Data Input/Output

Andrew Jaffe

January 4, 2016

Before we get Started: Working Directories

- R looks for files on your computer relative to the "working" directory
- It's always safer to set the working directory at the beginning of your script. Note that setting the working directory created the necessary code that you can copy into your script.
- Example of help file

```
## get the working directory
getwd()
# setwd("~/winterR_2016/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 slashed ("/") or two
    backslashes ("\")
  - Mac/Linux: Default is forward slashes, so you are okay
- Typical linux/DOS 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"               "Data_IO.Rmd"
[5] "monuments_newNames.csv"
```

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

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

Working Directory

- Copy the code to set your working directory from the History tab in RStudio (top right)
- Confirm the directory contains "day1.R" using `dir()`

Data Input

- We used several pre-installed sample datasets during previous modules (CO_2, iris)
- However, '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 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

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

- Download data from `http://www.aejaffe.com/winterR_2016/data/Monuments.csv`
- 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 "Tools –> Import Dataset" 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
            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: 0x7fe4f7104038>
<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,as.is
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

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

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

```r
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

```r
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 2120
 $ 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

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 = 
            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-formatted 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).