Decision Management Community Challenge August-2015

Dr. Riccardo Hertel

The August 2015 Challenge proposed by the *Decision Management Community* consists in identifying duplicate product lines in a list of sales. This document shows how the free programming language R can be used to address such tasks and to filter important information from a data set.

Reading and inspecting the data

There are numerous ways to load data into R. Probably the most frequently used method consists in reading a file stored on the hard disk drive with comma-separated entries (a `.csv` file) by using built-in commands such as `read.csv()`. There are also R packages offering the possibility to import Excel sheets, but for this particular case with a small data set we can conveniently use the function `read_table()`. This function reads the data and stores it internally in a format that within R is known as a *data frame*.

```
sales_dat <- read.table(text = "'Product SKU' Price Quantity
                                SKU-1000 10 10
                                SKU-1 20  3
                                SKU-2 30  1
                                SKU-1 20  4
                                SKU-3 40  6
                                SKU-3 42  8
                                SKU-4 15  2", header=TRUE)
```

We can verify that the data has been read correctly:

```
sales_dat
```

```
##   Product.SKU Price Quantity
## 1    SKU-1000    10       10
## 2       SKU-1    20        3
## 3       SKU-2    30        1
## 4       SKU-1    20        4
## 5       SKU-3    40        6
## 6       SKU-3    42        8
## 7       SKU-4    15        2
```

```
class(sales_dat)
```

```
## [1] "data.frame"
```

The first approach to address the given task could consist in identifying the entries in our data set with a `Product SKU` identifier that occurs more than once. The function `duplicated()` can be used for this purpose.

```
duplicates <- unique(with(sales_dat, (Product.SKU[duplicated(Product.SKU)])))
```

The function `unique()` has been used here to identify only the names of the duplicate `Product SKU` entries, irrespective of how many times an entry might be duplicated. We can now determine the number of groups which have the same `Product SKU` identifier:

```
length(duplicates)
```

```
## [1] 2
```

This result shows hat there are two groups with identical identifier in our data set. In the case of a small example like the one used here this result might seem obvious, but knowing the number of groups that could represent duplicates can be valuable information in larger data sets. Next we can identify which entries in the `Product SKU` column are not unique:

```
duplicates
```

```
## [1] SKU-1 SKU-3
## Levels: SKU-1 SKU-1000 SKU-2 SKU-3 SKU-4
```

Here, the first line of the output reveals that there are two subsets with the same `Product SKU`, and it specifically invites us to take a closer look at the entries with the identifiers SKU-1, SKU-3. We can achieve this programmatically by using `R`'s `%in%` operator:

```
sales_dat[sales_dat$Product.SKU %in% duplicates,]
```

```
##   Product.SKU Price Quantity
## 2       SKU-1    20        3
## 4       SKU-1    20        4
## 5       SKU-3    40        6
## 6       SKU-3    42        8
```

This already represents the relevant subset of possible duplicates. Assuming that the entries in the `Product SKU` column are correct, we could henceforth exclude all other observations (rows) from the analysis of possible duplicates. However, since the data set is small in this case, memory issues are not relevant and we can therefore preserve the entire data set without explicit subsetting.

Grouping entries with same identifier

To proceed with the analysis it is useful to group the occurrences of possible duplicate entries according to their `Product SKU` identifier entries.

Let us look at this subset more closely by separating the groups of rows with the same `Product SKU` identifier. One possibility consists in generating a list which we could call `same_id_rows`:

```
same_id_rows <- lapply(1:length(duplicates),
                       function(i) sales_dat$Product.SKU %in% duplicates[i])
same_id_rows <- lapply(same_id_rows, which)
```

We have now created a list with the relevant row numbers:

```
same_id_rows
```

```
## [[1]]
## [1] 2 4
##
## [[2]]
## [1] 5 6
```

The list shows that the rows number 2, 4 belong to the first group of identical `Product SKU` identifiers and that the rows number 5, 6 are those of the second group. Such a list can be very helpful, especially in large data sets, as it allows us to easily collect all rows with the same `Product SKU` identifier. For instance, all rows belonging to the first group with the same identifier can be displayed with

```
sales_dat[same_id_rows[[1]],]
```

```
##   Product.SKU Price Quantity
## 2       SKU-1    20        3
## 4       SKU-1    20        4
```

and accordingly for the second group with:

```
sales_dat[same_id_rows[[2]],]
```

```
##   Product.SKU Price Quantity
## 5       SKU-3    40        6
## 6       SKU-3    42        8
```

This analysis can be done for an arbitrary amount of groups and observations with equal identifiers on large data sets. Since we have established previously how many groups of duplicates exist, we can easily run a loop over all groups with the same `Product SKU` identifier if this should be necessary. In our case we know that there are only two such cases and we can hence simplify the analysis.

