Mastering RStudio – Develop, Communicate, and Collaborate with R

RStudio helps you to manage small to large projects by giving you a multi-functional integrated development environment combined with the power and flexibility of the R programming language, which is becoming the lingua franca of data science for developers and analysts worldwide.

This book begins by guiding you through the installation of RStudio and explaining the user interface step by step. From there, the next logical step is to use this knowledge to improve your data analysis workflow. You will do this by building up your toolbox to create interactive markdown reports, pretty data visualizations, graphs, and even professional web applications and dashboards with the Shiny framework. Start to collaborate and share your R code with others, by exploring how to use Git and GitHub with RStudio, and how to build your own packages to ensure top-quality results. Finally, you put it all together in an interactive dashboard written with R.

Who this book is written for

This book is aimed at all R developers and analysts who wish to take advantage of RStudio’s extensive functionalities to make their R statistical development efforts easier. R programming experience is assumed, as well as knowledge of R’s basic structures and a number of functions.

What you will learn from this book

- Discover the RStudio IDE and details about the user interface
- Communicate your insights with R Markdown in static and interactive ways
- Learn how to use different graphic systems to visualize your data
- Build interactive web applications with the Shiny framework, in order to present and share your results
- Apply your knowledge about RStudio and R development to create a real-world dashboard solution
- Understand the process of package development and assemble your own R packages
- Easily collaborate with other people on your projects by using Git and GitHub
- Manage the R environment for your organization with RStudio and the Shiny server


Julian Hillebrand
Maximilian H. Nierhoff

Harness the power of RStudio to create web applications, R packages, markdown reports and pretty data visualizations.

Prices do not include local sales tax or VAT where applicable.

$ 49.99 US
£ 31.99 UK

PUBLISHING

Open Source

Perfect Bound Cover Template

Lightning Source

Document Size: 19" x 12"
290 x 443 mm

7.50 x 9.25
239 mm x 191 mm

7.50 x 9.25
239 mm x 191 mm
In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 1 'The RStudio IDE – an Overview'
- A synopsis of the book’s content
- More information on Mastering RStudio – Develop, Communicate, and Collaborate with R
About the Authors

Julian Hillebrand studied international business marketing management at the Cologne Business School in Germany. His interest in the current questions of the business world showed him the importance of data-driven decision-making. Because of the growing size of available inputs, he soon realized the great potential of R for analyzing and visualizing data. This fascination made him start a blog project about using data science, especially for social media data analysis, which can be found at http://thinktostart.com/. He managed to combine his hands-on tutorials with his marketing and business knowledge.

Julian is always looking for new technological opportunities and is also interested in the emerging field of machine learning. He completed several digital learning offerings to take his data science capabilities to the next level.

Maximilian H. Nierhoff is an analyst for online marketing with more than half a decade of experience in managing online marketing channels and digital analytics. After studying economics, cultural activities, and creative industries, he started building online marketing departments and realized quickly that future marketing forces should also have programming knowledge. He has always been passionate about everything related to the topics of data, marketing, and customer journey analysis. Therefore, he has specialized in using R since then, which is his first-choice language for programming, data science, and analysis capabilities. He considers himself a lifelong learner and is an avid user of MOOCs, which are about R and digital analytics.
Preface

Data analysis, visualization, and the handling of complex statistical issues was reserved just for universities and very few organizations for a long time. In fact, an easy-to-use and free environment to make the concept of data analysis available to a broader audience was not available.

But in the early nineties, R saw the light of day, and since then, it has been on a meteoric rise. R has shaped the landscape of data science in recent years like no other programming language. Because of its open source nature, it became widely known and is often referred to as the lingua franca of data analysis. Another reason for this huge success is the availability of a sophisticated Integrated Development Environment (IDE) named RStudio.

The development of RStudio started in 2010, and now, it is the de facto, go-to IDE for everybody working with R. The mission statement of RStudio is "to provide the most widely used open source and enterprise-ready professional software for the R statistical computing environment."

