Explorations: Conducting Empirical Research in Canadian Political Science (4th Edition)

R Handbook

David Armstrong, Jason Roy and Loleen Berdahl

Welcome to the *Explorations: Conducting Empirical Research in Canadian Political Science R Handbook!* In this handbook, we provide you with a basic introduction to R. The procedures outlined follow from the statistical methods described in *Explorations: Conducting Empirical Research in Canadian Political Science (4th Edition)*. We encourage you to work closely with the textbook as you move through this handbook; here we cover the technical "how to" of basic statistics, but we do not cover the critical issues of which statistics to use when. We use 2019 Canadian Election Study (CES) data. As we explain throughout the textbook, the CES data are a great publicly available resource for studying politics in Canada. We encourage you to practice the techniques outlined in this handbook with the CES datasets.

To follow the procedures below, download and open the 2019 CES dataset (we use the telephone survey), available for free at https://doi.org/10.7910/DVN/8RHLG1.You should also download the 2019 CES technical documentation and codebook for reference.

Please note that the screen shots included are those captured working with RStudio (v1.2.1335) on a Mac. R is an open-source statistical computing environment and can be downloaded freely from https://cran.r-project.org/. Both SPSS and Stata – other software covered in *Explorations* handbooks are their own integrated development environments (or IDEs). To make R most useful, you need to download an IDE for R. There are lots of options here, but the one we are going to use is RStudio Desktop, which is also open-source and can be downloaded freely from https://rstudio.com/products/rstudio/download/. The appearance of the RStudio work environment may differ across operating systems.

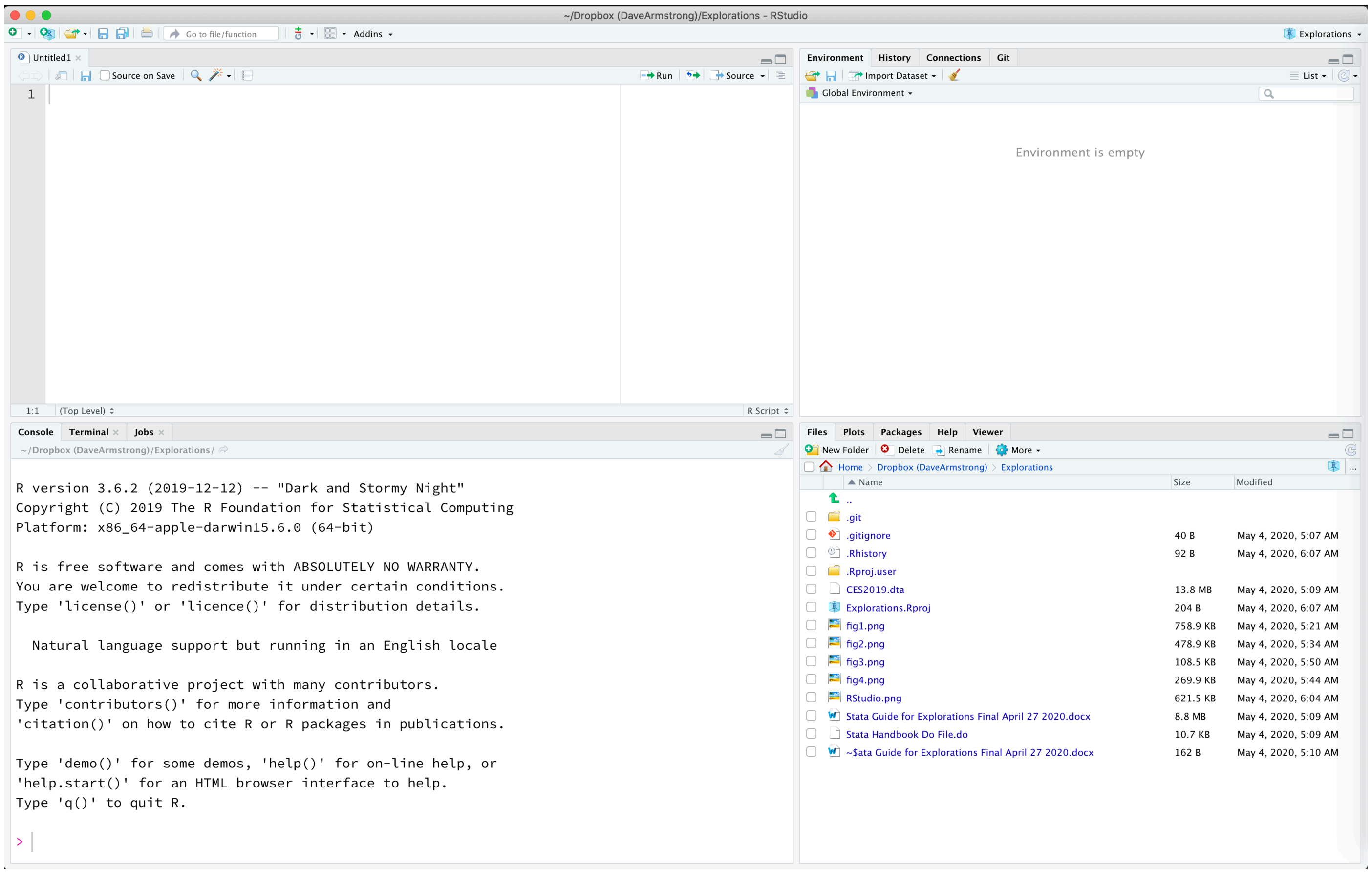
Part I: Getting Comfortable with R

Destination

By the end of this section, you will be able to:

- *navigate between RStudio windows;*
- *explain what an R script file is, and why it is a valuable tool for researchers;*
- *download and enable* R *packages;*
- *record your work in R;*
- *open a dataset in RStudio; and*
- *use search functions*

The RStudio work environment consists of a number of windows/screens, each with different information. (Note: The windows that are displayed when you first open RStudio will vary according to the current context and any user settings that have been applied.) The RStudio interface generally has four tiles (or sub-windows). You should see something similar to the view below if you choose *File > New File > R Script*.



```
R version 3.6.2 (2019-12-12) -- "Dark and Stormy Night"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

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

  Natural language support but running in an English locale

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

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

>
```

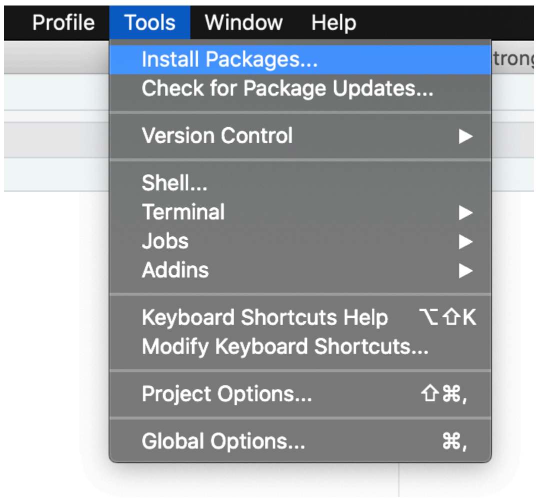
Name	Size	Modified
..		
.git		
.gitignore	40 B	May 4, 2020, 5:07 AM
.Rhistory	92 B	May 4, 2020, 6:07 AM
.Rproj.user		
CES2019.dta	13.8 MB	May 4, 2020, 5:09 AM
Explorations.Rproj	204 B	May 4, 2020, 6:07 AM
fig1.png	758.9 KB	May 4, 2020, 5:21 AM
fig2.png	478.9 KB	May 4, 2020, 5:34 AM
fig3.png	108.5 KB	May 4, 2020, 5:50 AM
fig4.png	269.9 KB	May 4, 2020, 5:44 AM
RStudio.png	621.5 KB	May 4, 2020, 6:04 AM
Stata Guide for Explorations Final April 27 2020.docx	8.8 MB	May 4, 2020, 5:09 AM
Stata Handbook Do File.do	10.7 KB	May 4, 2020, 5:09 AM
~$ata Guide for Explorations Final April 27 2020.docx	162 B	May 4, 2020, 5:10 AM

The features in the four tiles are described below.

- The **Script Editor window** (upper-left above) is a workspace where you can write, edit, and save R commands. Rather than entering these commands in the console window, you can run them from the script editor. The advantage is that you can easily edit, save and re-run all your analyses. We strongly recommend working with a script file in R. Doing so allows you to record all of the procedures that you run and easily re-produce all results as needed. It also allows you to easily collaborate with colleagues working on the same project with you, and to share a history of your analysis with others, thus increasing research transparency. (See Chapter 3 for a discussion of ethics and data analysis.) Users can add a number sign (#) at any point in a line of code and anything that follows it will be interpreted by R as a comment and not a command to be executed. You can use comments to leave notes to yourself or your colleagues or to add a title to the file. To run a command from the script file, place your cursor on the line you want to execute and click the "Run Current Line" icon at the top left corner of the script editor. You may also use the short-cut keys: command + return on a Mac or CTRL + Enter with a Windows operating system. Note that R will return an error if you select only part of a command. Therefore, be sure to only place your curser on the line you wish to execute or select the entire command.
- The **Environment Window** (upper-right above) gives you a list of all of the objects you have created. This could include data frames (data sets), statistical model results and sometimes graphs, depending on how they were created. This tile also has a "History" tab which captures the recent functions you have executed in R.
- The **Console window** (lower-left above) displays results. You can also interact with R directly in the console window by typing commands and having them executed in the window, but doing this is not as desirable as using the script editor since your commands will not be recorded for later use.

- The **Files window** (lower-right above) provides a lot of information. The **files** tab gives you a file explorer. The **plots** tab provides a history of the graphs you have made. The **packages** tab shows you the packages that you have available and the ones you have enabled. The **help** tab allows you to see help files for functions and packages. Occasionally, things you are trying to see , particularly any html output (for example, from the dataTable function) will show up in the **viewer** tab; it is pretty rare that things show up here.

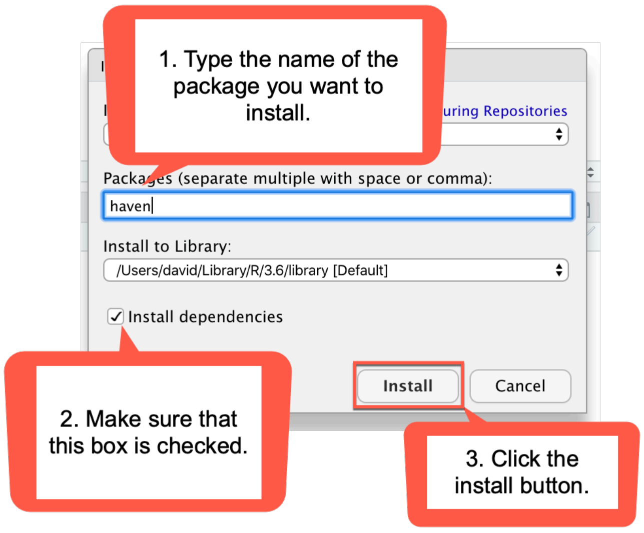
This brings us to the first big different between Stata/SPSS and R. R comes packaged with some very basic functionality, but for lots of tasks, you will need to download an R *package*[1] and then enable that package for your R session. You can do this through RStudio by navigating to the Tools dropdown menu and then choosing Install packages …



This will bring up the dialog box below. Follow the steps to install packages from the internet. You will only need to do this one time for each version of R. That is to say, until you upgrade R to a new version, you will not have to go through this step again. Note that packages often depend on other packages. Making sure the "Install dependencies" box is checked will ensure that you download all of the necessary packages.

For example, to import data into R, you will need to install a package called `{haven}`.[2] To do so, type the package name into the "Packages" textbox, make sure "Install dependencies" is checked, and click on the "Install" button, as shown in the screen capture below:

[1] An R package is a collection of functions that expand R's base functionality.
[2] Throughout this handbook, package names will be placed in braces and printed in Courier font, e.g., `{haven}`. Functions will have parentheses with them, e.g., `mean()`.


1. Type the name of the package you want to install.
uring Repositories
Packages (separate multiple with space or comma):
haven
Install to Library:
/Users/david/Library/R/3.6/library [Default]
Install dependencies
Install
Cancel
2. Make sure that this box is checked.
3. Click the install button.

