

An Introduction To R

Stuart Hertzog

Contents

Preface	5
1 Setup	7
1.1 Requirements	7
1.2 Quick Summary	7
1.3 Installation Details	8
1.4 GitHub Version Control	14
2 R For Data Science	21
2.1 What Is Data Science?	21
2.2 What Is R?	22
2.3 Installing R	23
2.4 Command Line R	23
2.5 The R Console Interface	25
2.6 R Packages	25
3 RStudio IDE	33
3.1 The RStudio Interface	34
3.2 RStudio Projects	34
3.3 Package Management In RStudio	34
3.4 Using Python In RStudio	40
4 The RStudio Ecosystem	41
4.1 R Notebooks	41
4.2 RMarkdown	42
4.3 Shiny	44
4.4 bookdown	45
4.5 blogdown	46
4.6 RStudio Products	47

5	The Fundamentals of R	49
5.1	Four Fundamentals	49
5.2	Basic Maths	50
5.3	Variables	50
5.4	Data Types	51
5.5	Data Structures	54
6	R Learning Resources	73
6.1	Books On R	73
6.2	R Blogs	74
6.3	R Tutorials	74
6.4	R Discussion Forums	74
6.5	R Mailing Lists	75
6.6	R Podcasts	75

Preface

This material is my own process of learning the R programming language. It can be yours, too.

Originally just notes on R, it grew into a series of presentations to the VicPiMakers Meetup Group in Victoria BC in April and May 2018. Each presentation was created in the RStudio IDE as a series of RMarkdown documents that were output to HTML then transferred to a Web hosting site by FTP.

This Series has now become what I hope is a useful tool for learning R. Instructions for setting up and using it are given in How To Set Up This Series.

Good luck with your R learning!

*Stuart Hertzog,
Victoria, BC Canada*

Chapter 1

Setup

You will get the most benefit from this Tutorial Series if you set it up as an RStudio Project and become familiar with the RStudio IDE.

1.1 Requirements

This series of tutorials on R can be run on the major computer platforms – Mac, Linux, and Windows.

The necessary files are available in a GitHub repository. To download and use them you will need these programs installed on your computer:

- R
- RStudio
- git

Once these are properly set up, you can proceed to Installation:

1.2 Quick Summary

The stages of installing this RStudio Project are:

1. Install `r` and `git` on your computer
2. Decide where to put the *Project Directory*
3. `cd` into what will be its parent directory
4. Clone the project files from GitHub
5. Install and open RStudio on your computer
6. Create a new RStudio Project in the *Project Directory*
7. Install and initialise `packrat`
8. Set up an account on GitHub if you don't have one
9. Create a new repository in your GitHub account
10. Add it to RStudio as a *Github destination*
11. Push the files in your Project Directory to GitHub
12. Continue to Part1 of your introduction to R

It may look intimidating, but it's fairly easy to work through.

1.3 Installation Details

Carefully follow these instructions, in this order.

1.3.1 Create The Project Directory

1. In your Terminal program, `cd` into the directory where you want to create the new Project. This can be located wherever you want. It will be called `Intro2R`, so there should not be an existing directory with the same name in that location.
2. Enter the following commands one at a time at the command line in your Terminal program. Follow each line by a `RETURN`:

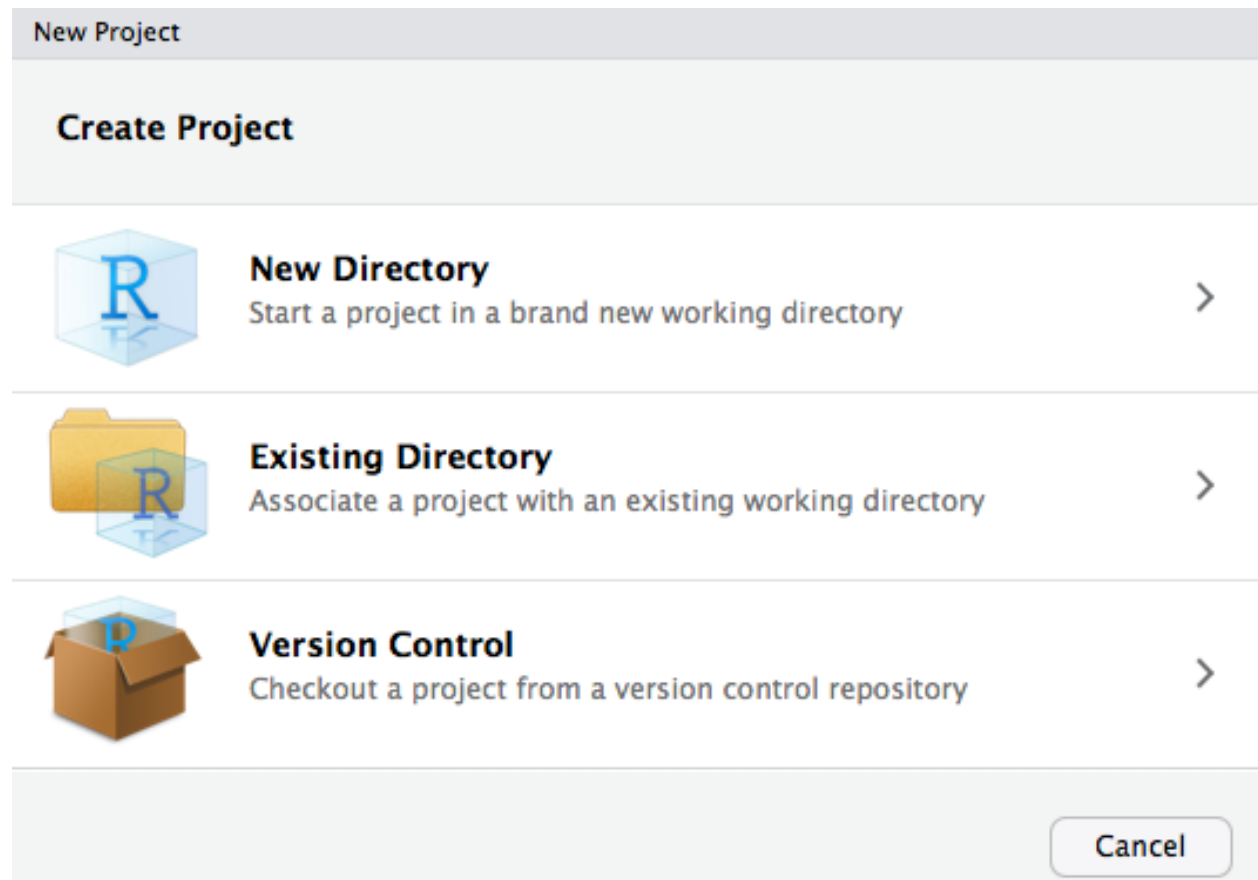
```
git clone https://github.com/stuzog/Intro2R.git
cd Intro2R
ls -al
```

You should see a listing of the GitHub repository files (with your user name):

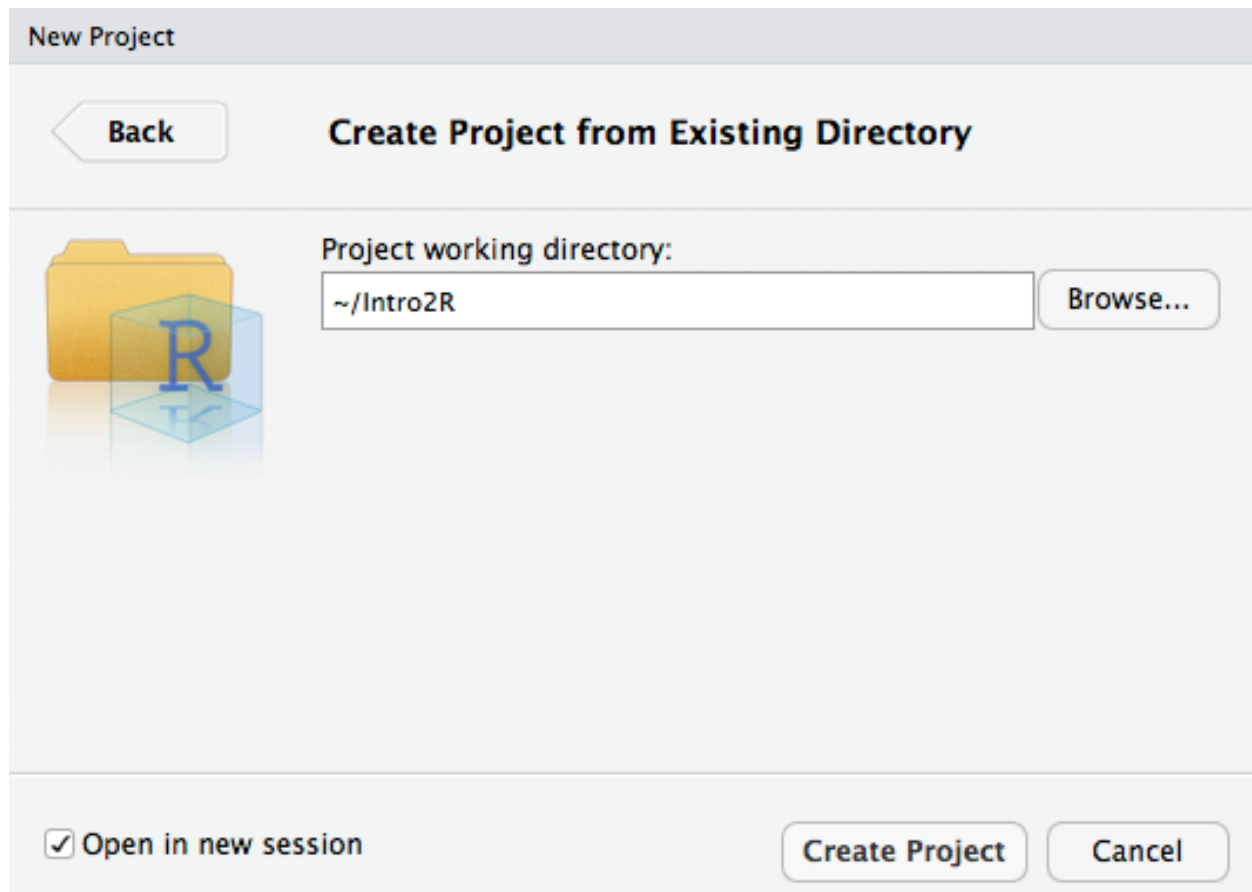
```
total 184
drwxr-xr-x  18 (username) staff   612 18 Apr 09:10 .
drwxr-xr-x@ 149 (username) staff  5066 18 Apr 09:10 ..
-rw-r--r--   1 (username) staff     0 18 Apr 09:10 .Rhistory
-rw-r--r--   1 (username) staff   117 18 Apr 09:10 .Rprofile
drwxr-xr-x  12 (username) staff   408 18 Apr 09:10 .git
-rw-r--r--   1 (username) staff    26 18 Apr 09:10 .gitignore
-rw-r--r--   1 (username) staff   536 18 Apr 09:10 How.Rmd
-rw-r--r--   1 (username) staff   225 18 Apr 09:10 Intro2R.Rproj
-rw-r--r--   1 (username) staff 18366 18 Apr 09:10 Part1_R_Data Science.Rmd
-rw-r--r--   1 (username) staff 13654 18 Apr 09:10 Part2_R_Programming.Rmd
-rw-r--r--   1 (username) staff   1360 18 Apr 09:10 README.md
-rw-r--r--   1 (username) staff 12253 18 Apr 09:10 RStudio.Rmd
-rw-r--r--   1 (username) staff   8000 18 Apr 09:10 RStudio_Ecosystem.Rmd
-rw-r--r--   1 (username) staff   2071 18 Apr 09:10 R_Learning_Resources.Rmd
-rw-r--r--   1 (username) staff    817 18 Apr 09:10 _site.yml
drwxr-xr-x  31 (username) staff  1054 18 Apr 09:10 images
-rw-r--r--   1 (username) staff   1682 18 Apr 09:10 index.Rmd
-rw-r--r--   1 (username) staff    687 18 Apr 09:10 intro.css
```

1.3.2 Make It An RStudio Project

3. **Launch RStudio on your computer.** Unless you have used RStudio before, you will see a fairly unpopulated RStudio window.
4. **Start a new RStudio project** by clicking on **R Projects** in the upper-right corner of the RStudio window, or via the *File > New Project...* menu item. The *Create New Project* window will appear:



5. Select **Existing Directory** and navigate to the newly-created **Intro2R** project directory. Click *Open in new session* if you wish to preserve the current RStudio session.



6. Click **Create Project**. You will see the following **Error** and **Warning** messages, which you can ignore for now. They will be corrected later in the setup.

```
Error in file(filename, "r", encoding = encoding) :
  cannot open the connection
In addition: Warning message:
In file(filename, "r", encoding = encoding) :
  cannot open file 'packrat/init.R': No such file or directory
```

1.3.3 Install packrat

Because **R** packages are updated frequently, it is recommended that you use a project package management system. This project uses the **packrat** package dependency management system for **R**, so this should now be installed.

