Kevin Chen

1/14/17

Notes

Getting Started

* Installing R on your computer
  + Go to r-project.org. Go to downloads, then pick a CRAN (comprehensive R archive network) mirror that’s closest to you. If you choose Windows, then choose base.
  + R is a command line application.
  + R is an implementation of the original language S (for statistics).
* Using RStudio
  + RStudio is an interface for R. Download and install it from www.rstudio.com
  + Choose the desktop version to install, not the web server.
  + On the left of RStudio is the console window that we originally had.
  + To change the various options of RStudio, go to Tools -> Global Options. Can change font size and editor theme, for example.
  + Create a new R script: File -> New File -> R Script. Can clear the console by doing Edit -> Clear Console or Ctrl + L
  + Workspace and history on the top right. Files, plots, available packages, and help.
* Taking a first look at the interface
  + Script window in the top left. Actual writing here.
  + # are comments, just like for Python. No multi-line comments. Shift + Ctrl + C comments selected lines of text.
  + To have something to go from the script window to show up in the console, press Ctrl + Enter (in the console, just press Enter).
  + While Matlab works on matrices, R works on vectors. Similar to Matlab, it prints the result of every line of code.
  + Example: Type a simple math equation (e.g. 8 + 5) in the script window, and run it. You get the original code and then the result, which in the example is [1] 13. (The [1] demonstrates this is the first index of the vector, 13 is the value at that index.)
  + a:b prints all numbers from a to b.
  + print(*string*) prints the string. String surrounded in double quotes.
  + Assignment operator: <- (equal sign can also be used, but this is frowned upon) Example x <- 1:5 assigns the values 1 through 5 to x. The output for this is shown in the workspace, not the console.
  + See what’s inside a variable by just typing the variable.
  + Instead of 1:5, you can do c(1, 2, 3, 4, 5) (c standing for concatenate). Can also do nonconsecutive numbers using c.
  + Can do a -> 2, but this is silly. Can do multiple assignments: a <- b <- c <- 3 assigns all of the variables to three.
  + If x and y are vectors of the same size, then can do vector math (e.g. x + y).
  + Google’s R Style Guide contains recommendations on how to write your code in R.
  + rm(*variables, separated, by, commas*) removes objects from the workspace. rm(list = ls()) clears the entire workspace.
* Installing and Managing packages
  + Go to the bottom right and select the packages tab.
  + Find all packages at <https://cran.r-project.org/web/views> (by category) or <https://cran.stat.ucla.edu/web/packages/available_packages_by_name.html> (by name) or crantastic.org
  + See current packages in the packages tab or run the command library().
  + Show the packages currently loaded by running the command search().
  + (To make the lines wrap, go to Tools -> Global Options -> Code. Check the soft-wrap R source files box.)
  + To install packages, go to Tools -> Install Packages or you can use scripts. (Scripts can be saved and replicated.) Use the command: install.packages(“*NameOfPackage*”).
  + (Getting help in R: Type help(*MethodName*) or ?*MethodName*)
  + library(“*NameOfPackage*”) or require(“*NameOfPackage*”) makes a package available; often used for loading in scripts. (require() is a bit better.)
  + Help for packages: library(help = “*NameOfPackage*”)
  + Bring up a list of vignettes (examples): vignette(package = “*NameOfPackage*”) brings it up in editor window. browseVignettes(package = “*NameOfPackage*”) opens a web page.
  + List of all vignettes for currently installed packages: vignette(). browseVignettes() gives you an HTML version. If links are dead, go to CRAN and search by name.
  + Update packages: Can be done by tools -> Check for Package updates – OR – in the bottom right window, packages -> Check for Updates – OR – update.packages() in the script.
  + Unload/remove packages: Be default all loaded packages are unloaded when R quits. Can also open Packages window and manually uncheck – OR – run the code detach(“*NameOfPackage*”, unload = TRUE)
  + To permanently remove (delete) a package: click the x bottom on the right of a package in the packages window – OR – remove.packages(“*NameOfPackage*”)
* Using built-in datasets in R
  + Use the datasets package to use built-in datasets (or data sets built into R)
  + See a list of the available datasets: data() or browseURL(“<http://stat.ethz.ch/R-manuel-R-devel/library/datasets/html/00Index.html>”)
  + More information on a specific dataset: ?*NameOfDataset*
  + Load a dataset from the package into the workspace: data(*NameOfDataset*)
  + To see the contents of a dataset, just type its name.
  + See the structure of a dataset: str(*NameOfDataset*). The first line in the output is contains the dimensions of the structure, and the number of entries within each category. Below that are the variables names (e.g. gender), each having its own categories (e.g. male, female).
  + If you click on the dataset in “Workspace”, you call the fix(*NameOfDataset*) function, which allows you to edit the dataset in a new window.
* Entering Data Manually
  + Creating sequential data: x1 <- 0:10 # assigns 0 to 10
  + Creating descending data: x2 <- 10:0 # assigns 10 to 0
  + Creating a sequence: x3 <- seq(10) #assigns 1 to 10
  + More complicated sequence: x4 <- seq(30, 0, by = - 3) # 30, 27, …, 0
  + Concatenate: x5 <- c(5, 2, 4, 1, 9)
  + Repeating values: x6 <-c(rep(10, 5)) #assigns five 10’s (10, 10, 10, 10, 10) to x6.
  + Enter data from the console: x6 <- scan # press enter after each entry. Press enter twice to stop.
  + List of all the objects in the workspace: ls()