But RStudio offers more than just a handy way to create R scripts; it grew to a real ecosystem by providing a variety of functionalities like package, application, interactive reporting creation, and more. Walking this way, RStudio has managed to bring data analysis to a broader audience. And because of its continuous desire to innovate R and its possibilities, it can be seen as a further development of the R language. RStudio combines the strong statistical power of R, the community, and open source spirit with cutting edge technologies of user interface development.

This made RStudio more than just a tool for statisticians; it became the platform for everybody who wants to generate insights from data and share them with others.

Therefore, we will hereafter guide you to develop, communicate, and collaborate with R by mastering RStudio.
What this book covers

Chapter 1, *The RStudio IDE – an Overview*, describes how to install RStudio, and gives a general overview of its user interface.

Chapter 2, *Communicating Your Work with R Markdown*, shows how to create R Markdown documents and presentations with the help of the concept of reproducible research.

Chapter 3, *R Lesson I – Graphics System*, gives an introduction to the landscape of plotting packages in R and the basic process of plot creation with different packages for interactive graphs.

Chapter 4, *Shiny – a Web-app Framework for R*, describes how to create web applications with the Shiny framework by explaining the basic concept of reactive programming.

Chapter 5, *Interactive Documents with R Markdown*, explains how to create interactive R Markdown documents with the Shiny framework and other R packages.

Chapter 6, *Creating Professional Dashboards with R and Shiny*, introduces the concept of dashboards, and how to build a professional dashboard with the shinydashboard package.

Chapter 7, *Package Development in RStudio*, describes the basic process of package development in R, and how to create R packages with RStudio.

Chapter 8, *Collaborating with Git and GitHub*, shows the fundamentals of Git and GitHub, and how to use them with RStudio.

Chapter 9, *R for your Organization – Managing the RStudio Server*, describes how to install R, RStudio, and the Shiny Server on a cloud server to create a fully flexible programming environment.

Chapter 10, *Extending RStudio and Your Knowledge of R*, explains where you can find additional resources to improve your work with R and RStudio.
The number of users adopting the R programming language has been increasing faster and faster in the last few years. It is not just used for smaller analyses, but also for bigger projects, and often, several people collaborating on the same project. The functions of the R console are limited when it comes to managing a lot of files, or when we want to work with version control systems. This is the reason, in combination with the increasing adoption rate, why a need for a better development environment arose. To serve this need, a team of R fans began to develop an integrated development environment (IDE) to make it easier to work on bigger projects and to collaborate with others. This IDE has the name, RStudio. We will introduce you to this fantastic software and show you how to take your R programming to the next level. Mastering the use of RStudio will help you solve real-world problems faster and more effectively.

In this chapter, we will introduce you to the RStudio interface and build the foundation for more advanced topics in the following chapters.

This chapter covers the following topics:

- Downloading and installing RStudio
- Getting to know the RStudio interface
- Working with RStudio projects

**Downloading and installing RStudio**

Before installing RStudio, you should install R on your computer. RStudio will then automatically search for your R installation.
Installing R

RStudio is based on the R framework and it requires, at least, R version 2.11.1, but we highly recommend that you install the latest version. The latest version of R is 3.2.2, as of September 2015.

We assume that most readers are using Windows or Mac OS systems. The installation of R is pretty simple. Just go to http://cran.rstudio.com, download the proper version of R for your system, and install it using the default setting.

We would like to leave more space to talk about installing R on different Linux distributions. As there are a huge number of different Linux distributions out there, we will focus, in this book, on the most used one: Ubuntu.

For Ubuntu

CRAN hosts repositories for Debian and Ubuntu. To install the latest version of R, you should add the CRAN repository to your system.

The supported releases are: Utopic Unicorn (14.10), Trusty Tahr (14.04; LTS), Precise Pangolin (12.04; LTS), and Lucid Lynx (10.04; LTS). However, only the latest Long Term Support (LTS) is fully supported by the R framework development team.

