*David J. Stanley*

# ***PSYC6060 Course Notes***

To my students,

from whom I’ve learn so much about teaching.

# *Contents*

# *List of Tables*

# *List of Figures*

# *Preface*

R's emerging role in psychology will be described here..

## Structure of the book

I'll put more about the structure of the book here in the future.

## Software information and conventions

I used the **knitr** package (Xie, 2015) and the **bookdown** package (Xie, 2020) to compile my book. My R session information is shown below:

```
xfun::session_info()
```

```
## R version 4.0.0 (2020-04-24)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Catalina 10.15.5, RStudio 1.3.959
##
## Locale: en_CA.UTF-8 / en_CA.UTF-8 / en_CA.UTF-8 / C / en_CA.UTF-8 / en_CA.UTF-8
##
## Package version:
##   base64enc_0.1.3 bookdown_0.19.1 compiler_4.0.0
##   digest_0.6.25   evaluate_0.14   glue_1.4.1
##   graphics_4.0.0  grDevices_4.0.0 highr_0.8
##   htmltools_0.4.0 jsonlite_1.6.1  knitr_1.28
##   magrittr_1.5    markdown_1.1    methods_4.0.0
##   mime_0.9        packrat_0.5.0   Rcpp_1.0.4.6
##   rlang_0.4.6     rmarkdown_2.2   rstudioapi_0.11
##   stats_4.0.0     stringi_1.4.6   stringr_1.4.0
##   tinytex_0.23    tools_4.0.0     utils_4.0.0
##   xfun_0.14       yaml_2.2.1
```

Package names are in bold text (e.g., **rmarkdown**), and inline code and filenames are formatted in a typewriter font (e.g., `knitr::knit('foo.Rmd')`). Function names are followed by parentheses (e.g., `bookdown::render_book()`).

# *About the Author*

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

# 1 Introduction

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

## 1.1 A focus on workflow

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

## 1.2 R works with plug-ins

R is a statistical language with many plug-ins called **packages** that you will use for analyses. You can think of R as being like your smartphone. To do things with your phone you need **an App** (R equivalent: a *package*) from the App Store (R equivalent: *CRAN*). Apps need to be **downloaded** (R equivalent: *install.packages*) before you can use them. To use the app you need **Open** it (R equivalent: *library command*). These similarities are illustrated in Table 1.1 below.

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

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

## 1.3 Create an account at R Studio Cloud

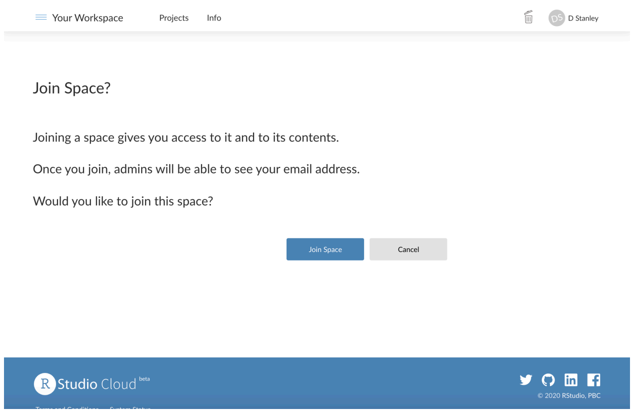
R Studio Cloud[1] accounts are free and required for this guide. Please go to the website and set up a new account.

## 1.4 Join the class workspace

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

1. Log into R Studio Cloud (if you haven't already done so).
2. Go to your university email account and find the message with the subject "R Studio Workspace Invitation". In this message there is a link to the class R Studio Cloud workspace.
3. Click on the workspace link in the email or paste it into your web browser. You should see a screen like the one below in Figure 1.1. Click on the Join button.
4. Then you should see the welcome message illustrated in Figure 1.2. Above this message is the Projects menu option. Click on the word Project.
5. You should now see the First Project displayed as in 1.3. Click the Start button. You will then move to a view of R Studio.

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



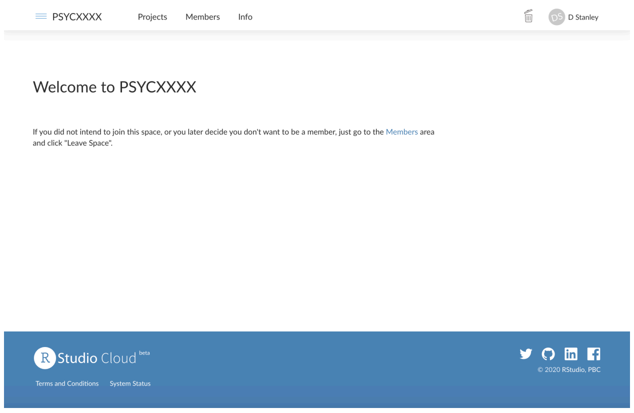
Join Space?

Joining a space gives you access to it and to its contents.

Once you join, admins will be able to see your email address.

Would you like to join this space?

**FIGURE 1.1:** Screen shot of workspace join message.



Welcome to PSYCXXXX

If you did not intend to join this space, or you later decide you don't want to be a member, just go to the Members area and click "Leave Space".

**FIGURE 1.2:** Screen shot of welcome messag

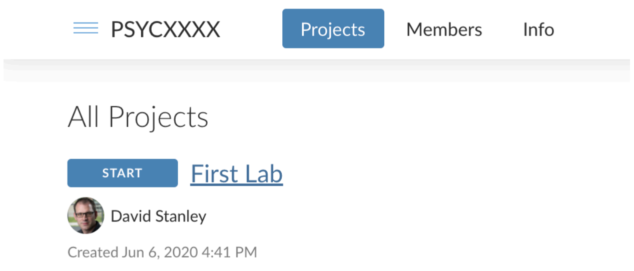
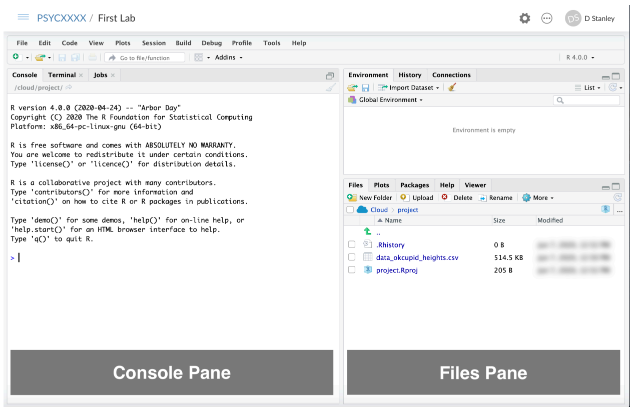