7. In the **Console window** at the bottom left of the RStudio interface, enter the following commands followed by **RETURN**, one at a time:

```
install.packages("packrat")
packrat::init()
```

The Console will install **packrat** and the dependencies required for the project:

```
> install.packages("packrat")
Installing package into '/Users/(username)/Library/R/3.4/library'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/macosx/el-capitan/contrib/3.4/packrat_0.4.9-1.tgz'
Content type 'application/x-gzip' length 214137 bytes (209 KB)
=====
downloaded 209 KB
```

```
The downloaded binary packages are in
  /var/folders/qt/6jh4c4yn5zdbf07t61nmz0_w0000gn/T//Rtmptyh6x4/downloaded_packages
> packrat::init()
Initializing packrat project in directory:
- "~/Intro2R"
```

Adding these packages to packrat:

```

      Rcpp      0.12.16
backports  1.1.2
base64enc  0.1-3
digest     0.6.15
evaluate   0.10.1
glue        1.2.0
highr      0.6
htmltools  0.3.6
jsonlite   1.5
knitr      1.20
magrittr   1.5
markdown   0.8
mime       0.5
packrat    0.4.9-1
rmarkdown  1.9
rprojroot  1.3-2
stringi    1.1.7
stringr    1.3.0
yaml       2.1.18
```

```

Fetching sources for Rcpp (0.12.16) ... OK (CRAN current)
Fetching sources for backports (1.1.2) ... OK (CRAN current)
Fetching sources for base64enc (0.1-3) ... OK (CRAN current)
Fetching sources for digest (0.6.15) ... OK (CRAN current)
Fetching sources for evaluate (0.10.1) ... OK (CRAN current)
Fetching sources for glue (1.2.0) ... OK (CRAN current)
Fetching sources for highr (0.6) ... OK (CRAN current)
Fetching sources for htmltools (0.3.6) ... OK (CRAN current)
Fetching sources for jsonlite (1.5) ... OK (CRAN current)
Fetching sources for knitr (1.20) ... OK (CRAN current)
Fetching sources for magrittr (1.5) ... OK (CRAN current)
Fetching sources for markdown (0.8) ... OK (CRAN current)
Fetching sources for mime (0.5) ... OK (CRAN current)
Fetching sources for packrat (0.4.9-1) ... OK (CRAN current)
Fetching sources for rmarkdown (1.9) ... OK (CRAN current)
Fetching sources for rprojroot (1.3-2) ... OK (CRAN current)
Fetching sources for stringi (1.1.7) ... OK (CRAN current)
```

```

Fetching sources for stringr (1.3.0) ... OK (CRAN current)
Fetching sources for yaml (2.1.18) ... OK (CRAN current)
Snapshot written to '/Users/Stuart/Intro2R/packrat/packrat.lock'
Installing Rcpp (0.12.16) ...
  OK (downloaded binary)
Installing backports (1.1.2) ...
  OK (downloaded binary)
Installing base64enc (0.1-3) ...
  OK (downloaded binary)
Installing digest (0.6.15) ...
  OK (downloaded binary)
Installing glue (1.2.0) ...
  OK (downloaded binary)
Installing highr (0.6) ...
  OK (downloaded binary)
Installing jsonlite (1.5) ...
  OK (downloaded binary)
Installing magrittr (1.5) ...
  OK (downloaded binary)
Installing mime (0.5) ...
  OK (downloaded binary)
Installing packrat (0.4.9-1) ...
  OK (downloaded binary)
Installing stringi (1.1.7) ...
  OK (downloaded binary)
Installing yaml (2.1.18) ...
  OK (downloaded binary)
Installing rprojroot (1.3-2) ...
  OK (downloaded binary)
Installing htmltools (0.3.6) ...
  OK (downloaded binary)
Installing markdown (0.8) ...
  OK (downloaded binary)
Installing stringr (1.3.0) ...
  OK (downloaded binary)
Installing evaluate (0.10.1) ...
  OK (downloaded binary)
Installing knitr (1.20) ...
  OK (downloaded binary)
Installing rmarkdown (1.9) ...
  OK (downloaded binary)
Initialization complete!

Restarting R session...

>

```

8. Click the **Packages** tab in the **Files... Viewer pane** (usually bottom right). It will now show a list of packages installed in the Project's dedicated *Packrat Library*:

9. Click **Update** in the **Packages tab toolbar** to ensure that all project packages are current. Packages are updated frequently, so it is advisable to do this from time to time.



The screenshot shows the 'Packrat Library' window with a menu bar (Files, Plots, Packages, Help, Viewer) and a toolbar (Install, Update, Packrat, search, refresh). Below the toolbar is a table of packages. Each row includes a checkbox, the package name, a description, the installed version, the available version, the source (CRAN), and a refresh icon.

	Name	Description	Version	Packrat	Sou...	
<input type="checkbox"/>	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.2	1.1.2	CRAN	●
<input type="checkbox"/>	base64enc	Tools for base64 encoding	0.1-3	0.1-3	CRAN	●
<input type="checkbox"/>	bitops	Bitwise Operations	1.0-6			●
<input type="checkbox"/>	caTools	Tools: moving window statistics, GIF, Base64, ROC AUC, etc.	1.17.1			●
<input type="checkbox"/>	digest	Create Compact Hash Digests of R Objects	0.6.15	0.6.15	CRAN	●
<input type="checkbox"/>	evaluate	Parsing and Evaluation Tools that Provide More Details than the Default	0.10.1	0.10.1	CRAN	●
<input type="checkbox"/>	glue	Interpreted String Literals	1.2.0	1.2.0	CRAN	●
<input type="checkbox"/>	highr	Syntax Highlighting for R Source Code	0.6	0.6	CRAN	●
<input type="checkbox"/>	htmltools	Tools for HTML	0.3.6	0.3.6	CRAN	●
<input type="checkbox"/>	jsonlite	A Robust, High Performance JSON Parser and Generator for R	1.5	1.5	CRAN	●
<input type="checkbox"/>	knitr	A General-Purpose Package for Dynamic Report Generation in R	1.20	1.20	CRAN	●
<input type="checkbox"/>	magrittr	A Forward-Pipe Operator for R	1.5	1.5	CRAN	●
<input type="checkbox"/>	markdown	'Markdown' Rendering for R	0.8	0.8	CRAN	●
<input type="checkbox"/>	mime	Map Filenames to MIME Types	0.5	0.5	CRAN	●
<input type="checkbox"/>	packrat	A Dependency Management System for Projects and their R Package Dependencies	0.4.9-1	0.4.9-1	CRAN	●
<input type="checkbox"/>	Rcpp	Seamless R and C++ Integration	0.12.16	0.12.16	CRAN	●
<input type="checkbox"/>	rmarkdown	Dynamic Documents for R	1.9	1.9	CRAN	●
<input type="checkbox"/>	rprojroot	Finding Files in Project Subdirectories	1.3-2	1.3-2	CRAN	●

Figure 1.1: *Above:* All Project packages will be installed in this project-specific library.

1.4 GitHub Version Control

It's a really good idea to track your R coding with some form of version control.

RStudio makes this easy for you.

git and GitHub have become the leading version control system — there's even a free eBook about **git** available for download . There also are some easy-to-understand explanations on how **git** works available on the Web.

git support is built-in to RStudio. This makes keeping track of your many ch-ch-ch-changes an integral part of your R programming workflow.

1.4.1 Setting Up git In RStudio

How you set up **git** in RStudio depends on whether you are:

- **Starting a new Project:**
 - first from GitHub
 - from RStudio
 - from a GitHub clone
- **Adding git to an existing Project**
 - starting from GitHub, or
 - starting from RStudio

Each alternative takes a slightly different path through the RStudio/GitHub maze. They are best detailed in Happy Git and GitHub for the useR by Jenny Bryan and the STAT 545 Teaching Assistants of UBC. RStudio's version control guide also shows how to use **git** or an SVN repository for version control.

1.4.2 Using git In RStudio

When git is properly set up, a new *Git* tab will appear in the *Environment... Build* pane. Clicking on this tab will show files and directories that have changed since the last synchronisation with the *GitHub repository* associated with this Project. If you keep this tab open, you will see its contents change each time you save a file, or rebuild an entire Web site.

The *GitHub Push* workflow is:

1. *Stage* any modified file or directory
2. *Commit* selected *Staged* files or directories, adding a brief *Commit message*
3. *Push* all *Committed* files/directories to a *GitHub repository*.

If you're new to git and somewhat intimidated by it, A Visual Introduction to Git gives an amusing introduction to the basic **git** workflow.

For a detailed overview from a developer perspective see Git and GitHub by Hadley Wickham. It particulaly refers to R package development.

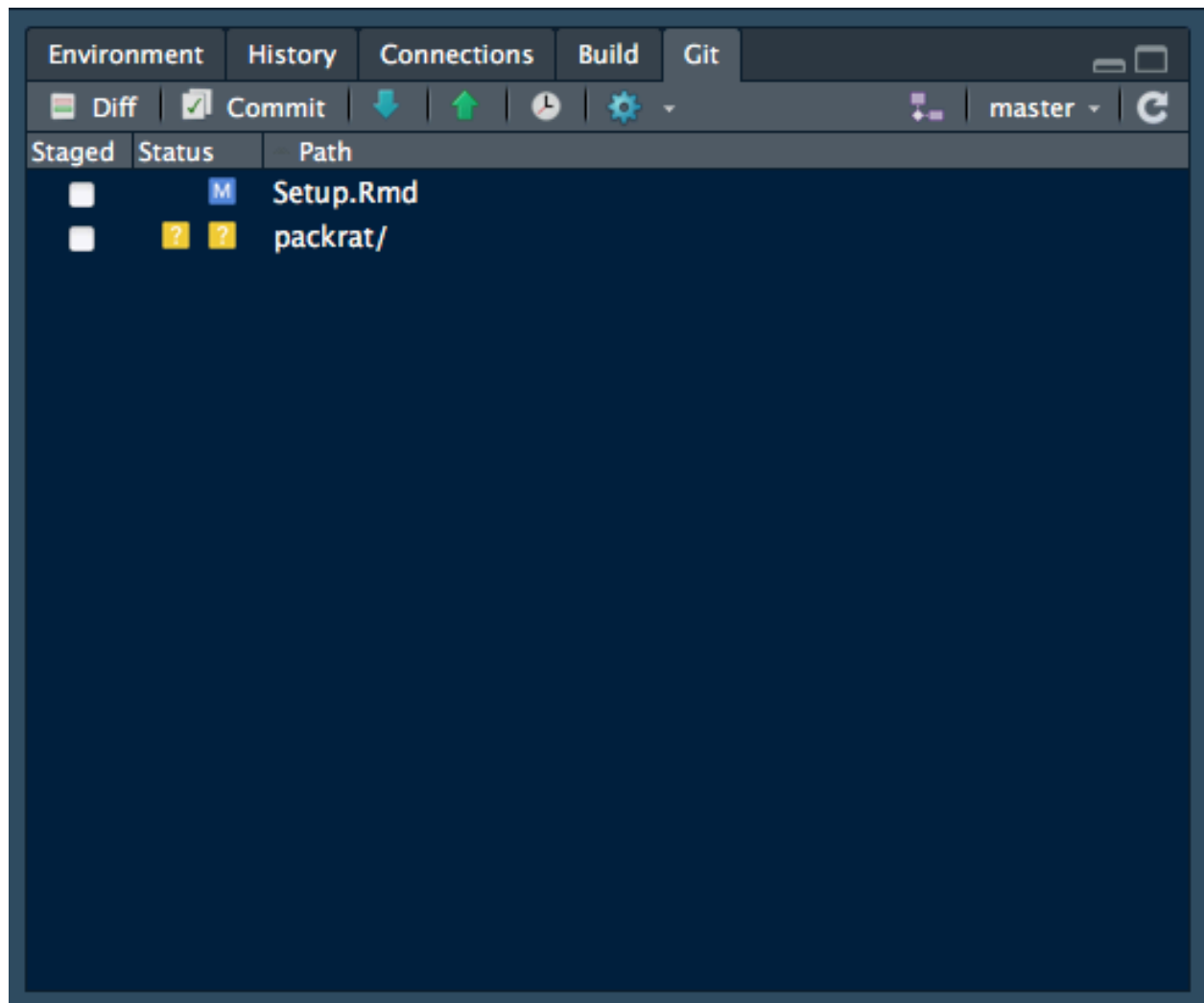


Figure 1.2: ***Above:** The Git pane showing changes during work on this .Rmd file*

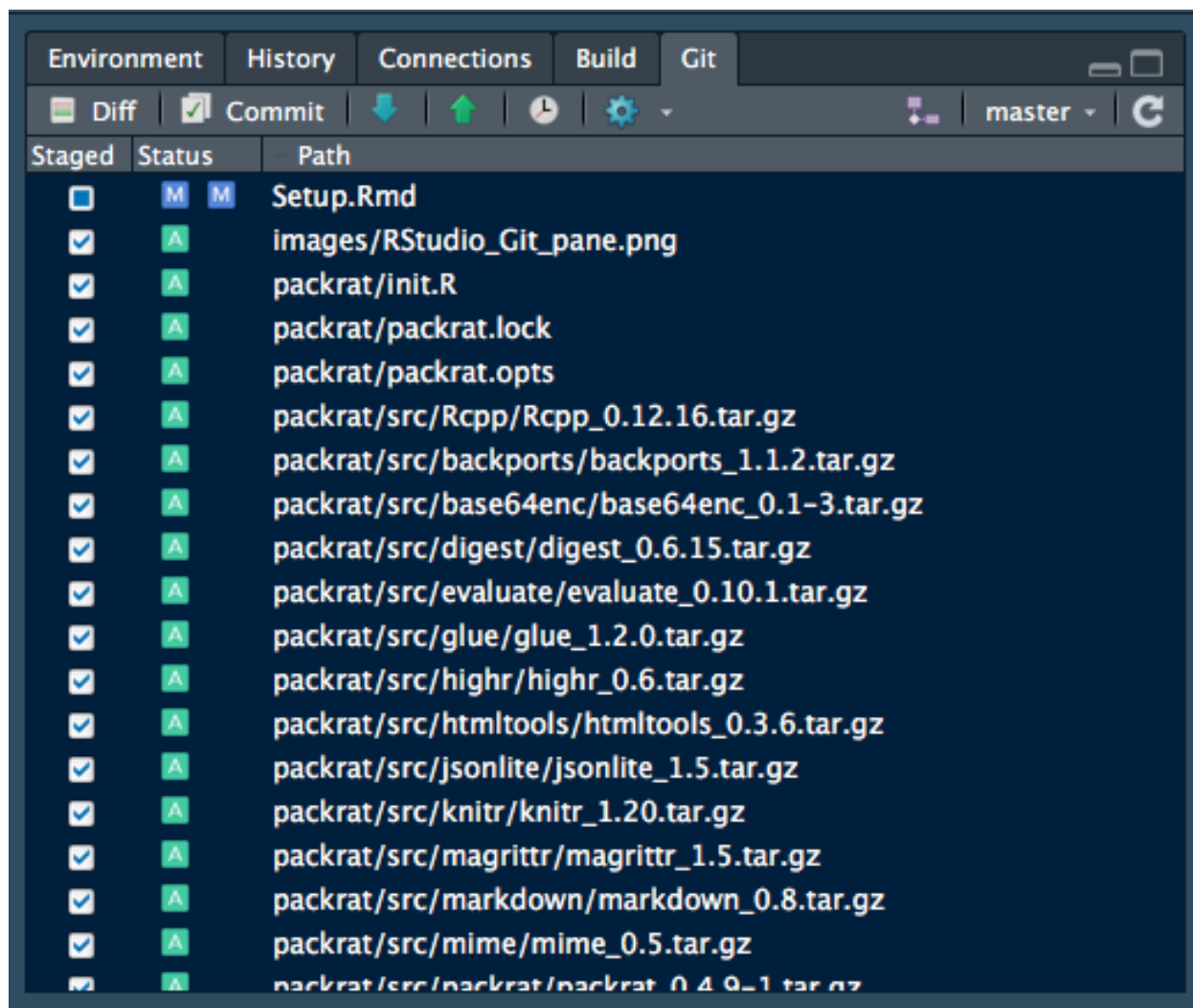





Figure 1.3: *Above:* Staging a directory will result in its contents all being shown Staged.

