for a particular correlation coefficient.

# Create a new object for a given correlation coefficient  
r <- -0.875  
  
# Define some total sample size (N) and some number of independent variables (K)  
N <- 100 # A number of participants  
K <- 4 # A number of independent variables  
  
# Compute adjusted R squared  
R2adj <- 1 - ((1-r^2)\*(N-1))/(N-K-1)  
  
# Use the paste() and round() functions to report a result rounded to 4 decimal places.  
paste("Adjusted R squared =", round(R2adj,4))

## [1] "Adjusted R squared = 0.7558"

**Activity**  
Using the formula above and the commented code steps below, create two scatterplots that show R2 and R2adjusted for r = 0.5, N = 25, and the following values of K: [1,3,5,10,25,50,500].

# First, create objects for r, N, and K. You will need to use the c() function for the different values of K.  
  
# Next, create a data frame that contains the values of K in a variable.  
  
# Next, create a variable equal to R squared for each value of K.  
# (Hint: It's OK if all values are the same!).  
  
# Next, create a variable equal to adjusted R squared for each value of K.  
  
# Next, load the ggplot2 package.  
  
# Next, create the first ggplot with an X aesthetic for your K variable, and a Y aesthetic for your R squared variable.  
  
# Finally, create the second ggplot with an X aesthetic for your K variable, and a Y   
# aesthetic for your adjusted R squared variable.   
  
# Comparing the two plots, write down some things you notice.

## Visualizing regression and correlation

The most common method for plotting linear regression and correlation models is to create a scatterplot with a line of best fit. Luckily, Lesson 3 provides a step-by-step guide to creating these exact plots! Rather than re-tread that ground here, I will just provide the syntax that one might use to make a publication-ready figure displaying the results of the model we created with the variables LSRP and HR\_Change. Challenge yourself to describe the purpose of each function and argument in the syntax below, and remember that you can check the documentation for any function using the “?” operator in your R console.

library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.2.3

regressionPlot <- ggplot(data = psychopathyData, aes(x=LSRP,y=HR\_Change)) +   
 labs(x = "Psychopathy Score", y = "Change in Heart Rate") +   
 geom\_point(color="black",size=2) +   
 geom\_smooth(method = "lm" , se = FALSE, color="black", linewidth=1.5)  
regressionPlot

A graph with a line and a line

Description automatically generated

## Logistic regression for dichotomous dependent variables

So far, we have looked at regression in the context of a single continuous independent variable and a single continuous dependent variable. But, there is so much more that it can do! *Multiple regression* methods can accommodate several (and in some rare cases, hundreds or thousands) of independent variables. *Regularized regression* is a special case of regression where model slopes are penalized (made more horizontal) in order to counteract the natural tendency for regression models to capitalize on random error variance when fitting trends. *Bayesian regression* models are sensitive to the probabilities of different outcomes occurring in our data. The list goes on and on, and miraculously, many of these methods can be mixed and matched to create complicated modelling strategies that are appropriate for whatever a particular dataset might look like.

For the last topic in this lesson, I want to highlight one special case of regression called **logistic regression**. Logistic regression is used whenever we have a *binary* or *dichotomous* dependent variable that is coded as a “this-or-that” event. Examples include whether or not a stoplight is green when your car enters an intersection, and whether or not a participant ate breakfast the day they participated in your experiment.

When our outcome variable in a regression model is dichotomous, we need to change our thinking about what the regression model output is saying. In a normal linear regression, the estimate associated with our independent variable tells us how much we expect a predicted score on the dependent variable to change given a one unit increase in the independent variable. In a logistic regression, we stop thinking in terms of predicted units on the Y variable, and instead think about the change in *likelihood* of occurrence for one of the dichotomous outcomes.

To go along with this difference, the estimates given by our independent variables in a logistic regression model are expressed as *log odds*, and the exponent of any estimate is an *odds ratio* (look back at our lesson on Chi Squared if you are still unsure about odds ratios).

For example, if you created a logistic regression predicting whether or not students ate breakfast from their standardized test scores later that day, and the exponent of the estimate for the test score variable was 1.06 (remember, this is just an odds ratio), then we would conclude that every unit increase in a given student’s standardized test scores increased the likelihood that they ate breakfast by 6%.

Here’s an example from our psychopathy dataset. Here, we will create a logistic regression model that predicts whether participants are over 25 years of age from the amount of heart rate change they showed during the cold pressor task. We hypothesize that different amounts of heart rate change will occur for people who are over 25 years of age.

To compute a logistic regression in R, we will need to use the glm() function. This function works very much the same as lm(), but allows us to include a new argument “family =”, where we will specify that we want to use a *binomial* distribution for our dependent variable. This is what makes a logistic regression happen.

There is one other problem to fix before we run this model: glm() is expecting to see a numeric binary variable. So, we will need to convert the character variable “Age” to a numeric variable where being under 25 is given a score equal to zero, and being over 25 is given a score equal to 1. We will use the **ifelse()** function to say that our new variable is equal to 1 if the score for Age in the same row is “Over25”, and otherwise it should be zero.

# Create our dependent variable and call it AgeBinary  
psychopathyData$AgeBinary <- ifelse(psychopathyData$Age == "Over25", 1, 0)  
  
# Fit our model and store it inside an object called "logisticModel".  
logisticModel <- glm(formula = AgeBinary~HR\_Change, family = 'binomial', data = psychopathyData)  
   
# Print a model summary to the console  
summary(logisticModel)

##   
## Call:  
## glm(formula = AgeBinary ~ HR\_Change, family = "binomial", data = psychopathyData)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -1.332 -1.280 1.058 1.070 1.087   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)  
## (Intercept) 0.224542 0.294111 0.763 0.445  
## HR\_Change 0.004137 0.029373 0.141 0.888  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 150.71 on 109 degrees of freedom  
## Residual deviance: 150.69 on 108 degrees of freedom  
## AIC: 154.69  
##   
## Number of Fisher Scoring iterations: 3

# Calculate an odds ratio for our independent variable using the exp() function  
paste("Odds ratio for HR\_Change =",exp(logisticModel[["coefficients"]][["HR\_Change"]]))

## [1] "Odds ratio for HR\_Change = 1.00414529732593"

From this model, notice that we have several familiar elements, including residuals, estimates, standard errors, and *p*-values. Notice how we used the exp() function to turn our estimate for HR\_Change into an odds ratio. We can interpret this result as indicating that each unit increase in HR\_Change increased the odds of being over 25 years of age by 0.4145%. This odds ratio is very small, and its associated *p*-value is not statistically significant. Nevertheless, logistic regression remains an excellent way to examine relationships between independent variables and dichotomous dependent variables.

**Reflect**  
Is it OK to do a logistic regression when the independent variable is also binary? Why or why not?  
(“\_\_\_\_\_\_\_\_\_\_”)  
What units do logistic regression standard errors appear to be in?  
(“\_\_\_\_\_\_\_\_\_\_”)

We have reached the end of another challenging lesson in R statistics! Before you go, make sure to write down your three major takeaways and review their key concepts when we come back next week.

**Key Takeaways**  
The most important lessons from this Lesson were:  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)

Lesson 7 - ANOVA

Last edited on: 2023-12-07

# Lesson 7 - One-way ANOVA

Welcome! In the last Lesson, we took a brief look at a few important topics in the wide world of regression. In this lesson, we turn back towards group mean differences. Here, we will consider the very common situation where we want to find out:

1. if statistically significant between-groups differences exist in an experiment with three or more groups, and if so…
2. which groups significantly differ from one another.

In order to do this, we will begin exploring the topic of Analysis of Variance, also known as ANOVA. This lesson assumes that you have previously encountered ANOVA in an introductory statistics course. We will not return to the conceptual or computational basics of ANOVA itself, but rather will expand on these concepts and consider several new and relevant dimensions to this test with which you are already familiar.

