Manipularea și vizualizarea datelor cu ajutorul pachetului Tidyverse

Elemente de analiză exploratorie a datelor

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

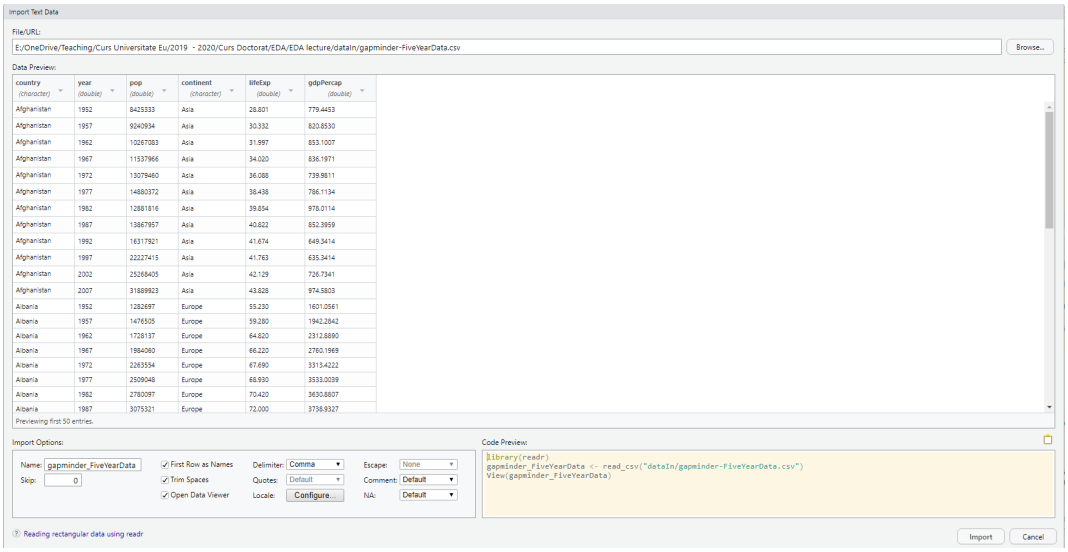


Fig. 1: Interfata de importare a datelor din RStudio

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

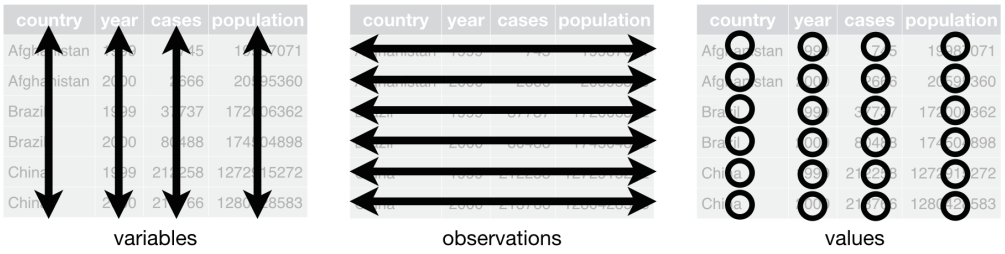


Fig. 2: Ilustrarea principiului de date tidy conform Wickham

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).

4 Vizualizarea datelor cu `ggplot2`

Tehnicile de vizualizarea a datelor joacă un rol important în orice analiză statistică, putând conduce la identificarea de noi caracteristici, perspective sau pattern-uri în acestea. Scopul acestei secțiuni este de a introduce o serie de elemente de vizualizare a datelor prin intermediul pachetului ggplot2. Pachetul a fost

dezvoltat inițial de Hadley Wickham (care încă menține și îmbunătățește activ pachetul), este parte integrantă din suita tidyverse și este bazat pe, ceea ce se numește în teoria vizualizării, *gramatica elemntelor grafice* introdusă de Leland Wilkinson în cartea The Grammar of graphics (de aici vine și prefixul `gg` - grammar of graphics). Similar cu gramatica unei limbi, în care diferite elemente precum substantive, verbe, articole, prepoziții, etc. se pot combina după anumite reguli și *gramatica elementelor grafice* prezintă o serie de reguli care să permită construcția de grafice statistice prin combinarea mai multor tipuri de *straturi* (*layers*) (Wilkinson 2005). Astfel, ideea principală a pachetului `ggplot2` este de a specifica independent o serie de componente constructive ale graficului (obiecte geometrice), organizate în straturi, ce urmează a le combina pentru a genera figura dorită (Wickham 2016).

Ne dorim ca la finalul acestei secțiuni, să fim capabili să creăm grafice care să semene cu cele de mai jos:

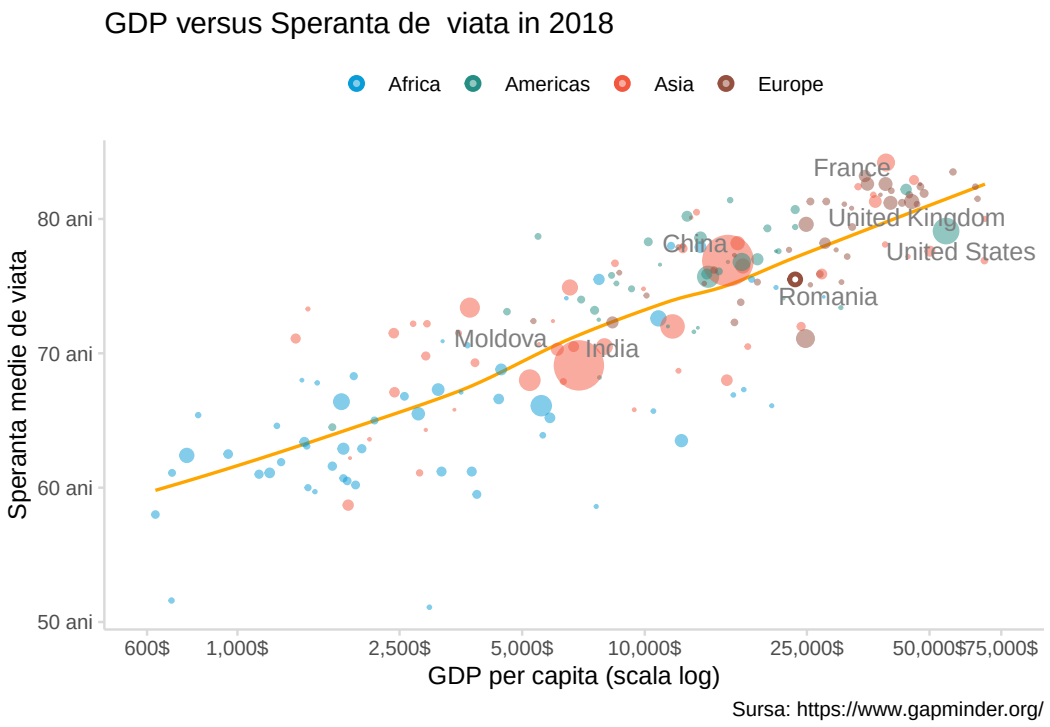


Fig. 3: Exemplu de vizualizare realizata cu ajutorul pachetului ggplot2

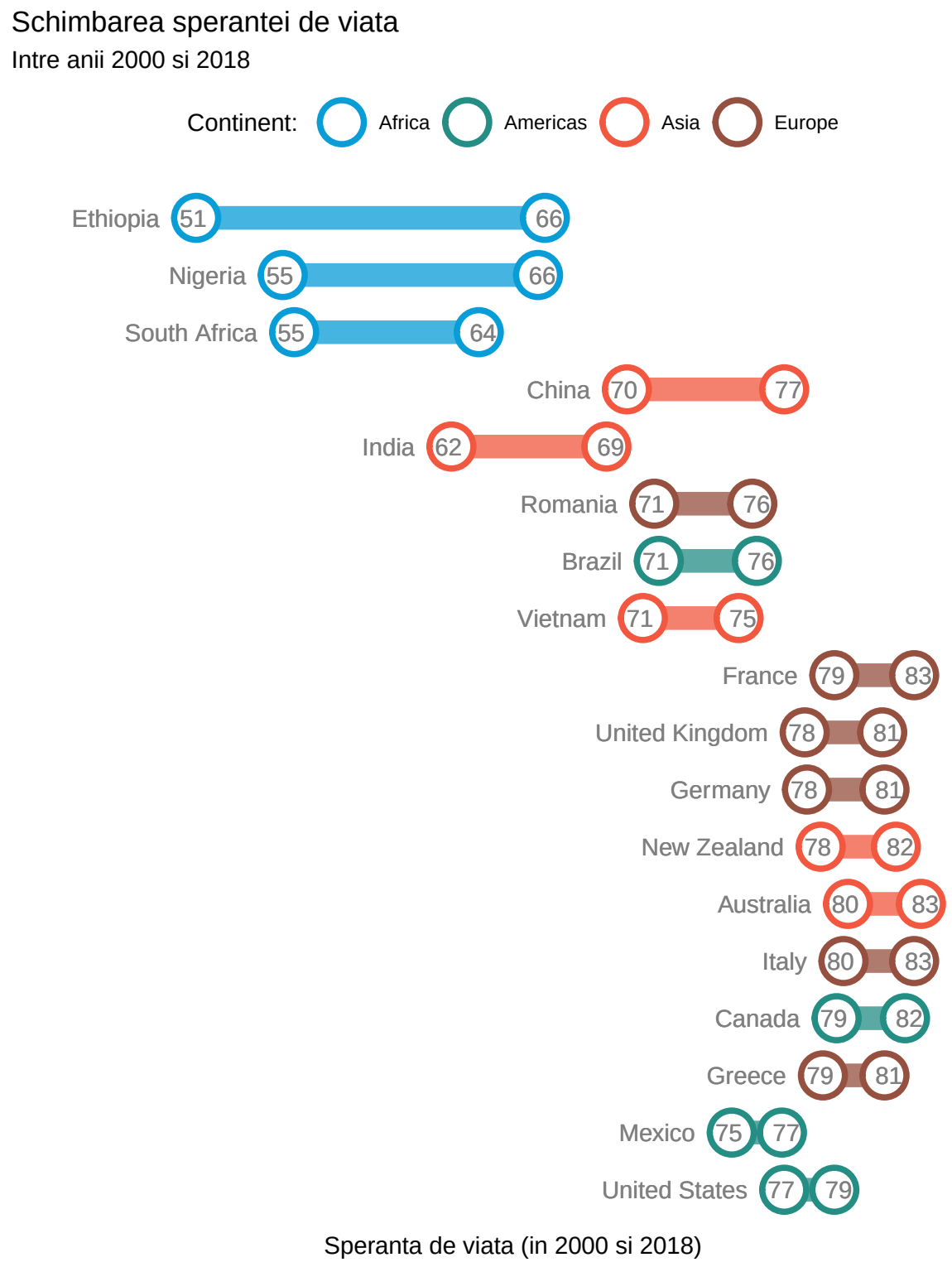


Fig. 4: Exemplul 2 de vizualizare realizata cu ajutorul pachetului ggplot2

Înainte de prezentarea elementelor de bază ce compun un grafic folosind pachetul `ggplot2` și de a descrie fiecare componentă în parte, vom specifica care sunt datele (setul/seturile de date) pe care le vom utiliza pe

parcursul acestui capitol.

4.1 Setul de date

Setul de date pe care îl vom folosi pentru a exemplifica elementele grafice introduse în acest capitol este creat prin manipularea unor date în format brut descărcate de pe platforma www.gapminder.org/data/. Mai precis vom descărca în format `csv` datele referitoare la durata medie de viață (sau speranța de viață) (*Health -> Life expectancy* sau http://gapm.io/ilex), produsul intern brut pe cap de locuitor ajustat la rata inflației (*Economy -> Incomes & growth -> Income* sau http://gapm.io/dgdppc) și respectiv populația totală (*Population -> Population* sau http://gapm.io/dpop) a peste 180 de state pe perioada 1800 - 2018. De asemenea, pentru a adăuga informații referitoare la locația geografică a țărilor din setul de date, respectiv continentul pe care se află fiecare țară, vom descărca fișierul excel de pe pagina https://www.gapminder.org/fw/four-regions/. Codul pentru crearea setului de date `gapminder_all` este ilustrat mai jos (acesta seamănă cu setul de date utilizat în capitolul de manipulare a datelor):

```
# citim seturile de date
life_exp_gap = read_csv("dataIn/life_expectancy_years.csv")
tot_pop_gap = read_csv("dataIn/population_total.csv")
gdp_gap = read_csv("dataIn/income_per_person_gdppercapita_ppp_inflation_adjusted.csv")
countries_gap = read_excel("dataIn/Data Geographies - v1 - by Gapminder.xlsx", sheet = 2)

# transformam seturile de date
life_exp_gap_pivot = life_exp_gap %>%
  pivot_longer(cols = -country, names_to = "year", values_to = "lifeExp")

tot_pop_gap_pivot =tot_pop_gap %>%
  pivot_longer(cols = -country, names_to = "year", values_to = "pop")

gdp_gap_pivot = gdp_gap %>%
  pivot_longer(cols = -country, names_to = "year", values_to = "gdpPerCap")

countries_gap_keep = countries_gap %>%
  select(geo, name, four_regions,
         six_regions, "World bank region",
         "World bank, 4 income groups 2017") %>%
  rename(wb_region = "World bank region",
         wb_income = "World bank, 4 income groups 2017",
         country = "name")

# unim seturile de date
gapminder_all <- life_exp_gap_pivot %>%
  left_join(tot_pop_gap_pivot) %>%
  left_join(gdp_gap_pivot) %>%
  left_join(countries_gap_keep) %>%
  # capitalizarea primei litere, i.e. transformarea: asia -> Asia
  mutate(four_regions = str_to_title(four_regions))
```

Pentru a avea o imagine de ansamblu asupra setului de date construit vom folosi funcția `glimpse`:

```
glimpse(gapminder_all)
Observations: 40,953
Variables: 10
$ country      <chr> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanis...
$ year         <chr> "1800", "1801", "1802", "1803", "1804", "1805", "1806"...
```

```
$ lifeExp      <dbl> 28.2, 28.2, 28.2, 28.2, 28.2, 28.2, 28.1, 28.1, 28.1, ...
$ pop          <dbl> 3280000, 3280000, 3280000, 3280000, 3280000, 3280000, ...
$ gdpPerCap    <dbl> 603, 603, 603, 603, 603, 603, 603, 603, 603, 604,...
$ geo          <chr> "afg", "afg", "afg", "afg", "afg", "afg", "afg", "afg"...
$ four_regions <chr> "Asia", "Asia", "Asia", "Asia", "Asia", "Asia", "Asia"...
$ six_regions  <chr> "south_asia", "south_asia", "south_asia", "south_asia"...
$ wb_region    <chr> "South Asia", "South Asia", "South Asia", "South Asia"...
$ wb_income    <chr> "Low income", "Low income", "Low income", "Low income"...
```

În multe din exemplele din această secțiune vom folosi doar datele corespunzătoare anului 2018 prin urmare vom crea și acest set de date și-l vom numi `gapminder_2018`:

```
# setul de date pentru anul 2018
gapminder_2018 = gapminder_all%>%
  filter(year == 2018) %>%
  select(-geo) %>%
  rename(continent = four_regions)
```

Putem sumariza variabilele din setul de date `gapminder_2018` pentru a înțelege mai bine tipul lor și a avea mai multe informații referitoare la statisticile elementare (media, mediana, valoarea minimă și respectiv maximă, etc.) asociate acestora:

```
summary(gapminder_2018)
   country              year             lifeExp           pop
 Length:187         Length:187         Min.   :51.10   Min.   :5.320e+04
 Class :character   Class :character   1st Qu.:67.10   1st Qu.:2.460e+06
 Mode  :character   Mode  :character   Median :74.05   Median :9.420e+06
                                       Mean   :72.66   Mean   :4.062e+07
                                       3rd Qu.:78.03   3rd Qu.:3.005e+07
                                       Max.   :84.20   Max.   :1.420e+09
                                       NA's   :3
   gdpPerCap       continent         six_regions         wb_region
 Min.   :   629   Length:187         Length:187         Length:187
 1st Qu.:  3605   Class :character   Class :character   Class :character
 Median : 11700   Mode  :character   Mode  :character   Mode  :character
 Mean   : 17984
 3rd Qu.: 25200
 Max.   :121000

  wb_income
 Length:187
 Class :character
 Mode  :character
```

Observăm că pentru variabila `lifeExp` avem trei observații care nu prezintă valori, prin urmare putem exclude acele observații din setul de date:

```
gapminder_2018 = gapminder_2018 %>%
  filter(!is.na(lifeExp))
```

4.2 Elemente de bază

Graficele (figurile) realizate prin intermediul pachetului `ggplot2` au la bază o serie de elementele constructive, printe care putem enumera:

- setul de date ce urmează să fie prezentat grafic (**data**) - acesta *trebuie* să fie sub forma unui `data.frame` (adus la un format *tidy*)
- obiectele geometrice (**geom**) ce urmează să apară pe grafic (puncte, linii, etc.)
- o serie de corespondențe (**mappings**) de la variabilele din setul de date la aspectul (estetica - **aesthetics**) obiectelor geometrice
- transformări statistice (**statistical transformations**) folosite pentru a calcula valorile datelor ce sunt utilizate în grafic
- ajustări ale pozițiilor (**position adjustments**) obiectelor geometrice pe grafic
- scalări și scale (**scales**) pentru fiecare corespondență estetică folosită
- sisteme de coordonate (**coordinate systems**)
- fațete (**facets**) sau grupuri de date folosite în figuri diferite

Aceste componente constructive ale unui grafic sunt organizate în straturi (*layers*), unde fiecare strat conține un singur obiect grafic, transformare statistică sau ajustare de poziție și astfel putem vizualiza o figură/un grafic ca pe o mulțime de straturi de imagini suprapuse, fiecare prezentând un anumit aspect al datelor.

```
Error in knitr::include_graphics("images/layers2.png"): Cannot find the file(s): "images/layers2.png"
```

Primul pas în crearea unui grafic folosind pachetul `ggplot2` este de a construi un obiect *ggplot*. Acest obiect, creat prin intermediul funcției `ggplot()`, inițializează suprafața de desen (*canvas*) fără a desena nimic pe ea. Prin apelarea acestei funcții se specifică, în general, atât setul de date (în format `data.frame`) care va fi folosit în ilustrația grafică precum și proprietățile estetice (ce țin de aspect) care vor fi aplicate (vor corespunde) obiectelor geometrice folosite ulterior. Cu alte cuvinte, vom spune funcției `ggplot` care sunt datele pe care urmărim să le ilustrăm grafic și cum fiecare varaibilă din acest set de date va fi folosită (e.g. ca și coordonate x, y sau ca variabile de colorare - *colour* ori mărime - *size*, etc.). Este important de specificat că setul de date **trebuie** să fie sub forma unui `data.frame` adus la un format *tidy*. Codul generic este:

```
# il atribuim unui obiect
obiect = ggplot(data = <DATAFRAME>, aes(x = <COLOANA_1>, y = <COLOANA_2>))

# nu il atribuim unui obiect
ggplot(data = <DATAFRAME>, aes(x = <COLOANA_1>, y = <COLOANA_2>))
```

Sumarizând avem următorii pași ilustrați în figura de mai jos:

1. Apelăm funcția `ggplot()` care ne va crea o suprafață de desen goală
2. Specificăm corespondențele estetice (*aesthetic mappings*) dintre variabile setului de date și elemente ce țin de aspect. În acest caz, pentru setul de date `gapminder_2018`, între variabilele `gdpPerCap` și `lifeExp` și axele x și y.
3. Adăugăm obiecte geometrice care vor apărea pe grafic. În cazul nostru vom folosi stratul `geom_point()` pentru a adăuga puncte ca elemente grafice care să reprezinte datele.

```
# crearea suprafetei de desenare
ggplot(gapminder_2018)

# variabilele de interes sunt afisate
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp))

# datele desenate
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point()
```

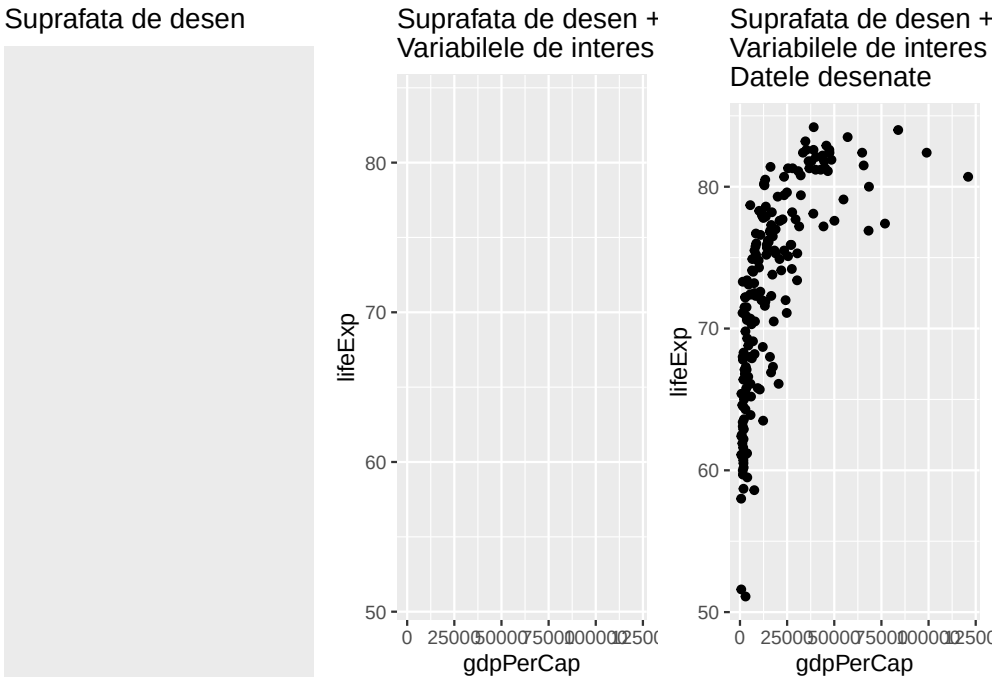


Fig. 5: Ilustrare a temelor predefinite

Este de remarcat faptul că atunci când am folosit stratul *geom* am utilizat operatorul `+`. De fiecare dată când vom adăuga un nou strat grafic la figura noastră o vom face prin intermediul operatorului `+`. Funcția `geom_point()` este doar o scriere prescurtată de trasare a graficului, în realitate se apelează funcția `layer()` care în fapt construiește un nou strat în care se specifică una din cele cinci componente principale `mapping`, `data`, `geom`, `stat` și `position`:

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  layer(
    mapping = NULL,
    data = NULL,
    geom = "point",
    stat = "identity",
    position = "identity"
    )
```

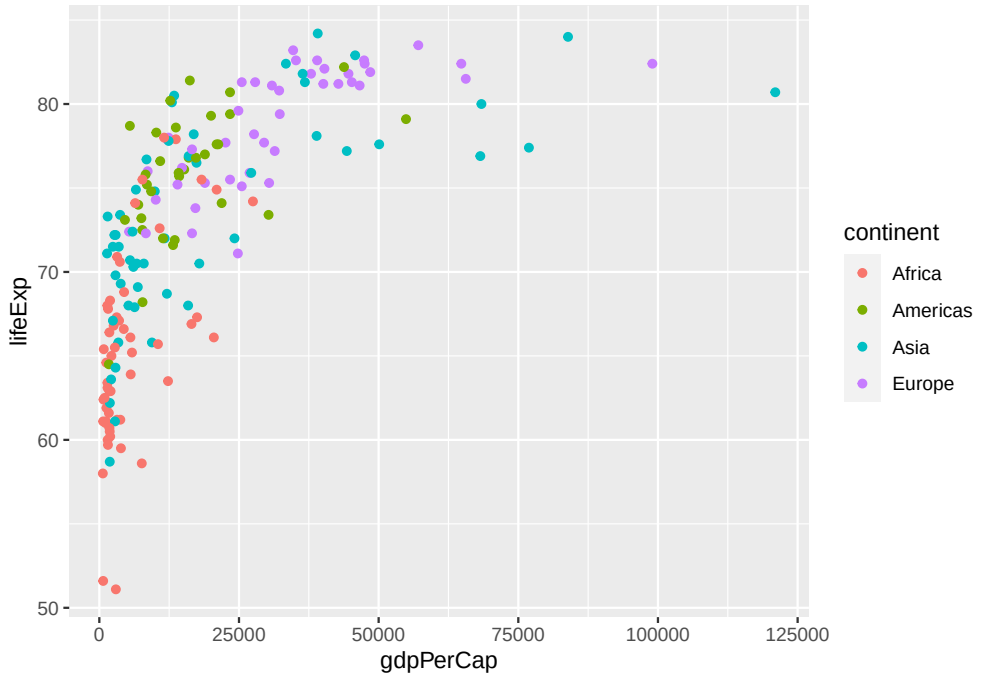
Structura generică a unui grafic realizat în `ggplot2` este:

```
ggplot(data = [DATA]) +         # element obligatoriu
 [GEOM_FUNCTION](
    mapping = aes([MAPPINGS]),  # element obligatoriu
    stat = [STAT],              # element optional
    position = [POSITION]       # element optional
  ) +
 [COORDINATE_FUNCTION] +        # element optional
 [FACET_FUNCTION] +             # element optional
 [SCALE_FUNCTION] +             # element optional
 [THEME_FUNCTION]               # element optional
```

4.3 Corespondențe estetice (*aesthetic mappings*)

Corespondențele estetice, de aspect, (aesthetic mappings) preiau proprietăți ale datelor și le folosesc pentru a influența o serie de caracteristici vizuale ale figurii, precum poziția, culoarea, mărimea, forma sau transparența. Astfel fiecare caracteristică vizuală poate codifica un aspect al datelor și, prin urmare, poate fi utilizată pentru a transmite noi informații. Toate elementele estetice se specifică folosind funcția `aes()` atât în interiorul funcției `ggplot` unde dobândesc un caracter global cât și în interiorul fiecărui strat definit de un obiect geometric (*geom*). De exemplu, în figura de mai jos, pentru setul de date `gapminder_2018`, culoarea (definită prin estetica `colour =`) descrie (corespunde la) apartenența la continent, poziția x arată produsul intern brut al statelor (`gdpPerCap`) iar poziția y arată speranța medie de viață (`lifeExp`).

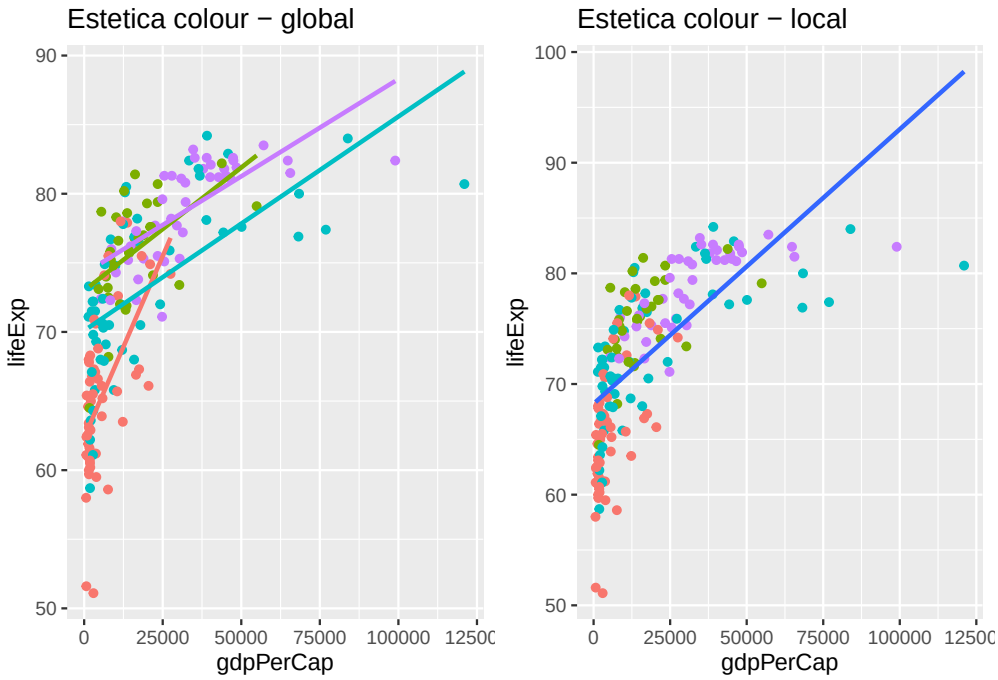
```
# caracter global pentru colour
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent)) +
  geom_point()

# sau local
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(aes(colour = continent))
```



Diferența dintre caracterul global și cel local al elementelor estetice (atunci când sunt aplicate funcției `ggplot` sau a straturilor individuale) se poate observa în figura de mai jos, unde adăugăm la grafic dreptele de regresie corespunzătoare țărilor de pe fiecare continent. Astfel, în cazul global, dreptele de regresie sunt asociate pentru fiecare subset de date (determinate de variabila `continent` prin intermediul esteticii de culoare) pe când atunci când elementul estetic este specificat în cadrul stratului geometric local (`geom_point()`)) se asociază dreapta de regresie asociată întregului set de date.