We will take Ubuntu 14.04 LTS as an example. Perform the following steps:

1. Open a new terminal window.
2. Add the repository for Ubuntu 14.04 to the file /etc/apt/sources.list:

   $ sudo sh -c "echo 'deb http://cran.rstudio.com/bin/linux/ubuntu trusty/'>/etc/apt/sources.list"

3. The Ubuntu archives on CRAN are signed with a key, which has the key ID, E084DAB9. So, we have to add the key to our system:

   $ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E084DAB9

4. Update the system and repository:

   $ sudo apt-get update

5. Install R with:

   $ sudo apt-get install r-base

6. Install the developer package:

   $ sudo apt-get install r-base-dev
Installing RStudio on Windows and Ubuntu is pretty much the same, as RStudio offers installers for nearly all platforms. The steps are listed as follows:

2. Download the newest installer for your system.
3. Install RStudio using the default settings.

**Using RStudio with different versions of R**

As R updates continuously, it is possible that you have, even after a short time, several versions of R installed on your system. Sometimes, you also have projects that require an older version of R to run properly.

**Windows**

When R is installed on Windows, it automatically writes the version being installed into the registry as the current version of R. And this will also be the version that RStudio uses. You can choose the version of R that you want to use by holding the Ctrl key during the launch of RStudio.

**Ubuntu**

On Linux, you can use a command with R to see which version of R, RStudio uses. If you want RStudio to use another version of R (maybe you want to use an older version or because you had to install R in your Documents folder because of missing admin rights) you can overwrite the settings with the following export: `RSTUDIO_WHICH_R=/usr/local/bin/R`. This line has to be added to your `~/.profile` file.

**Updating RStudio**

Updating RStudio is as easy as installing it. If you want to check if an update is available, navigate to Help | Check for Updates.

If an update is available, you can download the newest version and just install it. As RStudio saves all user information in the user's home directory, they will still be there after the update.
Getting to know the RStudio interface
Now, we can take a look at RStudio's user interface.

The four main panes
When you start RStudio for the first time, you will see four main panes. If you want

to customize the four main panes, you can do it by navigating to **Tools | Global

Options | Pane Layout.**

We will explain their use, but first we need to create a new R script file by clicking on

**File | New File | R Script.**
The new R script file is opened in a new pane and is named \textit{Untitled1}. 

[Image of RStudio interface]
You can see that we now have four panes. They are named as follows:

- The **Source editor** pane
- The **Environment** and **History** pane
- The **Console** pane
- The **Files, Plots, Packages, Help, and Viewer** pane

**The Source editor pane**

RStudio's source editor was developed in a fully functional R editor over the last few years. It has a powerful syntax highlighter that works with not only every format connected to R development, such as R Scripts, R Markdown, or R documentation files, but also C++, JavaScript, HTML, and many more.

We've already created a new R script file and can now demonstrate some of the code editor's functions. You can also open an existing R document by clicking on **File | Open File**, or by using the shortcut, `Ctrl + O`.

The code editor works with tabs, which gives you the possibility of opening several files at the same time, as you can see in the following screenshot. If there are unsaved changes in a file, their names will be highlighted in red and marked with an asterisk.
If you have several files opened, you will see a double arrow in the menu of the source code editor. This will open a small menu showing you an overview of all the opened files. You can also search for a specific file.

Under the tabs with the opened files, you can see a toolbox with tools for the code editor. For example, you have the Source on Save checkbox. This is a really handy tool especially when you are working on a reusable function. If activated, the function is automatically sourced to the global environment and we do not have to source it manually again after editing the code.

Another function you can find in the toolbox is the search and replace tool. This is known from a lot of text editors and helps you find existing code and replace it. RStudio also offers different options for your search, such as In selection, to just search in the code you selected in the editor or Match case, to make the search case-sensitive. This is demonstrated in the following screenshot:
Syntax highlighting
RStudio highlights parts of your code according to the R language definition. This makes your code much easier to read. The default settings are:

- The R keywords being blue
- The text strings being green
- Numbers being dark blue
- Comments being dull green

Code completion
One of the most important menus in the source editor is what you find when you click on the magic stick. If you forgot what exact arguments the selected function needs, just hit the Tab button and you will see a list of available arguments with a description, if available:

You can then scroll through the list and select the argument you want to use. This is especially useful when you have functions that can be called with a lot of different arguments; it would be very time-consuming to open the package documentation for every function call.
You can also find direct links to the help or function definition, which shows you where the current function is defined.

