A (very basic) R tutorial

Johannes Karreth

Applied Introduction to Bayesian Data Analysis

1 Getting started

The purpose of this tutorial is to show the very basics of the R language so that participants who have not used R before can complete the first assignment in this workshop. For information on the thousands of other features of R, see the suggested resources below.
In this tutorial, R code that you would enter in your script file or in the command line is preceded by the > character, and by + if the current line of code continues from a previous line. You do not need to type this character in your own code.

1.1 Installing R and RStudio

The most recent version of R for all operating systems is always located at `http://www.r-project.org/index.html`. Go directly to `http://lib.stat.cmu.edu/R/CRAN/`, and download the R version for your operating system. Then, install R.

To operate R, you should rely on writing R scripts. We will write these scripts in RStudio. Download RStudio from `http://www.rstudio.org`. Then, install it on your computer. Some text editors also offer integration with R, so that you can send code directly to R. RStudio is generally the best solution for running R and maintaining a reproducible workflow.

Lastly, install LaTeX in order to compile PDF files from within RStudio. To do this, follow the instructions under `http://www.jkarreth.net/latex.html`, "Installation". You won't have to use LaTeX directly or learn how to write LaTeX code in this workshop.

1.2 Opening RStudio

Upon opening the first time, RStudio will look like Figure 1.
The window on the left is named "Console". The point next to the blue "larger than" sign > is the "command line". You can tell R to perform actions by typing commands into this command line. We will rarely do this and operate R through script files instead.

1.3 R packages

Many useful and important functions in R are provided via packages that need to be installed separately. You can do this by using the Package Installer in the menu (Packages & Data > Package Installer in R or Tools > Install Packages... in RStudio), or by typing

```
> install.packages("foreign")
```

in the R command line. Next, every time you use R, you need to load the packages you want to use: type

```
> library(foreign)
```

in the R command line.

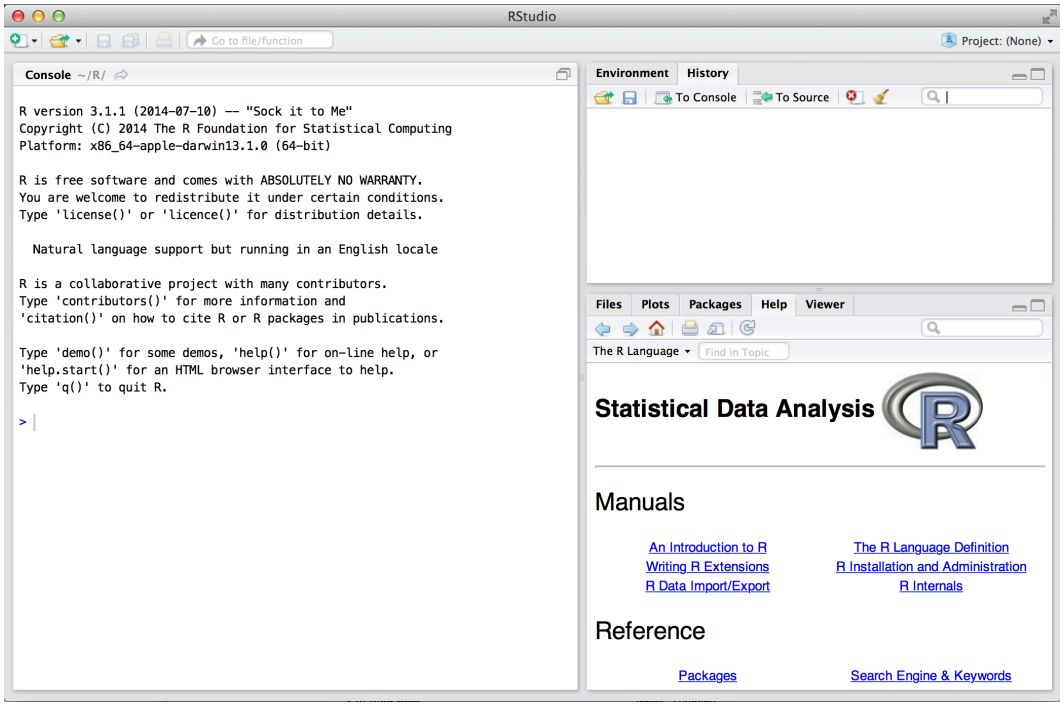


Figure 1: RStudio.

