Intro to R

Data Input/Output

Outline

- Part 1: reading CSV file, common new user mistakes in data reading, checking for problems in the read data
- Part 2: data input overview, working directories, relative vs. absolute paths, reading XLSX file (Excel file), other data inputs
- Part 3: writing CSV file
- Part 4: reading and saving R objects

Data We Use

- Everything we do in class will be using real publicly available data - there are few ‘toy’ example datasets and ‘simulated’ data
- Baltimore Open Data and Data.gov will be sources for the first few days
- We have also added functionality to load these datasets directly in the `jhur` package

Data Input

- ‘Reading in’ data is the first step of any real project/analysis
- R can read almost any file format, especially via add-on packages
- We are going to focus on simple delimited files first
  - comma separated (e.g. ‘.csv’)
  - tab delimited (e.g. ‘.txt’)
  - Microsoft Excel (e.g. ‘.xlsx’)

Data Input

Youth Tobacco Survey (YTS) dataset:

“The YTS was developed to provide states with comprehensive data on both middle school and high school students regarding tobacco use, exposure to environmental tobacco smoke, smoking cessation, school curriculum, minors’ ability to purchase or otherwise obtain tobacco products, knowledge and attitudes about tobacco, and familiarity with pro-tobacco and anti-tobacco media messages.”

- Check out the data at: https://catalog.data.gov/dataset/youth-tobacco-survey-yts-data

Data Input: Dataset Location

Dataset is located at
http://johnmuschelli.com/intro_to_r/data/Youth_Tobacco_Survey_YTS_Data.csv

- Download data by clicking the above link
    - Safari - if a file loads in your browser, choose File –> Save As, select, Format “Page Source” and save

Data Input: Read in Directly

```
# load library `readr` that contains function `read_csv`
library(readr)
dat = read_csv("http://johnmuschelli.com/intro_to_r/data/Youth_Tobacco_Survey_

# `head` displays first few rows of a data frame
head(dat, 5)
```

```
# A tibble: 5 x 31
   YEAR LocationAbbr LocationDesc TopicType  TopicDesc  MeasureDesc   DataSour
  <dbl> <chr>        <chr>        <chr>      <chr>      <chr>         <chr>
1  2015 AZ           Arizona      Tobacco U… Cessation… Percent of C… YTS
2  2015 AZ           Arizona      Tobacco U… Cessation… Percent of C… YTS
3  2015 AZ           Arizona      Tobacco U… Cessation… Percent of C… YTS
4  2015 AZ           Arizona      Tobacco U… Cessation… Quit Attempt… YTS
5  2015 AZ           Arizona      Tobacco U… Cessation… Quit Attempt… YTS
# … with 24 more variables: Response <chr>, Data_Value_Unit <chr>,
#   Data_Value_Type <chr>, Data_Value <dbl>, Data_Value_Footnote_Symbol <chr>,
#   Data_Value_Footnote <chr>, Data_Value_Std_Err <dbl>,
#   Low_Confidence_Limit <dbl>, High_Confidence_Limit <dbl>, Sample_Size <dbl>
#   Gender <chr>, Race <chr>, Age <chr>, Education <chr>, GeoLocation <chr>,
#   TopicTypeId <chr>, TopicId <chr>, MeasureId <chr>, StratificationID1 <chr>
#   StratificationID2 <chr>, StratificationID3 <chr>, StratificationID4 <chr>,
#   SubMeasureID <chr>, DisplayOrder <dbl>
```

Data Input: Read in Directly

So what is going on “behind the scenes”?