### Our objectives this lesson:

* Compute one-way, between-groups ANOVA
* Learn to test core assumptions of ANOVA
* Test some planned orthogonal contrasts
* Test post-hoc pairwise comparisons
* Compute Eta Squared, an effect size for ANOVA
* Visualize group means and standard errors

## Consider this dataset

*This example data comes (with thanks) from Zakary Draper*

Building Academic Retention through K9s (B.A.R.K.) is a dog therapy program at The University of British Columbia’s Okanagan Campus. BARK provides students opportunities to spend time with trained therapy dogs and dog handlers.

Several studies have shown that these and similar canine assisted interventions reduce acute stress and improve mood. However, it is unclear what aspects of these interventions are important for reducing stress. Does the benefit come from simply being with other people? Is the dog’s presence important? Or, does most of the benefit come from actual physical contact with the dog?

The researchers hypothesized that the effect of canine-assisted interventions would vary based on the presence of absence of a dog, and based on whether participants have physical contact with the canine.

Participants and Procedure

The current study sought to answer these questions using a randomized controlled trial. Participants (N = 300) were split into groups of 4–6 participants. Each group of participants was randomly assigned to one of three conditions:

1. direct contact with a BARK dog,
2. indirect contact with a BARK dog, or…
3. handler only

In the handler only condition, participants talked with the handler and other participants with no dog present. In the indirect contact condition, the handler was present with the therapy canine, but participants were instructed not to have physical contact with the dog. In the direct condition, participants were instructed to maintain physical contact with the canine for the duration of the session. Each session lasted 15–20 minutes.

After each canine therapy session, participants responded to a single-item measure of their self-perceived current stress, measured on a Likert scale ranging from 1 (“Not at all stressed”) to 5 (“Very stressed”).

The dataset for this experiment has 3 variables:

* pid - The participant ID number
* condition - The grouping variable, with values “Direct”, “Indirect”, and “Handler only”.
* stress - The dependent variable, stress ratings given on a 5-point scale.

This code loads our data:

# Clear our environment from the last lesson  
rm(list=ls())  
  
# Load the BARK dataset  
BARKdata <- read.csv("BARKdata.csv")

## Computing ANOVA

Based on the data we just loaded, we can test the hypothesis that at *least one experimental condition significantly differs from at least one other experimental condition*. Using R, there are several different ways to compute a one-way (meaning you have just one independent variable), between-groups (meaning that you have measured the dependent variable only once for each participant) analysis of variance.

This method uses the aov() function, and provides a good amount of information for our hypothesis test. AOV has a couple familiar arguments that you will want to include:

* “formula =” uses the same format as the lm() function we explored in Lesson 6.
* “data =” is an optional argument that can be helpful if you want to use variables that exist inside of a data frame.

# Use the aov() function to fit a model.  
aovMethod <- aov(formula = stress ~ condition, data = BARKdata)  
  
# Use the summary() function to print the results to the console.  
summary(aovMethod)

## Df Sum Sq Mean Sq F value Pr(>F)   
## condition 2 35.33 17.663 20.89 3.24e-09 \*\*\*  
## Residuals 297 251.11 0.845   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Many of the numbers in this output will be familiar from your previous statistics class. Here, we have two rows of data: with ‘condition’ referring to the between-subjects variability of our grouping variable that was called ‘condition’, and ‘Residuals’ referring to the within-subjects variability that is present in our data. You will notice that our F statistic comes from dividing the two Mean Sq values that are reported, and that each Mean Sq value is itself the result of dividing its Sum Sq value by its Df. When it comes to reporting this result using APA format, we might say something like:

“A significant effect of study condition was observed (*F*(2, 297) = 20.89, *p* < .001).”

With ANOVA models, we will typically report a lot more information that is relevant to specific comparisons we make between groups when this overall (“omnibus”) model is statistically significant. For the remainder of this lesson, we will frequently revisit this result and add new, relevant details to it.

Here’s another way to compute ANOVA models using R. This method uses the oneway.test() function, and is mostly the same as the aov() method with two key differences. First, the default results given by oneway.test() use fractional degrees of freedom. This is useful when the Homogeneity of Variance assumption is not tenable (see our next topic for a full explanation). In order to get a regular ANOVA out of this function, we must specify the argument “var.equal = TRUE”. We previously saw this argument used in the t.test() function in Lesson 5.

The second major difference is that oneway.test() does not use the summary() function the same way as aov(). Instead, it prints a shorter selection of model results when the object’s name is called.

# Use the oneway.test() function to fit a model.  
owtMethod <- oneway.test(formula = stress ~ condition, data = BARKdata, var.equal=TRUE)  
  
# Call the object's name to print the results to the console.  
owtMethod

##   
## One-way analysis of means  
##   
## data: stress and condition  
## F = 20.891, num df = 2, denom df = 297, p-value = 3.244e-09

Before we move on, I want to show you that ANOVA can also be computed as a form of linear regression using our friend the lm() function. This method works exactly the same way as the aov() method we looked at first, but it requires us to tell R that our independent variable called “condition” should be treated as a categorical variable. One way to do this is by using the factor() function.

# Use the lm() function to fit a model.  
regressionMethod <- lm(formula = stress ~ factor(condition), data = BARKdata)  
# Use the summary() function to print the results to the console.  
summary(regressionMethod)

##   
## Call:  
## lm(formula = stress ~ factor(condition), data = BARKdata)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1.9100 -0.5125 -0.0800 0.6200 2.6200   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 2.08000 0.09195 22.621 < 2e-16 \*\*\*  
## factor(condition)Handler Only 0.83000 0.13004 6.383 6.67e-10 \*\*\*  
## factor(condition)Indirect 0.30000 0.13004 2.307 0.0217 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.9195 on 297 degrees of freedom  
## Multiple R-squared: 0.1233, Adjusted R-squared: 0.1174   
## F-statistic: 20.89 on 2 and 297 DF, p-value: 3.244e-09

One major advantage of this approach is that the lm() model gives you a *lot* more information. We can see the results of the ANOVA model reported on the bottom row of the output. We also see an effect size reported as both Multiple R-squared and Adjusted R-squared. Finally, and perhaps most importantly, we can see that the direct contact group (represented by the “Intercept” row in this analysis) has a mean stress score of 2.08, while the Handler Only group reported higher stress equal to 2.08 + 0.83 = 2.91, and the Indirect contact condition had a mean stress score equal to 2.08 + 0.30 = 2.38. Looking at their associated *p*-values, we can see that both the Handler Only and Indirect study conditions were statistically different from our *reference category*[[14]](#footnote-14), the Direct contact condition.

The main disadvantages of this approach are: (1) the resulting model does not compare the Handler Only and Indirect conditions in order to determine statistical significance, and (2) the reported *p*-value for the omnibus model will be incorrect if the Homogeneity of Variance assumption is not tenable. So, sometimes you might be able to use a regression approach for your ANOVA calculations, and other times you might not[[15]](#footnote-15).

**Explain**  
When I know that the Homogeneity of Variance assumption is untenable, I will use the function called…  
(“\_\_\_\_\_\_\_\_\_\_”)  
When I have a single control group that I want to compare every experimental condition to, I will use the function called…  
(“\_\_\_\_\_\_\_\_\_\_”)  
When I want to clearly see the mean squares of my ANOVA model, I will use the function called…  
(“\_\_\_\_\_\_\_\_\_\_”)

## ANOVA assumptions

The assumptions underlying one-way, between-groups ANOVA are the same as we saw with *t*-tests in Lesson 5. That is, all observations should be *independent*, the distribution of dependent variable scores should be *normally* distributed, and the variances of dependent variable scores within each category of the indepependent variable need to be fairly similar (or *homogenous*).

### Independence

Once again, it is the case that independence will be assessed by verifying that each participant appears only once in our dataset, and that the sampling method selected for this experiment does not undermine the principle of random selection.