1.4 Working directory

In most cases, it is useful to set a project-specific working directory—especially if you work with graphics that you want to have printed to .pdf or .eps files. You can set the WD with this command:

```
> setwd("/Users/johanneskarreth/Documents/Uni/Teaching/CPH-Bayes/Tutorials/Intro to R")
```

RStudio offers a very useful function to set up a whole project (File > New Project...). Projects automatically create a working directory for you.

1.5 R help

Within R, you can access the help files for any command that exists by typing `?commandname` or, for a list of the commands within a package, by typing `help(package = packagename)`. So, for instance:

```
> ?rnorm
> help(package = "foreign")
```

1.6 Workflows and conventions

There are many resources on how to structure your R workflow (think of routines like the ones suggested by J. Scott Long in *The Workflow of Data Analysis Using Stata*), and I encourage you to search for and maintain a consistent approach to working with R. It will make your life much, much easier—with regards to collaboration, replication, and general efficiency. A few really important points that you might want to consider as you start using R:

- Never type commands into the R command line. Always use a script file, from which you can send (via RStudio, Emacs, ...) commands to R, or at least copy and paste them into R.
- Comment your script files! Comments are indicated by the # sign:

```
> # This is a comment
```

- Save your script files in a project-specific working directory.

- Use a consistent style when writing code. A good place to start is Google's style guide: `http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html`.
- Do not use the `attach()` command.

1.7 Error messages

R will return error messages when a command is incorrect or when it cannot execute a command. Often, these error messages are informative by themselves. You can often get more information by simply searching for an error message on the web. Here, I try to add 1 and the letter a, which does not (yet) make sense:

```
> 1 + a

## Error in eval(expr, envir, enclos):  object 'a' not found
```

As your coding will become more complex, you may forget to complete a particular command. For example, here I want to add 1 and the product of 2 and 4. But I forget to close the parentheses around the product:

```
> 1 + (2 * 4
+ )

## [1] 9
```

You will notice that the little > on the left changes into a +. This means that R is offering you a new line to finish the original command. If I type a right parenthesis, R returns the result of my operation.

1.8 Useful resources

As R has become one of the most popular programs for statistical computing, the number of resources in print and online has increased dramatically. Searching for terms like "introduction to R software" will return a huge number of results.
Some (of the many) good books and e-books that I have encountered and found useful are:

- Fox and Weisberg, *An R and S-Plus Companion to Applied Regression* (2011, print).
- `statmethods.net`. This website offers well-explained computer code to complete most of the data analysis tasks we use in this workshop.
- Maindonald and Braun, *Data Analysis and Graphics Using R* (2006, print).
- Verzani, *simpleR - Using R for Introductory Statistics* (`http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf`).

2 R and object-oriented programming

R is an object-based programming language. This means that you, the user, create objects and work with them. Objects can be:

- Numbers:

```
> x <- 1
> x

## [1] 1

> y <- 2
> x + y

## [1] 3

> x * y
```

```
## [1] 2

> x / y

## [1] 0.5

> y^2

## [1] 4

> log(x)

## [1] 0

> exp(x)

## [1] 2.718282
```

- Vectors:

```
> xvec <- c(1, 2, 3, 4, 5)
> xvec

## [1] 1 2 3 4 5

> xvec2 <- seq(from = 1, to = 5, by = 1)
> xvec2

## [1] 1 2 3 4 5

> yvec <- rep(1, 5)
> yvec

## [1] 1 1 1 1 1

> zvec <- xvec + yvec
> zvec

## [1] 2 3 4 5 6
```

