One year ago, I published a post titled 'Some everyday data tasks: a few hints with R'. In that post, I considered four data tasks, that we all need to accomplish daily, i.e.

1. subsetting
2. sorting
3. casting
4. melting

In that post, I used the methods I was more familiar with. And, as a long-time R user, I have mainly incorporated in my workflow all the functions from the base R implementation.

But now, the tidyverse is with us! Well, as far as I know, the tidyverse has been around long before my post. However, for a long time, I did not want to surrender to such a new paradygm. I am no longer a young scientist and, therefore, picking up new techniques is becoming more difficult: why should I abandon my effective workflow in favour of new techniques, which I am not familiar with? Yes I know, the young scientists are thinking that I am just an old dinosaur, who is trying to resist to progress by all means… It is a good point! I see that reading the code produced by my younger collegues is becoming difficult, due to the massive use of the tidyverse and the pipes. I still have a few years to go, before retirement and I do not yet fell like being set aside. Therefore, a few weeks ago I finally surrendered and 'embraced' the tidyverse. Here is how I revisited my previous post.

Subsetting the data

Subsetting means selecting the records (rows) or the variables (columns) which satisfy certain criteria. Let's take the 'students.csv' dataset, which is available on one of my repositories. It is a database of student's marks in a series of exams for different subjects and, obviously, I will use the 'readr' package to read it.

```
library(readr)
library(dplyr)
library(tidyr)
students <- read_csv("https://www.casaonofri.it/_datasets/students.csv")
students
## # A tibble: 232 x 6
##        Id Subject  Date        Mark  Year HighSchool
##
##   1     1 AGRONOMY 10/06/2002    30  2001 CLASSICO
##   2     2 AGRONOMY 08/07/2002    24  2001 AGRARIO
##   3     3 AGRONOMY 24/06/2002    30  2001 AGRARIO
##   4     4 AGRONOMY 24/06/2002    26  2001 CLASSICO
##   5     5 AGRONOMY 23/01/2003    30  2001 CLASSICO
##   6     6 AGRONOMY 09/09/2002    28  2001 AGRARIO
##   7     7 AGRONOMY 24/02/2003    26  2001 CLASSICO
##   8     8 AGRONOMY 09/09/2002    26  2001 SCIENTIFICO
##   9     9 AGRONOMY 09/09/2002    23  2001 RAGIONERIA
## 10    10 AGRONOMY 08/07/2002    27  2001 CLASSICO
## # … with 222 more rows
```

With respect to the usual `read.csv` function I saved some typing, as I did not need to specify the 'header = T' argument. Furthermore, printing the tibble only shows the first ten rows, which makes the 'head()' function no longer needed.

Let's go ahead and try to subset this tibble: we want to extract the good students, with marks higher than 28. In my previous post, I used the 'subset()' function. Now, I will use the 'filter()' function in the 'dplyr' package:

```
# subData <- subset(students, Mark >= 28)
subData <- filter(students, Mark >= 28)
subData
## # A tibble: 87 x 6
```

```
##       Id Subject  Date        Mark  Year HighSchool
##
##  1      1 AGRONOMY 10/06/2002    30  2001 CLASSICO
##  2      3 AGRONOMY 24/06/2002    30  2001 AGRARIO
##  3      5 AGRONOMY 23/01/2003    30  2001 CLASSICO
##  4      6 AGRONOMY 09/09/2002    28  2001 AGRARIO
##  5     11 AGRONOMY 09/09/2002    28  2001 SCIENTIFICO
##  6     17 AGRONOMY 10/06/2002    30  2001 CLASSICO
##  7     18 AGRONOMY 10/06/2002    30  2001 AGRARIO
##  8     19 AGRONOMY 09/09/2002    30  2001 AGRARIO
##  9     20 AGRONOMY 09/09/2002    30  2001 ALTRO
## 10     22 AGRONOMY 23/01/2003    30  2001 RAGIONERIA
## # … with 77 more rows
```

