PSYC 6380 Psychological Applications of Multivariate Analysis

David Stanley

2025-12-02

Table of contents

Preface

This is a Quarto book designed to supplement PSYC 6380.

1 An Emphasis on Workflow

1.1 Required Packages

The data files below are used in this chapter.

Required Data
data_ex_between.csv
data_ex_within.csv
data_item_scoring.csv

The following CRAN packages must be installed:

Required CRAN Packages
apaTables
Hmisc
janitor
psych
skimr
tidyverse

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

1.2 Objective

Due to a number of high profile failures to replicate study results (Nosek 2015) it has become increasingly clear that there is a general crisis of confidence in many areas of science (Baker 2016). Statistical (and other) explanations have been offered (Simmons, Nelson, and Simonsohn 2011) for why it's hard to replicate results across different sets of data. However, scientists are also finding it challenging to recreate the numbers in their own papers using their own

data. Indeed, the editor of Molecular Brain asked authors to submit the data used to create the numbers in published papers and found that the wrong data was submitted for 40 out of 41 papers (Miyakawa 2020).

Consequently, some researchers have suggested that it is critical to distinguish between replication and reproducibility (Patil P. 2019). **Replication refers to trying to obtain the same results from a different data set.** ***Reproducibility refers to trying to obtain the same results from the same data set.*** Unfortunately, some authors use these two terms interchangeably and fail to make any distinction between them. I encourage you to make the distinction and the use the terms consist with use suggested by (Patil P. 2019).

It may seem that reproducibility should be a given - but it's not. Correspondingly, there is a trend for journals and authors to adopt Transparency and Openness Promotion (TOP) guidelines. These guidelines involve such things as making your materials, data, code, and analysis scripts available on public repositories so anyone can check your data. A new open science journal rating system has even emerged called the TOP Factor.

The idea is not that open science articles are more trustworthy than other types of articles – the idea is that trust doesn't play a role. Anyone can inspect the data using the scripts and data provided by authors. It's really just the same as making your science available for auditing the way financial records can be audited. But just like in the world of business, some people don't like the idea of making it possible for others to audit their work. The problems reported in Molecular Brain (doubtless common to many journals) are likely avoided with open science - because the data and scripts needed to reproduce the statistics in the articles are uploaded prior to publication.

The TOP open science guidelines have made an impact and some newer journals, such as Meta Psychology, have fully embraced open science. Figure 1.1 shows the header from an article in Meta Psychology that clearly delineates the open science attributes of the article that used computer simulations (instead of participant data). Take note that the header even indicates who verified that the analyses in the article were reproducible.

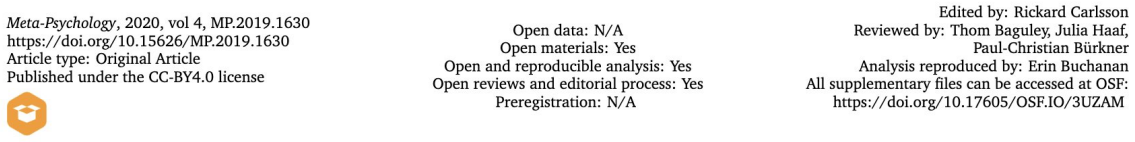


Figure 1.1: Open science in an article header

In Canada, the majority of university research is funded by the Federal Government's Tri-Agency (i.e., NSERC, SSHRC, CIHR). The agency has a new Data Management Policy in which they state that *"The agencies believe that research data collected through the use of public funds should be responsibly and securely managed and be, where ethical, legal and commercial*

obligations allow, ***available for reuse*** *by others. To this end, the agencies support the FAIR (Findable, Accessible, Interoperable, and Reusable) guiding principles for research data management and stewardship."* [emphasis added] The perspective of the funding agency on data ownership differs substantially from that of some researchers who incorrectly believe "they own their data". In Canada at least, the government makes it clear that when tax payers fund research (through the Tri-Agency) the research data is public property. Additionally the Tri-Agency Data Management Statement of Principles clearly indicates the responsibilities of funded researchers:

"Responsibilities of researchers include:

- incorporating data management best practices into their research;
- developing data management plans to guide the responsible collection, formatting, preservation and sharing of their data throughout the entire lifecycle of a research project and beyond;
- following the requirements of applicable institutional and/or funding agency policies and professional or disciplinary standards;
- acknowledging and citing datasets that contribute to their research; and
- staying abreast of standards and expectations of their disciplinary community."

As a result of this perspective on data, it's important that you think about structuring your data for reuse by yourself and others before you collect it. Toward this end, properly documenting your data file and analysis scripts is critical.

1.3 Begin with the end in mind

In this chapter we will walk you through the steps from data collection, data entry, loading raw data, and the creation of data you will analyze (analytic data) via pre-processing scripts. These steps are outlined in Figure 1.2. This figure makes a clear distinction between raw data and analytic data. Raw data refers to the data as you entered it into a spreadsheet or received it from survey software. Analytic data is the data that has been structured and processed so that it is ready for analysis. This pre-processing could include such things as identifying categorical variables to the computer, averaging multiple items into a scale score, and other tasks.

It's critical that you don't think of the analysis of your data as being completely removed from the data collection and data entry choices you make. Poor choices at the data collection and data entry stage can make your life substantially more complicated when it comes time to write the pre-processing script that will convert your raw data to analytic data. The mantra of this chapter is *begin with the end in mind.*

It's difficult to begin with the end in mind when you haven't read later chapters. So, here we will be providing you with some general thoughts around different approaches to structuring data files and the naming conventions you can use when creating those data files.

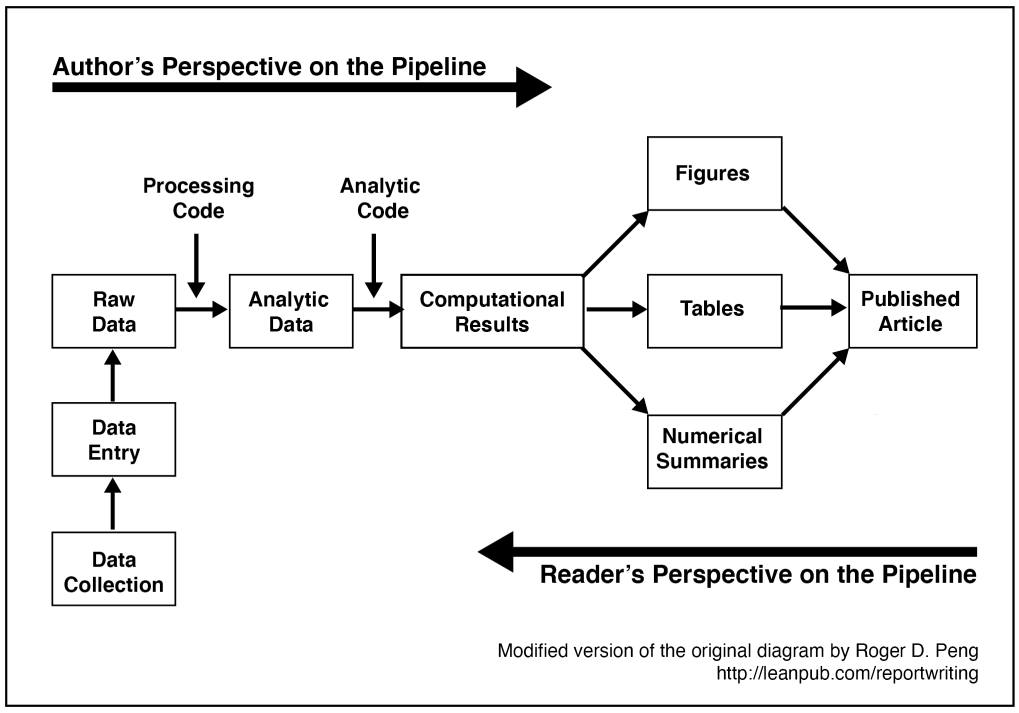


Figure 1.2: Data science pipeline by Roger Peng

Indeed, in this chapter we strongly advocate that you use a naming convention for file, variable, and column names. This convention will save you hours of hassles and permit easy application of certain tidyverse commands. However, we must stress that although the naming convention we advocate is based on the tidyverse style guide, it is not "right" or "correct" - there are other naming conventions you can use. Any naming convention is better than no naming convention. The naming convention we advocate here will solve many problems. We encourage to use this system for weeks or months over many projects - until you see the benefits of this system, and correspondingly its shortcomings. After you are well versed in the strengths/weaknesses of the naming conventions used here you may choose to create your own naming convention system.

1.3.1 Structuring data: Obtaining tidy data

When conducting analyses in R it is typically necessary to have data in a format called tidy data (Wickham 2014). Tidy data, as defined by Hadley, involves (among other requirements) that:

1. Each variable forms a column.
2. Each observation forms a row.

The tidy data format can be initially challenging for some researchers to understand because it is based on thinking about, and structuring data, in terms of observations/measurements instead of participants. In this section we will describe common approaches to entering animal and human participant data and how they can be done keeping the tidy data requirement in mind. It's not essential that data be entered in a tidy data format but it is essential that you enter data in a manner that makes it easy to later convert data to a tidy data format. When dealing with animal or human participant data it's common to enter data into a spreadsheet. Each row of the spreadsheet is typically used to represent a single participant and each column of the spreadsheet is used to represent a variable.

Between participant data. Consider Table 1.3 which illustrates between participant data for six human participants running 5 kilometers. The first column is id, which indicates there are six unique participants and provides and identification number for each of them. The second column is sex, which is a variable, and there is one observation per row, so sex also conforms to the tidy data specification. Finally, there is a last column elapsed_time which is a variable with one observation per row – also conforming to tidy data specification. Thus, single occasion between subject data like this conforms to the tidy data specification. There is usually nothing you need to do to convert between-participant data (or cross-sectional data) to be in a tidy data format.

Table 1.3: Between participant data entered one row per participant

id	sex	elapsed_time
1	male	40
2	female	35
3	male	38
4	female	33
5	male	42
6	female	36

Within participant data. Consider Table 1.4 which illustrates within participant data for six human participants running 5 kilometers - but on three different occasions. The first column is id, which indicates there are six unique participants and provides an identification number for each of them. The second column is sex, which is a variable, and there is one observation per row, so sex also conforms to the tidy data specification. Next, there are three different columns (march, may, july) each of which contains elapsed time (in minutes) for the runner in a different month. Elapsed run times are spread out over three columns so elapse_time is not in a tidy data format. Moreover, it's not clear from the data file that march, may, and july are levels of a variable called occasion. Nor is it clear that elapsed_times are recorded in each of those columns (i.e., the dependent variable is unknown/not labeled). Although this format is fine as a data entry format it clearly has problems associated with it when it comes time to analyze your data.

Table 1.4: Within participant data entered one row per participant

id	sex	march	may	july
1	male	40	37	35
2	female	35	32	30
3	male	38	35	33
4	female	33	30	28
5	male	42	39	37
6	female	36	33	31

Table 1.5: A tidy data version of the within participant data

id	sex	occasion	elapsed_time
1	male	march	40
1	male	may	37
1	male	july	35
2	female	march	35
2	female	may	32
2	female	july	30
3	male	march	38
3	male	may	35
3	male	july	33
4	female	march	33
4	female	may	30
4	female	july	28
5	male	march	42
5	male	may	39
5	male	july	37
6	female	march	36
6	female	may	33
6	female	july	31

Thus, a major problem with entering repeated measures data in the one row per person format is that there are hidden variables in the data and you need insider knowledge to know what the columns represent. That said, this is not necessarily a terrible way to enter your data as long as you have all of this missing information documented in a data code book.

Disadvantages one row per participant	Advantages one row per participant
1) Predictor variable (*occasion*) is hidden and spread over multiple columns 2) Unclear that each month is a level of the predictor variable *occasion* 3) Dependent variable (*elapsed_time*) is not indicated 4) Unclear that *elapsed_time* is the measurement in each month column	1) Easy to enter this way

Fortunately, the problems with Table 1.4 can be largely resolved by converting the data to a tidy data format. This can be done with the pivot_long() command that we will learn about in the Cookbook chapter. Thus, we can enter the data in the format of Table 1.4 and later convert it to a tidy data format. After this conversion the data will be appear as in Table 1.5. For elapsed_time variable this data is now in the tidy data format. Each row corresponds to a single elapsed_time observed. Each column corresponds to a single variable. Somewhat problematically, however, sex is repeated three times for each person (i.e., over the three rows) - and this can be confusing. However, if the focus in on analyzing elapsed time this tidy data format makes sense. Importantly, there is an id column for each participant so R knows that this information is repeated for each participant and is not confused by repeating the sex designation over three rows. Indirectly, this illustrates the importance of having an id column to indicate each unique participant.

Why did we walk you through this technical treatment of structuring data at this point in time - so that you pay attention to the advice that follows. You can see at this point that you may well need to restructure your data for certain analyses. The ability to do so quickly and easily depends upon following the advice in this chapter around naming conventions for variables and other aspects of your analyses. You can imagine the challenges for converting the data in Table 1.4 to the data in Table 1.5 by hand. You want to be able to automate that process and others - which is made substantially easier if you follow the forthcoming advice about naming conventions in the tidyverse.

1.4 Data collection considerations

Data can be collected in a wide variety of ways. Regardless of the method of data collection researchers typically come to data in one of two ways: 1) a research assistant enters the data into a spreadsheet type interface, or 2) the data is obtained as the output from computer software (e.g., Qualtrics, SurveyMonkey, Noldus, etc.).

Regardless of the approach, it is critical to name your variables appropriately. For those using software, such as Qualtrics, this means setting up the software to use appropriate variable names PRIOR to data collection - so the exported file has desirable column names. For spreadsheet users, this means setting up the spreadsheet in which the data will be recorded with column names that are amenable to the future analyses you want to conduct.

Although failure to take this thoughtful approach at the data collection stage can be overcome - it is only overcome with substantial manual effort. Therefore, as noted previously, we strongly encourage you to follow the naming conventions we espouse here when you set up your data recording regime. Additionally, we encourage you to give careful thought in advance to the codes you will use to record missing data.

1.4.1 File naming conventions

I **strongly** suggest you check out these excellent slides by Danielle Navarro on file name convention best practices.

1.4.2 Data column naming conventions

To make your life easier down the road, it is critical you set up your spreadsheet or online survey such that it uses a naming convention prior to data collection. The naming conventions suggested here are adapted from the tidyverse style guide.

- Lowercase letters only
- If using multiple words in a name (a good idea), only use the underscore ("_") character to separate words in the name.
- Avoid short decontextualized variable names like q1, q2, q3, etc.
- Do use moderate length column names. Aim to achieve a unique prefix for related columns so that those columns can be selected using the starts_with() command discussed in the previous chapter. Be sure to avoid short two or three letter prefixes for item names. Instead, use unique moderate length item prefixes so that it will be easy to select those columns using start_with() such that you don't accidentally get additionally columns you don't want - that have a similar prefix.
- Likert items. Be sure to indicate the following information in the name of each Likert item or you will make your life substantially more complicated when you start to analyze your data. The information to include: a) the name of the measure, b) the item number for the measure, c) that it is a Likert item, d) the number of Likert response options, and e) whether the item is reverse keyed. That's five things to include in each Likert item name. But it's easy to do so. Consider two "affective commitment" items, the 2nd and 3rd items on scale. Both items use a 5-point Likert response format. However, item 3 is reverse keyed. **Names that conform to this convention are: aff_com3_likert5, aff_com3_likert5rev.** Using this naming convention ensures you can easily select and convert the items later. You can select by "likert5", "likert5rev" or select by "aff_com".
- If you have a column name that represents the levels of two repeated measures variables only use the underscore character to separate the levels of the different variables. See within-participant ANOVA section below for details.
- Column content. Avoid numerical representation of categorical variables. Don't use 1 or 2 to represent a variable like sex. Use male and female in your spreadsheet - likewise in your survey program. Similarly, for between participant variables like drug_condition don't use 1 or 2 use "drug" and "placebo" but the actual drug name would be even better than the word "drug." Following this approach ensure the data can "stand alone" for

resuse by others (especially if a data codebook (example) is not provided.) Note you will covert categorical variables such as sex (male/female) to numeric representations in your script - but then it will be clear what each value means.

1.4.3 Likert-type items

A Likert-type item is typically composed of a statement with which participants are asked to agree or disagree. For example, participants could be asked to indicate the extent to which they agree with a number of statements such as "I like my job". Then they would be presented with response scale such as: 1 - Strongly Disagree, 2 - Moderately Disagree, 3 - Neutral, 4, Moderately Agree, 5 - Strongly Agree. A common question is, how should I enter the data?

Export text responses not numbers Software such as Qualtrics gives you the option of exporting the label (e.g., "Strongly Agree") or a value (e.g., 1). Make sure you export the text lable ("Strongly Agree). That way, the data file stands alone - and doesn't require additional knowledge to know what 1 means. You can easily convert the labels to numbers later.

- **High numbers should be associated with more of the construct being measured.** When designing your survey or data collection tools, it is important that you set the response options appropriately. If your scale measures job satisfaction, it is important that you collect data in a manner that ensures high numbers on the job satisfaction scale indicate high levels of job satisfaction. Therefore, assigning numbers makes sense using the 5-point scale: 1 - Strongly Disagree, 2 - Moderately Disagree, 3 - Neutral, 4, Moderately Agree, 5 - Strongly Agree. With this approach high response numbers indicate more job satisfaction. However, using the opposite scale would not make sense: 1 - Strongly Agree, 2 - Moderately Agree, 3 - Neutral, 4, Moderately Disagree, 5 - Strongly Disagree. With this opposite scale high numbers on a job satisfaction scale would indicate lower levels of job satisfaction - a very confusing situation. Avoid this situation, assign numbers so that higher numbers are associated with more of the construct being measured.
- **Use appropriate item names.** As described in the naming convention section, use moderate length names with different labels for each subscale.
- **Use moderate length column names unique to each subscale**. Imagine you have a survey with an 18-item commitment scale (Meyer, Allen, and Smith 1993) composed of three 6-item subscales: affective, normative, and continuance commitment. It would be a poor choice to prefix the labels of all 18 columns in your data with "commit" such that the names would be commit1, commit2, commit3, etc. The problem with this approach is that it fails to distinguish among the three subscales in naming convention; making it impossible to select the items for a single subscale using starts_with(). A better, but still poor choice for a naming convention would be use use a two letter prefix for the three scale such ac, nc, and cc. This would result in names for the columns like ac1, ac2, ac3, etc. This is an improvement because you could apparently (but likely not) select the columns using starts_with("ac"). The problem with these short names is that there

could be many columns in data set that start with "ac" beside the affective commitment items. You might want to select the affective commitment items using starts_with("ac"); but you would get all the affective commitment item columns; but also all the columns measuring other variables that also start with "ac". Therefore, it's a good idea to use a moderate length unique prefix for column names. For example, you might use prefixes like affect_com, norm_com, and contin_com for the three subscales. But see below because you need to include more than this in each name.

- **Indicate these 5 things in each Likert item name.** Be sure to indicate the following information in the name of each Likert item or you will make your life substantially more complicated when you start to analyze your data. The information to include: **1)** the name of the measure, **2)** the item number for the measure, **3)** that it is a Likert item, **4)** the number of Likert response options, and **5)** whether the item is reverse keyed. That's five things to include in each Likert item name. But it's easy to do so. Consider two "affective commitment" items, the 2nd and 3rd items on scale. Both items use a 5-point Likert response format. However, item 3 is reverse keyed. **Names that conform to this convention are: aff_com2_likert5, aff_com3_likert5rev.** Using this naming convention ensure you can easily select and convert the items later. You can select by "likert5", "likert5rev" or select by "aff_com" (or both).

Indicate in the item name if the item is reversed keyed. Sometimes with Likert-type items, an item is reverse keyed. For example, on a job satisfaction scale, participants will typically respond to items that reflect job satisfaction using the scale: 1 - Strongly Disagree, 2 - Moderately Disagree, 3 - Neutral, 4, Moderately Agree, 5 - Strongly Agree. Higher numbers indicate more job satisfaction. Sometimes, however, some items will use the same 1 to 5 response scale but be worded in the opposite manner such as "I hate my job". Responding with a 5 to this item would indicate high job *dissatisfaction*. But the columns for job satisfaction items should have high values that indicate high job satisfaction not high job dissatifaction. Consequently, we flag the names of columns with reversed responses (i.e., reverse-key items) so that we know to treat those column differently later. Columns with reverse-keyed items need to be processed by a script so that the values are flipped and scored in the right direction. The procedure for doing so is outlined in the next point.

Indicate in the item name the range for reverse-key items. If an item is reverse keyed, the process for the flipping the scores depends upon the range of a scale. Although 5-point scales are common, any number of points are possible. The process for correcting a reverse-key item depends upon: 1) the number of points on the scale, and 2) the range of the points on the scale. The reverse-key item correction process is different for an item that uses a 5-point scale ranging from 1 to 5 versus from 0 to 4. Both are 5-point scales but your correction process will be different. Therefore, for reverse-key items add a suffix at the end of each item name that indicates an item is reverse keyed and the range of the item. For example, if the third job satisfaction item was reversed keyed on scale using a 1 to 5 response format you might name the item: job_sat3_likert5rev. The suffix "_likert5rev" indicates the item is Likert item that

is reverse keyed and the range of responses used on the item is 1 to 5. Be sure to set up your survey with this naming convention when you collect your data.

- If you collect items over multiple time points use a prefix with a short code to indicate the time followed by an underscore. For example, if you had a multi-item self-esteem scale you might call the column for the first time "t1_esteem1_likert5rev". This indicate that you have for time 1 (t1), the first self-esteem item (esteem1) and that item is a likert item that is reverse keyed on a 1 to 5 scale.

1.5 Example: Single Occassion Survey

This section outlines a workflow appropriate for when you have cross-sectional single occasion survey data. Examples for other designs are presented in the Cookbook chapter. The data corresponds to a design where the researcher has measured, age, sex, eye color, self-esteem, and job satisfaction. Two of these, self-esteem and job satisfaction, were measured with multi-item scales with reverse-keyed items.

To Begin:

- Use the Files tab to confirm you have the data: data_item_scoring.csv
- Start a new script for this example. Don't forget to start the script name with "script_".

```
# Date: YYYY-MM-DD
# Name: your name here
# Example: Single occasion survey

# Load data
library(tidyverse)

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

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

```
Rows: 300 Columns: 14
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (2): sex, eye_color
dbl (12): id, age, esteem1_likert5, esteem2_likert5, esteem3_likert5, esteem...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We load the initial data into a raw_data_survey but immediately make a copy we will work with called analytic_data_survey. It's good to keep a copy of the raw data for reference if you encounter problems.

```
analytic_data_survey <- raw_data_survey
```

Remove empty row and columns from your data using the remove_empty_cols() and remove_empty_rows(), respectively. As well, clean the names of your columns to ensure they conform to tidyverse naming conventions.

```
library(janitor)
```

```
Attaching package: 'janitor'

The following objects are masked from 'package:stats':

    chisq.test, fisher.test
```

```
# Initial cleaning
analytic_data_survey <- analytic_data_survey %>%
  remove_empty("rows") %>%
  remove_empty("cols") %>%
  clean_names()
```

You can confirm the column names following our naming convention with the glimpse command - and see the data type for each column.

```
glimpse(analytic_data_survey)
```

```
Rows: 300
Columns: 14
$ id                <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ~
$ age               <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, NA, 20, 17, 24,~
$ sex               <chr> "male", "female", "male", "female", "male", "female~
$ eye_color         <chr> "blue", "brown", "hazel", "blue", NA, "hazel", "blu~
$ esteem1_likert5   <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, NA, NA, 3, 3, 3~
$ esteem2_likert5   <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, NA, 3, 2, 2, 2,~
$ esteem3_likert5   <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, 4, NA, 4, NA, ~
$ esteem4_likert5   <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, 3, 3, 4, NA, 3~
```

```
$ esteem5_likert5rev <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2, 3, 2, 2, 3, 3, NA, ~
$ jobsat1_likert5    <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4, 3, 3, 4, 3, ~
$ jobsat2_likert5rev <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3, 1, 3, 2, 1, 1, 2,~
$ jobsat3_likert5    <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 2,~
$ jobsat4_likert5    <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5, 4, 4, 4, 5, ~
$ jobsat5_likert5    <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA, 4, 5, 4, 4, 4~
```

1.5.1 Creating factors

Following initial cleaning, we identify categorical variables as factors. If you plan to conduct an ANOVA - it's critical that all predictor variables are converted to factors. Inspect the glimpse() output - if you followed our data entry naming conventions, categorical variables should be of the type character.

```
glimpse(analytic_data_survey)
```

```
Rows: 300
Columns: 14
$ id                 <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ~
$ age                <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, NA, 20, 17, 24,~
$ sex                <chr> "male", "female", "male", "female", "male", "female~
$ eye_color          <chr> "blue", "brown", "hazel", "blue", NA, "hazel", "blu~
$ esteem1_likert5    <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, NA, NA, 3, 3, 3~
$ esteem2_likert5    <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, NA, 3, 2, 2, 2,~
$ esteem3_likert5    <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, 4, NA, 4, NA, ~
$ esteem4_likert5    <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, 3, 3, 4, NA, 3~
$ esteem5_likert5rev <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2, 3, 2, 2, 3, 3, NA, ~
$ jobsat1_likert5    <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4, 3, 3, 4, 3, ~
$ jobsat2_likert5rev <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3, 1, 3, 2, 1, 1, 2,~
$ jobsat3_likert5    <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 2,~
$ jobsat4_likert5    <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5, 4, 4, 4, 5, ~
$ jobsat5_likert5    <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA, 4, 5, 4, 4, 4~
```

We have two variables, sex and eye_color, that are categorical variable of type character (i.e., chr). The participant id column is categorical as well, but of type double (i.e., dbl) which is a numeric column. You can quickly convert all character columns to factors using the code below:

```
analytic_data_survey <- analytic_data_survey %>%
  mutate(across(.cols = c(sex, eye_color),
                .fns = as_factor))
```

The participant identification number in the id column is a numeric column, so we have to handle that column on its own.

```
analytic_data_survey <- analytic_data_survey %>%
  mutate(id = as_factor(id))
```

You can ensure all of these columns are now factors using the glimpse() command.

```
glimpse(analytic_data_survey)
```

```
Rows: 300
Columns: 14
$ id                 <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ~
$ age                <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, NA, 20, 17, 24,~
$ sex                <fct> male, female, male, female, male, female, male, fem~
$ eye_color          <fct> blue, brown, hazel, blue, NA, hazel, blue, brown, h~
$ esteem1_likert5    <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, NA, NA, 3, 3, 3~
$ esteem2_likert5    <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, NA, 3, 2, 2, 2,~
$ esteem3_likert5    <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, 4, NA, 4, NA, ~
$ esteem4_likert5    <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, 3, 3, 4, NA, 3~
$ esteem5_likert5rev <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2, 3, 2, 2, 3, 3, NA, ~
$ jobsat1_likert5    <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4, 3, 3, 4, 3, ~
$ jobsat2_likert5rev <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3, 1, 3, 2, 1, 1, 2,~
$ jobsat3_likert5    <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 2,~
$ jobsat4_likert5    <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5, 4, 4, 4, 5, ~
$ jobsat5_likert5    <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA, 4, 5, 4, 4, 4~
```

Inspect the output of the glimpse() command and make sure you have converted all categorical variables to factors - especially those you will use as predictors.

Note: If you have factors like sex that have numeric data in the column (e.g, 1 and 2) instead of male/female you need to handle the situation differently. The preceding section, Experiment: Within N-way, illustrates how to handle this scenario.

1.5.2 Factor screening

Inspect the levels of each factor carefully. Make sure the factor levels of each variable are correct. Examine spelling and look for additional unwanted levels. For example, you wouldn't want to have the following levels for sex: male, mmale, female. Obviously, mmale is an incorrectly typed version of male. Scan all the factors in your data for erroneous factor levels. The code below displays the factor levels:

```
analytic_data_survey %>%
  select(where(is.factor)) %>%
  summary()
```

```
       id            sex      eye_color
 1      :  1   male    :147   blue : 99
 2      :  1   female  :149   brown: 98
 3      :  1   intersex:  2   hazel:100
 4      :  1   NA's    :  2   NA's :  3
 5      :  1
 6      :  1
 (Other):294
```

Also inspect the output of the above summary() command paying attention to the order of the levels in the factors. The order influences how text output and graphs are generated. In these data, the sex column has two levels: male and female in that order. Below we adjust the order of the sex variable because we want the x-axis of a future graph to display columns in the left to right order: female, male.

```
analytic_data_survey <- analytic_data_survey %>%
  mutate(sex = fct_relevel(sex,
                           "intersex",
                           "female",
                           "male"))
```

For eye color, we want a future graph to have the most common eye colors on the left so we reorder the factor levels:

```
analytic_data_survey <- analytic_data_survey %>%
  mutate(eye_color = fct_infreq(eye_color))
```

You can see the new order of the factor levels with summary():

```
analytic_data_survey %>%
  select(where(is.factor)) %>%
  summary()
```

```
       id            sex      eye_color
 1      :  1   intersex:  2   hazel:100
 2      :  1   female  :149   blue : 99
```

```
 3       :  1   male    :147   brown: 98
 4       :  1   NA's    :  2   NA's :  3
 5       :  1
 6       :  1
 (Other):294
```

1.5.3 Numeric screening

For numeric variables, it's important to find and remove impossible values. For example, in the context of this example you want to ensure none of the Likert responses are impossible (e.g., outside the 1- to 5-point rating scale) or clearly data entry errors.

Because we have several numeric columns that we are screening, we use the skim() command from the skimr package. The skim() command quickly provides basic descriptive statistics. In the output for this command there are also several columns that begin with p: p0, p25, p50, p75, and p100 (p25 and p75 omitted in output due to space). These columns correspond to the 0th, 25th, 50th, 75th, and 100th percentiles, respectively. The minimum and maximum values for the data column are indicated under the p0 and p100 labels. The median is the 50th percentile (p50). The interquartile range is the range between p25 and p75.

Start by examining the range of non-scale items. In this case it's only age. Examine the output to see if any of the age values are unreasonable. As noted, in the output p0 and p100 indicate the 0th percentile and the 100th percentile; that is the minimum and maximum values for the variable. Check to make sure none of the age values are unreasonably low or high. If they are, you may need to check the original data source or replace them with missing values.

```
library(skimr)
analytic_data_survey %>%
  select(age) %>%
  skim()
```

```
  skim_variable n_missing  mean   sd p0 p50 p100
1           age         3 20.52 2.05 17  20   24
```

With respect to the multi-item scales, it makes sense to look at sets of items rather than all of the items at once. This is because sometimes items from different scales use different response ranges. For example, one measure might use a response scale with a range from 1 to 5; whereas another measure might use a response scale with a range from 1 to 7. This is undesirable from a psychometric point of view, as discussed previously, but if it happens in your data - look at the scale items separately to make it easy to see out of range values.

We begin by looking at the items in the first scale, self-esteem. Possible items responses for this scale range from 1 to 5; make sure all responses are in this range. If any values fall outside

this range, you may need to check the original data source or replace them with missing values - as described previously.

```
analytic_data_survey %>%
  select(starts_with("esteem")) %>%
  skim()
```

```
       skim_variable n_missing mean   sd p0 p50 p100
1    esteem1_likert5        24 3.39 0.54  3   3    5
2    esteem2_likert5        28 2.35 0.48  2   2    3
3    esteem3_likert5        31 3.96 0.37  3   4    5
4    esteem4_likert5        15 3.54 0.50  3   4    4
5 esteem5_likert5rev        35 2.22 0.47  1   2    3
```

Follow the same process for the job satisfaction items. Write that code on your own now.

Possible item responses for the job satisfaction scale range from 1 to 5, make sure all responses are in this range. If any values fall outside this range, you may need to check the original data source or replace them with missing values - as described previously.

```
analytic_data_survey %>%
  select(starts_with("jobsat")) %>%
  skim()
```

```
       skim_variable n_missing mean   sd p0 p50 p100
1    jobsat1_likert5        25 3.34 0.51  3   3    5
2 jobsat2_likert5rev        27 1.51 0.61  1   1    3
3    jobsat3_likert5        28 2.84 0.37  2   3    3
4    jobsat4_likert5        35 4.29 0.70  3   4    5
5    jobsat5_likert5        24 4.57 0.61  3   5    5
```

1.5.4 Scale scores

For each person, scale scores involve averaging scores from several items to create an overall score. The first step in the creation of scales is correcting the values of any reverse-keyed items.

1.5.4.1 Reverse-key items

The way you deal with reverse-keyed items depends on how you scored them. Imagine you had a 5-point scale. You could have scored the scale with the values 1, 2, 3, 4, and 5. Alternatively, you could have scored the scale with the values 0, 1, 2, 3, and 4. The mathematical approach you use to correcting reverse-keyed items depends upon whether the scale starts with 1 or 0.

In this example, we scored the data using the value 1 to 5; so that is the approach illustrated here. See the extra information box for details on how to fixed reverse-keyed items when the scale begins with zero.

In this data file all the reverse-keyed items were identified with the suffix "_likert5rev" in the column names. This suffix indicates the item was reverse keyed and that the original scale used the response points 1 to 5. We can see those items with the glimpse() command below. Notice that there are two reverse-keyed items - each on difference scales.

```
analytic_data_survey %>%
  select(ends_with("_likert5rev")) %>%
  glimpse()
```

```
Rows: 300
Columns: 2
$ esteem5_likert5rev <dbl> 2, 2, 2, 2, 2, NA, NA, 2, 2, 2, 3, 2, 2, 3, 3, NA, ~
$ jobsat2_likert5rev <dbl> 1, 1, 1, NA, 1, 1, 2, 1, 2, 2, 3, 1, 3, 2, 1, 1, 2,~
```

To correct a reverse-keyed item where the lowest possible rating is 1 (i.e, 1 on a 1 to 5 scale), we simply subtract all the scores from a value one more than the highest point possible on the scale (i.e., one more than 5). For example, if a 1 to 5 response scale was used we subtract each response from 6 to obtain the recoded value.

Original value	Math	Recoded value
1	6 - 1	5
2	6 - 2	4
3	6 - 3	3
4	6 - 4	2
5	6 - 5	1

The code below:

- selects columns that end with "_likert5rev" (i.e., both esteem and jobsat scales)
- subtracts the values in those columns from 6

- renames the columns by removing "_likert5rev" from the name because the reverse coding is complete

```
analytic_data_survey <- analytic_data_survey %>%
  mutate(6 - across(.cols = ends_with("_likert5rev")) ) %>%
  rename_with(.fn = str_replace,
              .cols = ends_with("_likert5rev"),
              pattern = "_likert5rev",
              replacement = "_likert5")
```

You can use the glimpse() command to see the result of your work. If you compare these new values to those obtained from the previous glimpse() command you can see they have changed. Also notice the column names no longer indicate the items are reverse keyed.

```
glimpse(analytic_data_survey)
```

```
Rows: 300
Columns: 14
$ id              <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,~
$ age             <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, NA, 20, 17, 24, 17~
$ sex             <fct> male, female, male, female, male, female, male, female~
$ eye_color       <fct> blue, brown, hazel, blue, NA, hazel, blue, brown, haze~
$ esteem1_likert5 <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, NA, NA, 3, 3, 3, 3~
$ esteem2_likert5 <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, NA, 3, 2, 2, 2, 2,~
$ esteem3_likert5 <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, 4, NA, 4, NA, NA,~
$ esteem4_likert5 <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, 3, 3, 4, NA, 3, 3~
$ esteem5_likert5 <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4, 3, 4, 4, 3, 3, NA, 3, ~
$ jobsat1_likert5 <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4, 3, 3, 4, 3, 3, ~
$ jobsat2_likert5 <dbl> 5, 5, 5, NA, 5, 5, 4, 5, 4, 4, 3, 5, 3, 4, 5, 5, 4, 3,~
$ jobsat3_likert5 <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 2, 2,~
$ jobsat4_likert5 <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5, 4, 4, 4, 5, 4, ~
$ jobsat5_likert5 <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA, 4, 5, 4, 4, 4, 4~
```

> **Caution**
>
> If your scale had used response options numbered 0 to 4 the math is different. For each item you would use subtract values from the highest possible point (i.e, 4) instead of one larger than the highest possible point.

Original value	Math	Recoded value
0	4 - 0	4
1	4 - 1	3
2	4 - 2	2
3	4 - 3	1
4	4 - 4	0

Thus, the mutate command would instead be:
mutate(4 - across(.cols = ends_with("_likert5rev")))

1.5.4.2 Creating scores

The process we use for creating scale scores deletes item-level data from analytic_data_survey. This is a desirable aspect of the process because it removes information that we are no longer interested in from our analytic data. That said, before we create scale score, we create a backup on the item-level data called analytic_data_survey_items. We will need to use this backup later to compute the reliability of the scales we are creating.

```
analytic_data_survey_items <- analytic_data_survey
```

We want to make a self_esteem scale and plan to select items using starts_with("esteem"). But prior to doing this we make sure the start_with() command only gives us the items we want - and not additional unwanted items. The output below confirms there are not problems associated with using starts_with("esteem").

```
analytic_data_survey %>%
  select(starts_with("esteem")) %>%
  glimpse()
```

```
Rows: 300
Columns: 5
$ esteem1_likert5 <dbl> 3, 4, 4, 3, 3, 3, 3, 4, 4, 4, 3, 4, NA, NA, 3, 3, 3, 3~
$ esteem2_likert5 <dbl> 2, 3, 3, 2, 2, 3, 2, 3, 3, 3, 2, 2, NA, 3, 2, 2, 2, 2,~
$ esteem3_likert5 <dbl> 4, 4, 4, 3, 4, 4, NA, 4, 4, 3, 4, 4, 4, NA, 4, NA, NA,~
$ esteem4_likert5 <dbl> 3, 4, 4, 3, 4, 4, 4, 4, 3, 4, NA, 4, 3, 3, 4, NA, 3, 3~
$ esteem5_likert5 <dbl> 4, 4, 4, 4, 4, NA, NA, 4, 4, 4, 3, 4, 4, 3, 3, NA, 3, ~
```

Likewise, we want to make a job_sat scale and plan to select items using starts_with("jobsat"). The code and output below using starts_with("jobsat") only returns the items we are interested in.

```
analytic_data_survey %>%
  select(starts_with("jobsat")) %>%
  glimpse()
```

```
Rows: 300
Columns: 5
$ jobsat1_likert5 <dbl> 3, 5, 4, 3, 3, 3, 3, 5, 3, 3, 3, 4, 4, 3, 3, 4, 3, 3, ~
$ jobsat2_likert5 <dbl> 5, 5, 5, NA, 5, 5, 4, 5, 4, 4, 3, 5, 3, 4, 5, 5, 4, 3,~
$ jobsat3_likert5 <dbl> 3, NA, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 2, 2,~
$ jobsat4_likert5 <dbl> NA, 5, 5, 4, 4, 4, 4, 5, NA, 4, NA, 5, 4, 4, 4, 5, 4, ~
$ jobsat5_likert5 <dbl> 5, NA, 5, 4, 5, 4, 4, 5, 5, 5, 4, NA, 4, 5, 4, 4, 4, 4~
```

We calculate the scale scores using the rowwise() command. The mean() command provides the mean of columns by default - not people. We use the rowwise() command in the code below to make the mean() command work across columns (within participants) rather than within columns. The mutate command calculates the scale score for each person. The c_across() command combined with the starts_with() command ensures the items we want averaged together are the items that are averaged together. Notice there is a separate mutate line for each scale. The ungroup() command turns off the rowwise() command. We end the code block by removing the item-level data from the data set.

