# Assignment Two: Reading in Data

Your Name

today

## Lesson

- Today, we'll discuss how to set up a project directory, read in some data from IPEDS, and set up GitHub Copilot.
  - Note: There is no "incorrect" way of organizing (as long as it works), however, the principles we use in this lesson will help you avoid issues in the future

### Organizing a Project Folder

We'll begin with how to organize your course and project files.

- Note: we are now arriving at one of the first small differences from previous versions of the course; how the sub-folders are organized. There's a trade-off between simplicity and tidy-ness here, we have opted for simplicity

#### Skinner's Kitchen Metaphor

> "Every data analysis project should have its own set of organized folders. Just like you might organize a kitchen so that ingredients, cookbooks, and prepared food all have a specific cabinet or shelf, so too should you organize your project. We'll organize our course directory in a similar fashion."

- In Dr. Skinner's kitchen, think of pristine space where **NOTHING** sits out on the counter.
- However, we are going to have kitchen where some things are stored away in cupboards, but what we regularly work with (the scripts) sits out on the counter
  - The kitchen metaphor also works to explain why you might take these approaches
    - It's definitely tidier to keep everything in your kitchen stored away, but, it adds an extra step whenever you want to cook a meal. The same is true here.
    - Why? We will get to that later…

#### EDH 7916 Folder Setup

- With this kitchen metaphor in mind, let's set up our folder for the class
- I've made an EDH-7916 example folder you can download here which is also available on the class homepage

  - In here, you will see
    - An `.R` script template
    - A Quarto `.qmd` template
    - A set of numbered `.R` scripts (these are our lesson scripts)
    - A data folder
    - A reproducible-report folder (with it's own data sub-folder)
    - A `.pdf` copy of the syllabus

- Download and save this folder wherever you usually keep class folders

  - Here we use Desktop, but you can use your Documents folder etc. if you wish
  - You can rename the folder if you'd like (but please don't rename the internal folders)
    - See naming guidelines below on how best to name files

- Throughout the class (and especially in your final project) you may feel the need for other sub-folders for other items (such as one to keep graphs in), but this should be fine for now

**R Project Setup**

- RStudio has some really helpful features, one of which is creating R Projects easily
  - At their very simplest, these can be ways of keeping your RStudio environment saved (especially helpful switching between projects), but also enable more feature like using git (see extra credit lesson)
  - Note: If you're using posit.cloud, you already have a RStudio project by default
- It's pretty easy to set up a project now we have our class folder set up
  1. In the top right corner of RStudio you'll see a blue cube with "none" next to it
  2. Click there, then "new project"
  3. Then click "from existing directory"
  4. Find the class folder we just created, select it, and we're done!
- This is really useful for keeping track of multiple projects, but if this is all you use it for, it will be helpful to keep working directory correct!

**Naming Guidelines**

- Your class scripts and data files are already named, but there will be numerous files you need to create throughout the class (assignment scripts, everything for your final report, etc.). So it's best we get on the same page

- Always name your files something logical

  - The file name should always tell something about the purpose of that file or folder

- Script numbering

  - Following Hadley Wickham (the founder of RStudio)'s script numbering
    - Basically start all your script names with the number that they should be run in
      - `01-data-reading`
      - `02-data-cleaning`
      - `03-data-analysis`

    - This can be especially helpful if you're keeping scripts in the top level of the project directory to keep things organized

- Generally, a good programming tip is to avoid spaces at all costs, use dashes or underscores instead

- It's also good to be consistent with capitalization, most traditional programmers will avoid it completely, but if you do it, do it consistently throughout that project
  - We used no capitalization through this class
    - Whatever you do, never (ever, ever) have different versions of files with the same name but different capitalization

- Lastly, try to keep files names as short as possible
  - Later on we will be be in situations where we have to type out file names, so if you go too long, it can become frustrating

- How do we understand the names of our class scripts?
  1. Hadley Wickham's script numbering corresponding to the order of the lessons
  2. `set`, `wrangle`, `viz`, `quarto`, or `pro` indicate which group of lessons it belongs to
  3. Anything else is just descriptive, roman numerals for the lesson series, or a descriptive word

- With our class folder now set up, it's time to go over some other key organization principles

