Web Application Development with R Using Shiny
Second Edition

R is a highly flexible and powerful tool for analyzing and visualizing data. Most of the applications built using various libraries with R are desktop-based. But what if you want to go on the Web? Here comes Shiny to your rescue!

Shiny allows you to create interactive web applications using the excellent analytical and graphical capabilities of R. This book will guide you through basic data management and analysis with R through your first Shiny application, and then show you how to integrate Shiny applications with your own web pages. Finally, you will learn how to finely control the inputs and outputs of your application, along with using other packages to build state-of-the-art applications, including dashboards.

Who this book is written for
This book is for anybody who wants to produce interactive data summaries over the web, whether you want to share them with a few colleagues or the whole world. No previous experience with R, Shiny, HTML, or CSS is required to begin using this book, although you should possess some previous experience with programming in a different language.

What you will learn from this book
- Build interactive applications using Shiny’s built-in widgets
- Use the built-in layout functions in Shiny to produce user-friendly applications
- Integrate Shiny applications with web pages and customize them using HTML and CSS
- Harness the power of JavaScript and jQuery to customize your applications
- Engage your users and build better analytics using interactive plots
- Debug your applications using Shiny’s built-in functions
- Deliver simple and powerful analytics across your organization using Shiny dashboards
- Share your applications with colleagues or over the Internet using cloud services or your own server

Integrate the power of R with the simplicity of Shiny to deliver cutting-edge analytics over the Web
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Getting Started with R and Shiny!'
- A synopsis of the book’s content
- More information on Web Application Development with R using Shiny - Second Edition
About the Author

Chris Beeley works for Nottinghamshire Healthcare NHS Trust as the lead analyst and programmer for staff and patient experience. He uses a variety of open source tools (PHP/MySQL, Apache, R, Shiny, and Ubuntu) to collect, collate, analyze, and report on patient and staff experience throughout the organization. He was the author of the previous edition of this book.

He has been a keen user of R and a passionate advocate of open source tools in research and healthcare settings, having completed his PhD. He has made extensive use of R (and Shiny) to automate analysis and report on a new patient feedback website. This was funded by a grant from the NHS Institute for Innovation and made in collaboration with staff, service users, and carers within the Trust, particularly individuals from the Involvement Centre.
Preface

Harness the graphical and statistical power of R, and rapidly develop interactive and engaging user interfaces using the superb Shiny package, which makes programming for user interaction simple. R is a highly flexible and powerful tool used for analyzing and visualizing data. Shiny is the perfect companion to R, making it quick and simple to share analysis and graphics from R that users can interact with and query over the Web. Let Shiny do the hard work and spend your time generating content and styling, not writing code to handle user inputs. This book is full of practical examples and shows you how to write cutting-edge interactive content for the Web, right from a minimal example all the way to fully styled and extensible applications.

This book includes an introduction to Shiny and R and takes you all the way to advanced functions in Shiny as well as using Shiny in conjunction with HTML, CSS, and JavaScript to produce attractive and highly interactive applications quickly and easily. It also includes a detailed look at other packages available for R, which can be used in conjunction with Shiny to produce dashboards, maps, advanced D3 graphics, among many things.

What this book covers

Chapter 1, Getting Started with R and Shiny!, runs through the basics of statistical graphics, data input, and analysis with R. We also discuss data structures and programming basics in R in order to give you a thorough grounding in R before we look at Shiny.

Chapter 2, Building Your First Application, helps you build your first Shiny application. We begin with simply adding interactive content to a document written in markdown, and then delve deeper into Shiny, building a very primitive minimal example, and finally, looking at more complex applications and the inputs and outputs necessary to build them.
Preface

Chapter 3, Building Your Own Web Pages with Shiny, covers how Shiny works with existing web content in HTML, CSS, and JavaScript. We discuss the Shiny helper functions that allow you to add a custom HTML to a standard Shiny application and how to build a minimal example of a Shiny application in your own raw HTML with Shiny running in the background. Finally, we also discuss using JavaScript/jQuery with Shiny with examples given to add bells and whistles to an existing application as well as providing powerful interactive tools to communicate between the web page and Shiny using JavaScript.

