2
Getting Started with R

In this chapter, we lay out the case for using R for social media mining. We then walk readers through the processes of installing, getting help for, and using R. By the end of this chapter, readers will have gained familiarity with data import/export, arithmetic, vectors, basic statistical modeling, and basic graphing using R.

Why R?

We strongly prefer using the R statistical computing environment for social data mining. This chapter highlights the benefits of using R, presents an introductory lesson on its use, and provides pointers towards further resources for learning the R language.

At its most basic, R is simply a calculator. You can ask it what 2 + 2 is, and it will provide you with 4 as the answer. However, R is more flexible than the calculator you used in high school. In fact, its flexibility leads it to be described as a statistical computing environment. As such, it comes with functions that assist us with data manipulation, statistics, and graphing. R can also store, handle, and perform complex mathematical operations on data as well as utilize a suite of statistics-specific functions, such as drawing samples from common probability distributions. Most simply, R is a data analysis software adoringly promoted as being made by statisticians for statisticians. The R programming language is used by data scientists, statisticians, formal scientists, physical scientists, social scientists, and others who need to make sense of data for statistical analysis, data visualization, and predictive modeling. Fortunately, with the brief guidance provided by this chapter, you too will be using R for your own research. R is simple to learn, even for people with no programming or statistics experience.

R is a **GNU** (**GNU's Not Unix**), where GNU's Not Unix is a recursive acronym for GNU and is less commonly referred to as GNU S. R is freely available under the GNU General Public License, and precompiled binary versions are provided for most common operating systems. R uses a command-line interface; however, several integrated development environments are available for use with R, including our preferred one, RStudio.

The following nine important questions ought to drive whether to use R or some other statistical language:

- Does the software run natively on your computer?
  - R compiles and runs on a variety of Unix platforms as well as on Windows and Mac OS.
- Does the software provide the methods needed?
  - R comes with a moderate compliment of built-in functions and is wildly extensible through user-generated packages from a variety of disciplines.
- If not, how extensible is the software, if at all?
  - R is extremely extensible and extending it is simple. Packages are provided by a robust academic and practitioner community and are available for inclusion through simple downloads.
- Does the software fully support programming versus point-and-click?
  - Users can utilize R as an interactive programming language or a scripting language. There are also packages, such as Rcmdr, that allow limited point-and-click functionality.
- Are the visualization options adequate for your needs?
  - R has a very powerful, simple-to-use suite of graphical capabilities. Additionally, these capabilities are extensible just like R's other capabilities.
- Does the software provide output in the form you prefer?
  - R can output data files in many formats and can produce graphics in a wide range of formats as well.
- Does the software handle large datasets?
  - R handles data in memory; thus, users are constrained by the memory of their local system. However, within that constraint, R can handle vectors of up to 2 gigabytes in length. Packages can extend R to work in cloud computing environments.

- Can you afford the software?
    - R is free, as in free beer.

R is an open source software, which means that members of the public invented it and they now maintain and distribute it, as opposed to a corporation or other private entity. Mainstream reasons to use open source software have historically hinged on the free aspect, that is, free as in free beer. In the past, open source projects have often been plagued with serious drawbacks such as having limited functionality, being buggy, not staying up-to-date, and being difficult to get help with. However, open source projects such as R attract a large community of developers and users to overcome these issues. Furthermore, R has an expansive (and expanding) functionality and is constantly updated; thanks to the large number of people using and developing it, help is nearly always just a Google away. The open source nature of R makes it free, as in free beer, and also free, as in freedom from vendor lock-in, which is what Richard Stallman advocates as the best reason for moving to open source projects. As Mozilla's Firefox browser has commandingly demonstrated, open source software can be excellent and approachable as opposed to being aimed at niche users.

The excellence of R has several consequences, each of which in turn cause R to become better. First and foremost, R is extensible. Individuals can contribute add-on components called packages to R, which execute algorithms, create graphics, or perform other tasks. The number of these packages has grown exponentially over time; as of early 2014, there were over 5,000. Furthermore, many of these packages are multiplicatively useful when combined, making them more valuable as a whole than the sum of their individual utilities.

