A Starting with R and other tools

In this appendix, we'll show how you can install tools and start working with R. We'll demonstrate some example concepts and steps, but you'll want to follow up with additional reading.

A.1 Installing the tools

The primary tool for working our examples will be R, and possibly RStudio. But other tools (databases, version control) are also highly recommended. You may also need access to online documentation or other help to get all of these tools to work in your environment. The distribution sites we list are a good place to start.

A.1.1 Installing Tools

The R environment is set of tools and software that can be installed on Unix, Linux, Apple OSX, and Windows.

R

We recommend installing the latest version of R from the Comprehensive R Archive Network (CRAN), https://cran.r-project.org or mirror. CRAN is the authoritative central repository for R and R packages. CRAN and supported by The R Foundation and the R Development Core Team. R itself is an official part of the Free Software Foundation's GNU project distributed under a GPL 2 license. R is used at many large institutions including the United States Food and Drug Administration[1].

Footnote 1 Source: https://www.r-project.org/doc/R-FDA.pdf.

RSTUDIO

For R work we suggest using RStudio. RStudio is a popular cross platform integrated development environment supplied by the company RStudio, Inc. (https://www.rstudio.com). Is supplies a built in text editor, and convenient user interfaces for common tasks such as installing software, plotting, rendering RMarkdown documents, and working with source control. RStudio is not an official part of R or CRAN, and should not be confused with R or CRAN.

An important feature of RStudio is the file browser and the "set directory"/"go to directory" controls that are hidden in the gear icon of the file browsing pane, which we point out in figure A.1.

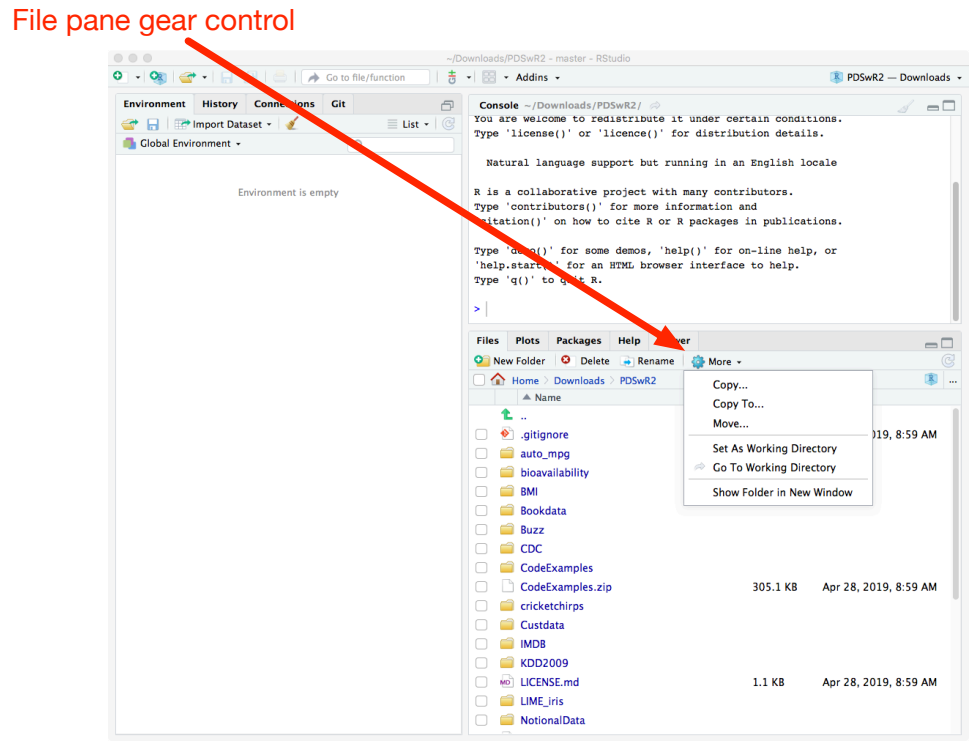

Figure A.1 RStudio file browsing controls

To work with R one needs a text editor specialized for working with non-formatted (or not-rich) text. Such editors include Atom, Emacs, Notepad++, Pico, Programmer's Notepad, RStudio, Sublime Text, text wrangler, vim, and many more. This are in contrast to rich text editors (which are not appropriate for programming tasks) such as Microsoft Word or Apple Text Edit.

