PSYC208 R Companion Website

Huey Woon LEE

2026-07-30

Contents

About This Website

This is the companion website for my PSYC208 class. It is written as a gentle introduction to R. So, it is most certainly not exhaustive or comprehensive. If you're interested in furthering your skills in R, there are plenty of resources available online which you can check out here, here or here. But it's okay if you don't wish to check them out; you should be able to get by with the information in this website.

If you need, here is the PDF version of this website: PSYC208.pdf.

Note. This website is developed linearly. What this means is that the later sections build on the earlier sections. For example, earlier sections may instruct you to install a particular package. Later sections will assume you have that package installed and make no mention of it. Because they are not stand-alone sections, if you jump to later sections without going through the earlier sections, you might get lost. :)

Chapter 1

Getting Started

To analyse data in R, you need both R and RStudio. These are *different* programmes. Think of R as the software that executes our commands and RStudio as the interface between us and R that makes interacting with R a much more pleasant experience. If that doesn't make sense, here's a (hopefully relatable) example.

Imagine you went to a hawker centre and there's an uncle manning the store. This uncle can only speak Hokkien but you can't. So, you tell your friend, who can speak Hokkien, what you want. Then, your friend translates that for the uncle, who then serves up a delicious meal for you. Think of the uncle in this example as R and your friend as RStudio. While you can try to "converse" with R directly, it's much easier to do so through RStudio. So… For the sake of your sanity, I strongly recommend you download both programmes!

1.1 Install R and RStudio

1.1.1 Install R

For Windows

1. Open an internet browser and go to https://cloud.r-project.org/.
2. Under **Download and Install R**, click on **Download R for Windows**. Click on **base**.
3. **Download the latest release** by saving the .exe file on your computer. **Double-click the file and follow the installation instructions**.

For macOS

1. Open an internet browser and go to https://cloud.r-project.org/.
2. Under **Download and Install R**, click on **Download R for macOS**.
3. **Download the latest release** by saving the .pkg file to your computer. **Double-click the file and follow the installation instructions**.

1.1.2 Install RStudio

Now that R is installed, download and install RStudio.

1. Go to https://docs.posit.co/ide/user/#rstudio-ide-oss-downloads.
2. Click the **download link for your operating system** (e.g., Windows or macOS).
3. Save the file. **Double-click the file and follow the installation instructions**.

1.2 The RStudio Interface

Now, start up RStudio. You should see something like this:

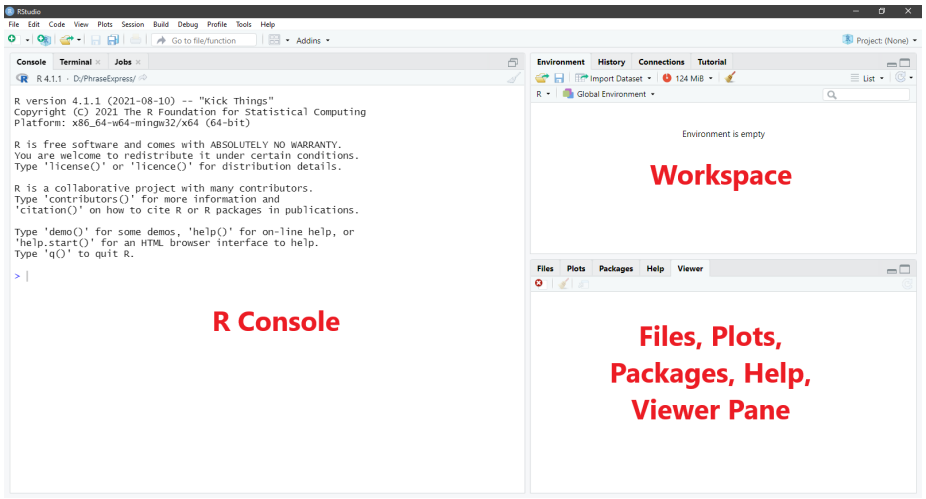


Figure 1.1: R Studio Interface With Three Panes

When you first start up RStudio, you will see three panes: the R console, the workspace, and the files, plots, packages, help, and viewer pane. Each pane serves different purposes.

1. **R console**: The R console is where commands are submitted to R for R to execute. It is also where we find some of the output from R (e.g., analysis results).

2. **Workspace**: I think of this as R's short-term memory. There are two tabs that are particularly useful.
    - **Environment tab**: We can find the list of objects (e.g., variables, data frames, functions) that we created in the session here.
    - **History tab**: Here is where we can find all the previous commands we submitted to R in the session.
3. **Files, plots, packages, help, and viewer**:
    - **Files**: We can create new folders on our computer, move, delete, and rename files here.
    - **Plots**: We can find all the plots we instructed R to produce during the session here.
    - **Packages**: We can find, install, and update packages here. Packages contain data, functions, help menus, etc. that other people have created to supplement those in R. We will talk more about specific packages later.
    - **Help**: We can find information about a given command or package. We can also find more information about various commands and the packages on this website: https://www.rdocumentation.org/

Note. Because the Terminal tab, the Connections tab, and the Viewer tab will not be used in this course, I will not talk about them.

1.3 R Script

To get R to do stuff (e.g., conduct analyses), we submit commands to R through RStudio. Although we can type the commands directly into the console, R users prefer to type the commands into what is called the script editor because we can save the commands in the script editor into script files (with the extension .R). The script files allow us to keep long-term records of the analyses that we have conducted. We can also share the script files with other R users so that they can reproduce our analyses. (In this class, I will use the words command and code interchangeably.)

To open a blank R script, go to **File > New File > R Script**. Or, you can use the shortcut Ctrl + Shift + N (Windows) or Cmd + Shift + N (macOS). Notice that now, your RStudio has four panes. The script editor should now take up the top half of the left hand side of the screen as shown below.

1.4 Some Tips and Tricks

Before we start coding proper, here are some tips to help you along your R journey. (These are things I wish I knew when I first started out!)

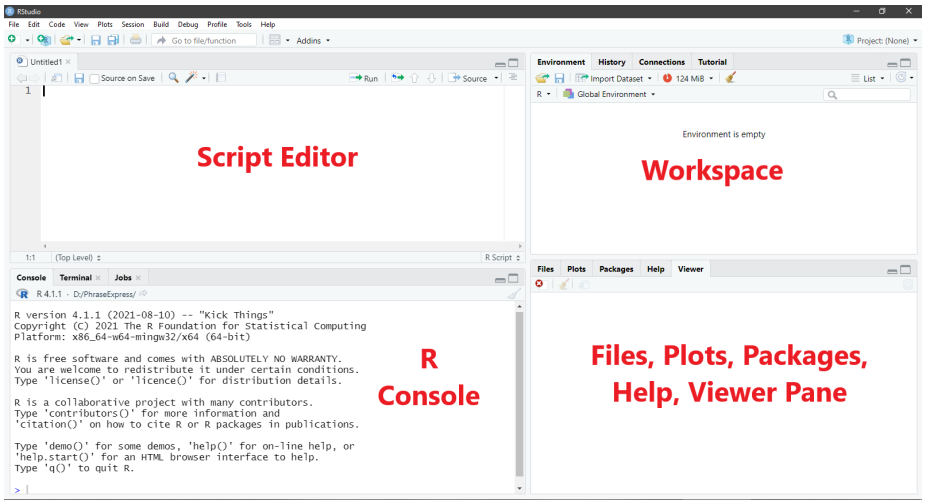


Figure 1.2: R Studio With Four Panes

1.4.1 Softwrap Long Lines

Sometimes, we might write commands in the Script editor section that are too long (horizontally) to fit the window. To see the entire command, we might need to scroll left and right. This can be frustrating. (It's like Notepad, without word wrap.) Fortunately, we can wrap the text such that the code fits into the size of the window. Go to **Code > Soft Wrap Long Lines**. I highly recommend you do this, especially if you tend to write a lot of comments in the script file like I do.

1.4.2 Make Notes or Comments

In R (and most programming languages), you can write notes or comments in the script to yourself and your readers. This is done in R by starting the line with a **#** sign.

```
# This is a comment.
```

Please make *liberal* use of comments. I cannot tell you how many times comments have helped me understand what I'm doing and why. Your future self will thank you. Trust me.

1.4.3 Multi-Line Comments

To make your super long comments readable, you may break them into several lines, starting each line with `#>`.

```
#> This is a comment
#> that has been broken
#> into multiple
#> lines.
#> :)
```

1.4.4 Create Code Sections in R Script

Often times when we are analysing data, we need to include multiple steps. It can feel incredibly confusing and overwhelming to navigate a long R script. We can use code sections to organise and structure the code, by grouping related tasks together. This way the R script is easier to navigate.

To insert a new code section, start the line with # and then use any of the following: *at least* four trailing dashes (-), equal signs (=), or pound signs (#).

```
# Section One ------------------------------
# Section One ----

# Section Two ==============================
# Section Two ====

### Section Three ############################
### Section Three ####
```

To fold all code sections (i.e., hide the content within all sections, leaving only the section header for navigation), go to **Edit > Folding > Collapse All** (the shortcut is `Alt + O`). To expand all code sections (i.e., show all content), go to **Edit > Folding > Expand All** (the shortcut is `Alt + Shift + O`).

1.4.5 Setting Working Directory

Typically when we analyse data, we need to reference external files (e.g., our data files). To tell R where to look for those data files, we need to specify the full file path (i.e., the file location). While this is fine if you only have one or two things to reference, it can be kind of tedious to keep typing the file path if you have many things to reference. Furthermore, if you choose to change your file location, it would be quite a hassle (and also error-prone) to have to update all those file paths in the script.

What we can do instead is to set a working directory in R using the function `setwd()`. This tells R where your data files are stored for the session, so it will know to look there. It will also be the place that R saves any output (e.g., plots).

To get the file location in Windows, we first go to the folder where the file is located, right-click on the address bar, and click **Copy address as text**. We then need to convert the backslashes in the file path to forward slashes before we can use it. Let's say the file path is `C:\Users\Win10\Desktop\R`. After converting all the backslashes to forward slashes, the file path to use is `C:/Users/Win10/Desktop/R`.

In macOS, there are several ways to get the file path. For instructions, please visit this website: https://www.dev2qa.com/how-to-get-file-path-in-mac/. Note that the file paths in macOS already use forward slashes, so changing backslashes to forward slashes is not an issue for macOS users.

After getting the file path, you can then set the working directory as follows, with the file path encased in open/close inverted commas, within the parentheses.

```
# Set working directory
setwd("C:/Users/Win10/Desktop/R")
```

Another (perhaps easier) way to set working directory is to go to **Session > Set Working Directory > Choose Directory**. Then, select the folder you want to set as your working directory. This process needs to be repeated for each session (i.e., each time you start up RStudio).

If you want to set a default working directory, go to **Tools > Global Options > General > Default working directory (when not in a project) > Browse**. Then, select the folder you want to set as your default working directory.

1.4.6 Using RStudio Projects

While setting the working directory manually is sufficient when we have only one or two projects, many of us have multiple projects on-going at the same time. If we have a bunch of different files from different projects all strewn in a single directory (folder), it can get quite messy. To stay organised, it is preferable that we create an RStudio project for each project we are working on. This allows us to group the files related to a single project (e.g., data, output) together.

To do this, go to **File > New Project**. A dialogue window with three options, "New Directory", "Existing Directory" and "Version Control", will appear. From here, you may choose either "New Directory" or "Existing Directory". If you chose "New Directory", R will create a new folder and create an R

project within that new folder. If you chose "Existing Directory", R will create an RStudio project within an existing folder that you specify. Either will work, but I usually select "Existing Directory" as I would already have created my own folder to house materials related to a specific project.

After RStudio has created the project, it will change the working directory to the project directory so that you can access all the files (e.g., data, script) related to this project in this directory. RStudio will also create a file with the extension .Rproj in the project directory. When you open this file, RStudio will automatically start a new session with the project directory as your working directory.

While it is not necessary to use RStudio projects in my classes, I recommend it because it will help keep you organized.

1.4.7 Compile Report

The "Compile Report" feature in RStudio generates a formatted document from your R script. It integrates code and output into a single document that can be easily shared. To compile a report, go to **File > Compile Report...**.

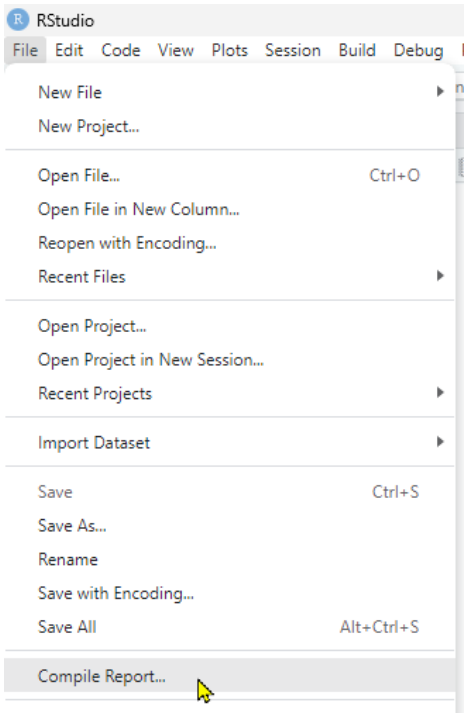


Figure 1.3: Compile Report Step 1

Although there are several different formats you may choose (e.g., PDF, Word, HTML), I find that compiling the report into Word format works best for my classes. So, under **Report output format:**, choose **MS Word**. Then click **Compile**.

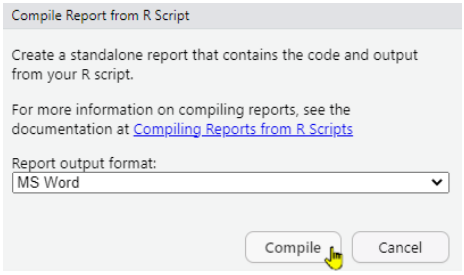


Figure 1.4: Compile Report Step 2

1.4.8 Debugging

When you are programming, you will make errors ("bugs") in your code. Trying to figure out where you made the error ("debugging") can be extremely time-consuming and frustrating. To help you along, here are some of the most common bugs that you'll run into.

1. Misspelled object or function
    - Misspelled object (e.g., condition vs condtion) will throw this error message: *Error: object not found*
    - Misspelled function (e.g., `t.test` vs `ttest`) or trying to use functions in packages that have not been loaded yet will throw this error message: *Error: could not find function*
    - R is case-sensitive. So if your object is called *Dataset*, you need to type *Dataset* and not *dataset*, else it will throw an error message.
    - If you're referring to a variable in the dataset, you must attach the `$` sign (e.g., dataset$variable). Otherwise, R will tell you it cannot find the object.
2. Punctuation mistake
    - Remember to close the parenthesis ()
    - Don't add a space where there shouldn't be. For example, if your object is called *dataset*, don't type *data set*.
    - Use the correct punctuation for the function. For example, if the function requires a comma, don't put in full stops. If it requires a double equal sign, don't put in a single equal sign.

If you've tried all the above and still can't figure out what's going on, just copy and paste the warning into the search engine or large language models (LLMs). 99% of the time, you'll be able to figure out what happened by reading the responses.

And speaking of LLMs…

1.4.9 On Using LLMs

I understand that this is the first time that some of you are using R. It can be quite intimidating. Fortunately, LLMs are *very* good at both generating code and interpreting the output. (This is so different from my time when I had to scour the forums to figure out what had gone wrong!)

The problem with LLMs is that they can lead you on a wild goose chase if you do not give them sufficient context. So, here are some things I've learnt that I think might be helpful.

1. **Give LLMs the full context.** The LLM typically doesn't know if a variable is a categorical variable or a quantitative one unless you tell it explicitly. (Of course, it can guess, but what if it guesses wrong?) It also won't know about your previous decisions (e.g., you decided to use dummy vs effects coding). If it doesn't know all these things, its interpretation of the output will be erroneous. Oh, and if you don't tell it you want R code, it is likely to give you Python code. So make sure you give it the full context.
2. **There are many ways of doing the same thing. Don't always accept the first set of code it gives you.** Due to the nature of my work, I typically prefer code that I can read and explain to someone else. So, if the LLM gives me a code that I don't understand or is too complex, I will ask for a simpler or different one. LLMs love giving me loop functions for things that can be done easily with the `map()` function from `tidyverse`, for example. (As an aside, usually I'll ask if it could give me a code in "tidyverse", as I find the "tidyverse" codes to be easier to understand.)
3. **Verify the code does what it's supposed to do.** Obviously, you need to check that the code runs properly and gives you the desired outcome. If it doesn't, copy the error message back into the LLM and ask it to de-bug. It usually does okay with de-bugging. However, I have, at times, found that it doesn't give correct explanations for why the errors appear. This is especially if the error is due to statistical issues (e.g., I forgot to convert a categorical variable into a factor so R thought it was a numeric / quantitative one). In such cases, you'll have to rely on your statistical understanding to de-bug. Other times, it's just a stupid typo (e.g., in the previous part of the code, I typed model1 and then 100 lines later, when referring to it in a subsequent code, I typed mod1). Unless R has all the code you typed, it won't pick that up either.

4. **Verify (almost) everything it says.** Do not trust the LLM's responses *prima facie*. Always ask for references. *And always click through to the references to check.* I have, on countless occasions, checked and found out the LLM was just hallucinating (i.e., the reference never stated whatever the LLM said it did), even in 2026, as of time of writing Or, the LLM output was just based on a single person's dissenting opinion on the web, and the LLM reported it as if it were scientific consensus. (There is currently sufficient research showing that LLMs' output can be "poisoned" by just a single Reddit post or a webpage.) So always click through to verify. Of course, there are some things that you don't have to verify if you already know them to be true or you don't really care about the veracity. :)
5. **Do not trust the LLM's reasoning unless it coheres with your own or you're able to independently confirm the reasoning.** It is very tempting for a novice to think, "The reasoning seems to make sense on the surface. I don't know any better. ¯_( )_/¯ I'll accept it then!" I have caught myself falling into that trap countless of times. It's only when I make myself go through the reasoning step by step that I realised the LLM was inconsistent or out-right wrong. An LLM once told me -0.75 + 1 = 0 and I didn't catch it until I worked through the calculation myself! If you find yourself stuck in a loop with the LLM, with it insisting it was right and you're wrong (happens to me a lot), it helps to start a new chat or to switch to a different model or an entirely different LLM. (I found that asking at different times yielded different reasoning and conclusions!)
6. **Saving tokens.** AI companies are now trying to turn a profit. As a result, they're starting to restrict the amount of tokens you have under the free version. Broadly speaking, tokens are words or part of words. The longer the query and/or the LLM output, the more tokens it will use. There are three ways to reduce tokens spent. 1) Use Caveman mode (see Caveman mode instructions below), 2) Start a new chat when possible, 3) Use the normal model for most tasks.

    1. **Caveman mode instructions (put this under personalisation or at the beginning of chat)**: Unless otherwise told, you will respond in Caveman Mode. You will use short, primitive sentences. Drop all filler words (like "the", "a", "of"), pleasantries, apologies, and explanations. Give only facts and direct commands. Retain full technical accuracy, but keep output as dense and short as possible.
    Note. I also add the following: "Where possible, provide sources for claims made in your responses."

        - This pushes the LLM output to be short and direct. To de-activate this, type "Use normal mode." To re-activate it (LLMs can start forgetting they were supposed to respond in Caveman mode after a while and revert to normal behaviour), type "Use Caveman mode."

    2. **Start a new chat.** When you ask additional questions in the same chat, the LLM essentially references everything (e.g., your previous

queries, its previous responses, its previous thought process) to answer your question. So, the more questions you ask within the same chat, the more tokens used each time. I recommend to start a new chat, if you can. Yes, you have to provide context all over again, but let's face it, not every response from the LLM was useful anyway and not every query was relevant. (See explanation from a computer science prof on why AI tokens are expensive here.)

- Yes, I know about prompt caching. But prompt caching only works if you ask a follow-up question reasonably quickly after the initial question. If you wait longer in between queries, that cache gets erased. Mike Pound, the prof in the video, talks about this as well.
- Some LLMs may choose to reference only the last X number of messages, reducing the number of tokens used. I don't know which LLMs do that. The start new chat method I recommend here works more generally.

3. **Use the basic LLM model.** For most coding questions, you don't need the most advanced model. Save your tokens for questions that *really* require the compute.

1.4.10 (Optional) Style Guide

I strongly believe that people should be able to reproduce each others' analyses. This provides the checks and balances that is important for science to advance. Therefore, you should be willing and able to share your code. But even if you do share your code, if it is unintelligible, it will not be of much help. Therefore, to make your code understandable, in addition to writing comments in the script, you should also try to follow certain conventions when writing code. These conventions are laid out in the tidyverse Style Guide. You can read the Style Guide here: https://style.tidyverse.org/syntax.html.

Now that you've installed R and RStudio, let's move on to some basics! =D

Chapter 2

Basics

2.1 Some Basic Commands

Before typing commands into the script editor, it might be useful for you to know some of the following basic commands.

1. To place a comment in the script file, begin the line with `#`.
2. To run (execute) a line in the script file, place the cursor on the line, hit `Ctrl/Cmd + Enter`.
    - If we typed the command directly into the console, we only need to hit `Enter`
3. To run multiple lines in the script file, select the lines, hit `Ctrl/Cmd + Enter`.
4. To run all lines in the script file, hit `Ctrl + Shift + Enter`
5. To clear the console when it becomes too messy, hit `Ctrl/Cmd + L`.
6. To request help, type a question mark in front of the command or the package name (e.g., `?cor`). Information about the command will appear in the Help tab (lower right pane).