`read_csv()` parses a “flat” text file (.csv) and turns it into a **tibble** – a rectangular data frame, where data are split into rows and columns

- First, a flat file is parsed into a rectangular matrix of strings
- Second, the type of each column is determined (heuristic-based guess)

Data Input: Read in Directly

`read_csv()` needs path to your file, will return a tibble

```
read_csv(file, col_names = TRUE, col_types = NULL,
  locale = default_locale(), na = c("", "NA"),
  quoted_na = TRUE, quote = "\"", comment = "", trim_ws = TRUE,
  skip = 0, n_max = Inf, guess_max = min(1000, n_max),
  progress = show_progress(), skip_empty_rows = TRUE
)
```

- `file` is the path to your file, in quotes
- can be path in your local computer – absolute file path or relative file path
- can be path to a file on a website

```
## Examples

dat = read_csv("/Users/martakaras/Downloads/Youth_Tobacco_Survey_YTS_Data.csv"

dat = read_csv("Youth_Tobacco_Survey_YTS_Data.csv")

dat = read_csv("www.someurl.com/table1.csv")
```

Data Input: Read in Directly

`read_csv()` is a special case of `read_delim()` – a general function to read a delimited file into a data frame

`read_delim()` needs path to your file and fileds delimiter, will return a tibble

```
read_delim(file, delim, quote = "\"", escape_backslash = FALSE,
  escape_double = TRUE, col_names = TRUE,  col_types = NULL,
  locale = default_locale(),na = c("", "NA"),  quoted_na = TRUE,
  comment = "", trim_ws = FALSE, skip = 0,
  n_max = Inf,  guess_max = min(1000, n_max),
  progress = show_progress(), skip_empty_rows = TRUE
)
```

- `file` is the path to your file, in quotes
- `delim` is what separates the fields within a record

```
## Examples
dat = read_delim("Youth_Tobacco_Survey_YTS_Data.csv", delim = ",")

dat = read_delim("www.someurl.com/table1.txt", delim = "\t")
```

Data Input: Read in Directly From File Path

```
dat = read_csv("../data/Youth_Tobacco_Survey_YTS_Data.csv")
```

```
── Column specification ────────────────────────────────────────────────────────
cols(
  .default = col_character(),
  YEAR = col_double(),
  Data_Value = col_double(),
  Data_Value_Std_Err = col_double(),
  Low_Confidence_Limit = col_double(),
  High_Confidence_Limit = col_double(),
  Sample_Size = col_double(),
  DisplayOrder = col_double()
)
ℹ Use `spec()` for the full column specifications.
```

The data is now successfully read into your R workspace. Colum specification of first few columns is printed to the console.

Common new user mistakes we have seen

1. **Working directory problems: trying to read files that R "can't find"**
   - Path misspecification
2. Typos (R is **case sensitive**, `x` and `X` are different)
   - RStudio helps with "tab completion"
3. Data type problems (is that a string or a number?)
4. Open ended quotes, parentheses, and brackets
5. Different versions of software

Data Input: Checking for problems

- The `spec()` and `problems()` functions show you the specification of how the data was read in.

```
problems(dat)
```

```
[1] row      col      expected actual
<0 rows> (or 0-length row.names)
```

```
spec(dat)
```

```
cols(
  YEAR = col_double(),
  LocationAbbr = col_character(),
  LocationDesc = col_character(),
  TopicType = col_character(),
  TopicDesc = col_character(),
  MeasureDesc = col_character(),
  DataSource = col_character(),
  Response = col_character(),
  Data_Value_Unit = col_character(),
  Data_Value_Type = col_character(),
  Data_Value = col_double(),
  Data_Value_Footnote_Symbol = col_character(),
  Data_Value_Footnote = col_character(),
  Data_Value_Std_Err = col_double(),
  Low_Confidence_Limit = col_double(),
  High_Confidence_Limit = col_double(),
  Sample_Size = col_double(),
  Gender = col_character(),
```

Data Input: Checking for problems

The `stop_for_problems()` function will stop if your data had any problem when reading in (even if that problem did not cause the data reading to fail).

