Laborator 2-1
Operații manipulare seturi de date

Import date

Pentru a prelua seturi de date care sunt disponibile într-un anumit format structurat R pune la dispoziție o serie de funcții pentru fiecare format. Pentru date în format "data" avem funcția read.table(), pentru formatul csv avem funcția read.csv() etc. Acestora le transmitem calea către fișier sub forma unui string de caractere – de aceea este nevoie să o încadrăm cu ghilimele. Pentru datele csv ce nu sunt separate de separatorul standard virgulă folosim read.csv2() . Daca fisierul sursa este plasat in directorul proiectului RStudio, atunci calea catre acesta poate fi omisa.

Comanda read.csv creaza un data frame.

```
#preluăm în variabila pop conținutul din fișierul populatie.csv
pop<-read.csv("C:/.../populatie.csv")
```

Puteți vizualiza conținutul setului de date direct în consolă scriind numele variabilei, pop, sau într-o fereastră separată în Rstudio cu ajutorul funcției **View**(), ce am mai întâlnit-o, sau a funcției **fix**() ce prezintă setul de date într-o fereastră spreadsheet (aceasta permite inclusiv editări ale datelor, adică putem să schimbăm valoarea din celule).

Variabilele conținute de setul de date nu vor putea fi referite în mod direct: dacă am dori să afișam conținutul variabilei denumire vom primi un mesaj de eroare. Acestea trebuie neapărat precedate de numele setului de date

```
#afișăm conținutul coloanei denumire
pop$denumire
```

Totuși, dacă dorim să facem referire la variabile folosind doar numele acestora putem folosi funcția attach()

```
#vizualizare valori din variabila denumire
attach(pop)
denumire
```

Operații de preluare/citire date tabelare sunt oferite ca funcții și în alte pachete de ex. pachetul ***readr***. Față de varianta oferită în distribuția standard aceste funcții din pachetul readr au avantajul că importă setul de date sub forma unei structuri tibble (în comparație cu structura de tip data frame anterioară).

Import pachete

Vom putea folosi funcțiile și obiectele din pachetul respectiv abia după ce îl vom încărca – această operație o vom efectua ori de câte ori avem nevoie să apelăm la conținutul pachetului. Asigurați-vă că aveți acest pachet încărcat (fie prin comanda library, fie bifați numele pachetului în ecranul Packages din colțul din dreapta jos din RStudio). Daca distributia de pe calculatorul dvs. nu are instalat un anume pachet, se poate folosi comanda install.packages(„nume pachet").

Pentru a lucra cu tibbles si a folosi o serie de functionalitati avansate referitoare la lucrul cu date avem nevoie de libraria tidyverse. Aceasta include si librariile

- readr: pentru citirea avansata de date. Se vor citi date din formate diverse, iar outputul va fi un tibble
- dplyr : pentru alte operatii avansate de lucru cu datele
- ggplot2 : o facilitate foarte puternica pentru vizualizarea datelor

```
install.packages(„tidyverse")
library(tidyverse)
```

În cazul în care nu se poate realiza încărcarea, probabil este nevoie să se instaleze pachetul. Aceasta operație o vom face o singură dată. Pentru a instala un packet în RStudio alegeți opțiunea Install din secțiunea Packages și e suficient să scriem numele pachetului sau tastați în consolă: install.packages(…)

La încărcarea pachetului tidyverse observați mesajele legate de posibile erori conflictuale: de exemplu că pachetul dplyr suprascrie anumite funcții din R. Dacă dorim să folosim varianta originală a acestor funcții după încărcarea dplyr atunci este nevoie să folosim numele complet a acestora stats::filter() sau stats::lag().

Pentru a citi date în format csv vom folosi funcția read_csv().

