Say it in R with "by", "apply" and friends

28 Jan 2012 14:43 aggregate , apply , by , data.table , doBy , language , plyr , R , sqldf , Tutorials

Iris versicolor

By Danielle Langlois

License: CC-BY-SA

R is a language, as Luis Apiolaza pointed out in his recent post. This is absolutely true, and learning a programming language is not much different from learning a foreign language. It takes time and a lot of practice to be proficient in it. I started using R when I moved to the UK and I wonder, if I have a better understanding of English or R by now.

Languages are full of surprises, in particular for non-native speakers. The other day I learned that there is *courtesy* and *curtsey*. Both words sounded very similar to me, but of course created some laughter when I mixed them up in an email.

With languages you can get into habits of using certain words and phrases, but sometimes you see or hear something, which shakes you up again. So did the following two lines in R with me:

```
f <- function(x) x^2
sapply(1:10, f)
[1]   1   4   9  16  25  36  49  64  81 100
```

It reminded me of the phrase that *everything is a list* in R. It showed me again how easily a *for* loop can be turned into a statement using the `apply` family of functions and how little I know about all the subtleties of R. I remember how happy I felt, when I finally understood the `by` function in R. I started to use it all the time, closing my eyes on `aggregate` and the `apply` functions family. Here is an example where I calculate the means of the various measurements by species of the famous iris data set using `by`.

by

```
do.call("rbind", as.list(
  by(iris, list(Species=iris$Species), function(x){
    y <- subset(x, select= -Species)
    apply(y, 2, mean)
  }
)))

           Sepal.Length Sepal.Width Petal.Length Petal.Width
setosa            5.006       3.428        1.462       0.246
versicolor        5.936       2.770        4.260       1.326
virginica         6.588       2.974        5.552       2.026
```

Now let's find alternative ways of expressing ourselves, using other words/functions of the R language, such as `aggregate, apply, sapply, tapply, data.table, ddply, sqldf`, and `summaryBy`.

aggregate

The `aggregate` function splits the data into subsets and computes summary statistics for each of them. The output of `aggregate` is a `data.frame`, including a column for species.

```
iris.x <- subset(iris, select= -Species)
iris.s <- subset(iris, select= Species)
aggregate(iris.x, iris.s, mean)

     Species Sepal.Length Sepal.Width Petal.Length Petal.Width
1     setosa        5.006       3.428        1.462       0.246
```

```
2 versicolor        5.936       2.770        4.260       1.326
3  virginica        6.588       2.974        5.552       2.026
```

Addition: As John Christie points out in the comments, `aggregate` has also a formula interface, which simplifies the call to:

```
aggregate( . ~ Species, iris, mean)
```

apply and tapply

The combination of `tapply` and `apply` achieves a similar result, but this time the output is a `matrix` and hence I lose the column with species. The species are now the row names.

```
apply(iris.x, 2, function(x) tapply(x, iris.s, mean))

           Sepal.Length Sepal.Width Petal.Length Petal.Width
setosa            5.006       3.428        1.462       0.246
versicolor        5.936       2.770        4.260       1.326
virginica         6.588       2.974        5.552       2.026
```

split and apply

Here I split the data first into subsets for each of the species and calculate then the mean for each column in the subset. The output is a `matrix` again, but transposed.

```
sapply(split(iris.x, iris.s), function(x) apply(x, 2, mean))

             setosa versicolor virginica
Sepal.Length  5.006      5.936     6.588
Sepal.Width   3.428      2.770     2.974
Petal.Length  1.462      4.260     5.552
Petal.Width   0.246      1.326     2.026
```

ddply

Hadley Wickham's `plyr` package provides tools for splitting, applying and combining data. The function `ddply` is similar to the by function, but it returns a `data.frame` instead of a `by` list and maintains the column for the species.

```
library(plyr)
ddply(iris, "Species", function(x){
    y <- subset(x, select= -Species)
    apply(y, 2, mean)
  })

     Species Sepal.Length Sepal.Width Petal.Length Petal.Width
1     setosa        5.006       3.428        1.462       0.246
2 versicolor        5.936       2.770        4.260       1.326
3  virginica        6.588       2.974        5.552       2.026
```