Chapter 4, Taking Control of Reactivity, Inputs, and Outputs, covers advanced functions in Shiny in detail, in particular, changing the UI based on user input or the state of the application, finely controlling reactivity in your application, and advanced methods used for reading user input as well as specialized graphics and data tables. We also cover debugging, which can pose challenges in Shiny applications.

Chapter 5, Advanced Applications I – Dashboards, contains detailed information of the layout in Shiny applications. We discuss simple ways to use layout functions described earlier in the book, and how to use the Bootstrap style on which Shiny is based. Finally, we also cover how a full dashboard is produced with several pages, specialized input and output widgets, and other advanced features accessible when using Shiny dashboards.

Chapter 6, Advanced Applications II – Using JavaScript Libraries in Shiny Applications, reviews some of the many JavaScript libraries, which can easily be integrated into Shiny, and how to use them in your own Shiny applications. We also cover how to draw graphics, which describe trends and predictions, heatmaps and highly interactive charts using D3, and 3D plots, along with an advice on how best to ensure that they work within Shiny.

Chapter 7, Sharing Your Creations, discusses the many different ways to share Shiny applications with your end users. There are many ways of doing this and they are described in detail, including the use of the Gist and GitHub website, locally using a simple ZIP file, hosting them yourself on your own server, or making use of RStudio's hosting services. We also cover reading and writing data using Shiny in a server (as opposed to a local) environment.
R is free and open source as well as being the pre-eminent tool for statisticians and data scientists. It has more than 6000 user-contributed packages, which help users with tasks as diverse as chemistry, biology, physics, finance, psychology, and medical science, as well as drawing extremely powerful and flexible statistical graphics.

In recent years, R has become more and more popular, and there are an increasing number of packages for R, which make cleaning, analyzing, and presenting data on the web easy for everybody. The Shiny package, in particular, makes it incredibly easy to deliver interactive data summaries and queries to end users through any modern web browser. You're reading this book because you want to use these powerful and flexible tools for your own content.

This book will show you how, right from starting with R, to build your own interfaces with Shiny and integrate them with your own websites. In this chapter, we're going to cover the following:

- Download and install R and choose a code editing environment/IDE
- Look at the power of R and learn about how RStudio and contributed packages can make writing code, managing projects, and working with data easier
- Install Shiny and run the examples
- Take a look at some awesome Shiny applications and some of the elements of the Shiny application we will build over the course of this book
R is a big subject, and this is a whistle-stop tour; so if you get a little lost along the way, don't worry. This chapter is really all about showing you what's out there and encouraging you to delve deeper into the bits that interest you and showing you places you can go for help if you want to learn more on a particular subject.

## Installing R

R is available for Windows, Mac OS X, and Linux at cran.r-project.org. Source code is also available at the same address. It is also included in many Linux package management systems; Linux users are advised to check before downloading from the web. Details on installing from source or binary for Windows, Mac OS X, and Linux are all available at cran.r-project.org/doc/manuals/R-admin.html.

## The R console

Windows and Mac OS X users can run the R application to launch the R console. Linux and Mac OS X users can also run the R console straight from the terminal by typing `R`.

In either case, the R console itself will look something like this:

```
R version 3.2.0 (2015-04-16) -- "Full of Ingredients"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (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.

> 
```
R will respond to your commands right from the terminal. Let's have a go:

```r
> 2 + 2
[1] 4
```

The [1] tells you that R returned one result, in this case, 4. The following command shows how to print Hello world:

```r
> print("Hello world!")
[1] "Hello world!"
```

The following command shows the multiples of pi:

```r
> 1:10 * pi
```

This example illustrates vector-based programming in R. `1:10` generates the numbers `1:10` as a vector, and each is then multiplied by `pi`, which returns another vector, the elements each being `pi` times larger than the original. Operating on vectors is an important part of writing simple and efficient R code. As you can see, R again numbers the values it returns at the console with the seventh value being 21.99.