### Normality

For ANOVA models, normality is handled in much the same way as we saw with *t*-tests. Normality can again be understood by plotting our dependent variable using a QQ plot (this is the same procedure seen in Lesson 5). Luckily, the aov() function has a diagnostic implementation of the plot() function that is identical to that of lm(). So, we can simply request a QQ plot by typing:

# print a QQ plot for our ANOVA model.  
plot(aovMethod, which = 2)

A graph of a normal q-q

Description automatically generated

Taking into account the fact that we have a limited number of possible scores on the stress variable (1, 2, 3, 4, or 5), I would generally say that the distribution of these data appears quite normal. Let’s see whether the Shapiro-Wilks test thinks this variable is normally distributed…

shapiro.test(BARKdata$stress)

##   
## Shapiro-Wilk normality test  
##   
## data: BARKdata$stress  
## W = 0.89585, p-value = 1.722e-13

Wow! That’s a very small *p*-value, and the Shapiro-Wilk test statistic (*W*) is pretty far from 1.0! A good rule of thumb is that ‘normal’ variables have a *W* statistic greater than or equal to 0.975. Based on our low *W* statistic and statistically-significant *p* value, we have evidence suggesting that the assumption of normality has been violated. In a case such as this, this statistically significant result is probably attributable mostly to our large sample size.

In a practical sense, our sample size is large enough that we are probably not going to create a huge problem by proceeding as though these data are normally distributed. Nevertheless, if we were concerned about violations of the assumption of normality, we could run a **Kruskal-Wallis** test instead of a parametric ANOVA. A footnote on the previous page explains how to accomplish this easily using the lm() and rank() functions. Let’s compare the results of our parametric (ANOVA; assumes normality) and non-parametric (Kruskal-Wallis; does not assume normality) tests.

# Assuming normality  
ANOVA <- lm(formula = stress ~ factor(condition), data = BARKdata)  
summary(ANOVA)

##   
## Call:  
## lm(formula = stress ~ factor(condition), data = BARKdata)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1.9100 -0.5125 -0.0800 0.6200 2.6200   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 2.08000 0.09195 22.621 < 2e-16 \*\*\*  
## factor(condition)Handler Only 0.83000 0.13004 6.383 6.67e-10 \*\*\*  
## factor(condition)Indirect 0.30000 0.13004 2.307 0.0217 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.9195 on 297 degrees of freedom  
## Multiple R-squared: 0.1233, Adjusted R-squared: 0.1174   
## F-statistic: 20.89 on 2 and 297 DF, p-value: 3.244e-09

# Not assuming normality (achieved by simply rank-ordering our dependent variable)  
KruskalWallis <- lm(formula = rank(stress) ~ factor(condition), data = BARKdata)  
summary(KruskalWallis)

##   
## Call:  
## lm(formula = rank(stress) ~ factor(condition), data = BARKdata)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -161.93 -48.48 -12.75 64.67 156.75   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 119.250 7.804 15.281 < 2e-16 \*\*\*  
## factor(condition)Handler Only 69.175 11.036 6.268 1.28e-09 \*\*\*  
## factor(condition)Indirect 24.575 11.036 2.227 0.0267 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 78.04 on 297 degrees of freedom  
## Multiple R-squared: 0.1197, Adjusted R-squared: 0.1138   
## F-statistic: 20.19 on 2 and 297 DF, p-value: 5.991e-09

Comparing the two tests, we can see that relatively little has changed. For the non-parametric Kruskal-Wallis version, our *p*-values are a little larger, and our Estimates are scaled as ranks rather than raw scores. But our conclusions about the null hypothesis remain the same. This will often be the case when you have a relatively large number of observations because, as sample size increases beyond about N = 30 (Ghasemi & Zahediasl, 2012), parametric tests become relatively robust against violations of the assumption of normality. Although a lot of data scientists do not use significance tests of normality and rely instead on the QQ plot and sample size, scientists looking for a reasonable inferential approach to assessing normality could follow these steps:

1. Always visualize your dependent variable to determine if the variable has some central tendency. If no central-tendency is apparent, run a Kruskal-Wallis test, or select a more-appropriate alternative test fitting the distribution of scores that occurred (e.g., a logistic regression or chi squared test may be more appropriate if your variable shows participants selected one of only two of your response options).
2. If your sample size is small (less than 30 or 40), run a Shapiro-Wilk test. If that test is not statistically-significant at an alpha-level = .05, proceed with a parametric ANOVA. If the test is statistically-significant, proceed with a Kruskal-Wallis test instead.
3. If your sample size is larger, and the visualization shows central tendency, proceed with a parametric ANOVA.
4. If your sample size is larger, and your visualization indicated some central tendency but you remain concerned about normality for any reason, it probably won’t hurt to proceed with a Kruskal-Wallist test.

### Homogeneity of Variance

Once again, we will use Levene’s test to check whether we are able to assume that the group variances in our experiment are homogeneous. Again, HoV is a more serious problem when group sizes are unbalanced. We have 100 participants in each condition of this experiment, but we will run a Levene’s test because it is good practice. As with every Levene’s test we compute, we will adopt a less-stringent alpha-level equal to .10. If our *p*-value is smaller than .10, we will reject the null hypothesis that our groups have homogeneous variances.

# Make sure to load the car package  
library(car)

## Loading required package: carData

# Run Levene's test and print the results to the console.  
# Note: We have used the factor() function here, because the leveneTest()  
# function is expecting to see a grouping variable that is a factor.  
leveneTest(y = BARKdata$stress, group = factor(BARKdata$condition))

## Levene's Test for Homogeneity of Variance (center = median)  
## Df F value Pr(>F)   
## group 2 3.151 0.04424 \*  
## 297   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Based on this *p*-value, we must reject the null hypothesis, conclude that the group variances are not homogenous, and adjust our ANOVA model using a fractional degrees of freedom. In R, this last step is as simple as setting the argument “var.equal = FALSE” inside the oneway.test() function. So, our omnibus model should looks like this:

# Use the oneway.test() function to fit a model.  
welchANOVA <- oneway.test(formula = stress ~ condition, data = BARKdata, var.equal=FALSE)  
  
# Call the object's name to print the results to the console.  
welchANOVA

##   
## One-way analysis of means (not assuming equal variances)  
##   
## data: stress and condition  
## F = 21.037, num df = 2.00, denom df = 195.13, p-value = 5.328e-09

Notice that our *F* statistic, *p*-value, and denominator degrees of freedom have all changed. In this case, our decision to reject the null hypothesis remains the same. But this will not always be the case because the Welch test operates with slightly less power than the ordinary ANOVA. Occasionally, you might see a statistically significant *p*-value when var.equal = TRUE, and a non-significant *p*-value when var.equal = FALSE. In order to find the correct conclusion, you will need to think clearly about whether or not the Homogeneity of Variance assumption is tenable in the data you collected.

Armed with this new information, we might amend our earlier conclusion to say:

“Levene’s test indicated significant differences in group variances (*F*(2, 297) = 3.15, *p* = .04). A significant effect of study condition was observed for the corrected omnibus model (*F*(2, 195.13) = 21.04, *p* < .001).”

**Reflect**  
What assumption might be violated if the researchers had asked participants if they felt afraid of dogs, and then put everyone who said ‘yes’ in the Handler Only condition?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Could there ever be a situation where the data truly are normally distributed, but the Shapiro-Wilk test returns a statistically-significant *p*-value?  
(“\_\_\_\_\_\_\_\_\_\_”)  
When might a researcher be justified in not testing for or reporting the results of a Levene’s test?  
(“\_\_\_\_\_\_\_\_\_\_”)

## Group comparisons

So far in this lesson, we have run an omnibus, one-way, between-groups ANOVA. The results of this analysis tell us that at least one of our groups is significantly different from at least one other group. But which ones?

A key problem we face with analysis of variance models (compared to *t*-tests) is that we do not know *which* groups significantly differ unless we somehow test them. In this topic, I will present two methods for comparing groups given a statistically significant omnibus ANOVA model.

### Planned Orthogonal Contrasts

The most-powerful way to identify statistically significant difference between groups is to create a set of **Planned Orthogonal Contrasts (POC)**. This requires you to specify what groups will be compared *before you compute your omnibus ANOVA model*. Since each contrast has a specific, pre-defined hypothesis that goes with it, you bypass the need to adopt a Family-Wise Error Rate and the correction for multiple comparisons that comes with it[[16]](#footnote-16).

