Extending R with Packages & Literate Programming with RMarkdown

UA R Users Group

May 14, 2019

Contents

Learning Objectives

- Know what packages are and how to install them from CRAN
- Understand why literate programming is useful
- Create and edit an RMarkdown file
- Know how to manipulate some common chunk options

Packages

Packages are bundles of code which extend the functionality of R.

Anyone can make an R package, and anyone can install anyone else's R package (if they make it available). This is part of the beauty of open source, and using different R packages is essential to modern R workflows.

You can get packages from many different places, but we'll focus on just the most common one: CRAN. CRAN is the Comprehensive R Archive Network, a global network of servers which make available for download a set of vetted R packages.

The next section is about RMarkdown, a package, so we'll install that now.

Installation

To download and install a package from CRAN, call the `install.packages` command on a string with the name of the desired package. You will get output describing the installation progress.

```
install.packages("rmarkdown", repos="http://cran.rstudio.com/")
```

```
## Installing package into 'C:/Users/jd033/Documents/R/win-library/3.5'
## (as 'lib' is unspecified)
```

```
## package 'rmarkdown' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\jd033\AppData\Local\Temp\RtmpIlWtam\downloaded_packages
```

You may be asked to choose a mirror; the RStudio mirror is a good choice as it will pick the nearest mirror automatically. This will also download and install packages which RMarkdown depends on.

You only need to install a package once per machine, unless you need to update an already-installed package. Calling `install.packages` for an existing package will update it if there's a more recent version on CRAN than on your machine.