Throughout this handbook, we will work with a number of packages. We recommend that you follow the steps outlined above to install each of the packages listed below in order to follow along with the procedures in this handbook. If you're working on a mac, you will also need to install the X11 windowing environment from (https://xquartz.macosforge.org).

```
{car}
{dplyr}
{ggplot2}
{haven}
{lattice}
{remotes}
{summarytools}
{survey}
{weights}
```

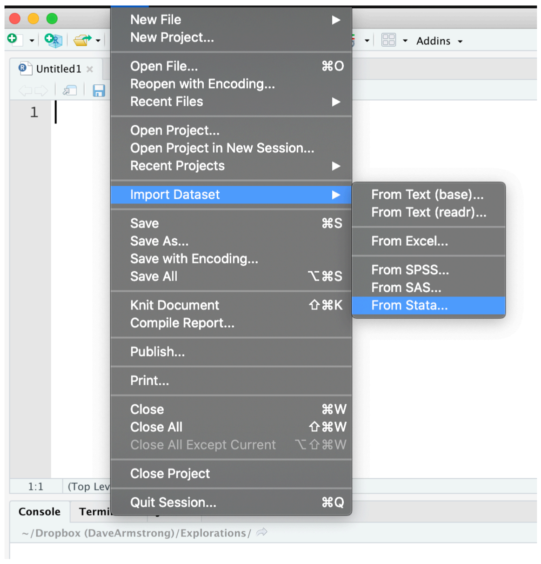
After the package is installed, you can enable its use in your R session by using the `library()` function. For example, to load the `{haven}` package, you could type `library(haven)` into your script file and then highlight that line and click the "run" button in you script editor. You will need to do this for each R session – that is, each time you open up RStudio. Add the script below to your script file and run it to enable each of the packages that you have installed (and be sure to save your script file frequently):

```
library(car)
library(dplyr)
library(ggplot2)
library(haven)
library(lattice)
library(remotes)
library(summarytools)
library(survey)
library(weights)
```

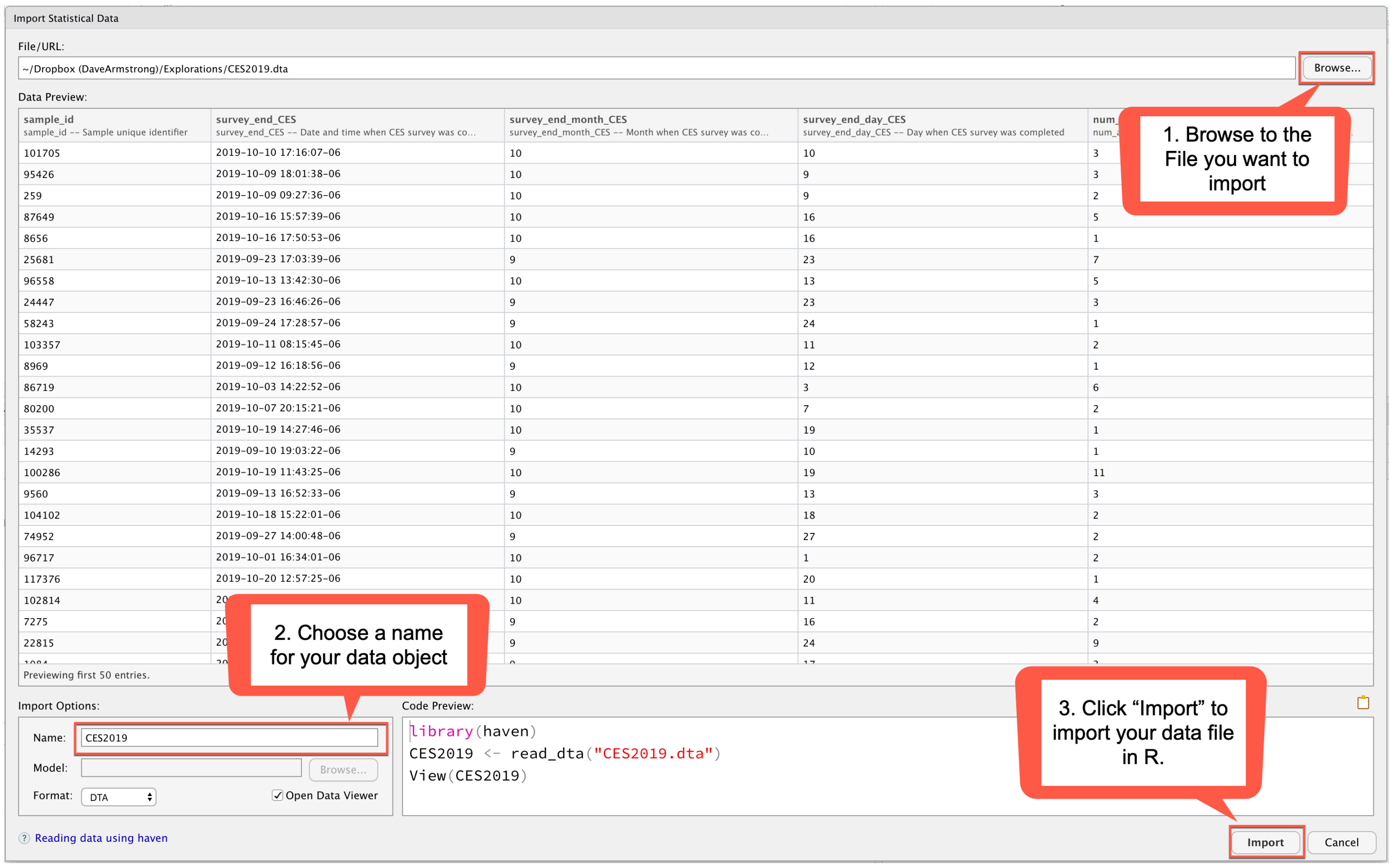
A final package required for use with this handbook, `{DAMisc}`, is available via a different repository. You can install it by running the following script in your script file:

```
install.packages('remotes')
remotes::install_github('davidaarmstrong/damisc')
library(DAMisc)
```

With the packages installed and enabled, we can now import the 2019 CES data. To do so, choose *File > Import Dataset > From Stata* (assuming you have downloaded the data is Stata format):



This will enable the dialog below. Follow the steps identified in the figure to read in the data.


Import Statistical Data
Browse...
1. Browse to the File you want to import
2. Choose a name for your data object
3. Click “Import” to import your data file in R.
Browse...
Reading data using haven
Import
Cancel

File/URL: ~/Dropbox (DaveArmstrong)/Explorations/CES2019.dta

Data Preview:

sample_id sample_id -- Sample unique identifier	survey_end_CES survey_end_CES -- Date and time when CES survey was co...	survey_end_month_CES survey_end_month_CES -- Month when CES survey was co...	survey_end_day_CES survey_end_day_CES -- Day when CES survey was completed	num_ num_a
101705	2019-10-10 17:16:07-06	10	10	3
95426	2019-10-09 18:01:38-06	10	9	3
259	2019-10-09 09:27:36-06	10	9	2
87649	2019-10-16 15:57:39-06	10	16	5
8656	2019-10-16 17:50:53-06	10	16	1
25681	2019-09-23 17:03:39-06	9	23	7
96558	2019-10-13 13:42:30-06	10	13	5
24447	2019-09-23 16:46:26-06	9	23	3
58243	2019-09-24 17:28:57-06	9	24	1
103357	2019-10-11 08:15:45-06	10	11	2
8969	2019-09-12 16:18:56-06	9	12	1
86719	2019-10-03 14:22:52-06	10	3	6
80200	2019-10-07 20:15:21-06	10	7	2
35537	2019-10-19 14:27:46-06	10	19	1
14293	2019-09-10 19:03:22-06	9	10	1
100286	2019-10-19 11:43:25-06	10	19	11
9560	2019-09-13 16:52:33-06	9	13	3
104102	2019-10-18 15:22:01-06	10	18	2
74952	2019-09-27 14:00:48-06	9	27	2
96717	2019-10-01 16:34:01-06	10	1	2
117376	2019-10-20 12:57:25-06	10	20	1
102814	20	10	11	4
7275	20	9	16	2
22815	20	9	24	9

Previewing first 50 entries.

Import Options:

Name: CES2019

Model:

Format: DTA

☑ Open Data Viewer

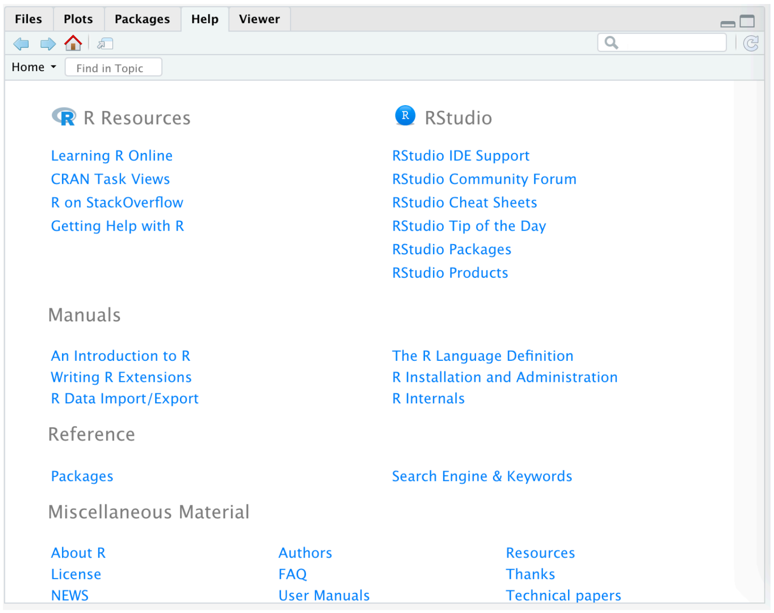
Code Preview:

```
library(haven)
CES2019 <- read_dta("CES2019.dta")
View(CES2019)
```

Clicking the “Import” button above will make a new data frame in R called “CES2019”. The name is one that we assign it and might or might not be the same as the file name.[3] To follow along with the sample script provided below, we recommend assigning the same name (CES2019) to your data frame.

As we near the end of this section, we want to highlight additional R resources. We provide the syntax necessary to run the procedures outlined in this handbook. However, because the syntax presented here is only a small sample of what can be done in R, you should familiarize yourself with how to find help and new packages and functions in R. The **help** tab (located in the lower- right tile) provides access to all of the help files that you have available on your computer along with some other resources. The home screen for the help tab looks like this:

[3] In R, the term data frame refers to a dataset. You can have multiple data frames open simultaneously, thus the need to provide it a name.



R Resources

- Learning R Online
- CRAN Task Views
- R on StackOverflow
- Getting Help with R

RStudio

- RStudio IDE Support
- RStudio Community Forum
- RStudio Cheat Sheets
- RStudio Tip of the Day
- RStudio Packages
- RStudio Products

Manuals

- An Introduction to R
- Writing R Extensions
- R Data Import/Export
- The R Language Definition
- R Installation and Administration
- R Internals

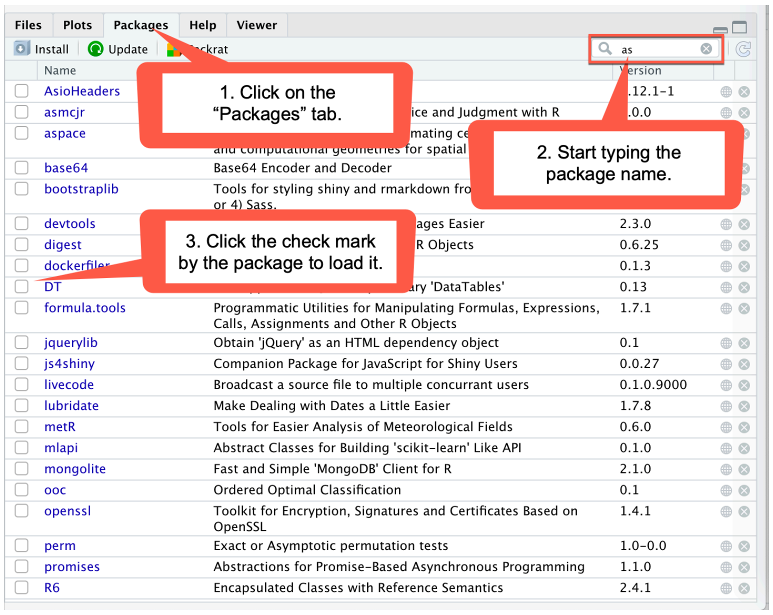
Reference

- Packages
- Search Engine & Keywords

Miscellaneous Material

- About R
- License
- NEWS
- Authors
- FAQ
- User Manuals
- Resources
- Thanks
- Technical papers

Typing a term into the search box in the upper-right corner of the help panel will search through the help files available on your computer. Note that there may be functions available to R that you have not installed yet. These will not be returned when searching the help files. There are a couple of great resources for R users to find new packages or help files that might not exist locally. The website https://rdrr.io/ is an online explorer for help files and documentation for R and its related packages. The website https://rseek.org/ is similar, though also provides access to packages and other web resources, too. The R Stack Overflow page is also a good source of help with R (https://stackoverflow.com/questions/tagged/r). Once you find a function that you need, you should see whether you have the package already (many packages get downloaded as dependencies, so you probably have downloaded more packages than you think). You can do that by going to the "Packages" tab in the lower- right tile of RStudio, as shown below:


Files
Plots
Packages
Help
Viewer
Install
Update
ckrat
as
1. Click on the "Packages" tab.
2. Start typing the package name.
3. Click the check mark by the package to load it.