How will we create a POC for our experiment? We will try to create a table that includes an individual weight for each group inside of each contrast. Supposing that we had hypothesized that: (1) stress ratings in both the Direct and Indirect conditions will differ from those in the Handler-Only condition, and (2) stress ratings in the Direct contact condition will differ from those in the Indirect condition, we might construct the following table:

| Condition | Direct | Indirect | Handler-only |
| --- | --- | --- | --- |
| Contrast 1 | 1 | 1 | -2 |
| Contrast 2 | 1 | -1 | 0 |

This table is a set of weights for each of our contrasts and conditions that satisfies the following rules:

1. If a group is singled out in one contrast, that group should generally be excluded from any subsequent contrasts.
2. Groups coded with positive weights will be compared against groups coded with negative weights. So, assign one side positive weights and the opposite side negative weights.
3. The sum of weights for any given contrast must be zero.
4. If a group is not involved in a contrast, assign it a weight of 0. If we give a group a weight of 0 then this eliminates that group from all calculations for that contrast.
5. When you have a set of potential contrasts, multiply and sum each pair of contrast weights within each pair of contrasts. The sum of these products must be ZERO for a pair of potential contrasts to be considered orthogonal. It ensures that our comparisons are UNCORRELATED with each other, giving us the right to NOT correct for a family-wise error rate.

OK, so we have a set of weights. Next, let’s find the results of our Planned Orthogonal Contrasts using R. To do this procedure, we will need the function emmeans() from the package of the same name. So, run install.packages("emmeans") to get started.

emmeans() is short for “Estimated Marginal Means”, which are means that are based on a model (not directly on our data). That’s why they are “estimates” of our groups’ means (although they will usually be very, very close to the true group means). The word “marginal” refers to the fact that, if you lay out all our data on a table, the means we estimate are calculated along the margins. The emmeans() function is expecting to see a handful of arguments:

* “object =” This is a fitted model. Our ANOVA model using the aov() function will work for this purpose.
* “specs =” Is expecting to see a character vector (an object with one or more character-type entries) that specifies the names of the predictors for which estimated marginal means will be computed. In our case, this will be the name of our grouping variable.
* “contr =” is expecting to see a character value or list that explains what contrasts should be used for any comparisons being done. Later, we will show how entering “pairwise” for this argument provides comparisons for all the pairs of conditions in our object. For our POC, we will instead supply a list of contrasts with names and weights. This will be shown in detail in the example below.
* “adjust =” can be specified as “bonferroni” or “sidak” to correct cases where post-hoc pairwise comparisons are being conducted instead of POC.[[17]](#footnote-17)

# load the emmeans package  
library(emmeans)  
  
# In order to specify our contrasts, we need to know what order they are in.  
# You can check the object 'xlevels' inside our model "aovMethod" to find out  
aovMethod[["xlevels"]]

## $condition  
## [1] "Direct" "Handler Only" "Indirect"

# Next, set up a list of contrasts using the list() function.   
# Make sure the order of your weights matches the order of the groups in xlevels.  
planned\_contrasts <- list(  
 "Direct and Indirect vs. Handler Only" = c(1, -2, 1),  
 "Direct vs. Indirect" = c(1,0,-1))  
  
# Ask the emmeans() function for our Planned Orthogonal Contrasts  
# Run our planned contrasts  
emmeans(aovMethod,  
 specs = "condition",  
 contr = planned\_contrasts)

## $emmeans  
## condition emmean SE df lower.CL upper.CL  
## Direct 2.08 0.092 297 1.90 2.26  
## Handler Only 2.91 0.092 297 2.73 3.09  
## Indirect 2.38 0.092 297 2.20 2.56  
##   
## Confidence level used: 0.95   
##   
## $contrasts  
## contrast estimate SE df t.ratio p.value  
## Direct and Indirect vs. Handler Only -1.36 0.225 297 -6.038 <.0001  
## Direct vs. Indirect -0.30 0.130 297 -2.307 0.0217

Let’s read these results from the top down. First, you’ll notice that the order of the condition variable matches what we saw when we looked at the model’s ‘xlevels’ object. Next, you will notice a marginal mean (“emmean”) is estimated for each group condition. We are also given a standard error (SE), degrees of freedom (df), and confidence intervals (CL) for the estimate that was computed. We can see that a 95% confidence interval was computed by default. Looking at our contrasts, we can see the estimated difference between the two sides of our contrasts and, most importantly, we can see a *t* statistic, a df value, and a *p*-value associated with each contrast. Now, we can clarify our results further to say:

“Levene’s test indicated significant differences in group variances (*F*(2, 297) = 3.15, *p* = .04). A significant effect of study condition was observed for the corrected omnibus model (*F*(2, 195.13) = 21.04, *p* < .001). Planned Orthogonal Contrasts revealed statistically significant differences between the Direct and Indirect groups (*t*(297) = -2.31, *p* = .02), as well as between the Handler Only group and all others (*t*(297) = -6.04, *p* < .001).”

**Activity**

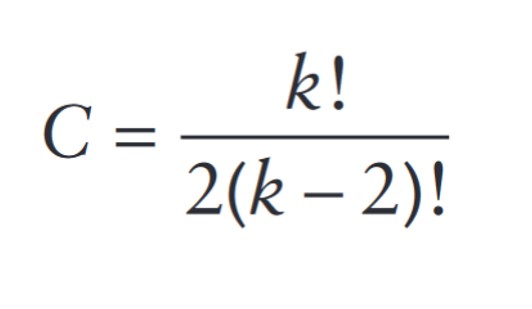
Imagine a randomized drug trial experiment with 5 independent groups: four experimental groups who were each given a different dosage of Drug X (25mg, 50mg, 100mg, and 150mg), and one control group. Create and insert a table of valid contrasts given the following rules:

* One contrast must compare all Drug X groups against the control group.
* One contrast must compare the 25mg and 50mg groups.
* There must be three total contrasts.

### Post hoc pairwise comparisons

Planned Orthogonal Contrasts are not always feasible. We may not always have the foresight to specify a set of POCs before running our omnibus ANOVA model. Sometimes, we may want to make multiple comparisons using a single isolated group. Whatever the reason, this is the situation where we might consider **post-hoc pairwise comparisons**. This refers to any situation where you decide to compare pairs of groups after your model has been fit. When this occurs, it is almost always a good idea to use a Bonferroni or Sidak correction based on the number of comparisons being made.

But be careful! The number of comparisons possible in any experiment follows the formula



**Figure 7.1. Formula for the number of possible pairwise comparisons.**

Where C is the total number of pairwise Comparisons, and k is the number of groups. For example, in a 5-group experiment, the number of pairwise comparisons is

C = 5 x 4 x 3 x 2 x 1 / 2(3 x 2 x 1) = 120 / 2(6) = 10

With this many comparisons, we can quickly find it very hard to reject the null hypothesis after using a Bonferroni or Sidak correction.

However, in a small experiment like this one, it is probably not going to do much harm to compute our pairwise comparisons using the code below. When we do this, the results can be reported using similar conventions as the POC example above (although it is a great idea to specify that you are reporting corrected pairwise comparisons).[[18]](#footnote-18)

# Use the emmeans() function to compute pairwise comparisons  
emmeans(aovMethod, specs = "condition", contr = "pairwise", adjust = "bonferroni")

## $emmeans  
## condition emmean SE df lower.CL upper.CL  
## Direct 2.08 0.092 297 1.90 2.26  
## Handler Only 2.91 0.092 297 2.73 3.09  
## Indirect 2.38 0.092 297 2.20 2.56  
##   
## Confidence level used: 0.95   
##   
## $contrasts  
## contrast estimate SE df t.ratio p.value  
## Direct - Handler Only -0.83 0.13 297 -6.383 <.0001  
## Direct - Indirect -0.30 0.13 297 -2.307 0.0652  
## Handler Only - Indirect 0.53 0.13 297 4.076 0.0002  
##   
## P value adjustment: bonferroni method for 3 tests

**Reflect**  
Is it generally better to use POC, or post-hoc pairwise comparisons? Why?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Is it always OK to report an estimated marginal mean as a true group mean? Why or why not?  
(“\_\_\_\_\_\_\_\_\_\_”)

