CHAPTER 6

Packaged Functions

R has over 10,000 packages, most of which contain functions. Functions are at the heart of R and provide R with R's great versatility. Functions are R objects, and they are of both `mode` and `class` function. Packaged functions are functions that have been created as a part of an R package. On the computer, packages are stored in libraries and are installed to be in a library.

The Libraries

When R is initially installed, currently, the packages `base`, `boot`, `class`, `cluster`, `codetools`, `compiler`, `datasets`, `foreign`, `graphics`, `grDevices`, `grid`, `KernSmooth`, `lattice`, `MASS`, `Matrix`, `methods`, `mgcv`, `nlme`, `nnet`, `parallel`, `rpart`, `spatial`, `splines`, `stats`, `stats4`, `survival`, `tcltk`, `tools`, and `utils` are also installed in a folder on the hard drive.

In Windows, any packages installed after the initial installation can be installed in a different library, in another folder. The folder is created when R is installed. In OS X, all installed packages are in the same folder and library. In Linux, any packages installed after the initial installation are installed in a different library, in another folder. The folder is created when the first extra package is installed.

To see a listing of the installed packages with descriptions of each package and the names of the package folders, enter ***library=(lib.loc = .Library)*** at the R prompt. Running the function `library()` with no arguments lists the packages, with descriptions, in the libraries. To view much more information about the packages, enter ***installed.packages()*** at the R prompt. In R Studio, the installed packages are listed under the Packages tab in the lower right window.

Some R functions require other R functions to run. When R is running, only those packages that have been loaded into R from the libraries are accessible to the program. R gives an error if an attempt is made to run a function where a necessary package(s) has not been loaded. Included in the error message are the name(s) of the missing package(s). If a package exists in one of the libraries on the computer, the package can be loaded (made accessible) by entering ***library('package name')*** at the R prompt, where *'package name'* is the name of the package. Or, in R Studio, you can put a checkmark in the box to the left of the package under the Packages tab to load the package.

If the package is not in one of the libraries, installing new packages is straightforward (see Chapter 1). Once installed, the package can be loaded using the `library()` function or by using the Packages tab in R Studio. (If called from inside a function use `require()` instead of `library{}`. See the `library()` help page.) At any given time, entering **search()** at the R prompt gives a list of the packages that are loaded in the workspace. In R Studio, which packages are loaded are those checkmarked.

To see the functions (and datasets) in a package, enter **help(package=*'package name'*)** at the R prompt, where *'package name'* is the name of the package. Note that the package must be installed for **help(package='package name')** to return the contents of the package. In R Studio, entering the package name under the Help tab will give access to all of the objects in the package. Some of the files in a package may be datasets, but for most packages the files are generally functions.

Default Packages and Primitive Functions

When a user starts an R session, the packages `base`, `datasets`, `utils`, `grDevices`, `graphics`, `stats`, and `methods` are the default packages to be loaded into the workspace. (Which default packages are loaded can be changed by changing **defaultPackages** in the function `options()`. See Chapter 15.) Often, depending on the computing needs of the user, no more packages are needed.

Functions that are written in C and compiled at the time R is compiled are called **primitive** functions. According to the help page found by entering **?primitive** at the R prompt or by entering primitive under the R Studio Help tab, all primitive functions are in the package base, which is always loaded. The advantage of using primitive functions is that the functions are already compiled, so the functions run faster. The primitive functions include the operators and most of the mathematical functions as well as functions basic to the running and structure of R. A list of the primitive functions can be found at `http://cran.r-project.org/doc/manuals/R-ints.html#g_t_002eInternal-vs-_002ePrimitive` or under the help page for base. Primitive functions are of type built-in or special, depending on how the argument(s) are handled. Functions that are written in R are of type closure. The function is.primitive() tests whether an object is a primitive function.

Using the Help Pages

Each function in R has a help page, and each help page has essentially the same structure. Like much else in R, the help pages can be daunting at first. However, the help pages often contain a wealth of information.

Given the name of a function, if the package containing the function has been loaded, entering **?*function*** or **help(*function*)** at the R prompt, where *function* is the name of the function, brings up the help page for the function. If the package has been installed but not loaded, entering ?*package::name*, where *package* is the name of the package and *name* is the name of the function, brings up the help page. In R Studio, the help page can also be accessed under the Help tab by entering the name in the Help tab search box.

