Manipularea datelor cu ajutorul pachetului Tidyverse

1 Introducere

Tidyverse, dezvoltat de Hadley Wickham cercetător principal la RStudio, este o librărie care însumează o colecție de pachete ce împărtășesc aceeași viziune (standard) asupra modului în care trebuie prelucrate, analizate și vizualizate datele. Acest pachet nu reprezintă doar o colecție de funcții care să înlocuiască funcțiile de bază din `R` ci mai degrabă este un mod de a *gândi* și de a analiza seturile de date.

Pachetele de bază din `tidyverse` sunt:

- `readr` și `readxl` care permit citirea datelor de tip dreptunghiular (.csv, .tsv, .fwf, .xls, .xlsx)
- `dplyr` și `tidyr` care permit manipularea și transformarea datelor într-un format consistent (`tidy`)
- `ggplot2` care asigură vizualizarea datelor
- `purrr` care îmbunătățește funcționalitățile de programare, în special permite lucrul cu vectori, liste și funcții
- `stringr` care asigură un set de funcționalități necesare analizei de text
- `forcats` care îmbunătățește lucrul cu elementele de tip factor

Structura de date primară pe care se bazează pachetul `tidyverse` este `data.frame`-ul (care, odată ce vom avansa în ecosistemul `tidyverse` se va transforma în `tibble`), prin urmare este indicat ca seturile de date să fie stocate sub această formă (spre deosebire de o matrice sau un vector). Ne putem imagina că datele noastre, stocate sub forma unui `data.frame`, reprezintă universul de lucru iar coloanele acestui `data.frame` sunt obiectele pe care vrem să le explorăm, manipulăm și modelăm.

Pentru a folosi funcționalitățile prezente în pachetul `tidyverse` putem instala individual pachetele componente

```
# trebuie rulat o singura data pentru a instala pachetul in sistem
install.packages("dplyr")
install.packages("ggplot2")
install.packages("purrr")
install.packages("tidyr")
install.packages("readr")
install.packages("tibble")

# pentru a folosi functionalitatile trebuie inregistrate
library(dplyr)
library(ggplot2)
library(purrr)
library(tidyr)
library(readr)
library(tibble)
```

sau putem instala pachetul integral

```
install.packages("tidyverse")

library(tidyverse)
```

care este mult mai ușor și include întreagă colecție de funcții.

Trebuie menționat că este posibil ca prin încărcarea librăriei `tidyverse`, o serie de funcționalități din alte pachete să fie mascate (acest fenomen apare atunci când funcțiile au același nume). Pentru a evita astfel de situații este indicat să se specifice numele integral al funcției folosite utilizând operatorul `::`, de exemplu `dplyr::filter` folosește funcția `filter` din pachetul `dplyr`.

În cele ce urmează vom include, atât cât este posibil, și o comparație între funcțiile din `tidyverse` și cele din `R`-ul de bază.

2 Importarea datelor

În această secțiune vom prezenta o serie de modalități de bază de importare a seturilor de date în `R/RStudio`. Interfața `RStudio` permite importarea datelor din diverse surse prin efectuarea următorilor pași (interfața generează și codul corespunzător importării datelor):

1. Mergeți în tab-ul *Environment* (fereastra din dreapta sus) și selectați *Import Dataset*
2. Selectați tipul de date corespunzător fișierului pe care doriți să-l importați
3. Selectați fișierul din repertoriul de date

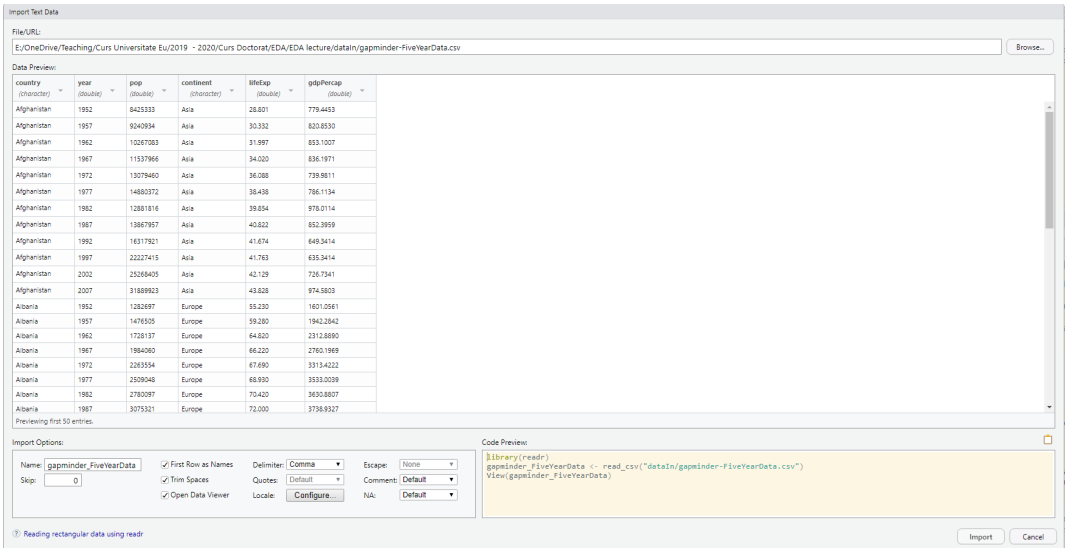

Fig. 1: Interfata de importare a datelor din RStudio

2.1 Fișiere de date de tip `csv`

Pachetul de bază din `tidyverse` care permite citirea fișierelor de tip `csv` este `readr`. Funcționalitățile acestui pachet permit și citirea/scrierea fișierelor de tip text, sau a fișierelor delimitate cu tab. Trebuie menționat că `R`-ul de bază vine cu funcționalități similare prin funcții precum `read.csv` sau `write.csv`. În cazul în care seturile de date sunt foarte mari atunci este indicată folosirea pachetului `data.table` prin `fread` și respectiv `fwrite`.

Funcția care permite citirea fișierelor de tip `csv` este `read_csv()`. Această funcție încearcă să detecteze automat tipurile de date și să le citească într-o structură de tip `data.frame`.

```
data_iris = read_csv("dataIn/iris.csv")
```

Funcția `read_csv()` admite o serie de argumete ce permit customizarea ei (a se vedea `?read_csv` pentru mai multe detalii):

- `col_names` - permite denumirea coloanelor din structura `data.frame` rezultată
- `col_types` - permite definirea tipurilor de date din fiecare coloană (înlocuid detectarea automată a acestora)
- `skip` - permite sărirea peste un anumit număr de linii atunci când este citit fișierul

```
data_iris = read_csv("dataIn/iris.csv",
                     col_names = c("sepal_length",
                                   "sepal_width",
                                   "petal_length",
                                   "petal_width",
                                   "species"),
                     skip=1)
```

Alternativ se pot folosi și alte funcții precum `read_csv2()` atunci când separatorul este `;`, `read_tsv()` atunci când separatorul este tab sau, mai general, `read_delim()` atunci când separatorul este un alt simbol.

În cazul în care dorim să scriem/salvăm un `data.frame`/set de date într-un fișier de tip `csv` atunci putem folosi funcția `write_csv()` din pachetul `readr` sau funcția `write.csv()` din pachetul de bază. Argumentele de bază ale funcției `write_csv()` sunt date de setul de date și numele și adresa fișierului în care salvăm:

```
write_csv(data_iris, "dataOut/iris.csv")
```

2.2 Fișiere de date de tip `xls` sau `xlsx`

Atunci când dorim să lucrăm cu fițiere de tip Excel, pachetul `readxl` asigură citirea acestor fișiere prin intermediul funcției `read_excel()` (citește doar primul sheet).

```
library(readxl)
data_iris_xlsx = read_excel("dataIn/iris.xlsx")
```

Ca și în cazul funcției `read_csv`, funcția `read_excel` admite o serie de argumente opționale.

```
data_iris_excel = read_excel("dataIn/iris.xlsx",
                             range = "B2:F13",
                             col_names = c("sepal_length",
                                           "sepal_width",
                                           "petal_length",
                                           "petal_width",
                                           "species"))
```

3 Metode de manipulare a datelor

Structurile de date de tip `data.frame` stau la baza analizei statistice în `R`. Pachetul `dplyr` furnizează o serie de funcționalități menite să asigure, într-un mod cât mai consistent și structurat - o gramatică, manipularea seturilor de date, (Wickham et al. 2019). Principalele operații sunt date de funcțiile:

- `%>%` - operatorul *pipe* permite scrierea/conectivitatea într-un mod logic a mai multor funcții
- `select()` - întoarce o submulțime de coloane (variabile) a `data.frame`-ului (setului de date) folosind o notație cât mai flexibilă
- `filter()` - extrage o submulțime de linii (observații) pe baza unor condiții/criterii logice
- `arrange()` - rearanjează observațiile
- `rename()` - redenumește variabilele
- `mutate()` - adaugă noi variabile sau modifică variabilele existente
- `group_by()` - grupează datele după diverse valori ale variabilelor calitative
- `summarise()` - sumarizează datele pentru diferite variabile, posibil pe straturi

Funcțiile pe care le vom prezenta în această secțiune prezintă o serie de caracteristici comune, precum:

- primul argument este un set de date sub formă de `data.frame`
- următoarele argumente descriu ce trebuie făcut cu setul de date specificat în primul argument (în acest caz se pot utiliza doar numele coloanelor (variabilelor) fără a mai folosi operatorul `$`)
- rezultatul obținut în urma aplicării funcției este tot un `data.frame`

3.1 Seturi de date folosite

În cele ce urmează vom descrie succint două seturi de date care ne vor ajuta la prezentarea noțiunilor/funcțiilor din pachetul `tidyverse`.