Distinguishing between similar and identical entries

It may be important to note that two entries with the same `Product SKU` identifier are not necessarily identical duplicates. In particular, they can display different prices. In such cases, an expert could decide whether there was a mistake in the data or if the different prices are correct; for instance if the entries were recorded at different times and the price of the product has changed. In contrast, the value stored in the `Quantity` column is not a relevant feature for identifying duplicates.
Since we have already identified the entries with the same `Product SKU` identifier, it remains to be seen whether the duplicates entries are identical or similar. We may also wish to measure the degree of similarity if the entries are not identical.

One possibility to quantify the differences between sets of numerical values is offered in `R` by the function `dist()`. With this function, we can define a relative difference between the rows of the groups of the first set by taking the difference between the prices and dividing the result by its average value:

```
dist(sales_dat$Price[same_id_rows[[1]]]) / mean(sales_dat$Price[same_id_rows[[1]]])
```

```
##   1
## 2 0
```

For the first group the difference in the `Price` value between the first and the second row of the group is equal to zero. These entries can hence be considered as identical duplicates.

Concerning the second group, we obtain a difference of about 5% between the two rows of the group by applying the same method:

```
dist(sales_dat$Price[same_id_rows[[2]]]) / mean(sales_dat$Price[same_id_rows[[2]]])
```

```
##            1
## 2 0.04878049
```

As a reminder, this group number `[[2]]` belongs to the `Product SKU` category SKU-3. The function `dist()` is particularly helpful when there is a larger number of duplicates in the same group, since its output provides a triangular matrix containing the difference between each entry. This allows to quantify the difference between each row of the group; a powerful feature that is not seen in this case, since both groups with the same `Product SKU` identifier consist only of two rows.

Concerning the non-identical possible duplicates mentioned above, as mentioned before, it may be best if an expert in the sales department determines whether there is an erroneous entry in the data set or if the two observations (rows) should be maintained as they are.

Reassembling and ordering the data set

The previous results allow us to conclude that the entries of the rows 2, 4 of the original data set are identical duplicates in the sense that they represent sales of the same product with the same price. It could therefore be convenient to assemble them into a single observation. For this, we can summarize the `Quantity` of the relevant rows and assign the result to a single observation. We can then retain only one observation of these multiple entries and delete the other occurrences. The function `aggregate()` accomplishes these tasks in one step:

```
sales_dat <- aggregate(Quantity ~ Product.SKU + Price, data=sales_dat, sum)
sales_dat
```

```
##   Product.SKU Price Quantity
## 1    SKU-1000    10       10
## 2       SKU-4    15        2
## 3       SKU-1    20        7
## 4       SKU-2    30        1
## 5       SKU-3    40        6
## 6       SKU-3    42        8
```

The output shows that there is only one row in the data set with the `Product` SKU-1, which now contains the sum of all `Quantity` entries of the previously identical duplicates. This command can be used in the same way also in the case of a large data set with numerous identical duplicates that need to be combined.

Unfortunately, the `aggregate()` function has rearranged the order of the rows and one might wish to restore an alphanumerically ordered set according to the `Product SKU` identifier. Moreover, we may wish to renumber the rows accordingly. This can be achieved with the following code:

```
sales_dat <- sales_dat[order(sales_dat$Product.SKU),]
rownames(sales_dat) <- seq(nrow(sales_dat))
sales_dat
```

```
##   Product.SKU Price Quantity
## 1       SKU-1    20        7
## 2    SKU-1000    10       10
## 3       SKU-2    30        1
## 4       SKU-3    40        6
## 5       SKU-3    42        8
## 6       SKU-4    15        2
```

Finally, we could also address the question of possible mistakes in the `Product SKU` column. For instance, the entry with `SKU-1000` could be wrong since the number is very different from the ones used in the other rows.

It is somewhat less straightforward to identify such possible mistakes programmatically, but a combination of `adist()` and `rowSums()` could help. The function `adist()` generates a matrix containing the literal differences between the entries according to Levenshtein (a non-negative integer), and here the function `rowSums()` takes the sum of these differences between each entry and all other entries. Finally, `which.max()` yields the row number with the largest total difference:

```
which.max(rowSums(adist(sales_dat$Product.SKU)))
```

```
## [1] 2
```

This shows that row number 2 has an entry in the `Product SKU` column with the largest literal difference between all other rows and may therefore need to be double-checked. Note that the row numbering has changed compared with the original data. We can display the suspicious row with the following command:

```
sales_dat[which.max(rowSums(adist(sales_dat$Product.SKU))),]
```

```
##   Product.SKU Price Quantity
## 2    SKU-1000    10       10
```

This entry could be inspected by an expert to verify whether it is correct.

Conclusion

In conclusion, using a few commands in `R`, we have identified and merged two identical entries in the data set, namely those belonging to SKU-1 in the original data set, and we have obtained information that the entries belonging to the Product identifier SKU-3 may require closer inspection. Additionally, the code has allowed to identify that the entry with SKU-1000 is a suspicious occurrence that might need to be verified.

Technical aspects and contact information

This document has been generated using `RStudio` version `0.99.467` and `rmarkdown`. The results are generated dynamically within the document by R version 3.2.2 (2015-08-14).

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