Decision Management Community Challenge "Collection of Cars"

Dr. Riccardo Hertel

September 2nd, 2015

Introduction

In the September 2015 Challenge proposed by the *Decision Management Community* the task consists in analyzing a collection of cars, with their type defined by characteristics stored in columns labelled "Make" and "Model". Eventually, the cars' price will also play a role. There are various groups, denoted as G1, G2, G3... with several cars, and generally one car type can be present in more than one of these groups.

In this challenge we are invited to

1. detect which car types are included in *all* groups,
2. identify which car types are only found in exactly *one* group, and
3. find the most expensive car in each group.

This document shows how the free programming language R can be used to address these tasks with a few lines of code.

Preliminaries

The solution described below uses two so-called packages that need to be installed in R. More specifically, we will use the MASS package to access a sample database of cars and their properties, and data.table to efficiently and rapidly manipulate the data, as we shall see later. Installing such external packages is a standard procedure in R, and in most cases it can be accomplished simply with the function `install.packages()`.

Once a package is installed, it provides a specific library, *i.e.*, a set of functions and/or data that can be loaded with the `library()` command. In a typical installation of R, many packages have been installed on previous occasions, but only few are needed in a session. Once a library has been installed it remains available for future use. Hence, most of the installed libraries are not used unless they are needed for a specific task, and then the necessary libraries can be loaded with the commands `library()` or `require()`.

The following code loads the aforementioned libraries into R and installs the corresponding packages in the case that this had not been done before.

```
install_missing <- function(lib){install.packages(lib); library(lib, character.only=TRUE)}
tryCatch(library("MASS"), error = function(e) install_missing("MASS"))
tryCatch(library("data.table"), error = function(e) install_missing("data.table"))
set.seed(123)
```

Attempting to load a library that has not yet been installed results in an error. Therefore, the function `tryCatch()` is used here to capture an error that could occur while loading the required libraries. In the case of an error, a function `install_missing()` will be called (defined in the first line) that installs the missing packages, instead of interrupting the code with an error message. Since we will randomly select different cars belonging to different groups, we have used the function `set.seed()` which allows one to obtain reproducible data when pseudo-random numbers are generated.

Preparing a sample data set

In order to work on such a problem, we need first of all a data set on which the analysis can be performed. The `MASS` library provides a data set called `Cars93` that contains a plethora of information on a large number of car types. More precisely, the data comprises 93 different car types, each characterized by 27 features. Here is a summary of the data:

```
str(Cars93)
```

```
## 'data.frame':    93 obs. of  27 variables:
##  $ Manufacturer      : Factor w/ 32 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
##  $ Model             : Factor w/ 93 levels "100","190E","240",..: 49 56 9 1 6 24 54 74 73 35 ...
##  $ Type              : Factor w/ 6 levels "Compact","Large",..: 4 3 1 3 3 3 2 2 3 2 ...
##  $ Min.Price         : num  12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
##  $ Price             : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
##  $ Max.Price         : num  18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
##  $ MPG.city          : int  25 18 20 19 22 22 19 16 19 16 ...
##  $ MPG.highway       : int  31 25 26 26 30 31 28 25 27 25 ...
##  $ AirBags           : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
##  $ DriveTrain        : Factor w/ 3 levels "4WD","Front",..: 2 2 2 2 3 2 2 3 2 2 ...
##  $ Cylinders         : Factor w/ 6 levels "3","4","5","6",..: 2 4 4 4 2 2 4 4 4 5 ...
##  $ EngineSize        : num  1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
##  $ Horsepower        : int  140 200 172 172 208 110 170 180 170 200 ...
##  $ RPM               : int  6300 5500 5500 5500 5700 5200 4800 4000 4800 4100 ...
##  $ Rev.per.mile      : int  2890 2335 2280 2535 2545 2565 1570 1320 1690 1510 ...
##  $ Man.trans.avail   : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...
##  $ Fuel.tank.capacity: num  13.2 18 16.9 21.1 21.1 16.4 18 23 18.8 18 ...
##  $ Passengers        : int  5 5 5 6 4 6 6 6 5 6 ...
##  $ Length            : int  177 195 180 193 186 189 200 216 198 206 ...
##  $ Wheelbase         : int  102 115 102 106 109 105 111 116 108 114 ...
##  $ Width             : int  68 71 67 70 69 69 74 78 73 73 ...
##  $ Turn.circle       : int  37 38 37 37 39 41 42 45 41 43 ...
##  $ Rear.seat.room    : num  26.5 30 28 31 27 28 30.5 30.5 26.5 35 ...
##  $ Luggage.room      : int  11 15 14 17 13 16 17 21 14 18 ...
##  $ Weight            : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
##  $ Origin            : Factor w/ 2 levels "USA","non-USA": 2 2 2 2 2 1 1 1 1 1 ...
##  $ Make              : Factor w/ 93 levels "Acura Integra",..: 1 2 4 3 5 6 7 9 8 10 ...
```

We obviously need only a small part of this set for the given problem, so we can begin by selecting three relevant columns:

```
cars <- Cars93[,c("Manufacturer", "Model", "Price")] # select only the columns: Model,
                                                     # Manufacturer, and Price
colnames(cars)[1] <- "Make"
```

In the second line of this code, the name of the column "Manufacturer" has been changed to "Make" in order to obtain a set that is similar to the one displayed in the problem description.