1.4.2.1 Staging Changed Items

1. Click its **Staged** checkbox ☐. A small blue box  appears in the left *Status* column, showing that the file or directory was *Modified* and ready for a *Commit*. If you had only added the file, a small green box  will appear. The *Staged* checkbox then changes to a tickmark ☒.
2. If you click on a listed directory, its contents will all appear, marked *Staged* and with small green boxes  signifying that they have been *Added*.
3. To **Unstage** added sub-directories or files you either deselect each individually then deselect the *Staged* parent directory (*Tip:* Start with the bottom one and keep clicking!), or SHIFT Click all of them and uncheck one *Selected* radio button.

1.4.2.2 Those little git icons?

In the *Status* column, you'll see little yellow, blue, or red icons. They mean:

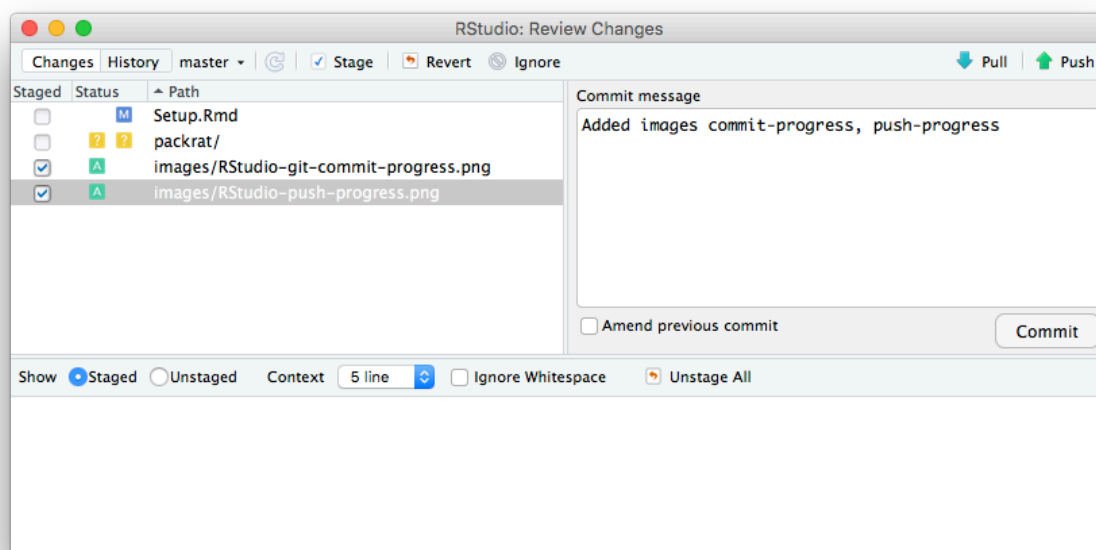






Figure 1.4: **Above:** The RStudio Review Changes window with a Commit message.





-  - New file or directory that does not exist in the `git` Repository
-  - Modified from what is in the `git` Repository
-  - Staging a file or adding it to be committed
-  - The item has been Deleted

The *Status* column has a *left* and *right* icon alignment:

icon on the left - the item has been *Staged*

icon on the right - the item is *Not Staged*

Sometimes you'll see icons in both columns:

-   - *Not in the Repository* and has changed recently
-   - *Staged* and *Modified* since it was *Staged*

In the last case, the *Staged* checkbox will be filled with a solid blue .

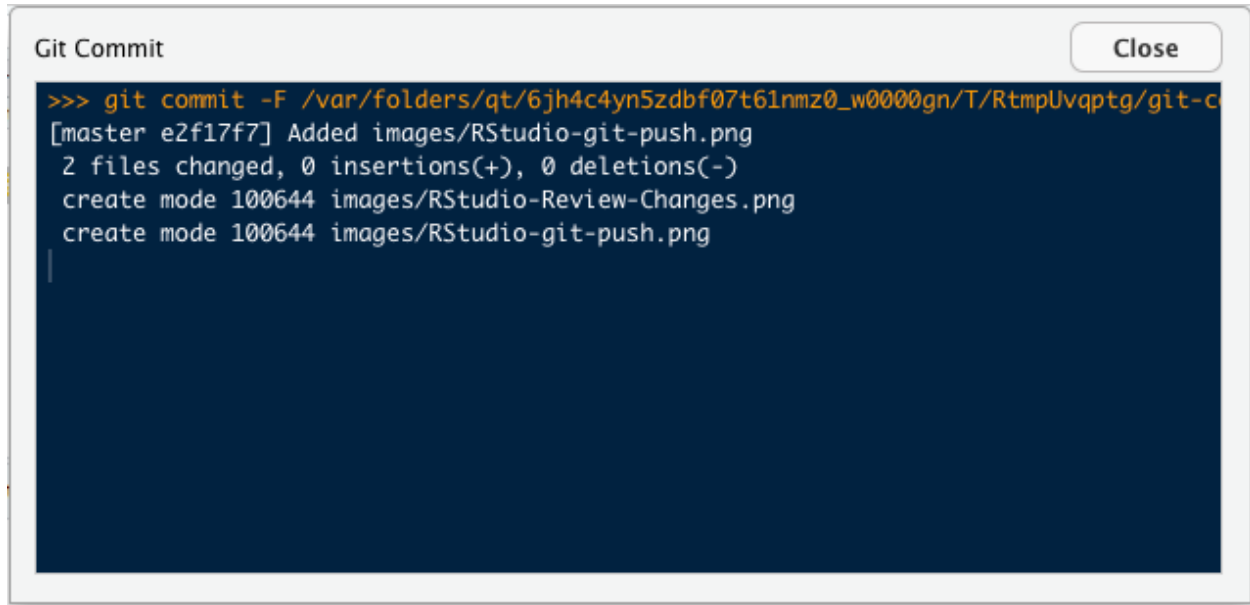
You'll see how it all works as you continue with RStudio.

1.4.2.3 Committing Staged Items

When you have *Staged* everything you want, the next step is to *Commit* those *Staged* items.

4. **Click the *Commit* button to do this.** The *Review Changes* window appears. You can add a message to remind you (and others) what you have done in this *Commit*. This is always advisable, especially for others reviewing your repository.

A smaller *Progress window* will open over the *Review window*. It is actually a small *Terminal window* showing what you would see if you were at the command line.



```
>>> git commit -F /var/folders/qt/6jh4c4yn5zdbf07t61nmz0_w0000gn/T/RtmpUvqptg/git-c
[master e2f17f7] Added images/RStudio-git-push.png
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 images/RStudio-Review-Changes.png
create mode 100644 images/RStudio-git-push.png
```

Figure 1.5: **Above:** The *Progress* window shows the result of your *Commit* command.

1.4.2.4 Pushing To GitHub

5. Your files are now Committed; the next step is to **Push** them to the Repository. The **green up-arrow** is the **Push button**. You may have to close the *Progress window* first.
6. If everything went well, you will see the above message in the *Progress window* (opens automatically if you had closed it).

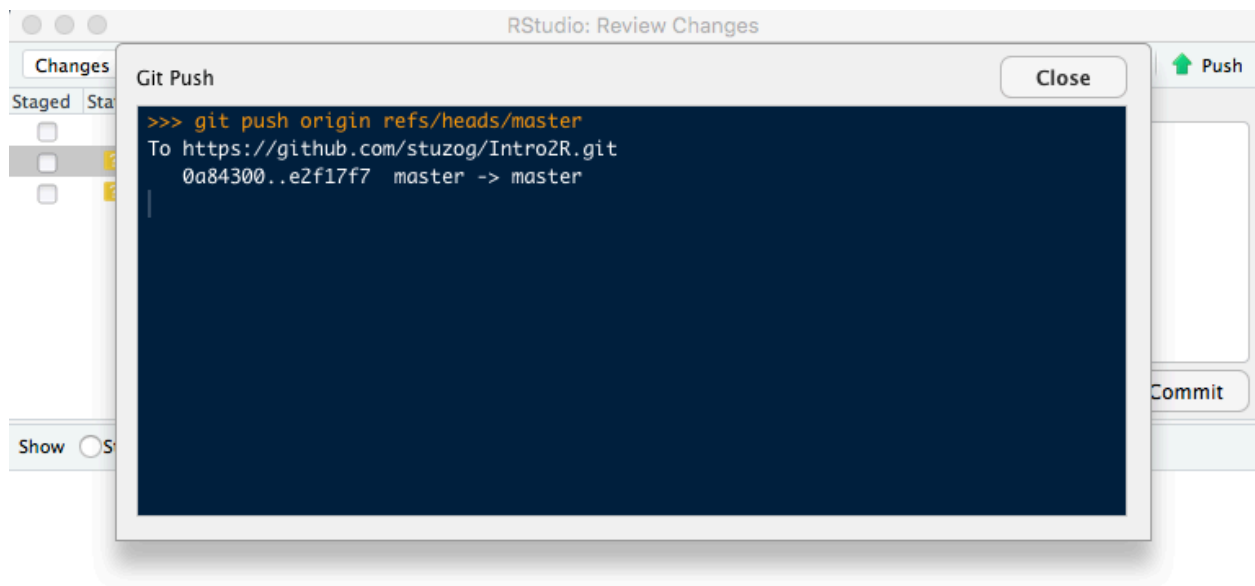


Figure 1.6: **Above:** The Push only happens when you click the green Push button.

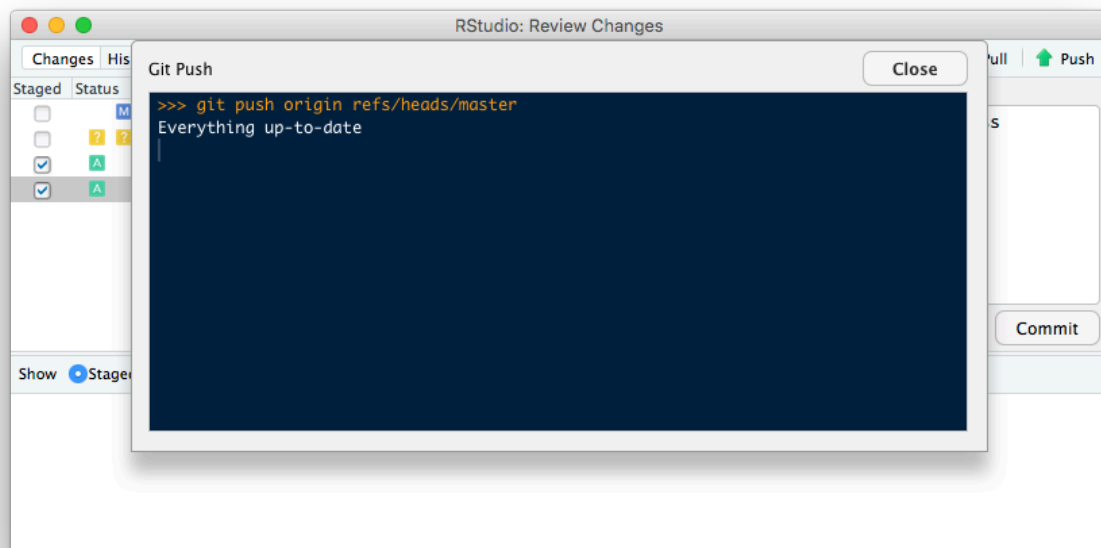


Figure 1.7: **Above:** After a successful Push, a second Push shows everything up-to-date.

Chapter 2

R For Data Science

“In Data Science there’s always a human in the loop: someone is understanding the insight, seeing the figure, or benefitting from the conclusion.” — *David Robinson, Variance Explained Blog*

That person can be called a **Data Scientist**.

2.1 What Is Data Science?

- Data Science is a discipline that uses *statistics, data analysis, machine learning* and their *related methods* to **understand and analyze actual phenomena**.
- It employs techniques and theories of **mathematics, statistics, information science, and computer science**, in particular the subdomains of *machine learning, classification, cluster analysis, uncertainty quantification, computational science, data mining, databases, and visualization*.

According to Wikipedia:

- Data Analysis is a **descriptive process** of inspecting, cleansing, transforming, and modeling data with the goal of **discovering useful information, suggesting conclusions, and supporting decision-making**.
- Data Mining is a particular data analysis technique for **discovering patterns in large data sets**. It focuses on modeling and knowledge discovery for **predictive** rather than purely **descriptive** purposes.
- Data Modeling is a process used to define and analyze the **data requirements needed to support business processes** within **information systems in organizations**.
- Business Intelligence comprises the strategies and technologies used by enterprises for the **data analysis of business information** to provide historical, current and predictive views of **business operations**.

R is used in two associated disciplines, which are often confused as also being Data Science:

- Machine Learning (ML) gives computer systems the ability to **progressively improve performance of a specific task** with data, without being explicitly programmed, and

- Artificial Intelligence (AI), in contrast to the natural intelligence displayed by humans and other animals, is intelligence demonstrated by machines through **Intelligent Agents** that **perceive their environment** and **take actions** to maximize the chance of successfully achieving their designers' intended goals.

Although they overlap, Data Science, Machine Learning, and Artificial Intelligence produce different outcomes:

- **Data Science** produces *insights*
- **Machine Learning** produces *predictions*
- **Artificial Intelligence** produces *actions*

2.2 What Is R?

R is a functional programming language designed to manipulate, display, and analyse statistical data.

R is a **free, case-sensitive, interpreted** language that uses statistical techniques **to perform** Data Analysis, Data Mining, Data Modeling, and Machine Learning, **and to create** Artificial Intelligence and Business Intelligence.

- R is a **functional programming language**, not **procedural**.
- It is **vector-based** and **highly-optimised** to speedily and efficiently manipulate and display all types of **numeric** and **graphical** data.
- R **holds data in memory** for speedy access, so it runs
 - **better on fast processors** with lots of memory for **large datasets**
 - but can be run on an **distributed cluster** running Apache Hadoop, either
 - on a **local network** or on a **Hadoop cloud service**.
- R was **developed from S**, which was **created in 1976** by **John Chambers** of **Bell Labs**.
- It was **written by Ross Ihaka and Robert Gentleman** at the **University of Auckland**, New Zealand.
- The name R comes partly from the names of the two authors, and partly as a play on S.
- R was **conceived in 1992** with an initial version released in **1995**, and a stable beta version in **2000**.
- While there are some important differences, most **S code can run unaltered in R**.
- R is currently being actively developed by the **R Development Core Team**, which includes John Chambers.
- Development of R is supported by the **R Project for Statistical Computing**.
- R is freely available under the **GNU General Public License**. Pre-compiled binary versions are provided for various operating systems. These are available on **local CRAN mirrors**.

The source code for the R software environment is written primarily in **C, Fortran, and R**, highly-optimised for fast computation of large collections of statistical data.

The **popularity of R has increased substantially** in recent years. As of March 2018, R is **in the top 20 of the TIOBE index**, (Python was ranked #4 at that time).

2.3 Installing R

R must be installed on a computer as currently it is **not included** in any operating system. Precompiled versions are available for major systems including **Windows**, **MacOS**, and **Linux**. It may **require compiling** on some Linux distributions.