You can view all of the installed packages using the `installed.packages` command. This will output a lot of information for each package, so if you only want a list of the installed package names, you can specify that you want the "Package" column. I like to look at this as a vector.

```
as.vector(installed.packages()[,"Package"])
```

```
##   [1] "abind"           "acepack"         "ade4"
##   [4] "ALL"             "ape"             "arm"
##   [7] "arules"          "arulesViz"       "askpass"
##  [10] "assertthat"      "backports"       "base64enc"
##  [13] "bayesm"          "bayesplot"       "bayestestR"
##  [16] "BH"              "Biobase"         "BiocGenerics"
##  [19] "BiocManager"     "BiocVersion"     "biomformat"
##  [22] "Biostrings"      "bit"             "bit64"
##  [25] "bitops"          "BMS"             "bootstrap"
##  [28] "brew"            "broom"           "callr"
##  [31] "car"             "carData"         "caTools"
##  [34] "cellranger"      "checkmate"       "classInt"
##  [37] "cli"             "clipr"           "clisymbols"
##  [40] "coda"            "colorspace"      "colourpicker"
##  [43] "Compositional"   "corrplot"        "cowplot"
##  [46] "crayon"          "crosstalk"       "curl"
##  [49] "data.table"      "datasets.load"   "DBI"
##  [52] "dbplyr"          "deldir"          "dendextend"
##  [55] "Deriv"           "desc"            "devtools"
##  [58] "digest"          "dirichlet"       "DirichletReg"
##  [61] "dirmult"         "doParallel"      "dotCall64"
##  [64] "dplyr"           "DT"              "dygraphs"
##  [67] "e1071"           "ellipsis"        "emmeans"
##  [70] "emplik"          "estimability"    "evaluate"
##  [73] "evd"             "expm"            "extrafont"
##  [76] "extrafontdb"     "fansi"           "farver"
##  [79] "fields"          "fitdistrplus"    "FNN"
```

```
##  [82] "forcats"          "foreach"          "Formula"
##  [85] "fs"               "gclus"            "gdata"
##  [88] "gdtools"          "generics"         "gganimate"
##  [91] "ggeffects"        "ggmap"            "ggplot2"
##  [94] "ggridges"         "ggthemes"         "gh"
##  [97] "gifski"           "GIGrvg"           "git2r"
## [100] "glmmTMB"          "glmnet"           "glue"
## [103] "gmodels"          "goftest"          "gplots"
## [106] "gridExtra"        "groupdata2"       "gtable"
## [109] "gtools"           "haven"            "hexbin"
## [112] "highr"            "Hmisc"            "hms"
## [115] "horseshoe"        "hrbrthemes"       "HSAUR"
## [118] "HSAUR2"           "htmlTable"        "htmltools"
## [121] "htmlwidgets"      "httpuv"           "httr"
## [124] "igraph"           "ini"              "inline"
## [127] "insight"          "IRanges"          "ISLR"
## [130] "iterators"        "jpeg"             "jsonlite"
## [133] "kableExtra"       "knitr"            "labeling"
## [136] "lars"             "later"            "latticeExtra"
## [139] "lazyeval"         "leaflet"          "LearnBayes"
## [142] "lme4"             "lmtest"           "loo"
## [145] "lpSolve"          "lsei"             "lubridate"
## [148] "lwgeom"           "magick"           "magrittr"
## [151] "manipulateWidget" "maps"             "maptools"
## [154] "mapview"          "markdown"         "MatrixModels"
## [157] "matrixStats"      "maxLik"           "mcmc"
## [160] "MCMCpack"         "memoise"          "mime"
## [163] "miniUI"           "minqa"            "miscTools"
## [166] "mixture"          "mnormt"           "modelr"
## [169] "monomvn"          "multtest"         "munsell"
## [172] "mvtnorm"          "neuralnet"        "nloptr"
## [175] "NLP"              "nnet"             "npsurv"
## [178] "numbers"          "numDeriv"         "openssl"
## [181] "openxlsx"         "packrat"          "pacman"
## [184] "pander"           "pbkrtest"         "performance"
## [187] "permute"          "philentropy"      "phyloseq"
## [190] "pillar"           "pkgbuild"         "pkgconfig"
## [193] "pkgload"          "plogr"            "plotly"
## [196] "pls"              "plyr"             "png"
## [199] "polyclip"         "prettyunits"      "processx"
## [202] "progress"         "promises"         "pryr"
## [205] "ps"               "psych"            "purrr"
## [208] "qap"              "quantreg"         "R6"
## [211] "randomForest"     "ranger"           "rappdirs"
## [214] "rapportools"      "raster"           "rcmdcheck"
## [217] "RColorBrewer"     "Rcpp"             "RcppArmadillo"
## [220] "RcppEigen"        "RcppGSL"          "RcppZiggurat"
## [223] "RCurl"            "readr"            "readxl"
```

```
## [226] "registry"         "rematch"          "remotes"
## [229] "reprex"           "reshape"          "reshape2"
## [232] "revealjs"         "Rfast"            "rgdal"
## [235] "rgl"              "RgoogleMaps"      "rhdf5"
## [238] "Rhdf5lib"         "rio"              "rjson"
## [241] "rlang"            "rmarkdown"        "rprojroot"
## [244] "rsconnect"        "rstan"            "rstantools"
## [247] "rstudioapi"       "Rttf2pt1"         "rtweet"
## [250] "RUnit"            "rvest"            "S4Vectors"
## [253] "sandwich"         "satellite"        "scales"
## [256] "scalreg"          "scatterplot3d"    "selectr"
## [259] "seriation"        "sessioninfo"      "sf"
## [262] "shiny"            "shinyjs"          "shinystan"
## [265] "shinythemes"      "sjlabelled"       "sjmisc"
## [268] "sjPlot"           "sjstats"          "slam"
## [271] "sn"               "sourcetools"      "sp"
## [274] "spam"             "SparseM"          "spatstat"
## [277] "spatstat.data"    "spatstat.utils"   "spData"
## [280] "spdep"            "StanHeaders"      "statmod"
## [283] "stringi"          "stringr"          "summarytools"
## [286] "svglite"          "sys"              "tensor"
## [289] "threejs"          "tibble"           "tidycensus"
## [292] "tidyr"            "tidyselect"       "tidyverse"
## [295] "tigris"           "tinytex"          "tm"
## [298] "TMB"              "transformr"       "tree"
## [301] "truncdist"        "TSP"              "tweenr"
## [304] "twitteR"          "units"            "usethis"
## [307] "utf8"             "uuid"             "vcd"
## [310] "vegan"            "viridis"          "viridisLite"
## [313] "visNetwork"       "webshot"          "whisker"
## [316] "withr"            "xfun"             "xml2"
## [319] "xopen"            "xtable"           "xts"
## [322] "XVector"          "yaml"             "zip"
## [325] "zlibbioc"         "zoo"              "base"
## [328] "boot"             "class"            "cluster"
## [331] "codetools"        "compiler"         "datasets"
## [334] "foreign"          "graphics"         "grDevices"
## [337] "grid"             "KernSmooth"       "lattice"
## [340] "MASS"             "Matrix"           "methods"
## [343] "mgcv"             "nlme"             "nnet"
## [346] "parallel"         "rpart"            "spatial"
## [349] "splines"          "stats"            "stats4"
## [352] "survival"         "tcltk"            "tools"
## [355] "translations"     "utils"
```

Loading

Most packages need to be loaded into the current environment to be accessible. RMarkdown is specially integrated in RStudio in a way that avoids this, but in general we load packages with the `library` command:

```
library(rmarkdown)  # notice the lack of quotes
```

This will come up again later in the lesson on `dplyr`, an external package that *does* need to be loaded.