We may also decide that 93 is too large a number of different cars to demonstrate the principle of the code, and we can hence select a subset of 20 cars, which we choose randomly with the `sample()` function:

```
cars <- cars[sample(nrow(cars),20),] # reduce the size of the data by selecting only 20
                                     # different types of cars
```

In a next step, we need to decide how many group there are, and which is the minimum and the maximum number of cars that can belong to one group. In this case, we have four groups, each consisting of a minimum of five and a maximum of ten cars.

```
num_groups <- 4 # number of different groups
min_group <- 5 # smallest number of cars in a group
max_group <- 10 # largest number of cars in a group
group_names <- paste0("G",1:num_groups) # define the names of the groups
```

In the last line we have specified the group names: G1, G2, G3, G4.

Now we can randomly determine the size of each group (within the limits described above) and choose randomly an according set of vehicles from our `cars` data:

```
group_list <- list("vector",num_groups) # initialize a list that will contain the index
                                        # of the cars in each group
for(i in 1:num_groups) group_list[[i]] <- sample(nrow(cars),sample(min_group:max_group,1))
                                          # randomly choose which cars are in each group
```

At this point, we already have the necessary information to assemble an ordered set of data, which in `R` is typically a *data frame*. We just need to combine and rearrange the data:

```
Group <- NULL
for (i in 1:num_groups) Group <- c(Group,rep(group_names[i],length(group_list[[i]])))
                                                        # generate "Group" column
cars_data <- cbind(Group,cars[unlist(group_list),]) # combine the columns to a data frame
rownames(cars_data) <- seq(nrow(cars_data)) # number the rows sequentially
```

Let us look at the data that we have generated:

```
print(cars_data)
```

```
##    Group       Make          Model Price
## 1     G1     Toyota         Previa  22.7
## 2     G1      Mazda            626  16.5
## 3     G1       Audi            100  37.7
## 4     G1 Volkswagen        Corrado  23.3
## 5     G1  Chevrolet       Corvette  38.0
## 6     G1    Hyundai         Sonata  13.9
## 7     G1      Buick        Riviera  26.3
## 8     G1     Subaru          Justy   8.4
## 9     G1    Pontiac         LeMans   9.0
## 10    G1 Volkswagen         Passat  20.0
## 11    G2     Toyota         Previa  22.7
## 12    G2 Oldsmobile   Eighty-Eight  20.7
## 13    G2      Dodge        Dynasty  15.6
## 14    G2    Hyundai         Sonata  13.9
## 15    G2      Mazda            626  16.5
## 16    G2     Subaru          Justy   8.4
## 17    G2     Toyota         Tercel   9.8
## 18    G2      Buick        Riviera  26.3
## 19    G2    Pontiac         LeMans   9.0
## 20    G2       Saab            900  28.7
```

```
## 21    G3    Pontiac     Bonneville  24.4
## 22    G3       Ford Crown_Victoria  20.9
## 23    G3      Dodge         Spirit  13.3
## 24    G3     Subaru          Justy   8.4
## 25    G3       Saab            900  28.7
## 26    G3  Chevrolet       Corvette  38.0
## 27    G3      Mazda            626  16.5
## 28    G4    Hyundai         Sonata  13.9
## 29    G4 Oldsmobile   Eighty-Eight  20.7
## 30    G4       Ford Crown_Victoria  20.9
## 31    G4        Geo          Metro   8.4
## 32    G4     Subaru          Justy   8.4
```

In this case we have a total of 32 cars distributed among 4 groups. The groups G1, G2, G3, G4 contain 10, 10, 7, 5 cars, respectively. This set is very similar to the one shown in the problem description, except for an additional column displaying the price, which we will use later.