Name		Version
AsioHeaders		.12.1-1
asmcjr	ice and Judgment with R	.0.0
aspace	mating ce and computational geometries for spatial	
base64	Base64 Encoder and Decoder	
bootstraplib	Tools for styling shiny and rmarkdown fro or 4) Sass.	
devtools	ages Easier	2.3.0
digest	R Objects	0.6.25
dockerfiler		0.1.3
DT	ary 'DataTables'	0.13
formula.tools	Programmatic Utilities for Manipulating Formulas, Expressions, Calls, Assignments and Other R Objects	1.7.1
jquerylib	Obtain 'jQuery' as an HTML dependency object	0.1
js4shiny	Companion Package for JavaScript for Shiny Users	0.0.27
livecode	Broadcast a source file to multiple concurrant users	0.1.0.9000
lubridate	Make Dealing with Dates a Little Easier	1.7.8
metR	Tools for Easier Analysis of Meteorological Fields	0.6.0
mlapi	Abstract Classes for Building 'scikit-learn' Like API	0.1.0
mongolite	Fast and Simple 'MongoDB' Client for R	2.1.0
ooc	Ordered Optimal Classification	0.1
openssl	Toolkit for Encryption, Signatures and Certificates Based on OpenSSL	1.4.1
perm	Exact or Asymptotic permutation tests	1.0-0.0
promises	Abstractions for Promise-Based Asynchronous Programming	1.1.0
R6	Encapsulated Classes with Reference Semantics	2.4.1

If you do not find the package you need, refer back to earlier in this section about how to install and load packages.

Check-In Point

If you are working alongside us, at this point you have opened the 2019 CES dataset in R. You have explored a variety of R windows to increase your familiarity with the various screens available to you. You now know what a script file is and why it is a valuable tool for researchers. Finally, being aware that this handbook teaches you only a small amount of R's capacities, you are familiar with how to search the help files and search for new functions.

With this foundation in place, you are ready to continue your explorations.

Part II: Familiarizing Yourself with Variables in the Dataset

Destination

By the end of this section, you will be able to:

- *use codebook and other commands to familiarize yourself with the variables in the dataset.*

Once you have opened your dataset, you will want to take a preliminary look at the variables.[4] There are several commands that are particularly useful:

dfSummary – displays a frequency distribution of the variable and its label for the entire dataset

inspect – displays a frequency distribution of the variable and its label for a single variable

sumStats– provides summary statistics, such as means and standard deviations.

You can use these commands by typing them in the Command window or in your Script file.

As discussed in Chapter 8, when using secondary datasets such as the CES, it is important to familiarize yourself with the dataset before conducting your analyses. Your first step to do so is to generate a codebook for the dataset. To do this, you can use the `dfSummary()` function from the `{summarytools}` package. Before doing that, though, we need to discuss how R stores variables. Broadly speaking, there are two general types of data in R – numeric (which covers interval and ratio) and factor (which are categorical variables – nominal and ordinal). The `{haven}` package, which we use to read in our data, has a class of data called *haven_labelled*. This indicates that the variable probably should be a factor. This class is used to ensure that the original attributes of the data are preserved (i.e., all of the original numerical values remain the same in the data when it is imported into R). For the data frame summary function to work properly, we should turn all of the *haven_labelled* variables into factors. Before working through the following commands, make sure that the `{dplyr}` and `{summarytools}` packages are loaded (see above). We could do this with the following commands in R.[5]

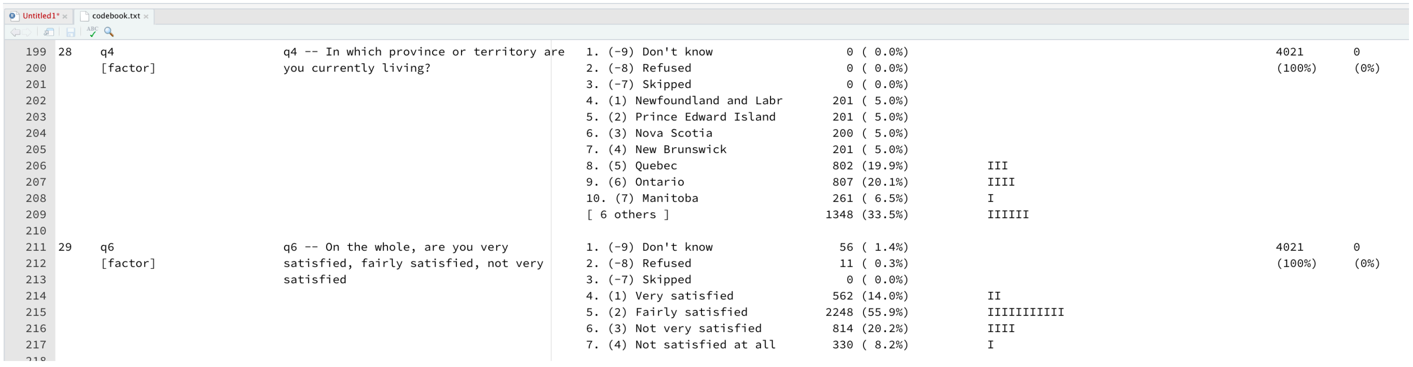
```
# enable the two packages needed if not already enabled (see above)
library(dplyr)
library(summarytools)
# change all haven_labelled variables to factors
CES2019b <- mutate_if(CES2019, is.labelled, as_factor)
# send all of the subsequent output to the file "codebook.txt"
```

[4] We also recommend reviewing the technical documentation and codebooks available for download with the dataset. The latter provides a listing of the survey questions asked, their corresponding variable name, and the response categories.

[5] Recall that in the code, the # indicates a comment. The script assume you have named your data frame "CES2019". If you have used another name, you will need to edit the script accordingly.

```
sink("codebook.txt")
# print a summary of the dataset
dfSummary(CES2019b, plain.ascii=TRUE)
# print the output back to the console
sink()
```

The `mutate_if()` function is from the `{dplyr}` package and it changes the variables in a dataset if the variables meet some criterion. In R there are lots of `is.X()` functions that identify whether a variable or object has a certain property. We are using the `is.labelled()` function to figure out whether or not a variable has the class *haven_labelled*. If it does, we want to turn that variable into a factor with the `as_factor()` function, also from the `{haven}` package. The `sink(<filename>)` function sends the subsequent output to the file and continues to do that until you issue the command `sink()` (without the filename) to return printing to the console (i.e., the console window in RStudio). The code above will produce a file called "codebook.txt" visible under the Files tab of the lower-right tile. Clicking on this file name will display the summary results in the upper-left tile. You can scroll through these results to view information on all of the variables included in the 2019 CES. Here is an example of how a couple of lines of the summary look (note that you can expand each of the tile windows as needed to view the content):



```
28    q4               q4 -- In which province or territory are   1. (-9) Don't know               0 ( 0.0%)                       4021     0
      [factor]         you currently living?                      2. (-8) Refused                  0 ( 0.0%)                       (100%)   (0%)
                                                                  3. (-7) Skipped                  0 ( 0.0%)
                                                                  4. (1) Newfoundland and Labr   201 ( 5.0%)
                                                                  5. (2) Prince Edward Island    201 ( 5.0%)
                                                                  6. (3) Nova Scotia             200 ( 5.0%)
                                                                  7. (4) New Brunswick           201 ( 5.0%)
                                                                  8. (5) Quebec                  802 (19.9%)    III
                                                                  9. (6) Ontario                 807 (20.1%)    IIII
                                                                  10. (7) Manitoba               261 ( 6.5%)    I
                                                                  [ 6 others ]                  1348 (33.5%)    IIIIII

29    q6               q6 -- On the whole, are you very           1. (-9) Don't know              56 ( 1.4%)                       4021     0
      [factor]         satisfied, fairly satisfied, not very      2. (-8) Refused                 11 ( 0.3%)                       (100%)   (0%)
                       satisfied                                  3. (-7) Skipped                  0 ( 0.0%)
                                                                  4. (1) Very satisfied          562 (14.0%)    II
                                                                  5. (2) Fairly satisfied       2248 (55.9%)    IIIIIIIIIII
                                                                  6. (3) Not very satisfied      814 (20.2%)    IIII
                                                                  7. (4) Not satisfied at all    330 ( 8.2%)    I
```

From the summary results, you can view the variable name, variable label, frequency distribution and graph as well as valid and missing observations for each variable in the dataset. For a very large dataset such as the CES, the information provided can be overwhelming, given the number of variables in the 2019 CES dataset. One solution is to limit the results by listing only the variables you wish to explore after the command. To do this, you need to find the variable names. You then simply list these after the codebook command.

> **Tip**: You can search for variables containing key words in the dataset by using the function `searchVarLabels()` from the `{DAMisc}` package. Running the following in the console or your script file will enable the {DAMisc} package that can then be used to generate a list of all variables that include the word "vote" in the variable name or label:

```
# enable the package needed if not already enabled (see above)
library(DAMisc)

# search the CES2019 data frame for any labels that contain the
word "vote"
searchVarLabels(CES2019, "vote")
```

To practice, let's look at two additional variables in the dataset. The first, *q6*, reports the level of satisfaction with the way democracy works in Canada. The second, *p3*, reports the respondents' vote choice in the 2019

Canadian federal election. In your script file, add the following commands:

```
# enable the packages needed if not already enabled (see above)
library(summarytools)
library(dplyr)


CES2019b %>%
select(q6, p3) %>%
mutate_if(is.labelled, as.factor) %>%
dfSummary(., plain.ascii=TRUE)
```

There are a couple of additional things here that we haven't seen already. The first is the pipe operator `%>%`. This operator takes whatever is on the left side of it and sends it to the function on the right side of it. Another new function is `select()` from the `{dplyr}` package – this function selects a subset of variables from the dataset, here q6 and p3. Then this dataset with two variables is passed to the `mutate_if()` function to turn labelled variables into factors and then that dataset with factors is sent to the `dfSummary()` function. Note that the first argument of `dfSummary()` is a dataset. Whenever you are using the pipe character, the result being passed to the right side of the pipe can always be accessed with the period. That's why the period is the first argument to the `dfSummary()` function.

To execute the commands, block (select) the lines and then hit the "Run" button at the top-right corner of the script window. (Reminder: you can also use the short-cut keys: command + return on a Mac or CTRL + Enter with a Windows operating system.) Compare your results with ours, reported below.

```
Data Frame Summary
CES2019b
Dimensions: 4021 x 2
Duplicates: 3972

----------------------------------------------------------------------------------------------------------------------------------------------------
No   Variable   Label                                  Stats / Values                   Freqs (% of Valid)  Graph            Valid      Missing
---- ---------- -------------------------------------- -------------------------------- ------------------- ---------------- ---------- ----------
1    q6         q6 -- On the whole, are you very       1. (-9) Don't know                 56 ( 1.4%)                         4021       0
     [factor]   satisfied, fairly satisfied, not very  2. (-8) Refused                    11 ( 0.3%)                         (100%)     (0%)
                satisfied                              3. (-7) Skipped                     0 ( 0.0%)
                                                       4. (1) Very satisfied             562 (14.0%)        II
                                                       5. (2) Fairly satisfied          2248 (55.9%)        IIIIIIIIIII
                                                       6. (3) Not very satisfied         814 (20.2%)        IIII
                                                       7. (4) Not satisfied at all       330 ( 8.2%)        I

2    p3         p3 -- Which party did you vote for?    1. (-9) Don't know                  8 ( 0.3%)                         2700       1321
     [factor]                                          2. (-8) Refused                   211 ( 7.8%)        I                (67.15%)   (32.85%)
                                                       3. (-7) Skipped                     0 ( 0.0%)
                                                       4. (1) Liberal Party              788 (29.2%)        IIIII
                                                       5. (2) Conservative Party         769 (28.5%)        IIIII
                                                       6. (3) NDP                        460 (17.0%)        III
                                                       7. (4) Bloc Québécois             139 ( 5.1%)        I
                                                       8. (5) Green Party                263 ( 9.7%)        I
                                                       9. (6) People's Party              42 ( 1.6%)
                                                       10. (7) Other                      14 ( 0.5%)
                                                       11. (8) Respondent cast inval       6 ( 0.2%)
----------------------------------------------------------------------------------------------------------------------------------------------------
```

The results above provide a wealth of information. For example, looking at the results for *q6*, we find the variable name (*q6*), the variable label (q6 -- On the whole, are you very satisfied, fairly satisfied, not very satisfied), the range of values, the frequency distribution (raw and relative frequencies) and the the number of missing (0) and valid (4021) cases. There is also a bar chart that plots the frequency distribution.

Other commands listed above can provide subsets of this information, for example, inspect () will report the variable name, variable and value labels, and the frequency distribution (raw and relative frequencies). Try this for yourself:

1. In your script file, type the following:
   ```
   library(DAMisc)
   inspect(CES2019b, "p3")
   inspect(CES2019b, "q6")
   ```
2. Select the line and then click on the "run" icon in the top right corner of the script file screen. (Reminder: you can also use the short-cut keys: Command + return on a Mac or CTRL + Enter with a Windows operating system.)
3. Compare your `inspect()` results with your `dfSummary()` results.
4. The `sumStats()` function from the `{DAMisc}` package has the same required arguments (syntax) as the `inspect()` function – `sumStats(data, variable-name).` Use it to get the summary statistics for the p3 and q6 variables.

Check- In Point

If you are working alongside us, at this point you have used a number of basic commands to examine two variables in the dataset. Before you move forward, be sure to practice these skills: Use the `searchVarLabels()` function to identify variables in an area of interest to you. Once you have the variable names, experiment with different commands, using your script file.