Addition: Sean mentions in the comments an alternative, using the `colMeans` function, while Andrew reminds us of the `reshape` package with its functions `melt` and `cast`.

```
ddply(iris, "Species", function(x) colMeans(subset(x, select= -Species)))
## or
ddply(iris, "Species", colwise(mean))
## same output as above
library(reshape)
cast(melt(iris, id.vars='Species'),formula=Species ~ variable,mean)
## same output as above
```

summaryBy

The `summaryBy` function of the `doBy` package by Søren Højsgaard and Ulrich Halekoh has a very intuitive interface, using formulas.

```
library(doBy)
summaryBy(Sepal.Length + Sepal.Width + Petal.Length + Petal.Width ~ Species, data=iris, FUN=mean)

     Species Sepal.Length.mean Sepal.Width.mean Petal.Length.mean Petal.Width.mean
1     setosa             5.006            3.428             1.462            0.246
2 versicolor             5.936            2.770             4.260            1.326
3  virginica             6.588            2.974             5.552            2.026
```

sqldf

If you are fluent in SQL, then the sqldf package by Gabor Grothendieck might be the one for you.

```
library(sqldf)
sqldf("select Species, avg(Sepal_Length), avg(Sepal_Width),
    avg(Petal_Length), avg(Petal_Width) from iris
    group by Species")

     Species avg(Sepal_Length) avg(Sepal_Width) avg(Petal_Length) avg(Petal_Width)
1     setosa             5.006            3.428             1.462            0.246
2 versicolor             5.936            2.770             4.260            1.326
3  virginica             6.588            2.974             5.552            2.026
```

data.table

The `data.table` package by M Dowle, T Short and S Lianoglou is the real rock star to me. It provides an elegant and fast way to complete the task. The statement reads in plain English from right to left: take columns 1 to 4, split them by the factor in column "Species" and calculate on the **s**ub **d**ata (`.SD`) the means.

```
library(data.table)
iris.dt <- data.table(iris)
iris.dt[,lapply(.SD,mean),by="Species",.SDcols=1:4]

        Species Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,]     setosa        5.006       3.428        1.462       0.246
[2,] versicolor        5.936       2.770        4.260       1.326
[3,]  virginica        6.588       2.974        5.552       2.026
```

apply

I should mention that R provides the `iris` data set also in an array form. The third dimension of the `iris3` array holds the species information. Therefore I can use the `apply` function again, I go down the third and then the second dimension to calculate the means.

```
apply(iris3, c(3,2), mean)

           Sepal L. Sepal W. Petal L. Petal W.
Setosa        5.006    3.428    1.462    0.246
Versicolor    5.936    2.770    4.260    1.326
Virginica     6.588    2.974    5.552    2.026
```

Conclusion

Many roads lead to Rome, and there are endless ways of explaining how to get there. I only showed a few I know of, and I am curious to hear yours. As a matter of *courtesy* I should mention the unknownR package by Matthew Dowle. It helps you to discover what you don't know that you don't know in R. Thus, it can help to build your R vocabulary. Of course there is a key difference between R and English. R tells me right away when I make a mistake. Human readers are far more forgiving, but please do point out to me where I made mistakes. I am still hopeful that I can improve, but I need your help.

R code

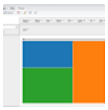
The R code of the examples is available on github. For more examples on the apply family see also Neil Saunders' post.

Newer Post | Older Post

FOLLOW BY EMAIL

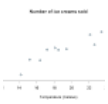
Email address... Submit

POPULAR POSTS



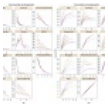
Interactive pivot tables with R

I love interactive pivot tables . That is the number one reason why I keep using spreadsheet software. The ability to look at data quickly ...



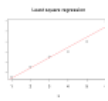
Generalised Linear Models in R

Linear models are the bread and butter of statistics, but there is a lot more to it than taking a ruler and drawing a line through a couple ...



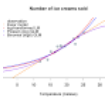
Data.table rocks! Data manipulation the fast way in R

I really should make it a habit of using data.table . The speed and simplicity of this R package are astonishing. Here is a simple exampl...



How to use optim in R

A friend of mine asked me the other day how she could use the function optim in R to fit data. Of course there are built in functions for f...



Visualising theoretical distributions of GLMs

Two weeks ago I discussed various linear and generalised linear models in R using ice cream sales statistics. The data showed not surprisin...

Say it in R with "by", "apply" and friends

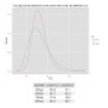
Iris versicolor By Danielle Langlois License: CC-BY-SA R is a language , as Luis Apiolaza pointed out in his recent post . This is ab...

First steps of using googleVis on shiny

The guys at RStudio have done a fantastic job with shiny . It is really easy to build web apps with R using shiny. With the help of Joe Ch...

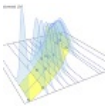
Changing colours and legends in lattice plots

Lattice plots are a great way of displaying multivariate data in R. Deepayan Sarkar, the author of lattice, has written a fantastic book ab...



Plotting tables alsongside charts in R

Occasionally I'd like to plot a table alongside a chart in R, e.g. to present summary statistics of the graph itself. Thanks to the grid...



Visualising the predictive distribution of a log-transformed linear model

Last week I presented visualisations of theoretical distributions that predict ice cream sales statistics based on linear and generalised l...

BLOG ARCHIVE

MY BLOG LIST