Foundational Skills

Getting started

First, you will need to download the latest versions of R Studio and R. Although you will likely exclusively use R Studio, this software (R Studio) needs to have R installed, as well, as R Studio uses R behind-the-scenes. Both are freely-available, cross-platform, and open-source.

Downloading R Studio

To download R Studio:

- Visit this page here to download R: https://cran.r-project.org/
- Find your operating system (Mac, Windows, or Linux)
- Download the 'latest release' on the page for your operating system and download and install the application

Don't worry; you will not mess anything up if you download (or even install!) the wrong file. Once you've installed both, you can get started.

If you do have issues, consider this page, and then reach out for help. One good place to start is the R Studio Community is a great place to start.

Check that it worked

Open R Studio. Find the console window and type in `2 + 2`. If what you can guess is returned (hint: it's what you expect!), then R Studio *and* R both work.

Help, I'm completely new to using R / R Studio!

If you're completely new, Swirl is a great place to start, as it helps you to learn R *from within R Studio*. Visit this page to see some directions: http://swirlstats.com.

If you have a bit more confidence but still feel like you need some time to get started, Data Camp is another good place to start.

And if you're ready to go, please proceed to the next sections on processing and preparing, plotting, loading, and modeling data and sharing results.

Creating projects

Before proceeding, we're going to take a few steps to set ourselves to make the analysis easier; namely, through the use of Projects, an R Studio-specific organizational tool.

To create a project, in R Studio, navigate to "File" and then "New Directory".

Then, click "New Project". Choose a directory name for the project that helps you to remember that this is a project that involves data science in education; it can be convenient if the name is typed in `lower-case-letters-separated-by-dashes`, like that. You can also choose the sub-directory. If you are just using this to learn and to test out creating a project, you may consider placing it in your downloads or another temporary directory so that you remember to remove it later.

Loading data from various sources

In this section, we'll start with loading data.

You might be thinking that an Excel file is the first that we would load, but there happens to be a format which you can open and edit in Excel that is even easier to use between Excel and R and among Excel, R, as well as SPSS and other statistical software, like MPlus, and even other programming languages, like Python. That format is CSV, or a comma-separated-values file.

The CSV file is useful because you can open it with Excel and save Excel files as CSV files. Additionally, and as its name indicates, a CSV file is rows of a spreadsheet with the columns separated by commas, so you can view it in a text editor, like TextEdit for Macintosh, as well. Not surprisingly, Google Sheets easily converts CSV files into a Sheet, and also easily saves Sheets as CSV files.

For these reasons, we start with reading CSV files.

Loading CSV files

The easiest way to read a CSV file is with the function `read_csv()`. It is from a package, or an add-on to R, called `readr`, but we are going to install `readr` as well as other packages that work well together as part of a group of packages named the `tidyverse`. To install all of the packages in the tidyverse, use the following command:

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

You can also navigate to the Packages pane, and then click "Install", which will work the same as the line of code above. Note, here there is a way to install a package using code or part of the R Studio interface. Usually, using code is a bit quicker, but sometimes (as we will see in a moment) using the interface can be very useful and sometimes complimentary to use of code.

We have now installed the tidyverse. We only have to install a package once, but to use it, we have to load it each time we start a new R session. We will discuss what an R session is later on; for now, know that we have to load a package to use it. We do that with `library()`.

Saving a file from the web (and a complicated walkthrough of some simple code)

Next, you'll need to copy this URL:

`https://goo.gl/bUeMhV`

Here's what it resolves to (it's a CSV file):

`https://raw.githubusercontent.com/data-edu/data-science-in-education/master/data/pisaUSA15/stu-quest.cs`

This bit of code downloads the file to your working directory. Run this to download it so in the next step you can read it into R. As a note: There are ways to read the file directory (from the web) into R. Also, of course, you could do what the next (two) lines of code do manually: Feel free to open the file in your browser and to save it to your computer (you should be able to 'right' or 'control' click the page to save it as a text file with a CSV extension).

```
student_responses_url <- "https://goo.gl/bUeMhV"
student_responses_file_name = paste0(getwd(), "/student-responses-data.csv")
download.file(url = student_responses_url, destfile = student_responses_file_name)
```

It may take a few seconds to download as it's around 20 MB.

