R Workshop Module 1: Introduction to R and RStudio

Prestented by Applied Statistical Lab
Jin Xie (`jin.xie@uky.edu`)
Department of Statistics, University of Kentucky
Janurary 23, 2017

Introduction to R

- *Getting Started:* R is a free software environment for statistical computing and graphics and can be downloaded from `http://www.r-project.org/`.
- According to wikipedia.com: "The R language is widely used among statisticians and data miners for developing statistical software and data analysis. Polls and surveys of data miners are showing R's popularity has increased substantially in recent years."
- *Advantages of R:* The R language is part of the GNU project which means that
  - the program is freely distributed,
  - the source code is available, and
  - any users can submit code/libraries so that other users can use the methods they have developed
- There are two (related) ways you can use R:
  1. you can simply write commands and use the preloaded functions already included, or
  2. you can write your own functions.
- In either case, it is generally a bad idea to type commands directly into R, since these commands are often hard to track. Also, if a mistake is made in a command, it is hard to find and fix.
- *Instead*, use a text editor to write a script (e.g. filename.R) and either copy and paste the commands into R or use the `source()` function to run the script in R. If you use Windows, editors such as RStudio, tinn-R, Winedt, notepad++ and others exist.

Introduction to RStudio

- *Getting Started:* RStudio is a free R-editor that can be used along with R. It can be downloaded from `http://www.rstudio.com/`. RStudio can be found under the start menu and the programs tab.
- There are four panels in the main RStudio window.
  1. *Console:* This is the place you can type R commands line-by-line.
  2. *Script Window:* This is where you can type R commands and save them so that you can reproduce or reanalyze your results.
     - To run commands, highlight the code you want to run and press `Ctrl + R` or click "Run" in the upper right hand corner of the panel.
  3. *Workspace/History:* Workspace shows all of the variables currently loaded in RStudio. History gives a list of all of the commands you have typed in this R Session.
  4. *Various Extra Features:* The two tabs I use most often are "Plots," which shows the current plot from R, and "Help," which displays the help for a function already built-in to R.

Help within R

The help files (as well as google) are very useful when learning about functions.

- If you <u>know</u> a function name (for instance, `mean()`) you can use either `help(mean)` or `?mean`.
- If you <u>do not</u> know a function name, search for applicable functions for what you want to do using either `help.search("mean")` or `??mean`.

There are *many* other resources for general help with R.

Setting the Working Directory

Before reading in data, it is convenient to set a "working directory." This specifies a default location for you to read files in from and write files to during a session. Using RStudio, you can set the working directory in two ways, a menu-driven way or by command.

Using Menus:

- In the bottom right window of the console, check to make sure the folder containing the data is shown. (If the folder is not shown, click "..." in the upper right corner of this window, and browse to the location of the folder.) Click "More" and "Set As Working Directory".

Using Commands:

Use the function `setwd()` to set the working directory path (see example below).

```
setwd("F:/Google Drive/ASL/Spring 2017/workshop") # Command to set working directory
getwd() # Displays current working directory

## [1] "F:/Google Drive/ASL/Spring 2017/workshop"
```

Notes:

- The "/" are forward slashes instead of backslashes here! Two backslashes, "\\" will also work. (We may do)
- The function `getwd()` will display your current working directory.
- Using `file.choose()` will bring up a window so that you can browse directly to the file you are reading in and use this path.

Reading in data

Each time you analyze data in R, you will need to call in the data at the beginning of your script. The two functions I most commonly use are `read.table()` and `write.table()`.

```
read.table() # Reads data into R from a file
scan() # Reads data into R from a file (good when you need to specify a data type
       # for each column.)
write.table() # Writes data from R to a file
```

Note: There are other "flavors" of `read.table` that we will not use (such as `read.csv`) since `read.table` is flexible enough (if you change the arguments) to include comma delimited data – so there is really no need to use the other function.

In RStudio: Select Tools – Import Dataset – From...