Dacă fișierul nostru folosește ; pentru separarea valorilor atunci vom apela la funcția read_csv2().

```
#preluăm în variabila pop conținutul din fișierul populatie.csv
pop <-read_csv("C:/.../populatie.csv")
```

Pe lângă argumentul obligatoriu dat de calea către fișierul cu date, funcția read_csv mai are și alte argumente ce pot fi specificate astfel încât să nu le rămână valoarea predefinită. De exemplu prin skip=nr putem defini câte rânduri ar trebui eliminate din date;

Informații statistice date- Data Profiling

În EDA (Exploratory Data Analysis) un prim pas îl reprezintă obținerea de date statistice despre setul de date cu scopul de a înțelege cât mai bine datele și care să ne permită vizualizări ale datelor și formularea de întrebări pe care să le analizăm în continuare.

Dacă dorim să obținem informații suplimentare despre setul nostru de date, cu alte cuvinte statistici despre fiecare atribut, o modalitate rapidă este de a folosi funcția summary()

O altă posibilitate alternativă la funcția summary() este skim() din pachetul ***skimr***.

Pachetul skimr nu este oferit în distribuția standard și probabil este nevoie să fie instalat.

Tipul variabilei va determina ce informații statistice pot fi oferite, de aceea variabilele vor fi grupate în funcție de tip (de exemplu, se prezintă întâi informații despre variabilele/coloanele de tip caracter, apoi pentru cele de tip numeric).

```
#afișează date statistice despre setul de date pop
library(skim)
skim(pop)
```

Interpretare: Există 3 variabile de tip caracter în setul nostru de date: judet, uat, denumire; nu există valori lipsă (n_missing este 0 și complete_rate este 1) valoarea min și max se referă la lungimea

șirului de caractere (de ex. pentru judet avem 4 – Alba, Arad respectiv 15 – Bistrita-Nasaud); n_unique ne dă numărul de valori unice pentru fiecare variabilă. Pentru variabilele numerice (numărul populației totale, a bărbaților și a femeilor) se va arăta valoarea medie, deviația standard și valorile quantile (p0 reprezintă valoarea min, p100 cea maximă, p50 este valoarea mediană – acea valoare pentru care se găsesc jumatate din date). O mini-histogramă este afișată pentru a sugera distribuția pentru fiecare variabilă.

```
#afișează date statistice despre coloana populatie din pop
skim(pop$Populație)
```

Putem vizualiza strict doar valoarea pentru un anume indicator:

```
#valoarea medie pentru coloana Populație
mean(pop$Populatie)
#valoarea max pentru coloana Populatie
max(pop$Populatie)
```

Observație: În cazul în care am avea în setul nostru de date valori lipsă (care sunt reprezentate prin NA) nu se vor putea calcula valori agregate: dacă datele de intrare conțin o singură valoare lipsă/necunoscută rezultatul va fi necunoscut. Însă toate funcțiile de agregare permit specificarea sub forma unui argument (na.rm=TRUE) a faptului că vrem ca aceste valori lipsă să fie excluse.

```
mean(pop$Populatie, na.rm=TRUE)
```

Valorile lipsă pot fi un indicator al calității datelor de aceea este indicat să se identifice câte valori lipsă există, respectiv a procentului pe care îl reprezintă în setul de date:

```
#numărare valori lipsă
sum(is.na(pop$Denumire))
#calculare procent valori lipsă în cadrul unui atribut
sum(is.na(pop$Denumire))/ sum(is.na(pop$Denumire), !is.na(pop$Denumire))*100
```

În unele cazuri este nevoie să calculăm valori agregate succesive în sensul că dorim să actualizăm o sumă sau medie pe un set de date cu fiecare valoare nouă de intrare. Acestea *sunt cummulative sum sau cummulative mean.* Le vom exemplifica pe un vector mai mic. Funcțiile pe care le folosim sunt incluse în pachetul ***dplyr***, astfel asigurați-vă că este încărcat înainte de a le apela:

```
library(dplyr)

#punem valori de la 1 la 10 în variabila x
x<-1:10
#calculăm o sumă nouă pentru fiecare intrare nouă
cumsum(x) #1 3 6 10 15 21 28 36 45 55
cummean(x) #1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

Cummulative difference se referă la calculul diferenței între termenii aflați într-un vector. Pentru aceasta ne vom folosi de o deplasare a termenilor din vector. Deplasarea poate fi fie în față (toți termenii din vector pot să înainteze un anumit număr de poziții) fie în spate (toți termenii din vector sunt "împinși" un anumit număr de poziții.