GIT

Git is a source control or version management system that is very useful for preserving and sharing work. To install git, please follow the appropriate instructions from https://git-scm.com.

Data science always involves a lot of tools and collaboration, so a willingness to try new tools is a flexibility one needs to develop.

THE BOOK SUPPORT MATERIALS

All of the book support materials are freely available from Github: https://github.com/WinVector/PDSwR2. The reader should download them in their entirety either using Git clone with the URL https://github.com/WinVector/PDSwR2.git or by download a complete zip file by using the "Clone or Download" control in the top right of the Github page.

Another way to download the book material is to use RStudio and Git. Select `File -> New Project -> Create Project from Version Control -> Git`. That will bring up a dialogue box as shown in figure A.1 where we can fill in the git-URL and download the book materials as a project.

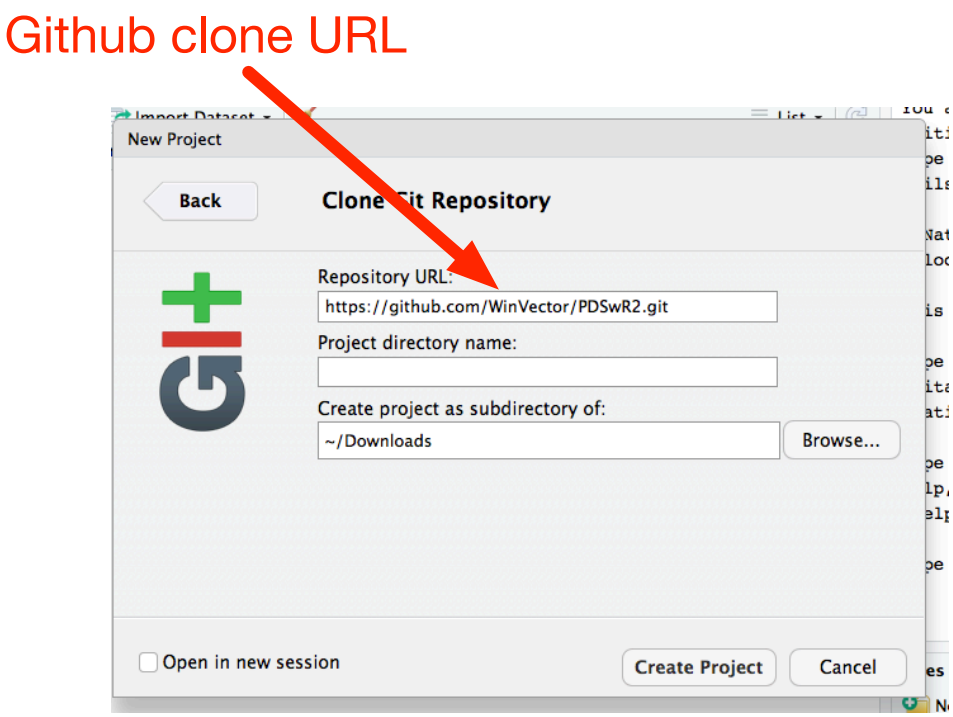

Figure A.2 Cloning the book repository

We will refer to this directory as `PDSwR2` throughout the book, and all files and paths we mention are either in this directory or a sub-directory.

Please be sure to look in this directory for any README or errata files.

Some features of the support directory include.

- All example data used in the book.
- All example code used in the book. The examples from the book are available in the sub-directory `CodeExamples`, and also as a zip-file `CodeExamples.zip`. In addition to this

the entire set of examples re-run and re-rendered are shared in `RenderedExamples`. (All paths relative to where you have unpacked the book directory `PDSwR2`.)

R PACKAGES

A great advantage of R is there is the CRAN central package repository. R has standardized package installation through the "`install.packages()`" command. An installed package is typically not fully available for use in a project until the package is also attached for use by the "`library()`" command.[2] A good practice is: any sort of R script or work should attach all of the packages it intends to use as a first step. Also in most cases scripts should *not* call `install.packages()` as this changes the R installation, which should not be done without user supervision.