- Particularly useful to put after the data reading code e.g. in some automated R script that should not proceed in case some data "weirdness" occurred.

```
stop_for_problems(dat)
```

Help

For any function, you can write `?FUNCTION_NAME`, or `help("FUNCTION_NAME")` to look at the help file:

```
?read_delim
help("read_delim")
```

Data Input: Read in From RStudio Toolbar

R Studio features some nice “drop-down” support, where you can run some tasks by selecting them from the toolbar.

For example, you can easily import text datasets using the `File --> Import Dataset --> From Text (readr)` command. Selecting this will bring up a new screen that lets you specify the formatting of your text file.

After importing a datatset, you get (printed in the R console) the corresponding R command that you can enter in the console if you want to re-import data.

Data Input: base R

There are also data importing functions provided in base R (rather than the `readr` package), like `read.delim()` and `read.csv()`.

These functions have slightly different syntax for reading in data (e.g. `header` argument).

However, while many online resources use the base R tools, the latest version of RStudio switched to use these new `readr` data import tools, so we will use them in the class for slides. They are also up to two times faster for reading in large datasets, and have a progress bar which is nice.

But you can use whatever function you feel more comfortable with.

Lab Part 1

Lab

Website

Data Input: `tbl_df` (a tibble)

The `read_csv()`, `read_delim()` and related functions from `readr` all return `tbl_df` object (colloquially: a tibble) that is a `data.frame` object but with improved printing and subsetting properties.

We also get `tbl_df` (tibble) when using drop-down menu in RStudio.

Alternatives:

- `read.csv()` – "base" R function, still largely used; returns `data.frame` object
- `fread()` – from `data.table` R package; returns `data.table` object

Data Input: `tbl_df` (a tibble)

```
head(dat, 3)
```

```
# A tibble: 3 x 31
   YEAR LocationAbbr LocationDesc TopicType   TopicDesc MeasureDesc   DataSour
  <dbl> <chr>        <chr>        <chr>       <chr>     <chr>         <chr>
1  2015 AZ           Arizona      Tobacco Us… Cessatio… Percent of C… YTS
2  2015 AZ           Arizona      Tobacco Us… Cessatio… Percent of C… YTS
3  2015 AZ           Arizona      Tobacco Us… Cessatio… Percent of C… YTS
# … with 24 more variables: Response <chr>, Data_Value_Unit <chr>,
#   Data_Value_Type <chr>, Data_Value <dbl>, Data_Value_Footnote_Symbol <chr>,
#   Data_Value_Footnote <chr>, Data_Value_Std_Err <dbl>,
#   Low_Confidence_Limit <dbl>, High_Confidence_Limit <dbl>, Sample_Size <dbl>
#   Gender <chr>, Race <chr>, Age <chr>, Education <chr>, GeoLocation <chr>,
#   TopicTypeId <chr>, TopicId <chr>, MeasureId <chr>, StratificationID1 <chr>
#   StratificationID2 <chr>, StratificationID3 <chr>, StratificationID4 <chr>,
#   SubMeasureID <chr>, DisplayOrder <dbl>
```

```
class(dat)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

Data Input

- `nrow()` displays the number of rows of a data frame
- `ncol()` displays the number of columns
- `dim()` displays a vector of length 2: # rows, # columns
- `colnames()` displays the column names (if any) and `rownames()` displays the row names (if any)

```
dim(dat)
```

```
[1] 9794   31
```

```
nrow(dat)
```

```
[1] 9794
```

```
ncol(dat)
```

```
[1] 31
```

Data Input

- `nrow()` displays the number of rows of a data frame
- `ncol()` displays the number of columns
- `dim()` displays a vector of length 2: # rows, # columns
- `colnames()` displays the column names (if any) and `rownames()` displays the row names (if any)

