

Using packages and writing functions in R

Sep. 15, 2006

1 Setting up

Choose a subdirectory to use for this R session. Go to the “Datasets” section of the course web page. Read the file called `Bap.info`, and download the data file called `Bap.txt` into the subdirectory you wish to use. Then call up R from that subdirectory.

2 Using on-line help in R

You can get help on any R function by typing `help (<command name>)`. For example, to get help on the library function, enter

```
> help(library)
```

To use “prettier” help in separate window, enter

```
> help.start()
```

After a while, a web window will come up. You can click on choices for help documents in that window. If you type `help (<command name>)` in the command window later in the R session, the results will appear in the web window.

Another function that is useful in learning R and getting help is `apropos`. It finds all functions whose *names* contain the character string given as the argument. Only packages that have been loaded into memory are searched. For example,

```
> apropos(rank)
```

Yet another useful function is `help.search`. It looks for any function in any installed package that mentions the search term in its help. The name of the package containing the function is in parentheses. For example,

```
> help.search("rank")
```

Help files with alias or concept or title matching rank using regular expression matching:

```
rank(base)           Sample Ranks
SignRank(stats)     Distribution of the Wilcoxon Signed Rank
                    Statistic
.                   .
.                   .
.                   .
```

3 Using built-in R datasets

Use the `search` function to determine which R packages are loaded automatically when you bring R up.

```
> search()
[1] ".GlobalEnv"          "package:methods"    "package:stats"
[4] "package:graphics"   "package:grDevices"  "package:utils"
[7] "package:datasets"   "Autoloads"          "package:base"
```

Notice that the `datasets`, `stats`, and `graphics` packages are listed. This means you can access any of the built-in R datasets and any functions in the `stats` and `graphics` packages without having to load the packages yourself. To get names and descriptions of the datasets, enter

```
> help(package=datasets)
```

To display the Orange dataset (it is a data frame), just enter

```
> Orange
```

To get descriptive information on the dataset named Orange, enter

```
> help(Orange, package=datasets)
```

Notice the example code at the end of the help. Copy the line that begins `coplot` into the command window and execute it. This is an example of a very powerful plotting function in the `graphics` package. Type in the necessary command to get help on this function.

4 Reading in an external file

To read the BaP data into a data frame called `Bap`, enter

```
BaP <- read.table("BaP.txt", header = T)
```

If the data file was in a different subdirectory, we would have to enter its full path name.

To get a scatterplot with the indoor measurements on the X axis and the outdoor measurements on the Y axis, enter:

```
plot(BaP$indoor, BaP$outdoor)
```

If we want to refer to the variables `indoor` and `outdoor` without referencing the dataframe, we need to `attach` the dataframe.

```
attach(BaP)
```

Note that this makes `BaP` the second item in the search list.

```
search()
```

Now we could just enter `plot(indoor, outdoor)`.

When we are done using the data frame, we should `detach` it to free up memory.

```
detach(BaP)
```

Use `search()` to verify that the `BaP` data frame has been detached.

5 Writing R programs

The safest way to write an R program in the Unix or Linux environment, is to open a terminal window (separate from the one in which you are using R) and from there, use a text editor to write the program code and save it as a plain text file with the extension `.R`. You may wish to develop the code interactively in R and copy the lines into the program file in the text editor. You can then run the program in R using the `source` function, for example, if the program was named “`dostuff.R`,”

```
> source("dostuff.R")
```

6 Writing R functions

Similarly, to write an R function that contains more than a few lines of code, write the code in a text editor outside of R and save it in a text file. Then you can use the `source` function in R to read in the function and assign it to an R object.

We will write a function that does the following:

1. Accepts a vector as an argument
2. If the vector is numeric and has at least 20 distinct values, returns the quantiles of the values
3. Otherwise returns a tabulation of the values

Use your text editor to create a plain text file named “`process.R`” that contains the following code.

```
function(x)
{
  if ( is.character(x) | is.factor(x)
      | (is.numeric(x) & length( unique(x) ) < 20 ))
      table(x)
  else
      quantile(x)
}
```

Now read it into an R function by entering the following in R:

```
> process <- source("process.R")$value
```

If R reports syntax errors, correct them in the text file; then try `source` again.

Test your function by creating different types of vectors and using them as arguments. For example,

```
myvect <- c(rep("a",5), rep("w",2), "n", "q", "v")
process(myvect)
```

```
myvect <- rnorm(10)
process(myvect)
```

```
myvect <- rnorm(500)
process(myvect)
```

7 R functions that return lists of values

Often a function must return several different kinds of values. It is convenient to put them into a list object.

Create a text file containing the following code:

```
function(x, y) {
  if ( !is.numeric(x) | ! is.numeric(y) | length(x) != length(y) )
    cat("Arguments must be two numeric vectors of equal length \n" )
  else {
    lmout <- lm(y ~ x)
    sumstatsx <- summary(x)
    sumstatsy <- summary(y)
    coefs <- lmout$coefficients
    list( statsx = sumstatsx, statsy = sumstatsy, coefs = coefs)
  }
}
```

Source it into an R function called “`twovars`” and test it.