

Lecture 10: Acquiring data

Introduction

Reading to do before next class:

- ▶ [REQUIRED] GW chapter 11 (Data import)

What we will do today

1. Introduction
2. Relative file paths
3. Common data formats
4. readr package
 - 4.1 readr functions
 - 4.2 readr arguments
 - 4.3 readr column specification
5. Set variable and value labels
6. Reading in data from the web
 - 6.1 download.file function
7. readxl package
 - 7.1 readxl arguments
8. haven package

Load the packages we will use today (output omitted)

- ▶ you must run this code chunk after installing these packages

```
library(dplyr)
library(readr)
library(haven)
library(readxl)
library(labelled)
```

If package not yet installed, then must install before you load. Install in “console” rather than .Rmd file

- ▶ Generic syntax: `install.packages("package_name")`
- ▶ Install “tidyverse”: `install.packages("tidyverse")`

Note: when we load package, name of package is not in quotes; but when we install package, name of package is in quotes:

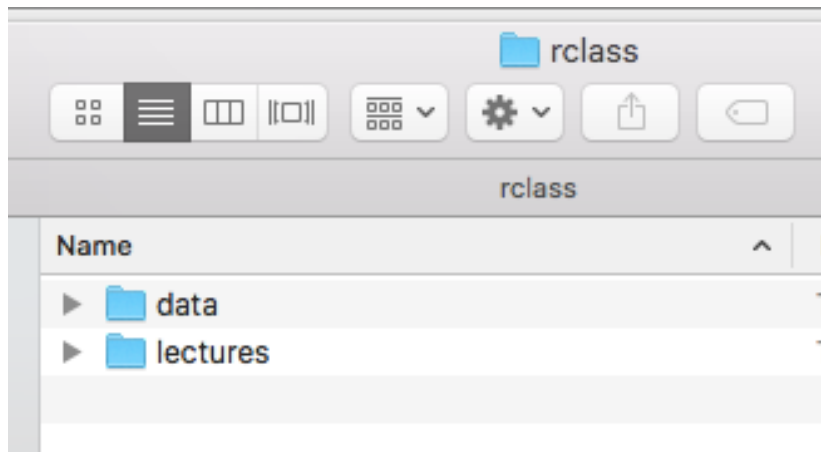
- ▶ `install.packages("tidyverse")`
- ▶ `library(tidyverse)`

Relative file paths

Rclass folder structure

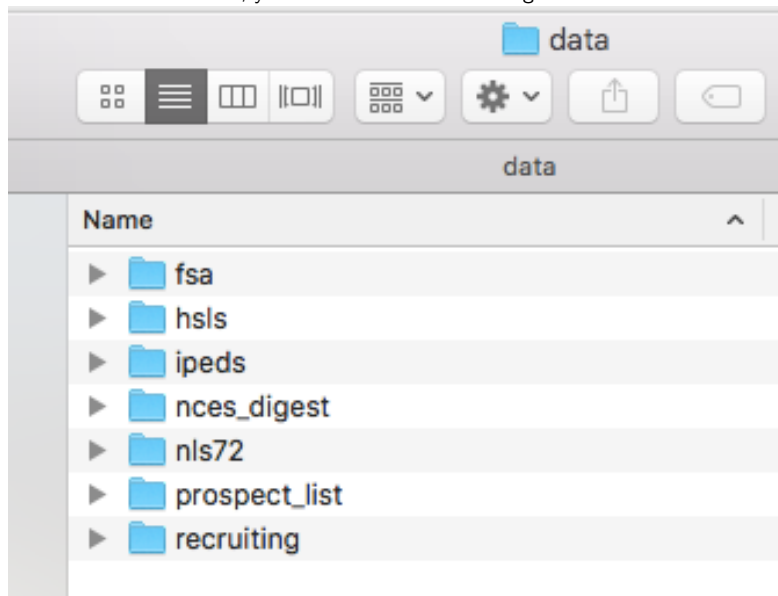
In lecture 1 we downloaded the rclass folder structure.

- ▶ You should have an rclass folder like this with a lectures and a data subfolder.



Rclass folder structure

- Inside the data subfolder, you should have the following subfolders:



Save data for lecture

We will not be providing links to data in this lecture. Please save data to the corresponding folders.

- ▶ First make sure to save **lecture10.Rmd** in your lectures folder
 - ▶ If you do not have class folder structure, you can download it [here](#)
- ▶ Download and unzip data folder we are using for this lecture [here](#)
- ▶ Save data files in the corresponding data folder
 - ▶ `ipeds_hd_2017_small.csv` should go in the **ic** folder inside the ipeds folder
 - ▶ `peps300.xlsx` should go in the **fsa** folder
 - ▶ `hsls_sch_small.dta` should go in the **hsls** folder

Working directory

- ▶ Your working directory will change depending on if you are using an R script or R markdown.
- ▶ **R script**
 - ▶ The default working directory is your computer's home directory
 - ▶ You only need to set your working directory `setwd()` in an R script once.
- ▶ **R markdown**
 - ▶ The default working directory is where the current R markdown file you are using is saved.
 - ▶ If you set your working directory in a code chunk, it will be reset to the default directory after the code chunk is finished running.
 - ▶ To avoid setting your working directory every time you read in data, you can read in data using a link to the data file or use the relative file path to save time.
- ▶ **R project**
 - ▶ The default working directory is in the R project location
 - ▶ If you are working with an R script in an R project, the working directory is the R project main folder (ex. `rclass`)
 - ▶ If you are working with an R markdown file in an R project, the working directory will be wherever the R markdown file is saved inside the R project.
 - ▶ For example, you have an R markdown file saved inside the lectures folder in an R project "rclass", your working directory may look like this ("`/Users/pm/Desktop/rclass/lectures`")

Absolute vs. relative filepath

Absolute file path: The absolute file path is the complete list of directories needed to locate a file or folder.

```
setwd("/Users/pm/Desktop/rclass/lectures/lecture8")
```