**Working Directory**

- The working directory is almost certainly the most common cause of issues in this class, and continues to be something I get tripped up by from time to time, so this may take a minute to get your head around

- As a general rule, no matter how you have your folders organized in the future, you usually want your working directory set to *where your script is*
  - That way, you're always giving directions from the common point of "where we are we are right now"
    - This will then be the same if I move the project folder on my computer, or run it on someone else's computer

- By default, when we open a project in RStudio, RStudio helpfully sets our working directory to the project folder
  - This is why we are keeping our scripts out on the counter top so to speak, the default working directory *should* be the correct one

- That said, there will be times when you need to change your working directory, so, let's go over the basics of that quickly
  - For instance, if you forget to open a project, RStudio will often the leave working directory as your root folder

  - You can see the currently working directory path next to the little R icon and version at the top of your console panel
  - If it's wrong, there are a few ways to change it

    1. Find "session" on the top drop-down menu

    - Then "set working directory"
    - Then "To source file location"
      - This should be the same as "To project directory" as our scripts are stored at the top level of the project folder

    2. Install the `this.path` package (recall how to do that from last week)

    - With that installed, call `setwd(this.path::here())` at the top of the script
      - Note: `this.path::here()` is the same as doing `library(this.path)` followed by `here()` but is more efficient if you only want one thing from a package
      - Assuming you want the working directory to be the script location, this never hurts to always leave at the top

    3. Navigate to the desired folder in the files pane (bottom right)

    - Select the cog symbol
      - Select "Set as working directory"
        - Note: "Go to working directory" can be useful to see what's in the folder if you navigate away

    4. The old school vanilla R way `setwd("<path to your script>")`

    - But, this really isn't usually the most efficient

- As I say, if we organize our folder as outlined in this lesson, and use an R project, we shouldn't need to change this much, but it's inevitable you will every now and then

**File Paths**

- When we are working with R, we (most of the time) need to bring in other items, such as data
  - In order to do that, our computer has to find these items, and there are two ways it can do that

**Absolute Paths**

- Absolute paths are directions to what you're looking for starting from the root of your computer, and list out exactly where a file is. For example, the absolute path of this Quarto script we are now looking is

```
"/Users/Jue/Desktop/7916/02-set-data.qmd"
```

- This is perfectly fine, assuming two things
  1. I don't move the project directory
  2. It only needs to run on my computer
- Oftentimes, we cannot rely on both these assumptions being true

- ‣ Plus, if we start with these absolute paths and then need to change, it will then become a real pain to update everything
  - – So, we should **ALWAYS** use relative paths instead (this one of the only strict rules for assignments)

**Relative Paths**

- Imagine I am giving you directions to a College of Education cookout, but, I give you directions from my house. That's only any use if you know where my house is…
  - ‣ Instead, you really want directions from somewhere we all know, like Norman Hall
    - – That is (basically) how relative paths work, we give directions to to our data from a common point
- Relative paths are directions to what you're looking for from where you are right now (a.k.a your "working directory")
- If we assume have our working directory set to our shiny new class folder, then, that becomes the starting point for all our directions
  - ‣ Therefore, to access `hd2007.csv` in out `data` sub-folder, we just need to say `file.path("data", "hd2007.csv")`

**The `file.path()` Function**

- == One last thing, you see how file paths are typically written with `/` or `\`? (which depends on your computer)

```
## R has a nice function that means we don't have to

```

a)  worry about which way around the slash should be
```


-


    b)  avoid issues with different computers expecting different slashes
```

  - ‣ `file.path()`
    - – Inside, we just type the name of each folder/file in "quotes", turning
      - `"Users/Matt/Desktop/7916"` into
      - `file.path("Users", "Matt", "Desktop", "7916")`
      - This may look longer, but, it's more compatible and easier to remember"

**What If I Need to Go Back a Level?**

- Sometimes we are in a folder, but want to go back a level, i.e. not the folder our current folder is in
  - ‣ This is very common if with we were using the "pristine kitchen" approach
- To do so is easy, we just add a `".."` to our `file.path()`

    - So, if we are in my class folder on my desktop, and we want to go to another folder on the desktop
        - `file.path("..", "<folder we want>")`