Important: Take note of how we name the scale variables (e.g., self_esteem, job_sat). We use a slightly different convention than our items. That is, these scale labels were picked so that they would *not* be selected by a starts_with("esteem") or starts_with("jobsat"). Why - because we later use those commands to remove the item-level data. We would not want the command designed to remove the item-level data to also remove the scale we just calculated! This example illustrates how carefully you need to think about your naming conventions.

```
analytic_data_survey <- analytic_data_survey %>%
  rowwise() %>%
  mutate(self_esteem = mean(c_across(starts_with("esteem")),
                            na.rm = TRUE)) %>%
  mutate(job_sat = mean(c_across(starts_with("jobsat")),
                            na.rm = TRUE)) %>%
  ungroup() %>%
  select(-starts_with("esteem")) %>%
  select(-starts_with("jobsat"))
```

We can see our data now has the self_esteem column, a job_sat column, and that all of the item-level data has been removed.

```r
glimpse(analytic_data_survey)
```

```
Rows: 300
Columns: 6
$ id          <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,~
$ age         <dbl> 23, 22, 18, 23, 22, 17, 23, 22, 17, NA, 20, 17, 24, 17, NA~
$ sex         <fct> male, female, male, female, male, female, male, female, ma~
$ eye_color   <fct> blue, brown, hazel, blue, NA, hazel, blue, brown, hazel, b~
$ self_esteem <dbl> 3.200000, 3.800000, 3.800000, 3.000000, 3.400000, 3.500000~
$ job_sat     <dbl> 4.000000, 5.000000, 4.400000, 3.500000, 4.000000, 3.800000~
```

You now have two data sets analytic_data_survey and analytic_data_survey_items. You can calculate descriptive statistics, correlations and most analyses using the analytic_data_survey. To obtain the reliability of the scales you just created though you will need to use the analytic_data_survey_items. Both sets of data are ready for analysis.

2 Qualtrics

2.1 Survey Response

Designing a survey process in way that ensure a high response rate is crucial to obtaining quality data. A low response rate can be problematic for many reasons. I encourage you to check out the resource below to help you design a survey process that maximizes your response rate.

Internet, Phone, Mail, and Mixed-Mode Surveys: The Tailored Design Method

2.2 Overview

There are two tasks covered in this chapter: **1)** importing surveys into Qualtrics and **2)** moving data from Qualtrics to R. An overview of each process is below.

1. Importing items into Qualtrics

- Create a survey codebook in Excel (or similar program) that has each item and the response options for it
- Convert the survey codebook to Qualtrics Advanced Text Format
- Import the items
- Tweak the survey after import

2. Qualtrics Data to R

- Export the raw data from Qualtrics with maximal information in the data file
- Load the raw data into R
- Convert the raw data to analytic data. This conversion includes: assigning values to response options (e.g., "Strongly Disagree" to a numeric value), flipping response for reverse-worded (i.e., reverse-keyed) items, and creating scale scores.

The overall steps are quite simple and I walk you through each step in detail below.

2.3 Required

The files below are used in this chapter. Right click to save each file.

Relevant files
survey_codebook.csv
items_qualtrics_format.txt
data_qualtrics.csv
script_qualtrics.R

The following CRAN packages must be installed:

Required CRAN Packages
tidyverse
janitor
remotes

The following GitHub packages must be installed:

Required GitHub Packages
dstanley4/qualtricsMaker

A GitHub package can be installed using the code below:

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

2.4 Items to Qualtrics

There are two approaches to entering survey items into Qualtrics.

- Enter the items one at a time using the Qualtrics web interface.
- Create a text file in the Qualtrics Advanced Text Format and import the items. Be warned though a bit of tweaking is often still needed using the web interface.

Here we focus only on the second approach to entering items into Qualtrics.

2.4.1 Create survey codebook

The easy way to create items in the Advanced Text Format is to use the qualtricsMaker package. With this package you create a spreadsheet with the items and then convert that spreadsheet to the Advanced Text Format.

Consider a scenario where we want to create a survey that contains 2 demographics items, 18 commitment items, and 4 job satisfaction items. The *three-component model of commitment* items are from (Meyer, Allen, and Smith 1993) and the *job satisfaction* items are from (Thompson and Phua 2012).

We begin by creating a survey codebook spreadsheet in Excel (or similar program). The spreadsheet should have the columns with the names: block, item_name, item_text, type, response_options. The completed file (survey_codebook.csv) is illustrated below. I provide a detailed description of what should be placed in each column after the figure. Remember to work from the guiding principle that one row is used for one item/question.

	A	B	C	D	
1	block	item_name	item_text	type	response_options
2	Demographics	year_of_birth	In what year where you born?	MC_dropdown	1940;1941;1942;1943;1944;1945;
3	Demographics	sex	What sex are you?	MC_dropdown	male; female; intersex
4	Commitment	aff_com1_likert7	I would be very happy to spend the rest of my career with this organization.	matrix	Strongly Disagree;Moderately Dis
5	Commitment	aff_com2_likert7	I really feel as if this organization's problems are my own.	matrix	Strongly Disagree;Moderately Dis
6	Commitment	aff_com3_likert7rev	I do not feel a strong sense of "belonging" to my organization.	matrix	Strongly Disagree;Moderately Dis
7	Commitment	aff_com4_likert7rev	I do not feel "emotionally attached" to this organization.	matrix	Strongly Disagree;Moderately Dis
8	Commitment	aff_com5_likert7rev	I do not feel like "part of the family" at my organization.	matrix	Strongly Disagree;Moderately Dis
9	Commitment	aff_com6_likert7	This organization has a great deal of personal meaning for me.	matrix	Strongly Disagree;Moderately Dis
10	Commitment	contin_com1_likert7	Right now, staying with my organization is a matter of necessity as much as desire.	matrix	Strongly Disagree;Moderately Dis
11	Commitment	contin_com2_likert7	It would be very hard for me to leave my organization right now, even if I wanted to.	matrix	Strongly Disagree;Moderately Dis
12	Commitment	contin_com3_likert7	Too much of my life would be disrupted if I decided I wanted to leave my organization now.	matrix	Strongly Disagree;Moderately Dis
13	Commitment	contin_com4_likert7	I feel that I have too few options to consider leaving this organization.	matrix	Strongly Disagree;Moderately Dis
14	Commitment	contin_com5_likert7	If I had not already put so much of myself into this organization, I might consider working elsewhere.	matrix	Strongly Disagree;Moderately Dis
15	Commitment	contin_com6_likert7	One of the few negative consequences of leaving this organization would be the scarcity of available alternatives.	matrix	Strongly Disagree;Moderately Dis
16	Commitment	norm_com1_likert7rev	I do not feel any obligation to remain with my current employer.	matrix	Strongly Disagree;Moderately Dis
17	Commitment	norm_com2_likert7	Even if it were to my advantage, I do not feel it would be right to leave my organization now.	matrix	Strongly Disagree;Moderately Dis
18	Commitment	norm_com3_likert7	I would feel guilty if I left my organization now.	matrix	Strongly Disagree;Moderately Dis
19	Commitment	norm_com4_likert7	This organization deserves my loyalty.	matrix	Strongly Disagree;Moderately Dis
20	Commitment	norm_com5_likert7	I would not leave my organization right now because I have a sense of obligation to the people in it.	matrix	Strongly Disagree;Moderately Dis
21	Commitment	norm_com6_likert7	I owe a great deal to my organization.	matrix	Strongly Disagree;Moderately Dis
22	Feelings About Your Job	job_aff1_likert5	I find real enjoyment in my jobI find real enjoyment in my job	MC	Strongly Disagree; Disagree; Neu
23	Feelings About Your Job	job_aff2_likert5	I like my job better than the average person	MC	Strongly Disagree; Disagree; Neu
24	Feelings About Your Job	job_aff3_likert5	Most days I am enthusiastic about my job	MC	Strongly Disagree; Disagree; Neu
25	Feelings About Your Job	job_aff4_likert5	I feel fairly well satisfied with my job	MC	Strongly Disagree; Disagree; Neu

2.4.1.1 block column

Enter any text you wish to use as a label for a block of items.

2.4.1.2 item_name column

To make your life easier down the road, it is critical you set up your spreadsheet or online survey such that it uses a naming convention prior to data collection. Recall the naming convention from the previous chapter.

2.4.1.3 item_text column

The item_text column contains the text for each item. Note that if you use commas in your item text do not save this file as a .csv file - it will not work. Rather save it as .tsv file (tab separated values). Then use read_tsv command instead of the read_csv command in the code that follows later.

2.4.1.4 type column

Code for type column	Qualtrics Item Type	Additional Information
matrix	matrix	If the first item in a block has type *matrix* all items in the block will be used to construct the matrix question. **Unfortunately, importing item_names for matrix questions is not supported by Qualtrics. You will need to manually restore your item_names following the directions below for matrix items.**
MC	multiple choice vertical format	
MC_horizontal	multiple choice horizontal format	
MC_multi_horizontal	multiple choice horizontal format multiple answers	
MC_select	select box	
MC_multi_select	select multiple boxes	
MC_dropdown	dropdown	

2.4.1.5 response_options column

The Likert items for the different blocks use different response options. The commitment items use a 7-point response option whereas the job satisfaction items use a 5-point response option.

2.4.1.5.1 Year of birth

We used a number of response options to indicate Year of Birth. We entered them as per below. Each response option is separated by a semi-colon. Each of the years below will be in the dropdown button.

Entered in the response_option column as below. Each option separated by a semicolon.

1940;1941;1942;1943;1944;1945;1946;1947;1948;1949;1950;1951;1952;1953;1954;1955;1956;1957;1958;1959;1960;196

2.4.1.5.2 Sex 3 points

Please note that in psychology we distinguish between sex and gender. We use the term "sex" to refer to underlying biology. In contrast, we use the term "gender" to refer a psychological construct (i.e., gender identity). A person's sex and gender may be quite different. Depending on the theory used in a study, you might be interested in sex or gender or both. In this example, we are using sex but not gender. If you were asking about gender - you would have a much longer (and quite different) list of options.

Entered in the response_option column as below. Each option separated by a semicolon.

male; female; intersex

2.4.1.5.3 Commitment 7 points

Entered in the response_option column as below. Each option separated by a semicolon.

Strongly Disagree;Moderately Disagree;Slightly Disagree;Neither Agree nor Disagree;Slightly Agree;Moderately Agree;Strongly Agree

2.4.1.5.4 Job Satisfaction 5 points

Entered in the response_option column as below. Each option separated by a semicolon.

Strongly Disagree; Disagree; Neutral; Agree; Strongly Agree

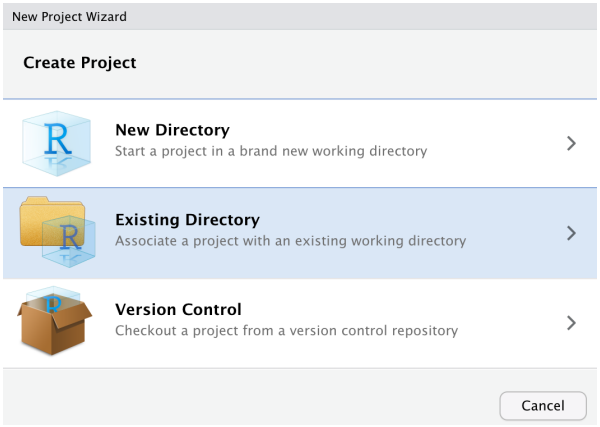
2.4.2 Convert to Qualtrics format

2.4.2.1 Create TXT file

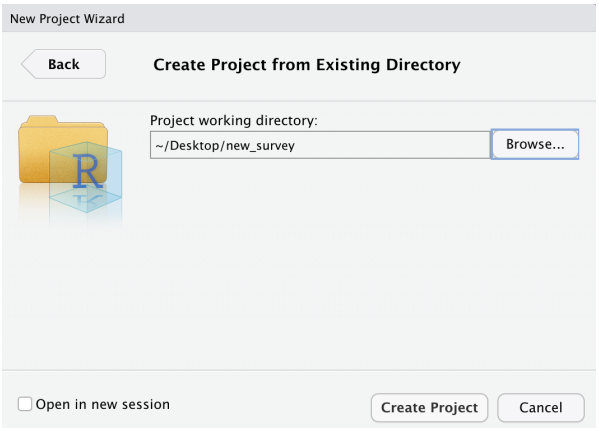
Once you have the survey codebook above, you need to convert it to an Advanced Text Format .txt file. We can do so by following the steps below. The instructions differ slightly for RStudio Cloud and RStudio - both are presented.

2.4.2.1.1 RStudio on your computer

1. Create a folder on your computer called "new_survey" **with the survey_codebook.csv file in it.**
2. Open RStudio. Then go to File > New Project... and select Existing Directory.



2. Then find and select the "new_survey" folder on your hard drive and click the Create Project button.



3. You should previously have installed the *remotes* and *tidyverse* packages from the CRAN and the qualtricsMaker package from GitHub. If you haven't done this already type the commands below into the **Console**.

Installing Packages from the CRAN

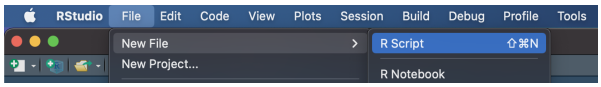
```
install.packages("tidyverse", dep = TRUE)
install.packages("remotes", dep = TRUE)
```

Installing Package from GitHub

Then install the qualtricsMaker package from GitHub:

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

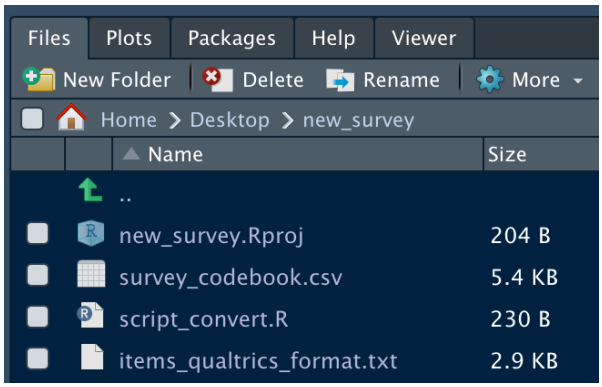
3. Create a new script by going to File > New File > R Script. A new script will appear in RStudio. Immediately press Control-S to save the script. Call it "script_convert.R".



4. Place the code below into the script and press Control-S to save your work.

```
library(tidyverse)
library(qualtricsMaker)

survey_codebook <- read_csv("survey_codebook.csv",
                            show_col_types = FALSE)

make_survey(survey_codebook,
            filename = "items_qualtrics_format.txt")
```

5. Run the script by pressing the Source button in the top right of the the Script window. Doing so will create the items_qualtrics_format.txt file that you will import into Qualtrics. You can see the file after it has been created in the Files tab below.



Congratulations - your items are ready to be imported into Qualtrics. Move on to the "Import the items" section.

2.4.2.1.2 RStudio Cloud

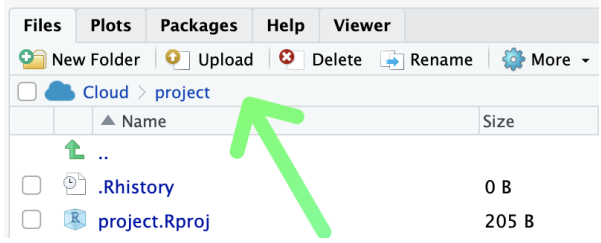
1. Open a new project in RStudio Cloud
2. You should previously have installed the remote and tidyverse package from the CRAN and the qualtricsMaker package from GitHub. If you haven't done this already type the commands below into the **Console**.However, if you are working from a class project I created **these packages will have been installed already for you.**

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

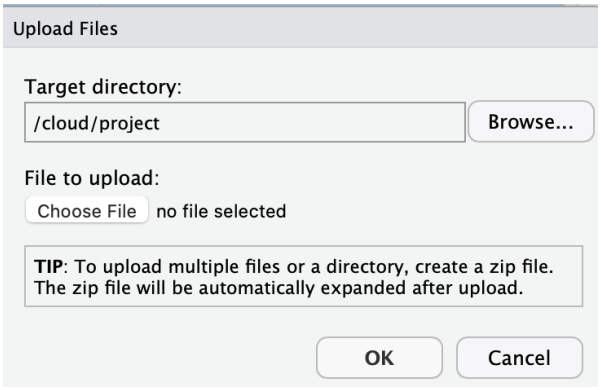
Then install the qualtricsMaker package from GitHub:

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

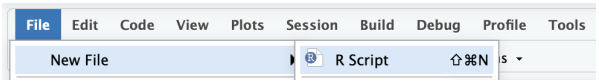
4. Import the survey_codebook.csv file by using the Upload button in the Files tab in the lower right panel.



5. Click the Choose File button and select the survey_codebook.csv file on your hard drive. Then click OK. The survey_codebook.csv file will then appear on the Files tab within RStudio Cloud.



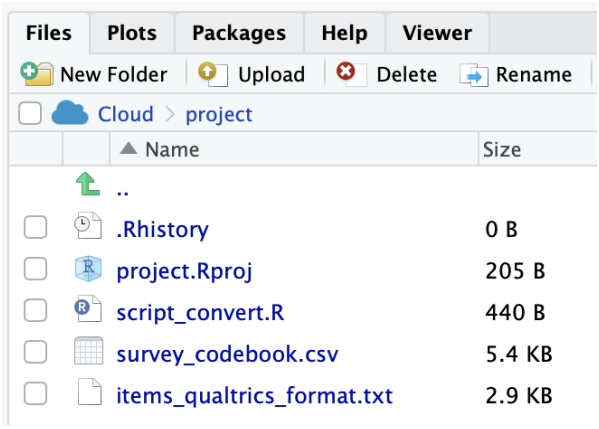
6. Create a new script by going to File > New File > R Script. A new script will appear in RStudio. Immediately press Control-S to save the script. Call it "script_convert.R".



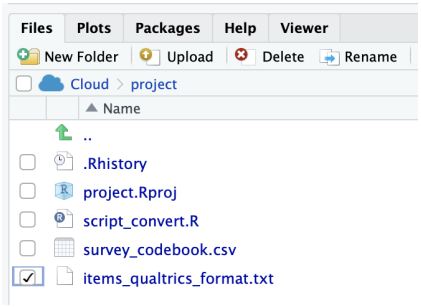
7. Place the code below into the script and press Control-S to save your work.

```
library(tidyverse)
library(qualtricsMaker)

survey_codebook <- read_csv("survey_codebook.csv",
                            show_col_types = FALSE)

make_survey(survey_codebook,
            filename = "items_qualtrics_format.txt")
```

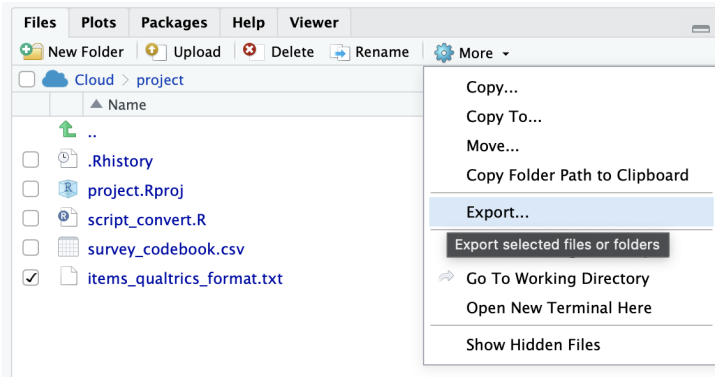
8. Run the script by pressing the Source button in the top right of the the Script window. Doing so will create the items_qualtrics_format.txt file that you will import into Qualtrics. You can see the file after it has been created in the Files tab below.



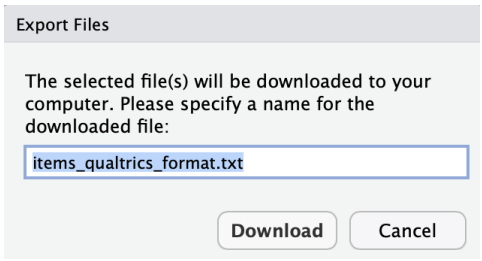
9. Select the checkbox beside the file you just created: items_qualtrics_format.txt



10. Select Export from the More menu.



11. Leave the filename as it appears and click the Download button. The file will appear in the Downloads directory on your computer.



12. Move the file (items_qualtrics_format.txt) from the Download folder to an easy to access place (e.g. the Desktop).

Congratulation - your items are ready to be imported into Qualtrics. Move on to the "Import the items" section.

2.4.3 Import the items

In the previous step you created items_qualtrics_format.txt. This file is illustrated below - well the middle of the file is illustrated below. You can think of the file created by the survey_maker command above as a starting point. You could open up this file in a text editor and make further modifications or additions based on learning the Advanced Text Format. For example, you might not want the matrix items to be in their own block. If so, you would simply remove the block text [[Block:Commitment]] from the text file prior to following the directions below. If you're a Mac user you might find the text editor BBEdit helpful for this purpose – the free version is sufficient. After you download BBEdit you have access to all of its features. Once the free trial ends - the core features are still free. You only need to purchase it if you want the advanced features (and you won't need the advanced features).

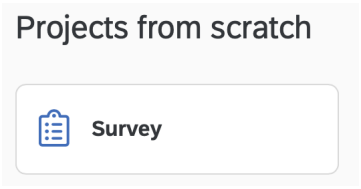
```
[[Block:Commitment]]

[[Question:Matrix]]
Indicate the extent to which you agree with each of the statements below.
[[Choices]]
I would be very happy to spend the rest of my career with this organization.
I really feel as if this organization's problems are my own.
I do not feel a strong sense of "belonging" to my organization.
I do not feel "emotionally attached" to this organization.
I do not feel like "part of the family" at my organization.
This organization has a great deal of personal meaning for me.
Right now, staying with my organization is a  matter of necessity as much as desire.
It would be very hard for me to leave my organization right now, even if I wanted to.
Too much of my life would be disrupted if I decided I wanted to leave my organization now.
I feel that I have too few options to consider leaving this organization.
If I had not already put so much of myself into this organization, I might consider working elsewhere.
One of the few negative consequences of leaving this organization would be the scarcity of available alternatives.
I do not feel any obligation to remain with my current employer.
Even if it were to my advantage, I do not feel it would be right to leave my organization now.
I would feel guilty if I left my organization now.
This organization deserves my loyalty.
I would not leave my organization right now because I have a sense of obligation to the people in it.
I owe a great deal to my organization.
[[AdvancedAnswers]]
[[Answer]]
Strongly Disagree
[[Answer]]
Moderately Disagree
[[Answer]]
Slightly Disagree
[[Answer]]
Neither Agree nor Disagree
[[Answer]]
Slightly Agree
[[Answer]]
Moderately Agree
[[Answer]]
Strongly Agree


[[Block:Feelings About Your Job]]

[[Question:MC:SingleAnswer:Vertical]]
[[ID:job_aff1_likert5]]
I find real enjoyment in my jobI find real enjoyment in my job
[[Choices]]
Strongly Disagree
 Disagree
 Neutral
 Agree
 Strongly Agree


[[Question:MC:SingleAnswer:Vertical]]
[[ID:job_aff2_likert5]]
I like my job better than the average person
[[Choices]]
Strongly Disagree
 Disagree
 Neutral
 Agree
 Strongly Agree
```

2.4.3.0.1 Begin the import by logging into Qualtrics

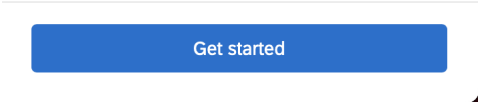
Begin by logging into Qualtrics via the University of Guelph site for Data Collection & Surveys. After you log in, continue below.

2.4.3.0.2 Click the Create New Project button in the top right of the page.

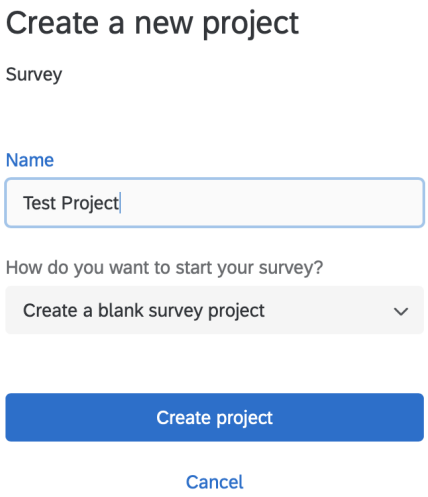
2.4.3.1 Under Projects from Scratch select "Survey" by clicking on it.



2.4.3.2 Click the Get Started button in the lower right of the page.

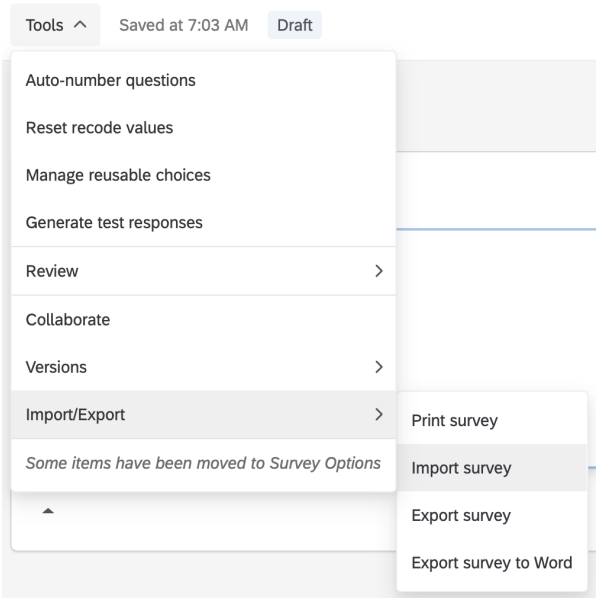


2.4.3.3 Enter project name, click Create Project



2.4.3.4 Use the Tools menu to import the items

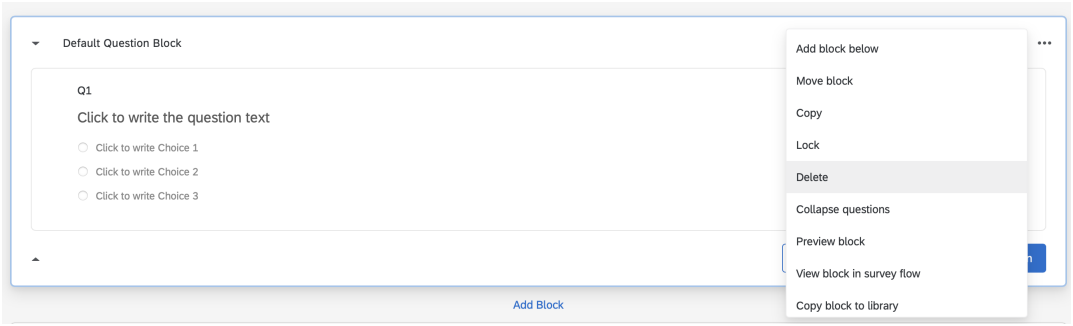
Select the Import option and then specify the filename (items_qualtrics_format.txt) for the Advanced Text import file with the items in it.



Congratulation - your items have been imported into Qualtrics.

2.4.3.5 Remove the default Empty Question block at the top of the survey

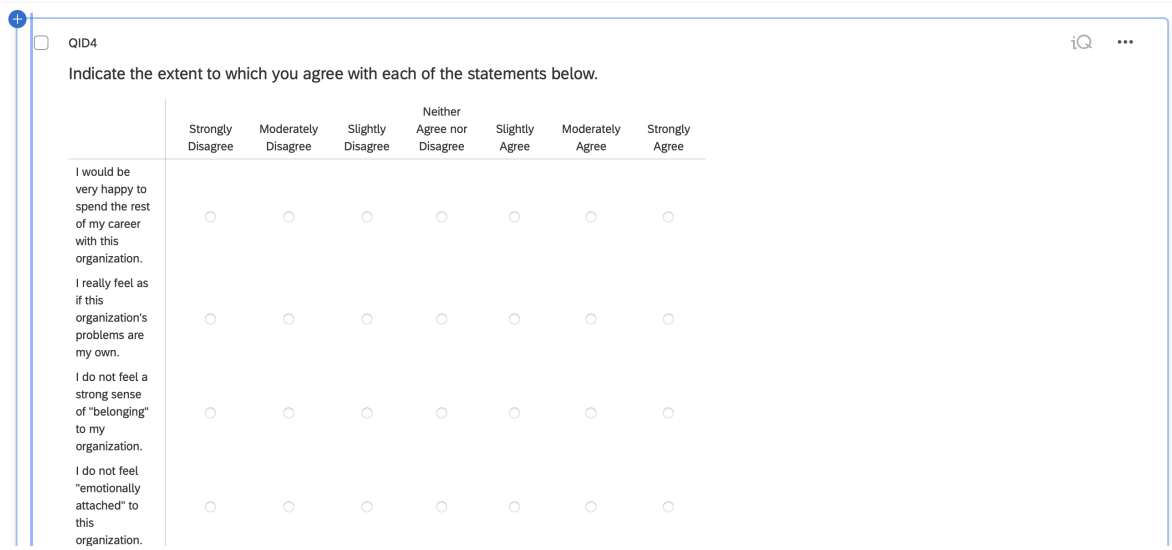
Scroll to the top of the survey and you will see a Default Question Block above your first block. Click on the three dots "..." in the top right of this block and select Delete to remove it.



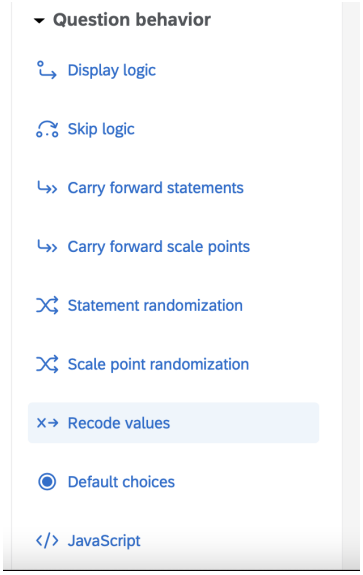
2.4.4 Tweak survey in Qualtrics

Now you need to tweak the survey using the Qualtrics interface. Recall that when you import items into Qualtrics, matrix-style questions will loose their item names. **You need to manually restore the item names for matrix questions now.**

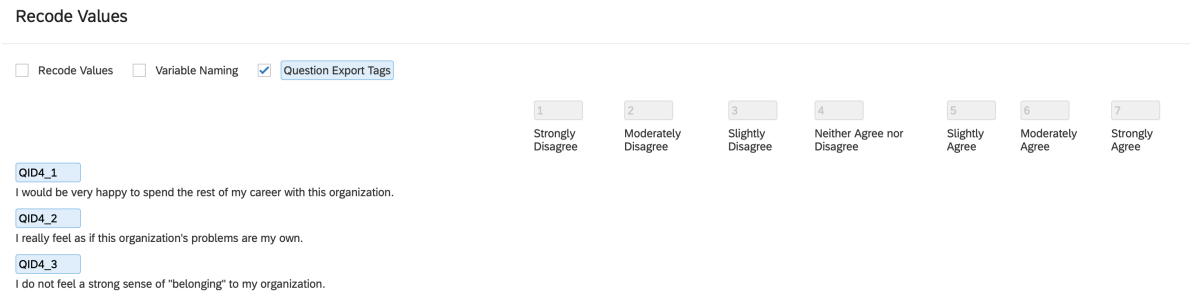
1. Begin by scrolling down to a block of matrix-style questions. Select the block so it gets a blue rectangle around it.



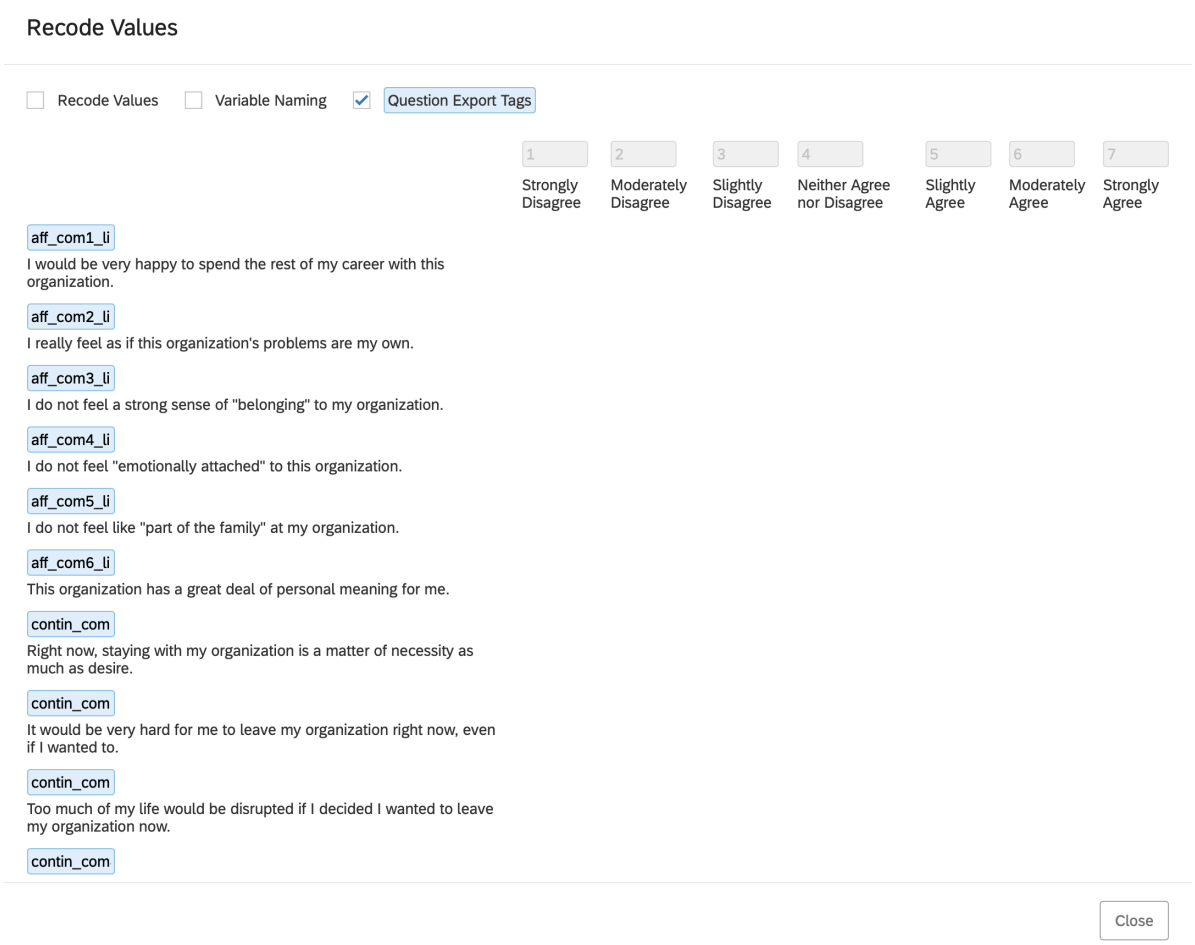
2. On the left side menu go the "Question Behaviour" menu and select "Recode values"



3. Click the Export Question Tags check box. You will see the default names for the items (instead of the ones you indicated in the codebook). Click the Close button in the lower right when you're done.



4. Manually restore the item names from the survey codebook by clicking the blue name for each item and fixing it. Don't forget to use proper item naming formats and include the reverse-key status of each item.



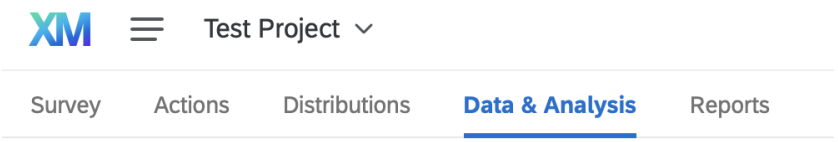
5. Collect your data!

2.5 Qualtrics data to R

Following data collection, you can obtain your data from Qualtrics. As of 2021, Tri-Agency (NSERC, SSHRC, CIHR) policy is that data collected with Tri-Agency funded research must be available for reuse by others. The data should follow the FAIR (Findable, Accessible, Interoperable, and Reusable) principle. Consequently, when you export the data from Qualtrics (and eventually post it) we want to ensure it has as much information in it as possible. This principle guides the options we select below.

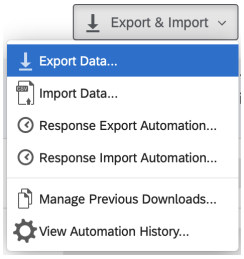
2.5.1 Export from Qualtrics

2.5.1.1 Click the Data & Analysis tab



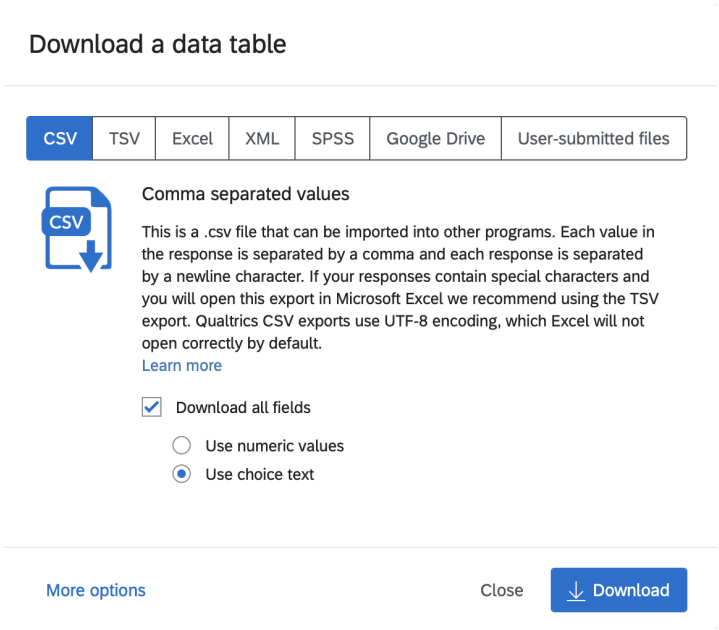
2.5.1.2 Click the Export & Import button

Then click the **Export Data...** option.



2.5.1.3 Click Use Choice Text

To ensure the data can be used by others, as per the FAIR policy described above, we SELECT the "Use Choice Text" option when exporting data. If you were to (non-optimally) select "Use numeric value" as an export option then the resulting data file would be missing information that make it difficult for others to use. Then click the Download button.



The file will appear in your downloads directory.

- It will likely be a zipped filed. Unzip the file before proceeding. On a Mac, you can just double click the file to do so.
- Rename the file to data_qualtrics.csv

2.5.2 Load data in R

To load the data in R follow the steps below

1. Create a new folder called "survey_data"
2. Drag the data_qualtrics.csv file to the survey_data folder.
3. Use the steps described previously on this page, use RStudio to create project for this data.
4. You may be tempted to load the data using the usual read_csv() process. This will be problematic due to the way the data is formatted in the .csv file. (see image below.)

5. You can see by inspecting the data screenshot above that a variety of columns have been included before our first data column (year_of_birth) which appears on the far right of screenshot. Also note that there are three header rows at the top of the file before the data which starts on row 4. So there are few problems that need to be addressed when we load data from Qualtrics.

- Obtain the names from the first header row but read the data from row 4 on.
- Remove some, or all, of the additional columns on the left side of data set added by Qualtrics
- Ensure our variable names follow the tidyverse style guide
- Ensure we DO NOT modify the data_qualtrics.csv file in Excel or other program. That would destroy the reproducibility of our work in the very first step.

2.5.3 Convert to analytic data

The issues described above are all solved with the script below.