Footnote 2 In R installing a package is a separate step from attaching the package contents to the software search path. `install.packages()` makes package contents potentially available, after that `library()` readies them for use. A handy mnemonic is: `install.packages()` sets up new appliances in your kitchen, and `library()` turns them on. You don't have to install things very often, however you often have to turn things back on.

INSTALLING THE REQUIRED PACKAGES

To install the set of packages required to work all of the examples in this book first download the book repository as described above. Then look in the first directory of top directory of this repository: `PDSwR2`. In this directory you will find the file `packages.R`. You can open this file with a text editor and it should look like the following (though it may be more up to date than what is shown here).

```
# Please have an up to date version of R (3.5.*, or newer)
# Answer "no" to:
# Do you want to install from sources the packages which need compilation?
update.packages(ask = FALSE, checkBuilt = TRUE)

pkgs <- c(
  'arules', 'bitops', 'caTools', 'cdata', 'data.table', 'DBI',
  'dplyr', 'e1071', 'fpc', 'gdata', 'ggplot2', 'glmnet', 'glmnetUtils',
  'gridExtra', 'hexbin', 'kernlab', 'knitr', 'lime',
  'lubridate', 'magrittr', 'MASS', 'mgcv', 'pander', 'plotly', 'pwr',
  'randomForest', 'readr', 'readxls', 'rmarkdown', 'rpart',
  'rpart.plot', 'rquery', 'RSQLite', 'scales', 'sigr', 'sqldf', 'tidyr',
  'vtreat', 'wrapr', 'WVPlots', 'xgboost', 'xts', 'zeallot', 'zoo')
install.packages(pkgs, dependencies = c("Depends", "Imports", "LinkingTo"))
```

To install everything run every line of code in this file from R. In RStudio this can be done by pressing the "Source" button in the top-right of the editor pane. Or in any case this can be accomplished by copying the code and pasting it into a running R prompt or R console.[3]

Footnote 3 The above code is in the book support directory as `packages.R`, which would refer to as `PDSwR2/packages.R`, and has the Github URL https://github.com/WinVector/PDSwR2.

Unfortunately, there are many reasons the install can fail: incorrect copy/paste, no internet connection, improperly configured R or RStudio, insufficient permissions to administrator the R install, out of date version of R or RStudio, and no or incorrect C/C++/Fortran compiler. If you run into these problems it is best to find a forum or expert to help work through these steps. Once everything is successfully installed, R is self-contained environment where "things just work."

OTHER TOOLS

R's capabilities can be enhanced by use of tools such as gcc/clang, gfortran, git, Rcpp, Tex, pandoc, ImageMagick, and bash shell. Each of these is managed outside of R and how to maintain them depends on your computer, operating system, and system permissions. Unix/Linux users have the easiest time installing these tools and R is primarily developed in an Unix environment.[4] RStudio will install some of the extra tools. OSX users may need Apple's XCode tools and Homebrew (https://brew.sh) to match expected open source tool chain. Windows users who wish to write package may want to research RTools (https://cran.r-project.org/bin/windows/Rtools/).

Footnote 4 For example we share notes on rapidly configuring R and RStudio Server on an Amazon EC2 instance here: http://www.win-vector.com/blog/2018/01/setting-up-rstudio-server-quickly-on-amazon-ec2/.

Windows users my need RTools to compile packages, however this should not be strictly necissary as most current packages are available from CRAN in a pre-compiled form (at lest for MacOS and 64 bit Windows). MacOS users may need to install the XCode compiler (available from Apple) to compile packages. All of these are steps you probably want to skip until you know you need the ability to compile.

A.1.2 The R package system

R is a broad and powerful language and analysis workbench in and of itself. But one of its real strengths is the depth of the package system and packages supplied through CRAN. To install a package from CRAN, just type `install.packages('nameofpackage')`. To use an installed package, type `library(nameofpackage)`.[5] Any time you type `library('nameofpackage')` or `require('nameofpackage')`, you're assuming you're using a built-in package or you're able to run `install.packages('nameofpackage')` if needed. We'll return to the package system again and again in this book. To see what packages are present in your session, type `sessionInfo()`.