2.2 Typing and Executing Commands

Now, let's actually get R to do stuff for us! Open a script file (`Ctrl/CMD + Shift + N`). Then copy and paste the following R codes into the script file. Run each line (`Ctrl/Cmd + Enter`) to see what they do. Note that `#` denotes a comment, and therefore it will run as a line of text. Experiment and have fun!

```r
# Assign a single value (e.g., 9) to an object, say x.
x <- 9  # this means "x gets the value of 9".

# Get the value for x.
x   # remember that R is case-sensitive. If you typed X, you'll
 ↪  get an error message.
X  # see how R complains here that it can't find the object?

# If you want to know what objects are in the workspace (i.e.,
 ↪  R's "short term" memory), look at the Environment tab or type
 ↪  ls().
ls()

# You may remove an object (e.g., x) from the workspace using
 ↪  rm(), where rm stands for remove.
rm(x)

# If there are too many objects in the workspace, you may remove
 ↪  all objects from the workspace using rm(list=ls()). To
 ↪  remember this command, I think of it as telling R to remove
 ↪  the list of objects in the Environment tab.
rm(list = ls())

# Assign a non-numerical value by putting the value in quotation
 ↪  marks.
y <- "hello!"

# Get the value of y. Notice the value of y is in quotation
 ↪  marks, indicating it is a non-numerical value.
y

# Perform the following mathematical operations in R.
11 + 10
11 - 10
11 * 10
11 / 10
11 ^ 10
11 ^ (1/2)
sqrt(11)   # this number should be the same as above line
log(11)    # taking natural log (log base e also known as ln)
log10(11)  # taking log base 10
exp(11)    # taking the exponential

# Perform mathematical operations in R with an object (e.g., a).
a <- 11  # a gets the value of 11
a + 10
```

```r
a - 10
a * 10
a / 10
a ^ (1/2)
sqrt(a)
log(a)
log10(a)
exp(a)

# Perform mathematical operations with more than one object.
y <- 2  # notice that 2 now replaces the value "hello".
a + y
a - y
a * y
a / y
a ^ (1/y)

# You cannot perform mathematical operations with non-numerical
 ↪  objects.
b <- "1"   # recall, putting things in between quotation marks
 ↪  makes it non-numerical, even if 1 is a number.
a + b  # you get an error here because you cannot perform
 ↪  mathematical operations with non-numerical objects.

# An object can store more than one value, such as a set of
 ↪  numbers or a set of characters. This is known as a vector and
 ↪  can be created using c().
num_vector <- c(1, 2, 3, 4, 5)   # numeric vector
fruits <- c("apple", "banana", "orange")   # non-numeric vector

# Get the values stored in the vectors
num_vector
fruits

# You can do mathematical operations with numerical vectors but
 ↪  not with non-numerical vectors.
num_vector * 5  # each value in num_vector is multiplied by 5
fruits * 5  # this throws an error that tells you that you cannot
 ↪  use non-numeric values for this operation
```

2.3 Entering Data Directly Into R

In reality, we rarely use R to do such simple mathematical calculations; we use R for data analyses. But before we can conduct any analysis, we need to give

R some data. One way is to key the data directly into R.

Let's say we have three people, Bob, Andrea, and Calvin. We have their ages, the number of children each of them has, and their gender.

Name	Age	No. of Children	Gender
Bob	48	1	Male
Andrea	47	3	Female
Calvin	49	2	Male

Let's figure out how to give R this set of data.

First, we create an object `name`, and assign the three people's names to it.

```
name <- c("Bob", "Andrea", "Calvin")
# the c in c() stands for combine
# here, we're telling R to combine the three names into a list of
 ↪  names
# notice the names are between quotation marks
# this tells R that name is a string (non-numerical) variable
```

If we type `name` into the console, we should get:

```
## [1] "Bob"    "Andrea" "Calvin"
```

Next, we create the object `age`, with the three people's ages.

```
age <- c(48, 47, 49)
# here, we're creating a list of ages
# the order of age should match the order of the names
```

If we type `age` into the console, we should get:

```
## [1] 48 47 49
```

Now, let's combine the two variables into a data frame. You can think of a data frame as a spreadsheet or as a table, where each row represents the information for one person, and `name` and `age` are in side-by-side columns.

We will use the `data.frame()` function and call this data frame `dataset`.

```
dataset <- data.frame(Name = name, Age = age)
# Name = name tells R that the column should have the header Name
 ↪  and the data for the column should be copied from the object
 ↪  name.
# Age = age tells R that the column should have the header Age
 ↪  and the data for the column should be copied from the object
 ↪  age.
```

Type `dataset` into the console. You should see:

```
##     Name Age
## 1    Bob  48
## 2 Andrea  47
## 3 Calvin  49
```

Notice that the output has two columns: `Name` and `Age`, where `Name` lists the names of the individuals and `Age` lists their ages.

The values of `Name` and `Age` in the data frame were copied from the original objects. This means that the original objects, `name` and `age`, are still in R's memory. You can see that this is the case from the Environment tab or when you use the ls() function.

```
## [1] "age"     "dataset" "name"
```

Let's remove the original variables `name` and `age`.

```
rm(name)
rm(age)
```

Now, when you type ls() into the console, you'll see that all is left is the data frame, `dataset`.

```
## [1] "dataset"
```

If you wish, you can directly type the data into a data frame instead of first creating separate objects for each variable.

```
dataset2 <- data.frame(Name = c("Bob", "Andrea", "Calvin"), Age =
 ↪  c(48, 47, 49))
# dataset and dataset2 will look exactly the same
```

Now, suppose we want to use the variables in the data frame `dataset`. We will need to attach `dataset$` (a dollar sign after the name of the data frame) before the variable name. This will tell R that that variable comes from the data frame `dataset`.

For example, if we want to know the ages of the three participants. We should type:

```
dataset$Age
```

```
## [1] 48 47 49
```

```
# this tells R to go to the object called dataset, locate the
 ↪  column Age and then produce the values in that column
# remember that R is case-sensitive. So if you'd typed
 ↪  dataset$age, you'll get an error.
# typing age or Age won't work either because there isn't an
 ↪  object called age or Age.
```

From the original table, we know that there are more variables (columns) we need to add to the data frame `dataset`: the number of children the person has and the person's gender. Let's label the number of children each person has as `Children` and the gender of each person as `Gender`.

```
dataset$Children <- c(1, 3, 2)
# this tells R to go to the object dataset and create a column
 ↪  called Children. Then, assign the column the values 1, 3, and
 ↪  2.

dataset$Gender <- c("male", "female", "male")
# this tells R to go to the object dataset and create a column
 ↪  called Gender. Then, assign the column the values "male",
 ↪  "female", and "male"

# again, notice the order of dataset$Children and dataset$Gender
 ↪  match the order of the names
```

Now, when you type `dataset` into the console, you'll get:

```
##     Name Age Children Gender
## 1    Bob  48        1   male
## 2 Andrea  47        3 female
## 3 Calvin  49        2   male
```

Note that we created `Children` and `Gender` within the data frame `dataset`. So if you type `Children` and `Gender` without `dataset$`, you will get an error message in the console telling you that the object cannot be found because there isn't an object (independent of the data frame) that is called `Children` or `Gender`.

Now, let's say we made a mistake and need to remove `Children` from the data frame, `dataset`. We can type:

```
dataset$Children <- NULL
# this tells R to assign the value of NULL to dataset$Children
# NULL basically represents non-existence. So, when we assign
 ↪  NULL to dataset$Children, it means we want that column to be
 ↪  non-existent / removed.

dataset
```

```
##     Name Age Gender
## 1    Bob  48   male
## 2 Andrea  47 female
## 3 Calvin  49   male
```

```
# notice that the Children column is now gone
```

Finally, if we want to save the data frame into a .csv file, we use `write.csv()`.

```
write.csv(dataset, "bobandreacalvin.csv")
# this tells R to save the object, dataset, as a .csv file, and
 ↪  to name the .csv file as bobandreacalvin.csv.
# remember to specify both the name and the extension of the file
 ↪  (e.g., bobandreacalvin.csv).
```

The .csv file will now be saved as `bobandreacalvin.csv` within your current directory. If you used an R project, it will be saved in your project directory. If you didn't use an R project but set the working directory, it will be in the working directory. Otherwise, it will be in the default directory. If you don't know what the directory is, type `getwd()` into the console.

As an aside, some students might wonder why we don't use `write.csv2` instead. `write.csv2` uses a comma for the decimal point whereas `write.csv` uses a full stop / period for the decimal point. Since it's more usual to use the full stop / period for the decimal point, we'll stick to `write.csv`.

Phew! That was a lot of information. Now… Try to apply what you learnt in this section to the exercise in the following section!

Chapter 3

Test Yourselves 1

Before working on this exercise, please read through the Getting Started and the Basics sections first, especially if you have no background in R.

Suggested answers are at the bottom of this page, but please do try the exercise before looking at them. :)

3.1 Instructions

1. Download and install R.
2. Download and install RStudio.
3. Start an R Project.
4. Open up a new script file in RStudio.

5. Suppose we have the starting monthly salary of 10 men and 10 women (see **Data** section below). Enter the data provided into a data frame (call this data frame `dataset`) where one column represents the biological sex of the individual (call this column `BioSex`) and the other column represents the starting monthly salary (call this column `Salary`) of the individual. Make sure you have 20 rows (one for each individual) and 2 columns (one for each variable).
6. Save the data frame into a .csv file. Name this file `data.csv`. Check in the working directory that the .csv file has been created correctly.

3.2 Data

Man	Woman
3530	3790
4730	2720
4330	3170
3560	3320
4050	3190
4880	2890
4190	2920
3620	3390
3530	3790
5070	3680

3.3 Suggested Answers

```
# Alternative 1
dataset <- data.frame(BioSex = c("man", "man", "man", "man",
 ↪  "man", "man", "man", "man", "man", "man", "woman", "woman",
 ↪  "woman", "woman", "woman", "woman", "woman", "woman",
 ↪  "woman", "woman"),
                  Salary = c(3530, 4730, 4330, 3560, 4050, 4880,
                   ↪  4190, 3620, 3530, 5070, 3790, 2720, 3170,
                   ↪  3320, 3190, 2890, 2920, 3390, 3790, 3680))

# Alternative 2
# In this alternative, I call the objects sex and money instead
 ↪  so that you can see how they work in data.frame function.

sex <- c(rep("man", 10),
         rep("woman", 10))
# tell R to replicate the string "man" 10 times and replicate the
 ↪  string "woman" 10 times. Then combine the results into a list
 ↪  and save it as sex. The first 10 values will be "man" and the
 ↪  next 10 will be "woman".

money <- c(3530, 4730, 4330, 3560, 4050, 4880, 4190, 3620, 3530,
 ↪  5070, 3790, 2720, 3170, 3320, 3190, 2890, 2920, 3390, 3790,
 ↪  3680)
# tell R to combine the 20 values into a list and save it as
 ↪  money

dataset <- data.frame(BioSex = sex,
```

```
                      Salary = money)
# create data frame

# Alternative 3
# Similar to Alternative 2 except that the three steps are
 ↪  combined into one command.

dataset <- data.frame(BioSex = c(rep("man", 10),
                                 rep("woman", 10)),
                      Salary = c(3530, 4730, 4330, 3560, 4050,
                       ↪  4880, 4190, 3620, 3530, 5070, 3790,
                       ↪  2720, 3170, 3320, 3190, 2890, 2920,
                       ↪  3390, 3790, 3680))

# Save data file
write.csv(dataset, "data.csv")
```

Chapter 4

Using tidyverse

Warning: This is a pretty long tutorial. However, it covers commands that are useful for data cleaning / wrangling, which I like to include in my midterms and exams (*hint hint*). I am unlikely to cover this extensively in class, so please do set aside the time to go through this on your own!

Now that you have some basics under your belt, let's move on to installing and using packages from `tidyverse`!

The `tidyverse` is a collection of R packages that many data analysts use. You can think of packages as collections of R codes (and more!) that other people have created to make R more powerful or easier to use. In this course, we will be heavily relying on a few of the packages such as `readr`, `dplyr` and `ggplot2` that are in the `tidyverse` collection. Instead of installing each package separately as we normally would, we can simply install `tidyverse` and that would install all the other packages that we want.

4.1 Installing and Loading tidyverse

To install `tidyverse`, use the function `install.packages()`.

```
install.packages("tidyverse")
```

Even after we have installed tidyverse, we cannot use it unless we load it. To load it, we need the `library()` function.

Note that we only need to install each package once. But we need to load the package each session (i.e., each time we re-start RStudio). I like to think of the package like a genie that goes to sleep each time you close RStudio. If you need the genie to do stuff for you the next time around, you need to awaken it again!

```
library(tidyverse)
```

Note. You can temporarily load a package without using the `library()` function using the notation `package::function`. This tells R to load the package for a specific chunk of code and not for the entire session. This allows anyone who reads the code to know which package the function comes from. I typically don't do this because it can be quite tiresome to keep typing the name of the package over and over.

However, `package::function` comes in especially handy when we want to help R distinguish between two packages with the same function names. For example, the packages `ggplot2` and `psych` both have a function called `alpha()`. If you load both together, R will "mask" the function from one of the packages (it will tell you which package is masked when loading the packages). What this means is that R will use the function from the package that is not masked. As you can imagine, this can be quite confusing. So, to specify that you want to use the function from the `psych` package, you should type `psych::alpha()`. Similarly, if you want to specify that the function comes from the `ggplot2` package, type `ggplot2::alpha()`.

Now that we have `tidyverse` loaded, let's do stuff with it. Let's start with importing data.

4.2 Import Data

Usually, R users do not have to enter their data into R directly. Instead, they already have their dataset in various formats, such as .csv, .txt, .sav, .xlsx, etc. So what they need to do is to import the dataset into R.

In this class, I will use mainly .csv files. To import .csv files, we can use the `read_csv()` function from the `readr` package in `tidyverse`. I have created a hypothetical dataset to demonstrate how to import .csv files. To follow along, please download the dataset here: SWB.csv.

4.2.1 SWB Dataset

In this hypothetical dataset, 343 participants provided demographic information and responded to the 9-item Materialism Values Survey (MVS) and the 5-item Satisfaction with Life Scale in 2019 (SWLS2019). The same participants were asked to respond to the Satisfaction with Life Scale in 2021 (SWLS2021). The legend for this dataset is as follows:

Variable Name	Variable Label	Value Label
pin	participant identification number	
gender	gender	0 = male, 1 = female
marital_status	marital status	1 = married, 2 = divorced, 3 = widowed
have_children	parental status	0 = no children, 1 = have children
mvs_1	My life would be better if I own certain things I don't have.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_2	The things I own say a lot about how well I'm doing.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_3	I'd be happier if I could afford to buy more things.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_4	It bothers me that I can't afford to buy things I'd like.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_5	Buying things gives me a lot of pleasure.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_6	I admire people who own expensive homes, cars, clothes.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree

Variable Name	Variable Label	Value Label
mvs_7	I like to own things that impress people.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_8	I like a lot of luxury in my life.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_9	I try to keep my life simple, as far as possessions are concerned. (R)	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
swls2019_1	In most ways my life is close to my ideal. (2019)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2019_2	The conditions of my life are excellent. (2019)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2019_3	I am satisfied with my life. (2019)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2019_4	So far I have gotten the important things I want in life. (2019)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2019_5	In most ways my life is close to my ideal. (2019)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree

Variable Name	Variable Label	Value Label
swls2021_1	In most ways my life is close to my ideal. (2021)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2021_2	The conditions of my life are excellent. (2021)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2021_3	I am satisfied with my life. (2021)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2021_4	So far I have gotten the important things I want in life. (2021)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2021_5	In most ways my life is close to my ideal. (2021)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree

Now that we have some idea of what the dataset comprises, let's read in the .csv file.

Before doing that though, we need to set the working directory. The reason is that we need to tell R where to find the .csv file that we saved. To do that, in RStudio, go to **Session > Set Working Directory > Choose Directory**. Select the folder where you saved the .csv file.

Now that we've set the working directory, we will read in the .csv file.

```
dataset <- read_csv("SWB.csv")

# this code is saying...
# read in the .csv file, "SWB.csv" and assign it to the object
 ↪  "dataset"
```

```
# from now on, "dataset" will refer to the data in this csv file
```

`read_csv()` is not the only function you can use to read in a .csv file. You can also use the built-in R function `read.csv()`.

```
dataset <- read.csv("SWB.csv")
```

If you use `read.csv()`, you don't have to load `tidyverse`. This is because `read_csv()` comes from the `readr` package (and therefore the `tidyverse` collection) whereas `read.csv()` comes pre-installed with R.

`read.csv()` is supposedly slower than `read_csv()`, which in turn is slower than `data.table`'s `fread()`. For our purposes, though, it really doesn't matter which you use, so long as you can successfully read the .csv file. Another thing about `read.csv()` is that it tries to identify categorical variables and then automatically import them as factors (see the Factors section below). I don't quite like that because I like to control the variable type myself. But, that's my personal preference. You're free to use whatever function that you want. A final thing about `read.csv()` is that it does not remove trailing spaces after a number (e.g., 3) whereas `read_csv()` does. So when importing with `read.csv()`, there might be some issues if you have such situations.

If you have other file types, such as .txt, .sav, .xlsx, you might need different packages. For example, for excel files (.xlsx), you will need the `readxl` package. For SPSS (.sav), you will need the `haven` package or the `foreign` package. For this class, I am very unlikely to use file types aside from .csv, but it is good to be aware of the packages to use if you want to import other file types.

4.2.2 Check Imported Dataset

Before conducting any analyses, check that the dataset has been imported correctly. Go to the `Environment` pane (top right pane of RStudio). Click on `dataset`. The top left pane should now show the imported data in what looks like a spreadsheet. Alternatively, you may type `View(dataset)` into the console.

1. **Rows**: The data for each participant is recorded in a single row (e.g., data for Participant 1 is in Row 1)
2. **Columns**: The data for each variable is recorded in a single column. Names of the variables are in the headers for each column

Scroll down to ensure all rows have been imported correctly. There should be 343 rows. Scroll right to ensure all columns have been imported correctly. There should be 23 columns.