Additionally, in the script we also:

- Convert categorical variable to factors
- Screen both numeric variables and factors for impossible values
- Convert Likert items from labels (e.g., "Strongly Disagree") to numbers (e.g., 1)
- Flip reverse-key (i.e., reverse worded items) so they are scored in the right direction
- Combine items (e.g., aff_com1_likert7) for the same measure into a single scale score (e.g., affective_commitment)
- Remove item columns (e.g., aff_com1_likert7) after that have been combined into the scale score while retaining the scale score column column (e.g., affective_commitment)

2.5.3.1 Load Data

```
# Date: YYYY-MM-DD
# Name: your name here
# Example: Single occasion survey

# Load data
library(tidyverse)
library(janitor)
```

```
library(skimr)

# read the file in the normal way and get just the names of the columns.
# put the names into the col_names
survey_file <- "data_qualtrics.csv"
col_names <- names(read_csv(survey_file,
                            n_max = 0,
                            show_col_types = FALSE))

# skip the first 3 rows than read in the data and use names from above
raw_data <- read_csv(survey_file,
                     col_names = col_names,
                     skip = 3,
                     show_col_types = FALSE)
```

Remove the Qualtrics columns you probably don't want:

```
# Qualtrics columns to remove (any_of() ignores names not present)
cols_to_remove <- c("StartDate",
                    "EndDate",
                    "Status",
                    "IPAddress",
                    "Progress",
                    "Finished",
                    "RecordedDate",
                    "ResponseId",
                    "RecipientLastName",
                    "RecipientFirstName",
                    "RecipientEmail",
                    "ExternalReference",
                    "LocationLatitude",
                    "LocationLongitude",
                    "DistributionChannel",
                    "UserLanguage")

# remove the unwanted Qualtrics columns
raw_data <- raw_data |>
  select(!any_of(cols_to_remove))
```

We make a copy of the data called analytic_data_survey since the goals of the next several steps are to prepare the data for analysis.

```
analytic_data_survey <- raw_data
```

In the command below, we ensure our column names follow the tidyverse style guide by using the clean_names() command from the janitor package. Additionally, we remove empty row and columns from the data using the remove_empty_cols() and remove_empty_rows(), respectively.

```
# Initial cleaning
## Convert column names to tidyverse style guide
## Remove empty rows and columns
analytic_data_survey <- analytic_data_survey %>%
  remove_empty("rows") %>%
  remove_empty("cols") %>%
  clean_names()
```

You can confirm the column names following our naming convention with the glimpse command - and see the data type for each column.

```
glimpse(analytic_data_survey)
```

```
Rows: 150
Columns: 25
$ duration_in_seconds  <dbl> 244, 134, 237, 110, 151, 92, 393, 165, 315, 262, ~
$ year_of_birth        <dbl> 1961, 2006, 2008, 1985, 1953, 1957, 1998, 1970, 1~
$ sex                  <chr> "female", "female", "female", "male", "intersex",~
$ aff_com1_likert7     <chr> "Slightly Agree", "Neither Agree nor Disagree", "~
$ aff_com2_likert7     <chr> "Slightly Agree", "Slightly Disagree", "Neither A~
$ aff_com3_likert7rev  <chr> "Strongly Disagree", "Neither Agree nor Disagree"~
$ aff_com4_likert7rev  <chr> "Moderately Disagree", "Strongly Agree", "Moderat~
$ aff_com5_likert7rev  <chr> "Strongly Disagree", "Neither Agree nor Disagree"~
$ aff_com6_likert7     <chr> "Moderately Agree", "Strongly Agree", "Strongly D~
$ contin_com1_likert7  <chr> "Strongly Agree", "Neither Agree nor Disagree", "~
$ contin_com2_likert7  <chr> "Slightly Agree", "Slightly Agree", "Slightly Dis~
$ contin_com3_likert7  <chr> "Strongly Disagree", "Slightly Disagree", "Strong~
$ contin_com4_likert7  <chr> "Moderately Agree", "Neither Agree nor Disagree",~
$ contin_com5_likert7  <chr> "Slightly Agree", "Slightly Agree", "Strongly Dis~
$ contin_com6_likert7  <chr> "Moderately Disagree", "Moderately Disagree", "St~
$ norm_com1_likert7rev <chr> "Neither Agree nor Disagree", "Slightly Disagree"~
$ norm_com2_likert7    <chr> "Moderately Disagree", "Moderately Agree", "Stron~
$ norm_com3_likert7    <chr> "Neither Agree nor Disagree", "Neither Agree nor ~
$ norm_com4_likert7    <chr> "Moderately Disagree", "Moderately Disagree", "St~
```

```
$ norm_com5_likert7    <chr> "Strongly Disagree", "Strongly Disagree", "Strong~
$ norm_com6_likert7    <chr> "Strongly Agree", "Strongly Disagree", "Moderatel~
$ job_aff1_likert5     <chr> "Strongly Agree", "Strongly Agree", "Neutral", "S~
$ job_aff2_likert5     <chr> "Strongly Disagree", "Strongly Agree", "Strongly ~
$ job_aff3_likert5     <chr> "Neutral", "Strongly Disagree", "Disagree", "Stro~
$ job_aff4_likert5     <chr> "Disagree", "Strongly Agree", "Agree", "Strongly ~
```

2.5.3.2 Creating factors

Following initial cleaning, we identify categorical variables as factors.

```
glimpse(analytic_data_survey)
```

```
Rows: 150
Columns: 25
$ duration_in_seconds  <dbl> 244, 134, 237, 110, 151, 92, 393, 165, 315, 262, ~
$ year_of_birth        <dbl> 1961, 2006, 2008, 1985, 1953, 1957, 1998, 1970, 1~
$ sex                  <chr> "female", "female", "female", "male", "intersex",~
$ aff_com1_likert7     <chr> "Slightly Agree", "Neither Agree nor Disagree", "~
$ aff_com2_likert7     <chr> "Slightly Agree", "Slightly Disagree", "Neither A~
$ aff_com3_likert7rev  <chr> "Strongly Disagree", "Neither Agree nor Disagree"~
$ aff_com4_likert7rev  <chr> "Moderately Disagree", "Strongly Agree", "Moderat~
$ aff_com5_likert7rev  <chr> "Strongly Disagree", "Neither Agree nor Disagree"~
$ aff_com6_likert7     <chr> "Moderately Agree", "Strongly Agree", "Strongly D~
$ contin_com1_likert7  <chr> "Strongly Agree", "Neither Agree nor Disagree", "~
$ contin_com2_likert7  <chr> "Slightly Agree", "Slightly Agree", "Slightly Dis~
$ contin_com3_likert7  <chr> "Strongly Disagree", "Slightly Disagree", "Strong~
$ contin_com4_likert7  <chr> "Moderately Agree", "Neither Agree nor Disagree",~
$ contin_com5_likert7  <chr> "Slightly Agree", "Slightly Agree", "Strongly Dis~
$ contin_com6_likert7  <chr> "Moderately Disagree", "Moderately Disagree", "St~
$ norm_com1_likert7rev <chr> "Neither Agree nor Disagree", "Slightly Disagree"~
$ norm_com2_likert7    <chr> "Moderately Disagree", "Moderately Agree", "Stron~
$ norm_com3_likert7    <chr> "Neither Agree nor Disagree", "Neither Agree nor ~
$ norm_com4_likert7    <chr> "Moderately Disagree", "Moderately Disagree", "St~
$ norm_com5_likert7    <chr> "Strongly Disagree", "Strongly Disagree", "Strong~
$ norm_com6_likert7    <chr> "Strongly Agree", "Strongly Disagree", "Moderatel~
$ job_aff1_likert5     <chr> "Strongly Agree", "Strongly Agree", "Neutral", "S~
$ job_aff2_likert5     <chr> "Strongly Disagree", "Strongly Agree", "Strongly ~
$ job_aff3_likert5     <chr> "Neutral", "Strongly Disagree", "Disagree", "Stro~
$ job_aff4_likert5     <chr> "Disagree", "Strongly Agree", "Agree", "Strongly ~
```

2.5.3.2.1 Sex

There is only one variable that we will convert to a factor and that's the sex variable. We convert it to a factor with the command below.

```
# Convert variables to factors as needed
## Convert sex to factor
analytic_data_survey <- analytic_data_survey %>%
  mutate(sex = as_factor(sex))
```

2.5.3.2.2 Participant identification

In many analyses we will need a column with a unique identification number that is a factor. We don't see one in this data set so we create it with the name participant_id.

```
## Participant identification to factor
## Participant identification column must be created first
## get the number of rows in the data set
N <- dim(analytic_data_survey)[1]

## create set of factor levels
participant_id_levels <- factor(seq(1, N))

## put the factor levels into a column called participant_id
analytic_data_survey <- analytic_data_survey %>%
  mutate(participant_id = participant_id_levels)
```

You can ensure all of these columns are now factors using the glimpse() command.

```
analytic_data_survey %>%
  select(sex, participant_id) %>%
  glimpse()
```

```
Rows: 150
Columns: 2
$ sex            <fct> female, female, female, male, intersex, intersex, inter~
$ participant_id <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, ~
```

2.5.3.3 Factor screening

Inspect the levels of each factor carefully. Make sure the factor levels of each variable are correct. Examine spelling and look for additional unwanted levels. For example, you wouldn't want to have the following levels for sex: male, mmale, female. Obviously, mmale is an incorrectly typed version of male. Scan all the factors in your data for erroneous factor levels. The code below displays the factor levels for sex:

```
# Screen factors
## screen
analytic_data_survey %>%
  select(sex) %>%
  summary()
```

```
       sex
 female  :43
 male    :49
 intersex:58
```

You can see in the above output there are three levels for sex in this data set: female, male and intersex. You can also see beside each of these labels the number of participants for each category.

Notice the order of the factor levels: female, male, and intersex. This will be the order they appear in graphs and tables. We might want another order. If we did want another order, we could use the code below to establish the custom order for graphs and tables: female, intersex, male. Of course, you can use any order you want that makes sense for the context and your research question.

```
## change to desired order
analytic_data_survey <- analytic_data_survey %>%
  mutate(sex = fct_relevel(sex,
                           "female",
                           "intersex",
                           "male"))
```

2.5.3.4 Numeric screening

For numeric variables, it's important to find and remove impossible values. We only have year_of_birth as a numeric column - so we can easily check for nonsense values with the code below:

```
# Screen numeric variables
analytic_data_survey %>%
  select(year_of_birth) %>%
  skim()
```

```
  skim_variable n_missing    mean    sd   p0  p50 p100
1 year_of_birth         0 1973.88 20.06 1940 1970 2010
```

2.5.3.5 Item values from labels

Recall that when we obtained our data from Qualtrics we selected "Use Choice Text". That means that, as desired, we didn't get values for our Likert items - we obtained labels. Now we need to switch the Likert responses to numeric values. The advantage of handling our data in this way is that anyone examining our data and code in an open access repository can see exactly what options survey participants were provided with for each question.

But prior to beginning check out the type for each Likert column below using the glimpse() command notice that are all of type "chr".

```
analytic_data_survey %>% glimpse()
```

```
Rows: 150
Columns: 26
$ duration_in_seconds  <dbl> 244, 134, 237, 110, 151, 92, 393, 165, 315, 262, ~
$ year_of_birth        <dbl> 1961, 2006, 2008, 1985, 1953, 1957, 1998, 1970, 1~
$ sex                  <fct> female, female, female, male, intersex, intersex,~
$ aff_com1_likert7     <chr> "Slightly Agree", "Neither Agree nor Disagree", "~
$ aff_com2_likert7     <chr> "Slightly Agree", "Slightly Disagree", "Neither A~
$ aff_com3_likert7rev  <chr> "Strongly Disagree", "Neither Agree nor Disagree"~
$ aff_com4_likert7rev  <chr> "Moderately Disagree", "Strongly Agree", "Moderat~
$ aff_com5_likert7rev  <chr> "Strongly Disagree", "Neither Agree nor Disagree"~
$ aff_com6_likert7     <chr> "Moderately Agree", "Strongly Agree", "Strongly D~
$ contin_com1_likert7  <chr> "Strongly Agree", "Neither Agree nor Disagree", "~
$ contin_com2_likert7  <chr> "Slightly Agree", "Slightly Agree", "Slightly Dis~
$ contin_com3_likert7  <chr> "Strongly Disagree", "Slightly Disagree", "Strong~
$ contin_com4_likert7  <chr> "Moderately Agree", "Neither Agree nor Disagree",~
$ contin_com5_likert7  <chr> "Slightly Agree", "Slightly Agree", "Strongly Dis~
$ contin_com6_likert7  <chr> "Moderately Disagree", "Moderately Disagree", "St~
$ norm_com1_likert7rev <chr> "Neither Agree nor Disagree", "Slightly Disagree"~
$ norm_com2_likert7    <chr> "Moderately Disagree", "Moderately Agree", "Stron~
$ norm_com3_likert7    <chr> "Neither Agree nor Disagree", "Neither Agree nor ~
```

```
$ norm_com4_likert7    <chr> "Moderately Disagree", "Moderately Disagree", "St~
$ norm_com5_likert7    <chr> "Strongly Disagree", "Strongly Disagree", "Strong~
$ norm_com6_likert7    <chr> "Strongly Agree", "Strongly Disagree", "Moderatel~
$ job_aff1_likert5     <chr> "Strongly Agree", "Strongly Agree", "Neutral", "S~
$ job_aff2_likert5     <chr> "Strongly Disagree", "Strongly Agree", "Strongly ~
$ job_aff3_likert5     <chr> "Neutral", "Strongly Disagree", "Disagree", "Stro~
$ job_aff4_likert5     <chr> "Disagree", "Strongly Agree", "Agree", "Strongly ~
$ participant_id       <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15~
```

When you have type chr for a column the computer sees the content of each column as text that is different for each participant. It doesn't even realize that two people who both responded "Slightly Agree" responded the same way. It sees every response in the column as unique. Check that out for the aff_com1_likert7 item.

```
analytic_data_survey %>%
    select(aff_com1_likert7) %>%
    summary()
```

```
 aff_com1_likert7
 Length:150
 Class :character
 Mode  :character
```

2.5.3.5.1 Commitment Items

To convert a column from a character column to factor column we need know the exact labels used in that column. We can obtain those with the code below. In this code, we use just one item aff_com1_likert7. We determine the response options for that one item and will apply them to all the commitment items. We use the pull() command to obtain all the participant responses from the aff_com1_likert7. column. Then we use the unique() command to give use just one instance of each response.

```
# Convert Commitment items to numeric  values
## Check levels for a likert7 item
analytic_data_survey %>%
  pull(aff_com1_likert7) %>%
  unique()
```

```
[1] "Slightly Agree"             "Neither Agree nor Disagree"
[3] "Moderately Disagree"        "Strongly Agree"
[5] "Strongly Disagree"          "Moderately Agree"
[7] "Slightly Disagree"
```

Now that we know the labels used in that column (and all the commitment columns that end in likert7) we create an ordered version of those labels with the code below. I copied and pasted the text (e.g., "Strongly Disagree") from the above output to avoid typos.

```
## write code to create ordered factor labels/levels
likert7_factor <- c("Strongly Disagree",
                    "Moderately Disagree",
                    "Slightly Disagree",
                    "Neither Agree nor Disagree",
                    "Slightly Agree",
                    "Moderately Agree",
                    "Strongly Agree")
```

Now we turn each column of text (i.e., type chr) into a factor column using the factor levels/labels we created:

```
## assign factor levels
analytic_data_survey <- analytic_data_survey %>%
  mutate(across(.cols = contains("likert7"),
                .fns = ~ factor(.x, levels = likert7_factor)))
```

In this code we use the mutate() command to modify columns. We use the across() command to have the single mutate command apply to several columns. Within the across() command we use contains("likert7") to apply the mutate() command only to columns with likert7 in the name. **You can see why column naming conventions are so important!** Next, within the across() command, we specify .fns. This indicate the function/command that will apply to the columns selected. The ~ before "factor" just warns the computer that what comes next is a function/command name. The factor() command is what will be applied to the columns. The .x within the factor command is a placeholder for each column name that will be obtained by the previous contains() command.

After this mutate() command is used you can see the columns with likert7 in the name are now factors.

```
glimpse(analytic_data_survey)
```

```
Rows: 150
Columns: 26
$ duration_in_seconds  <dbl> 244, 134, 237, 110, 151, 92, 393, 165, 315, 262, ~
$ year_of_birth        <dbl> 1961, 2006, 2008, 1985, 1953, 1957, 1998, 1970, 1~
$ sex                  <fct> female, female, female, male, intersex, intersex,~
$ aff_com1_likert7     <fct> Slightly Agree, Neither Agree nor Disagree, Moder~
```

```
$ aff_com2_likert7     <fct> Slightly Agree, Slightly Disagree, Neither Agree ~
$ aff_com3_likert7rev  <fct> Strongly Disagree, Neither Agree nor Disagree, Mo~
$ aff_com4_likert7rev  <fct> Moderately Disagree, Strongly Agree, Moderately A~
$ aff_com5_likert7rev  <fct> Strongly Disagree, Neither Agree nor Disagree, Mo~
$ aff_com6_likert7     <fct> Moderately Agree, Strongly Agree, Strongly Disagr~
$ contin_com1_likert7  <fct> Strongly Agree, Neither Agree nor Disagree, Stron~
$ contin_com2_likert7  <fct> Slightly Agree, Slightly Agree, Slightly Disagree~
$ contin_com3_likert7  <fct> Strongly Disagree, Slightly Disagree, Strongly Ag~
$ contin_com4_likert7  <fct> Moderately Agree, Neither Agree nor Disagree, Sli~
$ contin_com5_likert7  <fct> Slightly Agree, Slightly Agree, Strongly Disagree~
$ contin_com6_likert7  <fct> Moderately Disagree, Moderately Disagree, Strongl~
$ norm_com1_likert7rev <fct> Neither Agree nor Disagree, Slightly Disagree, Mo~
$ norm_com2_likert7    <fct> Moderately Disagree, Moderately Agree, Strongly D~
$ norm_com3_likert7    <fct> Neither Agree nor Disagree, Neither Agree nor Dis~
$ norm_com4_likert7    <fct> Moderately Disagree, Moderately Disagree, Strongl~
$ norm_com5_likert7    <fct> Strongly Disagree, Strongly Disagree, Strongly Di~
$ norm_com6_likert7    <fct> Strongly Agree, Strongly Disagree, Moderately Dis~
$ job_aff1_likert5     <chr> "Strongly Agree", "Strongly Agree", "Neutral", "S~
$ job_aff2_likert5     <chr> "Strongly Disagree", "Strongly Agree", "Strongly ~
$ job_aff3_likert5     <chr> "Neutral", "Strongly Disagree", "Disagree", "Stro~
$ job_aff4_likert5     <chr> "Disagree", "Strongly Agree", "Agree", "Strongly ~
$ participant_id       <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15~
```

Now examine aff_com1_likert7 again - you can see it has levels:

```
analytic_data_survey %>%
    select(aff_com1_likert7) %>%
    summary()
```

```
                  aff_com1_likert7
 Strongly Disagree         :21
 Moderately Disagree       :21
 Slightly Disagree         :21
 Neither Agree nor Disagree:18
 Slightly Agree            :28
 Moderately Agree          :21
 Strongly Agree            :20
```

The second step in our conversion process is converting each column from a factor to the numeric value associated with the factor level (i.e. Strongly Disagree becomes 1). We do so with the code below:

```r
## covert factor levels to numeric values
analytic_data_survey <- analytic_data_survey %>%
  mutate(across(.cols = contains("likert7"),
                .fns = as.numeric))
```

You can see there are numeric values with the code below. Notice the column types for likert7 items is now dbl (short for double - a type of high precision number.)

```r
glimpse(analytic_data_survey)
```

```
Rows: 150
Columns: 26
$ duration_in_seconds  <dbl> 244, 134, 237, 110, 151, 92, 393, 165, 315, 262, ~
$ year_of_birth        <dbl> 1961, 2006, 2008, 1985, 1953, 1957, 1998, 1970, 1~
$ sex                  <fct> female, female, female, male, intersex, intersex,~
$ aff_com1_likert7     <dbl> 5, 4, 2, 7, 1, 6, 6, 2, 4, 2, 5, 4, 7, 3, 1, 7, 3~
$ aff_com2_likert7     <dbl> 5, 3, 4, 2, 6, 7, 2, 3, 2, 7, 3, 4, 4, 1, 6, 2, 5~
$ aff_com3_likert7rev  <dbl> 1, 4, 2, 5, 7, 2, 6, 3, 5, 4, 3, 3, 2, 2, 6, 2, 7~
$ aff_com4_likert7rev  <dbl> 2, 7, 6, 6, 3, 5, 7, 6, 7, 2, 7, 2, 5, 6, 1, 4, 4~
$ aff_com5_likert7rev  <dbl> 1, 4, 6, 5, 3, 7, 5, 7, 5, 5, 4, 5, 3, 1, 3, 2, 4~
$ aff_com6_likert7     <dbl> 6, 7, 1, 7, 5, 6, 5, 7, 4, 2, 7, 7, 5, 3, 2, 4, 4~
$ contin_com1_likert7  <dbl> 7, 4, 1, 3, 2, 1, 1, 6, 7, 3, 6, 2, 2, 5, 5, 3, 5~
$ contin_com2_likert7  <dbl> 5, 5, 3, 2, 7, 6, 1, 3, 5, 2, 4, 5, 1, 6, 6, 6, 2~
$ contin_com3_likert7  <dbl> 1, 3, 7, 7, 5, 2, 7, 2, 7, 1, 4, 2, 4, 5, 6, 1, 6~
$ contin_com4_likert7  <dbl> 6, 4, 3, 7, 7, 5, 4, 5, 3, 2, 2, 4, 2, 7, 3, 7, 5~
$ contin_com5_likert7  <dbl> 5, 5, 1, 4, 4, 6, 5, 5, 5, 2, 5, 6, 3, 5, 2, 7, 6~
$ contin_com6_likert7  <dbl> 2, 2, 1, 4, 7, 7, 6, 6, 4, 5, 2, 5, 5, 5, 5, 5, 3~
$ norm_com1_likert7rev <dbl> 4, 3, 6, 3, 3, 6, 5, 7, 3, 7, 5, 1, 5, 3, 5, 4, 2~
$ norm_com2_likert7    <dbl> 2, 6, 1, 5, 3, 2, 3, 1, 7, 1, 1, 6, 6, 6, 7, 6, 4~
$ norm_com3_likert7    <dbl> 4, 4, 7, 3, 1, 1, 4, 3, 3, 5, 3, 5, 2, 3, 2, 3, 1~
$ norm_com4_likert7    <dbl> 2, 2, 1, 6, 4, 6, 1, 1, 5, 5, 5, 3, 2, 3, 3, 5, 5~
$ norm_com5_likert7    <dbl> 1, 1, 1, 5, 7, 4, 3, 6, 4, 4, 1, 3, 3, 1, 6, 1, 7~
$ norm_com6_likert7    <dbl> 7, 1, 2, 4, 6, 6, 1, 4, 6, 2, 1, 4, 1, 5, 6, 3, 4~
$ job_aff1_likert5     <chr> "Strongly Agree", "Strongly Agree", "Neutral", "S~
$ job_aff2_likert5     <chr> "Strongly Disagree", "Strongly Agree", "Strongly ~
$ job_aff3_likert5     <chr> "Neutral", "Strongly Disagree", "Disagree", "Stro~
$ job_aff4_likert5     <chr> "Disagree", "Strongly Agree", "Agree", "Strongly ~
$ participant_id       <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15~
```

If you run the summary() command on the aff_com1_likert7 you see we have values now.

```
analytic_data_survey %>%
    select(aff_com1_likert7) %>%
    summary()
```

```
 aff_com1_likert7
 Min.   :1.000
 1st Qu.:2.000
 Median :4.000
 Mean   :4.027
 3rd Qu.:6.000
 Max.   :7.000
```

Side note: We did a lot of explaining as we went through the steps above. But it's not as long as it seems. Some of the steps can even be combined. See the concise version of the code we used repeated below:

```
# Convert Commitment items to numeric  values
## Check levels for a likert7 item
analytic_data_survey %>%
  pull(aff_com1_likert7) %>%
  unique()

## write code to create ordered factor labels/levels
likert7_factor <- c("Strongly Disagree",
                    "Moderately Disagree",
                    "Slightly Disagree",
                    "Neither Agree nor Disagree",
                    "Slightly Agree",
                    "Moderately Agree",
                    "Strongly Agree")


## assign factor levels and then covert to numeric
analytic_data_survey <- analytic_data_survey %>%
  mutate(across(.cols = contains("likert7"),
                .fns = ~ factor(.x, levels = likert7_factor))) %>%
  mutate(across(.cols = contains("likert7"),
                .fns = as.numeric))
```

2.5.3.5.2 Job Satisfaction Conversion

We'll skip all the explanation steps for Job Satisfaction and just show you the essentials. First, get the labels for the items:

```
# Convert Job Satisfaction items to numeric  values
## Check levels for a likert5 item
analytic_data_survey %>%
  pull(job_aff1_likert5) %>%
  unique()
```

```
[1] "Strongly Agree"    "Neutral"           "Strongly Disagree"
[4] "Agree"             "Disagree"
```

Second, create the ordered factor labels:

```
## write code to create ordered factor labels/levels
likert5_factor <- c("Strongly Disagree",
                    "Disagree",
                    "Neutral",
                    "Agree",
                    "Strongly Agree")
```

Third, convert the columns to a factor and then to numbers.

```
## assign factor levels and then covert to numeric
analytic_data_survey <- analytic_data_survey %>%
  mutate(across(.cols = contains("likert5"),
                .fns = ~ factor(.x, levels = likert5_factor))) %>%
  mutate(across(.cols = contains("likert5"),
                .fns = as.numeric))
```

2.5.3.6 Scale scores

For each person, scale scores involve averaging scores from several items to create an overall score. The first step in the creation of scales is correcting the values of any reverse-keyed items.

2.5.3.6.1 Reverse-key items

A complete discussion of the logic for flipping reverse-key (i.e., reverse worded) items is provide the previous chapter in the Scale Scores section. We use that logic in the script below.

```
# Reverse key items
## Reverse key likert7 items
analytic_data_survey <- analytic_data_survey %>%
  mutate(8 - across(.cols = ends_with("_likert7rev")) ) %>%
  rename_with(.fn = str_replace,
              .cols = ends_with("_likert7rev"),
              pattern = "_likert7rev",
              replacement = "_likert7")
```

You can see that are all keyed in the right direction now.

```
glimpse(analytic_data_survey)
```

```
Rows: 150
Columns: 26
$ duration_in_seconds <dbl> 244, 134, 237, 110, 151, 92, 393, 165, 315, 262, 3~
$ year_of_birth       <dbl> 1961, 2006, 2008, 1985, 1953, 1957, 1998, 1970, 19~
$ sex                 <fct> female, female, female, male, intersex, intersex, ~
$ aff_com1_likert7    <dbl> 5, 4, 2, 7, 1, 6, 6, 2, 4, 2, 5, 4, 7, 3, 1, 7, 3,~
$ aff_com2_likert7    <dbl> 5, 3, 4, 2, 6, 7, 2, 3, 2, 7, 3, 4, 4, 1, 6, 2, 5,~
$ aff_com3_likert7    <dbl> 7, 4, 6, 3, 1, 6, 2, 5, 3, 4, 5, 5, 6, 6, 2, 6, 1,~
$ aff_com4_likert7    <dbl> 6, 1, 2, 2, 5, 3, 1, 2, 1, 6, 1, 6, 3, 2, 7, 4, 4,~
$ aff_com5_likert7    <dbl> 7, 4, 2, 3, 5, 1, 3, 1, 3, 3, 4, 3, 5, 7, 5, 6, 4,~
$ aff_com6_likert7    <dbl> 6, 7, 1, 7, 5, 6, 5, 7, 4, 2, 7, 7, 5, 3, 2, 4, 4,~
$ contin_com1_likert7 <dbl> 7, 4, 1, 3, 2, 1, 1, 6, 7, 3, 6, 2, 2, 5, 5, 3, 5,~
$ contin_com2_likert7 <dbl> 5, 5, 3, 2, 7, 6, 1, 3, 5, 2, 4, 5, 1, 6, 6, 6, 2,~
$ contin_com3_likert7 <dbl> 1, 3, 7, 7, 5, 2, 7, 2, 7, 1, 4, 2, 4, 5, 6, 1, 6,~
$ contin_com4_likert7 <dbl> 6, 4, 3, 7, 7, 5, 4, 5, 3, 2, 2, 4, 2, 7, 3, 7, 5,~
$ contin_com5_likert7 <dbl> 5, 5, 1, 4, 4, 6, 5, 5, 5, 2, 5, 6, 3, 5, 2, 7, 6,~
$ contin_com6_likert7 <dbl> 2, 2, 1, 4, 7, 7, 6, 6, 4, 5, 2, 5, 5, 5, 5, 5, 3,~
$ norm_com1_likert7   <dbl> 4, 5, 2, 5, 5, 2, 3, 1, 5, 1, 3, 7, 3, 5, 3, 4, 6,~
$ norm_com2_likert7   <dbl> 2, 6, 1, 5, 3, 2, 3, 1, 7, 1, 1, 6, 6, 6, 7, 6, 4,~
$ norm_com3_likert7   <dbl> 4, 4, 7, 3, 1, 1, 4, 3, 3, 5, 3, 5, 2, 3, 2, 3, 1,~
$ norm_com4_likert7   <dbl> 2, 2, 1, 6, 4, 6, 1, 1, 5, 5, 5, 3, 2, 3, 3, 5, 5,~
$ norm_com5_likert7   <dbl> 1, 1, 1, 5, 7, 4, 3, 6, 4, 4, 1, 3, 3, 1, 6, 1, 7,~
$ norm_com6_likert7   <dbl> 7, 1, 2, 4, 6, 6, 1, 4, 6, 2, 1, 4, 1, 5, 6, 3, 4,~
$ job_aff1_likert5    <dbl> 5, 5, 3, 1, 1, 3, 4, 4, 1, 4, 3, 5, 5, 1, 2, 1, 3,~
```

```
$ job_aff2_likert5    <dbl> 1, 5, 5, 2, 3, 1, 3, 3, 5, 5, 4, 3, 1, 1, 5, 3, 5,~
$ job_aff3_likert5    <dbl> 3, 1, 2, 5, 5, 3, 3, 1, 5, 2, 3, 4, 5, 4, 2, 3, 3,~
$ job_aff4_likert5    <dbl> 2, 5, 4, 1, 5, 3, 1, 3, 3, 5, 2, 5, 2, 3, 1, 5, 5,~
$ participant_id      <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,~
```

None of the job satisfaction items were reverse-keyed.

2.5.3.6.2 Creating scores

Important: Take note of how we name the scale variables (e.g., affective_commitment) with a different prefix than the items (e..g, aff_com1_likert7). This allows us to delete the items (i.e., columns that start with aff_com) but not the final scale score (i.e., a column starts with affective_commitment). This example again illustrates how carefully you need to think about your column and scale naming conventions.

```
# Create scale scores
## mutate commands create scale scores
## select commands with "-" remove items after scale creation
analytic_data_survey <- analytic_data_survey %>%
  rowwise() %>%
  mutate(affective_commitment = mean(c_across(starts_with("aff_com")),
                                     na.rm = TRUE)) %>%
  mutate(continuance_commitment = mean(c_across(starts_with("contin_com")),
                                       na.rm = TRUE)) %>%
  mutate(normative_commitment = mean(c_across(starts_with("norm_com")),
                                     na.rm = TRUE)) %>%
  mutate(job_satisfaction = mean(c_across(starts_with("job_aff")),
                                 na.rm = TRUE)) %>%
  ungroup() %>%
  select(-starts_with("aff_com")) %>%
  select(-starts_with("contin_com")) %>%
  select(-starts_with("norm_com")) %>%
  select(-starts_with("job_aff"))
```

We can see our data now has the columns: affective_commitment, continuance_commitment, normative_commitment, job satisfaction.

As well, all of the items used to create the scale scores have been removed.

```
glimpse(analytic_data_survey)
```

```
Rows: 150
Columns: 8
$ duration_in_seconds    <dbl> 244, 134, 237, 110, 151, 92, 393, 165, 315, 262~
$ year_of_birth          <dbl> 1961, 2006, 2008, 1985, 1953, 1957, 1998, 1970,~
$ sex                    <fct> female, female, female, male, intersex, interse~
$ participant_id         <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, ~
$ affective_commitment   <dbl> 6.000000, 3.833333, 2.833333, 4.000000, 3.83333~
$ continuance_commitment <dbl> 4.333333, 3.833333, 2.666667, 4.500000, 5.33333~
$ normative_commitment   <dbl> 3.333333, 3.166667, 2.333333, 4.666667, 4.33333~
$ job_satisfaction       <dbl> 2.75, 4.00, 3.50, 2.25, 3.50, 2.50, 2.75, 2.75,~
```

2.5.3.6.3 Removing fast responders

You might be interested in removing people who responded to the survey too quickly. Notice two things when we glimpse the ananlytic_data_survey data below.

```
glimpse(analytic_data_survey)
```

```
Rows: 150
Columns: 8
$ duration_in_seconds    <dbl> 244, 134, 237, 110, 151, 92, 393, 165, 315, 262~
$ year_of_birth          <dbl> 1961, 2006, 2008, 1985, 1953, 1957, 1998, 1970,~
$ sex                    <fct> female, female, female, male, intersex, interse~
$ participant_id         <fct> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, ~
$ affective_commitment   <dbl> 6.000000, 3.833333, 2.833333, 4.000000, 3.83333~
$ continuance_commitment <dbl> 4.333333, 3.833333, 2.666667, 4.500000, 5.33333~
$ normative_commitment   <dbl> 3.333333, 3.166667, 2.333333, 4.666667, 4.33333~
$ job_satisfaction       <dbl> 2.75, 4.00, 3.50, 2.25, 3.50, 2.50, 2.75, 2.75,~
```

First, there are 150 rows so there are 150 participants in this data set. Second, there is a column called duration_in_seconds. You can use this column to remove people who responded too quickly. For example, you might decide that anyone who responded in less than 120 seconds (i.e., 2 minutes) should be removed from the data set. You can do so with the code below:

```
analytic_data_survey <- analytic_data_survey %>%
  filter(duration_in_seconds >= 120)

glimpse(analytic_data_survey)
```

```
Rows: 129
Columns: 8
```

```
$ duration_in_seconds    <dbl> 244, 134, 237, 151, 393, 165, 315, 262, 370, 20~
$ year_of_birth          <dbl> 1961, 2006, 2008, 1953, 1998, 1970, 1983, 1959,~
$ sex                    <fct> female, female, female, intersex, intersex, mal~
$ participant_id         <fct> 1, 2, 3, 5, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17~
$ affective_commitment   <dbl> 6.000000, 3.833333, 2.833333, 3.833333, 3.16666~
$ continuance_commitment <dbl> 4.333333, 3.833333, 2.666667, 5.333333, 4.00000~
$ normative_commitment   <dbl> 3.333333, 3.166667, 2.333333, 4.333333, 2.50000~
$ job_satisfaction       <dbl> 2.75, 4.00, 3.50, 3.50, 2.75, 2.75, 3.50, 4.00,~
```

This reduces the data set to the 129 people who took at least 2 minutes (i.e., 120 seconds) to complete the survey.