Footnote 5 Actually, `library('nameofpackage')` also works with quotes. The unquoted form works in R because R has the ability to delay argument evaluation (so an undefined `nameofpackage` doesn't cause an error) as well as the ability to snoop the names of argument variables (most programming languages rely only on references or values of arguments). Given that a data scientist has to work with many tools and languages throughout the day, we prefer to not rely on features unique to one language unless we really need the feature. But the "official R style" is without the quotes.

TIP **Changing your CRAN mirror**

You can change your CRAN mirror at any time with the `chooseCRANmirror()` command. This is handy if the mirror you're working with is slow.

A.1.3 Installing Git

We advise installing Git version control before we show you how to use R and RStudio. This is because without Git, or a tool like it, you'll lose important work. Not just lose *your* work—you'll lose important *client* work. A lot of data science work (especially the analysis tasks) involves trying variations and learning things. Sometimes you learn something surprising and need to redo earlier experiments. Version control keeps earlier versions of all of your work, so it's exactly the right tool to recover code and settings used in earlier experiments. Git is available in precompiled packages from http://git-scm.com.

A.1.4 Installing RStudio

RStudio supplies a text editor (for editing R scripts) and an integrated development environment for R. Before picking up RStudio from http://rstudio.com, you should install both R and Git as we described earlier.

The RStudio product you initially want is called *RStudio Desktop* and is available precompiled for Windows, Linux, and OS X.

When first starting with RStudio we strongly recommend turning off both the "Restore .RData into workspace at startup" and "Save workspace to .RData on exit" features. Having these settings on (the default) makes it hard to reliably "work clean" (a point we will discuss in section A.1. To turn off these features open the RStudio options pane (The Global options is found by such as menus "RStudio -> Preferences", "Tools -> Global Options", "Tools -> Options", or similar- depending on what operating system you are using), and then alter the two settings as indicated in figure A.1.

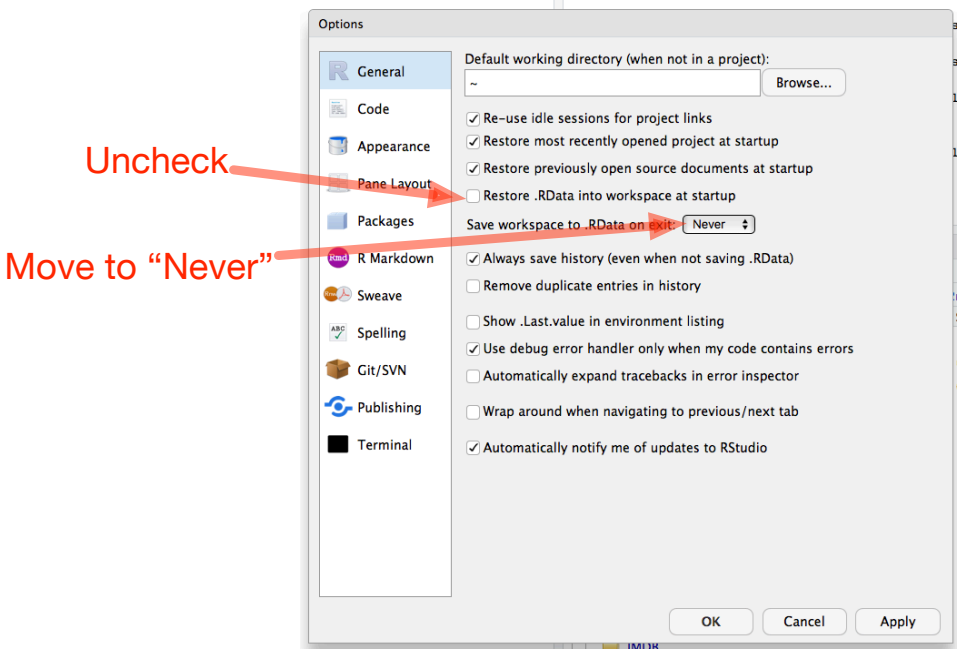

Figure A.3 RStudio Options

A.1.5 R resources

A lot of the power of R comes from the deep bench of freely available online resources. In this section, we'll touch on a few sources of code and documentation.

INSTALLING R VIEWS

R has an incredibly deep set of available libraries. Usually, R already has the package you want; it's just a matter of finding it. A powerful way to find R packages is using *views*: http://cran.r-project.org/web/views/.

