David J. Stanley

Psychology 6060

To my son,

without whom I should have finished this book two years earlier

Contents

List of Tables

List of Figures

Preface

Hi there, this is my great book.

Why read this book

It is very important...

Structure of the book

Chapters 1 introduces a new topic, and ...

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   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()`).

Acknowledgments

A lot of people helped me when I was writing the book.

Frida Gomam
on the Mars

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 journals. David also created the apaTables R package.

1

Introduction

We will use the statistical analysis software R in this course with the interface provided by R Studio. We can access R Studio using the website R Studio Cloud[1] at http://www.rstudio.cloud.

1.1 A focus on workflow

A key focus on this course is training you in a workflow that will avoid a large number of problms than can occur when using R.

1.1.0.1 Create an account at R Studio Cloud

R Studio Cloud accounts are free and required for this course. Please go to the website and set up a new account using your university email address.

1.2 R works with plug-in / add-ons

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

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

R terminology	Smartphone terminology
CRAN	App Store
Package	App
`install.packages`	Download app from App Store
`library`	Open app

Note: The analogy between R terminology and smartphone terminology is credited to Kim (2018).

FIGURE 1.1: Packages as smart phone apps

1.2.0.1 Join the class workspace

You need to join the class workspace to access class materials.

To do so:

1. Log into R Studio Cloud (if you haven't already done so).
2. Go to your email and find the message were you will emailed the link to the class R Studio workspace.
3. Click on that link or paste it into your web browser. You should see a screen like the one below. Click join.

INSERT IMAGE HERE OF JOIN SCREEN

1.2.0.2 Start the first lab

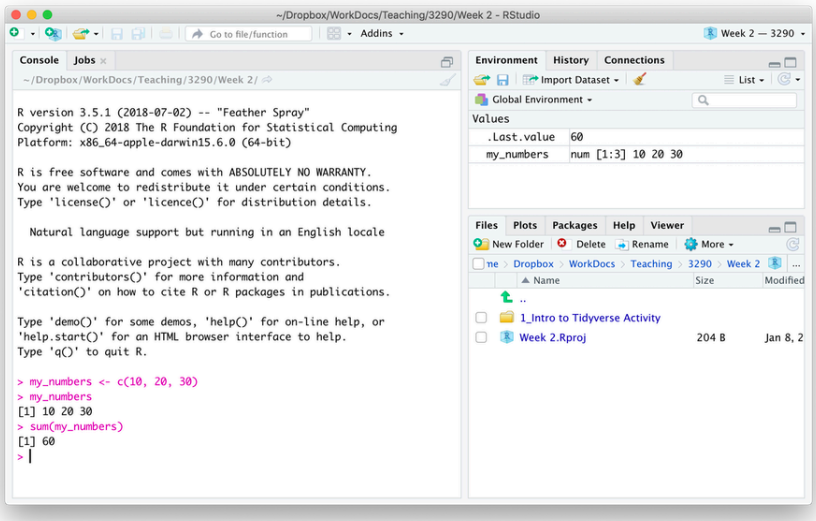
1.3 Exploring the R Studio Interface

1.3.0.1 File/Plot Window

Picture with arrow

1.3.0.2 Console Window

Notice the we can use R a bit like a calculator.



Picture with arrow

1.3.0.3 Script Window

Create a script in your R Studio project by using the menu File > New File > R Script. Save the file with an appropriate name. A common, and good, convention is to start all script names with the word "script" and separate words with an underscore "*_*". *You might save this file with the name "script_my_first_one.R". The advantage of this approach 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 prevously, we are trying to instill and effect workflow as you learn R.

1.4 Writing your first script

1.4.0.1 Add a comment with today's date

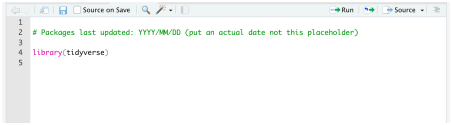
It's a good idea to add a comment with the date of your script. Like smartphone apps, packages are updated regularly. Sometimes when a package is updated it will no longer work with an older script. Fortunately, the **checkpoint** package let's users role back the clock and user older versions of packages. 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 (if needed). This won't be relevant to what we do in this course - but it is good practice. This is another important workflow tip.

```
## Packages last updated: YYYY/MM/DD (put use an actual date not this placeholder)
```

1.4.0.2 Add library(tidyverse) to your script

Now put the line below at the top of your script file:

```
library(tidyverse)
```



1.4.0.3 Activate tidyverse auto-complete for your script

Select the library(tidyverse) text with your mouse/track-pad and highlight it. Then click the Run button in the upper right of the script window.