Relative file path: The relative file path is the path relative to your current location/directory. Assuming your current working directory is in the “lecture8” folder and you want to change your directory to the data folder, your relative file path would look something like this:

```
setwd("../..data")
```

File path shortcuts

| Key | Description |
|-------|---|
| ~ | tilde is a shortcut for mac user's home directory |
| ../ | moves up a level |
| ../.. | moves up two level |

Practice using relative file paths

1. Using the code chunk below, get your working directory `getwd()`
2. Using your relative file path, set your working directory to fsa folder inside the data folder. `setwd("filepath")`
3. Using your relative file path, set your working directory to the ef folder inside the ipeds folder inside the data folder. `setwd("filepath")`

Common data formats

Common data formats

| Format | Package | Function |
|-------------------------------|---------|------------|
| Comma-separated values (.csv) | readr | read_csv |
| Text-formated data (.txt) | readr | read_table |
| Tab-separated values (.tsv) | readr | read_tsv |
| Stata (.dta) | haven | read_dta |
| SPSS (.sav) | haven | read_sav |
| SAS (.sas) | haven | read_sas |
| Excel (.xls or .xlsx) | readxl | read_excel |
| R (.Rdata or .rds) | base R | load() |

► Source: Professor Darin Christensen

readr package

readr

The readr package is part of [tidyverse](#), which is designed to read in flat data files in R and transform them into data frames.

- We could load **library(tidyverse)** if we wanted to load all packages in tidyverse (e.g. ggplot2, dplyr, tidyr, stringr, readr, etc...)

```
library(tidyverse)
#> -- Attaching packages -----
#>  ggplot2 3.0.0      purrr   0.2.5
#>  tibble  1.3.4      dplyr   0.7.6
#>  tidyr   0.8.1      stringr 1.3.1
#>  readr   1.1.1      forcats 0.3.0
#> Warning: package 'tibble' was built under R version 3.3.2
#> Warning: package 'readr' was built under R version 3.3.2
#> -- Conflicts -----
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
```

- For the purpose of this lecture, we will just need to load **library(readr)**

readr functions

readr functions

`readr` (tidyverse) functions

Refer to the `readr` package [pdf](#) for more detailed information on each function.

- Note: All of these functions follow similar syntax/rules in `readr`

| Format | Function |
|------------------------------|-------------------------|
| Comma-separated values (csv) | <code>read_csv</code> |
| Semicolon separated files | <code>read_csv2</code> |
| Tab-separated values (tsv) | <code>read_tsv</code> |
| Any delimiter | <code>read_delim</code> |
| Fixed width files | <code>read_fwf</code> |
| Text-formated data (txt) | <code>read_table</code> |
| Web log files | <code>read_log</code> |

readr arguments

readr arguments

```
read_csv(file, col_names = TRUE, col_types = NULL, na = c("", "NA"), comment =  
write_csv(x, path, na = "NA", append = FALSE, col_names = !append)
```

Arguments

- ▶ **col_types** : One of NULL, a cols() specification, or a string:
 - ▶ If NULL, all column types will be imputed from the first 1000 rows on the input. This is convenient (and fast), but not robust. If the imputation fails, you'll need to supply the correct types yourself.
 - ▶ If a column specification created by cols(), it must contain one column specification for each column. If you only want to read a subset of the columns, use cols_only().
- ▶ **skip** : Number of lines to skip before reading data.
- ▶ **comment** : A string used to identify comments.
- ▶ **col_names** : Either TRUE, FALSE or a character vector of column names.
 - ▶ If TRUE, the first row of the input will be used as the column names, and will not be included in the data frame.
 - ▶ If FALSE, column names will be generated automatically: X1, X2, X3 etc.
 - ▶ If col_names is a character vector, the values will be used as the names of the columns, and the first row of the input will be read into the first row of the output data frame.
- ▶ **na** : Character vector of strings to use for missing values

readr column specification

read_csv column specification, part 1

`readr` is pretty good at guessing each column's data type by looking at the first 1,000 rows (e.g. character, double, etc.), however it is good practice to manually specify the data type for each column.

```
mtcars <- read_csv(readr_example("mtcars.csv"))
```

```
#> Parsed with column specification:
```

```
#> cols(
```

```
#>   mpg = col_double(),
```

```
#>   cyl = col_double(),
```

```
#>   disp = col_double(),
```

```
#>   hp = col_double(),
```

```
#>   drat = col_double(),
```

```
#>   wt = col_double(),
```

```
#>   qsec = col_double(),
```

```
#>   vs = col_double(),
```

```
#>   am = col_double(),
```

```
#>   gear = col_double(),
```

```
#>   carb = col_double()
```

```
#> )
```

read_csv column specification, part 1

The output of the previous example shows us the column specification readr gave us. However, we could manually change column specification if we do not like readr's guess.

```
mtcars <- read_csv(readr_example("mtcars.csv"), col_types =  
  cols(  
    mpg = col_double(),  
    cyl = col_integer(),  
    disp = col_double(),  
    hp = col_integer(),  
    drat = col_double(),  
    vs = col_integer(),  
    wt = col_double(),  
    qsec = col_double(),  
    am = col_character(),  
    gear = col_integer(),  
    carb = col_character()  
  )  
)
```

readr column specification, part 2

| Data type | Arguments |
|------------------------|---|
| Logical (TRUE & FALSE) | <code>col_logical()</code> |
| Integers | <code>col_integer()</code> |
| Doubles | <code>col_double()</code> |
| Dates | <code>col_date(format = "")</code> |
| Characters | <code>col_character()</code> |
| Numbers | <code>col_numeric()</code> |
| Factors | <code>col_factors(levels, ordered)</code> |

```
read_csv("a, b, c
1,2,F
4,5,T", col_types =
  cols(
    a = col_factor(c("1", "2", "3", "4")),
    b = col_character(),
    c = col_logical()
  ))

#> # A tibble: 2 x 3
#>   a     b     c
#>   <fct> <chr> <lgl>
#> 1 1     2 FALSE
#> 2 4     5  TRUE
```


read_csv demonstration csv

`readr` automatically treats the first line of data as column names.

```
read_csv("column 1, column 2, column 3  
1,-2,3  
4,5,6", na = c(-2, "NA"))  
)
```

```
#> # A tibble: 2 x 3  
#>   `column 1` `column 2` `column 3`  
#>       <dbl>       <dbl>       <dbl>  
#> 1         1         NA         3  
#> 2         4         5         6
```