2.5.4 Complete script

The complete script script_qualtrics.R for the data file data_qualtrics.csv is presented below.

```r
# Date: YYYY-MM-DD
# Name: your name here
# Example: Single occasion survey
# Load data
library(tidyverse)
library(janitor)
library(skimr)

# read just the header row to get column names
survey_file <- "data_qualtrics.csv"
col_names <- names(read_csv(survey_file,
                            n_max = 0,
                            show_col_types = FALSE))

# skip the first 3 rows, read data using names from above
raw_data <- read_csv(survey_file,
                     col_names = col_names,
                     skip = 3,
                     show_col_types = FALSE)

# Qualtrics columns to remove (any_of() ignores names not present)
cols_to_remove <- c("StartDate",
                    "EndDate",
                    "Status",
                    "IPAddress",
                    "Progress",
```

```
                      "Finished",
                      "RecordedDate",
                      "ResponseId",
                      "RecipientLastName",
                      "RecipientFirstName",
                      "RecipientEmail",
                      "ExternalReference",
                      "LocationLatitude",
                      "LocationLongitude",
                      "DistributionChannel",
                      "UserLanguage")

# remove the unwanted Qualtrics columns
raw_data <- raw_data |>
  select(!any_of(cols_to_remove))

analytic_data_survey <- raw_data

# Initial cleaning
## Convert column names to tidyverse style guide
## Remove empty rows and columns
analytic_data_survey <- analytic_data_survey %>%
  remove_empty("rows") %>%
  remove_empty("cols") %>%
  clean_names()

glimpse(analytic_data_survey)

# Convert variables to factors as needed
## Convert sex to factor
analytic_data_survey <- analytic_data_survey %>%
  mutate(sex = as_factor(sex))


## Participant identification to factor
## Participant identification column must be created first
## get the number of rows in the data set
N <- dim(analytic_data_survey)[1]

## create set of factor levels
participant_id_levels <- factor(seq(1, N))
```

```r
## put the factor levels into a column called participant_id
analytic_data_survey <- analytic_data_survey %>%
  mutate(participant_id = participant_id_levels)

# Screen factors
## screen
analytic_data_survey %>%
  select(sex) %>%
  summary()

# Check existing levels (a different way than lines above)
levels(analytic_data_survey$sex)

## change to desired order
analytic_data_survey <- analytic_data_survey %>%
  mutate(sex = fct_relevel(sex,
                           "female",
                           "intersex",
                           "male"))

# Check for unexpected levels
expected_sex <- c("female", "intersex", "male")
unexpected <- setdiff(levels(analytic_data_survey$sex), expected_sex)
if (length(unexpected) > 0) {
  warning("Unexpected sex levels found: ", paste(unexpected, collapse = ", "))
}


# Screen numeric variables
analytic_data_survey %>%
  select(year_of_birth) %>%
  skim()


# Convert Commitment items to numeric  values
## Check levels for a likert7 item
analytic_data_survey %>%
  pull(aff_com1_likert7) %>%
  unique()

## Define word-to-number mapping 7-
likert7_recode <- c(
```

```
  "Strongly Disagree" = 1,
  "Moderately Disagree" = 2,
  "Slightly Disagree" = 3,
  "Neither Agree nor Disagree" = 4,
  "Slightly Agree" = 5,
  "Moderately Agree" = 6,
  "Strongly Agree" = 7
)

## Convert text responses to numeric values
analytic_data_survey <- analytic_data_survey %>%
  mutate(across(
    .cols = contains("likert7"),
    .fns = ~ likert7_recode[.x]
  ))



# Convert Job Satisfaction items to numeric  values
## Check levels for a likert5 item
analytic_data_survey %>%
  pull(job_aff1_likert5) %>%
  unique()

## Define word-to-number mapping 5-point scale
likert5_recode <- c(
  "Strongly Disagree" = 1,
  "Disagree" = 2,
  "Neutral" = 3,
  "Agree" = 4,
  "Strongly Agree" = 5
)

## Convert text responses to numeric values
analytic_data_survey <- analytic_data_survey %>%
  mutate(across(
    .cols = contains("likert5"),
    .fns = ~ likert5_recode[.x]
  ))


# Reverse key items
```

```r
## Reverse key likert7 items
analytic_data_survey <- analytic_data_survey %>%
  mutate(across(
    .cols = ends_with("_likert7rev"),
    .fns = ~ (7 + 1) - .x
  )) %>%
  rename_with(
    .fn = ~ str_replace(.x, "_likert7rev", "_likert7"),
    .cols = ends_with("_likert7rev")
  )


## No likert5 items are reverse-keyed but if they were
## You would adapt the code above replacing 8 (one higher than 7) to 6 (one higher than 5)


# Create scale scores
## mutate commands create scale scores
## select commands with "-" remove items after scale creation
analytic_data_survey <- analytic_data_survey %>%
  rowwise() %>%
  mutate(affective_commitment = mean(c_across(starts_with("aff_com")),
                                     na.rm = TRUE)) %>%
  mutate(continuance_commitment = mean(c_across(starts_with("contin_com")),
                                       na.rm = TRUE)) %>%
  mutate(normative_commitment = mean(c_across(starts_with("norm_com")),
                                     na.rm = TRUE)) %>%
  mutate(job_satisfaction = mean(c_across(starts_with("job_aff")),
                                 na.rm = TRUE)) %>%
  ungroup() %>%
  select(-starts_with("aff_com")) %>%
  select(-starts_with("contin_com")) %>%
  select(-starts_with("norm_com")) %>%
  select(-starts_with("job_aff"))


glimpse(analytic_data_survey)

analytic_data_survey <- analytic_data_survey %>%
  filter(duration_in_seconds >= 120)

glimpse(analytic_data_survey)
```

2.6 Extra Help

2.6.1 Qualtrics

- University of Guelph Qualtrics Login page
- University of Guelph Qualtrics FAQ page

2.6.2 Surveys

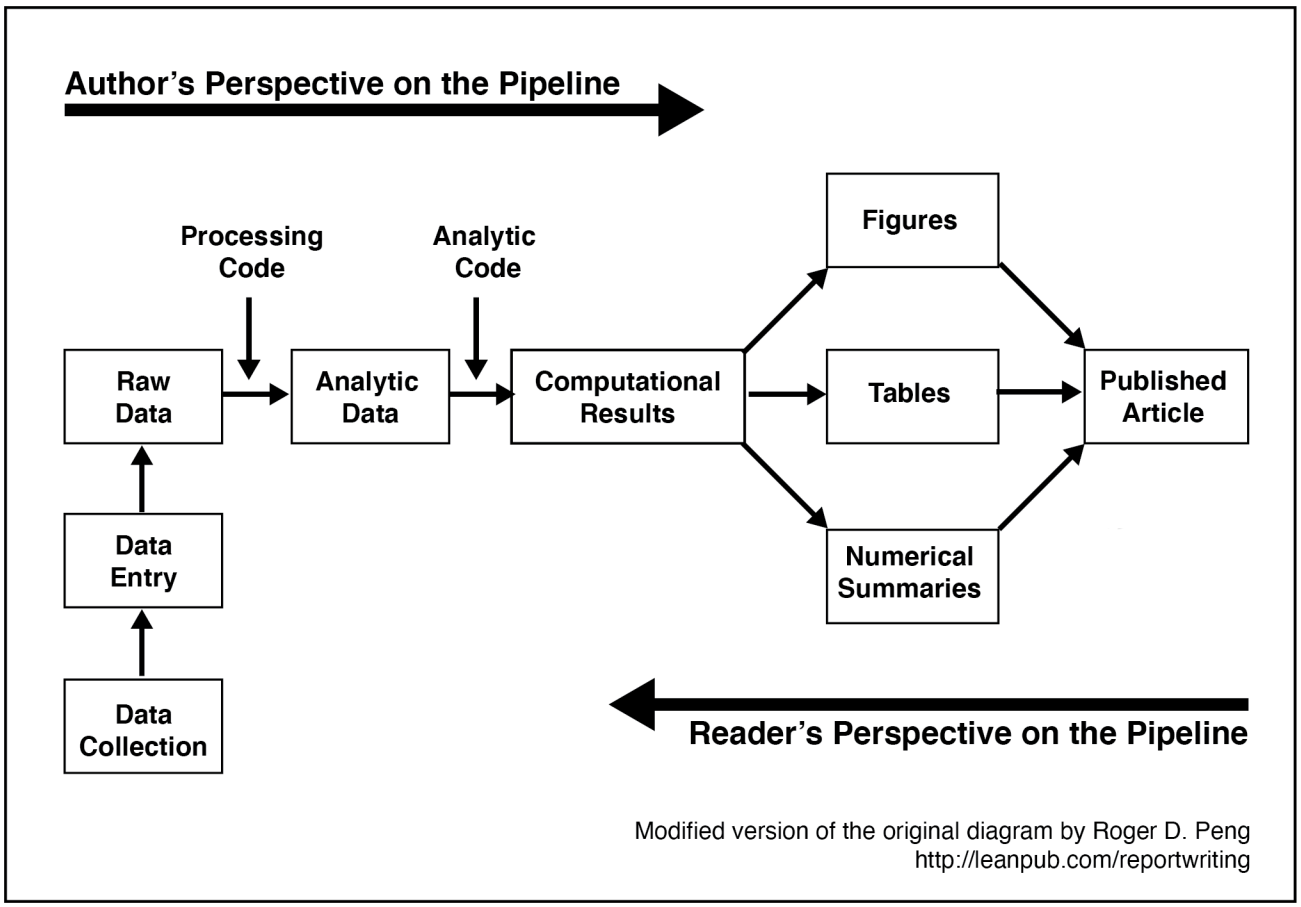
- Dillman Method book can be found here
- Duke University Survey site

3 Data Science Pipelines in R

Associated Files
data-qualtrics.csv
my-project.zip

3.1 Introduction

In the previous chapter, you learned how to read Qualtrics data and prepare it for analysis using a single R script. That approach works well for learning—but as your projects grow more complex, a single script becomes unwieldy. In this tutorial, you'll learn how to organize your work into a **data science pipeline**: a series of modular scripts that each handle one specific task.



Recall we first saw the pipeline in this figure in a previous chapter. Today we will create a series of scripts. Most of these scripts correspond to the "Processing Code" step in the above figure. That is, we have expanded the Processing Code to have its own multi-step pipeline. In this chapter, we walk you through creating that pipeline.

Note that that data set we use in this chapter is not the same as the one in the previous chapter. The data set for this chapter has been modified to include missing data to illustrate how to evaluate and handle missing data in a pipeline.

3.2 Why Use a Pipeline?

Consider the single-script approach you've been using. It might look something like this:

```
# Everything in one file...
library(tidyverse)
library(janitor)
```

```r
# Import
raw_data <- read_csv("data-qualtrics.csv", skip = 3)

# Clean
analytic_data <- raw_data |>
  select(-StartDate, -EndDate, ...) |>
  clean_names()

# Recode Likert scales
# ... 50 more lines ...

# Create scales
# ... 30 more lines ...

# Exclusions
# ... 20 more lines ...

# Analysis
# ... and so on
```

This approach has several problems:

1. **Difficult to debug**: When something goes wrong on line 247, finding the cause is tedious.
2. **Hard to modify**: Changing one part risks breaking another.
3. **Poor collaboration**: Lab members can't work on different sections simultaneously.
4. **No checkpoints**: If you make a mistake, you must re-run everything from scratch.

A pipeline solves these problems by breaking your workflow into discrete, manageable steps.

3.3 The Pipeline Structure

Our data science pipeline uses a specific file and folder structure to keep everything organized. Here's an overview of the structure. We will recreate this structure in this chapter. You will create each file and folder as we go. Then you will paste the code for each step into the appropriate script file. Alternatively, you can download the completed project from the associated files at the top of this page.

```
My Project/
   00-script-master.R      # Runs the entire pipeline     (Preprocessing Code)
```

```
01-import.R             # Load and initial setup       (Preprocessing Code)
02-clean-recode.R       # Factors and Likert recoding  (Preprocessing Code)
03-missing-data.R       # Evaluate missingness         (Preprocessing Code)
04-create-scales.R      # Compute scale scores         (Preprocessing Code)
05-exclusions.R         # Apply exclusion criteria     (Preprocessing Code)
06-analysis-wrapper.R   # Analysis and visualization   (Analytic Code)
07-analysis.R           # Analysis and visualization   (Analytic Code)
data-raw/                # Folder: Original data (NEVER modify)
data-interim/            # Folder: Checkpoint files between steps
data-processed/          # Folder: Final analytic dataset
output/                  # Folder: Analysis output, tables, figures, etc.
```

This structure follows the principle of separating **raw data**, **interim data**, **processed data**, and **output**. Each script reads from the previous step and writes to the next.

3.4 Output naming conventions

We use clear, consistent naming conventions for files we create. For examples figures will begin with `figure-`, tables with `table-`, and data files with `data-`. This makes it easy to identify file types at a glance.

3.5 Create Required Folders

Begin by creating the necessary folders in your project directory. You can do this manually or with R code. But we will begin mannually for clarity.

First, create a folder with the name of your project. This folder might be called "My-Thesis" or something similar. Today we will call this folder "My Project".

Second, create empty R script files inside "My Project" folder named as follows:

- `00-script-master.R`
- `01-import.R`
- `02-clean-recode.R`
- `03-missing-data.R`
- `04-create-scales.R`
- `05-exclusions.R`
- `06-analysis-wrapper.R`

- `07-analysis.R`

Third, inside "My Project", create the following folders:

- `data-raw/`
- `data-interim/`
- `data-processed/`
- `output/`

Next, place your raw data file (e.g., `data-qualtrics.csv`) inside the `data-raw/` folder. This file should never be modified.

3.6 The Master Script

The master script is the conductor of your pipeline. It loads all necessary packages once, then calls each step in order.

```
# 00-script-master.R

# Date: 2026-01-15 (Use YYYY-MM-DD format)
# Name: [Your Name]
# Purpose: Master - Runs all data preparation steps in order

# 1. Setup Environment ------------------------------------------------
library(tidyverse)
library(janitor)
library(skimr)
library(naniar) # Package for missing data visualization

# Clear memory to ensure reproducibility
rm(list = ls())

# Create directories if they don't exist (helpful for students)
if(!dir.exists("data-interim")) dir.create("data-interim")
if(!dir.exists("data-processed")) dir.create("data-processed")
if(!dir.exists("output")) dir.create("output")


# 2. Execute Pipeline -------------------------------------------------
```

```
# Step 1: Import and Anonymize
message("--- Running Step 1: Import ---")
source("01-import.R", echo = TRUE)

# Step 2: Cleaning and Recoding
message("--- Running Step 2: Cleaning ---")
source("02-clean-recode.R", echo = TRUE)

# Step 3: Evaluate missing data
message("--- Running Step 3: Evaluate missing data ---")
source("03-missing-data.R", echo = TRUE)

# Step 4: Create scales
message("--- Running Step 4: Create scales ---")
source("04-create-scales.R", echo = TRUE)

# Step 5: Exclusions
message("--- Running Step 5: Excluding participants ---")
source("05-Exclusions.R", echo = TRUE)

# Step 6: Analysis Wraper
message("--- Running Step 6: Analysis and Visualization ---")
source("06-analysis-wrapper.R", echo = TRUE)


message("--- Pipeline Complete! ---")
```

Notice several important features:

- `source()` with `echo = TRUE` shows you what each script is doing
- `message()` calls provide clear progress indicators
- All packages are loaded once at the beginning

3.7 Step 1: Import

The import script handles loading raw data and initial cleanup. This is essentially the first part of your single script, isolated:

```
# 01-import.R

# Step 1: Import and Initial Setup
```

```r
# Load Data ------------------------------------------------------------
survey_file <- "data-raw/data-qualtrics.csv"

# Read header row for names
col_names <- names(read_csv(survey_file, n_max = 0, show_col_types = FALSE))

# Read data (skipping Qualtrics header rows 2 and 3)
raw_data <- read_csv(survey_file,
                     col_names = col_names,
                     skip = 3,
                     show_col_types = FALSE,
                     na = c("", "NA","999")) # Treat blank cells as NA


# Select Columns -------------------------------------------------------
# NOTE: We comment out "Duration" so we can use it for exclusions later!
cols_to_remove <- c("StartDate", "EndDate", "Status", "IPAddress", "Progress",
                    "Finished", "RecordedDate", "ResponseId", "RecipientLastName",
                    "RecipientFirstName", "RecipientEmail", "ExternalReference",
                    "LocationLatitude", "LocationLongitude", "DistributionChannel",
                    "UserLanguage")

analytic_data_survey <- raw_data |>
  select(!any_of(cols_to_remove)) |>
  clean_names() |>
  remove_empty("rows") |>
  remove_empty("cols")

# Create Participant ID ------------------------------------------------
# Best practice: Create ID immediately so we can track rows if order changes
analytic_data_survey <- analytic_data_survey %>%
  mutate(participant_id = row_number()) %>%
  relocate(participant_id) # Move to first column

# Save Interim File ----------------------------------------------------
# We use .rds because it preserves R formatting (unlike CSV)
write_rds(analytic_data_survey, "data-interim/01-imported.rds")
```

Key insight: We save using `.rds` format, not `.csv`. The RDS format preserves R data types (factors, dates, etc.) exactly as they are. CSV files lose this information.

3.8 Step 2: Clean and Recode

This script handles factor creation and Likert scale recoding:

```
# 02-clean-recode.R

# Step 2: Cleaning, Factors, and Recoding

# Load Previous Step ------------------------------------------------------
analytic_data_survey <- read_rds("data-interim/01-imported.rds")

# Factor Handling ---------------------------------------------------------
# Convert sex to factor and fix levels
analytic_data_survey <- analytic_data_survey %>%
  mutate(sex = as_factor(sex)) %>%
  mutate(sex = fct_relevel(sex, "female", "intersex", "male"))

# Safety Check: Warn if new genders appear
expected_sex <- c("female", "intersex", "male")
unexpected <- setdiff(levels(analytic_data_survey$sex), expected_sex)
if (length(unexpected) > 0) {
  warning("Check Data! Unexpected sex levels found: ", paste(unexpected, collapse = ", "))
}

# Likert Recoding (Text to Numbers) ---------------------------------------

# Define Mappings
likert7_recode <- c(
  "Strongly Disagree" = 1,
  "Moderately Disagree" = 2,
  "Slightly Disagree" = 3,
  "Neither Agree nor Disagree" = 4,
  "Slightly Agree" = 5,
  "Moderately Agree" = 6,
  "Strongly Agree" = 7
)

likert5_recode <- c(
  "Strongly Disagree" = 1,
  "Disagree" = 2,
  "Neutral" = 3,
  "Agree" = 4,
```

```
  "Strongly Agree" = 5
)

# Apply Mappings
analytic_data_survey <- analytic_data_survey %>%
  # Recode 7-point scales
  mutate(across(
    .cols = contains("likert7"),
    .fns = ~ likert7_recode[.x]
  )) %>%
  # Recode 5-point scales
  mutate(across(
    .cols = contains("likert5"),
    .fns = ~ likert5_recode[.x]
  ))

# Reverse Keying -------------------------------------------------------------
analytic_data_survey <- analytic_data_survey %>%
  mutate(across(
    .cols = ends_with("_likert7rev"),
    .fns = ~ (7 + 1) - .x
  )) %>%
  # Rename columns to remove the 'rev' suffix now that they are fixed
  rename_with(
    .fn = ~ str_replace(.x, "_likert7rev", "_likert7"),
    .cols = ends_with("_likert7rev")
  )

# Save Interim File ----------------------------------------------------------
write_rds(analytic_data_survey, "data-interim/02-cleaned.rds")
```

Notice how our naming conventions (`likert7`, `likert7rev`, `likert5`) make it possible to apply recoding to multiple columns at once using `contains()` and `ends_with()`. This is why consistent naming matters.

3.9 Step 3: Missing Data Evaluation

Before creating scales, we should understand our missing data. This step generates reports but doesn't modify the data. Importantly, we need to check the reports it creates in the `output` directory after we run this code. The amount and location of the missing data may suggest a different course of action than the default approach used in these scripts.

```r
# 03-missing-data.R

# Step 3: Missing Data Evaluation

# Load Previous Step ------------------------------------------------------
analytic_data_survey <- read_rds("data-interim/02-cleaned.rds")


# Define Scale Items ------------------------------------------------------
# We select the columns relevant for scoring to check them specifically
scale_items <- analytic_data_survey %>%
  select(starts_with("aff_com"),
         starts_with("contin_com"),
         starts_with("norm_com"),
         starts_with("job_aff"))

# Missing Data Diagnosis --------------------------------------------------

# 1. Text Report: Items with the most missing data
message("--- Missing Data Report by Item ---")
missing_summary <- scale_items %>%
  miss_var_summary() %>%  # From naniar
  filter(n_miss > 0)

print(missing_summary)
write_csv(missing_summary, "output/data-table-missing-by-item.csv")

# 2. Text Report: Participants with too much missing data
# Check if any participant is missing more than 20% of the scale items
message("--- Participants with > 20% Missing Data ---")
high_missing_participants <- analytic_data_survey %>%
  rowwise() %>%
  mutate(pct_missing = mean(is.na(c_across(c(starts_with("aff_com"),
                                             starts_with("contin_com"),
                                             starts_with("norm_com"),
                                             starts_with("job_aff"))))) * 100) %>%
  ungroup() %>%
  filter(pct_missing > 20) %>%
  select(participant_id, pct_missing)

print(high_missing_participants)
write_csv(high_missing_participants, "output/data-table-missing-by-participant.csv")
```

```r
# 3. Visual Report
# Generates a map of missingness (Black = Missing, Grey = Present)
# We use 'print()' to ensure the plot renders when running from a script
message("--- Generating Missing Data Map ---")
plot_missing <- vis_miss(scale_items) +
  labs(title = "Missing Data Map (Scale Items Only)") +
  theme(axis.text.x = element_text(angle = 60, hjust = 0, vjust = 0, size = 16),
        axis.text.x.top = element_text(margin = margin(b = 2)),
        axis.text.y = element_text(size = 16),
        plot.title = element_text(size = 16),
        legend.text = element_text(size = 16),
        legend.title = element_text(size = 16),
        plot.margin = margin(t = 10, r = 10, b = 10, l = 10))

ggsave("output/figure-missing-data-map.png", plot = plot_missing, width = 20, height = 20, dp

# Data not altered so no need to save
```

This step saves diagnostic outputs to the `output/` folder. You can review these to decide whether any participants should be excluded for excessive missing data.

3.10 Step 4: Create Scales

Now we compute scale scores by averaging items:

```r
# 04-create-scales.R

# Step 4: Scale Creation

# Load Previous Step ------------------------------------------------------
analytic_data_survey <- read_rds("data-interim/02-cleaned.rds")


# Create Scale Scores -----------------------------------------------------
# Note: We use rowwise() for accurate mean calculations across columns
analytic_data_survey <- analytic_data_survey %>%
  rowwise() %>%
  mutate(
    affective_commitment = mean(c_across(starts_with("aff_com")), na.rm = TRUE),
    continuance_commitment = mean(c_across(starts_with("contin_com")), na.rm = TRUE),
```

```
    normative_commitment = mean(c_across(starts_with("norm_com")), na.rm = TRUE),
    job_satisfaction = mean(c_across(starts_with("job_aff")), na.rm = TRUE)
  ) %>%
  ungroup() # Always ungroup after rowwise()!

# Clean up Item columns (Optional - keeps dataset clean)
analytic_data_survey <- analytic_data_survey %>%
  select(-starts_with("aff_com"),
         -starts_with("contin_com"),
         -starts_with("norm_com"),
         -starts_with("job_aff"))

# Save Interim File ------------------------------------------------------
write_rds(analytic_data_survey, "data-interim/03-scales-created.rds")
```

Important: Always call `ungroup()` after `rowwise()`. Forgetting this can cause unexpected behavior in subsequent operations.

3.11 Step 5: Exclusions

This step applies your preregistered exclusion criteria:

```
# 05-exclusions.R

# Step 5: Exclusions

# Rules for excluding participants should be preregistered.
# In this example, we exclude participants who completed the survey in under 2 minutes.


# Load Previous Step -----------------------------------------------------
analytic_data_survey <- read_rds("data-interim/03-scales-created.rds")

# Exclusions -------------------------------------------------------------
# Filter out speeders (Requires 'duration_in_seconds' from Step 1)
initial_n <- nrow(analytic_data_survey)

# Only keep participants with duration >= 120 seconds (2 minutes)
analytic_data_survey <- analytic_data_survey %>%
  filter(duration_in_seconds >= 120)
```

```r
# Print a message telling the student how many were dropped
final_n <- nrow(analytic_data_survey)
message(paste("Dropped", initial_n - final_n, "participants due to speed checks."))

# Final Save ------------------------------------------------------------------
write_rds(analytic_data_survey, "data-processed/analytic-data-final.rds")

# Display final structure
glimpse(analytic_data_survey)
```

Note that this script saves to `data-processed/`, not `data-interim/`. This signals that the data is now ready for analysis.

3.12 Step 6: Analysis Wrapper

This script runs the analysis script but captures the output.

```r
# 06-analysis-wrapper.R

# Step 6: Analysis Wraper

# This wrapper runs the analysis script and captures its output to a text file
# The text file is saved in the output directory
# The analysis script is assumed to be "07-analysis.R"

capture.output(source("07-analysis.R", echo =  TRUE),
                      file = "output/analysis-output.txt")
```

3.13 Step 7: Analysis

Finally, the analysis script loads the clean data and performs your statistical analyses. Note that this file is not directly run in the 00-script-master.R; instead, it is called via the wrapper in Step 6. We do this so the output can be captered to a text file and placed in the output folder.

```r
# 07-analysis.R

# Step 7: Analysis
```

```r
# Rules for excluding participants should be preregistered.
# In this example, we exclude participants who completed the survey in under 2 minutes.


# Clear memory to ensure reproducibility
rm(list = ls())

# Set random seed for reproducibility (e.g., geom_jitter)
set.seed(12345)

# 1. Setup Environment for Analyses ------------------------------------------------
library(tidyverse)
library(GGally)
library(skimr)
library(apaTables)

# Load Previous Step ---------------------------------------------------------
analytic_data_survey <- read_rds("data-processed/analytic-data-final.rds")

# Analysis -------------------------------------------------------------------


skim(analytic_data_survey)



# Example Analysis: Descriptive Statistics

data_plot <- analytic_data_survey |>
  select(contains("commitment"))


apa.cor.table(data_plot,
                  filename = "output/table-correlation-descriptive-statistics.doc",
                  table.number = 1)


font_size <- 10
cor_plot <- ggpairs(data_plot,
                    upper = "blank",
                    lower = list(continuous = wrap("points", alpha = 0.3)),
                    diag = list(continuous = wrap("barDiag",
```

```
                                                    bins = 15,
                                                    color = "black",
                                                    fill = "black",
                                                    alpha = 1))) +
  theme_linedraw(font_size) +
  theme(panel.spacing = unit(.8, "lines"))

ggsave("output/figure-pairs-plot.png",
        plot = cor_plot,
        width = 7,
        height = 7,
        units = "in",
        dpi = 300)

print(cor_plot)
```

3.14 Converting Your Single Script

To convert your existing single script to a pipeline:

1. **Create the folder structure** shown above
2. **Identify natural breakpoints** in your code (import, cleaning, scales, exclusions, analysis)
3. **Move each section** to its own numbered script
4. **Add `read_rds()` at the start** of each script (except step 1)
5. **Add `write_rds()` at the end** of each script
6. **Create a master script** that sources each step in order

3.15 Benefits of the Pipeline Approach

The pipeline approach offers several advantages over a single script:

Single Script	Pipeline Approach
Everything in one file	Modular, numbered scripts
Difficult to debug	Each step is isolated and testable
Must re-run everything	Can re-run from any checkpoint
Hard to collaborate	Team members can work on different steps
No clear progress indicators	Messages show which step is running

The pipeline approach requires a bit more setup, but the benefits for debugging, collaboration, and reproducibility make it well worth the effort—especially as your projects grow in complexity.

> **Using AI is Easier with a Pipeline Approach**
>
> The modular pipeline structure makes it easier to work with AI assistants like ChatGPT or Claude. You can share individual scripts for help without overwhelming the AI with your entire project.

3.16 Output from the Pipeline

To run the entire pipeline, simply open `script-master.R` and run it. You can also run individual scripts if you need to re-do just one step (for example, if you change your exclusion criteria, you only need to re-run steps 5 and 6).

Let's run the master script now to see the pipeline in action!

Recall, running the script places the missing data evaluation in the output folder. Let's check that folder to see the results of that step.

3.16.1 Missing data by item

We can see the percentage of missing data for each item by examining the CSV file created in the output folder called `data-table-missing-by-item.csv`.

variable	n_miss	pct_miss
contin_com2_likert7	32	21.333333
norm_com1_likert7	9	6.000000
norm_com2_likert7	8	5.333333
norm_com3_likert7	8	5.333333
aff_com4_likert7	7	4.666667
aff_com5_likert7	7	4.666667
contin_com3_likert7	7	4.666667
contin_com4_likert7	7	4.666667
contin_com5_likert7	7	4.666667
contin_com6_likert7	7	4.666667
aff_com6_likert7	6	4.000000
contin_com1_likert7	6	4.000000
aff_com1_likert7	5	3.333333
aff_com2_likert7	5	3.333333

variable	n_miss	pct_miss
aff_com3_likert7	5	3.333333
norm_com4_likert7	2	1.333333
norm_com5_likert7	2	1.333333

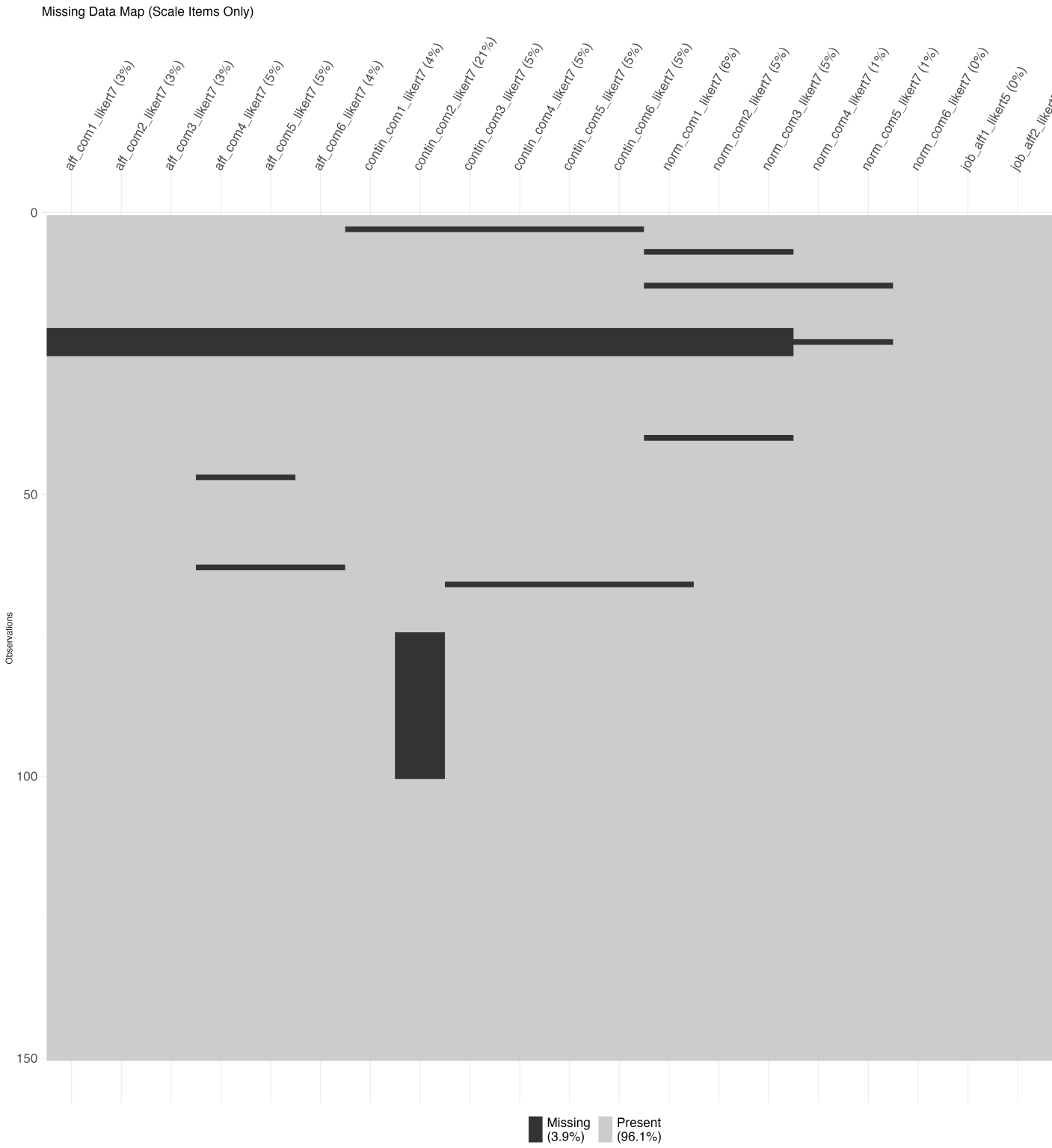
3.16.2 Missing data by person

We can see the percentage of missing data for each person by examining the CSV file created in the output folder called `data-table-missing-by-participant.csv`.

participant_id	pct_missing
3	27.27273
13	22.72727
21	68.18182
22	68.18182
23	77.27273
24	68.18182
25	68.18182
66	22.72727

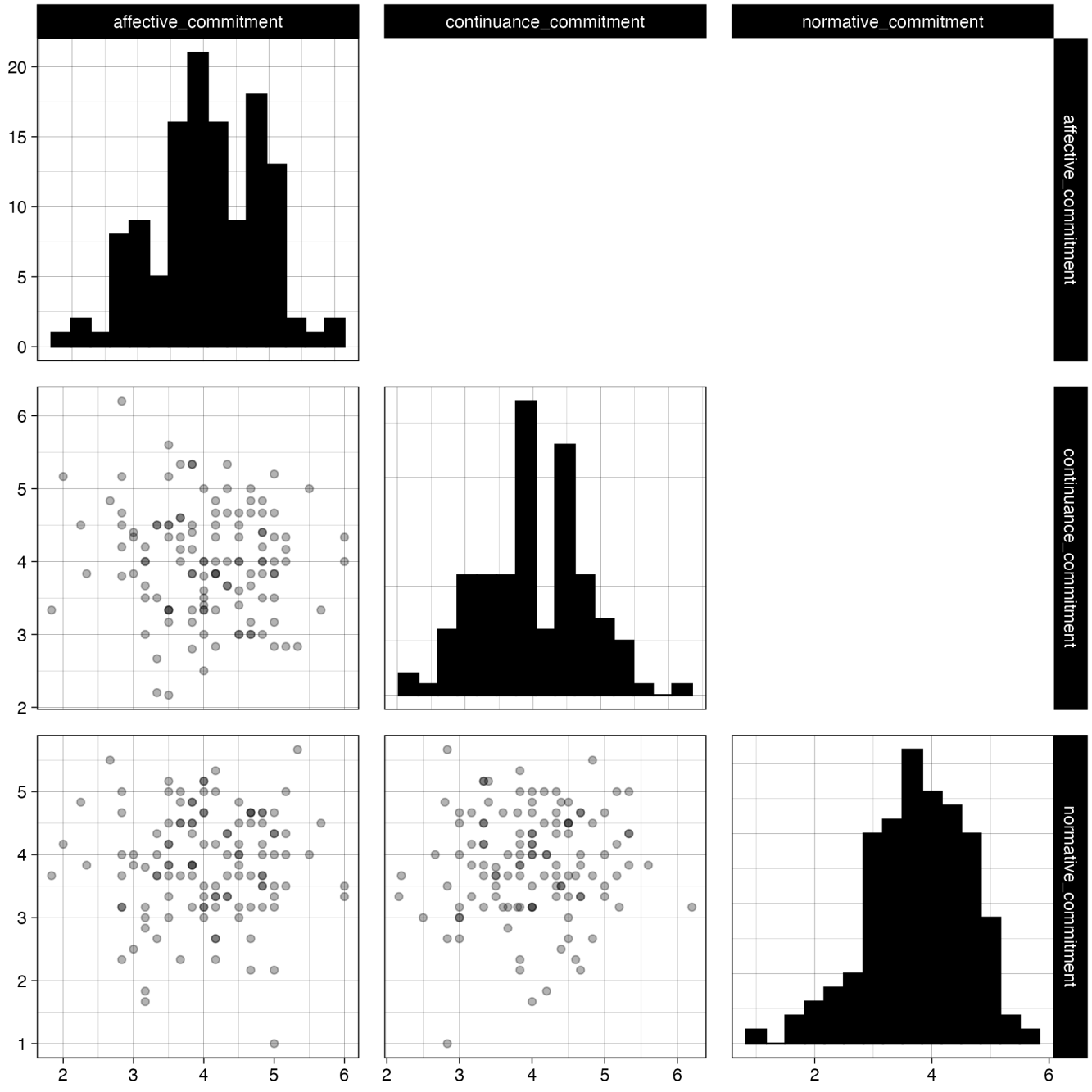
3.16.3 Visual missing data map

We can see the visual missing data map created in the output folder called `figure-missing-data-map.png`. This visual map helps identify patterns in the missing data across items and participants.


Missing Data Map (Scale Items Only)
aff_com1_likert7 (3%)
aff_com2_likert7 (3%)
aff_com3_likert7 (3%)
aff_com4_likert7 (5%)
aff_com5_likert7 (5%)
aff_com6_likert7 (4%)
contin_com1_likert7 (4%)
contin_com2_likert7 (21%)
contin_com3_likert7 (5%)
contin_com4_likert7 (5%)
contin_com5_likert7 (5%)
contin_com6_likert7 (5%)
norm_com1_likert7 (6%)
norm_com2_likert7 (5%)
norm_com3_likert7 (5%)
norm_com4_likert7 (1%)
norm_com5_likert7 (1%)
norm_com6_likert7 (0%)
job_aff1_likert5 (0%)
0
50
100
150
Observations
Missing (3.9%)
Present (96.1%)

3.16.4 Pairs plot

We can see the pairs plot created in the output folder called `figure-pairs-plot.png`. The ggpairs function lets us see relationships among all the scales at once - and see the histogram for each scale along the diagonal.



3.17 Using SPSS Data?

If your data is in SPSS format (.sav), you can easily read it into R using the `haven` package. Here's how to modify the import script to handle SPSS files. Simply replace the read_csv line in `01-import.R` with the following code:

```
library(haven)
# Read SPSS data
raw_data <- read_sav("data-raw/data-qualtrics.sav")

# Assign value labels (e.g., 1 in SPSS sex column becomes "male" if that's how you set up the
raw_data <- as_factor(raw_data)

# check which columns are now factors after using as_factor()
glimpse(raw_data)
```

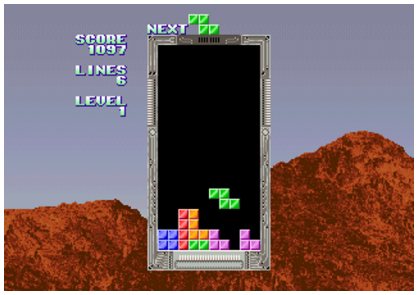
4 Regression and correlation

The following CRAN packages must be installed:

Required CRAN Packages
tidyverse
apaTables
cocor
janitor
psych

Required Data
data_cor_sample_video_game.csv

4.1 Population example

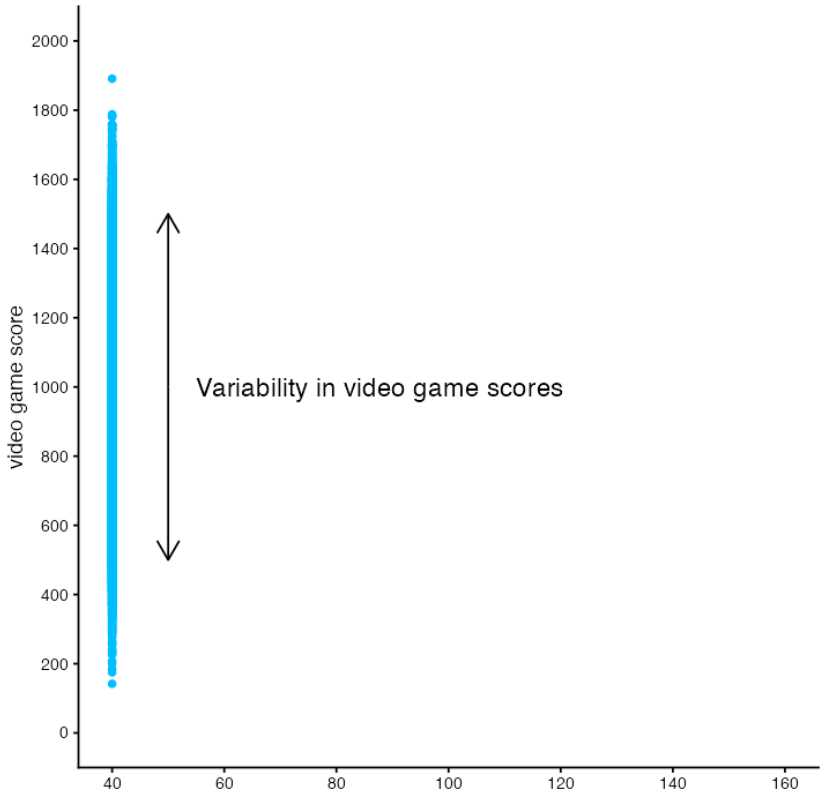


Consider the following the scenario, we want to examine the extent to which IQ predicts video game scores for people who live in the City of Guelph. We want to make conclusions about people who live in the City of Guelph so we refer to Guelph citizens as our population. Because we are interested in using IQ to predict video game score we refer to IQ as the **predictor**. The value being predicted is video game score and we refer to that variable as the **criterion** (i.e., the dependent variable). Imagine, for a moment, that we are actually able to get an IQ and a video game score for everyone in the City of Guelph (N = 100,000).

4.1.1 No predictor

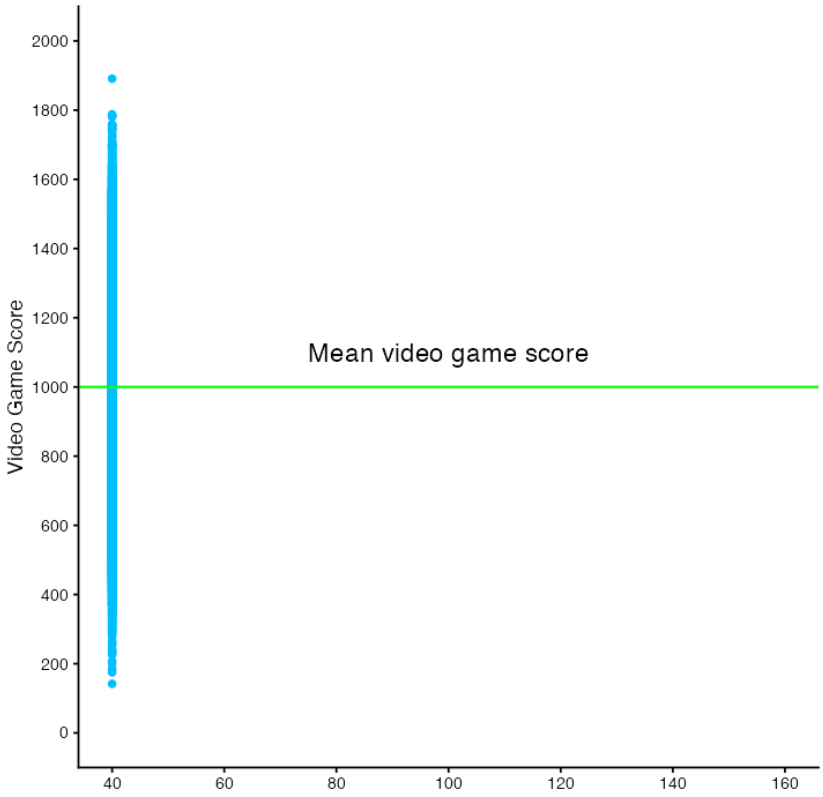
We illustrate the range and variability of video game scores for everyone in the population below. There are 100,000 blue dots on this graph. Each blue dot represents a person in the **population**. The vertical position of the dot indicates each person's video game.