After that, you can find the functions, Extract Function and Extract Variable. These functions help you in creating functions. When you click on Extract Function or use the shortcut, \textit{Ctrl + Alt + X}, RStudio creates a function from your selection and inserts it in the source code.

After executing the command, your code will look like this:

```
plot_sin_cos <- function(x) {
  plot(x, sin(x), type="l")
  lines(x, cos(x), type="l", col="red")
}
```

The next button is the Compile Notebook button. This helps you compile your currently opened source file into a notebook with the format, HTML, PDF, or MS Word:
The compiled report will then open in a new window.

This is the code we used for the preceding example; if you want to reproduce it, type the following code:

```r
x <- 10 + (1:20)/10
y <- x^2 + rnorm(length(x))  # Add Gaussian random number
plot(x, y)
```

**Executing R Code from the source pane**

On the extreme right of the source code menu, you will find the buttons needed to run the code. These buttons are:

- **The Run button** executes a single line and the shortcut is `Ctrl + Enter`
- **To re-run the previous region** (`Ctrl + Shift + P`)
- **The Source button** executes the entire source file (`Ctrl + Shift + Enter`)
Code regions are foldable regions of code in the code editor. We will explain later how you can create them.

If you want to execute a single line, or rather, if you want to run the current line where your cursor is, you can use the Run button or the shortcut, Ctrl + Enter. After the execution, the cursor will jump to the next line in the source file.

If you want to execute several lines of code, you can select the lines and press the Run button.

**Code folding**

RStudio supports both automatic and user-defined folding for regions of code. This is a very handy feature, especially when you work with functions and larger scripts. It lets you hide and show blocks to make the code easier to navigate.

RStudio automatically folds the following regions in the source editor:

- Braced regions (function definitions, conditional blocks, and so on)
- Code chunks within R Sweave or R Markdown documents
- Code sections (user-defined)

The output looks like this:

To define a code section on your own and to make it easier to navigate in larger source files, you can use three methods:

- # Section One ----------------------
- # Section Two =============
- ### Section Three #############
So, the line can start with any number of pound signs (#), but is has to end with at least four or more −, =, or # characters. RStudio then automatically defines the following code as the section. To navigate between code sections, you can use the Jump To menu at the bottom of the editor.

The menu at the bottom, on the right-hand side lets you choose the file format of the currently opened source file. Normally, RStudio chooses the right format automatically. If you change it manually, the code completion and the syntax highlighting will adapt to the new settings.
Debugging code

RStudio offers visual debuggers to help you understand code and find bugs and problems. Therefore, it uses the debugging functions of R but integrates them seamlessly into the RStudio user interface. You can find these tools in the Debug tab of the menu, or by pressing Alt + D:

You can set breakpoints right in the source editor by clicking on the number of the line, or by pressing Shift + F9:

The debugger output can help you find bugs in your code in a better way. In this example, the debugger output is debug.R:10. This means that we should look into the tenth line of the source file:
The Environment and History panes

With the default settings, this pane consists of the tabs, Environment and History. You can use the shortcut, Ctrl + 8, to switch to the Environment browser, and Ctrl + 4 to switch to the History window:

The Environment pane is one of the biggest advantages of RStudio. It gives you an overview over all objects currently available in an environment. So, you can see a list of all data, values, and functions.

The Environment browser shows you the number of observations and the number of variables in the second column. If you want to get a better overview of a dataset, you can click on the table symbol at the end of the row.
When you click on the blue and white arrow next to the name of an object, you will see its structure. This is basically the output of the `str()` function, but in a more structured way.

The **Import Dataset** button offers you an easy way to import data. It basically uses the `read.csv()` function but offers you a graphical interface to set the parameters for the import. You can either import the dataset from a local file, or you can choose an import from a URL.