If you're curious, what is going on here is pretty simple, but it also involves many core data science ideas and ideas from programming/coding. What is happening is that the *character string* `"https://goo.gl/wPmujv"`

is being saved to an *object*, `student_responses_url`. Then, that *object* is being passed as an *argument* to a *function*, `download.file()` (specifically, to the `url` argument), along with another object, `student_responses_file_name`, which is passed to the `destfile` argument. In short, the `download.file()` function needs to know a) where the file is coming from (which you tell it through the `url`) argument and b) where the file will be saved (which you tell it through the `destfile` argument). This bit - `paste0(getwd(), "/student-responses-data.csv")` - is just creating a file name with a *file path* with your working directory, so it saves the file in the folder that you are working in.

Of course, you don't really need to know all of this to use the function–or to use other functions. But understanding how R is working in these terms can be helpful for troubleshooting and reaching out for help. It also helps you to, for example, use functions that you have never used before, because you are familiar with how some functions work.

Now, in R Studio, you should see the downloaded file in the Files tab. This should be the case if you created a project with R Studio; if not, it should be whatever your working directory is set to; run `getwd()` to find out. If the file is there, great. If things are *not* working, consider downloading the file in the manual way; and then move it into the directory that the R Project you created it.

Okay, we're ready to go. Let's load the tidyverse library:

```
library(tidyverse) # so tidyverse packages can be used for analysis
```

You may have noticed the hash symbol after the code that says `library(tidyverse). It reads`# so tidyverse packages can be used for analysis‘. That is a comment and the code after it (but not before it) is not run (the code before it runs just like normal). Comments are useful for showing *why* a line of code does what it does.

After loading the tidyverse packages, we can now load a file. We are going to call the data `student_responses`:

```
# readr::write_csv(pisaUSA15::stu_quest, here::here("data", "pisaUSA15", "stu_quest.csv"))
student_responses <- read_csv("student-responses-data.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   CNTRYID = col_integer(),
##   CNT = col_character(),
##   CNTSCHID = col_integer(),
##   CYC = col_character(),
##   NatCen = col_character(),
##   Region = col_integer(),
##   STRATUM = col_character(),
##   SUBNATIO = col_integer(),
##   OECD = col_integer(),
##   ADMINMODE = col_integer(),
##   Option_CPS = col_integer(),
##   Option_FL = col_integer(),
##   Option_ICTQ = col_integer(),
##   Option_ECQ = col_integer(),
##   Option_PQ = col_integer(),
##   Option_TQ = col_integer(),
##   Option_UH = col_integer(),
##   Option_Read = col_character(),
##   Option_Math = col_character(),
##   LANGTEST_QQQ = col_integer()
##   # ... with 460 more columns
## )
```

```
## See spec(...) for full column specifications.
```