**FIGURE 1.3:** Screen shot of starting first assignment

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

## 1.5 Exploring the R Studio Interface

Once you have opened (or created) a Project folder, you are presented with the R Studio interface. There are a few key elements to the user interface that are illustrated in Figure 1.4 In the lower right of the screen you can see the a panel with several tabs (i.e., Files, Plots, Packages, etc) that I will refer to as the Files pane. You look in this pane to see all the files associated with your project. On the left side of the screen is the Console which is an interactive pane where you type and obtain results in real time. I've placed two large grey blocks on the screen with text to more clearly identify the Console and Files panes. Not shown in this figure is the Script panel where we can store our commands for later reuse.



```
R version 4.0.0 (2020-04-24) -- "Arbor Day"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

| | Name | Size | Modified |
|---|---|---|---|
| | .. | | |
| ☐ | .Rhistory | 0 B | |
| ☐ | data_okcupid_heights.csv | 514.5 KB | |
| ☐ | project.Rproj | 205 B | |

**FIGURE 1.4:** R Studio interface

### 1.5.1 Console panel

When you first start R, the Console panel is on the left side of the screen. Sometimes there are two panels on the left side (one above the other); if so, the Console panel is the lower one (and labeled accordingly). We can use R a bit like a calculator. Try typing the following into the Console window: 8 + 6 + 7 + 5. You can see that R immediately produced the result on a line preceded by two hashtags (##).

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

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

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

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

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

```
print(my_sum)
```

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

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

### 1.5.2 Script Panel

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

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

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

## 1.6 Writing your first script

### 1.6.1 Create the script file

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

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

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

### 1.6.2 Add a comment to your script

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

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

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

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

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

Note that in the above comment I used the internationally accepted date format order Year/Month/Day. Some people use the mnemonic *Your My Dream* to remember this order. Wikipedia provides more information about the Internationally Date Format ISO 8601[6].

Moving forward, I suggest you use comments to make your own personal notes in your own code as your write it.

### 1.6.3 Background about the tidyverse

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

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

```
##  [1] "broom"      "cli"        "crayon"
##  [4] "dbplyr"     "dplyr"      "forcats"
##  [7] "ggplot2"    "haven"      "hms"
## [10] "httr"       "jsonlite"   "lubridate"
```

[4]https://xkcd.com/1421/
[5]https://cran.r-project.org/web/packages/checkpoint/index.html
[6]https://en.wikipedia.org/wiki/ISO_8601

```
## [13] "magrittr"   "modelr"     "pillar"
## [16] "purrr"      "readr"      "readxl"
## [19] "reprex"     "rlang"      "rstudioapi"
## [22] "rvest"      "stringr"    "tibble"
## [25] "tidyr"      "xml2"       "tidyverse"
```

Before you can use a package it needs to be installed – this is the same as downloading an app from the App Store. Normally, you can install a **single** packages with the install.packages command. Previously, you needed run an install.package command for every package in the tidyverse as illustrated below (though we no longer use this approach).

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

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

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

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

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

### 1.6.4 Add library(tidyverse) to your script

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

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

### 1.6.5 Activate tidyverse auto-complete for your script

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

```
## -- Attaching packages ----------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.1     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

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

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

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

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

## 1.7 Loading your data

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

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

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

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

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

The output indicates that you have loaded a data file and the type of data in each column. The sex column is of type col_character which indicates it contains text/letters. Most of the columns are of the type character. The age and height columns contain numbers are correspondingly indicated to be the type col_double. The label col_double indicates that a column of numbers represented in R with high precision[7]. There are other ways of representing numbers in R but this is the type we will see/use most often.

## 1.8 Checking out your data

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

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

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

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

```
view(okcupid_profiles)
```

### 1.8.2 print(): See you data in the Console

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

```
print(okcupid_profiles)
```

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

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

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

```
head(okcupid_profiles)
```

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

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

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

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

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

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

```
tail(okcupid_profiles)
```

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

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

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

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

### 1.8.5 summary(): Quick summaries

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

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

```
summary(okcupid_profiles)
```

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

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

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

You can:

1. Console: Enter commands directly
2. Script: Select the command(s) and press the Run button.
3. Script: Source (Without Echo)
4. Script: Source With Echo

Two of these approaches involve using the Source button, see Figure 1.5. You bring up the options for the Source button, illustrated in this figure, by clicking on the small arrow to the right of the word Source.

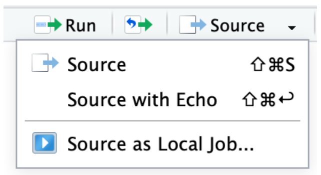


**FIGURE 1.5:** Source button options

### 1.9.1 Run select text

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

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

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

### 1.9.2 Source (without Echo)

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

### 1.9.3 Source with Echo

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

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

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

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

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

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

## 1.10 Trying Source with Echo

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

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

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

head(okcupid_profiles)

tail(okcupid_profiles)

summary(okcupid_profiles)
```

Now do the following:

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

You should see the output below:

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

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

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

```
head(okcupid_profiles)
```

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

```
tail(okcupid_profiles)
```

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

```
summary(okcupid_profiles)
```

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

Congratulations you just ran your first script!

## 1.11 A Few Key Points About R

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

#### 1.11.0.1 Vector of numbers

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

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

```
print(even_numbers)
```

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

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

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

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

#### 1.11.0.2 Vector of characters

We can also create vectors using only characters:

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

```
print(favourite_things)
```

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

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

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

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

### 1.11.1 Lists

Lists are similar to vectors in that you can create them and access items by their numeric position. Vectors must be all characters or all numbers. Lists can be a mix of characters or numbers. Most importantly items in lists can be accessed by their label.

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

print(my_list)
```

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

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

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

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

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

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

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

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

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

## 1.12 That's it!

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

# 2

# *Making your data ready for analysis*

A common problem in science is that after an article is published, the authors can't reproduce the numbers in the paper. You can read more about the important problem in a surprising article in the journal Molecular Brain[1]. In this article an editor reports how a request for the data underlying articles resulted in the wrong data for 40 out of 41 papers. Long story short – keep track of the data and scripts you use for your paper.