Furthermore, the Environment pane gives you the possibility of clearing the environment, which will delete all defined variables and also all sourced functions.
History pane
The History pane shows all the commands you entered in the console, and it also lets you send the selected command back from the history directly to the console with the To Console button or back to the opened source code file with the To Source button. You can also delete commands from the history by selecting them and pressing the paper icon with the red close sign above the history. Or you can clear the whole history by clicking the broom icon:

Console pane
The console pane is basically an R console but it is enhanced with some RStudio functions. This includes the command completion known from the source editor, and a history popup, which shows you the recent commands you used.

The keyboard shortcuts for the console pane are:

- **Command completion**: Tab
- **Command history popup**: Ctrl + arrow up
- **Clear console**: Ctrl + L
- **Go through historical command**: arrow up

The Files, Plots, Packages, Help, and Viewer panes
This pane is, like the name says, divided into five sub panes: Files, Plots, Packages, Help, and Viewer.
The Files pane

This pane is one of RStudio's biggest enhancements in comparison to the normal R console. The Files pane shows you all the files in the current working directory. It includes information about the file size and when the data was last modified. Clicking on an item will open it with the appropriate application.

![Files pane](image1)

The Plot pane

The Plot pane in RStudio handles all of your graphics output. This makes working with graphical output much easier than in the regular R console, as it opens a new window for every graphic.

![Plot pane](image2)

Furthermore, the Plot pane gives some more tools. These tools include the option to zoom into a graphic. This will open a new window with a bigger version of the current plot. This plot will then arrange itself to the current window size.
You can also export the current plotted graphic with the Export button. The Export menu has three options:

- To save the plot as an image
- To save the plot as a PDF
- To copy the plot to the clipboard

When you choose the Save as Image... option, RStudio will open a popup that lets you define the export image format, the directory, and the file name, as well as the width and height.
The **Save as PDF...** option will create a single page PDF document with your plot. Based on the width and height settings, it will be either in the landscape or portrait format.

RStudio also offers the option to publish your plots on RPubs. This is a free and very simple web service from the makers of RStudio to upload R graphics and R Markdown documents, which will then be publicly available on the web and you can share the link. We will talk about the possibilities of R markdown in a later chapter.

When you click on the **Publish** button, a window will open and guide you through the process.

![Publish window](image)

After clicking on **Publish**, a new browser window will open and show your uploaded report:

![Uploaded report](image)
The Packages pane

The Packages pane helps you install, update, or load packages. It gives you an overview about all installed packages, a short description, and the installed version.

If you tick a checkbox in front of a package, it will automatically be loaded, and if you remove the tick again, RStudio will automatically detach it from the environment. So, it basically unloads it again.

The Packages pane also provides a handy tool to install new packages with the help of a graphical interface. We just have to click on the Install button and we will be guided through the installation process. The Install packages dialog also allows us to install packages that we have saved locally on our computer:

You can see next what RStudio does in the R console:

```
> library("RCurl", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.1")
Loading required package: bitops
> detach("package:RCurl", unload=TRUE)
> 
```
The Help pane
A big advantage of the R language is that every package on CRAN will come with package documentation. You can find these files on the CRAN website but RStudio bundles them in a handy Help pane. You can search the help through the search bar, or you can just press F1:

![Help pane diagram]

The Viewer pane
The Viewer pane in RStudio can be used to view local web content, such as web graphics created with packages such as rCharts, googleVis, and others. It can also show local web applications created with Shiny or OpenCPU.

![Viewer pane diagram]
Now, we will click on **Save as Web Page...** in the Export menu.

The export menu of the viewer pane offers, basically, the same option to export your work but replaces the **Save as image** option with **Save as Web Page**. This creates a standalone web page.

### Customizing RStudio

The default options of RStudio are the best for most people, but you can also change the appearance and the pane layout completely according to your needs and wishes. We can open the **Options** menu by clicking on **Tools | Global Options**:
RStudio offers a lot of ways to personalize the code editing. We can, for example, set the spaces that will be inserted when we use the Tab key, or change the diagnostics information shown. You also have the Appearance tab, as shown next:

Here you can edit, for example, the font used in the code editor, or the editor theme. This way, you can make RStudio look the way you want it to.
And the Pane Layout tab: In this pane, we can change the content of the four main panes in the Pane Layout tab. You can make each of them a source, a console, or an individualized pane. So, the last option means that you can easily add elements to the pane with the help of the checkboxes.