```
#deplasare în față cu funcția lead()cu o poziție
lead(x) #2 3 4 5 6 7 8 9 10 NA
#deplasare în față cu funcția lead()cu 3 poziții
lead(x, 3) #4  5  6  7  8  9 10 NA NA NA
#deplasare în spate cu funcția lag()
```

lag(x) #NA 1 2 3 4 5 6 7 8 9
#aflarea diferenței între termenii din vector
x-lag(x) # NA 1 1 1 1 1 1 1 1 1
#exemplu cu valori diferite
y<-c(12,4,5,17,8,9,17,12,21)
y-lag(y) # NA -8 1 12 -9 1 8 -5 9
#găsește valoarea minimă , respectiv maximă odată cu fiecare intrare
cumin(y) # 12 4 4 4 4 4 4 4 4
cummax(y) # 12 12 12 17 17 17 17 17 21

Aflarea poziției de rank: uneori este util să determinăm care este poziția de ordine pentru fiecare element (rankul acestuia). Funcția min_rank() raportează pozițiile după valoarea cea mai mică (1 fiind poziția cea mai mică pentru valoarea cea mai mică).

#raportează pozițiile pentru fiecare element
min_rank(y) # 5 1 2 7 3 4 7 5 9
#raportează descrescător pozițiile pentru fiecare element
min_rank(desc(y)) # 4 9 8 2 7 6 2 4 1
#aproximativ același efect îl are și row_number() însă aici elementele cu aceeași valoare nu pot ocupa aceeași poziție
row_number(y) # 5 1 2 7 3 4 8 6 9

Pentru transformări pe setul de date vom apela la pachetul ***dplyr*** ce este inclus in tidyverse:

- folosim funcția **filter**() pentru a realiza selecția de observații pe baza valorilor pe care le au pentru anumite variabile
- reordonăm rândurile (sortări) cu funcția **arrange**()
- selectăm anumite variabile/coloane cu funcția **select**()
- creăm noi variabile/coloane pornind de la cele existente cu funcția **mutate**()

Toate funcțiile amintite preiau ca și argument o structură de date de tip data frame iar următoarele argumente ce sunt transmise descriu ce ar trebui să se facă cu setul de date folosind nume de variabile. De asemenea, toate funcțiile pot fi folosite în conjuncție cu **group_by()** care definește domeniul de aplicare pentru fiecare funcție astfel încât să nu se mai aplice întregului set de date.

Filtrări date - Data Reduction

Funcția filter() ne permite să selectăm observații pe baza valorilor acestora. Funcția va returna o nouă structură de date (un nou tibble) și nu va afecta datele de intrare. Primul argument va fi numele setului de date iar următoarele argumente vor fi expresii folosite pentru a specifica condițiile de selecție. De obicei acestea vor fi formate din numele unei variabile/coloane, operator de comparație și valoarea cu care se realizează selecția. Filter este echivalentul clauzei where din Select SQL.

Exemple filtrări:

#filtrează datele din județul Alba
filter(pop, Judet=="Alba")
#filtrează datele din judetul Alba cu o populație mai mare de 10000
filter(pop, Judet=="Alba", Populatie>10000)
#filtrează datele din judetul Alba din municipii; observați modul de încadrare a denumirii de variabile cu ghilimele întoarse datorită spațiilor
filter(pop, Judet=="Alba", `Unitate administrativ teritoriala`=="MUNICIPIU")