One of the big strengths of using R is the graphics capability, which is excellent even in a vanilla installation of R (these graphics are referred to as base graphics because they ship with R). When adding packages such as ggplot2 and some of the JavaScript-based packages, R becomes a graphical tour de force, whether producing statistical, mathematical, or topographical figures, or indeed many other types of graphical output. To get a flavor of the power of base graphics, simply type the following at the console:

```r
> demo(graphics)
```

You can also type the following command:

```r
> demo(persp)
```

There is more on ggplot2 and base graphics later in the chapter and a brief introduction to JavaScript and D3-based packages for R in Chapter 6, Advanced Applications II–Using JavaScript Libraries in Shiny Applications.

Enjoy! There are many more examples of R graphics at gallery.r-enthusiasts.com/.
Code editors and IDEs

The Windows and OSX versions of R both come with built-in code editors, which allow code to be edited, saved, and sent to the R console. It's hard to recommend that you use this because it is rather primitive. Most users would be best served by RStudio (rstudio.com/), which includes project management and version control (including support for Git, which is covered in Chapter 7, Sharing Your Creations), viewing of data and graphics, code completion, package management, and many other features. The following is an illustrative screenshot of an RStudio session:

As can be seen, in the top-left corner, there is the code editing pane (with syntax highlighting). The clockwise move from there will take you to the environment pane (in which the different objects loaded into the session can be seen); the viewing pane containing various options such as Files, Plots, Build, Help, and the console (again, with syntax highlighting). In the middle, there is one of the most useful features of RStudio—the ability to view dataframes. This function also comes with sorting and filtering by column.
However, if you already use an IDE for other types of code, it is quite likely that R can be well integrated into it. Examples of IDEs with good R integration include the following:

- Emacs with the Emacs Speaks Statistics plugin
- Vim with the Vim-R plugin
- Eclipse with the StatET plugin

### Learning R

There are almost as many uses for R as there are people using it. It is not possible to cover your specific needs within this book. However, it is likely that you wish to use R to process, query, and visualize data such as sales figures, satisfaction surveys, concurrent users, sporting results, or whatever types of data your organization processes. Later chapters will concentrate on Google Analytics data downloaded from the API, but for now, let's just take a look at the basics.

### Getting help

There are many books and online materials that cover all aspects of R. The name R can make it difficult to come up with useful web search hits (substituting CRAN for R can sometimes help); nonetheless, searching for R tutorial brings back useful results. Some useful resources include the following:

- Excellent introduction to syntax and data structures in R (goo.gl/M0RQ5z)
- Videos on using R from Google (goo.gl/A3uRah)
- Swirl (swirlstats.com)
- Quick-R (statmethods.net)

At the R console, ?functionname (for example, ?help) brings up help materials and use of ??help will bring up a list of potentially relevant functions from installed packages.

Subscribing to and asking questions on the R-help mailing list at r-project.org/mail.html allows you to communicate with some of the leading figures in the R community as well as many other talented enthusiasts. Do read the posting guide and research your question before you ask any questions because it's a busy and sometimes unforgiving list.
Getting Started with R and Shiny!

There are two Stack Exchange communities, which can provide further help at [stats.stackexchange.com/](http://stats.stackexchange.com/) (for questions about statistics and visualization with R) and [stackoverflow.com/](http://stackoverflow.com/) (for questions about programming with R).

There are many ways to learn R and related subjects online; RStudio has a very useful list on their website available at [goo.gl/8tX7FP](http://goo.gl/8tX7FP).

### Loading data

The simplest way of loading data into R is probably using a comma-separated value (.csv) spreadsheet file, which can be downloaded from many data sources and loaded and saved in all spreadsheet software (such as Excel or LibreOffice). The `read.table()` command imports data of this type by specifying the separator as a comma, or there is a function specifically for .csv files, `read.csv()`, as shown in the following command:

```
> analyticsData <- read.table("~/example.csv", sep = ",")
```

Otherwise, you can use the following command:

```
> analyticsData <- read.csv("~/example.csv")
```

Note that unlike in other languages, R uses `<-` for assignment as well as `=`. Assignment can be made the other way using `->`. The result of this is that `y` can be told to hold the value of 4 like this, `y <- 4`, or like this, `4 -> y`. There are some other more advanced things that can be done with assignment in R, but don't worry about them now. Just write code using the assignment operator in the preceding example and you'll be just like the natives that you will come across on forums and blog posts.