Analyzing the data

Now that we have a data set to work with, we can address the tasks specified in the Challenge. A large portion of these tasks can be accomplished with a single command, using the `aggregate()` function:

```
condensed_tab <- aggregate(Group ~ . ,data = cars_data, unique)
```

This line of code has transformed the data in a way that allows us to see immediately to which group each car belongs:

```
print(condensed_tab)
```

```
##          Make          Model Price      Group
## 1      Subaru          Justy   8.4 1, 2, 3, 4
## 2         Geo          Metro   8.4          4
## 3     Pontiac         LeMans   9.0       1, 2
## 4      Toyota         Tercel   9.8          2
## 5       Dodge         Spirit  13.3          3
## 6     Hyundai         Sonata  13.9    1, 2, 4
## 7       Dodge        Dynasty  15.6          2
## 8       Mazda            626  16.5    1, 2, 3
## 9  Volkswagen         Passat  20.0          1
## 10 Oldsmobile   Eighty-Eight  20.7       2, 4
## 11       Ford Crown_Victoria  20.9       3, 4
## 12     Toyota         Previa  22.7       1, 2
## 13 Volkswagen        Corrado  23.3          1
## 14    Pontiac     Bonneville  24.4          3
## 15      Buick        Riviera  26.3       1, 2
## 16       Saab            900  28.7       2, 3
## 17       Audi            100  37.7          1
## 18  Chevrolet       Corvette  38.0       1, 3
```

Task 1: Find cars that are common to all groups

Using the transformed data shown above, we can analyze the rightmost column to obtain information on the number of groups to which each car belongs. In order to find the cars that are common to all groups, we

search for cars belonging to a number of groups that corresponds to the total number of groups. For this purpose we can use the `length()` function in combination with `lapply()`. The latter is a command that internally runs a loop over a given list:

```
in_all_groups_tab <- condensed_tab[lapply(condensed_tab$Group, length) == num_groups,]
in_all_groups_tab <- in_all_groups_tab[, -ncol(condensed_tab)] #remove last column
```

In the second line have removed the last column (containing the list of the groups to which the selected cars belong) since it contains superfluous information as we know that each car in this subset is common to *every* group. In this example the number of cars that are present in each group is equal to 1, and we can easily display the result:

```
print(in_all_groups_tab)
```

```
##     Make Model Price
## 1 Subaru Justy   8.4
```

Task 2: Find those cars which are present in exactly one group

Using a very similar approach as in the first task, we can determine the subset of cars that belong to exactly one group. From a programming perspective, the difference is that now the number of groups to which a car should belong is equal to 1, in contrast to the previous case, where we were searching for entries where this number was equal to the total number of groups (in this example 4).

```
unique_tab <- condensed_tab[lapply(condensed_tab$Group, length) == 1,] #select cars
                                                                        #appearing only in one group
```

Although this command already yields the desired result, we can perform some "cosmetic" changes to the data in order to restore a form similar to the one in the original data set:

```
unique_tab$Group <- group_names[unlist(unique_tab$Group)] # use group names, not numbers
unique_tab <- unique_tab[c("Group", "Make", "Model", "Price")] # reorder column sequence
```

The result now looks like this:

```
print(unique_tab)
```

```
##    Group       Make      Model Price
## 2     G4        Geo      Metro   8.4
## 4     G2     Toyota     Tercel   9.8
## 5     G3      Dodge     Spirit  13.3
## 7     G2      Dodge    Dynasty  15.6
## 9     G1 Volkswagen     Passat  20.0
## 13    G1 Volkswagen    Corrado  23.3
## 14    G3    Pontiac Bonneville  24.4
## 17    G1       Audi        100  37.7
```

The subset displayed above shows all the car types that belong to exactly one group in our example.

Task 3: Find the most expensive car in each group

This is a task that can be solved in several ways, but in this case I prefer using the functionalities of the `data.table` package mentioned in the beginning of the document. This package is tailored to perform very quickly complex operations and generate subsets on data sets. Moreover, I find that the syntax used in this package is rather elegant, although it differs quite significantly from R's usual syntax to manipulate data frames. In any case, using the `data.table` package, we can obtain the result of task 3 with a single line of code:

```
most_expensive <- setDT(cars_data)[, .SD[which.max(Price)], by = Group]
print(most_expensive)
```

```
##    Group      Make          Model Price
## 1:    G1 Chevrolet       Corvette  38.0
## 2:    G2      Saab            900  28.7
## 3:    G3 Chevrolet       Corvette  38.0
## 4:    G4      Ford Crown_Victoria  20.9
```

Conclusion

The free programming language R offers several powerful methods to analyze data sets and to create subsets according to specific rules. Arguably the most complex part of the programming discussed here consists in generating the sample data set. Each task of this challenge can be solved with a few lines of code in R. The code described here can be used for an arbitrary number of groups (collections) and cars. For simplicity, we have chosen to use only 20 car types and 4 groups, and we have imposed certain limits regarding the minimum and maximum number of cars within a group. Each of these parameters can be changed easily without affecting the operation of the code.

Technical aspects and contact information

This document has been produced with RStudio version 0.99.467, using the data.table package version 1.9.4 and, for the data set of the cars, the MASS package version 7.3-44. The rmarkdown package version 0.8 was used for the typesetting, and the results were generated dynamically within the document using R version 3.2.2 (2015-08-14) and the knitr package version 1.11.

The author can be contacted at rh@riccardo-hertel.com
www.riccardo-hertel.com