```
# global
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)

# local
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
```

```
  geom_point(aes(colour = continent)) +
  geom_smooth(method = "lm", se = FALSE)
```



Elementele estetice ce pot fi utilizate pentru trasarea unei figuri depind de tipul de obiect/strat geometric (*geom*) folosit. Pentru mai multe informații se poate consulta documentația funcțiilor `geom` la secțiunea *Aesthetics* (e.g. `?geom.point`) unde elementele obligatorii apar îngroșate iar cele opționale apar normal.

Printre cele mai uzuale elemente estitice ale unui grafic enumerăm:

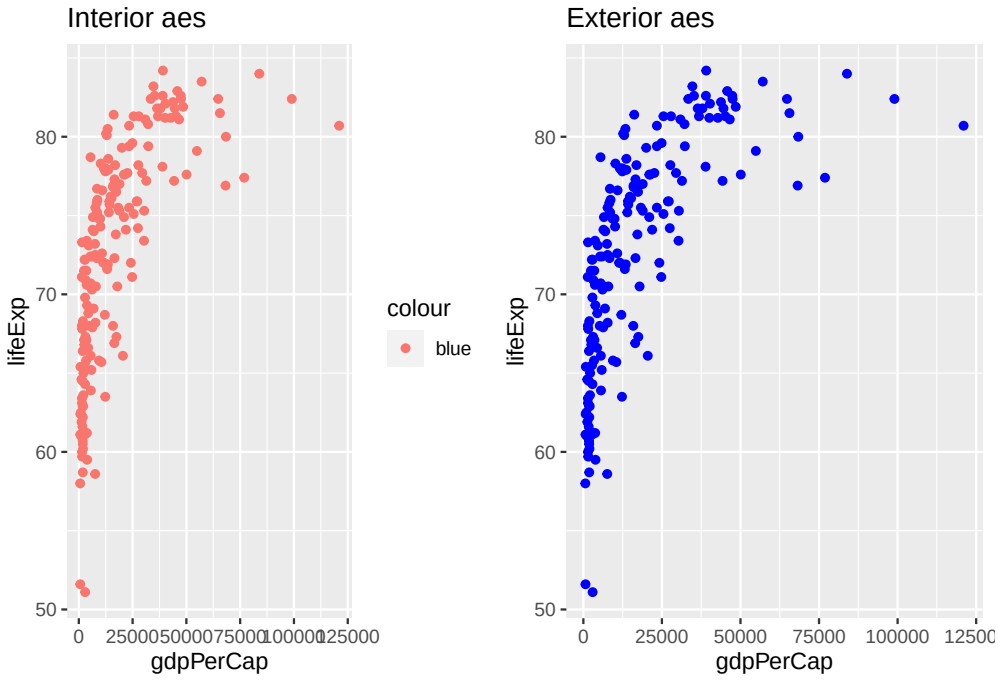
Element estetic (aesthetic)	Descriere
`x`	Poziția pe axa absciselor (x-axis)
`y`	Poziția pe axa ordonatelor (y-axis)
`shape`	Forma punctelor
`color/colour`	Culoarea marginală a elementelor
`fill`	Culoarea de umplere, interioară, a elementelor
`size`	Mărimea
`alpha`	Tranparență (1 - opac, 0 - transparent)
`linetype`	Tipul liniei (e.g. solidă, punctată, etc.)

O prezentare generală și mai amănunțită a acestora se regăsește în vigneta pachetului `ggplot2` intitulată `Aesthetic specifications` care poate fi apelată prin

```
vignette("ggplot2-specs")
```

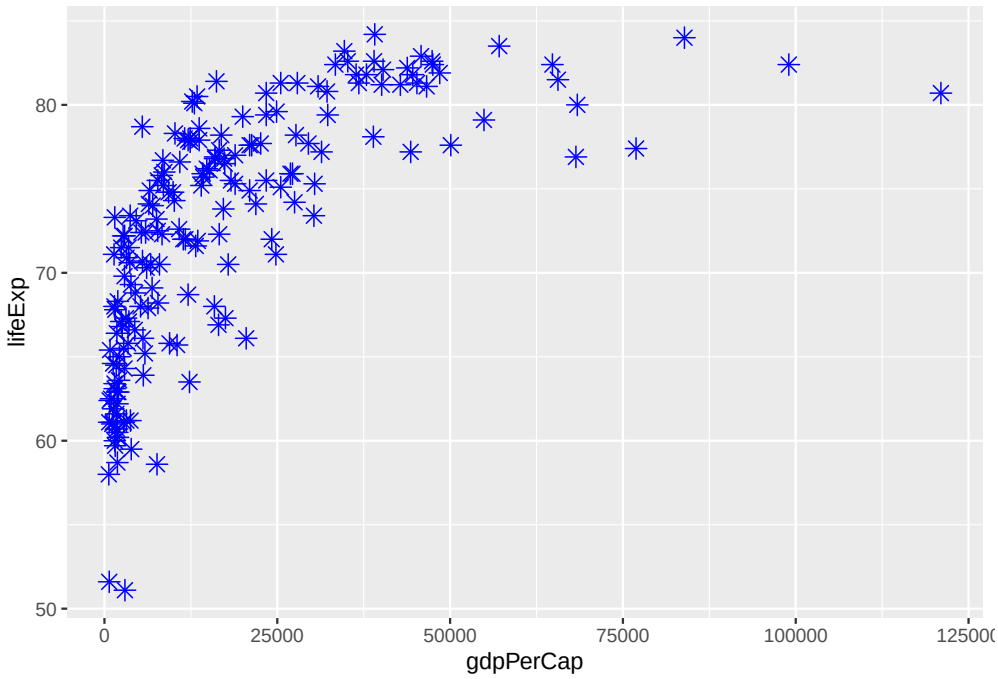
Trebuie remarcat faptul că prin utilizarea funcției `aes()`, canalul vizual va fi determinat să se bazeze pe datele specificate în argument. De exemplu, atunci când folosim comanda `aes(colour = "blue")` nu va face ca obiectul geometric (în cazul de mai sus punctele) să aibă culoarea albastră, ci va face maparea ca și cum am avea un singur tip numit "albastru". În cazul în care dorim să aplicăm valoarea elementului estetic întregului obiect geometric, de exemplu să schimbăm culoarea punctelor în albastru, atunci trebuie să specificăm elementul estetic în afara funcției `aes()`:

```
# interior
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(aes(colour = "blue"))
# exterior
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(colour = "blue")
```



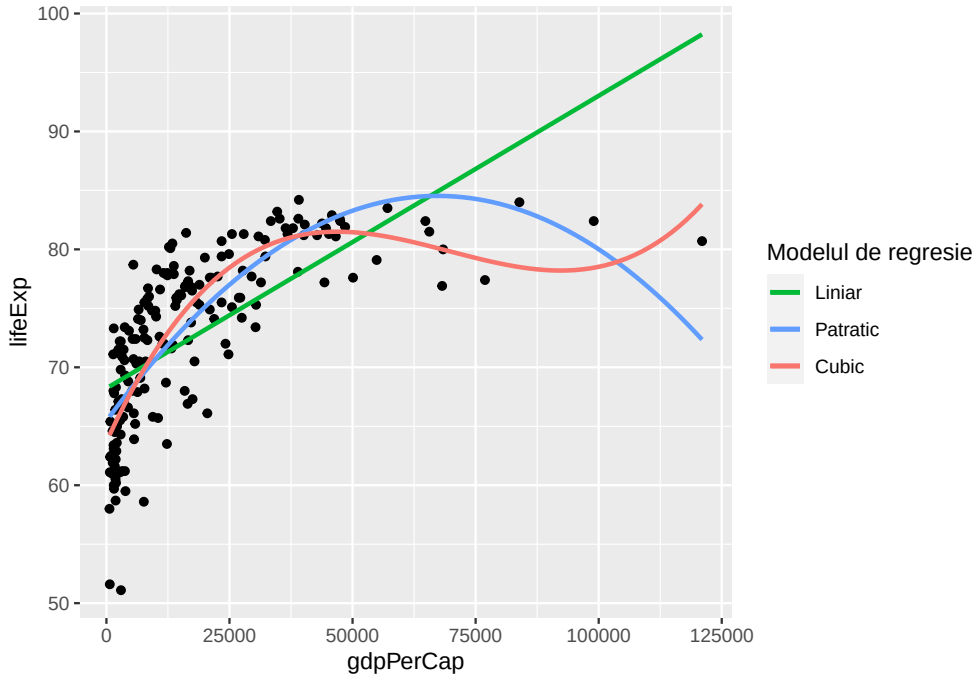
Putem face acest lucru pentru fiecare caracteristică estetică, incluzând `colour`, `fill`, `shape` sau `size`. În exemplul de mai jos schimbăm culoarea, mărimea și forma punctelor:

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(colour = "blue", size = 3, shape = 8)
```



Atribuirea unei valori fixate (unei constante) unei estetici în interiorul funcției `aes()` poate fi utilă în special atunci când dorim să ilustrăm mai multe straturi cu diverși parametrii și în care dorim să numim acești parametrii. Spre exemplu să presupunem că dorim să ajustăm mai multe modele de regresie la setul de date `gapminder_2018` și să scoatem în evidență curbele generate.

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point() +
  geom_smooth(method = "lm",
              se = FALSE,
              aes(color = "Liniar") ) +
  geom_smooth(method = "lm",
              formula = y ~ poly(x, 2),
              se = FALSE,
              aes(color = "Patratic") ) +
  geom_smooth(method = "lm",
              formula = y ~ poly(x, 3),
              se = FALSE,
              aes(color = "Cubic") )
```

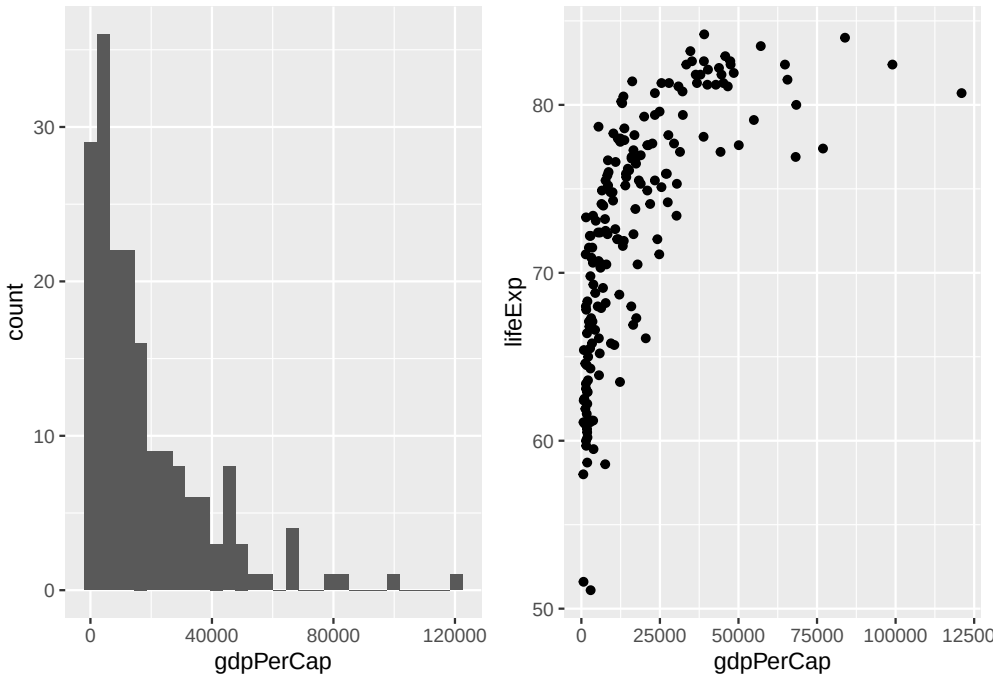


4.4 Obiecte geometrice (*geometric objects*)

Obiectele geometrice (*geometric objects*), sau `geom` pe scurt, sunt elementele fundamentale ale unui grafic creat în `ggplot2`, prin intermediul lor se efectuează reprezentarea a datelor, controlând tipul de grafic dorit. Neincluderea unui strat `geom` în desen conduce la o figură goală (a se vedea mai sus). Majoritatea obiectelor geometrice sunt asociate unui grafic a cărui denumire este prestabilită, astfel `geom_bar` corespunde unei diagrame cu bare (*barplot*), `geom_line` corespunde unei diagrame liniare (*linegraph*), `geom_histogram` corespunde unei histograme, `geom_boxplot` corespunde unei diagrame de tip boxplot (*cutie cu mustăți*) ș.a.m.d. O excepție de la această regulă este dată de diagrama de împrăștiere (*scatterplot*) pentru care se folosește `geom_point`.

Fiecare funcție `geom` are propriile elemente estetice și argumente necesare pentru a ajusta modul în care este creat graficul. De exemplu, funcția `geom_histogram` necesită doar elementul estetic `x` pe când dacă dorim să trasăm o diagramă de împrăștiere avem nevoie atât de estetica `x` cât și de `y`. În contextul setului de date `gapminder_2018`, variabila `gdpPerCap` ne arată produsul intern brut pe cap de locuitor pentru fiecare țară iar variabila `lifeExp` prezintă speranța medie de viață din țara respectivă. Pentru a vedea cum este distribuit produsul intern brut pentru anul 2018 vom trasa o histogramă a acestuia iar pentru a investiga relația dintre variabilele `gdpPerCap` și `lifeExp` vom ilustra o diagramă de împrăștiere:

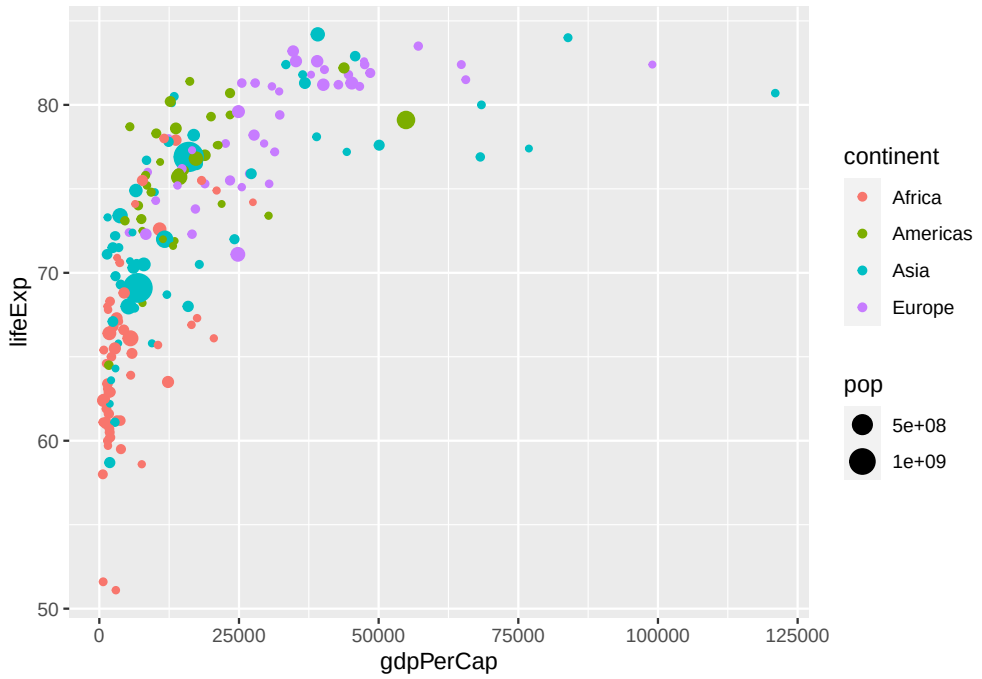
```
# histograma
ggplot(gapminder_2018, aes(x = gdpPerCap)) +
  geom_histogram()

# diagrama de imprastiere - scatterplot
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point()
```



Pentru fiecare funcție `geom` există elemente estetice obligatorii și opționale. Majoritatea obiectelor grafice necesită elementele estetice `x` și `y` dar sunt și excepții precum `geom_bar` sau `geom_histogram`. De exemplu, în cazul funcției `geom_point`, elementele estetice obligatorii sunt `x` și `y` iar cele opționale pot include: `alpha` (transparență), `colour`, `fill`, `size`, `shape` sau `stroke`. Trebuie menționat că elementele estetice pot să difere de la `geom` la `geom` în funcție de specificul vizual al fiecărui obiect grafic, de exemplu putem utiliza `shape` ca estetică pentru `geom_point` dar nu o putem utiliza și pentru `geom_line` precum putem utiliza `linetype` pentru `geom_line` dar nu și pentru `geom_point`.

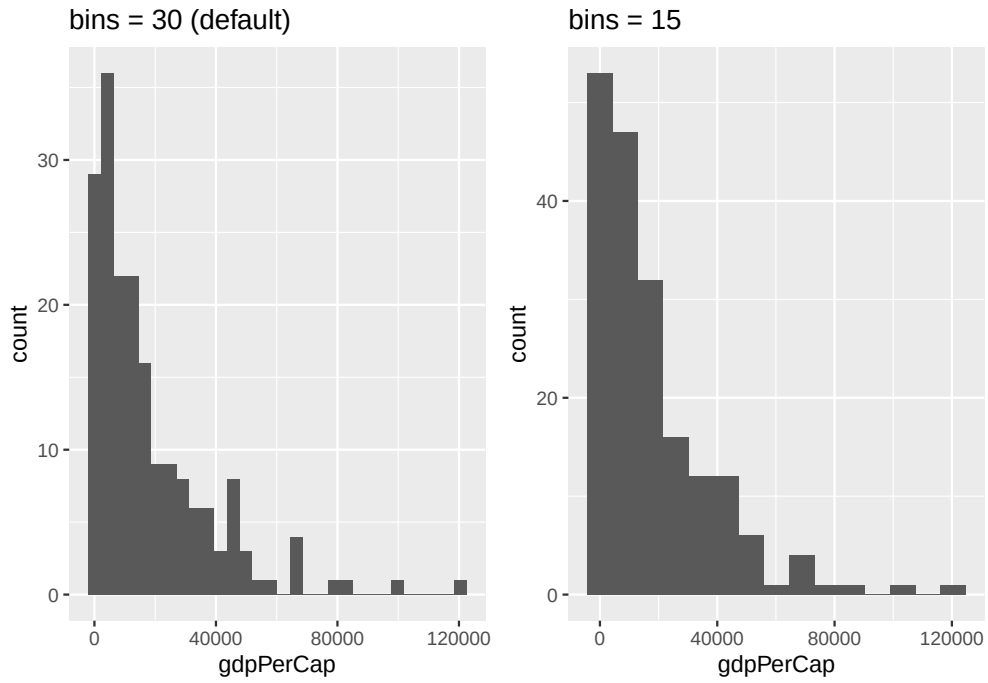
În contextul setului de date `gapminder_2018` vom utiliza atât estetica `colour` pentru a diferenția țările în funcție de `continent` cât și estetica `size` pentru a scoate în evidență țările cu o populație (`pop`) mai mare:

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent, size = pop)) +
  geom_point()
```



Pe lângă elementele estetice, funcțiile `geom` admit și o serie de argumente specifice (care se regăsesc la secțiunea *arguments* în documentația acestora, e.g. `?geom_point`), de exemplu dacă dorim să modificăm numărul de subintervale (*bins*) folosit pentru trasarea unei histograme atunci putem utiliza argumentul `bins =`:

```
# histograma
ggplot(gapminder_2018, aes(x = gdpPerCap)) +
  geom_histogram(bins = 15)
```



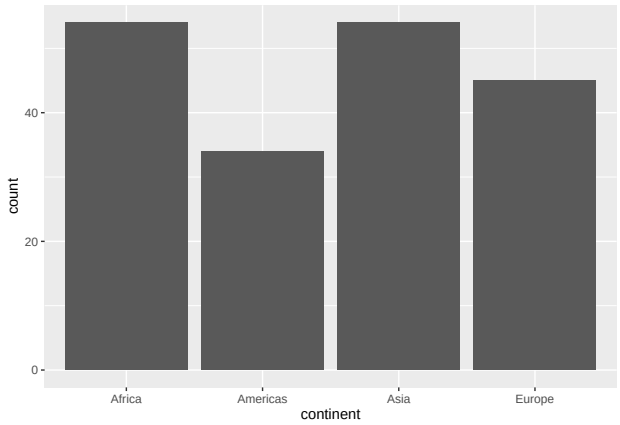
Următorul tabel prezintă o serie de funcții `geom` împreună cu elementele estetice obligatorii și o parte din argumentele specifice ale acestora.

Funcția	Elemente estetice	Argumente opționale	Descriere
`geom_point`	x, y		Trasează o diagramă de împrăștiere
`geom_line`	x, y	arrow, na.rm	Trasează o diagramă liniară
`geom_segment`	x, y, xend, yend	arrow, na.rm	Trasează un segment de dreaptă
`geom_path`	x, y	na.rm	Trasează o linie poligonală
`geom_polygon`	x, y		Trasează o linie poligonală închisă (umplută)
`geom_rect`	xmin, xmax, ymin, ymaxl		Trasează un dreptunghi
`geom_histogram`	x	bins, binwidth	Trasează o histogramă
`geom_density`	x		Trasează o diagramă de densitate estimată prin nuclee
`geom_dotplot`	x	bins, binwidth	Trasează o diagramă cu puncte similară diagramei cu bare
`geom_freqpoly`	x	binwidth	Trasează o linie poligonală de frecvență
`geom_bar`	x sau x, y	width	Trasează o diagramă cu bare
`geom_abline`	intercept, slope		Trasează drepte în care sunt specificate pantele și ordonatele la origine
`geom_hline`	yintercept		Trasează drepte de referință orizontale
`geom_vline`	xintercept		Trasează drepte de referință verticale
`geom_smooth`	x, y	method, se, span	Adaugă o curbă de regresie
`geom_text`	x, y, label		Adaugă etichete text la puncte
`geom_bin2d`	x, y	bins	Ilustrează o estimare a densității bivariate prin dreptunghiuri
`geom_hex`	x, y	bins	Ilustrează o estimare a densității bivariate prin hexagoane (faguri)

Vom prezenta mai jos o parte dintre graficele generate de aceste funcții în contextul seturilor de date `gapminder_all` și `gapminder_2018`. Vom face separarea graficelor după numărul (una sau două) și tipul (discrete sau continue) variabilelor analizate:

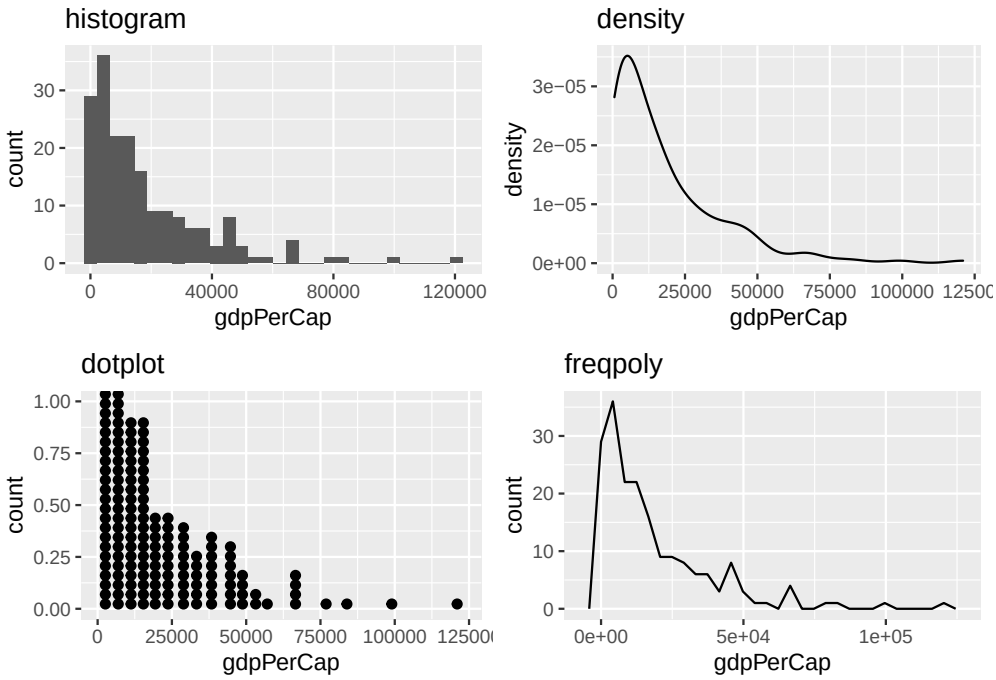
- o variabilă discretă - `geom_bar()`

```
ggplot(gapminder_2018, aes(x = continent)) +
  geom_bar()
```



- o variabilă continuă - `geom_histogram`, `geom_density`, `geom_dotplot` și `geom_freqpoly`

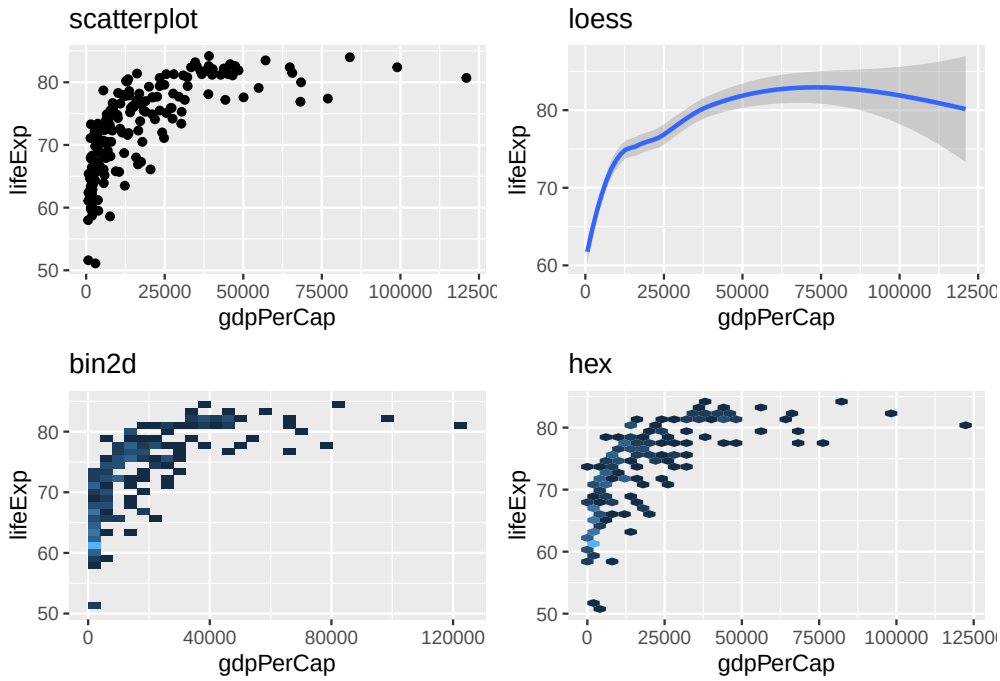
```
# histograma
ggplot(gapminder_2018, aes(x = gdpPerCap)) +
  geom_histogram()

# estimarea densitatii
ggplot(gapminder_2018, aes(x = gdpPerCap)) +
  geom_density()

# dotplot
ggplot(gapminder_2018, aes(x = gdpPerCap)) +
  geom_dotplot()

# diagrama de frecvente
ggplot(gapminder_2018, aes(x = gdpPerCap)) +
  geom_freqpoly()
```



- două variabile ambele continue - `geom_point`, `geom_line`, `geom_smooth`, `geom_bin2d`, `geom_hex`, etc.