## Script Template

- In our shiny new class folder, you'll see an `r-script-template.R` file (thanks to Matt and Ben)
    - This a resource for you to use for assignments and other work, feel free to change it to suit your needs
        - Generally you can just "Save As" the template everytime you make a new script
- The script header block is a useful way to keep more info than a file name can
- The main reason to use a template is to keep your work organized into sections
    - This template has
        - `Libraries` to load needed packages
        - `Input` to load data
        - `Prep` to clean the data
        - `Analysis` to run our analyses
        - `Output` to save our modified data
    - However, these will not always be the sections you need
        - In bigger projects, you might have a whole script for data cleaning
        - In other projects, you might want a section or script just for making plots
        - In your assignments, you'll likely want a section for each question
    - The main point is to ensure you have some kind of sections in your scripts
        - Scripts can be really hard to navigate if you don't!

### Quarto Template

- You should also see an `quarto-template.qmd` file in your folder
    - This is a template for your Quarto files, which you will use for your assignment and final project. We will go through Quarto in more detail in a few weeks, but for now you only need to know that you can write text along with codes in Quarto, and that you will render it to a PDF file for your assignment submission.

## Reading in Data

- Next, let's apply some of this thrilling knowledge about file paths and working directories to read in some data from IPEDS

- To do this, open `02-set-data.R` from your class folder

- First up, check your working directory by either

    - Looking at the top of your console or
    - Typing `getwd()` into the console

- This should be your class folder, but if not, we need to set it there

    - On the top drop-down menu, select "Session", "Set Working Directory,"To Source File Location"

    - Quick Question: Without scrolling up, who can remember the other ways of doing this?