Either of the preceding code examples will assign the contents of the Analytics.csv file to a dataframe named `analyticsData`, with the first row of the spreadsheet providing the variable names. A dataframe is a special type of object in R, which is designed to be useful for the storage and analysis of data.

### Data types and structures

There are many data types and structures of data within R. The following topics summarize some of the main types and structures that you will use when building Shiny applications.
Dataframes, lists, arrays, and matrices

Dataframes have several important features, which make them useful for data analysis:

- Rectangular data structures with the typical use being cases (for example, days in one month) down the rows and variables (page views, unique visitors, or referrers) along the columns.
- A mix of data types is supported. A typical dataframe might include variables containing dates, numbers (integers or floats), and text.
- With subsetting and variable extraction, R provides a lot of built-in functionality to select rows and variables within a dataframe.
- Many functions include a data argument, which makes it very simple to pass dataframes into functions and process only the variables and cases that are relevant, which makes for cleaner and simpler code.

We can inspect the first few rows of the dataframe using the `head(analyticsData)` command. The following screenshot shows the output of this command:

As you can see, there are four variables within the dataframe—one contains dates, two contain integer variables, and one contains a numeric variable. There is more about variable types in R shown in the following paragraphs.

Variables can be extracted from dataframes very simply using the $ operator as follows:

```r
> analyticsData$pageViews
[1] 836 676 940 689 647 899 934 718 776 570 651 816
[13] 731 604 627 946 634 990 994 599 657 642 894 983
[25] 646 540 756 989 965 821
```

Also, variables can be extracted from dataframes using [], as shown in the following command:

```r
> analyticsData[, "pageViews"]
```
Note the use of the comma with nothing before it to indicate that all rows are required. In general, dataframes can be accessed using `dataObject[x,y]` with `x` being the number(s) or name(s) of the rows required and `y` being the number(s) or name(s) of the columns required. For example, if the first 10 rows were required from the `pageViews` column, it could be achieved like this:

```r
> analyticsData[1:10,"pageViews"]
[1] 836 676 940 689 647 899 934 718 776 570
```

Leaving the space before the comma blank returns all rows, and the space after the comma blank returns all variables. For example, the following command returns the first three rows of all variables:

```r
> analyticsData[1:3,]
```

The following screenshot shows the output of this command:

![Dataframe output](image)

Dataframes are a special type of list. Lists can hold many different types of data including lists. As with many data types in R, their elements can be named, which can be useful to write code that is easy to understand. Let's make a list of the options for dinner, with drink quantities expressed in milliliters.

In the following example, please note also the use of the `c()` function, which is used to produce vectors and lists by giving their elements separated by commas. R will pick an appropriate class for the return value, string for vectors that contain strings, numeric for those that only contain numbers, logical for Boolean values, and so on:

```r
> dinnerList <- list("Vegetables" =
    c("Potatoes", "Cabbage", "Carrots"),
    "Dessert" = c("Ice cream", "Apple pie"),
    "Drinks" = c(250, 330, 500)
)
```

Note that code is indented throughout, although entering directly into the console will not produce indentations; it is done for readability.
Indexing is similar to dataframes (which are, after all, just a special instance of a list). They can be indexed by number, as shown in the following command:

```r
> dinnerList[1:2]
$Vegetables
 [1] "Potatoes"  "Cabbage"  "Carrots"

$Dessert
 [1] "Ice cream"  "Apple pie"
```

This returns a list. Returning an object of the appropriate class is achieved using `[]`:

```r
> dinnerList[[3]]
[1] 250 330 500
```

In this case a numeric vector is returned. They can be indexed also by name:

```r
> dinnerList["Drinks"]
$Drinks
 [1] 250 330 500
```

Note that this, also, returns a list.