After you click the Run button you should see text like the following the Console window:

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

## v ggplot2 3.3.0     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## 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()
```

Running the library(tidyverse) prior to entering the rest of your script allows R Studio to present auto-complete options when typing your text.

1.5 Loading your data

1.5.0.1 Use read_csv (not read.csv) to open the file.

If you followed the steps above, you should have auto-complete for the tidyverse. This means you should see an auto-complete option for read_csv. Be sure to use read_csv not read.csv. The use of the _ indicates we are using the command from the tidyverse package.

We already have the **data_okcupid_heights.csv** data file in our project directory. We will load this data with the command below. Notice how we format our command below. After each command option for read_csv there is a comma followed by a **shift-return**. The shift return moves to the next line and keeps the text aligned.

This command puts the data_okcupid_heights.csv data into the R data.frame/tibble called my_data. Note that we avoid the use of capital letters in my_data. Also note that we avoid using a period. It's good to have a style guide when you are typing variable/object names to avoid mistakes. I suggest the following style guide (which is common in R):

- Use the underscore (_) to represent a space in your variable object name
- Avoid using periods in your variable/object names
- Use only lower case - avoid upper case

Add the line below to your script file:

```
my_data <- read_csv(file = "data_okcupid_heights.csv",
                    na = c("", "NA", "-999"))
```

In the command above, *file* indicate the name of the file in the project directory. Likewise, *na* indicate the values that are used in that .csv file to represent missing values. In R, *NA* is often used to indicate missing values (i.e., NA indicates Not Available). Here we are telling read_csv that when we read the data file is there is nothing present between two commas (i.e., ""), or if there is the text"NA", or if there is the text"-999", treat all of text strings as missing values.

Select the read_csv lines and click the Run button in the top right of the script window. You should see text like that below.

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

This text indicates that you have loaded a data file with two columns named sex and height. The sex column is of type col_character which indicates it contains text/letters. The height column is of type col_double which merely indicates that it is a column of numbers represented with high precision (double indicates it is high precision - technical description of exactly what is meant by double is left for a computer science course).

1.5.0.2 Missing values and read_csv

Tell people how to identify missing values when they load their data.

1.6 Getting your data ready for analysis

1.6.0.1 Make sure you are using well-formatted variable names

Janitor package here..

1.6.0.2 Tell R which variables are categorical (i.e., a factor)

A key part of using R is telling it which variables are categorical variables. If you don't do this some analysis won't run. Worse - other analyses will run but provide the wrong numbers.

Below are a series of command where we load our data and identify which variables are categorical. In R categorical variables are called factors.

```
my_data <- read_csv(file = "data_okcupid_heights.csv",
                    na = c("", "NA", "-999"))
```

```
## Parsed with column specification:
## cols(
##   sex = col_character(),
```

```
##   height = col_double()
## )
```

```
glimpse(my_data)
```

```
## Rows: 59,826
## Columns: 2
## $ sex    <chr> "male", "male", "male", "male", "ma...
## $ height <dbl> 75, 70, 68, 71, 66, 67, 65, 70, 72,...
```

We know that sex is a categorical variable (i.e., a factor). Yet when we look at the output above from glimpse it is of type chr - which means character. We need to make the data into a factor.

```
my_data <- mutate(my_data, sex = as.factor(sex))
```

In later exercise we will deal with situation where numbers like 1 and 2 are used to designate sex instead of a label like male or female. When numbers are used in factors we have to provide labels for them (i.e., value labels in SPSS terms) using the level command; but that's a future exercise.

Notice now that we have made sex a factor, there is fct beside it in the glimpse output.

```
glimpse(my_data)
```

```
## Rows: 59,826
## Columns: 2
## $ sex    <fct> male, male, male, male, male, male,...
## $ height <dbl> 75, 70, 68, 71, 66, 67, 65, 70, 72,...
```

1.7 Checking out the structure of your data

1.7.0.1 glimpse(): A way to check out your data structure

Check out the structure of the data with the glimpse command. Add the command below to your script file, highlight it, and click Run.

```
glimpse(my_data)
```

```
## Rows: 59,826
## Columns: 2
## $ sex    <fct> male, male, male, male, male, male,...
## $ height <dbl> 75, 70, 68, 71, 66, 67, 65, 70, 72,...
```

You may also encounter the str function (or structure function) as a way to glimpse the data. This is an older command that provides a bit more information. However, the output of the str function isn't as user friendly as that from the glimpse function.

```
str(my_data)
```

```
## tibble [59,826 x 2] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ sex   : Factor w/ 2 levels "female","male": 2 2 2 2 2 2 2 2 2 2 ...
##  $ height: num [1:59826] 75 70 68 71 66 67 65 70 72 72 ...
```

You can also see the names of the columns using just:

```
names(my_data)
```

```
## [1] "sex"    "height"
```

1.7.0.2 head(): Check out the first few rows

You can see the first few rows of the data with the head command. Add the command below to your script file, highlight it, and click Run.