There are instances where you may want to tell R from what line to begin reading in data.

read_csv skip argument

Notice the example below. The first two lines are comments about the data. We would need to use **skip = n** to skip n lines.

```
read_csv("This file contains data on student charges for the academic year.  
File name: IC2016_AY  
a, b, c  
1,2,3  
4,5,6", skip = 2  
)  
#> # A tibble: 2 x 3  
#>       a       b       c  
#>   <dbl> <dbl> <dbl>  
#> 1     1     2     3  
#> 2     4     5     6
```

read_csv comment argument

We could also tell R to drop lines we specify as comments. With **comment = n**

```
read_csv("# This file contains data on student charges for the academic year.  
  a, b, c  
  1,2,3  
  4,5,6", comment = "#"  
)  
#> # A tibble: 2 x 3  
#>       a     b     c  
#>   <dbl> <dbl> <dbl>  
#> 1     1     2     3  
#> 2     4     5     6
```

```
read_csv("* This file contains data on student charges for the academic year.  
  a, b, c  
  1,2,3  
  4,5,6", comment = "*"  
)  
#> # A tibble: 2 x 3  
#>       a     b     c  
#>   <dbl> <dbl> <dbl>  
#> 1     1     2     3  
#> 2     4     5     6
```

read_csv column names argument

We could tell R there are no column names with `col_names = FALSE` or we could manually give R column names with `col_names = c("", "", "")`

```
read_csv("1,2,3
         4,5,6", col_names = FALSE
         )
```

```
#> # A tibble: 2 x 3
#>       X1     X2     X3
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
#> 2     4     5     6
```

```
read_csv("1,2,3
         4,5,6", col_names = c("column 1", "column 2", "column 3")
         )
```

```
#> # A tibble: 2 x 3
#>   `column 1` `column 2` `column 3`
#>       <dbl>      <dbl>      <dbl>
#> 1         1         2         3
#> 2         4         5         6
```

```
read_csv("a,b,c
         1,2,3
         4,5,6", col_names = c("column 1", "column 2", "column 3"), skip = 1
         )
```

```
#> # A tibble: 2 x 3
#>   `column 1` `column 2` `column 3`
#>       <dbl>      <dbl>      <dbl>
```

read_csv Student exercise

1. Create a 3x3 tibble like the examples above (e.g. `read_csv("a,b,c...")`), treating the first line as column names
2. Now on the first line add a sentence and use the `skip` argument to skip this line
3. This time add a special character (`*`, `#`, `!`) at the beginning of the sentence and indicate it is a comment
4. Delete the sentence and column names (should have a 2x2 tibble) and manually tell R column names

read_csv Student exercise solutions

1. Create a 3x3 tibble like the examples above (e.g. `read_csv("a,b,c...")`), treating the first line as column names

```
read_csv("a, b, c
1,2,3
4,5,6"
)
#> # A tibble: 2 x 3
#>       a     b     c
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
#> 2     4     5     6
```

2. Now on the first line add a sentence and use the `skip` argument to skip this line

```
read_csv("Do not read this sentence
a, b, c
1,2,3
4,5,6", skip = 1
)
#> # A tibble: 2 x 3
#>       a     b     c
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
#> 2     4     5     6
```

read_csv Student exercise solutions continued

3. This time add a special character (*, #, !) at the beginning of the sentence and indicate it is a comment

```
read_csv("#This is a comment
         a, b, c
         1,2,3
         4,5,6",comment = "#")
#> # A tibble: 2 x 3
#>       a     b     c
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
#> 2     4     5     6
```

4. Delete the sentence and column names (should have a 2x3 tibble) and manually tell R column names

```
read_csv("1,2,3
         4,5,6",col_names = c("a", "b", "c"))
#> # A tibble: 2 x 3
#>       a     b     c
#>   <dbl> <dbl> <dbl>
#> 1     1     2     3
#> 2     4     5     6
```

1. readr Running into errors

1. Make sure you have downloaded and saved the flat file
2. Make sure to know the file path of where data is downloaded or saved (“../../data”)
 - ▶ You can use the `list.files()` function to view what files you have in a particular folder
3. Make sure you set your working `setwd()` directory in R. To check your current working directory type `getwd()` in console.

read_csv demonstration using IPEDS data

Integrated Postsecondary Education Data System (IPEDS)

- ▶ Postsecondary education data from NCES
- ▶ There are [12 survey components](#) and 3 collection periods

We will be working with [Institutional Characteristics data of 2017](#)

read_csv demonstration using IPEDS data

Tying it all together

Use `read_csv()` function from `readr` to import csv dataset into R without column specification. Follow along on your computers.

```
ipeds <- read_csv(file="../../data/ipeds/ic/ipeds_hd_2017_small.csv")
#> Parsed with column specification:
#> cols(
#>   unitid = col_double(),
#>   instnm = col_character(),
#>   stabbr = col_character(),
#>   sector = col_double(),
#>   iclevel = col_double(),
#>   control = col_double()
#> )
# glimpse(ipeds)
```

read_csv demonstration using IPEDS data

Use `read_csv()` function from `readr` to import csv dataset into R with column specification

```
ipeds <- read_csv(file="../../data/ipeds/ic/ipeds_hd_2017_small.csv",  
                  col_types =  
                    cols(  
                      unitid = col_character(),  
                      instnm = col_character(),  
                      stabbr = col_character(),  
                      sector = col_integer(),  
                      iclevel = col_integer(),  
                      control = col_integer()  
                    )  
)
```

We changed unitid to character type, but could be left as is or changed to double type for example.

readr variable and value labels

Let's view variable and value labels

```
ipeds %>% select(sector) %>% var_label()
#> $sector
#> NULL
ipeds %>% select(sector) %>% val_labels()
#> $sector
#> NULL
```

- ▶ There are no variable and value labels for this data. IPEDS has a separate do file with variable and value labels.
- ▶ Let's practice manually adding variable and value labels using the `labelled` package.