Another way to check the imported dataset is using the `str()` or the `glimpse()` functions. Both give you roughly similar information (e.g., that there are 343 rows and 23 columns, the names of the variables, the values for the first 10 or so participants). Compare them below.

```
str(dataset)
```

```
## spc_tbl_ [343 x 23] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ pin           : num [1:343] 1 2 3 4 5 6 7 8 9 10 ...
##  $ gender        : num [1:343] 0 1 1 1 1 0 1 0 1 0 ...
##  $ marital_status: num [1:343] 1 1 3 1 2 1 2 1 2 1 ...
##  $ have_children : num [1:343] 0 1 0 0 0 1 0 0 0 1 ...
##  $ mvs_1         : num [1:343] 4 4 5 2 2 3 2 2 3 2 ...
##  $ mvs_2         : num [1:343] 3 3 4 3 2 4 3 3 4 3 ...
##  $ mvs_3         : num [1:343] 3 3 4 3 3 4 3 3 4 3 ...
##  $ mvs_4         : num [1:343] 3 3 3 2 2 5 4 4 3 4 ...
##  $ mvs_5         : num [1:343] 3 3 4 3 3 4 3 3 4 2 ...
##  $ mvs_6         : num [1:343] 4 4 5 2 2 3 4 4 5 2 ...
##  $ mvs_7         : num [1:343] 3 2 4 2 3 3 3 3 4 3 ...
##  $ mvs_8         : num [1:343] 4 4 5 4 4 5 3 3 4 3 ...
##  $ mvs_9         : num [1:343] 2 2 1 3 3 2 3 3 2 4 ...
##  $ swls2019_1    : num [1:343] 4 5 4 6 5 5 2 4 5 5 ...
##  $ swls2019_2    : num [1:343] 4 6 4 5 4 4 3 3 5 5 ...
##  $ swls2019_3    : num [1:343] 5 5 4 7 4 4 3 3 5 5 ...
##  $ swls2019_4    : num [1:343] 5 6 5 6 4 5 3 3 4 6 ...
##  $ swls2019_5    : num [1:343] 4 6 5 5 5 4 3 3 5 6 ...
##  $ swls2021_1    : num [1:343] 4 4 3 5 4 5 2 4 4 5 ...
##  $ swls2021_2    : num [1:343] 4 6 3 5 3 4 2 3 4 5 ...
##  $ swls2021_3    : num [1:343] 5 4 4 6 4 3 4 3 5 4 ...
##  $ swls2021_4    : num [1:343] 5 5 5 6 4 4 3 3 4 5 ...
##  $ swls2021_5    : num [1:343] 4 5 5 5 5 4 3 3 5 6 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   pin = col_double(),
##   ..   gender = col_double(),
##   ..   marital_status = col_double(),
##   ..   have_children = col_double(),
##   ..   mvs_1 = col_double(),
##   ..   mvs_2 = col_double(),
##   ..   mvs_3 = col_double(),
##   ..   mvs_4 = col_double(),
##   ..   mvs_5 = col_double(),
##   ..   mvs_6 = col_double(),
##   ..   mvs_7 = col_double(),
##   ..   mvs_8 = col_double(),
```

```
##   ..   mvs_9 = col_double(),
##   ..   swls2019_1 = col_double(),
##   ..   swls2019_2 = col_double(),
##   ..   swls2019_3 = col_double(),
##   ..   swls2019_4 = col_double(),
##   ..   swls2019_5 = col_double(),
##   ..   swls2021_1 = col_double(),
##   ..   swls2021_2 = col_double(),
##   ..   swls2021_3 = col_double(),
##   ..   swls2021_4 = col_double(),
##   ..   swls2021_5 = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
glimpse(dataset)
```

```
## Rows: 343
## Columns: 23
## $ pin            <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, ~
## $ gender         <dbl> 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0~
## $ marital_status <dbl> 1, 1, 3, 1, 2, 1, 2, 1, 2, 1, 3, 2, 3, 3, 3, 1, 1, 1, 3~
## $ have_children  <dbl> 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0~
## $ mvs_1          <dbl> 4, 4, 5, 2, 2, 3, 2, 2, 3, 2, 2, 3, 2, 2, 3, 3, 3, 4, 3~
## $ mvs_2          <dbl> 3, 3, 4, 3, 2, 4, 3, 3, 4, 3, 2, 4, 1, 3, 2, 3, 2, 4, 3~
## $ mvs_3          <dbl> 3, 3, 4, 3, 3, 4, 3, 3, 4, 3, 3, 4, 2, 2, 3, 4, 4, 5, 3~
## $ mvs_4          <dbl> 3, 3, 3, 2, 2, 5, 4, 4, 3, 4, 4, 4, 2, 2, 4, 3, 3, 4, 4~
## $ mvs_5          <dbl> 3, 3, 4, 3, 3, 4, 3, 3, 4, 2, 2, 3, 3, 3, 4, 4, 4, 5, 3~
## $ mvs_6          <dbl> 4, 4, 5, 2, 2, 3, 4, 4, 5, 2, 2, 3, 3, 3, 4, 2, 2, 3, 4~
## $ mvs_7          <dbl> 3, 2, 4, 2, 3, 3, 3, 3, 4, 3, 1, 4, 4, 1, 5, 1, 3, 2, 3~
## $ mvs_8          <dbl> 4, 4, 5, 4, 4, 5, 3, 3, 4, 3, 3, 4, 2, 2, 3, 3, 3, 4, 4~
## $ mvs_9          <dbl> 2, 2, 1, 3, 3, 2, 3, 3, 2, 4, 4, 3, 4, 4, 3, 3, 3, 2, 3~
## $ swls2019_1     <dbl> 4, 5, 4, 6, 5, 5, 2, 4, 5, 5, 5, 3, 4, 5, 4, 5, 5, 6, 4~
## $ swls2019_2     <dbl> 4, 6, 4, 5, 4, 4, 3, 3, 5, 5, 5, 4, 3, 5, 4, 4, 4, 6, 5~
## $ swls2019_3     <dbl> 5, 5, 4, 7, 4, 4, 3, 3, 5, 5, 4, 4, 3, 4, 3, 4, 4, 5, 4~
## $ swls2019_4     <dbl> 5, 6, 5, 6, 4, 5, 3, 3, 4, 6, 4, 4, 4, 5, 4, 4, 5, 6, 4~
## $ swls2019_5     <dbl> 4, 6, 5, 5, 5, 4, 3, 3, 5, 6, 5, 4, 3, 4, 3, 5, 5, 6, 4~
## $ swls2021_1     <dbl> 4, 4, 3, 5, 4, 5, 2, 4, 4, 5, 5, 2, 3, 5, 4, 4, 5, 6, 4~
## $ swls2021_2     <dbl> 4, 6, 3, 5, 3, 4, 2, 3, 4, 5, 4, 3, 2, 4, 3, 4, 4, 6, 4~
## $ swls2021_3     <dbl> 5, 4, 4, 6, 4, 3, 4, 3, 5, 4, 4, 4, 3, 4, 2, 3, 3, 4, 4~
## $ swls2021_4     <dbl> 5, 5, 5, 6, 4, 4, 3, 3, 4, 5, 4, 4, 4, 5, 3, 3, 4, 5, 4~
## $ swls2021_5     <dbl> 4, 5, 5, 5, 5, 4, 3, 3, 5, 6, 5, 4, 3, 4, 3, 4, 5, 6, 4~
```

4.3 Factors

When you look at the output from `str(data)` or `glimpse(data)`, you can see that some values don't seem to make sense on their own. For example, what do 0 and 1 in gender mean? From the legend table earlier, we can see that 0 stands for male and 1 stands for female. But having to consult the table every time we do an analysis is tiresome and also error-prone. So, we use the `factor()` command to tell R what those values mean for categorical variables.

```
dataset$gender <- factor(dataset$gender,
                         levels = c(0, 1),
                         labels = c("male", "female"))
# this code is saying...
# we want to convert the variable gender in the dataset into a
 ↪  factor
# this variable has two values (levels), 0 and 1
# label 0 as "male" and 1 as "female"
# notice that male and female are enclosed in quotation marks
 ↪  (because they are non-numerical or string variables).

# Check that gender is now a factor with either glimpse() or
 ↪  str()
# They'll give you the same output
glimpse(dataset$gender)
```

```
##  Factor w/ 2 levels "male","female": 1 2 2 2 2 1 2 1 2 1 ...
```

4.4 Data Wrangling With dplyr

What should we do after importing a dataset? Well, there are many things we can do! We might want to, say, select specific columns to analyse. Or we might want to select only participants who completed all the questions in the survey. Or we might want to create new variables from existing ones. Or we might want to get some descriptive statistics like mean and standard deviation. All of that is made possible with functions in the `dplyr` package in `tidyverse`.

Some of the functions I've found useful in the `dplyr` package are:

dplyr functions	Description
select()	select specific columns
filter()	filter (keep, select) specific rows
mutate()	create new variables / columns
summarise()	summarise values in form of summary statistics

dplyr functions	Description
group_by()	apply operations to different groups

There are more functions in the dplyr package! Check out the dplyr cheat sheet here: https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf.

When you go through the code below, you'll notice that in `dplyr` (and also `tidyverse` in general), the pipe operator `|>` is used a lot. You can think of it as a symbol for "then". For example `X |> Y` would be read as: Do X then take the result from that to do Y. Because it gets tiring to type the pipe operator so frequently, there's a shortcut for it: `Ctrl/Cmd + Shift + M`. (You may see the pipe operator written as `%>%` as well because that was what the pipe operator looked like before 2023.)

With that out of the way… Let's go through the five functions!

4.4.1 Select()

The `select()` function allows us to select specific *columns*. This is especially useful if we have many columns to work with and we only want to focus on a few.

Let's say we only want to select `gender` and the `swls2019_1` to `swls2019_5` columns in the `SWB` dataset. Here's how we would do it:

```
swls2019_only <- dataset |>   # create the subset called
 ↪  "swls2019_only" from "dataset", and then
  select(gender, swls2019_1:swls2019_5)  # select gender and the
   ↪  columns swls2019_1 to swls2019_5
```

This is what the first six rows of `swls2019_only` look like:

```
## # A tibble: 6 x 6
##   gender swls2019_1 swls2019_2 swls2019_3 swls2019_4 swls2019_5
##   <fct>       <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 male            4          4          5          5          4
## 2 female          5          6          5          6          6
## 3 female          4          4          4          5          5
## 4 female          6          5          7          6          5
## 5 female          5          4          4          4          5
## 6 male            5          4          4          5          4
```

I sometimes also use `select()` to re-arrange the order of the columns. (In `dplyr`, the "correct" function to re-arrange the order of the columns is `arrange()`. `arrange()` allows us to re-arrange rows in addition to columns. But `select()` works too!)

So, let's say we want the `swls2019_1` to `swls2019_5` columns to come *before* `gender`. We simply list the `swls2019_1` to `swls2019_5` columns before `gender`.

```
swls2019_genderlast <- dataset |>
  select(swls2019_1:swls2019_5, gender)
```

This is what the first six rows of `swls2019_genderlast` look like:

```
## # A tibble: 6 x 6
##   swls2019_1 swls2019_2 swls2019_3 swls2019_4 swls2019_5 gender
##        <dbl>      <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1          4          4          5          5          4 male
## 2          5          6          5          6          6 female
## 3          4          4          4          5          5 female
## 4          6          5          7          6          5 female
## 5          5          4          4          4          5 female
## 6          5          4          4          5          4 male
```

4.4.2 Filter()

Maybe we don't want to select specific columns. Instead, we want to select specific kinds of participants to conduct our analyses on (e.g., say from specific treatment groups or maybe only male participants). In other words, we want to select specific *rows*. We use `filter()` to do that.

New R users sometimes confuse `select()` with `filter()` so it bears repeating: `select()` is used to select columns (i.e., variables) whereas `filter()` is used to select rows (i.e., participants).

So, let's say we want to select only male participants. In other words, we only want to keep the rows that come from male participants.

```
male_only <- dataset |>  # create "male_only" subset from
 ↪  "dataset", and then
  filter(gender == "male")  # filter (keep) only male
   ↪  participants
```

Notice that the double equal sign `==` is used here. In programming languages, the `==` sign is used when we are comparing the left and the right hand side to see if they match. Here, we're comparing each row of the column `gender` to the

word "male". If that row matches the word "male", we will filter (keep) that row. Otherwise, we will toss it out.

This is what the first six rows of `male_only` look like:

```
## # A tibble: 6 x 23
##     pin gender marital_status have_children mvs_1 mvs_2 mvs_3 mvs_4 mvs_5 mvs_6
##   <dbl> <fct>           <dbl>         <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1 male                1             0     4     3     3     3     3     4
## 2     6 male                1             1     3     4     4     5     4     3
## 3     8 male                1             0     2     3     3     4     3     4
## 4    10 male                1             1     2     3     3     4     2     2
## 5    11 male                3             0     2     2     3     4     2     2
## 6    14 male                3             0     2     3     2     2     3     3
## # i 13 more variables: mvs_7 <dbl>, mvs_8 <dbl>, mvs_9 <dbl>, swls2019_1 <dbl>,
## #   swls2019_2 <dbl>, swls2019_3 <dbl>, swls2019_4 <dbl>, swls2019_5 <dbl>,
## #   swls2021_1 <dbl>, swls2021_2 <dbl>, swls2021_3 <dbl>, swls2021_4 <dbl>,
## #   swls2021_5 <dbl>
```

4.4.3 Mutate()

Sometimes, we might want to create new variables, say averages or totals. We can do this with `mutate()`.

Let's say we want to find the average of `swls2019_1` to `swls2019_5` for each participant. We will use the subset we created just now, `swls2019_only` to do this.

```
swls2019_only <- swls2019_only |>
  mutate(swls2019_avg = (swls2019_1 + swls2019_2 + swls2019_3 +
    ↪  swls2019_4 + swls2019_5) / 5)
# this code is saying...
# go to swls2019_only, and then
# create a variable called `swls2019_avg` by adding up swls2019_1
 ↪  to swls2019_5 and then dividing the total by 5
# overwrite swls2019_only with this new dataset (that includes
 ↪  `swls2019_avg`)
```

The first 6 rows of `swls2019_only` now looks like this:

```
## # A tibble: 6 x 7
##   gender swls2019_1 swls2019_2 swls2019_3 swls2019_4 swls2019_5 swls2019_avg
##   <fct>       <dbl>      <dbl>      <dbl>      <dbl>      <dbl>        <dbl>
## 1 male            4          4          5          5          4          4.4
## 2 female          5          6          5          6          6          5.6
```

```
## 3 female          4          4          4          5          5          4.4
## 4 female          6          5          7          6          5          5.8
## 5 female          5          4          4          4          5          4.4
## 6 male            5          4          4          5          4          4.4
```

Notice that the `swls2019_avg` now appears as the last column of `swls2019_only`.

You can achieve the same results using `rowMeans()`. (As an aside, if you want the total instead of the average, use `rowSums()`.)

```
swls2019_only <- swls2019_only |>
  mutate(swls2019_avg_rowmeans =
    ↪  rowMeans(across(c(swls2019_1:swls2019_5))))
# rowMeans(across(c(swls2019_1:swls2019_5))) tells R that we want
 ↪  to find the mean (average) for each row (hence rowMeans)
 ↪  across the set of five variables, swls2019_1 to swls2019_5.
```

Let's compare `swls2019_avg` and `swls2019_avg_rowmeans` for the first 6 rows:

```
## # A tibble: 6 x 2
##   swls2019_avg swls2019_avg_rowmeans
##          <dbl>                 <dbl>
## 1          4.4                   4.4
## 2          5.6                   5.6
## 3          4.4                   4.4
## 4          5.8                   5.8
## 5          4.4                   4.4
## 6          4.4                   4.4
```

You should notice that the two columns give you the exact same results.

4.4.4 Summarise()

The `summarise()` function is used when we want to get summary statistics such as the mean, median, maximum, minimum, etc for a given column in the data frame.

Let's say we're interested to know the mean (`avg_swls`), minimum (`min_swls`), maximum (`max_swls`), variance (`var_swls`), standard deviation (`sd_swls`), total number of participants (`total`) for `swls2019_avg`. Again, we will use our `swls2019_only` subset.

```
swls2019_only |>
  summarise(avg_swls = mean(swls2019_avg),
            min_swls = min(swls2019_avg),
            max_swls = max(swls2019_avg),
            var_swls = var(swls2019_avg),
            sd_swls = sd(swls2019_avg),
            total = n())
```

```
## # A tibble: 1 x 6
##   avg_swls min_swls max_swls var_swls sd_swls total
##      <dbl>    <dbl>    <dbl>    <dbl>   <dbl> <int>
## 1     4.35        2        6    0.588   0.767   343
```

To be honest, though, I think it might be more convenient to get these summary statistics using the `describe()` function from the `psych()` package.

```
# Load package (install the package before loading!)
library(psych)

# Get the most common descriptive statistics
describe(swls2019_only$swls2019_avg)
```

```
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 343 4.35 0.77    4.4    4.33 0.89   2   6     4  0.1    -0.48 0.04
```

4.4.5 Group_by()

Sometimes, we might want to apply the same function to different groups of people. For example, we might want to know what the maximum swls score is for males and for females separately. We can use the `group_by()` function to do this.

Let's say we're interested to know the mean (`avg_swls`), minimum (`min_swls`), maximum (`max_swls`), variance (`var_swls`), standard deviation (`sd_swls`), total number of participants (`total`) for `swls2019_avg` for males and females separately.

```
swls2019_only |>
  group_by(gender) |>
  summarise(avg_swls = mean(swls2019_avg),
            min_swls = min(swls2019_avg),
            max_swls = max(swls2019_avg),
            var_swls = var(swls2019_avg),
```

```
            sd_swls = sd(swls2019_avg),
            total = n())
```

```
## # A tibble: 2 x 7
##   gender avg_swls min_swls max_swls var_swls sd_swls total
##   <fct>     <dbl>    <dbl>    <dbl>    <dbl>   <dbl> <int>
## 1 male       4.08        3      5.8    0.373   0.611   168
## 2 female     4.62        2      6      0.657   0.810   175
```

Again, I think that the `describeBy()` function from the `psych` package would be much more convenient here.

```
describeBy(swls2019_only$swls2019_avg,
           group = swls2019_only$gender)
```

```
##
##  Descriptive statistics by group
## group: male
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 168 4.08 0.61    4.2    4.06 0.59   3 5.8   2.8 0.26    -0.46 0.05
## ------------------------------------------------------------
## group: female
##    vars   n mean   sd median trimmed  mad min max range  skew kurtosis   se
## X1    1 175 4.62 0.81    4.6    4.65 0.89   2   6     4 -0.35    -0.25 0.06
```

I've shown in some examples above that we can combine different dplyr functions such as `group_by()` and `summarise()` to achieve specific results. I encourage you to play around with the combinations to see what results they lead to!

4.5 Tables and Plots with ggplot2

Okay, so what else can we do with the data? Well, we can create tables and graphs! But why might we do this? Well… Tables and graphs provide us a way to visualize the data. They show us the distribution of the data and allow us to discover any errors or interesting patterns.

Let's start with the frequency (distribution) tables.

4.5.1 Frequency Tables

A frequency (distribution) table tells us the number of times each data value occurs for a given variable.

Let's say we want to know how many males and how many females participated in the study using the `swls2019_only` subset. The variable is `gender` and the data values we are interested in are `male` and `female`. Let's find out!

Before looking at my code below, try coming up with your own! (HINT: Use the `summarise()` function!)

```
swls2019_only |>
  group_by(gender) |>  # Group the result by gender
  summarise(freq = n())  # Count how many of each value in the
   ↪  data
```

```
## # A tibble: 2 x 2
##   gender  freq
##   <fct>  <int>
## 1 male     168
## 2 female   175
```

The output tells us 168 males and 175 females participated in the study.

Here's a faster alternative.

```
swls2019_only |>
  select(gender) |>  # Select the column gender
  table()   # Count how many of each value in the data and
   ↪  present it in a table
```

```
## gender
##   male female
##    168    175
```

The number of times a data value occurs for a given variable is called the absolute frequency. If we want the relative frequency (i.e., the number of times a data value occurs relative to the total number of observations for a given variable), then we divide the absolute frequency by the total number of values.

Again, try this on your own before looking at my code. (HINT: Use the `summarise()` function and then the `mutate()` function!)

```
swls2019_only |>
  group_by(gender) |>   # Group the result by gender
  summarise(freq = n()) |>  # Count how many of each value in the
   ↪  data
  mutate (total = sum(freq),  # Count the total number of
   ↪  observations
          rel_freq = freq / total)  # Take the frequency of each
           ↪  value / total number of observations
```

```
## # A tibble: 2 x 4
##   gender  freq total rel_freq
##   <fct>  <int> <int>    <dbl>
## 1 male     168   343    0.490
## 2 female   175   343    0.510
```

The output (rel_freq) tells us that slightly less than half the sample are males (0.49) and slightly more than half are females (0.51).

Here's a faster alternative, using `table()` and `prop.table()`.

```
swls2019_only |>
  select(gender) |>
  table() |>
  prop.table()
```

```
## gender
##      male    female
## 0.4897959 0.5102041
```

Relative frequency is a proportion and may be difficult for some people to interpret. For easier interpretation, we can convert the proportions into percentages by multiplying 100%.

Try converting the proportions into percentages on your own before looking at my code below!

```
swls2019_only |>
  group_by(gender) |>
  summarise(freq = n()) |>
  mutate (total = sum(freq),
          rel_freq = freq / total,
          percentage = rel_freq * 100)
```

```
## # A tibble: 2 x 5
##   gender  freq total rel_freq percentage
##   <fct>  <int> <int>    <dbl>      <dbl>
## 1 male     168   343    0.490       49.0
## 2 female   175   343    0.510       51.0
```

Here's a faster alternative, using `table()`, `prop.table()`, and `mutate()`.

```
swls2019_only |>
  select(gender) |>
  table() |>
  prop.table() |>
  data.frame() |>   # force table into a data frame because
   ↪  mutate is incompatible with tables
  mutate(Percentage = Freq * 100,
         Rounded = round(Freq * 100, 2))  # you may round the
          ↪  results to 2 d.p. using the round() function
```

```
##   gender      Freq Percentage Rounded
## 1   male 0.4897959   48.97959   48.98
## 2 female 0.5102041   51.02041   51.02
```

4.5.2 Bar Plots with ggplot2

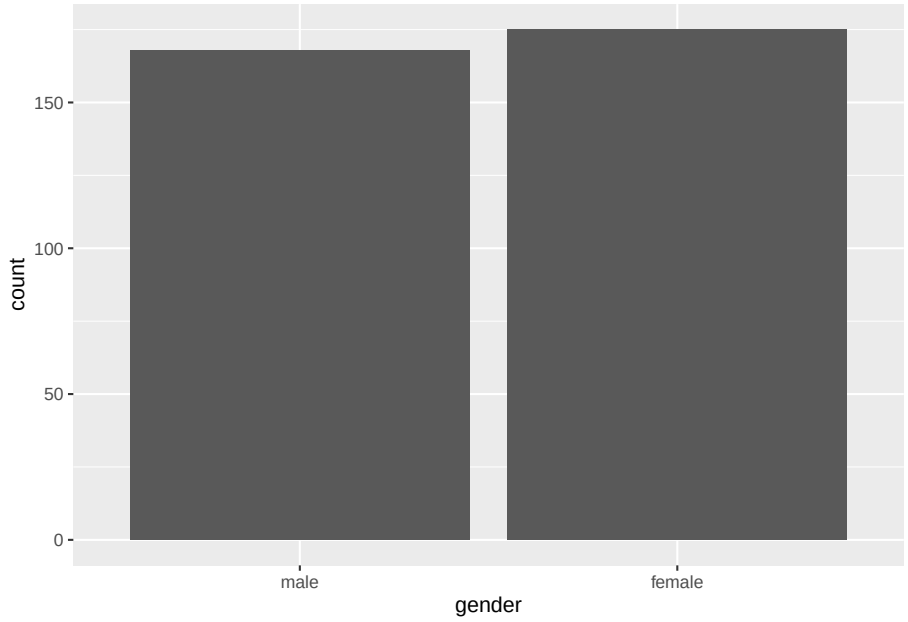
Instead of using a frequency table, maybe you want a bar chart or bar plot to visually represent the frequency distribution instead. (Note that bar charts or bar plots are used for categorical variables only.)

We will use `ggplot2`, a package developed for data visualization, which is part of the `tidyverse` collection. Every plot in `ggplot2` is plotted with three components: data, aesthetics, and geometry.