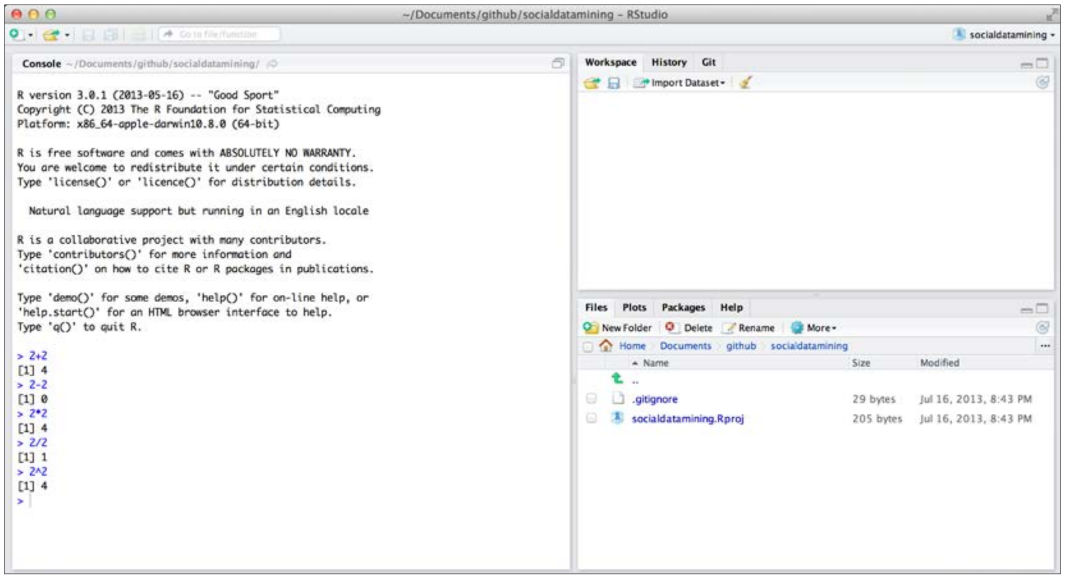
Secondly, R has a large and growing community of users and contributors, largely due to its excellence and broad utility. R has proven useful to so many that the traffic flow about it on e-mail discussion forums now outstrips the traffic on all of its main commercial contemporaries such as Stata, SAS, and SPSS. Similarly, the traffic related to R on Stack Overflow (`http://stackoverflow.com`), a software help forum, has outstripped SAS as well as some generic computing languages, such as PERL. Perhaps what's most telling is the fact that, at the time of writing this book (early 2014), more than half of the users on Kaggle (`http://www.r-bloggers.com/how-kaggle-competitors-use-r/`)—a site that promotes high-end data analysis competitions—use R.

R's popularity is indicative of its quality and broad utility. Additionally, the large number of active users make it much easier to get help with R through forums such as Stack Overflow and others (if R's built-in help documentation doesn't already answer your questions). Additionally, there are many books currently available in print that walk users through how to perform intermediate and advanced general programming in R as well as demonstrate R's use for particular domains (such as this one).

The justification for using R is overwhelming. We find R to have an excellent combination of freedom (both kinds), flexibility, and power. In addition, R has growing capabilities in handling Big Data in distributed systems or in parallel; some examples include **Distributed Storage and List** (**dsl**), **HadoopInteractiVE** (**hive**), **Text Mining Distributed Corpus Plug-In** (**tm.plug.dc**), **Hadoop Steaming** (**HadoopSteaming**), and **Amazon Web Services** (**AWS.tools**). So, let's get started.

Quick start

To install R, simply navigate to `http://www.r-project.org` and choose a mirror near you. Then, select whether you want R for Linux, Windows, or Mac. Finally, just follow the instructions from there, and you'll be up in no time. For additional FAQs, refer to `http://cran.r-project.org/faqs.html`.



```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (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.

> 2+2
[1] 4
> 2-2
[1] 0
> 2*2
[1] 4
> 2/2
[1] 1
> 2^2
[1] 4
> 
```

```
..
.gitignore                  29 bytes    Jul 16, 2013, 8:43 PM
socialdatamining.Rproj      205 bytes   Jul 16, 2013, 8:43 PM
```

In addition to installing R, you will almost certainly want to install an **integrated development environment** (**IDE**). An IDE is a programming environment that offers features beyond what is found by using the terminal or a command-line environment. These features can include code editing/highlighting/completion/generation, code compiling, code debugging, a file manager, a package manager, and a **graphical user interface** (**GUI**). These features will make generating and managing your R code simpler. There are a plethora of options, but we have a slight preference for RStudio, which is shown in the previous screenshot. It is recommended that you install RStudio (`http://www.rstudio.com`) before working through the examples discussed in this book. As we move forward, note that all of the following R code will be available online on the authors' web page and GitHub.