Part III: Applying Survey Weights

Destination

By the end of this section, you will be able to:

- *explain why survey weights are often used in analysis.*

As we discuss in Chapter 5, when researchers sample from populations, they often over-sample certain population segments and then create design weights to adjust for over or under representation of certain segments of the population. When you use secondary survey datasets, be sure to consult the metadata (technical documentation, as discussed in Chapter 8) to review information on the sampling procedures and weight variables.

The 2019 CES employs a disproportionate random sampling technique that oversamples in some areas of the country, such as Quebec, while under sampling in others. The CES dataset includes two weight variables, weight_CES and weight_PES, to account for provincial over/under sampling as well as phone ownership (landline and/or cell phone. See CES technical documentation for more details). The former weights according to the full sample and the latter is a weight based on only those respondents who completed both waves of the survey (campaign period and post-election). We use the full sample weight (weight_CES) in our analyses.

The `{survey}` package in R allows you to incorporate weights into many common statistical routines.

The main method for doing this is by defining a survey design object that will get used as the data. For example, with the 2019 CES, using the "weight_CES" variable, we would do the following:

```
library(survey)
ces_svy <- svydesign(ids=~1, strata=NULL, weights=~weight_CES,
     data=CES2019, digits=3)
```

Then, for some commands, instead of using the "CES2019" data object as the source of the data, we will use the "ces_svy" object. Unlike other programs, such as Stata, R only has one way of including weights generally (which is similar to the ***svyset*** in Stata). R also does not make the distinction between analytical weights and probability weights, as Stata does, but in different settings it provides results that are equivalent to either probability weights or analytical weights. We include the syntax to weight the CES data in the examples in the following sections where possible. As you work through this handbook, be sure to reflect upon how the use of survey weights affects the results.

Check-In Point

Survey weights can be a challenging idea for many new researchers. Before you move forward, ensure that you are comfortable with your understandings. Why do researchers use survey weights? How can you as a user of secondary survey datasets determine how the original researchers constructed their survey weights? We encourage you to review both Chapter 5 and Chapter 8 of the *Explorations* textbook before moving forward to the next section.

Part IV: Examining Frequency Distributions and Univariate Statistics

Destination

By the end of this section, you will be able to:

- *generate frequency distributions;*
- *apply survey weights; and*
- *generate univariate statistics.*

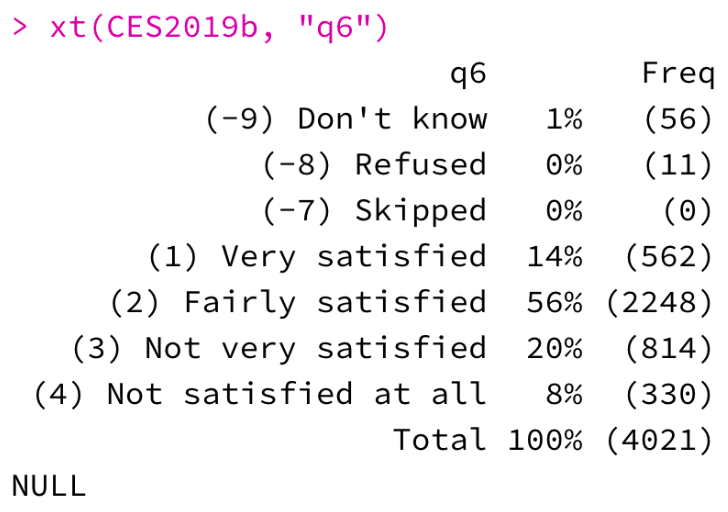
In Chapter 12, we discuss how researchers start their analyses by examining the frequency distributions and summary statistics for each individual variable in their analysis. These can be generated in several ways in R. The way that will be most useful for us, particularly in terms of pivoting between weighted and unweighted data, is the `xt()` function from the `{DAMisc}` package:

```
xt(data, row-variable)
```

Try this for yourself to generate a frequency table for the satisfaction with the way democracy works in Canada variable:

1. In your script file, type the following: `xt(CES2019b, "q6")`
2. Select the line and then click on the "run" icon in the top right corner of the script file screen. (Reminder: you can also use the short-cut keys: Command + return on a Mac or CTRL + Enter with a Windows operating system.)

3. Compare your results with the results displayed below.



```
> xt(CES2019b, "q6")
                            q6        Freq
              (-9) Don't know   1%    (56)
                 (-8) Refused   0%    (11)
                 (-7) Skipped   0%     (0)
           (1) Very satisfied  14%   (562)
         (2) Fairly satisfied  56%  (2248)
       (3) Not very satisfied  20%   (814)
     (4) Not satisfied at all   8%   (330)
                         Total 100%  (4021)
NULL
```

Note that the results include the raw frequency and relative frequency.

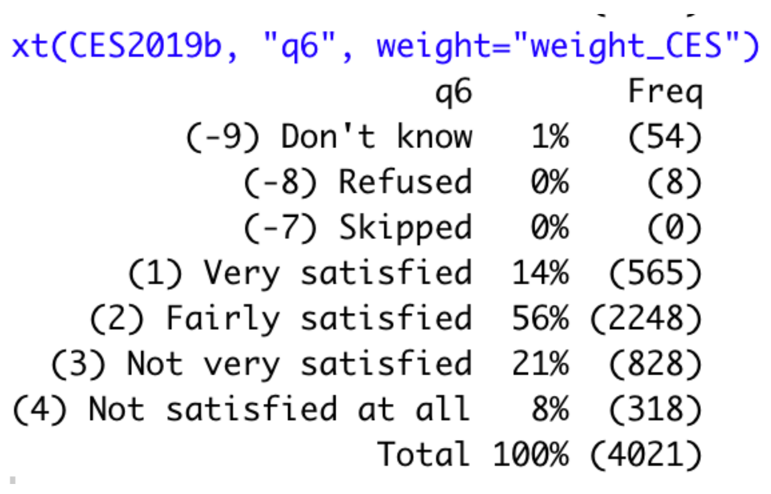
In the previous section, we discussed survey weights, and noted that to apply weights we simply generate a survey design object and then use that as input to the function. Let's do this now, re-running our function to account for the disproportionate random sample by weighting the data.

1. In your Script file, type the following:

   `xt(CES2019b, "q6", weight="weight_CES")`

2. Select the line and then click on the "run" icon in the top right corner of the script file screen. (Reminder: you can also use the short-cut keys: command + return on a Mac or CTRL + Enter with a Windows operating system.)

3. Compare your results with the results displayed below, and with your original results. Note how the addition of the weight variable affects the results.

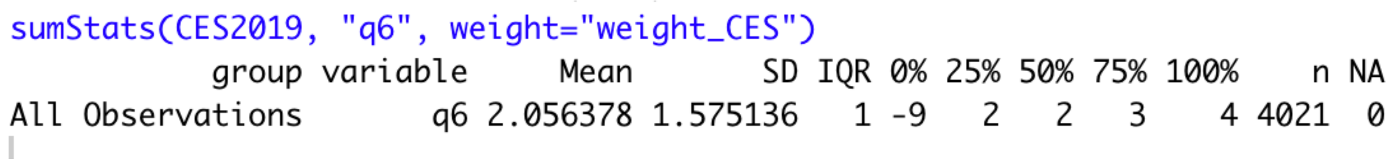


```
xt(CES2019b, "q6", weight="weight_CES")
                            q6        Freq
              (-9) Don't know   1%    (54)
                 (-8) Refused   0%     (8)
                 (-7) Skipped   0%     (0)
           (1) Very satisfied  14%   (565)
         (2) Fairly satisfied  56%  (2248)
       (3) Not very satisfied  21%   (828)
     (4) Not satisfied at all   8%   (318)
                         Total 100%  (4021)
```

In examining a variable, you will also want to consider the appropriate measures of central tendency and dispersion. (Review Chapter 12 if you need refreshing on the appropriate measures of central tendency and dispersion by variable level.) As noted earlier, the frequency distribution results include the raw frequency and the relative frequency. This is an ordinal variable, and from these results you can visually identify the appropriate measure of central tendency (median) and dispersion (range). To move beyond a visual assessment, you can use a related R command that allows you to specify the summary statistics that you wish to view. Note that R does not include the mode or the variation ratio as summary statistics. Fortunately, both are easily identified with the information reported in the frequency distribution table (again, see Chapter 12).

For example, to obtain the median and range for this variable we would run the following command to produce the results reported below:

1. In your script file, type the following:
   ```
   sumStats(CES2019, "q6", weight="weight_CES")
   ```
2. Select the line and then click on the "run" icon in the top right corner of the script file screen. (Reminder: you can also use the short-cut keys: Command + return on a Mac or CTRL + Enter with a Windows operating system.)
3. Compare your results with the results displayed below.



```
sumStats(CES2019, "q6", weight="weight_CES")
           group variable     Mean       SD IQR 0% 25% 50% 75% 100%    n NA
All Observations       q6 2.056378 1.575136   1 -9   2   2   3    4 4021  0
```

Let's interpret these univariate statistics, starting with the median. R reports lots of things, but the two we are interested in for this example are the median and the range. The former is reported as 50% (the 50th percentile), which is reported as "2". Note that these are the actual values for the variable, not the value labels as reported in the frequency distribution. To determine the value labels associated with these values, you could use the following command:

```
attr(CES2019$q6, "labels")
```

Based on the results, we see that the value label for 2 is "Fairly satisfied". Recall from Chapter 12 that the range is estimated by subtracting the lowest value from the highest value. We could calculate the range as the difference between the 100% value (4) and the 0% value (-9). In this case, the range is equal to 13. But how do we interpret this? When you look at the range result, the number should strike you as a bit curious – how does a variable with four possible response categories have a range of 13? When you see results like this, you should always ask questions and seek out the answer. In this case, the answer lies with the coding. In the CES dataset "don't know" is coded as -9 (see above). Thus, R produced the range as (4)-(-9) for a range of 13. In this example, some recoding is necessary to estimate the range. We will return to this topic shortly; for now, simply know that it is always important to critically assess your results, as statistical software will not catch such issues for you!

Check-In Point

At this point, you should understand how to use the R `xt()` and `sumStats()` functions to generate frequency distributions and univariate statistics (specifically the median and the range). You should also be able to add syntax to your command to apply survey weights. Before moving forward, be sure to practice these skills with other variables, using the codebook to help interpret categories and to identify curious results that may reflect coding. Be sure as well to continue to compare how results change with the addition of the survey weight.

Part V: Creating and Recoding Variables

Destination

By the end of this section, you will be able to:

- *explain why you should never recode original variables;*
- *create new variables;*
- *recode variables; and*
- *rename variables and add or alter variable labels.*

As we have already observed, it is often necessary to recode variables before you can work them. As a rule, we recommend never altering original variables within a dataset. We will repeat this, in case you are reading quickly: **never alter original variables in a dataset**. Instead, you should generate a new variable from the original and then make the transformations you need to your new variable. There are two reasons for this: (1) it allows you to check your work by comparing the recoded variable against the original one, and (2) maintaining the original variable allows you to use a different transformation processes if you need to do so at a later time.

Creating New Variables

To create a new variable, we use the `mutate()` function from the `{dplyr}` package. We select a new variable name for the new variable, and then command R to create a new variable from the existing variable. Let's consider this with variable *q6* from the 2019 CES. You are going to create a new variable named *satdemocracy* (remember that the variable looks at satisfaction with democracy).

1. In your Script file, type the following: `CES2019 <- mutate(CES2019, satdemocracy=q6)`
2. Select the line and then click on the "run" icon in the top right corner of the script file screen. (Reminder: you can also use the short-cut keys: command + return on a Mac or CTRL + ENTER with a Windows operating system.)

When you generate a new variable, you can confirm your work by comparing the original variable and the new variable in a cross-tabulation (we discuss cross-tabulation in more detail below). Note that using a cross-tabulation to check your recoding is only advisable when working with nominal or ordinal level variables. For interval/ratio level variables, we use a frequency distribution, as demonstrated below. To produce this cross-tabulation, you can use the `table()` function:

1. In your Script file, type the following: `with(CES2019, table(q6, satdemocracy))`[6]
2. Select the line and then click on the "run" icon in the top right corner of the Script file screen. (Reminder: you can also use the short-cut keys: command + return on a Mac or CTRL + ENTER with

[6] Because R allows you to have multiple data frames in memory at once, unlike Stata and SPSS, you need to identify in R functions where to find the variables (i.e., which data frame the variables are in). You can do this in several ways. In many functions, there is a "data" argument, where you could say, for example, data=CES2019. The table() function doesn't have such an argument, so we used the with() function which takes the following form: with(data, function()) where this forces function() to use the designated data file. The other option is to use the dollar sign $ to extract a variable from a dataset. We could have done the same thing above with: table(CES2019$q6, CES2019$satdem).

a Windows operating system.)

3. Look at the intersection of the original values in the row and the new values in the column.

Because we didn't change the values of the variable, we simply copied it, we should only see observations in the cells where the old and new values are the same (e.g., the -9 row and the -9 column). The "satdemocracy" variable will have all of the same properties as the "q6" variable.

