January 18, 2018

RE: A few parting thoughts on pmap

Hi everyone,

I hope this e-mail finds you all well and sticking to your purrr resolution. The purrr function for Week 3 will be announced in a separate e-mail. For now, I wanted to reflect on my own experience with learning more about pmap() in the hope that you may find this reflection useful.

As we discovered last week, **pmap()** is useful in situations where we need to apply the **same operation** (e.g., constructing a plot, fitting a model, producing a confidence interval, computing a predicted value) simultaneously to all items which are collected in a **list of lists**.

This means that pmap() needs (at least) two inputs: a list of lists and a function which will encapsulate the operation that needs to be applied to each item included in the list of lists. Additional inputs include arguments which are needed for the function to work properly.

Lists can be confusing to understand in R but one conceptualization that can help with this is envisioning a list as a **jar which is used to store information**. The jar can store an entire data set, or a sequence of numbers, or a constant, or a plot, etc. The jar can have a label or it can be unlabelled. Most importantly, if the jar is big enough, it can include other smaller jars (some of which may themselves include even smaller jars, etc.). The smaller jars could also be either labelled or unlabelled.

So in thinking about the meaning of a **list of lists**, it helps to think about a **bigger jar which includes other smaller jars**. The bigger jar is what the pmap function will require us to specify first. Once we make that specification, pmap will apply our desired operation to the information content of each of the smaller jars nested in the bigger jar. In other words, pmap will count how many smaller jars are included in the bigger jars and then proceed to (simultaneously) apply that operation to the information content of each jar.

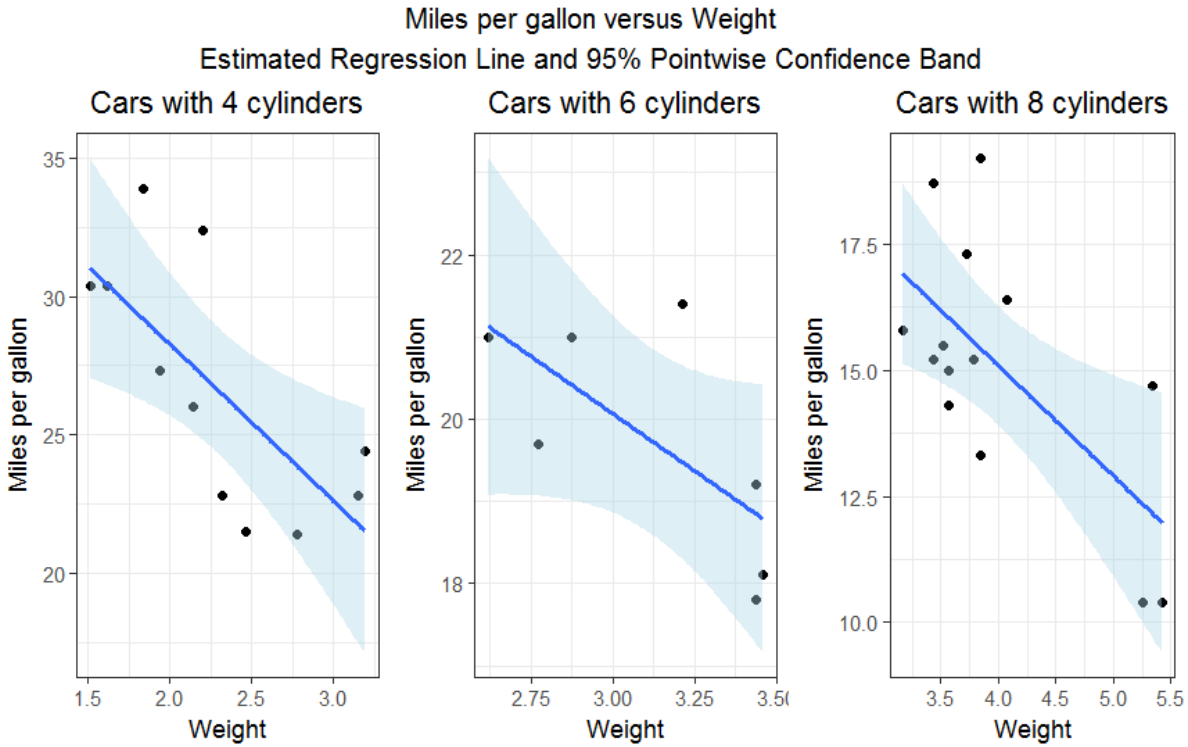
Imagine that each of the smaller jars contains a data set. If these data sets are collected together in a bigger jar, then we can apply the same type of model to each data set, provided we define a function which will use a generic data set as an input. However, in the body of the function, we will need to invoke variables stored in each of our actual data sets by their specific names.