- Data: specifies the data frame
- Aesthetics: specifies the variables that we want to plot as well as aesthetics, like colour
- Geometry: specifies the type of plot & other modifications

These three components apply when we make any plot in ggplot2. Let's make a bar chart with `ggplot2` with the code below.

```
ggplot(data = swls2019_only, aes(x = gender)) +  # dataset used
 ↪  is swls2019_only, x (horizontal axis) is gender
  geom_bar(stat = "count")  # we want a bar plot where the
   ↪  statistic is the count of the values in x
```



Here's an alternative that will give you the exact same output.

```
# Alternative
swls2019_only |>  # dataset used is swls2019_only
  ggplot(.) +  # R takes swls2019_only in the previous line and
   ↪  places it in the . on this line to tell ggplot that dataset
   ↪  used is swls2019_only
  geom_bar(aes(x = gender), stat="count")  # we want a bar plot
   ↪  where x (horizontal axis) is gender and the statistic is
   ↪  the count of values in x
```

Notice that the `aes()` command can be either in the ggplot line or the geom function line. In this case, where you type it doesn't affect the results at all. (Try this out yourself to confirm!)

I personally prefer the alternative because it's easier for me to read, but it doesn't matter to me which you use so long as it works!

4.5.3 Histograms with ggplot2

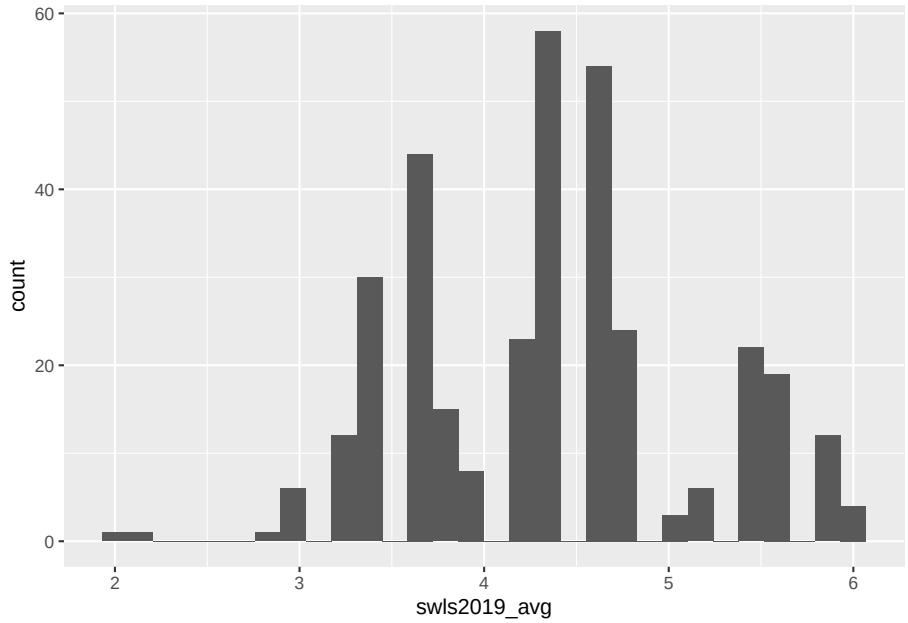
We create bar plots for categorical variables (such as gender). For continuous variables which can take on infinite values (such as time), we create histograms.

A histogram is a visual representation of the frequency table for a given continuous variable (i.e., it shows us the frequency distribution, or the frequency of scores for a given continuous variable).

Let's use `ggplot2` to make a histogram for `swls2019_avg` with a very basic code.

```
ggplot(data = swls2019_only, aes(x = swls2019_avg)) +  # tells R
 ↪  what dataset to use and what variable should be on the x axis
  geom_histogram()  # we want a histogram, so geom_histogram() is
   ↪  used
```

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



As with the bar chart, you could do this instead and the results would be the same:

```
# Alternative
swls2019_only |>
  ggplot() +
  geom_histogram(aes(x = swls2019_avg))
```

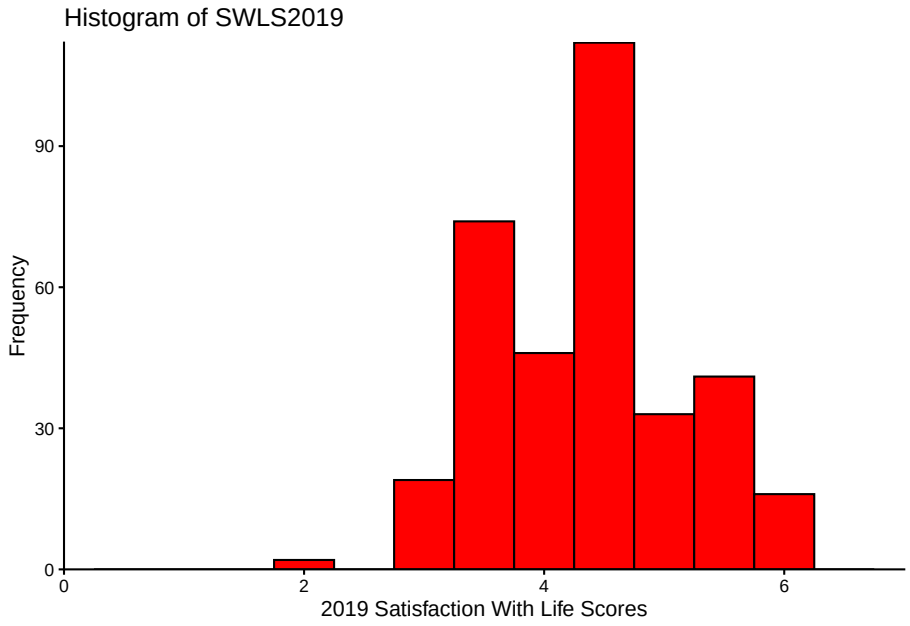
Now, the histogram that you got is okay, but kinda basic… Maybe you want to customise it. For example, you might want to add a specific label for the y-axis or the x-axis… Or maybe you want to change the colour of the graph! You can customise all of that! Here are some ideas.

```r
# Assign color "red" to the object barfill (which we will use
 ↪  below to change the colour of the bars).
barfill <- "red"

# Assign color "black" to the object barlines (which we will use
 ↪  below to change the colour of the barlines).
barlines <- "black"

# Specify the data frame and the variables to plot
ggplot(swls2019_only, aes(x = swls2019_avg)) +
# Specify the type of plot and binwidth
  geom_histogram(binwidth = 0.5,
# Specify the border of the bars
                 color = barlines,
# Specify the color of the bar
                 fill = barfill) +
# Label the x-axis
  scale_x_continuous(name = "2019 Satisfaction With Life Scores",
   ↪
# Force the graph to start at x = 0
                     expand = c(0, 0),
# Force the graph to end at x = 7
                     limits = c(0, 7)) +
# Label the y-axis
  scale_y_continuous(name = "Frequency",
# Force the graph to start at y = 0
                     expand = c(0, 0),
# No maximum limit to y (hence, NA)
                     limits = c(NA, NA)) +
# Give the histogram a title.
    ggtitle("Histogram of SWLS2019") +
# Choose a theme. I like theme_classic() for its clean, white
 ↪  background
    theme_classic()
```

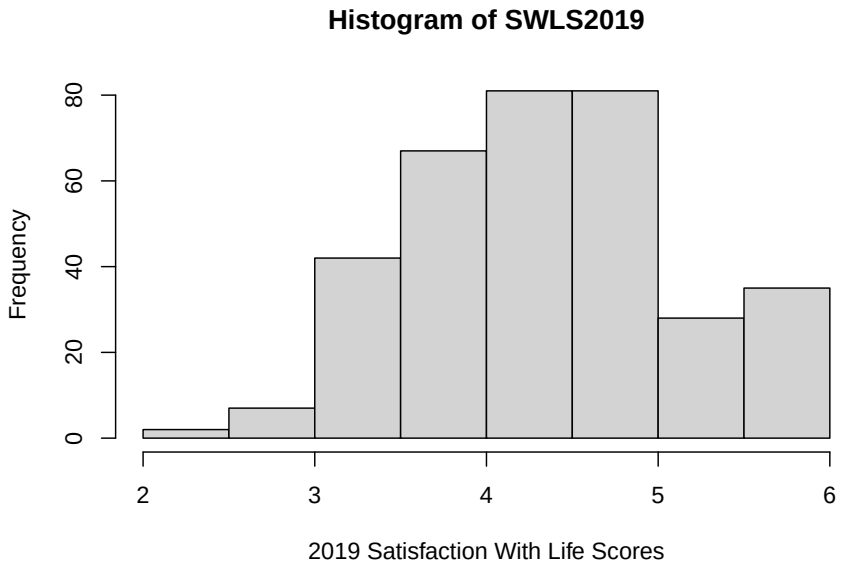
```
## Warning: Removed 2 rows containing missing values or values outside the scale
## range (`geom_bar()`).
```



Note that when you set limits on the graph, it throws up a warning: *Warning: Removed 2 rows containing missing values (`geom_bar()`).* This is a known issue, but is apparently too complex to resolve (see: https://github.com/tidyverse/ggplot2/issues/4083). You may safely ignore it if your limits are beyond the minimum and maximum values of your axes. For example, suppose your minimum value for the x-axis is 1 and your maximum value is 7. If you set the limits to 0 and 8, it should comprise all values in your dataset. You can find the minimum and maximum values using the `summarise()` function, or simply use `max()` or `min()`. As an aside, you may find other themes here: https://ggplot2.tidyverse.org/reference/ggtheme.html.

Now… I confess that `ggplot2` intimidates me. It's very powerful and can make super fancy graphs. But it has so many moving parts! So here's a simple code in base R that does not require `ggplot2`.

```
hist(x = swls2019_only$swls2019_avg,  # variable
     xlab = "2019 Satisfaction With Life Scores",  # label for
      ↪  x-axis
     ylab = "Frequency",  # label for y-axis
     main = "Histogram of SWLS2019")  # title for the histogram
```



4.5.4 Scatterplot with ggplot2

So far, we've been focusing on single variables (e.g., `gender` and `swls2019_avg`). What if we're interested in the relationship between two continuous variables? Here is where we might want to plot a scatterplot!

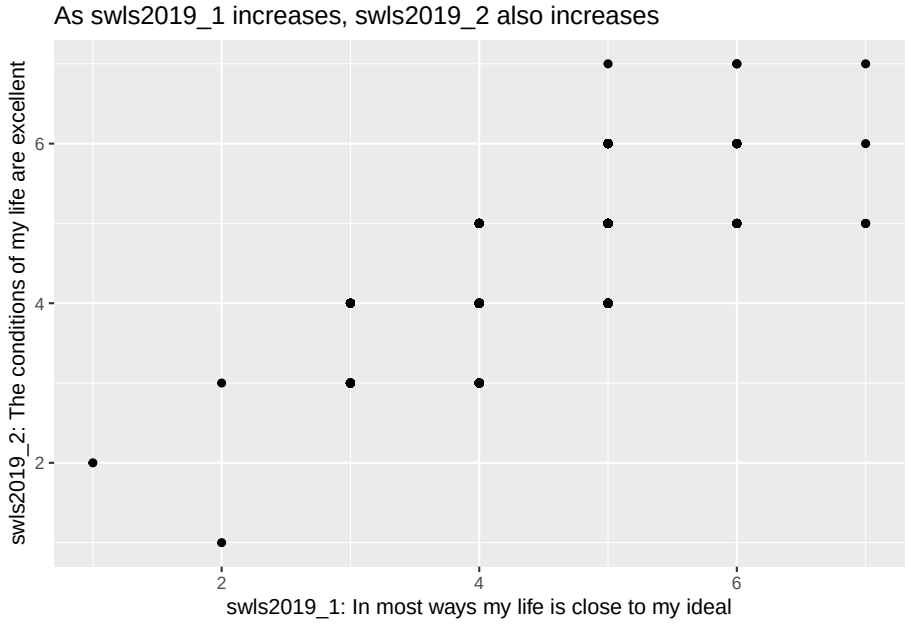
A scatterplot is a graph of pairs of values for each subject or individual. Specifically, each individual provides two observations, one for each variable. One of the variables will be plotted along the x-axis and the other along the y-axis.

So, suppose we're interested in plotting the relationship between two items from the SWLS2019 dataset: `swls2019_1` and `swls2019_2`. Why might we be interested in this? Well, the reasoning goes something like this: Both items supposedly measure satisfaction with life. So, if a participant scores higher on one item than another participant, they should also score higher on the other item. To confirm this, we can plot a scatterplot.

Here's the basic code for a scatterplot in `ggplot2`:

```
swls2019_only |>
  ggplot() +
  geom_point(aes(x = swls2019_1, y = swls2019_2)) +  # A
   ↪ scatterplot is a plot of points on a graph. So we use
   ↪ geom_point.
  labs(x = "swls2019_1: In most ways my life is close to my
   ↪ ideal",
```

```
       y = "swls2019_2: The conditions of my life are excellent",
        ↪
       title = "As swls2019_1 increases, swls2019_2 also
        ↪  increases")  # labs() is another way to add labels.
        ↪  You can also add a title using the labs() function,
        ↪  which by default is left-aligned.
```

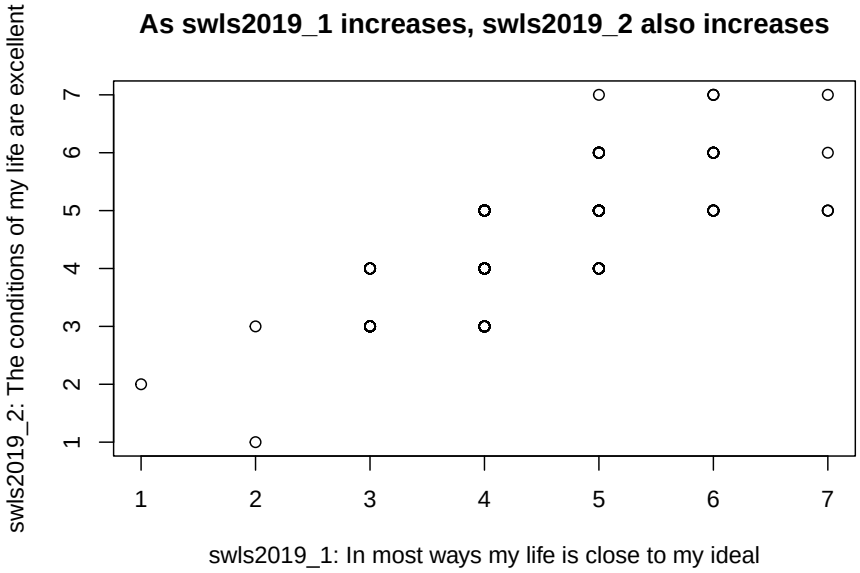


From the scatterplot, we see that indeed, as `swls2019_1` increases, `swls2019_2` increases.

Like the histogram, there are many different customisations beyond the basics I've shown you here. You can refer to this website for more: https://t-redactyl.io/blog/2016/02/creating-plots-in-r-using-ggplot2-part-5-scatterplots.html.

And like the histogram, there's a function in base r for plotting scatterplots.

```
plot(x = swls2019_only$swls2019_1,
     y = swls2019_only$swls2019_2,
     xlab = "swls2019_1: In most ways my life is close to my
      ↪  ideal",
     ylab = "swls2019_2: The conditions of my life are
      ↪  excellent",
     main = "As swls2019_1 increases, swls2019_2 also increases")
```



4.5.5 Last Words

I've only just scratched the surface of plots in this tutorial. For more, visit here: https://t-redactyl.io/tag/r-graphing-tutorials.html. Or here: https://r4ds.had.co.nz/data-visualisation.html. And here's a link to the ggplot2 cheatsheet: https://posit.co/wp-content/uploads/2022/10/data-visualization-1.pdf.

Now that you've gotten to this point, I think you're ready to do the next exercise! Proceed to the next section!

Chapter 5

Test Yourselves 2

Please read the Using tidyverse section before working on this exercise, especially if you have never used tidyverse before.

As before, suggested answers are at the bottom of this page, but please do try the exercise before looking at them. :)

5.1 Instructions

1. Download the data file here: SWB.csv.
2. Start an R Project in the folder where you saved the data file.
3. Open up a new script file in RStudio.
4. Load `tidyverse` and `psych` packages.
5. Read in the data file `SWB.csv`. Give this dataset the name `dataset`.
6. Examine `dataset`. Minimally you should be able to tell there are 343 rows and 23 columns. You should also be able to tell the variable names and the data type for each variable.
7. Convert `marital_status` and `have_children` from integer to factor. Add value labels to the factors. See the legend below to see how each variable is coded.
8. Check that the two variables are indeed converted to factors.
9. Create a subset of the dataset of only married parents (i.e., married people who have children). Further subset the dataset such that only the variables `swls2021_1`, `swls2021_2`, `swls2021_3`, `swls2021_4`, and `swls2021_5` are in the dataset. You should have 68 rows and 5 columns in this subset. Give this subset the name `subset`.
10. Using `subset`, compute the average of `swls2021_1`, `swls2021_2`, `swls2021_3`, `swls2021_4`, and `swls2021_5` for each person. Call this variable `swls2021_avg`.

11. Create a histogram for `swls2021_avg`. Although I'd be okay with just the basic histogram, I'd like you to play around with the different customisations! Change the bar fill color, change the range of the x-axis, make the background white, etc. (I just want you to explore and have fun here!)
12. Using `subset`, get the following summary statistics for `swls2021_avg`: maximum, minimum, mean, and sd. Specifically, use the `summarise()` function from `dplyr` package. Then confirm the results using the `describe()` function from the `psych` package. Check that the maximum is 5.4, the minimum is 2.6, the mean is 4.18, and the sd is 0.707.
13. Combine different functions from the `dplyr` package to find out the number of married parents who have `swls2021_avg` greater than or equal to 4.18. Check that the number is 36.

5.2 Legend

Variable Name	Variable Label	Value Label
pin	participant identification number	
gender	gender	0 = male, 1 = female
marital_status	marital status	1 = married, 2 = divorced, 3 = widowed
have_children	parental status	0 = no children, 1 = have children
mvs_1	My life would be better if I own certain things I don't have.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_2	The things I own say a lot about how well I'm doing.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_3	I'd be happier if I could afford to buy more things.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree

Variable Name	Variable Label	Value Label
mvs_4	It bothers me that I can't afford to buy things I'd like.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_5	Buying things gives me a lot of pleasure.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_6	I admire people who own expensive homes, cars, clothes.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_7	I like to own things that impress people.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_8	I like a lot of luxury in my life.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_9	I try to keep my life simple, as far as possessions are concerned. (R)	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
swls2019_1	In most ways my life is close to my ideal. (2019)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2019_2	The conditions of my life are excellent. (2019)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree

Variable Name	Variable Label	Value Label
swls2019_3	I am satisfied with my life. (2019)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2019_4	So far I have gotten the important things I want in life. (2019)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2019_5	In most ways my life is close to my ideal. (2019)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2021_1	In most ways my life is close to my ideal. (2021)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2021_2	The conditions of my life are excellent. (2021)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2021_3	I am satisfied with my life. (2021)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2021_4	So far I have gotten the important things I want in life. (2021)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree
swls2021_5	In most ways my life is close to my ideal. (2021)	1 = strongly disagree, 2 = disagree, 3 = slightly disagree, 4 = neither disagree nor agree, 5 = slightly agree, 6 = agree, 7 = strongly agree

5.3 Suggested Answers

```r
# Skipped Steps 1 to 3.

# 4. Load packages
library(tidyverse)
library(psych)

# 5. Read .csv file
dataset <- read_csv("SWB.csv")
# read.csv("SWB.csv") also okay

# 6. Examine the data
# Take a look at the variable names and data types
glimpse(dataset)  # str(dataset) or View(dataset) also okay

# 7. Convert integer to factor and add value labels to the
 ↪  factor.
dataset$marital_status <- factor(dataset$marital_status,
                                 levels = c(1, 2, 3),
                                 labels = c("married",
                                  ↪  "divorced", "widowed"))

dataset$have_children <- factor(dataset$have_children,
                                levels = c(0, 1),
                                labels = c("no children", "have
                                 ↪  children"))

# 8. Check variables are now factors
glimpse(dataset$marital_status)  # str(dataset$marital_status)
 ↪  also okay
glimpse(dataset$have_children) # str(dataset$have_children) also
 ↪  okay

# 9. Create subset
subset <- dataset |>
  filter(marital_status == "married" & have_children == "have
   ↪  children") |>
  select(swls2021_1:swls2021_5)

# 10. Compute swls2021_avg
subset <- subset |>
  mutate(swls2021_avg =
   ↪  rowMeans(across(c(swls2021_1:swls2021_5))))
```

```r
# 11. Create a histogram for swls2021_avg
ggplot(data = subset) +
  geom_histogram(aes(x = swls2021_avg)) +
  scale_x_continuous(expand = c(0, 0),
                     limits = c(0, 8),
                     # Every 0.5 units, a tick will appear on
                      ↪  x-axis
                     breaks = seq(0, 8, 0.5)) +
  scale_y_continuous(expand = c(0, 0),
                     # Every 1 unit, a tick will appear on y-axis
                     breaks = seq(0, 20, 1)) +
  labs(x = "Average 2021 Satisfaction With Life Scores",
       y = "Frequency",
       title = "Histogram of Average 2021 Satisfaction With Life
        ↪  Scores") +
  # Choose the theme (there are different themes to choose, just
   ↪  type theme and some suggestions will pop up)
  theme_classic()

# 12A. Get summary statistics using the summarise() function
# Maximum is 5.4, minimum is 2.6, mean is 4.18, and sd is 0.707.
subset |>
  summarise(avg_swls = mean(swls2021_avg),
            min_swls = min(swls2021_avg),
            max_swls = max(swls2021_avg),
            sd_swls = sd(swls2021_avg))

# 12B. Get summary statistics using the describe() function
# Maximum is 5.4, minimum is 2.6, mean is 4.18, and sd is 0.707.
describe(subset$swls2021_avg)

# 13A. Create a subset that only contains people with an average
 ↪  SWLS of 4.18 and above
# Then, count the number of people remaining
subset |>
  filter(swls2021_avg >= 4.18) |>
  summarise (total = n())

# 13B. Create a subset that only contains people with an average
 ↪  SWLS of 4.18 and above
# Then, count the number of people remaining
subset |>
  filter(swls2021_avg >= 4.18) |>
  nrow  # count the number of rows remaining
```

Chapter 6

Multi-Item Measures

6.1 Overview

This section guides you through the steps to deal with multi-item measures.