Matrices and arrays, which, unlike dataframes, only hold one type of data, also make use of square brackets for indexing, with `analyticsMatrix[, 3:6]` returning all rows of the third to sixth column, `analyticsMatrix[1, 3]` returning just the first row of the third column, and `analyticsArray[1, 2, ]` returning the first row of the second column across all of the elements within the third dimension.

**Variable types**

R is a dynamically typed language and you are not required to declare the type of your variables. It is worth knowing, of course, about the different types of variable that you might read or write using R. The different types of variable can be stored in a variety of structures, such as vectors, matrices, and dataframes, although some restrictions apply as detailed previously (for example, matrices must contain only one variable type):

- Declaring a variable with at least one string in will produce a vector of strings (in R, the character data type):
  ```r
  > c("First", "Third", 4, "Second")
  [1] "First"  "Third"  "4"  "Second"
  ```
You will notice that the numeral 4 is converted to a string, "4". This is as a result of coercion, in which elements of a data structure are converted to other data types in order to fit within the types allowed within the data structure. Coercion occurs automatically, as in this case, or with an explicit call to the \texttt{as()} function, for example, \texttt{as.numeric()}, or \texttt{as.Date()}.

- Declaring a variable with just numbers will produce a numeric vector:
  
  \begin{verbatim}
  > c(15, 10, 20, 11, 0.4, -4)
  [1] 15.0 10.0 20.0 11.0  0.4 -4.0
  \end{verbatim}

- R includes, of course, also a logical data type:
  
  \begin{verbatim}
  > c(TRUE, FALSE, TRUE, TRUE, FALSE)
  [1] TRUE FALSE TRUE TRUE FALSE
  \end{verbatim}

- A data type exists for dates, often a source of problems for beginners:
  
  \begin{verbatim}
  > as.Date(c("2013/10/24", "2012/12/05", "2011/09/02"))
  [1] "2013-10-24" "2012-12-05" "2011-09-02"
  \end{verbatim}

- The use of the \texttt{factor} data type tells R all of the possible values of a categorical variable, such as gender or species:
  
  \begin{verbatim}
  > factor(c("Male", "Female", "Female", "Male", "Male"),
  >     levels = c("Female", "Male"))
  [1] Male Female Female Male Male
  Levels: Female Male
  \end{verbatim}

\section*{Functions}

As you grow in confidence with R you will wish to begin writing your own functions. This is achieved very simply and in a manner quite reminiscent of many other languages. You will no doubt wish to read more about writing functions in R in a fuller treatment, but just to give you an idea, here is a function called the \texttt{sumMultiply} function which adds together \(x\) and \(y\) and multiplies by \(z\):

\begin{verbatim}
sumMultiply <- function(x, y, z){
    final = (x+y) * z
    return(final)
}
\end{verbatim}

This function can now be called using \texttt{sumMultiply(2, 3, 6)}, which will return 2 plus 3 times 6, which gives 30.
Objects

There are many special object types within R which are designed to make it easier to analyze data. Functions in R can be polymorphic, that is to say they can respond to different data types in different ways in order to produce the output that the user desires. For example, the `plot()` function in R responds to a wide variety of data types and objects, including single dimension vectors (each value of y plotted sequentially) and two-dimensional matrices (producing a scatterplot), as well as specialized statistical objects such as regression models and time series data. In the latter case, plots specialized for these purposes are produced.

As with the rest of this introduction, don't worry if you haven't written functions before, or don't understand object concepts and aren't sure what this all means. You can produce great applications without understanding all these things, but as you do more and more with R you will start to want to learn more detail about how R works and how experts produce R code. This introduction is designed to give you a jumping off point to learn more about how to get the best out of R (and Shiny).

Base graphics and ggplot2

There are lots of user-contributed graphics packages in R that can produce some wonderful graphics. You may wish to take a look for yourself at the CRAN task view cran.r-project.org/web/views/Graphics.html. We will have a very quick look at two approaches: base graphics, so-called because it is the default graphical environment within a vanilla installation of R, and ggplot2, a highly popular user-contributed package produced by Hadley Wickham, which is a little trickier to master than base graphics, but can very rapidly produce a wide range of graphical data summaries. We will cover two graphs familiar to all, the bar chart and the line chart.