There are a number of more advanced options for the `mutate()` function, including the option of combining multiple variables to generate a single measure. This is part of a host of mathematical calculations that can be used with `mutate()`. We do not cover these more advanced procedures in this introductory handbook but encourage interested users to seek out additional information on the various expressions that can be used with `mutate()` (along with other ways that the `mutate()` function can be used**)** via the R help files.

Recoding Variables

Once you have created the new variable, you can begin making transformations to meet your research needs. For example, let's say you want to transform *satdemocracy* to remove cases that report "Don't know" or "Refused". Recall from the last section that the range for *q6* was nonsensical given the inclusion of these responses, which are coded as -9 and -8, respectively. Given that we cannot be sure how individuals who answered "Don't know" or "Refused" feel about the way democracy works in Canada, we want to exclude these cases from our new variable for our analysis.

In R, missing cases are denoted as `NA`. The way that recoding works in R depends on the type of variable you're working with. Recall that we made two different versions of our dataset. The *CES2019* data frame has only numeric values and the *CES2019b* data frame converted anything with variable labels to factors. In Stata and SPSS, variables with labels are still treated as numeric and the numeric information in the variable is preserved. This is not the case for factors in R – R treats these variables more like words than numbers. First, we will discuss how to recode the -9 and -8 values to missing. For this part of the analysis, we will use the `recode()` function from the `{car}` package.[7] We can do this for individual values (option A), we can use a range instead of noting each value separately (option B) or we can specify a set of values to all be recoded as the same value (option C):

Option A:
```
CES2019 <- mutate(CES2019,
          satdemocracy = car::recode(q6, "-9=NA; -8=NA"))
```

Option B:
```
CES2019 <- mutate(CES2019,
          satdemocracy = car::recode(q6, "-9:-8=NA"))
```

Option C:
```
CES2019 <- mutate(CES2019,
          satdemocracy = car::recode(q6, "c(-9,-8)=NA"))
```

Not that in all cases, the recode() syntax has the following characteristics. First, all recodes are inside a single set of quotation marks. Each different recode statement, within the one single set of quotation marks, is separated by a semi-colon (;). The range operator used in option B is the colon (:). It indicates every value

[7] One of the down-sides of having many of the useful functions in R being contributed in packages by developers is that sometimes two packages have the same function in them. This happens here as there are recode() functions in both the {car} package (which is the one we want to use) and the {dplyr} package, which we loaded to use the mutate() function. These two functions work differently. You can ensure you're using the right function by executing it with the following convention: package::function(), in this case it would be car::recode().

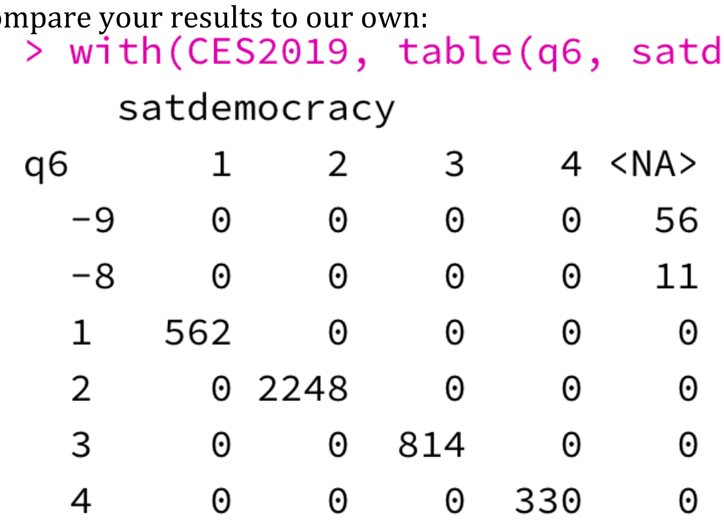
between (and including) the two end points. The `c()` function used in option C is the concatenate operator. It allows us to specify a single object with multiple values. Whereas option B includes all values between (and including) -9 and -8, option C only includes the integer values -9 and -8, but nothing in between.

Recode your missing values now with any of the options above by entering the function into your script file, selecting the text, and clicking "run." It does not matter which option you select; there are often a number of ways to achieve the same outcome in R. We outline some of the more simplistic procedures within this handbook but recognize that there are other commands that you may use to achieve the same results.

With the newly recoded variable, you should confirm that you have not made any errors. This can be done by comparing the new variable to the original variable the same way we did above. To include the missing cases (those cases you set to `NA`) you add the `useNA="ifany"` argument to the table function as below:

```
with(CES2019, table(q6, satdemocracy, useNA="ifany"))
```

Compare your results to our own:



```
> with(CES2019, table(q6, satdemocracy, useNA="ifany"))
    satdemocracy
q6      1    2    3    4 <NA>
  -9    0    0    0    0   56
  -8    0    0    0    0   11
  1   562    0    0    0    0
  2     0 2248    0    0    0
  3     0    0  814    0    0
  4     0    0    0  330    0
```

From the results, we can confirm that we have not made any errors. Notice that for the "q6" rows -9 and -8, we see that their values are now in the `<NA>` column for "satdemocracy".

Things get a bit more complex when we are trying to recode the factor variable in the *CES2019b* data frame. Here, the range operator in Option B above doesn't work, but options A and C still work. Here is how we would do this.

Option A:
```
CES2019b <- mutate(CES2019b, satdemocracy = car::recode(q6,
          "\"(-9) Don't know\" = NA;
          '(-8) Refused' = NA"))
```

Option C:
```
CES2019b <- mutate(CES2019b, satdemocracy = car::recode(q6,
          "c(\"(-9) Don't know\",  '(-8) Refused')=NA"))
```

The reason things are a bit more complicated here is that we have to put all recode statements in a set of quotation marks (they could be either double quotes or single quotes). Then, we have to identify the values we want to recode as strings, too. So, if we used double quotes to encapsulate the entire set of recode statements, we could identify specific values with single quotes. The problem here (and you will run into it a lot in survey data) is that the word "don't" has an apostrophe in it, which is the same as a single quotation mark. So, you'll notice in the code above, we wrap the entire set of recode statements in double quotes and we wrap (-9) Don't Know in what we would call escaped quote marks. The escape character (\) here makes it so that these quotation marks don't close the double quote mark that started the string and they ensure that the apostrophe in Don't isn't recognized as an open single quote. These are kind of esoteric concerns and it may not be completely clear how it works, but if you follow the formula above, you should do the right thing.

You can compare your results to those below:

```
> table(CES2019b$q6, CES2019b$satdemocracy, useNA="ifany")

                          (1) Very satisfied (2) Fairly satisfied (3) Not very satisfied (4) Not satisfied at all <NA>
  (-9) Don't know                          0                    0                      0                        0   56
  (-8) Refused                             0                    0                      0                        0   11
  (-7) Skipped                             0                    0                      0                        0    0
  (1) Very satisfied                     562                    0                      0                        0    0
  (2) Fairly satisfied                     0                 2248                      0                        0    0
  (3) Not very satisfied                   0                    0                    814                        0    0
  (4) Not satisfied at all                 0                    0                      0                      330    0
```

You'll notice the same thing as above; the -9 and -8 original values now only have values in the `<NA>` column for "satdemocracy"

What would you do if you discovered that you made a mistake? (For example, say you accidently set the value of 4 as missing, when it is in fact not missing). First, you would congratulate yourself for creating new variables rather than transforming an original variable, as your problem can be easily fixed. Second, you can simply delete a mis-transformed variable in R using the `select()` function and then re-create it with the corrected syntax. For the example above, you would use the following syntax to delete the *satdemocracy* variable:

```
CES2019 <- select(CES2019, -satdemocracy)
```

(Please don't actually do this – we are going to continue working with *satdemocracy!*)

Renaming variables and adding or altering value labels

The majority of the variables in the 2019 CES dataset include variable and value labels – descriptions of each variable, and descriptions of the values for categorical variables. However, for any new variables that you generate, you will need to either create or modify variable names and labels. The most useful commands for doing so are listed below.

To change the name of an existing variable:

```
data <- rename(data,  "new_varname" = "old_varname")
```

To add variable and value labels, you must define the labels, and then attach those labels to the variable:

```
data <- mutate(data,
        varname = labelled(varname,
                labels=c("label" = #, "label" = #),
        label="Descriptive Variable Label"))
```

For example, we can rename our newly generated variable, *satdemocracy*, and add the appropriate variable and value labels:

```
          CES2019 <- rename(CES2019, satdem = satdemocracy)
```

Next, we will add variable and value labels:

```
CES2019 <- mutate(CES2019, satdem = labelled(satdem,
     labels=c("Very Satisfied" = 1, "Fairly Satisfied" = 2,
              "Not Very Satisfied" = 3, "Not Satisfied at All" = 4),
     label = "Satisfaction with Democracy"))
```

To "activate" the labels (i.e., to have them show up when you make a table of the variable, you would have to turn it into a factor with the `as_factor()` function as follows:

```
     CES2019 <- mutate(CES2019, satdem = as_factor(satdem))
```

You should once again check your work by comparing the new variable against the original variable.

```
with(CES2019, table(q6, satdem, useNA="ifany"))
```

The exclusion of respondents who answered "don't know" or "refused" in the new variable will change things. Repeat the steps outlined in Part IV to see how the removal of these response categories affects the frequency distribution and summary statistics.

Check-In Point

We covered a bit of ground in this section: you now know the reasons why you should never recode original variables (and thus will avoid future despair when discovering recoding errors). You know how to create new variables, and then recode, rename, and add variable and value labels to those new variables. These are likely to be some of the most frequently used procedures when working with data.

Part VI: Creating Bar Graphs and Pie Charts

Destination

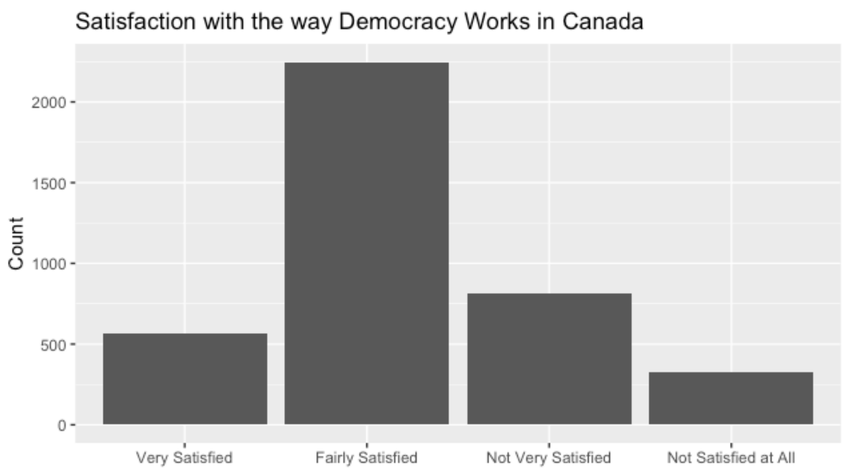
By the end of this section, you will be able to:

- *create bar graphs and pie charts.*

R offers a range of graphing options. There are several different graphing packages, including R's base graphics, `{lattice}` and `{ggplot2}`. While the numerous options available for graphing within R are beyond the scope of this handbook, we do outline the steps to create basic graphs and charts that you may use to display univariate frequency data.

Let's report the frequency distribution of our recoded satisfaction with democracy in a bar chart with the following syntax to produce the graph below (the graph is visible in the Plots tab of the lower-right tile):

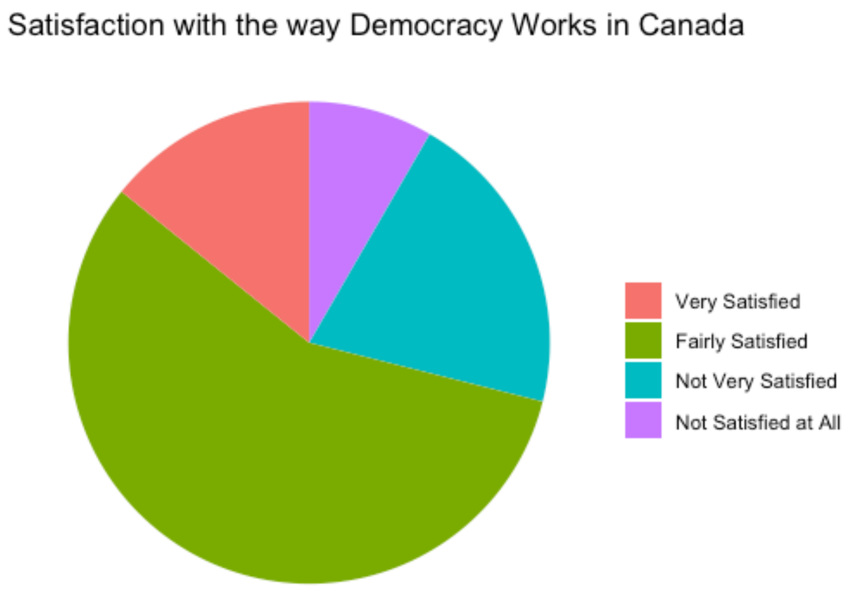
```
filter(CES2019, !is.na(satdem)) %>%
ggplot(aes(x=satdem)) +
  geom_bar() +
  labs(x = "", y="Count") +
  ggtitle("Satisfaction with the way Democracy Works in Canada")
```



The first line of the functions above filters out the missing data on the "satdem" variable. The second line initializes a plot and sets the x variable to "satdem". The third line adds bars for the satdem variable. The fourth line sets the x-axis label to blank and the y-axis label to "Count". The final line adds the title to the top of the figure.