Set variable and value labels

Set variable and value labels

Let's create a tibble first

```
df <- tribble(
  ~id, ~edu, ~sch,
  #--/--/----
  1, 2, 2,
  2, 1, 1,
  3, 3, 2,
  4, 4, 2,
  5, 1, 2
)
```

df

```
#> # A tibble: 5 x 3
#>       id    edu  sch
#>   <dbl> <dbl> <dbl>
#> 1     1     2     2
#> 2     2     1     1
#> 3     3     3     2
#> 4     4     4     2
#> 5     5     1     2
```

Set variable labels

- ▶ Use `set_variable_labels` function to manually set variable labels
 - ▶ `df %>% set_variable_labels(variable = "Variable label")`
 - ▶ `ipeds %>% set_variable_labels(unitid = "Unit identification number")`

```
class(df$sch)
#> [1] "numeric"
df <- df %>%
  set_variable_labels(
    id = "Unique identification number",
    edu = "Education level",
    sch = "Type of school attending"
  )
class(df$sch)
#> [1] "numeric"
```

Set value labels

- ▶ Use `set_value_labels` function to manually set value labels



```
df %>% set_value_labels(variable = c("Value label" = 1, "Value label" = 2))
```



```
ipeds %>% set_value_labels(sector = "Administrative Unit" = 0)
```

```
class(df$sch)
#> [1] "numeric"
df <- df %>%
  set_value_labels(
    edu = c("High School" = 1,
            "AA degree" = 2,
            "BA degree" = 3,
            "MA or higher" = 4),
    sch = c("Private" = 1,
            "Public" = 2))
class(df$sch)
#> [1] "haven_labelled"
```


readr labelled data

- ▶ Open the data dictionary file for [hd2017](#) data and select “Frequencies” sheet
- ▶ We are only working with these 6 variables (unitid, instnm, stabbr, sector, iclevel, control)

```
# Lets view values for sector
```

```
ipeds %>%
```

```
  count(sector)
```

```
#> # A tibble: 11 x 2
```

```
#>   sector      n
```

```
#>   <int> <int>
```

```
#> 1      0    75
```

```
#> 2      1   775
```

```
#> 3      2  1701
```

```
#> 4      3   661
```

```
#> 5      4   981
```

```
#> 6      5   169
```

```
#> 7      6   864
```

```
#> 8      7   248
```

```
#> 9      8    85
```

```
#> 10     9  1562
```

```
#> 11     99    32
```

readr variable and value labels student exercise

1. Using the codebook for hd2017 data, add variable labels for all 6 variables (unitid, instnm, stabbr, sector, iclevel, control) and save to new object `ipeds_labelled`
2. Add value labels for sector, iclevel, and control for `ipeds_labelled`

hint: you can do both tasks within the same piping

readr variable and value labels student exercise solution

Need to manually assign variable and value labels using labelled package

```
ipeds_labelled <- ipeds %>%  
  set_variable_labels(unitid = "Unit identification number",  
    instnm = "Institution name",  
    stabbr = "State abbreviation",  
    sector = "Sector of institution",  
    iclevel = "Level of institution",  
    control = "Control of institution") %>%  
  set_value_labels(sector = c("Administrative Unit" = 0,  
    "Public, 4-year or above" = 1,  
    "Private not-for-profit, 4-year or above" = 2,  
    "Private for-profit, 4-year or above" = 3,  
    "Public, 2-year" = 4,  
    "Private not-for-profit, 2-year" = 5,  
    "Private for-profit, 2-year" = 6,  
    "Public, less-than 2-year" = 7,  
    "Private not-for-profit, less-than 2-year" = 8,  
    "Private for-profit, less-than 2-year" = 9,  
    "Sector unknown (not active)" = 99),  
    iclevel = c("Four or more years" = 1,  
    "At least 2 but less than 4 years" = 2,  
    "Less than 2 years (below associate)" = 3,  
    "{Not available}" = -3),  
    control = c("Public" = 1, "Private not-for-profit" = 2,  
    "Private for-profit" = 3,  
    "{Not available}" = -3))
```

readr Let's view new labelled data

```
typeof(ipeds_labelled$iclevel)
#> [1] "integer"
class(ipeds_labelled$iclevel)
#> [1] "haven_labelled"
attributes(ipeds_labelled$iclevel)
#> $label
#> [1] "Level of institution"
#>
#> $labels
#>           Four or more years   At least 2 but less than 4 years
#>                             1                             2
#> Less than 2 years (below associate)   {Not available}
#>                             3                             -3
#>
#> $class
#> [1] "haven_labelled"
```

`as_factor()` function

Usage (i.e., syntax): `as_factor(x, ..., only_labelled = TRUE)`

Arguments

- ▶ `x` : object to coerce to a factor
- ▶ `...` : other arguments passed down to method
- ▶ `only_labelled` : only apply to labelled columns
- ▶ `levels` : How to create the levels of the generated factor:
 - ▶ "default": uses labels where available, otherwise the values. Labels are sorted by value.
 - ▶ "both": like "default", but pastes together the level and value
 - ▶ "label": use only the labels; unlabelled values become NA labelled 3
 - ▶ "values": use only the values
- ▶ `ordered` : If TRUE create an ordered (ordinal) factor, if FALSE (the default) create a regular (nominal) factor