I have noted that all other subsetting examples in my previous post can be solved by simply replacing 'subset()' with 'filter()', with no other changes. However, differences appear when I try to select the columns. Indeed, 'dplyr' has a specific function 'select()', which should be used for this purpose. Therefore, in the case that I want to select the students with marks ranging from 26 to 28 in Maths or Chemistry and, at the same time, I want to report only the three columns 'Subject', 'Mark' and 'Date', I need to split the process in two steps (filter and, then, select):

```
# subData <- subset(students, Mark <= 28 & Mark >=26 &
#                     Subject == "MATHS" |
#                     Subject == "CHEMISTRY",
#                   select = c(Subject, Mark, HighSchool))
subData1 <- filter(students, Mark <= 28 & Mark >=26 &
                   Subject == "MATHS" |
                   Subject == "CHEMISTRY")
subData <- select(subData1, c(Subject, Mark, HighSchool))
```

Looking at the above two-steps process I could easily understand the meaning of the pipe operator: it simply replaces the word 'then' between the two steps (`filter` and then `select` is translated into `filter %>% select`). Here is the resulting code:

```
subData <- students %>%
  filter(Mark <= 28 & Mark >=26 &
                   Subject == "MATHS" |
                   Subject == "CHEMISTRY") %>%
  select(c(Subject, Mark, HighSchool))
subData
## # A tibble: 50 x 3
##    Subject    Mark HighSchool
##
##  1 CHEMISTRY    20 AGRARIO
##  2 CHEMISTRY    21 CLASSICO
##  3 CHEMISTRY    21 CLASSICO
##  4 CHEMISTRY    18 AGRARIO
##  5 CHEMISTRY    28 ALTRO
##  6 CHEMISTRY    23 RAGIONERIA
##  7 CHEMISTRY    26 RAGIONERIA
##  8 CHEMISTRY    27 AGRARIO
##  9 CHEMISTRY    27 SCIENTIFICO
## 10 CHEMISTRY    23 RAGIONERIA
## # … with 40 more rows
```

In the end: there is not much difference between 'subset()' and 'filter()'. However, I must admit I am seduced by the 'pipe' operator… my younger collegues may be right: it should be possible to chain several useful data management steps, producing highly readable code. But… how about debugging?

Sorting the data

In my previous post I showed how to sort a data frame by using the 'order()' function. Now, I can use the 'arrange()' function:

```
# sortedData <- students[order(-students$Mark, students$Subject), ]
# head(sortedData)
sortedData <- arrange(students, desc(Mark), Subject)
sortedData
## # A tibble: 232 x 6
##       Id Subject  Date        Mark  Year HighSchool
##
##  1     1 AGRONOMY 10/06/2002    30  2001 CLASSICO
##  2     3 AGRONOMY 24/06/2002    30  2001 AGRARIO
##  3     5 AGRONOMY 23/01/2003    30  2001 CLASSICO
##  4    17 AGRONOMY 10/06/2002    30  2001 CLASSICO
##  5    18 AGRONOMY 10/06/2002    30  2001 AGRARIO
##  6    19 AGRONOMY 09/09/2002    30  2001 AGRARIO
##  7    20 AGRONOMY 09/09/2002    30  2001 ALTRO
##  8    22 AGRONOMY 23/01/2003    30  2001 RAGIONERIA
##  9    38 BIOLOGY  28/02/2002    30  2001 AGRARIO
## 10    42 BIOLOGY  28/02/2002    30  2001 RAGIONERIA
## # … with 222 more rows
# sortedData <- students[order(-students$Mark, -xtfrm(students$Subject)), ]
# head(sortedData)
sortedData <- arrange(students, desc(Mark), desc(Subject))
sortedData
## # A tibble: 232 x 6
##       Id Subject Date        Mark  Year HighSchool
##
##  1   116 MATHS   01/07/2002    30  2001 ALTRO
##  2   117 MATHS   18/06/2002    30  2001 RAGIONERIA
##  3   118 MATHS   09/07/2002    30  2001 AGRARIO
##  4   121 MATHS   18/06/2002    30  2001 RAGIONERIA
##  5   123 MATHS   09/07/2002    30  2001 CLASSICO
##  6   130 MATHS   07/02/2002    30  2001 SCIENTIFICO
##  7   131 MATHS   09/07/2002    30  2001 AGRARIO
##  8   134 MATHS   26/02/2002    30  2001 AGRARIO
##  9   135 MATHS   11/02/2002    30  2001 AGRARIO
## 10   143 MATHS   04/02/2002    30  2001 RAGIONERIA
## # … with 222 more rows
```

