Module 4

Data Input

Andrew Jaffe
Instructor

Data Input

We used several pre-installed sample datasets during previous modules (`CO2`, `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 Input

`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 inputs
read.table( file, # filename
            header = FALSE, # are there column names?
            sep = "", # what separates columns?
            as.is = !stringsAsFactors, # do you want character strings as factors or characters
            quote = "\"'",  dec = ".", row.names, col.names,
            na.strings = "NA", nrows = -1,
            skip = 0, check.names = TRUE, fill = !blank.lines.skip,
            strip.white = FALSE, blank.lines.skip = TRUE, comment.char = "#",
            stringsAsFactors = default.stringsAsFactors())

# for example: `read.table("file.txt", header = TRUE, sep="\t", as.is=TRUE)`
```

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 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 completness and currentness of these data are uncertain."

Navigate to: https://data.baltimorecity.gov/Community/Monuments/cpxf-kxp3

Export --> Download --> Download As: CSV

Save it (or move it) to the same folder as your day2.R script

Within RStudio: Session --> Set Working Directory --> To Source File Location

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 = quote,
##     dec = dec, fill = fill, comment.char = comment.char, ...)
## <bytecode: 0x000000000a326ac8>
## <environment: namespace:utils>
```

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

Data Input

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

```
                            name zipCode neighborhood councilDistrict
1           James Cardinal Gibbons   21201     Downtown               11
2              The Battle Monument   21202     Downtown               11
3 Negro Heroes of the U.S Monument   21202     Downtown               11
4              Star Bangled Banner   21202     Downtown               11
5  Flame at the Holocaust Monument   21202     Downtown               11
6                   Calvert Statue   21202     Downtown               11
  policeDistrict                          Location.1
1        CENTRAL  408 CHARLES ST\nBaltimore, MD\n
2        CENTRAL
3        CENTRAL
4        CENTRAL 100 HOLLIDAY ST\nBaltimore, MD\n
5        CENTRAL    50 MARKET PL\nBaltimore, MD\n
6        CENTRAL  100 CALVERT ST\nBaltimore, MD\n
```

```
> colnames(mon)
```

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

```
> head(mon$zipCode)
```

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

```
> head(mon$neighborhood)
```

```
[1] "Downtown" "Downtown" "Downtown" "Downtown" "Downtown" "Downtown"
```

Aside: Working Directory

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()
```

```
[1] "C:/Users/Andrew/Dropbox/R_CLASS/SummerR_2014/Lectures"
```

```
> setwd("~/Dropbox/SummerR_2014/Lectures")
```

```
Error: cannot change working directory
```

Aside: 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

Try some directory navigation:

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

```
 [1] "assets"                    "cache"
 [3] "charmcirc.rda"             "charmcitycirc_reduced.csv"
 [5] "data"                      "figure"
 [7] "libraries"                 "module1.html"
 [9] "module1.md"                "module1.Rmd"
[11] "module10.Rmd"              "module11.Rmd"
[13] "module12.Rmd"              "module13.Rmd"
[15] "module2.html"              "module2.md"
[17] "module2.Rmd"               "module3.html"
[19] "module3.md"                "module3.Rmd"
[21] "module4.html"              "module4.md"
[23] "module4.Rmd"               "module5.html"
[25] "module5.md"                "module5.Rmd"
[27] "module6.html"              "module6.md"
[29] "module6.Rmd"               "module7.html"
[31] "module7.md"                "module7.Rmd"
[33] "module8.html"              "module8.md"
[35] "module8.Rmd"               "module9.Rmd"
[37] "PDFs"                      "plots"
[39] "Reports"                   "Things_To_Do_A_Lot.R"
```

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

```
 [1] "Computing_in_R.zip"          "desktop.ini"
 [3] "exercises"                   "figure"
```

Working Directory

Copy the code to set your working directory from the History tab in RStudio (top right)

Confirm the directory contains "day2.R" using `dir()`

Data Input

The `read.table()` function returns a `data.frame`

```
> class(mon)
```

```
[1] "data.frame"
```

```
> str(mon)
```

```
'data.frame':   84 obs. of  6 variables:
 $ name           : chr  "James Cardinal Gibbons" "The Battle Monument" "Negro Heroes of the U
 $ zipCode        : int  21201 21202 21202 21202 21202 21202 21202 21211 21213 21211 ...
 $ neighborhood   : chr  "Downtown" "Downtown" "Downtown" "Downtown" ...
 $ councilDistrict: int  11 11 11 11 11 11 11 7 14 14 ...
 $ policeDistrict : chr  "CENTRAL" "CENTRAL" "CENTRAL" "CENTRAL" ...
 $ Location.1     : chr  "408 CHARLES ST\nBaltimore, MD\n" "" "" "100 HOLLIDAY ST\nBaltimore, I
```

Data Input

Changing variable names in `data.frame`s 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"   "councilDistrict"
[5] "policeDistrict" "Location.1"
```