3.1.1 Setul de date `gapminder`

Pentru ilustrarea noțiunilor vom folosi setul de date gapminder care are 1704 observații (linii) ce conțin informații despre populația, durata de viață, GDP per capita pe an (perioada 1952 - 2007) și țară.

Pentru început înregistrăm setul de date (în memorie) folosind funcția `read_csv()`:

```
# inregistram setul de date
gapminder = read_csv("dataIn/gapminder-FiveYearData.csv")
```

Investigăm structura setului de date folosind funcții precum `dim()` (funcție de bază ce permite vizualizarea dimensiunii setului de date), `str()` (funcție de bază ce permite ilustrarea structurii setului de date), `glimpse()` (funcție din pachetul `tibble` ce prezintă într-o manieră mai compactă rezultatele funcției `str()`) și `head()/tail()` (funcții de bază ce afișează primele respectiv ultimele observații din setul de date):

```
# vedem dimensiunea acestuia
dim(gapminder)
[1] 1704    6

# ne uitam la structura lui
str(gapminder)
Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame':    1704 obs. of  6 variables:
 $ country  : chr  "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...
 $ year     : num  1952 1957 1962 1967 1972 ...
 $ pop      : num  8425333 9240934 10267083 11537966 13079460 ...
 $ continent: chr  "Asia" "Asia" "Asia" "Asia" ...
 $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
 $ gdpPercap: num  779 821 853 836 740 ...
 - attr(*, "spec")=
  .. cols(
  ..   country = col_character(),
  ..   year = col_double(),
  ..   pop = col_double(),
  ..   continent = col_character(),
  ..   lifeExp = col_double(),
  ..   gdpPercap = col_double()
  .. )

# sau folosind functia glimpse
glimpse(gapminder)
Observations: 1,704
Variables: 6
$ country   <chr> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan...
```

```
$ year      <dbl> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 199...
$ pop       <dbl> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372,...
$ continent <chr> "Asia", "Asia", "Asia", "Asia", "Asia", "Asia", "Asia", "...
$ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 4...
$ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.113...

# sau ne uitam la primele observatii
head(gapminder)
# A tibble: 6 x 6
  country      year      pop continent lifeExp gdpPercap
  <chr>       <dbl>    <dbl> <chr>       <dbl>     <dbl>
1 Afghanistan  1952  8425333 Asia         28.8      779.
2 Afghanistan  1957  9240934 Asia         30.3      821.
3 Afghanistan  1962 10267083 Asia         32.0      853.
4 Afghanistan  1967 11537966 Asia         34.0      836.
5 Afghanistan  1972 13079460 Asia         36.1      740.
6 Afghanistan  1977 14880372 Asia         38.4      786.

# sau ultimele observatii
tail(gapminder)
# A tibble: 6 x 6
  country   year      pop continent lifeExp gdpPercap
  <chr>    <dbl>    <dbl> <chr>       <dbl>     <dbl>
1 Zimbabwe  1982  7636524 Africa       60.4      789.
2 Zimbabwe  1987  9216418 Africa       62.4      706.
3 Zimbabwe  1992 10704340 Africa       60.4      693.
4 Zimbabwe  1997 11404948 Africa       46.8      792.
5 Zimbabwe  2002 11926563 Africa       40.0      672.
6 Zimbabwe  2007 12311143 Africa       43.5      470.
```

3.1.2 Setul de date `msleep`

Al doilea set de date pe care îl vom investiga este setul de date msleep (mammals sleep) care conține informații referitoare la timpii de somn și greutatea unor mamifere:

```
# importam datele
msleep = read_csv("dataIn/msleep_ggplot2.csv")

# vedem dimensiunea acestuia
dim(msleep)
[1] 83 11

# ne uitam la structura lui
str(msleep)
Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame':    83 obs. of  11 variables:
 $ name        : chr  "Cheetah" "Owl monkey" "Mountain beaver" "Greater short-tailed shrew" ...
 $ genus       : chr  "Acinonyx" "Aotus" "Aplodontia" "Blarina" ...
 $ vore        : chr  "carni" "omni" "herbi" "omni" ...
 $ order       : chr  "Carnivora" "Primates" "Rodentia" "Soricomorpha" ...
 $ conservation: chr  "lc" NA "nt" "lc" ...
 $ sleep_total : num  12.1 17 14.4 14.9 4 14.4 8.7 7 10.1 3 ...
 $ sleep_rem   : num  NA 1.8 2.4 2.3 0.7 2.2 1.4 NA 2.9 NA ...
 $ sleep_cycle : num  NA NA NA 0.133 0.667 ...
```

```
 $ awake       : num  11.9 7 9.6 9.1 20 9.6 15.3 17 13.9 21 ...
 $ brainwt     : num  NA 0.0155 NA 0.00029 0.423 NA NA NA 0.07 0.0982 ...
 $ bodywt      : num  50 0.48 1.35 0.019 600 ...
 - attr(*, "spec")=
  .. cols(
  ..   name = col_character(),
  ..   genus = col_character(),
  ..   vore = col_character(),
  ..   order = col_character(),
  ..   conservation = col_character(),
  ..   sleep_total = col_double(),
  ..   sleep_rem = col_double(),
  ..   sleep_cycle = col_double(),
  ..   awake = col_double(),
  ..   brainwt = col_double(),
  ..   bodywt = col_double()
  .. )

# sau folosind functia glimpse
glimpse(msleep)
Observations: 83
Variables: 11
$ name         <chr> "Cheetah", "Owl monkey", "Mountain beaver", "Greater s...
$ genus        <chr> "Acinonyx", "Aotus", "Aplodontia", "Blarina", "Bos", "...
$ vore         <chr> "carni", "omni", "herbi", "omni", "herbi", "herbi", "c...
$ order        <chr> "Carnivora", "Primates", "Rodentia", "Soricomorpha", "...
$ conservation <chr> "lc", NA, "nt", "lc", "domesticated", NA, "vu", NA, "d...
$ sleep_total  <dbl> 12.1, 17.0, 14.4, 14.9, 4.0, 14.4, 8.7, 7.0, 10.1, 3.0...
$ sleep_rem    <dbl> NA, 1.8, 2.4, 2.3, 0.7, 2.2, 1.4, NA, 2.9, NA, 0.6, 0....
$ sleep_cycle  <dbl> NA, NA, NA, 0.1333333, 0.6666667, 0.7666667, 0.3833333...
$ awake        <dbl> 11.9, 7.0, 9.6, 9.1, 20.0, 9.6, 15.3, 17.0, 13.9, 21.0...
$ brainwt      <dbl> NA, 0.01550, NA, 0.00029, 0.42300, NA, NA, NA, 0.07000...
$ bodywt       <dbl> 50.000, 0.480, 1.350, 0.019, 600.000, 3.850, 20.490, 0...

# sau ne uitam la primele observatii
head(msleep)
# A tibble: 6 x 11
  name  genus vore  order conservation sleep_total sleep_rem sleep_cycle awake
  <chr> <chr> <chr> <chr> <chr>              <dbl>     <dbl>       <dbl> <dbl>
1 Chee~ Acin~ carni Carn~ lc                  12.1      NA        NA      11.9
2 Owl ~ Aotus omni  Prim~ <NA>                17         1.8      NA       7
3 Moun~ Aplo~ herbi Rode~ nt                  14.4       2.4      NA       9.6
4 Grea~ Blar~ omni  Sori~ lc                  14.9       2.3       0.133   9.1
5 Cow   Bos   herbi Arti~ domesticated         4         0.7       0.667  20
6 Thre~ Brad~ herbi Pilo~ <NA>                14.4       2.2       0.767   9.6
# ... with 2 more variables: brainwt <dbl>, bodywt <dbl>
```

Variabilele, reprezentate prin coloane, corespund la: `name`- numele generic; `genus`- rangul taxonomic; `vore`- dacă este sau nu carnivor, omnivor sau ierbivor; `oreder`- ordinul taxonomic; `conservation`- statutul de conservare; `sleep_total`- durata totală de somn măsurată în ore; `sleep_rem`- numărul de ore în rem; `sleep_cycle`- durata ciclului de somn; `awake`- timpul petrecut treaz; `brainwt`- greutatea creierului în kg; `bodywt`- greutatea corporală în kg.