![Pane Layout Tab](image)

**Using keyboard shortcuts**

The fastest way to use RStudio is by using it with keyboard shortcuts. In the previous text, we already mentioned some of them. But we put the most important ones together in a table, which is as follows:

<table>
<thead>
<tr>
<th>Description</th>
<th>Windows and Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move the focus to the Source editor</td>
<td>Ctrl + 1</td>
<td>Ctrl + 1</td>
</tr>
<tr>
<td>Move the focus to console</td>
<td>Ctrl + 2</td>
<td>Ctrl + 2</td>
</tr>
<tr>
<td>Move the focus to Help</td>
<td>Ctrl + 3</td>
<td>Ctrl + 3</td>
</tr>
<tr>
<td>Show the History pane</td>
<td>Ctrl + 4</td>
<td>Ctrl + 4</td>
</tr>
</tbody>
</table>
### Description

<table>
<thead>
<tr>
<th>Description</th>
<th>Windows and Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show the <strong>Files</strong> pane</td>
<td>Ctrl + 5</td>
<td>Ctrl + 5</td>
</tr>
<tr>
<td>Show the <strong>Plots</strong> pane</td>
<td>Ctrl + 6</td>
<td>Ctrl + 6</td>
</tr>
<tr>
<td>Show the <strong>Packages</strong> pane</td>
<td>Ctrl + 7</td>
<td>Ctrl + 7</td>
</tr>
<tr>
<td>Show the <strong>Environment</strong> pane</td>
<td>Ctrl + 8</td>
<td>Ctrl + 8</td>
</tr>
<tr>
<td>Open the document</td>
<td>Ctrl + O</td>
<td>Command + O</td>
</tr>
<tr>
<td>Run the current line/section</td>
<td>Ctrl + Enter</td>
<td>Command + Enter</td>
</tr>
<tr>
<td>Clear the console</td>
<td>Ctrl + L</td>
<td>Command + L</td>
</tr>
<tr>
<td>Extract the function from the selection</td>
<td>Ctrl + Alt + X</td>
<td>Command + Option + X</td>
</tr>
<tr>
<td>Source the current document</td>
<td>Ctrl + Shift + Enter</td>
<td>Command + Shift + Enter</td>
</tr>
<tr>
<td>Toggle the breakpoint</td>
<td>Shift + F9</td>
<td>Shift + F9</td>
</tr>
</tbody>
</table>

---

**Working with RStudio and projects**

In the times before RStudio, it was very hard to manage bigger projects with R in the R console, as you had to create all the folder structures on your own.

When you work with projects or open a project, RStudio will instantly take several actions. For example, it will start a new and clean R session, it will source the `.Rprofile` file in the project's main directory, and it will set the current working directory to the project directory. So, you have a complete working environment individually for every project. RStudio will even adjust its own settings, such as active tabs, splitter positions, and so on, to where they were when the project was closed.

But just because you can create projects with RStudio easily, it does not mean that you should create a project for every single time that you write R code. For example, if you just want to do a small analysis, we would recommend that you create a project where you save all your smaller scripts.
Creating a project with RStudio

RStudio offers you an easy way to create projects. Just navigate to File | New Project and you will see a popup window with the following options:

- New Directory
- Existing Directory
- Version Control

These options let you decide from where you want to create your project. So, if you want to start it from scratch and create a new directory, associate your new project to an existing one, or if you want to create a project from a version control repository, you can avail of the respective options. For now, we will focus on creating a new directory.

The following list will show you the next options available:

- Empty Project
- R Package
- Shiny Web Application
Chapter 1

We will look in the categories, R Package and Shiny Web Application later in this book, so for now we will concentrate on the Empty Project option.

