

# Chapter 19: Split function

## Section 19.1: Using split in the split-apply-combine paradigm

A popular form of data analysis is [split-apply-combine](#), in which you split your data into groups, apply some sort of processing on each group, and then combine the results.

Let's consider a data analysis where we want to obtain the two cars with the best miles per gallon (mpg) for each cylinder count (cyl) in the built-in mtcars dataset. First, we split the `mtcars` data frame by the cylinder count:

```
(spl <- split(mtcars, mtcars$cyl))
# $`4`
#           mpg cyl disp hp drat    wt  qsec vs am gear carb
# Datsun 710 22.8   4 108.0 93 3.85 2.320 18.61  1  1    4    1
# Merc 240D  24.4   4 146.7 62 3.69 3.190 20.00  1  0    4    2
# Merc 230   22.8   4 140.8 95 3.92 3.150 22.90  1  0    4    2
# Fiat 128   32.4   4  78.7 66 4.08 2.200 19.47  1  1    4    1
# ...
#
# $`6`
#           mpg cyl disp hp drat    wt  qsec vs am gear carb
# Mazda RX4  21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
# Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
# Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
# Valiant    18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
# ...
#
# $`8`
#           mpg cyl disp hp drat    wt  qsec vs am gear carb
# Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
# Duster 360     14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
# Merc 450SE     16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
# Merc 450SL     17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
# ...
```

This has returned a list of data frames, one for each cylinder count. As indicated by the output, we could obtain the relevant data frames with `spl$`4``, `spl$`6``, and `spl$`8`` (some might find it more visually appealing to use `spl$"4"` or `spl[["4"]]` instead).

Now, we can use `lapply` to loop through this list, applying our function that extracts the cars with the best 2 mpg values from each of the list elements:

```
(best2 <- lapply(spl, function(x) tail(x[order(x$mpg), ], 2)))
# $`4`
#           mpg cyl disp hp drat    wt  qsec vs am gear carb
# Fiat 128   32.4   4  78.7 66 4.08 2.200 19.47  1  1    4    1
# Toyota Corolla 33.9   4  71.1 65 4.22 1.835 19.90  1  1    4    1
#
# $`6`
#           mpg cyl disp hp drat    wt  qsec vs am gear carb
# Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
# Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
#
# $`8`
#           mpg cyl disp hp drat    wt  qsec vs am gear carb
# Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
# Pontiac Firebird 19.2   8  400 175 3.08 3.845 17.05  0  0    3    2
```

Finally, we can combine everything together using `rbind`. We want to call `rbind(best2[["4"]], best2[["6"]], best2[["8"]])`, but this would be tedious if we had a huge list. As a result, we use:

```
do.call(rbind, best2)
#          mpg cyl disp hp drat wt qsec vs am gear carb
# 4.Fiat 128    32.4   4 78.7 66 4.08 2.200 19.47 1 1 4 1
# 4.Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90 1 1 4 1
# 6.Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02 0 1 4 4
# 6.Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44 1 0 3 1
# 8.Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02 0 0 3 2
# 8.Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05 0 0 3 2
```

This returns the result of `rbind` (argument 1, a function) with all the elements of `best2` (argument 2, a list) passed as arguments.

With simple analyses like this one, it can be more compact (and possibly much less readable!) to do the whole split-apply-combine in a single line of code:

```
do.call(rbind, lapply(split(mtcars, mtcars$cyl), function(x) tail(x[order(x$mpg), ], 2)))
```

It is also worth noting that the `lapply(split(x, f), FUN)` combination can be alternatively framed using the `?by` function:

```
by(mtcars, mtcars$cyl, function(x) tail(x[order(x$mpg), ], 2))
do.call(rbind, by(mtcars, mtcars$cyl, function(x) tail(x[order(x$mpg), ], 2)))
```

## Section 19.2: Basic usage of split

`split` allows to divide a vector or a data.frame into buckets with regards to a factor/group variables. This ventilation into buckets takes the form of a list, that can then be used to apply group-wise computation (`for` loops or `lapply/sapply`).