Example: Change labelled columns to factor

```
as_factor(df, only_labelled = TRUE)
```

readr labelled data to factor

Let's change class to factor

```
ipeds_factor <- as_factor(ipeds_labelled, only_labelled = TRUE, levels="both")
typeof(ipeds_factor$sector)
#> [1] "integer"
class(ipeds_factor$sector)
#> [1] "factor"
attributes(ipeds_factor$sector)
#> $levels
#> [1] "[0] Administrative Unit"
#> [2] "[1] Public, 4-year or above"
#> [3] "[2] Private not-for-profit, 4-year or above"
#> [4] "[3] Private for-profit, 4-year or above"
#> [5] "[4] Public, 2-year"
#> [6] "[5] Private not-for-profit, 2-year"
#> [7] "[6] Private for-profit, 2-year"
#> [8] "[7] Public, less-than 2-year"
#> [9] "[8] Private not-for-profit, less-than 2-year"
#> [10] "[9] Private for-profit, less-than 2-year"
#> [11] "[99] Sector unknown (not active)"
#>
#> $class
#> [1] "factor"
#>
#> $label
#> [1] "Sector of institution"
```

Reading in data from the web

Reading in data from the web

- ▶ Save time
 - ▶ Reduce the steps of downloading, saving, and reading in data
 - ▶ Read in data directly from the internet
 - ▶ **note** not all packages will work with downloading data from the web (`read_excel`)

For example, rather than downloading Raj Chetty data and saving it in a folder, we could download the data directly from the web.

Reading in data from web example using Raj Chetty data

Equality of Opportunity Project

- ▶ [Equality of Opportunity Project](#) uses two data sources– federal tax records and Department of Education records (1999-2013)– to investigate intergenerational income mobility at colleges in the US.

We will use [Mobility Report Cards: The Role of Colleges in Intergenerational Mobility data](#)

Reading in data from web example using Raj Chetty data

1. Follow this [link](#) and under the “Mobility Report Cards...” tab select “click to view data”.
2. Choose “Online Data Table 1”
3. Right click and copy link address for “Excel” (Note: it is actually a csv file)

Mobility Report Cards: The Role of Colleges in Intergenerational Mobility

Chetty, Friedman, Saez, Turner, and Yagan (2017)

Mobility Statistics and Student Outcomes by College and Birth Cohort

[Click to view data](#) ▼

| Data Description | Download | | |
|--|-----------------------|-----------------------|------------------------|
| | Stata | Excel | Readme |
| Online Data Table 1 Preferred Estimates of Access and Mobility Rates by College | Stata | Excel | Readme |
| Online Data Table 2 Baseline Cross-Sectional Estimates by College | Stata | Excel | Readme |
| Online Data Table 3 Baseline Longitudinal Estimates by College and Child's Cohort | Stata | Excel | Readme |

Reading in data from the web

Approach #1

```
#Paste url to excel "csv" file
```

```
data_url <- "http://www.equality-of-opportunity.org/data/college/mrc_table1.csv"
```

```
#Download data and read in using read_csv (readr)
```

```
mrc <- read_csv(data_url)
```

```
#View first 4 rows and 4 columns
```

```
mrc[1:4, 1:4]
```

```
#> # A tibble: 4 x 4
```

```
#>   super_opecid name
```

```
#>   <dbl> <chr>
```

```
czname state
```

```
<chr> <chr>
```

```
#> 1      2665 Vaughn College Of Aeronautics And Technology New York NY
```

```
#> 2      7273 CUNY Bernard M. Baruch College New York NY
```

```
#> 3      2688 City College Of New York - CUNY New York NY
```

```
#> 4      7022 CUNY Lehman College New York NY
```

Reading in data from the web

Approach #2

```
#Download data and read in link directly using read_csv (readr)
mrc <- read_csv("http://www.equality-of-opportunity.org/data/college/mrc_table1")
#> Parsed with column specification:
#> cols(
#>   super_opeid = col_double(),
#>   name = col_character(),
#>   czname = col_character(),
#>   state = col_character(),
#>   par_median = col_double(),
#>   k_median = col_double(),
#>   par_q1 = col_double(),
#>   par_top1pc = col_double(),
#>   kq5_cond_parq1 = col_double(),
#>   ktop1pc_cond_parq1 = col_double(),
#>   mr_kq5_pq1 = col_double(),
#>   mr_ktop1_pq1 = col_double(),
#>   trend_parq1 = col_double(),
#>   trend_bottom40 = col_double(),
#>   count = col_double()
#> )
```

Reading in data from the web

```
#View first 4 rows and 4 columns
```

```
mrc[1:4, 1:4]
```

```
#> # A tibble: 4 x 4
```

```
#>   super_opecid name                                czname    state
```

```
#>   <dbl> <chr>                                <chr>    <chr>
```

```
#> 1      2665 Vaughn College Of Aeronautics And Technology New York NY
```

```
#> 2      7273 CUNY Bernard M. Baruch College                New York NY
```

```
#> 3      2688 City College Of New York - CUNY                New York NY
```

```
#> 4      7022 CUNY Lehman College                          New York NY
```

Problems downloading data (zip files) using IPEDS

1. Follow this [link](#) and under the “Survey Data” tab select “Complete data files”.
2. Choose “All years” and “All surveys” and click continue
3. Right click and copy link address for “IC2017_AY”

Years & Surveys

All years ▾ All surveys ▾ **Continue**

Data files are available in ZIP format.

| Year | Survey | Title | Data File | Stata Data File | Programs | Dictionary |
|------|-------------------------------|--|---------------------------|---------------------------------|----------------------------------|----------------------------|
| 2017 | Institutional Characteristics | Directory information (Preliminary) | HD2017 | HD2017_STATA | SPSS, SAS, STATA | Dictionary |
| 2017 | Institutional Characteristics | Educational offerings, organization, services and athletic associations (Preliminary) | IC2017 | IC2017_STATA | SPSS, SAS, STATA | Dictionary |
| 2017 | Institutional Characteristics | Student charges for academic year programs (Preliminary) | IC2017_AY | IC2017_AY_STATA | SPSS, SAS, STATA | Dictionary |

Figure 1: ipeds data center

Downloading data (zip files) using IPEDS

► Paste url and read in using `read_csv`

► `ipeds <- read_csv("link")` What happens when you try reading in this file?

Need to download **and** unzip

download.file function

download.file function

`download.file` is a function use to download a file from the internet.