R is available from the Comprehensive R Archive Network (CRAN), via a multinational network of CRAN Mirrors. **Canadian CRAN mirrors** are hosted at:

- Simon Fraser University
- Manitoba Unix User Group
- University of Toronto
- Dalhousie University

Precompiled versions are available for:

- Download for Linux (SFU)
- Download for OS X (SFU)
- Download for Windows (SFU)

2.4 Command Line R

R can be run from the command line in a Terminal window:

- **which R** – find the location of the R executable
- **R --version** – show the currently installed version of R
- **R** – start an interactive R session

Version information is shown (as above) and the R command prompt **>** is available.

- **quit()** – quit an R session**

2.4.1 Help and Manuals

- Help is available in the usual way with **R --help**
- Downloadable manuals for R are available in **HTML**, **PDF**, and **EPUB** formats.
- The **LaTeX** or **Texinfo** sources of the latest version of these documents are contained in every R source distribution, in the subdirectory **doc/manual** of the extracted archive.
- The **HTML versions of the manuals** are also part of most R installations, accessible by entering the function **help.start()** at the R prompt to launch a page in your default Web browser with links to local R documentation.
- **Manuals for older R versions** are available in the archives of the R sources.

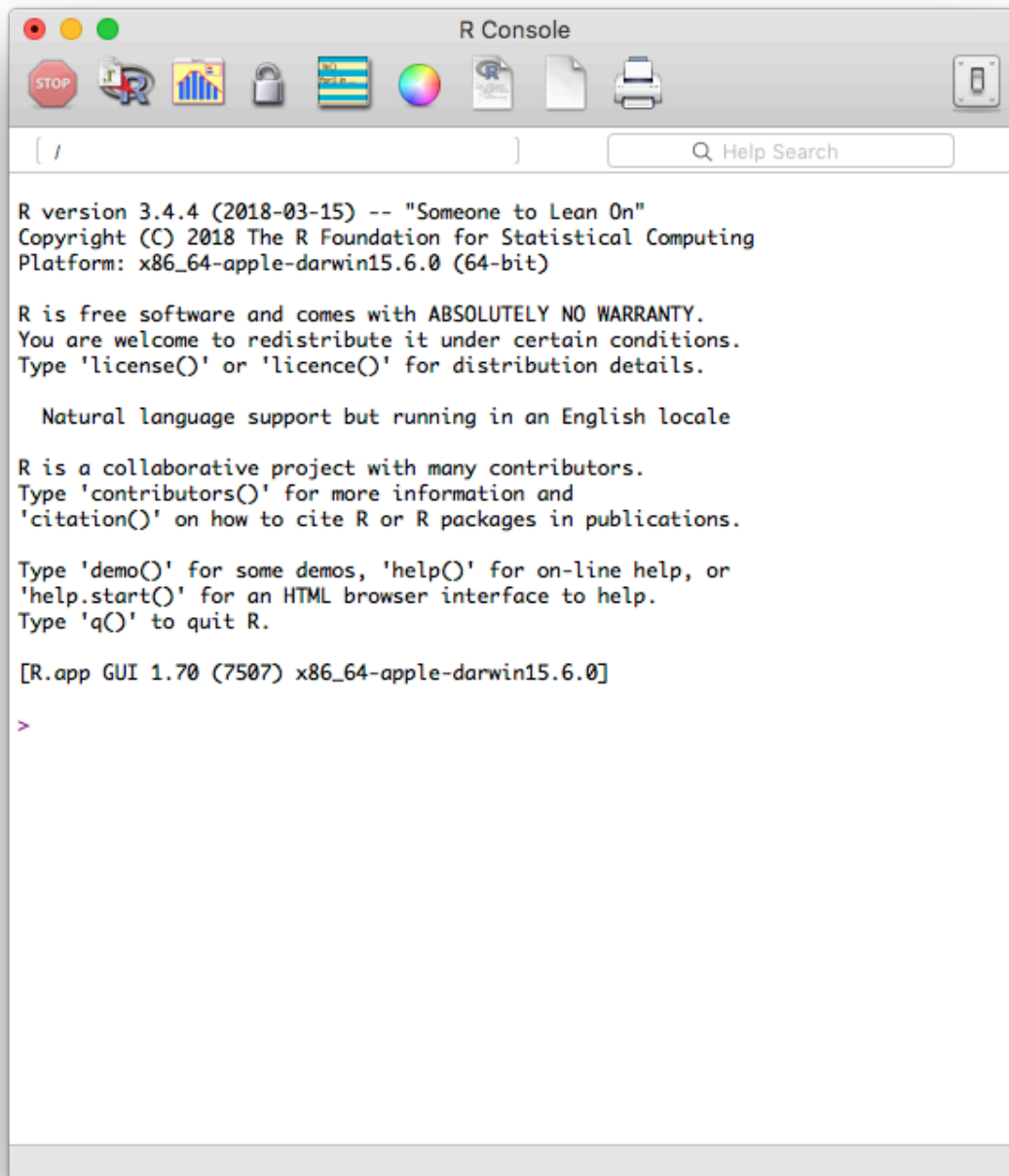


Figure 2.1: **Above:** The R Console startup screen in OS X 10.11.6 El Capitan

2.5 The R Console Interface

While R has a command line interface, a graphical interface is installed as an executable application **R.app** or **R.exe** for **Windows**, **MacOS**, and **Linux**. With minor platform differences, they are similar to this startup screen on **OS X**:

R commands can be **run from the command prompt** in R Console, similar to using the **command line interface** in a terminal program.

2.5.1 Access To R Help

Tooltip help appears on hovering the cursor over the **icons in the Toolbar** along the top of the Console window. **Access to R Help files** is invoked by entering `help.start()` at the command prompt. This will launch the **R Help server** on your computer and open a **comprehensive Help page** in your Web browser.

```
> help.start()
starting httpd help server ... done
If the browser launched by '/usr/bin/open' is already running, it is
    *not* restarted, and you must switch to its window.
Otherwise, be patient ...
>
```

2.5.2 Data File Management

An R installation includes a treasure-trove of **statistical data files**. These can be seen using the **Data Manager**, available under the *Packages & Data* menu. Clicking on a dataset will display its **documentation in the lower pane** of the R Data Manager.

2.5.3 Package Management

R Console has a built-in Package Management system that can be invoked from the **Packages & Data** menu. The top pane of the **R Package Manager** window lists all installed packages. Clicking on a package brings up its **Documentation** in the lower window.

2.6 R Packages

2.6.1 CRAN Package Repository

The **Comprehensive R Archive Network** (CRAN) is a network of **ftp** and Web servers around the world that store identical, up-to-date versions of code and documentation for R. Using the nearest CRAN Mirror minimises network load.

Official releases of R source code (Unix and Windows) are available on CRAN. The 2018-03-15 release is R-3.4.4 *Someone to Lean On*. Older releases are available.

R documentation is available on CRAN in HTML, PDF, and EPUB formats. These were created on Debian Linux and may differ for Mac or Windows, but most parts will be identical. The applicable manual is included in the R installation for each OS.

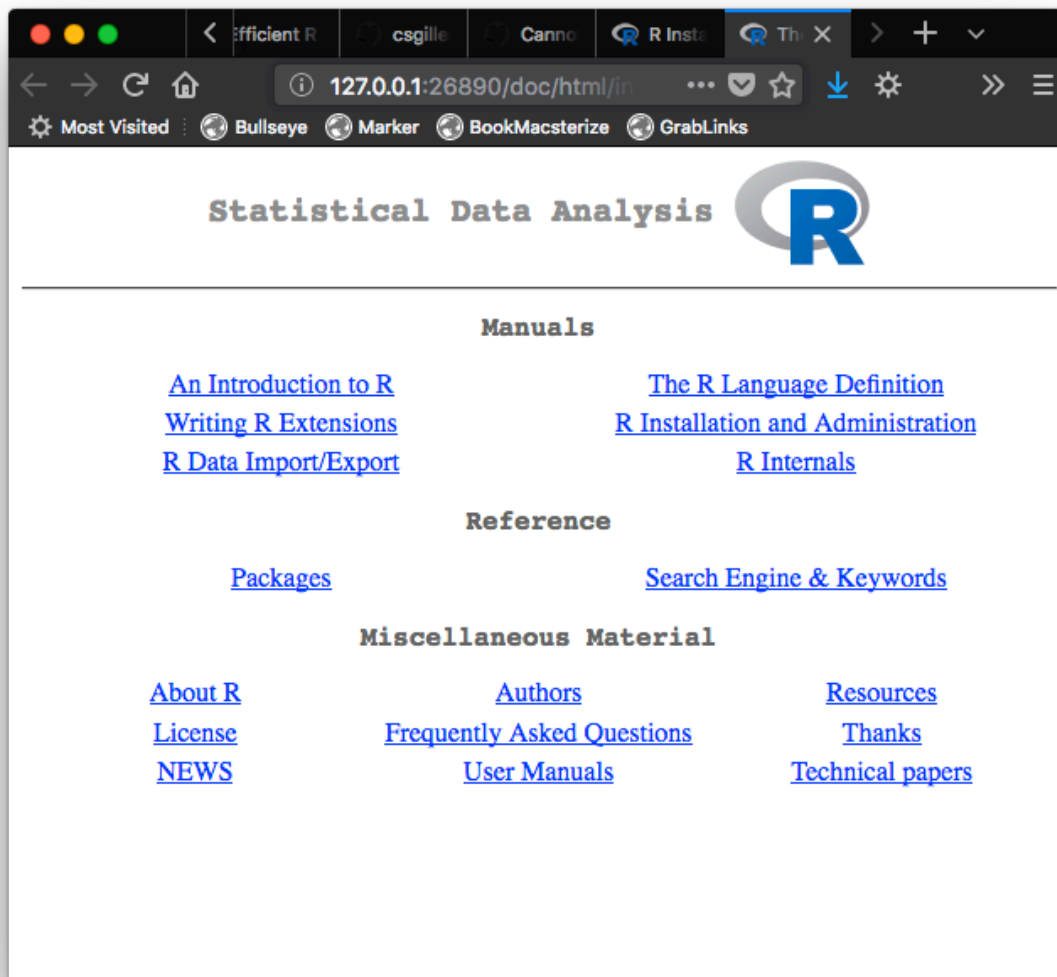


Figure 2.2: **Above:** The R Language Help page opens in your default Web browser

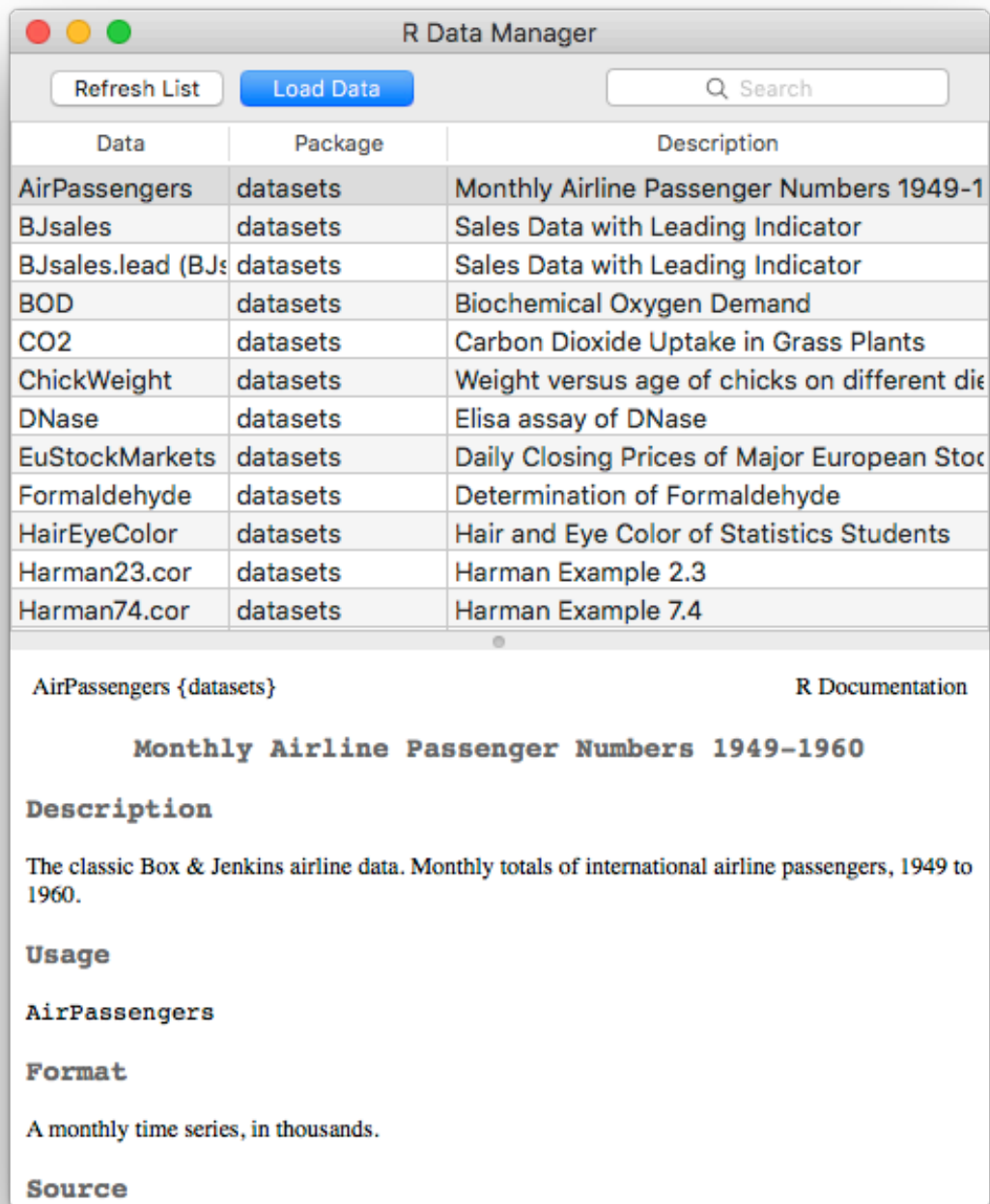


Figure 2.3: ***Above:** The Data Manager lists the extensive statistical datasets included with R.*