The basics – assignment and arithmetic

R gives you access to the central math operators through the characters discussed later. R respects the order of operations; as such, the list discussed in the next section is in that order.

The carrot (>) symbol denotes lines of code being inputted to R, while lines without (>) denote the output from R. In some circumstances, R will number its output; we'll point that out as it arises. Finally, R does not read code following a pound sign (#), which allows users to write comments for themselves and others right in the code itself:

```
>2^4-3
13
>2^(4-3)
2
```

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Functions, arguments, and help

R has many built-in functions. A function is a programming construct that takes input, calls arguments, and turns them into output. Some functions only take one argument. An example of this is as follows:

```
>sqrt(16)
4
```

Other functions take several arguments. Generally, R can use the order in which you provide the arguments to understand the arguments respectively; however, it is a good practice to explicitly label your arguments, which are generally separated by commas. R does not read or care about spaces, but it is a good practice to include spaces between operators and after commas for better readability, as follows:

```
# Take the log of 100 with base 10
>log(100, 10)
2
```

```
# Though not necessary, it is best practice to label arguments
# This avoids confusion when functions take many arguments
>log(100, base=10)
2
```

To get help with a function, you can use the help function or type a question mark before a term. Using double question marks broadens the search, as shown in the following code:

```
>help(log)
> ?log
> ??log
```

Assignment is an important concept in R. We can assign values to an object, then treat that object as if it were the value it stores. An example should make this much more clear. Note the use of the left-facing arrow (<-) for assignment. Although you can assign with a right arrow or a single equals sign, only using the left arrow helps avoid confusion. An example of the assignment concept is shown as follows:

```
# Assign the value 3 to the object called 'my.variable'
> my.variable<- 3
# Work with the object
>my.variable * 2
[1] 6
# Create a new variable
>other.object<- my.variable + 7
>other.object * 2
[1] 20
```

R utilizes logical operators in addition to arithmetic operators. Logical operators are those that compare entities and return values as either `TRUE` or `FALSE`. Note that the double equals sign is used to ask a question, while the single equals sign is used for assignment (though the arrow should be strongly preferred for assignment to avoid confusion):

```
> 2==3
[1] FALSE
> 4 >= 4
[1] TRUE
"hello" != "HELLO"
[1] TRUE
```

Vectors, sequences, and combining vectors

Many R operations can be performed, or performed more efficiently, on vectors or matrices. Vectors are strings of objects; matrices are two-dimensional collections of objects, usually numbers. The `c` operator, which means concatenate, creates simple vectors, while the colon (`:`) operator generates simple sequences. To construct matrices, one simply passes a vector of data, the dimensions of the matrix to be created, and whether to input the data by row or by column (the default behavior is to input data by row). Examples of vectors and sequences are given as follows:

```
>c(1,2,3,4,5)
1 2 3 4 5

> 1:4
1 2 3 4

> 5:-1
5 4 3 2 1 0 -1

>matrix(data=c(1, 2, 3, 4), byrow=TRUE, nrow=2)
1 2
3 4
```

For more complex sequence-like vectors, you can use the `seq()` function. At a minimum, it takes two arguments: `from` and `to`. You can additionally specify a `by` argument as well:

```
>seq(from=1, to=5)
1 2 3 4 5

>seq(from=2, to=6, by=2)
2 4 6
```

R also contains several constructs that allow access to individual elements or subsets through indexing operations. In the case of basic vector types, one can access the *i* th element by using `x[i]`, but there is also indexing of lists (which are simply collections of other data types), matrices, and multidimensional arrays (that is, matrices with more than two dimensions). In addition, R has a data type called a data frame, which is what many readers familiar with Stata, SPSS, or Microsoft Excel would think of as a dataset or spreadsheet. Data frames have columns and possibly row names as well. R has three basic indexing operators with syntax, which is displayed in the following examples:

```
x[i]    # read the i-th element of a vector
x[i, j] # read i-th row, j-th column element of a matrix
x[[i]]  # read the i-th element of a list
x$a     # read the variable named "a" in a data frame named x
```

