Introduction to R

Andy Grogan-Kaylor

September 24, 2017

Contents

Figure 1: R Logo

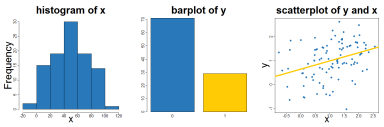

Figure 2: Graphical Possibilities in Base R

1 *Background*

R is open source, and therefore free, statistical software with particular strengths in obtaining, analyzing and visualizing data. R has an admittedly steep learning curve. However, I believe that it is possible to teach R in an accessible way, and that a little bit of R can take you a long way.

This document is a brief introduction to R[1], in order to create a kind of "cheat sheet" that can be presented in a few pages. Commands that you actually type into R are represented in `courier font`. `mydata` is the name of your data set. `x` and `y` and `z` refer to variables in your data. More documentation on any command is usually available via `help(command)` or `??command`. The R interface makes it extremely easy to do rapid interactive data analysis. Hit **"Up-Arrow"**

[1] This document is inspired by my long-standing "Two Page Stata" document: (PDF) (HTML) .

to recall the most recent command, which you can then quickly edit and resubmit. Remember also that one often submits a command or set of commands from a script window. The general idea of many R commands[2] is:

[2] The $ sign is a kind of "connector". mydata$x means: "The variable x in the dataset called mydata".

```
command(data=mydata, ...variables..., options)
```

or

```
command(mydata$xvar, options)
```

Sometimes, it is not necessary to use any options since some authors of R have done a good job of thinking about the defaults. R can make use of long pathnames[3] to files like:

[3] Note that R uses forward slashes "/" instead of backslashes "\".

```
C:/Users/user1/Desktop/mydata.sav
```

2 *Base R and Libraries*

Much of this guide makes use of what is most often called **Base R**, the R that you get when you install the R software, and RStudio, on your computer.

A great deal can be accomplished with **Base R**. However, as you grow in your use of R, you will likely frequently need to make use of libraries, which are invoked by the `library(...)` command. Before using a library you need to install it. Below is an example of installing the *ggplot2* advanced graphics library.

You would need to install the library only once. Installation can also be accomplished from the "Packages" tab in RStudio.

```
install.packages("ggplot2")
```

Then start the library when you are using R by typing...

```
library(ggplot2)
```

3 *Working Directory*

It is often helpful to simply set your working directory to a particular location and by default, files will be accessed from, and saved to, that directory e.g.:

```
getwd() # "get", or find out, your working directory

setwd("C:/Users/user1/Desktop/") # set your working directory
```

4 *Writing R Code or Script*

R is a command or syntax based program, and many advanced functions are only available via syntax.

> R Commands are stored in a *script* or *code* file that usually ends in .R, e.g. `myRscript.R`. The command file is distinct from your actual data, stored in an .RData file, e.g. `mydata.RData`.
>
> Base R can sometimes be cryptic.
>
> However, a little bit of Base R can go a long way, and you can get a great learning return for a little bit of investment in learning Base R.

I should mention here the new additions to the R language of the new libraries which make up the **tidyverse**. Learning the **tidyverse** requires an additional investment in learning, however the **tidyverse** makes many improvements to the R language and functionality.

5 *Graphical User Interface*

A good **Graphical User Interface** (GUI) can make some of the base functionality of R available without the use of syntax. **RCommander** is the best GUI, and can be installed from the command line by typing:

```
install.packages("Rcmdr", dependencies=TRUE)
```

RCommander can make some tasks easier, but the syntax that it produces can sometimes be non-intuitive. Often it is easiest (and more in the interests of replicable research) just to learn how to write the `R code` that accomplishes a particular task. Further, your learning may go quicker if you bypass **RCommander** altogether and simply learn how to write R code.

RStudio is an **Integrated Development Environment (IDE)** that can be run simultaneously with **RCommander** and provides an easier working enivronment for R Software. If all the software is installed, Start **RStudio** to start R, then type `library(Rcmdr)` to start **RCommander**.

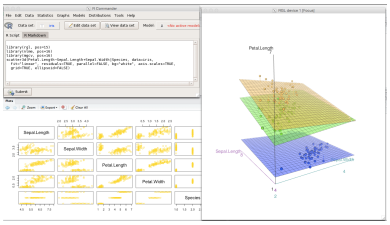

Figure 3: screenshot of RCommander

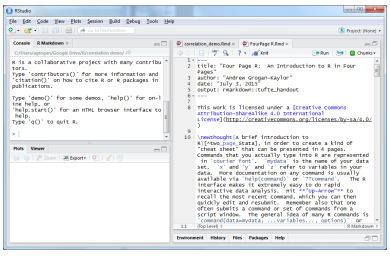

Figure 4: screenshot of RStudio

6 *Get Your Data*

R most easily makes use of data in R format. Data can be loaded with the `load()` command.

```
load("pathto/myRfile.RData")
```

R can also read *comma separated values (csv)*.

```
library(readr) # to read csv

mydata <- read_csv("myCSVfile.csv")
```

R can easily import well-formatted data from other packages} like SPSS, Stata, or Excel[4].

[4] These instructions assume you have `setwd()` appropriately, or alternatively are specifying a full pathname and filename.

```
library(foreign) # library for importing from stats software

mydata <- read.spss("mySPSSfile.sav") # SPSS

mydata <- read.spss("myStatafile.dta") # Stata

library(readxl) # library for importing Excel files

mydata <- read_excel("mySpreadsheet.xls") # Excel
```

Working with a *subset* of your data (i.e. fewer variables rather than many many variables) is often helpful. The subset function can be especially helpful.

```
mydata_subset <- subset(mydata,
                        select = c(id, sex, income))
```

7 Save and Document Your Work

Use the Script Editor to save R commands that you want to use again, or to modify for the next project, as well as to create an "audit trail" of your work so that your workflow is documented and replicable.

