Packages

2022-06-26

## Terminology

#### Packages and libraries

Packages are a collection of R functions, data, and compiled code.

A library is the location where the packages are stored on your computer.

Usage: “Install the {tidyverse} package from CRAN. It will automatically save to your R library.”

#### CRAN

CRAN stands for the Comprehensive R Archive Network. Along with providing downloads and updates to the base R computing software, CRAN is the main source for stable, published versions of packages. The function install.packages() automatically looks in CRAN for whatever package you name inside the function.

Using other installation functions, it’s also possible to download packages from various other sources, like GitHub. But most of the time, those packages are still in development, which means various features could become inoperable over time (as opposed to making defunct code backwards-compatible). Getting packages from CRAN is the best way to ensure that your code will still run months and sometimes years into the future

#### Objects, functions, and arguments

**Object** refers to any data structure in R, with special attributes and methods according to the object type. Most objects are open to exploration and transformation.

***Examples of objects:*** Data frames, linear models, vectors, tables, variables, ggplot objects, matrices, etc.

**Functions** are the verbs of the R language. They operate on objects in your R environment, and behave according to their arguments. On this website, functions are written in monospace font and followed by a pair of parentheses, ()

***Examples of functions:*** mean(), summary(), factor(), exp()

**Arguments** are the settings and information supplied to a given function. Each new argument should be separated from the rest with a comma.

Some functions don’t require any arguments, like R.Version() and sessionInfo(). These functions, when run, supply information about the R environment, but don’t operate on an object.

Other functions only require a single argument: an object. For example, the function head() takes many different types of objects, and supplies the first 6 observations within a given object.

But say you want a different number of observations. head() allows you to manually set those observations with n =.

Typing head(object, n = 10) will return the first 10 observations in a given object.

Most functions come with default arguments that are submitted to the function implicitly. In the case of head(), the default method for n = is 6.

ALL arguments can be specified with the syntax argumentName = value. But most functions are written to recognize certain types of values as belonging to a specific argument, and operate on them accordingly.

So head(object, 10) will also give the first 10 observations of object, since head() knows to recognize an object of type integer as belonging to n =.

You can view any function’s arguments by going to its documentation. In the R console, run ?functionName, and it will appear in the help window. Under the “Usage” section, you will find information on the function’s syntax and arguments. If an argument is alone, without an equals sign, =, that argument doesn’t have default and must be submitted by the user to run. However, if a given argument contains an equals sign and a value to the right of the equals sign, that value is the argument’s default.

#### Documentation

Any function or package’s documentation is the primary authority on the appropriate usage of that function. Typically, a function’s documentation will contain:

* A brief description of the function
* An overview of the functin’s usage
* A detailed description of each argument needed for the function to run
* Further function details
* Value, which indicates what gets returned when the function is run
* Examples of the function in use

The documentation for any function is available via the R console by running ?functionName.

Sometimes, you can find documentation that goes into greater detail by searching the function’s documentation online.

When you are having problems with your code, it’s smart to check the function’s documentation before looking elsewhere. With practice and a little persistence, you will become more comfortable with reading and interpreting help documentation.

## Installing packages

Packages are collections of functions. As we’ll see shortly, we use functions as code to view, manipulate, and analyze our data.

There are packages that come built-in with R. These have names like {base}, {utils}, and {stats}.

Since R is open-source, users are able to create their own packages so that other R users can use them. These packages are available in places like [the Comprehensive R Archive Network (CRAN for short)](https://cran.r-project.org/web/packages/) and [GitHub](https://github.com/).

Lucky for us, packages are easily retrieved from the R Console. If you haven’t already, run the following code from your console to download the packages that we’ll be needing for Fall Semester. **You only need to do this once**:

install.packages("tidyverse")  
install.packages("skimr")  
install.packages("epiR")  
install.packages("devtools")  
devtools::install\_github("potato-nathan/epiAssist")

## Loading libraries

To enable R to use a package’s functions in our current project environment, we need to load the packages using the library() function.

In a fresh code chunk, call in the {tidyverse} and {skimr} packages using the following code:

library(tidyverse)  
library(skimr)

**Notice that when we install packages, we need to specify their names using quotes, but when we load them into R using library(), R recognizes them as package objects automatically, and so don’t need quotes. You can surround them with quotes and it will load all the same.**