Usage (i.e., syntax):

```
download.file(url, destfile, method, quiet = FALSE, mode = "w")
```

Arguments

- ▶ `url` : A character string naming the URL of a resource to be downloaded
- ▶ `destfile` : A character string with the name where the downloaded file is saved.
- ▶ `method` : Method to be used for downloading files. Current download methods are "internal", "wininet" (Windows only) "libcurl", "wget" and "curl", and there is a value "auto"
- ▶ `quiet` : If TRUE, suppress status messages (if any), and the progress bar.
- ▶ `mode` : character. The mode with which to write the file. Useful values are "w", "wb" (binary), "a" (append) and "ab". Not used for methods "wget" and "curl"
- ▶ `cacheOK` : logical. Is a server-side cached value acceptable?
- ▶ `extra` : character vector of additional command-line arguments for the "wget" and "curl" methods.
- ▶ `...` : allow additional arguments to be passed, unused

download.file function

Details: The function `download.file` can be used to download a single file as described by url from the internet and store it in `destfile`. The url must start with a scheme such as [http://](#), [https://](#), [ftp://](#) or [file://](#).

Example: Download data from the web

```
#Data from the Equality of Opportunity Project
```

```
download.file(url = "http://www.equality-of-opportunity.org/data/college/mrc_ta  
             destfile = "table_1.csv")
```

unzip files

`unzip` is used to extract files from or list a zip archive.

Usage (i.e., syntax):

```
unzip(zipfile, files = NULL, list = FALSE, overwrite = TRUE, unzip = "internal")
```

Arguments

- ▶ `zipfile` : The pathname of the zip file
- ▶ `files` : A character vector of recorded filepaths to be extracted: the default is to extract all files.
- ▶ `list` : If TRUE, list the files and extract none.
- ▶ `overwrite` : If TRUE, overwrite existing files, otherwise ignore such files.
- ▶ `junkpaths` : If TRUE, use only the basename of the stored filepath when extracting.
- ▶ `exdir` : The directory to extract files to
- ▶ `unzip` : The method to be used.

Downloading data (zip files) using IPEDS

```
#Set path to where data will be saved
#download file
#getwd()
setwd("../..../data/ipeds/ic")
download.file("https://nces.ed.gov/ipeds/datacenter/data/IC2018_AY.zip",
              destfile = "ic2018_ay", mode = "wb")
#unzip zip file
unzip(zipfile = "ic2018_ay" , unzip = "unzip")

ic2018_ay <- read_csv("ic2018_ay.csv") %>%
  select(-starts_with("X"))
#> Parsed with column specification:
#> cols(
#>   .default = col_character(),
#>   UNITID = col_double()
#> )
#> See spec(...) for full column specifications.

names(ic2018_ay) <- tolower(names(ic2018_ay)) #lowercase column names

#names(ic2018_ay)
```

Student exercise

Tying it all together

- ▶ Using everything we learned today, read in a csv data file from the web
- ▶ Go back to the ipeds data center [here](#)
- ▶ Right click and copy the link address to a different data file ("HD2017", "EFFY2017")
- ▶ Make sure to download the link first (download.file) before reading in the data
- ▶ Change column names to lowercase `names(df) <- tolower(names(df))`
- ▶ Report dimensions of data `dim(df)`
- ▶ Create a subset of your data (filter, select, etc.)

readxl package

The `readxl` package is part of [tidyverse](#), which is designed to read data from Excel and into R.

- ▶ We could install tidyverse `install.package("tidyverse")` to access `readxl`, but have to explicitly load `readxl` because it is not a core tidyverse package
 - ▶ `library(readxl)`
- ▶ Or install `readxl` `install.packages("readxl")` and load it
 - ▶ `library(readxl)`
- ▶ For the purpose of this lecture, we just need to load `library(readxl)`.

readxl

`readxl` supports both `.xls` and `.xlsx` formats and is designed to work with tabular data. It does not require dependencies— making installing and operating fairly simple.

`readxl` has several example files where we could use as practice. The files include:

```
readxl_example()
#> [1] "clippy.xls"      "clippy.xlsx"    "datasets.xls"   "datasets.xlsx"
#> [5] "deaths.xls"      "deaths.xlsx"    "geometry.xls"   "geometry.xlsx"
#> [9] "type-me.xls"     "type-me.xlsx"
```

For now, lets use "datasets.xlsx"

```
excel_example <- readxl_example("datasets.xlsx")
```


readxl arguments

readxl arguments

Refer to the `readxl` package [pdf](#) for more detailed information on each function.

Arguments

- ▶ `sheet` : Sheet to read. Either a string (the name of a sheet), or an integer (the position of the sheet).
- ▶ `n_max` : Maximum number of data rows to read
- ▶ `range` : A cell range to read from
- ▶ `cell_rows` : Cell rows to read from
- ▶ `cell_cols` : Cell columns to read from
- ▶ `na` : Character vector of strings to interpret as missing values

readxl sheet

Select the excel sheet you want to work with.

```
#To view sheets in excel file
```

```
excel_sheets(excel_example)
```

```
#> [1] "iris"      "mtcars"    "chickwts"  "quakes"
```

```
xl_example <- read_excel(excel_example, sheet = "quakes")
```

```
head(xl_example)
```

```
#> # A tibble: 6 x 5
```

```
#>   lat long depth mag stations
```

```
#>   <dbl> <dbl> <dbl> <dbl>   <dbl>
```

```
#> 1 -20.4 182. 562 4.8      41
```

```
#> 2 -20.6 181. 650 4.2      15
```

```
#> 3 -26   184. 42 5.4      43
```

```
#> 4 -18.0 182. 626 4.1      19
```

```
#> 5 -20.4 182. 649 4       11
```

```
#> 6 -19.7 184. 195 4       12
```

readxl n_max

Maximum number of rows to read

```
read_excel(excel_example, sheet = "quakes", n_max = 3)
```

```
#> # A tibble: 3 x 5
```

```
#>   lat long depth mag stations
```

```
#>   <dbl> <dbl> <dbl> <dbl>   <dbl>
```

```
#> 1 -20.4 182. 562 4.8      41
```

```
#> 2 -20.6 181. 650 4.2      15
```

```
#> 3 -26   184. 42 5.4      43
```