For lists, one generally uses `[[` to select any single element, whereas `[` returns a list of the selected elements. Many operators can work over vectors, as shown in the following code:

```
# divides each number in vector by 2
>c(1,2,3,4,5) / 2
0.5 1.0 1.5 2.0 2.5

# first vector divided by second
> c(1,2,3,4,5) / c(5,4,3,2,1)
0.2 0.5 1.0 2.0 5.0

# log base 10 of vector
>log(c(1,2.5,5), base=10)
0.00000 0.39794 0.69897

# new variable x is assigned resultant set
> x <- c(1,2,3,4,5) / 2
>x
0.5 1.0 1.5 2.0 2.5

# generic function 'summary' on variable x
>summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.5     1.0     1.5     1.5     2.0     2.5

# function to find mean
# notice mean is also captured by the generic function 'summary'
>mean(x)
1.5
```

A quick example – creating data frames and importing files

Getting data into R is often the first step in an analysis. R has a suite of functions called `read`, such as `read.csv()`, to help import data. Here, we assign the values read from a CSV file to an object called `mydata` as shown in the following code:

```
>mydata<- read.csv("http://www.ats.ucla.edu/stat/data/binary.csv")

# returns the first few rows of the data
>head(mydata)
```

```
  admitgre  gpa rank
1     0 380 3.61    3
2     1 660 3.67    3
3     1 800 4.00    1
```

To the initial confusion of some, several R functions behave differently depending on the type of object on which they act. As we saw earlier, the `summary()` function outputs descriptive statistics when it is given a vector. When given a data frame, it outputs summary statistics for each variable, as shown in the following code:

```
>summary(mydata)
     admit             gre             gpa
Min.   :0.0000   Min.   :220.0   Min.   :2.260
 1st Qu.:0.0000   1st Qu.:520.0   1st Qu.:3.130
Median :0.0000   Median :580.0   Median :3.395
 Mean   :0.3175   Mean   :587.7   Mean   :3.390
 3rd Qu.:1.0000   3rd Qu.:660.0   3rd Qu.:3.670
 Max.   :1.0000   Max.   :800.0   Max.   :4.000
```

R has many built-in functions for fitting statistical models. For example, we can estimate a linear regression model, that is, a model that predicts the level of a continuous variable with another continuous variable(s), by **ordinary least squares** (**OLS**) with the first two lines of the next code. Note that the tilde (~) in the following code is used to separate the left-hand side of the equation from the right-hand side of the equation. In this simple regression example, we are regressing y on x, or gre (`mydata$gre`) on gpa (`mydata$gpa`). When the `summary` command is used with a model as the argument, parameter estimates are displayed along with other auxiliary information. Finally, we present the regression example as a demonstration of a classical method in social science used on structured data. This book departs from these classical methods and structured data:

```
>mydata.model<- lm(mydata$gre~mydata$gpa)
>summary(mydata.model)

Call:
lm(formula = mydata$gre ~ mydata$gpa)

Residuals:
    Min       1Q   Median       3Q      Max
-302.394  -62.789   -2.206   68.506  283.438

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   192.30      47.92   4.013 7.15e-05 ***
mydata$gpa    116.64      14.05   8.304 1.60e-15 ***
---
Signif.codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
```

One of R's great features is its extensibility. For instance, the `foreign` package allows users to import data formats other than those that R supports natively. To install a package, simply enter the following command in the terminal:

```
>install.packages("foreign", dependencies=TRUE)
```

The first argument to the function is the package name, and the second argument tells R to additionally install any other packages on which the one being installed is dependent. You will be asked to pick a mirror, that is, a location to download from. Choose one (it doesn't really matter which) and then input the following command to load the package:

```
>library("foreign")
```

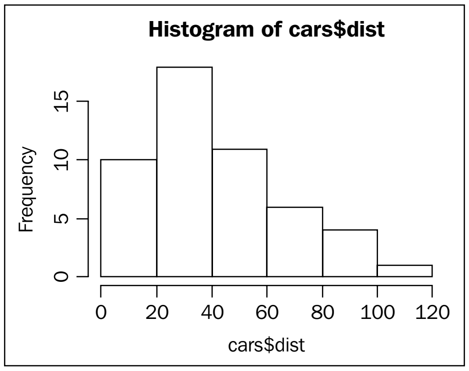
To see all the different uses for this package, type `?foreign` as a command. One package that is particularly useful is the `sos` package, which allows you to search for other packages using colloquial search terms with the `findFn()` function. For example, when searching for a package that does non-linear regression, one could use the following command:

```
>library("sos")
findFn("non-linear regression")
```

Visualization in R

Visualization is a powerful tool for analyzing data and for presenting results. Many relationships and patterns that are obscured by summary statistics can be brought to light through visualization. The next graph shows a potent example of this. To begin with, let's look at some data that R comes with on the stopping distance of cars. This variable is contained in a dataset called cars, in a variable called **dist**. Histograms provide an informative way to visualize single variables. We can make a histogram with one line of code:

```
>hist(cars$dist)
```