Since we loaded the data, we now want to look at it. We can just type its name, and a summary of the data will print:

```
student_responses
```

```
## # A tibble: 5,712 x 922
##    CNTRYID CNT   CNTSCHID CNTSTUID CYC   NatCen Region STRATUM SUBNATIO
##      <int> <chr>    <int>    <dbl> <chr> <chr>   <int> <chr>      <int>
##  1     840 USA   84000001 84006899 06MS  084000  84000 USA0103  8400000
##  2     840 USA   84000001 84000625 06MS  084000  84000 USA0103  8400000
##  3     840 USA   84000001 84007720 06MS  084000  84000 USA0103  8400000
##  4     840 USA   84000001 84001279 06MS  084000  84000 USA0103  8400000
##  5     840 USA   84000001 84000532 06MS  084000  84000 USA0103  8400000
##  6     840 USA   84000001 84005284 06MS  084000  84000 USA0103  8400000
##  7     840 USA   84000001 84001664 06MS  084000  84000 USA0103  8400000
##  8     840 USA   84000001 84010771 06MS  084000  84000 USA0103  8400000
##  9     840 USA   84000001 84003969 06MS  084000  84000 USA0103  8400000
## 10     840 USA   84000001 84010965 06MS  084000  84000 USA0103  8400000
## # ... with 5,702 more rows, and 913 more variables: OECD <int>,
## #   ADMINMODE <int>, Option_CPS <int>, Option_FL <int>, Option_ICTQ <int>,
## #   Option_ECQ <int>, Option_PQ <int>, Option_TQ <int>, Option_UH <int>,
## #   Option_Read <chr>, Option_Math <chr>, LANGTEST_QQQ <int>,
## #   LANGTEST_COG <int>, LANGTEST_PAQ <dbl>, CBASCI <int>, BOOKID <int>,
## #   ST001D01T <int>, ST003D02T <int>, ST003D03T <int>, ST004D01T <int>,
## #   ST005Q01TA <dbl>, ST006Q01TA <dbl>, ST006Q02TA <dbl>,
## #   ST006Q03TA <dbl>, ST006Q04TA <dbl>, ST007Q01TA <dbl>,
## #   ST008Q01TA <dbl>, ST008Q02TA <dbl>, ST008Q03TA <dbl>,
## #   ST008Q04TA <dbl>, ST011Q01TA <dbl>, ST011Q02TA <dbl>,
## #   ST011Q03TA <dbl>, ST011Q04TA <dbl>, ST011Q05TA <dbl>,
## #   ST011Q06TA <dbl>, ST011Q07TA <dbl>, ST011Q08TA <dbl>,
## #   ST011Q09TA <dbl>, ST011Q10TA <dbl>, ST011Q11TA <dbl>,
## #   ST011Q12TA <dbl>, ST011Q16NA <dbl>, ST011D17TA <chr>,
## #   ST011D18TA <chr>, ST011D19TA <chr>, ST012Q01TA <dbl>,
## #   ST012Q02TA <dbl>, ST012Q03TA <dbl>, ST012Q05NA <dbl>,
## #   ST012Q06NA <dbl>, ST012Q07NA <dbl>, ST012Q08NA <dbl>,
## #   ST012Q09NA <dbl>, ST013Q01TA <dbl>, ST123Q01NA <dbl>,
## #   ST123Q02NA <dbl>, ST123Q03NA <dbl>, ST123Q04NA <dbl>,
## #   ST019AQ01T <dbl>, ST019BQ01T <dbl>, ST019CQ01T <dbl>,
## #   ST021Q01TA <dbl>, ST022Q01TA <dbl>, ST124Q01TA <chr>,
## #   ST125Q01NA <dbl>, ST126Q01TA <dbl>, ST127Q01TA <dbl>,
## #   ST127Q02TA <dbl>, ST127Q03TA <dbl>, ST111Q01TA <dbl>,
## #   ST118Q01NA <dbl>, ST118Q02NA <dbl>, ST118Q03NA <dbl>,
## #   ST118Q04NA <dbl>, ST118Q05NA <dbl>, ST119Q01NA <dbl>,
## #   ST119Q02NA <dbl>, ST119Q03NA <dbl>, ST119Q04NA <dbl>,
## #   ST119Q05NA <dbl>, ST121Q01NA <dbl>, ST121Q02NA <dbl>,
## #   ST121Q03NA <dbl>, ST082Q01NA <dbl>, ST082Q02NA <dbl>,
## #   ST082Q03NA <dbl>, ST082Q08NA <dbl>, ST082Q09NA <dbl>,
## #   ST082Q12NA <dbl>, ST082Q13NA <dbl>, ST082Q14NA <dbl>,
## #   ST034Q01TA <dbl>, ST034Q02TA <dbl>, ST034Q03TA <dbl>,
## #   ST034Q04TA <dbl>, ST034Q05TA <dbl>, ST034Q06TA <dbl>,
## #   ST039Q01NA <dbl>, ST039Q02NA <dbl>, ...
```

Woah, that's a big data frame (with a lot of variables with confusing names, to boot). This was a minor task, but if you loaded a file and printed it, give yourself a pat on the back. It is no joke to say that many times

simply being able to load a file into new software. We are now well on our way to carrying out analysis of our data.

Loading Excel files

We will now do the same with an Excel file. You might be thinking that you can simply open the file in Excel and then save it as a CSV. This is generally a good idea. At the same time, sometimes you may need to directly read a file from Excel, and it is easy enough to do this.