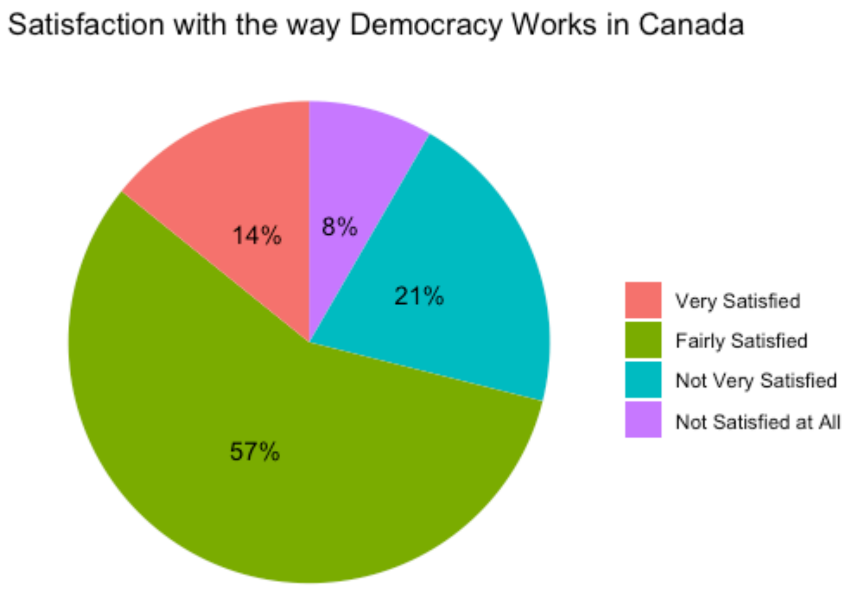
There is no pie chart function natively in `{ggplot2}`, so we will use the `ggpie()` function in the `{DAMisc}` package. The ggpie() function returns a ggplot to which other elements can be added as below.

```
ggpie(CES2019, "satdem", addPct="none") +
  labs(fill="") +
  ggtitle("Satisfaction with the way Democracy Works in Canada")
```



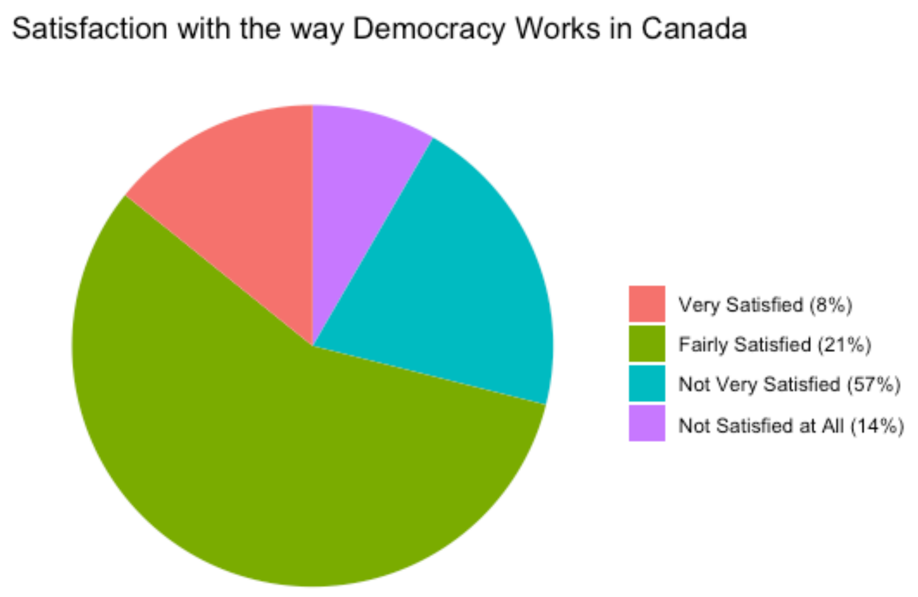
You will notice that these graphs include the title that you gave the functions above and the variable's value labels. We can include additional information in the graph by adding additional instructions in the syntax. Let's add the percentage of the sample within each category to the pie chart with the following syntax:

```
ggpie(CES2019, "satdem", addPct="pie") +
  labs(fill="") +
  ggtitle("Satisfaction with the way Democracy Works in Canada")
```



You could also add the percentages to the legend, in the event that some of the pie pieces are too small:

```
ggpie(CES2019, "satdem", addPct="legend") +
  labs(fill="") +
  ggtitle("Satisfaction with the way Democracy Works in Canada")
```



Since the graph is a ggplot, you can modify it by adding other elements to it. The document for the ggplot package should give you some ideas about other things you could do.

Check-In Point

Congratulations – you are now able to create basic univariate graphs. If you have an interest in more advanced graphing features, be sure to review the `{ggplot2}` documentation and other online resources.

Part VII: Comparing Two Independent Samples

Destination

By the end of this section, you will be able to:

- *use a t-test to assess differences of means between two independent samples.*

In Chapter 13, we consider how we often wish to compare the means of two independent groups to see if they differ. To assess differences of means between two groups, we can use a t-test. To do this, we need a variable with our two groups of interest (for example, a treatment group and a control group from an experimental study) and an interval/ratio variable for which we expect a difference between the two groups. (Recall that means should only be used with interval/ratio variables.)

For example, let's compare the average income (interval/ratio variable) of men and women (dichotomous variable).[8] We used the following syntax to find, examine, recode, and check the variables that we will use for this analysis:

```
#Use the searchVarLabels() command to find the gender variable
searchVarLabels(CES2019, "gender")

#Look at the value labels and distribution of the original
variable

inspect(CES2019, "q3")
```

[8] Note that the CES asks about household income. We have used this as a proxy for personal income in this example.

```
*Generate, recode and label the new variable

CES2019 <- mutate(CES2019, gender = car::recode(q3,
              "1=1; 2=2; else=NA"))

CES2019 <- mutate(CES2019, gender = labelled(gender,
              labels=c("Male" = 1, "Female" = 2),
              label = "Dichotomous gender variable"))

CES2019 <- mutate(CES2019, gender = as_factor(gender))


#Check original and new variable distributions

with(CES2019, table(q3, gender, useNA="ifany"))

#Use the searchVarLabels() function  to find the income
variable

searchVarLabels(CES2019, "income")

# Look at the value labels and distribution of the original
variable

inspect(CES2019, "q69")

# Generate, recode and label the new variable (note that we
have added "IR" to the end of the new variable name to indicate
interval/ratio variable)

CES2019 <- mutate(CES2019, incomeIR = car::recode(q69,
              "c(-9,-8) = NA"))
attr(CES2019$incomeIR, "label") <-
          "Household income in dollars"


# Check original and new variable distributions
with(filter(CES2019, is.na(incomeIR)), table(q69))
```

The last command above is just finding the values that are missing in the “incomeIR” variable and then making a frequency distribution of the “q69” variable for those observations that are missing on “incomeIR”.

R has a built-in t-test function called `t.test()`. However, it doesn’t give quite as much information as you like, so we’ll use the `tTest()` function in the `{DAMisc}` package as follows:

```
tTest("gender", "incomeIR", CES2019)
```

```
> tTest("gender", "incomeIR", CES2019)
Summary:
           mean     n    se
Male       112299.4 1790 127079
Female     94962.62 1284 100085
Difference 17336.82 3074 4101.615
p-value < 0.001
---------------------------------

        Welch Two Sample t-test

data:  incomeIR by gender
t = 4.2268, df = 3045.4, p-value = 2.44e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  9294.61 25379.04
sample estimates:
  mean in group Male mean in group Female
           112299.44             94962.62
```

Do the incomes of men and women differ? The results suggest that they do. From the table above we see that the mean income for men is $112,299.40 compared to a mean income of $94,962.62 for women, a difference of $17,336.82 (the value reported as "Difference" in the Summary table). In other words, the results indicate that, on average, men earn $17,336.82 a year more than women. We provide a more detailed discussion of this test and the results reported here in the text (see Chapter 13).

In Chapter 13, we also discuss statistical significance. You will recall that researchers use a one-tailed test when they hypothesize a specific direction to a relationship and use a two-tailed test when they do not hypothesize a direction. The R `tTest()` function by default reports the two-tailed hypothesis. If you wanted a one-tailed test where you thought the difference was greater than zero, you could add an additional argument to the `tTest()` function, `alternative="greater"`. You could use `alternative="less"` for the other one-tailed hypothesis. Given that we did not make any assumptions about the direction of the relationship in advance, we would use the results from the default two-tailed test, a result that indicates a statistically significant relationship at p<0.001. R reports the p-value as 2.44e-05. This scientific notation means that the p-value is 2.44 with the decimal point moved five places to the left, so 0.0000244.

Check-In Point
If you are following along with the examples by running your own data, you should now be able to use a t-test to assess differences of means between two independent samples. This is a useful skill, and we encourage you to practice by exploring other income differences between other dichotomous groups in the CES dataset. For example, do the average incomes of university and non-university graduates differ?

Part VIII: Examining Bivariate Relationships for Nominal and/or Ordinal Variables

Destination

By the end of this section, you will be able to:

- *create crosstabulations;*

- *calculate measures of association; and*
- *calculate Chi Square.*

In Chapter 12, we introduce you to the four questions we must answer when assessing whether there is a relationship between two variables:

1. What is the form/direction of the relationship?
2. How strong is the relationship?
3. Is the relationship statistically significant?
4. What happens to the relationship when we control for other variables?

In this section, we will focus on how to use R to help answer the first three of these questions for nominal and/or ordinal variables. In the section that follows, we will look at interval/ratio variables. The final section of this handbook will consider the fourth and final question.

To examine the relationship between two nominal and/or ordinal variables, we create a cross-tabulation (contingency table), calculate the appropriate measure of association, and calculate the appropriate inferential statistic. With R, we can do all of this with just a couple of functions.

For example, let's test the hypothesis that those with higher levels of income will also be more interested in politics. Before we can do so, we must recode our variables. For this example, we will recode our dependent variable (DV), political interest, into a three-point measure ranging from low to high (terciles), as follows[9]:

```
# Use the searchVarLabels function to find the interest
variable

searchVarLabels(CES2019, "interest")

#Look at the value labels and distribution of the original
variable

inspect(CES2019, "q9")

# Generate, recode and label the new variable into terciles
(note that the tercile divisions are based on the cumulative
frequency)

CES2019 <- mutate(CES2019,
     polinterest = car::recode(q9, "c(-9,-8) = NA"),
     polinterest = binVar(polinterest, 3,
          method="proportions", labels=c("Low interest",
          "Middle interest", "High interest")))

# Check original and new variable distributions
```

[9] An alternative way to recode this variable would be to set respondents that choose 0-4 as low interest, 5 as the mid-point, and all responses over 5 as high interest. We opt to use the cut-off points based on the cumulative frequency to produce three roughly equal sized groups.

```
          with(CES2019, table(q9, polinterest, useNA="ifany"))
```

In the code above, we first use the `recode()` function from the `{car}` package to change -9 and -8 to `NA`. Next, we use the `binVar()` function from the `{DAMisc}` package to cut our continuous variable into three roughly equally sized groups.

We also will recode our independent variable (IV), income, into terciles (low, middle, and high income). It is important to note that many respondents (approximately 25% in the 2019 CES) did not report their actual income. To help reduce the number of non-responses, individuals who refuse or report that they do not know their actual income are asked a follow-up question that provides income categories for the respondent to choose instead of stating their actual income. In this example, we combine the responses from the two questions to generate a new income category variable. Note that we use the income variable that we generated for the ttest above in this example:

```
# Use the lookfor command to find the interest variable

searchVarLabels(CES2019, "income")

#Look at the value labels and distribution of the original variable

inspect(CES2019, "income")

#Generate, recode and label the new variable into terciles (note that
this combines the two income measures included in the CES)

CES2019 <- mutate(CES2019, incomegrptemp = car::recode(incomeIR,
                    "0=1; 1:30000=2; 30001:60000=3; 60001:90000=4;
                    90001:110000=5; 110001:150000=6;
                    150001:200000=7; 200000:hi=8; else=NA"))

attr(CES2019$incomegrptemp, "label") <- "Temp income grp var"

# Use the sumStats function to see what the range of income is
# for each of the new categories you generated to confirm your recoding

sumStats(CES2019, "incomeIR", byvar="incomegrptemp")

# Merge two income group variables

CES2019 <- mutate(CES2019,
    igtmp = car::recode(q70, "c(-8,-9) = NA"),
    incomegrpmerged = ifelse(is.na(incomegrptemp), igtmp, incomegrptemp),
    incomegrpmerged = labelled(incomegrpmerged, labels =
        attr(q70,"labels")))


names(attr(CES2019$incomegrpmerged, "labels"))[6] <- "(3) $30,001 to
$60,000"
```

```
CES2019$incomegrpmerged <- as_factor(CES2019$incomegrpmerged)

xt(CES2019, "incomegrpmerged")

# Create new income tercile variable

CES2019 <- mutate(CES2019, incometercile =
     binVar(as.numeric(incomegrpmerged), 3,
            labels=c("Low income", "Middle income", "High income")))

#Check original and new variable distributions

with(CES2019, table(incomegrpmerged, incometercile))
```

Note that we corrected an error in the value label by changing the names of the labels attribute for the variable. Also note that we used a conditional statement `ifelse()` when we combined two variables to create *incomegrpmerged*. This if-else statement tells R to evaluate the expression in its first argument, here `is.na(incomegrptemp)` is true if "incomegrptemp" is missing and false otherwise. The second argument tells R what to do if the expression is true. In this case, it replaces "incomegrpmerged" with the "q70" variable, which we had recoded directly above to remove -8 and -9. The third argument is what to do if the expression is false. In this case, that's using the "incomegrptemp" variable that we just defined. Use the help function to learn more about the `ifelse()` function.