As I was trying to read up more on pmap, one thing I found was that most of the examples available online seemed to be rather simplistic in that they assumed that each smaller jar contained a single number or a single string, etc. So I set out to create my own example, which I include below.

```
#=============================================================================
# Require necessary packages
#=============================================================================

require(dplyr)
require(ggplot2)
require(magrittr)
require(gridExtra)

#=============================================================================
# Create a set of 3 smaller "data" jars using the mtcars data frame.
# The first "data" jar will contain the variables mpg (miles per gallon) and wt (weight)
# for cars with 4 cylinders.
# The second "data" jar will contain the variables mpg (miles per gallon) and wt (weight)
# for cars with 6 cylinders.
# The third "data" jar will contain the variables mpg (miles per gallon) and wt (weight)
# for cars with 8 cylinders.
# The 3 smaller "data" jars will be stored in a list. To apply the pmap() function to all of
# these jars simultaneously, we will need to nest them inside a larger "data" jar.
#=============================================================================

by_cyl_list <- mtcars %>% select(mpg,wt,cyl) %>% split(.$cyl)

str(by_cyl_list)

#=============================================================================

# Create a bigger "data" jar which will contain the 3 smaller "data" jars (in other words,
create a list of lists).

#=============================================================================

by_cyl_list_of_lists <- list(by_cyl)
```

```
str(by_cyl_list_of_lists)

#=============================================================================================
# Define a scatterplot function which will be applied to each of the smaller "data" jars
# nested within the larger "data" jar. Because the smaller "data" jars contain data sets (or
# data frames), the input argument for the function will be generic and will be called data -
# however, inside the body of the function, we can refer to variables inside each of
# the smaller "data" jars by their actual particular names.
# The scatterplot() function will plot the mpg variable against the hp variable and also
# show the fitted OLS regression line and the corresponding 95% pointwise confidence
# band.
#=============================================================================================

scatterplot <- function(data){

     ggplot(data=data, aes(x=wt, y=mpg)) +
      geom_point() +
       geom_smooth(method="lm", fill="lightblue") +
       labs(x="Weight",y="Miles per gallon") +
        ggtitle(paste("Cars with", unique(data$cyl),"cylinders")) +
         theme_bw() +
          theme(plot.title = element_text(hjust = 0.5))

}


#=============================================================================================
# Apply the scatterplot() function to each of the data sets in the smaller jars nested
# within the larger jar - this will produce a list of "scatterplot" subplots.
#=============================================================================================

plots_list <- by_cyl_list_of_lists %>% pmap(scatterplot)
```

```
#=============================================================================================
# Display all the "scatterplot" subplots in the same plotting window.
#=============================================================================================

marrangeGrob(plots_list, nrow=1, ncol=3,
             top="Miles per gallon versus Weight\nEstimated Regression Line
                  and 95% Pointwise Confidence Band")
```