Typically, for constructs that are measured (e.g., manipulation check items, dependent variables), we use several items to assess the construct. Because each item is supposed to measure the same construct, instead of looking at each item separately, it is more efficient to combine participants' responses to these multiple items into a single score. This single score is then the variable that operationalises our construct. However, because participants do not always respond to items in the way that researchers intend, we should first check that the pattern of responses is consistent with our expectations (i.e., check which items are positively / negatively related to each other). We also need to check whether the items have adequate reliability. Then, we can create an average or a total score out of all the items of the scale.

6.2 Dataset

To illustrate how we deal with multi-item measures, I will use the hypothetical dataset `SWB.csv`, used in the `tidyverse` section earlier and also in Assignment 2. If you do not have a copy of the dataset, download it here: SWB.csv.

For this tutorial, we will only focus on the items measuring materialism. In this hypothetical study, materialism is measured with 9 items from the Materialism Values Survey (i.e., `mvs_1` to `mvs_9`). `mvs_1` to `mvs_8` are positively keyed items, meaning that higher scores should indicate greater endorsement of materialistic values. `mvs_9` is a reverse (or negatively) keyed item, meaning that higher scores on `mvs_9` should indicate lesser endorsement of materialistic values.

The specific wording of each item in the MVS is listed below.

Legend

Variable Name	Variable Label	Value Label
pin	participant identification number	
mvs_1	My life would be better if I own certain things I don't have.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_2	The things I own say a lot about how well I'm doing.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_3	I'd be happier if I could afford to buy more things.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_4	It bothers me that I can't afford to buy things I'd like.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_5	Buying things gives me a lot of pleasure.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_6	I admire people who own expensive homes, cars, clothes.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree

Variable Name	Variable Label	Value Label
mvs_7	I like to own things that impress people.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_8	I like a lot of luxury in my life.	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree
mvs_9	I try to keep my life simple, as far as possessions are concerned. (R)	1 = strongly disagree, 2 = disagree, 3 = neither disagree nor agree, 4 = agree, 5 = strongly agree

Let's start by reading in the dataset, then creating a subset of the dataset. We will keep `mvs_1` - `mvs_9` and also the participant identification number (`pin`) so that if there are any issues with the responses (e.g., if there is missing data), we know which participant that issue came from.

```
# Load package
library(tidyverse)

# Read in the data
dataset <- read_csv("SWB.csv")

# Create subset with the required variables and call it MVS
MVS <- dataset |>
  select(pin, mvs_1:mvs_9)

# Check that the subset gives you what you want
glimpse(MVS)
```

```
## Rows: 343
## Columns: 10
## $ pin   <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1~
## $ mvs_1 <dbl> 4, 4, 5, 2, 2, 3, 2, 2, 3, 2, 2, 3, 2, 2, 3, 3, 3, 4, 3, 3, 4, 4~
## $ mvs_2 <dbl> 3, 3, 4, 3, 2, 4, 3, 3, 4, 3, 2, 4, 1, 3, 2, 3, 2, 4, 3, 2, 4, 3~
## $ mvs_3 <dbl> 3, 3, 4, 3, 3, 4, 3, 3, 4, 3, 3, 4, 2, 2, 3, 4, 4, 5, 3, 3, 4, 4~
## $ mvs_4 <dbl> 3, 3, 3, 2, 2, 5, 4, 4, 3, 4, 4, 4, 2, 2, 4, 3, 3, 4, 4, 4, 5, 4~
```

```
## $ mvs_5 <dbl> 3, 3, 4, 3, 3, 4, 3, 3, 4, 2, 2, 3, 3, 3, 4, 4, 4, 5, 3, 3, 4, 4~
## $ mvs_6 <dbl> 4, 4, 5, 2, 2, 3, 4, 4, 5, 2, 2, 3, 3, 3, 4, 2, 2, 3, 4, 4, 5, 4~
## $ mvs_7 <dbl> 3, 2, 4, 2, 3, 3, 3, 3, 4, 3, 1, 4, 4, 1, 5, 1, 3, 2, 3, 3, 4, 3~
## $ mvs_8 <dbl> 4, 4, 5, 4, 4, 5, 3, 3, 4, 3, 3, 4, 2, 2, 3, 3, 3, 4, 4, 4, 5, 3~
## $ mvs_9 <dbl> 2, 2, 1, 3, 3, 2, 3, 3, 2, 4, 4, 3, 4, 4, 3, 3, 3, 2, 3, 3, 2, 3~
```

Before we combine the 9 items into one overall measure of materialism, though, we need to check a number of things.

6.3 Check the Pattern of Responses: Correlation Coefficient

First, we need to check the pattern of responses for our 9 items. We do this using the correlation coefficient.

The correlation coefficient, r, tells us if two variables are linearly related. In general, there are two directions, positive and negative. When two variables increase or decrease in the same direction (e.g., when one increases, the other also increases; when one decreases, the other also decreases), we call this a positive linear relationship. The correlation coefficient will be a positive number. When the two variables move in opposite directions (e.g., when one increases, the other decreases and vice versa), we call this a negative linear relationship. The correlation coefficient will be a negative number. The correlation coefficient always has a minimum value of -1 (i.e., variables are perfectly negatively correlated) and a maximum value of 1 (i.e., variables are perfectly positively correlated). Values beyond this range are impossible (because of the way r is calculated).

Because all the items of the Materialism Values Survey are supposed to measure the same construct, i.e., materialism, we expect positive-keyed items to correlate positively with each other and reverse-keyed items to correlate negatively with positive-keyed items. So, we should expect all items to be positively related to each other except for `mvs_9`.

To confirm our expectations, we need the correlations between each pair of items in the Materialism Values Survey.

```
# cor() tells R to calculate the correlations for each pair of
 ↪  items/variables in the dataframe
cor(MVS)
```

```
##                pin       mvs_1       mvs_2      mvs_3        mvs_4       mvs_5
## pin    1.000000000  0.01820791  0.00261561  0.0190207 -0.004592464  0.02088333
## mvs_1  0.018207909  1.00000000  0.43763755  0.3899635  0.350306014  0.56608026
```

```
## mvs_2  0.002615610  0.43763755  1.00000000  0.3956159  0.342974781  0.43869719
## mvs_3  0.019020703  0.38996352  0.39561590  1.0000000  0.358897021  0.50948105
## mvs_4 -0.004592464  0.35030601  0.34297478  0.3588970  1.000000000  0.26055546
## mvs_5  0.020883333  0.56608026  0.43869719  0.5094811  0.260555462  1.00000000
## mvs_6  0.029931812  0.67775570  0.28114358  0.3502713  0.455624190  0.44042258
## mvs_7  0.014597497  0.42771247  0.29644060  0.4019504  0.494644383  0.38935168
## mvs_8  0.012333886  0.57560443  0.39774601  0.2562786  0.361006990  0.62365310
## mvs_9 -0.031136263 -0.75002163 -0.43323415 -0.6384993 -0.384562433 -0.53922937
##             mvs_6      mvs_7       mvs_8       mvs_9
## pin    0.02993181  0.0145975  0.01233389 -0.03113626
## mvs_1  0.67775570  0.4277125  0.57560443 -0.75002163
## mvs_2  0.28114358  0.2964406  0.39774601 -0.43323415
## mvs_3  0.35027129  0.4019504  0.25627862 -0.63849929
## mvs_4  0.45562419  0.4946444  0.36100699 -0.38456243
## mvs_5  0.44042258  0.3893517  0.62365310 -0.53922937
## mvs_6  1.00000000  0.6200303  0.56460779 -0.74614447
## mvs_7  0.62003030  1.0000000  0.40459133 -0.55145900
## mvs_8  0.56460779  0.4045913  1.00000000 -0.64079832
## mvs_9 -0.74614447 -0.5514590 -0.64079832  1.00000000
```

Personally, I think the output is much easier to read if you coerce (force) it into a data frame with this code:

```
# Alternative
cor.output <- data.frame(cor(MVS))
```

You will find `cor.output` in the Environment tab (top right hand pane of RStudio). Click on it and you can read it like a normal data frame. Alternatively, you can use the `View()` function. Either way, you'll see a data frame, where each row by column cell shows the correlation between the row item and the column item (e.g., item `mvs_1` and item `mvs_2` have a correlation of .4525015).

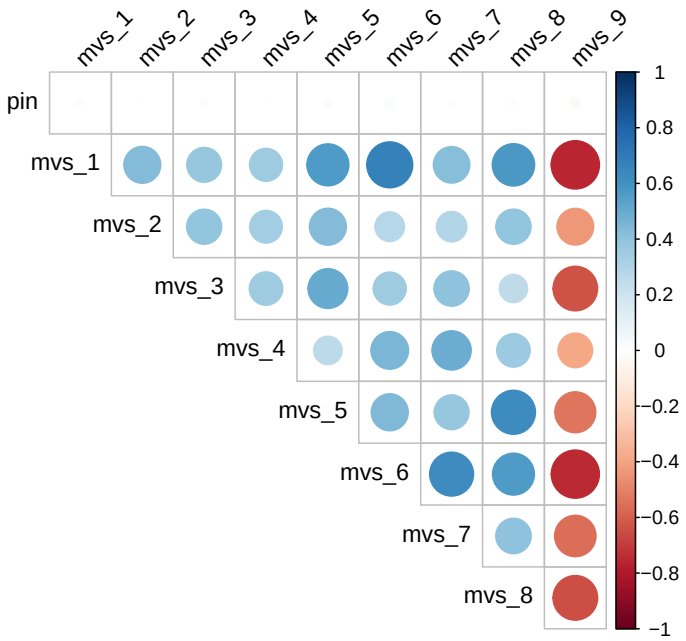
Of course, this alternative method only serves to make the output more readable. There is no difference in the actual result. Still, I think readability is important, especially when the original output is broken up into two rows, which can make it more difficult to understand.

Regardless which method you use, the output shows you that positive-keyed items correlate positively with each other (i.e., the correlations are positive for all pairs of positive-keyed items). Also, positive-keyed items correlate negatively with the reverse-keyed item. The correlations between `mvs_9` (in the rightmost column) and each of the other items are negative. These results indicate that participants are responding to the items as we expect.

I find that when there are too many variables, the correlation matrix might be a bit overwhelming. So, it might help to create a correlation plot using the `corrplot()` function from the `corrplot` package.

```
# Load package (remember to install package before loading
 ↪  package)
library(corrplot)

# Produce the correlation plot
corrplot(cor(MVS),
         type = "upper", # we only want the correlations
          ↪  reflected in the upper triangle (because the lower
          ↪  triangle and upper triangle will give us the same
          ↪  result, choosing one or the other helps reduce
          ↪  clutter)
         tl.col="black",  # text label colour = black
         tl.srt = 45,  # rotate the text label by 45 degrees for
          ↪  readability
         diag = FALSE)  # do not show the diagonal, which is
          ↪  essentially the correlations of the variables with
          ↪  themselves. Removing this will help reduce clutter
          ↪  in the output
```



Blue represents positive correlation, red represents negative correlation, and the size of the circles represent the strength of the correlation. Notice that the column for `mvs_9` are all red circles, which is indeed what we would expect, since `mvs_9` is the only reverse keyed item.

To be honest, I much prefer to first reverse code all the reverse keyed items

before getting the correlation coefficients. That way, it's much easier to spot a problem–the moment I see a negative sign, it's a problem. So, let's do that reverse coding now.

The simple formula to use to reverse code data is this: (max possible value on likert scale + min possible value on likert scale) - observation. If you refer to the legend, you'll see that the maximum possible value is 5 and the minimum possible value is 1 for the 9 items.

```
# Recode mvs_9 by creating a new variable called mvs_9r
MVS <- MVS |>
  mutate(mvs_9r = 5 + 1 - mvs_9)

# Check mvs_9 is correctly recoded for the first 6 rows
# A 1 on mvs_9 should be a 5 on mvs_9r, 2 should be a 4, and so
 ↪  on
MVS |>
  select(mvs_9, mvs_9r) |>
  head(6)
```

```
## # A tibble: 6 x 2
##   mvs_9 mvs_9r
##   <dbl>  <dbl>
## 1     2      4
## 2     2      4
## 3     1      5
## 4     3      3
## 5     3      3
## 6     2      4
```

Note. Notice that I labelled my recoded variable as `mvs_9r` (where r stands for recoded). Some people might choose to replace (overwrite) the variable instead of giving the recoded variable a new name. So they might do the following: `mutate(mvs_9 = 5 + 1 - mvs_9)`. I strongly recommend against this as it might confuse you later on–you cannot tell at a glance whether you have recoded the variable. You'll have to keep checking your code to confirm this, which would increase the probability of making errors. (Sadly, I know this from experience…)

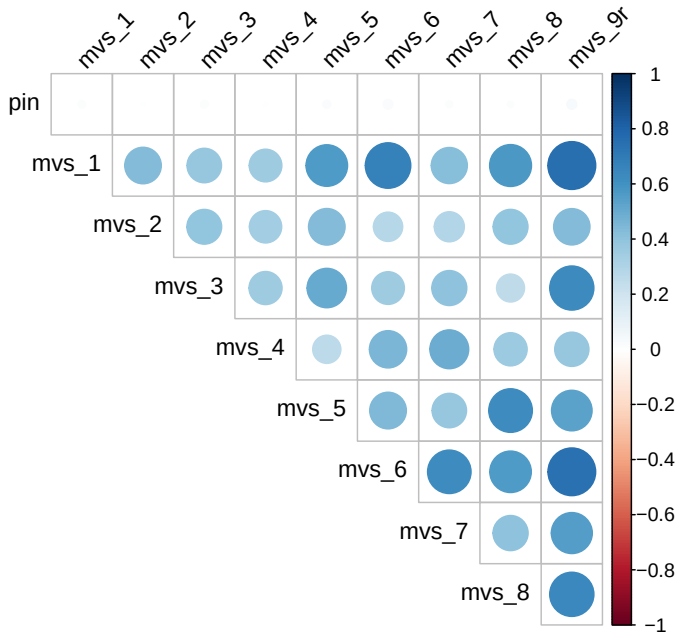
Once you have confirmed that you correctly recoded the variable, you can drop `mvs_9` by using `select()`. Then conduct the correlation analysis to look at the correlation matrix.

```
# Drop mvs_9
MVS <- MVS |>
  select(pin, mvs_1:mvs_8, mvs_9r)
```

```
# Examine the correlation matrix
cor(MVS)
```

```
##                pin      mvs_1      mvs_2     mvs_3        mvs_4      mvs_5
## pin     1.000000000 0.01820791 0.00261561 0.0190207 -0.004592464 0.02088333
## mvs_1   0.018207909 1.00000000 0.43763755 0.3899635  0.350306014 0.56608026
## mvs_2   0.002615610 0.43763755 1.00000000 0.3956159  0.342974781 0.43869719
## mvs_3   0.019020703 0.38996352 0.39561590 1.0000000  0.358897021 0.50948105
## mvs_4  -0.004592464 0.35030601 0.34297478 0.3588970  1.000000000 0.26055546
## mvs_5   0.020883333 0.56608026 0.43869719 0.5094811  0.260555462 1.00000000
## mvs_6   0.029931812 0.67775570 0.28114358 0.3502713  0.455624190 0.44042258
## mvs_7   0.014597497 0.42771247 0.29644060 0.4019504  0.494644383 0.38935168
## mvs_8   0.012333886 0.57560443 0.39774601 0.2562786  0.361006990 0.62365310
## mvs_9r  0.031136263 0.75002163 0.43323415 0.6384993  0.384562433 0.53922937
##             mvs_6     mvs_7      mvs_8     mvs_9r
## pin    0.02993181 0.0145975 0.01233389 0.03113626
## mvs_1  0.67775570 0.4277125 0.57560443 0.75002163
## mvs_2  0.28114358 0.2964406 0.39774601 0.43323415
## mvs_3  0.35027129 0.4019504 0.25627862 0.63849929
## mvs_4  0.45562419 0.4946444 0.36100699 0.38456243
## mvs_5  0.44042258 0.3893517 0.62365310 0.53922937
## mvs_6  1.00000000 0.6200303 0.56460779 0.74614447
## mvs_7  0.62003030 1.0000000 0.40459133 0.55145900
## mvs_8  0.56460779 0.4045913 1.00000000 0.64079832
## mvs_9r 0.74614447 0.5514590 0.64079832 1.00000000
```

```
# Produce the correlation plot
corrplot(cor(MVS), type = "upper", tl.col="black", tl.srt = 45,
 ↪  diag = FALSE)
```



Now that we recoded `mvs_9` into `mvs_9r`, you can see, all correlation coefficients here are positive and the correlation plot shows all blue circles. So, all is good!

Before we move on to the next bit, I should mention that `cor()` doesn't provide the p values for the correlations. If you would like the p values, then you should consider using `corr.test()` from the `psych` package instead.

6.4 Reliability Analysis: Cronbach's Alpha

Before averaging (or summing) the items to get an overall MVS score, we must demonstrate that the items have adequate reliability. Cronbach's alpha is one of the most common measure of internal consistency reliability. It's a number that ranges from 0 to 1 and indicates how well the items on the scale measure the same construct (in this case, materialism). An alpha of .70 is accepted by many researchers as adequate reliability. However, for certain purposes, an alpha of .60 to .70 can be accepted if justified (e.g., research is new and exploratory or focused on theory or items measure different facets of the construct). If reliability is too low, however, this might mean the responses were too inconsistent across items. Then it might not make sense to average them together. If this happens in your project, it would be an important limitation to discuss in the report.

We will use the `alpha()` function in the `psych` package to conduct the reliability analysis. Note that this function is used on the data frame with the reverse-

keyed items all reverse coded already. So, if you haven't reverse-coded `mvs_9` yet (see above for instructions), do so before continuing.

At this point, it should also be noted that `ggplot2` also has an `alpha()` function. If you have both `tidyverse` and `psych` loaded, R might be confused as to which package to use and throw up an error. You can specify the package like this: `package::function()`.

```
# First, ensure the dataset only has the items from the scale
 ↪  (meaning, remove pin and keep only the 9 items)
MVS <- MVS |>
  select(mvs_1:mvs_8, mvs_9r)

# tell R we want to use the alpha() function from the psych
 ↪  package
# the alpha function will give us the Cronbach's alpha for all
 ↪  the items in MVS.
psych::alpha(MVS)
```

Notice that we do not need to load the package using `library(psych)`. `psych::alpha()` temporarily loads the package (just for that chunk of code). If you want to use other functions in the `psych` package, then you'll probably want to load the package.

```
## Warning in response.frequencies(x, max = max): response.frequency has been
## deprecated and replaced with responseFrequecy.  Please fix your call
```

```
##
## Reliability analysis
## Call: psych::alpha(x = MVS)
##
##   raw_alpha std.alpha G6(smc) average_r S/N    ase mean   sd median_r
##       0.88      0.89    0.91      0.47 7.8 0.0098  3.4 0.68     0.44
##
##     95% confidence boundaries
##          lower alpha upper
## Feldt     0.86  0.88   0.9
## Duhachek  0.86  0.88   0.9
##
##  Reliability if an item is dropped:
##        raw_alpha std.alpha G6(smc) average_r S/N alpha se var.r med.r
## mvs_1       0.86      0.87    0.89      0.45 6.5    0.012 0.016  0.42
## mvs_2       0.88      0.89    0.91      0.49 7.7    0.010 0.019  0.48
## mvs_3       0.87      0.88    0.89      0.48 7.4    0.011 0.018  0.44
## mvs_4       0.88      0.89    0.91      0.49 7.7    0.010 0.018  0.44
```

```
## mvs_5       0.87      0.87    0.89      0.46 6.9    0.011 0.019  0.42
## mvs_6       0.86      0.87    0.89      0.45 6.6    0.012 0.015  0.42
## mvs_7       0.87      0.88    0.90      0.47 7.1    0.011 0.020  0.44
## mvs_8       0.87      0.87    0.89      0.46 6.9    0.011 0.018  0.44
## mvs_9r      0.85      0.86    0.87      0.43 6.1    0.012 0.013  0.40
##
##  Item statistics
##          n raw.r std.r r.cor r.drop mean   sd
## mvs_1  343  0.78  0.79  0.78   0.71  3.4 0.92
## mvs_2  343  0.63  0.62  0.54   0.51  2.9 1.08
## mvs_3  343  0.66  0.66  0.63   0.56  3.5 0.91
## mvs_4  343  0.63  0.61  0.54   0.51  3.3 1.00
## mvs_5  343  0.72  0.73  0.70   0.64  3.5 0.89
## mvs_6  343  0.78  0.79  0.78   0.71  3.7 0.89
## mvs_7  343  0.72  0.70  0.65   0.61  3.1 1.15
## mvs_8  343  0.73  0.74  0.72   0.65  3.6 0.85
## mvs_9r 343  0.86  0.87  0.89   0.82  3.6 0.79
##
## Non missing response frequency for each item
##           1    2    3    4    5 miss
## mvs_1  0.00 0.19 0.34 0.35 0.11    0
## mvs_2  0.10 0.25 0.34 0.24 0.07    0
## mvs_3  0.00 0.15 0.31 0.40 0.14    0
## mvs_4  0.00 0.28 0.28 0.32 0.12    0
## mvs_5  0.00 0.15 0.35 0.38 0.12    0
## mvs_6  0.00 0.11 0.26 0.45 0.17    0
## mvs_7  0.11 0.17 0.35 0.25 0.12    0
## mvs_8  0.00 0.10 0.33 0.43 0.14    0
## mvs_9r 0.00 0.06 0.36 0.44 0.13    0
```

The `Reliability analysis` section of the output tells us that the Cronbach's alpha is .88 (`raw_alpha`). The last two lines tell us that the 95% confidence interval surrounding the Cronbach's alpha is 95% CI [.86, .90]. That is, we can be 95% confident that this interval contains the population Cronbach's alpha. We can ignore the other parts in this section as they are not relevant and look at the next section.