As for sorting, there is no competition! The 'arrange()' function, together with the 'desc()' function for descending order, represents a much clearer way to sort the data, with respect to the traditional 'order()' function.

Casting the data

When we have a dataset in the LONG format, we might be interested in reshaping it into the WIDE format. This is the same as what the 'pivot table' function in Excel does. For example, take the 'rimsulfuron.csv' dataset in my repository. This contains the results of a randomised block experiment, where we have 16 herbicides in four blocks. The dataset is in the LONG format, with one row per plot.

```
rimsulfuron <- read_csv("https://www.casaonofri.it/_datasets/rimsulfuron.csv")
## Parsed with column specification:
## cols(
##   Herbicide = col_character(),
##   Block = col_character(),
```

```
##   Height = col_double(),
##   Yield = col_double()
## )
rimsulfuron
## # A tibble: 60 x 4
##    Herbicide Block Height Yield
##
##  1 A         B1      183.  93.9
##  2 B         B1      187  103.
##  3 C         B1      188.  92.7
##  4 D         B1      183.  88.7
##  5 E         B1      184.  91.0
##  6 F         B1      179.  98.4
##  7 G         B1      192. 105.
##  8 H         B1      191  121.
##  9 I         B1      209. 111.
## 10 J         B1      210   82.7
## # … with 50 more rows
```

Let's put this data frame in the WIDE format, with one row per herbicide and one column per block. In my previous post, I used to the 'cast()' function in the 'reshape' package. Now I can use the 'pivot_wider()' function in the 'tidyr' package: the herbicide goes in the first column, the blocks (B1, B2, B3, B4) should go in the next four columns, and each unique level of yield should go in each cell, at the crossing of the correct herbicide row and block column. The 'Height' variable is not needed and it should be removed. Again, a two steps process, that is made easier by using the pipe:

```
# library(reshape)
# castData <- cast(Herbicide ~ Block, data = rimsulfuron,
#      value = "Yield")
# head(castData)

castData <- rimsulfuron %>%
  select(-Height) %>%
  pivot_wider(names_from = Block, values_from = Yield)
castData
## # A tibble: 15 x 5
##    Herbicide    B1    B2    B3    B4
##
##  1 A          93.9 105.   94.1 111.
##  2 B         103.   98.5 106.  128.
##  3 C          92.7 107.   97.1 118.
##  4 D          88.7 114.  105.  127.
##  5 E          91.0 113.  104.  113.
##  6 F          98.4  99.9 102.   93.9
##  7 G         105.  115.  101.  122.
##  8 H         121.  113.  111.  107.
##  9 I         111.  101.  119.  118.
## 10 J          82.7  67.7  65.9  78.5
## 11 K          41.5  44.1  46.1  11.3
## 12 L         137.  113.  114.  138.
## 13 M         138.  122.  109.  118.
## 14 N         130.  119.  127.  123.
## 15 O          66.1  33.2  58.4  11.8
```

Here, I am not clear with which it is more advantageous than which. Simply, I do not see much difference: none of the two methods is as clear as I would expect it to be!