First example shows the usage of `split` on a vector:

Consider following vector of letters:

```
testdata <- c("e", "o", "r", "g", "a", "y", "w", "q", "i", "s", "b", "v", "x", "h", "u")
```

Objective is to separate those letters into vowels and consonants, ie split it accordingly to letter type.

Let's first create a grouping vector:

```
vowels <- c('a', 'e', 'i', 'o', 'u', 'y')
letter_type <- ifelse(testdata %in% vowels, "vowels", "consonants")
```

Note that `letter_type` has the same length that our vector `testdata`. Now we can `split` this test data in the two groups, vowels and consonants :

```
split(testdata, letter_type)
#$consonants
#[1] "r" "g" "w" "q" "s" "b" "v" "x" "h"

#$vowels
#[1] "e" "o" "a" "y" "i" "u"
```

Hence, the result is a list which names are coming from our grouping vector/factor `letter_type`.

`split` has also a method to deal with data.frames.

Consider for instance `iris` data:

```
data(iris)
```

By using `split`, one can create a list containing one data.frame per iris specie (variable: Species):

```
> liris <- split(iris, iris$Species)
> names(liris)
[1] "setosa"      "versicolor"   "virginica"
> head(liris$setosa)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1        3.5         1.4        0.2    setosa
2          4.9        3.0         1.4        0.2    setosa
3          4.7        3.2         1.3        0.2    setosa
4          4.6        3.1         1.5        0.2    setosa
5          5.0        3.6         1.4        0.2    setosa
6          5.4        3.9         1.7        0.4    setosa
```

(contains only data for setosa group).

One example operation would be to compute correlation matrix per iris specie; one would then use `lapply`:

```
> (lcor <- lapply(liris, FUN=function(df) cor(df[,1:4])))

$setosa
  Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length  1.0000000  0.7425467  0.2671758  0.2780984
Sepal.Width   0.7425467  1.0000000  0.1777000  0.2327520
Petal.Length  0.2671758  0.1777000  1.0000000  0.3316300
Petal.Width   0.2780984  0.2327520  0.3316300  1.0000000

$versicolor
  Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length  1.0000000  0.5259107  0.7540490  0.5464611
Sepal.Width   0.5259107  1.0000000  0.5605221  0.6639987
Petal.Length  0.7540490  0.5605221  1.0000000  0.7866681
Petal.Width   0.5464611  0.6639987  0.7866681  1.0000000

$virginica
  Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length  1.0000000  0.4572278  0.8642247  0.2811077
Sepal.Width   0.4572278  1.0000000  0.4010446  0.5377280
Petal.Length  0.8642247  0.4010446  1.0000000  0.3221082
Petal.Width   0.2811077  0.5377280  0.3221082  1.0000000
```

Then we can retrieve per group the best pair of correlated variables: (correlation matrix is reshaped/melted, diagonal is filtered out and selecting best record is performed)

```
> library(reshape)
> (topcor <- lapply(lcor, FUN=function(cormat){
  correlations <- melt(cormat, variable_name="correlation");
  filtered <- correlations[correlations$X1 != correlations$X2,];
  filtered[which.max(filtered$correlation),]
}))
```

```

$setosa
      X1          X2    correlation
2 Sepal.Width Sepal.Length      0.7425467

$versicolor
      X1          X2    correlation
12 Petal.Width Petal.Length     0.7866681

$virginica
      X1          X2    correlation
3 Petal.Length Sepal.Length    0.8642247

```

Note that one computations are performed on such groupwise level, one may be interested in stacking the results, which can be done with:

```

> (result <- do.call("rbind", topcor))

      X1          X2    correlation
setosa   Sepal.Width Sepal.Length      0.7425467
versicolor  Petal.Width Petal.Length     0.7866681
virginica  Petal.Length Sepal.Length    0.8642247

```