Lesson 09 - Getting data into R

Last Updated 09-20-2020

Introduction

In this lesson we will introduce ways to get data from external files into R, how R works with data, and how to use functions to explore your data frame.

Student Learning Objectives

After completing this lesson learners will be able to

- Import data that is stored in an external Excel, CSV or tab-delimited text file into R.
- Compare and contrast the three file types discussed in this lesson.
- Use functions to examine data objects to ensure data was imported correctly.

Prior to this lesson learners should

- Download the [09_import_notes.Rmd] R markdown file and save into your `Math130/notes` folder.
- Download the three data sets listed on the Week 4 overview below.

File types

In this lesson we are only going to explore reading files that exist on your computer into R from three most commonly used data sources: A tab-delimited text file, A CSV file and an Excel file.

The three different files we will be using have different file types, or extensions.

- `email.txt` is a `.txt` or "text" file.
- `NCBirths.csv` is a `.csv` or "comma separated values" file.
- `fatal-police-shootings-data.xlsx` is a `.xlsx` or Excel file.

Each of these file types differ in the type of *delimiter* used. The *delimiter* is a character or symbol that separates columns of data from each other when stored in an external file. Recall back to the earlier lesson on data frames and matrices. Each column in the matrix represented data on a specific variable. Something had to tell R how to distinguish which values went with which variable.

There are two main types of delimiters we will consider in this class; comma and tab. That does not mean that data can't be stored in other ways, these are just the two most common.

Each of these different data types requires a different function or mechanism to import the data into R. If you use the wrong mechanism, the data may not be read in correctly if at all.

General Importing data

To import data into R, we have to tell the program explicitly where to find the files you just downloaded. To do that we need to find the file's *path*.

The *path* is a programmatic way to direct your computer to the location of a file. It's like the file's address, or, where it lives on your computer. (link to picture to explain absolute vs relative path if there is time)

In all the examples below, the path shown (`data/`) is the path on **MY** computer.You will have to update this path to **YOUR** path on **YOUR** machine that points to your MATH130 folder.

You can find the path of a file by navigating to the desired file in your finder or browser window, right click and copy the file to the clipboard, then right click and pasting into a R markdown file. This will show a path that looks something like this:

```
file:///C:/Math 130/Data/Challenger.txt
```

You will need to remove the `file:///` part at the beginning before using this path. You will have three chances in this lab to practice this.

Special Instructions for Mac

- Navigate to the file or folder you wish to copy the path for Right-click (or Control+Click, or a Two-Finger click on trackpads) on the file or folder in the Mac Finder
- While in the right-click menu, hold down the OPTION key to reveal the "Copy (item name) as Pathname" option, it replaces the standard Copy option Once selected, the file or folders path is now in the clipboard, ready to be pasted anywhere

Source: https://stackoverflow.com/questions/52695546/how-to-copy-path-of-a-file-in-mac-os

Checking the import was successfull

The first thing you should always do after importing a data file is look at the raw data and ask yourself the following things:

1. Were the variable names read in correctly?
2. Is there the expected number of rows and columns?
3. Are the data types for each variable as expected?

There are many ways this can be done, here are three

1. `str()` to show you the structure of the data frame.
2. `head()` to show you the top 6 rows
3. `data[1:x, 1:x]` - when the data frame is too large for the above two to work well, we can use matrix notation to view only the first X columns, and first X rows.

Next we will read in three different data sets, each with unique file types, and look at each one to make sure it was read in correctly.

Text files

Text files are very simple files that have a `.txt` file extension. Common delimiters include a space, a comma (,) or a tab. Uncommon delimiters could include a % or even a semi-colon. By opening the file directly on your computer (not in R) you can see what delimiter is being used.

We will use the `read.table()` function that is in base R to read in any type of delimited file. A tab-delimited text file can be read in using `"\t"` as the deliminator character. In this class you **ALWAYS** want to include `header=TRUE` to signify that the data in the first row contains our column names.

```
email <- read.table("../data/email.txt", header=TRUE, sep="\t")
```

Here we call the `str()` or *structure* function to examine the data that was imported.

```
str(email)
```

```
## 'data.frame':    3921 obs. of  21 variables:
##  $ spam        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ to_multiple : int  0 0 0 0 0 0 1 1 0 0 ...
```

```
##  $ from        : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ cc          : int  0 0 0 0 0 0 0 1 0 0 ...
##  $ sent_email  : int  0 0 0 0 0 0 1 1 0 0 ...
##  $ time        : chr  "2011-12-31 22:16:41" "2011-12-31 23:03:59" "2012-01-01 08:00:32" "2012-01-01 (
##  $ image       : int  0 0 0 0 0 0 0 1 0 0 ...
##  $ attach      : int  0 0 0 0 0 0 0 1 0 0 ...
##  $ dollar      : int  0 0 4 0 0 0 0 0 0 0 ...
##  $ winner      : chr  "no" "no" "no" "no" ...
##  $ inherit     : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ viagra      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ password    : int  0 0 0 0 2 2 0 0 0 0 ...
##  $ num_char    : num  11.37 10.5 7.77 13.26 1.23 ...
##  $ line_breaks : int  202 202 192 255 29 25 193 237 69 68 ...
##  $ format      : int  1 1 1 1 0 0 1 1 0 1 ...
##  $ re_subj     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ exclaim_subj: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ urgent_subj : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ exclaim_mess: int  0 1 6 48 1 1 1 18 1 0 ...
##  $ number      : chr  "big" "small" "small" "small" ...
```