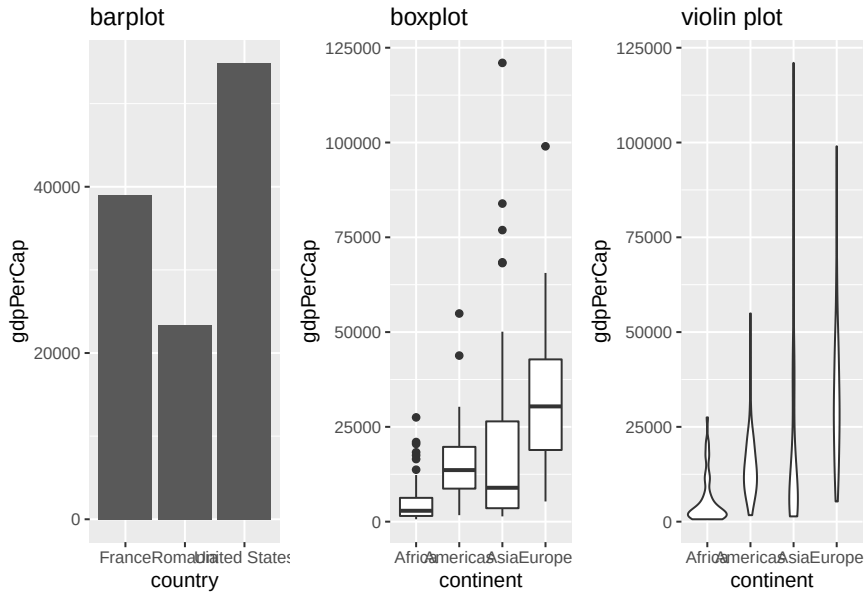
```
# Diagrama de imprastiere
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point()

# Functia de regresie - default loess
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_smooth()

# Densitate bivariata
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_bind2d()

# Densitate bivariata
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
```

```
  geom_hex()
```



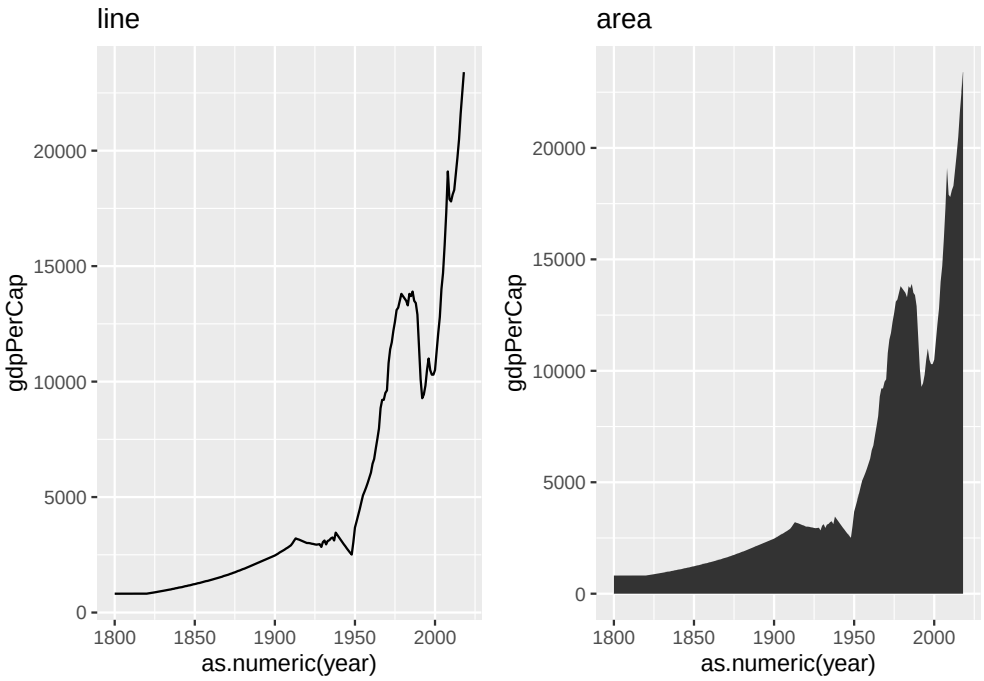
- o variabilă continuă și una discretă - `geom_bar(stat = "identity")` (sau `geom_col`), `geom_boxplot`, `geom_violin`, etc.

```
# datele
gap1 = gapminder_2018 %>%
  filter(country %in% c("Romania", "United States", "France"))

# barplot
ggplot(gap1, aes(x = country, y = gdpPerCap)) +
  geom_bar(stat = "identity")
# sau
ggplot(gap1, aes(x = country, y = gdpPerCap)) +
  geom_col()

# boxplot
ggplot(gapminder_2018, aes(x = continent, y = gdpPerCap)) +
  geom_boxplot()
```



- o variabilă continuă și alta de tip temporal: `geom_line`, `geom_area`

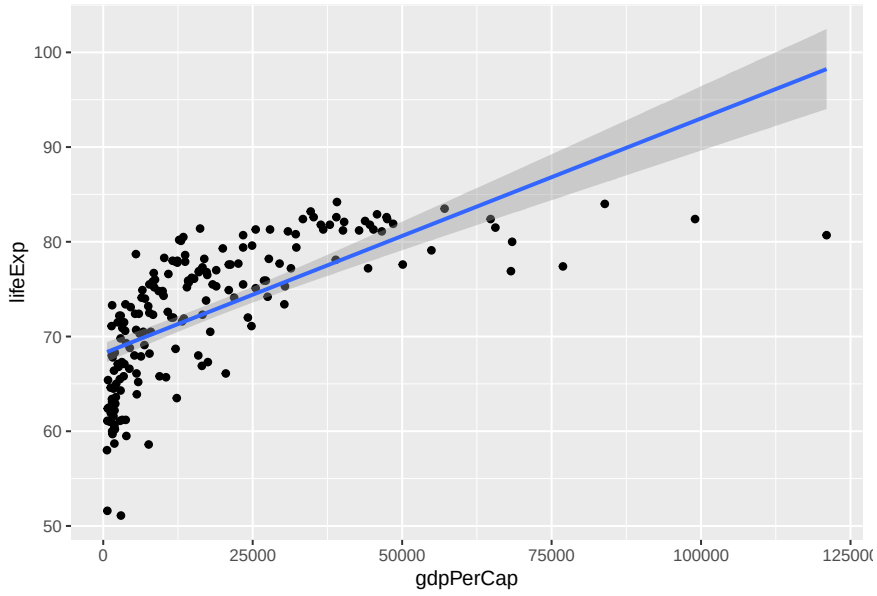
```
# diagrama liniara
ggplot(gapminder_all %>% filter(country == "Romania"),
       aes(x = as.numeric(year), y = gdpPerCap)) +
  geom_line()

# area plot
ggplot(gapminder_all %>% filter(country == "Romania"),
       aes(x = as.numeric(year), y = gdpPerCap)) +
  geom_line()
```



Ceea ce face pachetul `ggplot2` cu adevărat puternic este că acesta permite construcția de figuri complexe

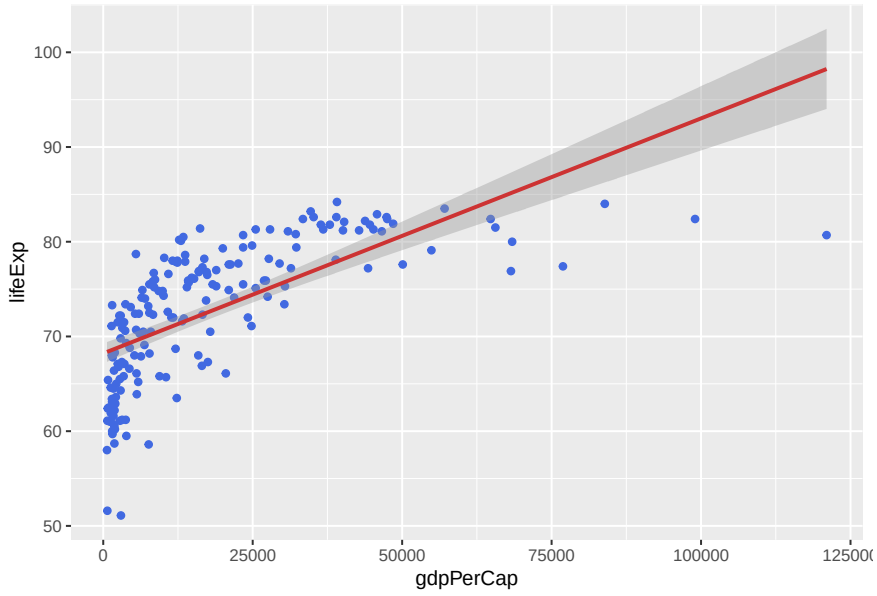
prin adăugarea mai multor obiecte grafice la aceeași figură, suprapunând straturile determinate de ele. Să considerăm setul de date `gapminder_2018` și să adăugăm la diagrama de împrăștiere a variabilelor `gdpPerCap` și `lifeExp` relația liniară determinată de dreapta de regresie:

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point() +
  geom_smooth(method = "lm")
```



Trebuie remarcat faptul că putem utiliza elemente estetice diferite pentru fiecare obiect geometric folosit, de exemplu în figura anterioară putem schimba culoarea punctelor și a dreptei de regresie.

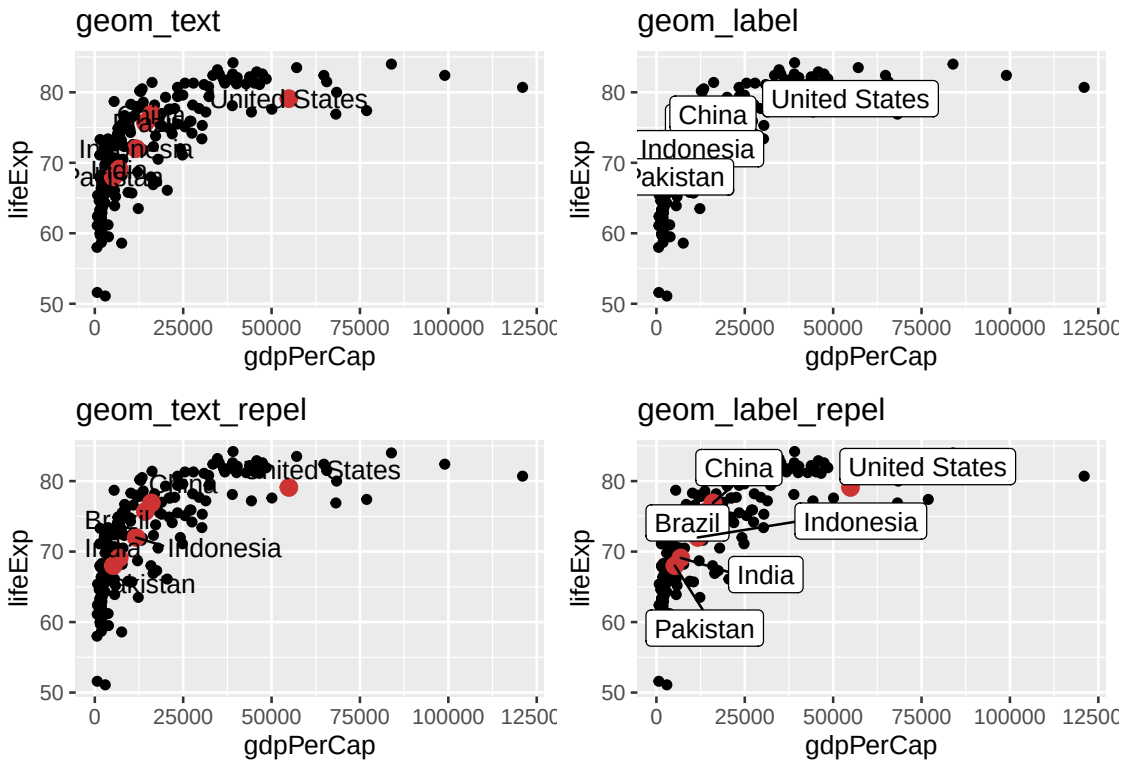
```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(color = "royalblue") +
  geom_smooth(method = "lm", color = "brown3")
```



De asemenea, este posibil să atribuim fiecărui strat `geom` propriul set de date astfel încât să putem utiliza

mai multe seturi de date pentru a scoate în evidență caracteristicile de interes ale graficului (variabilelor). În contextul diagramei de împrăștiere dintre produsul intern brut pe cap de locuitor și speranța medie de viață pentru setul de date `gapminder_2018` vrem să adăugăm numele (etichetele - label) țărilor care au mai mult de 200 milioane de locuitori și să colorăm punctele în roșu. Pentru început, vom construi setul de date cu țările corespunzătoare și apoi vom adăuga informația folosind `geom_text`. Etichetarea elementelor grafice se poate face cu `geom_text` sau `geom_label` dar atunci când punctele sunt foarte aglomerate se pot folosi și funcții alternative precum cele din pachetul `ggrepel` (`geom_text_repel` sau `geom_label_repel`). Pentru a colora punctele vom folosi estetica `colour` într-un nou strat `geom_point`. :

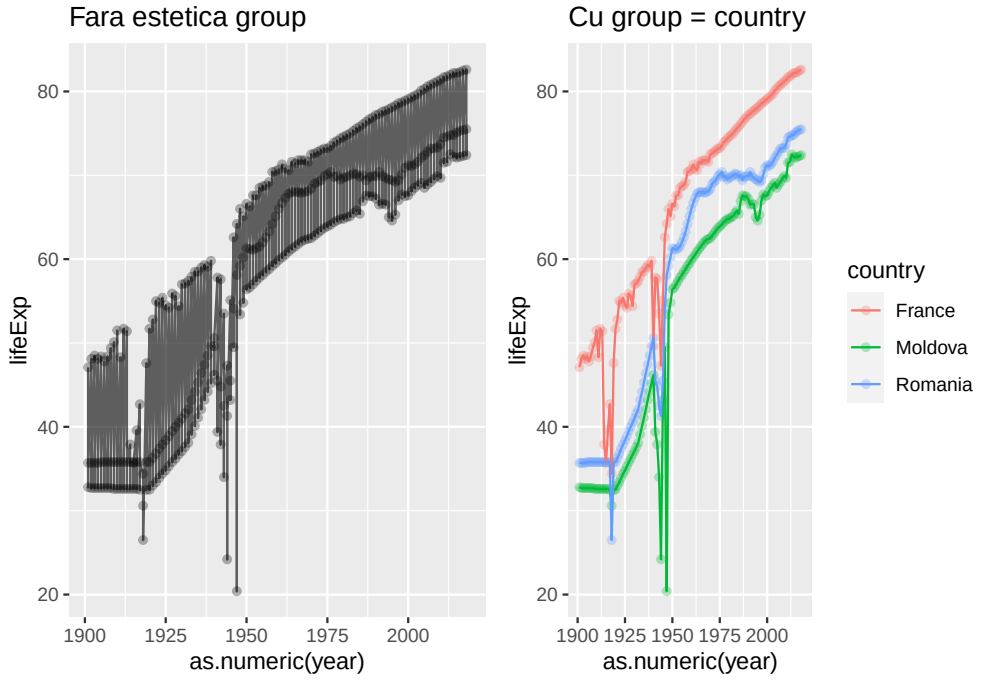
```
tari_mari = gapminder_2018 %>%
  filter(pop > 2e8)

ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point() +
  geom_point(data = tari_mari,
             aes(x = gdpPerCap, y = lifeExp),
             colour = "brown3", size = 3) +
  geom_text(data = tari_mari, aes(label = country))
```



Sunt și situații în care dorim să separăm datele în grupuri dar vrem să le redăm în același mod, de exemplu atunci când avem un studiu longitudinal (studiu care presupune observații repetate asupra aceloraşi variabile) cu mai mulți subiecți. Acest tip de situație, întâlnit în cazul în care un `geom` afișează observații multiple printr-un singur obiect geometric (e.g. `geom_boxplot`), necesită separarea pe grupe a observațiilor, fapt realizat prin intermediul elementului estetic `group`. Pentru o mai bună înțelegere vom ilustra acest concept folosind setul de date `gapminder_all`. Să presupunem că dorim să afișăm evoluția duratei medii de viață după anul 1900 pentru o submulțime de țări (*Romania*, *Moldova* și *France*).

```
gapminder_all %>%
  filter(as.numeric(year) > 1900) %>%
  filter(country %in% c("Romania", "Moldova", "France")) %>%
```

```
ggplot(aes(x = year, y = lifeExp)) +
  geom_point()+
  geom_line()

gapminder_all %>%
  filter(as.numeric(year) > 1900) %>%
  filter(country %in% c("Romania", "Moldova", "France")) %>%
ggplot(aes(x = year, y = lifeExp, group = country, colour = country)) +
  geom_point()+
  geom_line()
```



4.5 Transformări statistice (*statistical transformations*)

Pachetul `ggplot2` folosește o serie de transformări statistice, sau `stat`, pentru a aduce datele la formatul dorit pentru trasare. Forma generală a acestora este `stat_[nume statistică]`. De obicei aceste transformări sumarizează datele într-un anume fel, de exemplu în cazul unui *boxplot* pentru valorile de pe `y` se calculează mediana (Q_2), prima și a treia cuartila (Q_1 și Q_3) și distanța dintre acestea (IQR), iar în cazul unei curbe de regresie (`geom_smooth`) se calculează valoarea medie a lui `y` condiționată la `x`. Un alt exemplu constă în trasarea unei diagrame cu bare (*barplot*) în care axa `y` este definită ca fiind numărul de elemente din fiecare clasă (`count`) a lui `x`, număr care nu aparține setului de date inițial ci este obținut prin aplicarea unei transformări statitice `stat_count`. Fiecare obiect geometric `geom` admite o transformare statistică predefinită și vice versa dar acestea pot fi schimbate. De exemplu, transformarea statistică predefinită pentru `geom_bar` este `stat_count`

```
# argumentele functiei geom_bar
args(geom_bar)
function (mapping = NULL, data = NULL, stat = "count", position = "stack",
    ..., width = NULL, binwidth = NULL, na.rm = FALSE, orientation = NA,
    show.legend = NA, inherit.aes = TRUE)
NULL
```

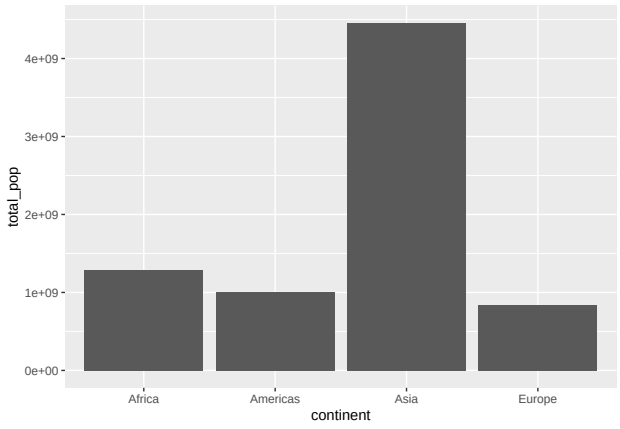
```
# argumentele functiei stat_count
args(stat_count)
function (mapping = NULL, data = NULL, geom = "bar", position = "stack",
    ..., width = NULL, na.rm = FALSE, orientation = NA, show.legend = NA,
    inherit.aes = TRUE)
NULL
```

Următorul tabel prezintă o serie de transformări statistice și obiectele geometrice pentru care acestea reprezintă un comportament implicit:

Transformare statistică	Obiect geometric
`stat_identity`	`geom_point`
`stat_count()`	`geom_bar()`
`stat_bin()`	`geom_freqpoly()`, `geom_histogram()`
`stat_boxplot()`	`geom_boxplot()`
`stat_smooth()`	`geom_smooth()`
`stat_bin2d()`	`geom_bin2d()`
`stat_binhex()`	`geom_hex()`
`stat_bindot()`	`geom_dotplot()`
`stat_contour()`	`geom_contour()`

De exemplu transformarea *identitate* (*identity*) va lăsa datele așa cum sunt și folosită în `geom_bar` conduce la construcția unei diagrame cu bare în care pe axa `y` se află valorile variabilei `y` fără a mai efectua transformarea (în acest caz este nevoie atât de estetica `x` cât și de `y`). În figura următoare afișăm populația totală a țărilor de pe fiecare dintre cele patru continente considerate în setul de date `gapminder_2018`:

```
gapminder_2018 %>%
  group_by(continent) %>%
  summarise(n = n(),
            max_pop = max(pop),
            total_pop = sum(pop)) %>%
  ggplot(aes(x = continent, y = total_pop)) +
  geom_bar(stat = "identity")
```

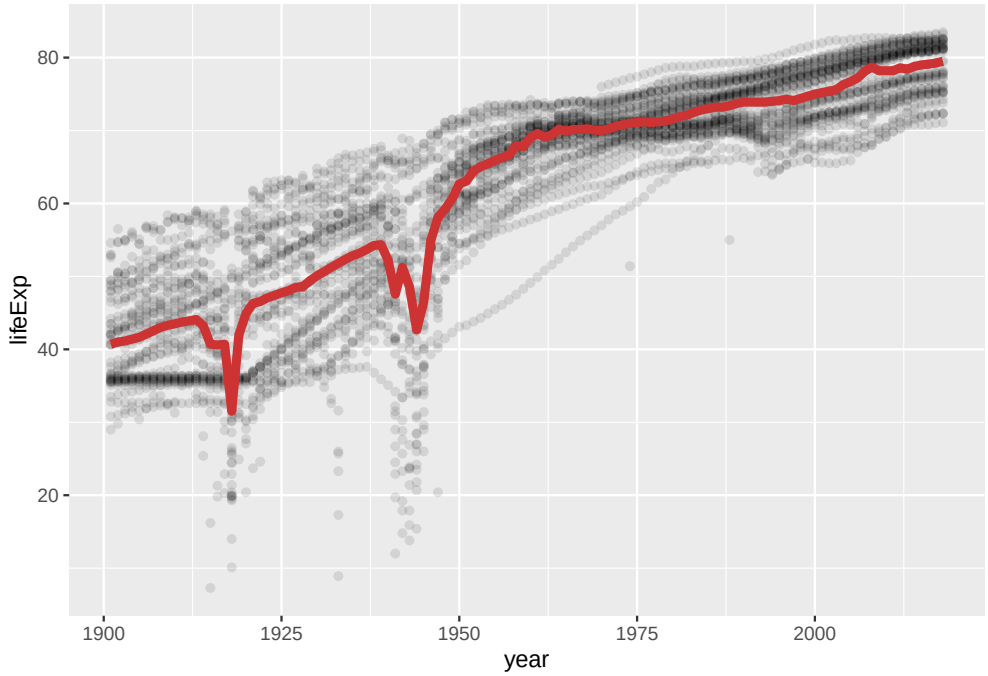


Pe lângă transformările implicite există și o serie de transformări care nu pot fi create prin aplicarea funcțiilor `geom_`. Tabelul de mai jos prezintă o parte dintre acestea (pentru mai multe informații se poate consulta documentația pachetului `ggplot2` la adresa https://ggplot2.tidyverse.org/reference/index.html#section-layer-stats):

Transformare statistică	Descriere
`stat_ecdf()`	calculează și ilustrează funcția de repartiție empirică
`stat_function()`	calculează valorile lui y aplicând o funcție valorilor lui x
`stat_summary()`	sumarizează valorile lui y la clase diferite în x
`stat_summary2d()`, `stat_summary_hex()`	este o versiune bidimensională a lui `stat_summary`
`stat_unique()`	elimină valorile duplicate
`stat_qq()`	efectuează calculele pentru un grafic cuantilă-cuantilă

Pentru a folosi aceste funcții putem sau să le apelăm în mod direct utilizând `stat_[nume]` și astfel suprascriind obiectul geometric sau să le apelăm în interiorul unui strat geometric. Pentru ilustrare vom folosi setul de date `gapminder_all` în care vom trasa o diagramă de împrăștiere unde pe axa x avem anii de după 1900 iar pe y speranța medie de viață a țărilor europene (pentru a evita supraaglomerarea punctelor - *ovreplotting* vom folosi estetic `alpha = 0.1`). Vom adăuga cu roșu la acestă diagramă valoarea mediană a duratei medii de viață a tuturor țărilor pentru fiecare an și vom uni aceste valori printr-o linie:

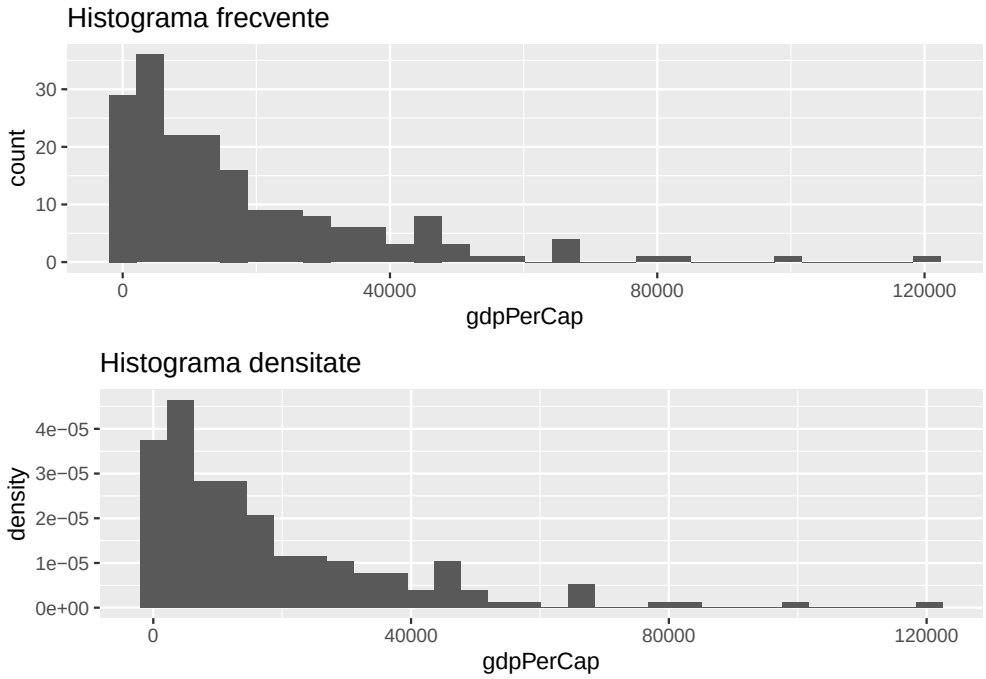
```
# transformam datele
gapminder_all %>%
  mutate(year = as.numeric(year),
         continent = four_regions) %>%
  filter(year > 1900,
         continent == "Europe") %>%
  ggplot(aes(x = year, y = lifeExp)) +
  # trasam diagrama de imprastiere
    geom_point(alpha = 0.1) +
  # unim punctele mediane
    geom_line(stat = "summary", fun = "median",
              colour = "brown3", linetype = 1, size = 2)
```



Trebuie avut în vedere că atunci când funcțiile `stat_` transformă datele, în realitate se construiește un alt `data.frame` la care pot să apară *noi* variabile (noi coloane) și prin urmare este posibil să utilizăm aceste

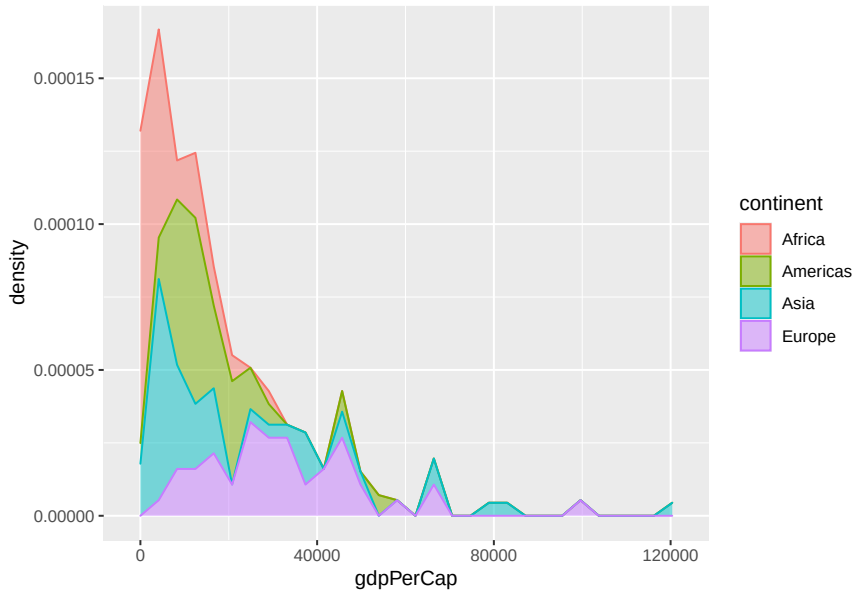
noi variabile în trasarea graficului. De exemplu, în cazul transformării `stat_bin` (statistica folosită pentru trasarea unei histograme) sunt construite următoarele variabile: `count` - numărul de observații din fiecare subinterval (*bin*), `density` - densitatea observațiilor din fiecare subinterval (procentul din total pe lungimea subintervalului), `x` - centrul subintervalului. Aceste variabile pot fi folosite în trasarea graficului folosind `..` în jurul numelui acestora (i.e. `..[nume variabilă]..`) pentru a evita o eventuală confuzie cu o variabilă din setul de date inițial care are aceeași denumire. Spre exemplu atunci când trasăm o histogramă observăm că în mod automat (implicit) înălțimea barelor este egală cu numărul de observații din subintervalul corespunzător (`count`) prin urmare vorbim de o histogramă de frecvențe și nu una de densitate. Pentru a obține-o pe cea din urmă trebuie să specificăm că înălțimea barelor este dată de variabila `density`. În contextul setului de date `gapminder_2018` aceasta se traduce prin

```
# histograma frecvente
ggplot(gapminder_2018, aes(x = gdpPerCap)) +
  geom_histogram()

# histograma densitate
ggplot(gapminder_2018, aes(x = gdpPerCap)) +
  geom_histogram(aes(y = ..density..))
```



Mai mult dacă dorim să comparăm modul de repartiție a produsului intern brut pe cap de locuitor pentru fiecare continent atunci este recomandată folosirea datelor standardizate:

```
ggplot(gapminder_2018, aes(x = gdpPerCap, colour = continent, fill = continent)) +
  geom_area(stat = "bin", aes(y = ..density..), alpha = 0.5)
```



4.6 Gestionarea scalelor (*scales*)

De fiecare dată când specificăm o corespondență de estetică/aspect (*aesthetic mapping*), funcția `ggplot` folosește o anumită *scală* (predefinită în funcție de tipul esteticii) pentru a determina intervalul de valori pe care datele trebuie să le verifice. O corespondență estetică (i.e. specificată cu `aes()`) ne spune doar că o variabilă ar trebui asociată unei caracteristici ce ține de aspect și nu cum trebuie să aibă loc această relație. De exemplu, atunci când folosim estetica `colour` (`aes(colour = x)`) nu specificăm nicăieri ce culori trebuie folosite pentru trasarea elementelor specifice nivelelor variabilei `x`. Aceeași observație este valabilă și pentru alte atribute estetice, i.e. `size`, `fill`, `shape`, etc. Pentru a specifica ce culoare, mărime, formă, etc. să fie utilizată trebuie modificată scala corespunzătoare. În pachetul `ggplot2`, scalele (*scales*) pot fi modificate prin intermediul funcțiilor de tipul

```
scale_[nume estetică]_[tip]
```

unde `nume estetică` reprezintă numele elemntului estetic (i.e. `x`, `y`, `colour`, `fill`, etc.) iar `tip` reprezintă tipul scalei folosite pentru elementul ales (i.e. `continuous`, `discrete`, `manual`, `gradient`, etc.). În funcție de estetică, tipul scalei poate să difere

Trebuie menționat că `ggplot` asociază în mod automat fiecărui element estetic folosit în trasarea graficului o scală. De exemplu dacă scriem

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(aes(colour = continent))
```