readxl range

A cell range to read from

```
read_excel(excel_example, sheet = "quakes", range = "C1:E4")
```

```
#> # A tibble: 3 x 3
```

```
#>   depth  mag stations
```

```
#>   <dbl> <dbl>   <dbl>
```

```
#> 1   562   4.8     41
```

```
#> 2   650   4.2     15
```

```
#> 3    42   5.4     43
```

```
read_excel(excel_example, sheet = "quakes", range = cell_rows(1:3))
```

```
#> # A tibble: 2 x 5
```

```
#>   lat long depth  mag stations
```

```
#>   <dbl> <dbl> <dbl> <dbl>   <dbl>
```

```
#> 1 -20.4 182.   562   4.8     41
```

```
#> 2 -20.6 181.   650   4.2     15
```

```
head(read_excel(excel_example, sheet = "quakes", range = cell_cols("A:C")))
```

```
#> # A tibble: 6 x 3
```

```
#>   lat long depth
```

```
#>   <dbl> <dbl> <dbl>
```

```
#> 1 -20.4 182.   562
```

```
#> 2 -20.6 181.   650
```

```
#> 3 -26   184.   42
```

```
#> 4 -18.0 182.   626
```

```
#> 5 -20.4 182.   649
```

```
#> 6 -19.7 184.   195
```

```
#> #> main: head() to only view first 6 rows
```

Character vector of strings to interpret as missing values

```
read_excel(excel_example, sheet = "quakes", na = "-20.42")
```

```
#> # A tibble: 1,000 x 5
```

```
#>       lat  long depth  mag stations
```

```
#>   <dbl> <dbl> <dbl> <dbl>   <dbl>
```

```
#> 1  NA    182.   562  4.8     41
```

```
#> 2 -20.6  181.   650  4.2     15
```

```
#> 3 -26    184.    42  5.4     43
```

```
#> 4 -18.0  182.   626  4.1     19
```

```
#> 5  NA    182.   649  4      11
```

```
#> 6 -19.7  184.   195  4      12
```

```
#> 7 -11.7  166.    82  4.8     43
```

```
#> 8 -28.1  182.   194  4.4     15
```

```
#> 9 -28.7  182.   211  4.7     35
```

```
#> 10 -17.5  180.   622  4.3     19
```

```
#> # ... with 990 more rows
```

read_excel using FSA data

Federal Student Aid

- ▶ Federal Student Aid Data Center provides information for federal assistance programs and is divided into four categories:
 - Student Aid Data
 - School Data
 - Federal Family Education Loan (FFEL) Program
 - Business Information Resources

We will be working with [School Data](#)

read_excel student exercise using FSA data

Read in Federal Student Aid data using `readxl` function

- Excel file "peps300.xlsx" is saved in the fsa folder inside the data folder

1. Use relative file path to read in data

▶ `read_excel("filepath/excelfile")`

2. Read in first four rows (`n_max`)

3. Read in columns from range Names to State **hint** `cell_cols`

4. Set value "P" to missing (na) **note** : you need to investigate in detail before setting anything to missing

read_excel Student exercise solutions

1. Use relative file path to read in data

```
#Read in data using readxl function
#getwd()
fsa <- read_excel("../../data/fsa/peps300.xlsx")
```

2. Read in first four rows (n_max)

```
#Read in first four rows (n_max)
#setwd()
read_excel("../../data/fsa/peps300.xlsx", n_max = 4)
#> # A tibble: 4 x 29
#>   OPEID Name Address City State `State Desc` `Zip Code` `Zip Ext`
#>   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
#> 1 0010~ ALAB~ 4900 M~ NORM~ AL ALABAMA 35762 1357
#> 2 0010~ FAUL~ 5345 A~ MONT~ AL ALABAMA 36109 3398
#> 3 0010~ UNIV~ PALMER~ MONT~ AL ALABAMA 35115 6000
#> 4 0010~ ALAB~ 915 SO~ MONT~ AL ALABAMA 36104 5714
#> # ... with 21 more variables: `Prog|r\nLength` <dbl>,
#> # `School|r\nType` <dbl>, `Year 1` <dbl>, `Dual|r\nNum 1` <dbl>,
#> # `Dual|r\nDenom 1` <dbl>, `DRate 1` <dbl>, `PRate 1` <chr>, `Ethnic
#> # Code` <chr>, Program <chr>, `Cong Dis` <chr>, Region <chr>, `Year
#> # 2` <dbl>, `Dual|r\nNum 2` <dbl>, `Dual|r\nDenom 2` <dbl>, `DRate
#> # 2` <dbl>, `PRate 2` <chr>, `Year 3` <dbl>, `Dual|r\nNum 3` <dbl>,
#> # `Dual|r\nDenom 3` <dbl>, `DRate 3` <dbl>, `PRate 3` <chr>
```

read_excel Student exercise solutions continued

3. Read in columns from range Names to State **hint** cell_cols

```
#Read in column Names to column State
#setwd()
head(read_excel("../data/fsa/peps300.xlsx", range = cell_cols("B:E")))
```