Some functions share the same help page. The help page can be brought up using any of the function names. In Windows and OS X R, the help pages open in a separate window. In Linux, the help pages display in the terminal. In R Studio, the help pages open in the lower right window.

Identifier

The first line of the help page lists the function name, followed by the function package in curly brackets, then the text "R Documentation."

Title

Below the identifier is a title that says something about the function(s). For example, for the function `lm()`, the title is "Fitting Linear Models."

Description

Below the title is a description of how the function(s) is used, headed by the word "Description." The description can be long or short, depending on the complexity of the function(s). For the function `lm()`, you will find the following description:

`lm` *is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although aov may provide a more convenient interface for these).*

Usage

The section "Usage" is found below the description. In the "Usage" section, the function(s) is listed with all of the possible arguments to the function(s). For arguments with default values, the default values are given. The Usage section lists the S4 usage. For many functions, S3 usages are also listed.

For the function `lm()`, the "Usage" section contains the following:

```
lm(formula, data, subset, weights, na.action, method = "qr",
model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
singular.ok = TRUE, contrasts = NULL, offset, ...)
```

The arguments with default values are the arguments for which the arguments have been set equal to a value.

Arguments

Below the "Usage" section is a section entitled "Arguments." In the "Arguments" section, the arguments found in the "Usage" section are listed with a description of each argument. The description includes the legal values for the argument.

For example, from the `lm()` help page, the first two arguments listed are as follows:

formula	*an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under "Details."*
data	*an optional data frame, list, or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from* `environment(formula)`*, typically the environment from which* `lm` *is called.*

So, for the function `lm()`, the first argument is a formula, and the second argument can be a `data.frame`, but the second argument is optional.

Details

Sometimes, there is a section entitled "Details," which gives details related to the arguments. In the `lm()` function example, the section on details gives the rules for setting up a formula and how the function behaves for differing inputs to the formula.

Value

The next section is entitled "Value." The "Value" section gives a description of what is returned from the function(s). For some functions, what functions can operate on the output and what components can be subsetted from the output are relevant and listed in this section.

The first few lines of the "Value" section for the function `lm()` are as follows:

`lm` *returns an object of class* `"lm"` *or for multiple responses of class* `c("mlm", "lm")`.

The functions `summary` *and* `anova` *are used to obtain and print a summary and analysis of variance table of the results. The generic accessor functions* coefficients, effects, fitted.values*, and* residuals *extract various useful features of the value returned by* `lm`.

An object of class `"lm"` *is a list containing at least the following components:*

`coefficients` *a named vector of coefficients*

`residuals` *the residuals, that is response minus fitted values.*

…

Some Other Optional Sections

Following the “Value” section, there may be other sections giving more information. For the function `lm()`, there are three other sections: “Using time series,” “Note,” and “Author(s).” Some sections for other functions might be “Warning,” “Source,” or other headings.

References

The next section is called “References.” The “References” section gives references to books and articles related to the method, both for more information and for how the method was derived.

For the function `lm()`, the “References” section contains

Chambers, J. M. (1992) Linear models. Chapter 4 of Statistical Models in S eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Wilkinson, G. N. and Rogers, C. E. (1973). Symbolic descriptions of factorial models for analysis of variance. *Applied Statistics*, **22**, 392-399. doi: 10.2307/2346786.

See Also

The section “See Also” follows the “References” section. The “See Also” section gives information about other functions related to the help page function(s). For the function `lm()`, the first three lines of the “See Also” section are the following:

summary.lm *for summaries and* anova.lm *for the ANOVA table;* aov *for a different interface.*

The generic functions coef, effects, residuals, fitted, vcov.

`predict.lm` *(via* `predict`*) for prediction, including confidence and prediction intervals;* `confint` *for confidence intervals of parameters.*

The "See Also" section is a good source for clues to functions related to the method the user is applying.

Examples

The final section, which most pages have, is "Examples." The "Examples" section gives examples of the use of the function(s). Seeing actual examples of usage can be very helpful. From the help page of the function `lm()`, part of the example includes the following:

```
require(graphics)

## Annette Dobson (1990) "An Introduction to Generalized Linear
Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
weight <- c(ctl, trt)
lm.D9 <- lm(weight ~ group)
lm.D90 <- lm(weight ~ group - 1) # omitting intercept

anova(lm.D9)
summary(lm.D90)
```

In this example, the structure of a formula is shown rather than explained. Some of the functions that operate on an object of class `lm` are also shown. Since the package `graphics` is loaded by default, the call to **require(graphics)** would not normally be necessary.