în realitate are loc

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(aes(colour = continent)) +
  scale_x_continuous() +
  scale_y_continuous() +
  scale_color_discrete()
```

dar pentru a evita scrierea manuală a fiecărei scale, `ggplot2` face lucrurile automat. Este foarte important de menționat că scalele pot modifica unul din cele două elemente de ghidaj (*guides*) ale unui grafic: axele și legendele. În `ggplot2` cele două elemente, chiar dacă vizual sunt diferite, ele sunt tratate în mod similar, legătura fiind evidențiată în tabelul următor:

Axă	Legendă	Numele parametrului
Nume (*Label*)	Titlu (*Title*)	`name`
Markeri (*Ticks & grid line*)	Intrările legendei (*Key*)	`breaks`
Etichete Markeri (*Tick label*)	Etichetele intrărilor legendei (*Key label*)	`labels`

În general, scalele disponibile în `ggplot2` pot fi împărțite în patru categorii, primele trei fiind cele mai utilizate:

- *scale continue de poziție* folosite pentru a face corespondența dintre datele de tip numeric sau de tip temporal (date/time) și poziția pe x sau y (i.e. `scale_x_continuous`, `scale_y_continuous`)
- *scale de culori* care permit corespondența dintre valorile continue sau discrete ale datelor și culori (i.e. `scale_color_discrete`, `scale_fill_continuous`, `scale_fill_gradient`)
- *scale manuale*, utilizate pentru a defini relația dintre valorile discrete ale datelor și mărimea (*size*), tipul liniei (*linetype*), forma (*shape*) sau culoarea (*colour*) dorită
- *scale identice*, folosite pentru a trasa variabilele fără a le scala (de exemplu atunci când avem deja un vector de culori)

O scală continuă va fi folosită cu precădere atunci când vorbim de date numerice (unde există un set continuu de numere), în timp ce o scală discretă va gestiona elemente precum culorile, forma punctelor sau tipul liniilor (deoarece există o listă mică de culori distincte).

În tabelul de mai jos sunt prezentate o parte dintre scalele cele mai utilizate (pentru mai multe detalii se poate consulta documentația pachetului ggplot2)

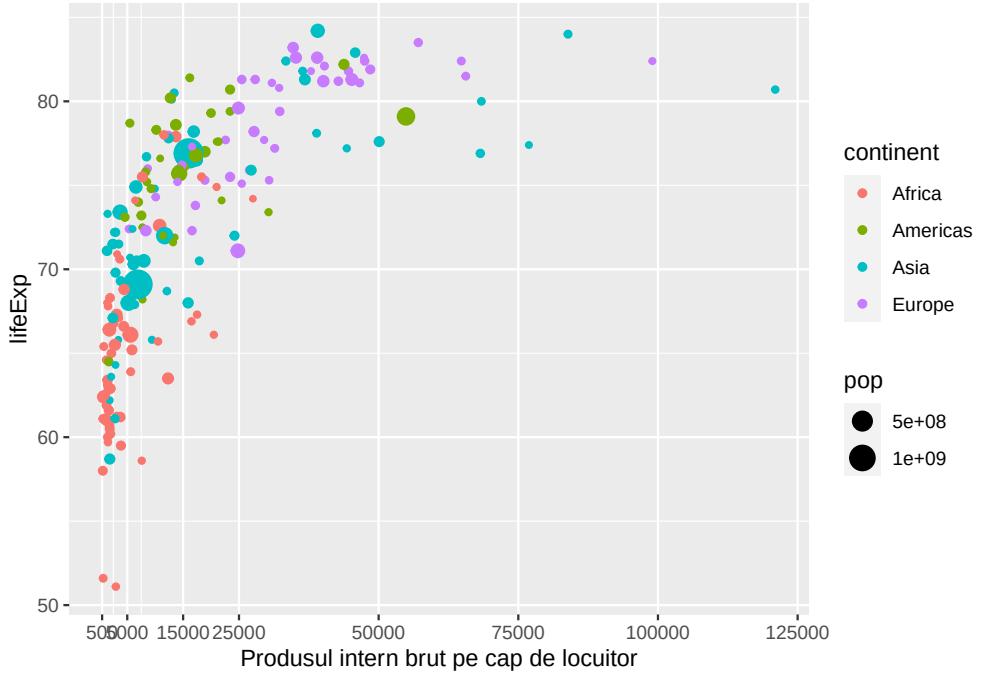
Scala	Tipul	Exemple
`scale_color_`	`identity`	`scale_fill_continuous`
`scale_fill_`	`manual`	`scale_color_discrete`
`scale_size_`	`continuous`	`scale_size_manual`
	`discrete`	`scale_size_discrete`
`scale_shape_`	`discrete`	`scale_shape_discrete`
`scale_linetype_`	`identity`	`scale_shape_manual`
	`manual`	`scale_linetype_discrete`
`scale_x_`	`continuous`	`scale_x_continuous`
`scale_y_`	`discrete`	`scale_y_discrete`
	`reverse`	`scale_x_log`
	`log`	`scale_y_reverse`
	`date`	`scale_x_date`
	`datetime`	`scale_y_datetime`

4.6.1 Scale de poziție

Pentru a ajusta scala axei x pentru o variabilă continuă vom folosi `scale_x_continuous`. Utilizarea funcțiilor de scală permite modificarea mai multor proprietăți ale elementului estetic, de exemplu în cazul scalei pe axa x putem modifica aspecte ale axei precum eticheta/titlul axei (*name*), poziția (*position*) sau distanța dintre markeri (*breaks* sau *minor_breaks*) și numele acestora (*labels*). Pentru alte elemente estetice în afara lui x și y, *axa* va fi de obicei legenda graficului și astfel modificarea scalei unui asemenea element permite modificarea elementelor componente ale legendei (nume, etichete, culori, etc.).

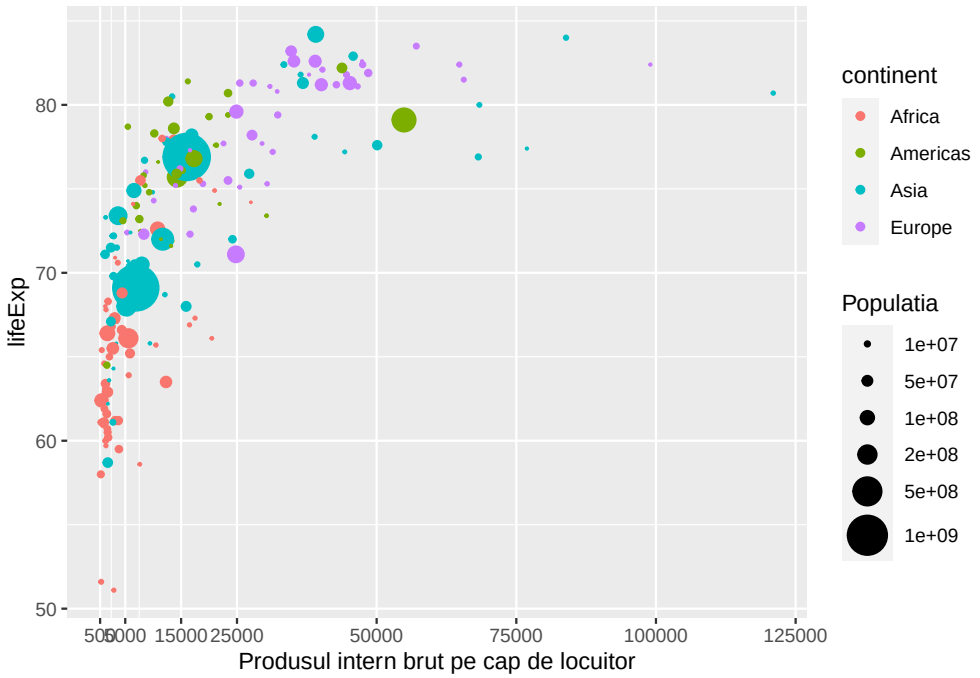
Ca exemplu, vom folosi setul de date `gapminder_2018` și folosind `scale_x_continuous` vom modifica numele axei x, poziția markerilor de separare (*breaks*) principali și secundari (*minor_breaks*):

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent, size = pop)) +
  geom_point() +
  scale_x_continuous(name = "Produsul intern brut pe cap de locuitor",
                     breaks = c(500, 5000, 15000, 25000,
                                50000, 75000, 100000, 125000),
                     minor_breaks = c(500, 2500, 7500))
```



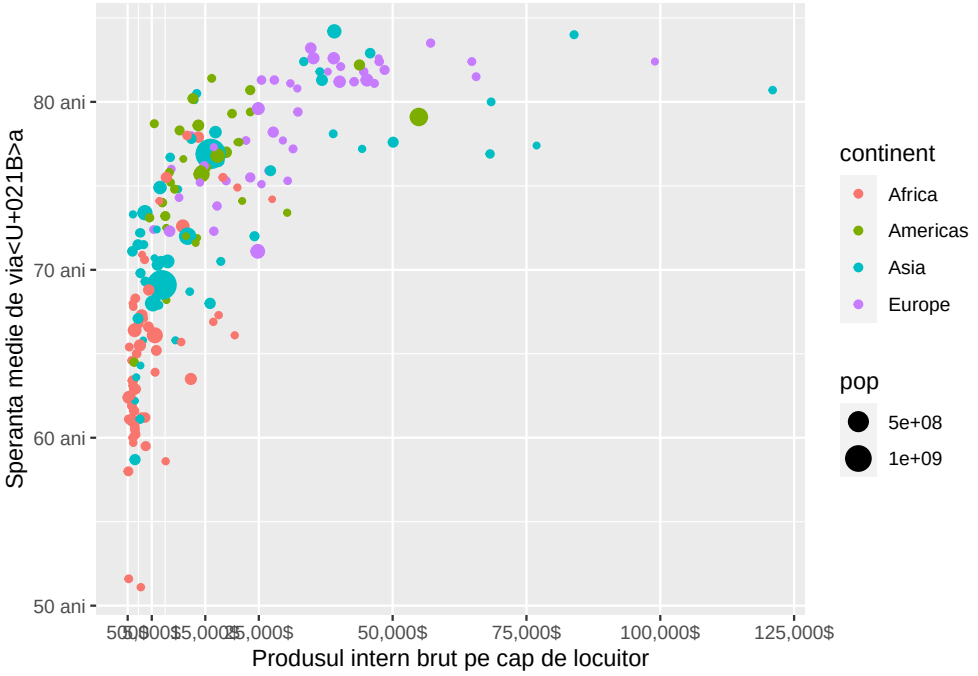
Este posibil să dorim de asemenea schimbarea numelui legendei ce corespunde variabilei `pop` (populației) din *pop* în *Populația*, să afișăm mărimile 1e7, 5e7, 1e8, 2e8, 5e8, 1e9 (*breaks*) și să schimbăm intervalul de valori (minim-maxim) pentru mărimea simbolului (*range*):

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent, size = pop)) +
  geom_point() +
  scale_x_continuous(name = "Produsul intern brut pe cap de locuitor",
                     breaks = c(500, 5000, 15000, 25000,
                                50000, 75000, 100000, 125000),
                     minor_breaks = c(500, 2500, 7500)) +
  scale_size(name = "Populatia",
             breaks = c(1e7, 5e7, 1e8, 2e8, 5e8, 1e9),
             range = c(0.1, 10))
```



În plus, putem să adăugăm funcții ca argument al parametrilor `breaks` și `labels`, în primul caz funcția returnând un vector de valori numerice care să corespundă elementelor de marcaj iar în cazul etichetelor va primi ca date de intrare un vector numeric de elemente de marcaj și va returna un vector de etichete. De exemplu, pentru a adăuga simbolul `$` în dreptul valorilor produsului intern brut vom folosi funcții din pachetul `scales` precum `dollar_format()` iar pentru a modifica axa y vom crea propria funcție:

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent, size = pop)) +
  geom_point() +
  scale_x_continuous(name = "Produsul intern brut pe cap de locuitor",
                     breaks = c(500, 5000, 15000, 25000,
                                50000, 75000, 100000, 125000),
                     minor_breaks = c(500, 2500, 7500),
                     labels = scales::dollar_format(prefix = "", suffix = "$")) +
  scale_y_continuous(name = "Speranta medie de viață",
                     labels = function(x){paste(x, "ani")})
```

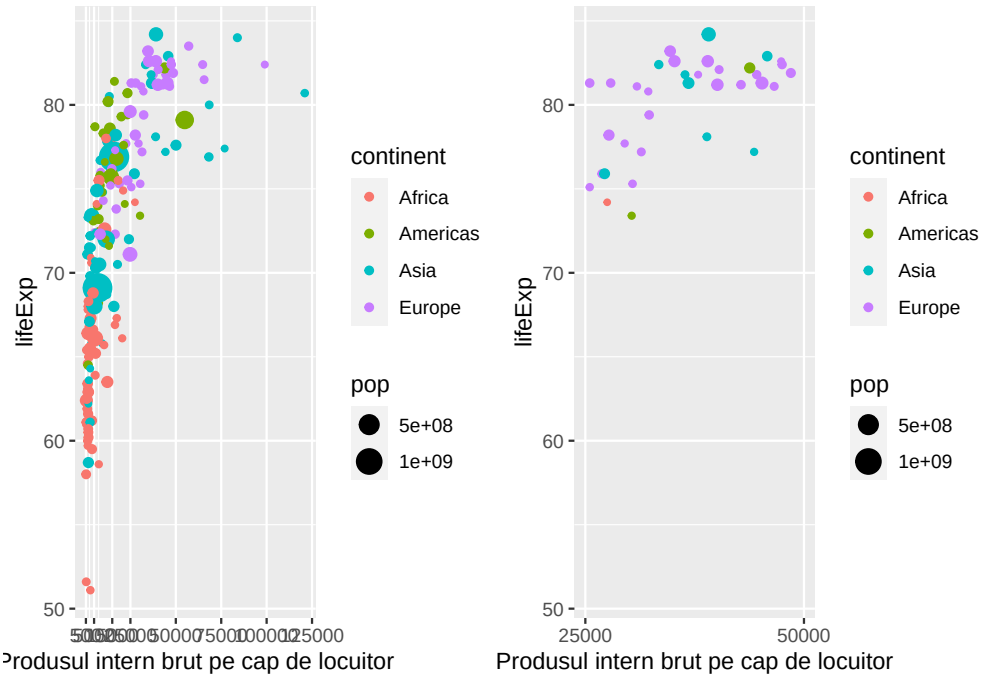


Fiecare funcție de scalare permite utilizarea unui număr diferit de parametrii, o parte dintre aceștia regăsindu-se în tabelul de mai jos (pentru a vizualiza toți parametrii disponibili se poate consulta documentația funcției, i.e. `?scale_x_continous`):

Parametru	Descriere
`name`	Eticheta (label) axei sau numele legendei
`breaks`	Vector de puncte de marcaj
`minor_breaks`	Vector de puncte de marcaj intermediare
`labels`	Etichete folosite pentru fiecare punct de marcaj
`limits`	Limitele intervalului de valori ale axei

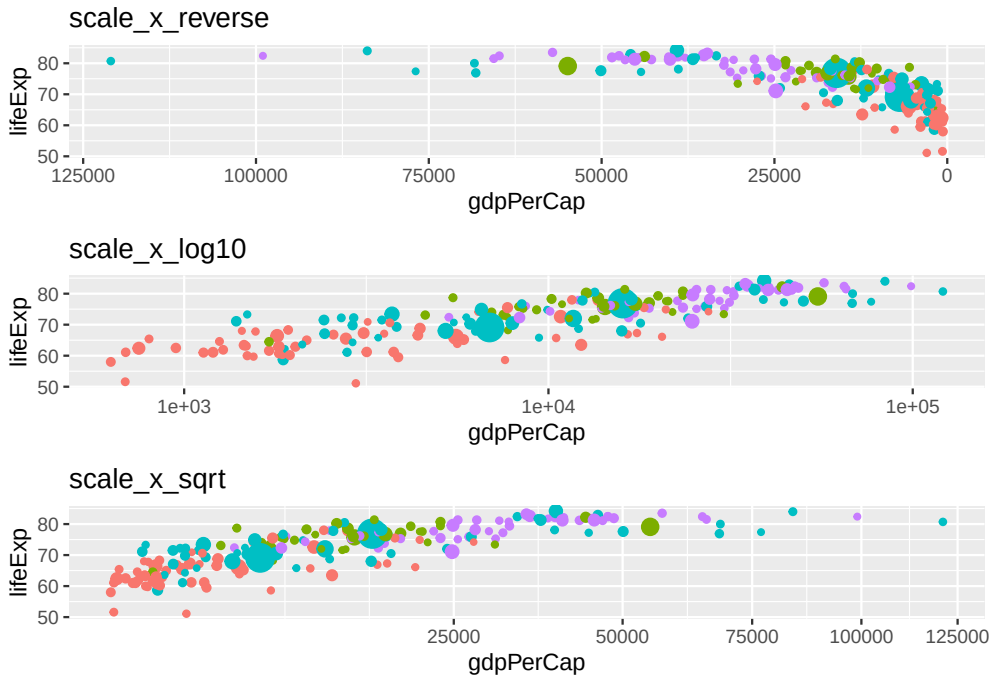
Modul de folosire a parametrilor `name`, `breaks`, `minor.breaks` și `labels` a fost ilustrat în exemplele anterioare. Parametrul `limits` este derivat din domeniul de valori al setului de date și este folosit cu precădere pentru a scoate în evidență o subregiune a graficului. Atunci când facem referire la scale continue parametrul `limits` primește ca argument de intrare un vector cu două valori (minim și maxim).

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent, size = pop)) +
  geom_point() +
  scale_x_continuous(name = "Produsul intern brut pe cap de locuitor",
                     breaks = c(500, 5000, 15000, 25000,
                                50000, 75000, 100000, 125000),
                     minor_breaks = c(500, 2500, 7500),
                     limits = c(25000, 50000))
```



Deoarece modificarea limitelor axelor este o operație des întânlnită atunci când efectuăm analize exploratorii a datelor cu care lucrăm, pachetul `ggplot2` pune la dispoziție o serie de funcții predefinite precum `xlim()`, `ylim()` sau `lims()`.

Pachetul `ggplot2` pune la dispoziție și o serie de scale care permit modificarea axelor, de exemplu, prin inversarea valorilor (i.e. `scale_x_reverse`) sau prin transformarea acestora într-o scală logaritmică (folosind `scale_x_log10` sau `scale_x_continuous(trans = "log10")`) ori printr-o altă funcție (i.e. `scale_x_sqrt` sau `scale_x_continuous(trans = "sqrt")`). Pentru mai multe transformări se poate consulta documentația funcției `scale_x_continuous` la argumentul `trans`.

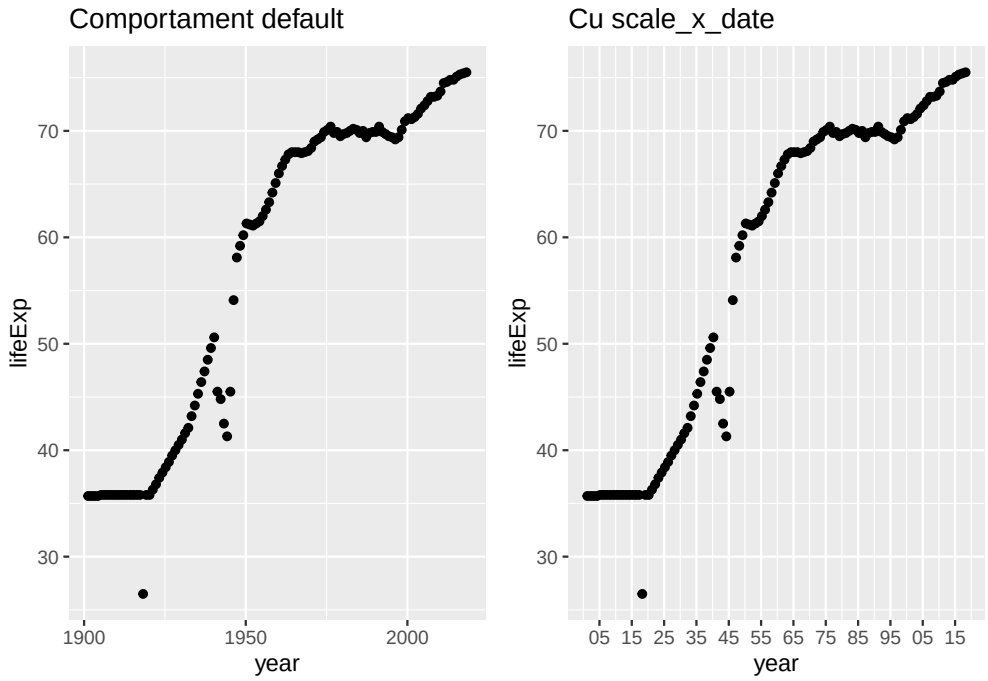


Atunci când avem de-a face cu variabile/date care sunt în format de date de timp (format `Date` sau `POSIXct`) este recomandată utilizarea funcțiilor predefinite `scale_x_date` și respectiv `scale_x_datetime` care dispun în plus de argumentele `date_breaks` și `date_labels`. Argumentul `date_breaks` permite poziționarea elementelor de marcaj în funcție de unitățile de timp (ani, luni, săptpmâni, etc.), astfel `date_breaks = 2 years` va pasa marcaje din doi în doi ani. Argumentul `date_labels` asigură ilustrarea etichetelor pentru marcaje în format `strptime()`.

Sir de caractere	Descriere
`%S`	secunde (00-59)
`%M`	minute (00-59)
`%l`	ore, format ceas 12 ore (1-12)
`%I`	ore, format ceas 12 ore (01-12)
`%p`	am/pm
`%H`	ore, format ceas 24 ore (00-23)
`%a`	zilele săptămânii, sub forma (Mon-Sun)
`%A`	zilele săptămânii, sub forma (Monday-Sunday)
`%e`	ziua din lună (1-31)
`%d`	ziua din lună (01-31)
`%m`	luna, format numeric (01-12)
`%b`	luna, format abreviat (Jan-Dec)
`%B`	luna, nume întreg (January-December)
`%y`	an, fără specificarea secolului (00-99)
`%Y`	an, cu specificarea secolului (0000-9999)

De exemplu dacă vrem să afișăm date precum 23/08/1944 vom folosi șirul de caractere `"%d/%m/%Y"`. Următorul cod ilustrează aceste opțiuni în contextul setului de date `gapminder_all` unde ne interesăm la evoluția duratei medii de viață în țara noastră după anul 1900.

```
gapminder_all %>%
  filter(country == "Romania",
         as.numeric(year) > 1900) %>%
  mutate(year = as.Date(year, format = "%Y")) %>%
ggplot(aes(x = year, y = lifeExp)) +
  geom_point() +
  scale_x_date(date_labels = "%y", date_breaks = "10 years")
```



4.6.2 Scale de culori

O altă clasă de scale des folosite sunt cele care corespund esteticii de culoare (`colour` - culoarea marginală/exterioară și respectiv `fill` - culoarea de umplere). Pachetul `ggplot2` pune la dispoziție mai multe tipuri de scale diferențiate după cum variabilele de scalare sunt continue sau discrete. Spațiul culorilor din `ggplot2` este de tip HCL (*hue*, *chroma* și *luminance*), pentru mai multe detalii se poate consulta http://www.handprint.com/HP/WCL/color7.html, dar se pot folosi si palete de culoare de tip Brewer (pentru explorarea acestora se poate vizita https://colorbrewer2.org/). Acestea din urmă pot fi clasificate în trei categorii: secvențiale, divergente și calitative unde primele două categorii corespund variabilelor continue iar cea de-a treia corespunde variabilelor cu valori discrete.

Set1 (qualitative)

PuBuGn (sequential)

Accent (qualitative)

Spectral (divergent)

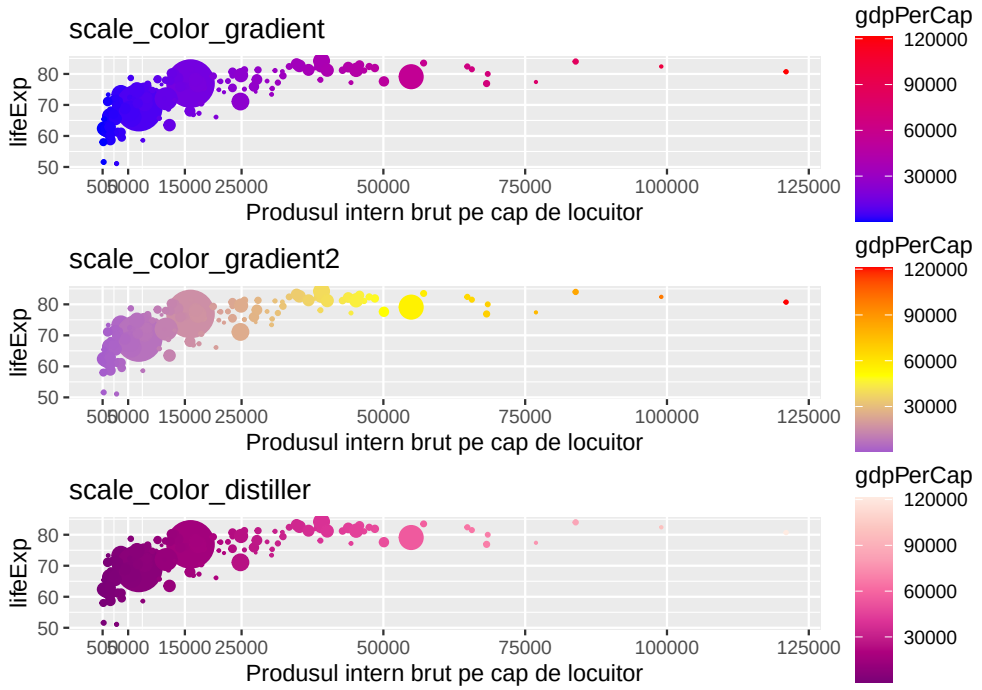
Atunci când utilizăm variabile continue se folosesc gradiente de culoare prin intermediul funcțiilor generice de forma `scale_[estetică culoare]_[tip]`, după cum urmează

Funcție generică	Scală	Tip	Descriere
`scale_[estetică culoare]_[tip]`	`colour`	`gradient`	Scală predefinită (similară cu `scale_colour_continuous()`) care folosește un gradient de culoare bazat pe două culori (*low* și *high*)
	`fill`	`gradient2`	Scală care folosește un gradient de culoare cu trei valori (*low*, *high* și *mid* - pentru acesta din urmă se poate selecta valoarea de *midpoint*)
		`gradientn`	Scală care folosește un gradient de culoare bazat pe n valori
		`distiller`	Scală care este bazată pe palete de tip Brewer

Pentru ilustrare vom folosi setul de date `gapminder_2018`:

```
# grafic de baza
baza = ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = gdpPerCap,
                                  size = pop)) +
  geom_point()

# gradient 2 culori
baza +
  scale_color_gradient(low = "blue", high = "red")

# gradient 3 culori
baza +
  scale_color_gradient(low = "blue", high = "red")

# gradient Brewer - RdPu
baza +
```

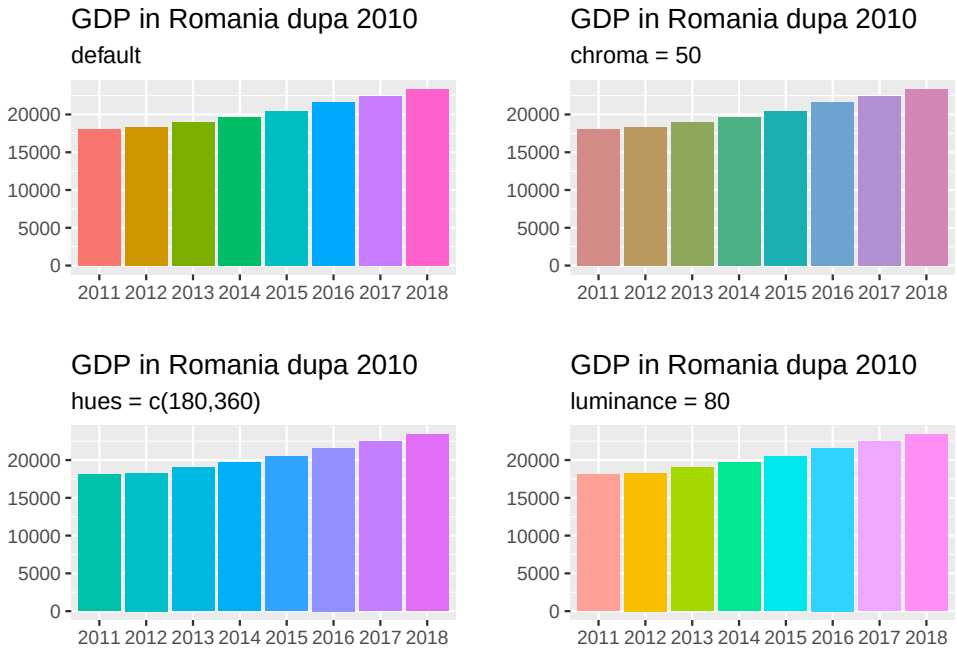
```
  scale_color_distiller(palette = "RdPu")
```



În cazul variabilelor discrete, putem folosi patru scale pentru fiecare dintre esteticile `colour` și `fill`: `scale_[estetică culoare]_hue()`, `scale_[estetică culoare]_brewer()`, `scale_[estetică culoare]_gray()` și `scale_[estetică culoare]_manual()`. Scala predefinită în cazul variabilelor discrete este scala `scale_[estetică culoare]_hue()` care alege culori egal depărtate pe roata de culori de tip HCL. Scala `scale_[estetică culoare]_brewer()` folosește paletele de culori de tip Brewer iar scala `scale_[estetică culoare]_gray()` folosește o paletă de griuri, de la gri deschis la gri închis.

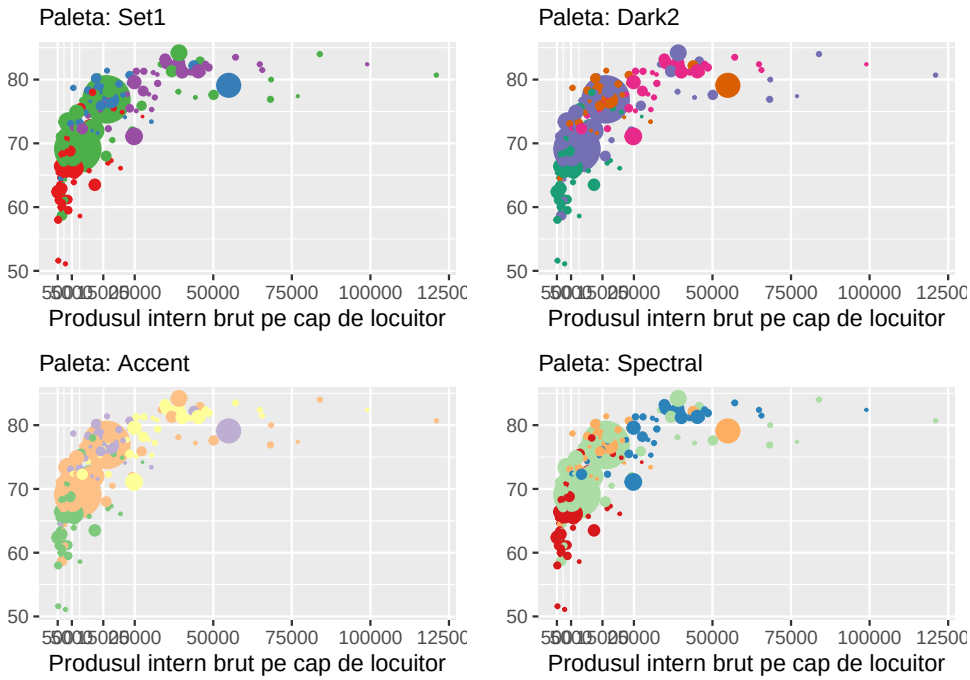
Pentru a ilustra scala `scale_[estetică culoare]_hue` vom folosi setul de date `gapminder_all` și vom afișa variația produsului intern brut pe cap de locuitor în România după anul 2000.