## Effect sizes for ANOVA models

One final statistic to consider when reporting ANOVA results is *η*2 (“Eta squared”). This metric provides an indication of practical significance that is most comparable to R2. We can compute it by dividing the sum of squares between groups (35.33) by the total sum of squares in our ANOVA model (35.33+251.11). For example, in our aovMethod model, we saw:[[19]](#footnote-19)

summary(aovMethod)

## Df Sum Sq Mean Sq F value Pr(>F)   
## condition 2 35.33 17.663 20.89 3.24e-09 \*\*\*  
## Residuals 297 251.11 0.845   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

So, we could compute η2 as:

35.33 / (35.33+251.11)

## [1] 0.1233417

Using the summary.lm() function to get a regression-style summary of our aovMethod model, we notice an interesting thing about the Multiple R2 statistic…

# This is one way to find eta squared without hand-computing it.  
summary.lm(aovMethod)

##   
## Call:  
## aov(formula = stress ~ condition, data = BARKdata)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1.9100 -0.5125 -0.0800 0.6200 2.6200   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 2.08000 0.09195 22.621 < 2e-16 \*\*\*  
## conditionHandler Only 0.83000 0.13004 6.383 6.67e-10 \*\*\*  
## conditionIndirect 0.30000 0.13004 2.307 0.0217 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.9195 on 297 degrees of freedom  
## Multiple R-squared: 0.1233, Adjusted R-squared: 0.1174   
## F-statistic: 20.89 on 2 and 297 DF, p-value: 3.244e-09

It’s the same number! Isn’t that convenient? We can interpret *η*2 the same way we interpret *R*2! Finally, we can complete our APA-style results section by saying:

“Levene’s test indicated significant differences in group variances (*F*(2, 297) = 3.15, *p* = .04). A significant effect of study condition was observed for the corrected omnibus model (*F*(2, 195.13) = 21.04, *p* < .001, *η*2 = .12). Planned Orthogonal Contrasts revealed statistically significant differences between the Direct and Indirect groups (*t*(297) = -2.31, *p* = .02), as well as between the Handler Only group and all others (*t*(297) = -6.04, *p* < .001).”

**Activity**

Read the paper “Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and ANOVAs” by Daniel Lakens (2013; <https://www.frontiersin.org/articles/10.3389/fpsyg.2013.00863/full>). Use the space below to briefly explain the statistic *ω*2 (omega squared), and why it is an improved form of *η*2.

## Visualizing ANOVA models

Most often, when researchers display the results of an ANOVA model, they will do one of two things: (1) they will show a boxplot of each group’s data, or (2) they will show a bar graph of each group’s data (with standard error bars). The choice to show either comes down to your choice of tests[[20]](#footnote-20), and your own view of which plot will best represent your data.

Here is an example of a box plot for this Lesson’s study:

# First, make sure that ggplot2 is loaded.  
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.2.3

# Create the box plot.  
BARKboxplot <- ggplot(BARKdata, aes(x=condition, y = stress, group=condition))+  
 geom\_boxplot()  
# Call the plot's name to display it.  
BARKboxplot

A diagram of a graph

Description automatically generated with medium confidence

Notice how we have a single dot representing an outlier in our data. This could be considered an advantage of using a box plot to illustrate our results. Notice also how the median quartiles for each boxplot fall on integer values in this plot. This is a normal feature of box plots of integer variables that might be considered a disadvantage in this specific case.

A bar graph of the same data might look like this:

# Create the bar graph  
BARKbar <- ggplot(BARKdata, aes(x = condition, y = stress)) +   
 geom\_bar(stat = "summary\_bin", fun = mean) +   
 stat\_summary\_bin(aes(x = condition, y = stress), fun.data = mean\_se)  
  
# Call the plot's name to display it.  
BARKbar

A graph of a bar chart

Description automatically generated with medium confidence

In this case, we can more-clearly see where the group means fall. Using the standard error bars, we can also reasonably guess which pairwise analyses might be statistically significant before correcting for multiple comparisons. A good rule of thumb is: if there is no overlap between two groups’ standard error bars on the Y-axis, then it is very likely that these groups significantly differ. Obviously, this is no substitute for actually running your analysis, but it is useful in a variety of less serious situations (such as when reading bar graphs in published papers).

**Activity**  
Find a real research paper that shows either a box plot or a bar graph. Try to notice how the plot was formatted in that published work. How big was the text? What labels were used for the X and Y axes? Were the bars colorful, or uniform?

Now, make some edits to these plots using the information in Chapter 3 and anything you can find online to make the resulting plot more similar to the one you found in a real, published paper. Add both plots as images in the space below.

We have reached the end of another challenging lesson in R statistics! Before you go, make sure to write down your three major takeaways and review their key concepts when we come back next week.

**Key Takeaways**  
The most important lessons from this Lesson were:  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)

Lesson 8 - Factorial ANOVA

Last edited on: 2023-12-07

# Lesson 8 - Factorial ANOVA

Welcome! In all Lessons so far, we dealt with *univariate* (‘one variable’) and *bivariate* (‘two variable’) statistics. As we look to your future in psychological science, it is extremely likely that you will soon begin to develop hypotheses too complex to be answered using just one or two variables.

For example, you might ask whether a relationship between childrens’ ability to resist eating a marshmallow when left unsupervised depends on *both* their school grade level (where two levels may be defined as Kindergarten, and Grade 1), *and* their having been offered a second marshmallow if they can hold out long enough (where two levels would be defined as “offered” and “not offered”). This more-complex research question uses a *factorial* design. Here, we still have one dependent variable (how many minutes each child waited before eating the first marshmallow), but we now have two different independent variables: grade level, and whether or not a second marshmallow was promised.

When two factors are in play, a variety of unique scenarios could unfold regarding our dependent variable, including:

* we might find that Grade 1-aged children wait longer than Kindergarten-aged children(or vice versa), but no effect of a second marshmallow.
* we might find that offering a second marshmallow affects childrens’ resistance to the temptation of the first marshmallow, but that this is unaffected by age group.
* we might find that neither variable impacts wait times in any way.
* or, we might find that wait times depend on a second marshmallow differently among the age groups

As we begin thinking about factorial ANOVA, we will see that our independent variables can exert *main effects* that occur irrespective of any other variables measured, and *interaction effects* wherein the effects of one independent variable are dependent upon another independent variable.

### Our objectives this lesson:

* Compute power for factorial ANOVA
* Learn the assumptions of factorial ANOVA models
* Compute factorial ANOVA
* Test comparisons in factorial ANOVA models
* Learn about effect sizes for factorial ANOVA
* Plot and report the results of factorial ANOVA

## Consider this dataset

The study described here is an example of a follow-up experiment to the Canine-Assisted Intervention (CAI) experiment described in the previous lesson. It is based on a study conducted by Zakary Draper and colleagues during the immediate response to the emergence of COVID-19 and associated public health measures which sought to limit close contacts between people.

“As university courses moved online during the COVID-19 pandemic, opportunities for in-person group Canine-Assisted Interventions (CAIs) were limited. The Building Academic Retention through K9s (B.A.R.K.) team thought here may be value in providing online CAIs. The current study investigates the benefits of participating in an online CAI. As with in-person CAIs, students can join their peers for brief social interactions with a trained dog handler and dog. Online CAIs also provide the opportunity for on-demand sessions. Students could watch a pre-recorded video with content similar to a synchronous CAI session. Although asynchronous sessions lack the benefit of live communication with peers, they have the benefit of being available on-demand, and require less resources in the long run.

Like with in-person CAIs, the dog is complementary to the therapy. Research has shown that the dogs are important to in-person CAIs (Binfet et al., 2022); however, the same cannot be said for online CAIs. Without the opportunity for physical contact, the dogs may be unimportant to online interventions. Given the time and money required to train dogs and handlers, it is important that their benefit be justified.

