The example

A few days ago, I met a colleague from the plant pathology group, who asked my co-operation for an experiment where he studied the content in 62 mycotoxins in wheat caryopses, as recorded in 70 samples coming from different Italian regions. The dataset was structured as 70 x 64 table, as shown in the scheme below: there is one row for each wheat sample, while the first column represents the sample id, the second column represents the region from where that sample was collected and the other columns represent the concentrations of the 62 toxins (one column per toxin).

That colleague wanted me to calculate, for each toxin and for each region, the following stats:

1. number of collected samples
2. mean concentration value
3. maximum concentration value
4. standard error
5. number of contaminated samples (concentration higher than 0)
6. percentage of contaminated samples
7. mean value for contaminated samples
8. standard error for contaminated samples

He also wanted me to return to him a list of 62 tables (one per toxin), with all the regions along with their descriptive stats.

I have already done something like this!

When facing this task, my first reaction was to search in my ‘computer’s attic’, looking for previously written codes to accomplish the very same task. As I said, this is a fairly common task in my everyday work and I could find several examples of old codes. Looking at those, I realised that, when the number of variables is high, I have so far made an extensive use of `for` loops to repeat the same task across columns. This is indeed no wonder, as I came across R in 2000, after an extensive usage of general purpose languages, such as Quick Basic and Visual Basic.

In practise, most of my earliest R codes were based on the `tapply()` function to calculate statistics for grouped data and `for` loops to iterate over columns. In the box below I show an example of such an approach, by using a factitious dataset, with the very same structure as my colleague’s dataset.

```
rm(list = ls())
dataset <- read.csv("https://casaonofri.it/_datasets/Mycotoxins.csv", header = T)
# str(dataset)
# 'data.frame': 70 obs. of  64 variables:
#  $ Sample : int  1 2 3 4 5 6 7 8 9 10 ...
#  $ Region : chr  "Lombardy" "Lombardy" "Lombardy" "Lombardy" ...
#  $ DON    : num  8.62 16.2 18.19 27.08 10.97 ...
#  $ DON3G  : num  21.7 28.4 34.7 26.9 26.4 ...
#  $ NIV    : num  23.5 25.3 22.6 27.8 29.4 ...
#  $ NIVG   : num  38.6 26.2 13.5 21.4 15.4 ...
#  $ T2     : num  23.9 23.6 19.7 25.7 21.7 ...
#  $ HT2    : num  19 37.6 32.1 22.3 25.6 ...
#  $ HT2G   : num  13.7 25.7 25.8 33.4 32.7 ...
#  $ NEO    : num  29.06 7.56 27.52 32.91 24.49 ...

returnList <- list()
for(i in 1:(length(dataset[1,]) - 3)){
  y <- as.numeric( unlist(dataset[, i + 3]) )
  Count <- tapply(y, dataset$Region, length)
  Mean <- tapply(y, dataset$Region, mean)
  Max <- tapply(y, dataset$Region, max)
  SE <- tapply(y, dataset$Region, sd)/sqrt(Count)
  nPos <- tapply(y != 0, dataset$Region, sum)
  PercPos <- tapply(y != 0, dataset$Region, mean)*100
  muPos <- tapply(ifelse(y > 0, y, NA), dataset$Region, mean, na.rm =
T)
  muPos[is.na(muPos)] <- 0
  sdPos <- tapply(ifelse(y > 0, y, NA), dataset$Region, sd, na.rm = T)
  SEpos <- sdPos/sqrt(nPos)
  returnList[[i]] <- data.frame(cbind(Count, Mean, Max, SE, nPos,
PercPos, muPos, SEpos))
  names(returnList)[[i]] <- colnames(dataset)[i + 3]
}
print(returnList$CRV, digits = 2)
##                 Count Mean Max   SE nPos PercPos muPos SEpos
## Abruzzo             4   28  30 0.85    4     100    28  0.85
## Apulia              9   25  40 2.67    9     100    25  2.67
## Campania            2   20  21 0.74    2     100    20  0.74
## Emilia Romagna      8   23  33 2.80    8     100    23  2.80
## Latium              7   20  33 2.76    7     100    20  2.76
## Lombardy            4   25  32 5.12    4     100    25  5.12
## Molise              1   18  18   NA    1     100    18    NA
## Sardinia            6   25  38 3.32    6     100    25  3.32
## Sicily              6   21  30 2.94    6     100    21  2.94
## The Marche          5   19  23 1.58    5     100    19  1.58
## Tuscany             5   30  34 1.22    5     100    30  1.22
## Umbria              9   21  32 2.83    9     100    21  2.83
## Veneto              4   23  28 1.99    4     100    23  1.99
```