## 2.1 Using R the old way or the new way (the tidyverse way)

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

##### 2.1.0.1 Tibbles vs Data Frames: Why use read_csv instead of read.csv

When you load data into R it is typically represented in one of two formats inside the computer - depending on the command you used. The original format for representing a data set in R is the data frame. You will see this term used frequently when you read about R. When you load data using read.csv your data is loaded into a data frame in the computer. That is your data is represented in the memory of the computer in particular format and structure called a data frame.

[1]https://molecularbrain.biomedcentral.com/articles/10.1186/s13041-020-0552-2

#### 2.1.0.2 read.csv puts data into a data frame

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

Notice that when you print a data frame it does NOT show you the number of rows or columns above the data like our example did with the okcupid_profiles data. It also list ALL of your data rather than just the first few rows. As a result in the output below I show only the first 10 rows of the output - because all the rows are printed in your Console (too much to show here).

```
print(my_dataframe)
```

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

#### 2.1.0.3 read_csv puts data into a tibble

When you use the read_csv command the data you load is stored in the computer as a tibble. The tibble is modern version of the data frame. Notice that when you print a tibble it DOES show you the number of rows and columns. As well, the tibble only provides the first few rows of output so it doesn't fill your screen.

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

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

```
print(my_tibble)
```

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

#### 2.1.0.4 Deeper differences between data frames and tibbles

In short you should always use tibbles (i.e., use read_csv) - they are simply enhanced data frames (i.e., the new version of the data frame). The differences between data frames and tibbles run deeper than the superficial output provided here. On some rare occasions an old package or command may not work with a tibble so you need to make it a data frame. You can do so with the commands below:

#### 2.1.0.5 Converting a tibble into a data frame

This command creates a new data set called new_data_frame (use any name you want) from the tibble data.

```
new_dataframe <- as.data.frame(my_tibble)
```

## 2.2 Required Packages

This chapter requires the following packages are installed:

| Required Packages |
|---|
| apaTables |
| janitor |
| psych |
| tidyverse |

**Important Note:** that you should NOT use library(psych) at any point. There are major conflict between the psych package and the tidyverse. We will access the psych package command by preceding each command with psych:: instead of using library(psych).

## 2.3 Objective

## 2.4 Context

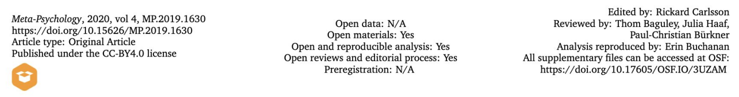
There is a growing interest in ensuring your analyses are reproducible. That is, that a third part could generate the numbers in the research article from materials provided by the researcher. Although this sounds like a low bar for rigor, it is in fact a surprisingly challenging bar. Indeed, the editor of Molecular Brain[2]. reported that a request for the data underlying articles resulted in the wrong data for 40 out of 41 papers.

Consequently, a trend is for journals and authors to adopt Transparency and Openness Promotion (TOP) guidelines[3]. These guidelines involve such things

[2] https://molecularbrain.biomedcentral.com/articles/10.1186/s13041-020-0552-2
[3] https://www.cos.io/our-services/top-guidelines

as making your materials, data, code, and analysis scripts available on public repositories so anyone can check your data. A new journal rating system even emerged call the TOP Factor[4].

The idea is not that open science articles are not more trust worthy that other types of articles – the idea is that trust doesn't play a role. Anyone can inspect the data using the scripts provided by authors. It's really just the same as making your science available for auditing the way financial records can be audited. This process avoids the problem reported at Molecular Brain (doubless is common to many journals) - because the data and scripts need to have been uploaded at the time of publication. The TOP open science guidelines have made an impact and newer journals such as Meta Psycchology have fully embraced open science. Figure 2.1 shows the header from an article[5] in Meta Psychology that clearly delineates the open science attributes of the article. Take note that the header even specifies who checked that the analyses in the article were reproducible.



*Meta-Psychology*, 2020, vol 4, MP.2019.1630
https://doi.org/10.15626/MP.2019.1630
Article type: Original Article
Published under the CC-BY4.0 license

Open data: N/A
Open materials: Yes
Open and reproducible analysis: Yes
Open reviews and editorial process: Yes
Preregistration: N/A

Edited by: Rickard Carlsson
Reviewed by: Thom Baguley, Julia Haaf, Paul-Christian Bürkner
Analysis reproduced by: Erin Buchanan
All supplementary files can be accessed at OSF: https://doi.org/10.17605/OSF.IO/3UZAM

**FIGURE 2.1:** Open science in an article header

## 2.5 A mindset for moving foward

In this chapter we walk you though the process of going from raw data to analytic data by creating processing script.

see Figure 2.2

[4]https://topfactor.org
[5]https://open.lnu.se/index.php/metapsychology/article/view/1630/2266

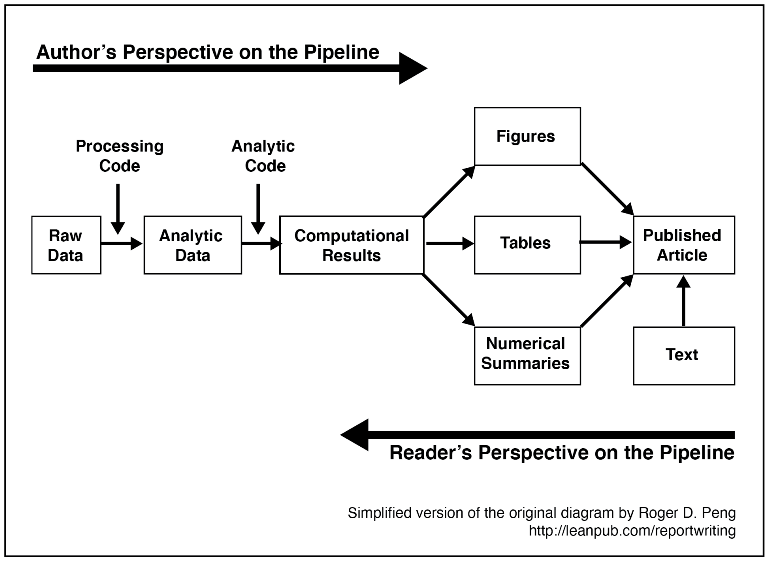

Author's Perspective on the Pipeline
Processing Code
Analytic Code
Raw Data
Analytic Data
Computational Results
Figures
Tables
Numerical Summaries
Published Article
Text
Reader's Perspective on the Pipeline
Simplified version of the original diagram by Roger D. Peng
http://leanpub.com/reportwriting

**FIGURE 2.2:** Data science pipeline by Roger Peng.

# 3

# *Begin with the end in mind*

-sample size analysis -unfalsible article by lakens

TIDYVERSE NAMING CONVENTION ARE IMPORTANT BECAUSE IN SOME CASES THEY ARE REQUIRED for the SCRIPTS to work. If you haven't following the naming conventions you may be making your life quite difficult.

