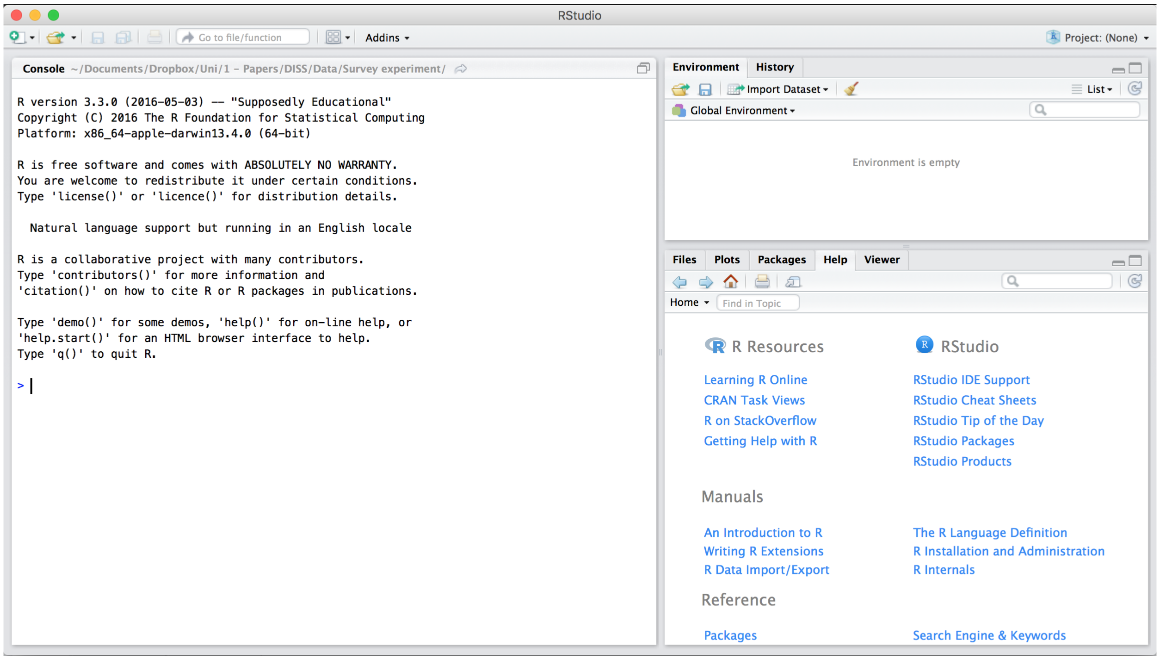
Applied Bayesian Modeling
A (very basic) R tutorial

Johannes Karreth
Ursinus College
jkarreth@ursinus.edu

ICPSR Summer Program 2017

Contents



```
R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Figure 1: RStudio.

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. Note that copying and pasting code from the PDF version of this tutorial may lead to errors when trying to execute code. Please copy code from the R script used to produce this tutorial; this script can be found at `https://www.dropbox.com/s/jv1nvep7bqfsekv/Lab1_R.R?raw=1`.

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 Typing R commands

In the following pages, I walk you through some basic R commands. In this tutorial and most other materials you will see in this workshop, R commands and the resulting R output will appear in light grey boxes. Output in this tutorial is always preceded by two ## signs.

To begin, see how R responds to commands. If you type a simple mathematical operation, R will return its result(s):

```
> 1 + 1

## [1] 2
```

```
> 2 * 3

## [1] 6
```

```
> 10/3

## [1] 3.333333
```

1.4 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. 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 as I haven't defined an object `a` yet and numbers and letters cannot be added:

```
> 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 unless I add the parenthesis at the end of the line, or in the immediately following line, this code won't execute:

```
> 1 + (2 * 4)

## [1] 9
```

While executing this command and looking at the console, 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.

1.5 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("rio")
```

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

```
> library("rio")
```

in the R command line. You only need to *install* packages once on your (or any) computer, but you need to *load* them anew in each R session.

Alternatively, if you only want to access one particular function from a package, but do not want to load the whole package, you can use the `packagename::function` option.

1.6 Working directory

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

```
> setwd("/Users/johanneskarreth/Documents/Dropbox/Uni/9 - ICPSR/2017/Applied Bayes/Tutorials/1 - R Basics/")
```

You can typically see your current working directory on top of the R console in RStudio, or you can obtain the working directory with this command:

```
> getwd()

## [1] "/Users/johanneskarreth/Documents/Dropbox/Uni/9 - ICPSR/2017/Applied Bayes/Tutorials/1 - R Basics"
```

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

1.7 Comments

R scripts contain two types of text: R commands and comments. Commands are executed and perform actions. Comments are part of a script, but they are not executed. Comments begin with the # sign. Anything that follows after a # sign in the same line will be ignored by R. Compare what happens with the following two lines:

```
> 1 + 1

## [1] 2

> # 1 + 1
> 1 + 1  # + 3

## [1] 2
```

You should use comments frequently to annotate your script files in order to explain to yourself what you are doing in a script file.

1.8 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 = "rio")
```

1.9 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 or the console. Always use a script file in RStudio and execute your code from this script file using the Run button or the Command + Return (Mac) or Control + Return (Windows) key combination.
- Always create and specify a working directory at the beginning of a script file. This will ensure that all input and output of your project-specific work is in a location that makes sense.
- Comment your script files!
- Save your script files in a project-specific working directory (e.g. via the Project structure in RStudio).
- Use a consistent style when writing code. A good place to start is this style guide: `http://adv-r.had.co.nz/Style.html`. Read through this style guide in the first week of the semester and use this style from then on.
- In script files, try to break lines after 80 characters to keep your files readable.
- Do not use the `attach()` command.

1.10 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 resources that I have encountered and found useful are:

- ICPSR workshops on R
- 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-oriented programming language. This means that you, the user, create objects and work with them. Objects can be of different types. To **create** an object, first type the object name, then the “assignment character”, a leftward arrow `<-`, then the content of an object. To **display** an object, simply type the object’s name, and it will be printed to the console. [1]

You can then apply functions to objects. Most functions have names that are somewhat descriptive of their purpose. For example, `mean()` calculates the mean of the numbers within the parentheses, and `log()` calculates the natural logarithm of the number(s) within the parentheses.

Functions consist of a function name, the function’s arguments, and specific values passed to the arguments. In symbolic terms:

```
> function_name(argument1 = value, argument2 = value)
```

Here is a specific example of the function `abbreviate`, its first argument `names.arg`, and the value `"Regression"` that I provide to the argument `x`:

```
> abbreviate(names.arg = "Regression")
```

The following are the types of objects you need to be familiar with:

- Scalars
- Vectors of different types
  - Numeric (numbers)
  - Character (words or letters): always entered between quotation marks `"`
  - Factor (numbers with labels)
  - Logical (TRUE or FALSE)
- Matrices
- Data frames
- Lists

Below, you find some more specific examples of different types of objects.

- Numbers:

[1] A good overview of types of objects is here: `http://www.statmethods.net/input/datatypes.html`. Read this page and then continue this tutorial.

```
> 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:

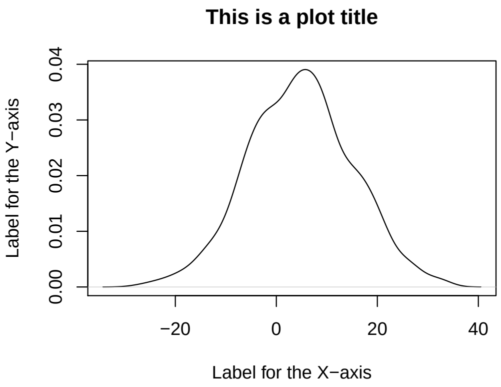
```
> draws <- rnorm(1000, mean = 5, sd = 10)
> summary(draws)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -31.490  -1.030   5.282   5.267  11.470  34.260
```

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

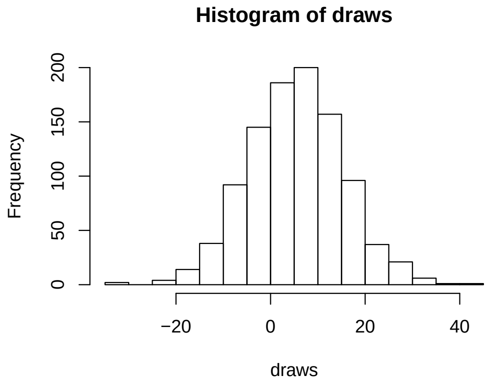
- Density plots:

```
> 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")
```



- 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. The rio package allows you to import data sets in a variety of formats with one single function, `import()`. You need to first load the package:

```
> library("rio")
```

The `import()` function ”guesses” the format of the data from the file type extension, so that a file ending in `.csv` is read in as a comma-separated value file. If the file typ extension does not reveal the type of data (e.g., a tab-separated file saved with a `.txt` extension), you need to provide the `format` argument, as you see in the first example below. See the help file for `import()` for more information. 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.

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

  ```
  > mydata_from_tsv <- import("http://www.jkarreth.net/files/mydata.txt", format = "tsv")
  > head(mydata_from_tsv)