As a separate note, Feldt and Duhachek are different ways of calculating the 95% confidence interval. (Feldt only considers mean covariances whereas Duhachek considers variance of the covariances.) The numbers for both methods should match for larger sample sizes. For smaller sample sizes, the numbers might differ. If they do, there is currently no fixed rule of thumb. So, for this class, if the numbers differ, report the one for Duhachek. It is likely to give you more accurate confidence intervals given that it also considers variance of the covariances. (If you are very concerned about confidence intervals, you might instead do bootstrapping, which is also available in the `psych` package under

`n.iter`.)

The `Reliability if an item is dropped` section is especially helpful if we are looking at a scale that we have developed (i.e., NOT an established, well-validated scale that has been published), and are considering which items to exclude from our final scale.

The table's second column, `raw_alpha`, tells you what the Cronbach's alpha would be if we re-ran the Reliability Analysis excluding that item. For example, if we computed the reliability without `mvs_1` (first row), the Cronbach's alpha would drop from .88 (from the `Reliability analysis` section) to .86. This means that the scale becomes less reliable when we remove `mvs_1`. So, we want to keep this item in our final scale. If deleting an item increases the overall Cronbach's alpha (in this case, if the raw_alpha is larger than .88), then it means that the scale becomes more reliable when we remove the item. So, we would probably exclude the item from our final scale. In that case, re-analyse the data excluding that item. Repeat this process until there are no more items with a raw_alpha value greater than the Cronbach's alpha for the overall scale. Take note of which items you excluded because you will need to exclude them when calculating the average or the total score.

If the scale is obtained from a published source (as the MVS is), we do NOT exclude items. This is because we want to be able to compare our results with previous research that has used the same scale (i.e., with all the items). Excluding items means our results will not be directly comparable anymore. However, the `Reliability if an item is dropped` table is still helpful because it tells us if each item is contributing to the overall scale in the way we expect. If the items are not working in the way we expect, this would be an important limitation to discuss in the paper.

We will ignore the `Item statistics` section and go straight to the section on `Non missing response frequency for each item`.

This is a relative frequency table for all the items in the scale. The most important thing to note is whether there are any missing data for the items (see the column `miss`). In this case, there were no missing data. So hooray!

6.5 Compute the average score for the scale

Now that we're sufficiently satisfied with the reliability of the scale, let's compute the average score.

```
MVS <- MVS |>
  mutate(MVS_avg = rowMeans(across(c(mvs_1:mvs_8, mvs_9r)))) #
    ↪ calculate the average MVS score across the set of 9 items
    ↪ and call the average MVS_avg
```

You might want to check if `MVS_avg` is correctly calculated. Go to `View(MVS)` and manually calculate the average of `mvs_1` to `mvs_9r` for the first row. You should get 3.44 (to 2 d.p.) which should match the value in the `MVS_avg` column.

Of course, you could create `MVS_avg` by adding all the items together (`mvs_1 + mvs_2 + ... + mvs_9r`) and dividing that total by 9. I think that's a bit tedious, but some people are intimidated by `rowMeans()`. So, do what makes more sense for you!

Congratulations on completing this section! You're now ready to try the next exercise! Proceed to the next section!

Chapter 7

Test Yourselves 3

Please read through the Multi-Item Measures section before working on this exercise.

As before, suggested answers are at the bottom of this page, but please do try the exercise before looking at them. :)

7.1 Instructions

1. Download the data file here: WVS2012 Singapore BFI10 Data.csv.This dataset contains the responses of 1972 Singaporean participants to the 10-item Big Five Inventory (BFI10) in the 2012 World Values Survey. The BFI10 uses 2 items to measure each personality trait: extraversion, agreeableness, conscientiousness, openness to experience, and neuroticism. One item is positively-keyed and the other is reverse-keyed (R). See the legend below for more information.
2. Start an R Project in the folder where you saved the data file.
3. Open up a new script file in RStudio.
4. Load `tidyverse` and `psych` packages.
5. Read in the data file `WVS2012 Singapore BFI10 Data.csv`. Name this data frame `dataset`.
6. Examine `dataset`. Minimally you should be able to tell there are 1972 rows and 10 columns. You should also be able to tell the variable names and the data type for each variable.
7. Create a subset of the data by removing all participants who had values below 1 for ANY of the BFI10 items (e.g., -5, -2). Name this subset `subset`. (Hint: Use the `filter()` function in combination with the `if_all()` function. You'll need to read up a bit on the internet to figure out how they work and then experiment.) This is probably the hardest part of this ex-

ercise. But if all goes well, you should have 1963 rows (participants) in your subset.

8. Using `subset`, reverse-code all the reverse-keyed items (i.e., V160A, V160C, V160D, V160E, V160G).
9. Get the correlation coefficient for each pair of items for the same personality trait. Check that your answer matches mine: $r = 0.1177$ (extraversion), $r = 0.0636$ (agreeableness), $r = 0.0921$ (conscientiousness), $r = 0.0478$ (neuroticism), and r = -0.2959 (openness to experience). At this point, you should be surprised that openness to experience is negatively related, especially because we had already re-coded the reverse-keyed items!
10. Get the Cronbach's alpha for each pair of items for the same personality trait. Check that your answer matches mine: = 0.21 (extraversion), = 0.12 (agreeableness), = 0.17 (conscientiousness), = 0.09 (neuroticism), and = ERROR or -0.84 because items were negatively correlated (openness to experience). This suggests that the BFI10 has low reliability for the Singapore data. This finding mirrors findings elsewhere which suggest that the BFI10 has low reliability for non WEIRD (Western, Educated, Industrial, Rich, and Democratic) countries (see: https://renebekkers.wordpress.com/2017/03/21/hunting-game-targeting-the-big-five/).

7.2 Legend

Variable Name	Variable Label	Construct	Value Label
V160A	I see myself as someone who: is reserved	Extraversion (R)	-5 = Missing, Unknown; -4 = Not asked in survey; -3 = Not applicable; -2 = No answer; -1 = Don't know; 1 = Disagree strongly; 2 = Disagree a little; 3 = Neither agree nor disagree; 4 = Agree a little; 5 = Agree strongly
V160B	I see myself as someone who: is generally trusting	Agreeableness	-5 = Missing, Unknown; -4 = Not asked in survey; -3 = Not applicable; -2 = No answer; -1 = Don't know; 1 = Disagree strongly; 2 = Disagree a little; 3 = Neither agree nor disagree; 4 = Agree a little; 5 = Agree strongly

Variable Name	Variable Label	Construct	Value Label
V160C	I see myself as someone who: tends to be lazy	Conscientiousness (R)	-5 = Missing, Unknown; -4 = Not asked in survey; -3 = Not applicable; -2 = No answer; -1 = Don't know; 1 = Disagree strongly; 2 = Disagree a little; 3 = Neither agree nor disagree; 4 = Agree a little; 5 = Agree strongly
V160D	I see myself as someone who: is relaxed, handles stress well	Neuroticism (R)	-5 = Missing, Unknown; -4 = Not asked in survey; -3 = Not applicable; -2 = No answer; -1 = Don't know; 1 = Disagree strongly; 2 = Disagree a little; 3 = Neither agree nor disagree; 4 = Agree a little; 5 = Agree strongly

Variable Name	Variable Label	Construct	Value Label
V160E	I see myself as someone who: has few artistic interests	Openness to experience (R)	-5 = Missing, Unknown; -4 = Not asked in survey; -3 = Not applicable; -2 = No answer; -1 = Don't know; 1 = Disagree strongly; 2 = Disagree a little; 3 = Neither agree nor disagree; 4 = Agree a little; 5 = Agree strongly
V160F	I see myself as someone who: is outgoing, sociable	Extraversion	-5 = Missing, Unknown; -4 = Not asked in survey; -3 = Not applicable; -2 = No answer; -1 = Don't know; 1 = Disagree strongly; 2 = Disagree a little; 3 = Neither agree nor disagree; 4 = Agree a little; 5 = Agree strongly

Variable Name	Variable Label	Construct	Value Label
V160G	I see myself as someone who: tends to find fault with others	Agreeableness (R)	-5 = Missing, Unknown; -4 = Not asked in survey; -3 = Not applicable; -2 = No answer; -1 = Don't know; 1 = Disagree strongly; 2 = Disagree a little; 3 = Neither agree nor disagree; 4 = Agree a little; 5 = Agree strongly
V160H	I see myself as someone who: does a thorough job	Conscientiousness	-5 = Missing, Unknown; -4 = Not asked in survey; -3 = Not applicable; -2 = No answer; -1 = Don't know; 1 = Disagree strongly; 2 = Disagree a little; 3 = Neither agree nor disagree; 4 = Agree a little; 5 = Agree strongly

Variable Name	Variable Label	Construct	Value Label
V160I	I see myself as someone who: gets nervous easily	Neuroticism	-5 = Missing, Unknown; -4 = Not asked in survey; -3 = Not applicable; -2 = No answer; -1 = Don't know; 1 = Disagree strongly; 2 = Disagree a little; 3 = Neither agree nor disagree; 4 = Agree a little; 5 = Agree strongly
V160J	I see myself as someone who: has an active imagination	Openness to experience	-5 = Missing, Unknown; -4 = Not asked in survey; -3 = Not applicable; -2 = No answer; -1 = Don't know; 1 = Disagree strongly; 2 = Disagree a little; 3 = Neither agree nor disagree; 4 = Agree a little; 5 = Agree strongly

7.3 Suggested Answers

```
# Skipped Steps 1 to 3

# 4. Load packages
library(tidyverse)
library(psych)

# 5. Read .csv file
dataset <- read_csv("WVS2012 Singapore BFI10 Data.csv")

# 6. Take a look at the variable names and types
glimpse(dataset)  #str(data) also okay

# Extra. Run descriptives to get min and max values for each
 ↪  column
# I didn't get you guys to do this, but in actual analysis, you
 ↪  should run descriptives to check what the minimum and maximum
 ↪  values are for this dataset.
# Values below 1 are invalid numbers (see legend) so we need to
 ↪  remove them
# Notice that the minimum value is -5 (because there are negative
 ↪  values)
dataset |>
  describe(.)

# 7. Remove all participants who have values below 1 for the
 ↪  BFI10 items.
subset <- dataset |>
  filter(if_all(c("V160A":"V160J"), ~ .x > 0))
# The code above is saying...
# KEEP (filter) the row
# IF ALL (if_all) of the
# COLUMNS from V160A to V160J (c("V160A":V160J") have
# VALUES GREATER THAN 0 (.x > 0).
# Note. If you have multiple conditions, you can just add them
 ↪  on... For example greater than 0 and less than 6 would be ~.x
 ↪  > 0 & .x < 6

# Extra. Run descriptives to check the min and max values for
 ↪  each column
# If you did the above step correctly, the minimum value should
 ↪  be 1 now, which is the lowest value on the likert scale used
subset |>
  describe(.)
```

```r
# 8. Reverse-Code all the Reverse-Keyed Items
# Then, select only the variables that you are interested in
 ↪  analyzing in the subset
subset <- subset |>
  mutate(5 + 1 - across(c(V160A, V160C, V160D, V160E, V160G),
                        .names = "{col}r")) |>
  select(V160Ar, V160F, V160B, V160Gr, V160Cr, V160H, V160Dr,
   ↪  V160I, V160Er, V160J)

# The above code is saying...
# Mutate (i.e., create new variables) by taking 5 (max) + 1 (min)
 ↪  - observation, and do this across this set of columns: V160A,
 ↪  V160C, V160D, V160E, V160G. Name those columns by adding an r
 ↪  after the original column name (represented by {col}r)
# Then, select the columns you'd like to keep

# 9. Get the correlation coefficients
# I present two alternatives below.

# Alternative 1: Correlation Matrix
# This is an acceptable method, although I find it difficult to
 ↪  pick out the important correlations from here because the
 ↪  matrix is a bit large.
cor.output <- data.frame(cor(subset))

# Alternative 2: Individual correlations
# This takes a bit more effort, but clearly indicates which
 ↪  variable we're analyzing.
cor(subset$V160Ar, subset$V160F) # extraversion
cor(subset$V160B, subset$V160Gr) # agreeableness
cor(subset$V160Cr, subset$V160H) # conscientiousness
cor(subset$V160Dr, subset$V160I) # neuroticism
cor(subset$V160Er, subset$V160J) # openness to experience

# 10. Cronbach's alpha for each pair of items
# Here's a general explanation of the code below.
# We take the data frame, subset. Then we keep only the variables
 ↪  we're interested in (e.g., V160Ar, V160F). This results in a
 ↪  data frame with two variables. This two-variable data frame
 ↪  is then passed into the alpha() function.
# Note that alpha() can only be used with data frames or matrices
 ↪  and not with vectors. So alpha(subset$V160Ar, subset$V160F)
 ↪  won't work since each column (e.g., subset$V160F) is a vector
 ↪  (think of vector as a list of values of the same type) and
 ↪  not a data frame.
```

```
# Extraversion
subset |>
  select(V160Ar, V160F) |>
  alpha()

# Agreeableness
subset |>
  select(V160Gr, V160B) |>
  alpha()

# Conscientousness
subset |>
  select(V160Cr, V160H) |>
  alpha()

# Neuroticism
subset |>
  select(V160Dr, V160I) |>
  alpha()

# Openness to Experience
subset |>
  select(V160Er, V160J) |>
  alpha()
```

Chapter 8

Hypothesis Testing

8.1 Overview

The bulk of what we cover in PSYC208 is hypothesis testing. This section guides you through the steps of conducting the various hypothesis tests. This only serves as an overview for students who are interested to read ahead. I will demonstrate how to conduct the hypothesis tests in class as well as how to interpret the results.

To illustrate the various tests, we will use the same hypothetical dataset, `SWB.csv` as before. To follow along, please download the dataset: SWB.csv.

8.2 Load Packages and Dataset

```
# Load packages
library(tidyverse)
library(psych)

# Read in the dataset
# Call the dataset df
df <- read_csv("SWB.csv")
```

8.3 Scientific Notation

In addition to loading the packages and reading in the data file, it might also be helpful to run the code below to turn off scientific notation.

```
options(scipen = 999)  # Turn off scientific notation
options(digits = 9)  # Display results to 9 decimal places
```

Scientific notation is a way of expressing numbers that are very large or very small. It usually takes the form of `m × 10^n` for very large numbers and `m × 10^-n` for very small numbers, where `^` stands for "to the power of". So, suppose we have the following number: 0.000000000000000477. In scientific notation, this would be represented as`4.77 x 10^-16` (i.e., 16 decimal places). Alternatively, it might be represented as `4.77e-16`. They mean the same thing.

Here's a quick question to see if you understood scientific notation.

Which of the following is the scientific notation for 0.0123?

A) 1.23e-1 / 1.23 x 10^-1
B) 1.23e-2 / 1.23 x 10^-2
C) 1.23e-3 / 1.23 x 10^-3
D) 1.23e-4 / 1.23 x 10^-4

(Answer after the next paragraph because I couldn't find a better way to hide the answer!)

By default, R presents very small or very large numbers in scientific notation. However, this can be difficult to understand for people who don't use scientific notation in their daily lives. I confess I have difficulty with scientific notation myself so I usually turn the scientific notation off. It's really a matter of personal choice whether you want to turn it off or not, though! Just thought you'd like an option :)

```
# Answer to the quiz above is Option B: 1.23e-2 / 1.23 x 10^-2.
```

And now, let's begin with the hypothesis tests!

8.4 One-Sample T Test

8.4.1 When to Use A One-Sample T Test

We use a one-sample t test when we want to compare the data from one group to some hypothesised mean. You can think of it as taking a set of observations or scores and then comparing it to some value.

So, suppose we're interested to know whether on average, in 2019, people were more or less satisfied with their lives than the neutral value of 4. (Why 4? Well, in this data set, the satisfaction with life items were measured on a 7-point scale, where 4 is the neutral point.) To answer this question, we will conduct a one-sample t test.

8.4.2 Conducting and Interpreting the One-Sample T Test

Before jumping into any hypothesis testing, though, it is good practice to first get the descriptive statistics for the variable(s) you're investigating. At the minimum, you should get the mean and the standard deviation. (Anyway, you need to report means and standard deviations in APA style write-up!) I often also like to look at the minimum and maximum values to make sure the values are not out of ordinary (e.g., a value of -999 might throw up some red flags if, say, the likert scale only comprises values from 1 to 7).

```
# Calculate the average satisfaction with life score in 2019 for
 ↪  each participant
df <- df |>
  mutate(swls2019_avg =
   ↪  rowMeans(across(c(swls2019_1:swls2019_5))))

# Get the descriptive statistics
describe(df$swls2019_avg)
```

```
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 343 4.35 0.77    4.4    4.33 0.89   2   6     4  0.1    -0.48 0.04
```

Then, we can conduct the t test.

```
# Conduct the one-sample t test to
# compare the satisfaction with life score in 2019 against
# the neutral point of 4
t.test(x = df$swls2019_avg,  # variable we're analysing
       mu = 4)  # value we're comparing against
```

```
##
##  One Sample t-test
##
## data:  df$swls2019_avg
## t = 8.530937, df = 342, p-value = 0.00000000000000000477477
## alternative hypothesis: true mean is not equal to 4
## 95 percent confidence interval:
##  4.27188244 4.43482310
## sample estimates:
##  mean of x
## 4.35335277
```

```
# By default, R assumes a two-tailed test,
# that each observation comes from different individuals (i.e.,
 ↪  they are not "paired"),
# and that the alpha level is .05, therefore confidence interval
 ↪  is 95%
```

The results tell you that the average of `swls2019_avg` is 4.35 (but notice it doesn't tell you any other descriptive statistics like standard deviation or maximum, so it's still important to get the descriptives in other ways).

When we compare 4.35 to the neutral point of 4, the resulting t value is 8.53. With degrees of freedom 342, the p value is very small, at 0.00000000000000000477477. Because p value is smaller than the alpha level of .05, the result is statistically significant. Because 4.35 is statistically significantly greater than 4, we would conclude that there is sufficient evidence that on average, the satisfaction with life scores in 2019 is greater than the neutral value of 4.

In addition, the 95% confidence interval is [4.27, 4.43], which is interpreted as follows: We are 95% confident that the population mean 2019 satisfaction with life score is between 4.27 and 4.43.

8.4.3 Effect Size: Cohen's d

In addition to hypothesis testing, we also need to look at (standardised) effect size. Effect sizes tell you how large the effect (or the difference) is. The effect size we usually calculate for t tests is the Cohen's d. By convention, in psychology, $d = 0.2$ is considered a small effect size, $d = 0.5$ is considered a medium effect size, and $d = 0.8$ is considered a large effect size.

We can calculate Cohen's d using the `cohens_d()` function from the `effectsize` package.

```
# Load package (remember to install package before loading
 ↪  package)
library(effectsize)
```

```
## Warning: package 'effectsize' was built under R version 4.5.3
```

```
# Cohen's d, with mu = 4 (i.e., the comparison value is 4)
cohens_d(x = df$swls2019_avg,
         mu = 4)
```

```
## Cohen's d |       95% CI
## ------------------------
## 0.46      | [0.35, 0.57]
##
## - Deviation from a difference of 4.
```

Given the Cohen's d is 0.46, we can consider this a small to medium effect size. Meaning, there is a small to medium difference between the average 2019 satisfaction with life scores and the neutral point of 4.

8.5 Correlated Groups T test

8.5.1 When to Use A Correlated Groups T Test

We use a correlated groups t test when we want to compare two sets of data to see if they are different from each other. Importantly, the two sets of data are paired or correlated in some way (e.g., they come from the same person).

Suppose we want to know from our hypothetical dataset whether people's satisfaction with life scores changed from 2019 to 2021. We have (or can calculate) the same individuals' satisfaction with life scores in 2019 and in 2021. Given the same individual provided the 2019 and the 2021 scores, the two sets of scores are considered paired or correlated.

To find out whether there is a difference between these two sets of (paired) scores, we should therefore conduct a correlated groups t test.