# 4 Collecting your data

- data are collected in different ways
- data collected by programs in the lab that require
- lab measurementson paper and entering later
- paper surveys and entering later
- website surveys
- a mix of all of the above

each trial is a row - multiple rows per person…

Distinguish between entering data and not. But also think about what's not confusing… talk about wide not showing DV or IV just levels of IV

if you're writing a cognitive program or making a survey think about the variable names think about how the data is outputed.

If you have entirely within person data then tidy is the obvious choice If you have entirely between then wide makes sense If you have a mix - it's more difficult. Different research area. Think about the problems.

For example, a good data code book is essential if you have wide with repeated measures varibles because there is no way to tell what the IV or DV is by inspecting the data.

# 5 Entering your data

# 6 Making analytic data

Think about - missing data - column names - representation of categorical variables in the data set

# 7

# *Entering data*

- computer/web collection

## 7.1 Begin with a project

I suggest you begin every R Studio task in the following way:

R Studio in the Cloud 1. Create a new Project using the web interface 2. Upload your data files in using the upload button in the Files pane

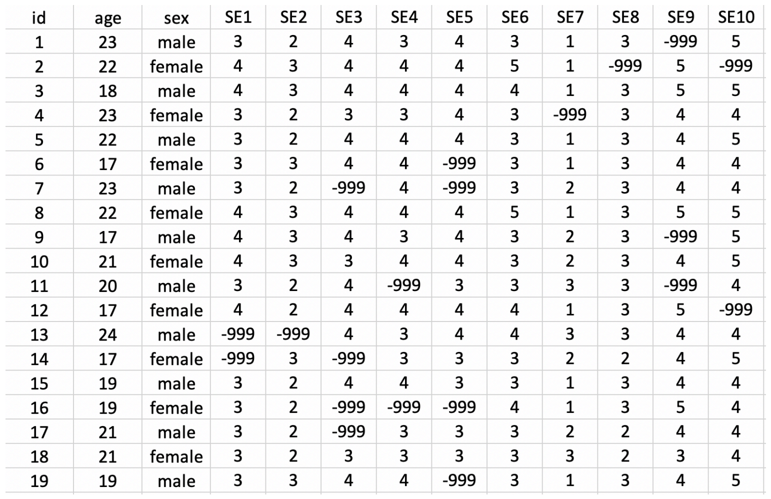
R Studio on Your Computer 1. Create a folder on your computer for the analysis 2. Place your data files in that folder 3. Use the menu item File > New Project… to start the project 4. On the window that appears select "Existing Directory" 5. On the next screen, press the "Browse" button and find/select the folder with your data 6. Press the Create Project Button

Regardless of whether your are working from the cloud or locally you should now have an R Studio project with your data files in it. Using Projects.

**Class note: You dont' need to do either of these approach. You will just "Start" each assignment in the class workspace on R Studio Cloud".**

## 7.2 Raw data

We begin by examining the data as originally entered into a spreadsheet. In Figure 7.1 you see a screen shot of the intial raw data as a researcher might receive it. Take careful note of the numerous -999 values used to indicate missing values. As part of creating the analytic data that we will analyze we need to indicate to the computer that the -999 are not data but codes to represent missing values.



| id | age | sex | SE1 | SE2 | SE3 | SE4 | SE5 | SE6 | SE7 | SE8 | SE9 | SE10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 23 | male | 3 | 2 | 4 | 3 | 4 | 3 | 1 | 3 | -999 | 5 |
| 2 | 22 | female | 4 | 3 | 4 | 4 | 4 | 5 | 1 | -999 | 5 | -999 |
| 3 | 18 | male | 4 | 3 | 4 | 4 | 4 | 4 | 1 | 3 | 5 | 5 |
| 4 | 23 | female | 3 | 2 | 3 | 3 | 4 | 3 | -999 | 3 | 4 | 4 |
| 5 | 22 | male | 3 | 2 | 4 | 4 | 4 | 3 | 1 | 3 | 4 | 5 |
| 6 | 17 | female | 3 | 3 | 4 | 4 | -999 | 3 | 1 | 3 | 4 | 4 |
| 7 | 23 | male | 3 | 2 | -999 | 4 | -999 | 3 | 2 | 3 | 4 | 4 |
| 8 | 22 | female | 4 | 3 | 4 | 4 | 4 | 5 | 1 | 3 | 5 | 5 |
| 9 | 17 | male | 4 | 3 | 4 | 3 | 4 | 3 | 2 | 3 | -999 | 5 |
| 10 | 21 | female | 4 | 3 | 3 | 4 | 4 | 3 | 2 | 3 | 4 | 5 |
| 11 | 20 | male | 3 | 2 | 4 | -999 | 3 | 3 | 3 | 3 | -999 | 4 |
| 12 | 17 | female | 4 | 2 | 4 | 4 | 4 | 4 | 1 | 3 | 5 | -999 |
| 13 | 24 | male | -999 | -999 | 4 | 3 | 4 | 4 | 3 | 3 | 4 | 4 |
| 14 | 17 | female | -999 | 3 | -999 | 3 | 3 | 3 | 2 | 2 | 4 | 5 |
| 15 | 19 | male | 3 | 2 | 4 | 4 | 3 | 3 | 1 | 3 | 4 | 4 |
| 16 | 19 | female | 3 | 2 | -999 | -999 | -999 | 4 | 1 | 3 | 5 | 4 |
| 17 | 21 | male | 3 | 2 | -999 | 3 | 3 | 3 | 2 | 2 | 4 | 4 |
| 18 | 21 | female | 3 | 2 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 4 |
| 19 | 19 | male | 3 | 3 | 4 | 4 | -999 | 3 | 1 | 3 | 4 | 5 |