```
head(my_data)
```

```
## # A tibble: 6 x 2
##   sex   height
##   <fct>  <dbl>
## 1 male      75
## 2 male      70
## 3 male      68
## 4 male      71
## 5 male      66
## 6 male      67
```

1.7.0.3 tail(): Check out the last few rows

You can see the last few rows of the data with the head command. Add the command below to your script file, highlight it, and click Run.

```
tail(my_data)
```

```
## # A tibble: 6 x 2
##   sex    height
##   <fct>   <dbl>
## 1 female     64
## 2 female     63
## 3 female     67
## 4 female     61
## 5 female     62
## 6 female     62
```

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

You can view the data in a spreadsheet style window using the command below.

Do NOT add this command to your script file. Just type it in the Console. Adding it to the script can cause problems.

```
view(my_data)
```

1.7.0.5 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 course. The output of the summary command changes depending on what you give it (data, regression results, etc). Add the command below to your script file, highlight it, and click Run.

Later you will use the summary command in other ways. It is a very interesting command because it can give the summary of many different things (data, regression results, etc).

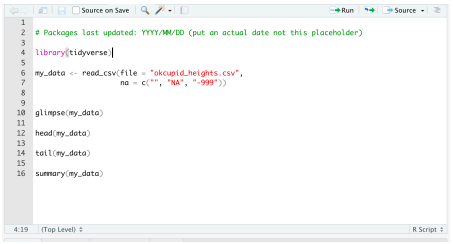
```
summary(my_data)
```

```
##      sex            height
##  female:24089   Min.   :55.0
##  male  :35737   1st Qu.:66.0
##                 Median :68.0
##                 Mean   :68.3
##                 3rd Qu.:71.0
```

```
##                   Max.   :80.0
```

1.8 Run *vs.* Source with Echo *vs.* Source

You have the script below in your Script window. We will discuss how, Run, Source with Echo, and Source (without Echo) work.



1.8.0.1 Run

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

I strongly suggest you ONLY use the Run button when testing a command to make sure it works or to debug a script.

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.

1.8.0.2 Source (without Echo)

Source (without Echo) is not designed for the typical analysis workflow. It is mostly helpful when you run simulations. You can see below that when you run Source (without Echo) much of the output you would wish to read is suppressed.

In general, avoid this option.

```
Parsed with column specification:
cols(
```

```
  sex = col_character(),
  height = col_double()
)
Observations: 59,826
Variables: 2
$ sex    <chr> "male", "male", "male", "male", "male", "male", "male", "male", "male", m…
$ height <dbl> 75, 70, 68, 71, 66, 67, 65, 70, 72, 72, 70, 71, 72, 69, 71, 73, 70, 72, 67…
>
```

1.8.0.3 Source with Echo

The Source with Echo command runs all of the contents of a script and presents the output in the R console.

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: * Menu item: **Session > Restart R** * Click the down arrow beside Source, and click on Source With Echo

This will run the script and provide you with the desired output. This should be your default workflow.

Inspect the output of the script we created below using this workflow. Notice how you can see the results of every command presented.

```
> library(tidyverse)

> my_data <- read_csv(file = "data_okcupid_heights.csv",
+                     na = c("", "NA", "-999"))
Parsed with column specification:
cols(
  sex = col_character(),
  height = col_double()
)

> glimpse(my_data)
Observations: 59,826
Variables: 2
$ sex    <chr> "male", "male", "male", "male", "male", "male", "male", "male", "male", m…
$ height <dbl> 75, 70, 68, 71, 66, 67, 65, 70, 72, 72, 70, 71, 72, 69, 71, 73, 70, 72, 67…

> head(my_data)
```

```
## A tibble: 6 x 2
  sex   height
  <chr>  <dbl>
1 male      75
2 male      70
3 male      68
4 male      71
5 male      66
6 male      67

> tail(my_data)
## A tibble: 6 x 2
  sex    height
  <chr>   <dbl>
1 female     64
2 female     63
3 female     67
4 female     61
5 female     62
6 female     62