8.5.2 Conducting and Interpreting the Correlated Groups T Test

Again, before conducting the correlated groups t test, remember to get the descriptive statistics for each set of scores.

```r
# Calculate the average satisfaction with life score in 2021 for
 ↪  each participant.
# If you did not calculate the average satisfaction with life
 ↪  score in 2019 earlier, you will want to do that before
 ↪  continuing.
df <- df |>
  mutate(swls2021_avg =
   ↪  rowMeans(across(c(swls2021_1:swls2021_5))))

# Get the descriptives
describe(df$swls2019_avg)
```

```
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 343 4.35 0.77    4.4    4.33 0.89   2   6     4  0.1    -0.48 0.04
```

```
describe(df$swls2021_avg)
```

```
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 343 3.85 0.67    3.8    3.84 0.89   2 5.8   3.8  0.1    -0.37 0.04
```

From the descriptive statistics results, you'll notice that the 2019 satisfaction with life scores appear to be higher than that for the 2021 satisfaction with life scores. But we need to conduct the correlated groups t test to find out if this difference is statistically significant.

```
# Conduct the correlated groups t test to
# see if there is any significant difference between
# the satisfaction with life scores in 2019 and in 2021.

t.test(x = df$swls2019_avg,  # scores in 2019
       y = df$swls2021_avg,  # scores in 2021
       mu = 0,  # value we expect if null hypothesis of no
        ↪  difference is true
       paired = TRUE)  # observations are paired, meaning each
        ↪  individual offered pairs of observations (i.e.,
        ↪  swls2019_avg and swls2021_avg)
```

```
##
##  Paired t-test
##
## data:  df$swls2019_avg and df$swls2021_avg
## t = 27.82957, df = 342, p-value < 0.000000000000000222
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  0.46493219 0.53565090
## sample estimates:
## mean difference
##     0.500291545
```

```
# By default, R assumes a two-tailed test,
# and that the alpha level is .05, therefore confidence interval
 ↪  is 95%.
# Since observations are paired, we need to specify that using
 ↪  paired = TRUE.
```

The results indicate that the average of the difference between swls2019_avg and swls2021_avg is 0.50. When we compare this difference against 0, the resulting t value is 27.83. With degrees of freedom 342, the p value is very small, at 0.000000000000000222. Because p value is smaller than the alpha level of .05, the result is statistically significant. Because 4.35 (mean for 2019) is statistically significantly greater than 3.85 (mean for 2021), we would conclude that there is sufficient evidence that on average, the satisfaction with life scores in 2019 are higher than the satisfaction with life scores in 2021.

The 95% confidence interval is [0.46, 0.54], which is interpreted as follows: We are 95% confident that the population mean difference between the 2019 and 2021 satisfaction with life scores is between 0.46 and 0.54.

8.5.3 Effect Size: Cohen's d

Finally, we need to get the effect size, Cohen's d, for correlated groups t test using the `cohens_d()` function from the `effectsize` package.

```
# Cohen's d with mu = 0
cohens_d(x = df$swls2019_avg,
         y = df$swls2021_avg,
         mu = 0,
         paired = TRUE)
```

```
## For paired samples, 'repeated_measures_d()' provides more options.
```

```
## Cohen's d |       95% CI
## ------------------------
## 1.50      | [1.35, 1.66]
```

Given the Cohen's d is 1.50, we can consider this a large effect size. In other words, there is a large difference between the 2019 and 2021 satisfaction with life scores.

8.6 Independent Groups T Test

8.6.1 When to Use An Independent Groups T Test

We use an independent groups t test when we want to compare two sets of data to see if they are different from each other. Importantly, the two sets of data are independent (i.e., they are NOT paired).

Suppose we want to know whether men and women differ in satisfaction with life in 2019. In other words, we're comparing the satisfaction with life scores for the group of men with that for the group of women. In this case, because the men and women are not paired with each other in any way, we consider the two sets of satisfaction with life scores as independent.

Therefore, to answer our research question, we will conduct an independent groups t test.

8.6.2 Conducting and Interpreting the Independent Groups T Test

Before conducting the independent groups t test, we need to convert the grouping variable into a factor. Here, the grouping variable is gender because we're comparing the male group against the female group.

After converting the grouping variable into a factor, we also want to get the descriptive statistics for each group.

```
# First, we need to convert gender into a factor and add the
 ↪  labels accordingly
df$gender <- factor(df$gender,
                    levels = c(0, 1),
                    labels = c("male", "female"))

# Next, we get the descriptive statistics for each group, male
 ↪  and female
describeBy(df$swls2019_avg,
           group = df$gender)
```

```
##
##  Descriptive statistics by group
## group: male
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 168 4.08 0.61    4.2    4.06 0.59   3 5.8   2.8 0.26    -0.46 0.05
## ------------------------------------------------------------
## group: female
##    vars   n mean   sd median trimmed  mad min max range  skew kurtosis   se
## X1    1 175 4.62 0.81    4.6    4.65 0.89   2   6     4 -0.35    -0.25 0.06
```

The descriptive statistics tell us that on average, the males have lower satisfaction with life scores than females. To find out whether this difference is statistically significant, we need to conduct the independent groups t test.

```r
# We conduct the independent groups t test to
# determine if men and women differ in
# satisfaction with life in 2019
t.test(df$swls2019_avg ~ df$gender,  # DV (outcome) ~ IV (group)
       mu = 0,   # value we expect if null hypothesis of no
        ↪  difference is true
       var.equal = TRUE)  # homogeneity of variance assumption is
        ↪  assumed to be met (if homogeneity of variance is
        ↪  violated, use var.equal = FALSE and R will conduct
        ↪  Welch corrections)
```

```
##
##  Two Sample t-test
##
## data:  df$swls2019_avg by df$gender
## t = -6.898557, df = 341, p-value = 0.0000000000256373
## alternative hypothesis: true difference in means between group male and group female is not eq
## 95 percent confidence interval:
##  -0.689132523 -0.383343668
## sample estimates:
##   mean in group male mean in group female
##            4.0797619            4.6160000
```

```r
# Again... By default, R assumes a two-tailed test,
# that each observation comes from different individuals (i.e.,
 ↪  they are not "paired"),
# and that the alpha level is .05, therefore confidence interval
 ↪  is 95%
```

The results indicate that the average 2019 satisfaction with life score for those who identify as male is 4.08 and 4.62 for those who identify as female. When we compare the difference between the two genders against 0, the resulting t value is -6.90. With degrees of freedom 341, the p value is very small, at 0.0000000000256373. Because p value is smaller than the alpha level of .05, the result is statistically significant. Because 4.62 is statistically significantly greater than 4.08, we would conclude that there is sufficient evidence that on average, the satisfaction with life scores in 2019 is greater for females than for males.

The 95% confidence interval is [-0.69, -0.38] which is interpreted as follows: We are 95% confident that the population mean difference in 2019 satisfaction with life score between males and females is between -0.69 and -0.38.

Note. In calculating the t statistic, R took male - female (since 0 = male and 1 = female, and R takes the group coded with the smaller number and subtracts

the group coded with the larger number). Because males have a smaller 2019 SWLS than females, the t statistic is negative. However, if R had taken female - male (say if it had been 0 = female and 1 = male), the t statistic would be positive. To me, whether the t statistic is positive or negative doesn't matter so long as you know which group has a higher mean and therefore can interpret correctly the direction of the effect. (And this is why descriptive statistics is important–it helps you interpret the results accurately!)

8.6.3 Effect Size: Cohen's d

Finally, we need to get the effect size, Cohen's d, for the independent groups t test using the `cohens_d()` function from the `effectsize` package.

```
# Cohen's d, with mu = 0
cohens_d(swls2019_avg ~ gender,
         data = df,
         mu = 0)
```

```
## Cohen's d |         95% CI
## -------------------------
## -0.75     | [-0.96, -0.53]
##
## - Estimated using pooled SD.
```

Given the Cohen's d is -0.75, we can consider this a medium-large effect size. In other words, there is a medium-large difference in the 2019 satisfaction with life scores between males and females.

Note. Again, note that there is a negative sign here because R took `male - female`. If R took `female - male`, Cohen's d will be a positive number. This is not an issue so long as you know which group had a higher mean and can interpret correctly.

8.7 One-Way Between-Subjects ANOVA

8.7.1 When to Use A One-Way Between-Subjects ANOVA

The independent groups t test is used when you want to compare the scores between two independent groups. But what if you have more than two independent groups? You would use the one-way between-subjects ANOVA.

So, suppose we want to know whether people with different marital status report differing levels of satisfaction with life scores. In other words, we might want to

find out whether there is at least one mean difference between: a) married vs divorced, b) married vs widowed, c) divorced vs widowed. To find out, we will conduct the one-way between-subjects ANOVA.

8.7.2 Conducting and Interpreting the One-Way Between-Subjects ANOVA

Before conducting the ANOVA, we need to convert the grouping variable into a factor. Here, the grouping variable is marital status because we grouped the individuals into the three groups: married, widowed, and divorced.

After converting the grouping variable into a factor, we also want to get the descriptive statistics for each group.

```
# Convert marital status into factor
df$marital_status <- factor(df$marital_status,
                            levels = c(1, 2, 3),
                            labels = c("married", "divorced",
                             ↪  "widowed"))

# Get the descriptive statistics by marital status
describeBy(df$swls2019_avg,
           group = df$marital_status)
```

```
##
##  Descriptive statistics by group
## group: married
##    vars   n mean   sd median trimmed  mad min max range  skew kurtosis   se
## X1    1 118 4.58 0.85    4.5     4.6 1.19   3   6     3 -0.01    -1.08 0.08
## ------------------------------------------------------------
## group: divorced
##    vars   n mean   sd median trimmed  mad min max range  skew kurtosis   se
## X1    1 114 4.16 0.72    4.3    4.16 0.74   2 5.6   3.6 -0.17     -0.1 0.07
## ------------------------------------------------------------
## group: widowed
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 111  4.3 0.65    4.4    4.27 0.59 3.2   6   2.8 0.18    -0.74 0.06
```

From the means, we can see that married people have the highest satisfaction with life scores, followed by widowers and then divorcees. To find out whether the result is statistically significant, we conduct the ANOVA.

```
# Run the ANOVA
# Set up the model being tested and store the model to [mod1]
```

```r
# The model has the following format: DV ~ IV
mod1 <- aov(swls2019_avg ~ marital_status,
            data = df)

# Look at the summary of the results in mod1
summary(mod1)
```

```
##                 Df   Sum Sq Mean Sq F value      Pr(>F)
## marital_status   2  10.7832 5.39159 9.62428 0.000085953 ***
## Residuals      340 190.4705 0.56021
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The F ratio is 9.62. With the degrees of freedom 2 and 340, the p value is 0.000085953, which is smaller than our alpha level of .05. Therefore, we conclude that there is sufficient evidence indicating that there is at least one mean difference in 2019 satisfaction with life scores between the three marital statuses.

8.7.3 Posthoc Analysis: Tukey's HSD

Because the ANOVA results are statistically significant, we want to follow up with a *post hoc* analysis such as the Tukey's HSD to investigate which group(s) differ from which group(s).

The Tukey's HSD conducts all possible pairwise comparisons, while adjusting the p value to control inflation of family-wise error rate. In this example, we would have three pairwise comparisons: a) married vs widowed, b) married vs divorced, and c) widowed vs divorced.

```r
# Perform a post hoc test on the results using Tukey's HSD and
 ↪  get the 95% CI
TukeyHSD(mod1, conf.level = .95)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = swls2019_avg ~ marital_status, data = df)
##
## $marital_status
##                           diff           lwr           upr       p adj
## divorced-married -0.423342254 -0.6547275679 -0.1919569400 0.000064299
## widowed-married  -0.280241258 -0.5132115136 -0.0472710028 0.013569662
## widowed-divorced  0.143100996 -0.0918420376  0.3780440291 0.324684308
```

The results indicate that married people were significantly more satisfied with their lives than divorced and widowed people. However, divorced and widowed people did not differ significantly in their satisfaction with life scores.

Note. Tukey's HSD is one of the most common *post hoc* tests. But there are others. If you need other *post hoc* tests, you might want to check out the `DescTools` package.

8.7.4 Effect Size: Eta-Squared

Just like the t tests, we need to calculate the effect size for ANOVA to see how large the effect is. One of the most commonly reported effect sizes for ANOVA is eta-squared (2). By convention, $2 = 0.01$ indicates a small effect, $2 = 0.06$ indicates a medium effect, and $2 = 0.14$ indicates a large effect.

We can get eta-squared using the `eta_squared()` function from `effectsize` package.

```
eta_squared(mod1, partial = FALSE)
```

```
## # Effect Size for ANOVA (Type I)
##
## Parameter      | Eta2 |       95% CI
## ------------------------------------
## marital_status | 0.05 | [0.02, 1.00]
##
## - One-sided CIs: upper bound fixed at [1.00].
```

```
# partial = FALSE means we do NOT want partial eta-squared; we
 ↪  want eta-squared
# stating partial = TRUE / FALSE actually doesn't matter here.
# For a one-way between-subjects ANOVA, eta-squared and partial
 ↪  eta-squared will be exactly the same value.
# There is no other factor to attribute the variance to, so
 ↪  partial eta-squared is irrelevant here.
```

The eta-squared value here is .05, suggesting a small to medium effect. In other words, there is a small to medium effect of marital status on satisfaction with life scores.

Note. Eta-squared tends to overestimate the effect size. This means it will tell you that the differences between the groups are larger than they actually are. Therefore, non-biased alternatives have been developed in place of eta-squared. This would include omega-squared (2) and epsilon-squared (2), which you can find in the `effectsizes` package as well (look for `omega_squared()`

and `epsilon_squared()`). Omega-squared is more frequently used than epsilon-squared, so if you want to report an unbiased effect size, choose omega-squared.

8.8 Two-Way Between-Subjects Factorial ANOVA

8.8.1 When to Use A Two-Way Between-Subjects Factorial ANOVA

A two-way between-subjects ANOVA is used when we want to examine an interaction. More specifically, it is used when we're trying to investigate whether the effect of one variable depends on a second variable (which is usually called a moderator).

Let's say we're investigating the effect of alcohol on aggression levels. We think that for men, ingesting alcohol makes them more aggressive (compared to not ingesting alcohol). However, for women, ingesting alcohol does not lead to any change in aggression levels (compared to not ingesting alcohol). In such a case, we're interested in an interaction: We want to know whether the effect of alcohol on aggression (the difference between ingesting and not ingesting alcohol) depends on biological sex.

There are many kinds of factorial ANOVAs. A between-subjects ANOVA is used when the observations are all independent of each other (i.e., they are not paired or correlated). (If the observations are paired or related, we would use either repeated measures or mixed ANOVA. These would follow different procedures than shown below.)

So, suppose we want to know whether the difference in satisfaction with life scores between male and female depends on whether they have children. To find out, we will need to conduct a two-way between-subjects ANOVA.

8.8.2 Conducting and Interpreting the Two-Way Between-Subjects ANOVA

We need the `Anova()` function from the `car` package to run this type of ANOVA. So, we need to load the package, change the grouping variables into factors, and get the descriptives for each group.

```
# Load package (remember to install if you've not done so!)
library(car)

# Change have_children into a factor
# If you've not already done so, you'll also need to change
 ↪  marital status to a factor
```

```
df$have_children <- factor(df$have_children,
                           levels = c(0, 1),
                           labels = c("no children", "have
                            ↪  children"))

# Get descriptive statistics for each group
describeBy(df$swls2019_avg, group = df$gender)
```

```
##
##  Descriptive statistics by group
## group: male
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 168 4.08 0.61    4.2    4.06 0.59   3 5.8   2.8 0.26    -0.46 0.05
## ------------------------------------------------------------
## group: female
##    vars   n mean   sd median trimmed  mad min max range  skew kurtosis   se
## X1    1 175 4.62 0.81    4.6    4.65 0.89   2   6     4 -0.35    -0.25 0.06
```

```
describeBy(df$swls2019_avg, group = df$have_children)
```

```
##
##  Descriptive statistics by group
## group: no children
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 169 4.09 0.66    4.2    4.06 0.89 2.8 5.8     3 0.38     -0.4 0.05
## ------------------------------------------------------------
## group: have children
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 174 4.61 0.78    4.6    4.63 0.89   2   6     4 -0.3    -0.06 0.06
```

```
describeBy(df$swls2019_avg, group = list(df$gender,
 ↪  df$have_children))
```

```
##
##  Descriptive statistics by group
## : male
## : no children
##    vars  n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 92 3.95 0.59      4    3.94 0.74   3 5.8   2.8 0.32    -0.53 0.06
## ------------------------------------------------------------
## : female
## : no children
##    vars  n mean   sd median trimmed  mad min max range skew kurtosis   se
```

```
## X1    1 77 4.25 0.71    4.4    4.21 0.89 2.8 5.8     3 0.25    -0.63 0.08
## ------------------------------------------------------------
## : male
## : have children
##    vars  n mean  sd median trimmed  mad min max range skew kurtosis   se
## X1    1 76 4.23 0.6    4.4    4.21 0.59   3 5.8   2.8 0.19    -0.42 0.07
## ------------------------------------------------------------
## : female
## : have children
##    vars  n mean   sd median trimmed  mad min max range  skew kurtosis   se
## X1    1 98 4.91 0.77    4.8    4.97 0.89   2   6     4 -1.02     1.75 0.08
```

Now that we have the descriptives (which will help us interpret the results later on), we can move on to conduct the ANOVA.

```
# IMPORTANT: Change contrasts settings when estimating Type-3 sum
 ↪  of squares
options(contrasts = c('contr.sum', 'contr.poly'))

# Set up the model being tested and store the model to [mod2]
# The model has the following format: DV ~ IV1 + IV2 + IV1*IV2
# You can actually omit IV1 and IV2 and it would still run the
 ↪  factorial ANOVA
mod2 <- aov(swls2019_avg ~ gender + have_children +
 ↪  gender*have_children,
            data = df)

# Conduct Two-Way ANOVA and get the ANOVA summary table
# type = "3" refers to Type 3 sum of squares. There are different
 ↪  types of sum of squares depending on how the sum of squares
 ↪  is partitioned.
# We will use Type 3 sum of squares as we want to test for an
 ↪  interaction.
Anova(mod2, type = "3")
```

```
## Anova Table (Type III tests)
##
## Response: swls2019_avg
##                         Sum Sq  Df     F value                  Pr(>F)
## (Intercept)           6366.863   1 13955.51399 < 0.000000000000000222 ***
## gender                  19.801   1    43.40250       0.00000000017028 ***
## have_children           18.578   1    40.72079       0.00000000057671 ***
## gender:have_children     3.093   1     6.77873              0.0096307 **
## Residuals              154.660 339
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note. Read up more on the four types of sum of squares here: https://md.psych.bio.uni-goettingen.de/mv/unit/lm_cat/lm_cat_unbal_ss_explained.html#type-iii-1. tldr: if we are trying to test an interaction, we should use Type 3 sum of squares.

From the results, we can see that the main effects as well as the interaction effect are statistically significant. To interpret the results, we need to look at the descriptive statistics. The main effect of gender tells us that overall, females reported being significantly more satisfied than males (first set of descriptives). The main effect of presence of children tells us that overall, those with children reported being significantly more satisfied than those without children (second set of descriptives). However, these main effects are qualified by the interaction effect. The interaction effect tells us that the effect of gender on satisfaction with life scores depends on the presence of children.

8.8.3 Conducting and Interpreting the Simple Effects Analysis

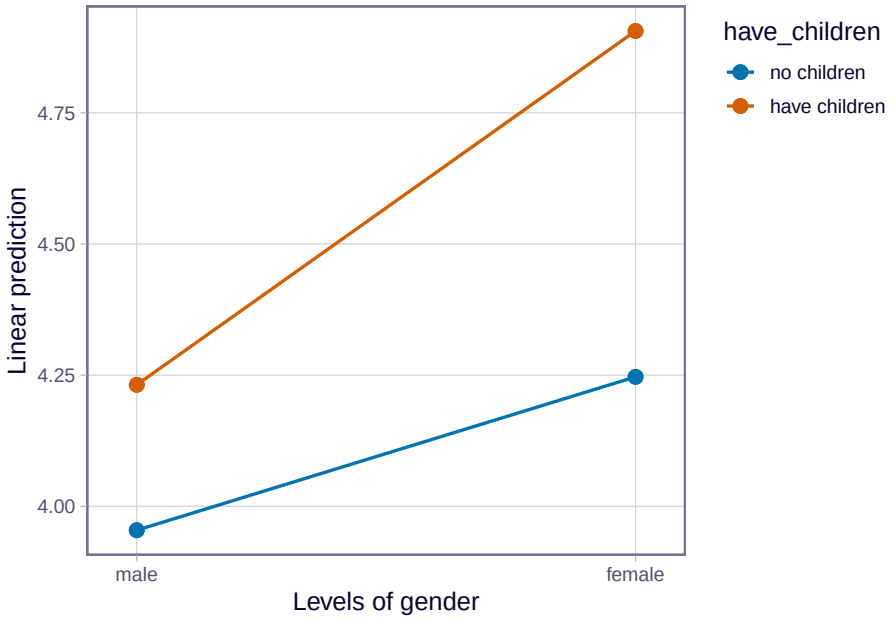
Because the interaction is statistically significant, we will follow up with the simple effects analysis. This is essentially like conducting independent groups t tests (with some corrections). Since we want to find out whether the difference in satisfaction with life scores between males and females depends on having children, we will test the: 1) difference in the satisfaction with life scores of males and females who have children; 2) difference in the satisfaction with life scores of males and females who do not have children.