- Notice the large number of dots (i.e., people) - there are so many that it's hard to see individual dots/people.
- The dots illustrate the range of video game scores - everyone did not obtain the same score.
- We want to try to understand why, in this population, some people have higher vs lower video game scores.
- Said another way, we want to explain, for this population, the variability in video game scores.
- Correlation/regression can never provide evidence of causation (or explanation) but we can use those analyses to find a pattern in our data that is *consistent* with a causal relation. Then we conduct a second **experimental** study to determine if their really is a causal relation.



Without a predictor variable, our best estimate of a person's video game score is the population mean. Moreover, without a predictor, we have the same estimate for everyone in the population - the mean video game score for the population. We have no way of creating an individualized estimate of someone's video game score. The mean video game score is illustrated in the figure below with a horizontal green line.

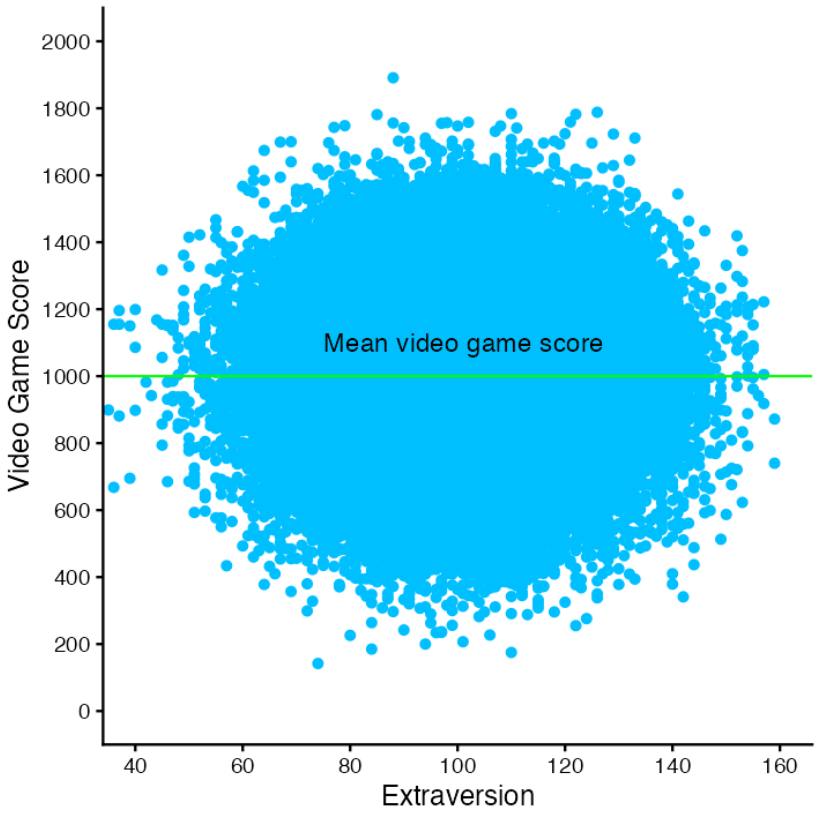
```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



4.1.2 Weak relation

We can plot a variable (e.g., extraversion) against video game score to see if there is a relation between the two. In this case there doesn't appear to be a relation. Indeed, this graph illustrates a zero correlation between extraversion and video game scores - the weakest possible relation for a predictor. Effectively, there is no linear relation between extraversion and video game scores.

As before, we place a horizontal green line on the graph to indicate the mean video game score. We also place a red regression line (i.e., best-fit line) on the graph; however, the red line is completely hidden by the green line representing the mean video game score. In this case, with these data, knowing a person's extraversion score **does not** allow us to provide an individualized estimate of a person's video game score. As a result, extraversion does not help us explain the variability in video game scores. That is, extraversion scores do not allow us to explain why some people have high video game score whereas other people have low video game scores.



4.1.3 Strong relation

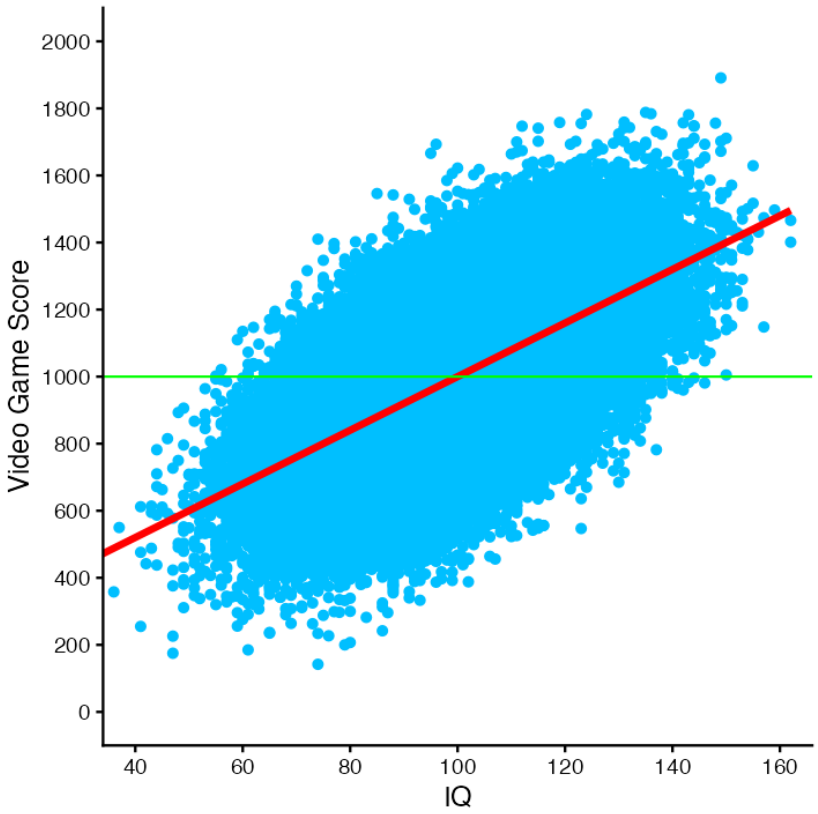
What if we were to try another variable – like IQ? The graph below illustrates a positive linear relation between IQ and video game score. As before, we place a green horizontal line on the graph to indicate the population mean. Additionally, we place a red regression line (i.e., a best-fit line) on the graph. This red regression line represents, for each person, an individualized estimate of video game score based on their IQ. The population-level regression line has the slope 8.00; which indicate that as IQ increase by 1.0 point video game score increases by 8.00 points. The equation for the regression line is:

$$\widehat{score} = 8.00(IQ) + 200.31$$

Or using the more generic X/Y notation:

$$\hat{Y} = 8.00(X) + 200.31$$

You can see in the graph below that, for some people, the individualized estimate of video game score (i.e., the y-axis position of the red line) is higher than the population mean. That is, in some cases the red regression is line is higher than the green line for the population mean. For other people, the individualized estimate of video game score (i.e., the y-axis position of the red line) is lower than the population mean. That is, in some cases the red regression is line is lower than the green line for the population mean. It appears that individuals with a high IQ tend to have a high video game score whereas individuals with a low IQ tend of have a low video game score. The regression line provides a more nuanced estimate for individual's video game score than you can obtain by simply using the same estimation (i.e., the population mean) for everyone.



But how good is this model of the data? There are a variety of way of assessing model fit. We present two below. First, when you are only concerned about how well a single predictor performs, as is the case here, you can use a correlation. The symbol for the population correlation is ρ. In these data, $\rho = .60$. The correlation coefficient can range from -1 to 1. The further the correlation is from zero - the stronger the relation between the predictor and the criterion. That is, the further the correlation is from zero the better a linear model fits the data. A positive correlation indicates that as one variable increases the other variable also increases. A negative correlation indicates that as one variable increases the other variable decreases.

Cohen's benchmarks are below:

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

An alternative, and more general, means of assessing the quality of statistical model is to use R^2. This indexes the proportion of video game scores that are accounted for by a statistical model. One important attribute of R^2 is that can be used when there are multiple predictors. When there is only one predictor, R^2 is simply the correlation squared. Thus, at the population level, **when there is only one predictor**:

$$R^2 = \rho^2$$

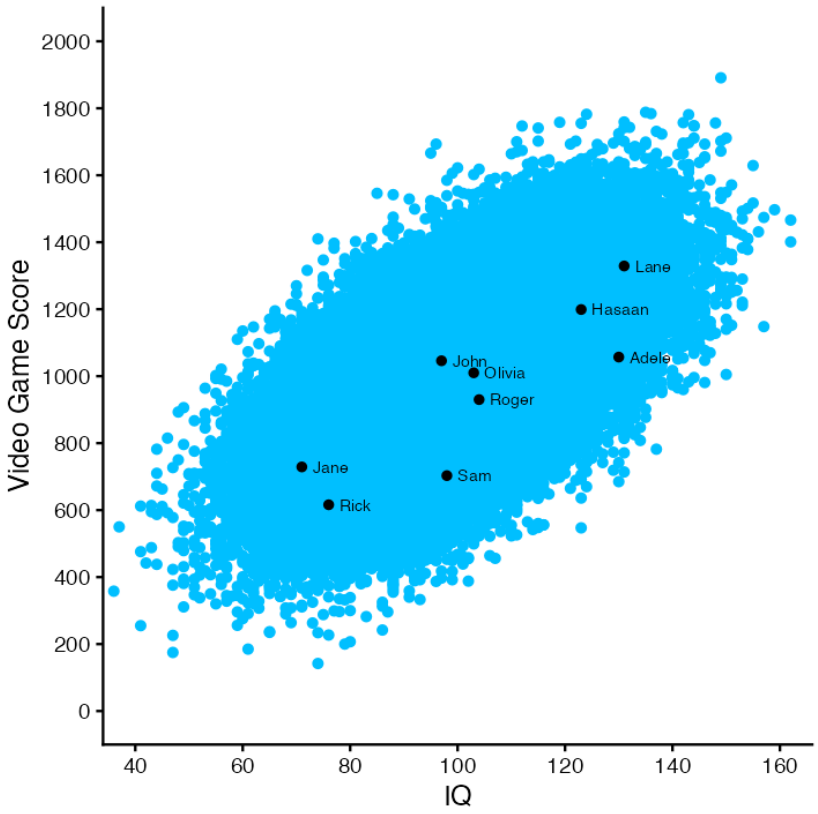
To summarize, the population-level values (i.e., parameters):

- Slope = 8.00
- $\rho = .60$
- $R^2 = .36$

In the next section we use sample-level data to estimate these population-level values.

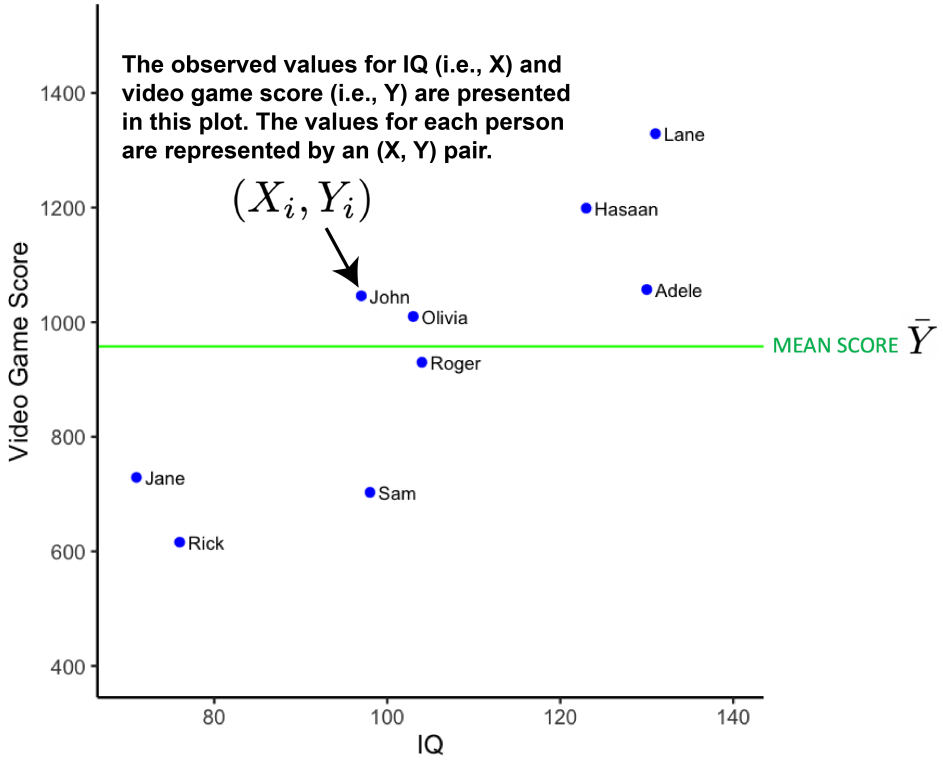
4.2 Consider a sample

Unfortunately, we never/rarely have data for everyone in the population (in this example everyone in the City of Guelph). Consequently, we usually have to select a subset of the population as a sample and use sample data for our analyses. In the figure below there are 100,000 blue dots - each dot represents an individual in the population. Additionally, there are also 9 black dots. These black dots are a random subset of the population – our sample. We will use the data from our sample (i.e., the 9 black dots) to estimate the slope, correlation, and R^2 for the population (i.e., the 100,000 blue dots).



We always need to remember the values calculated from our sample (statistics) are only estimates of the population-level parameters; estimates that are likely are likely to differ from population parameters due to sampling error.

Let's look at our sample in more detail. Notice, in the figure below, that there is one data point (X_i, Y_i) for each of the 9 people. We illustrate the mean video game score for the 9 people with the horizontal green line.



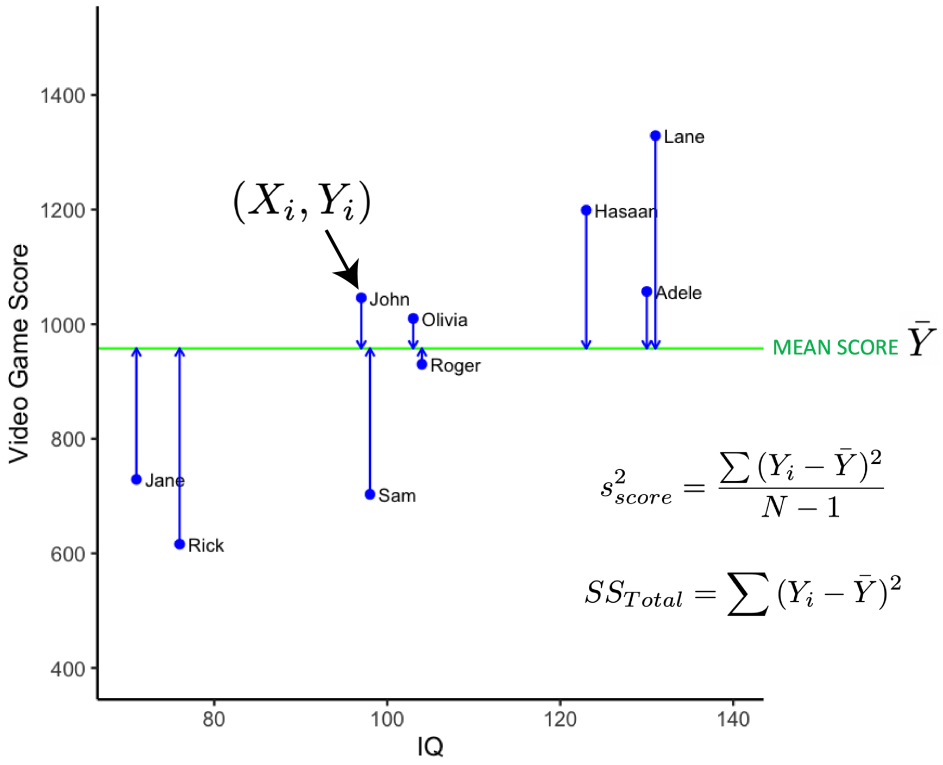
We can calculate the variance of the video game scores using the formula below.

$$s^2_{score} = \frac{\sum (Y_i - \bar{Y})^2}{N - 1}$$

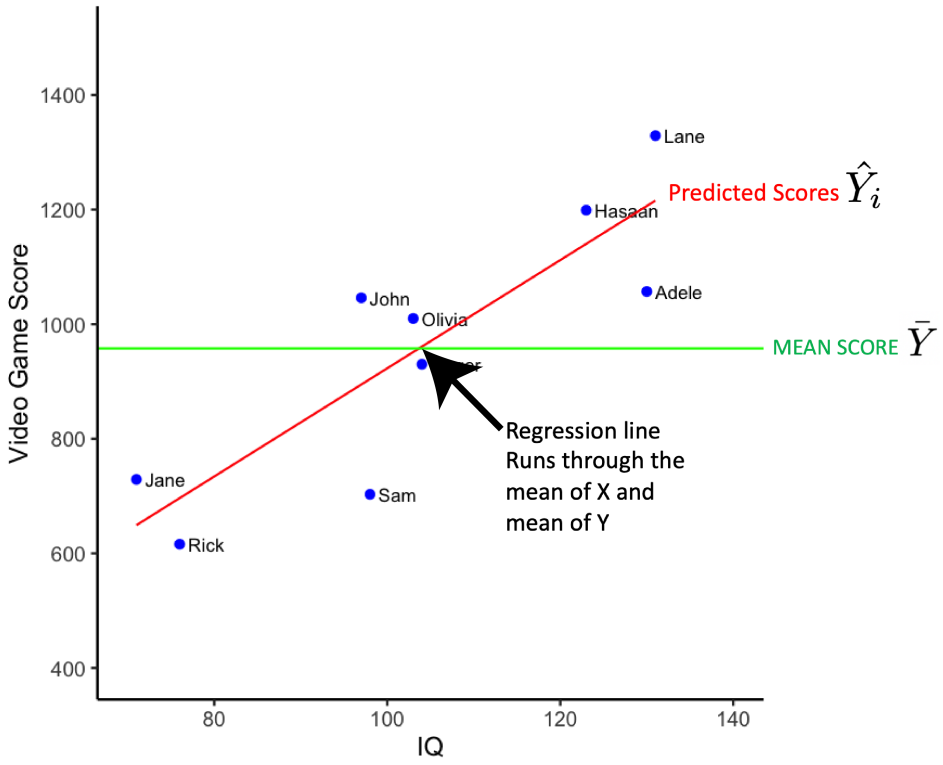
It's often useful to just focus on the numerator of this equation. We call this Sum of Squares Total (SSR):

$$SS_{Total} = \sum (Y_i - \bar{Y})^2$$

The values used in the **Sum of Squares Total** calculation are illustrated in the figure below. The vertical blue line indicates the difference between Y_i and $\bar{Y}$. The SS_{Total} value indexes the variability of the **actual** video game scores around the sample mean.



In the figure below we add the regression line in red. The regression line is a statistical model for the data (a best-fit line). The regression line will always run through the joint mean of the two variables (i.e., the $[\bar{X}, \bar{Y}]$ point).



Recall that the regression line represents an individualized estimate of each person's video game score derived from their IQ (via the regression equation). Later we will show you how to calculate the regression line. For now, accept that the regression equation for the sample in standard notation is:

$$\hat{Y}_i = 9.44(X_i) - 21.21$$

And contextualized to the variable names is:

$$\widehat{score}_i = 9.44(IQ_i) - 21.21$$

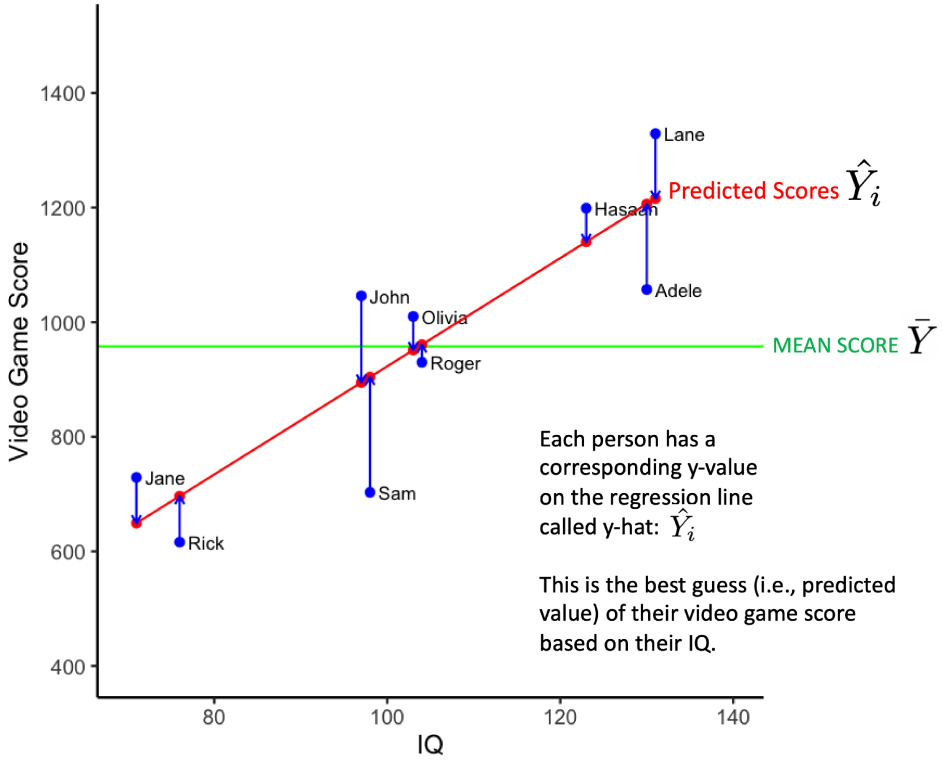
Therefore, the predicted value for Jane is:

$$\begin{aligned} \widehat{score}_i &= 9.44(IQ_i) - 21.21 \\ &= 9.44(71) - 21.21 \\ &= 649(\text{rounded}) \end{aligned}$$

You can see this calculation for everyone in the sample:

	X	Y	Y-hat
name	iq	video_game	video_game_hat
Jane	71	729	649
Rick	76	616	696
John	97	1046	895
Sam	98	703	904
Olivia	103	1010	951
Roger	104	930	961
Hasaan	123	1199	1140
Adele	130	1057	1206
Lane	131	1329	1216

In the graph below each person is represented by a blue dot. The estimate of each person's video game score, $\hat{Y}_i$, derived from the regression equation, is indicated by a red dot on the red regression line. The vertical blue lines are used to indicate, for each person (i.e., blue dot, Y_i), the estimate of their video game score (i.e., red dot, $\hat{Y}_i$).

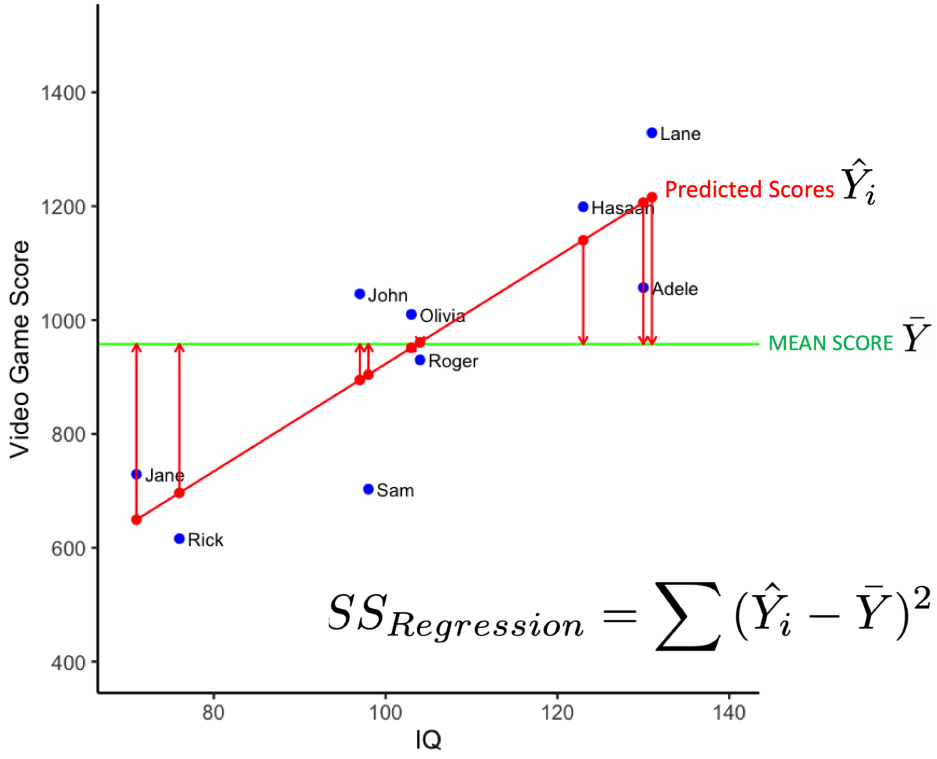


We can calculate the extent to which individualized estimates are better than the sample mean for modeling the data. That is, we can calculate the extent to which the regression line is better at modeling the data than the mean line. We do so by calculating the extent to which

the individualized estimates on the regression line differ from the sample mean. This is done with the calculation below for the **Sum of Squares Regression**.

$$SS_{Regression} = \sum (\hat{Y}_i - \bar{Y})^2$$

The values used in the **Sum of Squares Regression** calculation are illustrated in the figure below. The vertical red line indicates the difference between $\hat{Y}_i$ and $\bar{Y}$. The $SS_{Regression}$ value indexes the variability of the **estimates** of video game scores around the sample mean. The longer the vertical red line (i.e., the larger the $(\hat{Y}_i - \bar{Y})$ difference) the better the model. Longer vertical red lines are associated with models that do a better job of accounting for variability in video scores.



So far we have calculated two values, $SS_{Regression}$ and SS_{Total}. The SS_{Total} value indexes the variability of **actual** video game scores about the sample mean (it's the numerator for the variance calculation). In contrast, $SS_{Regression}$ indexes the variability of **estimated** video game scores about the sample mean. We can calculate the proportion of the variability in actual scores accounted for the statistical model (i.e., regression line) using R^2:

$$R^2 = \frac{SS_{Regression}}{SS_{Total}}$$

We can also think of this as in terms of variance (because N-1 terms cancel each other out).

$$
\begin{aligned}
R^2 &= \frac{\text{Variance of predicted scores}}{\text{Variance of actual scores}} \\
&= \frac{\frac{\sum(\hat{Y}_i - \bar{Y})^2}{N-1}}{\frac{\sum(Y_i - \bar{Y})^2}{N-1}} \\
&= \frac{\sum(\hat{Y}_i - \bar{Y})^2}{\sum(Y_i - \bar{Y})^2} \\
&= \frac{SS_{Regression}}{SS_{Total}}
\end{aligned}
$$

4.2.1 Regression

Let's obtain the actual value for R^2, as well as the slope, using R. We can obtain the regression model (i.e.. linear model or lm) using the command below:

```
lm_object <- lm(video_game ~ iq,
                data = sample_data)
```

We display the result using apaTables:

```
library(apaTables)

apa.reg.table(lm_object,
              table.number = 1,
              filename = "regression_table.doc")
```

Which produces the output:

Table 1

Regression results using video_game as the criterion

Predictor	*b*	*b* 95% CI [LL, UL]	*beta*	*beta* 95% CI [LL, UL]	sr^2	sr^2 95% CI [LL, UL]	*r*	Fit
(Intercept)	-21.21	[-555.27, 512.85]						
iq	9.44**	[4.39, 14.50]	0.86	[0.40, 1.32]	.74	[.17, .86]	.86**	R^2 = .736** 95% CI[.17,.86]

Note. A significant *b*-weight indicates the beta-weight and semi-partial correlation are also significant. *b* represents unstandardized regression weights. *beta* indicates the standardized regression weights. sr^2 represents the semi-partial correlation squared. *r* represents the zero-order correlation. *LL* and *UL* indicate the lower and upper limits of a confidence interval, respectively.
* indicates $p < .05$. ** indicates $p < .01$.

If we examine the fit column on the far right of the output above we see $R^2 = .74$, 95% CI [.17, .86]. This value indicates that in this sample 74% of the variability in video scores is accounted accounted for by the statistical model (i.e., red regression line). The confidence interval suggests a plausible range of values for the R^2 at the population-level is .17 to .86. Notice that this range captures that population-level R^2 of .36 that we calculated from the entire population previously.

$$R^2 = .736 = .74$$

If we examine the b column in the output we can create the regression equations below:

$$\hat{Y}_i = 9.44(X_i) - 21.21$$
$$\widehat{score}_i = 9.44(IQ_i) - 21.21$$

This tells use the slope in our sample is 9.44, 95% CI [4.39, 14.50]. That is, in the sample, each IQ point is associated with an additional 9.44 points in the video game. The population regression line might have a smaller/larger slope. The 95% confidence intervals tells us that a plausible range of values for the slope of the regression line at the population-level is 4.39 to 14.50. Notice that this range captures that population-level slope of 8.00 that we calculated from the entire population previously.

Additional regression details are provided with the command below.

```
summary(lm_object)
```

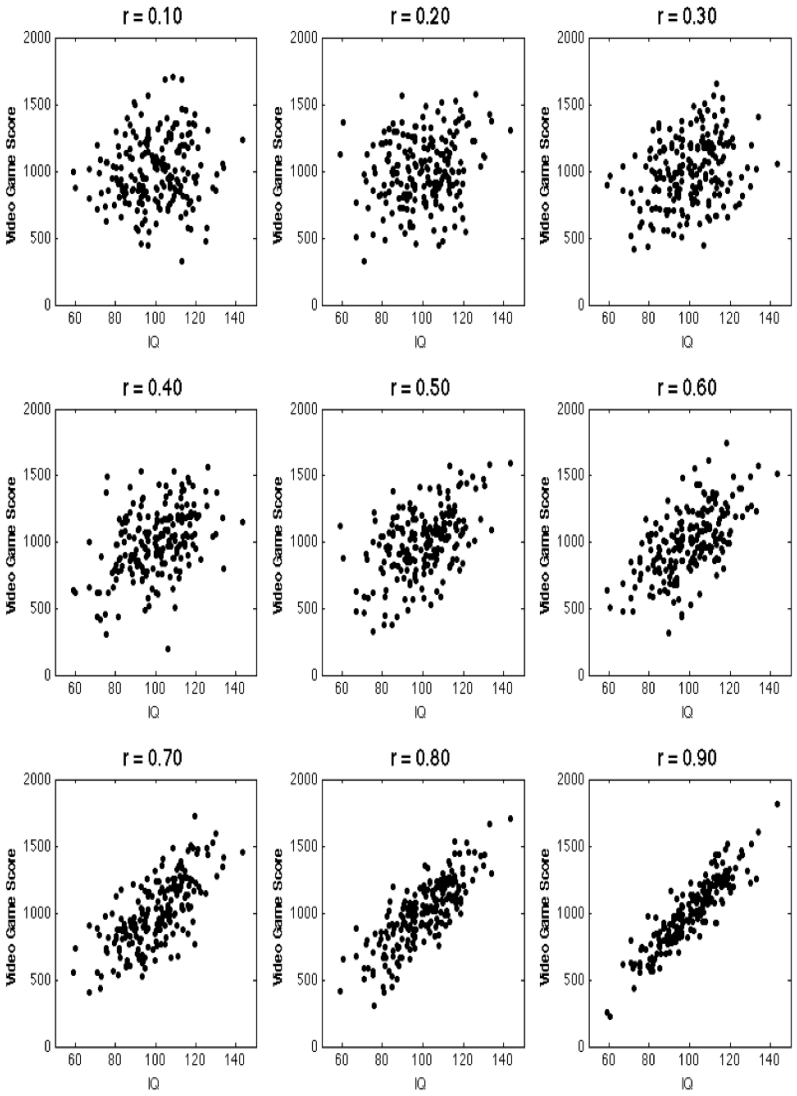
```
Call:
lm(formula = video_game ~ iq, data = sample_data)

Residuals:
    Min      1Q  Median      3Q     Max
-201.16  -80.42   58.63   79.79  151.28

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -21.209    225.855  -0.094  0.92781
iq             9.443      2.138   4.417  0.00309 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 130.5 on 7 degrees of freedom
Multiple R-squared:  0.7359,    Adjusted R-squared:  0.6982
F-statistic: 19.51 on 1 and 7 DF,  p-value: 0.003093
```

4.2.2 Correlation

As discussed previously, a correlation can be considered a fit index for a linear regression line. That is, a correlation indicates the extent to which the data fit a straight line (i.e., the extent to which the data fit a linear model). Correlation values range between -1 and +1. The further a correlation value is from 0 the more tightly points will cluster around the regression line.

- A **positive correlation** indicates that as one value **increases** the other value **increases**. For example, as height increases weight increases.
- A **negative correlation** indicates that as one value **increases** the other values **decreases**. For example, as study time increases the number of errors on an exam decreases.

A few possible **positive** correlations are illustrated below – notice the relation between the graph and the strength of the correlation.



You can obtain the correlation from our sample data with the command below:

```
cor.test(sample_data$iq, sample_data$video_game,
         na.action = "pairwise.complete.obs")
```

```
    Pearson's product-moment correlation
```

```
data:  sample_data$iq and sample_data$video_game
t = 4.4167, df = 7, p-value = 0.003093
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.4502656 0.9695860
sample estimates:
      cor
0.8578603
```

From this output we extract the numbers below in APA reporting style:

There was a positive relation between IQ and video game score, such that as IQ increased, so did video game score, $r = .86$, 95% CI[.45, .97], $p = .003$, $N = 9$.

4.2.3 Graphing

A scatter plot can be made code below:

```
my_plot <- ggplot(data = sample_data,
                  mapping = aes(x = iq, y = video_game)) +
  geom_point(color = "blue", size = 4) +
  coord_cartesian(xlim = c(70, 140), ylim = c(400, 1500)) +
  scale_y_continuous(breaks = seq(400, 1500, by = 200)) +
  scale_x_continuous(breaks = seq(70, 140, by = 20)) +
  labs(x = "IQ", y = "Video Game Score") +
  theme_classic(24)
```

4.3 Comparing correlations

In this part of the chapter we compare correlations within and across studies we do so with the cocor package. We begin by obtaining a data set from the psych package. Note that we **do not** use the library(psych) command due to conflicts with the tidyverse.

```
# Obtain the bfi data set from the psych package
bfi <- psych::bfi

# remove empty rows/columns and clean the variable names
bfi <- bfi %>%
  remove_empty("rows") %>%
```

```
  remove_empty("cols") %>%
  clean_names()
```

Check out the large number of columns.

```
glimpse(bfi)
```

```
Rows: 2,800
Columns: 28
$ a1        <int> 2, 2, 5, 4, 2, 6, 2, 4, 4, 2, 4, 2, 5, 5, 4, 4, 4, 5, 4, 4, ~
$ a2        <int> 4, 4, 4, 4, 3, 6, 5, 3, 3, 5, 4, 5, 5, 5, 5, 3, 6, 5, 4, 4, ~
$ a3        <int> 3, 5, 5, 6, 3, 5, 5, 1, 6, 6, 5, 5, 5, 5, 2, 6, 6, 5, 5, 6, ~
$ a4        <int> 4, 2, 4, 5, 4, 6, 3, 5, 3, 6, 6, 5, 6, 6, 2, 6, 2, 4, 4, 5, ~
$ a5        <int> 4, 5, 4, 5, 5, 5, 5, 1, 3, 5, 5, 5, 4, 6, 1, 3, 5, 5, 3, 5, ~
$ c1        <int> 2, 5, 4, 4, 4, 6, 5, 3, 6, 6, 4, 5, 5, 4, 5, 5, 4, 5, 5, 1, ~
$ c2        <int> 3, 4, 5, 4, 4, 6, 4, 2, 6, 5, 3, 4, 4, 4, 5, 5, 4, 5, 4, 1, ~
$ c3        <int> 3, 4, 4, 3, 5, 6, 4, 4, 3, 6, 5, 5, 3, 4, 5, 5, 4, 5, 5, 1, ~
$ c4        <int> 4, 3, 2, 5, 3, 1, 2, 2, 4, 2, 3, 4, 2, 2, 2, 3, 4, 4, 4, 5, ~
$ c5        <int> 4, 4, 5, 5, 2, 3, 3, 4, 5, 1, 2, 5, 2, 1, 2, 5, 4, 3, 6, 6, ~
$ e1        <int> 3, 1, 2, 5, 2, 2, 4, 3, 5, 2, 1, 3, 3, 2, 3, 1, 1, 2, 1, 1, ~
$ e2        <int> 3, 1, 4, 3, 2, 1, 3, 6, 3, 2, 3, 3, 3, 2, 4, 1, 2, 2, 2, 1, ~
$ e3        <int> 3, 6, 4, 4, 5, 6, 4, 4, NA, 4, 2, 4, 3, 4, 3, 6, 5, 4, 4, 4,~
$ e4        <int> 4, 4, 4, 4, 4, 5, 5, 2, 4, 5, 5, 5, 2, 6, 6, 6, 5, 6, 5, 5, ~
$ e5        <int> 4, 3, 5, 4, 5, 6, 5, 1, 3, 5, 4, 4, 4, 5, 5, 4, 5, 6, 5, 6, ~
$ n1        <int> 3, 3, 4, 2, 2, 3, 1, 6, 5, 5, 3, 4, 1, 1, 2, 4, 4, 6, 5, 5, ~
$ n2        <int> 4, 3, 5, 5, 3, 5, 2, 3, 5, 5, 3, 5, 2, 1, 4, 5, 4, 5, 6, 5, ~
$ n3        <int> 2, 3, 4, 2, 4, 2, 2, 2, 2, 5, 4, 3, 2, 1, 2, 4, 4, 5, 5, 5, ~
$ n4        <int> 2, 5, 2, 4, 4, 2, 1, 6, 3, 2, 2, 2, 2, 2, 2, 5, 4, 4, 5, 1, ~
$ n5        <int> 3, 5, 3, 1, 3, 3, 1, 4, 3, 4, 3, NA, 2, 1, 3, 5, 5, 4, 2, 1,~
$ o1        <int> 3, 4, 4, 3, 3, 4, 5, 3, 6, 5, 5, 4, 4, 5, 5, 6, 5, 5, 4, 4, ~
$ o2        <int> 6, 2, 2, 3, 3, 3, 2, 2, 6, 1, 3, 6, 2, 3, 2, 6, 1, 1, 2, 1, ~
$ o3        <int> 3, 4, 5, 4, 4, 5, 5, 4, 6, 5, 5, 4, 4, 4, 5, 6, 5, 4, 2, 5, ~
$ o4        <int> 4, 3, 5, 3, 3, 6, 6, 5, 6, 5, 6, 5, 5, 4, 5, 3, 6, 5, 4, 3, ~
$ o5        <int> 3, 3, 2, 5, 3, 1, 1, 3, 1, 2, 3, 4, 2, 4, 5, 2, 3, 4, 2, 2, ~
$ gender    <int> 1, 2, 2, 2, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 2, 2, ~
$ education <int> NA, NA, NA, NA, NA, 3, NA, 2, 1, NA, 1, NA, NA, NA, 1, NA, N~
$ age       <int> 16, 18, 17, 17, 17, 21, 18, 19, 19, 17, 21, 16, 16, 16, 17, ~
```

Let's select a small subset of the columns for our example:

```
bfi <- bfi %>%
  select(a1, c1, e1, o1, gender)
```

You can confirm the smaller set of columns:

```
glimpse(bfi)
```

```
Rows: 2,800
Columns: 5
$ a1     <int> 2, 2, 5, 4, 2, 6, 2, 4, 4, 2, 4, 2, 5, 5, 4, 4, 4, 5, 4, 4, 5, ~
$ c1     <int> 2, 5, 4, 4, 4, 6, 5, 3, 6, 6, 4, 5, 5, 4, 5, 5, 4, 5, 5, 1, 4, ~
$ e1     <int> 3, 1, 2, 5, 2, 2, 4, 3, 5, 2, 1, 3, 3, 2, 3, 1, 1, 2, 1, 1, 3, ~
$ o1     <int> 3, 4, 4, 3, 3, 4, 5, 3, 6, 5, 5, 4, 4, 5, 5, 6, 5, 5, 4, 4, 6, ~
$ gender <int> 1, 2, 2, 2, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 2, 2, 1, ~
```

These columns are single items from a personality measure.

- a1 (Agreeableness)
- c1 (Conscientiousness)
- e1 (Extraversion)
- o1 (Opennness)

You can obtain a condensed correlation matrix using the cor() command. You can specify use = “pairwise.complete.obs” for pairwise correlation - the documentation covers other options. The round(2) command rounds the correlations to two decimal places.

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

```
          a1    c1    e1    o1 gender
a1      1.00  0.03  0.11  0.01  -0.16
c1      0.03  1.00 -0.02  0.17   0.01
e1      0.11 -0.02  1.00 -0.10  -0.13
o1      0.01  0.17 -0.10  1.00  -0.10
gender -0.16  0.01 -0.13 -0.10   1.00
```

Or we could use apaTable apa.cor.table() command:

```
library(apaTables)

apa.cor.table(bfi,
              table.number = 1,
              filename = "table_1_bfi.doc")
```

Table 1

Means, standard deviations, and correlations with confidence intervals

Variable	*M*	*SD*	1	2	3	4
1. a1	2.41	1.41				
2. c1	4.50	1.24	.03 [-.01, .07]			
3. e1	2.97	1.63	.11** [.07, .14]	-.02 [-.06, .01]		
4. o1	4.82	1.13	.01 [-.02, .05]	.17** [.13, .20]	-.10** [-.14, -.06]	
5. gender	1.67	0.47	-.16** [-.19, -.12]	.01 [-.02, .05]	-.13** [-.16, -.09]	-.10** [-.14, -.07]

Note. M and *SD* are used to represent mean and standard deviation, respectively. Values in square brackets indicate the 95% confidence interval for each correlation. The confidence interval is a plausible range of population correlations that could have caused the sample correlation (Cumming, 2014). * indicates $p < .05$. ** indicates $p < .01$.

Inspecting the above table you see that the correlation between a1 and c1 with is $r = .03$, 95% CI [-.01, .07]. Likewise, the correlation between e1 and o1 is $r = -.10$, 95% CI [-.14, -.06]. We obtain the p-values for these relations below.

4.3.1 p-values

The code below obtains the p-value for the a1/c1 relation - a value of .144.