With our variables prepared for analysis, we can now use the following syntax to test the relationship:

```
xt(CES2019, "polinterest", byvar="incometercile", weight="weight_CES")
```

Note that in creating crosstab tables, it is important to be clear on the order of your dependent and independent variables. We have followed the format DV in the rows and IV in the columns. This makes sense because the `xt()` function only produces column percentages. Unlike in Stata, we can use weights in producing cross-tab measures of fit and association. The way that it works is a bit different, though. The Chi-squared test that you will see is done properly through the `{survey}` package in R. The other measures of fit are calculated on the weighted contingency tables where each count is rounded to the nearest integer value. To recover the unweighted statistics, simply specify the `xt()` function without the weighting variable. This produced the following results:[10]

[10] Note that these results differ slightly from those reported in Chapter 13.

```
> xt(CES2019, "polinterest", byvar="incometercile", weight="weight_CES")
  polinterest/incometercile  Low income Middle income High income       Total
1              Low interest  35%  (482)    24%  (264)  23%  (281)  28% (1027)
2           Middle interest  31%  (428)    36%  (391)  35%  (432)  34% (1251)
3             High interest  34%  (477)    39%  (425)  43%  (529)  39% (1431)
4                     Total 100% (1387)   100% (1080) 100% (1242) 100% (3709)

        Pearson's X^2: Rao & Scott adjustment

data:  svychisq(as.formula(paste0("~", var, "+", byvar)), d)
F = 10.615, ndf = 3.9972, ddf = 16068.6819, p-value = 1.388e-08

Measures of Association
                       statistic p-value
Cramers V             0.08843254       0
Kruskal-Goodman Gamma 0.14109405       0
Tau-b                 0.09370769       0
```

Is there a relationship between income and political interest? Recall from Chapter 12 that the first step when assessing the results from a contingency table with two ordinal level variables is to look for a consistent increase/decrease in the percentage of respondents across categories of the IV in the top row and the opposite pattern in the bottom row. In this example, reading across the top row ("Low interest"), we find that the percentages decrease as we move from left to right (low to high income): high income earners are approximately twelve percentage points less likely to indicate low political interest compared to their low income counterparts. Looking at the bottom row ("High interest") we find the reverse pattern, with high income earners approximately 9 percentage points more likely to indicate high political interest relative to those in the low income category.

As noted in Chapter 12, our next step is to consider the correct correlation coefficient. Given that both variables are ordinal, we can assess the strength of the association by looking at the Gamma or the Tau value. Since gamma tends to inflate the strength of the relationship, we will opt for the more conservative Tau estimate. In this example, we find an extremely weak, positive (as income increases, political interest increases) relationship with a Tau value of 0.09.

Finally, recall from Chapter 13 that we can assess whether or not the relationship is statistically significant by looking at the Pearson chi^2 value. The results indicate that the relationship is statistically significant at $p<0.001$. As such, we would conclude that our results support our hypothesis: those with higher levels of income appear to be more interested in politics.

Check-In Point

This section covered an incredible amount of information – creating crosstabulations, how to calculate measures of association, and calculating Chi Square. This is powerful, but it is always critical to keep in mind that your decisions with respect to recoding have great influence on the results, so always check your recoding carefully before assessing relationships.

Part IX: Examining Bivariate Relationships between Interval/Ratio Variables

Destination

By the end of this section, you will be able to:

- *create scatterplots;*
- *calculate Pearson's Correlation Coefficient; and*
- *conduct basic linear regression.*

To assess the relationship between two continuous (interval/ratio) variables, we continue to ask the same four questions noted in the last section (and, of course, in Chapter 12), but we use different statistical techniques.

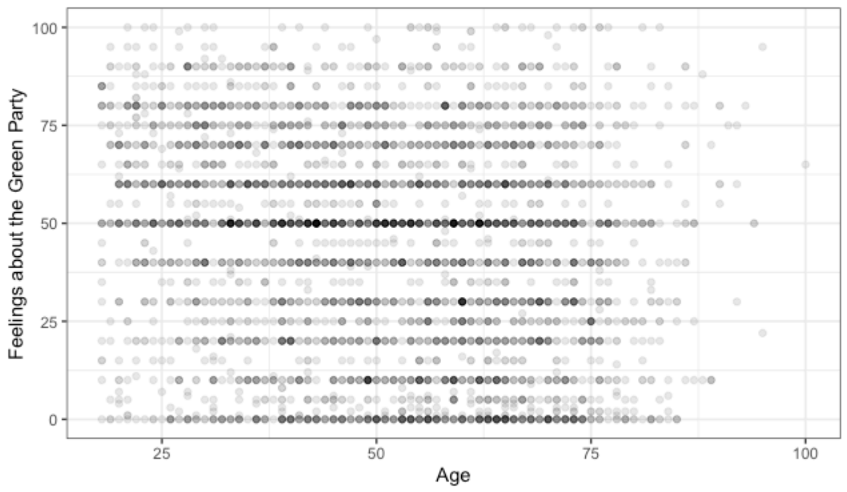
For example, we might theorize that younger individuals are more apt to like the Green party. To test this, we first recode the variables for analysis. To do this, you can use the following syntax:

```
# Use the searchVarLabels() function to find the interest variable

searchVarLabels(CES2019, "green")

#Look at the value labels and distribution of the original variable

inspect(CES2019, "q18")

# Generate, recode and label the new variable

CES2019 <- mutate(CES2019,
     greenfeelings = car::recode(q18, "-9:-6 = NA"))

attr(CES2019$greenfeelings, "label") <-
     "Feelings about the Green Party"


#Check original and new variable distributions

with(filter(CES2019, is.na(greenfeelings)), table(q18))

# Use the searchVarLabels() function to find the interest variable

searchVarLabels(CES2019, "age")

# Look at the value labels and distribution of the original variable

inspect(CES2019, "age")
```

NOTE: The age variable does not require recoding.

With our variables ready for analysis, we can produce a scatterplot to visually inspect whether or not there appears to be a linear relationship between age and feelings towards the Green party using the following syntax to produce the scatterplot shown below:

```
ggplot(CES2019, aes(x=age, y=greenfeelings)) +
  geom_point(alpha=.1) +
  theme_bw() +
  labs(x="Age", y="Feelings about the Green Party")
```



Now, we assess the scatter plot. What do you see? Don't panic – we don't see anything either. Based on the graph, it is difficult to interpret any type of relationship! It may be that age is *not* associated with feelings about the Green party. To be sure, we need to look to further, either using Pearson's *r* or basic linear regression.

Pearson's *r*

We can find the measure of association between the two variables, Pearson's *r*. To estimate this value, we will use the command for a pairwise correlation, ***pwcorr,*** and we will add the command ***sig*** so that the output includes the level of statistical significance for the relationship. We will also apply the sample weight with this command. All together, our syntax is as follows:

```
pwCorrMat(~age + greenfeelings, data=CES2019, weight=CES2019$weight_CES)
```

Compare your results to our own.

```
> pwCorrMat(~ age + greenfeelings, CES2019,
+            weight=CES2019$weight_CES)
Pairwise Correlations
               age     greenfeelings
age
greenfeelings -0.150*
```

In the above code, we identify the variables to be correlated by specifying them on the right hand side of a formula separated by pluses. In interpreting the results, we will start with the correlation coefficient. In R, the output does not include the diagonal element, which is always 1. The results indicate a weak, negative relationship between age and feelings about the Green party (-0.15): as age increases, feelings about the Green party decrease.

We next turn to the inferential statistic to see if this weak, negative relationship is statistically significant. Correlations that are statistically significant at the specified level are flagged with a single asterisk. The default level is 0.05, but you can specify a different one with the `level` argument to the function. For example, if you wanted to evaluate significance at the p=0.001 level, you would add the argument `level=.001` to the function call above. The value below the correlation coefficient is the probability of observing a relationship of this strength in the sample if a similar relationship ***did not*** exist in the population from which the sample was drawn. In this case, the relationship is found to be statistically significant at p<0.05.

Basic linear regression

While the measure of association and the strength of the relationship between two variables is informative, we can also use information about the independent variable to predict scores on the dependent variable using basic linear regression. R allows us to easily produce regression models with the use of the ***regress*** (***reg***) command.

To continue with our example, we can estimate how feelings for the Green party changes for every year increase in age with the syntax below. In the first line, we turn the "greenfeelings" variable into a numeric rather than labelled class. It's not necessary, but it does streamline the output a bit. You would only have to do this once per R session, but it wouldn't cause problems if by accident you happened to do it again. In the second and third lines, we use the `lm()` function which estimates least squares regression. In this case, we use the weighting variable, too. The first argument to the function is a formula where the dependent variable is on the left-hand side of the tilde (~) and the independent variables separated by pluses (in this case).

```
CES2019$greenfeelings <- as.numeric(CES2019$greenfeelings)
green.mod <- lm(greenfeelings ~ age, data=CES2019,
     weight=weight_CES)
summary(green.mod)
```

```
> CES2019$greenfeelings <- as.numeric(CES2019$greenfeelings)
> green.mod <- lm(greenfeelings ~ age, data=CES2019, weight=weight_CES)
> summary(green.mod)

Call:
lm(formula = greenfeelings ~ age, data = CES2019, weights = weight_CES)

Weighted Residuals:
    Min      1Q  Median      3Q     Max 
-69.899 -18.605   2.974  18.159  85.274 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 55.20274    1.41275  39.075   <2e-16 ***
age         -0.24326    0.02609  -9.322   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 26.71 on 3756 degrees of freedom
  (263 observations deleted due to missingness)
Multiple R-squared:  0.02261,	Adjusted R-squared:  0.02235 
F-statistic:  86.9 on 1 and 3756 DF,  p-value: < 2.2e-16
```

Be sure to check your results against ours.

There is considerably more information presented here than in the Pearson's *r* results. Let's walk through some of it:

- Age coefficient. The age coefficient is -0.24. This indicates that for every year increase in age, feelings about the Green party decrease by 0.24 points. We also find that there is a statistically significant relationship between age and feelings about the Green party based on the value reported under the P(> |t|) column (0.000). We would report this a p<0.001 in our written interpretation of the results.
- Intercept: The constant (intercept), 55.20, is the value on the feelings about the Green party variable when age is equal to 0 (an impossible value given that respondents for the CES are a minimum of 18 years of age).
- r^2: How much of the variance in the dependent variable does our independent variable explain? Not much. Recall from Chapter 12 that we can determine the proportion of the dependent variable that can be explained by the independent variable by squaring the Pearson's *r* value, producing a result known as r^2. This value is reported with the R output as "Multiple R-squared", 0.02 in this example. In other words, using age as a predictor of feelings about the Green party reduces our prediction errors by two percent.

Check-In Point

If you are continuing to follow along by running all of the examples in your own dataset, you now have the ability to create a scatter plot, calculate Pearson's *r*, and conduct basic linear regression with R. You have come a long way!

Part X: Assessing Relationships Using Control Variables

Destination

By the end of this section, you will be able to:

- *add control variables to your analyses.*

The final question in assessing a relationship between two variables is to consider what happens to the relationship once other important variables are controlled. We discuss this question fully in Chapter 13, and in this section of the handbook we direct you to the appropriate R syntax.