  ##       y    x1    x2
  ## 1 -0.56  1.22 -1.07
  ## 2 -0.23  0.36 -0.22
  ## 3  1.56  0.40 -1.03
  ## 4  0.07  0.11 -0.73
  ## 5  0.13 -0.56 -0.63
  ## 6  1.72  1.79 -1.69
  ```

  Alternatively, use `read.table()` specifically for tab-separated files:

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


  ##       y    x1    x2
  ## 1 -0.56  1.22 -1.07
  ## 2 -0.23  0.36 -0.22
  ## 3  1.56  0.40 -1.03
  ## 4  0.07  0.11 -0.73
  ## 5  0.13 -0.56 -0.63
  ## 6  1.72  1.79 -1.69
  ```

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

  ```
  > mydata_from_csv <- import("http://www.jkarreth.net/files/mydata.csv")
  > head(mydata_from_csv)


  ##       y    x1    x2
  ## 1 -0.56  1.22 -1.07
  ## 2 -0.23  0.36 -0.22
  ## 3  1.56  0.40 -1.03
  ## 4  0.07  0.11 -0.73
  ## 5  0.13 -0.56 -0.63
  ## 6  1.72  1.79 -1.69
  ```

  Alternatively, use `read.csv()` specifically for comma-separated files:

  ```
  > mydata_from_csv <- read.csv("http://www.jkarreth.net/files/mydata.csv")
  > head(mydata_from_csv)


  ##       y    x1    x2
  ## 1 -0.56  1.22 -1.07
  ## 2 -0.23  0.36 -0.22
  ## 3  1.56  0.40 -1.03
  ## 4  0.07  0.11 -0.73
  ## 5  0.13 -0.56 -0.63
  ## 6  1.72  1.79 -1.69
  ```

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

  ```
  > mydata_from_spss <- import("http://www.jkarreth.net/files/mydata.sav")
  > head(mydata_from_spss)

  ##       y    x1    x2
  ## 1 -0.56  1.22 -1.07
  ## 2 -0.23  0.36 -0.22
  ## 3  1.56  0.40 -1.03
  ## 4  0.07  0.11 -0.73
  ## 5  0.13 -0.56 -0.63
  ## 6  1.72  1.79 -1.69
  ```

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

  ```
  > mydata_from_dta <- import("http://www.jkarreth.net/files/mydata.dta")
  > head(mydata_from_dta)

  ##       y    x1    x2
  ## 1 -0.56  1.22 -1.07
  ## 2 -0.23  0.36 -0.22
  ## 3  1.56  0.40 -1.03
  ## 4  0.07  0.11 -0.73
  ## 5  0.13 -0.56 -0.63
  ## 6  1.72  1.79 -1.69
  ```

  Alternatively, use `read.dta()` from the foreign package specifically for Stata files:

  ```
  > library("foreign")
  > mydata_from_dta <- read.dta("http://www.jkarreth.net/files/mydata.dta")
  > head(mydata_from_dta)

  ##       y    x1    x2
  ## 1 -0.56  1.22 -1.07
  ## 2 -0.23  0.36 -0.22
  ## 3  1.56  0.40 -1.03
  ## 4  0.07  0.11 -0.73
  ## 5  0.13 -0.56 -0.63
  ## 6  1.72  1.79 -1.69
  ```

Describing data

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

```
> summary(mydata_from_dta)

##        y                 x1                x2
##  Min.   :-1.2700   Min.   :-1.970   Min.   :-1.6900
##  1st Qu.:-0.5325   1st Qu.:-0.325   1st Qu.:-1.0600
##  Median :-0.0800   Median : 0.380   Median :-0.6800
##  Mean   : 0.0740   Mean   : 0.208   Mean   :-0.4270
##  3rd Qu.: 0.3775   3rd Qu.: 0.650   3rd Qu.: 0.0575
##  Max.   : 1.7200   Max.   : 1.790   Max.   : 1.2500

> summary(mydata_from_dta$y)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.2700 -0.5325 -0.0800  0.0740  0.3775  1.7200
```

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_from_dta$y)

## [1] 0.9561869

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

##     5%    95%
## -1.009  1.648
```

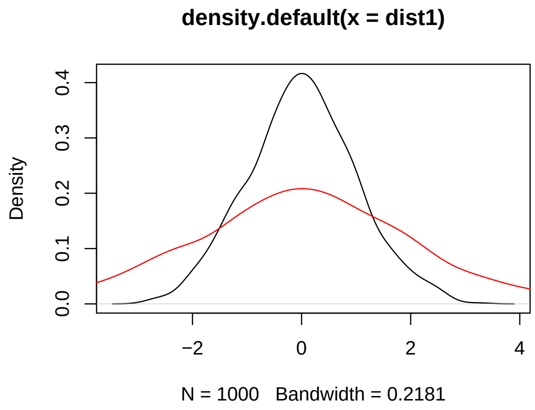
4 Creating figures

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")
```