```
cor.test(bfi$a1, bfi$c1)
```

```
    Pearson's product-moment correlation

data:  bfi$a1 and bfi$c1
t = 1.4617, df = 2762, p-value = 0.1439
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.009490233  0.065018669
sample estimates:
       cor
0.02780283
```

The code below obtains the p-value for the e1/o1 relation - a value sufficiently small we report it as $p < .001$

```
cor.test(bfi$e1, bfi$o1)
```

```
    Pearson's product-moment correlation

data:  bfi$e1 and bfi$o1
t = -5.2971, df = 2757, p-value = 1.27e-07
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.13717646 -0.06329333
sample estimates:
       cor
-0.1003733
```

With p-values in hand we can write this up as:

Inspecting the above table you see that the correlation between a1 and c1 with is $r = .03$, 95% CI [-.01, .07], $p = .144$. Likewise, the correlation between e1 and o1 is $r = -.10$, 95% CI [-.14, -.06], $p < .001$.

4.3.2 Within a data set

In this section we look at comparing correlation within a single data set.

4.3.2.1 Non-overlapping correlations

We will compare the correlation to between (a1, c1) to the correlation between (e1, o1) with the cocor package. In this case, because neither of the variables in the first correlation (a1, c1) are in the second correlation (e1, o1) we refer to this as a non-overlapping correlation comparison.

The cocor command will provide a lot of output. We are most interested in the last part of the output corresponding to Zou (2007) which provides the confidence interval for the difference in the correlations. But we also want to examine the first part of the output which will show us the two original correlations (a1, c1) = 0.0276, and (e1, o1) = -0.1002 (these are the .03 and -.10 values prior to rounding). As well it also shows us the difference between them, Difference: r.jk - r.hm = 0.1278.

```r
library(cocor)

cocor( ~ a1 + c1 | e1 + o1, data = as.data.frame(bfi))
```

```
  Results of a comparison of two nonoverlapping correlations based on dependent groups

Comparison between r.jk (a1, c1) = 0.0276 and r.hm (e1, o1) = -0.1002
Difference: r.jk - r.hm = 0.1278
Related correlations: r.jh = 0.1036, r.jm = 0.0125, r.kh = -0.0259, r.km = 0.1688
Data: as.data.frame(bfi): j = a1, k = c1, h = e1, m = o1
Group size: n = 2724
Null hypothesis: r.jk is equal to r.hm
Alternative hypothesis: r.jk is not equal to r.hm (two-sided)
Alpha: 0.05

pearson1898: Pearson and Filon's z (1898)
  z = 4.7832, p-value = 0.0000
  Null hypothesis rejected

dunn1969: Dunn and Clark's z (1969)
  z = 4.7676, p-value = 0.0000
  Null hypothesis rejected

steiger1980: Steiger's (1980) modification of Dunn and Clark's z (1969) using average correla
  z = 4.7671, p-value = 0.0000
  Null hypothesis rejected

raghunathan1996: Raghunathan, Rosenthal, and Rubin's (1996) modification of Pearson and Filo
  z = 4.7676, p-value = 0.0000
  Null hypothesis rejected

silver2004: Silver, Hittner, and May's (2004) modification of Dunn and Clark's z (1969) usin
  z = 4.7671, p-value = 0.0000
  Null hypothesis rejected

zou2007: Zou's (2007) confidence interval
  95% confidence interval for r.jk - r.hm: 0.0753 0.1800
  Null hypothesis rejected (Interval does not include 0)
```

We could write this as:

There was a negative weak relation between extraversion (e1) and openness (o1) such that as extraversion increased openness decreased, r = -.10, 95% CI [-.14, -.06], $p < .001$. In contrast, the relation between agreeableness (a1) and conscientiousness(c1) was non-significant, r = .03, 95% CI [-.01, .07], $p = .144$. The extraversion/openness relation was stronger than the agreeableness/conscientiousness relation, $\Delta r = .13$, 95% CI [.07, .18], $p < .001$.

4.3.2.2 Overlapping correlations

We will compare the correlation to between (a1, c1) to the correlation between (a1, e1) with cocor. In this case, because a1 is common to the first correlation (a1, c1) and the second correlation (e1, o1) we refer to this as an overlapping correlation comparison.

We obtain the (a1, e1) correlation, below, and find: $r = .11$, 95% [.07, .14], $p < .001$.

```
cor.test(bfi$a1, bfi$e1)
```

```
    Pearson's product-moment correlation

data:  bfi$a1 and bfi$e1
t = 5.6102, df = 2759, p-value = 2.221e-08
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.06917483 0.14294144
sample estimates:
      cor
0.1062043
```

We obtain the (a1, c1) correlation, below, and find: $r = .03$, 95% [-.00, .07], $p = .14$.

```
cor.test(bfi$a1, bfi$c1)
```

```
    Pearson's product-moment correlation

data:  bfi$a1 and bfi$c1
t = 1.4617, df = 2762, p-value = 0.1439
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.009490233  0.065018669
sample estimates:
```

```
       cor
0.02780283
```

We use the cocor command to compare the two relations:

```
library(cocor)

cocor( ~ a1 + c1 | a1 + e1, data = as.data.frame(bfi))
```

```
  Results of a comparison of two overlapping correlations based on dependent groups

Comparison between r.jk (a1, c1) = 0.0263 and r.jh (a1, e1) = 0.1047
Difference: r.jk - r.jh = -0.0784
Related correlation: r.kh = -0.0248
Data: as.data.frame(bfi): j = a1, k = c1, h = e1
Group size: n = 2741
Null hypothesis: r.jk is equal to r.jh
Alternative hypothesis: r.jk is not equal to r.jh (two-sided)
Alpha: 0.05

pearson1898: Pearson and Filon's z (1898)
  z = -2.8820, p-value = 0.0040
  Null hypothesis rejected

hotelling1940: Hotelling's t (1940)
  t = -2.8827, df = 2738, p-value = 0.0040
  Null hypothesis rejected

williams1959: Williams' t (1959)
  t = -2.8793, df = 2738, p-value = 0.0040
  Null hypothesis rejected

olkin1967: Olkin's z (1967)
  z = -2.8820, p-value = 0.0040
  Null hypothesis rejected

dunn1969: Dunn and Clark's z (1969)
  z = -2.8775, p-value = 0.0040
  Null hypothesis rejected

hendrickson1970: Hendrickson, Stanley, and Hills' (1970) modification of Williams' t (1959)
```

```
  t = -2.8827, df = 2738, p-value = 0.0040
  Null hypothesis rejected

steiger1980: Steiger's (1980) modification of Dunn and Clark's z (1969) using average correla
  z = -2.8765, p-value = 0.0040
  Null hypothesis rejected

meng1992: Meng, Rosenthal, and Rubin's z (1992)
  z = -2.8754, p-value = 0.0040
  Null hypothesis rejected
  95% confidence interval for r.jk - r.jh: -0.1325 -0.0251
  Null hypothesis rejected (Interval does not include 0)

hittner2003: Hittner, May, and Silver's (2003) modification of Dunn and Clark's z (1969) usi
  z = -2.8765, p-value = 0.0040
  Null hypothesis rejected

zou2007: Zou's (2007) confidence interval
  95% confidence interval for r.jk - r.jh: -0.1317 -0.0250
  Null hypothesis rejected (Interval does not include 0)
```

The cocor command provides a lot of output. We are most interested in the last part of the output corresponding to Zou (2007) which provides the confidence interval for the difference: -.13 to -.03. But we also want to examine the first part of the output which will show us the two original correlations $r_{(a1,c1)} = 0.0276$, and $r_{(a1,e1)} = 0.10472$ As well, the output also shows us the difference between thesee two correlations, Difference: r.jk - r.jh = -0.0784 (with rounding, -.08).

We can write this up as:

Agreeableness and extraversion were weakly related, $r = .11$, 95% [.07, .14], $p < .001$, such that as agreeableness increased so did extraversion. The relation between agreeableness and conscientiousness was non-significant, $r = .03$, 95% [-.00, .07], $p = .14$. The agreeableness/extraversion relation was significantly stronger than the agreeableness/conscientiousness relation, $\Delta r = .08$, 95% CI [.03, .13], $p = .004$.

4.3.3 Between data sets

In this section we look at comparing correlations from two data sets.

4.3.3.1 Create seperate data files for men and women (if needed)

We begin by creating two separate data sets - one for men and one for women:

```
bfi_men   <- bfi %>%
  filter(gender == 1) %>%
  select(-gender)

bfi_women <- bfi %>%
  filter(gender == 2) %>%
  select(-gender)
```

Use glimpse() to check out the subgroups. Note that it also tells you the number of participants in each subgroup.

```
glimpse(bfi_men)
```

```
Rows: 919
Columns: 4
$ a1 <int> 2, 2, 2, 4, 4, 4, 2, 5, 4, 4, 5, 5, 1, 4, 1, 4, 5, 1, 1, 1, 1, 5, 1~
$ c1 <int> 2, 4, 5, 3, 6, 4, 5, 4, 5, 5, 5, 4, 4, 5, 1, 4, 2, 5, 6, 6, 4, 5, 5~
$ e1 <int> 3, 2, 4, 3, 5, 1, 3, 2, 3, 1, 2, 3, 2, 3, 6, 2, 3, 6, 3, 1, 3, 6, 6~
$ o1 <int> 3, 3, 5, 3, 6, 5, 4, 5, 5, 6, 5, 6, 6, 6, 6, 4, 4, 5, 5, 6, 4, 3, 6~
```

```
glimpse(bfi_women)
```

```
Rows: 1,881
Columns: 4
$ a1 <int> 2, 5, 4, 6, 2, 5, 4, 4, 4, 1, 2, 1, 2, 2, 2, 4, 1, 2, 2, 1, 1, 5, 2~
$ c1 <int> 5, 4, 4, 6, 6, 5, 4, 5, 1, 5, 3, 5, 6, 4, 5, 5, 5, 5, 4, 6, 4, 5, 5~
$ e1 <int> 1, 2, 5, 2, 2, 3, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 3, 4, 6, 2~
$ o1 <int> 4, 4, 3, 4, 5, 4, 5, 4, 4, 5, 6, 6, 5, 6, 2, 4, 4, 5, 5, 5, 4, 3, 5~
```

4.3.3.2 Check out the subgroup correlations

For **men**, we can obtain the correlation between a1/e1 with the code below. From this we determine, $r = .14$, 95% [.07, .20], $p < .001$.

```
cor.test(bfi_men$a1, bfi_men$e1)
```

```
    Pearson's product-moment correlation

data:  bfi_men$a1 and bfi_men$e1
t = 4.2256, df = 910, p-value = 2.623e-05
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.07447871 0.20182343
sample estimates:
      cor
0.1387245
```

For **women**, we can obtain the correlation between a1/e1 with the code below. From this we determine, $r = .06$, 95% [.02, .11], $p = .007$.

```
cor.test(bfi_women$a1, bfi_women$e1)
```

```
    Pearson's product-moment correlation

data:  bfi_women$a1 and bfi_women$e1
t = 2.6809, df = 1847, p-value = 0.007408
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.01672024 0.10753950
sample estimates:
       cor
0.06225875
```

4.3.3.3 Comparison r(a1, e1)

We run the R commands below to compare the correlations for men and women.

```
library(cocor)
bfi_men_dataframe   <- as.data.frame(bfi_men)
bfi_women_dataframe <- as.data.frame(bfi_women)

cocor( ~ a1 + e1 | a1 + e1,
       data = list(bfi_men_dataframe, bfi_women_dataframe))
```

```
  Results of a comparison of two correlations based on independent groups

Comparison between r1.jk (a1, e1) = 0.1387 and r2.hm (a1, e1) = 0.0623
Difference: r1.jk - r2.hm = 0.0765
Data: list(bfi_men_dataframe, bfi_women_dataframe): j = a1, k = e1, h = a1, m = e1
Group sizes: n1 = 912, n2 = 1849
Null hypothesis: r1.jk is equal to r2.hm
Alternative hypothesis: r1.jk is not equal to r2.hm (two-sided)
Alpha: 0.05

fisher1925: Fisher's z (1925)
  z = 1.9074, p-value = 0.0565
  Null hypothesis retained

zou2007: Zou's (2007) confidence interval
  95% confidence interval for r1.jk - r2.hm: -0.0021 0.1543
  Null hypothesis retained (Interval includes 0)
```

The output reveals the correlation for men, $r_{(a1,e1)} = 0.1387$, .14 rounded, and women $r_{(a1,e1)}=$ 0.0623, .06 rounded, in the output. We also see the comparison: Difference: r1.jk - r2.hm = 0.0765. Finally, we see the zou2007 95% confidence interval: -.00 to .15.

We write this up as:

For men, there was a positive relation between agreeableness (a1) and extraversion (e1), $r =$ .14, 95% [.07, .20], $p < .001$, such that as agreeableness increased so did extraversion. Likewise, for women, there was a similar positive relation between agreeableness (a1) and extraversion (e1), $r = .06$, 95% [.02, .11], $p = .007$. We did not find a significant difference in the strength of these relations, $\Delta r = .08$, 95% CI [-.00, .15], $p = .057$.

5 Multiple regression

The following CRAN packages must be installed:

Required CRAN Packages
tidyverse
apaTables
janitor
remotes
skimr

The following GitHub packages must be installed:

Required GitHub Packages
dstanley4/fastInteraction

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

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

Required Data
data_mr_ex.csv

5.1 Overview

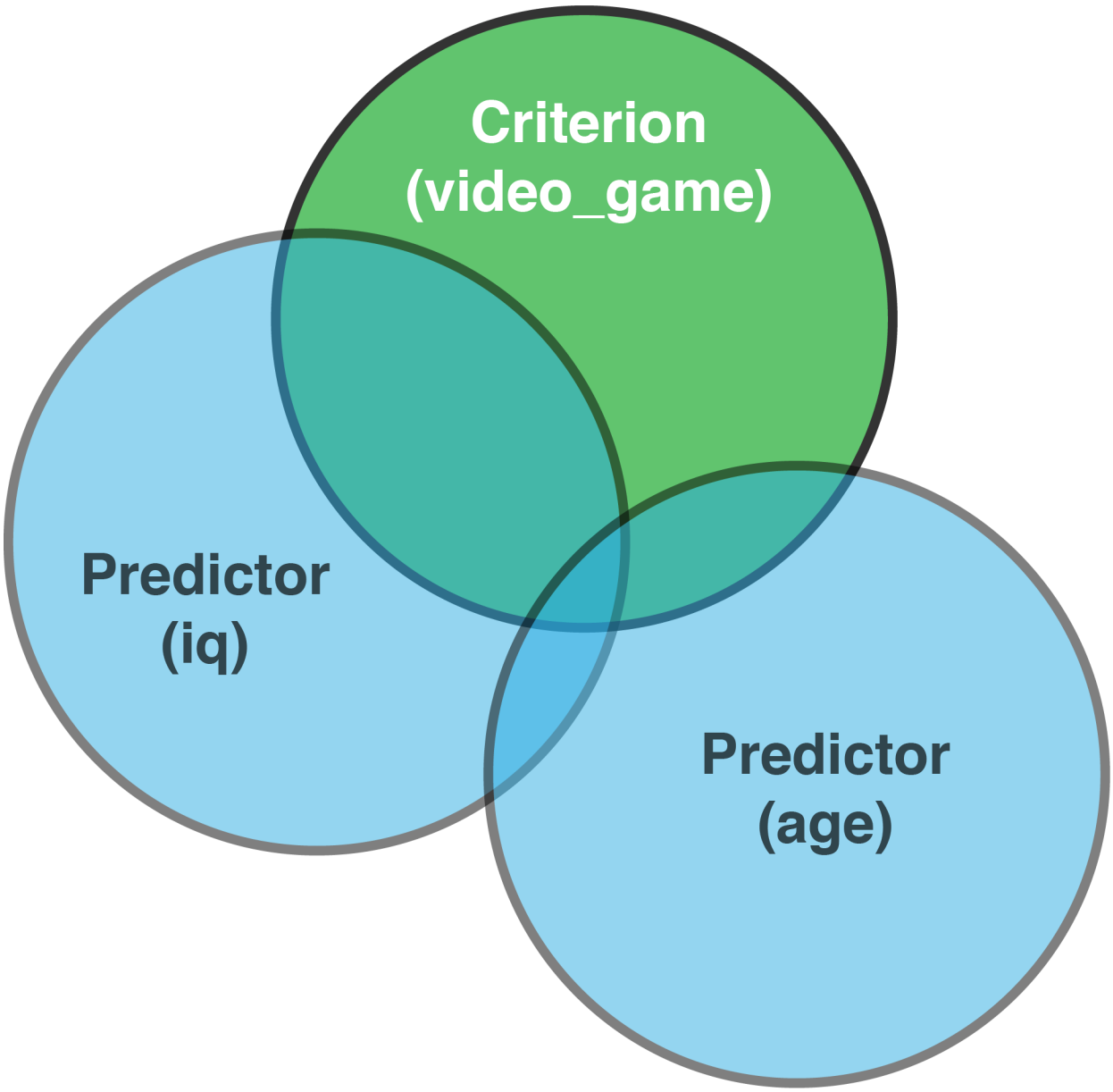
Multiple regression is a means relating multiple predictor variables to a single criterion variable. We can determine the amount of variance in the criterion accounted for by the set of predictors (R^2), a single predictor on it's own (r^2), or the extent to which a single predictor accounts for unique variance account for in the criterion that is not accounted for by any of the other predictors (sr^2).

5.2 Example

In this chapter, we examine the extent to video game scores are predicted by age and IQ for people who live in the City of Guelph. As before, we treat citizens of Guelph as our population – though in this chapter we move straight to the sample data (without showing the population-level data).

We are interested in using both IQ and age to predict video game scores so we refer to these variables as **predictor** variables. The value being predicted is video game score and we refer to that variable as the **criterion** (i.e., the dependent variable).

You can think of the multiple regression problem using a venn diagram:



Multiple Regression Venn Diagram

5.3 Load the data

We use a data set called "data_mr_ex.csv" in this example. The data can be loaded with the command:

```
library(tidyverse)
library(janitor)

my_data <- read_csv("data_mr_ex.csv")

# ensure column names match desired naming convention
my_data <- my_data %>%
  clean_names()
```

We see the structure of the data with the glimpse() command:

```
glimpse(my_data)
```

```
Rows: 200
Columns: 3
$ video_game <dbl> 122.02, 108.67, 130.44, 123.38, 121.49, 125.67, 122.88, 109~
$ iq         <dbl> 107.6, 100.9, 89.0, 90.3, 97.1, 108.1, 120.4, 90.5, 118.9, ~
$ age        <dbl> 41.1, 55.1, 43.9, 46.7, 42.1, 41.2, 41.7, 48.3, 48.2, 48.1,~
```

5.4 Bivariate relations

When you conduct multiple regression analyses you should always report a correlation matrix with your predictors and criterion.

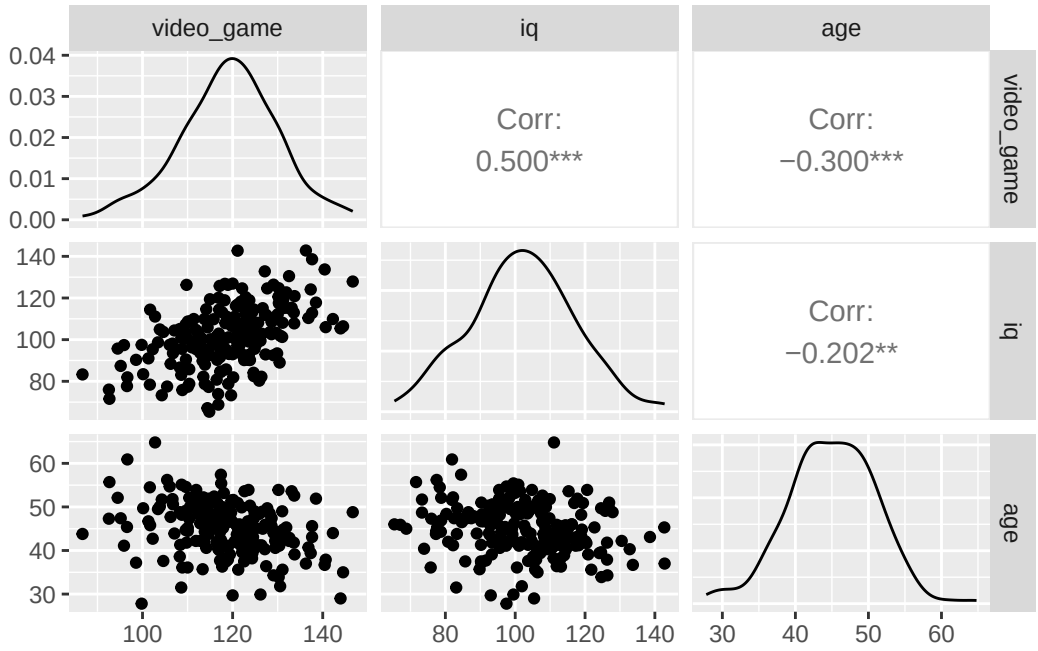
```
library(apaTables)

apa.cor.table(my_data)
```

```
Table 0

Descriptive Statistics and Correlations
```

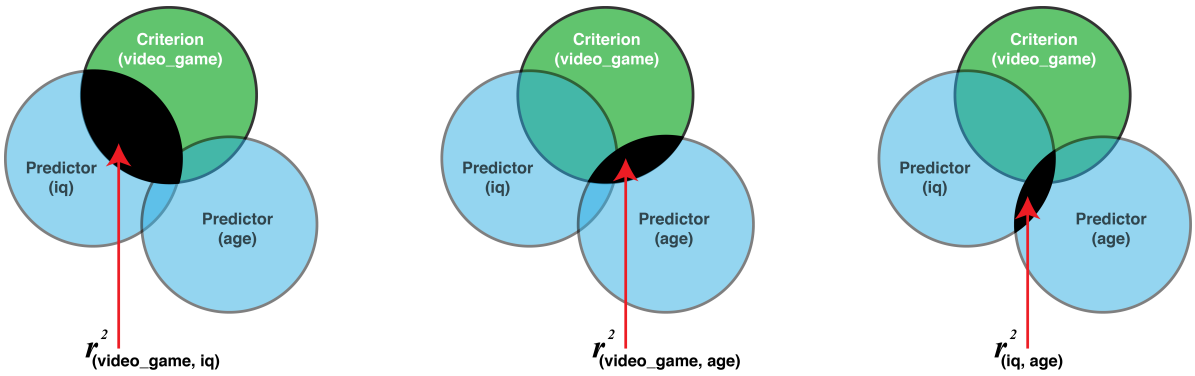
```
  Variable      N   M      SD    1            2
  1. video_game 200 119.03 10.75

  2. iq         200 102.00 15.00 .50**
                                 [.39, .60]
                                 p < .001

  3. age        200 45.00  6.00  -.30**       -.20**
                                 [-.42, -.17] [-.33, -.07]
                                 p < .001     p = .004

Note. N = number of cases. M = mean. SD = standard deviation.
Values in square brackets indicate the 95% confidence interval.
 * indicates p < .05. ** indicates p < .01.
```

You should always check for curvilinear relations when reporting correlations via a graph. We quickly create a graph for doing so with the code below. In this case we don't see any curvilinear relations.

```
library(GGally)

ggpairs(my_data)
```



You can think of each bivariate correlation as the shaded areas on the figure below. In this type of figure the degree of overlap between two circles is determined by the correlation squared (i.r., r^2). The value you obtain for r^2 indicates the proportion of the criterion that is covered by the predictor. For example, if $r = .50$ then $r^2 = .25$ which indicates 25% of the criterion should be covered by the circle representing that predictor.



5.5 Single best predictor

What is the best predictor of video_game? Many researchers incorrectly believe you need multiple regression to answer this question - you do not. To determine the single best predictor in a set of predictors just look at the correlation matrix above - no need for regression

(or beta-weights). The strongest correlation is the best predictor. In our current example, video_game_score is predicted by iq (r = .50) and age (r = -.30). The best predictor of video_game_score is iq because it has the strongest correlation (using absolute values .50 is larger than .30).

5.6 Multiple regression

Multiple regression is frequently used to ask two questions:

- How well we can predict the criterion using a set of predictors (see R^2)?
- What is the unique contribution of a single variable in a set or predictors? In other words, how much does one variable predict the criterion above and beyond another variable? For example, does study time predict exam grades above and beyond iq? Or phrased differently: Does study time predict unique variance in exam grades that is not accounted for by iq? (see sr^2).

We want to use age and IQ to predict video game score (Y). More specifically, we want to combine age and IQ to create a new variable ($\hat{Y}$) that correlates as highly as possible with video game score. We do with the code below:

```
lm_object <- lm(video_game ~ age + iq,
                    data = my_data)
```

Now look at the brief output:

```
print(lm_object)
```

```

Call:
lm(formula = video_game ~ age + iq, data = my_data)

Coefficients:
(Intercept)          age           iq
   102.2333      -0.3712       0.3285
```

This output shows you how we can combine age, IQ, and a constant to create ($\hat{Y}$).

More specifically: $\hat{Y} = 102.233 - 0.0371(age) + 0.328(iq)$

The slopes for age and iq are -0.0371 and 0.328, respectively. These are also referred to as the b-weights or unstandardized regression coefficients. The computer picks these weight so that

predicted video game scores ($\hat{Y}$) will corresponds as closely as possible to actual video game scores.

A quick way to get more comprehensive regression output is to use the *apa.reg.table* function in the apaTables package.

```
library(apaTables)

apa.reg.table(lm_object,
              filename = "table_regression.doc")
```

Table 1

Regression results using video_game as the criterion

Predictor	*b*	*b* 95% CI [LL, UL]	*beta*	*beta* 95% CI [LL, UL]	sr^2	sr^2 95% CI [LL, UL]	*r*	Fit
(Intercept)	102.23**	[87.76, 116.71]						
age	-0.37**	[-0.59, -0.15]	-0.21	[-0.33, -0.09]	.04	[-.01, .09]	-.30**	
iq	0.33**	[0.24, 0.42]	0.46	[0.34, 0.58]	.20	[.11, .30]	.50**	
								R^2 = .291** 95% CI[.19,.38]

Note. A significant *b*-weight indicates the beta-weight and semi-partial correlation are also significant. *b* represents unstandardized regression weights. *beta* indicates the standardized regression weights. sr^2 represents the semi-partial correlation squared. *r* represents the zero-order correlation. *LL* and *UL* indicate the lower and upper limits of a confidence interval, respectively.
* indicates $p < .05$. ** indicates $p < .01$.

5.7 *b*-weights

An inspection of the *b* column in the above table reveals the *b*-weights we previously discussed. The *b*-weights are also known as the slopes or the unstandardized regression weights. The *b*-weights are used to create predicted/estimated video game score via the regression equation:

$$\hat{Y} = 102.233 + 0.3285(Z) - 0.3712(X)$$
$$\widehat{\text{video game}} = 102.233 + 0.3285(iq) - 0.3712(age)$$

You can think of the regression equation as a receipe for making $\hat{Y}$. The variables in the regression (e.g., age and iq) are the ingredients. The *b*-weights (e.g., 0.3285 and 00.3712) are the amount of each ingredient you need to make $\hat{Y}$.

As previously noted, the computer picks the *b*-weights using a process that ensures the predicted video game scores ($\hat{Y}$) corresponds as closely as possible to actual video game scores (Y). Consider the regression calculation for Person 1, below, who is 41.7 years old and has an IQ of 107.6.

$$\begin{aligned}
\hat{Y} &= 102.233 + 0.3285(Z) - 0.3712(X) \\
\widehat{\text{video game}} &= 102.233 + 0.3285(iq) - 0.3712(age) \\
&= 102.233 + 0.3285(107.6) - 0.3712(41.1) \\
&= 122.3
\end{aligned}$$

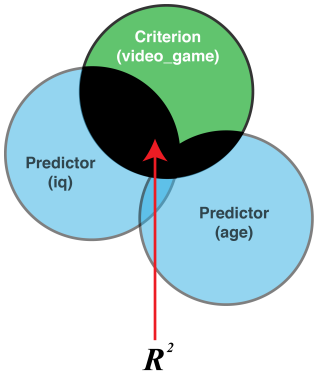
The above calculation reveals an estimated video game score for Person 1 of 122.3 (i.e., $\hat{Y} = 122.3$) – which differs only slightly from their actual video game score of 122 (i.e.. $Y = 122$). You can see the similarity between actual video game scores (Y) and predicted video game scores ($\hat{Y}$) for the first several participants in the table below.

	X	Z	Y	$\hat{Y}$
	age	iq	video_game	video_game_hat
Person 1	41.1	107.6	122.0	122.3
Person 2	55.1	100.9	108.7	114.9
Person 3	43.9	89	130.4	115.2
Person 4	46.7	90.3	123.4	114.6
Person 5	42.1	97.1	121.5	118.5
Person 6	41.2	108.1	125.7	122.4
Person 7	41.7	120.4	122.9	126.3
Person 8	48.3	90.5	109.6	114.0
Person 9	48.2	118.9	118.2	123.4
Person 10	48.1	107.9	118.4	119.8
Person 11	40.6	114.6	114.2	124.8
Person 12	51.9	117.8	138.5	121.7
Person 13	47.3	85.8	110.4	112.9
Person 14	43.8	77.3	114.7	111.4
Person 15	49.5	98.7	112.2	116.3
Person 16	43.9	104.4	107.1	120.2
Person 17	45.2	96.6	117.9	117.2
⋮	⋮	⋮	⋮	⋮

You can interpret the b-weights as indicating how much the criterion changes when a predictor changes – **holding the other predictors constant**. In this context, a 0.3285 b-weight for IQ indicates that for each one-unit increase in IQ video game score will increase 0.3285 points – holding the effect of age constant. Don't forget the b-weight can only be interpreting in context of that specific regression equation. If you replaced the age predictor with, say, height as a predictor, then the b-weight for IQ would change. Context matters when interpreting b-weights.

5.8 R^2

How effective is the set of predictors? We can calculate R^2 to determine the proportion of variance the criterion that is accounted for by the set of predictors. This value is illustrated graphically below:



You will simply obtain R^2 from computer output. But how is R^2 calculated? Understanding how R^2 is calculated can help you to understand how to interpret it. There are two methods for doing so - that produce the same number:

5.8.1 Method 1: Ratio Approach

What does the R^2 mean? It is the proportion variability in criterion scores (Y) accounted for by ($\hat{Y}$). In other words, it is the proportion of the variability in criterion scores that can be accounted for by (a linear combination of) iq and age.

We begin by obtaining predicted video game scores ($\hat{Y}$) using the regression:

$\hat{Y} = 102.233 + 0.3285(iq) - 0.3712(age)$

predicted_video_game_scores $= 102.233 + 0.3285(iq) - 0.3712(age)$

The code below uses the above equation to calculate predicted video game score for each person:

```
predicted_video_game_scores  <- predict(lm_object)

actual_video_game_scores <- my_data$video_game
```

Recall the formula for R^2:

$$R^2 = \frac{\text{Variance of predicted scores}}{\text{Variance of actual scores}}$$

We implement this formula using the code below:

```
var_predicted_video_game_scores <- var(predicted_video_game_scores )
var_actual_video_game_scores   <-  var(actual_video_game_scores)

R2 <- var_predicted_video_game_scores / var_actual_video_game_scores

print(R2)
```

```
[1] 0.2911182
```

Thus, 29% of the variability in video game scores is predicted by the combination of age and IQ.

5.8.2 Method 2: Correlation Approach

An alternative way of thinking about R^2 is as the squared correlation between predicted criterion scores and actual criterion scores:

$$\begin{aligned} R^2 &= r^2_{\hat{Y},Y} \\ &= r^2_{(predicted_{v}ideo_{g}ame_{s}cores, actual_{v}ideo_{g}ame_{s}cores)} \end{aligned}$$

We implement this formula with the code below and obtain the same value:

```
R <- cor(predicted_video_game_scores, actual_video_game_scores)

R2 <- R * R
print(R2)
```

```
[1] 0.2911182
```

5.8.3 R^2 in practice

In practice we simply look at the apa.reg.table() output and obtain the R^2 value and 95% CI from this output:

Table 1

Regression results using video_game as the criterion

Predictor	*b*	*b* 95% CI [LL, UL]	*beta*	*beta* 95% CI [LL, UL]	sr^2	sr^2 95% CI [LL, UL]	*r*	Fit
(Intercept)	102.23**	[87.76, 116.71]						
age	-0.37**	[-0.59, -0.15]	-0.21	[-0.33, -0.09]	.04	[-.01, .09]	-.30**	
iq	0.33**	[0.24, 0.42]	0.46	[0.34, 0.58]	.20	[.11, .30]	.50**	R^2 = .291** 95% CI[.19,.38]

Note. A significant *b*-weight indicates the beta-weight and semi-partial correlation are also significant. *b* represents unstandardized regression weights. *beta* indicates the standardized regression weights. sr^2 represents the semi-partial correlation squared. *r* represents the zero-order correlation. *LL* and *UL* indicate the lower and upper limits of a confidence interval, respectively.
* indicates $p < .05$. ** indicates $p < .01$.

From this table we determine: $R^2 = .29$, 95% CI [.19, .38]. to obtain the required p-value we use the summary() command on the previously calculated lm_object:

```
summary(lm_object)
```

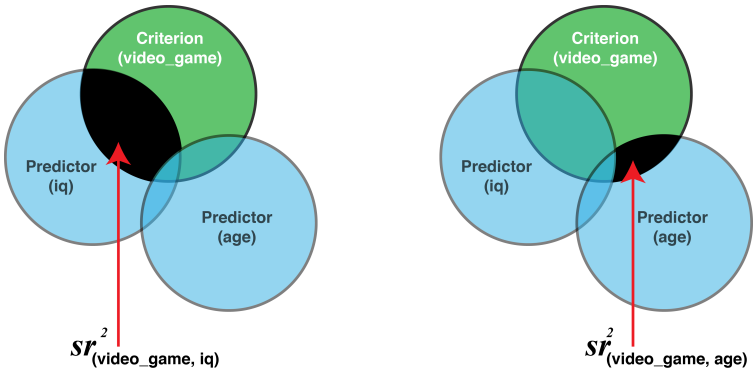
```
Call:
lm(formula = video_game ~ age + iq, data = my_data)

Residuals:
     Min       1Q   Median       3Q      Max
-26.6583  -5.4797   0.6023   6.3278  20.5394

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 102.23331    7.33860  13.931  < 2e-16 ***
age          -0.37115    0.10983  -3.379 0.000876 ***
iq            0.32846    0.04391   7.480  2.4e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.101 on 197 degrees of freedom
Multiple R-squared:  0.2911,    Adjusted R-squared:  0.2839
F-statistic: 40.45 on 2 and 197 DF,  p-value: 1.912e-15
```

At the bottom of this output we see that the p-value is 1.912e-15 or 0.000000000000001912. Consequently, we report the R^2 value as: $R^2 = .29$, 95% CI [.19, .38], $p < .001$. Thus, 29% of the variability in video game scores is predicted by the combination of age and IQ.

5.9 Semi-partial (sr)

A semi-partial correlation is represented by the symbol sr and correspondingly a squared semi-partial correlation is represented by the symbol sr^2. What is a squared semi-partial correlation and why is it useful?

Semi-partial correlations are a way of determining the unique contribution of a variable to predicting the criterion (in the context of the other predictors). The semi-partial correlation is the correlation of one predictor (with all the other predictors removed) with the criterion. The semi-partial correlation squared is the amount R^2 would drop by if that variable was removed from the regression. It is the percentage of variability in criterion scores that is uniquely accounted for by a predictor. This is illustrated in the venn diagram below:



5.9.1 sr^2 in theory

In the text below we go "inside the black box" to show you how semi-partial correlations are computed. In practice, they are just displayed in R output - but understanding the text below where we calculate them "old school" will help with you interpret sr^2.

Overall, squared semi-partial correlations provide an index of how much that predictor contributes to the overall R^2 (with the effect of the other predictors removed). We calculate sr^2 for IQ (removing the effect of age) to demonstrate this fact. We do this with a regression equation in which **we make IQ the criterion** (Y). Then we predict IQ with age. This produces my_iq_regression which has inside of it a predictor version of IQ, $\widehat{Y_{iq}}$, which in this case represents a best guess of IQ based on age.

```
my_iq_regression <- lm(iq ~ age, data = my_data)

print(my_iq_regression)
```

```
Call:
lm(formula = iq ~ age, data = my_data)

Coefficients:
(Intercept)          age
   124.7626      -0.5059
```

. Thus, we find: $\widehat{Y_{iq}} = 124.763 - 0.506(age)$

Consequently, when you see $\widehat{Y_{iq}}$ recognize that it is really just an estimate of IQ **created entirely from age**. In contrast, Y_{iq} is the actual IQ score we obtained from participants.

We want IQ with the effect of age removed. Therefore, we want IQ (i.e., Y_{iq}) with the effect of age (i.e.,$\widehat{Y_{iq}}$) removed.

Thus we want: residual = Y_{iq} - $\widehat{Y_{iq}}$

or another way of thinking of it is:

iq_without_age = Y_{iq} - $\widehat{Y_{iq}}$

iq_without_age = iq - (124.763 - 0.506(age))

We do this below:

```
iq_without_age <- resid(my_iq_regression)
```

Then we correlate IQ without age (i.e., iq_without_age) with video game scores (i.e., video_game). This tells us how IQ correlates with video game scores when the effects of age have been removed from IQ; that is, the semi-partial correlation (i.e., sr). Once again refer to the venn diagram above illustrating sr^2.