This study took place entirely online. Undergraduate students completed a brief online survey, which asked about their stress levels both before, and after the session. The therapy session involved encouraging students to reflect on their experience at university in a positive way. There were two independent variables pertaining to the content and delivery of the intervention: participants were randomly assigned to a synchronous or asynchronous session, where a B.A.R.K. dog was – or was not – visible on camera. Synchronous sessions occurred via Zoom with 2–4 participants in each session. Asynchronous sessions were pre-recorded videos with similar content, and were viewed on YouTube. In all conditions, the intervention lasted approximately 5 minutes.

Because the intervention is so short, it is necessary to keep the survey similarly brief. As such, stress will be measured using with a single item (“how stressed do you feel right now?”) with response options ranging from 1 (Not at all stressed) to 5 (Very stressed). 10.2.2.3 Analytic Strategy

The hypotheses will be tested using a 2 (asynchronous vs. synchronous) x 2 (dog present vs. absent) analysis of variance predicting the difference between stress at pre- and post-intervention.

* Hypothesis 1 will be considered supported if the main effect of platform (asynchronous vs. synchronous) is significant in the model.
* Hypothesis 2 will be considered supported if the main effect of the dog (present vs. absent) is significant in the model.
* Hypothesis 3 will be considered supported if the interaction term (between platform and dog) is significant, and if follow-up analyses indicate that this interaction is as hypothesized (i.e., that the presence of the dog is primarily of important for synchronous delivery).

This code will clear out our R environment and load the new dataset I just described.

# Clean out the environment  
rm(list=ls())  
  
# Load the dog data as a new object we will call "factorialData"  
factorialData <- read.csv("dog\_data.csv")  
  
# Show the first few rows of data  
head(factorialData)

## pid platform dog pre post  
## 1 1 async no dog 2 1  
## 2 2 sync no dog 3 3  
## 3 3 sync dog 4 4  
## 4 4 async no dog 3 2  
## 5 5 sync dog 3 2  
## 6 6 sync no dog 2 2

Looking at the data, we see five variables:

* pid, the participant identification number.
* platform, a factor that indicates whether a participant was assigned to the synchronous (sync) or asynchronous (async) content delivery condition.
* dog, a factor that indicates whether a dog was visible on camera during each participant’s session.
* pre, a numeric variable indicating pre-session stress levels, and
* post, a numeric variable indicating post-session stress levels.

You may have noticed that we do not yet have a variable indicating whether self-reported stress levels increased or decreased as a result of the intervention. Let’s compute that variable now, and call it stressChange.

# Compute our dependent variable  
factorialData$stressChange <- factorialData$post - factorialData$pre  
  
# Print some basic descriptive statistics for it  
quantile(factorialData$stressChange)

## 0% 25% 50% 75% 100%   
## -3 -2 -1 -1 1

Our dependent variable, stressChange was computed as the change in stress between the post- and pre-session ratings. It was calculated this way so that increases in stress are recorded as positive numbers, decreases in stress are coded as negative numbers, and differences between individuals’ baseline (pre-intervention) stress levels are mathematically removed from the analyses we plan to do. Looking at some quantiles for this variable, we can see that the maximum recorded value was an increase of 1 out of 5 stress rating points, while the minimum value indicated a decrease of 3 stress points and the median participant reported a decrease of 1 stress point. This is beginning to look like stress ratings may have decreased as a result of the intervention, but was this due to either the platform or dog variable?

Let’s do a factorial ANOVA to find out! But first, let’s learn more about our dataset; do we have adequate statistical power for this kind of model, and are this model’s assumptions met?

## Power and factorial ANOVA

Do we have enough statistical power to lend some credibility to our statistical approach even if we do not find any statistically significant effects? Remember that our alpha level is set (typically at .05) such that we allow ourselves to make incorrect conclusions (Type I errors) a certain percentage of the time when the null hypothesis is true. The quantity beta is a related concept that dictates the proportion of incorrect conclusions (Type II errors) we will make when the null hypothesis is false. Recall also that the inverse of beta, (1 - beta) is called power, and it refers to the proportion of correct conclusions we will make when the null hypothesis is false. However, you will recall from Lesson 5 that we do not have the ability to dictate precisely how much power an experiment will have. Rather, we typically determine (*a priori*) the sample size of our experiments based on the power level we would like to achieve (typically 80% at minimum) given the effect size we reasonably expect we might observe.

Let’s do that here. We will use the function pwr.anova.test() to determine what sample size would be required to achieve 80% power to detect real main effects (i.e., an effect of either dog or platform on stressChange) given an ANOVA model with:

* an alpha level equal to .05,
* two independent variables (this will be coded using the variable k), and
* an medium effect size of f = .025.[[21]](#footnote-21)

# Load the pwr package  
library(pwr)  
  
# Determine required sample size for adequate power to detect medium-sized main effects  
powerF <- pwr.anova.test(  
 n = NULL,  
 k = 2,  
 f = 0.25,  
 sig.level = .05,  
 power = .80)  
  
# Print our results to the console.  
powerF

##   
## Balanced one-way analysis of variance power calculation   
##   
## k = 2  
## n = 63.76561  
## f = 0.25  
## sig.level = 0.05  
## power = 0.8  
##   
## NOTE: n is number in each group

Our results indicate that approximately 64 participants are needed in each group to achieve adequate power. Since we have two groups in both our dog and platform variables, and since we have 128 participants in this dataset, and since 128 divided by 2 equals 64, it appears that we have adequate power to test each of our two factors and the interaction between them. Given that a different hypothesis is tested for the main effect of dog, the main effect of platform, and the dog by platform interaction, we do not need to adjust our alpha level to account for multiple comparisons, and can proceed to the next step of this analysis.

**Reflect**  
What could have been a consequence if we found this study did not have adequate power for the analysis we have planned?  
(“\_\_\_\_\_\_\_\_\_\_”)  
Imagine that this study was found to have 50% power, and the results we obtained were not statistically significant. Can we trust this result? Why or why not?  
(“\_\_\_\_\_\_\_\_\_\_”)

## Factorial ANOVA assumptions

As in our Lesson 7 between-groups ANOVA models, we will need to check three key assumptions before we run our model here:

### Independence

As always, the assumption of independence is not something we check using a statistical procedure. Rather, we consider any ways that our participants might be related to each other that are not accounted-for by our model. In this case, we might have a slight problem with the fact that groups of participants were tested together in clusters of two to four people. This could be accounted for in our model by including a third factor that identifies specific participant groups, but this new variable would have between 32 and 64 groups inside it, and we would need to have sampled hundreds more participants to achieve adequate power for that factor. When an issue like this is present in our data, we scientists often feel obligated to highlight it as a limitation to our findings in the paper we write about the experiment.

### Normality

The assumption of normality will be handled in precisely the same manner as we saw in the previous lesson on ANOVA. Once again, we will use the plot() function alongside our aov() object for our ANOVA model in order to generate a QQ plot. If the data appear non-normal, or the sample size is smaller than about N = 30, we will choose to perform a Kruskal-Wallis test instead of an ordinary Analysis of Variance.

# create the ANOVA model for this analysis (see below for detailed explanation)  
aovModel <- aov(stressChange ~ platform \* dog, data = factorialData)  
# generate a QQ plot  
plot(aovModel, which = 2)

A graph of a normal q-q

Description automatically generated

This plot show some evidence of non-normality. Our data points do not fall along the diagonal line that defines expected quantiles given a normal distribution. This is especially the case for lower scores on our dependent variable. Although our larger sample size means that this likely will not create huge problems for our ANOVA model, it may be worth considering a non-parametric alternative.

### Homogeneity of Variance (HoV)

To test whether the different groups of our dog and platform variables have different variances, we will need to do a Levene’s test that captures both variables. Placing an asterisk symbol (\*) in between two independent variables in a *regression-style* formula (see Lesson 5) tells R that we are interested in *crossing* the two variables. In other words, we want to look at the first variable, and the second variable, and the interaction between them. As always, we will use a more-lenient alpha level equal to .10, and consider violations of HoV to be potentially-impactful on our ability to draw conclusions about the null hypothesis when group sizes differ and Levene’s test is statistically significant.