```
baza = gapminder_all %>%
  filter(country == "Romania",
         as.numeric(year) > 2010) %>%
  ggplot(aes(x = year, y = gdpPerCap, fill = year)) +
  geom_bar(stat = "identity")

baza +
  scale_fill_hue(c = 50)

baza +
  scale_fill_hue(h = c(180, 300))

baza +
  scale_fill_hue(l = 80)
```

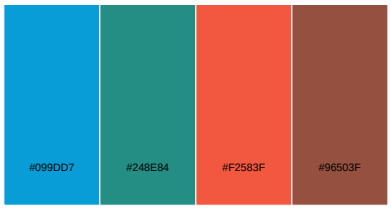


Următoarele grafice prezintă aceeași figură (situarea țărilor din setul de date `gapminder_2018` în funcție de produsul intern brut pe cap de locuitor și durată medie de viață) ilustrată cu ajutorul funcției `scale_[estetică culoare]_brewer()` pentru patru palete de culori diferite:

```
baza = ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent,
                                  size = pop), alpha = 0.7) +
  geom_point() +
  scale_x_continuous(name = "Produsul intern brut pe cap de locuitor",
                     breaks = c(500, 5000, 15000, 25000,
                                50000, 75000, 100000, 125000),
                     minor_breaks = c(500, 2500, 7500)) +
  scale_size(name = "Populatia",
             breaks = c(1e7, 5e7, 1e8, 2e8, 5e8, 1e9),
             range = c(0.1, 10)) +
  guides(size = "none", colour = "none") +
  labs(x = "", y = "")

p1 = baza +
  scale_color_brewer(palette = "Set1") +
  labs(subtitle = "Paleta: Set1")

p2 = baza +
  scale_color_brewer(palette = "Dark2") +
  labs(subtitle = "Paleta: Dark2")

p3 = baza +
  scale_color_brewer(palette = "Accent") +
  labs(subtitle = "Paleta: Accent")

p4 = baza +
  scale_color_brewer(palette = "Spectral") +
  labs(subtitle = "Paleta: Spectral")
```

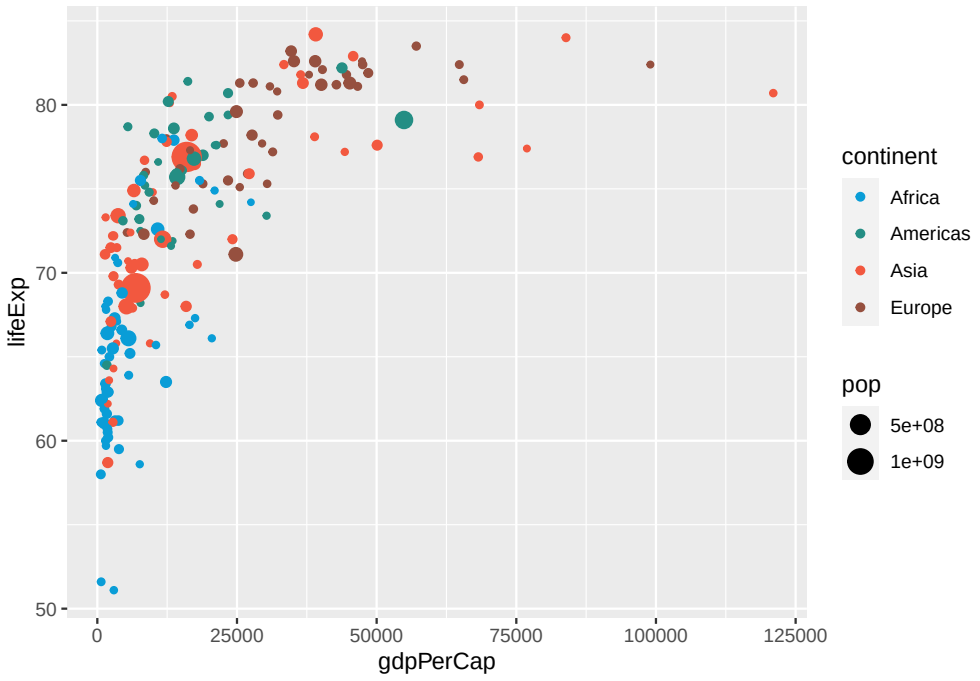


Funcţia `scale_[estetică culoare]_manual()` este folosită în special atunci când dorim să folosim propria paletă de culori. De exemplu, să presupunem că dorim să colorăm statele de pe fiecare continent cu o culoare corespunzătoare



atunci putem scrie

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent, size = pop)) +
  geom_point() +
  scale_colour_manual(values = c("#099DD7","#248E84","#F2583F", "#96503F"))
```

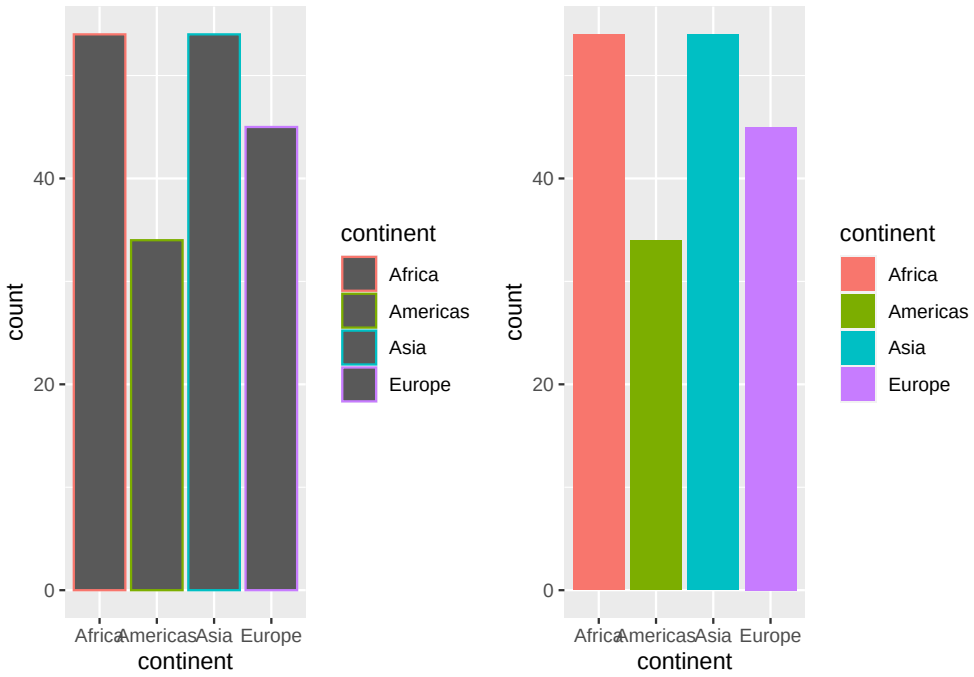


4.7 Ajustarea poziţiilor (*position adjustments*)

Fiecare strat geomentric *geom* prezintă, pe lângă o transformare statistică implicită, și o ajustare implicită a poziției care specifică un set de *reguli* privind modul în care componentele diferite ale graficului trebuie poziționate unele față de altele. Această poziționare implicită este vizibilă în special în cadrul funcției `geom_bar` atunci când aplicăm o variabilă diferită caracteristicii vizuale de colorare (e.g. `colour` sau `fill`).

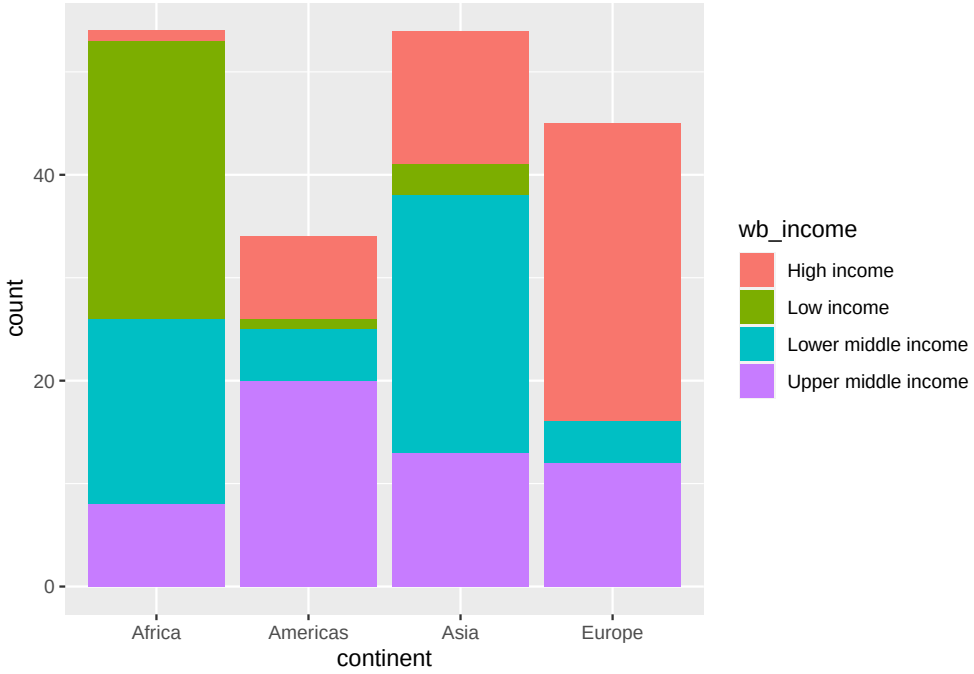
De exemplu, să considerăm diagrama cu bare care ilustrează numărul de tări de pe fiecare continent folosind setul de date `gapminder_2018`. Putem colora barele folosind sau estetica `colour` sau estetica `fill` (de preferat).

```
gapminder_2018 %>%
  ggplot() +
  geom_bar(aes(x = continent, colour = continent))

gapminder_2018 %>%
  ggplot() +
  geom_bar(aes(x = continent, fill = continent))
```



Acum dacă colorăm diagrama cu bare de mai sus în raport cu nivelul de venit (conform Băncii Mondiale) obținem o diagramă cu bare suprapuse colorate în funcție de această varaibilă.

```
gapminder_2018 %>%
  ggplot() +
  geom_bar(aes(x = continent, fill = wb_income))
```

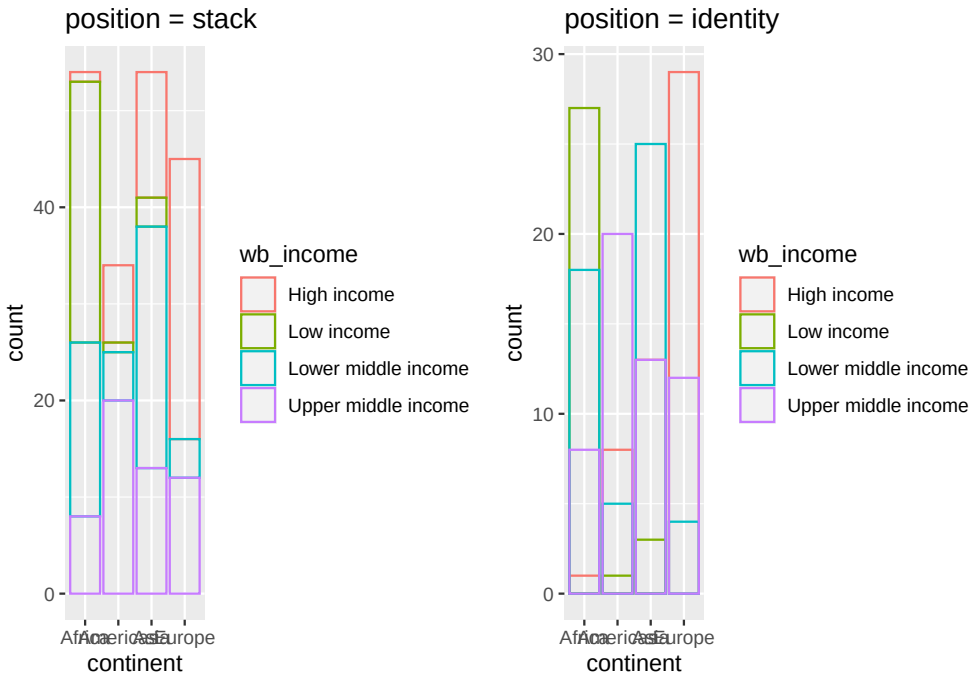


Suprapunerea barelor (`stack`) în `geom_bar` (înălțimea dreptunghiurilor este proporțională cu valoarea lor) se face în mod automat prin ajustarea poziției specificate de argumentul `position` (default `position = "stack"`). Dacă nu se dorește suprapunerea barelor atunci se poate folosi una din următoarele ajustări:

`identity`, `dodge` sau `fill`.

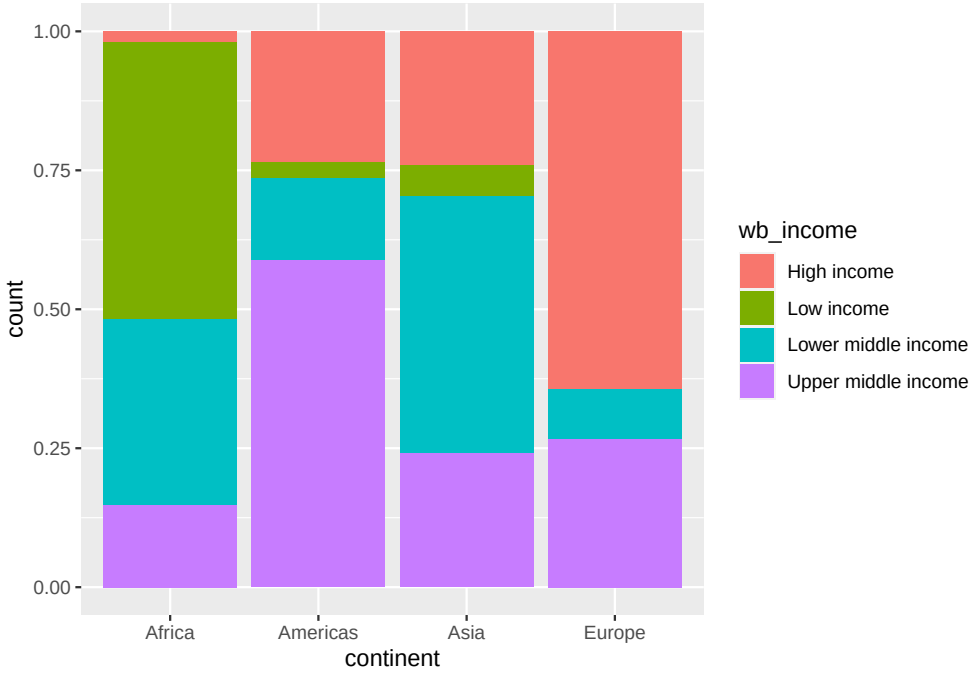
- `position = "identity"` - opțiunea impune plasarea fiecărui obiect exact unde se află în contextul figurii (nu este foarte folositoare atunci când trasăm diagrame cu bare dar este opțiunea de default pentru diagramele de împrăștiere)

```
gapminder_2018 %>%
  ggplot(aes(x = continent, colour = wb_income)) +
  geom_bar(position = "identity", fill = NA)
```



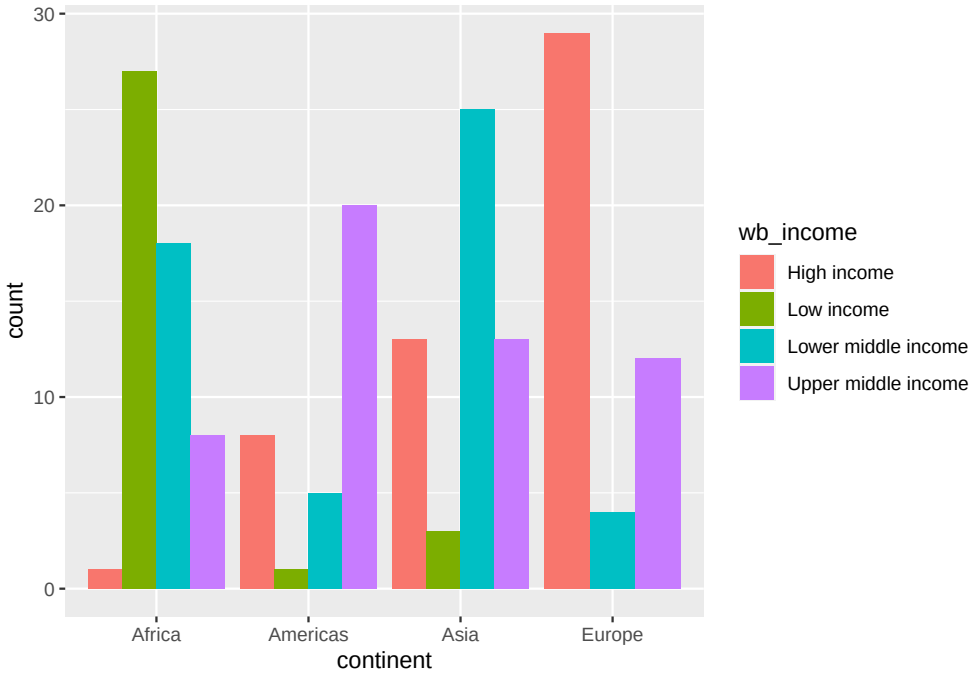
- `position = "fill"` - opțiunea este similară cu `stack`, suprapunând barele în așa fel încât înălțimea acestora să fie constantă (este de preferat atunci când dorim să comparăm proporții între grupuri)

```
gapminder_2018 %>%
  ggplot(aes(x = continent, fill = wb_income)) +
  geom_bar(position = "fill")
```



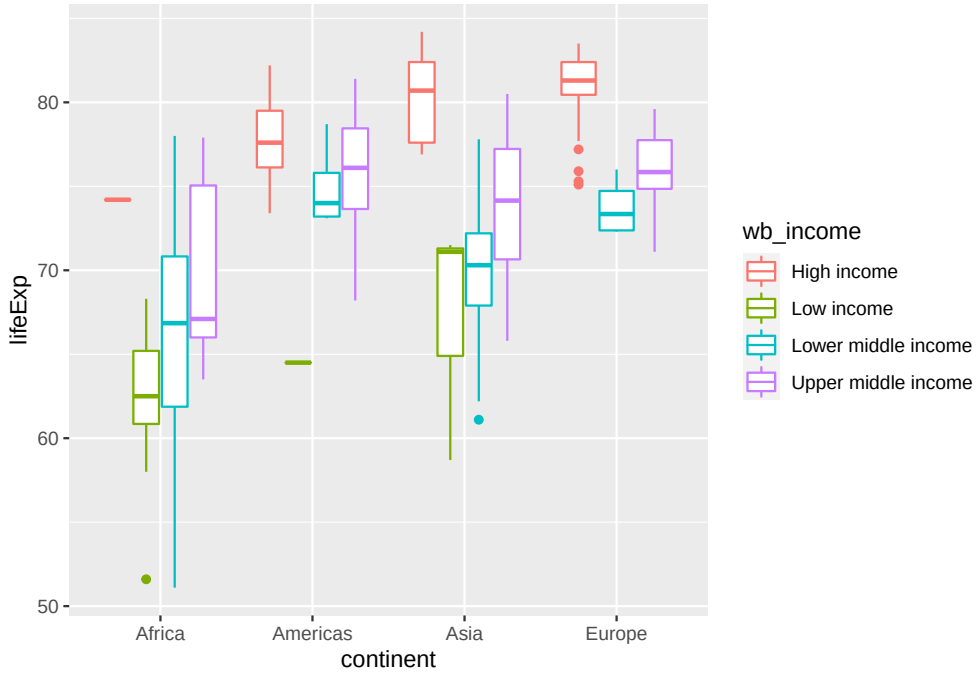
- `position = "dodge"` - opțiunea plasează obiectele unul lângă celălalt și permite astfel o mai bună comparare între valorile individuale

```
gapminder_2018 %>%
  ggplot(aes(x = continent, fill = wb_income)) +
  geom_bar(position = "dodge")
```



Trebuie menționat că `geom_boxplot` are ca și poziționare implicită poziționarea `dodge` după cum se poate observa și din exemplul de mai jos.

```
# position dodge - comportament de default pentru boxplot
gapminder_2018 %>%
  ggplot(aes(x = continent, y = lifeExp)) +
  geom_boxplot(aes(colour = wb_income))
```

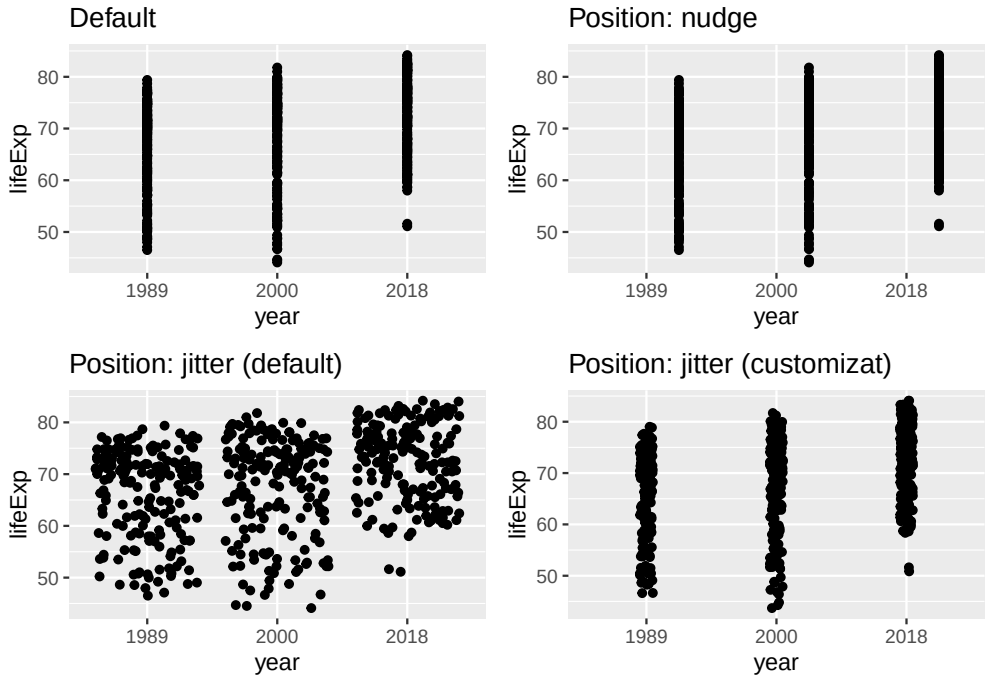


Tipurile de poziții prezentate mai sus se aplică în special diagramelor cu bare (sau boxplot-urilor) dar, trebuie menționat că pachetul `ggplot2` pune la dispoziție și trei metode de ajustare a pozițiilor în cazul punctelor: `position_nudge()` - asigură mutarea punctelor printr-o valoare fixată (default în cazul `geom_text()`); `position_jitter()` - adaugă poziției fiecărui punct un zgomot aleator pentru a dispersa un pic datele; `position_jitterdodge()` - permite separarea/evitarea (*dodge*) puntelor în interiorul grupurilor și apoi adaugă zgomot aleator pozițiilor.

Vom ilustra primele două ajustări de poziții în contextul setului de date `gapminder_all` în care afișăm pentru anii 1989, 2000 și 2018 speranța medie de viață pentru fiecare țară.

```
# initial
gapminder_all %>%
  filter(year %in% c("1989", "2000", "2018")) %>%
  ggplot(aes(x = year, y = lifeExp)) +
  geom_point()

# position nundge
gapminder_all %>%
  filter(year %in% c("1989", "2000", "2018")) %>%
  ggplot(aes(x = year, y = lifeExp)) +
  geom_point(position = position_nudge(x = 0.25, y = 0))

# position jitter - default
gapminder_all %>%
  filter(year %in% c("1989", "2000", "2018")) %>%
  ggplot(aes(x = year, y = lifeExp)) +
  geom_point(position = "jitter")
```

```
#position jitter - cu specificatii
gapminder_all %>%
  filter(year %in% c("1989", "2000", "2018")) %>%
  ggplot(aes(x = year, y = lifeExp)) +
  geom_point(position = position_jitter(width = 0.05, height = 0.5))
```



Pentru mai multe informații referitoare la tipurile de ajustare a pozițiilor în obiectele grafice din `ggplot2` se poate consulta documentația fiecărei funcții `geom` sau se poate apela comanda

```
help.search("position_", package = "ggplot2")
```

4.8 Sisteme de coordonate (*coordinate systems*)

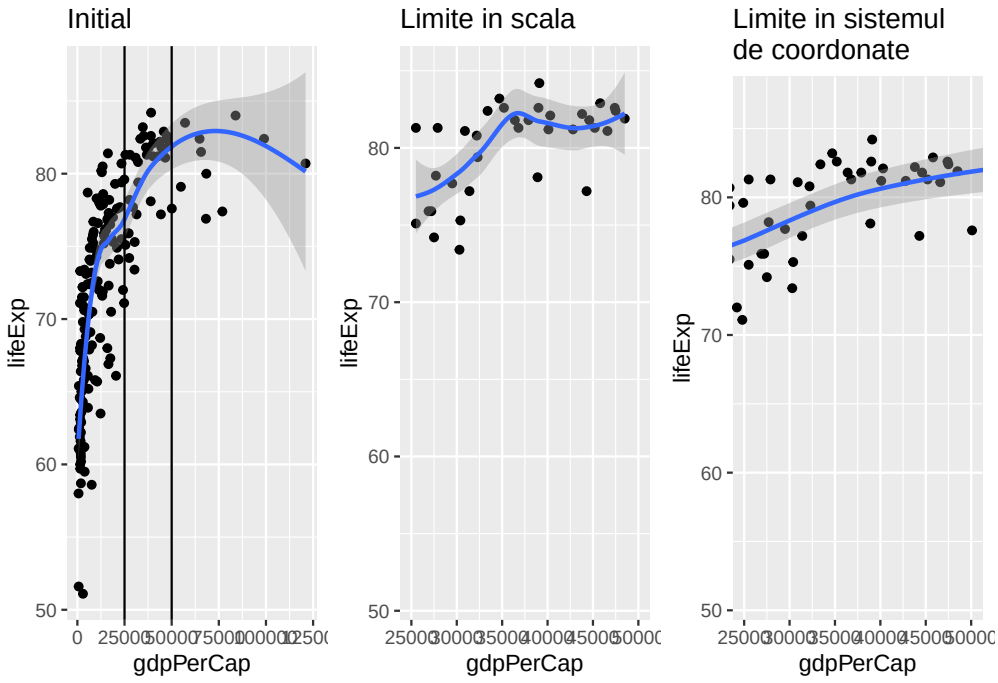
Pachetul `ggplot2` pune la dispiziție și o serie de funcții care fac referire la sistemul de coordonate în care sunt trasate elementele grafice. Sistemele de coordonate sunt specificate prin funcții care încep prin `coord_` și sunt adăugate ca straturi la graficul existent. Pachetul dispune de mai multe sisteme de coordonate printre care enumerăm:

- `coord_cartesian` - sistemul cartezian care este și cel de default și în care trebuie specificare valorile pe x și y
- `coord_flip` - sistemul cartezian în care axele sunt schimbate între ele
- `coord_fixed` - un sistem cartezian care păstrează fixat aspectul (*aspect ratio* - numărul de unități pe axa y care corespund unei unități pe axa x) cu raportul de bază de 1
- `coord_polar` - sistemul de coordonate polar
- `coord_quickmap` - un sistem de coordonate care aproximează sistemul sferic al planeti în vederea trasării hărților

