Data Science with R

Part VII: Working with Data frame

Raphael Schleutker

Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program.
— Linus Torvalds (Initiator of Linux)

Table of contents

General info

Starting with this lecture I will successively decrease the extent of explanations of basic principles. I expect that you have already a little background: You know how to use functions and how to use the help system plus you know the data structures in R.

When you recap the lectures you will more and more have to read the help pages for the used functions and you will have to run the examples on your own to fully understand them (and yes, this will take some time every now and then).

Splitting data frames

Splitting data frames

We have already seen how we can split data frames using `split`. It takes a data frame (but also a vector, for instance) and a variable of groups and returns a list each element of which is one of the resulting groups. Since we are using the same groups for merging the data frames it makes sense to store it in a separate variable.

```r
grouping <- iris$Species
split_iris <- split(iris, grouping)
```

Please note, that the second argument does not have to be related to the first argument. We could split `iris` also by a completely different grouping variable of the same length.

Splitting data frames

The result of this operation is a list of data frames, each containing the data for one species.

```r
str(split_iris, max.level = 1)
```

```
# List of 3
#  $ setosa    :'data.frame': 50 obs. of  5 variables:
#  $ versicolor:'data.frame': 50 obs. of  5 variables:
#  $ virginica :'data.frame': 50 obs. of  5 variables:
```

We could no work on these data frames separately.

Splitting data frames

`unsplit` merges the data frames. We have to provide the same grouping as for splitting.

```
merged_iris <- unsplit(split_iris, grouping)
str(merged_iris)

# 'data.frame': 150 obs. of  5 variables:
#  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
#  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
#  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
#  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
#  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Splitting data frames

We can also provide a list of grouping variables. In this case data frames for each combination will be generated.

```r
grouping <- CO2[c("Type", "Treatment")] # Data frames are lists
split_co2 <- split(CO2, grouping)
merged_co2 <- unsplit(split_co2, grouping)
```

Splitting data frames

Sometimes, we'd like to computer some values for grouped data. We could do so using the list of data frames produced by split but this is very inconvenient. Instead we can use aggregate.

```r
aggregate(iris[-5], by = iris[5], FUN = mean)
```

```
#      Species Sepal.Length Sepal.Width
# 1     setosa        5.006       3.428
# 2 versicolor        5.936       2.770
# 3  virginica        6.588       2.974
#   Petal.Length Petal.Width
# 1        1.462       0.246
# 2        4.260       1.326
# 3        5.552       2.026
```

The function takes an R object like a vector, list of vectors or a data frame, groups it by its second argument, and applies the function to each group.

Splitting data frames

A convenient way to express what the grouping variable is, is given by the formula interface.

```
aggregate(Sepal.Length ~ Species, data = iris, FUN = mean)

#      Species Sepal.Length
# 1     setosa        5.006
# 2 versicolor        5.936
# 3  virginica        6.588
```

The formula interface was originally a way to specify the relationship between dependent and independent variables in linear models but is nowadays used for many things especially when working with data frames.

Combining data frames

Combining data frames

When working with real data e.g. from files it often occurs that the data is split among several files. For instance, the data for the number of corona infections is split among many files, one for each day. In R however, we'd like to have a single data frame with which we can work.

```r
data_20200317 <- read.csv("./data/03-17-2020.csv", stringsAsFactors = FALSE)
data_20200318 <- read.csv("./data/03-18-2020.csv", stringsAsFactors = FALSE)
```

Combining data frames

When both data frames have a compatible structure (same number of columns in the same order) we can combine them row-wise using `rbind`.

```r
data_full <- rbind(data_20200317, data_20200318)
```

Combining data frames

When both data frames have a compatible structure (same number of columns in the same order) we can combine them row-wise using `rbind`.

```
str(data_20200317, max.level = 0)

# 'data.frame': 276 obs. of  8 variables:

str(data_20200318, max.level = 0)

# 'data.frame': 284 obs. of  8 variables:

str(data_full, max.level = 0)

# 'data.frame': 560 obs. of  8 variables:
```

Combining data frames

A similar function is `cbind`, which combines data frames column-wise. In this case, both data frames have to have the same number of rows. Also, we have to make sure that the ordering in both data frames is the same.

Combining data frames

We can also combine two data frames by a key column. Let us assume we are interested only in some europoean countries.