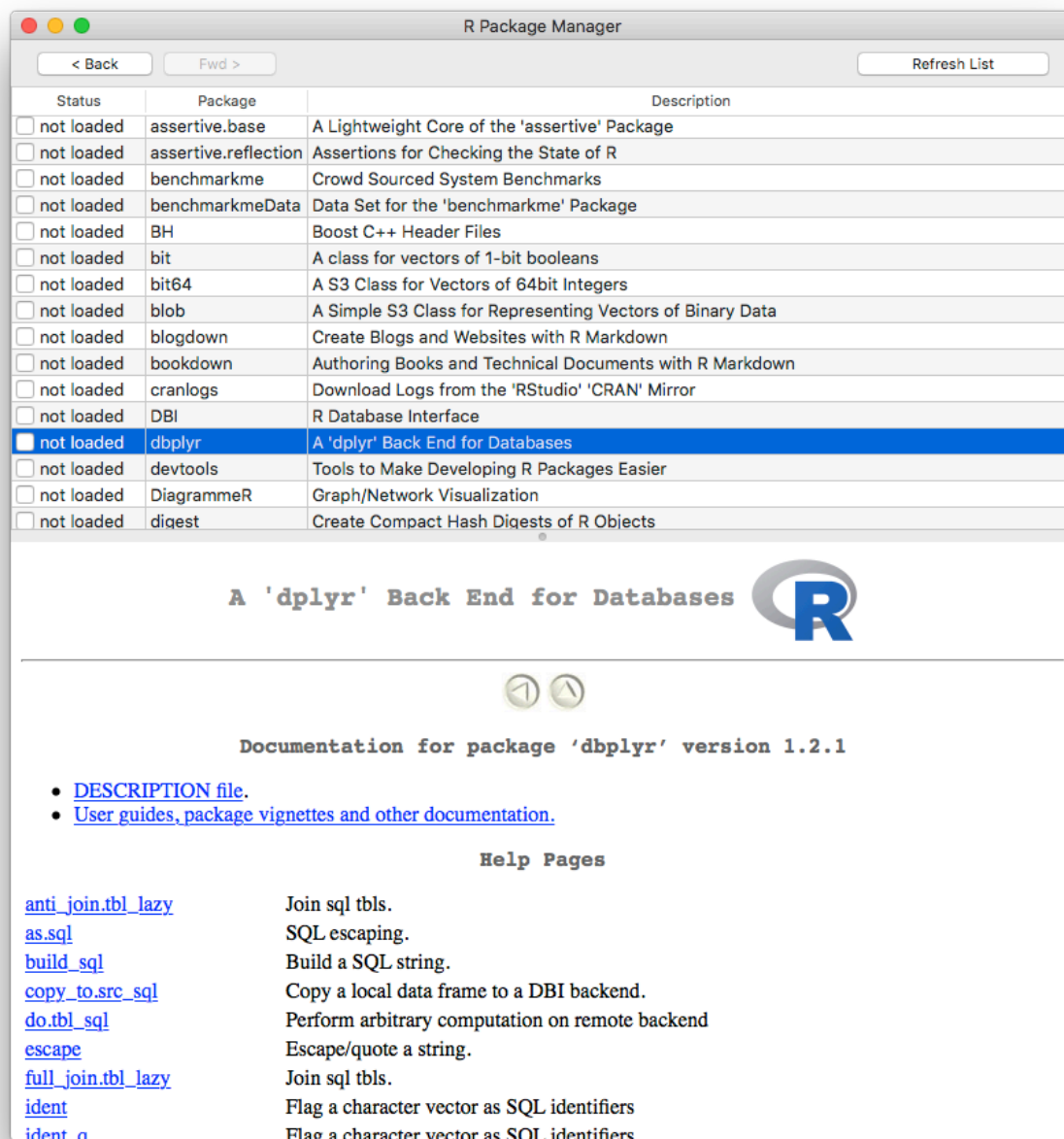


Figure 2.4: **Above:** The Package Manager window lists installed R packages and documentation.

CRAN maintains tables of available R packages sorted by date of publication and name. To assist searching for appropriate packages, CRAN offers discipline-based Task Views. On April 6 2018, the CRAN package repository contained 12,411 packages.

All submitted packages are tested regularly on Debian GNU/Linux, Fedora, OS X, Solaris and Windows. The results are summarized in the check summary.

2.6.2 Installing CRAN R Packages

The R command `help("INSTALL")` or `help("install.packages")` displays information on how to install packages from CRAN. The general package installation command is:

To download and install a package – `install.packages("PackageName")`

To make its library available – `library(PackageName)`

2.6.3 Installing Packages From GitHub

However, not all packages are available on CRAN, but must be downloaded from GitHub. To enable this the `devtools` package must be downloaded and made available:

```
install.packages("devtools")
library(devtools)
```

- `devtools'` `install_github("DeveloperName/PackageName")` installs R packages from GitHub, but it requires the developers GitHub name.
- `githubinstall's` `githubinstall("PackageName")` does not require the developers GitHub name. It also provides some helpful information on gitHub-hosted R packages hosted. This package can be installed from CRAN.

If a repository package requires compiling the appropriate compilers, usually for *C++* or *FORTRAN*, must be already installed on your system.

A more detailed discussion for installing from GitHub is available [here](#).

2.6.4 The tidyverse

The tidyverse is a collection of R packages designed to clean up datasets for analysis. The *tidyverse philosophy* has become an important tool of Data Science. tidyverse packages share the same design philosophy, grammar, and structure. Ease of adoption and use are fundamental principles for all packages of the tidyverse.

Core tidyverse packages:

- `ggplot2` for data visualisation
- `dplyr` for data manipulation
- `tidyr` for data tidying
- `readr` for data import
- `purrr` for functional programming
- `tibble` for tibbles, a modern re-imagining of data frames

Two other packages are considered part of tidyverse core:

- `stringr` for working with strings
- `forcats` for resolving factors

tidyverse packages sometimes conflict with other packages. You'll see this displayed on installing a tidyverse package. You can:

- **list conflicts with other installed R packages at any time**
with `tidyverse_conflicts()`, and
- **Check that all tidyverse packages are up-to-date**
with `tidyverse_update()`.

Chapter 12 of the seminal book *R For Data Science* explains in detail how to use tidyverse packages to clean up data prior to analysis. Learning how to use the tidyverse is the recommended first step to mastering R programming for Data Science.

2.6.4.1 `ggplot2`

A system for declaratively creating graphics based on The Grammar of Graphics, `ggplot2` was created by Hadley Wickham. You provide the data; tell `ggplot2` how to map variables to aesthetics and what graphical primitives to use; it takes care of the details.

2.6.4.2 `dplyr`

A grammar of data manipulation, `dplyr` provides a consistent set of verbs to solve most common data manipulation challenges. It abstracts how the data is stored, so it works as well with remote databases as with local data frames, using exactly the same R code.

2.6.4.3 `tidyr`

The goal of `tidyr` is to create tidy data. Tidy data describes a standard way of storing data so it can be used anywhere in the tidyverse. If you ensure that your data is tidy, you'll spend less time fighting with the tools and more time working on analysis.

2.6.4.4 `readr`

Provides a fast and friendly way to read rectangular data — comma-separated, tab-separated, fixed-width, and general-delimited files, as well as tables separated by white space, and web log files. `readr` flexibly parses data while failing cleanly if the data changes.

2.6.4.5 `purrr`

A complete toolset for working with functions and vectors, `purrr` functions offer many advantages over the equivalents in base R. For example, the `purrr` family of `map()` functions replace many `for` loops with code that is both more succinct and easier to read.

2.6.4.6 `tibble`

A modern reimagining of the `data.frame`, *tibbles* keep what has proven to be effective. *tibbles* are both lazy and surly: they do less (they don't change variable names or types, or do partial matching), and they complain more (for example when a variable does not exist).

2.6.4.7 stringr

Designed to make working with strings as easy as possible. stringr focusses on the most important and commonly-used string manipulation functions. It is built on top of stringi, which provides a comprehensive set covering almost anything you can imagine.

2.6.4.8 forcats

A suite of tools to solve problems with factors. R uses factors to handle categorical variables that have a fixed and known set of possible values. Many base R functions convert character vectors to factors, but tidyverse packages don't create them automatically.

Chapter 3

RStudio IDE

is a **free, open-source, professionally-designed IDE for programming in R** (and other languages including Python and associated Python packages).

RStudio is available in two free versions:

- RStudio Desktop, run locally as a desktop application. and
- RStudio Server, which runs on a networked or remote Linux server.

RStudio Server Pro is a full-service, paid edition for business and government use.

The advantages of using RStudio include:

- Free and open source for non-commercial use
- Easy to install on Windows, Mac, and Linux
- Well-designed, multi-pane GUI interface
- Paid, full-service, professional versions
- Git and Subversion versioning support
- Can import data from online sources
- Outputs to HTML, PDF, Word files
- Built-in R package management
- Keyboard shortcuts display
- Active community support
- Tab key code completion
- Ongoing development
- Well-documented
- Code folding
- It's So Cool
- So is R!

From the RStudio web Site:

An IDE that was built just for R

- Syntax highlighting, code completion, and smart indentation
- Execute R code directly from the source editor
- Quickly jump to function definitions

Bring your workflow together

- Integrated R help and documentation
- Easily manage multiple working directories using projects
- Workspace browser and data viewer

Powerful authoring & Debugging

- Interactive debugger to diagnose and fix errors quickly
- Extensive package development tools
- Authoring with Sweave and R Markdown

3.1 The RStudio Interface

The RStudio Console interface is flexible, rich, and capable:

- Tabs in each pane display different aspects of the current project
- Panes can share a column or be raised to fill an entire column
- File and Viewer panes can be zoomed into a larger new window
- Toolbars in each tab allow direct control of content elements
- Console commands are integrated into the running OS and files
- Console display can be changed by selecting a different theme
- Code syntax colors can be changed with another editor theme

3.2 RStudio Projects

RStudio advises that **all data files and packages** belonging to a project be contained in a dedicated directory as an **RStudio Project** with an executable **.Rproj file**.

Project management is built into RStudio. Start a new Project either from the *File -> New Project* main menu, or from the drop-down **Project menu** at the top right of the RStudio window (left).

New Project... opens a dialogue window that enables a Project to be created in a *New* or *Existing* directory, or from a *Version Control Repository*.

If an existing Project has unsaved changes, you will be asked to save them.

Open Project... brings up a File dialogue to locate an existing project directory and double-click its **.Rproj** file.

Open Project in New Session... allows you to create a new Project in a new R session without closing the existing Project R session.

Close Project will close the running R session and after saving any changes.

Existing RStudio Projects are listed below these to facilitate switching between active projects. Entries are created when a project is started by RStudio. This list can be emptied by selecting *Clear Project List*.

Project Options lets you set options specific only to the running Project.

3.3 Package Management In RStudio

Package management is an integral part of the RStudio interface. The *Packages* tab in the Files/Viewer pane shows all locally-installed and system packages.

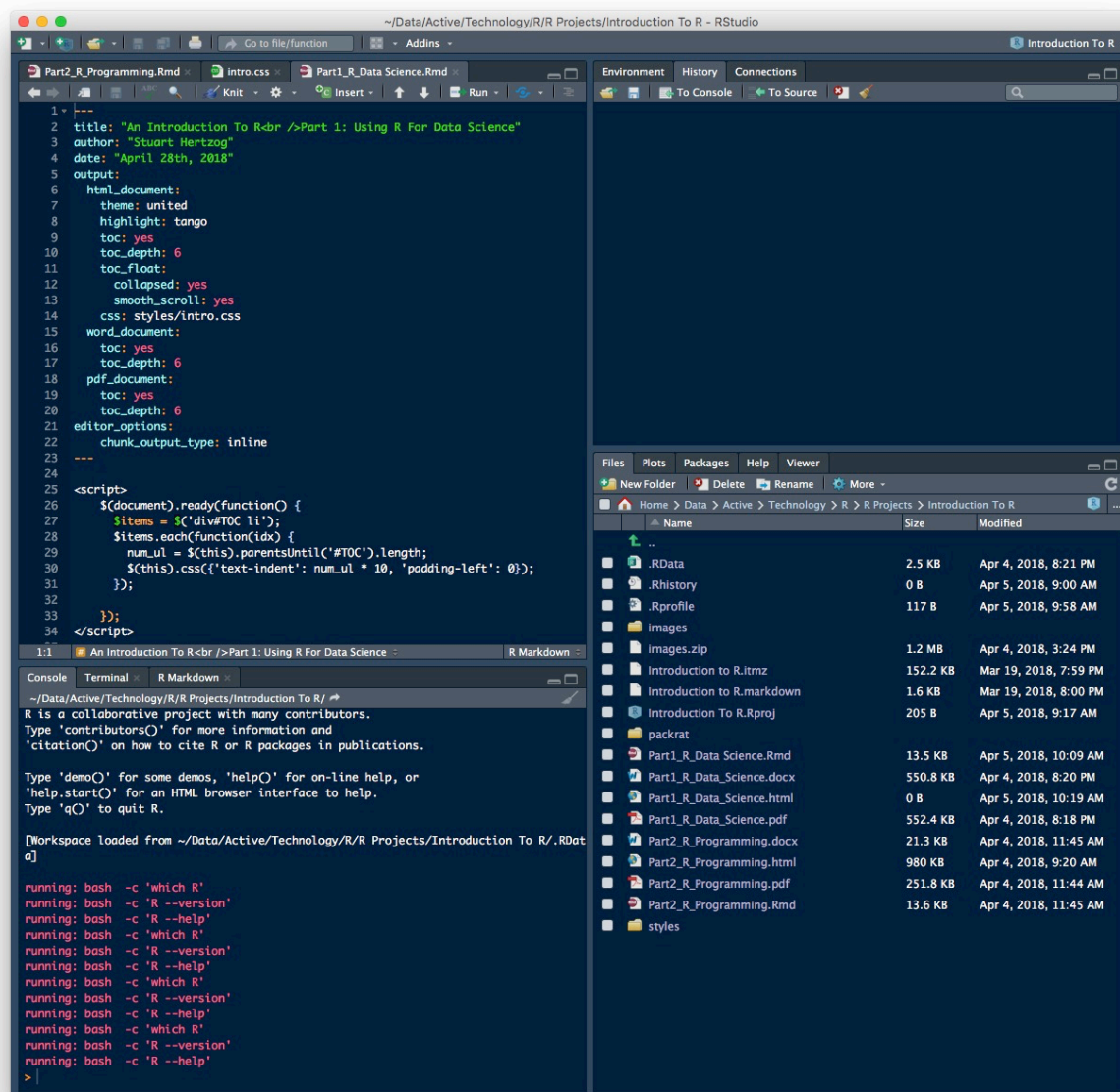


Figure 3.1: *Above: The four-pane default RStudio Console. Pane arrangement can be changed.*



Clicking the checkbox to the left of each package listing will load a package's library into memory and make its functions available. The corresponding installation command will appear immediately in the

Console plus any warning messages.

Packages with checkmarks have already been loaded. Unclicking a loaded package will unload it unless it is required by RMarkdown, in which case a warning message will inform you of that restriction and the package will remain loaded.

There are two ways of installing packages in the RStudio interface:

- Using the ***Install Packages GUI***
- Entering commands in the **Console**.

The progress of any compilation or installation will be displayed in the Console.