Folosind sistemul cartezian cu opțiunea `xlim` (respectiv `ylim`) putem vizualiza doar acea regiune a graficului care ne interesează (putem face zoom). Diferența majoră dintre utilizarea limitelor în interiorul funcției `coord_cartesian` și utilizarea limitelor în funcțiile de scală (e.g. `scale_x_continuous`) este că în primul caz utilizăm tot setul de date pe când în cel de al doilea caz sunt folosite doar observațiile care se află între

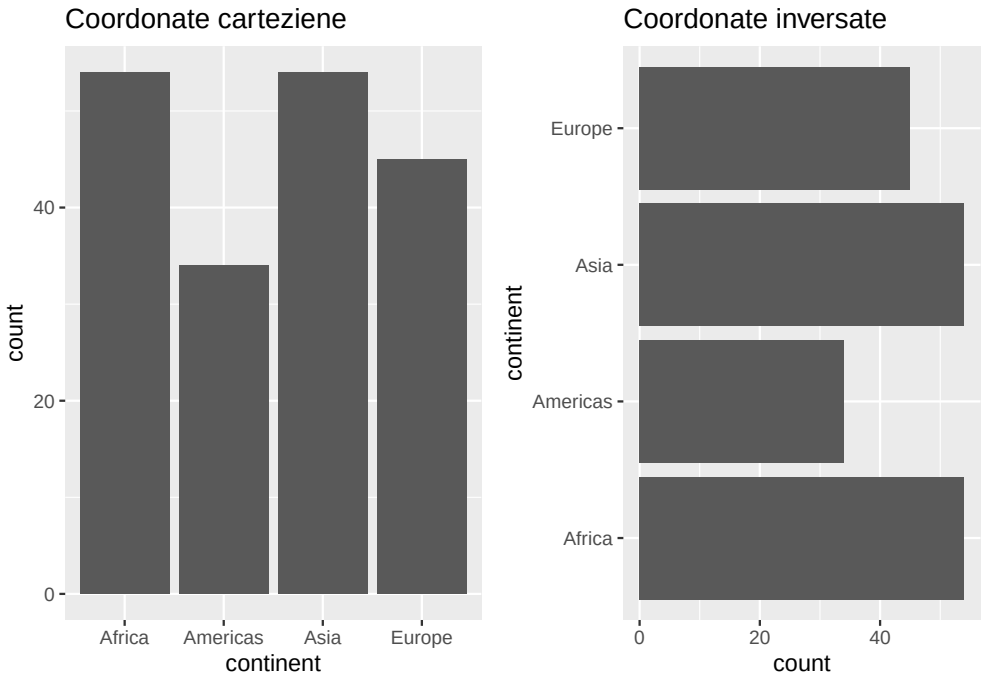
limitele setate, toate celelalte puncte fiind excluse.

```
# folosind scala - fitam curba de regresie pe datele ramase
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point() +
  geom_smooth() +
  scale_x_continuous(limits = c(25000, 50000))

# folosind sistemul de coordonate - fitam curba de regresie
# pe toate datele si evidentiem doar zona de interes
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point() +
  geom_smooth()+
  coord_cartesian(xlim = c(25000, 50000))
```

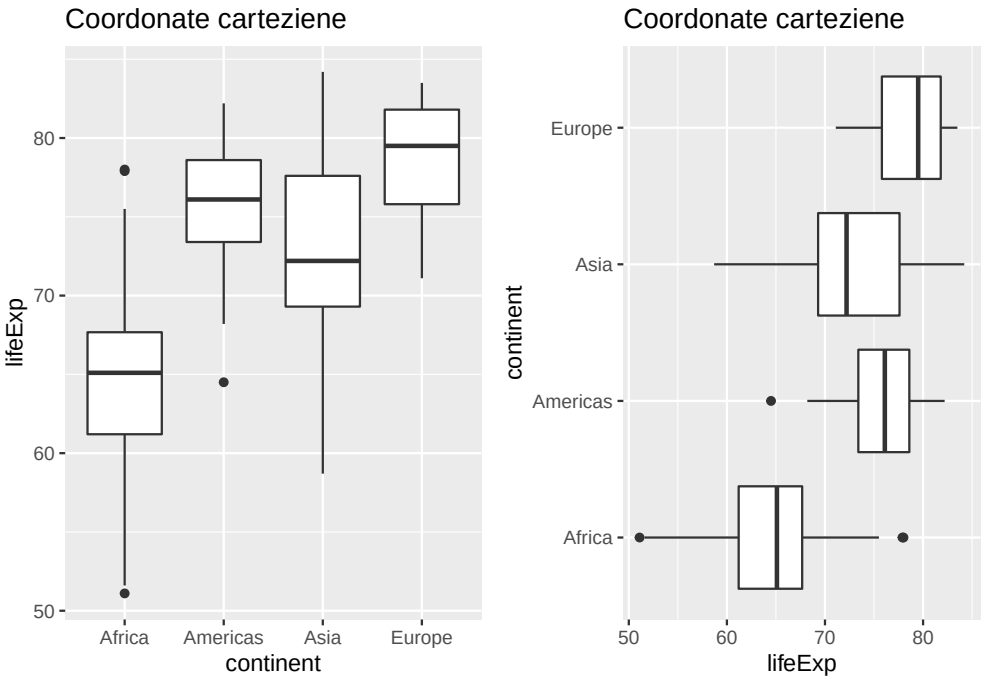


Sunt multe situațiile în care suntem interesați de valorile variabilei de pe axa x condiționate la variabila de pe axa y și în acest caz este recomandată schimbarea (întoarcerea) sistemului de coordonate prin apelarea funcției `coord_flip()`. De asemenea, o altă situație în care se recomandă rotirea coordonatelor cu 90 de grade este în cazul trasării unei diagrame cu bare sau a unui boxplot în funcție de o variabilă calitativă cu un număr mare de categorii sau cu categorii a căror denumire este lungă. Vom ilustra această transformare a sistemului de coordonate folosind diagrama cu bare pentru a evidenția numărul țărilor din setul de date `gapminder_2018` de pe fiecare continent.

```
# coordonate carteziene
gapminder_2018 %>%
  ggplot(aes(x = continent)) +
  geom_bar()

# coordonate flip
gapminder_2018 %>%
  ggplot(aes(x = continent)) +
  geom_bar() +
  coord_flip()
```



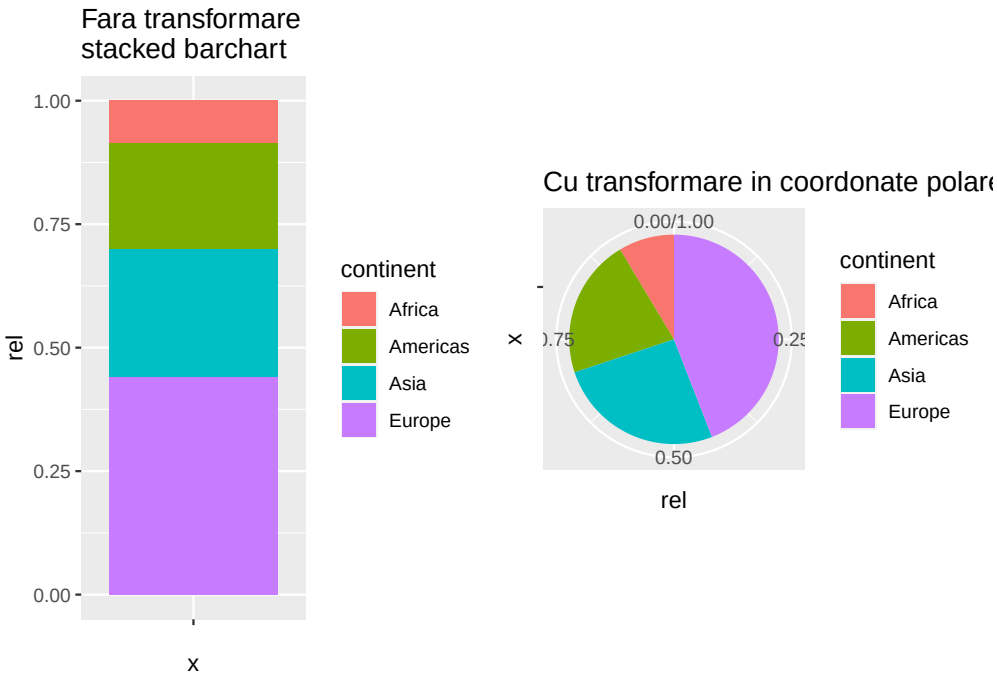
În mod similar putem vizualiza prin intermediul unui boxplor diferențele dintre duratele medii de viață de pe fiecare continent:



Un alt mod de vizualizare a datelor calitative (categoriale), frecvent utilizat în practică, este reprezentat de diagrama circulară (pie chart). Trebuie menționat că acest tip de grafic nu este recomandat având în vedere că oamenii tind să supraestimeze unghiurile mai mari de 90 de grade și să subestimeze unghiurile mai mici de 90 de grade astfel fiind dificilă compararea mărimii relative a două sectoare din diagrama circulară (Robbins 2013). Pentru a crea o diagramă circulară în `ggplot2` vom folosi transformarea în coordonate polare `coord_polar`. Ideea este de a pleca de la o diagramă cu bare suprapuse (*stacked barchart*) și apoi să o transpunem în coordonate polare.

Figura de mai jos prezintă o diagramă circulară pentru proporția țărilor de pe fiecare continent care depășesc produsul intern brut median global.

```
gapminder_2018 %>%
  mutate(gdp_avg = median(gdpPerCap)) %>%
  filter(gdpPerCap > gdp_avg) %>%
  mutate(n_all = n()) %>%
  group_by(continent) %>%
  summarize(rel = n() / unique(n_all)) %>%
  ggplot(aes(x = "", y = rel)) +
  geom_col(aes(fill = continent)) +
  coord_polar("y")
```



Mai multe detalii despre sistemele de coordonate se pot găsi în lucrarea de referință (Wickham 2016).

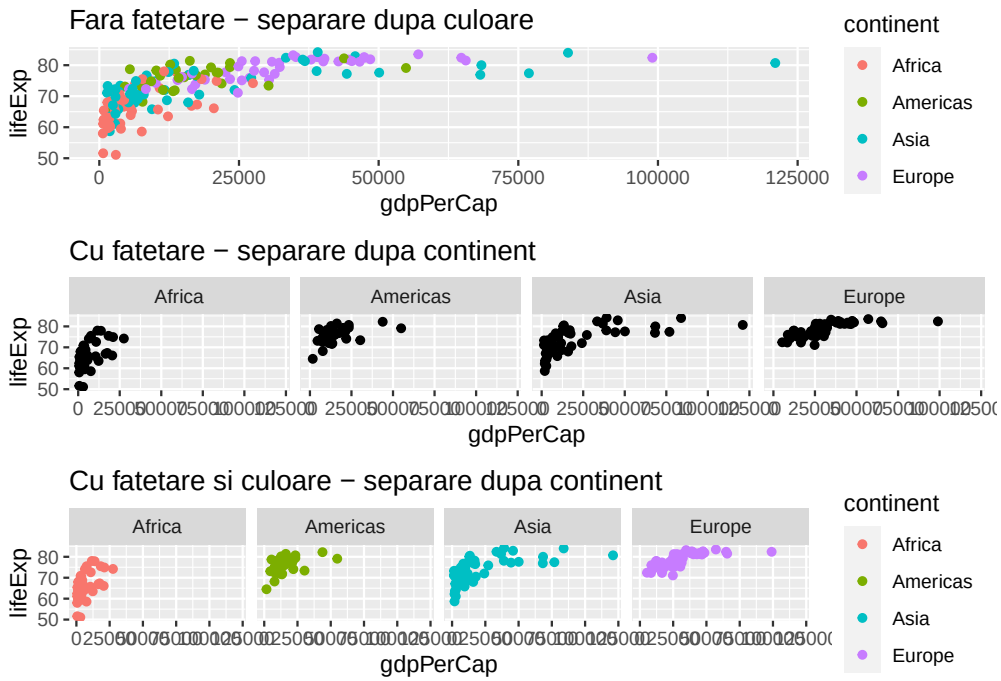
4.9 Fațetare (*facets*)

Procesul de *fațetare* (*facets*) presupune divizarea unei figuri în subfiguri determinate de valorile unei variabile categoriale sau a mai multor variabile categoriale. Astfel, în cazul în care vorbim de o singură varaibilă, pentru fiecare categorie a variabilei este trasat un subgrafic. Pachetul `ggplot2` pune la dispoziție două funcții utile pentru a efectua această operație:

- `facet_wrap()` - folosită atunci când avem de-a face cu o singură varaibilă de grupare a datelor
- `facet_grid()` - folosită atunci când dorim secționarea datelor după două sau mai multe variabile

Spre exemplu, setul de date `gapminder_2018` conține informații referitoare la produsul intern brut ajustat (`gdpPerCap`) și speranța medie de viață (`lifeExp`) pentru mai mult de 180 de țări de pe patru continente (`continents`). Dacă dorim să vedem relația dintre cele două variabile în funcție de continent atunci una dintre variante ar fi să folosi estetica `colour` pentru a diferenția. O variantă alternativă, care uneori ilustrează mai clar diferențele, este de a subsecționa graficul în subgrafice ce corespund fiecărui continent folosind funcția `facet_grid()`.

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent)) +
  geom_point()

ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point() +
  facet_grid(.~continent)
```

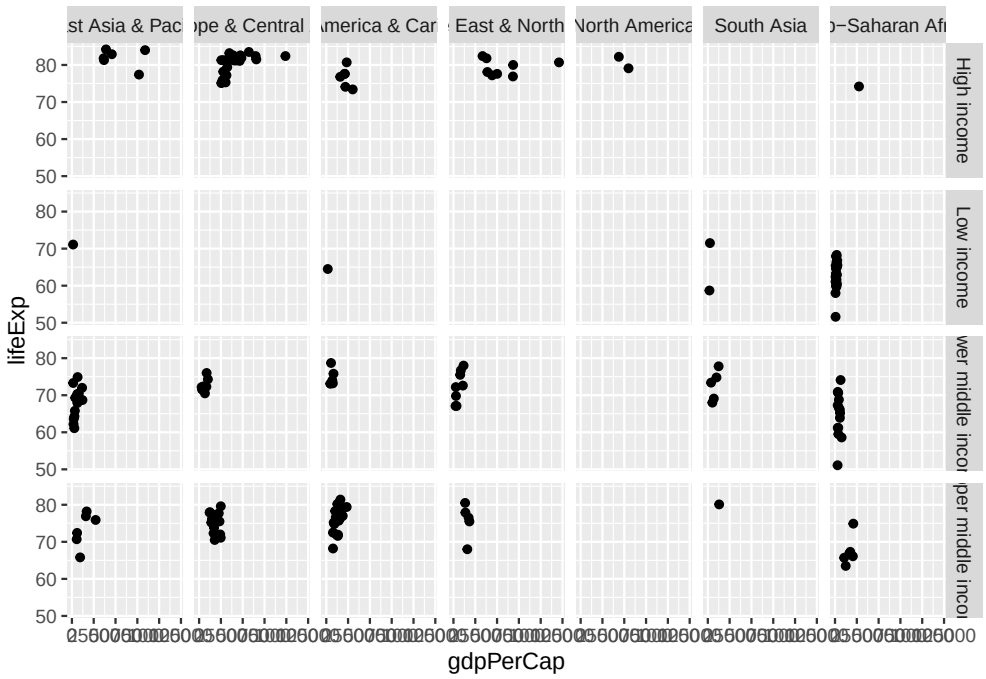


În figura de mai sus am folosit `facet_grid` pentru a separa în subgrafice chiar dacă separarea s-a făcut în funcție de o singură variabilă (*continent*). Observăm că în apelarea funcției `facet_grid` (`?facet_grid`) am folosit simbolul `~` care separă variabila/variabilele din stânga (trasată/trasate de-a lungul liniilor) de cea/cele din dreapta (trasată/trasate pe coloane). Codul generic pentru funcția `facet_grid` este

```
facet_grid([factor pentru linii] ~ [factor pentru coloane])
```

Pentru ilustrare să considerăm că ne dorim să investigăm relația dintre `gdpPerCap` și `lifeExp` în funcție de regiunea și nivelul de venit stabilite de Banca Mondială (variabilele `wb_region` și respectiv `wb_income`).

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp)) +
  geom_point() +
  facet_grid(wb_income ~ wb_region)
```



Funcția `facet_wrap()` este folosită cu precădere atunci când dorim secționarea datelor după o singură variabilă calitativă (dar poate fi folosită și cu mai multe - `?facet_wrap`). Prin aplicarea acestei funcții, subgraficele corespunzătoare fiecărui nivel al varaibilei de separare nu trebuie să fie repartizate pe o singură linie sau coloană în schimb se poate specifica numărul de coloane (sau de linii) dorite. Codul generic pentru funcția `facet_wrap` este

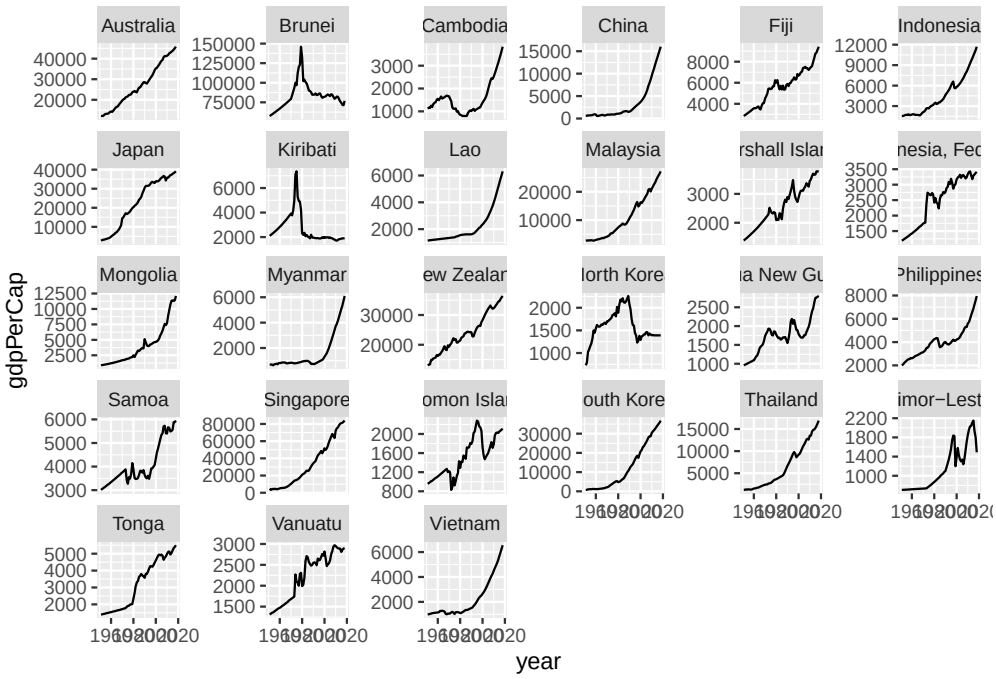
```
facet_wrap( ~ [variabila de fatetare],
            ncol = [numar de coloane])
```

Vom ilustra aplicarea acestei funcții în contextul setului de date `gapminder_all`. Vom afișa cum a evoluat evoluția duratei medii de viață pentru patru țări (*Statele Unite*, *UK*, *Franța* și *România*) după Cel de-al Doilea Război Mondial:

```
gapminder_all %>%
  mutate(year = as.numeric(year)) %>%
  filter(country %in% c("United States", "France", "United Kingdom", "Romania"),
         year > 1945) %>%
  ggplot(aes(x = year, y = lifeExp)) +
  geom_line() +
  facet_wrap(~country,
             ncol = 2)
```

În figura de mai jos ilustrăm evoluția produsului intern brut pentru statele din Asia de Est după anul 1950. Opțiunea `ncol` descrie numărul de coloane în care vrem să fie împărțit grid-ul de subgrafice iar opțiunea `scales = "free_y"` permite ajustarea scalei pentru axa y pentru fiecare categorie (țară) în parte.

```
gapminder_all %>%
  mutate(year = as.numeric(year)) %>%
  filter(six_regions == "east_asia_pacific",
         year > 1950) %>%
  ggplot(aes(x = year, y = gdpPerCap)) +
  geom_line()+
  facet_wrap(~country,
             ncol = 6, scales = "free_y")
```



4.10 Customizarea graficelor

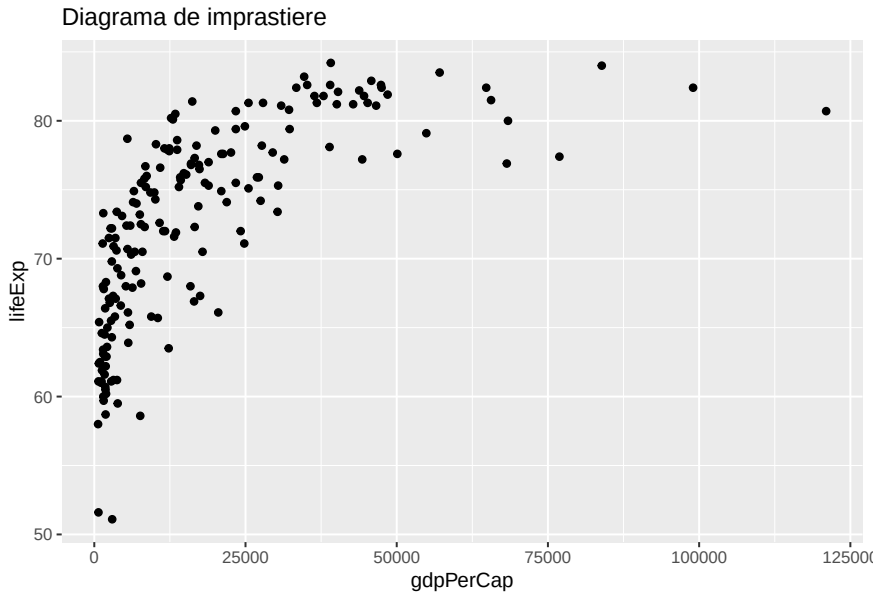
În această secțiune vom prezenta o serie de metode și tehnici de personalizare/customizare a graficelor atât prin adăugarea de elemente clasice precum titlu, etichetă pentru axe, etc. cât și prin modificarea temelor grafice.

4.10.1 Personalizarea elementelor grafice (axe, titlu, etc.)

Titlurile, etichetele axelor și adnotările textuale (pe grafic, axe, obiecte geometrice și legendă) reprezintă un aspect important în crearea unei figuri prin care vrem să transmitem o serie de informații. Pachetul `ggplot2` pune la dispoziție mai multe funcții care permit efectuarea adnotării elementelor grafice fără mare dificultate.

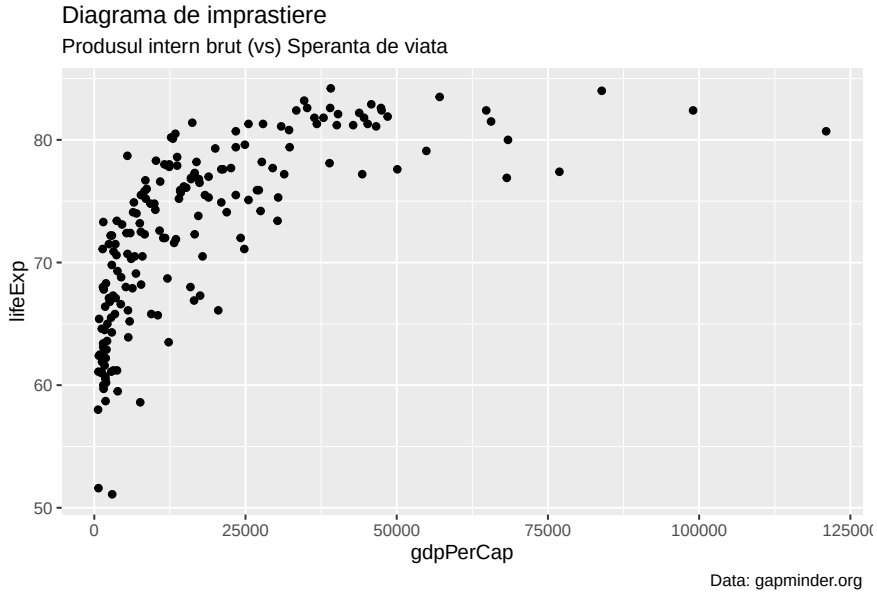
Astfel putem adauga/modifica titlul unui grafic folosind comanda `ggtitle([titlu])`:

```
p = ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp))+
  geom_point()

p + ggtitle("Diagrama de imprastiere")
```



sau alternativ putem utiliza funcția `labs(title = [titlu])` care în plus permite adăugarea unui subtitlu (`subtitle`) si a unei surse/credit (`caption`):

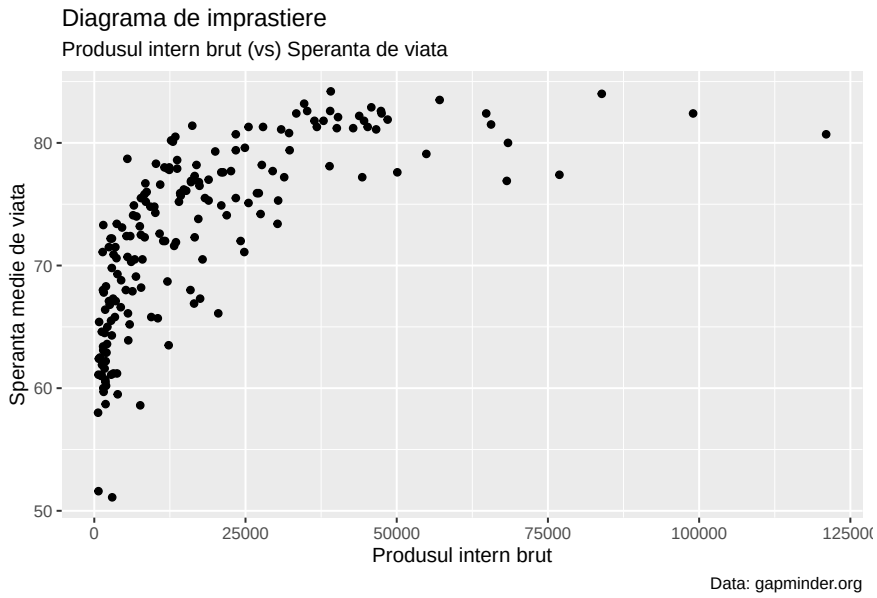
```
p +
  labs(title = "Diagrama de imprastiere",
       subtitle = "Produsul intern brut (vs) Speranta de viata",
       caption = "Data: gapminder.org")
```



Pentru modificarea etichetelor axelor de coordonate putem folosi sau funcțiile `xlab()` pentru axa x și respectiv `ylab()` pentru axa y sau comanda `labs(x = [eticheta axei x], y = [eticheta axei y])`.

