Packages and Subsetting

Topics

1. Packages
2. Apply functions
3. Subsetting

Packages in R

With many programs, everything is included in the initial installation. R works differently. When you first install R, it installs the program and a set of “base” packages. Packages are just scripts which define functions for subsequent use. Installed packages are simply text files (".R" extension) with functions defined in them that are stored on your computer\(^1\). However, having a package installed does not necessarily mean its functions will be available for use. To make the functions within a package available for use, the package must be “sourced”, which essentially means that the contents of the scripts are run through your current R session. The base libraries are a special group of packages that are not only installed by default but also are sourced automatically every time you open R. If you want to use functions in packages that aren’t part of the base distribution, you have to do two things:

1. **Install the package.** Again, this just puts the files on your computer in the correct place so R can find them. You can use the command `install.packages("package-name")` (note the quotes are necessary), or do it using the drop-down menus. You can install packages directly (meaning that R downloads them from the CRAN mirror you choose and installs them), or you can download packages as compressed archives (usually `.tar.gz`) that you subsequently install using the above command (I recommend the first approach so you don’t end up with multiple versions of different packages in different places).

2. **Load (source) the package.** Simply type `library("package-name")`. I recommend that you place all `library()` calls in one place in your project scripts so it is clear to others what packages are necessary to run your analyses (see script example in Week 3-Data). You can also use `require()` which does the same thing\(^2\), and may make more sense to you conceptually as in, “I require this package for the rest of the script to work”.

Apply functions

These are useful functions for doing the sorts of data exploration that you want to (and should) do all the time. I introduce them here because, although they are more “advanced”, it’s hard to imagine doing any manipulation of your data without using them. For the short exercises below, you’ll use a small dataset (littleData.txt) included in the package you downloaded. Try the commands below to begin using these functions. [It is unlikely that these will be obvious to you at first—the more you play with them the more natural they will seem.]

→ Download littleData.txt and save it to your working directory for this class. You’ll have to change the `read.table()` below if your data and script aren’t in the same directory (but you already know that now that you have mastered the concept of the working directory).

→ Create a new script, littleData.R in Rstudio.

\(^1\)They are compiled, so this is not exactly true. To find out where they are stored, type `.Library in R.

\(^2\)The difference between the two likely won’t be noticeable for you: `require()` returns TRUE/FALSE, so will work inside functions and allows for proper error-checking should you decide to start building packages.
apply

apply() is good for performing function across columns or rows.

Read in data:

```r
> data <- read.table("littleData.txt", header=T, sep="\t")
> names(data)
> str(data)
```

Take the mean of columns 3 and 4:

```r
> apply(data[, -(1:2)], 2, mean)
```

Here we are first subsetting data to remove columns 1 and 2 (`data[, -(1:2)]`). The “2” indicates operate over columns (use “1” for rows), and “mean” is the function we are applying to those columns.

You can add these types of calculations to your data frame:

```r
> data$means <- apply(data[, -(1:2)], 1, mean)
> data
```

Now you’ll see a new column that shows the mean of every row. We’ve used mean in these examples, but any function can be used:

```r
> par(mfrow = c(2, 2)) # we’ll learn more about this when we do plotting
> apply(data[, -1:2], 2, function(x) hist(x))
> apply(data, 1, function(x) paste(x, collapse="/"))
```

Be careful because these apply functions are messed up by NAs. Create an NA value to see this:

```r
> data[1, 3] <- NA
> apply(data[, -(1:2)], 2, mean)
```

This is easily fixed with the `na.rm` option:

```r
> apply(data[, -(1:2)], 2, mean, na.rm = T)
```

tapply

A really useful function for summarizing data (hint: you can use it for the exercises today).

Find the mean weight for every treatment in the data:

```r
> tapply(data$weight, data$treatment, mean, na.rm = T)
```

The first argument to `tapply` is the variable we want to do something with. The second is the "splitting" variable(s), and the third is the function we want to apply. In this case, we separate data by treatments then find the mean of the weights in each of those treatments (removing NAs in the process). The second argument can be multiple variables:

```r
> tapply(data$weight, list(data$treatment, data$individual), mean, na.rm = T)
```

Here we find the mean weights for every individual/treatment combination (which doesn’t make much sense for this data set, but you get the point).

`aggregate()` is like `tapply` with different syntax and different output. This following is equivalent to the last `tapply()` statement, but outputs a data frame, which is often more useful:

```r
> aggregate(data$weight, data[, c("treatment", "individual")], mean, na.rm = T)
```

You might also check the help files for `lapply` and `sapply`, which are useful for lists.
Subsetting Data in R

For in-class practice, we’ll be doing Chapter 3 from Zuur et al, 2009.

→ Download the chapter and the zipped data files and scripts

Follow along with the chapter text and exercises in BGR_chap3.R and try to answer the questions I have embedded in that script. Where appropriate, try to use the apply-type functions we went over together to do the exercises.

Homework

1. Finish BGR chap 3 and compare your answers to the Exercises to the answer script. Be sure you understand the basics of subsetting!