3.3.1 The Install Packages GUI

Clicking ***Install*** on the **Packages** tab toolbar opens an *Install Packages* dialogue (left) that gives the choice of installing a package directly from CRAN or from a local package archive.

Multiple packages can be installed by separating the package names with commas. This installs and makes the packages available for use.

The ***Install to Library*** drop-down menu allows you to chose the Library where you want the package installed. The *default location* is selected: this is usually acceptable.

The location of the R installation on your system can be found by entering `R.home(component = "home")` in the Console. Other component values for the paths to specific directories of the current R installation are "bin", "doc", "etc", "include", library, "modules" and "share".

Clicking the ***Install dependencies*** checkbox will also download and install any other packages required for the requested package(s). This is recommended.

NOTE: This option will not work when installing from a local archive. In this case the dependencies must be installed first, or the installation will fail.

###Installing From The Console

Packages can be installed and loaded from the Console. The command sequence is:

```
install.packages("PackageName")
library(PackageName)
```

The delimiters " " or ' ' must be used to install a package.

Most R packages are hosted on CRAN, but some are only available on GitHub or Bitbucket. To install from these locations, the `devtools` package first must be installed, as described above.

3.3.2 Project Package Management

RStudio projects are built with different versions of R using the then-current versions of any installed packages. As both R and R packages are constantly being updated, this can cause problems when developing packages or RMarkdown documents.

There are four solutions to project package management:

- Installing the `packrat` package to write package versions in a `packrat.lock` text file and store the packages in a project `packrat` directory;
- Using MRAN (Microsoft R Application Network) to determine dependencies based on the “snapshot” of CRAN that Microsoft has stored on a given day;

- Installing the `checkpoint` package to choose package versions based on a given day in MRAN history.

For more information, see Package Management for Reproducible R Code

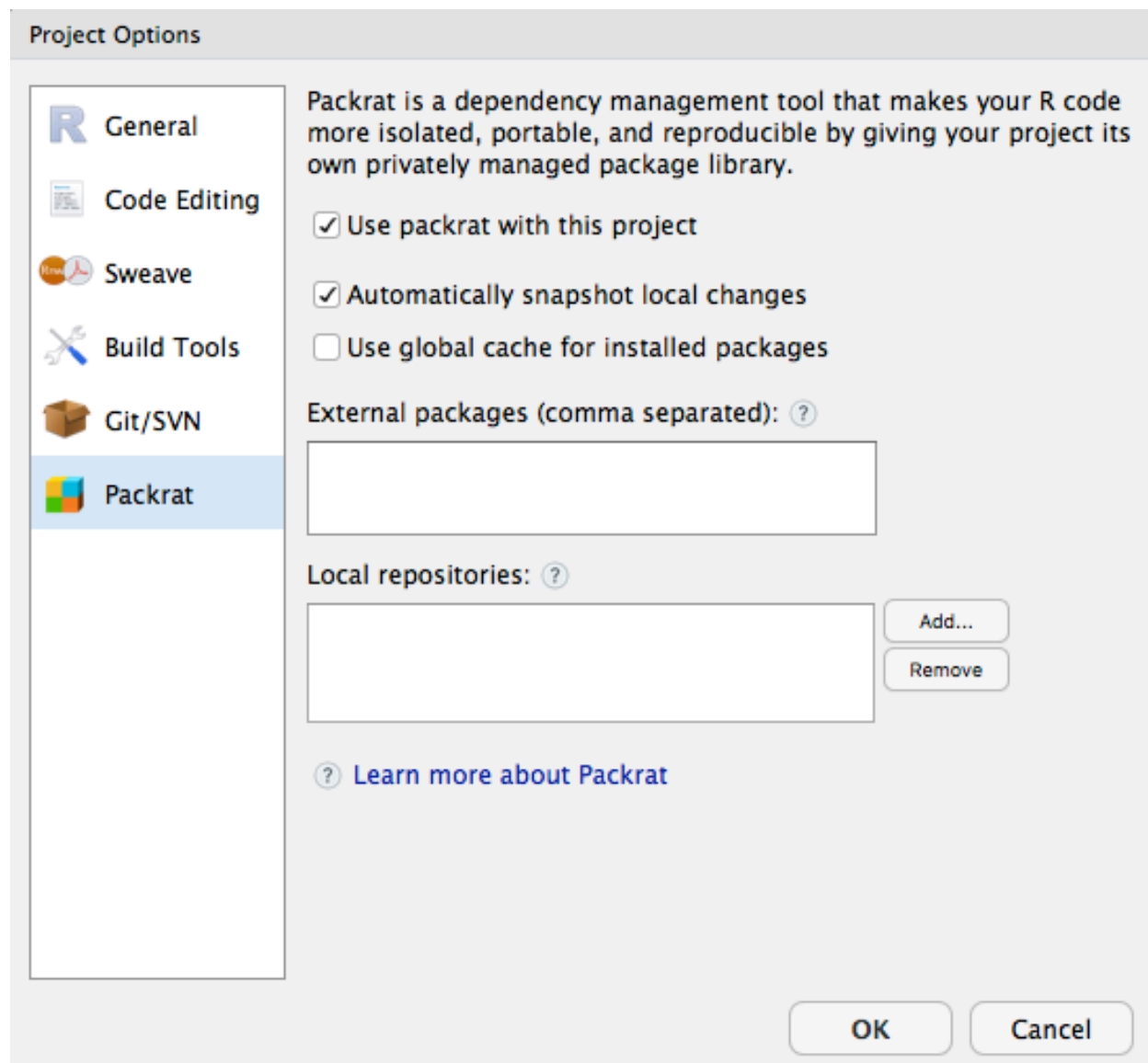
3.3.3 Installing And Using packrat

Packrat stores all added packages in its own directory library, rather than relying on the user R library that is shared across all R sessions.

The advantages of this are:

- **Isolation:** Installing a new or updated package for one project does not break other projects, and vice versa;
- **Portability:** Enables moving projects from one computer to another, even across different platforms. (NOTE: this requires installation for each OS as the compiled libraries are not always identical);
- **Reproducibility:** Ensures the correct package versions are assembled where and whenever the project is installed.

packrat is integrated into the RStudio environment so it is easy to install and initiate. The *Packages* tab in the *Files/Viewer* window has a `packrat` item menu. Clicking on this brings up the **Packrat Options** window:



It's advisable to select the *Automatically snapshot local changes* option.

Clicking **OK** will download, install, and initiate **packrat** in exactly the same way the same as issuing the commands from the Console:

```
install.packages("packrat")
packrat::init()
```

There also is an option in the *New Project* window to create a new project with **packrat** pre-installed and initiated.

Once **packrat** is up and running the *Packrat* link will become a drop-down menu (left), containing the following options:

Check Library Status... – shows if any packages require updating

Clean Unused Packages – removes unnecessary packages. If you want to keep a package from appearing in the Unused list, just add a `library(PackageName)` call to any .R file in your Project directory.

Export Project Bundle – creates a zipped bundle of your Project in the `packrat` directory, for transport to another system. The bundle contains application code, a manifest file, an html file, and a `packrat` folder with a `packrat.lock` file.

Packrat Options... – brings up the options window. (See above)

For more information see

- Packrat
- Packrat Commands
- Packrat by example
- Using Packrat with RStudio
- Bundle Downloads

3.4 Using Python In RStudio

3.4.1 reticulate

Until recently it has not been possible to easily use Python in RStudio. The newly-developed `reticulate` package provides a **comprehensive set of tools for interoperability between Python and R** within RStudio.

reticulate includes facilities for:

- **Calling Python from R** in a variety of ways including R Markdown, sourcing Python scripts, importing Python modules, and using Python interactively within an R session.
- **Translation between R and Python objects** (for example, between R and Pandas data frames, or between R matrices and NumPy arrays).
- **Flexible binding to different versions of Python** including virtual environments and Conda environments.

reticulate embeds a Python session within your R session, enabling seamless, high-performance interoperability. If you are a Python developer, or an R developer who uses Python for some of your work, or a member of data science team that uses both languages, `reticulate` can dramatically streamline your workflow.

Chapter 4

The RStudio Ecosystem



The RStudio IDE is the flagship of a comprehensive RStudio ecosystem.

4.1 R Notebooks

R Notebooks are the most basic type of RMarkdown document. They are the RStudio equivalent to Jupyter Notebooks. As described by RStudio:

An R Notebook is an R Markdown document with chunks that can be executed independently and interactively, with output visible immediately beneath the input.

Any R Markdown document can be used as a Notebook, and all R Notebooks can be rendered to other R Markdown document types.

4.1.1 Creating An R Notebook File

The RStudio menu item *File -> New File -> R Notebook* creates a new RStudio Notebook containing instructions on how to use an R Notebook.

```
---  
title: "R Notebook"  
output:  
  html_notebook  
---
```

This is an [R Markdown](<http://rmarkdown.rstudio.com>) Notebook. When you execute code within the notebook

Try executing this chunk by clicking the **Run** button within the chunk or by placing your cursor inside

```

...
{r}
plot(cars)
...

```

Add a new chunk by clicking the **Insert Chunk** button on the toolbar or by pressing **Cmd+Option+I**.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (cli

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike **Knit**, :

For more information see RStudio Notebook.

4.1.2 R Notebook HTML files

An R Notebook HTML file is a special type of R Notebook.

It is an HTML file that uses the `.nb.html` extension and it enables:

- **the source .Rmd document** and
- **chunk outputs** to be recovered.

The `html_notebook` output format is specified in the YAML metadata:

```

title: "my_document_title"
output:
  html_notebook

```

With the `_yaml` file written this way, the command `rmarkdown::render()` will create an `.nb.html` R Notebook HTML Format file.

For more information, see Details of the R Notebook HTML Format.

Note: The RStudio instructions show an incorrect YAML header. You must use the `_yaml` format shown above or RStudio will throw an error and not compile.

4.2 RMarkdown

R Markdown documents are fully reproducible. They use a productive notebook interface to weave together narrative text and code to produce elegantly-formatted output from multiple languages including R, Python, and SQL.— *RStudio RMarkdown*

RMarkdown documents are the default RStudio document. They are plain-text files with the `.Rmd` extension and can be used to generate many different types of output. The main ones are:

- HTML documents – `.html`
- PDF Documents – `.pdf`
- Word Documents – `.docx`

In addition to the standard output formats (above), RMarkdown documents can generate output in many other formats, via the pandoc conversion engine.

Beamer, HTML5 slides, Tufte-style handouts, books, dashboards, interactive documents, scientific articles, and websites.

RStudio maintains a gallery of examples of the many uses of RMarkdown.

4.2.1 knitr

RMarkdown uses the knitr report generation package to output to the different file formats.

The **knitr** package enables integration of R code into LaTeX, LyX, HTML, Markdown, AsciiDoc, and reStructuredText documents. It can execute code chunks in a variety of languages via knitr Language Engines, including for:

- R
- Python
- SQL
- Bash
- Rcpp
- Stan
- JavaScript
- CSS

To process a code chunk using an alternate language engine simply use the name of the engine in place of `{r}` in the chunk declaration, for example:

```
```bash
cat flights1.csv flights2.csv flights3.csv > flights.csv
```
```

4.2.1.1 Code Chunk Options

RMarkdown code chunk processing can be controlled by including *Chunk Options* in the *Chunk Declaration*, for example:

`{r, include = FALSE}` prevents code and results from appearing.

R Markdown still runs the code and the results can be used by other chunks.

`{r, eval = FALSE}` prevents to code from running but it still appears. `{r, echo = FALSE}` prevents code but not the results from appearing.

This is a useful way to embed figures.

`{r, message = FALSE}` prevents messages generated by code from appearing.

`{r, warning = FALSE}` prevents warnings generated by code from appearing.

`{r, fig.cap = "..."} adds a caption to graphical results.`

The R Markdown Reference Guide (.pdf) has a list of useful options. Yihui's **knitr** options has full documentation of all **knitr** code processing controls.

4.2.2 pandoc

The RStudio installation includes pandoc, which is used for conversion between RMarkdown and the various output formats. You may have to install pandoc on your system for conversion to take place, especially for .pdf output, which goes through an intermediate LaTeX stage.

pandoc can also be run from the command line if it is installed separately. This is explained in detail in the pandoc manual.

4.3 Shiny

Shiny is an R package that makes it easy to build interactive web apps straight from R. You can host standalone apps on a webpage or embed them in R Markdown documents or build dashboards. You can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions. — *RStudio Shiny*

RStudio's Shiny is a Web application framework for R that takes static RMarkdown documents and Web sites to the next level.

There is no limit to building online documentation, interactive tutorials, and statistical data interfaces with these powerful R packages:

- **htmlwidgets** enables JavaScript visualisations and interactive components to be incorporated into R Markdown documents and Shiny web applications, as shown in this showcase and this gallery
- **crosstalk** wires widgets together, including linked brushing between widgets and client side filtering
- **shinydashboard** makes it easy to create all kinds of Web dashboards, as shown in these examples

and others:

- **intrval** can be used to build for example, a Web slider

We will explore building Shiny apps in a future Tutorial.

4.3.1 Shiny Server

Shiny Server makes your Shiny Web apps accessible on a local network, an Enterprise server, or on the Internet. There are two flavours of server:

- Shiny Server Open Source and
- Shiny Server Pro

Documentation is available for:

- Installing Shiny Server Open Source
- Shiny Server Pro Admin Guide
- shinyapps.io User Guide

Shiny Apps can be available to an Intranet or to the Internet via:

- An Enterprise Server
- RStudio Connect
- shinyapps.io

The difference between the latter two is explained here.

4.4 bookdown

bookdown is an open-source R package that makes it really easy to create online books and technical documents using RMarkdown.

bookdown books open a new avenue for Web-enabled book publishing.

They are essentially **responsive RMarkdown Web sites**, complete with content navigation, plus the ability to display and process computer code using a wide range of programming languages, including interactive Shiny dashboards and Web apps.

bookdown has added a few important missing features related to writing books, such as figure and table caption numbering and cross-references, and the ability to embed HTML widgets or Shiny apps.

Developed and maintained by RStudio software engineer Yihui Xie, **bookdown** books can be published on bookdown.org, a free service provided by RStudio Inc. Books are available for download, and the author holds full copyright.

Fully-documented in — of course! — a **bookdown** book called *bookdown*, the **bookdown** package uses many of the same conventions as other RMarkdown document types, so conversion of any RStudio project to an online book is relatively straightforward.

A Minimal Book Example can be cloned from GitHub. It can also be downloaded as a .zip file if you are not familiar with GitHub.

4.4.1 bookdown Installation

Installation requires the devtools R package to be first installed, after which **bookdown** can be installed from its GitHub repository:

```
install.packages("devtools")
devtools::install_github("rstudio/bookdown")
```

4.4.2 tinytex

PDF output requires some version of the LaTeX typesetting system to be installed. LaTeX is not a stand-alone typesetting program in itself, but document preparation software that runs on top of Donald E. Knuth's TeX typesetting system.