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

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

Data Subsetting

Now we will introduce subsetting rows/observations of data using logical statements. Recall that the `logical` class consists of either `TRUE` or `FALSE`

```
> z = c(TRUE,FALSE,TRUE,FALSE)
> class(z)
```

```
[1] "logical"
```

```
> sum(z) # number of TRUEs
```

```
[1] 2
```

And recall again that the logical class does NOT use quotes.

```
> z2 = c("TRUE","FALSE","TRUE","FALSE")
> class(z2)
```

```
[1] "character"
```

```
> sum(z2)
```

```
Error: invalid 'type' (character) of argument
```

```
> identical(z,z2)
```

```
[1] FALSE
```

Useful: `identical()` checks if two R objects are exactly identical/equal.

Logical Statements

Almost every R object can be evaluated and converted to the `logical` class using different logical statements (this mirrors computer science/programming syntax)

- '==': equal to
- '!=': not equal to (it is NOT '~' in R, e.g. SAS)
- '>': greater than
- '<': less than
- '>=': greater than or equal to
- '<=': less than or equal to

Logical Statements

```
> x = 1:6
> x > 4
```

```
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE
```

```
> x == 3
```

```
[1] FALSE FALSE  TRUE FALSE FALSE FALSE
```

Logical Statements

These logical statements can be then used to subset your data.

```
> Index = (mon$zipCode == 21202)
> sum(Index)
```

```
[1] 16
```

```
> table(Index)
```

```
Index
FALSE  TRUE
   68    16
```

```
> mon2 = mon[Index,]
```

```
> dim(mon2)
```

```
[1] 16  6
```

```
> head(mon2)
```

```
                                  name zipCode neighborhood
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
```

Which

`which()`: "Give the `TRUE` indices of a logical object, allowing for array indices."

```
> mon$Location.1 != ""
```

```
 [1]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
[12] FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
[34]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
[45]  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE
[56] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
[67] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE
[78]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
```

```
> which(mon$Location.1 != "")
```

```
 [1]  1  4  5  6  7  8  9 11 14 15 17 18 19 20 21 22 23 24 25 26 27 28 29
[24] 31 32 33 34 35 36 37 38 41 42 43 44 45 46 47 50 54 55 57 58 59 60 61
[47] 68 69 70 71 72 73 76 78 79 80 81 84
```

Missing Data

In R, missing data is represented by the symbol `NA` (note that it is NOT a character, and therefore not in quotes, just like the `logical` class)

`is.na()` is a logical test for which variables are missing

Many summarization functions do not the calculation you expect (e.g. they return `NA`) if there is ANY missing data, and these ofen have an argument `na.rm=FALSE`. Changing this to `na.rm=TRUE` will ignore the missing values in the calculation (i.e. `mean()`, `median()`, `max()`, `sum()`)

Here is a good link with more information: http://www.statmethods.net/input/missingdata.html

Lab

Swirl Module 4: Monuments

Question 1

Names are just an attribute of the data frame (recall `str`) that you can change to any valid character name

Valid character names are case-sensitive, contain a-z, 0-9, underscores, and periods (but cannot start with a number).

For the `data.frame` class, `colnames()` and `names()` return the same attribute.

```
> names(mon)
```

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

```
> names(mon)[6] = "location"
> names(mon)
```

```
[1] "name"           "zipCode"        "neighborhood"   "councilDistrict"
[5] "policeDistrict" "location"
```

These naming rules also apply for creating R objects

Question 2

There are several ways to return the number of rows of a data frame or matrix

```
> nrow(mon)
```

```
[1] 84
```

```
> dim(mon)
```

```
[1] 84  6
```

```
> length(mon$name)
```

```
[1] 84
```