Bar chart

Useful when comparing quantities across categories, bar charts are very simple within base graphics, particularly when combined with the `table()` command. We will use the mpg dataset which comes with the ggplot2 package; it summarizes different characteristics of a range of cars. First, let's install the ggplot2 package. You can do this straight from the console:

```r
> install.packages("ggplot2")
```
Alternatively, you can use the built in package functions in IDEs like RStudio or RKWard. We'll need to load the package at the beginning of each session in which we wish to use this dataset or the ggplot2 package itself. From the console type the following command:

```r
> library(ggplot2)
```

We will use the `table()` command to count the number of each type of car featured in the dataset:

```r
> table(mpg$class)
```

This returns a table object (another special object type within R) that contains the following columns shown in the screenshot:

```
> table(mpg$class)

   2seater compact  midsize minivan pickup subcompact   suv
      5       47       41       11      33       35     62
```

Producing a bar chart of this object is achieved simply like this:

```r
> barplot(table(mpg$class), main = "Base graphics")
```

The `barplot` function takes a vector of frequencies. Where they are named, as here (the `table()` command returning named frequencies in table form), names are automatically included on the x axis. The defaults for this graph are rather plain; explore `?barplot` and `?par` to learn more about fine-tuning your graphics.

We've already loaded the ggplot2 package in order to use the `mpg` dataset, but if you have shut down R in between these two examples you will need to reload it by using the following command:

```r
> library(ggplot2)
```

The same graph is produced in ggplot2 as follows:

```r
> ggplot(data = mpg, aes(x = class)) + geom_bar() +
  ggtitle("ggplot2")
```
This ggplot call shows the three fundamental elements of ggplot calls—the use of a dataframe (data = mpg), the setup of aesthetics (aes(x = class)), which determines how variables are mapped onto axes, colors, and other visual features, and the use of geom_xxx(). A ggplot call sets up the data and aesthetics, but does not plot anything. Functions such as geom_bar() (there are many others, see ??geom) tell ggplot what type of graph to plot, as well as taking optional arguments, for example, geom_bar() optionally takes a position argument which defines whether the bars should be stacked, offset, or stretched to a common height to show proportions instead of frequencies.

These elements are the key to the power and flexibility that ggplot2 offers. Once the data structure is defined, ways of visualizing that data structure can be added and taken away easily, not only in terms of the type of graphic (bar, line, or scatter) but also the scales and co-ordinate system (log10; polar coordinates) and statistical transformations (smoothing data, summarizing over spatial co-ordinates). The appearance of plots can be easily changed with pre-set and user-defined themes, and multiple plots can be added in layers (that is, adding to one plot) or facets (that is, drawing multiple plots with one function call).

**Line chart**

Line charts are most often used to indicate change, particularly over time. This time we will use the longley dataset, featuring economic variables between 1947 and 1962:

```r
> plot(x = 1947 : 1962, y = longley$GNP, type = "l",
     xlab = "Year", main = "Base graphics")
```

The x axis is given very simply by 1947 : 1962, which enumerates all the numbers between 1947 and 1962, and the type = "l" argument specifies the plotting of lines, as opposed to points or both.

The ggplot call looks a lot like the bar chart except with an x and y dimension in the aesthetics this time. The command looks as follows:

```r
> ggplot(longley, aes(x = 1947 : 1962, y = GNP)) + geom_line() +
   xlab("Year") + ggtitle("ggplot2")
```
Base graphics and ggplot versions of the bar chart are shown in the following screenshot for the purposes of comparison:

Advanced tasks with dplyr and ggvis

Just to get a taste of how easy it can be to do some really cool things with R and to start to build the foundation of the Shiny application that we are going to build through the course of this book, let’s build a few graphics using some Google Analytics data and present them in an interactive document. We are going to use two contributed packages, dplyr and ggvis. The dplyr package provides very powerful functions for selecting, filtering, combining, and summarizing datasets. As you will see throughout this book, dplyr allows you to very rapidly process data to your exact specifications. The ggvis package provides very simple functions to make your visualizations interactive.
We’re going to run through some of the code very quickly indeed, so you can get a feeling for some of the tasks and structures involved, but we’ll return to this application later in the book where everything will be explained in detail. Just relax and enjoy the ride for now. If you want to browse or run all the code, it is available at chrisbeeley.net/website/index.html.