You can also view the packages that you have loaded into your workspace.

```
(.packages())
```

```
## [1] "rmarkdown" "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
```

R Markdown

R Markdown is a special file format which allows us to combine text, code, *and the output of that code* in a single file. This combination of explanation, code, and results is called *literate programming* and is a powerful way to share research and data explorations.

RMarkdown is an extended version of the Markdown (`.md`) file format, which is an easy way to make nicely formatted text documents without endlessly tinkering with the formatting (as you might with LaTeX). The software community loves Markdown because in addition to being straightforward, it has good support for formatting code, which can be a pain in other formats.

RMarkdown takes this a step further by allowing you to *run* the code in your document, and having the output appear below the code that made it.

If you've used ipython/Jupyter notebooks before, R Markdown will feel similar. All the lessons in this workshop were created with R Markdown!

My First Rmd

Rstudio makes it easy to create a new RMarkdown file, and it even starts with a demo file that shows off most of the basic features of the `Rmd` format. In the upper-left corner, click the "new file" icon and select RMarkdown. A window should appear to help you configure this file initially. There's a lot of options (R Markdown can do so much!), but for now, make sure your name is in the "Author" field, and change the "Title" to be something like "AARUG Workshop".

Before we delve into what each of these pieces mean, let's "knit" the document so we can see what kind of output RMarkdown produces. Above the file, press the `knit` button, the one that looks like a ball of yarn.

You should see a new pane open in RStudio that shows R "knitting" the document, and when it's done, a pop-up will appear showing the knitted output.

This new output being displayed as an `html` file; look in the file browser pane, and you'll see a `.html` file next to your `.Rmd` file (may need to refresh), because RStudio automatically saved this output when the document finished knitting.

Let's look at the individual pieces in this document:

Header

This is the section at the top, with three dashes before and after. This lists some metadata about the object. The title, date, and author form the start of the output document, and the `output:` line instructs the knitting process to generate an html file.

Section Titles

You can enlarge text be preceding it with one or more pound signs (`#`). This is mainly useful for organizing a document into sections. The more pound signs, the smaller the text, so when you make sub-section you should add at least one more pound sign than used in the parent sections' title.

Link

You can make text clickable by including a link to a different website. An example can be found above, where we included a link to CRAN.

There are two parts to creating linked text. The first part is including the text you want to see, surrounded by square brackets `[CRAN]`. Immediately after that, add the link surrounded by parentheses `(https://cran.r-project.org)`. The final product looks like `[CRAN](https://cran.r-project.org)`.

Bold/Italic Text

The double-asterisks surrounding the word "Knit" in the second paragraph cause that piece of text to be bold. This phrase can be multiple words, but should not have spaces immediately on the inside of the asterisks. You can make text italix by similarly wrapping in underscores (`_`) or using single asterisks.

Code Chunks

This is the real meat of the document! An R Markdown code chunk is a section which starts and ends with triple-backticks (‘, not `'`). After the initial set, the curly-bracketed section which starts with `{r` is what forces this to be ran as R code; without this piece, the section would get formatted like code, but would not be executed when knitting. The phrase after the `r` is the *chunk name*. Chunks do not need to be named, but no two chunks can have the same name. Naming chunks can help keep code organized and make it easier to track down the source of errors when they occur.

Chunk Options

As the second default section discusses, we can hide the code in a code chunk by placing a comma after out chunk name and setting an option `echo=FALSE`. The code will still execute, and its output will be inserted in the knitted document, but it will not be shown.

Similarly, you can set `eval=FALSE` to avoid running a code chunk.

Digging Deeper

There's a lot more to RMarkdown than just this; as the demo document shows, you can visit http://rmarkdown.rstudio.com to learn more.

We'll be using R Markdown for the rest of this workshop to keep a running log of what we're learning. This will allow you to walk away with a knit document which has not only the code commands you've learned to use, but the output of those commands and some explanatory text. That's literate programming!