Question 3

`unique()` returns the unique entries in a vector

```
> unique(mon$zipCode)
```

```
 [1] 21201 21202 21211 21213 21217 21218 21224 21230 21231 21214 21223
[12] 21225 21251
```

```
> unique(mon$policeDistrict)
```

```
[1] "CENTRAL"       "NORTHERN"      "NORTHEASTERN" "WESTERN"
[5] "SOUTHEASTERN" "SOUTHERN"      "EASTERN"
```

```
> unique(mon$councilDistrict)
```

```
 [1] 11  7 14 13  1 10  3  2  9 12
```

```
> unique(mon$neighborhood)
```

```
 [1] "Downtown"                          "Remington"
 [3] "Clifton Park"                      "Johns Hopkins Homewood"
 [5] "Mid-Town Belvedere"                "Madison Park"
 [7] "Upton"                             "Reservoir Hill"
 [9] "Harlem Park"                       "Coldstream Homestead Montebello"
[11] "Guilford"                          "McElderry Park"
[13] "Patterson Park"                    "Canton"
[15] "Middle Branch/Reedbird Parks"      "Locust Point Industrial Area"
[17] "Federal Hill"                      "Washington Hill"
```

```
> length(unique(mon$zipCode))
```

```
[1] 13
```

```
> length(unique(mon$policeDistrict))
```

```
[1] 7
```

```
> length(unique(mon$councilDistrict))
```

```
[1] 10
```

```
> length(unique(mon$neighborhood))
```

```
[1] 32
```

Also note that table() can work, which tabulates a specific variable (or cross-tabulates two variables)

```
> table(mon$zipCode)
```

```
21201 21202 21211 21213 21214 21217 21218 21223 21224 21225 21230 21231
   11    16     8     4     1     9    14     4     8     1     3     4
21251
    1
```

```
> length(table(mon$zipCode))
```

```
[1] 13
```

Question 4

The "by hand" way is cross-tabulating the zip codes and neighborhoods,

```
> tab = table(mon$zipCode, mon$neighborhood)
> # tab
> tab[,"Downtown"]
```

```
21201 21202 21211 21213 21214 21217 21218 21223 21224 21225 21230 21231
    2     9     0     0     0     0     0     0     0     0     0     0
21251
    0
```

```
> length(unique(tab[,"Downtown"]))
```

```
[1] 3
```

```
> tt = tab[,"Downtown"]
> tt
```

```
21201 21202 21211 21213 21214 21217 21218 21223 21224 21225 21230 21231
    2     9     0     0     0     0     0     0     0     0     0     0
21251
    0
```

```
> tt == 0 # which entries are equal to 0
```

```
21201 21202 21211 21213 21214 21217 21218 21223 21224 21225 21230 21231
FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
21251
 TRUE
```

```
> tab[,"Downtown"] !=0
```

```
21201 21202 21211 21213 21214 21217 21218 21223 21224 21225 21230 21231
 TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
21251
FALSE
```

```
> sum(tab[,"Downtown"] !=0)
```

```
[1] 2
```

```
> sum(tab[,"Johns Hopkins Homewood"] !=0)
```

```
[1] 2
```

We could also subset the data into neighborhoods:

```
> dt = mon[mon$neighborhood == "Downtown",]
> head(mon$neighborhood == "Downtown",10)
```

```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
```

```
> dim(dt)
```

```
[1] 11  6
```

```
> length(unique(dt$zipCode))
```

```
[1] 2
```

Question 5

```
> head(mon$location)
```

```
[1] "408 CHARLES ST\nBaltimore, MD\n"  ""
[3] ""                                 "100 HOLLIDAY ST\nBaltimore, MD\n"
[5] "50 MARKET PL\nBaltimore, MD\n"    "100 CALVERT ST\nBaltimore, MD\n"
```

```
> table(mon$location != "") # FALSE=DO NOT and TRUE=DO
```

```

FALSE  TRUE
   26    58
```

Question 6

```
> tabZ = table(mon$zipCode)
> head(tabZ)
```

```
21201 21202 21211 21213 21214 21217
   11    16     8     4     1     9
```

```
> max(tabZ)
```

```
[1] 16
```

```
> tabZ[tabZ == max(tabZ)]
```

```
21202
   16
```

`which.max()` returns the FIRST entry/element number that contains the maximum and `which.min()` returns the FIRST entry that contains the minimum

```
> which.max(tabZ) # this is the element number
```

```
21202
    2
```

```
> tabZ[which.max(tabZ)] # this is the actual maximum
```

```
21202
   16
```

```
> tabN = table(mon$neighborhood)
> tabN[which.max(tabN)]
```

```
Johns Hopkins Homewood
                    17
```

```
> tabC = table(mon$councilDistrict)
> tabC[which.max(tabC)]
```

```
11
29
```

```
> tabP = table(mon$policeDistrict)
> tabP[which.max(tabP)]
```

```
CENTRAL
     27
```

Question 7

```
> monTab = read.delim("http://biostat.jhsph.edu/~ajaffe/summerR/data/Monuments-tab.txt",
+                     header=TRUE, as.is=TRUE)
> identical(mon$name,monTab$name)
```

```
[1] TRUE
```