#> # A tibble: 6 x 4

| #> | Name | Address | City | State |
|------|------------------------------------|----------------------|-------------|-------|
| #> | <chr> | <chr> | <chr> | <chr> |
| #> 1 | ALABAMA AGRICULTURAL & MECHANICAL~ | 4900 MERIDIAN STREET | NORMAL | AL |
| #> 2 | FAULKNER UNIVERSITY | 5345 ATLANTA HIGHWAY | MONTGOMERY | AL |
| #> 3 | UNIVERSITY OF MONTEVALLO | PALMER CIRCLE | MONTEVALLO | AL |
| #> 4 | ALABAMA STATE UNIVERSITY | 915 SOUTH JACKSON S~ | MONTGOMERY | AL |
| #> 5 | CENTRAL ALABAMA COMMUNITY COLLEGE | 1675 CHEROKEE ROAD | ALEXANDER ~ | AL |
| #> 6 | ATHENS STATE UNIVERSITY | 300 NORTH BEATY STR~ | ATHENS | AL |

read_excel Student exercise solutions continued

4. Set value "P" to missing (na) **note** : you need to investigate in detail before setting anything to missing

```
#setwd()
read_excel("../../data/fsa/peps300.xlsx", n_max = 4, na = "P")
#> # A tibble: 4 x 29
#>   OPEID Name Address City State `State Desc` `Zip Code` `Zip Ext`
#>   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
#> 1 0010~ ALAB~ 4900 M~ NORM~ AL ALABAMA 35762 1357
#> 2 0010~ FAUL~ 5345 A~ MONT~ AL ALABAMA 36109 3398
#> 3 0010~ UNIV~ PALMER~ MONT~ AL ALABAMA 35115 6000
#> 4 0010~ ALAB~ 915 SO~ MONT~ AL ALABAMA 36104 5714
#> # ... with 21 more variables: `Prog\r\nLength` <dbl>,
#> # `School\r\nType` <dbl>, `Year 1` <dbl>, `Dual\r\nNum 1` <dbl>,
#> # `Dual\r\nDenom 1` <dbl>, `DRate 1` <dbl>, `PRate 1` <chr>, `Ethnic
#> # Code` <chr>, Program <chr>, `Cong Dis` <chr>, Region <chr>, `Year
#> # 2` <dbl>, `Dual\r\nNum 2` <dbl>, `Dual\r\nDenom 2` <dbl>, `DRate
#> # 2` <dbl>, `PRate 2` <chr>, `Year 3` <dbl>, `Dual\r\nNum 3` <dbl>,
#> # `Dual\r\nDenom 3` <dbl>, `DRate 3` <dbl>, `PRate 3` <chr>
```

readxl Running into problems

1. Make sure you have downloaded and saved excel file
2. Make sure to know the file path of where data is downloaded or saved
("../../data")
 - ▶ `list.files()`
3. Make sure you set your working `setwd()` directory in R. To check your current working directory type `getwd()` in console.
4. Make sure to choose the correct sheet (if applicable)
5. Pay attention to column names when setting range

haven package

Recap from Augmented Vectors Lecture:

haven is part of [tidyverse](#), which enables users to import and export data from the following statistical packages:

- ▶ SAS
- ▶ SPSS
- ▶ Stata

Similar to `readr`, we could load the entire **library(tidyverse)** package to get `haven`. For the purpose of this lecture, we will just need to load **library(haven)**.

haven functions

Refer to the `haven` package [pdf](#) for more detailed information on each function.

haven's (tidyverse) functions

| Format | Function |
|--------|-----------------------|
| SPSS | <code>read_sav</code> |
| SAS | <code>read_sas</code> |
| Stata | <code>read_dta</code> |

haven read and write Stata arguments

```
read_dta(file, encoding = NULL)
```

```
write_dta(data, path, version = 14)
```

Arguments

- ▶ `file` : file path to data
- ▶ `encoding` : the character encoding used for the file
- ▶ `data` : data frame to save (write)
- ▶ `path` : file path to where data will be saved
- ▶ `version` : file version to use. Supports versions 8-15.

[Link](#)

haven using HSLs data

High school longitudinal surveys from National Center for Education Statistics (NCES)

- ▶ Follow U.S. students from high school through college, labor market

We will be working with [High School Longitudinal Study of 2009 \(HSLs:09\)](#)

- ▶ Follows 9th graders from 2009
- ▶ Data collection waves - Base Year (2009) - First Follow-up (2012) - 2013 Update (2013) - High School Transcripts (2013-2014) - Second Follow-up (2016)

haven Student exercise using HSLS data

1. Use `read_dta()` function from `haven` to import Stata dataset into R
2. Use `write_dta()` function from `haven` to save Stata dataset
3. If you have time, explore data (View, glimpse, head, etc.)
 - ▶ View variable and value labels
 - ▶ Change class == labelled to class == factor

haven Student exercise solutions

Use `read_dta` function from `haven` to import State data

```
hsls <- read_dta("../..data/hsls/hsls_sch_small.dta", encoding=NULL)
```

```
# View data
```

```
head(hsls)
```

```
glimpse(hsls)
```

Use `write_dta` function from `haven` to write State data

```
write_dta(dataframe, path = "")
```

```
write_dta(hsls, path = "../..data/hsls/hsls_sch_small.dta")
```

haven Student exercise Solution continued...

Variable and Value labels

```
# View variable labels
hs1s %>% var_label()
#> $sch_id
#> [1] "School ID"
#>
#> $x1control
#> [1] "X1 School control"
#>
#> $x1locale
#> [1] "X1 School locale (urbanicity)"
#>
#> $x1region
#> [1] "X1 School geographic region"
#>
#> $a1schcontrol
#> [1] "A1 A02 School control"
```

haven Student exercise Solution cont...

```
#View value label for x1locale
hsls %>% select(x1locale) %>% val_labels()
#> $x1locale
#>                                     Missing
#>                                     1
#> Unit non-response/component not applicable
#>                                     2
#>                                     Item legitimate skip/NA
#>                                     3
#>                                     City
#>                                     4
#> Suburb
#>                                     5
#> Town
#>                                     6
#> Rural
#>                                     7
```

haven Student exercise Solution cont...

```
# Change class == labelled to class == factor
hsls <- as_factor(hsls, only_labelled = TRUE)

typeof(hsls$x1region)
#> [1] "integer"
class(hsls$x1region)
#> [1] "factor"
attributes(hsls$x1region)
#> $levels
#> [1] "Missing"
#> [2] "Unit non-response/component not applicable"
#> [3] "Item legitimate skip/NA"
#> [4] "Northeast"
#> [5] "Midwest"
#> [6] "South"
#> [7] "West"
#>
#> $class
#> [1] "factor"
#>
#> $label
#> [1] "X1 School geographic region"
```