3.2 Operatorul %>%

Operatorul `%>%` (pipe) permite legarea/utilizarea împreunată a mai multor funcții, eliminând nevoia de a defini multiple obiecte intermediare ca elemente de input pentru funcțiile ulterioare. Acest operator vine din pachetul `magrittr` și poate fi citit *și apoi* (*and then*). Ca exemplu să considerăm o situație ipotetică în care vrem să aplicăm unui set de date `x` o serie de operații prin intermediul unor funcții `f()`, `g()` și `h()`: luăm `x` și apoi folosim `x` ca argument de intrare pentru `f` și apoi folosim rezultatul `f(x)` ca argument de intrare pentru `g` și apoi folosim rezultatul `g(f(x))` ca argument pentru `h` în vederea obținerii `h(g(f(x)))`. Putem folosi operatorul `%>%` pentru a obține această înșiruire de operații astfel:

```
x %>%
  f() %>%
  g() %>%
  h()
```

Un alt exemplu pe setul de date `gapminder`

```
gapminder %>%
  filter(continent == "Asia", year == "2007") %>%
  select(country, lifeExp)
```

poate fi citit ca *considerăm setul de date gapminder și apoi filtrăm după continentul Asia și anul 2007 și apoi selectăm țările și durata de viață*:

```
# setul de date gapminder
gapminder %>%
  # si filtram dupa continentul Asia si anul 2007
  filter(continent == "Asia", year == 2007) %>%
  # ilustram care sunt tarile si valorile duratei de viata pentru acestea
  select(country, lifeExp)
# A tibble: 33 x 2
   country         lifeExp
   <chr>             <dbl>
 1 Afghanistan        43.8
 2 Bahrain            75.6
 3 Bangladesh         64.1
 4 Cambodia           59.7
 5 China              73.0
 6 Hong Kong China    82.2
 7 India              64.7
 8 Indonesia          70.6
 9 Iran               71.0
10 Iraq               59.5
# ... with 23 more rows
```

În cazul în care nu am dori să folosim operatorul `%>%` atunci am fi putut scrie

```
gapminder_filtered = filter(gapminder, continent == "Asia", year == 2007)
gapminder_filtered_selected = select(gapminder_filtered, country, lifeExp)
gapminder_filtered_selected
```

dar versiunea inițială adaugă un plus de claritate la citire.

O scriere echivalentă a codului de mai sus folosind doar instrucțiunile de bază din `R` ar putea fi:

```
# identificam care linii corespund continentului Asia si anului 2007
continent_year_index <- which(gapminder["continent"] == "Asia" & gapminder["year"] == 2007)
```

```
# extragem acele linii si afisam tara si durata de viata
gapminder[continent_year_index, c("country", "lifeExp")]
```

3.3 Selectarea variabilelor - `select()`

Atunci când dorim să alegem un set de variabile (o serie de coloane) din setul nostru de date putem aplica funcția `select()`. Argumentele funcției `select()` specifică numele variabilelor pe care dorim să le păstrăm (putem folosi numele coloanelor fără să utilizăm ghilimele - dar se poate și cu ghilimele) separate prin virgulă:

- selectăm variabilele `country` și `gdpPercap` din setul de date `gapminder`

```
gapminder %>%
  select(country, gdpPercap) %>%
  head()
# A tibble: 6 x 2
  country     gdpPercap
  <chr>           <dbl>
1 Afghanistan      779.
2 Afghanistan      821.
3 Afghanistan      853.
4 Afghanistan      836.
5 Afghanistan      740.
6 Afghanistan      786.
```

- selectăm coloanele `name` și `sleep_total` din setul de date `msleep`

```
sleepData = select(msleep, name, sleep_total)
head(sleepData)
# A tibble: 6 x 2
  name                          sleep_total
  <chr>                               <dbl>
1 Cheetah                              12.1
2 Owl monkey                           17
3 Mountain beaver                      14.4
4 Greater short-tailed shrew           14.9
5 Cow                                   4
6 Three-toed sloth                     14.4
```

Dacă în setul nostru de date avem multe variabile pe care vrem să le păstrăm dar doar un număr mic pe care vrem să le excludem atunci putem folosi operatorul `-` în fața numelui coloanei/coloanelor pe care vrem să o/le excludem:

```
# scoatem variabila continent
gapminder %>%
  select(-continent) %>%
  head()
# A tibble: 6 x 5
  country      year      pop lifeExp gdpPercap
  <chr>       <dbl>    <dbl>   <dbl>     <dbl>
1 Afghanistan  1952  8425333    28.8      779.
2 Afghanistan  1957  9240934    30.3      821.
3 Afghanistan  1962 10267083    32.0      853.
4 Afghanistan  1967 11537966    34.0      836.
5 Afghanistan  1972 13079460    36.1      740.
6 Afghanistan  1977 14880372    38.4      786.
```

În situația în care dorim să selectăm/deselectăm toate variabilele cuprinse între `variabila1` și `variabila2` atunci putem folosi operatorul `:`.

```
# selectam toate variabilele intre var1 si var2
gapminder %>%
  select(year:lifeExp) %>%
  head()
# A tibble: 6 x 4
   year      pop continent lifeExp
  <dbl>    <dbl> <chr>       <dbl>
1  1952  8425333 Asia         28.8
2  1957  9240934 Asia         30.3
3  1962 10267083 Asia         32.0
4  1967 11537966 Asia         34.0
5  1972 13079460 Asia         36.1
6  1977 14880372 Asia         38.4

# deselectam toate variabilele intre var1 si var2
gapminder %>%
  select(-(year:lifeExp)) %>%
  head()
# A tibble: 6 x 2
  country     gdpPercap
  <chr>           <dbl>
1 Afghanistan      779.
2 Afghanistan      821.
3 Afghanistan      853.
4 Afghanistan      836.
5 Afghanistan      740.
6 Afghanistan      786.
```

Funcția `select()` poate fi folosită de asemenea și pentru a reordona coloanele/variabilele din setul de date atunci când este utilizată în conjucție cu `everything()`. De exemplu să presupunem că variabilele `pop`, `year` vrem să apară înaintea variabilei `country` și să păstrăm și celelalte variabile:

```
gapminder_reorder <- gapminder %>%
  select(pop, year, country, everything())
glimpse(gapminder_reorder)
Observations: 1,704
Variables: 6
$ pop       <dbl> 8425333, 9240934, 10267083, 11537966, 13079460, 14880372,...
$ year      <dbl> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992, 199...
$ country   <chr> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan...
$ continent <chr> "Asia", "Asia", "Asia", "Asia", "Asia", "Asia", "Asia", "...
$ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854, 4...
$ gdpPercap <dbl> 779.4453, 820.8530, 853.1007, 836.1971, 739.9811, 786.113...
```

Funcția `select()` permite, în momentul selectării coloanelor, și redenumirea acestora folosind argumente cu nume. Cu toate acestea, deoarece prin intermediul funcției `select()` se păstrează doar acele variabile menționate spre a fi selectate, această proprietate de redenumire nu este foarte folosită recomandată fiind utilizarea funcției `rename()`.

```
gapminder %>%
  select(gdp = gdpPercap) %>%
  head()
# A tibble: 6 x 1
```

```
    gdp
  <dbl>
1  779.
2  821.
3  853.
4  836.
5  740.
6  786.
```

Trebuie menționat că pachetul `tidyverse` (prin `tidyselect`) pune la dispoziție o serie de funcții ajutătoare care permit selectarea coloanelor setului de date după nume:

- `starts_with()` - întoarce coloanele în care șirul de caractere introdus se află la începutul numelui coloanei
- `ends_with()` - întoarce coloanele în care șirul de caractere introdus se află la sfârșitul numelui coloanei
- `contains()` - întoarce coloanele în care șirul de caractere introdus se află oriunde în numele coloanei
- `num_range()` - întoarce coloanele cu nume de tipul `Prefix 2017` până la `Prefix 2020` prin adăugarea prefixului și a valorilor numerice pe care vrem să le selectăm
- `matches()` - întoarce coloanele după un pattern
- `one_of()` - întoarce coloanele după un șir de nume predefinite

```
# daca vrem sa selectam coloanele/variabilele dupa un prefix/sufix
gapminder %>%
  select(starts_with("c")) %>%
  head()
# A tibble: 6 x 2
  country     continent
  <chr>       <chr>
1 Afghanistan Asia
2 Afghanistan Asia
3 Afghanistan Asia
4 Afghanistan Asia
5 Afghanistan Asia
6 Afghanistan Asia

gapminder %>%
  select(ends_with("p")) %>%
  head()
# A tibble: 6 x 3
      pop lifeExp gdpPercap
    <dbl>   <dbl>     <dbl>
1  8425333    28.8      779.
2  9240934    30.3      821.
3 10267083    32.0      853.
4 11537966    34.0      836.
5 13079460    36.1      740.
6 14880372    38.4      786.

# daca vrem sa selectam coloanele/variabilele dupa un text continut

gapminder %>%
  select(contains("en")) %>%
  head()
# A tibble: 6 x 1
  continent
```

```
  <chr>
1 Asia
2 Asia
3 Asia
4 Asia
5 Asia
6 Asia
```

Pentru mai multe detalii despre funcțiile ajutătoare se poate executa `?select_helpers`.

%TODO - scoping variables (advanced selection)

3.4 Selectarea observațiilor - `filter()`

Un alt aspect important atunci când prelucrăm/manipulăm un set de date este acela de a păstra doar observațiile care ne interesează sau pentru care analiza pe care urmează să o efectuăm este aplicabilă. În această secțiune vom prezenta două funcții care permit selectarea observațiilor (liniilor): `slice()` și `filter()`.