The Google Analytics code is not included because it requires a login for the Google Analytics API; instead, you can download the actual data from the previously mentioned link. Getting your own account for Google Analytics and downloading data from the API is covered in Chapter 5, Advanced Applications I – Dashboards. I am indebted to examples at goo.gl/rPFpF9 and at goo.gl/eL4Lrl for helpful examples of showing data on maps within R.

Preparation of the data

In order to prepare the data for plotting, we will make use of dplyr. As with all packages that are included on the CRAN repository of packages (cran.r-project.org/web/packages/), it can be installed using the package management functions in RStudio or other GUIs, or by typing install.packages("dplyr") at the console. It's worth noting that there are even more packages available elsewhere (for example, on GitHub), which can be compiled from the source.

The first job is to prepare the data that will demonstrate some of the power of the dplyr package using the following code:

```r
groupByDate =
filter(gadf, networkDomain %in% topThree$networkDomain) %>%
group_by(YearMonth, networkDomain) %>%
summarise(meanSession = mean(sessionDuration, na.rm = TRUE),
users = sum(users),
newUsers = sum(newUsers),
sessions = sum(sessions))
```

This single block of code, all executed in one line, produces a dataframe suitable for plotting and uses chaining to enhance the simplicity of the code. Three separate data operations, filter(), group_by(), and summarise(), are all used, with the results from each being sent to the next instruction using the %>% operator. The three instructions carry out the following tasks:

- **filter()**: This is similar to subset(). This operation keeps only rows that meet certain requirements, in this case, data for which networkDomain (the originating ISP of the page view) is in the top three most common ISPs. This has already been calculated and stored within topThree$networkDomain (this step is omitted here for brevity).
• `group_by()`: This allows operations to be carried out on subsets of data points, in this case, data points subsetted by the year and month and by the originating ISP.

• `summarise()`: This carries out summary functions such as `sum` or `mean` on several data points.

So, to summarize, the preceding code filters the data to select only the ISPs with the most users overall, groups it by the year or month and the ISP, and finds the sum or mean of several of the metrics within it (`sessionDuration`, `users`, and so on).

### A simple interactive line plot

We already saw how easy it is to draw line plots in `ggplot2`. Let’s add some Shiny magic to a line plot now. This can be achieved very easily indeed in RStudio by just navigating to **File | New | R Markdown | New Shiny document** and installing the dependencies when prompted. This will create a new R Markdown document with interactive Shiny elements. R Markdown is an extension of Markdown ([daringfireball.net/projects/markdown/](http://daringfireball.net/projects/markdown/)), which is itself a markup language, such as HTML or LaTeX, which is designed to be easy to use and read. R Markdown allows R code chunks to be run within a Markdown document, which renders the contents dynamic. There is more information about Markdown and R Markdown in Chapter 2, *Building Your First Application*. This section gives a very rapid introduction to the type of results possible using Shiny-enabled R Markdown documents.

For more details on how to run interactive documents outside RStudio, refer to [goo.gl/NGubdo](http://goo.gl/NGubdo). Once the document is set up, the code is as follows:

```r
# add interactive UI element
inputPanel(
  checkboxInput("smooth", label = "Add smoother?", value = FALSE)
)

# draw the plot
renderPlot({
  thePlot = ggplot(groupByDate, aes(x = Date, y = meanSession,
                  group = networkDomain, colour = networkDomain)) +
  geom_line() + ylim(0, max(groupByDate$meanSession))
  if(input$smooth){
    thePlot = thePlot + geom_smooth()
  }
  print(thePlot)
})
```
That's it! You’ll have an interactive graphic once you run the document (click on Run document in RStudio or use the run() command from the rmarkdown package), as shown in the following screenshot:

As you can see, Shiny allows us to turn on or off a smoothing line courtesy of `geom_smooth()` from the ggplot2 package.