I must admit that the above code is ugly: first, `for` loops in R are not very efficient and, second, reusing that code requires quite a bit of editing and it is prone to errors. Therefore, I asked myself how I could write more efficient code …

First of all, I thought I should use `apply()` instead of the `for` loop. Thus I wrote a function to calculate the required stats for each column and `apply()-ed` that function to all columns of my data-frame.

```
funBec <- function(y, group){
  # y <- as.numeric( unlist(dataset[, i + 3]) )
  Count <- tapply(y, group, length)
  Mean <- tapply(y, group, mean)
  Max <- tapply(y, group, max)
  SE <- tapply(y, group, sd)/sqrt(Count)
  nPos <- tapply(y != 0, group, sum)
  PercPos <- tapply(y != 0, group, mean)*100
  muPos <- tapply(ifelse(y > 0, y, NA),group, mean, na.rm = T)
  muPos[is.na(muPos)] <- 0
  sdPos <- tapply(ifelse(y > 0, y, NA), group, sd, na.rm = T)
  SEpos <- sdPos/sqrt(nPos)
  data.frame(cbind(Count, Mean, Max, SE, nPos, PercPos, muPos, SEpos))
}
returnList2 <- apply(dataset[3:length(dataset[1,])], 2,
                     function(col) funBec(col, dataset$Region))
# kable(returnList2$CRV, digits = 2)
```

I am pretty happy with the above approach; it feels 'natural', to me and the coding appears to be pretty clear and easily reusable. However, it makes only use of the `base` R implementation and, therefore, it looks a bit out-fashioned… a dinosaur's code…

What could I possibly do to write more 'modern' code? A brief survey of posts from the R world suggested the ultimate solution: "I must use the 'tidyverse'!". Therefore, I tried to perform my column-wise task by using `dplyr` and related packages and I must admit that I could not immediately find an acceptable solution. After some careful thinking, it was clear that the tidyverse requires a strong change in programming habits, which is not always simple for those who are over a certain age. Or, at least, that is not simple for me…

Eventually, I thought I should switch from a `for` attitude to a `split-apply-combine` attitude; indeed, I discovered that, if I 'melted' the dataset so that the variables were stacked one above the other, I could, consequently:

1. split the dataset in several groups, consisting of one toxin and one region,
2. calculate the stats for each group, and
3. combine the results for all groups.

I'd like to share the final code: I used the `pivot_longer()` function to melt the dataset, the `group_by()` function to create an 'internal' grouping structure by regions and toxins, the `summarise()` and `across()` functions to calculate the required stats for each group. In the end, I obtained a tibble, that I split into a list of tibbles, by using the `group_split()` function. The code is shown in the box below.

```
library(tidyverse)
returnList6 <- dataset %>%
  select(-Sample) %>%
  pivot_longer(names_to = "Toxin", values_to = "Conc",
               cols = c(3:length(dataset[1,]) - 1)) %>%
  group_by(Toxin, Region) %>%
  summarise(across("Conc", .fns =
               list(Mean = mean,
                    Max = max,
                    SE = function(df) sd(df)/sqrt(length(df)),
                    nPos = function(df) length(df[df > 0]),
                    percPos = function(df) length(df[df >
0])/length(df)*100,
                    muPos =  function(df) mean(df[df > 0]),
                    SEpos =  function(df) sd(df[df >
0])/sqrt(length(df[df > 0]))
                    ))) %>%
  ungroup() %>%
  group_split(Toxin, .keep = F)
names(returnList6) <- names(dataset)[3:64]
# returnList6$CRV
```

Well, I can say that with the above code I have, at least, tried not to be a dinosaur… but, I am sure that many young scientists out there can suggest better solutions. If so, please drop me a line at the email address below. Thanks for reading!