- Matrices:

```
> mat1 <- matrix(data = c(1, 2, 3, 4, 5, 6), nrow = 3, byrow = TRUE)
> mat1

##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6

> mat2 <- matrix(data = seq(from = 6, to = 3.5, by = -0.5),
+                nrow = 2, byrow = T)
> mat2

##      [,1] [,2] [,3]
## [1,]  6.0  5.5  5.0
## [2,]  4.5  4.0  3.5

> mat1 %*% mat2

##      [,1] [,2] [,3]
## [1,]   15 13.5   12
## [2,]   36 32.5   29
## [3,]   57 51.5   46
```

- Data frames (equivalent to data sets):

```
> y <- c(1, 1, 3, 4, 7, 2)
> x1 <- c(2, 4, 1, 8, 19, 11)
> x2 <- c(-3, 4, -2, 0, 4, 20)
> name <- c("Student 1", "Student 2", "Student 3", "Student 4",
+           "Student 5", "Student 6")
> mydata <- data.frame(name, y, x1, x2)
> mydata


##        name y x1 x2
## 1 Student 1 1  2 -3
## 2 Student 2 1  4  4
## 3 Student 3 3  1 -2
## 4 Student 4 4  8  0
## 5 Student 5 7 19  4
## 6 Student 6 2 11 20
```

2.1 Random numbers and distributions

You can use R to generate (random) draws from distributions. This will be important in the first assignment. For instance, to generate 1000 draws from a normal distribution with a mean of 5 and standard deviation of 10, you would write:

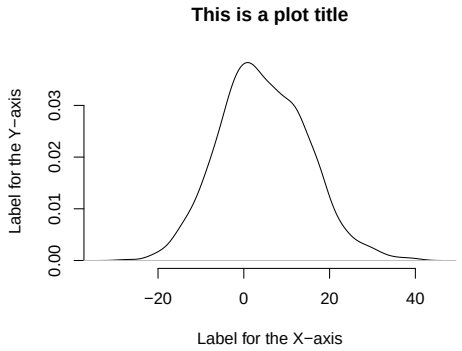
```
> draws <- rnorm(1000, mean = 5, sd = 10)
> summary(draws)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -26.750  -1.392   5.065   4.951  11.070  34.460
```

You can then use a variety of plotting commands (see for more below) to visualize your draws:

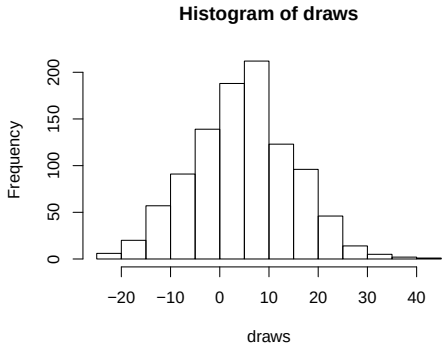
- Density plots:

```
> draws <- rnorm(1000, mean = 5, sd = 10)
> plot(density(draws), main = "This is a plot title",
+      xlab = "Label for the X-axis", ylab = "Label for the Y-axis",
+      frame = FALSE)
```



- Histograms:

```
> draws <- rnorm(1000, mean = 5, sd = 10)
> hist(draws)
```



2.2 Extracting elements from an object

- Elements from a vector:

```
> vec <- c(4, 1, 5, 3)
> vec[3]

## [1] 5
```

- Variables from a data frame:

```
> mydata$x1

## [1]  2  4  1  8 19 11

> mydata$names

## NULL
```

- Columns from a matrix:

```
> mat1[ ,1]

## [1] 1 3 5
```

- Rows from a matrix:

```
> mat1[1, ]

## [1] 1 2
```

- Elements from a list

```
> mylist <- list(x1, x2, y)
> mylist[[1]]

## [1]  2  4  1  8 19 11
```

3 Working with data sets

In most cases, you will not type up your data by hand, but use data sets that were created in other formats. You can easily import such data sets into R. Here are the most common options. Most of them use the foreign package, hence you need to load that package before using these commands:

```
> library(foreign)
```

Note that for each command, many options (in R language: arguments) are available; you will most likely need to work with these options at some time, for instance when your source dataset (e.g., in Stata) has value labels. Check the help files for the respective command in that case.