Melting the data

In this case we do the reverse: we transform the dataset from WIDE to LONG format. In my previous post I used the 'melt()' function in the 'reshape2' package; now, I will use the 'pivot_longer()' function in 'tidyr'.

```
# library(reshape2)
# castData <- as.data.frame(castData)
# mdati <- melt(castData,
#                variable.name = "Block",
#                value.name = "Yield",
#                id=c("Herbicide"))
#
# head(mdati)
#
pivot_longer(castData, names_to = "Block", values_to = "Yield",
             cols = c(2:5))
## # A tibble: 60 x 3
##    Herbicide Block Yield
##
##  1 A         B1     93.9
##  2 A         B2    105.
##  3 A         B3     94.1
##  4 A         B4    111.
##  5 B         B1    103.
##  6 B         B2     98.5
##  7 B         B3    106.
##  8 B         B4    128.
##  9 C         B1     92.7
## 10 C         B2    107.
## # … with 50 more rows
```

As before with casting, neither 'melt()', nor 'pivot_longer()' let me completely satisfied.

Tidyverse or not tidyverse?

This post is the result of using some functions coming from the 'tidyverse' and related packages, to replace other functions from more traditional packages, which I was more accustomed to, as a long-time R user. What's my feeling about this change? Let me try to figure it out.

1. First of all, it didn't take much time to adjust. I need to thank the authors of 'tidyverse' for being very respectful of tradition.
2. In one case (ordering), adjusting to the new paradigm brought to a easier coding. In all other cases, the ease of coding was not affected.

Will I stick to the new paradigm or will I go back to my familiar approaches? Should I only consider the simple tasks above, my answer would be: "I'll go back!". However, this would be an unfair answer. Indeed, my data tasks are not as simple as those above. More frequently, my data tasks are made of several different steps. For example:

1. Take the 'students' dataset
2. Filter the marks included between 26 and 28
3. Remove the 'Id', 'date' and 'high-school' columns
4. Calculate the mean mark for each subject in each year
5. Spread those means along Years
6. Get the overall mean for each subject across years

Let's try to accomplish this task by using both a 'base' approach and a 'tidyverse' approach.

```
# Traditional approach
library(reshape)
students2 <- subset(students, Mark >= 26 | Mark <= 28,
```

```
                    select = c(-Id, -Date, -HighSchool))
mstudents2 <- cast(Subject ~ Year, data = students2,
      value = "Mark", fun.aggregate = mean)
mstudents2$Mean <- apply(mstudents2[,2:3], 1, mean)
mstudents2
##       Subject     2001     2002     Mean
## 1   AGRONOMY 26.69565 26.25000 26.47283
## 2    BIOLOGY 26.48000 26.41379 26.44690
## 3  CHEMISTRY 24.21429 22.19048 23.20238
## 4  ECONOMICS 27.73077 27.11111 27.42094
## 5 FRUIT TREES      NaN 26.92857      NaN
## 6      MATHS 26.59259 25.00000 25.79630
# Tidyverse approach
students %>%
  filter(Mark >= 26 | Mark <= 28) %>%
  select(c(-Id,-Date,-HighSchool)) %>%
  group_by(Subject, Year) %>%
  summarise(Mark = mean(Mark)) %>%
  pivot_wider(names_from = Year, values_from = Mark) %>%
  mutate(Mean = (`2001` + `2002`)/2)
## # A tibble: 6 x 4
## # Groups:   Subject [6]
##   Subject     `2001` `2002`  Mean
##
## 1 AGRONOMY      26.7   26.2  26.5
## 2 BIOLOGY       26.5   26.4  26.4
## 3 CHEMISTRY     24.2   22.2  23.2
## 4 ECONOMICS     27.7   27.1  27.4
## 5 FRUIT TREES   NA     26.9  NA
## 6 MATHS         26.6   25    25.8
```

I must admit the second piece of code flows much more smooothly and it is much closer to my natural way of thinking. A collegue of mine said that, when it comes to operating on big tables and making really complex operations, the tidyverse is currently considered ‘the most powerful tool in the world’. I will have to dedicate another post to such situations. So far, I have started to reconsider my attitute towards the tidyverse.