TeX distributions usually bundle together all the parts needed for a working TeX system. LaTeX and many of the packages built on it form an important component of any modern TeX distribution.

As this is a large package consisting of many gigabytes of files that users may not wish to install, Yihui Xie wrote **tinytex**, a lightweight, cross-platform, portable, and easy-to-maintain LaTeX distribution based on TeX Live.

4.4.2.1 tinytex Installation

Installation of **tinytex** is accomplished with:

```
install.packages(c('tinytex', 'rmarkdown'))
tinytex::install_tinytex()
```

PDF output (including Beamer slides) requires a full TeX installation.

4.5 blogdown

Just as WordPress changed the Web by bringing blogging to the masses, blogdown has opened a path for the more computer-savvy to create quick-loading, static blog sites using RMarkdown.

Not content with authoring `bookdown`, RStudio developer Yihui Xie took the next logical step by writing the `blogdown` R package.

Based on the Hugo website framework, `blogdown` lets you write posts in RStudio as RMarkdown.Rmd files. These are automatically converted to blog posts for uploading to a hosting service, including GitHub pages.

4.5.1 Learning blogdown

It is advisable to first read Yihui Xie and Alison Presmanes Hill's `bookdown` book *Creating Websites with R Markdown* to become familiar with `bookdown` concepts and the Hugo framework before you embark on creating your first blog site.

These links offer practical Setup hints and guidance:

4.5.1.1 blogdown Setup

- Up and running with blogdown | Alison Presmanes Hill
- Getting Started With Blogdown | Borasky Research Journal
- R Blogdown Setup in GitHub (2) | Tales of R
- A Technical Introduction to Yihui's personal website
- A not-so-technical introduction to Daijiang's personal website

4.5.1.2 blogdown Links

- `blogdown`: Creating Websites with R Markdown
- Announcing `blogdown` - Yihui Xie, RStudio 2017-09-11
- GitHub - rstudio/blogdown: Create Blogs and Websites with R Markdown
- GitHub - rbind/blogdown-demo: A minimal website example using `blogdown`
- Twitter hashtag `#blogdown`

4.5.2 Hugo Themes

`blogdown` uses **Hugo themes to style the look and feel of its blogs**. The default theme for `blogdown` is *hugo-lithium*, a minimal theme that offers several options, detailed in Section 2.4.1 of the `blogdown` book. The base theme can be modified to suit your taste, or it can be replaced by another theme.

The Hugo web site showcases many user-contributed themes. These can be examined online and downloaded for use as-is or as a starting point for themeing.

4.5.2.1 Hugo Links

- Hugo — world's fastest (?) framework for building websites
- Hugo Setup Guide — applies to `blogdown` blog sites
- Hugo Themeing — how to choose and fine-tune a blog
- Contributed Hugo Themes – hundreds of excellent Hugo themes
- Converting Old Wordpress Posts To Hugo — goodbye WordPress!

4.5.3 blogdown Blogs

A list of R-related `blogdown` blogs is kept at `rbind` on Github. R-blogs made with `blogdown` can be viewed [here](#).

An interesting and uncluttered example is Simply Statistics, a statistics blog by Rafa Irizarry, Roger Peng, and Jeff Leek.

4.6 RStudio Products



Some RStudio products are open-source and free for student and personal use. Others are priced for professional or Enterprise deployment.

4.6.1 Software

- R Notebooks — equivalent to Jupyter Notebooks
- RMarkdown — creates `.Rmd` files using Pandoc Markdown
- R Packages — available on CRAN and GitHub
- RStudio Desktop — the RStudio IDE, also on GitHub
- RStudio Server — free for personal and local Network use
- Shiny — build interactive Web pages and JavaScript apps
- Shiny Server — free for local network and Web access behind a firewall

4.6.2 Cloud Services

- RStudio Server Pro – enhanced admin, Enterprise security and pricing
- RStudio Connect – team management, Enterprise security and pricing
- RStudio Cloud (in alpha) — Do, Share, Teach, and Learn Data Science
- Shiny Server Pro – share Shiny apps, Enterprise security and pricing
- shinyapps.io – deploy Shiny apps, enhanced security, scalable
- Bookdown Book Publishing — free for `bookdown` static page books

4.6.3 Resources

- RStudio Cheat Sheets — PDF format
- RStudio Community — register for full Forum access
- RStudio Documentation — Help articles for all RStudio products
- RStudio Server> Getting Started — lists many links for setup and use
- RStudio Server Pro Admin. Guide — web-accessible User Manual

Chapter 5

The Fundamentals of R

5.1 Four Fundamentals

The essence of R:

```
R <- c(1:4)
R
```

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

(See Vectors later).

[One] important difference about R:

- **Vector-based:** R is not a procedural language

[Two] reasons to use R for Data Science:

- **Designed for data:** R can manipulate big data sets
- **Graphics Are Graspable:** people understand graphical data

[Three] fundamental principles of R per John Chambers:

- **Objects:** Everything that exists in R is an object
- **Functions:** Everything that happens in R is a function call
- **Interfaces:** to other softwares are an integral part of R

[Four] ways of programming R:

- **Command line:** entering R commands in a terminal
- **Source file:** running a set of commands from a saved file
- **R GUI interface:** available for Mac, Windows, and Linux
- **Code chunks in RStudio:** allows debugging as you write

5.2 Basic Maths

R has all the basic mathematical functions:

```
1 + 1
```

```
## [1] 2
```

```
1 + 2 + 3
```

```
## [1] 6
```

```
3 * 7 * 2
```

```
## [1] 42
```

```
4 / 3
```

```
## [1] 1.333333
```

R obeys the standard order of mathematical operations (**PEMDAS**):

1. **P**arentheses ()
2. **E**xponents ^
3. **M**ultiplication x
4. **D**ivision
5. **A**ddition +
6. **S**ubtraction -

```
(2 ^ 5) + (2 * 5)
```

```
## [1] 42
```

The use of white space between operators is recommended.

5.3 Variables

Unlike statically-typed languages such as C++, R does not require variable types to be declared. An R variable can represent any data type or R object, such as a function, result, or graphical plot. R variables can be redeclared.

- Variable names can **contain alphanumeric characters**
 - but not **periods .** or **underscores _**
- They **cannot start with a number or underscore**
- Variable names are **case sensitive**

5.3.1 Assigning variables

R variable assignment operators are <- (default) and = (acceptable).

```
x <- 2
x
```

```
## [1] 2
```

```
y = 5
y
```

```
## [1] 5
```

You can also assign left-to-right with `->`, but variables are not often assigned that way.

```
7 -> z
z
```

```
## [1] 7
```

Assignment operations can be used successively to assign a value to multiple variables

```
a <- b <- 42
a
```

```
## [1] 42
```

```
b
```

```
## [1] 42
```

You can also use the built-in `assign` function:

```
assign("q", 4)
q
```

```
## [1] 4
```

5.3.2 Removing variables

`rm(variablename)` removes a variable.

```
rm(q)
```

5.4 Data Types

R has four main data types:

- Numeric
- Character (a.k.a Nominal)

- Date
- Logical

You can check the type of variable with `class(variablename)`

```
x <- "eh?"  
x
```

```
## [1] "eh?"
```

```
class(x)
```

```
## [1] "character"
```

```
y <- 99  
y
```

```
## [1] 99
```

```
class(y)
```

```
## [1] "numeric"
```

5.4.1 Numeric data types

Numeric data includes both integers and decimals — positive, negative, and zero — similar to `float` or `double` in other languages. A numeric value stored in a variable is automatically assumed to be numeric in R.

You can test whether data is numeric with `is.numeric()`:

```
is.numeric(y)
```

```
## [1] TRUE
```

And if it's an integer with `is.integer()`:

```
is.integer(y)
```

```
## [1] FALSE
```

The response of `FALSE` is because to set an integer as a variable you must append the value with `L`:

```
y <- 99L  
is.integer(y)
```

```
## [1] TRUE
```

R promotes `integers` to `numeric` when needed.

5.4.2 Character data types

R handles Character data in two primary ways: as `character` and as `factor`. They are treated differently:

```
x <- "data"
x
```

```
## [1] "data"
```

```
class(x)
```

```
## [1] "character"
```

and

```
y <- factor("data")
y
```

```
## [1] data
## Levels: data
```

The `levels` are attributes of that factor.

To find the length of a `character` (or `numeric`):

```
nchar(x)
```

```
## [1] 4
```

This does not work for `factor` data.

5.4.3 Date data types

R has numerous types of dates. `Date` and `POSIXct` are the most useful.

```
date1 <- as.Date("2018-03-28")
date1
```

```
## [1] "2018-03-28"
```

```
class(date1)
```

```
## [1] "Date"
```

```
as.numeric(date1)
```

```
## [1] 17618
```

and

```
date2 <- as.POSIXct("2018-03-28 10:45")
date2
```

```
## [1] "2018-03-28 10:45:00 PDT"
```

```
class(date2)
```

```
## [1] "POSIXct" "POSIXt"
```

```
as.numeric(date2)
```

```
## [1] 1522259100
```

Using `as.numeric` also changes the underlying type:

```
class(date1)
```

```
## [1] "Date"
```

```
class(as.numeric(date1))
```

```
## [1] "numeric"
```

5.4.4 Logical data types

Logicals can be either `TRUE` (T or 1) or `FALSE` (F or 0). `T` and `F` are not recommended as they are simply shortcuts to `TRUE` and `FALSE` and can be overwritten, causing woe, anguish, mayhem, and rioting. (`TRUE` or `F`?)

Logical data types have a similar test function `is.logical()`:

```
k <- TRUE
class(k)
```

```
## [1] "logical"
```

```
is.logical(k)
```

```
## [1] TRUE
```

5.5 Data Structures

R data structures are containers for data elements:

- **Vectors** – collections of only *same-type elements*
- **Matrices** – rectangular containers of only *same-type elements*
- **Data Frames** – contain *many types of vectors*, all of the same length
- **Arrays** – Vectors with *dimensions* for each *same-type element*
- **Lists** – containers for elements of *multi-type data types*

5.5.1 Vectors

Vectors are the heart of R; it is a **vectorised language**. An R Vector is:

A collection of elements of the same type.

Operations are applied to each element of a vector without the need to loop through them. This separates R from other programming languages and makes it most suited to manipulation and graphical presentation of data.

Vectors do not have a dimension: there is no column or row vector. Unlike mathematical vectors there is no difference between column or row orientation.

5.5.1.1 Creating a vector

Vectors are created with `c`, meaning “combine”:

```
x <- c(1, 2, 3, 4, 5, 6, 7, 8)
x
```

```
## [1] 1 2 3 4 5 6 7 8
```

Operations are applied to all elements at once:

```
x + 2
```

```
## [1] 3 4 5 6 7 8 9 10
```

```
x - 3
```

```
## [1] -2 -1 0 1 2 3 4 5
```

```
x * 2
```

```
## [1] 2 4 6 8 10 12 14 16
```

```
x / 4
```

```
## [1] 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
```

```
x^2
```

```
## [1] 1 4 9 16 25 36 49 64
```

```
sqrt(x)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
```

5.5.1.2 Vector creation shortcuts

```
1:8
```

```
## [1] 1 2 3 4 5 6 7 8
```

```
8:1
```

```
## [1] 8 7 6 5 4 3 2 1
```

```
-3:4
```

```
## [1] -3 -2 -1 0 1 2 3 4
```

```
4:-3
```

```
## [1] 4 3 2 1 0 -1 -2 -3
```

5.5.1.3 Accessing vector elements

Any element of a **Vector** can be directly access using [square brackets] to point to it:

```
x
```

```
## [1] 1 2 3 4 5 6 7 8
```

```
x[4]
```

```
## [1] 4
```

```
x[8]
```

```
## [1] 8
```

5.5.1.4 Counting within Vectors

You can check the length of a vector:

```
x
```

```
## [1] 1 2 3 4 5 6 7 8
```

```
length(x)
```

```
## [1] 8
```



```
y
```

```
## [1] data
## Levels: data
```

```
length(y)
```

```
## [1] 1
```

```
length(x + y)
```

```
## Warning in Ops.factor(x, y): '+' not meaningful for factors
```

```
## [1] 8
```

and count the number of characters in a vector:

```
q <- c("One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight")
q
```

```
## [1] "One"  "Two"  "Three" "Four"  "Five"  "Six"  "Seven" "Eight"
```

```
nchar(q)
```

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

5.5.1.5 Combining Vectors

Two vectors of the same or different length can be combined:

5.5.1.5.1 Vectors of the same length

```
x <- 1:8
x
```

```
## [1] 1 2 3 4 5 6 7 8
```

```
y <- -3:4
y
```

```
## [1] -3 -2 -1 0 1 2 3 4
```

```
x + y
```

```
## [1] -2 0 2 4 6 8 10 12
```

```
x - y
```

```
## [1] 4 4 4 4 4 4 4 4
```

```
x * y
```

```
## [1] -3 -4 -3 0 5 12 21 32
```

```
x / y
```

```
## [1] -0.3333333 -1.0000000 -3.0000000      Inf  5.0000000  3.0000000
## [7]  2.3333333  2.0000000
```

```
x^y
```

```
## [1]  1.0000000  0.2500000  0.3333333  1.0000000  5.0000000
## [6] 36.0000000 343.0000000 4096.0000000
```

5.5.1.5.2 Vectors of different lengths

For two vectors of different lengths, the shorter vector is recycled, and R may issue a warning:

```
x + c(1, 2)
```

```
## [1] 2 4 4 6 6 8 8 10
```

```
x + c(1, 2, 3)
```

```
## Warning in x + c(1, 2, 3): longer object length is not a multiple of
## shorter object length
```

```
## [1] 2 4 6 5 7 9 8 10
```

5.5.1.6 Comparison of two Vectors

```
x <- c(1:8)
x
```

```
## [1] 1 2 3 4 5 6 7 8
```

```
x > 5
```

```
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

```
y <- c(3:10)
y
```

```
## [1] 3 4 5 6 7 8 9 10
```

```
x > y
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

The `all()` function tests whether all elements are TRUE

```
x <- 10:1
y <- -4:5
x
```

```
## [1] 10 9 8 7 6 5 4 3 2 1
```

```
y
```

```
## [1] -4 -3 -2 -1 0 1 2 3 4 5
```

```
all(x < y)
```

```
## [1] FALSE
```

The `any()` function tests if any element is 'TRUE':

```
any(x < y)
```

```
## [1] TRUE
```

including vectors, matrices, data frames (similar to datasets), and lists (collections of objects).