```r
eu_countries <- subset(
        data_full,
        Country.Region %in% c("Germany", "Italy", "Norway", "Spain"),
        select = 2:6
)

str(eu_countries)
```

```
# 'data.frame': 8 obs. of  5 variables:
#  $ Country.Region: chr  "Italy" "Spain" "Germany" "Norway" ...
#  $ Last.Update   : chr  "2020-03-17T18:33:02" "2020-03-17T20:53:02" "2020-03-17T18:53:02" "2020-03-17T19:53:02" .
#  $ Confirmed     : int  31506 11748 9257 1463 35713 13910 12327 1550
#  $ Deaths        : int  2503 533 24 3 2978 623 28 6
#  $ Recovered     : int  2941 1028 67 1 4025 1081 105 1
```

Combining data frames

We also have another data frame with some general information about each country like total population.

```
population <- data.frame(
        Country = c("Germany", "Italy", "Norway", "Spain"),
        Population = c(83149300, 60317546, 5367580, 46733038)
)

population
```

```
#   Country Population
# 1 Germany   83149300
# 2   Italy   60317546
# 3  Norway    5367580
# 4   Spain   46733038
```

Combining data frames

We can add this information to our first data frame by using the country as a key.

```
data_with_pop <- merge(
        eu_countries,
        population,
        by.x = "Country.Region",
        by.y = "Country"
)
```

Combining data frames

```
data_with_pop
```

```
#   Country.Region          Last.Update Confirmed
# 1        Germany 2020-03-17T18:53:02      9257
# 2        Germany 2020-03-18T19:33:02     12327
# 3          Italy 2020-03-17T18:33:02     31506
# 4          Italy 2020-03-18T17:33:05     35713
# 5         Norway 2020-03-17T19:53:02      1463
# 6         Norway 2020-03-18T15:53:09      1550
# 7          Spain 2020-03-17T20:53:02     11748
# 8          Spain 2020-03-18T13:13:13     13910
#   Deaths Recovered Population
# 1     24        67   83149300
# 2     28       105   83149300
# 3   2503      2941   60317546
# 4   2978      4025   60317546
# 5      3         1    5367580
# 6      6         1    5367580
# 7    533      1028   46733038
# 8    623      1081   46733038
```

Constructing environments from data frames

Constructing environments from data frames

When working with data frams we will often refer to columns of that data frame by name. This is a bit annoying cause we always have to add the name of the data frame in front of it.

```
eu_countries$Confirmed
```

```
# [1] 31506 11748  9257  1463 35713 13910 12327
# [8]  1550
```

Constructing environments from data frames

Fortunately, R provides two functions that allow us to construct an environment from a data frame or a list. The first function is called `with`. It takes the data frame as the first argument and some code as the second argument. We can group that code by curly braces to allow several lines. The effect is that we can refer to columns in the data frame like variables.

```
with(
    eu_countries,
    {
        Confirmed / sum(Confirmed) # Will not be returned.
        Deaths / Confirmed # Will be returned.
    }
)


# [1] 0.079445185 0.045369425 0.002592633
# [4] 0.002050581 0.083387002 0.044787922
# [7] 0.002271437 0.003870968
```

Constructing environments from data frames

The return value of the function is the result of the last line in the code block. However, we can use previous lines of code to calculate intermediate results. We can assign variables and use them within that code block but they will not be exported.

Of course we can store the result of `with` in a variable.

```r
recovery_rate <- with(eu_countries, Recovered / Confirmed)
recovery_rate


# [1] 0.0933472989 0.0875042560 0.0072377660
# [4] 0.0006835270 0.1127040573 0.0777138749
# [7] 0.0085178876 0.0006451613
```

For one-liners we can omit the curly braces.

Constructing environments from data frames

A similiar function is `within`. It also constructs an environment from a list or data frame and evaluates the code in that environment. However it creates a copy of the original data frame and attaches newly created variables. The return value is than that modified copy of the original list / data frame.

```
eu_countries <- within(eu_countries, death_rate <- Deaths / Confirmed)
str(eu_countries)

# 'data.frame': 8 obs. of  6 variables:
#  $ Country.Region: chr  "Italy" "Spain" "Germany" "Norway" ...
#  $ Last.Update   : chr  "2020-03-17T18:33:02" "2020-03-17T20:53:02" "2020-03-17T18:53:02" "2020-03-17T19:53:02" .
#  $ Confirmed     : int  31506 11748 9257 1463 35713 13910 12327 1550
#  $ Deaths        : int  2503 533 24 3 2978 623 28 6
#  $ Recovered     : int  2941 1028 67 1 4025 1081 105 1
#  $ death_rate    : num  0.07945 0.04537 0.00259 0.00205 0.08339 ...
```

Constructing environments from data frames

Both functions can be very handy but we should not put to much code in the expression block as this soon becomes confusing.