Locating your project
A very important question you have to ask yourself when creating a new project is where you want to save it? There are several options and details you have to pay attention to especially when it comes to collaboration and different people working on the same project.

You can save your project locally, on a cloud storage or with the help of a revision control system such as Git.

Using RStudio with Dropbox
An easy way to store your project and to be able to access it from everywhere is the use of a cloud storage provider like Dropbox. It offers you a free account with 2 GB of storage, which should be enough for your first project.

Preventing Dropbox synchronization conflicts
RStudio actively monitors your project files for changes, which allows it to index functions and files to enable code completion and navigation. But when you use Dropbox at the same time to remotely sync your work, it will also monitor your files and this can cause conflicts. So you should tell Dropbox to ignore the .Rproj.user directory in your RStudio project.

To ignore a file in Dropbox, navigate to Preferences | Account | Selective Sync and uncheck the .Rproj.user directory.
Dropbox also helps you with version control, as it keeps previous versions of a file.

Creating your first project

To begin your first project, choose the New Directory option we described before and create an empty project. Then, choose a name for the directory and the location that you want to save it in. You should create a projects folder on your Dropbox.

The first project will be a small data analysis based on a dataset that was extracted from the 1974 issue of the Motor Trend US magazine. It comprises fuel consumption and ten aspects of automobile design and performance, such as the weight or number of cylinders for 32 automobiles, and is included in the base R package. So, we do not have to install a separate package to work with this dataset, as it is automatically loaded when you start R.
As you can see, we left the Use packrat with this project option unchecked. Packrat is a dependency management tool that makes your R code more isolated, portable, and reproducible by giving your project its own privately managed package library. This is especially important when you want to create projects in an organizational context where the code has to run on various computer systems, and has to be usable for a lot of different users. This first project will just run locally and will not focus on a specific combination of package versions.

Organizing your folders
RStudio creates an empty directory for you that includes just the file, Motor-Car-Trend-Analysis.Rproj. This file will store all the information on your project that RStudio will need for loading. But to stay organized, we have to create some folders in the directory. Create the following folders:

- **data**: This includes all the data that we need for our analysis
- **code**: This includes all the code files for cleaning up data, generating plots, and so on
- **plots**: This includes all graphical outputs
- **reports**: This comprises all the reports that we create from our dataset

This is a very basic folder structure and you have to adapt it to your needs in your own projects. You could, for example, add the folders, raw and processed, in the data folder. Raw for unstructured data that you started with, and processed for cleaned data that you actually used for your analysis.
**Saving the data**

The Motor Trend Car Road Tests dataset is part of the `dataset` package, which is one of the preinstalled packages in R. But, we will save the data in a CSV file in our data folder, after extracting the data from the `mtcars` variable, to make sure our analysis is reproducible:

```r
# write data into csv file
write.csv(mtcars, file = "data/cars.csv", row.names=FALSE)
```

Put the previous line of code in a new R script and save it as `data.R` in the `code` folder.

**Analyzing the data**

The analysis script will first have to load the data from the CSV file with the following line:

```r
cars_data <- read.csv(file = "data/cars.csv", header = TRUE, sep = ",")
```

**Correcting the path for report exporting**

If you want to create a report from your R script, you have to specify the relative path to the data file, beginning with two dots:

```r
cars_data <- read.csv(file = "../data/cars.csv", header = TRUE, sep = ",")
```

Next, we can take a look at the different variables and see if we can find any correlations on the first look. We can create a `pairs` matrix with the following line:

```r
pairs(cars_data)
```
We can then save the created matrix with the export function of the Plots Pane option. Then, we can save it as an image in the plots folder:

As you can see, we can expect a lot of different variable combinations, which could correlate very well. The most obvious one is surely weight of the car (wt) and Miles per Gallon (mpg): a heavy car seems to need more gallons of fuel than a lighter car.