> summary(my_data)
     sex                height
 Length:59826       Min.   :55.00
 Class :character   1st Qu.:66.00
 Mode  :character   Median :68.00
                    Mean   :68.29
                    3rd Qu.:71.00
                    Max.   :80.00
```

1.8.0.4 Source: Use Session > Restart R EVERY time before you click Source

Avoid a large number of problems, prior to using the Source button, go to **Session > Restart R**.

This ensure each script runs in a fresh environment.

Do **Session > Restart R** before click source – **ALWAYS**.

1.9 A few analyses with scripts and Source with Echo

1.9.0.1 select(): Obain a subset of columns

1.9.0.2 filter(): Obtain a subset of rows

1.9.0.3 summarise(): Obtain summary statistics for a column

1.10 Avoid problems in your scripts

I suggest you do the following things to avoid problems in your scripts

1.10.0.1 Source: Use Session > Restart R EVERY time before you click Source

As noted above, prior to using the Source button, go to Session > Restart R.

This ensure each script runs in a fresh environment.

1.10.0.2 Avoid View() in your scripts

Depending on your version of R Studio, putting View() in your scripts can cause problems. Try to restrict use of View to the Console window.

1.10.0.3 Package Installation Advice

In the first part of the course, the package will be installed for you. To use those package you will merely need to activate them with the library command. Later in the course you will install your packages. When you do so there a few things you need to keep in mind to avoid problems:

1.10.0.3.0.1 Do NOT put install.package() commands in your scripts

Try to avoid putting install.package commands in your scripts. Doing so can corrupt your R installation. You may need to re-install R if you do so. Worse, you may create problems on your computer so that even re-installing R won't get it working again. As discussed above, install.packages commands in scripts are problematic because they often follow library commands. When this happen your R installation becomes corrupted.

1.10.0.3.0.2 install.packages: Use Session > Restart R EVERY time before you install.packages

Prior to using install.packages, go to Session > Restart R.

This ensure each installation occurs in a fresh environment.

1.11 Using R the old way or the new way (the tidyverse way)

Talk about difference. A key differences

1.11.0.1 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. The original format for representing a data set in R is the data frame. When you load data using read.csv your data is loaded into a data frame.

1.11.0.2 read.csv puts data into a data frame

```
my_dataframe <- read_csv(file = "data_okcupid_heights.csv",
                         na = c("", "NA", "-999"))
```

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

Notice that when you print a data frame it does NOT show you the number of rows or columns above the data. Here I actually pasted just the first 10 rows of the output - because all the rows are printed in your console.

```
print(my_dataframe)
```

```
## # A tibble: 10 x 2
##    sex   height
##    <chr>  <dbl>
```

```
##  1 male      75
##  2 male      70
##  3 male      68
##  4 male      71
##  5 male      66
##  6 male      67
##  7 male      65
##  8 male      70
##  9 male      72
## 10 male      72
```

1.11.0.3 read_csv puts data into a tibble

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_heights.csv",
                      na = c("", "NA", "-999"))
```

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

```
print(my_tibble)
```

```
## # A tibble: 59,826 x 2
##    sex   height
##    <chr>  <dbl>
##  1 male      75
##  2 male      70
##  3 male      68
##  4 male      71
##  5 male      66
##  6 male      67
##  7 male      65
##  8 male      70
##  9 male      72
## 10 male      72
## # ... with 59,816 more rows
```

1.11.0.4 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:

1.11.0.4.0.1 Converting a tibble into a data frame

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

1.12 A Few Key Points About R

Sometimes you will need to send a command in R additional information. Occasionally this will involve using vectors of numbers, vectors of characters, or lists.

1.12.0.1 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
```

```
print(even_numbers *2)
```

```
## [1]  4  8 12 16 20
```

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

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

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

1.12.0.2 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"
```

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

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

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

1.12.0.2.0.1 Using character vectors with a command

Previously, we used a vector to indicate what codes represent missing values:

```
missing_value_codes <- c("", "NA", "-999")

print(missing_value_codes)
```

```
## [1] ""     "NA"   "-999"
```

You could load data like this:

```
missing_value_codes <- c("", "NA", "-999")

my_data <- read_csv(file = "data_okcupid_heights.csv",
                    na = missing_value_codes)
```

Or the way we used did previously:

```
my_data <- read_csv(file = "data_okcupid_heights.csv",
                    na = c("", "NA", "-999"))
```

1.12.0.3 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 = "Stanley",
                first_name = "David",
                office_number = 4002)

print(my_list)
```

```
## $last_name
## [1] "Stanley"
##
## $first_name
## [1] "David"
##
## $office_number
## [1] 4002
```

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

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

```
## [1] "Stanley"
```

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

```
## [1] 4002
```

One way of indicating to R that you are using the value 1 and 2 to code Male and Female in your data is by using a list that you pass to another command (i.e., with levels commands from a future exercise). Here is how you would represent how you coded sex:

```
sex_coding <- list("male" = 1,
                   "female" = 2)

print(sex_coding)
```

```
## $male
## [1] 1
##
## $female
## [1] 2
```

```
print(sex_coding$male)
```

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

2

The FOO Method

We talk about the *FOO* method in this chapter.

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.

Index