- The `email` data set has 3921 observations, and 21 variables.
- The variable names were read in correctly, as evidenced by the words in the far right corner after the `$`.
- Most variables are treated as numeric integers, except a few are factors. Including `time`. If we wanted to examine a time trend, we would need to convert the class into a `datetime`

  What happens if you forget to include the arguments for `header` or `sep`? Try that now and discuss with your neighbor what happened.

Forgetting both arguments

```
email <- read.table("../data/email.txt")
```

```
## Error in read.table("../data/email.txt"): duplicate 'row.names' are not allowed
```

Forgetting the header

```
email <- read.table("../data/email.txt", sep="\t")
head(email)
```

Forgetting the separator

```
email <- read.table("../data/email.txt", header=TRUE)
head(email)
```

CSV: Comma Separated Values

CSV is a fancy way of saying a text file with comma-separated values (i.e. CSV). This file type can also open in Excel, the icons even look similar. We could use `read.table()` to import this data file, but `read.csv()` is optimized to read in CSV files.

```
NCbirths <- read.csv("../data/NCbirths.csv", header=TRUE)
head(NCbirths)
```

```
##   fage mage      mature weeks    premie visits marital gained weight
## 1   NA   13 younger mom    39 full term     10 married     38   7.63
## 2   NA   14 younger mom    42 full term     15 married     20   7.88
## 3   19   15 younger mom    37 full term     11 married     38   6.63
## 4   21   15 younger mom    41 full term      6 married     34   8.00
## 5   NA   15 younger mom    39 full term      9 married     27   6.38
```

```
## 6   NA   15 younger mom    38 full term     19 married     22   5.38
##   lowbirthweight gender     habit  whitemom
## 1        not low   male nonsmoker not white
## 2        not low   male nonsmoker not white
## 3        not low female nonsmoker     white
## 4        not low   male nonsmoker     white
## 5        not low female nonsmoker not white
## 6            low   male nonsmoker not white
```

- The `NCbirths` data set has 1000 rows and 13 columns.
- Variable names are read in OK.
- Most the data is begin read in as factors and integers.
- There are some `NA` values for the variable `fage` (fathers age).

Excel files

The best method I have found so far to read in Excel files is from the `readxl` package by Hadley Wickham. This package need to be installed first, and then can be simply loaded using the `library()` function each time you start an `R` session where you will be reading in this type of data.

The `read_excel()` function is what we are going to use. Note the use of the underscore `_` instead of a period `.` between `read` and `excel`.

```
library(readxl)
police  <- read_excel("../data/fatal-police-shootings-data.xlsx", sheet=1, col_names=TRUE)
police[1:10,1:5]
```

```
## # A tibble: 10 x 5
##       id name                date                manner_of_death  armed
##    <dbl> <chr>               <dttm>              <chr>            <chr>
##  1     3 Tim Elliot          2015-01-02 00:00:00 shot             gun
##  2     4 Lewis Lee Lembke    2015-01-02 00:00:00 shot             gun
##  3     5 John Paul Quintero  2015-01-03 00:00:00 shot and Tasered unarmed
##  4     8 Matthew Hoffman     2015-01-04 00:00:00 shot             toy weapon
##  5     9 Michael Rodriguez   2015-01-04 00:00:00 shot             nail gun
##  6    11 Kenneth Joe Brown   2015-01-04 00:00:00 shot             gun
##  7    13 Kenneth Arnold Buck 2015-01-05 00:00:00 shot             gun
##  8    15 Brock Nichols       2015-01-06 00:00:00 shot             gun
##  9    16 Autumn Steele       2015-01-06 00:00:00 shot             unarmed
## 10    17 Leslie Sapp III     2015-01-06 00:00:00 shot             toy weapon
```

- The variable `date` is a `dttm` or *date-time* variable. This means `R` recognizes it directly as a date, not some string of numbers.
- Categorical variables such as `manner_of_death` and `city` are read in as `character` instead of factor. We may or may not want to change that later.
- Numeric variables such as `age` are of type `dbl` (double). This is similar to `integer` or `numeric`, so we are fine.

Go Back to Week 4