To do the simple effects analyses, we need the `emmeans` package.

```
# Load package (install package if you've not already done so)
library(emmeans)
```

```
## Warning: package 'emmeans' was built under R version 4.5.3
```

```
# First, let's look at a plot of the means for the four groups
# The four groups being: males with children, males without
 ↪  children, females with children, and females without children
emmip(mod2, have_children ~ gender) # Separate Lines:
 ↪  have_children, X axis: gender
```



In the plot, the x axis is gender and the four points represent the means for each condition (i.e., male without children, male with children, female without children, and female with children). The red line represents the difference between male and female who have no children whereas the blue line represents the difference between male and female who have children. Here, we can see that the blue line is steeper than the red line, indicating that the effect of gender (difference between males and females) is bigger for those who have children than for those who don't have children. This is what the significant interaction effect in the ANOVA is telling us–that the effect of gender is different for those who have children and those who don't.

Now, we conduct the simple effects analysis to find out if there is a difference a) between males and females with children and b) between males and females without children. This is because even though it looks like there are differences on the graph (e.g., the red line is not flat), we don't know whether the differences are statistically significant.

We conduct the simple effects analysis using the `emmeans()` function from the `emmeans` package.

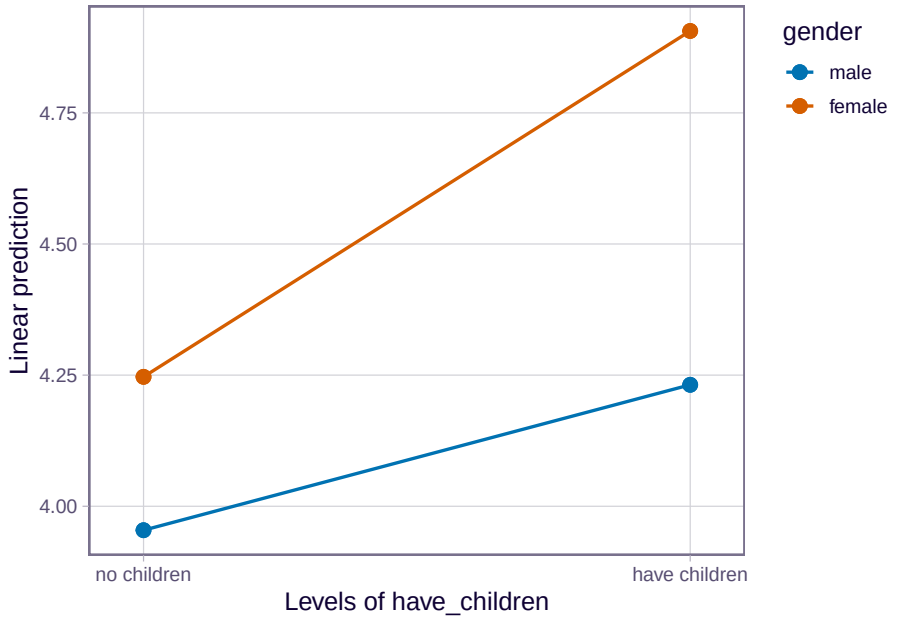
```
# We conduct the simple effects investigating 1) the effect of
 ↪  gender (difference between males and females) for those who
 ↪  have children, and 2) the effect of gender for those who do
 ↪  not have children.
emmeans(mod2, pairwise ~ gender | have_children)  # comparing the
 ↪  pair of conditions in gender for each level of have_children
```

```
## $emmeans
## have_children = no children:
##  gender emmean     SE  df lower.CL upper.CL
##  male     3.95 0.0704 339     3.82     4.09
##  female   4.25 0.0770 339     4.10     4.40
##
## have_children = have children:
##  gender emmean     SE  df lower.CL upper.CL
##  male     4.23 0.0775 339     4.08     4.38
##  female   4.91 0.0682 339     4.77     5.04
##
## Confidence level used: 0.95
##
## $contrasts
## have_children = no children:
##  contrast      estimate    SE  df t.ratio p.value
##  male - female   -0.292 0.104 339  -2.803  0.0054
##
## have_children = have children:
##  contrast      estimate    SE  df t.ratio p.value
##  male - female   -0.675 0.103 339  -6.534 <0.0001
```

The results under `$contrasts` indicate that females with children have significantly higher satisfaction with life scores than males with children. Similarly, females without children have significantly higher satisfaction with life scores than males without children. (Although, remember, as the interaction effect tells us, the effect of gender for those with children is significantly stronger/larger than that for those without children.)

Suppose, though, you were interested in looking at the effect of having children by gender. (This means your moderator is now gender instead of presence of children.) In other words, you're interested in looking at the difference 1) between males with and males without children, and 2) between females with and females without children. In this case, you will swap the variables around.

```
# First, let's look at a plot
# Separate Lines: gender, X axis: have_children
emmip(mod2, gender ~ have_children)
```




```
# Next, we conduct the simple effects investigating 1) the effect
 ↪  of having children (difference between having children and
 ↪  not having children) for males, and 2) the effect of having
 ↪  children (difference between having children and not having
 ↪  children) for females.
emmeans(mod2, pairwise ~ have_children | gender)  # comparing the
 ↪  pair of conditions in have_children for each level of gender
```

```
## $emmeans
## gender = male:
##  have_children emmean     SE  df lower.CL upper.CL
##  no children     3.95 0.0704 339     3.82     4.09
##  have children   4.23 0.0775 339     4.08     4.38
##
## gender = female:
##  have_children emmean     SE  df lower.CL upper.CL
##  no children     4.25 0.0770 339     4.10     4.40
##  have children   4.91 0.0682 339     4.77     5.04
##
## Confidence level used: 0.95
##
## $contrasts
## gender = male:
##  contrast                   estimate    SE  df t.ratio p.value
```

```
##  no children - have children   -0.277 0.105 339  -2.648  0.0085
##
## gender = female:
##  contrast                   estimate    SE  df t.ratio p.value
##  no children - have children   -0.659 0.103 339  -6.410 <0.0001
```

The results under `$contrasts` now indicate that females with children have significantly higher satisfaction with life scores than females without children. Similarly, males with children have significantly higher satisfaction with life scores than males without children. (Note that the interaction effect tells us, the effect of having children for women is significantly stronger/larger than the effect of having children for men.)

8.8.4 Effect Size: Partial Eta-Squared

We can get the effect size, partial eta-squared (2p), for each of the main effects as well as the interaction effect using the `eta_squared()` function from `effectsize` package.

```
eta_squared(Anova(mod2, type = "3"), partial = TRUE)
```

```
## Type 3 ANOVAs only give sensible and informative results when covariates
##   are mean-centered and factors are coded with orthogonal contrasts (such
##   as those produced by `contr.sum`, `contr.poly`, or `contr.helmert`, but
##   *not* by the default `contr.treatment`).
```

```
## # Effect Size for ANOVA (Type III)
##
## Parameter            | Eta2 (partial) |       95% CI
## -----------------------------------------------------
## gender               |           0.11 | [0.07, 1.00]
## have_children        |           0.11 | [0.06, 1.00]
## gender:have_children |           0.02 | [0.00, 1.00]
##
## - One-sided CIs: upper bound fixed at [1.00].
```

The results indicate that we have a large effect for gender, a medium-large effect for having children, and a small effect for the interaction.

8.9 Correlation

8.9.1 When to Use A Correlation

A correlation is used when we want to examine whether two variables are related to each other. Suppose we want to find out whether people who have higher satisfaction with life scores in 2019 also have higher satisfaction with life scores in 2021. Here is where you will conduct a correlation. There are many different types of correlations (e.g., Pearson correlation, Spearman correlation, phi coefficient). The most commonly used is the Pearson correlation, so that's what we'll focus on here.

8.9.2 Conducting and Interpreting the Correlation

As usual, we get the descriptive statistics before conducting the hypothesis test. We should also test whether the variables have a linear relationship (because the Pearson correlation coefficient should only be used for variables that have a linear relationship).

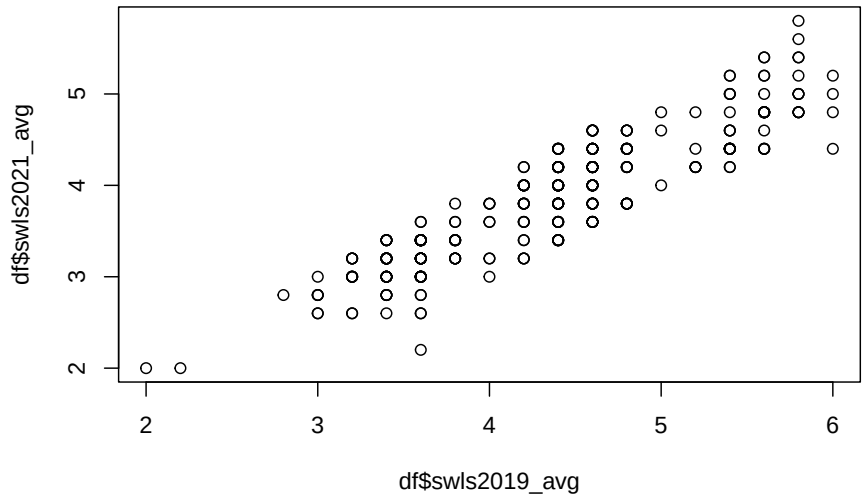
```
# Get the descriptive statistics
describe(df$swls2019_avg)
```

```
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 343 4.35 0.77    4.4    4.33 0.89   2   6     4  0.1    -0.48 0.04
```

```
describe(df$swls2021_avg)
```

```
##    vars   n mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 343 3.85 0.67    3.8    3.84 0.89   2 5.8   3.8  0.1    -0.37 0.04
```

```
# Check linearity assumption
plot(df$swls2019_avg, df$swls2021_avg)
```




```
# Conduct the correlation with cor.test
# Unlike cor(), cor.test() gives you the p value
cor.test(df$swls2019_avg, df$swls2021_avg)
```

```
##
##  Pearson's product-moment correlation
##
## data:  df$swls2019_avg and df$swls2021_avg
## t = 38.39589, df = 341, p-value < 0.000000000000000222
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.879200201 0.919351528
## sample estimates:
##         cor
## 0.901191727
```

The correlation is $r = 0.90$, with a 95% confidence interval [0.88, 0.92]. The p value is very small, smaller than .05, and so we will conclude that the relationship is positive. In other words, people who have higher satisfaction with life scores in 2019 also have significantly higher satisfaction with life scores in 2021.

There are other ways we can write the R code. Here are two alternatives.

```r
# Alternative 1
cor.test(~ swls2019_avg + swls2021_avg,
         data = df)

# Alternative 2
df |>
  cor.test(formula = ~ swls2019_avg + swls2021_avg, .)
```

Both ways would give us the same result as the earlier one but neither are intuitive. I personally prefer the much simpler `cor.test(df$swls2019_avg, df$swls2021_avg)`.

8.9.3 Difference Between Correlated Groups T Test and Correlation

Students sometimes get confused between correlated groups t test and correlation. I'll attempt to explain the difference below.

Correlated groups t test and correlation have different purposes. Correlated groups t test is used when we are trying to find the *difference* between two paired scores. For example, whether there is a difference between satisfaction with life scores in 2019 and in 2021. Correlation is used when we are trying to find the *relationship* between two variables. This would be applied when we're trying to see whether the higher one variable is, the higher/lower another variable is. It is possible for us to have a correlated groups t test that is not statistically significant and a correlation that is significant when we run both on the same data.

Below is a worked example to illustrate the difference.

```r
# Two Variables X and Y
x <- c(1, 2, 3, 4, 5)
y <- c(1, 2, 3, 4, 5)

# Conduct Correlated Groups T Test
# To see if there is a difference between x and y scores
t.test(x = x,
       y = y,
       mu = 0,
       paired = TRUE)
```

```
##
##  Paired t-test
##
```

```
## data:  x and y
## t = NaN, df = 4, p-value = NA
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  NaN NaN
## sample estimates:
## mean difference
##               0
```

```
# Conduct the Correlation
cor.test(x, y)
```

```
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = 82191237, df = 3, p-value < 0.000000000000000222
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  1 1
## sample estimates:
## cor
##   1
```

From the output, you will see that the correlated groups t test is not statistically significant, whereas the correlation is. This is because they're testing different things.

The correlated groups t test is telling us that on average, there is no difference between x scores and y scores. The correlation is telling us there is a perfect positive relationship between x and y. The two variables increase with a fixed proportion: As x increases, y increases proportionately by 1.

8.10 Regression

8.10.1 When to Use A Regression

A regression is used when we want to use one variable to predict another. The simplest form of linear regression is called simple linear regression. It only has two variables: one predictor and one outcome. If you have multiple predictors, then we would use a multiple linear regression instead. For this course, we will stick to simple linear regression.

Suppose we want to predict `swls2021_avg` from `swls2019_avg`. Here, we want to conduct a simple linear regression as we have one predictor and one outcome.

8.10.2 Conducting and Interpreting the Regression

As the descriptives have been calculated previously, and the linearity assumption tested, we will go straight to the regression here.

```
# Conduct regression
# the model takes the form of Y ~ X (where Y = outcome, X =
 ↪  predictor)
mod3 <- lm(df$swls2021_avg ~ df$swls2019_avg)
summary(mod3)
```

```
##
## Call:
## lm(formula = df$swls2021_avg ~ df$swls2019_avg)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -1.056645 -0.248328 -0.006664  0.193336  0.801654
##
## Coefficients:
##                  Estimate Std. Error  t value               Pr(>|t|)
## (Intercept)     0.4065882  0.0911404  4.46112            0.000011087 ***
## df$swls2019_avg 0.7916825  0.0206189 38.39589 < 0.000000000000000222 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.292508 on 341 degrees of freedom
## Multiple R-squared:  0.812147,   Adjusted R-squared:  0.811596
## F-statistic: 1474.24 on 1 and 341 DF,  p-value: < 0.000000000000000222
```

The results indicate that `swls2019_avg` significantly predicted `swls2021_avg` (p value is < 0.000000000000000222). The slope is 0.7916825, meaning that a unit increase in `df$swls2019_avg` is associated with an average increase of 0.7916825 in `df$swls2021_avg`.

Unfortunately, the summary doesn't give us the confidence interval. To get that, we need to specify that we want the confidence interval with the `confint()` function.

```
# To get the 95% confidence interval
confint(mod3)
```

```
##                       2.5 %      97.5 %
## (Intercept)     0.227320000 0.585856416
## df$swls2019_avg 0.751126141 0.832238772
```

We interpret the 95% confidence interval as, "We are 95% confident that the slope in the population is between 0.75 and 0.83."

There are other ways to code a linear regression. One of those alternatives is shown below.

```
# Alternative

# Linear regression
data |>
  lm(swls2021_avg ~ swls2019_avg, data = .) |>
  summary(.)

# 95% confidence interval
data |>
  lm(swls2021_avg ~ swls2019_avg, data = .) |>
  confint(.)
```

You should get the same result as earlier.

8.10.3 R-squared

R-squared (R^2), also known as the coefficient of determination, tells us the amount of variability in the dependent variable that is accounted for by the independent variable. It takes the values between 0 and 1 (inclusive), where 0 indicates that the independent variable explains none of the variance and 1 indicates it explains all of it. The value for this can be found in the regression output under `Multiple R-squared`.

According to Cohen (1988, p. 413-414), R^2 = 0.02 is a small effect size, R^2 = 0.13 is medium, and R^2 = 0.26 is large.

Reference

Cohen J. (1988). Statistical Power Analysis for the Behavioral Sciences, 2nd Ed. Hillsdale, NJ: Laurence Erlbaum Associates

8.11 Chi-square Test of Goodness of Fit

8.11.1 When to Use Chi-square Test of Goodness of Fit

We use the chi-square test of goodness of fit when we want to compare the the observed frequency distribution of a categorical variable to an expected frequency distribution.

Suppose we know that we have the same proportion of men and women in the Singapore population (i.e., 50% of the population are men, 50% are women). We want to know whether the proportion of men and women in our study (observed frequency distribution) is the same as or different from that of the Singapore population (expected frequency distribution). To answer this question, we will conduct a chi-square test of goodness of fit.

8.11.2 Conducting and Interpreting the Chi-Square Test of Goodness of Fit

First, we need the observed frequencies for men and women separately.

```
# Observed frequencies for men and women (i.e., number of men and
 ↪  women in our study)

observedfreq <- table(df$gender)

observedfreq
```

```
##
##   male female
##    168    175
```

The descriptives above indicate that there are 168 males and 175 females. We want to know whether the observed frequencies are significantly different from the frequencies we expect if we have an equal proportion of men and women (expected frequencies).

```
# Set up the expected proportions for the chisq.test()
expectedprop <- c(0.5, 0.5)  # expected proportions would be 0.5
 ↪  for males, 0.5 for females

# Conduct the chi square test of goodness of fit
chisq.test(x = observedfreq, p = expectedprop)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  observedfreq
## X-squared = 0.1428571, df = 1, p-value = 0.705457
```

```
# NOTE. The order in which you enter the values for expectedprop
 ↪  should be the same as that for observedfreq
# If the first value for observedfreq is for males, then the
 ↪  first value for expectedprop should be for males also.
# (And vice versa, of course.)
```

Because p value is .705457 (greater than alpha level of .05), the results tell us that the proportion of men and women in our study is not significantly different from that in the Singapore population.

8.11.3 Effect Size: Cohen's *w*

The effect size usually calculated for chi-square test of goodness of fit is the Cohen's w (omega). By convention, in psychology, $w = 0.1$ is considered a small effect size, $w = 0.3$ is considered a medium effect size, and $w = 0.5$ is considered a large effect size.

We can calculate Cohen's w using the `cohens_w()` function from the `effectsize` package.

```
# Cohen's w
cohens_w(x = observedfreq, p = expectedprop)
```

```
## Cohen's w |       95% CI
## -------------------------
## 0.02      | [0.00, 1.00]
##
## - One-sided CIs: upper bound fixed at [1.00].
```

The Cohen's w is 0.02, which is a very, very small effect size (almost negligible!).

As an aside, Cohen's w has no upper bound when there are more than two categories or when the expected distribution is non-uniform. What this means is that it can be arbitrarily large. Ben-Shachar et al. (2023) proposed an adjustment to Cohen's w, which they call fei. Unlike Cohen's w, fei is bounded between 0 and 1.

To calculate fei, use the function `fei(x = observedfreq, p = expectedprop)` instead.

8.12 Chi-square Test of Independence

8.12.1 When to Use Chi-square Test of Independence

We use the chi-square test of independence to determine if there is a relationship between two categorical variables.

Let's say we believe that people who have children are more likely to stay married (instead of divorcing) than people who don't have children. Put it differently, we think that whether someone is married or divorced depends on whether they have children. To test whether the relationship between the two categorical variables–marital status and whether someone has children–exists, we will conduct a chi-square test of independence.

8.12.2 Conducting and Interpreting the Chi-square Test of Independence

First, we need to filter the data such that we only have married and divorced participants.

```
# Filter (keep) only divorced and married participants
div_and_mar_only <- df |>
  filter(marital_status == "married" | marital_status ==
   ↪  "divorced")

# Drop the "widowed" level for marital status since it is now
 ↪  redundant
div_and_mar_only$marital_status <-
 ↪  droplevels(div_and_mar_only$marital_status)
```

Then, we perform the chi-square test of independence.

```
# Get the observed frequencies for each cell and save it as
 ↪  observedfreq1
observedfreq1 <- table(div_and_mar_only$marital_status,
 ↪  div_and_mar_only$have_children)

# Perform the chi-square test of independence
chisq.test(observedfreq1)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
```

```
## data:  observedfreq1
## X-squared = 1.689424, df = 1, p-value = 0.193677
```

The chi-square test of independence results indicate that there is no significant difference in marital status for people who have children and those who don't ($p = .19$). This means that whether a person is married or divorced does not depend on whether they have children. In other words, marital status is INDEPENDENT of having children.

8.12.3 Effect Size: Phi and Cramer's V

The effect size usually calculated for a 2 x 2 table is phi. (A 2 x 2 table is one in which each of the two variables has two categories.) By convention, in psychology, phi = 0.1 is considered a small effect size, phi = 0.3 is considered a medium effect size, and phi = 0.5 is considered a large effect size.

We can calculate phi using the `phi()` function from the `effectsize` package.

```
# phi
phi(x = observedfreq1)
```

```
## Phi (adj.) |       95% CI
## ------------------------
## 0.07       | [0.00, 1.00]
##
## - One-sided CIs: upper bound fixed at [1.00].
```

Phi is 0.07, which is a small effect size.

As an aside, phi is used for a 2 x 2 table. If you have a larger table (e.g., 2 x 3 or 3 x 4), then use Cramer's V. Note that it is also possible to use Cramer's V for a 2 x 2 table.

```
# Cramer's V
cramers_v(x = observedfreq1)
```

```
## Cramer's V (adj.) |       95% CI
## --------------------------------
## 0.07              | [0.00, 1.00]
##
## - One-sided CIs: upper bound fixed at [1.00].
```

Both phi and Cramer's V will be the same value. So reporting either will be fine. The thresholds for small, medium, and large effect sizes will also remain the same.

8.13 Next steps

Congratulations!!! You've come to the end of the hypothesis testing section and the end of this companion website! I really hope this has convinced you that R is manageable and not as scary as you thought! :)

What remains is… practice, practice, practice! "But where am I gonna find the datasets to practice on?", I hear you wail. Well, here's a start: https://www.openintro.org/data/!

Enjoy~! :)