- Okay, with this set, it's time to read in our first dataset!
    - Quick Question 1: Where is our data?
    - Quick Question 2: Who remembers how we assign something in R?
        - With those questions answered, we have everything we need

```
library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0
──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.1     ✔ tibble    3.2.1
✔ lubridate 1.9.3     ✔ tidyr     1.3.1
✔ purrr     1.0.2
── Conflicts ──────────────────────────────────────────── tidyverse_conflicts()
──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors
```

```
df_ipeds <- read_csv(file.path("data", "hd2007.csv"))
```

```
Rows: 7052 Columns: 59
──                          Column                              specification
────────────────────────────────────────────────────────
Delimiter: ","
chr (16): INSTNM, ADDR, CITY, STABBR, ZIP, CHFNM, CHFTITLE, EIN, OPEID, WEBA...
dbl (43): UNITID, FIPS, OBEREG, GENTELE, OPEFLAG, SECTOR, ICLEVEL, CONTROL, ...

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

Success!

We will cover other ways of reading in data over the course of the class (we can download data directly to R somtimes), but this is most common way, so we are ready for some analysis next week!

That's it for R today, phew!

Now let's go and explore IPEDS Data Center and see where a lot of contemporary higher education research data comes from!

## IPEDS Exploration Key Points (for review)

- IPEDS is an annual federally mandated data collection process (and compliance is a significant portion of many Institutional Researcher jobs)
- There are a few ways of downloading IPEDS data, if you click through the website you may well find a point-and-click way of selecting specifc variables
  - This is **NOT** reproducible and therefore **NOT** the best practice for research
- We want to use the IPEDS data center to access the complete data files then select and join variables to get our desired data set
  - Fear not, we will go over how to do those things in the first two data wrangling lessons!

### "Data File" vs. "STATA Data File"

- For some NCES data sets, such as HSLS downloaded from NCES DataLab, selecting the Stata file option will download a `.dta` STATA file format version of the data, which is often nicely labelled
  - We can actually read these into R using the `haven` library from the `tidyverse`
- However, for IPEDS, the STATA file option is actually just another `.csv` file, it's formatted slightly differently to read into STATA, so there's no reason to bother with it when using R

### Using IPEDS Codebooks

- To be able to use most of these big data sets, you need to be able to understand the code. Let's look together at the codebook for EFFY (headcount enrollment) for 2021
  - For IPEDS, the code book is called the dictionary, and is always an Excel file (`.xlsx`). Other data sources will look different but the general principle will be the same

## Setting up GitHub Copilot

One major change of the course this year is the introduction of GitHub Copilot.

### What is GitHub Copilot?

GitHub Copilot is an AI-powered coding assistant that helps write code faster by offering suggestions, autocompleting code, or generating code snippets based on comments or existing code. This is a tool that can be very helpful, but also can be a crutch if not used correctly. We will go over how to set it up and some best practices for using it.

### How to set up GitHub Copilot?

- First, you will need a GitHub account. If you don't have one, you can sign up one. Feel free to sign up for GitHub Student Developer Pack as a student.

- Next, you will need to enable GitHub Copilot from the Copilot page.

- Then, enable GitHub Copilot Plugin for RStudio.

  - On your top menu bar, go to Tools > Global Options > Copilot, and enable it
  - You will be prompted to sign in to your GitHub account and authorize GitHub Copilot to use your account

**Using GitHub Copilot for your projects**

If you do use GitHub Copilot to help with your codes, please keep your prompts in your comment line and acknowledge with a `## h/t`

## Other common data sources for higher education research

- National Center for Educational Statistics (also the owner of IPEDS)
  - Longitudinal Surveys such as HSLS-High School Longitudinal Study and ECLS-Early Childhood Longitudinal Studies
  - Administrative Data (including IPEDS)
  - NCES has a good amount of publicly available data, but they also have a LOT of restricted data
    - Typically publicly available data will be either institution level (school, college, university wide) or fully anonymized. Meanwhile restricted data will often be student level and have some more detailed information
      - Getting restricted data is tough, but not impossible
        - You will need a clear purpose of your study and to know exactly what data you want access to (see available data here)
        - You'll then need to take this idea to your advisor
      - For your final project in this class, your data **MUST** be publicly available<>
        - This means we must be able to go and download it ourselves, you won't submit data with your final project submission
- College Scorecard
  - Designed more as a tool for potential college students, college scorecard has data points of interest to this audience, but some things useful for our research too, in particular graduate earning levels
    - Similarly to IPEDS, if using College Scorecard, we want to avoid the point-and-click interface and download the entire data files available here
- National Bureau of Labor Statistics
  - Longitudinal surveys, some have educational variables similar to NCES but are often much broader in scope
    - National Longitudinal Survey of Youth (NLSY) is one of the most used
  - There are publicly available portions of these surveys, but other sections are restricted, see BLS's accessing data page for more info
- Census & American Community Survey
  - Useful for population statistics, not student specific
    - Common variables for higher ed research include education and income levels for a population
      - For example of what is available, see the variables available in the 2019 ACS here
    - We will actually do some cool stuff to download ACS data directly to R later in Data Viz III using the tidycensus package
- MSI Data Project
  - Details about MSI classification and funding, includes IPEDS ID numbers to easily link to additional data
- Many, many more, have fun exploring!

## Assignment

Assignment

## Question One

**a) Create an Excel spreadsheet with three columns; name, degree program, and years at UF**

**b) Add your information to it**

**c) Optional: Add some of your classmates information (recall from class introductions, or, re-introduce yourself after class)**

**d) Read this file into R and assign it to an object called `data` (just like we did in class)**

## Question Two

**a) Pick any single data file from IPEDS that peaks your interest**

**b) Download it and save it to your `data` folder**

**c) Read it into R and assign it to an object called `data_ipeds`**

1. Create a new R script in the top level of the project directory (use the template provided if you'd like)

- Note: to keep grading time reasonable, *please* keep your scripts in the top-level of your class folder, that way everyone is using the working directory and file paths
- To keep things organized, I'd suggest naming it something like
  - `assignment-01.R` or `a-02-set-data.R`
    - If you use the lesson name, I suggest adding the `a-` just to keep your assignments grouped separately

2. Create an Excel file (your last name_your first name.xlsx) with three columns detailing your name, your degree program, and your year. Save it to your `data` folder, read it in to R, and assign it an object (just like we did in class)

- Hint: you'll need a new package to read Excel files, between Google and asking your friends, you should be able to figure it out

3. Pick any single data file from IPEDS that peaks your interest, download it, save it to your `data` folder, and read it in to R, assigning it to an object (again, just like we did in class)

Once complete, turn in the .R script (no data etc.), .qmd file, and the PDF output to Canvas by the due date (Tuesday 12:00pm following the lesson). Assignments will be graded before next lesson on Wednesday in line with the grading policy outlined in the syllabus.