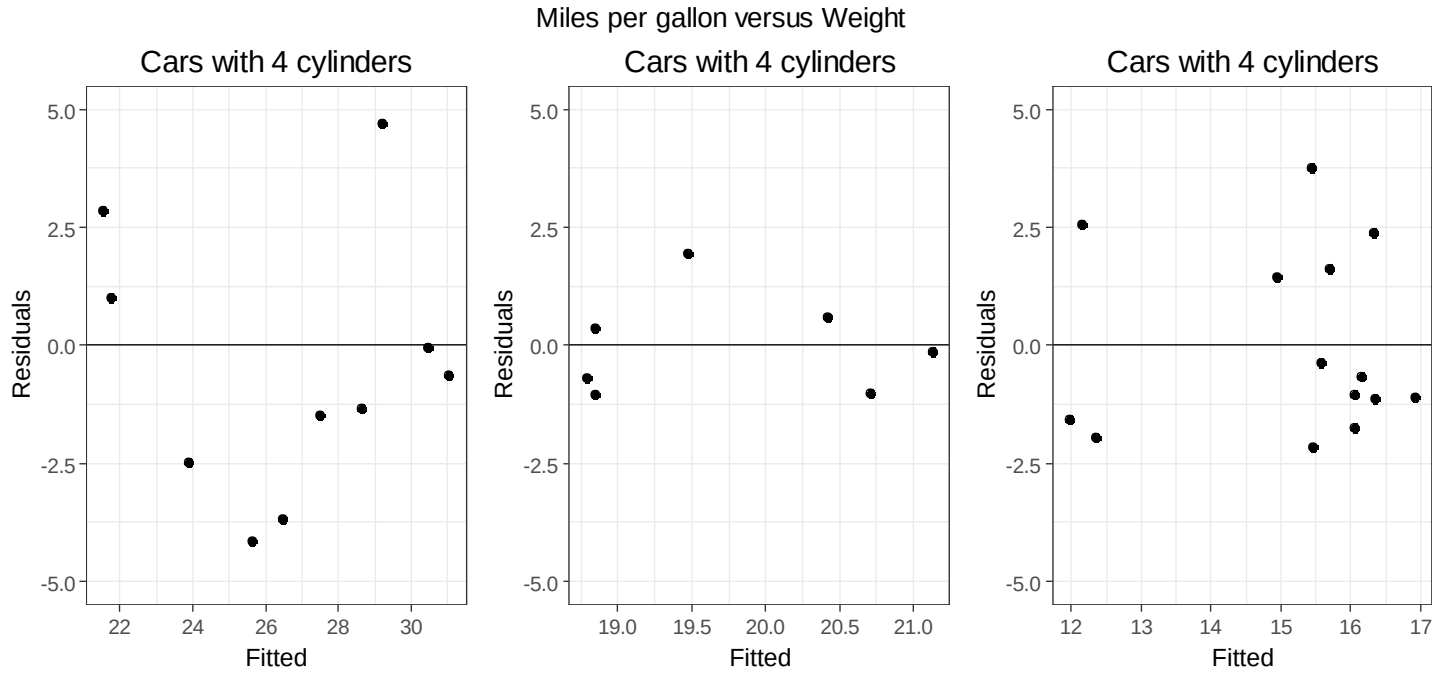
```
#=============================================================================================
# Fit a simple linear regression model relating mpg (dependent variable) to hp
# (independent variable) to each of the data sets in the smaller "data" jars; this will
# produce three smaller "model" jars stored in a list.
#=============================================================================================


models_list <- by_cyl_list_of_lists %>%          # apply model using a function defined

               pmap( ~ lm(mpg ~ wt, data = .) ) # on the fly; however, it is also

str(models_list)                                 # possible to define a model fitting

                                                 # function outside of pmap as seen below.



model <- function(data) {
         lm(mpg ~ wt, data=data)
}

models_list <- by_cyl_list_of_lists %>% pmap(model)  # apply model using a function defined

                                                     # outside of pmap

# Note:  We define the model() function on the fly inside of pmap() by replacing:
#
#
#        1) the portion highlighted in yellow below with a tilde and
#        2) the data argument highlighted in blue below with a dot.
#         model <- function(data) {
#                 lm(mpg ~ wt, data=data)
#        }
#        The function define on the fly will therefore look like this:
#        ~ lm(mpg ~ wt, data= .)
```

```
#==========================================================================================
# Store the smaller "model" jars into a bigger "model" jar (that is,
# create a list of lists).
#==========================================================================================

models_list_of_lists <- list(models_list)

str(models_list_of_lists)

# R output listed in purple - do not run this as commands

> models_list_of_lists
[[1]]
[[1]]$`4`

Call:
lm(formula = mpg ~ wt, data = data)

Coefficients:
(Intercept)           wt
     39.571       -5.647


[[1]]$`6`

Call:
lm(formula = mpg ~ wt, data = data)

Coefficients:
(Intercept)           wt
      28.41        -2.78
```

```
[[1]]$`8`

Call:
lm(formula = mpg ~ wt, data = data)

Coefficients:
(Intercept)           wt
     23.868       -2.192

#=============================================================================================
# Extract confidence intervals for the true slope and true intercept from the smaller
# "model" jars contained in the bigger "model" jar by using the function confint()
#=============================================================================================

confint_list <- models_list_of_lists %>% pmap( ~ confint(.) )

confint_list



# R output listed in purple - do not run this as commands

> confint_list
$`4`
                2.5 %    97.5 %
(Intercept) 29.738544 49.403848
wt          -9.832284 -1.461766

$`6`
               2.5 %     97.5 %
(Intercept) 17.65258 39.1651069
wt          -6.21162  0.6514082

$`8`
                2.5 %     97.5 %
(Intercept) 17.319690 30.4163679
wt          -3.803102 -0.5817738
```

```
#=============================================================================================
# Compute residuals and fitted values for the fitted simple linear regression models
# and store them in smaller "model diagnostics" jars
#=============================================================================================

resfit_list <- models_list_of_lists %>%
               pmap( ~ cbind.data.frame(residuals=residuals(.), fitted=fitted(.)) )

#=============================================================================================
# Nest the smaller "model diagnostics" jars into a bigger "model diagnostics" jar
# (i.e., create a list of lists)
#=============================================================================================

resfit_list_of_lists <- list(resfit_list)

