I recently had a question from a client about the simplest way to subset a data.frame and apply a function to each subset. “Simplest” could mean many things, of course, since what is simple for one person could appear very difficult to another. In this specific case I suggested using base::split() as a possible option since it is one I find fairly approachable.

I turns out I don’t have a go-to example for how to get started with a split() approach. So here’s a quick blog post about it! 

**Load R packages**

I’ll load **purrr** for looping through lists.

library(purrr) # 0.3.3

**A dataset with groups**

I made a small dataset to use with split(). The id variable contains the group information. There are three groups, a, b, and c, with 10 observations per group. There are also two numeric variables, var1 and var2.

dat = structure(list(id = structure(c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L,

1L, 1L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 3L, 3L, 3L, 3L,

3L, 3L, 3L, 3L, 3L, 3L), .Label = c("a", "b", "c"), class = "factor"),

var1 = c(4, 2.7, 3.4, 2.7, 4.6, 2.9, 2.2, 4.5, 4.6, 2.4,

3, 3.8, 2.5, 4, 3.6, 2.7, 4.5, 4.1, 4.2, 2.2, 4.9, 4.4, 3.6,

3.3, 2.7, 3.9, 4.9, 4.9, 4.3, 3.4), var2 = c(6, 22.3, 19.4,

22.8, 18.6, 14.2, 10.9, 22.7, 22.4, 11.7, 6, 13.3, 12.5,

6.3, 13.6, 20.5, 23.6, 10.9, 8.9, 20.9, 23.7, 15.9, 22.1,

11.6, 22, 17.7, 21, 20.8, 16.7, 21.4)), class = "data.frame", row.names = c(NA,

-30L))

head(dat)

# id var1 var2

# 1 a 4.0 6.0

# 2 a 2.7 22.3

# 3 a 3.4 19.4

# 4 a 2.7 22.8

# 5 a 4.6 18.6

# 6 a 2.9 14.2

**Create separate data.frames per group**

If the goal is to apply a function to each dataset in each group, we need to pull out a dataset for each id. One approach to do this is to make a subset for each group and then apply the function of interest to the subset. A classic approach would be to do the subsetting within a for() loop.

This is a situation where I find split() to be really convenient. It splits the data by a defined group variable so we don’t have to subset things manually.

The output from split() is a list. If I split a dataset by groups, each element of the list will be a data.frame for one of the groups. Note the group values are used as the names of the list elements. I find the list-naming aspect of split() handy for keeping track of groups in subsequent steps.

Here’s an example, where I split dat by the id variable.

dat\_list = split(dat, dat$id)

dat\_list

# $a

# id var1 var2

# 1 a 4.0 6.0

# 2 a 2.7 22.3

# 3 a 3.4 19.4

# 4 a 2.7 22.8

# 5 a 4.6 18.6

# 6 a 2.9 14.2

# 7 a 2.2 10.9

# 8 a 4.5 22.7

# 9 a 4.6 22.4

# 10 a 2.4 11.7

#

# $b

# id var1 var2

# 11 b 3.0 6.0

# 12 b 3.8 13.3

# 13 b 2.5 12.5

# 14 b 4.0 6.3

# 15 b 3.6 13.6

# 16 b 2.7 20.5

# 17 b 4.5 23.6

# 18 b 4.1 10.9

# 19 b 4.2 8.9

# 20 b 2.2 20.9

#

# $c

# id var1 var2

# 21 c 4.9 23.7

# 22 c 4.4 15.9

# 23 c 3.6 22.1

# 24 c 3.3 11.6

# 25 c 2.7 22.0

# 26 c 3.9 17.7

# 27 c 4.9 21.0

# 28 c 4.9 20.8

# 29 c 4.3 16.7

# 30 c 3.4 21.4

**Looping through the list**

Once the data are split into separate data.frames per group, we can loop through the list and apply a function to each one using whatever looping approach we prefer.

For example, if I want to fit a linear model of var1 vs var2 for each group I might do the looping with purrr::map() or lapply().