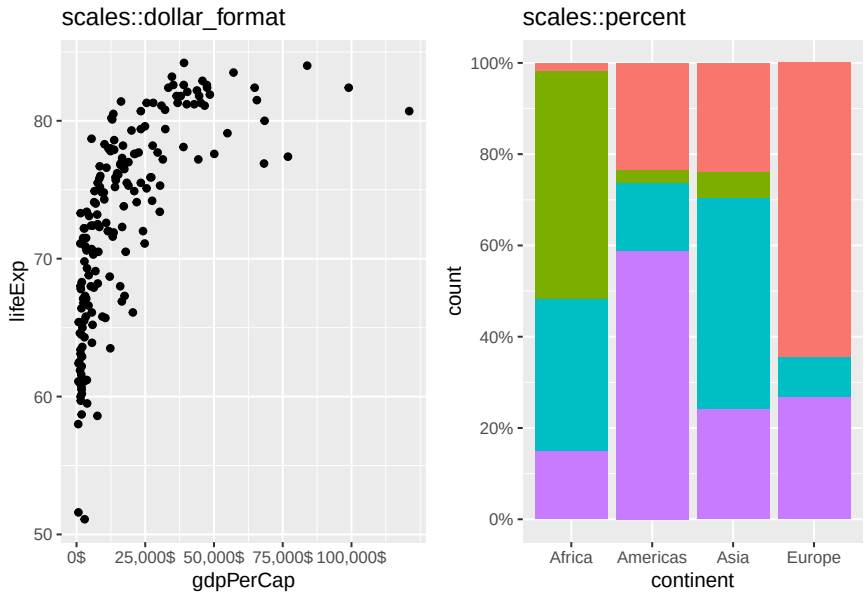
```
p +
  labs(title = "Diagrama de imprastiere",
       subtitle = "Produsul intern brut (vs) Speranta de viata",
       caption = "Data: gapminder.org",
       x = "Produsul intern brut",
```

```
       y = "Speranta medie de viata")
```



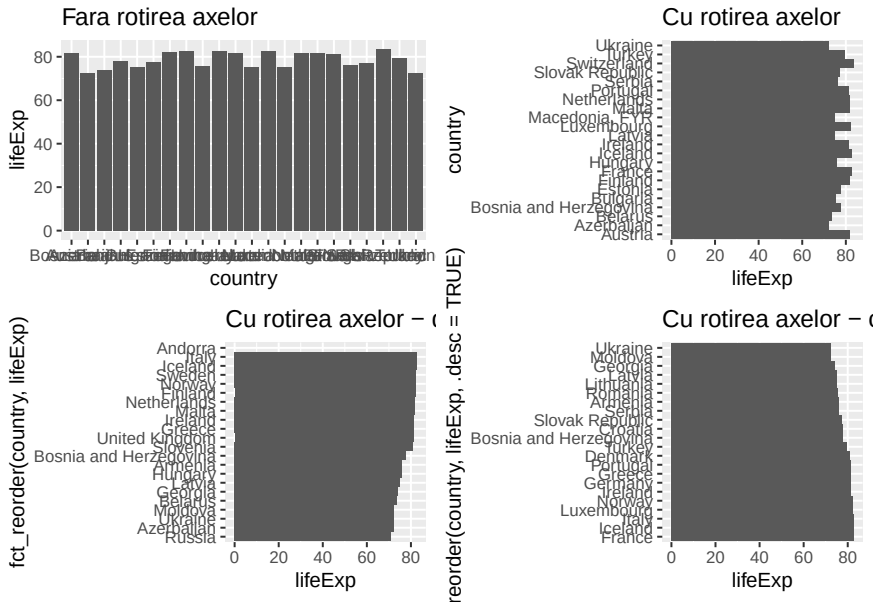
De asemenea, am văzut că putem modifica/adăuga nume axelor și prin specificarea acestuia parametrului `name` într-o scală (e.g. `scale_x_continuous`). Atunci când situația impune este recomandat să folosim unităle de măsură ale valorilor variabilelor. Pachetul `scales`, https://scales.r-lib.org/, pune la dispoziție o serie de funcții ajutătoare, de exemplu atunci când unitățile de măsură sunt `$` sau `%`:

```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp))+
  geom_point() +
  scale_x_continuous(labels = scales::dollar_format(prefix = "", suffix = "$")) +
  ggtitle("scales::dollar_format")

gapminder_2018 %>%
  ggplot(aes(x = continent, fill = wb_income)) +
  geom_bar(position = "fill") +
  scale_y_continuous(breaks = seq(0, 1, by = .2), labels = scales::percent)+
  ggtitle("scales::percent")
```



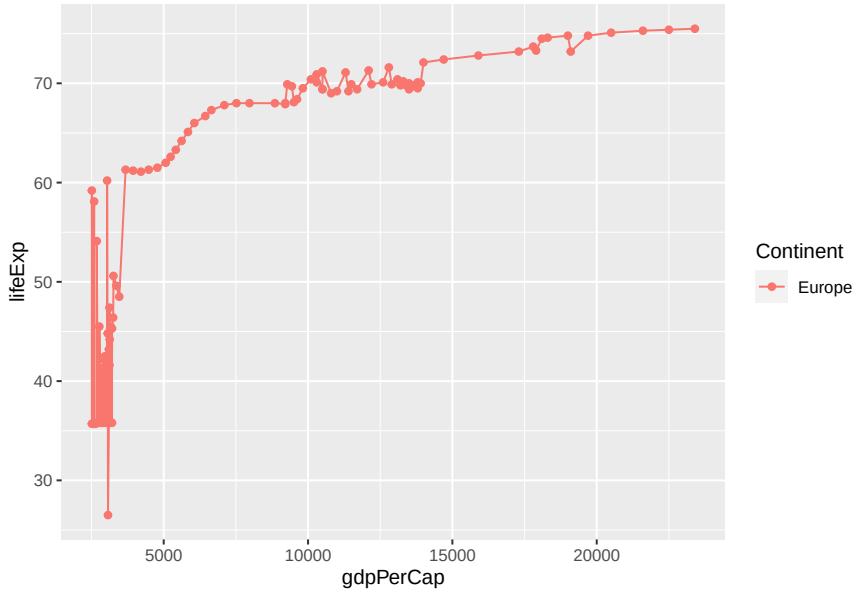
Totodată se pot întâlni situații în care axa x necesită etichete de lungime mai lungă, conducând astfel la o supraaglomerare a acestora, și în acest caz este recomandată rotirea axelor (`coord_flip()` - poate și ordonarea nivelelor `fct_reorder()`).

```
set.seed(1234)

gapminder_2018 %>%
  filter(continent == "Europe") %>%
  sample_frac(0.5) %>%
  ggplot(aes(x = country, y = lifeExp)) +
  geom_bar(stat = "identity") +
  ggtitle("Fara rotirea axelor")

set.seed(1234)

gapminder_2018 %>%
  filter(continent == "Europe") %>%
  sample_frac(0.5) %>%
  ggplot(aes(x = country, y = lifeExp)) +
  geom_bar(stat = "identity") +
  ggtitle("Cu rotirea axelor") +
  coord_flip()
```



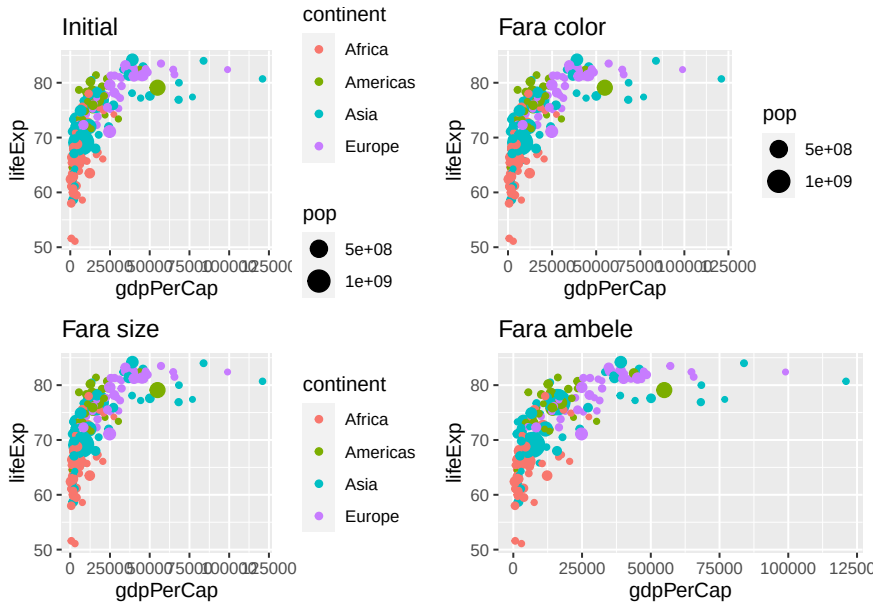
Un alt element grafic important în crearea unei figuri este legenda. Am văzut că în `ggplot2` legenda este creată automat atunci când folosim un element estetic precum `colour`, `fill`, `size`, `shape`, etc. După cum am văzut, legendele și axele sunt cele două elemente de ghidaj (*guides*) care pot fi modificate prin intermediul funcțiilor de scalare. Spre deosebire de axe, legendele sunt elemente mai complexe deoarece, pe lângă faptul că pot fi poziționate în diverse locuri, ele pot afișa mai multe estetici simultan (e.g. `colour`, `shape`) provenite de la mai multe straturi geometrice.

```
gapminder_all %>%
  filter(country == "Romania",
         as.numeric(year) > 1900) %>%
ggplot(aes(x = gdpPerCap, y = lifeExp, colour = four_regions)) +
  geom_point() +
  geom_line() +
  # schimbam titlul legendei
  labs(colour = "Continent")
```



Atunci când vrem să eliminăm legenda sau stratul care nu vrem să apară în legendă avem la dispoziție mai multe variante: folosim funcția `guides([nume estetica] = FALSE)`, funcția `scale_[nume estetică]_[tip](guide = FALSE)` sau argumentul `show.legend = FALSE` în elementul geometric:

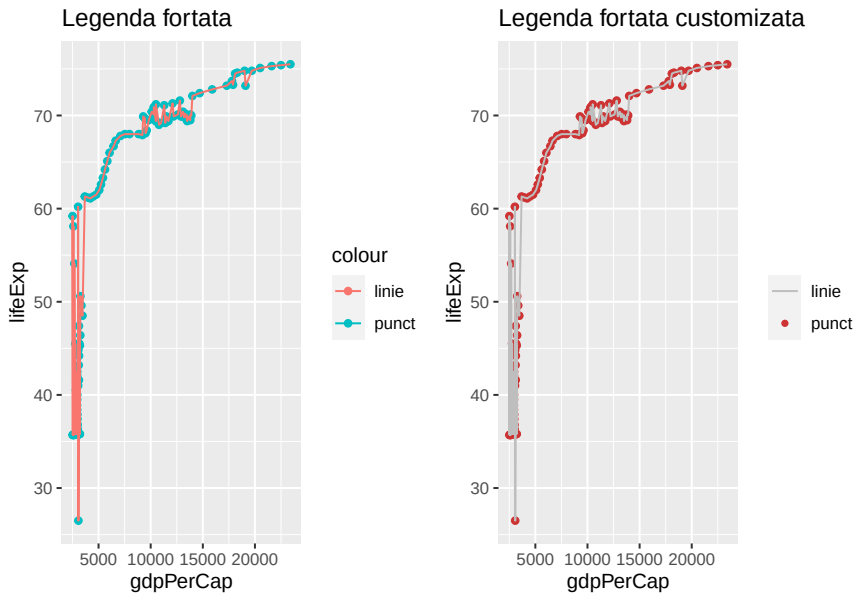
```
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent, size = pop)) +
  geom_point() +
  ggtitle("Initial")

ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent, size = pop)) +
  geom_point() +
  scale_color_discrete(guide = FALSE) +
  ggtitle("Initial")

ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent, size = pop)) +
  geom_point() +
  guides(size = FALSE) +
  ggtitle("Initial")
```



Pachetul `ggplot2` nu va adăuga o legendă în mod automat decât dacă avem o corespondență între un element estetic (`colour`, `size`, etc.) și o variabilă dar sunt multe situațiile în care dorim să evidențiem elementele din grafic printr-o legendă. Putem *forța* apariția unei legende prin maparea în interiorul obiectului geometric (atribuirea) a unei estetici într-o valoare fixată. Mai mult, în cazul în care dorim modificarea aspectului elementelor geometrice din legendă putem folosi funcția `guide_legend()` cu opțiunea `override.aes`.

```
gapminder_all %>%
  filter(country == "Romania",
         as.numeric(year) > 1900) %>%
ggplot(aes(x = gdpPerCap, y = lifeExp)) +
  geom_point() +
  geom_line() +
  labs(title = "Initial")

gapminder_all %>%
  filter(country == "Romania",
         as.numeric(year) > 1900) %>%
```

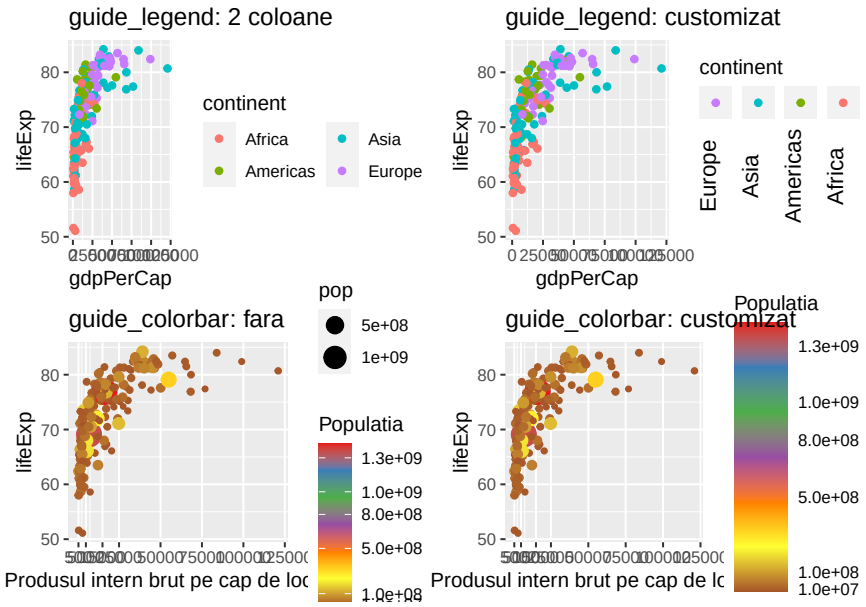
```
ggplot(aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(aes(colour = "punct")) +
  geom_line(aes(colour = "linie")) +
  labs(title = "Legenda fortata")

gapminder_all %>%
  filter(country == "Romania",
         as.numeric(year) > 1900) %>%
ggplot(aes(x = gdpPerCap, y = lifeExp)) +
  geom_point(aes(colour = "puncte")) +
  geom_line(aes(colour = "linie")) +
  # modificam culorile
  scale_color_manual("", guide = "legend",
                     values = c("puncte" = "brown3",
                                "linie" = "gray")) +
  # vrem sa apara doar linie si punct
  guides(color = guide_legend(override.aes = list(linetype = c(1, 0),
                                                  shape = c(NA, 16)))) +
  labs(title = "Legenda fortata customizata")
```



Funcțiile ajutătoare `guide_legend()` și `guide_colorbar()` oferă un control suplimentar asupra detaliilor de prezentare ale unei legende, astfel prima poate fi aplicată atât elementelor estetice discrete cât și celor continue pe când cea de-a doua este aplicată doar elementelor estetice continue (este folosită pentru reprezentarea gamelor continue de culori). Pentru utilizarea acestor funcții, ele trebuie atribuite parametrului `guide` (i.e. `guide = guide_legend(...)`) din funcția de scală corespunzătoare esteticii pe care dorim să o modificăm sau mai simplu în interiorul funcției ajutătoare `guides` (i.e. `guides([estetică] = guide_legend(...)`).

```
# guide_legend
# pe doua coloane
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent)) +
  geom_point() +
  guides(colour = guide_legend(ncol = 2))
# customizata
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = continent)) +
  geom_point() +
```

```
  guides(colour = guide_legend(reverse = TRUE,
                               direction = "horizontal",
    title.position = "top",
    label.position = "bottom",
    label.hjust = 0.5,
    label.vjust = 1,
    label.theme = element_text(angle = 90)))

# guide_colorbar
ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = pop, size = pop)) +
  geom_point() +
  scale_x_continuous(name = "Produsul intern brut pe cap de locuitor",
                     breaks = c(500, 5000, 15000, 25000,
                                50000, 75000, 100000, 125000),
                     minor_breaks = c(500, 2500, 7500)) +
  scale_colour_distiller(palette = "Set1",
                         breaks = c(1e7, 1e8, 5e8, 8e8, 1e9, 1.3e9),
                         name = "Populatia")

ggplot(gapminder_2018, aes(x = gdpPerCap, y = lifeExp, colour = pop, size = pop)) +
  geom_point() +
  scale_x_continuous(name = "Produsul intern brut pe cap de locuitor",
                     breaks = c(500, 5000, 15000, 25000,
                                50000, 75000, 100000, 125000),
                     minor_breaks = c(500, 2500, 7500)) +
  scale_colour_distiller(palette = "Set1",
                         breaks = c(1e7, 1e8, 5e8, 8e8, 1e9, 1.3e9),
                         name = "Populatia") +
  guides(colour = guide_colorbar(barwidth = 2,
                                 barheight = 10,
                                 ticks = FALSE),
         size = FALSE)
```

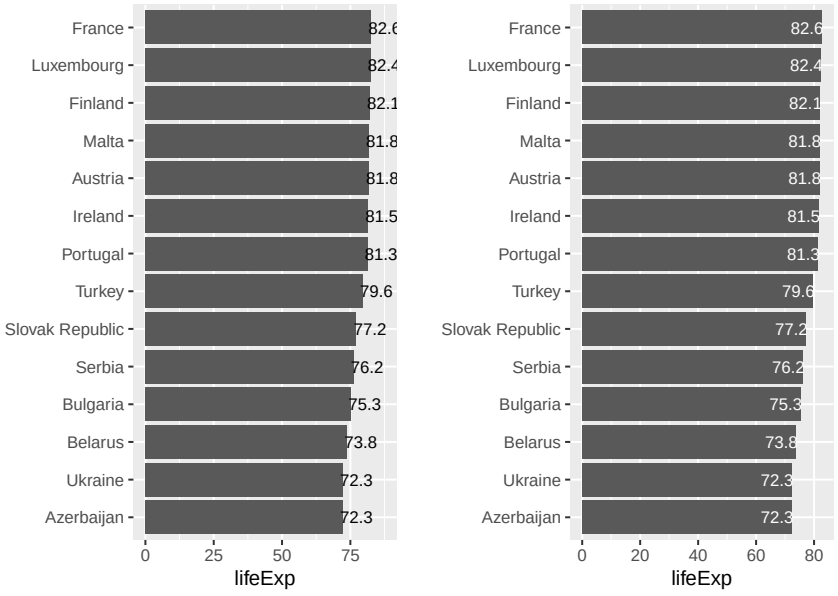


4.10.2 Adăugarea elementelor textuale la grafic

Sunt multe situații (e.g. în cazul unor prezentări, rapoarte, proiecte, etc.) în care pentru a face un grafic mai expresiv/ilustrativ este necesară adăugarea de elemente textuale explicative. Aceste elemente pot fi adăugate de cele mai multe ori folosind funcțiile `geom_text`, `geom_label`, `annotate` sau funcțiile corespunzătoare din pachetul `ggrepel`. Vom prezenta mai jos câteva exeple de grafice în care adăugarea de elemente textuale facilitează înțelegerea mesajului transmis de figură.

Pentru început vom folosi un barplot orizontal în care dorim să adăugăm marcaje în dreptul fiecărei bare:

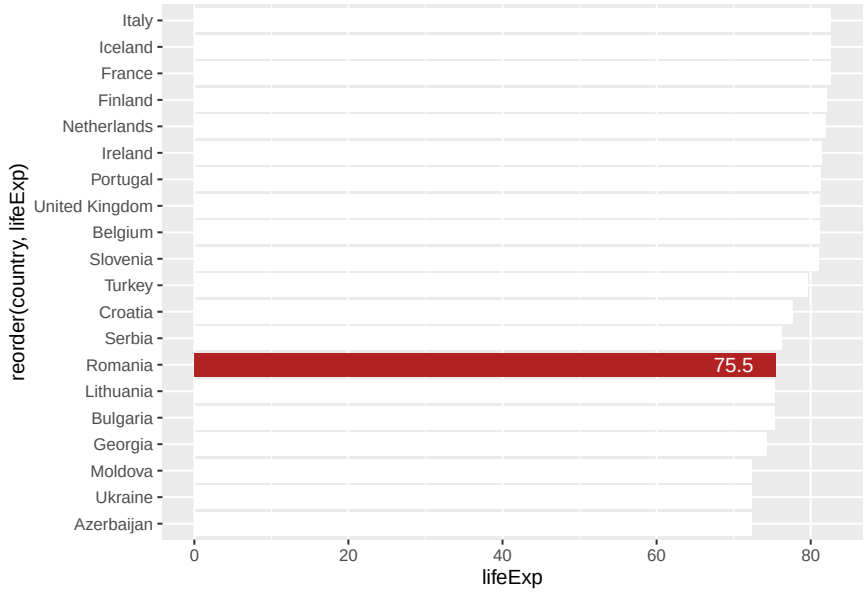
```
set.seed(1234)

baza = gapminder_2018 %>%
  filter(continent == "Europe") %>%
  sample_frac(0.3) %>%
  ggplot(aes(x = fct_reorder(country, lifeExp), y = lifeExp)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(x = "")

baza +
  geom_text(aes(label = lifeExp),
            size = 3, nudge_y = 5)

baza +
  geom_text(aes(label = lifeExp), size = 3,
            nudge_y = - 5, color = "white")
```



În cazul în care dorim să atragem atenția doar asupra unei bare atunci:

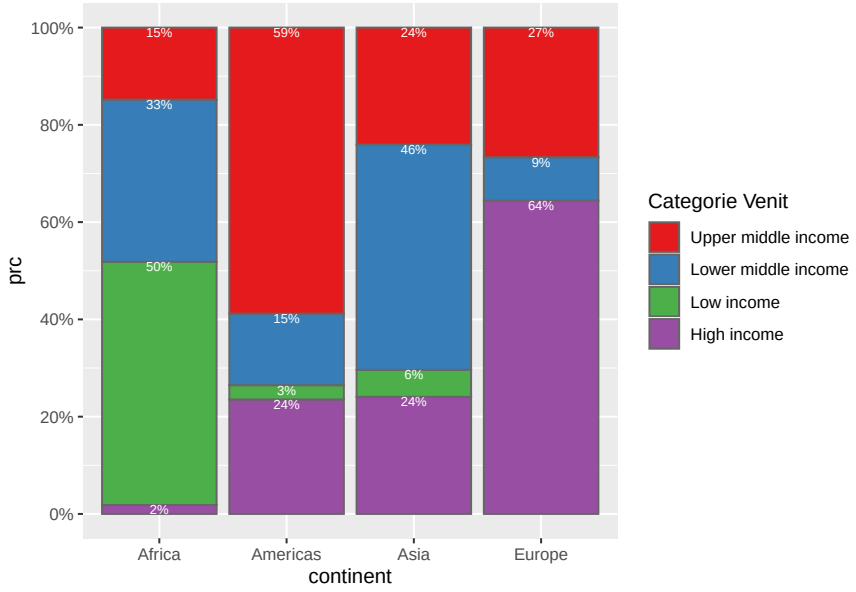
```
countries = c("Moldova", "Georgia", "Italy", "Croatia", "Slovenia",
              "Ukraine", "Azerbaijan", "Romania", "Turkey", "Macedonia",
              "Belgium", "France", "Finland", "Iceland", "Netherlands",
              "Lithuania", "Serbia", "Portugal", "Ireland", "Bulgaria",
              "United Kingdom")
```

```
baza_gap = gapminder_2018 %>%
  filter(country %in% countries) %>%
  mutate(ID = ifelse(country == "Romania", TRUE, FALSE))

ggplot(baza_gap, aes(x = reorder(country, lifeExp), y = lifeExp, fill = ID)) +
        geom_bar(stat = "identity") +
        coord_flip() +
        scale_fill_manual(values = c("white", "firebrick"),
                          guide = FALSE) +
        annotate("text", x = "Romania", y = 70, label = "75.5", color = "white")
```



Pentru a adăuga etichete la o diagramă cu bare proporțională (pentru care avem ajustată poziția la `fill`) trebuie să creăm o nouă variabilă care să stocheze locația (suma cumulativă a proporțiilor):

```
# pregatim setul de date
gapminder_2018_bf = gapminder_2018 %>%
  group_by(continent, wb_income) %>%
  # cate elemente am din fiecare wb_income pentru
  # fiecare continent in parte
  tally() %>%
  # grupez dupa continent
  group_by(continent) %>%
  # determin proportia
  mutate(prc = n / sum(n),
         label_y = cumsum(prc))


gapminder_2018_bf %>%
  ggplot(aes(x = continent, y = prc, fill = fct_rev(wb_income))) +
  geom_bar(stat = "identity", color = "grey40") +
  scale_y_continuous(breaks = seq(0, 1, by = .2), labels = scales::percent) +
  geom_text(aes(label = scales::percent(prc), y = label_y),
            vjust = 1, color = "white", size = 2.5) +
  scale_fill_manual(values = c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3")) +
  labs(fill = "Categorie Venit")
```



4.10.3 Teme grafice

Putem schimba întregul aspect al graficului prin modificarea *temei* (funcția `theme()`) acestuia. Temele reprezintă o modalitate consistentă de a customiza/personaliza elementele grafice ale unei figuri, elemente care nu fac referire la setul de date (titlu, etichetele axelor, mărimea și tipul fontului, culoarea de fundal, liniile de marcaj - gridlines, legendele, etc.).

Pentru a utiliza o temă trebuie să adăugăm unui obiect de tip `ggplot` o funcție de temă care are forma generală `theme_<nume temă>`. Pachetul `ggplot2` vine cu o serie de teme predefinite printre care menționăm: `theme_grey`, `theme_linedraw`, `theme_bw`, `theme_minimal`, `theme_void`, `theme_dark`, `theme_classic`, `theme_light`. Figura de mai jos prezintă fiecare dintre aceste teme pentru setul de date `gapminder_2018`:

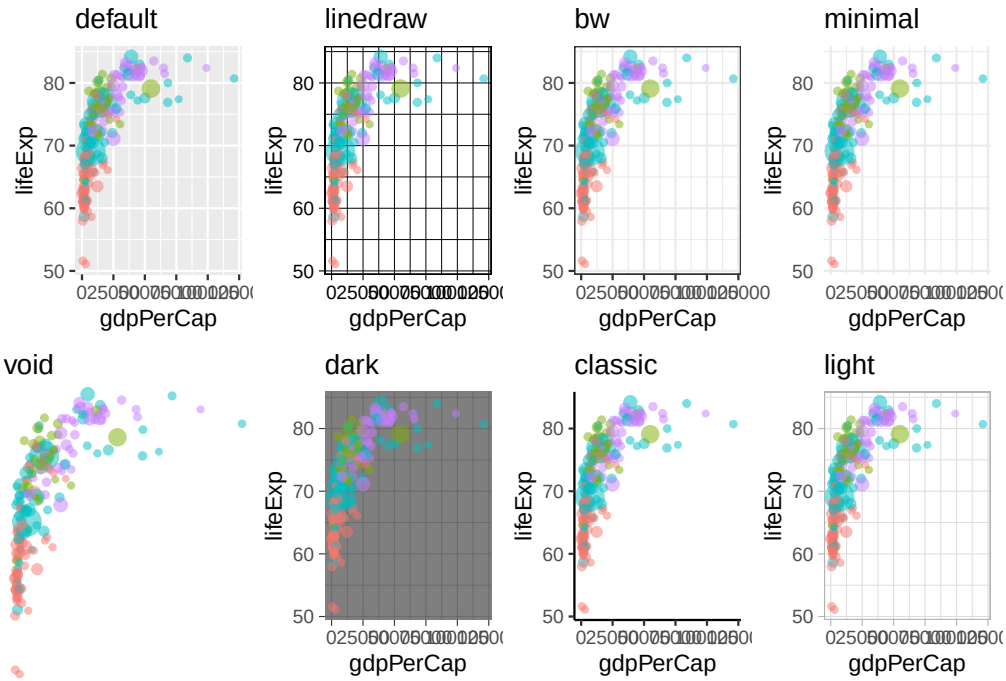


Fig. 6: Ilustrare a temelor predefinite

Pe lângă temele predefinite în `ggplot2` se pot folosi și teme din pachetul `ggthemes` dezvoltat de Jeffrey Arnold (pentru mai multe detalii vizitați site-ul ggthemes) care cuprinde mai mult de 20 de astfel de teme unele fiind reprezentări fidele ale tematicilor grafice folosite de reviste sau programe consacrate (de exemplu *Economist* sau *Excel*). Mai jos sunt prezentate doar patru dintre acestea:

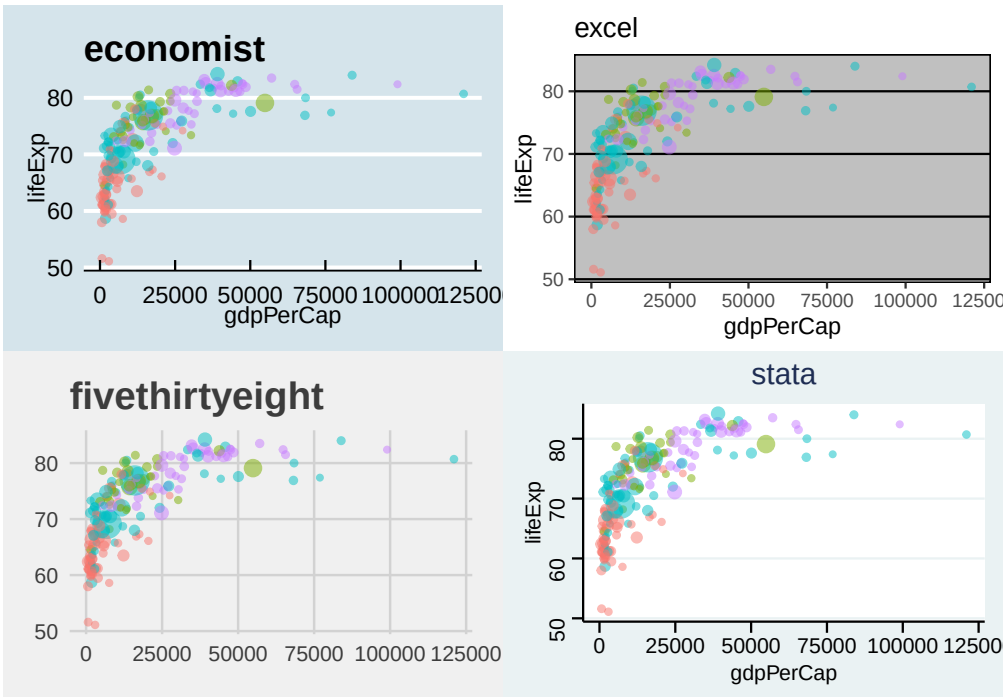


Fig. 7: Ilustrare a temelor din pachetul ggthemes

Pentru a modifica elementele unei temei vom folosi funcția `theme()`. Această funcție ne permite controlul (aproape total) asupra elementelor grafice care nu țin de setul de date dar prin intermediul cărora figura devine mai expresivă: marimea fontului, culoarea de fundal, fontul și culoarea etichetelor axelor și ale legendei, etc.