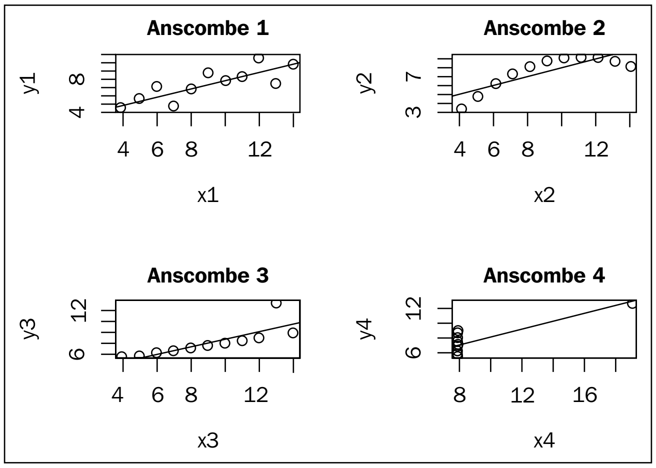


R makes the histogram, decides how to break up the data, and provides default labels for the graph title and the x and y axes. Type the `?hist()` command to see other arguments to this function that change the number of bars, the labels, and other features of the histogram.

Anscombe's quartet comprises four small datasets with two variables each. Each of the sets has similar mean and variance for both variables, and regressions of y on x in each dataset generate nearly identical regression estimates. Overall, we might be tempted to infer that these datasets are nearly identical. However, bivariate visualization (of the x and y axes from each dataset) using the generic `plot()` function shows otherwise. At a minimum, the `plot()` function takes two arguments, each as a vector of the same length. To create the following four plots, enter the following commands for each pair of x and y (x1 and y1, x2 and y2, x3 and y3, and finally x4 and y4):

```
# par can be used to set or query graphical parameters.
# subsequent figures will be drawn in a n-row-by-n-column array (e.g.
2,2)
#par(mfrow=c(2,2))
>plot(anscombe$x1, anscombe$y1, xlab="x1", ylab="y1",
  main="Anscombe 1")
>abline(lm(anscombe$y1~anscombe$x1)
```

The code discussed earlier shows how to create your own x and y axis labels and plot titles. The `abline` call adds straight lines to an existing plot, in this case, the best fit (or regression) line of y on x. The output of the previous code is shown as follows:



This exercise not only demonstrates some simple graphical commands, but also the importance of visualization generally. In later chapters, we elaborate on these graphical methods to enhance analyses and the presentation of analytical results.

Style and workflow

Statistical programmers can think of R code—like other languages—as being dysfunctional, functional but awkward, or graceful. Graceful code is clear and readable, which helps prevent errors. Here are a few tips on writing graceful R code:

- Filenames should end in `.R` and be meaningful.
- Variable names should be short. If necessary, use a period to delineate multiword variable names (for example, `my.variable`).
- Keep every line short. R will not terminate a command until all parentheses are closed, so feel free to wrap commands across lines for brevity.
- Use spaces before and after the assignment, after commas, and around parentheses and operators (such as +) for readability.
- Use the left arrow (<-) for assignment, never the single equals sign.

For more details on writing good R code, refer to the guide at `http://google-styleguide.googlecode.com/svn/trunk/Rguide.xml`. Again, though R can be used interactively from within the terminal, it is best practice to develop code within an IDE, such as RStudio, so that it can be saved, changed, and rerun. Additionally, building version controls and persistence into your code by storing it on GitHub may be important, especially if you find yourself working in a group environment. Finally, many users will find the creation of projects useful—the RStudio documentation offers useful tips on this topic.

Additional resources

This chapter provides what we hope is a useful, though necessarily brief, introduction to R. There are many resources available that will help you expand your R programming skill set. What follows is a short list of our favorites:

- *A First Course in Statistical Programming with R* by *Braun and Murdoch* (2007)
- *The R Cookbook* by *Teetor* (2011)
    - Quick-R: `http://www.statmethods.net/`

Summary

This chapter has set out a case for using the R mathematical computing environment for data handling and analysis due to R's zero cost feature, flexibility, and large support community. By now, you've seen how to import, summarize, and visualize datasets as well as run and plot bivariate regression models. In the next chapter, we discuss some ways of obtaining data from social media sources, such as Twitter.