```
# apa.reg.table does this for you - this is for learning/illustration only.

sr  <- cor(iq_without_age, my_data$video_game)
sr2 <- sr * sr
```

5.9.2 sr^2 in practice

The sr^2 values with confidence intervals are reported in apa.reg.table() output:

Table 1

Regression results using video_game as the criterion

Predictor	*b*	*b* 95% CI [LL, UL]	*beta*	*beta* 95% CI [LL, UL]	sr^2	sr^2 95% CI [LL, UL]	*r*	Fit
(Intercept)	102.23**	[87.76, 116.71]						
age	-0.37**	[-0.59, -0.15]	-0.21	[-0.33, -0.09]	.04	[-.01, .09]	-.30**	
iq	0.33**	[0.24, 0.42]	0.46	[0.34, 0.58]	.20	[.11, .30]	.50**	
								R^2 = .291** 95% CI[.19,.38]

Note. A significant *b*-weight indicates the beta-weight and semi-partial correlation are also significant. *b* represents unstandardized regression weights. *beta* indicates the standardized regression weights. sr^2 represents the semi-partial correlation squared. *r* represents the zero-order correlation. *LL* and *UL* indicate the lower and upper limits of a confidence interval, respectively.
* indicates $p < .05$. ** indicates $p < .01$.

From this table we determine:

- age: $sr^2 = .04$, 95% CI [-.01, .09]
- iq: $sr^2 = .20$, 95% CI [.11, .30]

However, to obtain the *p*-value for each sr^2 value we need to use the summary() command:

```
summary(lm_object)
```

```
Call:
lm(formula = video_game ~ age + iq, data = my_data)

Residuals:
     Min       1Q   Median       3Q      Max
-26.6583  -5.4797   0.6023   6.3278  20.5394

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 102.23331    7.33860  13.931  < 2e-16 ***
age          -0.37115    0.10983  -3.379 0.000876 ***
iq            0.32846    0.04391   7.480  2.4e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.101 on 197 degrees of freedom
```

```
Multiple R-squared:  0.2911,    Adjusted R-squared:  0.2839
F-statistic: 40.45 on 2 and 197 DF,  p-value: 1.912e-15
```

The p-value for the b-weight (i.e. Estimate) is the p-value for sr^2, Therefore, we simply look in the Pr(>|t|) column to obtain the required p-value. Adding this value to our reporting, we find:

- age: $sr^2 = .04$, 95% CI [-.01, .09], $p < .001$
- iq: $sr^2 = .20$, 95% CI [.11, .30], $p < .001$

If a predictor is significant, this indicates that the predictor contributes unique variance to $\widehat{Y_{videogame}}$ that can not be contributed by any of the other predictors. The amount of unique variance contributed by a predictor is indicated by sr^2 (semi-partial correlation squared).

5.9.3 Blocks regression

Some researchers are unfamiliar with semi-partial correlations and prefer to think in term of how the R^2 value changes over two different regression. This approach is just an indirect way of calculating sr^2.

Consider the example below where the researcher conducts the first regression, block 1, in which age is the predictor. Then he conducts a second regression, block 2, in which both age and iq are the predictors.

```
# apa.reg.table does this for you - this is for learning/illustration only.

block1 <- lm(video_game ~ age,
             data = my_data)

block2 <- lm(video_game ~ age + iq,
             data = my_data)
```

The goal is to examine the R^2 when only age is the predictor and see how much it increases when you have both age and iq as predictors. The resulting difference, ΔR^2 tells you how much iq predicted video game score beyond age alone. Examine the output below.

```
apa.reg.table(block1, block2,
              filename = "table_mr_blocks.doc")
```

Table 2

Regression results using video_game as the criterion

Predictor	*b*	*b* 95% CI [LL, UL]	*beta*	*beta* 95% CI [LL, UL]	sr^2	sr^2 95% CI [LL, UL]	*r*	Fit	Difference
(Intercept)	143.21**	[132.33, 154.10]							
age	-0.54**	[-0.78, -0.30]	-0.30	[-0.43, -0.17]	.09	[.03, .17]	-.30**		
								R^2 = .090** 95% CI[.03,.17]	
(Intercept)	102.23**	[87.76, 116.71]							
age	-0.37**	[-0.59, -0.15]	-0.21	[-0.33, -0.09]	.04	[-.01, .09]	-.30**		
iq	0.33**	[0.24, 0.42]	0.46	[0.34, 0.58]	.20	[.11, .30]	.50**		
								R^2 = .291** 95% CI[.19,.38]	ΔR^2 = .201** 95% CI[.11, .30]

Note. A significant *b*-weight indicates the beta-weight and semi-partial correlation are also significant. *b* represents unstandardized regression weights. *beta* indicates the standardized regression weights. sr^2 represents the semi-partial correlation squared. *r* represents the zero-order correlation. *LL* and *UL* indicate the lower and upper limits of a confidence interval, respectively.
* indicates $p < .05$. ** indicates $p < .01$.

This table illustrates that for the first regression when just age was a predictor that $R^2 = .09$. When both age and iq were predictors $R^2 = .29$. That indicates that R^2 increased by .20 when we added iq as a predictor. Thus, $\Delta R^2 = .20$. This is the amount R^2 increased by due to adding iq as predictor. Conceptually, and mathematically, this is identical to the sr^2 value for iq. Indeed, if you look at this output in detail you see that for iq $sr^2 = .20$.

5.10 Beta-weights

Beta-weights are often referred to as *standardized regression weights*. This is a poor description that makes beta weights hard to understand. A better description of *beta-weights* is the regression weights for standardized variables; that is variables with a mean of 0 and a standard deviation of 1.0.

5.10.1 In practice

Recall we ran a regression with the command below. This command used the original/raw form of the variables.

```
lm_object <- lm(video_game ~ age + iq,
                    data = my_data)
```

From lm_object created we used apa.reg.table() to obtain this output:

Table 1

Regression results using video_game as the criterion

Predictor	*b*	*b* 95% CI [LL, UL]	*beta*	*beta* 95% CI [LL, UL]	sr^2	sr^2 95% CI [LL, UL]	*r*	Fit
(Intercept)	102.23**	[87.76, 116.71]						
age	-0.37**	[-0.59, -0.15]	-0.21	[-0.33, -0.09]	.04	[-.01, .09]	-.30**	
iq	0.33**	[0.24, 0.42]	0.46	[0.34, 0.58]	.20	[.11, .30]	.50**	R^2 = .291** 95% CI[.19,.38]

Note. A significant *b*-weight indicates the beta-weight and semi-partial correlation are also significant. *b* represents unstandardized regression weights. *beta* indicates the standardized regression weights. sr^2 represents the semi-partial correlation squared. *r* represents the zero-order correlation. *LL* and *UL* indicate the lower and upper limits of a confidence interval, respectively.
* indicates $p < .05$. ** indicates $p < .01$.

Notice the beta column in this output. It reports beta weights of -.21 and .46 for age and iq, respectively. Where did these values come from?

To answer this question, we need to start with the lm_object. The apa.reg.table() command just formats the information contained in the lm_object. We can see an unformatted version of this information with the summary() command:

```
summary(lm_object)
```

```
Call:
lm(formula = video_game ~ age + iq, data = my_data)

Residuals:
     Min       1Q   Median       3Q      Max
-26.6583  -5.4797   0.6023   6.3278  20.5394

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 102.23331    7.33860  13.931  < 2e-16 ***
age          -0.37115    0.10983  -3.379 0.000876 ***
iq            0.32846    0.04391   7.480  2.4e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.101 on 197 degrees of freedom
Multiple R-squared:  0.2911,    Adjusted R-squared:  0.2839
F-statistic: 40.45 on 2 and 197 DF,  p-value: 1.912e-15
```

Notice that the values in the Estimates column correspond to the b-weights column in the apa.reg.table() output. We will use this fact to create beta-weights.

5.10.2 Old school

To obtain beta-weights there are two steps. First, we create standardized score versions of each column. Second, we run a normal regression using those columns.

A set of standardized scores have a mean of 0 and a standard deviation of 1.0. To create the standardized score versions of each column in the regression we use the z-score formula:

$$\text{standardized scores} = z_X = \frac{X - \bar{X}}{\sigma_X}$$

Consider the age column. We can calculate the mean for this column, mean(age), and the standard deviation for this column, sd(age). Then for every value in the age column we subtract the column mean and then divide by the column standard deviation. We do so with the calculation: (age-mean(age))/sd(age). The code below creates standardized score versions of the iq, age, and video_game columns called z_iq, z_age, and z_video_game, respectively.

```
my_data <- my_data %>%
  mutate(z_iq = (iq-mean(iq))/sd(iq),
         z_age = (age-mean(age))/sd(age),
         z_video_game = (video_game-mean(video_game))/sd(video_game))
```

We can confirm a mean of 0 and a standard deviation of 1.0 for these new columns with the skim command:

```
library(skimr)
skim(my_data)
```

Now we conduct the regression again with standardized variables (i.e., z-score versions).

```
lm_object_zscores <- lm(z_video_game ~ z_iq + z_age,
                        data = my_data)
```

We can obtain the regression weights for analysis using these standardized scores with the summary() command:

```
summary(lm_object_zscores)
```

```
Call:
lm(formula = z_video_game ~ z_iq + z_age, data = my_data)

Residuals:
    Min      1Q  Median      3Q     Max
-2.4788 -0.5095  0.0560  0.5884  1.9098

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.732e-16  5.984e-02   0.000 1.000000
z_iq         4.582e-01  6.125e-02   7.480  2.4e-12 ***
z_age       -2.070e-01  6.125e-02  -3.379 0.000876 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8462 on 197 degrees of freedom
Multiple R-squared:  0.2911,    Adjusted R-squared:  0.2839
F-statistic: 40.45 on 2 and 197 DF,  p-value: 1.912e-15
```

Notice that the estimates above are 4.582e-01 -2.070e-01 which are, in decimal form, .46 and -.21, respectively. These are the beta-weights from the apa.reg.table on the previous page. Thus, when you conduct a normal regression **but used standardized variables in it** you obtain beta-weights.

Interpretation. The typically reported **b-weights** describe how a 1 unit change in IQ influences video_game points. Similarly, **beta-weights**, describe how a 1 unit change in z_iq influences z_video_game. Keep in mind, however, that 1 unit of z_iq (and z_video_game) is 1 standard deviation. As a result, a beta-weight indicates how much the criterion scores will change in SD units when a predictor increases by 1 SD – holding the effect of the other predictors constant. Similar to b-weights, beta-weights can only be interpreted in the context of the other variables in the equation.

Hopefully, this description has made it clear that although beta-weights are often referred to as *standardized regression weights*; it would be more accurate to describe them as the **weights for standardized variables**.

5.11 Graphing

Let's take a minute to consider the nature of the data we have so far - examine the first few rows of the data below (that includes the predicted value for each person).

	X	Z	Y	$\hat{Y}$
	age	iq	video_game	video_game_hat
Person 1	41.1	107.6	122.0	122.3
Person 2	55.1	100.9	108.7	114.9
Person 3	43.9	89	130.4	115.2
Person 4	46.7	90.3	123.4	114.6
Person 5	42.1	97.1	121.5	118.5
Person 6	41.2	108.1	125.7	122.4
Person 7	41.7	120.4	122.9	126.3
Person 8	48.3	90.5	109.6	114.0
Person 9	48.2	118.9	118.2	123.4
Person 10	48.1	107.9	118.4	119.8
Person 11	40.6	114.6	114.2	124.8
Person 12	51.9	117.8	138.5	121.7
Person 13	47.3	85.8	110.4	112.9
Person 14	43.8	77.3	114.7	111.4
Person 15	49.5	98.7	112.2	116.3
Person 16	43.9	104.4	107.1	120.2
Person 17	45.2	96.6	117.9	117.2
⋮	⋮	⋮	⋮	⋮

You can see that each person has three measured variables associated with them: video_game_score, iq, and age. These columns are in blue to indicate the fact they are measured variables. Because we have three measured variables we can't create a typical 2D scatter plot. That type of plot only work when there is one predictor and one criterion. Now we have two predictors and one criterion. Consequently, we need make a 3D scatter plot.

Correspondingly, because we have two predictors, we can't obtain a regression line (i.e., best-fit line). A regression line is only possible when there is one predictor. Now we have two predictors. Consequently, a regression surface is required to show the predicted values for combinations of age and iq. In this case, when there are two predictor variables, the regression surface is a plane.

Let's create the 3D scatter plot with a regression surface. You recall we previously created the lm_object when we ran our regression:

```
lm_object <- lm(video_game ~ age + iq,
                    data = my_data)
```

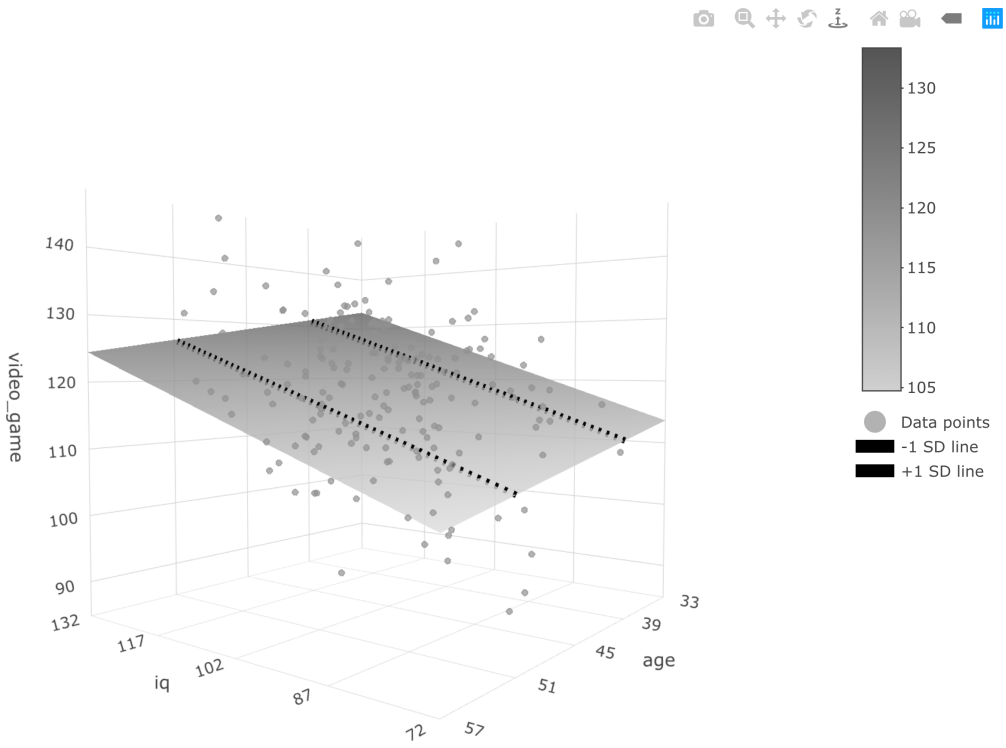
The lm_object has the three data point for each person (age, iq, video_game) embedded inside of it. So we can use the lm_object to create the scatter plot with regression plane. We do so using the fastInteraction package. More specifically, we use the fast.plot command as illustrated below.Note: The fast.plot command uses the argument "moderator" which is not appropriate in our context. When you see moderator in the command below just think of it as another predictor.

```
library(fastInteraction)

surface_plot <- fast.plot(lm_object,
                          criterion = video_game,
                          predictor = iq,
                          moderator = age)
```

Then just type:

```
surface_plot
```

You can see the graph below:



All of the predicted scores fall on the regression surface. You can think of the regression surface as a best-fit plane. In some sense you can think of the plane as a collection of best-fit lines. Indeed, two illustrative best-fit lines are place on this surface. For all the best-fit lines on the plane the slopes are the same - just the intercepts differ. You can see the people relative to the predicted surface/plane by looking at the dots. Each dot represents a person.

Try the interactive version of the graph below. Caveat, sometimes it doesn't appear correctly on the web. If the graph appears incorrectly (e.g., no data points or no surface) try using the Chrome browser. Safari sometimes has problems with this web object. You can rotate the graph to see a better view of the surface. The graph illustrates that predicted video game scores (i.e., the surface) change as a function of both age and iq.

5.12 Control variables

Many people use "control" variables in their multiple regression. Before using this type of analysis I urge you to read:

Wysocki, A. C., Lawson, K. M., & Rhemtulla, M. (2022). Statistical control requires causal justification. Advances in Methods and Practices in Psychological Science, 5(2), 25152459221095823.

6 Sample size for multiple regression

When you conduct a multiple regression there are two ways to think about conducting a sample size analysis.

- Focus on having a sufficient power to the variance accounted for by the set of predictors (i.e. R^2)
- Focus on having a sufficient power to the unique variance account for by a single predictor (i.e. sr^2)

6.1 Sample size using R^2

You want to run a multiple regression study in an existing research area using 3 predictors. A past study, $N = 100$, found an $R^2 = .20$. How many people do you need in your study to have 80% power for the overall regression equation (i.e., R^2).

We need to determine two pieces of information to proceed: 1) the effect size as f^2 instead of R^2 and 2) the degrees of freedom for the predictors. We use these two pieces of information in the *pwr* command.

6.1.1 Determining f^2

We need to represent R^2 as f^2. We do this with the formula:

$$f^2 = \frac{R^2}{1 - R^2}$$

So in R we type:

```
my_f2 <- .20 / (1 - .20)
print(my_f2)
```

```
[1] 0.25
```

Thus, $f^2 = .25$.

6.1.2 Determine degrees of freedom

There are degrees of freedom for the predictors (u) and error (v) in the original study. The degrees of freedom for the predictors is easy to compute:

Degrees of Freedom Predictors: u = number of predictors = 3.

6.1.3 Calculate sample size

With multiple regression we send R the effect size (.25) the degrees of freedom for the predictor (3). The output provide returns the degrees of freedom for the denominator - which requires a bit of work to convert to sample size.

We begin with the R code below:

```
library(pwr)
pwr.f2.test(u = 3,
            f2 = .25,
            power = .90)
```

```
     Multiple regression power calculation

              u = 3
              v = 56.75331
             f2 = 0.25
      sig.level = 0.05
          power = 0.9
```

What does this output mean? It provides us with the degrees of freedom error (i.e., v) for your study (not the original study). We need to use the degrees of freedom error to calculate the N needed.

Degrees of Freedom Error: $v = N - u - 1$ therefore $N = u + v + 1$.

We know $u = 3$ because there were three predictors. Our power analysis revealed we need $v = 57$ in our study.

Therefore $N = u + v + 1 = 3 + 57 + 1$

```
N = 3 + 57 + 1
print(N)
```

```
[1] 61
```

Thus we want an N of 61 in our study.

6.2 Sample size using sr^2

You want to run a multiple regression study in an existing research area using 3 predictors (A, B, and C). A past study, $N = 100$, found an $R^2 = .20$. You are particularly interested in one predictor (predictor C) that you think will account for 2% of the variance in the criterion above and beyond the other two predictors. Said another way you believe .02 of the .20 will be a result of the unique contribution of one predictor. What sample size do you need to ensure you have power of .90 for detecting this increment in variance.

Let's think about this scenario in terms of Blocks to make it a bit clear. You are describing a scenario where if you put predictors A and B are in Block 1 you would obtain an overall $R^2 = .18$. Then when you put Predictor A, B, and C into Block 2 you would obtain an overall $R^2 = .20$, an increment of .02 so sr^2 for predictor C is .02.

We want to conduct a power analysis to determine how many people you need to ensure power of .90 for this incremental prediction effect.

6.2.1 Determine degrees of freedom

We are interested in the incremental prediction of one variable so $u = 1$.

6.2.2 Determine f^2

We know $sr^2 = .02$ and $R^2 = .20$. We can use these to compute the needed f^2.

$$f^2 = \frac{sr^2}{1 - R^2}$$

So we type:

```
my_f2 <- .02 / (1 - .20)

print(my_f2)
```

```
[1] 0.025
```

6.2.3 Calculate power

We can calculate the required degrees of freedom with the command below:

```
library(pwr)
pwr.f2.test(u = 1,
            f2 = 0.025,
            power = .90)
```

```
     Multiple regression power calculation

              u = 1
              v = 420.2268
             f2 = 0.025
      sig.level = 0.05
          power = 0.9
```

Thus, degrees of freedom error (i.e., v) is 421

We then calculate $N = u + v + 1$

In R we type:

```
N = 1 + 421 + 1
print(N)
```

```
[1] 423
```

Thus, you need an N of 423.

6.3 Power obtained sr^2

Imagine your power analysis tells you that you need an N of 423 but you only get an N of 75. What is your power? Simply calculate the degrees of freedom for the predictors and error and run the command again. This time you leave out power. It will be calculated for you. You can do this before you collect your data if you know in advance you will have a low/restricted N - as is the case for many theses.

In this example, $R^2 = .20$ and $sr^2 = .02$ are the expected effect sizes.

Degrees of Freedom Predictors: $u = 1$ as above (incremental prediction of one variable)

In terms of error:

Degrees of Freedom Error: $v = N - u - 1 = 75 - 1 - 1 = 73$.

In terms of effect size:

$f^2 = \frac{sr^2}{1-R^2} = \frac{.02}{1-.20} = .025$.

```
pwr.f2.test(u = 1,
            v = 73,
            f2 = 0.025)
```

```
     Multiple regression power calculation

              u = 1
              v = 73
             f2 = 0.025
      sig.level = 0.05
          power = 0.2718443
```

In this case, power is .27. Thus, there is only a 27% chance of finding the effect if it exists. Given this, would it make sense to spend the time and energy to run the study?

7 Moderated multiple regression

The following CRAN packages must be installed:

Required CRAN Packages
tidyverse
remotes

Data
data_endurance.csv

The following GitHub packages must be installed:

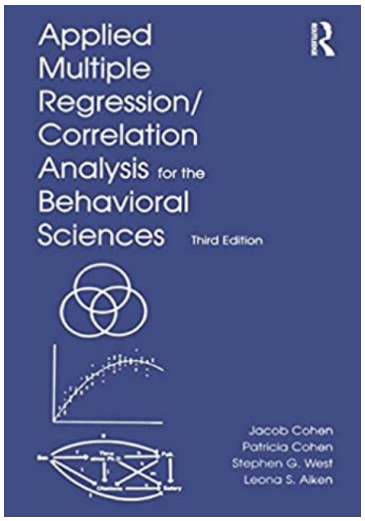
Required GitHub Packages
dstanley4/fastInteraction

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

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

7.1 Overview

In this chapter we present a brief overview of moderated multiple regression. In an ANOVA you can have two variables interact to predict a dependent variable. In this ANOVA scenario, the predictors are the categorical ANOVA variables. When our predictors are continuous variables (e.g., height, weight, etc) they can still interact to predict the dependent variable (i.e., criterion). In this chapter we primarily focus on how to obtain the required information from R to write up a continous variable interaction (also known as a moderated multiple regression). For an understanding of the underlying theory I strongly encourage you to read *Chapter 7: Interactions among continuous variables* in Cohen, Cohen, West, and Aiken (2003):



Cohen, J., Cohen, P., West, S. G., & Aiken, L. S. (2013). Applied multiple regression/correlation analysis for the behavioral sciences. Routledge.

7.2 Scenario

Imagine a scenario where we are interested in predicted endurance from participant age and exercise.

We can load the data:

```
library(tidyverse)

data_endurance <- read_csv("data_endurance.csv")
```

We can see the structure of the data:

```
glimpse(data_endurance)
```

```
Rows: 245
Columns: 4
$ id        <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1~
$ age       <dbl> 60, 40, 29, 47, 48, 42, 55, 43, 39, 51, 54, 52, 53, 68, 57, ~
$ exercise  <dbl> 10, 9, 2, 10, 9, 6, 8, 19, 9, 14, 15, 4, 3, 17, 24, 4, 4, 16~
$ endurance <dbl> 18, 36, 51, 18, 23, 30, 8, 40, 28, 15, 49, 27, 12, 43, 47, 2~
```

7.3 Overview

7.3.1 No interaction

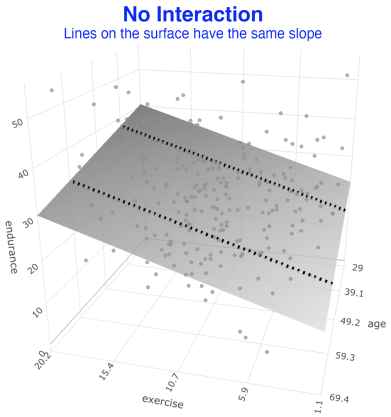
In a typical two variable regression we would attempt to solve this equation:

$$\hat{Y} = b_0 + b_1 age + b_2 exercise$$

We would do so using this R code:

```
lm_no_int <- lm(endurance ~ age + exercise,
                data = data_endurance)
```

The result would be the a regression surface that fits the data as illustrated below. In this graph we did **not** assess whether there was an interaction (i.e., a moderating effect). This is just a standard two predictor regression. We illustrate the extent to which exercise and age predict endurance. You can consider the regression surface a series of best-fit lines. All of the lines on this surface have the same slope. That means in this statistical model the relation between exercise and endurance is not influenced by age – because the slopes of the lines on the surface are the same and do not change with age of participants.



7.3.2 Interaction

We might wonder if the relation between exercise (a predictor) and endurance (the criterion) depends on the age of participants (a predictor). In other words, we might wonder if the relation between exercise and endurance is **moderated** by age. This is conceptually identical to an interaction effect in an ANOVA.

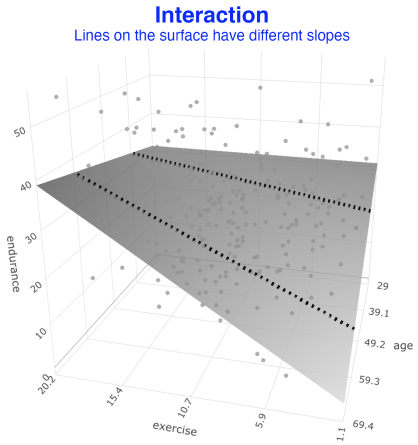
We determine if there is a moderated relation by adding a product term to the regression. This is simply a new data column created by multiplying the age and exercise columns. If the b-weight (i.e., b_3) for for this product column (i.e., age*exercise) is significant - we say there is an interaction or a moderated relation. The regression equation we need to solve is below:

$$\hat{Y} = b_0 + b_1 age + b_2 exercise + b_3 (age)(exercise)$$

We solve this equation using the code:

```
lm_int <- lm(endurance ~ age + exercise + I(age*exercise),
             data = data_endurance)
```

When we examined the output for this regression, we found that that b_3 was significant. This indicates there is a moderated relation. The relation between exercise and endurance does depend on age. We can see this moderated relation in the graph below. As before imagine the surface is composed of a series of best-fit lines. Because there is a moderated relation the slopes of the lines change as we move across this surface. You can see this clearly in the graph below. This illustrate the slope for the exercise – endurance relation changes depending on the age of participants. That is, it illustrates the nature of the moderated relation.

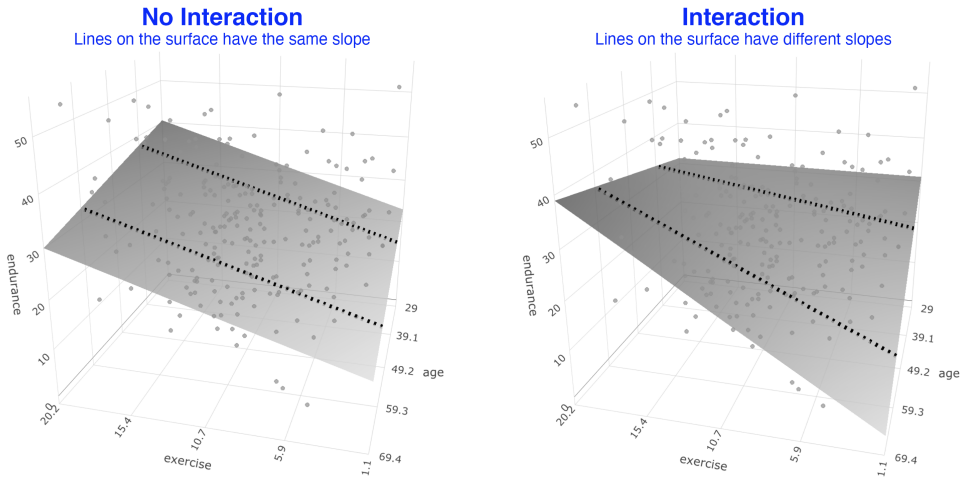


7.3.3 Comparison

We present both graphs below so you can more easily see the difference between them. When you look at the *No Interaction* graph you can see the best-fit surface is composed of a series of straight lines that are parallel (i.e., all have the same slope) but all orientated a particular angle. In contrast, when you look at the *Interaction* graph you can see that it is also composed

of a series of straight lines but the overall surface looks curves because the lines composing the best-fit surface have different slopes.

Interpretation note: Notice that the difference between the *No Interaction* surface and the *Interaction* surface is the largest near the edges of the surface. Consequently, the inclusion of product terms typically makes the largest improvement in fit for the people who score at the extremes of the surface. In contrast, the inclusion of product terms typically makes a minor improvement in fit for the people who are near the middle of the surface (i.e., the majority of people). Keep that in mind as you interpret the increase in fit that results from the inclusion of the product terms.



7.4 fastInteraction

If you want to actually conduct a moderated regression and solve the equation below, there is an easy way to do it.

$$\hat{Y} = b_0 + b_1 age + b_2 exercise + b_3 (age)(exercise)$$

You simply use the fast.int() command in the fastInteraction package does the following:

1. Conducts the regression (i.e., the lm command); including mean centering of predictors if desired (see Cohen, Cohen, West, and Aiken, 2003).
2. Creates the 2D graph
3. Creates the 3D graph
4. Creates two APA style tables to describe the analyses

To run the analysis you use the code below:

7.4.1 Graphing in 3D

You can obtain the 3D graph (which can be rotated) using the code below. The graph may not appear on this webpage if you are using Chrome.

7.4.2 Unformatted 2D graph

You can obtain the 2D graph (cross-section of the 3D surface) with the code below. We need to adjust the graph a bit before it's presentable. But to do so we first need to inspect tables created by the fast.int() command.

7.4.3 Tables

We inspect MS Word file created by the fast.int() command below. The commands save the table in the file "tables_mmr.doc" – as specified. It presents the results of the various regression in tables corresponding to APA style.

Table 1

Moderated regression results using endurance as the criterion, age as the predictor, and exercise as the moderator

Predictor	*b*	*b* 95% *CI* [*LL, UL*]	sr^2	sr^2 95% *CI* [*LL, UL*]	*p*	Fit
(Intercept)	25.89**	[24.61, 27.16]			0.000	
age	-0.26**	[-0.39, -0.14]	.05	[.00, .11]	0.000	
exercise	0.97**	[0.70, 1.24]	.17	[.08, .25]	0.000	
age:exercise	0.05**	[0.02, 0.07]	.04	[-.00, .08]	0.001	
						R2 = .206** 95% CI[.12,.28]

Note. A significant *b*-weight indicates the semi-partial correlation is also significant. *b* represents unstandardized regression weights. sr^2 represents the semi-partial correlation squared. LL and UL indicate the lower and upper limits of a confidence interval, respectively. *p* indicates the *p*-value.
* *p* indicates $p < .05$. ** indicates $p < .01$.

Table 2

Simple slope regression results using endurance as the criterion, age as the predictor, and exercise as the moderator

Moderator	Moderator Value	b_1 *slope*	b_1 *LL*	b_1 *UL*	b_0 *intercept*	b_1 *SE*	*t*	*p*
-1 SD exercise	-4.775	-0.487	-0.669	-0.306	21.244	0.092	-5.289	0.0000
Mean exercise	0.000	-0.262	-0.388	-0.135	25.889	0.064	-4.085	0.0001
+1 SD exercise	4.775	-0.036	-0.214	0.142	30.534	0.090	-0.400	0.6896

Note. b_1 represents the slope. b_0 represents the intercept. SE represents standard error. LL and UL indicate the lower and upper limits of a confidence interval, respectively. *t* represents the *t*-obtained value. *p* represents the *p*-value.

7.4.4 Formatted 2D graph

We inspect the Table output and find the value for -1/+1 SD of the moderator (age) in Table 2. In this case, we find the value is 4.775. So we use this to set the x-axis ticks That way the ticks on the x-axis correspond to standard deviations.

7.4.4.1 Saving

We can save the 2D plot with the following command:

```
ggsave("graph_2D_interaction.png", custom_formatted_ggplot_graph)
```

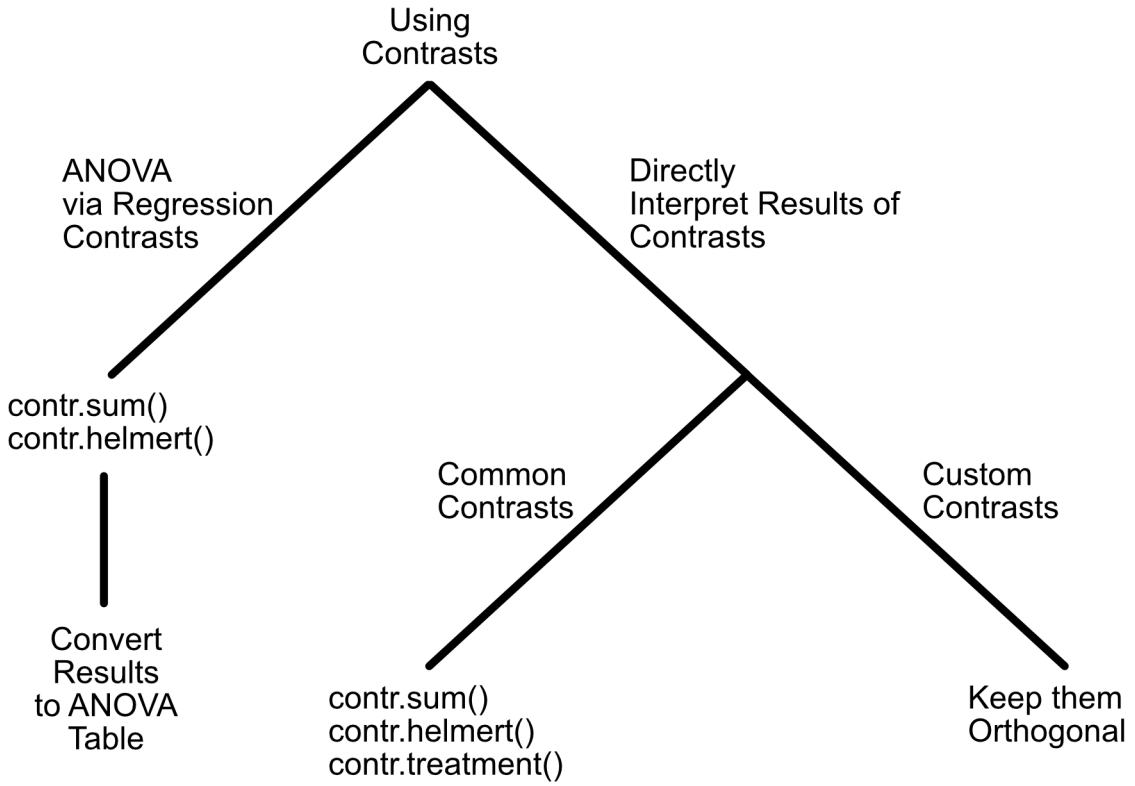
7.5 Power/sample size analysis

If you examine a large number of studies involving moderated multiple regression you will see that the sr^2 (i.e., ΔR^2) for the product term (e.g., age*exercise) typically ranges from .01 to .04 and only rarely falls out of this range - regardless of the variables involved. Consequently, to conduct a sample size analysis for a moderated mulitple regression you need to do three things:

1. Estimate the size of the sr^2 value (i.e., in the .01 to .04 range) for your study.
2. Estimate the overall R^2 value for your study.
3. Plug both of those values into the sample size calculation provided in the multiple regression lecture.

8 One-way ANOVA via Regression

8.1 Using Contrasts



8.2 Treatement/Dummy Coding

Use the Treatment contrast ONLY when you are interested in the contrast itself. DO NOT use if you are interested in typical ANOVA results (main effect, main effect, interaction, etc.).

This approach is the default approach in R unless you specify otherwise. In most cases, this is NOT what you want in Psychology analyses.

Comparisons are to one specific level of the Independent Variables that we call the reference group.

8.2.1 Original Data

```
print(viagra)
```

```
   libido      dose
1       3   placebo
2       2   placebo
3       1   placebo
4       1   placebo
5       4   placebo
6       5  low_dose
7       2  low_dose
8       4  low_dose
9       2  low_dose
10      3  low_dose
11      7 high_dose
12      4 high_dose
13      5 high_dose
14      3 high_dose
15      6 high_dose
```

Note the means for the three groups are:

```
viagra %>% group_by(dose) %>% summarise(group_mean = mean(libido))
```

```
# A tibble: 3 x 2
  dose      group_mean
  <fct>          <dbl>
1 placebo          2.2
2 low_dose         3.2
3 high_dose        5
```

8.2.2 Set Factor with Reference Group

```r
viagra <- viagra %>%
  mutate(dose = as_factor(dose)) %>%
  mutate(dose = relevel(dose, ref = "placebo"))
```

8.2.3 Regression with Treatment Contrast

The computer will always use contrasts when there are categorical variables. So you should set the contrast you want. Here we set the contrast as Treatment (or Dummy) Coding. We use treatment contrasts when we are interested in directly interpreting the regression results.