You can also install all of the packages (with help documentation) from a view in a single command (though be warned: this can take an hour to finish). For example, here we're installing a huge set of time series libraries all at once:

```
install.packages('ctv', repos='https://cran.r-project.org')
library('ctv')
# install.views('TimeSeries') # can take a LONG time
```

Once you've done this, you're ready to try examples and code.

ONLINE R RESOURCES

A lot of R help is available online. Some of our favorite resources include these:

- *CRAN*—The main R site: http://cran.r-project.org
- *Stack Overflow R section*—A question-and-answer site: http://stackoverflow.com/questions/tagged/r
- *Quick-R*—A great R resource: http://www.statmethods.net

- *LearnR*—A translation of all the plots from *Lattice: Multivariate Data Visualization with R (Use R!)* (by D. Sarker; Springer, 2008) into ggplot2: http://learnr.wordpress.com
- *R-bloggers*—A high-quality R blog aggregator: http://www.r-bloggers.com
- *RStudio community*—A high-quality R blog aggregator: https://community.rstudio.com/ A RStudio/"tidyverse" oriented company site.

A.2 Starting with R

R implements a dialect of a statistical programming language called *S*. The original implementation of S evolved into a commercial package called S+. So most of R's language design decisions are really facts about S. To avoid confusion, we'll mostly just say *R* when describing features. You might wonder what sort of command and programming environment S/R is. It's a pretty powerful one, with a nice command interpreter that we encourage you to type directly into.

SIDEBAR

Work clean

In R or R studio, it is important to "work clean" that is to start with an empty workspace and explicitly bring in the packages, code, and data you want. This ensures you know how to get into your ready to go state (as you have to do or write-down the steps to get there) and you aren't held hostage to state you don't know how to restore (what we call the "no alien artifact" rule).

To work clean in R you must turn off any sort of auto-restore of the workspace. In "base R" this is done as follows.

We discussed how to work clean with RStudio in section .1.4.

Working with R and issuing commands to R is in fact scripting or programming. We assume you have some familiarity with scripting (perhaps using Visual Basic, Bash, Perl, Python, Ruby, and so on) or programing (perhaps using C, C#, C++, Java, Lisp, Scheme, and so on), or are willing to use one of our references to learn. We don't intend to write long programs in R, but we'll have to show how to issue R commands. R's programming, though powerful, is a bit different than many of the popular programming languages, but we feel that with a few pointers, anyone can use R. If you don't know how to use a command, try using the `help()` call to get at some documentation.

Throughout this book, we'll instruct you to run various commands in R. This will almost always mean typing the text or the text following the command prompt > into the RStudio console window, followed by pressing Return. For example, if we tell you to type `1/5`, you can type that into the console window, and when you press Enter you'll see a result such as `[1] 0.2`. The `[1]` portion of the result is just R's way of labeling result rows (and is to be ignored), and the `0.2` is the floating point representation of one-fifth, as requested.

TIP **Help**

Always try calling `help()` to learn about commands. For example, `help('if')` will bring up help about R's `if` command.

Let's try a few commands to help you become familiar with R and its basic data types. R commands can be terminated with a line break or a semicolon (or both), but interactive content isn't executed until you press Return. The following listing shows a few experiments you should run in your copy of R.

Listing A.1 Trying a few R commands

```
1
## [1] 1
1/2
## [1] 0.5
'Joe'
## [1] "Joe"
"Joe"
## [1] "Joe"
"Joe"=='Joe'
## [1] TRUE
c()
## NULL
is.null(c())
## [1] TRUE
is.null(5)
## [1] FALSE
c(1)
## [1] 1
c(1,2)
## [1] 1 2
c("Apple",'Orange')
## [1] "Apple"  "Orange"
length(c(1,2))
## [1] 2
vec <- c(1,2)
vec
## [1] 1 2
```

TIP **Multiline commands in R**

R is good with multiline commands. To enter a multiline command, just make sure it would be a syntax error to stop parsing where you break a line. For example, to enter `1+2` as two lines, add the line break after the plus sign and not before. To get out of R's multiline mode, press Escape. A lot of cryptic R errors are caused by either a statement ending earlier than you wanted (a line break that doesn't force a syntax error on early termination) or not ending where you expect (needing an additional line break or semicolon).