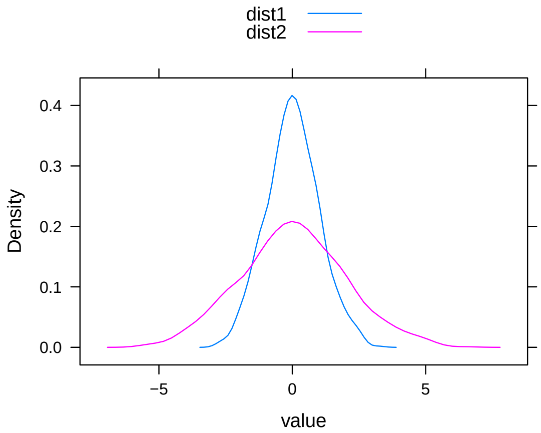


4.2 The lattice package

The lattice package has long been popular for visualizing more complex data structures, e.g. nested data. For plotting Bayesian model output, lattice offers some useful features.

lattice needs to be first loaded as an external package. It offers a variety of plots, some of them specifically built-in (`densityplot` or `dotplot`) and many other plots can be built with `xyplot`. 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. Here, I use the ggplot2 package’s `melt` command to reshape the data so they can be plotted in one figure. When trying the code below, have a look at the structure of the `dist.dat` object to see what’s going on.

```
> library("lattice")
> 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)
> densityplot(~value, data = dist.df, groups = variable, plot.points = FALSE, auto.key = TRUE)
```

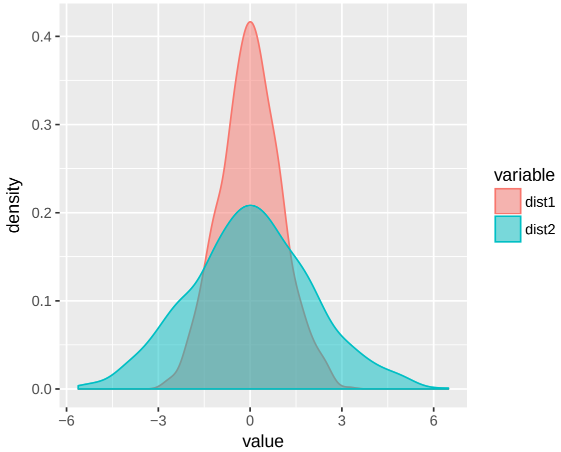


4.3 The ggplot2 package

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. I will mostly use ggplot2 in this workshop because (in my opinion) it offers a quick and scalable way to produce figures that are useful for diagnostics and publication-quality output alike.

ggplot2 needs to be first loaded as an external package. Its key commands are `ggplot()` and various types of plots, passed to R via `geom_...` 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. Here, I use the ggplot2 package's `melt` command to reshape the data so they can be plotted in one figure. 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`.

4.4 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,
+     unit = "in")
```

5 Integrating writing and data analysis

For project management and replication purposes, it is advantageous 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 and all slides in Weeks 3-4 are written using knitr. We will offer another lab that will address RMarkdown as a tool for reproducible research, among other topics.

6 Exercises

1. Create an R script that performs the following operations:
   - Sets an assignment-specific working directory, e.g. `Users/thomasbayes/Work/ICPSR/Homework/Lab1`
   - Calculates the mean and standard deviation of this series of numbers: $37, -6, -38, 12, -2$
2. Go to `http://gss.norc.org/` and download the General Social Survey raw data for 2014 in SPSS or Stata format. Save this file in an assignment-specific working directory. Then, create an R script that performs the following operations:
   - Reads the GSS data into R
   - Shows the number of respondents in this survey file
   - Creates a histogram of the distribution of age of all respondents in the survey
3. Create an R script that creates a fake data set representing 1500 observations. The data set should contain three variables:
   - an index variable that identifies each respondent, ranging from 1 to 1500
   - a binary variable that represents (fake) information on whether a respondent voted in the 2012 U.S. presidential election (1 if yes, 0 if not). Create this (fake) variable assuming that the respondents are randomly drawn from the U.S. population and that the turnout rate in this (fake) sample is more or less representative of the actual turnout rate in the 2012 presidential election.
   - a categorical variable for one of the four U.S. census regions (Northeast, South, Midwest, West). Assume that respondents are randomly drawn from these regions at equal probability.

   Save this fake dataset as a .csv file to your working directory. Then provide summary statistics for it.