5.5.1.7 Factor Vectors

Factors are an important concept in R. **Factors contain levels**, which are the unique values of that factor variable.

```
q
```

```
## [1] "One" "Two" "Three" "Four" "Five" "Six" "Seven" "Eight"
```

```
qFactor <- as.factor(q)
qFactor
```

```
## [1] One Two Three Four Five Six Seven Eight
## Levels: Eight Five Four One Seven Six Three Two
```

Note that the order of `levels` does not matter unless the `ordered` argument is set TRUE:

```
factor(x=c("High School", "Doctorate", "Masters", "College"),
      levels=c("High School", "College", "Masters", "Doctorate"),
      ordered=TRUE)
```

```
## [1] High School Doctorate Masters College
## Levels: High School < College < Masters < Doctorate
```

5.5.2 Matrices

A familiar mathematical structure, **matrices** are essential to statistics.

A **Matrix** is a rectangular structure of rows and columns in which every element is of the same type, often all numerics.

Matrices can be acted upon similarly to **Vectors**, with **PEDMAS**-style element-by-element addition, subtraction, division, and equality.

5.5.2.1 Creating a Matrix

```
A <- matrix(1:12, nrow=3)
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Any element of a **matrix** can be directly accessed using [square bracket] co-ordinates:

```
A[2,3]
```

```
## [1] 8
```

```
A[3,4]
```

```
## [1] 12
```

5.5.2.2 Dimensions of a Matrix

```
nrow(A)
```

```
## [1] 3
```

```
ncol(A)
```

```
## [1] 4
```

```
dim(A)
```

```
## [1] 3 4
```

5.5.2.3 Adding Matrices

```
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
B <- matrix(13:24, nrow=3)
```

```
B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
```

```
A + B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   14   20   26   32
## [2,]   16   22   28   34
## [3,]   18   24   30   36
```

5.5.2.4 Multiplying Matrices

```
A * B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   13   64  133  220
## [2,]   28   85  160  253
## [3,]   45  108  189  288
```

5.5.2.5 Logical querying

```
A == B
```

```
##      [,1] [,2] [,3] [,4]
## [1,] FALSE FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE FALSE
## [3,] FALSE FALSE FALSE FALSE
```

5.5.2.6 Naming rows and columns

```
colnames(A) <- c("A1", "A2", "A3", "A4")
rownames(A) <- c("First", "Second", "Third")
A
```

```
##           A1 A2 A3 A4
## First      1  4  7 10
## Second     2  5  8 11
## Third      3  6  9 12
```

```
A["First", "A2"]
```

```
## [1] 4
```

```
A[1,2]
```

```
## [1] 4
```

Two special vectors – `letters` and `LETTERS` – create lowercase and UPPERCASE letter named matrix columns or rows:

```
C <- matrix(21:40, nrow=2)
colnames(C) <- LETTERS[1:10]
rownames(C) <- c(letters[1:2])
C
```

```
##    A  B  C  D  E  F  G  H  I  J
## a 21 23 25 27 29 31 33 35 37 39
## b 22 24 26 28 30 32 34 36 38 40
```

5.5.3 Dataframes

The `data.frame` is perhaps the primary reason for R's growing popularity as a powerful, focussed, and flexible language for use in all aspects of Data Science.

A `data.frame` is a rectangular collection of vectors, all of which are of the same length but differing data types.

A **Data Frame** looks like an **Excel spreadsheet** in that the data is organised into **columns** and **rows**. In statistical terms, each column is a *variable* while each row contains specific *observations*. Similar to a Matrix only in that it is also rectangular, a `data.frame` is a much more flexible and comprehensive data structure.

5.5.3.1 Creating a Dataframe

Using the existing functions:

```
(x <- 8:1)

## [1] 8 7 6 5 4 3 2 1

(y <- -3:4)

## [1] -3 -2 -1 0 1 2 3 4

(q <- c("One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight"))

## [1] "One" "Two" "Three" "Four" "Five" "Six" "Seven" "Eight"
```

The simplest way of creating a `Dataframe` is with the `data.frame()` function:

```
theDF <- data.frame(x, y, q)
theDF

##   x y   q
## 1 8 -3 One
## 2 7 -2 Two
## 3 6 -1 Three
## 4 5 0 Four
## 5 4 1 Five
## 6 3 2 Six
## 7 2 3 Seven
## 8 1 4 Eight
```

This creates an 8x3 `data.frame` consisting of three `vectors`. Notice that the data types are included below the column headings.

To assign names to the `vectors`:

```
theDF <- data.frame(First=x, Second=y, Third=q)
theDF

##   First Second Third
## 1     8     -3   One
## 2     7     -2   Two
## 3     6     -1 Three
## 4     5      0  Four
## 5     4      1  Five
## 6     3      2   Six
## 7     2      3 Seven
## 8     1      4 Eight
```

To assign names to the rows:

```
rownames(theDF) <- c("One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight")
theDF
```

```
##      First Second Third
## One      8     -3   One
## Two      7     -2   Two
## Three    6     -1 Three
## Four     5      0   Four
## Five     4      1   Five
## Six      3      2   Six
## Seven    2      3 Seven
## Eight    1      4 Eight
```

5.5.3.2 Examining a Dataframe

The `nrow()`, `ncol()`, `dim()`, `rownames()`, and `names()` functions are available to investigate its properties:

```
(nrow(theDF))
```

```
## [1] 8
```

```
(ncol(theDF))
```

```
## [1] 3
```

```
(dim(theDF))
```

```
## [1] 8 3
```

```
(rownames(theDF))
```

```
## [1] "One" "Two" "Three" "Four" "Five" "Six" "Seven" "Eight"
```

```
(names(theDF))
```

```
## [1] "First" "Second" "Third"
```

Elements of any vector of a `data.frame` can be directly accessed using the `$` or `[row, col]` operators:

```
(theDF$Second)
```

```
## [1] -3 -2 -1 0 1 2 3 4
```

```
(theDF[7, 3])
```

```
## [1] Seven
```

```
## Levels: Eight Five Four One Seven Six Three Two
```

To specify an entire row, leave out the column specification, *vice versa* for specifying an entire column:


```
(theDF[2, ])
```

```
##      First Second Third
## Two      7      -2   Two
```

```
(theDF[, 2])
```

```
## [1] -3 -2 -1  0  1  2  3  4
```

To specify more than one row or column, use a **vector** of indices:

```
(theDF[3:5, 2:3])
```

```
##      Second Third
## Three      -1 Three
## Four        0 Four
## Five         1 Five
```

To specify multiple columns by name, use a **character vector** of the column names:

```
(theDF[, c("First", "Third")])
```

```
##      First Third
## One       8    One
## Two       7    Two
## Three     6 Three
## Four      5   Four
## Five      4   Five
## Six       3    Six
## Seven     2 Seven
## Eight     1 Eight
```

To find the **class** of the entire **data.frame**:

```
(class(theDF))
```

```
## [1] "data.frame"
```

or the **class** of any vector:

```
(class(theDF$Third))
```

```
## [1] "factor"
```

5.5.3.3 Displaying a Dataframe

data.frames can be small, large, big, huge, or ginormous, depending on their size. The **head()** and **tail()** functions print only the first or last few rows, or the number of rows you set:

```
(head(theDF))
```

```
##      First Second Third
## One      8      -3   One
## Two      7      -2   Two
## Three    6      -1 Three
## Four     5       0   Four
## Five     4       1   Five
## Six      3       2   Six
```

```
(head(theDF, n=5))
```

```
##      First Second Third
## One      8      -3   One
## Two      7      -2   Two
## Three    6      -1 Three
## Four     5       0   Four
## Five     4       1   Five
```

```
(tail(theDF, n=5))
```

```
##      First Second Third
## Four     5       0   Four
## Five     4       1   Five
## Six      3       2   Six
## Seven    2       3 Seven
## Eight    1       4 Eight
```

5.5.4 Arrays

An Array is a multidimensional Vector whose elements are all the same type, but which also have attributes having dimensions (`dim`) that can also be named (`dimnames`).

5.5.4.1 Creating Arrays

To create an Array, the first element is the row index, the second the column index, and the remaining elements are for the outer dimensions `row`, `column`, `number of arrays`:

```
theArray <- array(1:12, dim = c(2, 3, 2))
theArray
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

5.5.4.2 Accessing Arrays

Individual elements of an **Array** are accessed using square brackets similar to a **Vector** but in this case by [row, column, array #].

```
theArray[1, , ]
```

```
##      [,1] [,2]
## [1,]    1    7
## [2,]    3    9
## [3,]    5   11
```

```
theArray[2, , ]
```

```
##      [,1] [,2]
## [1,]    2    8
## [2,]    4   10
## [3,]    6   12
```

```
theArray[1, , 1]
```

```
## [1] 1 3 5
```

```
theArray[1, , 2]
```

```
## [1] 7 9 11
```

```
theArray[, , 1]
```

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

```
theArray[, , 2]
```

```
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

5.5.5 Lists

Lists are used to store any number of items of any type: all **numeric** or all **character** vectors, or a mix of them; complete **data.frames**; and even other **lists**.

5.5.5.1 Creating Lists

Lists are created with the `list()` function. Each argument to the function becomes an element of the list:

```
list(1, 2, 3)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

Single-element lists can contain multi-element vectors:

```
list(c(1, 2, 3))
```

```
## [[1]]
## [1] 1 2 3
```

Here's a two-element list with the second element a five-element vector:

```
list1 <- list(c(1, 2, 3), 3:7)
list1
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 3 4 5 6 7
```

A two-element list with the first element an **array**, the second element a ten-element **vector**:

```
list2 <- list(theArray, 1:10)
list2
```

```
## [[1]]
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
##
## [[2]]
## [1] 1 2 3 4 5 6 7 8 9 10
```

5.5.5.2 Creating Empty Lists

Empty lists of a determined length are created using a `vector`:

```
(emptyList <- vector(mode = "list", length = 4))
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
```

Note: Enclosing an expression in round brackets displays the results immediately after execution.

5.5.5.3 Naming Lists

Lists can have names, and each element of a `list` can have a unique name

```
names(list2)
```

```
## NULL
```

```
(names(list2) <- c("The Array", "The Vector"))
```

```
## [1] "The Array" "The Vector"
```

```
list2
```

```
## $`The Array`
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
##
## $`The Vector`
## [1] 1 2 3 4 5 6 7 8 9 10
```

5.5.5.4 Naming List Elements

Names can also be assigned to `list` elements during creation using name-value pairs. This can also include naming the `list` itself:

```
(list3 <- list(theARR=theArray, theVECT=1:10, List3=list2))
```

```
## $theARR
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
##
## $theVECT
## [1]  1  2  3  4  5  6  7  8  9 10
##
## $List3
## $List3$`The Array`
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
##
## $List3$`The Vector`
## [1]  1  2  3  4  5  6  7  8  9 10
```

5.5.5.5 Adding To A List

New elements can be added to a `list` by appending a `numeric` or `named` index that does not yet exist:

```
length(list3)
```

```
## [1] 3
```

Adding a `numeric` index:

```
list3[[4]] <- 11
length(list3)
```

```
## [1] 4
```

```
list3
```

```
## $theARR
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
##
## $theVECT
## [1]  1  2  3  4  5  6  7  8  9 10
##
## $List3
## $List3$`The Array`
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
##
## $List3$`The Vector`
## [1]  1  2  3  4  5  6  7  8  9 10
##
##
## [[4]]
## [1] 11
```

Adding a named index:

```
list3[["AddedElement"]] <- 12:16
length(list3)
```

```
## [1] 5
```

```
list3
```

```
## $theARR
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
##
## $theVECT
## [1]  1  2  3  4  5  6  7  8  9 10
##
## $List3
## $List3$`The Array`
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
##
## $List3$`The Vector`
## [1]  1  2  3  4  5  6  7  8  9 10
##
##
## [[4]]
## [1] 11
##
## $AddedElement
## [1] 12 13 14 15 16
```


Chapter 6

R Learning Resources

A rich and responsive constellation of print and online information, explanation, guidance, and discussion make up the lively and growing R universe. Help is available at any time — you just have to know where to find it.

6.1 Books On R

- Learn R : 12 Free Books and Online Resources
- Practical Data Science With R
- R in Action, Second Edition
- **R For Data Science** by Hadley Wickham
- R for Everyone by Jared Lander
- Teach Yourself R In 24 Hours
- The R Series | CRC Press Online

6.1.1 Online R Books

- bookdown: Authoring Books With RMarkdown by Yihui Xie
- bookdown.org: R books made with bookdown
- Data Visualization by Kieran Healy
- Efficient R programming by Colin Gillespie & Robin Lovelace
- Happy Git and GitHub for the useR by Jenny Bryan & Others
- **R For Data Science** by Hadley Wickham
- R packages by Hadley Wickham
- Text Mining with R by Julia Silge and David Robinson

6.1.2 WikiBooks on R

- Data Mining Algorithms In R
- R Programming
- Statistical Analysis: an Introduction using R

6.2 R Blogs

6.2.1 Aggregation Sites

- **R-bloggers | R news and tutorials by R bloggers**
- RWeekly.org - Blogs to Learn R from the Community
- Statistics Blogs @ StatsBlogs.com | All About Statistics |

6.2.2 Corporate/NGO

- RStudio Blog
- RStudio RViews
- Revolutions
- rOpenSci Blog

6.2.3 Personal

- Jared Lander
- John Myles White
- Andrew Gelman
- Learning Slowly
- TRinker's R Blog
- You Canalytics

6.3 R Tutorials

- Awesome-R: A curated list | GitHub
- Data Visualization
- Getting Started | RStudio Support
- How to get started with Data Science using R | R-bloggers
- Implementation of the Matplotlib 'viridis' color map in R | GitHub
- Learn R, Python & Data Science Online | DataCamp
- Quick list of useful R packages | RStudio Support
- Quick R | DataCamp
- R Package Management Tools
- R-Courses | R-Exercises
- Start here to learn R! | R-Exercises
- Try R | Code School
- Tutorials for learning R | R-bloggers
- UBC STAT 545
- Using R | Working With Data Series

6.4 R Discussion Forums

- Cross Validated | Newest 'r' Questions
- Stack Overflow | Newest 'r' Questions
- R Forum | nabble
- R Learning Resources
- R Resources for Beginners

- [R-help Info Page](#)
- [R-Help Subscription Page](#)

6.5 R Mailing Lists

- [R Project Mailing Lists](#)
- [R Mailing List Archive](#)
- [R Mailing List Subscription](#)
- [R-help | Mailing List Archive](#)
- [R-help | Mailing List Subscription](#)

6.6 R Podcasts

If you're into listening to podcasts, there are many great podcasts about R, Data Science, and Artificial Intelligence. Here are some:

- [Not So Standard Deviations](#)
- [DataFramed](#)
- [The R Podcast](#)
- [This Week in Machine Learning and AI](#)
- [The Microsoft Research Podcast](#)
- [Bombshell](#)
- [Marketplace](#)
- [Revolutions](#)

Collected from: [A few podcast recommendations](#) | [R-bloggers](#)