Atunci când vrem să selectăm observațiile după poziție vom folosi funcția `slice()` care primește ca argumente un vector de valori numerice întregi, pozitive sau negative după cum vrem să includem sau să excludem observațiile. Această funcție poate fi folosită și împreună cu funcția ajutătoare `n()` care întoarce numărul de linii al setului de date. Funcția `n()` poate fi utilizată doar în interiorul funcției `slice()` dar și în funcții precum `filter()`, `mutate()` sau `summarise()`.

```
# selectam primele 5 observatii
gapminder %>% slice(1:5)
# A tibble: 5 x 6
  country      year      pop continent lifeExp gdpPercap
  <chr>       <dbl>    <dbl> <chr>       <dbl>     <dbl>
1 Afghanistan  1952  8425333 Asia         28.8      779.
2 Afghanistan  1957  9240934 Asia         30.3      821.
3 Afghanistan  1962 10267083 Asia         32.0      853.
4 Afghanistan  1967 11537966 Asia         34.0      836.
5 Afghanistan  1972 13079460 Asia         36.1      740.

# excludem prima treime de observatii
gapminder %>%
  slice(-(1:floor(n()/3)))
# A tibble: 1,136 x 6
   country  year      pop continent lifeExp gdpPercap
   <chr>   <dbl>    <dbl> <chr>       <dbl>     <dbl>
 1 Germany  1972 78717088 Europe       71      18016.
 2 Germany  1977 78160773 Europe       72.5    20513.
 3 Germany  1982 78335266 Europe       73.8    22032.
 4 Germany  1987 77718298 Europe       74.8    24639.
 5 Germany  1992 80597764 Europe       76.1    26505.
 6 Germany  1997 82011073 Europe       77.3    27789.
 7 Germany  2002 82350671 Europe       78.7    30036.
 8 Germany  2007 82400996 Europe       79.4    32170.
 9 Ghana    1952  5581001 Africa       43.1      911.
10 Ghana    1957  6391288 Africa       44.8     1044.
# ... with 1,126 more rows
```

De cele mai multe ori, în practică, selectăm/filtrăm observațiile (liniile setului de date) după o serie de condiții logice. Funcția pe care o folosim atunci când vrem să selectăm observațiile după un criteriu logic este

funcția `filter()` (aceasta seamănă cu opțiunea *Filter* din Microsoft Excel). Primul argument al funcției este setul de date (un `data.frame`) iar următoarele argumete fac referire la expresii logice în care intervin variabilele (coloanele) acestuia. Funcția `filter()` întoarce acele linii (observații) pentru care expresiile logice sunt evaluate cu `TRUE`. De exemplu, dacă dorim să selectăm doar acele observații pentru care variabila `pop` (populația) este mai mare de 10^8 locuitori putem folosi o filtrare logică astfel

```
# toate obs/liniile pt care populatia este mai mare de 100 milioane de locuitori
gapminder %>%
  filter(pop > 1e8)
# A tibble: 77 x 6
   country     year       pop continent lifeExp gdpPercap
   <chr>      <dbl>     <dbl> <chr>       <dbl>     <dbl>
 1 Bangladesh  1987 103764241 Asia         52.8      752.
 2 Bangladesh  1992 113704579 Asia         56.0      838.
 3 Bangladesh  1997 123315288 Asia         59.4      973.
 4 Bangladesh  2002 135656790 Asia         62.0     1136.
 5 Bangladesh  2007 150448339 Asia         64.1     1391.
 6 Brazil      1972 100840058 Americas     59.5     4986.
 7 Brazil      1977 114313951 Americas     61.5     6660.
 8 Brazil      1982 128962939 Americas     63.3     7031.
 9 Brazil      1987 142938076 Americas     65.2     7807.
10 Brazil      1992 155975974 Americas     67.1     6950.
# ... with 67 more rows
```

O versiune echivalentă a codului de mai sus folosind instrucțiunile din `R`-ul de bază ar fi:

```
gapminder[gapminder$pop > 1e8, ]
# A tibble: 77 x 6
   country     year       pop continent lifeExp gdpPercap
   <chr>      <dbl>     <dbl> <chr>       <dbl>     <dbl>
 1 Bangladesh  1987 103764241 Asia         52.8      752.
 2 Bangladesh  1992 113704579 Asia         56.0      838.
 3 Bangladesh  1997 123315288 Asia         59.4      973.
 4 Bangladesh  2002 135656790 Asia         62.0     1136.
 5 Bangladesh  2007 150448339 Asia         64.1     1391.
 6 Brazil      1972 100840058 Americas     59.5     4986.
 7 Brazil      1977 114313951 Americas     61.5     6660.
 8 Brazil      1982 128962939 Americas     63.3     7031.
 9 Brazil      1987 142938076 Americas     65.2     7807.
10 Brazil      1992 155975974 Americas     67.1     6950.
# ... with 67 more rows
```

De asemenea, funcția `filter()` permite specificarea în paralel a mai multor condiții logice (folosind operatorii logici uzuali: `==`, `<`, `<=`, `>`, `>=`, `!=`, `%in%`) separate prin virgulă sau prin intermediul operatorilor AND - `&` sau OR - `|`.

```
# tarile din Asia din anii 1952, 1957

gapminder %>%
  filter(year %in% c(1952, 1957), continent == "Asia")
# A tibble: 66 x 6
   country      year       pop continent lifeExp gdpPercap
   <chr>       <dbl>     <dbl> <chr>       <dbl>     <dbl>
 1 Afghanistan  1952   8425333 Asia         28.8      779.
 2 Afghanistan  1957   9240934 Asia         30.3      821.
 3 Bahrain      1952    120447 Asia         50.9     9867.
```

```
 4 Bahrain      1957    138655  Asia         53.8    11636.
 5 Bangladesh   1952  46886859  Asia         37.5      684.
 6 Bangladesh   1957  51365468  Asia         39.3      662.
 7 Cambodia     1952   4693836  Asia         39.4      368.
 8 Cambodia     1957   5322536  Asia         41.4      434.
 9 China        1952 556263528. Asia         44        400.
10 China        1957 637408000  Asia         50.5      576.
# ... with 56 more rows

# si care erau tarile cu o astfel de populatie in 1992
gapminder %>%
  filter(pop > 100000000, year == 1992)
# A tibble: 8 x 6
  country        year        pop continent lifeExp gdpPercap
  <chr>         <dbl>      <dbl> <chr>       <dbl>     <dbl>
1 Bangladesh     1992  113704579 Asia         56.0      838.
2 Brazil         1992  155975974 Americas     67.1     6950.
3 China          1992 1164970000 Asia         68.7     1656.
4 India          1992  872000000 Asia         60.2     1164.
5 Indonesia      1992  184816000 Asia         62.7     2383.
6 Japan          1992  124329269 Asia         79.4    26825.
7 Pakistan       1992  120065004 Asia         60.8     1972.
8 United States  1992  256894189 Americas     76.1    32004.
```

%TODO - scoping variables (advanced filtering)

3.5 Rearanjarea datelor - `arrange()`

Sunt multe situațiile în care dorim să aranjăm setul de date sau prin schimbarea poziției variabilelor sau prin ordonarea observațiilor după o ordine alphanumerică efectuată în funcție de valorile unei variabile date.

Atunci când dorim rearanjarea variabilelor (a coloanelor) setului de date putem utiliza funcția `select()` împreună cu funcția ajutătoare `everything()`. După cum am văzut într-un exemplu anterior, să presupunem că variabilele `country`, `continent` vrem să apară înaintea celorlalte variabile:

```
gapminder %>%
  select(starts_with("c"), everything()) %>%
  head()
# A tibble: 6 x 6
  country     continent  year      pop lifeExp gdpPercap
  <chr>       <chr>     <dbl>    <dbl>   <dbl>     <dbl>
1 Afghanistan Asia       1952  8425333    28.8      779.
2 Afghanistan Asia       1957  9240934    30.3      821.
3 Afghanistan Asia       1962 10267083    32.0      853.
4 Afghanistan Asia       1967 11537966    34.0      836.
5 Afghanistan Asia       1972 13079460    36.1      740.
6 Afghanistan Asia       1977 14880372    38.4      786.
```

Dacă dorim sortarea alfabetică a variabilelor atunci avem nevoie să extragem numele acestora, pas efectuat prin aplicarea funcției `current_vars()` (sau mai nou `tidyselect::peek_vars()`), și apoi sortarea acestora:

```
gapminder %>%
  select(sort(current_vars())) %>%
  head()
# A tibble: 6 x 6
```

```
  continent country     gdpPercap lifeExp      pop  year
  <chr>     <chr>           <dbl>   <dbl>    <dbl> <dbl>
1 Asia      Afghanistan      779.    28.8  8425333  1952
2 Asia      Afghanistan      821.    30.3  9240934  1957
3 Asia      Afghanistan      853.    32.0 10267083  1962
4 Asia      Afghanistan      836.    34.0 11537966  1967
5 Asia      Afghanistan      740.    36.1 13079460  1972
6 Asia      Afghanistan      786.    38.4 14880372  1977
```

În situația în care dorim să ordonăm observațiile după valorile unei/sau mai multor variabile/coloane atunci folosim funcția `arrange()`. Funcția `arrange()` primește ca argumente numele coloanei sau a coloanelor (separate prin virgulă) după care se efectuază sortarea. Sortarea se face în ordine crescătoare, în caz că se dorește sortarea în ordine descrescătoare se aplică funcția `desc()` variabilei respective.