The package that we use, `readxl`, is not a part of the tidyverse, so we will have to install it first (remember, we only need to do this once), and then load it using `library(readxl)`. Note that the command to install `readxl` is grayed-out below: The `#` symbol before `install.packages("readxl")` indicates that this line should be treated as a comment and not actually run, like the lines of code that are not grayed-out. It is here just as a reminder that the package needs to be installed if it is not already.

Once we have installed readxl, we have to load it (just like tidyverse):

```
# install.packages("readxl")
library(readxl)
```

We can then use `read_excel()` in the same way as `read_csv()`, where "path/to/file.xlsx" is where an Excel file you want to load is located (note that this code is not run here):

```
my_data <- read_excel("path/to/file.xlsx")
```

Of course, were this run, you can replace `my_data` with a name you like. Generally, it's easy to use short and easy-to-type names for data, as you will be typing and using it a lot.

Note that one easy way to find the path to a file is to use the "Import Dataset" menu. It is in the Environment window of R Studio. Click on that menu bar option, select the option corresponding to the type of file you are trying to load (e.g., "From Excel"), and then click The "Browse" button beside the File/URL field. Once you click on the, R Studio will automatically generate the file path - and the code to read the file, too - for you. You can copy this code or click Import to load the data.

Loading SAV files

The same factors that apply to reading Excel files apply to reading `SAV` files (from SPSS). First, install the package `haven`, load it, and the use the function `read_sav()`:

```
# install.packages("haven")
library(haven)
my_data <- read_sav("path/to/file.xlsx")
```

Google Sheets

Finally, it can sometimes be useful to load a file directly from Google Sheets, and this can be done using the Google Sheets package.

```
# install.packages("googlesheets")
library(googlesheets)
```

When you run the command below, a link to authenticate with your Google account will open in your browser.

```
my_sheets <- gs_ls()
```

You can then simply use the `gs_title()` function in conjunction with the `gs_read()` function:

```
df <- gs_title('title')
df <- gs_read(df)
```

Saving files

Saving files is relatively easy (compared to loading them). Using our data frame `student_responses`, we can save it as a CSV (for example) with the following function:

```
write_csv(student_responses, "student-responses.csv")
```

That will save a CSV file entitled `student-responses.csv` in the working directory. If you want to save it to another directory, simply add the file path to the file, i.e. `path/to/student-responses.csv`. To save a file for SPSS, load the haven package and use `write_sav()`. There is not a function to save an Excel file, but you can directly and simply load a CSV file in Excel.

Conclusion

For more on reading files, we will discuss how to use functions to read every file in a folder (or, to write many different files to a folder).

Processing data

Now that we have loaded `student_responses`, we can process it. This section highlights some common data processing functions.

We're also going to introduce a powerful, unusual *operator* in R, the pipe. The pipe is this symbol: `%>%`. It lets you *compose* functions. It does this by passing the output of one function to the next.

Here's an example. Let's say that we want to select just a few variables from the `student_responses` dataset. Here's how we would do that using `select()`.

```
student_mot_vars <- select(student_responses, SCIEEFF, JOYSCIE, INTBRSCI, EPIST, INSTSCIE)
```

Note that we saved the output from the `select()` function to a new data frame, this one called `student_mot_vars`. We could save it back to `student_responses`, which would simply overwrite the existing data frame (with more variables), i.e. (the following code is not run here):

```
student_responses <- select(student_responses, SCIEEFF, JOYSCIE, INTBRSCI, EPIST, INSTSCIE)
```

We could also rename them at the same time we select them; I put these on separate lines so I could add the comment, but you could do this all in the same line, too; it does not make a difference in terms of how `select()` will work.

```
student_mot_vars <- select(student_responses,
                           student_efficacy = SCIEEFF,
                           student_joy = JOYSCIE,
                           student_broad_interest = INTBRSCI,
                           student_epistemic_beliefs = EPIST,
                           student_instrumental_motivation = INSTSCIE)
```

[will add more on creating new variables, filtering grouping and summarizing, and joining data sets]

Visualizing data

[not yet added - will add scatter plots, bar plots, and time series plots]

Modeling data

[not yet added - will add about regression/ANOVA]

Communicating / sharing results

[not yet added - will add about R Markdown]