Structura generală este `theme([element al temei] = element_[nume functie asociată]([proprietati schimbate]))` unde

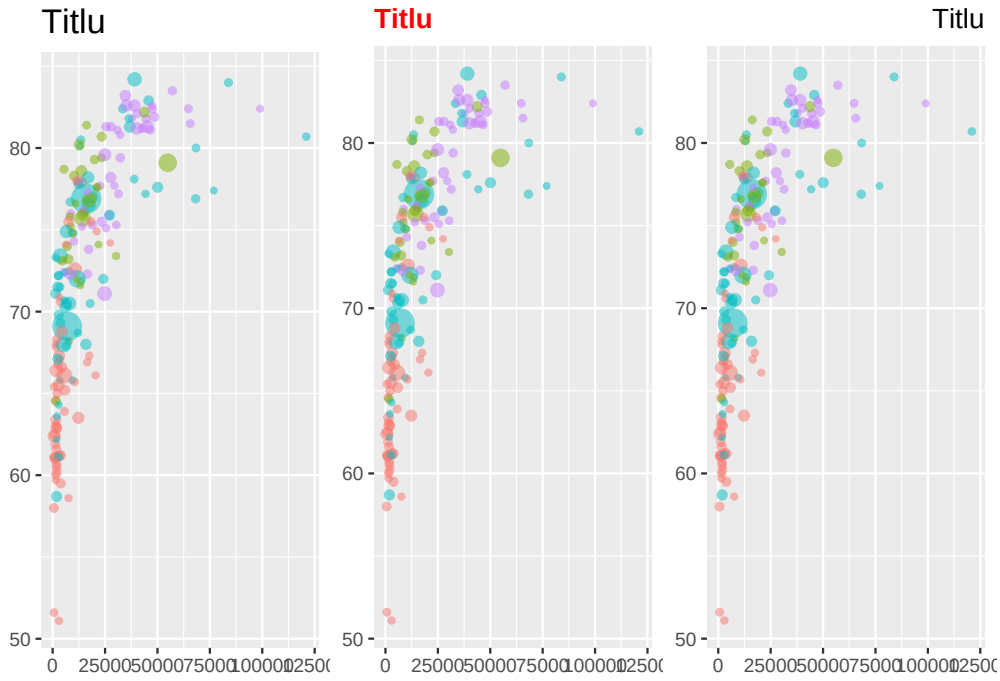
- `[element al temei]` face referire la un element grafic precum `plot.title` (controleaza modul de afișare a titlului graficului), `axis.ticks.x` (elemente de marcaj de pe axa x), `legend.key.height` (înălțimea elementelor din legendă) etc. Pentru a vizualiza toate elementele grafice ale unei teme ce pot fi modificate se poate consulta documentația https://ggplot2.tidyverse.org/reference/theme.html
- `element_[nume functie asociată]` reprezintă o funcție asociată elementelor temei care descrie caracteristicile vizuale ale elementului (toate elementele au asociate o astfel de funcție). Sunt patru astfel de funcții: `element_text`, `element_line`, `element_rect` și `element_blank`.
- `[proprietati schimbate]` reprezintă caracteristicile vizuale ce urmează să fie modificate, i.e. `font`, `family`, `linetype`, `colour`, `size`, etc.

Tabelul de mai jos prezintă o serie de caracteristici vizuale pentru fiecare funcție asociată elementelor unei teme grafice:

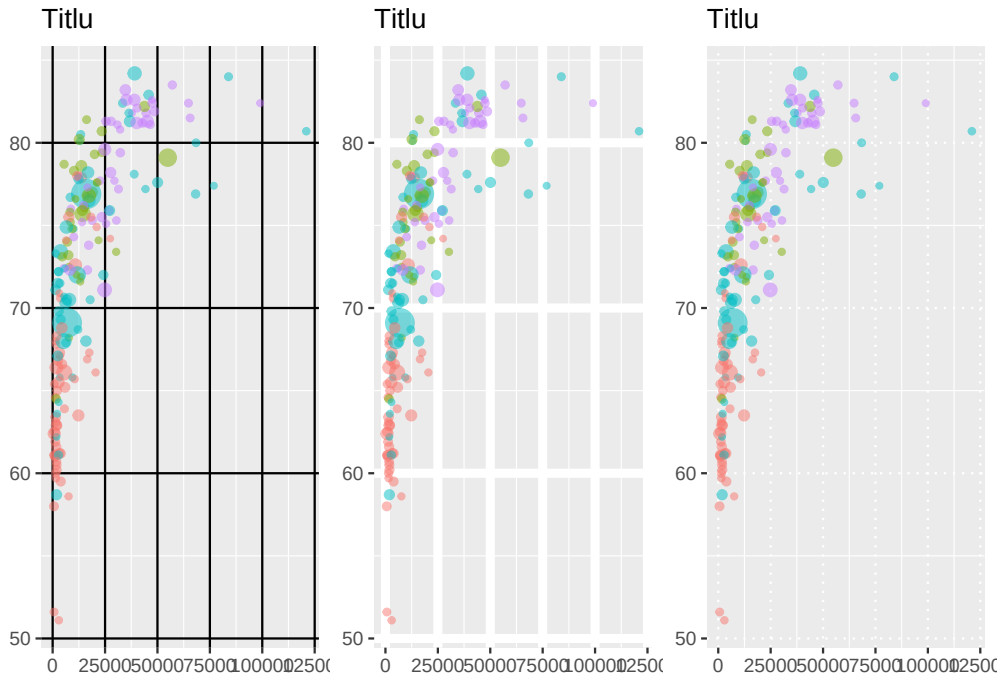
Funcție	Caracteristici
`element_text()`	`family`, `face`, `colour`, `size` (în points), `hjust`, `vjust`, `angle` (în grade), `lineheight`
`element_line()`	`colour`, `size`, `linetype`, `lineend`
`element_rect()`	`fill`, `colour`, `size`, `linetype`
`element_blank()`	

Exemple:

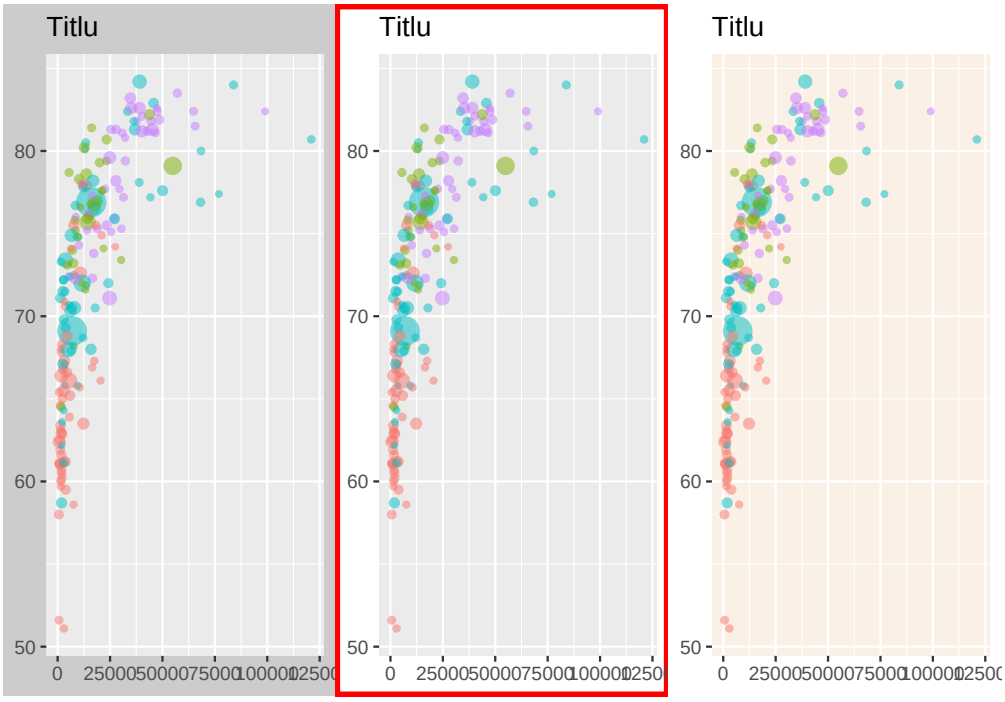
```
baza = ggplot(gapminder_2018) +
  geom_point(aes(x = gdpPerCap, y = lifeExp, color = continent, size = pop),
             alpha = 0.5, show.legend = FALSE)

baza <- baza + labs(title = "Titlu") + xlab(NULL) + ylab(NULL)
baza + theme(plot.title = element_text(size = 16))
baza + theme(plot.title = element_text(face = "bold", colour = "red"))
baza + theme(plot.title = element_text(hjust = 1))
```




```
baza + theme(panel.grid.major = element_line(colour = "black"))
baza + theme(panel.grid.major = element_line(size = 2))
baza + theme(panel.grid.major = element_line(linetype = "dotted"))
```



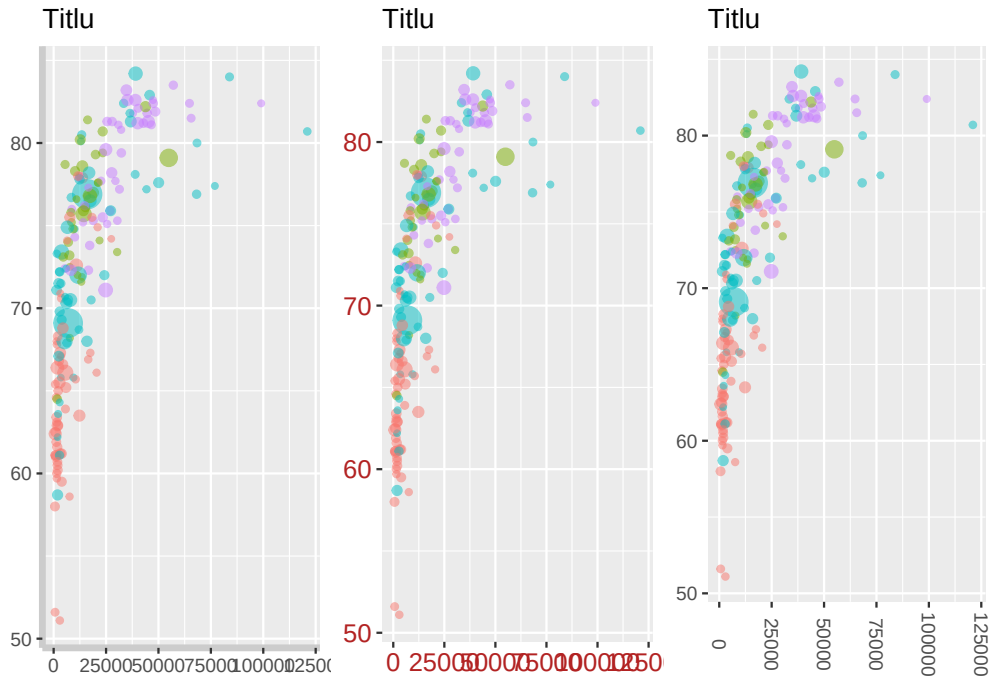

```
baza + theme(plot.background = element_rect(fill = "grey80", colour = NA))
baza + theme(plot.background = element_rect(colour = "red", size = 2))
baza + theme(panel.background = element_rect(fill = "linen"))
```



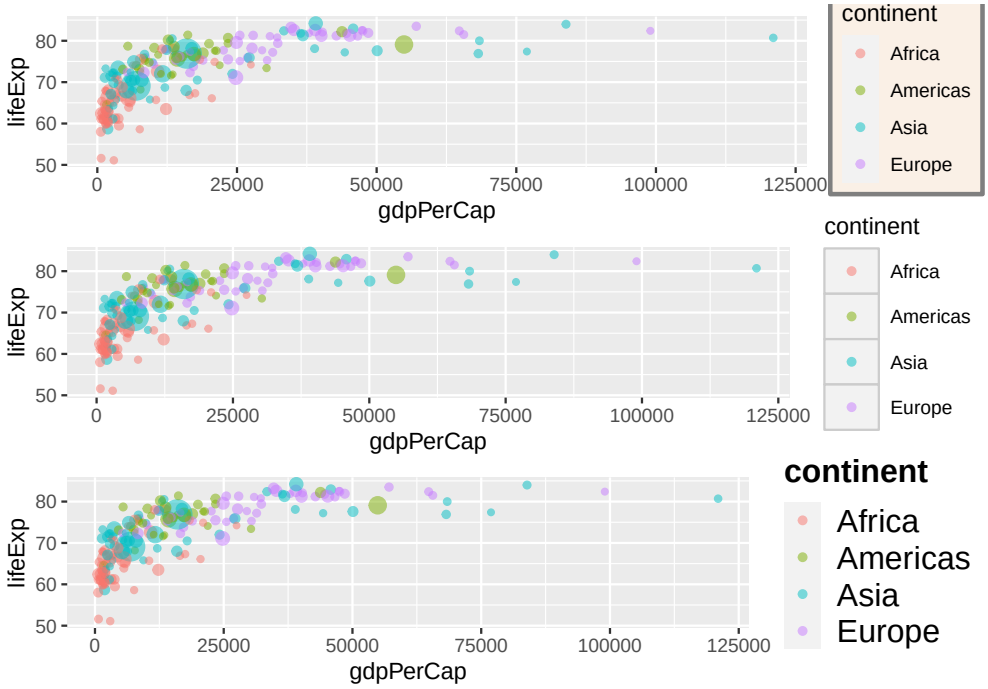
Mai jos regăsim o serie de elemente care permit controlul vizual al axelor și legendelor împreună cu funcțiile asociate acestora:

Obiect	Element	Setter	Description
axe	axis.line	`element_line()`	drepte paralele cu axele (nu se văd implicit)
	axis.text	`element_text()`	etichetele axelor (x și y)
	axis.text.x	`element_text()`	etichetele axei x
	axis.text.y	`element_text()`	etichetele axei y
	axis.title	`element_text()`	titlurile axelor
	axis.title.x	`element_text()`	tiltlul axei x
	axis.title.y	`element_text()`	tiltlul axei y
	axis.ticks	`element_line()`	elementele de marcaj ale axelor
	axis.ticks.length	`unit()`	lungimea elementelor de marcaj
legendă	legend.background	`element_rect()`	fundalul legendei
	legend.key	`element_rect()`	fundalul elementelor legendei
	legend.key.size	`unit()`	marimea elementelor legendei
	legend.key.height	`unit()`	înălțimea elementelor legendei
	legend.key.width	`unit()`	lățimea elementelor legendei
	legend.margin	`unit()`	marginile legendei
	legend.text	`element_text()`	etichetele legendei
	legend.text.align	0–1	alinierea etichetelor legendei (0 = dreapta, 1 = stânga)
	legend.title	`element_text()`	titlul legendei
	legend.title.align	0–1	alinierea titlului legendei (0 = dreapta, 1 = stânga)

```
baza + theme(axis.line = element_line(colour = "grey80", size = 1.5))
baza + theme(axis.text = element_text(color = "firebrick", size = 12))
baza + theme(axis.text.x = element_text(angle = -90, vjust = 0.5))
```




```
baza2 = ggplot(gapminder_2018) +
  geom_point(aes(x = gdpPerCap, y = lifeExp, color = continent, size = pop),
             alpha = 0.5) +
  guides(size = "none")

baza2 + theme(
  legend.background = element_rect(
    fill = "linen",
    colour = "grey50",
    size = 1
  )
)
baza2 + theme(
  legend.key = element_rect(color = "grey80"),
  legend.key.width = unit(0.9, "cm"),
  legend.key.height = unit(0.75, "cm")
)
baza2 + theme(
  legend.text = element_text(size = 15),
  legend.title = element_text(size = 15, face = "bold")
)
```



Pentru a mai multe detalii și exemple privind elementele grafice ale unei teme ce pot fi modificate se poate consulta documentația https://ggplot2.tidyverse.org/reference/theme.html.

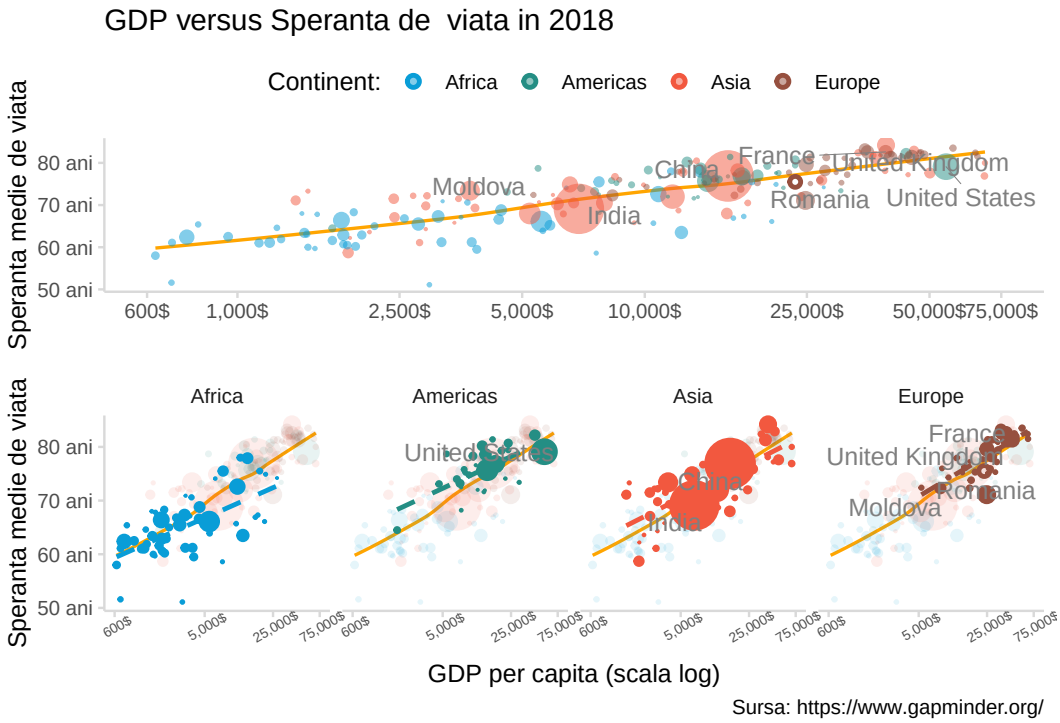
4.11 Exemple folosind setul de date `gapminder`

Noțiunile prezentate în acest capitol, puse cap la cap, permit generarea figurii de la începutul capitolului. Mai jos este prezentat codul folosit pentru generarea figurii (pentru anul 2018):

```
ggplot(gapminder_2018) +
  # adaugam curba de regresie loess
  geom_smooth(aes(x = gdpPerCap, y = lifeExp),
              se = FALSE, method = "loess",
              color = "orange", size = 0.7, alpha = 0.7) +
  # adaugam puncte fara Romania
  geom_point(data = gapminder_2018 %>% filter(country != "Romania"),
             aes(x = gdpPerCap, y = lifeExp, color = continent, size = pop),
             alpha = 0.5) +
  # adaugam Romania si o facem de forma diferita - cerc gol
  geom_point(data = gapminder_2018 %>% filter(country == "Romania"),
             aes(x = gdpPerCap, y = lifeExp, color = continent, size = pop),
             shape = 1,
             stroke = 1.5)+
  # adaugam text pentru o serie de state (ggrepel)
  geom_text_repel(aes(x = gdpPerCap, y = lifeExp, label = country),
            color = "grey50",
            segment.size = 0.2,
            data = gapminder_2018 %>%
              filter(pop > 1e9 | country %in% c("Moldova",
                                                "United States",
                                                "Romania",
                                                "United Kingdom",
```

```
                                                "France"))) +
# schimbam scala pe x in scala logaritmica
scale_x_log10(limits = c(600, 75000),
              breaks = c(600, 1000, 2500, 5000, 10000,
                         25000, 50000, 75000),
              label = scales::dollar_format(prefix = "", suffix = "$", accuracy = 1)) +
# adaugam ani la scala de pe y
scale_y_continuous(label = function(x) {return(paste(x, "ani"))}) +
# schimbam etichetele si adaugam titlu
labs(title = "GDP versus Speranta de  viata in 2018",
     caption = "Sursa: https://www.gapminder.org/",
     x = "GDP per capita (scala log)",
     y = "Speranta medie de viata",
     size = "Populatia",
     color = "Continent") +
# schimbam marimea scalei
scale_size(range = c(0.1, 10),
           # stergem legenda pentru size
           guide = "none") +
# adaugam culori customizate pentru continente
scale_color_manual(name = "",
                   values = c("#099DD7",
                              "#248E84",
                              "#F2583F",
                              "#96503F"),
                   guide = guide_legend(nrow = 1, order=1)) +
# schimbam tema
theme_minimal() +
# plasam legenda sus si scoatem gridul
theme(legend.position = "top",
      axis.line = element_line(color = "grey85"),
      axis.ticks = element_line(color = "grey85"),
      panel.grid.major = element_blank(),
      panel.grid.minor = element_blank())
```

Cu un pic mai mult efort se poate construi figura de mai jos care în plus evidențiază pentru fiecare continent în parte tendința (dreapta de regresie) corespunzătoare:



Sursa: https://www.gapminder.org/

Pentru cel de-al doilea grafic avem

```
countries = c("Romania", "France", "Germany", "United Kingdom", "Italy", "Greece",
              "United States", "Canada", "Brazil", "Mexico",
              "China", "India", "Vietnam",
              "Ethiopia", "South Africa", "Nigeria",
              "Australia", "New Zealand")

gapminder_life_exp_diff <- gapminder_all %>%
  mutate(year = as.integer(year),
         continent = four_regions) %>%
  # filtram anii de start si de final
  filter(year == 2000 | year == 2018) %>%
  # ne asiguram ca datele sunt aranjate astfel ca anul 2000 sa fie inaintea lui 2018
  arrange(country, year) %>%
  # pentru fiecare tara adaugam variabila care ne da diferenta sperantei de viata
  group_by(country) %>%
  mutate(lifeExp_diff = lifeExp[2] - lifeExp[1]) %>%
  ungroup() %>%
  # aranjam in ordinea diferentelor cele mai mari
  arrange(lifeExp_diff) %>%
  # restrangem la tarile selectate
  filter(country %in% countries) %>%
  select(country, year, continent, lifeExp, lifeExp_diff)


gapminder_life_exp_diff %>%
  mutate(country = fct_inorder(country)) %>%
  # pentru fiecare tara definim minimul si maximul sperantei de viata
  group_by(country) %>%
  mutate(max_lifeExp = max(lifeExp),
```

```
        min_lifeExp = min(lifeExp)) %>%
  ungroup() %>%
  ggplot() +
  # trasam segmentele
  geom_segment(aes(x = min_lifeExp, xend = max_lifeExp,
                   y = country, yend = country,
                   col = continent), alpha = 0.5, size = 5) +
  # adaugam punctele de capat
  geom_point(aes(x = lifeExp, y = country, col = continent), size = 8,
             shape = 21, fill = "white", stroke = 2) +
  # adaugam elementele textuale
  geom_text(aes(x = min_lifeExp + 0.47, y = country,
                label = paste(country, " ", round(min_lifeExp))),
            col = "grey50", hjust = "right") +
  geom_text(aes(x = max_lifeExp - 0.4, y = country,
                label = round(max_lifeExp)),
            col = "grey50", hjust = "left") +
  # delimitam axa x
  scale_x_continuous(limits = c(45, 85)) +
  # alegem culorile
  scale_color_manual(values = c("#099DD7",
                                "#248E84",
                                "#F2583F",
                                "#96503F")) +
  # stabilim titlul si axele
  labs(title = "Schimbarea speranței de viață",
       subtitle = "Între anii 2000 și 2018",
       x = "Speranța de viață (în 2000 și 2018)",
       y = NULL,
       col = "Continent: ") +
  # folosim tema clasica
  theme_classic() +
  # eliminam axele si pozitionam legenda
  theme(legend.position = "top",
        axis.line = element_blank(),
        axis.ticks = element_blank(),
        axis.text = element_blank())
```

Mai mult dacă dorim să ilustrăm evoluția speranței de viață în Europa evidențiind pentru fiecare țară în parte parcurul atunci putem obține un grafic de forma

```
# setul de background
gap_bg = gapminder_all %>%
  mutate(continent = four_regions,
         year = as.integer(year)) %>%
  filter(continent == "Europe",
         year > 1900) %>%
  rename(country2 = country)

# capetele
gap_bg_endpoints = gap_bg %>%
  filter(year == 2018) %>%
  rename(country = country2)
```

```
## tarile
gap_bg_countries = gap_bg %>%
  group_by(country2) %>%
  filter(year == min(year)) %>%
  ungroup() %>%
  mutate(year = 1900,
         country = country2,
         lifeExp = max(gap_bg$lifeExp, na.rm = TRUE) - 2)

# constructia graficului
gapminder_all %>%
  mutate(continent = four_regions,
         year = as.integer(year)) %>%
  filter(continent == "Europe",
         year > 1900) %>%
ggplot(aes(x = year, y = lifeExp)) +
  # trasarea curbelore de background
  geom_line(data = gap_bg,
            aes(x = year, y = lifeExp, group = country2),
            size = 0.15, color = "gray80", alpha = 0.5)+
  # trasarea liniilor evidentiate
  geom_line(aes(group = country),
            color = "royalblue",
            lineend = "round") +
  # adaugarea punctelor la final
  geom_point(data = gap_bg_endpoints,
             aes(x = year, y = lifeExp),
             size = 1.1,
             shape = 21,
             color = "royalblue",
             fill = "royalblue") +
  # adaugarea tarilor - stanga sus
  geom_text(data = gap_bg_countries,
            mapping = aes(label = country),
            vjust = "inward",
            hjust = "inward",
            fontface = "bold",
            size = 2.1) +
  # fatetarea dupa country reordonat
  facet_wrap(~ reorder(country, - lifeExp), ncol = 5)+
  # redenumirea axelor
  labs(x = "Perioada 1900 - 2018",
       y = "Speranta de viata",
       title = "Evolutia sperantei de viata in Europa",
       caption = "Sursa: https://www.gapminder.org/") +
  # customizarea temei
  theme_classic()+
  theme(plot.title = element_text(size = rel(1), face = "bold"),
        plot.caption = element_text(size = rel(1)),
        # scoatem titlurile fatetelor
        strip.text = element_blank(),
        panel.spacing.x = unit(-0.05, "lines"),
        panel.spacing.y = unit(0.3, "lines"),
```

```
        axis.text.y = element_text(size = rel(0.5)),
        axis.title.x = element_text(size = rel(1)),
        axis.title.y = element_text(size = rel(1)),
        axis.text.x = element_text(size = rel(0.5)),
        legend.text = element_text(size = rel(1)))
```

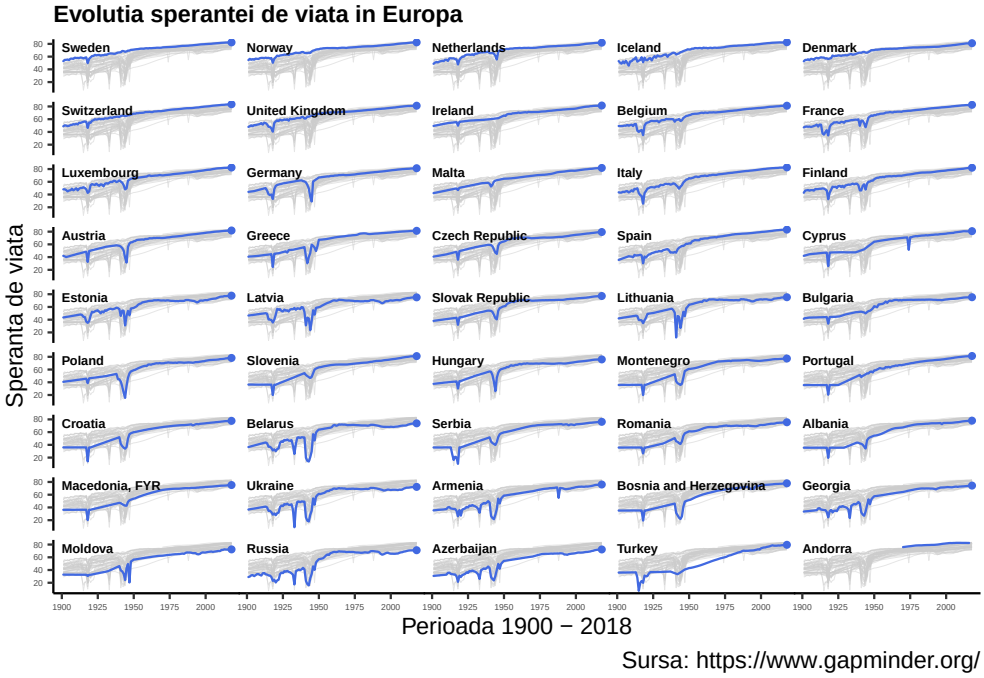


Fig. 8: Evolutia sperantei de viata in Europa dupa anul 1900

Atunci când dorim să salvăm un grafic putem folosi funcția `ggsave()` care prin comportamentul de bază salvează ultimul grafic generat. Funcția permite specificarea graficului pe care dorim să îl salvăm (`plot =`), locația și numele fișierului salvat (`filename =`), dimensiunea (`width =`, `height =`) precum și tipul acestuia (`png`, `jpeg`, `tiff`, `pdf`, `tex`, etc.).

De exemplu, pentru a salva figura generată de codul de mai sus (figura de la începutul capitolului) într-un fișier *pdf* în format *landscape* putem scrie

```
ggsave(filename = "grafic.pdf",
       width = 11, height = 8.5)
```

ținând cont că acesta este ultimul grafic generat, în caz contrar trebuie să specificăm pe care grafic dorim să îl salvăm. Astfel, dacă figura este atribuită elementului `p` (`p = ggplot(gapminder_2018) + ...`) atunci pentru a salva scriem

```
ggsave(filename = "grafic.pdf",
       plot = p,
       width = 11, height = 8.5)
```

Ex. 4.1

Încercați să recreați grafice similare cu cel de la începutul capitolului (sau cel de mai sus) pentru alte seturi de date ce pot fi descărcate de pe platforma www.gapminder.org/data/.

Referințe

Robbins, Naomi. 2013. *Creating More Effective Graphs*. First. New York, NY: Chart House.

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

———. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

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.

Wilkinson, Leland. 2005. *The Grammar of Graphics (Statistics and Computing)*. First. Secaucus, NJ: Springer-Verlag.