De exemplu dacă dorim să aranjăm observațiile crescător după durata de viață atunci

```
gapminder %>%
  arrange(lifeExp) %>%
  head
# A tibble: 6 x 6
  country       year     pop continent lifeExp gdpPercap
  <chr>        <dbl>   <dbl> <chr>       <dbl>     <dbl>
1 Rwanda        1992 7290203 Africa       23.6      737.
2 Afghanistan   1952 8425333 Asia         28.8      779.
3 Gambia        1952  284320 Africa       30        485.
4 Angola        1952 4232095 Africa       30.0     3521.
5 Sierra Leone  1952 2143249 Africa       30.3      880.
6 Afghanistan   1957 9240934 Asia         30.3      821.
```

iar dacă dorim să le aranjăm crescător după an și descrescător după populație atunci

```
gapminder %>%
  arrange(year, desc(pop)) %>%
  head
# A tibble: 6 x 6
  country        year        pop continent lifeExp gdpPercap
  <chr>         <dbl>      <dbl> <chr>       <dbl>     <dbl>
1 China          1952 556263528. Asia         44        400.
2 India          1952 372000000  Asia         37.4      547.
3 United States  1952 157553000  Americas     68.4    13990.
4 Japan          1952  86459025  Asia         63.0     3217.
5 Indonesia      1952  82052000  Asia         37.5      750.
6 Germany        1952  69145952  Europe       67.5     7144.
```

%TODO - advanced ordering

3.6 Modificarea/redenumirea variabilelor - `mutate()/rename()`

Atunci când vrem să schimbăm numele unor variabile din setul de date cu care lucrăm vom folosi comanda `rename()` (am văzut că putem schimba numele variabilelor de interes și prin intermediul funcției `select()`). Această funcție permite redenumirea variabilelor, prin intermendiul operatorului `=` (*noul nume* = *vechiul nume*), de interes și *păstrarea* celorlalte variabile.

```
gapminder %>%
  rename(gdp = gdpPercap) %>%
```

```
  head
# A tibble: 6 x 6
  country      year      pop continent lifeExp   gdp
  <chr>       <dbl>    <dbl> <chr>       <dbl> <dbl>
1 Afghanistan  1952  8425333 Asia         28.8  779.
2 Afghanistan  1957  9240934 Asia         30.3  821.
3 Afghanistan  1962 10267083 Asia         32.0  853.
4 Afghanistan  1967 11537966 Asia         34.0  836.
5 Afghanistan  1972 13079460 Asia         36.1  740.
6 Afghanistan  1977 14880372 Asia         38.4  786.
```

Sunt multe situațiile în care, pe parcursul analizei, ne dorim să adăugăm la setul de date (sau să lucrăm cu) noi variabile care să fie obținute prin transformarea unor variabile deja existente. Funcția `mutate()` permite exact acest lucru, i.e. crearea de variabile (adăugate la sfârșitul setului de date) derivate din variabilele deja existente. De exemplu putem construi variabila `gdp_total` ca fiind obținută prin înmulțirea dintre variabilele `gdpPerCap` și `pop`:

```
gapminder %>%
  mutate(gdp_total = gdpPercap * pop) %>%
  head()
# A tibble: 6 x 7
  country      year      pop continent lifeExp gdpPercap    gdp_total
  <chr>       <dbl>    <dbl> <chr>       <dbl>     <dbl>        <dbl>
1 Afghanistan  1952  8425333 Asia         28.8      779.  6567086330.
2 Afghanistan  1957  9240934 Asia         30.3      821.  7585448670.
3 Afghanistan  1962 10267083 Asia         32.0      853.  8758855797.
4 Afghanistan  1967 11537966 Asia         34.0      836.  9648014150.
5 Afghanistan  1972 13079460 Asia         36.1      740.  9678553274.
6 Afghanistan  1977 14880372 Asia         38.4      786. 11697659231.
```

Atunci când folosim funcția `mutate()` putem crea atât variabile care depind de coloanele existente cât și variabile care pot depinde de variabile construite în același timp cu acestea. Pentru a ilustra această proprietate vom construi pe lângă variabila `gdp_total` și variabila `gdp_trend` obținută prin scăderea din variabila `gdpPercap` a mediei variabilei `gdp_total`:

```
gapminder %>%
  mutate(gdp_total = gdpPercap * pop,
         gdp_trend = gdpPercap - mean(gdp_total)) %>%
  head()
# A tibble: 6 x 8
  country      year      pop continent lifeExp gdpPercap    gdp_total gdp_trend
  <chr>       <dbl>    <dbl> <chr>       <dbl>     <dbl>        <dbl>     <dbl>
1 Afghanistan  1952  8425333 Asia         28.8      779.  6567086330.  -1.87e11
2 Afghanistan  1957  9240934 Asia         30.3      821.  7585448670.  -1.87e11
3 Afghanistan  1962 10267083 Asia         32.0      853.  8758855797.  -1.87e11
4 Afghanistan  1967 11537966 Asia         34.0      836.  9648014150.  -1.87e11
5 Afghanistan  1972 13079460 Asia         36.1      740.  9678553274.  -1.87e11
6 Afghanistan  1977 14880372 Asia         38.4      786. 11697659231.  -1.87e11
```

O funcție similară cu `mutate()` este funcția `transmute()` excepție făcând faptul că aplicarea acesteia conduce la un set de date în care sunt păstrate doar variabilele transformate *nu* și cele netransformate:

```
gapminder %>%
  transmute(gdp_total = gdpPercap * pop,
         gdp_trend = gdpPercap - mean(gdp_total)) %>%
  head()
```

```
# A tibble: 6 x 2
     gdp_total gdp_trend
         <dbl>     <dbl>
1  6567086330.  -1.87e11
2  7585448670.  -1.87e11
3  8758855797.  -1.87e11
4  9648014150.  -1.87e11
5  9678553274.  -1.87e11
6 11697659231.  -1.87e11
```

Pachetul `dplyr` pune la dispoziție o serie de funcții ajutătoare care pot fi folosite împreună cu funcția `mutate()`: `row_numbers()`, `lead()`, `lag()`, `case_when()`, etc.. De exemplu atunci când dorim să adăugăm coloana `ID` la setul de date putem folosi funcția `row_number()`:

```
gapminder %>%
  mutate(ID = row_number()) %>%
  head()
# A tibble: 6 x 7
  country      year      pop continent lifeExp gdpPercap    ID
  <chr>       <dbl>    <dbl> <chr>       <dbl>     <dbl> <int>
1 Afghanistan  1952  8425333 Asia         28.8      779.     1
2 Afghanistan  1957  9240934 Asia         30.3      821.     2
3 Afghanistan  1962 10267083 Asia         32.0      853.     3
4 Afghanistan  1967 11537966 Asia         34.0      836.     4
5 Afghanistan  1972 13079460 Asia         36.1      740.     5
6 Afghanistan  1977 14880372 Asia         38.4      786.     6
```

iar când dorim să comparăm valorile în timp putem folosi funcțiile `lag()` și respectiv `lead()`:

```
gapminder %>%
  mutate(gdpPercap_prev = lag(gdpPercap),
         gdpPercap_future = lead(gdpPercap)) %>%
  select(country, gdpPercap, gdpPercap_prev, gdpPercap_future) %>%
  head()
# A tibble: 6 x 4
  country     gdpPercap gdpPercap_prev gdpPercap_future
  <chr>           <dbl>          <dbl>            <dbl>
1 Afghanistan      779.           NA               821.
2 Afghanistan      821.          779.              853.
3 Afghanistan      853.          821.              836.
4 Afghanistan      836.          853.              740.
5 Afghanistan      740.          836.              786.
6 Afghanistan      786.          740.              978.
```

Funcția `case_when()` poate fi utilă atunci când dorim să scriem mai multe expresii condiționale, evitând să folosim `ifelse` în mod repetat (în special când dorim să creăm variabile calitative). Ca argumente de intrare a funcției avem nevoie de una sau mai multe expresii (condiționale) care să returneze valori booleene (TRUE sau FALSE) și pentru care asociem eticheta corespunzătoare prin intermediul simbolului `~`. Funcția returnează pentru fiecare observație eticheta (label-ul) primei expresii care întoarce valoarea de adevăr `TRUE`. Pentru a returna o valoare pentru situațiile neprevăzute în expresiile condiționale se folosește ca ultimă condiție `TRUE ~ 'altele'`:

```
gapminder %>%
  mutate(size = case_when(
    pop < mean(pop, trim = 0.1) ~ "small",
    pop > mean(pop, trim = 0.1) ~ "large",
```

```
    TRUE ~ "moderate"
  )) %>%
  head()
# A tibble: 6 x 7
  country      year      pop continent lifeExp gdpPercap size
  <chr>       <dbl>    <dbl> <chr>       <dbl>     <dbl> <chr>
1 Afghanistan  1952  8425333 Asia         28.8      779. small
2 Afghanistan  1957  9240934 Asia         30.3      821. small
3 Afghanistan  1962 10267083 Asia         32.0      853. small
4 Afghanistan  1967 11537966 Asia         34.0      836. large
5 Afghanistan  1972 13079460 Asia         36.1      740. large
6 Afghanistan  1977 14880372 Asia         38.4      786. large
```

Pentru mai multe detalii se poate consulta (Wickham and Grolemund 2017, Capitolul 3).

3.7 Gruparea și sumarizarea datelor - `group_by()/summarise()`

O altă funcție importantă în manipularea seturilor de date este funcția `summarise()`. Aceasta se folosește de cele mai multe ori împreună cu funcția `group_by()` și permite agregarea datelor pe grupuri/straturi (determinate de valorile unei variabile discrete).

Atunci când funcția `summarise()` este folosită singură pe un `data.frame` aceasta restrânge setul de date la un singur rând a cărui valori sunt obținute prin agregarea valorilor de pe coloanele respective. Funcția primește ca prim argument setul de date urmat de o listă de variabile care vor apărea ca valori de output. Este important de specificat că fiecare variabilă de ieșire trebuie să fie definită prin operații care se efectuează pe *vectori* și nu pe scalari (e.g. `sum`, `max`, `min`, `mean`, `sd`, `var`, `median`, etc.).