- **Tables:** If you have a text file with a simple tab-delimited table, where the first line designates variable names:

```
> mydata.table <- read.table("http://www.jkarreth.net/files/data.txt",
+                            header = TRUE)
> head(mydata.table)


##            y         x1         x2
## 1 -0.1629267  1.6535472  0.3001316
## 2  1.3985720  1.4152763 -0.9544489
## 3  0.8983962  0.4199516 -0.4580181
## 4 -1.6484948  0.7212208  0.9356037
## 5  0.2285570 -1.1969352 -1.1368931
```

- **CSV files:** If you have a text file with a simple tab-delimited table, where the first line designates variable names:

```
> mydata.csv <- read.csv("http://www.jkarreth.net/files/data.csv",
+                        header = TRUE)
> head(mydata.csv)


##            y         x1         x2
## 1 -0.1629267  1.6535472  0.3001316
## 2  1.3985720  1.4152763 -0.9544489
## 3  0.8983962  0.4199516 -0.4580181
## 4 -1.6484948  0.7212208  0.9356037
## 5  0.2285570 -1.1969352 -1.1368931
```

- **SPSS files:** If you have an SPSS data file, you can do this:

```
> # mydata.spss <- read.spss("http://www.jkarreth.net/files/data.sav",
> # use.value.labels = TRUE)
```

- **Stata files:** If you have a Stata data file, you can do this:

```
> mydata.dta <- read.dta("http://www.jkarreth.net/files/data.dta",
+                        convert.dates = TRUE, convert.factors = TRUE)
> head(mydata.dta)


##            y         x1         x2
## 1 -0.1629267  1.6535472  0.3001316
## 2  1.3985720  1.4152763 -0.9544489
## 3  0.8983962  0.4199516 -0.4580181
## 4 -1.6484948  0.7212208  0.9356037
## 5  0.2285570 -1.1969352 -1.1368931
```

Describing data

To obtain descriptive statistics of a dataset, or a variable, use the `summary` command:

```
> summary(mydata.dta)

##        y                 x1                x2
##  Min.   :-1.6485   Min.   :-1.1969   Min.   :-1.1369
##  1st Qu.:-0.1629   1st Qu.: 0.4200   1st Qu.:-0.9544
##  Median : 0.2286   Median : 0.7212   Median :-0.4580
##  Mean   : 0.1428   Mean   : 0.6026   Mean   :-0.2627
##  3rd Qu.: 0.8984   3rd Qu.: 1.4153   3rd Qu.: 0.3001
##  Max.   : 1.3986   Max.   : 1.6535   Max.   : 0.9356

> summary(mydata$y)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    1.25    2.50    3.00    3.75    7.00
```

You can access particular quantities, such as standard deviations and quantiles (in this case the 5th and 95th percentiles), with the respective functions:

```
> sd(mydata$y)

## [1] 2.280351

> quantile(mydata$y, probs = c(0.05, 0.95))

##   5%  95%
## 1.00 6.25
```

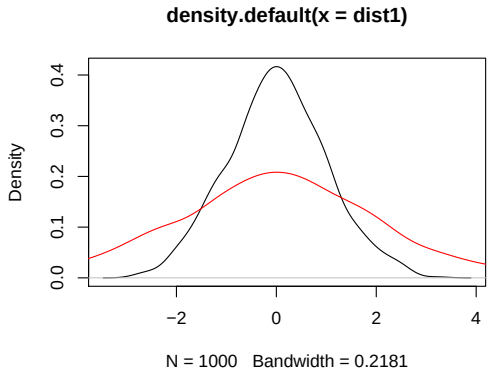
4 Creating figures

R offers several options to create figures. We will work with the so-called "base graphics", mostly using the `plot()` function, and the ggplot2 package.

