Data Input/Output

Introduction to R for Public Health Researchers

Common new users mistakes we have seen

1. Different versions of software
2. Data type problems (is that a string or a number?)
3. **Working directory problems: trying to read files that R "can't find"**
   - RStudio can help, and so do RStudio Projects
   - discuss in Data Input/Output lecture
4. Typos (R is **case sensitive**, `x` and `X` are different)
   - RStudio helps with "tab completion"
   - discussed throughout

Before we get Started: Working Directories

- R "looks" for files on your computer relative to the "working" directory
- Many recommend not setting a directory in the scripts
  - assume you're in the directory the script is in
  - If you open an R file with a new RStudio session, it does this for you.
- 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:

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

```
[1] "Data_IO.html"              "Data_IO.pdf"
[3] "Data_IO.R"                 "Data_IO.Rmd"
[5] "monuments_newNames.csv"
```

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

```
[1] "lab"     "lecture"
```

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

In RStudio, go to `Session â†' Set Working Directory â†' To Source File Location`

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

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

Setting the Working Directory

Again, if you open an R file with a new RStudio session, it does this for you. You may need to make this a default.

1. Make sure RStudio is the default application to open .R files
   - Mac - right click â†' Get Info â†' Open With: RStudio â†' Change All
   - Windows - Andrew will show
2. Close RStudio Double click `day1.R`

- Confirm the directory contains "day1.R" using `dir()`:
  - Type `dir()` in the R Console (`day1.R` should be there)

Help

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

```
?dir
help("dir")
```

Data Aside

- Everything we do in class will be using real publicly available data - there are few 'toy' example datasets and 'simulated' data
- OpenBaltimore and Data.gov will be sources for the first few days

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
  - tab delimited (e.g. '.txt')
  - comma separated (e.g. '.csv')
  - Microsoft excel (e.g. '.xlsx')

Data Input

Monuments Dataset: "This data set shows the point location of Baltimore City monuments. However, the completness and currentness of these data are uncertain."

- Download data from `http://www.aejaffe.com/winterR_2017/data/Monuments.csv`
  - Safari - if a file loads in your browser, choose File â† Save As, select, Format "Page Source" and save
- Save it (or move it) to the same folder as your `day1.R` script
- Within RStudio: Session –> Set Working Directory –> To Source File Location
- (data downloaded from `https://data.baltimorecity.gov/Community/Monuments/cpxf-kxp3`)

Data Input

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 CSV" 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 the corresponding R commands that you can enter in the console if you want to re-import data.

Data Input

So what is going on "behind the scenes"?

`read.table()`: Reads a file in table format and creates a `data.frame` from it, with cases corresponding to lines and variables to fields in the file.

```
# the four ones I've put at the top are the important input
read.table( file, # filename
            header = FALSE, # are there column names?
            sep = "", # what separates columns?
            as.is = !stringsAsFactors, # do you want charact
                   #factors or characters?
            quote = "\"'",  dec = ".", row.names, col.names,
            na.strings = "NA", nrows = -1,
            skip = 0, check.names = TRUE, fill = !blank.line
            strip.white = FALSE, blank.lines.skip = TRUE, co
            stringsAsFactors = default.stringsAsFactors())

# for example: `read.table("file.txt", header = TRUE, sep="
```

Data Input

- The filename is the path to your file, in quotes
- The function will look in your "working directory" if no absolute file path is given
- Note that the filename can also be a path to a file on a website (e.g. 'www.someurl.com/table1.txt')

Data Input

There is a 'wrapper' function for reading CSV files:

```
read.csv
```

```
function (file, header = TRUE, sep = ",", quote = "\"", dec
    fill = TRUE, comment.char = "", ...)
read.table(file = file, header = header, sep = sep, quote =
    dec = dec, fill = fill, comment.char = comment.char, ..
<bytecode: 0x000000001f4b1bc0>
<environment: namespace:utils>
```

Note: the `...` designates extra/optional arguments that can be passed to `read.table()` if needed

Data Input

- Here would be reading in the data from the command line, specifying the file path:

```
mon = read.csv("../../data/Monuments.csv", header = TRUE, a
head(mon)
```

```
                                name zipCode neighborhood cou
1           James Cardinal Gibbons   21201     Downtown
2              The Battle Monument   21202     Downtown
3 Negro Heroes of the U.S Monument   21202     Downtown
4              Star Bangled Banner   21202     Downtown
5  Flame at the Holocaust Monument   21202     Downtown
6                   Calvert Statue   21202     Downtown
  policeDistrict                          Location.1
1        CENTRAL  408 CHARLES ST\nBaltimore, MD\n
2        CENTRAL
3        CENTRAL
4        CENTRAL 100 HOLLIDAY ST\nBaltimore, MD\n
```