Spre exemplu, să considerăm setul de date `gapminder` pentru care vrem să calculăm media duratei de viață și produsul intern brut total:

```
gapminder %>%
  summarise(count = n(),
            mean_lifeExp = mean(lifeExp),
            total_gdp = sum(gdpPercap)) %>%
  head()
# A tibble: 1 x 3
  count mean_lifeExp total_gdp
  <int>        <dbl>     <dbl>
1  1704         59.5 12294917.
```

Funcția ajutătoare `n()` întoarce numărul de linii pentru care se aplică sumarizarea datelor și este recomandată utilizarea ei ori de câte ori are loc o agregare a datelor. Alte funcții ajutătoare sunt: `n_distinct()` - întoarce numărul valorilor unice dintr-o variabilă; `first()`, `last()` și `nth()` - întorc prima, ultima și respectiv a n-a valoare.

```
gapminder %>%
  summarise(count = n(),
            unique_countries = n_distinct(country),
            first_country = first(country),
            last_country = last(country),
            nth_country = nth(country, 20))
# A tibble: 1 x 5
  count unique_countries first_country last_country nth_country
  <int>           <int> <chr>         <chr>        <chr>
1  1704             142 Afghanistan   Zimbabwe     Albania
```

Funcția `summarise()` se folosește predominant în conjuncție cu funcția `group_by()` care permite efectuarea de operații și agregarea datelor pe straturi definite de valorile uneia sau a mai multor variabile. Aplicarea funcției `group_by()` permite schimbarea unității de analiză de la întregul set de date la partiția definită de valorile variabilelor selectate (putem interpreta că avem mai multe seturi de date). Toate funcțiile care se aplică după variabila de grupare se aplică pentru fiecare nivel al acesteia (se aplică separat pentru fiecare grup). Astfel funcțiile de manipulare deja specificate se vor aplica pentru fiecare grup/partiție din setul de date. În cazul în care dorim să lucrăm iar pe întregul set de date apelăm funcția `ungroup()`.

În exemplul de mai jos, setul de date este filtrat după acele țări și acei ani pentru care durata de viață este mai mare decât media duratei de viață pe continent:

```
gapminder %>%
  group_by(continent) %>%
  filter(lifeExp > mean(lifeExp)) %>%
  ungroup()
# A tibble: 873 x 6
   country  year      pop continent lifeExp gdpPercap
   <chr>   <dbl>    <dbl> <chr>       <dbl>     <dbl>
 1 Albania  1987  3075321 Europe       72       3739.
 2 Albania  1997  3428038 Europe       73.0     3193.
 3 Albania  2002  3508512 Europe       75.7     4604.
 4 Albania  2007  3600523 Europe       76.4     5937.
 5 Algeria  1967 12760499 Africa       51.4     3247.
 6 Algeria  1972 14760787 Africa       54.5     4183.
 7 Algeria  1977 17152804 Africa       58.0     4910.
 8 Algeria  1982 20033753 Africa       61.4     5745.
 9 Algeria  1987 23254956 Africa       65.8     5681.
10 Algeria  1992 26298373 Africa       67.7     5023.
# ... with 863 more rows
```

Pentru a evidenția diferența dintre rezultatul grupat și cel negrupat după variabila `continent` vom selecta datele corespunzătoare anului 2007 și număra, folosind funcția `count()`, câte țări de pe fiecare continent au durata de viață mai mare decât media pe continent, în cazul în care grupăm, și respectiv media toatală, în cazul în care nu grupăm:

```
gapminder %>%
  filter(year == 2007) %>%
  group_by(continent) %>%
  filter(lifeExp > mean(lifeExp)) %>%
  ungroup() %>%
  count(continent)
# A tibble: 5 x 2
  continent     n
  <chr>     <int>
1 Africa       22
2 Americas     12
3 Asia         20
4 Europe       18
5 Oceania       1
```

Observăm că pe continentul African sunt considerabil mai puține țări care au o durată de viață mai mare decât media totală de 67 ani pe când în Asia, Europa și America sunt mai multe țări a căror durată de viață medie o depășește pe cea totală.

```
gapminder %>%
  filter(year == 2007) %>%
  filter(lifeExp > mean(lifeExp)) %>%
```

```
  count(continent)
# A tibble: 5 x 2
  continent     n
  <chr>     <int>
1 Africa        7
2 Americas     23
3 Asia         23
4 Europe       30
5 Oceania       2
```

Un alt exemplu în care vrem să calculăm durata de viață medie și produsul intern brut total pentru fiecare an din setul nostru de date este:

```
gapminder %>%
  # grupam pe an
  group_by(year) %>%
  summarise(count = n(),
            mean_life_yr = mean(lifeExp),
            total_gdp_yr = sum(gdpPercap))%>%
  head()
# A tibble: 6 x 4
   year count mean_life_yr total_gdp_yr
  <dbl> <int>        <dbl>        <dbl>
1  1952   142         49.1      528989.
2  1957   142         51.5      610516.
3  1962   142         53.6      671065.
4  1967   142         55.7      778679.
5  1972   142         57.6      961352.
6  1977   142         59.6     1038470.
```

În cazul în care vrem să adăugăm la setul de date o nouă coloană care să conțină media pe ani (anii disponibili în setul de date) a produsului intern brut pentru fiecare țară scriem:

```
gapminder %>%
  group_by(country) %>%
  mutate(mean_gdp = mean(gdpPercap)) %>%
  head()
# A tibble: 6 x 7
# Groups:   country [1]
  country      year      pop continent lifeExp gdpPercap mean_gdp
  <chr>       <dbl>    <dbl> <chr>       <dbl>     <dbl>    <dbl>
1 Afghanistan  1952  8425333 Asia         28.8      779.     803.
2 Afghanistan  1957  9240934 Asia         30.3      821.     803.
3 Afghanistan  1962 10267083 Asia         32.0      853.     803.
4 Afghanistan  1967 11537966 Asia         34.0      836.     803.
5 Afghanistan  1972 13079460 Asia         36.1      740.     803.
6 Afghanistan  1977 14880372 Asia         38.4      786.     803.
```

Pentru a vedea dacă am obținut într-adevăr valorile corecte să ne oprim asupra țării "Afghanistan" și să calculăm media valorilor produsului intern brut:

```
gapminder %>%
  filter(country == "Afghanistan") %>%
  summarise(mean_Afghanistan_gdp = mean(gdpPercap))
# A tibble: 1 x 1
  mean_Afghanistan_gdp
```

```
                <dbl>
1                803.
```

De asemenea putem afișa primele trei țări de pe fiecare continent ordonate descrescător în funcție de media produsului intern brut (media calculată pe ani):

```
gapminder %>%
  group_by(continent, country) %>%
  summarise(gdp = mean(gdpPercap)) %>%
  group_by(continent) %>%
  arrange(desc(gdp)) %>%
  slice(1:3)
# A tibble: 14 x 3
# Groups:   continent [5]
   continent country          gdp
   <chr>     <chr>          <dbl>
 1 Africa    Libya         12014.
 2 Africa    Gabon         11530.
 3 Africa    South Africa   7247.
 4 Americas  United States 26261.
 5 Americas  Canada        22411.
 6 Americas  Puerto Rico   10863.
 7 Asia      Kuwait        65333.
 8 Asia      Saudi Arabia  20262.
 9 Asia      Bahrain       18078.
10 Europe    Switzerland   27074.
11 Europe    Norway        26747.
12 Europe    Netherlands   21749.
13 Oceania   Australia     19981.
14 Oceania   New Zealand   17263.
```

Să presupunem de asemenea că dorim să cunoaștem care sunt valorile medii ale duratei de viață în raport cu cuantilele produsului intern brut:

```
q_gdp <- quantile(gapminder$gdpPercap, seq(0, 1, 0.2), na.rm = TRUE)

gapminder %>%
  mutate(q_gdp = cut(gdpPercap, q_gdp)) %>%
  group_by(q_gdp) %>%
  summarise(life_mean = mean(lifeExp, na.rm = TRUE))
# A tibble: 6 x 2
  q_gdp               life_mean
  <fct>                   <dbl>
1 (241,976]                45.2
2 (976,2.28e+03]           51.3
3 (2.28e+03,5.15e+03]      59.5
4 (5.15e+03,1.14e+04]      68.0
5 (1.14e+04,1.14e+05]      73.4
6 <NA>                     45.0
```

3.8 Aducerea seturilor de date la un format `tidy`

Sunt multe situațiile în care seturile de date pe care urmează să le analizăm nu au formatul *dreptunghiular* cu care ne-am obișnuit, i.e. observațiile pe linii și variabilele pe coloane, și în aceste cazuri analiza poate fi

dificilă. De multe ori aceleași date/informații pot fi organizate în moduri diferite conducând la forme mai complexe sau mai simple de analizat. De exemplu, următorul set de date

country	year	pop	lifeExp
Afghanistan	1997	22227415	41.763
Afghanistan	2007	31889923	43.828
Brazil	1997	168546719	69.388
Brazil	2007	190010647	72.390
China	1997	1230075000	70.426
China	2007	1318683096	72.961
Romania	1997	22562458	69.720
Romania	2007	22276056	72.476

poate fi scris și sub forma

country	year	type	count
Afghanistan	1997	pop	2.222742e+07
Afghanistan	1997	lifeExp	4.176300e+01
Afghanistan	2007	pop	3.188992e+07
Afghanistan	2007	lifeExp	4.382800e+01
Brazil	1997	pop	1.685467e+08
Brazil	1997	lifeExp	6.938800e+01
Brazil	2007	pop	1.900106e+08
Brazil	2007	lifeExp	7.239000e+01
China	1997	pop	1.230075e+09
China	1997	lifeExp	7.042600e+01
China	2007	pop	1.318683e+09
China	2007	lifeExp	7.296100e+01
Romania	1997	pop	2.256246e+07
Romania	1997	lifeExp	6.972000e+01
Romania	2007	pop	2.227606e+07
Romania	2007	lifeExp	7.247600e+01

sau sub forma compactă

country	year	rate
Afghanistan	1997	41.763/22227415
Afghanistan	2007	43.828/31889923
Brazil	1997	69.388/168546719
Brazil	2007	72.39/190010647
China	1997	70.426/1230075000
China	2007	72.961/1318683096
Romania	1997	69.72/22562458
Romania	2007	72.476/22276056

sau încă sub forma a două tabele

country	1997	2007
Afghanistan	22227415	31889923
Brazil	168546719	190010647
China	1230075000	1318683096
Romania	22562458	22276056

country	1997	2007
Afghanistan	41.763	43.828
Brazil	69.388	72.390
China	70.426	72.961
Romania	69.720	72.476