* Importing data
  + Don’t import data from Excel. Better to do it from txt or csv file.
  + Text files: Load a spreadsheet that has been saved as tab-delimited text file. This command gives an error on missing data but works on complete data: *variable* <- read.table(“/path/to/file/filename.txt”, header = TRUE). (header = TRUE means the first line is a header, i.e. contains the column names)
  + Specifying the separators fixes the issue of the missing data: read.table(“/path/to/file/filename.txt”, header = TRUE, sep = “\t”) (if the data is separated by tabs)
  + Can View the data by clicking the variable in the workspace or by typing the command View(*variablename*)
  + CSV file: don’t have to specify delimiters for missing data because CSV means “comma separated values”. The command is read.csv(“/path/to/file/filename.csv”, header = TRUE, sep = “\t”)
* Converting tabular data to row data
  + (A data set in R is called UCBAdmissions, which contains the admissions statistics from the 1973 admissions in UCB. It demonstrates Simpson’s Paradox.)
  + plot(*dataset*) plots the data set.
  + Get marginal frequencies from original table: margin.table(*dataset*, *variablenumber*). Get variable numbers by looking at the order the variables are shown when displaying the structure of the data set (use the function str(*dataset*))
  + Get the grand total: margin.table(*dataset*)
  + Can save the marginal as new tables.
  + Show as proportions (instead of the actual count): prop.table(*marginal*)
  + (round(*dataset*, *numberofdecimals*) rounds all the entries in *dataset* to *numberofdecimals* decimal places.)
  + Go from table to one row per case: *variable1* <- as.data.frame.table(*dataset*) (coerces to data frame). *variable2* <- lapply(*variable1*, function(x)rep(x, *variable1*$Freq)) (repeats each row by Freq). *variable3* = as.data.frame(*variable2*) (converts from list back to data frame). You can then remove the frequencies column: *variable4* <- *variable3*[, -*Freqvariablenumber*] (this takes *variable3*, copies all the rows since we specify first rows then columns within the brackets – leaving it blank means take all the rows, and the minus means to discard a column)
  + *dataset*.rows[1:10, ] shows the first ten rows of data (and all columns)
* Working with colors in R
  + Create a bar plot: barplot(*dataset*). Default color is grey.
  + R has names for 657 colors. Get list of color names by entering colors() or referencing them here: <http://research.stowers-institute.org/efg/R/Color/Chart/>
  + Change color of bar plot: barplot(*dataset*, col = “*colorname*”). Can type in color instead of col, but you only need to feed in enough letters to allow R to distinguish it from the other arguments.
  + Can refer by index instead: barplot(*dataset*, col = colors()[*index*])
  + Can use RGB Triplets: can specify colors in the 0-1 range, or the 0-255 range by adding “max = 255”. barplot(*dataset*, col = rgb(*redvalue*, *greenvalue*, *bluevalue*) allows you to use rgb values in the 0-1 range. barplot(*dataset*, col = rgb(*redvalue*, *greenvalue*, *bluevalue*, max = 255) uses 0-255 range. barplot(*dataset*, col = “#*HEXRGBValue*”) allows you to use hex values for the RGB value. (Can use function col2rgb(“*colorname*”) to convert a color name to its RGB value.)
  + Specifying multiple colors: use a vector. barplot(x, col = c(“colors”, “separated”, “by”, “commas”)). If there are fewer colors than you have columns, the colors will cycle through.
  + Using color palettes: use the colorspace package (can find more info using help(package = colorspace) and ?palette). Various built-in palettes. pallete() returns the current palette(). Use the function: barplot(*dataset*, col = *palettename*(*valuesneeded*)). Return to default palette: palette(“default”)
* Exploring color with Colorbrewer
  + One of the best ways to work with color is to use an external package called Colorbrewer. Explore it here: <http://colorbrewer2.org>.
  + Install and load the package: install.packages(“RColorBrewer”) require(“RColorBrewer”)
  + Show all the palettes in a graphics window: display.brewer.all(), and then click zoom.
  + Types of palettes: sequential (progression from light to dark), qualitative (completely different), divergent(dark to light to dark).
  + To see palette colors in a separate window: display.brewer.pal(*NumberOfDesiredColors*, “*NameOfPalette*”)
  + Save the palette as a vector: *variable* = brewer.pal(*NumberOfDesiredColors*, “*NameOfPalette*”). Then make the bar plot: barplot(x, col = *variable*)

Charts for One Variable

* Creating bar charts for categorical variables
  + Quickest method: plot(*dataset*$*variable*)
  + To create a bar plot, you must create a table that has the frequencies for each level of the variable. *storingvariable* <- table(*dataset*$*variable*). Then make a barplot: barplot(*storingvariable*).
  + Ordering a bar plot: “barplot(*storingvariable*[order(*orderingvariable*, decreasing = TRUE)]” orders the bar plot in decreasing order. *orderingvariable* is often the same as the *storingvariable*, but you can order the variable by another variable. To order it in increasing order, omit “, decreasing = TRUE”.
  + Setting margins: par(oma = c(*bot*, *left*, *top*, *right*)) sets the outside margins. par(mar = c(*bot*, *left*, *top*, *right*)) sets the plot margins.
  + Displaying a horizontal bar chart: Pass in additional arguments into the barplot method: horiz = TRUE, las = 1. (las gives the orientation of axis labels. 1 means always horizontal.)
  + Some other options: pass in additional arguments into the barplot method: border = NA (no borders on bars), main = *TitleString* (sets the title of the graph), xlab = *XVariableString* (sets the name of the variable). Look above for colors.
* Creating pie charts for categorical variables