Activity Worksheet

Seeking Help

Dan Maxwell

2018-09-27

A MyDataStory / THE CARPENTRIES Learning Experience

Reading Help files

R, and every package, provide help files for functions. To search for help on a function from a specific function that is in a package loaded into your namespace (your interactive R session):

```
?function_name
help(function_name)
```

This will load up a help page in RStudio (or as plain text in R by itself).

Each help page is broken down into sections:

- Description: An extended description of what the function does.
- Usage: The arguments of the function and their default values.
- Arguments: An explanation of the data each argument is expecting.
- Details: Any important details to be aware of.
- Value: The data the function returns.
- See Also: Any related functions you might find useful.
- Examples: Some examples for how to use the function.

Different functions might have different sections, but these are the main ones you should be aware of.

> **Tip: Reading help files**
>
> One of the most daunting aspects of R is the large number of functions available. It would be prohibitive, if not impossible to remember the correct usage for every function you use. Luckily, the help files mean you don't have to!

Special Operators

To seek help on special operators, use quotes:

```
?"+"
```

Getting help on packages

Many packages come with "vignettes": tutorials and extended example documentation. Without any arguments, `vignette()` will list all vignettes for all installed packages; `vignette(package="package-name")` will list all available vignettes for `package-name`, and `vignette("vignette-name")` will open the specified vignette.

If a package doesn't have any vignettes, you can usually find help by typing `help("package-name")`.

When you kind of remember the function

If you're not sure what package a function is in, or how it's specifically spelled you can do a fuzzy search:

```
??function_name
```

When you have no idea where to begin

If you don't know what function or package you need to use CRAN Task Views is a specially maintained list of packages grouped into fields. This can be a good starting point.

When your code doesn't work: seeking help from your peers

If you're having trouble using a function, 9 times out of 10, the answers you are seeking have already been answered on Stack Overflow. You can search using the `[r]` tag.

If you can't find the answer, there are a few useful functions to help you ask a question from your peers:

```
?dput
```

Will dump the data you're working with into a format so that it can be copy and pasted by anyone else into their R session.

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 14393)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.5.1  backports_1.1.2 magrittr_1.5    rprojroot_1.3-2
##  [5] tools_3.5.1     htmltools_0.3.6 yaml_2.2.0      Rcpp_0.12.18
##  [9] stringi_1.1.7   rmarkdown_1.10  knitr_1.20      stringr_1.3.1
## [13] digest_0.6.17   evaluate_0.11
```

Will print out your current version of R, as well as any packages you have loaded. This can be useful for others to help reproduce and debug your issue.

Challenge 1

Look at the help for the `c` function. What kind of vector do you expect you will create if you evaluate the following:

```
c(1, 2, 3)
c('d', 'e', 'f')
c(1, 2, 'f')
```

```
# Solution to Challenge 1
# The `c()` function creates a vector, in which all elements are the
# same type. In the first case, the elements are numeric, in the
# second, they are characters, and in the third they are characters:
# the numeric values are "coerced" to be characters.
```

Challenge 2

Look at the help for the `paste` function. You'll need to use this later. What is the difference between the `sep` and `collapse` arguments?

```
# Solution to Challenge 2

# To look at the help for the `paste()` function, use:

help("paste")
?paste

# The difference between `sep` and `collapse` is a little
# tricky. The `paste` function accepts any number of arguments, each of which
# can be a vector of any length. The `sep` argument specifies the string
# used between concatenated terms - by default, a space. The result is a
# vector as long as the longest argument supplied to `paste`. In contrast,
# `collapse` specifies that after concatenation the elements are *collapsed*
# together using the given separator, the result being a single string.
# e.g.

paste(c("a","b"), "c")
paste(c("a","b"), "c", sep = ",")
paste(c("a","b"), "c", collapse = "|")
paste(c("a","b"), "c", sep = ",", collapse = "|")

# For more information,
# scroll to the bottom of the `?paste` help page and look at the
# examples, or try `example('paste')`.)
```

Challenge 3

Use help to find a function (and its associated parameters) that you could use to load data from a csv file in which columns are delimited with “͡” (tab) and the decimal point is a “.” (period). This check for decimal separator is important, especially if you are working with international colleagues, because different countries have different conventions for the decimal point (i.e. comma vs period).

hint: use `??csv` to lookup csv related functions.

```
# Solution to Challenge 3

# The standard R function for reading tab-delimited files with a period
# decimal separator is read.delim(). You can also do this with
# `read.table(file, sep="\t")` (the period is the *default* decimal
# separator for `read.table()`, although you may have to change
# the `comment.char` argument as well, if your data file contains
# hash (#) characters
```

Other ports of call

- Quick R
- RStudio cheat sheets
- Cookbook for R