Spunem că un set de date este în format *tidy*, are o structură organizată, dacă îndeplinește următoarele condiții (a se vedea (Wickham 2014)):

- fiecare variabilă formează o coloană
- fiecare observație formează o linie
- fiecare valoare are celula sa proprie

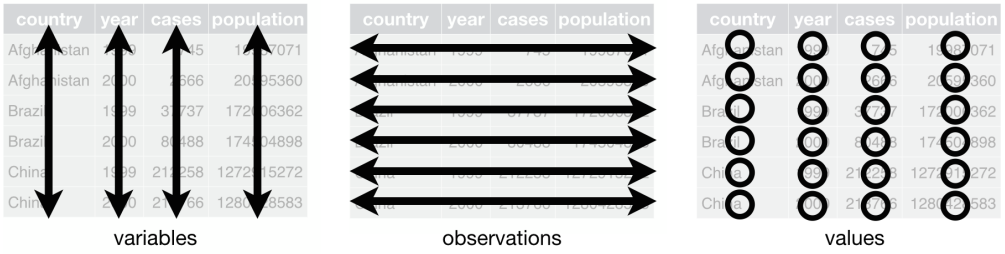

Fig. 2: Ilustrarea principiului de date tidy conform Wickham

În exemplul nostru, doar primul tabel verifică structura de date de tip *tidy*, i.e. fiecare unitate observațională corespunde unei linii și fiecare variabilă corespunde unei coloane. Avantajul datelor de tip *tidy* este că, având o formă standardizată de reprezentare a datelor, permite analistului să extragă cu mai mare ușurință informațiile necesare (Wickham and Grolemund 2017).

Scopul acestei secțiuni este de a introduce două funcții care permit aducerea/transformarea datelor la tipul de date *tidy*: `pivot_longer()` și `pivot_wider()` (Wickham and Henry 2020). În general, datele reale nu vin întotdeauna în formatul *tidy* iar aducerea lor la acest format necesită, în primul rând, identificarea variabilelor și a observațiilor. Două dintre problemele cel mai des întâlnite sunt că valorile unei variabile pot fi împrăștiate pe mai multe coloane și că o observație poate fi împărțită pe mai multe linii.

3.8.1 Funcția `pivot_longer()`

Funcția `pivot_longer()` primește ca prim argument un `data.frame` care specifică în mod precis cum metadatele stocate în numele coloanelor devin valorile unor variabile. Această funcție transformă un set de date (în acest caz mai lat - *wide*) într-un alt set de date mai lung prin creșterea numărului de linii și scăderea numărului de coloane.

Pentru a exemplifica modul de aplicare a acestei funcții vom folosi un set de date în format brut descărcat de pe platforma www.gapminder.org/data/, mai precis vom descărca în format `csv` datele referitoare la durata de viață (*Health -> Life expectancy* sau http://gapm.io/ilex) și respectiv populația totală (*Population -> Population* sau http://gapm.io/dpop) pe perioada 1800 - 2018.

```
gap_life_exp = read_csv("dataIn/life_expectancy_years.csv")
gap_pop_total = read_csv("dataIn/population_total.csv")
```

Observăm că ambele seturi de date au un număr mare de coloane, 220 și respectiv 302, tabelul de mai jos ilustrând o parte dintre acestea pentru setul `gap_life_exp`:

country	1800	1801	1802	2016	2017	2018
Afghanistan	28.2	28.2	28.2	58.0	58.4	58.7
Albania	35.4	35.4	35.4	77.7	77.9	78.0
Algeria	28.8	28.8	28.8	77.4	77.6	77.9
Andorra	NA	NA	NA	82.5	NA	NA
Angola	27.0	27.0	27.0	64.7	64.9	65.2
Antigua and Barbuda	33.5	33.5	33.5	77.3	77.4	77.6

Putem remarca faptul că setul `gap_life_exp` conține trei variabile: variabila `country` înregistrată pe linii, variabila `year` împrăștiată pe coloane și variabila `lifeExp` a cărei valori corespund valorilor din celule. Pentru a aduce acest set de date la formatul *tidy* vom aplica funcția `pivot_longer` astfel

```
gap_life_exp %>%
  pivot_longer(-country, names_to = "year", values_to = "lifeExp") %>%
  head()
# A tibble: 6 x 3
  country     year  lifeExp
  <chr>       <chr>   <dbl>
1 Afghanistan 1800     28.2
2 Afghanistan 1801     28.2
3 Afghanistan 1802     28.2
4 Afghanistan 1803     28.2
5 Afghanistan 1804     28.2
6 Afghanistan 1805     28.2
```

Structura primară a funcției `pivot_longer()` este următoarea :

- primul argument este setul de date `gap_life_exp` (unde s-a folosit notația pipe `%>%`)
- al doilea argument este dat de coloanele care trebuie transformate (poate fi specificată și prin argumetul `cols`), în cazul nostru toate coloanele cu excepția coloanei `country` (sau `cols = -country`)
- argumentul `names_to` precizează numele variabilei care va fi creată în noul set de date și a cărei valori vor fi date de numele coloanelor selectate în setul de date original, în cazul nostru `year`
- argumentul `values_to` precizează numele variabilei din setul de date *tidy* care va conține ca valori datele stocate în celulele setului de date original, în cazul nostru `lifeExp`

Dacă dorim să selectăm doar o submulțime de valori care corespund numelor coloanelor din setul de date original atunci putem specifica care sunt aceste valori atributului `cols`:

```
gap_life_exp %>%
  pivot_longer(cols = `1800`:`1850`,
               names_to = "year",
               values_to = "lifeExp") %>%
  distinct(year) %>% # valorile distincte ale variabilei year
  count() # numarul valorilor distincte
# A tibble: 1 x 1
      n
  <int>
1    51
```

De asemenea funcțiile ajutătoare `starts_with()`, `ends_with()`, `contains()`, etc. pot fi folosite împreună cu atributul `cols`:

```
gap_life_exp %>%
  pivot_longer(cols = starts_with("19"),
               names_to = "year",
               values_to = "lifeExp") %>%
  distinct(year) %>%
  count()
# A tibble: 1 x 1
      n
  <int>
1   100
```

Să presupunem acum că am fi încărcat setul de date `life_expectancy_years.csv` folosind comanda de bază `read.csv()`. În această situație variabilele `1800-2018` vor prezenta un prefix `X` în față, i.e. `X1800-X2018`. Putem folosi funcția `pivot_longer()` și în acest caz înlăturând prefixul dat astfel:

```
gap_life_exp2 = read.csv("dataIn/life_expectancy_years.csv")

gap_life_exp2 %>%
  pivot_longer(cols = starts_with("X"),
               names_to = "year",
               names_prefix = "X",
               names_ptypes = list(year = integer()),
               values_to = "lifeExp") %>%
  head()
# A tibble: 6 x 3
  country      year lifeExp
  <fct>       <int>   <dbl>
1 Afghanistan  1800    28.2
2 Afghanistan  1801    28.2
3 Afghanistan  1802    28.2
4 Afghanistan  1803    28.2
5 Afghanistan  1804    28.2
6 Afghanistan  1805    28.2
```

unde atributul `names_prefix` înlătură prefixul `X` iar atributul `names_ptypes` specifică tipul de date pe care îl va avea variabila `year`, în acest caz întreg. Pentru mai multe detalii și exemple despre cum poate fi folosită funcția `pivot_longer` în diferite contexte se poate folosi documentația apelând `vignette("pivot")`.

3.8.2 Funcția `pivot_wider()`

Funcția `pivot_wider()` are rolul opus funcției `pivot_longer()` și anume transformă un set de date într-un set de date lat (*wide*) prin creșterea numărului de coloane și scăderea numărului de linii. Se folosește în special atunci când mai multe variabile sunt stocate într-o singură coloană.