8 Process Your Data

R recognizes two basic kinds of variables: *continuous variables* (which R calls *"numeric" variables*) which are often scales like income, mental health, or neighborhood quality; and *categorical variables* (which R calls *"factor variables"*) like race, gender or religion. R seems to make a stronger distinction between these two types of variables than some other statistical software.[5] It can thus sometimes be useful to change variables from one type to another:

[5] In many cases, this is very helpful in that R recognizes that the type of variable calls for a certain kind of statistic or graph, or vice versa. In other cases, this may be the source of an *error message*.

```
mydata$x <- as.factor(mydata$y)

mydata$y <- as.numeric(mydata$x)
```

Data with missing values, often represented as negative numbers (e.g. -99, -9, -8) needs to be recoded so that the missing values are represented as a missing value character ("NA") that R knows to exclude from calculations.

```
mydata$x[mydata$x == -9] <-NA # Example 1

mydata$x[mydata$x == -8] <-NA # Example 2
```

It is often convenient to rename your data so that the variables have more intuitively understandable names e.g.

```
mydata$age <- mydata$var123

mydata$gender <- mydata$var456
```

It is sometimes useful to sort your data. `sort(mydata$x)` will sort mydata by the values of `x`.

9 *Scales or Measures*

You can sum the items of a scale into a scale as follows:

```
myscale <- x1 + x2 + x3
```

You can test the alpha reliability of this scale with the following syntax:

```
myscale_data <- subset(mydata, select = c(x1, x2, x3))
```

The syntax above create a dataframe of only the scale items.
Then `library(psych)` and `alpha(myscale_data)`.

10 *Descriptive Statistics*

`summary(mydata$x)` gives you basic descriptive statistics for a variable, such as the mean (average). Especially useful for continuous variables. Use `summary(mydata)` to summarize every variable in your data. `describe(mydata$x)` from `library(psych)` will often give you a nicer summary of your variables that is closer to what you want for an academic paper or agency report.

`table(mydata$x)` gives you a frequency distribution for your variable. Especially useful for *factor variables*. `prop.table(table(mydata$x))` will give you a table of proportions. Calling up `library(descr)` and then using `freq(mydata$x)` will give you a more nicely formatted frequency distribution.

11 *Bivariate Statistics*

Tabulating two categorical variables (*factor variables*) together gives you a cross-tabulation of those variables, e.g:

```
table(mydata$x, mydata$y)
```

then

```
chisq.test(table(mydata$x, mydata$y))
```

will give you a chi-square test of the relationship of `x` and `y`.

```
cor(mydata[,c("x","y")], use="complete.obs")
```

will give you the correlation of continuous variables `x` and `y`[6].

[6] Here is an example where R turns a simple issue into a difficult one, and the syntax is frankly less than elegant, and not intuitive. I don't have this syntax memorized. I use `library(Rcmdr)` if I need to test, or create a script for, a correlation.

```
cor.test(mydata$x, mydata$y,
         alternative="two.sided",
         method="pearson")
```

will test the statistical significance of this correlation.

```
numSummary(mydata$x, groups=mydata$z)
```

gives you a summary of continuous variable `x` by *factor variable* `z`.

```
t.test(mydata$x~mydata$z)
```

runs a t-test of continuous variable `x` over *factor variable* `z`.

```
aov(x ~ z, data=mydata)
```

runs the corresponding ANOVA of continuous variable `x` across *factor variable* `z`.

12 *Multivariate Statistics*

```
mymodel <- lm(y ~ x + z, data=mydata)
```

runs a regression (linear model) of `y` on `x` and `z`, type

```
summary(mymodel)
```

to display the results

13 *Graphing*

```
hist(mydata$x)
```

will give you a nice display of one continuous variable.

```
hist(mydata$x, main="...", xlab="...")
```

gives a nicer looking graph.

```
barplot(table(mydata$x))
```

gives similar results when x is a *factor variable*.

```
plot(mydata$y, mydata$x)
```

gives you a twoway scatterplot of your data
A more nicely labelled graph can be obtained with:

```
plot(y, x,
     main= "...",
     xlab= "...",
     ylab= "...")
```

`abline(lm(mydata$y~mydata$x))` will add a linear fit line to a scatterplot that you have already constructed.
`abline(lm(mydata$y~mydata$x), col="gold", lwd=5)` will be a nicer looking fit line.

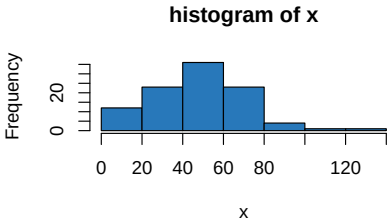

Figure 5: histogram of x

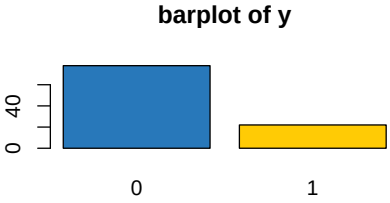

Figure 6: barplot of y

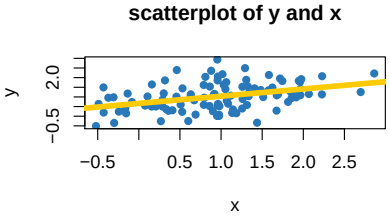

Figure 7: scatterplot of y against x

14 *Comments, Questions and Corrections*

Comments, questions and corrections most welcome and may be sent to: Andrew Grogan-Kaylor @ `http://www.umich.edu/~agrogan` & @ `agrogan@umich.edu`.

Last updated: `September 24 2017` at `12:12`