# Do group sizes differ? Yes.  
table(factorialData$dog,factorialData$platform)

##   
## async sync  
## dog 43 28  
## no dog 32 25

# Test for violations of HoV (we will need the car package for this)  
library(car)

## Loading required package: carData

leveneTest(stressChange ~ platform \* dog, data = factorialData)

## Levene's Test for Homogeneity of Variance (center = median)  
## Df F value Pr(>F)  
## group 3 1.6293 0.186  
## 124

Although we noticed clear differences in the sizes of our groups, we did not find that Levene’s test was significant. We can thus proceed with an ordinary factorial ANOVA analysis.

**Explain**  
When I want to determine if the independence assumption is met, I will…  
(“\_\_\_\_\_\_\_\_\_\_”)  
When I want to determine if the normality assumption is met, I will…  
(“\_\_\_\_\_\_\_\_\_\_”)  
When I want to determine if the homogeneity of variance assumption is met, I will…  
(“\_\_\_\_\_\_\_\_\_\_”)

## Compute factorial ANOVA

The code for testing factorial ANOVA models is extremely similar to the code for testing one-way ANOVA models. We have simply added a \* to indicate that we wish to see the main effects and interaction effects for the two factors (dog and platform) we specified. This is the same as if we told R we wanted to look at the main effect of dog, the main effect of platform, and the interaction between the two (in terms of our code, this would could also be written as like platform + dog + platform:dog)

# Create an object called aovModel that is our ANOVA model  
aovModel <- aov(stressChange ~ platform \* dog, data = factorialData)  
  
# Print a summary of the model to the console.  
summary(aovModel)

## Df Sum Sq Mean Sq F value Pr(>F)   
## platform 1 0.72 0.7170 1.178 0.2799   
## dog 1 0.02 0.0246 0.040 0.8410   
## platform:dog 1 2.64 2.6359 4.329 0.0395 \*  
## Residuals 124 75.50 0.6089   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

These results can be read in the same manner we handled ANOVA results in the previous Lesson. However, one key difference concerns our interpretation of the results: we now have both main effects and interactions to contend with. Looking at the row for the platform factor, we see that there was no statistically significant main effect. It did not matter whether content delivery was synchronous or asynchronous across all levels of the dog variable. Similarly, there was no significant main effect of dog; it didn’t matter whether a dog was present or absent across all levels of the platform variable. However, we see a significant platform by dog interaction, indicating that stress change ratings significantly differed for at least one combination of groups compared to at least one other combination of groups.

At this point, you may be wondering whether this model, like other models we have seen in this manual, could have been tested ‘regression-style’ using the lm() function. The answer to this question is an emphatic *yes*, but there are some important caveats that should be considered in depth. If this is a question you care about, I happily direct you to 16.6.7 in Dr. Danielle Navarro’s exceptional textbook “Learning Statistics with R” (<https://kpu.pressbooks.pub/learningstatistics/>).

## Group comparisons in factorial ANOVA

Pairwise comparisons work very much the same way for one-way and factorial ANOVA models. Using a similar code to what we saw in Lesson 7, we can obtain the following post-hoc pairwise comparisons:

# Make sure emmeans is loaded.  
library(emmeans)  
  
# Compute estimated marginal means  
emmeans(aovModel, specs = c("platform", "dog"),   
 contr = "pairwise",   
 adjust = "bonferroni")

## $emmeans  
## platform dog emmean SE df lower.CL upper.CL  
## async dog -1.186 0.119 124 -1.42 -0.951  
## sync dog -1.071 0.147 124 -1.36 -0.780  
## async no dog -0.969 0.138 124 -1.24 -0.696  
## sync no dog -1.440 0.156 124 -1.75 -1.131  
##   
## Confidence level used: 0.95   
##   
## $contrasts  
## contrast estimate SE df t.ratio p.value  
## async dog - sync dog -0.115 0.189 124 -0.605 1.0000  
## async dog - async no dog -0.217 0.182 124 -1.193 1.0000  
## async dog - sync no dog 0.254 0.196 124 1.294 1.0000  
## sync dog - async no dog -0.103 0.202 124 -0.509 1.0000  
## sync dog - sync no dog 0.369 0.215 124 1.717 0.5313  
## async no dog - sync no dog 0.471 0.208 124 2.263 0.1524  
##   
## P value adjustment: bonferroni method for 6 tests

Notice how none of these contrasts is statistically significant, despite the significant interaction effect we found earlier. This could be due to adjustments made when we corrected these comparisons using the Bonferroni method. As always, the more-powerful approach is to pre-specify a set of pairwise comparisons. For example, supposing that we wanted to test Hypothesis 3 from this study (that the presence of the dog is primarily of important for synchronous delivery) using a planned orthogonal contrast, we might run the following set of orthogonal contrasts based on the order of conditions we see in the output above:

emmeans(aovModel, specs = c("platform", "dog"),   
 contr = list("sync dog vs. all" = c(-1, 3, -1, -1),  
 "async dog vs. no dog" = c(2, 0, -1, -1),  
"async no dog vs. sync no dog" = c(0, 0, 1, -1)))

## $emmeans  
## platform dog emmean SE df lower.CL upper.CL  
## async dog -1.186 0.119 124 -1.42 -0.951  
## sync dog -1.071 0.147 124 -1.36 -0.780  
## async no dog -0.969 0.138 124 -1.24 -0.696  
## sync no dog -1.440 0.156 124 -1.75 -1.131  
##   
## Confidence level used: 0.95   
##   
## $contrasts  
## contrast estimate SE df t.ratio p.value  
## sync dog vs. all 0.3805 0.503 124 0.756 0.4510  
## async dog vs. no dog 0.0367 0.316 124 0.116 0.9079  
## async no dog vs. sync no dog 0.4713 0.208 124 2.263 0.0254

Interestingly, we do not see a statistically significant effect here either. This leaves us in the somewhat mysterious predicament of not knowing which group(s) was/were responsible for the significant interaction effect we observed. We may, however, still get some clue as to the culprit(s) when we visualize these results later on in the lesson.

Finally, let’s compute an effect size for the *t*-statistic in this contrast, as we will need it later on. We will use the same method used with other *t* statistics in Lesson 5.

# Load the psych package  
library(psych)

##   
## Attaching package: 'psych'

## The following object is masked from 'package:car':  
##   
## logit

# Compute an effect size  
t2d(0.756, n1 = 28, n2 = 100)

## [1] 0.1616396

**Activity**  
Imagine a test a new set of three Planned Orthogonal Contrasts using these data. Make sure to check the rules for POC in Lesson 7 to ensure that your contrasts are truly orthogonal.

# Make sure emmeans is loaded.  
  
# Compute estimated marginal means for your groups  
  
# Define a list of Planned Orthogonal Contrasts  
  
# Test the Planned Orthogonal Contrasts

## Effect sizes and factorial ANOVA

Let’s calculate some effect sizes for this model. This code is rather simple, and uses the effectsize() function from the effectsize package. You can download this package using the code install.packages("effectsize").

# load the effectsize package  
library(effectsize)  
  
# Give the ANOVA mode -- an 'aov'-type object as an argument to get partial eta squares.  
effectsize(aovModel)

## # Effect Size for ANOVA (Type I)  
##   
## Parameter | Eta2 (partial) | 95% CI  
## --------------------------------------------  
## platform | 9.41e-03 | [0.00, 1.00]  
## dog | 3.26e-04 | [0.00, 1.00]  
## platform:dog | 0.03 | [0.00, 1.00]  
##   
## - One-sided CIs: upper bound fixed at [1.00].

The interpretation of these effect sizes remains the same as with one-way ANOVA models. These values would all typically be considered quite small. Recall that Partial eta squared statistics equal to .01, .06, and .14 correspond to small, medium, and large effect sizes, respectively. As such, we can conclude that the main effect of platform had a very small effect size, the main effect of dog had a very small effect size, and the interaction between them had an effect that was a little larger, but still quite small.

## Visualizing factorial ANOVA models

The most typical way factorial ANOVA models are visualized is with a line graph that has standard error bars. This works well whenever your independent variables are two factors that each have a reasonably small number of levels. A good rule of thumb if your factors have different numbers of levels is to place the factor with more levels on the X axis and plot individual lines for each level of the other factor.