**FIGURE 7.1:** Raw data for item scoring

## 7.3 Loading raw data

Create an RStudio project for this activity Create a new script in your project and save it wite

```r
library(tidyverse)
library(janitor)
```

```
##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```r
my_missing_value_codes <- c("-999", "", "NA")

raw_data <- read_csv(file = "data_item_scoring.csv",
                     na = my_missing_value_codes)
```

```
## Parsed with column specification:
## cols(
##   id = col_double(),
##   age = col_double(),
##   sex = col_character(),
##   SE1 = col_double(),
##   SE2 = col_double(),
##   SE3 = col_double(),
##   SE4 = col_double(),
##   SE5 = col_double(),
##   SE6 = col_double(),
##   SE7 = col_double(),
##   SE8 = col_double(),
##   SE9 = col_double(),
##   SE10 = col_double()
## )
```

## 7.4 Initial inspection

We use glimipse to do an initial inspection of the column names in this data set. All of the column name conform to tidyverse style guidelines[1] so we do not need to run the clean_name() function from the janitor package

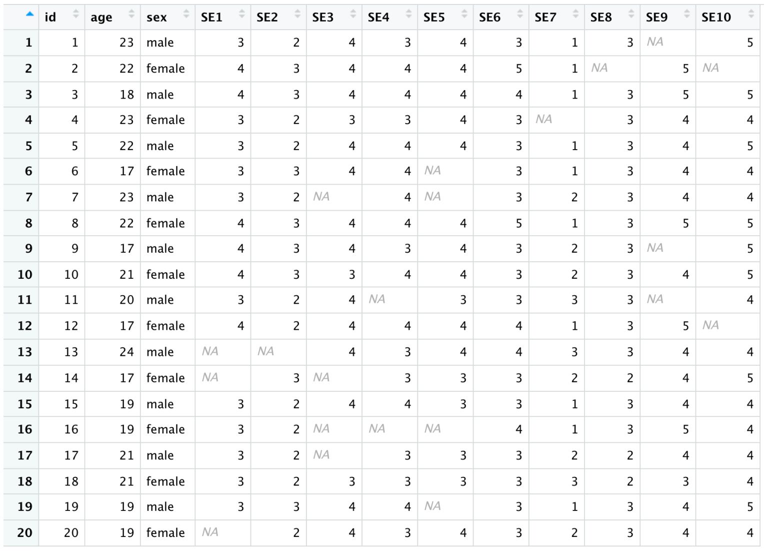
```
glimpse(raw_data)
```

```
## Rows: 300
## Columns: 13
## $ id   <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
## $ age  <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, 2...
## $ sex  <chr> "male", "female", "male", "female", "...
## $ SE1  <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, N...
## $ SE2  <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, N...
## $ SE3  <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, ...
## $ SE4  <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, ...
## $ SE5  <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4, 3, 4,...
## $ SE6  <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4...
## $ SE7  <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3, 1, ...
## $ SE8  <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
## $ SE9  <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5...
## $ SE10 <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA,...
```

[1] https://style.tidyverse.org

```r
view(raw_data)
```

See Figure 7.2



| | id | age | sex | SE1 | SE2 | SE3 | SE4 | SE5 | SE6 | SE7 | SE8 | SE9 | SE10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 23 | male | 3 | 2 | 4 | 3 | 4 | 3 | 1 | 3 | NA | 5 |
| 2 | 2 | 22 | female | 4 | 3 | 4 | 4 | 4 | 5 | 1 | NA | 5 | NA |
| 3 | 3 | 18 | male | 4 | 3 | 4 | 4 | 4 | 4 | 1 | 3 | 5 | 5 |
| 4 | 4 | 23 | female | 3 | 2 | 3 | 3 | 4 | 3 | NA | 3 | 4 | 4 |
| 5 | 5 | 22 | male | 3 | 2 | 4 | 4 | 4 | 3 | 1 | 3 | 4 | 5 |
| 6 | 6 | 17 | female | 3 | 3 | 4 | 4 | NA | 3 | 1 | 3 | 4 | 4 |
| 7 | 7 | 23 | male | 3 | 2 | NA | 4 | NA | 3 | 2 | 3 | 4 | 4 |
| 8 | 8 | 22 | female | 4 | 3 | 4 | 4 | 4 | 5 | 1 | 3 | 5 | 5 |
| 9 | 9 | 17 | male | 4 | 3 | 4 | 3 | 4 | 3 | 2 | 3 | NA | 5 |
| 10 | 10 | 21 | female | 4 | 3 | 3 | 4 | 4 | 3 | 2 | 3 | 4 | 5 |
| 11 | 11 | 20 | male | 3 | 2 | 4 | NA | 3 | 3 | 3 | 3 | NA | 4 |
| 12 | 12 | 17 | female | 4 | 2 | 4 | 4 | 4 | 4 | 1 | 3 | 5 | NA |
| 13 | 13 | 24 | male | NA | NA | 4 | 3 | 4 | 4 | 3 | 3 | 4 | 4 |
| 14 | 14 | 17 | female | NA | 3 | NA | 3 | 3 | 3 | 2 | 2 | 4 | 5 |
| 15 | 15 | 19 | male | 3 | 2 | 4 | 4 | 3 | 3 | 1 | 3 | 4 | 4 |
| 16 | 16 | 19 | female | 3 | 2 | NA | NA | NA | 4 | 1 | 3 | 5 | 4 |
| 17 | 17 | 21 | male | 3 | 2 | NA | 3 | 3 | 3 | 2 | 2 | 4 | 4 |
| 18 | 18 | 21 | female | 3 | 2 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 4 |
| 19 | 19 | 19 | male | 3 | 3 | 4 | 4 | NA | 3 | 1 | 3 | 4 | 5 |
| 20 | 20 | 19 | female | NA | 2 | 4 | 3 | 4 | 3 | 2 | 3 | 4 | 4 |

**FIGURE 7.2:** Missing values now NA

## 7.5 Handling categorical variables

```r
# Turn all columns that are of type character into factors
raw_data <- raw_data %>%
  mutate(across(.cols = where(is.character),
                .fns = as.factor))
```

We can see there was only one column that changes, but if there had been many columns that were characters they all would have changed.

```r
glimpse(raw_data)
```

```
## Rows: 300
## Columns: 13
## $ id   <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
## $ age  <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, 2...
## $ sex  <fct> male, female, male, female, male, fem...
## $ SE1  <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, N...
## $ SE2  <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, N...
## $ SE3  <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, ...
## $ SE4  <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, ...
## $ SE5  <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4, 3, 4,...
## $ SE6  <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4...
## $ SE7  <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3, 1, ...
## $ SE8  <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
## $ SE9  <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5...
## $ SE10 <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA,...
```