#=============================================================================================
# Create a function residualsvsfitted() for plotting the residuals versus the fitted values
# based on the information content of the smaller "model diagnostics" jars.
# The information content consists of the variables residuals and fitted.
# In addition to having a generic data argument, the function also includes other arguments
# used to specify the plot title as well as the limits of the vertical axis.
#=============================================================================================


residualsvsfitted <- function(data, titlestring, ymin, ymax){

     ggplot(data=data, aes(x=fitted, y=residuals)) +
      geom_point() +
      geom_hline(yintercept = 0) +
       labs(x="Fitted",y="Residuals") +
       ylim(ymin, ymax) +
        ggtitle(titlestring) +
         theme_bw() +
          theme(plot.title = element_text(hjust = 0.5))

}
```

```
#=============================================================================================
# Apply the function residualsvsfitted() simultaneously to each of the smaller
# "model diagnostic" jars located inside the larger "model diagnostic" jar in order to
# produce plots of residuals versus fitted values for each of the three types of cylinders.
# This will produce a list of "residuals versus fitted values" subplots.
#=============================================================================================


diagplots_list <- resfit_list_of_lists %>%

pmap(residualsvsfitted, paste("Cars with", levels(factor(mtcars$cyl)), "cylinders "), -5, 5)



diagplots_list

#=============================================================================================
# Display all the "residuals versus fitted values" subplots in the same plotting window.
#=============================================================================================

marrangeGrob(diagplots_list, nrow=1, ncol=3, top="Miles per gallon versus Weight")
```




```
#=============================================================================================
# Predict the individual miles per gallon for a car with a weight of 5 tonnes,
# assuming the car has 4, 6 or 8 cylinders, respectively
#=============================================================================================

predict_list <- pmap(models_list_of_lists,
                     predict,
                     newdata=data.frame(wt=5),    # additional argument of predict();
                     interval="prediction")       # additional argument of predict()
```

```
# R output listed in purple - do not run this as commands

> predict_list

$`4`
       fit       lwr     upr
1 11.33607 -2.485564 25.1577

$`6`
       fit      lwr     upr
1 14.50831 7.297328 21.7193

$`8`
       fit      lwr      upr
1 12.90584 8.064697 17.74698
```

Those of you familiar with the map() function may realize that, in the above examples, we could abstain from nesting our smaller jars into a larger jar. If we do that, we can apply the map() function directly to the smaller jars. For example, the commands below are applied directly to the smaller jars (which are stored in a list, rather than a list of lists) using the map() function:

```
by_cyl_list %>% map(scatterplot)

models_list <- by_cyl_list %>% map( ~ lm(mpg ~ wt, data = .) )

confint_list <- models_list %>% map( ~ confint(.) )


diagplots_list <- resfit_list %>%

                  map(residualsvsfitted,

                      paste("Cars with", levels(factor(mtcars$cyl)), "cylinders "),

                       -5, 5)
```

```
predict_list <- map(models_list, predict, newdata=data.frame(wt=5), interval="prediction")
```

Keep on purring!

Isabella

Isabella R. Ghement, Ph.D.
Ghement Statistical Consulting Company Ltd.
301-7031 Blundell Road, Richmond, B.C., Canada, V6Y 1J5
Tel: 604-767-1250
Fax: 604-270-3922
E-mail: isabella@ghement.ca
Web: www.ghement.ca