Data Input

```
colnames(mon) # column names
```

```
[1] "name"           "zipCode"        "neighborhood"    "
[5] "policeDistrict" "Location.1"
```

```
head(mon$zipCode) # first few rows
```

```
[1] 21201 21202 21202 21202 21202 21202
```

Data Input

The `read.table()` function returns a `data.frame`, which is the primary data format for most data cleaning and analyses. The `str` function gives "structure" of data:

```
str(mon) # structure of an R object
```

```
'data.frame':   84 obs. of  6 variables:
 $ name           : chr  "James Cardinal Gibbons" "The Batt
 $ zipCode        : int  21201 21202 21202 21202 21202 212(
 $ neighborhood   : chr  "Downtown" "Downtown" "Downtown" "
 $ councilDistrict: int  11 11 11 11 11 11 11 7 14 14 ...
 $ policeDistrict : chr  "CENTRAL" "CENTRAL" "CENTRAL" "CEN
 $ Location.1     : chr  "408 CHARLES ST\nBaltimore, MD\n"
```

Data Input with `tbl_dfs`

- When using the dropdown menu in RStudio, it uses `read_csv`, which is an improved version of reading in CSVs. It is popular but `read.csv` is still largely used. It returns a `tbl` (tibble), that is a `data.frame` with improved printing and subsetting properties:

```
library(readr)
mon_tbl = read_csv("../../data/Monuments.csv")
head(mon_tbl)
```

```
# A tibble: 6 × 6
                                  name zipCode neighborhood cou
                                 <chr>   <int>        <chr>
1           James Cardinal Gibbons   21201     Downtown
2              The Battle Monument   21202     Downtown
3 Negro Heroes of the U.S Monument   21202     Downtown
4              Star Bangled Banner   21202     Downtown
5  Flame at the Holocaust Monument   21202     Downtown
```

Data Input

Changing variable names in `data.frames` works using the `names()` function, which is analagous to `colnames()` for data frames (they can be used interchangeably)

```
names(mon)[1] = "Name"
names(mon)
```

```
[1] "Name"              "zipCode"           "neighborhood"     "
[5] "policeDistrict"    "Location.1"
```

```
names(mon)[1] = "name"
names(mon)
```

```
[1] "name"              "zipCode"           "neighborhood"     "
[5] "policeDistrict"    "Location.1"
```

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.

`write.table()`: prints its required argument `x` (after converting it to a `data.frame` if it is not one nor a `matrix`) to a file or connection.

```
write.table(x,file = "", append = FALSE, quote = TRUE, sep
            eol = "\n", na = "NA", dec = ".", row.names = T
            col.names = TRUE, qmethod = c("escape", "double
            fileEncoding = "")
```

Data Output

`x`: the R `data.frame` or `matrix` you want to write

`file`: the file name where you want to R object written. It can be an absolute path, or a filename (which writes the file to your working directory)

`sep`: what character separates the columns?

- "," = .csv - Note there is also a `write.csv()` function
- "" = tab delimited

`row.names`: I like setting this to FALSE because I email these to collaborators who open them in Excel

Data Output

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

```
names(mon)[6] = "Location"
write.csv(mon, file="monuments_newNames.csv", row.names=FAI
```

Note that `row.names=TRUE` would make the first column contain the row names, here just the numbers `1:nrow(mon)`, which is not very useful for Excel. Note that row names can be useful/informative in R if they contain information (but then they would just be a separate column).

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 `xlsx`, `readxl`, or `openxlsx` package for reading in the data.

Data Input - Other Software

- **haven** package (`https://cran.r-project.org/web/packages/haven/index.html`) reads in SAS, SPSS, Stata formats
- **readxl** package - the `read_excel` function can read Excel sheets easily
- **readr** package - Has *read_csv*/*write_csv* and *read_table* functions similar to *read.csv*/*write.csv* and *read.table*. Has different defaults, but can read **much faster** for very large data sets
- **sas7bdat** reads .sas7bdat files
- **foreign** package - can read all the formats as **haven**. Around longer (aka more testing), but not as maintained (bad for future).