It's often helpful to has id as a factor rather than a number so we add and extra command that changes this value:

```
raw_data <-raw_data %>%
  mutate(id = as.factor(id))
```

Now it looks like our data is ready for the creation of scale scores:

```
glimpse(raw_data)
```

```
## Rows: 300
## Columns: 13
## $ id   <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
## $ age  <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, 2...
## $ sex  <fct> male, female, male, female, male, fem...
## $ SE1  <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, N...
## $ SE2  <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, N...
## $ SE3  <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, ...
## $ SE4  <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, ...
## $ SE5  <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4, 3, 4,...
## $ SE6  <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4...
## $ SE7  <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3, 1, ...
## $ SE8  <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
## $ SE9  <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5...
## $ SE10 <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA,...
```

## 7.6 Seeing your data

See the first six rows of the data with the *head* command below. If you wanted to see all of the data you would use View(raw_data). The NA values in the output indicate missing values (NA = Not Available).

```
head(raw_data)
```

```
## # A tibble: 6 x 13
##   id      age sex     SE1   SE2   SE3   SE4   SE5   SE6
##   <fct> <dbl> <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 1        23 male      3     2     4     3     4     3
## 2 2        22 fema~     4     3     4     4     4     5
## 3 3        18 male      4     3     4     4     4     4
## 4 4        23 fema~     3     2     3     3     4     3
## 5 5        22 male      3     2     4     4     4     3
## 6 6        17 fema~     3     3     4     4    NA     3
## # ... with 4 more variables: SE7 <dbl>, SE8 <dbl>,
## #   SE9 <dbl>, SE10 <dbl>
```

## 7.7 Dealing with reverse key items

Our first step is dealing with reverse key items. The way you deal with these items depends on how you scored them. Imagine you had a 5-point scale. You could have scored the scale with the values 1, 2, 3, 4, and 5. Alternatively, you could have scored the scale with the values 0, 1, 2, 3, and 4. In this example, we scored the data using the 1 to 5 system. So we'll use that. Later I'll show you how to deal with the other scoring system (0 to 4).

### 7.7.1 Scoring items where the ratings scale starts with 1

We need to take items that were reversed-key when the participant wrote them and recode those responses. We do that with using the *mutate* command from the *dplyr* package.

In this data file the only reverse-key item was SE7 (we known this from when we created the survey). We use the command below to reverse key an item with reponse options ranging from 1 to 5. So we use 6 in the command (i.e., one higher than 5).

```
raw_data <-raw_data %>%
  mutate(SE7c = 6 - SE7)
```

The command above creates a new column in raw_data called SE7c that has the reverse-keyed values for SE7 in it. You can see the new SE7c column using command below that displays the first six rows of the data. The SE7c column is at the far right of the data displayed.

```
glimpse(raw_data)
```

```
## Rows: 300
## Columns: 14
## $ id   <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12...
## $ age  <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, 2...
## $ sex  <fct> male, female, male, female, male, fem...
## $ SE1  <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, N...
## $ SE2  <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, N...
## $ SE3  <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, ...
## $ SE4  <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, ...
## $ SE5  <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4, 3, 4,...
## $ SE6  <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4...
## $ SE7  <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3, 1, ...
## $ SE8  <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, ...
## $ SE9  <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5...
## $ SE10 <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA,...
## $ SE7c <dbl> 5, 5, 5, NA, 5, 5, 4, 5, 4, 4, 3, 5, ...
```

We have reverse keyed one item. So now when we create our scale we will use item SE7c (the c stands for correctly coded) instead of the original item SE7. That is, we will use items SE1, SE2, SE3, SE4, SE5, SE6, **SE7c**, SE8, SE9, and SE10 to form the scale.

**Note for a box** multiple items what it looks like

**Note FOR A BOX** . If you had used response options numbered 0 to 4 for each item you would use the command below instead. Note that we use 4 in the command this time instead of a value one higher.

```
raw_data <- mutate(raw_data, SE7c = 4 - SE7)
```

## 7.8 Creating the scale score

```
raw_data <- raw_data %>%
  rowwise() %>%
  mutate(self_esteem = mean(c(SE1, SE2, SE3, SE4, SE5, SE6, SE7c, SE8, SE9, SE10),
                            na.rm = TRUE)) %>%
  ungroup()
```

When you see ungroup() in ths context you can think of it as "turn off row-wise".

We can see our data now has the self esteem column:

```
glimpse(raw_data)
```

```
## Rows: 300
## Columns: 15
## $ id          <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,...
## $ age         <dbl> 23, 22, 18, 23, 22, 17, 23, 22...
## $ sex         <fct> male, female, male, female, ma...
## $ SE1         <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, ...
## $ SE2         <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, ...
## $ SE3         <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3,...
## $ SE4         <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, ...
## $ SE5         <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4...
## $ SE6         <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, ...
## $ SE7         <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2,...
## $ SE8         <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3,...
## $ SE9         <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4...
## $ SE10        <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5,...
## $ SE7c        <dbl> 5, 5, 5, NA, 5, 5, 4, 5, 4, 4,...
## $ self_esteem <dbl> 3.556, 4.250, 4.100, 3.222, 3....
```

(Alternative code: removing ITEM 7 and using begings with: show full example) Think about this when creating names for your column

## 7.9 Creatinig analytic data from raw data

Select only the columns you will use in your analysis:

```
analytic_data <- raw_data %>%
  select(id, age, sex, self_esteem)
```

We can see our new data set has only these columns of interest:

```
glimpse(analytic_data)
```

```
## Rows: 300
## Columns: 4
## $ id          <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,...
## $ age         <dbl> 23, 22, 18, 23, 22, 17, 23, 22...
## $ sex         <fct> male, female, male, female, ma...
## $ self_esteem <dbl> 3.556, 4.250, 4.100, 3.222, 3....
```

## 7.10 Wait: I need alpha!

We return to the raw_data file that has the original item data to obtain Cronbach's alpha which is labeled "raw alpha" in the output.

```
self_esteem_item_analysis <- raw_data %>%
  select(SE1, SE2, SE3, SE4, SE5, SE6, SE7c, SE8, SE9, SE10) %>%
  psych::alpha()

print(self_esteem_item_analysis$total)
```

```
##  raw_alpha std.alpha G6(smc) average_r   S/N    ase
##     0.8278    0.8333  0.8276    0.3333 4.999 0.0143
##   mean     sd median_r
##  3.656 0.3392   0.3277
```

```
# To see the full item analysis use:
# print(self_esteem_item_analysis)
```