Producing an interactive map (click to examine the value associated with each country) using the ggvis package is as simple as the following:

```r
getUsers = function(x){
  if(is.null(x)) return(NULL)
  theCountry = head(filter(map.df, id == x$id), 1)$CNTRY_NAME
  return(filter(groupByCountry, country == theCountry)$users)
}

map.df %>%
group_by(group, id) %>%
ggvis(~long, ~lat) %>%
layer_paths(fill = ~ users) %>%
scale_numeric("fill", trans = "log", label = "log(users)") %>%
add_tooltip(getUsers, "click") %>%
hide_axis("x") %>% hide_axis("y")
```
The final result looks like the following screenshot:

As you can see, the number of users is shown for the USA. This has been achieved simply by clicking on this country. Don't worry if you can't follow all of this code; this section is just designed to show you how quick and easy it is to produce effective and interactive visualizations.

**Installing Shiny and running the examples**

Shiny can be installed using standard package management functions as described previously (using the GUI or running `install.packages("shiny")` at the console).

Let's run some of the examples:

```r
> library(shiny)
> runExample("01_hello")
```
Your web browser should launch and display the following screenshot (note that I clicked on the show below button on the app to better fit the graphic on the page):

The graph shows the frequency of a set of random numbers drawn from a statistical distribution known as the normal distribution, and the slider allows users to select the size of the draw, from 0 to 1000. You will note that when you move the slider, the graph updates automatically. This is a fundamental feature of Shiny, which makes use of a reactive programming paradigm.

This is a type of programming that uses reactive expressions, which keep track of the values on which they are based that can change (known as reactive values) and update themselves whenever any of their reactive values change. So, in this example, the function that generates the random data and draws the graph is a reactive expression, and the number of random draws that it makes is a reactive value on which the expression depends. So, whenever the number of draws changes, the function re-executes.
Getting Started with R and Shiny!

You can find more information about this example as well as a comprehensive tutorial for Shiny at shiny.rstudio.com/tutorial/.

Also, note the layout and style of the web page. Shiny is based by default on the bootstrap theme (getbootstrap.com/). However, you are not limited by the styling at all and can build the whole UI using a mix of HTML, CSS, and Shiny code.

Let's look at an interface made with bare-bones HTML and Shiny. Note that in this and all subsequent examples, we're going to assume that you run `library(shiny)` at the beginning of each session. You don't have to run it before each example except at the beginning of each R session. So, if you have closed R and come back, then run it at the console. If you can't remember, run it again to be sure, as follows:

```r
> runExample("08_html")
```

And here it is, in all its customizable glory:

![HTML UI](image-url)
Now there are a few different statistical distributions to pick from and a different method of selecting the number of observations. By now, you should be looking at the web page and imagining all the possibilities there are to produce your own interactive data summaries and styling them just how you want, quickly and simply. By the end of the next chapter, you'll have made your own application with the default UI, and by the end of the book, you'll have complete control over the styling and be pondering where else you can go.

There are lots of other examples included with the Shiny library; just type `runExample()` at the console to be provided with a list.

To see some really powerful and well-featured Shiny applications, take a look at the showcase at shiny.rstudio.com/gallery/.

Summary

In this chapter, we installed R and explored the different options for GUIs and IDEs, and looked at some examples of the power of R. We saw how R makes it easy to manage and reformat data and produce beautiful plots with a few lines of code. You also learned a little about the coding conventions and data structures of R. We saw how to format a dataset and produce an interactive plot in a document quickly and easily. Finally, we installed Shiny, ran the examples included in the package, and got introduced to a couple of basic concepts within Shiny.

In the next chapter, we will go on to build our own Shiny application using the default UI.
Where to buy this book


Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.