```r
options(contrasts = c("contr.treatment", "contr.poly"))

lm_viagra <- lm(libido ~ dose + 1, data = viagra)
```

```r
tidy(lm_viagra)
```

term	estimate	std.error	statistic	p.value
(Intercept)	2.2	0.6271629	3.507860	0.0043189
doselow_dose	1.0	0.8869423	1.127469	0.2815839
dosehigh_dose	2.8	0.8869423	3.156913	0.0082681

What is going on here? The single dose column has disappeared. Instead we get b-weights for doselow_dose and dosehigh_dose. How do you interpret that information?

8.2.4 Treatment Contrasts Explained

When we used the treatment contrast we converted the use the rules below:

```r
contr.treatment(3)
```

```
  2 3
1 0 0
2 1 0
3 0 1
```

- The first contrast row (0 0) indicates the mean of group 1 (placebo) is the intercept plus the 0 times first slope and 0 times the second slope.

$$placebo - mean = 2.20 + 0(1.00) + 0(2.80) = 2.20$$

- The second contrast row (1 0) indicates the mean of group 2 (low dose) is the intercept plus the 1 times first slope and 0 times the second slope.

$$low - dose - mean = 2.20 + 1(1.00) + 0(2.80) = 3.20$$

- The third contrast row (0 1) indicates the mean of group 3 (high dose) is the intercept plus the 0 times first slope and 1 times the second slope.

$$low - dose - mean = 2.20 + 0(1.00) + 1(2.80) = 5.00$$

Recall the levels of the dose variable:

```
levels(viagra$dose)
```

```
[1] "placebo"   "low_dose"  "high_dose"
```

We set the order of the levels previously with the fct_relevel() command.

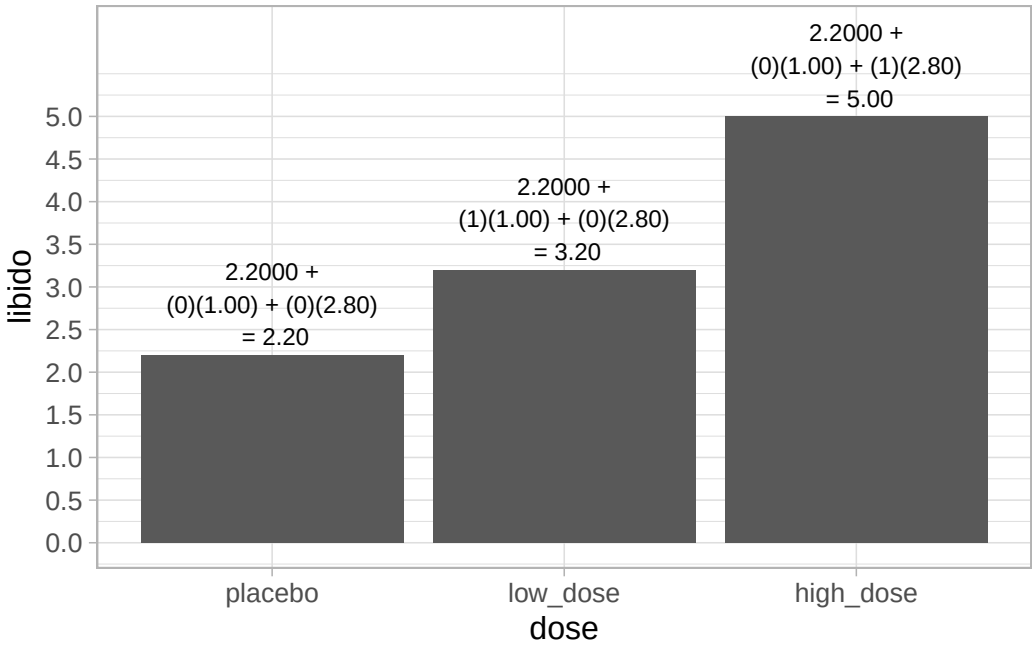
When we ran the regression we effectively use the predictors below

```
print(viagra_dummy_coded)
```

```
# A tibble: 15 x 4
   libido intercept dose_low_dose dose_high_dose
    <int>     <dbl>         <dbl>          <dbl>
 1      3         1             0              0
 2      2         1             0              0
 3      1         1             0              0
 4      1         1             0              0
 5      4         1             0              0
 6      5         1             1              0
 7      2         1             1              0
 8      4         1             1              0
 9      2         1             1              0
10      3         1             1              0
11      7         1             0              1
12      4         1             0              1
13      5         1             0              1
```

```
14       3          1                0                1
15       6          1                0                1
```

Examine the weights in the above table and see how they can be used to recreate the group means.



8.2.5 ANOVA Summary Information

With a one-way ANOVA, it's easy to exact ANOVA information from the Regression output.

```
glance(lm_viagra)
```

r.squared	adj.r.squared	sigma	statistic	p.value	df	logLik	AIC	BIC	deviance	df.residual	nobs
0.4603659	0.3704268	1.402379	5.118644	0.0246943	2	-24.68305	57.36616	60.1983	23.6	12	15

From this output you can see that for this one-way ANOVA, $F(2, 12) = 5.119$, $p = .025$. In a one-way ANOVA the effect size is $\eta^2 = \eta^2_{partial} = R^2 = .46$. Note that in a one-way ANOVA, $\eta^2 = \eta^2_{partial}$ but this is not the case when you move to N-way ANOVA.

```r
summary(lm_viagra)
```

```
Call:
lm(formula = libido ~ dose + 1, data = viagra)

Residuals:
   Min     1Q Median     3Q    Max
  -2.0   -1.2   -0.2    0.9    2.0

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)     2.2000     0.6272   3.508  0.00432 **
doselow_dose    1.0000     0.8869   1.127  0.28158
dosehigh_dose   2.8000     0.8869   3.157  0.00827 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.402 on 12 degrees of freedom
Multiple R-squared:  0.4604,    Adjusted R-squared:  0.3704
F-statistic: 5.119 on 2 and 12 DF,  p-value: 0.02469
```

From this output you can (AGAIN) see that for this one-way ANOVA, $F(2, 12) = 5.119$, $p = .025$. In a one-way ANOVA the effect size is $\eta^2 = \eta^2_{partial} = R^2 = .46$.

Note: A good exam question would be to present a table like this an then ask you the mean for each group. With treatment/dummy coding the regression weight indicate for the reference group the mean of that group. For the other groups, the regression weights indicate the difference from the reference group.

8.3 Sum Contrast / Effect Contrast

We typically use Sum Coding or Effect Coding when we are not interested in the directly interpretting the regression results. Although the results can be directly interpretted this is not commonly done. Instead we use Sum Coding or Effect Coding when we use regression to run an ANOVA. Here we will directly interpret the results, however, to show that it can be done.

With sum coding, the contrasts create b-weight represent comparisons of each group mean to the to the grand mean. The contrasts we use for each group are presented below:

```
contr.sum(3)
```

```
  [,1] [,2]
1    1    0
2    0    1
3   -1   -1
```

We run the regression with the code below. Notice we take care to set the contrast to sum before the regression.

```
# Check levels
levels(viagra$dose)
```

```
[1] "placebo"   "low_dose"  "high_dose"
```

```
# select sum coding
options(contrasts = c("contr.sum", "contr.poly"))

lm_viagra_sum_coded <- lm(libido ~ dose + 1,
                          data = viagra)
```

We see the results are below. We use as.data.frame() just to see all the decimals.

```
tidy(lm_viagra_sum_coded) %>% as.data.frame()
```

```
         term   estimate std.error  statistic      p.value
1 (Intercept)  3.4666667 0.3620927  9.5739760 5.720565e-07
2       dose1 -1.2666667 0.5120764 -2.4735893 2.930022e-02
3       dose2 -0.2666667 0.5120764 -0.5207556 6.120112e-01
```

In the results above the intercept corresponds to the grand mean.

Recall the sum contrast below:

```
contr.sum(3)
```

```
  [,1] [,2]
1    1    0
2    0    1
3   -1   -1
```

- The first contrast row (1 0) indicates the mean of group 1 (placebo) is the intercept plus the 1 times first slope and 0 times the second slope.

$$placebo - mean = 3.4666667 + 1(-1.2666667) + 0(-0.2666667) = 2.20$$

- The second contrast row (0 1) indicates the mean of group 2 (low dose) is the intercept plus the 0 times first slope and 1 times the second slope.

$$low - dose - mean = 3.4666667 + 0(-1.2666667) + 1(-0.2666667) = 3.20$$

- The third contrast row (-1 -1) indicates the mean of group 3 (high dose) is the intercept plus -1 times first slope and -1 times the second slope.

$$low - dose - mean = 3.4666667 + (-1)(-1.2666667) + (-1)(-0.2666667) = 5.00$$

8.3.1 Behind the scenes

What is the grand mean? It's just the mean of the dependent variable column across all conditions.

```
summary_stat = viagra %>%
  summarise(grand_mean = mean(libido))

print(summary_stat)
```

```
  grand_mean
1   3.466667
```

In the above analysis, when we used this code block:

```
options(contrasts = c("contr.sum", "contr.poly"))

lm_viagra_sum_coded <- lm(libido ~ dose + 1,
                          data = viagra)
```

We were doing a regression with the predictors below:

```
print(viagra_sum_coded)
```

```
   libido intercept dose1 dose2
1       3         1     1     0
2       2         1     1     0
3       1         1     1     0
4       1         1     1     0
5       4         1     1     0
6       5         1     0     1
7       2         1     0     1
8       4         1     0     1
9       2         1     0     1
10      3         1     0     1
11      7         1    -1    -1
12      4         1    -1    -1
13      5         1    -1    -1
14      3         1    -1    -1
15      6         1    -1    -1
```

That is, when we specified this:

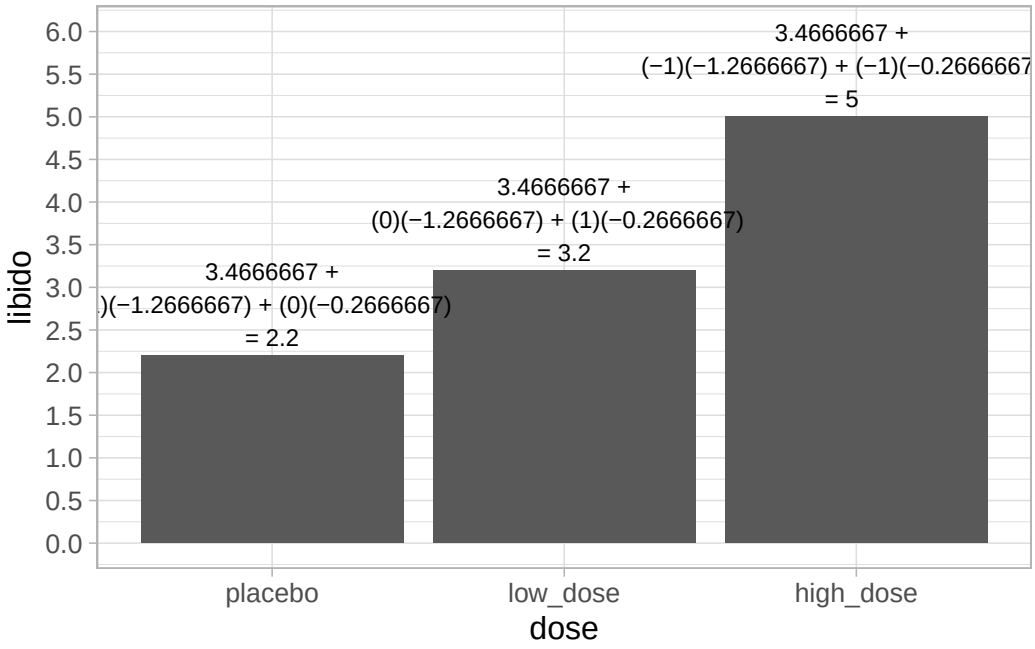
```
lm_viagra <- lm(libido ~ dose + 1, data = viagra)
```

The computer actually ran this:

```
lm_viagra_sum <- lm(libido ~ dose1 + dose2 + 1,
                    data = viagra_sum_coded)
```

```
tidy(lm_viagra_sum)
```

term	estimate	std.error	statistic	p.value
(Intercept)	3.4666667	0.3620927	9.5739760	0.0000006
dose1	-1.2666667	0.5120764	-2.4735893	0.0293002
dose2	-0.2666667	0.5120764	-0.5207556	0.6120112



8.3.2 ANOVA values

```
glance(lm_viagra_sum)
```

r.squared	adj.r.squared	sigma	statistic	p.value	df	logLik	AIC	BIC	deviance	df.residual	nobs
0.4603659	0.3704268	1.402379	5.118644	0.0246943	2	-24.68305	57.36616	60.1983	23.6	12	15

From this output you can see that for this one-way ANOVA, F(2,12) = 5.118644, p = 0.0246943.

```
summary(lm_viagra_sum)
```

```
Call:
lm(formula = libido ~ dose1 + dose2 + 1, data = viagra_sum_coded)

Residuals:
```

```
   Min     1Q Median     3Q    Max
  -2.0   -1.2   -0.2    0.9    2.0

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.4667     0.3621   9.574 5.72e-07 ***
dose1        -1.2667     0.5121  -2.474   0.0293 *
dose2        -0.2667     0.5121  -0.521   0.6120
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.402 on 12 degrees of freedom
Multiple R-squared:  0.4604,    Adjusted R-squared:  0.3704
F-statistic: 5.119 on 2 and 12 DF,  p-value: 0.02469
```

From this output you can (again) see that for this one-way ANOVA, F(2,12) = 5.118644, p = 0.0246943.

8.4 Helmert Contrast

Use the Helmert Contrast if you are interested in typical ANOVA results (main effect, main effect, interaction, etc.).

8.5 Summary: Contrast Types

Name/Synonym	R command Example	Nature of Comparison	Note

Treatment Contrast / Dummy Contrast	`> contr.treatment(4)`
	`  2 3 4`
	`1 0 0 0`
	`2 1 0 0`
	`3 0 1 0`
	`4 0 0 1`
		Intercept is the reference group mean. Unstandardized weights indicate difference between a group's mean and reference group mean. In this example, group 1 is the reference group. Notice the first contrast column is for the second group - because group 1 is the intercept.	Default in R. **Do NOT use for ANOVA.**

Sum Contrast / Effect Contrast	`> contr.sum(4)`
	`  [,1] [,2] [,3]`
	`1  1  0  0`
	`2  0  1  0`
	`3  0  0  1`
	`4 -1 -1 -1`
		Intercept is the grand mean. Unstandardized weights are used to create a predicted score that corresponds to the mean for each group.	**USE for ANOVA.** This works because the intercept is the grand mean.

Name/Synonym	R command Example	Nature of Comparison	Note

Helmert Contrast	`> contr.helmert(4)`
	`  [,1] [,2] [,3]`
	`1   -1   -1   -1`
	`2    1   -1   -1`
	`3    0    2   -1`
	`4    0    0    3`
		Intercept is the grand mean. First contrast, unstandardized weight indicate difference between Group 2 and Group 1 means. Second contrast, unstandardized weight indicate difference between Group 3 mean and the average of the Group 1 and Group 2 means. Third contrast, unstandardized weight indicate difference between Group 4 mean and the average of the Group 1, Group 2, and Group 3 means. And so on.	**USE for ANOVA.** This works because the intercept is the grand mean.

9 Two-way ANOVA via Regression

On this page we illustrate what is happening "under the hood" when you run an ANOVA using regression.

9.1 Conducting a 2-way ANOVA

9.1.1 Activate Packages

```
library(tidyverse)
library(janitor)
library(pracma)
library(recipes)
library(forcats)
library(tidymodels)
library(apaTables)
```

9.1.2 Load Data

```
gdata = read_csv("gdata.csv")
```

9.1.3 Inspect Data

This example is from the Andy Field book. The example uses alcohol(0, 2 pints, 4 pints) and sex(male,female) as predictors of attractiveness. See the Discovering Statistics Using R (2012) for the complete example. We use this example to illustrate sum contrasts in an ANOVA context.

```
glimpse(gdata)
```

```
Rows: 41
Columns: 3
$ attractiveness <dbl> 60, 60, 55, 60, 55, 70, 65, 60, 70, 65, 60, 60, 50, 55,~
$ gender         <chr> "female", "female", "female", "female", "female", "fema~
$ alcohol        <chr> "none", "none", "none", "none", "none", "pint2", "pint2~
```

```
print(gdata)
```

```
   attractiveness gender alcohol
1              60 female    none
2              60 female    none
3              55 female    none
4              60 female    none
5              55 female    none
6              70 female   pint2
7              65 female   pint2
8              60 female   pint2
9              70 female   pint2
10             65 female   pint2
11             60 female   pint2
12             60 female   pint2
13             50 female   pint2
14             55 female   pint4
15             65 female   pint4
16             70 female   pint4
17             55 female   pint4
18             55 female   pint4
19             60 female   pint4
20             50 female   pint4
21             50 female   pint4
22             50   male    none
23             55   male    none
24             80   male    none
25             65   male    none
26             70   male    none
27             75   male    none
28             75   male    none
29             65   male    none
30             45   male   pint2
31             60   male   pint2
32             85   male   pint2
33             65   male   pint2
```

```
34             70   male   pint2
35             70   male   pint2
36             80   male   pint2
37             60   male   pint2
38             30   male   pint4
39             30   male   pint4
40             30   male   pint4
41             55   male   pint4
```

9.1.4 Make Factors

```
gdata <- gdata %>%
  mutate(gender = as_factor(gender)) %>%
  mutate(alcohol = as_factor(alcohol))

glimpse(gdata)
```

```
Rows: 41
Columns: 3
$ attractiveness <dbl> 60, 60, 55, 60, 55, 70, 65, 60, 70, 65, 60, 60, 50, 55,~
$ gender         <fct> female, female, female, female, female, female, female,~
$ alcohol        <fct> none, none, none, none, none, pint2, pint2, pint2, pint~
```

9.1.5 Linear Model

Because we want to run an ANOVA we use a sum contrast.

```
options(contrasts = c("contr.sum", "contr.poly"))
```

```
lm_output <- lm(attractiveness ~  gender*alcohol,
                data = gdata)
```

But note that when we run the command above with gender*alcohol we need to realize is is a shortcut convention for the code below - which implicitly includes an intercept:

```
lm_output <- lm(attractiveness ~ gender+ alcohol + gender:alcohol,
                data = gdata)
```

Which is in turn a shortcut for the code below which explicitly includes the intercept:

```
lm_output <- lm(attractiveness ~ gender+ alcohol + gender:alcohol + 1,
                data = gdata)
```

```
table1 <- apa.aov.table(lm_output, table.number = 1)
apa.save("table1aov.doc", table1)
```

Predictor	SS	df	MS	F	p	$\eta^2_{partial}$	90% CI
(Intercept)	127477.89	1	127477.89	1490.97	<.001		
gender	67.37	1	67.37	0.79	.381	.02	[.00, .15]
alcohol	2230.76	2	1115.38	13.05	<.001	.43	[.19, .56]
gender x alcohol	1508.65	2	754.33	8.82	.001	.34	[.11, .48]
Error	2992.50	35	85.50				

The table above is helpful for interpretting the data, but, how did we obtain it? There is a lot that happens to get the above table. Especially when you remember that if we look at the results of the regression itself it looks quite different:

```
tidy(lm_output)
```

```
# A tibble: 6 x 5
  term              estimate std.error statistic  p.value
  <chr>                <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)         58         1.50    38.6   2.76e-30
2 gender1              1.33      1.50     0.888 3.81e- 1
3 alcohol1             4.44      2.14     2.08  4.54e- 2
4 alcohol2             6.69      2.01     3.33  2.07e- 3
5 gender1:alcohol1    -5.77      2.14    -2.70  1.06e- 2
6 gender1:alcohol2    -3.52      2.01    -1.75  8.85e- 2
```

9.2 Regression Becoming ANOVA

A bit of magic seems to happen in the above. We conduct a regression and then somehow get an ANOVA table out at the end. How does that work? The key is understanding that when we specify the regression with factors in it - we a really giving the computer a set of instruction and a starting point - rather than an actual analysis. The computer does a few things "under the hood":

1) Factors are turned into contrasts columns
2) Multiple columns be be required for a single factor
3) The number of contrast columns required for a single factor column is equal to the number levels minus one.
4) The "actual analysis" is conducted using the contrast columns - not the factor column from your data set.

9.3 Contrasts for Categorical Variables

Let's load a new data set that has some contrast columns created already.

```
gdata <- read_csv("gdata_contrasts.csv")
```

9.3.1 Gender Contrasts

In R when you use the line:

```
options(contrasts = c("contr.sum", "contr.poly"))
```

It effectively runs the contr.sum() command on each factor column, when a regression is run, and creates contrasts based on the number of levels of each factor. For example, the sex factor, with levels, causes the command below to be run.

```
contr.sum(2)
```

```
  [,1]
1    1
2   -1
```

These rules are applied to the gender column. We create a new column called sex where this rule has been applied. In the output below, I have already applied this rule and put the result in the sex column. Normally this happens "under the hood" and you don't see it. Notice how every female is coded 1 in the sex column where as males are coded -1 in the sex column; consistent with the contr.sum(2) command.

```
gdata %>%
  select(attractiveness, gender, sex) %>%
  as.data.frame()
```

```
   attractiveness gender sex
1              60 female   1
2              60 female   1
3              55 female   1
4              60 female   1
5              55 female   1
6              70 female   1
```

```
7              65 female   1
8              60 female   1
9              70 female   1
10             65 female   1
11             60 female   1
12             60 female   1
13             50 female   1
14             55 female   1
15             65 female   1
16             70 female   1
17             55 female   1
18             55 female   1
19             60 female   1
20             50 female   1
21             50 female   1
22             50   male  -1
23             55   male  -1
24             80   male  -1
25             65   male  -1
26             70   male  -1
27             75   male  -1
28             75   male  -1
29             65   male  -1
30             45   male  -1
31             60   male  -1
32             85   male  -1
33             65   male  -1
34             70   male  -1
35             70   male  -1
36             80   male  -1
37             60   male  -1
38             30   male  -1
39             30   male  -1
40             30   male  -1
41             55   male  -1
```

9.3.2 Alcohol Contrasts

```
contr.sum(3)
```

```
  [,1] [,2]
```

```
1    1    0
2    0    1
3   -1   -1
```

In the output below, I have already applied this rule and put the result in the alc1 and alc2 columns. Normally this happens "under the hood" and you don't see it. Notice how every levels of alcohol are coded using this scheme; consistent with the contr.sum(3) command.

```
gdata %>%
  select(attractiveness, alcohol, alc1, alc2) %>%
  as.data.frame()
```

```
   attractiveness alcohol alc1 alc2
1              60    none    1    0
2              60    none    1    0
3              55    none    1    0
4              60    none    1    0
5              55    none    1    0
6              70   pint2    0    1
7              65   pint2    0    1
8              60   pint2    0    1
9              70   pint2    0    1
10             65   pint2    0    1
11             60   pint2    0    1
12             60   pint2    0    1
13             50   pint2    0    1
14             55   pint4   -1   -1
15             65   pint4   -1   -1
16             70   pint4   -1   -1
17             55   pint4   -1   -1
18             55   pint4   -1   -1
19             60   pint4   -1   -1
20             50   pint4   -1   -1
21             50   pint4   -1   -1
22             50    none    1    0
23             55    none    1    0
24             80    none    1    0
25             65    none    1    0
26             70    none    1    0
27             75    none    1    0
28             75    none    1    0
29             65    none    1    0
```

```
30              45   pint2    0    1
31              60   pint2    0    1
32              85   pint2    0    1
33              65   pint2    0    1
34              70   pint2    0    1
35              70   pint2    0    1
36              80   pint2    0    1
37              60   pint2    0    1
38              30   pint4   -1   -1
39              30   pint4   -1   -1
40              30   pint4   -1   -1
41              55   pint4   -1   -1
```

9.3.3 Interaction Contrasts

We also need contrasts for the interaction. We create the interaction contrasts by multiplying the columns for sex, alc1, and alc2. You can see how we do so in the code below.

```
gdata <- gdata %>%
  mutate(int1 = sex*alc1,
         int2 = sex*alc2)
```

You can see these new interaction columns below:

```
print(gdata)
```

```
   attractiveness sex alc1 alc2 int1 int2
1              60   1    1    0    1    0
2              60   1    1    0    1    0
3              55   1    1    0    1    0
4              60   1    1    0    1    0
5              55   1    1    0    1    0
6              70   1    0    1    0    1
7              65   1    0    1    0    1
8              60   1    0    1    0    1
9              70   1    0    1    0    1
10             65   1    0    1    0    1
11             60   1    0    1    0    1
12             60   1    0    1    0    1
13             50   1    0    1    0    1
14             55   1   -1   -1   -1   -1
```

15	65	1	-1	-1	-1	-1
16	70	1	-1	-1	-1	-1
17	55	1	-1	-1	-1	-1
18	55	1	-1	-1	-1	-1
19	60	1	-1	-1	-1	-1
20	50	1	-1	-1	-1	-1
21	50	1	-1	-1	-1	-1
22	50	-1	1	0	-1	0
23	55	-1	1	0	-1	0
24	80	-1	1	0	-1	0
25	65	-1	1	0	-1	0
26	70	-1	1	0	-1	0
27	75	-1	1	0	-1	0
28	75	-1	1	0	-1	0
29	65	-1	1	0	-1	0
30	45	-1	0	1	0	-1
31	60	-1	0	1	0	-1
32	85	-1	0	1	0	-1
33	65	-1	0	1	0	-1
34	70	-1	0	1	0	-1
35	70	-1	0	1	0	-1
36	80	-1	0	1	0	-1
37	60	-1	0	1	0	-1
38	30	-1	-1	-1	1	1
39	30	-1	-1	-1	1	1
40	30	-1	-1	-1	1	1
41	55	-1	-1	-1	1	1

9.4 Regression command (i.e., lm) overview

To get ANOVA results that are consistent with what are typically used in psychology you need to 1) Specify the contr.sum() contrast 2) Calculate the Sum of Squares using the logic for Type III Sum of Squares

When you run an ANOVA using the command:

```
options(contrasts = c("contr.sum", "contr.poly"))

lm_output <- lm(attractiveness ~ gender + alcohol + gender:alcohol,
                data = gdata)
```

You actually run the regression below:

```
lm_output <- lm(attractiveness ~ gender1 + alcohol1 + alcohol2 + gender:alcohol1 + gender:alc
                data = gdata)
```

Which uses these columns - notice there is a column of all ones for the intercept:

```
print(gdata)
```

```
   attractiveness intercept sex alc1 alc2 int1 int2
1              60         1   1    1    0    1    0
2              60         1   1    1    0    1    0
3              55         1   1    1    0    1    0
4              60         1   1    1    0    1    0
5              55         1   1    1    0    1    0
6              70         1   1    0    1    0    1
7              65         1   1    0    1    0    1
8              60         1   1    0    1    0    1
9              70         1   1    0    1    0    1
10             65         1   1    0    1    0    1
11             60         1   1    0    1    0    1
12             60         1   1    0    1    0    1
13             50         1   1    0    1    0    1
14             55         1   1   -1   -1   -1   -1
15             65         1   1   -1   -1   -1   -1
16             70         1   1   -1   -1   -1   -1
17             55         1   1   -1   -1   -1   -1
18             55         1   1   -1   -1   -1   -1
19             60         1   1   -1   -1   -1   -1
20             50         1   1   -1   -1   -1   -1
21             50         1   1   -1   -1   -1   -1
22             50         1  -1    1    0   -1    0
23             55         1  -1    1    0   -1    0
24             80         1  -1    1    0   -1    0
25             65         1  -1    1    0   -1    0
26             70         1  -1    1    0   -1    0
27             75         1  -1    1    0   -1    0
28             75         1  -1    1    0   -1    0
29             65         1  -1    1    0   -1    0
30             45         1  -1    0    1    0   -1
31             60         1  -1    0    1    0   -1
32             85         1  -1    0    1    0   -1
33             65         1  -1    0    1    0   -1
```

```
34              70          1  -1    0    1    0   -1
35              70          1  -1    0    1    0   -1
36              80          1  -1    0    1    0   -1
37              60          1  -1    0    1    0   -1
38              30          1  -1   -1   -1    1    1
39              30          1  -1   -1   -1    1    1
40              30          1  -1   -1   -1    1    1
41              55          1  -1   -1   -1    1    1
```

9.5 Full and Restricted Models

R actually runs a whole series of regressions for you and combines them into the single output table you saw above. These models fall into two categories Full and Restricted Models.

9.5.1 Full Model

First, the Full Model is run that includes all of the predictor columns:

```
lm_full <- lm(attractiveness ~ sex + alc1 + alc2 + int1 + int2,
              data = gdata)
```

9.5.2 Restricted Models

Next a series of restricted models are run that excluded an effect of interest for each restricted model.

```
lm_restricted_no_sex <- lm(attractiveness ~ alc1 + alc2 + int1 + int2,
                           data = gdata)

lm_restricted_no_alcohol <- lm(attractiveness ~ sex + int1 + int2,
                               data = gdata)

lm_restricted_no_interaction <- lm(attractiveness ~ sex + alc1 + alc2,
                                   data = gdata)

lm_restricted_no_intercept <- lm(attractiveness ~ sex + alc1 + alc2 + int1 + int2 - 1,
                                 data = gdata)
```

9.6 Logic: Model Comparison

The ANOVA table is created by comparing each of these restricted models to the full model. For example, to determine the main effect for gender we compare the model lm_restricted_no_sex to the model lm_full. If lm_full accounts for substantially more variance than lm_restricted_no_sex it is significant.

This is effectively identical to when we looked at comparing two regression models previous. Using that logic, we could just write the code:

```
# try this, it works!
library(apaTables)
apa.reg.table(lm_restricted_no_sex, lm_restricted_all)
```

The result would tell us if the main effect of sex is significant. The logic of calculating things this way is the Type III Sum of Squares logic. HOWEVER, we don't tend to use the output in the form provided in this type of table. Rather the output if reformatted to by consistent with the way ANOVA's are typically presented. The next few sections show you how the table below is created:

Predictor	*SS*	*df*	*MS*	*F*	*p*	$\eta^2_{partial}$	90% CI
(Intercept)	127477.89	1	127477.89	1490.97	<.001		
gender	67.37	1	67.37	0.79	.381	.02	[.00, .15]
alcohol	2230.76	2	1115.38	13.05	<.001	.43	[.19, .56]
gender x alcohol	1508.65	2	754.33	8.82	.001	.34	[.11, .48]
Error	2992.50	35	85.50				

9.7 Explanation 1: Comparing Models

9.7.1 Sex

```
lm_restricted_no_sex <- lm(attractiveness ~ alc1 + alc2 + int1 + int2,
                           data = gdata)

lm_full <- lm(attractiveness ~ sex + alc1 + alc2 + int1 + int2,
              data = gdata)

anova(lm_restricted_no_sex, lm_full)
```

```
Analysis of Variance Table
```

```
Model 1: attractiveness ~ alc1 + alc2 + int1 + int2
Model 2: attractiveness ~ sex + alc1 + alc2 + int1 + int2
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     36 3059.9
2     35 2992.5  1    67.368 0.7879 0.3808
```

Compare the $F-$ and p-values in the above output to those in the table below.

Predictor	SS	df	MS	F	p	$\eta^2_{partial}$	90% CI
(Intercept)	127477.89	1	127477.89	1490.97	<.001		
gender	67.37	1	67.37	0.79	.381	.02	[.00, .15]
alcohol	2230.76	2	1115.38	13.05	<.001	.43	[.19, .56]
gender x alcohol	1508.65	2	754.33	8.82	.001	.34	[.11, .48]
Error	2992.50	35	85.50				

9.7.2 Alcohol

```r
lm_restricted_no_alcohol <- lm(attractiveness ~ sex + int1 + int2,
                               data = gdata)

lm_full <- lm(attractiveness ~ sex + alc1 + alc2 + int1 + int2,
              data = gdata)

anova(lm_restricted_no_alcohol, lm_full)
```

```
Analysis of Variance Table

Model 1: attractiveness ~ sex + int1 + int2
Model 2: attractiveness ~ sex + alc1 + alc2 + int1 + int2
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     37 5223.3
2     35 2992.5  2    2230.8 13.045 5.843e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Compare the $F-$ and p-values in the above output to those in the table below.

Predictor	SS	df	MS	F	p	$\eta^2_{partial}$	90% CI
(Intercept)	127477.89	1	127477.89	1490.97	<.001		
gender	67.37	1	67.37	0.79	.381	.02	[.00, .15]
alcohol	2230.76	2	1115.38	13.05	<.001	.43	[.19, .56]
gender x alcohol	1508.65	2	754.33	8.82	.001	.34	[.11, .48]
Error	2992.50	35	85.50				

9.7.3 Interaction

```
lm_restricted_no_interaction <- lm(attractiveness ~ sex + alc1 + alc2,
                                   data = gdata)

lm_full <- lm(attractiveness ~ sex + alc1 + alc2 + int1 + int2,
              data = gdata)

anova(lm_restricted_no_interaction, lm_full)
```

```
Analysis of Variance Table

Model 1: attractiveness ~ sex + alc1 + alc2
Model 2: attractiveness ~ sex + alc1 + alc2 + int1 + int2
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     37 4501.2
2     35 2992.5  2    1508.7 8.8225 0.0007896 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Compare the $F-$ and p-values in the above output to those in the table below.

Predictor	SS	df	MS	F	p	$\eta^2_{partial}$	90% CI
(Intercept)	127477.89	1	127477.89	1490.97	<.001		
gender	67.37	1	67.37	0.79	.381	.02	[.00, .15]
alcohol	2230.76	2	1115.38	13.05	<.001	.43	[.19, .56]
gender x alcohol	1508.65	2	754.33	8.82	.001	.34	[.11, .48]
Error	2992.50	35	85.50				

9.8 Degrees of Freedom

When you look at the the columns in the above output notice the number of columns we use for each predictor corresponds the degrees of freedom for that predictor.

Predictor	df	Number of contrast columns	Contrast column names
sex	$df_a = a - 1 = 2 - 1 = 1$	1	sex
alcohol	$df_b = b - 1 = 3 - 1 = 2$	2	alc1, alch2

Predictor	df	Number of contrast columns	Contrast column names
sex by alcohol	$df_{int} = df_a * df_b = (a-1)(b-1) = 1(2) = 2$	2	int1, int2

9.9 Explanation 2: Sum of Squares to ANOVA

Notice the first column of the ANOVA table above is the Sum of Squares column. How are those values calculated? Let's consider the example of alcohol as a predictor. We want to determine the Sum of Squares for alcohol.

We begin by calculating predicted scores for the Full Model (lm_full):

$$\hat{y}_{\text{full}} = b_0 + b_1 sex + b_2 alc1 + b_3 alc2 + b_4 int1 + b_5 int2$$

Next, we calculate the predicted scores for the alcohol restricted model (lm_restricted_no_alcohol)

$$\hat{y}_{\text{restricted no alcohol}} = b_0 + b_1 sex + b_2 int1 + b_3 int2$$

Then we calculated the difference between these two sets of predicted scores:

$$\hat{y}_{\text{difference}} = \hat{y}_{\text{full}} - \hat{y}_{\text{restricted no alcohol}}$$

Then we square these values and add them up.

$$SS_{alcohol} = \sum \hat{y}^2_{\text{difference}}$$

We follow this process below for each predictor (including the intercept).

9.9.1 Intercept

```
## Sum of squares intercept
sum( ( predict(lm_full) - predict(lm_restricted_no_intercept) )^2 )
```

```
[1] 127477.9
```

Compare the Sum of Squares values in the above output to the one in the table below.

Predictor	SS	df	MS	F	p	$\eta^2_{partial}$	90% CI
(Intercept)	127477.89	1	127477.89	1490.97	<.001		
gender	67.37	1	67.37	0.79	.381	.02	[.00, .15]
alcohol	2230.76	2	1115.38	13.05	<.001	.43	[.19, .56]
gender x alcohol	1508.65	2	754.33	8.82	.001	.34	[.11, .48]
Error	2992.50	35	85.50				

9.9.2 Sex

```
## Sum of squares sex
sum( ( predict(lm_full) - predict(lm_restricted_no_sex) )^2 )
```

```
[1] 67.36842
```

Compare the Sum of Squares values in the above output to the one in the table below.

Predictor	SS	df	MS	F	p	$\eta^2_{partial}$	90% CI
(Intercept)	127477.89	1	127477.89	1490.97	<.001		
gender	67.37	1	67.37	0.79	.381	.02	[.00, .15]
alcohol	2230.76	2	1115.38	13.05	<.001	.43	[.19, .56]
gender x alcohol	1508.65	2	754.33	8.82	.001	.34	[.11, .48]
Error	2992.50	35	85.50				

9.9.3 Alcohol

```
## Sum of squares alcohol
sum( ( predict(lm_full) - predict(lm_restricted_no_alcohol) )^2 )
```

```
[1] 2230.757
```

Compare the Sum of Squares values in the above output to the one in the table below.

Predictor	SS	df	MS	F	p	$\eta^2_{partial}$	90% CI
(Intercept)	127477.89	1	127477.89	1490.97	<.001		
gender	67.37	1	67.37	0.79	.381	.02	[.00, .15]
alcohol	2230.76	2	1115.38	13.05	<.001	.43	[.19, .56]
gender x alcohol	1508.65	2	754.33	8.82	.001	.34	[.11, .48]
Error	2992.50	35	85.50				

9.9.4 Interaction

```
## Sum of squares interaction
sum( ( predict(lm_full) - predict(lm_restricted_no_interaction) )^2 )
```

```
[1] 1508.651
```

Compare the Sum of Squares values in the above output to the one in the table below.

Predictor	SS	df	MS	F	p	$\eta^2_{partial}$	90% CI
(Intercept)	127477.89	1	127477.89	1490.97	<.001		
gender	67.37	1	67.37	0.79	.381	.02	[.00, .15]
alcohol	2230.76	2	1115.38	13.05	<.001	.43	[.19, .56]
gender x alcohol	1508.65	2	754.33	8.82	.001	.34	[.11, .48]
Error	2992.50	35	85.50				

9.9.5 Error

```
## Sum of squares error
sum( lm_full$residuals^2 )
```

```
[1] 2992.5
```

Compare the Sum of Squares values in the above output to the one in the table below.

Predictor	SS	df	MS	F	p	$\eta^2_{partial}$	90% CI
(Intercept)	127477.89	1	127477.89	1490.97	<.001		
gender	67.37	1	67.37	0.79	.381	.02	[.00, .15]
alcohol	2230.76	2	1115.38	13.05	<.001	.43	[.19, .56]
gender x alcohol	1508.65	2	754.33	8.82	.001	.34	[.11, .48]
Error	2992.50	35	85.50				

9.9.6 SS to ANOVA

Based on the above analyses we know the Sum of Squares and the degrees of freedom for everything. We can put that information in the table below.

Predictor	SS	df	MS= $\frac{SS}{df}$	F= $\frac{MS}{MS_{error}}$	p
(Intercept)	127477.89	1			
sex	67.37	1			

Predictor	SS	df	MS= $\frac{SS}{df}$	F= $\frac{MS}{MS_{error}}$	p
alcohol	2230.76	2			
sex by alcohol	1508.65	2			
Error	2992.5	35			

A few hand calculations, and an F to p-value look-up table, provides us with the rest of the information we need:

Predictor	SS	df	MS= $\frac{SS}{df}$	F= $\frac{MS}{MS_{error}}$	p
(Intercept)	127477.89	1	$\frac{127477.89}{1} = 127477.89$	$\frac{127477.89}{85.5} = 1490.97$	<.001
sex	67.37	1	$\frac{67.37}{1} = 67.37$	$\frac{67.37}{85.5} = 0.79$	.381
alcohol	2230.76	2	$\frac{2230.76}{2} = 1115.38$	$\frac{1115.38}{85.5} = 13.05$	<.001
sex by alcohol	1508.65	2	$\frac{1508.65}{2} = 754.325$	$\frac{754.325}{85.5} = 8.82$	.001
Error	2992.5	35	$\frac{2992.5}{35} = 85.5$		

References

Baker, M. 2016. "1500 Scientists Lift the Lid on Reproducibility." *Nature* 533. https://doi.org/10.1038/533452a.

Meyer, John P, Natalie J Allen, and Catherine A Smith. 1993. "Commitment to Organizations and Occupations: Extension and Test of a Three-Component Conceptualization." *Journal of Applied Psychology* 78 (4): 538.

Miyakawa, T. 2020. "No Raw Data, No Science: Another Possible Source of the Reproducibility Crisis." *Mol Brain* 13 (24). https://doi.org/10.1186/s13041-020-0552-2.

Nosek. 2015. "Estimating the Reproducibility of Psychological Science." *Science* 349. http://doi.org/10.1126/science.aac4716.

Patil P., & Leek J. T, Peng R. D. 2019. "A Visual Tool for Defining Reproducibility and Replicability." *Nat Hum Behav* 3: 650–52. https://doi.org/10.1038/s41562-019-0629-z.

Simmons, Joseph P, Leif D Nelson, and Uri Simonsohn. 2011. "False-Positive Psychology: Undisclosed Flexibility in Data Collection and Analysis Allows Presenting Anything as Significant." *Psychological Science* 22 (11): 1359–66.

Thompson, Edmund R, and Florence TT Phua. 2012. "A Brief Index of Affective Job Satisfaction." *Group & Organization Management* 37 (3): 275–307.

Wickham, Hadley. 2014. "Tidy Data." *The Journal of Statistical Software* 59. http://www.jstatsoft.org/v59/i10/.