Although line graphs are usually reserved for continuously-scaled variables, their use with factorial designs makes it particularly easy to see main effects and interactions between factors. Whenever multiple variables or levels are included on a graph, it is also a good idea to distinguish them using different shapes or colours, and to ‘dodge’ their positioning slightly so that error bars can be easily associated with the data points to which they refer.

Below, I have included an example plot for our data. First, we will need to create an object that contains group means and 95% confidence intervals. As we saw in the last lesson, the emmeans package is well suited to reporting this information.

# Create an object with estimated marginal means to be used for plotting.  
emm <- emmeans(aovModel, specs = c("platform", "dog"))  
  
# Convert the marginal means object to a data frame so that ggplot knows how to use it.  
emm\_df <- as.data.frame(emm)  
  
# Show what the emm\_df object looks like  
emm\_df

## platform dog emmean SE df lower.CL upper.CL  
## async dog -1.186 0.119 124 -1.42 -0.951  
## sync dog -1.071 0.147 124 -1.36 -0.780  
## async no dog -0.969 0.138 124 -1.24 -0.696  
## sync no dog -1.440 0.156 124 -1.75 -1.131  
##   
## Confidence level used: 0.95

Next, let’s make our plot where the X axis represents our variable platform, the Y axis represents scores on our variable stressChange, and each level of the variable dog is given a different line. I have used a couple of new arguments here. The aesthetics ymin and ymax define the points on the graph where lower and upper confidence intervals should be placed. The argument size can be given to the geom\_point() function to increase or decrease the size of points on the plot. Larger numbers specify larger points. The aesthetics shape and colour can be given to the labs()function in order to change the shape and colour given to geom\_point(). Color is also given as an aesthetic to geom\_point() to make this code function properly. For all geoms, the function position\_dodge() can be given a proportion value that identifies the amount of space by which the geom should dodge each other.

# Make sure ggplot2 is loaded.  
library(ggplot2)

## Warning: package 'ggplot2' was built under R version 4.2.3

##   
## Attaching package: 'ggplot2'

## The following objects are masked from 'package:psych':  
##   
## %+%, alpha

# Create the factorial effects plot.  
anovaPlot <- ggplot(emm\_df, aes(x = platform, y = emmean, shape = dog, ymin = lower.CL, ymax = upper.CL)) +  
 geom\_line(aes(group = dog), position = position\_dodge(.25)) +  
 geom\_errorbar(position = position\_dodge(.25)) +   
 geom\_point(aes(colour = dog), position = position\_dodge(.25), size = 3.5) +  
 labs(x = "Platform", y = "Change in Stress", shape = "Dog", colour = "Dog")  
  
# Print the plot  
anovaPlot

A diagram of a dog and a dog

Description automatically generated

With this plot, we can clearly see that the largest decreases in stress occurred among participants exposed to a synchronous session with no dog visible. Finally, we may summarize our results using APA format by saying something like:

A 2x2 (platform by dog presence) between-subjects factorial ANOVA found that there was no significant main effect of platform (Hypothesis 1; *F*(1,124) = 1.18, *p* = .28, *η*2 < .01 ) nor dog presence (Hypothesis 2; *F*(1,124) = 0.04, *p* = .84, *η*2 < .01), but there was a significant platform-by-dog presence interaction effect (*F*(1,124) = 4.33, *p* = .04, *η*2 =.03). A planned orthogonal contrast defined to test the prediction that the presence of a dog would be distinctly important in the synchronous condition (Hypothesis 3) did not yield statistically significant results (*t*(124) = 0.76, *p* = .45, *d* = 0.16).

We have reached the end of another challenging lesson in R statistics! Before you go, make sure to write down your three major takeaways and review their key concepts when we come back next week.

**Key Takeaways**  
The most important lessons from this Lesson were:  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)  
(“\_\_\_\_\_\_\_\_\_\_”)

1. Many students get in the habit of loading every package we ever use at the beginning of their R scripts. Please do not do this. Very often, you will want to share your R scripts with other people so they can reproduce your analyses. Other people may not want to install and load all your R packages as this can hog their computers’ memory and storage space, and might create problems for their own analyses. [↑](#footnote-ref-1)
2. Whenever we do not write a required argument’s name inside a function, the first value we give will be passed to the first unnamed argument as defined in the help documentation. Then, the second value we give will be passed to the second unnamed argument, and so on. [↑](#footnote-ref-2)
3. You can also remove variables by adding a minus sign - in front of their names. Think of this as a way to ‘de-select’ or remove variables from an object. [↑](#footnote-ref-3)
4. Note: In this example, you will see a message that geom\_smooth() is using the formula ‘y ~ x’. This is not an error, or even a warning. It’s just to let you know how the line of best fit was determined. We will look at this ‘y ~ x’ language in depth in our Lesson on regression. [↑](#footnote-ref-4)
5. Notice that we have a warning for this bit of code. Here, geom\_histogram() lets us know that it has chosen to ignore some arguments that make sense for continuously-scaled x-axis variables. [↑](#footnote-ref-5)
6. Note: R wants the probabilities to be entered in the same order as the categories first appear in the table. [↑](#footnote-ref-6)
7. <https://www.amazon.ca/Understanding-Statistics-Behavioral-Sciences-Robert/dp/1111839204> [↑](#footnote-ref-7)
8. Collecting additional participants to achieve higher power after looking at your results is called **p-hacking**, and doing it will bring a swift end to your career in science. [↑](#footnote-ref-8)
9. We will look at the different kinds of R-squared in the next lesson on Regression. [↑](#footnote-ref-9)
10. Kim & Park, 2019 [↑](#footnote-ref-10)
11. Knief & Forstmeier, 2021 [↑](#footnote-ref-11)
12. Note: There is currently some disagreement about whether one should report the test results of a Levene’s test alongside each reported *t*-test. I am of the opinion that this is not very important because reporting fractional degrees of freedom clearly indicates that an HOV correction has been performed. Your view may differ, and that is perfectly valid. [↑](#footnote-ref-12)
13. The next Lesson on one-way ANOVA provides a detailed explanation and tutorial. [↑](#footnote-ref-13)
14. When we used the factor() function, we allowed R to create a set of “dummy variables”. These are variables that have the value 1 when a particular experiment condition is true, and 0 when it is false. So our variable “condition” with its three categories was used to generate only 2 dummy variables: one that identifies rows of data in the Handler Only condition, and one that identifies rows of data in the Indirect condition. By looking at these two variables side-by-side, we could see that (1,0) corresponds to the Handler Only condition, (0,1) corresponds to the Indirect condition, (0,0) corresponds to the Direct condition, and (1,1) never occurs. Dummy variables will always be arranged this way, with one fever dummy variable than the number of variable conditions. [↑](#footnote-ref-14)
15. Although, a non-parametric approach could be used with these data and the lm() function by wrapping our outcome (Y) variable with the rank() function. This is called a Kruskal-Wallis test. [↑](#footnote-ref-15)
16. This is only true if the Rule 5 below is satisfied. [↑](#footnote-ref-16)
17. Note: Computation gets more complicated when we need to account for violations of HOV. For the sake of this simple example, we will use the model aovMethod, which has not been corrected for HOV violations. [↑](#footnote-ref-17)
18. Note: Computation gets more complicated when we need to account for violations of HOV. For the sake of this simple example, we will use the model aovMethod, which has not been corrected for HOV violations. [↑](#footnote-ref-18)
19. Note: Computation gets more complicated when we need to account for violations of HOV. For the sake of this simple example, we will use the model aovMethod, which has not been corrected for HOV violations. [↑](#footnote-ref-19)
20. For example, a common alternative to ANOVA models and HoV tests called the Brown-Forsythe procedure relies on sample medians instead of means. When this test is used, it might be a good idea to display a box plot which also shows sample medians. [↑](#footnote-ref-20)
21. Note: This statistic is called “Cohen’s f”. It is a measure of effect size quit similar to Cohen’s d. However, Cohen (1988) suggests that his f statistic should be interpreted such that f values of 0.1, 0.25, and 0.4 represent small, medium, and large effects respectively. [↑](#footnote-ref-21)