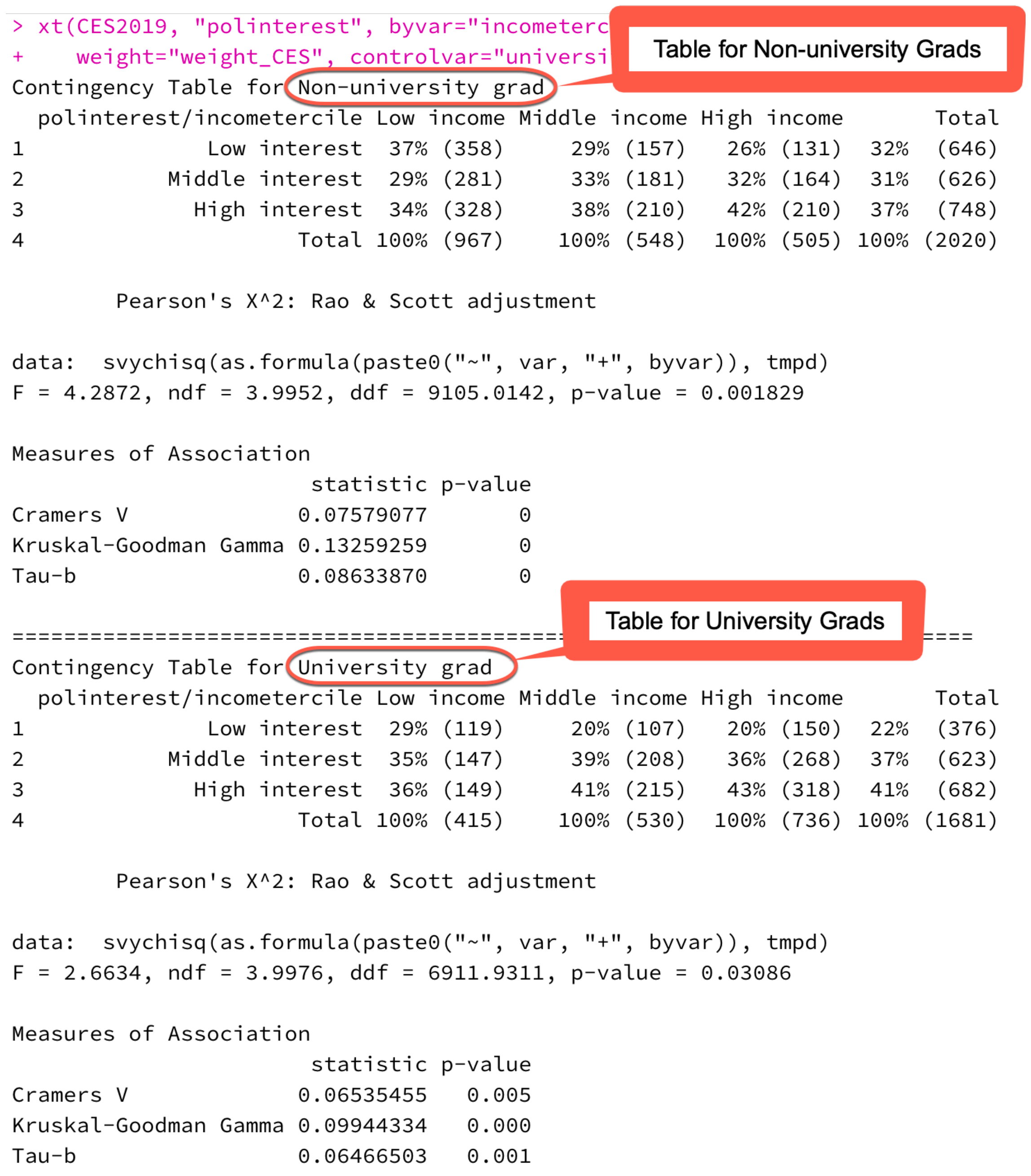
Cross tabs and control variables

Recall from Chapter 13 that to test a control variable using a crosstabulation, you assess the IV -DV relationship separately for each category of the control and compare these results against those obtained in the original (full) model.

Let's consider the relationship between income and political interest (recall that we recoded these variables previously), controlling for education (university graduate versus non-university graduate). We can use the `xt()` function with the `controlvar` argument command, which instructs R to generate contingency tables and measures of fit separately for each sub-group (category) of the variable indicated. To test the income-political interest relationship while controlling for education, we instruct R to create crosstabulations for all categories of education. Try it by first generating the dichotomous education variable and then using the `xt()` function as follows:

```
# Use the searchVarLabels function to find the interest variable

searchVarLabels(CES2019, "education")

# Look at the value labels and distribution of the original
variable

inspect(CES2019, "q61")

# Generate, recode and label the new variable

CES2019 <- mutate(CES2019, universitygrad = car::recode(q61,
  "1:8='Non-university grad'; 9:11='University grad'; else=NA",
  as.factor=TRUE,
  levels=c("Non-university grad", "University grad")))

attr(CES2019$universitygrad, "label") <-
  "University graduate versus non-university graduate"

# Check original and new variable distributions

with(CES2019, table(q61, universitygrad, useNA="ifany"))

# Generate crosstab with control variable
xt(CES2019, "polinterest", byvar="incometercile",
     weight="weight_CES", controlvar="universitygrad")
```

This will produce the following results for non-graduates and university graduates, respectively:



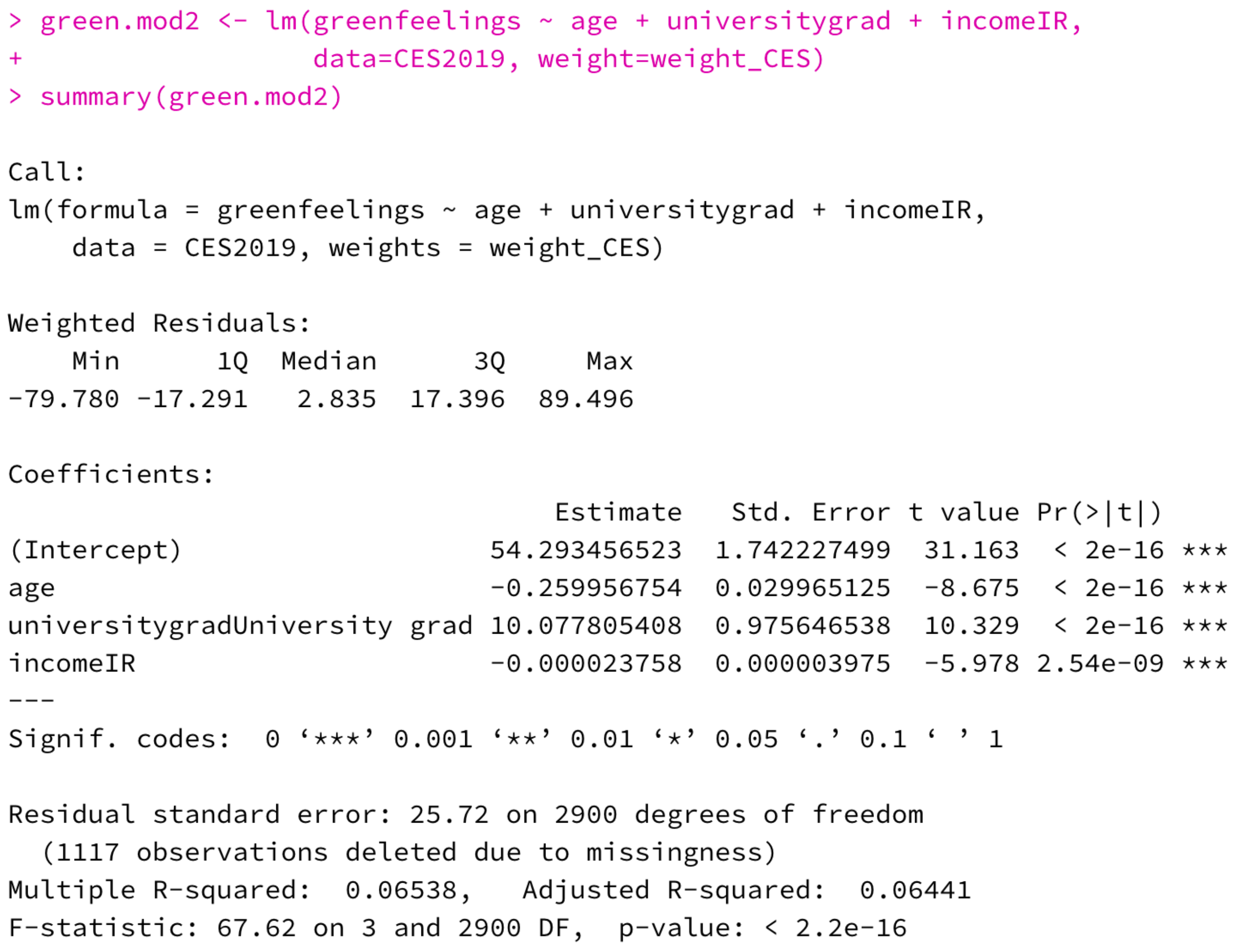
```
> xt(CES2019, "polinterest", byvar="incometerc
+    weight="weight_CES", controlvar="universi
Contingency Table for Non-university grad
  polinterest/incometercile Low income Middle income High income        Total
1               Low interest  37% (358)     29% (157)   26% (131)   32%  (646)
2            Middle interest  29% (281)     33% (181)   32% (164)   31%  (626)
3              High interest  34% (328)     38% (210)   42% (210)   37%  (748)
4                      Total 100% (967)    100% (548)  100% (505)  100% (2020)

        Pearson's X^2: Rao & Scott adjustment

data:  svychisq(as.formula(paste0("~", var, "+", byvar)), tmpd)
F = 4.2872, ndf = 3.9952, ddf = 9105.0142, p-value = 0.001829

Measures of Association
                       statistic p-value
Cramers V              0.07579077       0
Kruskal-Goodman Gamma  0.13259259       0
Tau-b                  0.08633870       0

==========================================                              ====
Contingency Table for University grad
  polinterest/incometercile Low income Middle income High income        Total
1               Low interest  29% (119)     20% (107)   20% (150)   22%  (376)
2            Middle interest  35% (147)     39% (208)   36% (268)   37%  (623)
3              High interest  36% (149)     41% (215)   43% (318)   41%  (682)
4                      Total 100% (415)    100% (530)  100% (736)  100% (1681)

        Pearson's X^2: Rao & Scott adjustment

data:  svychisq(as.formula(paste0("~", var, "+", byvar)), tmpd)
F = 2.6634, ndf = 3.9976, ddf = 6911.9311, p-value = 0.03086

Measures of Association
                       statistic p-value
Cramers V              0.06535455   0.005
Kruskal-Goodman Gamma  0.09944334   0.000
Tau-b                  0.06466503   0.001
```

The interpretation of the relationship with the inclusion of a control variable is the same as the process you followed to interpret the original relationship, only you need to do so for each category of the control variable. You then compare the results from each of the control variable categories to that of the original relationship to assess whether or not the control variable affects the relationship as anticipated (see Chapter 13 for a full interpretation of the results with the addition of the control variable). Recall that we originally observed an extremely weak, statistically significant relationship (Tau = 0.09; p<0.001) in the original model, with levels of political interest increasing with income. When we control for education, we find that the relationship is essentially replicated for non-university graduates. The same basic relationship holds for university graduates, too though it is a bit weaker (Tau = .06, p=.001). Accordingly, education does not appear to be a source of spuriousness.

Multivariate linear regression

The syntax for multivariate linear regression in R is the same as that for basic linear regression, only we add the additional independent and/or control variables to the model. For example, in addition to age, we can assess how education and income influence feelings about the Green party with the following syntax:

```
green.mod2 <- lm(greenfeelings ~ age + universitygrad + incomeIR,
     data=CES2019, weight=weight_CES)
summary(green.mod2)
```



```
> green.mod2 <- lm(greenfeelings ~ age + universitygrad + incomeIR,
+                  data=CES2019, weight=weight_CES)
> summary(green.mod2)

Call:
lm(formula = greenfeelings ~ age + universitygrad + incomeIR,
    data = CES2019, weights = weight_CES)

Weighted Residuals:
    Min      1Q  Median      3Q     Max 
-79.780 -17.291   2.835  17.396  89.496 

Coefficients:
                                  Estimate   Std. Error t value Pr(>|t|)    
(Intercept)                    54.293456523  1.742227499  31.163  < 2e-16 ***
age                            -0.259956754  0.029965125  -8.675  < 2e-16 ***
universitygradUniversity grad  10.077805408  0.975646538  10.329  < 2e-16 ***
incomeIR                       -0.000023758  0.000003975  -5.978 2.54e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 25.72 on 2900 degrees of freedom
  (1117 observations deleted due to missingness)
Multiple R-squared:  0.06538,	Adjusted R-squared:  0.06441 
F-statistic: 67.62 on 3 and 2900 DF,  p-value: < 2.2e-16
```

The results show that, holding education and income equal, for every year increase in age, feelings about the Green party decrease by 0.26 points (p<0.001). In the case of education, given that we are using a dichotomous variable, we would interpret the results as indicating that university graduates are more likely (10 points) to have more positive feelings about the Green party than those who have not completed university, net of age and income. While the coefficient for income is statistically significant, the impact on feelings about the Greens is marginal, a decrease of 0.00002 points for every unit increase in income, holding age and education constant. How much of the variance in the dependent variable does our model explain? We can use the value reported as the adjusted R-squared, which takes into account the number of variables in the model, to determine the proportion of the dependent variable that can be explained by the independent variables. In this example, our model reduces our prediction errors by 6 percent.

Check-In Point and Conclusion

As we come to the end of this introductory R handbook, we hope that the procedures outlined here have provided you with the basic skills necessary to conduct your own statistical analyses. We also hope that this introduction has encouraged you to learn more about the many possibilities to use this type of statistical program for your research. As we have noted, this is only a very small sampling of the many options available in R, which we hope serves as the starting point for your continued exploration of the possibilities that this software has to offer.