4.1 Base graphics

R's base graphics are very versatile and, in our workshop, ideal for creating quick plots to inspect objects. These graphs are built sequentially, beginning with the `plot()` command applied to an object. So, for instance to plot the density of 1000 draws from a normal distribution, you would use the following code. I'm using the `set.seed()` command here before every simulation to ensure that the same values are drawn when you try these commands and make these plots.

```
> set.seed(123)
> dist1 <- rnorm(n = 1000, mean = 0, sd = 1)
> set.seed(123)
> dist2 <- rnorm(1000, mean = 0, sd = 2)
> plot(density(dist1))
> lines(density(dist2), col = "red")
```

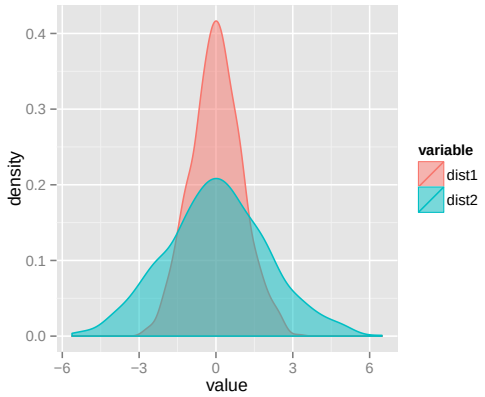


4.2 The ggplot2 package

The ggplot2 package has become popular because its language and plotting sequence can be somewhat more convenient (depending on users' background), especially when working with more complex datasets. For plotting Bayesian model output, ggplot2 offers some useful features.
ggplot2 needs to be first loaded as an external package. Its key commands are `ggplot()` and various plotting commands. All commands are added via +, either in one line or in a new line to an existing ggplot2 object. The command below contains a couple more data manipulation steps that will come in handy for us later; we will discuss them in the workshop. When trying the code below, have a look at the structure of the `dist.dat` object to see what's going on.

```
> library(ggplot2); library(reshape2)
> set.seed(123)
> dist1 <- rnorm(n = 1000, mean = 0, sd = 1)
> set.seed(123)
```

```
> dist2 <- rnorm(1000, mean = 0, sd = 2)
> dist.df <- data.frame(dist1, dist2)
> dist.df <- melt(dist.df)
> normal.plot <- ggplot(data = dist.df, aes(x = value, colour = variable, fill = variable))
> normal.plot <- normal.plot + geom_density(alpha = 0.5)
> normal.plot
```



ggplot2 offers plenty of opportunities for customizing plots; we will also encounter these later on in the workshop. You can also have a look at Winston Chang's *R Graphics Cookbook* for plenty of examples of ggplot2 customization: `http://www.cookbook-r.com/Graphs`.

4.3 Exporting graphs

Plots created via base graphics can be printed to a PDF file using the `pdf()` command. This code:

```
> set.seed(123)
> dist1 <- rnorm(n = 1000, mean = 0, sd = 1)
> set.seed(123)
> dist2 <- rnorm(1000, mean = 0, sd = 2)
> pdf("normal_plot.pdf", width = 5, height = 5)
> plot(density(dist1))
> lines(density(dist2), col = "red")
> dev.off()

## pdf
##   2
```

will print a plot named normal_plot.pdf of the size 5×5 inches to your working directory.
Plots created with ggplot2 are best saved using the `ggsave()` command:

```
> ggsave(plot = normal.plot, filename = "normal_ggplot.pdf", width = 5, height = 5)
```

5 Integrating writing and data analysis

For project management and replication purposes, it is a great idea to combine your data analyis and writing in one framework. RMarkdown, Sweave and knitr are great solutions for this. The RStudio website has a good explanation of these options: `http://rmarkdown.rstudio.com` and `https://support.rstudio.com/hc/en-us/articles/200552056-Using-Sweave-and-knitr`. This tutorial was written using knitr.