Each element of the new list still has the grouping information attached via the list names.

map(dat\_list, ~lm(var1 ~ var2, data = .x) )

# $a

#

# Call:

# lm(formula = var1 ~ var2, data = .x)

#

# Coefficients:

# (Intercept) var2

# 2.64826 0.04396

#

#

# $b

#

# Call:

# lm(formula = var1 ~ var2, data = .x)

#

# Coefficients:

# (Intercept) var2

# 3.80822 -0.02551

#

#

# $c

#

# Call:

# lm(formula = var1 ~ var2, data = .x)

#

# Coefficients:

# (Intercept) var2

# 3.35241 0.03513

I could also create a function that fit a model and then returned model output. For example, maybe what I really wanted to do is the fit a linear model and extract \(R^2\) for each group model fit.

r2 = function(data) {

fit = lm(var1 ~ var2, data = data)

broom::glance(fit)

}

The output of my r2 function, which uses broom::glance(), is a data.frame.

r2(data = dat)

# # A tibble: 1 x 11

# r.squared adj.r.squared sigma statistic p.value df logLik AIC BIC

#

# 1 0.0292 -0.00550 0.867 0.841 0.367 2 -37.3 80.5 84.7

# # ... with 2 more variables: deviance , df.residual

Since the function output is a data.frame, I can use purrr::map\_dfr() to combine the output per group into a single data.frame. The .id argument creates a new variable to store the list names in the output.

map\_dfr(dat\_list, r2, .id = "id")

# # A tibble: 3 x 12

# id r.squared adj.r.squared sigma statistic p.value df logLik AIC

#

# 1 a 0.0775 -0.0378 0.968 0.672 0.436 2 -12.7 31.5

# 2 b 0.0387 -0.0815 0.832 0.322 0.586 2 -11.2 28.5

# 3 c 0.0285 -0.0930 0.808 0.235 0.641 2 -10.9 27.9

# # ... with 3 more variables: BIC , deviance , df.residual

**Splitting by multiple groups**

It is possible to split data by multiple grouping variables in the split() function. The grouping variables must be passed as a list.

Here’s an example, using the built-in mtcars dataset. I show only the first two list elements to demonstrate that the list names are now based on a combination of the values for the two groups. By default these values are separated by a . (but see the sep argument to control this).

mtcars\_cylam = split(mtcars, list(mtcars$cyl, mtcars$am) )

mtcars\_cylam[1:2]

# $`4.0`

# mpg cyl disp hp drat wt qsec vs am gear carb

# 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

# Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1

#

# $`6.0`

# mpg cyl disp hp drat wt qsec vs am gear carb

# 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

# Merc 280 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4

# Merc 280C 17.8 6 167.6 123 3.92 3.440 18.90 1 0 4 4

If all combinations of groups are not present, the drop argument in split() allows us to drop missing combinations. By default combinations that aren’t present are kept as 0-length data.frames.

**Other thoughts on split()**

I feel like split() was a gateway function for me to get started working with lists and associated convenience functions like lapply() and purrr::map() for looping through lists. I think learning to work with lists and “list loops” also made the learning curve for list-columns in data.frames and the nest()/unnest() approach of analysis-by-groups a little less steep for me.

**Just the code, please**

Here’s the code without all the discussion. Copy and paste the code below or you can download an R script of uncommented code [from here](https://aosmith.rbind.io/script/2019-11-26-split-example.R).

library(purrr) # 0.3.3

dat = structure(list(id = structure(c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L,

1L, 1L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 3L, 3L, 3L, 3L,

3L, 3L, 3L, 3L, 3L, 3L), .Label = c("a", "b", "c"), class = "factor"),

var1 = c(4, 2.7, 3.4, 2.7, 4.6, 2.9, 2.2, 4.5, 4.6, 2.4,

3, 3.8, 2.5, 4, 3.6, 2.7, 4.5, 4.1, 4.2, 2.2, 4.9, 4.4, 3.6,

3.3, 2.7, 3.9, 4.9, 4.9, 4.3, 3.4), var2 = c(6, 22.3, 19.4,

22.8, 18.6, 14.2, 10.9, 22.7, 22.4, 11.7, 6, 13.3, 12.5,

6.3, 13.6, 20.5, 23.6, 10.9, 8.9, 20.9, 23.7, 15.9, 22.1,

11.6, 22, 17.7, 21, 20.8, 16.7, 21.4)), class = "data.frame", row.names = c(NA,

-30L))

head(dat)

dat\_list = split(dat, dat$id)

dat\_list

map(dat\_list, ~lm(var1 ~ var2, data = .x) )

r2 = function(data) {

fit = lm(var1 ~ var2, data = data)

broom::glance(fit)

}

r2(data = dat)

map\_dfr(dat\_list, r2, .id = "id")

mtcars\_cylam = split(mtcars, list(mtcars$cyl, mtcars$am) )

mtcars\_cylam[1:2]