Example: Let's read in some practice data. The data file is 'practicedata.txt' and can be downloaded from http://web.as.uky.edu/statistics/users/klthomd/practicedata.csv.

```
practicedata = read.table('practicedata.csv', # Give filename first
               header=TRUE, # If filename has variable names, set header to TRUE.
                     # Otherwise, use header=FALSE
               sep=",", # Symbol separating data values (comma here)
               na.strings="NA", # Characters used to denote missing values
               #comment.char='#', # Character used to indicate comments in your file
               #skip=0, # number of lines of data file to skip before reading in data
               #nrows=1000 # maximum number of lines of data file to read in
                )
```

Once the data is read in, we can check to see what it looks like by clicking on 'practicedata' in the upper right panel of the RStudio window.

```
practicedata[1:5,] # Prints first 5 rows of the data
practicedata[,1:2] # Prints first 2 columns of the data

practicedata[,"expvar"] # One way to call the variable, expvar
practicedata$expvar # Another way to call the variable, expvar
```

Suppose the first 50 data points are from a control group and the last 50 are from a treatment group, and you want to consider only the treatment group. This means you need to define a new variable (we'll call it `trtmtdata`) containing only the data associated with the treatment group.

```
k=50 # number of observations in each group
n=100 # total number of observations
trtmtdata = practicedata[(k+1):n, ] # Print the 51st through 100th rows of the data
```

Data Formats

Check the format of the data we just read in:

```
class(practicedata)

## [1] "data.frame"

is.data.frame(practicedata)

## [1] TRUE
```

A data frame has the following properties:

- Each row must have the same number of columns (like a matrix variable in R)
- Each column can be a different data type (numerical, categorical, factor, etc.)

  ```
  class(practicedata$expvar)

  ## [1] "numeric"

  class(practicedata$groupvar)

  ## [1] "factor"
  ```

This makes a data frame more flexible than other data types in R, including:

- a matrix (which can only have data of one type)
- a character variable (always start and end with quotes; good for any categorical variable)
- a factor (if you are running an ANOVA, factors are good to use on grouping variables)
- a logical variable
- a list (good if you have variables of different sizes/types to store and manipulate)

There are certain times that we wish to convert certain types of variables into others. For instance, R reads in 'groupvar' as a factor by default. If we want to convert it to a character, we can use:

```
practicedata$groupvar = as.character(practicedata$groupvar)
is.character(practicedata$groupvar) # Checks to make sure conversion worked
```

Similar functions exist for matrices, numeric variables, and factors (`as.matrix(), as.numeric(), as.list()`, and `as.factor()`).

Note: If your dataset has multiple variable types (e.g., numeric and character), you do not want to try and coerce R into giving you a matrix. If we do, R converts the whole dataset into character variables (with the exception of any missing data).

```
mydatamatrix<-as.matrix(practicedata)
mydatamatrix[1,]

##      expvar      toyvar     toyvar2    groupvar     respvar   groupvar2
##  "6.437888" "10.013365" " 8.686682"   "Control"  "33.20271"         "A"
```

Hint for Reading in One Variable: If you read in one variable (a column of numbers), R creates a matrix with one column instead of a vector (what you want). In this situation, you can keep the first column of what you read in, and everything should work smoothly. This can be done in the same line as reading in the file. For example:

```
# d = read.table('onevariablefilename.txt')[,1]
```

Writing Data to Files

To write data to a file, the function `write.table()` is very flexible in terms of data formatting in the new file. As a default, the new file is created in the current working directory. For example, suppose you want to write the variables, `expvar, groupvar`, and the natural log of `respvar` to a new file. (Although, by saving your R script, it is not necessary to save the natural log of your data. If you need it again, you can re-run that line of code.)

```
getwd() #Check current working directory

## [1] "F:/Google Drive/ASL/Spring 2017/workshop"

resp.log=log(practicedata$respvar) # take the natural log of the response variable
data.to.write=cbind(practicedata$expvar,practicedata$groupvar,resp.log)
          # Binds the columns (variables) together
colnames(data.to.write) # print out column names of the new data

## [1] ""         ""         "resp.log"

colnames(data.to.write)<-c('expvar','groupvar','logrespvar') # rename the columns of
          # the new dataset
colnames(data.to.write) # print out column names of the new data again

## [1] "expvar"     "groupvar"   "logrespvar"

##To write data to a new .csv file:
write.table(data.to.write, # data to write to a file
  file='logdata.csv', # name of file you want to save data in
  quote=FALSE, # whether or not to put quotations around data
  col.names=TRUE, # whether or not to write column names to file
```

```
row.names=FALSE, # whether or not to write row names to file
sep=',', # what you want to put between data entries (commas and spaces are common)
append=FALSE, # whether or not to append existing data to the current file
na='NA' # string to use for missing values
)
```

Special thanks to Dr. Thompson for all her help in this presentation.