We can now test this hypothesis by calculating the correlation and plotting a scatterplot of these two variables. In addition, we can also do a linear regression and see how it performs:

```r
cor(cars_data$wt, cars_data$mpg)
```

```r
install.packages("ggplot2")
require(ggplot2)
```
```r
ggplot(cars_data, aes(x=wt, y=mpg)) +
  geom_point(aes(shape=factor(am, labels = c("Manual","Automatic")))) +
  geom_smooth(method=lm) + scale_shape_discrete(name = "Transmission Type")

firstModel <- lm(mpg~wt, data = cars_data)
```

We can see more details with:

```r
summary(firstModel)$coef
```

```
[1] -0.8676594
```

```r
print(c('R-squared', round(summary(firstModel)$r.sq,2)))
```

```
[1] "R-squared" "0.75"
```
We can see that there is a high negative correlation between these two variables, and the first model is a pretty good fit with an R-squared value of 0.75.

But we also have to test other combinations and see how they perform. And what we basically do is test all the correlations and use the best model.

We will not explain the statistical functions behind this approach, as it would be out of the scope of this chapter:

```r
# Test other correlations
completeModel <- lm(mpg ~ ., data=cars_data)
stepSolution <- step(completeModel, direction = "backward")

# get the best model
bestModel <- stepSolution$call
bestModel
```

The output will look like this:

```
> bestModel
lm(formula = mpg ~ wt + qsec + am, data = cars_data)
```

The best model now has the following formula:

mpg ~ wt + qsec + am

So, we will create a final model with this formula and see how it performs:

```r
finalModel <- lm(mpg~wt + factor(am) + qsec, data = cars_data)
summary(finalModel)$coef
print(c('R-squared', round(summary(finalModel)$r.sq,2)))
```

```
Estimate  Std. Error  t value Pr(>|t|)     
(Intercept)     9.617781       6.9595930  1.381946  1.779152e-01
wt              -3.916504       0.7112016 -5.506882  6.952711e-06
factor(am)manual 2.935837      1.41899045  2.098819  4.675551e-02
qsec             1.225886       0.2886696  4.246676  2.161737e-04
```

[1] "R-squared"  "0.85"
As we can see, the final model also includes the variable, \textit{qsec}, which is the time the car needs for a quarter mile, and \textit{am}, which is the type of transmission (automatic or manual).

But, we can also see that just the transmission type, manual, seems to play a significant role when it comes to mileage.

After you execute the analysis script, you can see that all your results are still in RStudio, which is a big advantage in contrast to the R console.

So, you can go through all the graphs you produced in the plot viewer with the arrows.
Or, you can see which variables are set in the environment. These are all the models you calculated in this analysis, as well as in your initial dataset.

You can click on the table icon behind `cars_data` in the Environment pane to open the data frame in the Source pane.
Exporting your analysis as a report

You can also export the `analysis.R` script as a report in the HTML, PDF, or MS Word format, and you will then find the report in your code folder. Therefore, just click on the Publish button and RStudio will guide you through the process.

```r
# Transform the dataset
cars_data$am <- as.factor(cars_data$am)
cars_data$cyl <- as.factor(cars_data$cyl)
cars_data$vs <- as.factor(cars_data$vs)
cars_data$gear <- as.factor(cars_data$gear)

# According to the dataset description
# Transmission (0 = automatic, 1 = manual)
levels(cars_data$am) <- c("automatic", "manual")

# Plot
library(ggplot2)
ggplot(cars_data, aes(x = wt, y = mpg)) +
ggplot2::geom_point(aes(shape = cars_data$am)) +
ggplot2::geom_smooth(method = "lm") +
ggplot2::scale_shape_discrete(name = "Transmission Type")
```
Summary
In this chapter, we learned how to install RStudio and got a general overview of its user interface. This consists of four main panes: the Source Editor pane, the console pane, the Environment and Help pane, and the Files, Plot, Help, and Viewer pane. We learned their different functions and saw what tools each pane has.

Furthermore, we learned how to create a project with RStudio in combination with Dropbox, and we started our first small data analysis.

In the next chapter, we will learn how to communicate our work with the help of R Markdown, and how to create reproducible research.
Where to buy this book

You can buy Mastering RStudio – Develop, Communicate, and Collaborate with R from the Packt Publishing website.

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

Click here for ordering and shipping details.