Să presupunem că avem următorul set de date

```
gap_tab2
# A tibble: 16 x 4
   country      year type           count
   <chr>       <dbl> <chr>          <dbl>
 1 Afghanistan  1997 pop        22227415
 2 Afghanistan  1997 lifeExp          41.8
 3 Afghanistan  2007 pop        31889923
```

```
 4 Afghanistan  2007 lifeExp         43.8
 5 Brazil       1997 pop      168546719
 6 Brazil       1997 lifeExp         69.4
 7 Brazil       2007 pop      190010647
 8 Brazil       2007 lifeExp         72.4
 9 China        1997 pop     1230075000
10 China        1997 lifeExp         70.4
11 China        2007 pop     1318683096
12 China        2007 lifeExp         73.0
13 Romania      1997 pop       22562458
14 Romania      1997 lifeExp         69.7
15 Romania      2007 pop       22276056
16 Romania      2007 lifeExp         72.5
```

în care observăm că variabila `type` conține valorile a două variabile `pop` și respectiv `lifeExp`. Vom folosi funcția `pivot_wider()` pentru a aduce setul de date la formatul *tidy* dorit:

```
gap_tab2 %>%
  pivot_wider(names_from = "type", values_from = "count") %>%
  head()
# A tibble: 6 x 4
  country      year        pop lifeExp
  <chr>       <dbl>      <dbl>   <dbl>
1 Afghanistan  1997   22227415    41.8
2 Afghanistan  2007   31889923    43.8
3 Brazil       1997  168546719    69.4
4 Brazil       2007  190010647    72.4
5 China        1997 1230075000    70.4
6 China        2007 1318683096    73.0
```

Funcția `pivot_woder()` primește următoarele argumente de bază:

-primul argument este setul de date

- al doilea argument este `names_from` care specifică numele variabilei din setul de date original care corespunde la numele variabilelor din setul de date transformat
- al treilea argument este `values_from` și specifică numele variabilei din setul de date original unde se regăsesc valorile corespunzătoare variabilelor din setul de date transformat

Pentru mai multe opțiuni și exemple de utilizare ale funcției `pivot_wider()` se poate consulta documentația `vignette("pivot")`.

3.9 Combinarea mai multor seturi de date

De cele mai multe ori analistul/statisticianul are de-a face cu mai multe seturi de date pentru a efectua analiza de interes și în această situație este important să dispună de instrumente care să-i permită să le combine într-un mod cât mai facil. În această secțiune vom prezenta o serie de funcții, disponibile în pachetul `dplyr`, care permit unirea a două seturi de date (`data.frame`) prin combinarea variabilelor din acestea. Interpretând că primul set de date este tabelul din stânga (`x`) iar cel de-al doilea, cel a cărui coloane vrem să le adăugăm primului, este cel din dreapta (`y`), avem mai multe posibilități de a le combina: `inner_join`, `left_join`, `right_join`, `full_join`, `anti_join` și `semi_join`. Pentru a ilustra cel șase tipuri de join vom considera următoarele două seturi de date (`df_x` în stânga și `df_y` în dreapta):

```
df_x = gapminder %>%
  select(country, year, pop) %>%
```

```
  filter(year %in% 1975:1990,
         country %in% c("Romania", "Belgium"))

df_y = gapminder %>%
  select(country, year, continent, lifeExp) %>%
  filter(year %in% 1985:1998,
         country %in% c("Romania", "Belgium", "France"))
```

country	year	pop
Belgium	1977	9821800
Belgium	1982	9856303
Belgium	1987	9870200
Romania	1977	21658597
Romania	1982	22356726
Romania	1987	22686371

country	year	continent	lifeExp
Belgium	1987	Europe	75.35
Belgium	1992	Europe	76.46
Belgium	1997	Europe	77.53
France	1987	Europe	76.34
France	1992	Europe	77.46
France	1997	Europe	78.64
Romania	1987	Europe	69.53
Romania	1992	Europe	69.36
Romania	1997	Europe	69.72

Avem:

a) `inner_join` - permite obținerea unui tabel a cărui linii sunt cele din `df_x` care au un corespondent în `df_y` și păstrează toate coloanele din `df_x` și `df_y`

country	year	pop	continent	lifeExp
Belgium	1987	9870200	Europe	75.35
Romania	1987	22686371	Europe	69.53

Observăm că unirea celor două seturi de date s-a făcut după potrivirea valorilor din variabilele `country` și respectiv `year`. Comportamentul de default al funcției `inner_join` (dar și a celorlalte) este de a face potrivirea după toate variabilele disponibile, i.e. cele care se regăsesc în ambele tabele. Dacă dorim să face potrivirea după o variabilă anume atunci putem specifica numele acesteia atributului `by`:

```
df_x %>%
  inner_join(df_y, by = "country")
# A tibble: 18 x 6
   country year.x      pop year.y continent lifeExp
   <chr>    <dbl>    <dbl>  <dbl> <chr>       <dbl>
 1 Belgium   1977  9821800   1987 Europe       75.4
 2 Belgium   1977  9821800   1992 Europe       76.5
 3 Belgium   1977  9821800   1997 Europe       77.5
 4 Belgium   1982  9856303   1987 Europe       75.4
 5 Belgium   1982  9856303   1992 Europe       76.5
 6 Belgium   1982  9856303   1997 Europe       77.5
 7 Belgium   1987  9870200   1987 Europe       75.4
 8 Belgium   1987  9870200   1992 Europe       76.5
 9 Belgium   1987  9870200   1997 Europe       77.5
10 Romania   1977 21658597   1987 Europe       69.5
11 Romania   1977 21658597   1992 Europe       69.4
12 Romania   1977 21658597   1997 Europe       69.7
```

```
13 Romania   1982 22356726   1987 Europe       69.5
14 Romania   1982 22356726   1992 Europe       69.4
15 Romania   1982 22356726   1997 Europe       69.7
16 Romania   1987 22686371   1987 Europe       69.5
17 Romania   1987 22686371   1992 Europe       69.4
18 Romania   1987 22686371   1997 Europe       69.7
```

În cazul în care variabila/variabilele de potrivire nu poartă același nume, atunci se poate specifica atributului `by` numele corespondenței `by = c("nume_x" = "nume_y")`.

b) `left_join` - permite obținerea unui tabel care va conține toate liniile din `df_x` și toate coloanele din `df_x` și `df_y` iar în cazul în care nu există un corespondent al liniilor primului tabel în cel de-al doilea va întoarce `NA`.

country	year	pop	continent	lifeExp
Belgium	1977	9821800	NA	NA
Belgium	1982	9856303	NA	NA
Belgium	1987	9870200	Europe	75.35
Romania	1977	21658597	NA	NA
Romania	1982	22356726	NA	NA
Romania	1987	22686371	Europe	69.53

Observăm că setul de date nou obținut conține toate liniile primului table și coloanele amandurora iar în dreptul valorilor variabilelor din `df_y` ce corespund liniilor din `df_x` care nu se potrivesc în `df_y` avem trecută valoarea `NA`.

Funcția `right_join` se comportă în mod similar cu `left_join` inversând locurile seturilor de date.

country	year	pop	continent	lifeExp
Belgium	1987	9870200	Europe	75.35
Belgium	1992	NA	Europe	76.46
Belgium	1997	NA	Europe	77.53
France	1987	NA	Europe	76.34
France	1992	NA	Europe	77.46
France	1997	NA	Europe	78.64
Romania	1987	22686371	Europe	69.53
Romania	1992	NA	Europe	69.36
Romania	1997	NA	Europe	69.72

c) `full_join` - permite obținerea unui tabel în care apar toate liniile și coloanele din ambele seturi de date iar pe pozițiite unde nu există corespondență între cele două se pune automat valoarea `NA`.

country	year	pop	continent	lifeExp
Belgium	1977	9821800	NA	NA
Belgium	1982	9856303	NA	NA
Belgium	1987	9870200	Europe	75.35
Romania	1977	21658597	NA	NA
Romania	1982	22356726	NA	NA
Romania	1987	22686371	Europe	69.53
Belgium	1992	NA	Europe	76.46
Belgium	1997	NA	Europe	77.53
France	1987	NA	Europe	76.34
France	1992	NA	Europe	77.46
France	1997	NA	Europe	78.64
Romania	1992	NA	Europe	69.36
Romania	1997	NA	Europe	69.72

d) `anti_join` - permite obținerea unui tabel în care se regăsesc doar coloanele din primul set de date și doar acele linii ale acestuia care nu au corespondent în cel de-al doilea set de date

country	year	pop
Belgium	1977	9821800
Belgium	1982	9856303
Romania	1977	21658597
Romania	1982	22356726

e) `semi_join` - permite obținerea unui tabel în care se regăsesc doar coloanele din primul set de date și doar acele linii ale acestuia care au corespondent în cel de-al doilea set de date

country	year	pop
Belgium	1987	9870200
Romania	1987	22686371

Pachetul `dplyr` pune la dispoziție și o serie de funcții care permit efectuarea de operații cu mulțimi. Aceste funcții primesc ca argumente două `data.frame`-uri care au aceleași coloane (aceleași variabile) și consideră observațiile ca elemente ale unei mulțimi:

- `intersect(df_x, df_y)` - întoarce un tabel doar cu observațiile comune din cele două seturi de date
- `union(df_x, df_y)` - întoarce un tabel cu observațiile unice din cele două seturi de date
- `setdiff(df_x, df_y)` - întoarce un tabel cu observațiile din primul set de date care nu se regăsesc în al doilea set de date

Pentru mai multe informații și exemple referitoare la modurile în care se pot combina două seturi de date se poate consulta (Wickham et al. 2019; Wickham and Grolemund 2017).

Referințe

Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software, Articles* 59 (10): 1–23. https://doi.org/10.18637/jss.v059.i10.

Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and R Studio. 2019. "Dplyr: A Grammar of Data Manipulation Ver. 0.8.3."

Wickham, Hadley, and Garrett Grolemund. 2017. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. 1st ed. O'Reilly Media, Inc.

Wickham, Hadley, and Lionel Henry. 2020. *Tidyr: Tidy Messy Data*. https://CRAN.R-project.org/package=tidyr.