## 7.11 Wait: I need item correlations and descriptive statistics

```
SE_items <- raw_data %>%
  select(starts_with("SE", ignore.case = FALSE))

psych::describe(SE_items)
```

```
##      vars   n mean   sd median
## SE1     1 276 3.39 0.54      3
## SE2     2 272 2.35 0.48      2
## SE3     3 269 3.96 0.37      4
## SE4     4 285 3.54 0.50      4
## SE5     5 265 3.78 0.47      4
## SE6     6 275 3.34 0.51      3
## SE7     7 273 1.51 0.61      1
## SE8     8 272 2.84 0.37      3
## SE9     9 265 4.29 0.70      4
## SE10   10 276 4.57 0.61      5
## SE7c   11 273 4.49 0.61      5
```

When you run the cor command you have to indicate how the it will handle missing the data. The options are below. You can learn more about what each one of these options means by typing: **?cor** into the Console, this will bring up the help page for the cor command.

| Missing data options for cor |
|---|
| everything |
| all.obs |
| complete.obs |
| na.or.complete |
| pairwise.complete.obs |

```
SE_items %>%
  cor(use = "pairwise.complete.obs") %>%
  round(2)
```

```
##        SE1  SE2  SE3  SE4  SE5  SE6   SE7  SE8
## SE1   1.00 0.31 0.20 0.34 0.34 0.33 -0.32 0.28
## SE2   0.31  1.00  0.24  0.22  0.31  0.32 -0.29  0.29
## SE3   0.20  0.24  1.00  0.28  0.30  0.23 -0.43  0.38
## SE4   0.34  0.22  0.28  1.00  0.29  0.34 -0.37  0.32
## SE5   0.34  0.31  0.30  0.29  1.00  0.39 -0.41  0.41
## SE6   0.33  0.32  0.23  0.34  0.39  1.00 -0.32  0.24
## SE7  -0.32 -0.29 -0.43 -0.37 -0.41 -0.32  1.00 -0.44
## SE8   0.28  0.29  0.38  0.32  0.41  0.24 -0.44  1.00
## SE9   0.38  0.36  0.35  0.32  0.43  0.34 -0.42  0.51
## SE10  0.31  0.34  0.28  0.19  0.40  0.25 -0.39  0.35
## SE7c  0.32  0.29  0.43  0.37  0.41  0.32 -1.00  0.44
##        SE9  SE10  SE7c
## SE1   0.38  0.31  0.32
## SE2   0.36  0.34  0.29
## SE3   0.35  0.28  0.43
## SE4   0.32  0.19  0.37
## SE5   0.43  0.40  0.41
## SE6   0.34  0.25  0.32
## SE7  -0.42 -0.39 -1.00
## SE8   0.51  0.35  0.44
## SE9   1.00  0.36  0.42
## SE10  0.36  1.00  0.39
## SE7c  0.42  0.39  1.00
```

# 8

# *Getting data into R Studio*

SOMETHING ABOUT USING HELP IN TIDYVERSE CHAPTER BEFORE THIS ONE

A large segment of the R community as moved toward a standardized way of naming your columns as desribed in the tidyverse style guide[1]. Likewise, in order to use the tidyverse packages effectively it helps to have tidy data[2]. In animal or human participant research, however, it can be

## 8.1 Creating data files

Moving from raw data to analytic data The following steps - and more if you have repeated measures data

### 8.1.1 Column name conventions

### 8.1.2 Categorical columns

### 8.1.3 Missing data representation

Before you start entering data, stop and consider how you want to represent missing data. You may want a single missing data code for all data (e.g., -999) or you might want to have a variety of different missing data codes.

As you do so, be sure to distinguish between codes for missing data and codes for particpants responses such as "Not applicable". Missing data is data that is missing - which is very different that a question being not applicable for a person. This distinction is particularly important if you start to use strategies for estimating the values for missing data (see McKnight et al., 2007). Consider the case of a particpant presented with two questions. The first question asks

[1]`https://style.tidyverse.org`
[2]`https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html`

if they have children and the second question asks the age of their first born child. Imagine the participant indicates they have no children on the first question and then on the second question indicates “No applicable” when asked about the age of their first born child. It would not make sense to treat this “Not applicable” as a missing value and try to estimate it with missing data strategies. Consequently, when you enter your data (and conduct your analyses) be sure to distinguish missing data from not applicable and similar responses.

### 8.1.4 Making a data codebook

Data codebooks are useful because they document the nature of the data in your data file. For example, you might have a column named “ac1” that contains the responses to a Likert-type question. The data codebook could clarify that “ac1” refers an affective commitment item and provide the text of the item. Likewise, sometimes people use value to represent categorical responses in data sets (but try to avoid this practice). In older data it’s not uncommon to use 1 to represent male and 2 to represent female (or the other way around). A data codebook clarifies which sex you mean by 1 and which sex you mean by 2. Though as noted previously, you’re better off entering “male” and “female” directly into your data file rather than using numerical proxies like 1 and 2.

A data code book is often a spreadsheet that where the rows in the data code book

reverse coded

Figure showing data and data codebook.