```
colnames(dat)
```

```
 [1] "YEAR"                      "LocationAbbr"
 [3] "LocationDesc"              "TopicType"
 [5] "TopicDesc"                 "MeasureDesc"
 [7] "DataSource"                "Response"
 [9] "Data_Value_Unit"           "Data_Value_Type"
[11] "Data_Value"                "Data_Value_Footnote_Symbol"
[13] "Data_Value_Footnote"       "Data_Value_Std_Err"
[15] "Low_Confidence_Limit"      "High_Confidence_Limit"
[17] "Sample_Size"               "Gender"
[19] "Race"                      "Age"
[21] "Education"                 "GeoLocation"
[23] "TopicTypeId"               "TopicId"
[25] "MeasureId"                 "StratificationID1"
[27] "StratificationID2"         "StratificationID3"
[29] "StratificationID4"         "SubMeasureID"
[31] "DisplayOrder"
```

Working Directories

- R "looks" for files on your computer relative to the **working directory**
- Many people recommend not setting a working directory in the scripts
  - If you open an R file with a new RStudio session, it assumes the working directory is where the script is in
- If you do set a working directory, do it at the beginning of your script
- Example of getting and setting the working directory:

```
# get the working directory
getwd()
setwd("~/Lectures")
```

Setting a Working Directory

- Setting the directory can sometimes be finicky
  - **Windows**: Default directory structure involves single backslashes (“\”), but R interprets these as “escape” characters. So you must replace the backslash with forward slashes (“/”) or two backslashes (“\\”)
  - **Mac/Linux**: Default is forward slashes, so you are okay
- Typical directory structure syntax applies
  - “..” - goes up one level
  - “./” - is the current directory
  - “~” - is your “home” directory

Working Directory

Note that the `dir()` function interfaces with your operating system and can show you which files are in your current working directory.

You can try some directory navigation:

```
dir("./") # shows directory contents
```

```
 [1] "Data_IO.html"                  "Data_IO.pdf"
 [3] "Data_IO.R"                     "index.html"
 [5] "index.pdf"                     "index.R"
 [7] "index.Rmd"                     "lab"
 [9] "makefile"                      "YouthTobacco_newNames.csv"
[11] "yts_dataset.rds"
```

```
dir("..")
```

```
 [1] "all_functions.xlsx"
 [2] "all_the_functions.csv"
 [3] "all_the_packages.txt"
 [4] "Arrays_Split"
 [5] "Basic_R"
 [6] "Best_Model_Coefficients.csv"
 [7] "Best_Model_Coefficients.xlsx"
 [8] "bibliography.bib"
 [9] "black_and_white_theme.pdf"
[10] "bloomberg.logo.small.horizontal.blue.png"
[11] "data"
[12] "Data_Classes"
```

Relative vs. absolute paths (From Wiki)

*An **absolute or full path** points to the same location in a file system, regardless of the current working directory. To do that, it must include the root directory.*

This means if I try your code, and you use absolute paths, it won't work unless we have the exact same folder structure where R is looking (bad).

*By contrast, a **relative path starts from some given working directory**, avoiding the need to provide the full absolute path. A filename can be considered as a relative path based at the current working directory.*

Setting the Working Directory From RStudio Toolbar

In RStudio, go to `Session -> Set Working Directory -> To Source File Location`

RStudio should put code in the Console, similar to this:

```
setwd("~/Lectures/Data_IO/lecture")
```

Again, if you open an R file with a new RStudio session, RStudio sets working directory to the file's location for you.

Setting the Working Directory

You may need to make sure that RStudio is the default application to open .R files

- Mac - right click –> `Get Info --> Open With: RStudio --> Change All`

Data Input - Excel

Many data analysts collaborate with researchers who use Excel to enter and curate their data. Often times, this is the input data for an analysis. You therefore have two options for getting this data into R:

- Saving the Excel sheet as a .csv file, and using `read_csv()`
- Using an add-on package, like `xlsx`, `readxl`, or `openxlsx`

For single worksheet .xlsx files, I often just save the spreadsheet as a .csv file (because I often have to strip off additional summary data from the columns)

For an .xlsx file with multiple well-formated worksheets, I use the `readxl` package for reading in the data.

Data Input - Other Software

- **readr** package - has `read_csv()`/`write_csv()` and `read_table()` functions similar to `read.csv()`/`write.csv()` and `read.table()` from base R; has different defaults, but can read **much faster** for very large data sets
- **readxl** package - the `read_excel()` function can read Excel sheets easily
- **haven** package reads in SAS, SPSS, Stata formats
- **sas7bdat** reads .sas7bdat files (SAS files)
- **foreign** package - can read all the formats as **haven**. Around longer (aka more testing), but not as maintained (bad for future)

Some of these are now available in the RStudio drop-down list.

Lab Part 2

Lab

Website

Data Output

While its nice to be able to read in a variety of data formats, it's equally important to be able to output data somewhere.

There are also data exporting functions in the `readr` package, which have the pattern `write_*` like `write_csv()` and `write_delim()`

```
write_csv(x, file,
  na = "NA", append = FALSE,
  col_names = !append, quote_escape = "double",
  eol = "\n", path = deprecated()
)

write_delim(x, file, delim = " ",
  na = "NA", append = FALSE,
  col_names = !append, quote_escape = "double",
  eol = "\n", path = deprecated()
)
```

Data Output

`x`: data frame or tibble you want to write

`file`: the file name where you want to R object written

- It can be an absolute path, a relative path (relative to your working directory), or a filename (which writes the file to your working directory)

`delim`: what character separates the columns?

- `","` = comma delimted
- `"\t"` = tab delimited

Data Output

There other functions in base R to do the writing, like `write.table()` and `write.csv()`

- Some of their arguments may be named differently than in `write_csv()` – look up teir documentation (e.g. run `?write.table`)
- Note base R writing functions by default do write out row names, which you can change by setting e.g. `row.names = TRUE`. I do this a lot since I often email these to collaborators who open them in Excel

Data Output

For example, we can write back out the Youth Tobacco dataset with the new column name:

```
# load `dplyr` package that has `rename` function
library(dplyr)
dat = rename(dat, Location_Abbr = LocationAbbr)

write_csv(dat, path = "YouthTobacco_newNames.csv")
```

Lab Part 3

Lab

Website

Write a single R object to a file

If you want to save **one** R object, you can use `write_rds()` from `readr` package to save an R object (e.g., a tibble) to an R binary file with `rds` extension

```
write_rds(dat, file = "yts_dataset.rds")
```

`rds` is R's native format for storing single objects

Read a single R object from a file

To read this back in to R, you need to use `read_rds()`. Note you **need to assign it**:

```
dat2 = read_rds(file = "yts_dataset.rds")
identical(dat, dat2) # test if they are the same
```

```
[1] TRUE
```

Lab Part 4

Lab

Website

Write multiple R objects to a file

The `save` command can save a set of `R` objects into an "R data file", with the extension `.rda` or `.RData`.

```
x = 5; # can have semicolons a the end!

# calling read_csv function and pasting a long string together
yts = readr::read_csv(
  paste0("http://johnmuschelli.com/intro_to_r/",
         "data/Youth_Tobacco_Survey_YTS_Data.csv"))

# saving two files
save(yts, x, file = "yts_data.rda")
```

Load multiple R objects from a file

The opposite of `save` is `load`.

The `ls()` command lists the items in the workspace/environment and `rm()` removes them:

```
ls() # list things in the workspace
```

```
[1] "dat"  "dat2" "x"    "yts"
```

```
rm(list = c("x", "yts"))
ls()
```

```
[1] "dat"  "dat2"
```

```
z = load("yts_data.rda")
ls()
```

```
[1] "dat"  "dat2" "x"    "yts"  "z"
```

Load multiple R objects from a file

```
z = load("yts_data.rda")
print(z)
```

```
[1] "yts" "x"
```

Note, `z` is a **character vector** of the **names** of the objects loaded, **not** the objects themselves.