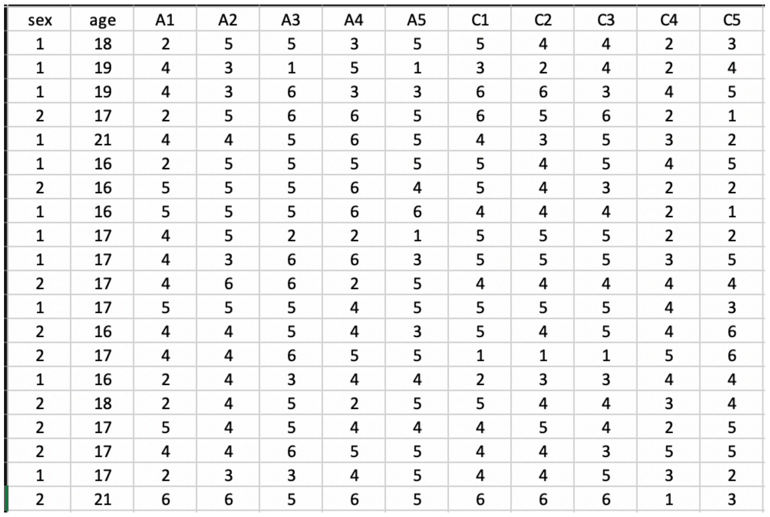
Figure 8.1 Figure 8.2

There are a many ways to create data codeboos

## 8.2 Loading files

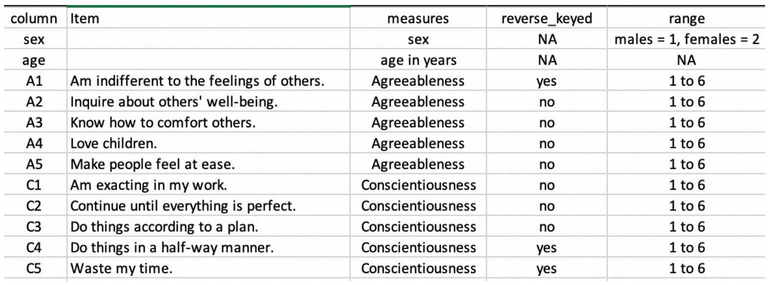
### 8.2.1 Loading CSV files

https://readr.tidyverse.org



| sex | age | A1 | A2 | A3 | A4 | A5 | C1 | C2 | C3 | C4 | C5 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 18 | 2 | 5 | 5 | 3 | 5 | 5 | 4 | 4 | 2 | 3 |
| 1 | 19 | 4 | 3 | 1 | 5 | 1 | 3 | 2 | 4 | 2 | 4 |
| 1 | 19 | 4 | 3 | 6 | 3 | 3 | 6 | 6 | 3 | 4 | 5 |
| 2 | 17 | 2 | 5 | 6 | 6 | 5 | 6 | 5 | 6 | 2 | 1 |
| 1 | 21 | 4 | 4 | 5 | 6 | 5 | 4 | 3 | 5 | 3 | 2 |
| 1 | 16 | 2 | 5 | 5 | 5 | 5 | 5 | 4 | 5 | 4 | 5 |
| 2 | 16 | 5 | 5 | 5 | 6 | 4 | 5 | 4 | 3 | 2 | 2 |
| 1 | 16 | 5 | 5 | 5 | 6 | 6 | 4 | 4 | 4 | 2 | 1 |
| 1 | 17 | 4 | 5 | 2 | 2 | 1 | 5 | 5 | 5 | 2 | 2 |
| 1 | 17 | 4 | 3 | 6 | 6 | 3 | 5 | 5 | 5 | 3 | 5 |
| 2 | 17 | 4 | 6 | 6 | 2 | 5 | 4 | 4 | 4 | 4 | 4 |
| 1 | 17 | 5 | 5 | 5 | 4 | 5 | 5 | 5 | 5 | 4 | 3 |
| 2 | 16 | 4 | 4 | 5 | 4 | 3 | 5 | 4 | 5 | 4 | 6 |
| 2 | 17 | 4 | 4 | 6 | 5 | 5 | 1 | 1 | 1 | 5 | 6 |
| 1 | 16 | 2 | 4 | 3 | 4 | 4 | 2 | 3 | 3 | 4 | 4 |
| 2 | 18 | 2 | 4 | 5 | 2 | 5 | 5 | 4 | 4 | 3 | 4 |
| 2 | 17 | 5 | 4 | 5 | 4 | 4 | 4 | 5 | 4 | 2 | 5 |
| 2 | 17 | 4 | 4 | 6 | 5 | 5 | 4 | 4 | 3 | 5 | 5 |
| 1 | 17 | 2 | 3 | 3 | 4 | 5 | 4 | 4 | 5 | 3 | 2 |
| 2 | 21 | 6 | 6 | 5 | 6 | 5 | 6 | 6 | 6 | 1 | 3 |

**FIGURE 8.1:** Illustrative Data



| column | Item | measures | reverse_keyed | range |
|---|---|---|---|---|
| sex | | sex | NA | males = 1, females = 2 |
| age | | age in years | NA | NA |
| A1 | Am indifferent to the feelings of others. | Agreeableness | yes | 1 to 6 |
| A2 | Inquire about others' well-being. | Agreeableness | no | 1 to 6 |
| A3 | Know how to comfort others. | Agreeableness | no | 1 to 6 |
| A4 | Love children. | Agreeableness | no | 1 to 6 |
| A5 | Make people feel at ease. | Agreeableness | no | 1 to 6 |
| C1 | Am exacting in my work. | Conscientiousness | no | 1 to 6 |
| C2 | Continue until everything is perfect. | Conscientiousness | no | 1 to 6 |
| C3 | Do things according to a plan. | Conscientiousness | no | 1 to 6 |
| C4 | Do things in a half-way manner. | Conscientiousness | yes | 1 to 6 |
| C5 | Waste my time. | Conscientiousness | yes | 1 to 6 |

**FIGURE 8.2:** Illustrative Codebook

### 8.2.2 Loading SPSS files

https://haven.tidyverse.org

## 8.3 Cleaning column names

clean_names() remove_empty() http://sfirke.github.io/janitor/articles/janitor.html

## 8.4 Numerical variables

### 8.4.1 Creating scales scores

A brief explanation of commands:

You can read the "%>%" pipe command as "as then" - it is a way of linking one command to the next within one line.

mean() provides the average the values it is sent. The na.rm = TRUE argument needs explaining though. na stand for "not available". rm stands for "remove". na.rm = TRUE tells the computer to remove missing values prior to calculating the mean. Always use mean to create scale scores.

mutate() creates a new column based on directions in the brackets. Here mutate is creating a new affective_commitment column that this the average (for each person/row) the

- Use mean() to create scale scores. Do not add the items and divide by the number of items - or you will likely not take missing values into accout appropriately. If there is any missing data this process will result in incorrect scale scores for anyone with missing data.

```
attitude %>% rowwise() %>% mutate(affective_commitment = mean(rating, complaints, na.rm =

attitude %>% select(rating, complaints, learning) %>% psych::alpha()
```

## 8.5 Categorical varibles

### 8.5.1 Identifying categorical varibles

### 8.5.2 Levels of categorical varibles

## 8.6 Analytic data

### 8.6.1 Columns: Just what you need

https://dplyr.tidyverse.org

### 8.6.2 Rows: Just what you need

## 8.7 Initial demographics

summary() review for nonenses

Getting demographics tabyl() %>% tabyl(meat_colour) tabyl(meat_colour, plant)

# A

# *More to Say*

Yeah! I have finished my book, but I have more to say about some topics. Let me explain them in this appendix.

To know more about **bookdown**, see `https://bookdown.org`.

# *Bibliography*

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2020). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.19.1.

# *Index*