Condițiile multiple ce sunt puse în funcția filter() sunt realizate concomitent (echivalent cu existența operatorului ȘI între ele). Putem să folosim și alți operatori logici precum | pentru SAU , și ! pentru NOT. În schimb, dacă dorim să adăugăm operatorul SAU între valorile de comparație (de exemplu

vrem să extragem date de la municipii SAU orase) vom primi o eroare. O soluție este de a folosi operatorul %in%

#filtrează datele din judetul Alba din municipii sau orase
filter(pop, Judet=="Alba", "Unitate administrativ teritoriala" %in% c("MUNICIPIU", "ORAS"))

Valorile lipsă pentru o variabilă nu vor fi incluse în rezultatele filtrării decât dacă se specifică explicit în condiția de filtrare. R folosește denumirea NA (not available) pentru acestea și funcția is.na() care verifică dacă argumentul transmis este sau nu o valoare lipsă.

#filtrează toate datele din judetul Alba sau din județul la care nu avem denumire
filter(pop, Judet=="Alba"| is.na(Judet))

Ordonări date

Pentru ordonări am mai folosit funcțiile sort() și order().

In pachetul dplyr pentru ordonări folosim funcția arrange() în mod asemănător cu funcția filter() în sensul că preia numele setului de date asupra căruia se aplică și un set de nume de coloane sau expresii după care se realizează ordonarea.

#ordonare descrescătoare după numele judetului
#criteriul descrescător este transmis sub forma unei funcții deoarece pe langă numele setului de date sunt permise doar nume de coloane sau expresii
arrange(pop, desc(Judet))
#ordonare multicriterială
arrange(pop, desc(Judet), Populatie)

Valorile lipsă sunt adăugate la final.

Selecție coloane

Există seturi de date cu sute de coloane astfel, de multe ori este necesar să selectăm doar acele coloane care ne interesează. Și această funcție preia ca prim argument numele setului de date iar următoarele argumente sunt pentru a specifica coloanele selectate sau expresii aplicate supra coloanelor. În aceste expresii ar putea fi incluse funcții precum starts_with("sir_de_comparat"), ends_with("sir_de_comparat"), contains("sir_de_comparat") care se aplică coloanelor de tip caracter. Funcția everything() face referire la toate coloanele din setul de date și este folosită mai ales în situațiile în care dorim să reordonăm coloanele .

#enumerăm coloanele ce ne interesează, de ex doar Judet si Populatie
select(pop, Judet, Populatie)
#selectăm toate coloanele cuprinse între Judet și Populatie folosind operatorul :
select(pop, Judet:Populatie)
#selectăm toate coloanele mai puțin cele cuprinse între Populatie și Femei
select(pop, -(Populatie:Femei))
#selectăm coloanele care încep cu cuvântul Populatie – in cazul nostru inutil deoarece avem o singură coloană cu numele acesta
select(pop, starts_with("Populatie"))
#punem coloana denumire urmată de celelalte coloane
select(pop, Denumire, everything())

Adăugare noi coloane

Uneori avem nevoie să adăugăm noi coloane derivate din cele existente. Cu ajutorul funcției mutate() putem adăuga noi coloane la finalul setului de date. Coloanele noi create pot fi refolosite în crearea altora:

```
#adăugarea coloanei Rata_barbati și a coloanei Rata_femei calculată din coloana anterioară
mutate(pop, Rata_barbati=Barbati/Populatie*100, Rata_femei=100-Rata_barbati)
```

Dacă dorim să geneream doar coloane noi fara sa pastram setul vechi, vom folosi funcția transmute():

```
transmute(pop, Rata_barbati=Barbati/Populatie*100, Rata_femei=100-Rata_barbati)
```

Pentru a modifica setul existent, putem folosi operatia de atribuire:

```
pop <- mutate(pop, Rata_barbati=Barbati/Populatie*100, Rata_femei=100-Rata_barbati)
```

Grupare date în setul de date

Funcția summarize() sau summarise() (ambele forme sunt acceptate) preia ca argumente numele setului de date și mai multe funcții de agregare care se vor calcula pe setul respectiv:

```
#calculează media pentru coloana Populație
summarize(pop, medie=mean(Populatie))
```

Funcția summarize() este însă mai folositoare dacă o combinăm cu group_by(), ce schimbă unitatea de analiză din setul întreg de date în grupuri. Pentru aceasta ar trebui deci întâi să avem setul de date grupat după un anumit atribut:

```
#grupare pe județe
pop_grup <- group_by(pop, Judet)
#aflare medie populatie pe fiecare grup
summarize(pop_grup, mean(Populatie))
#aflarea mediei și atribuirea unui nume rezultatului
summarize(pop_grup, medie=mean(Populatie))
#sau într-un singur pas
summarize(group_by(pop, Judet), medie=mean(Populatie))

#aflarea mediei pentru fiecare judet și numărarea de valori din fiecare grup: observați funcția n() pentru numărare
summarize(pop_grup, medie=mean(Populatie), numar=n())
```

Observație: pentru a număra dimensiunea dintr-un set sau grup folosim funcția n() fără argumente. Dacă dorim să obținem numărul de elemente distincte (valori unice) folosim n_distinct(x). Acestea vor fi apelate din funcția summarize()

```
#câte județe distinte avem
summarize(pop, n_distinct(Judet))
```

Numărarea este foarte utilă astfel pachetul dplyr ne oferă și funcția count() unde se poate transmite direct prin argument atributul pentru care dorim să facem numărarea.

```
#câte observații avem pentru fiecare judet
pop %>% count(Judet)
```

Observam folosirea operatorului special %>% care inseamna *pipe*: adica inlantuirea mai multor comenzi. R executa fiecare dintre comenzile inlantuite prin %>% si afiseaza rezultatul final.

Pe lângă metodele amintite care au ca rezultat tot structuri de date de tip data frame, mai există funcția **table**() care creează un tabel cu frecvențe. Acestea sunt folosite în determinarea probabilităților în metode de analiză precum Naive Bayes.

```
#câte observații avem pentru fiecare judet
table (pop$Judet)
```

Presupunem că dorim mai multe ***operații înlănțuite***: grupare pe județe, apoi calculul mediei la Populație, filtrarea acelor județe care au media mai mare de 5500. Realizarea acestor operații într-un singur pas ar duce la o expresie greu de urmărit. Pe de altă parte dacă am face aceste operații pe rând am fi nevoiți să folosim variabile intermediare (pop_grup, pop_grup_filter) care solicită atribuire de nume:

```
pop_grup <- group_by(pop, Judet)
pop_grup_medie <-summarize(pop_grup, mean(Populatie))
filter(pop_grup_medie, medie>5500)
```

Folosind operatorul ***pipe*** %>% putem să scriem aceste operații înlănțuite fără să avem nevoie de folosirea acestor variabile intermediare, ceea ce ne permite să ne concentrăm asupra operațiilor în sine și nu a ceea ce este transformat, făcând codul mai ușor de citit. În plus, funcțiile nu mai solicită, ca argument, numele unui set de date, acesta va fi considerat implicit rezultatul anterior. Pentru a sugera această înlănțuire de operații operatorul pipe poate fi citit "then".

```
#aceleași operații din exemplul anterior
pop %>%
group_by(Judet) %>%
summarize(medie=mean(Populatie)) %>%
 filter(medie>5500)
```

Putem renunța la grupare în setul de date folosind funcția ungroup(), astfel operațiile care sunt înlănțuite pot să aibă domenii de aplicare diferite.

Grupare date de pe linii și pe coloane

Câteodată, pentru a putea realiza analiza datelor este nevoie să realizăm operații de a aduna pe o singură coloană valori care sunt pe mai multe coloane sau invers să extindem datele/categoriile care sunt într-o coloană sub forma unor coloane noi. Pentru aceste operații am putea folosi funcțiile **gather**() respectiv **spread**() din pachetul tidyr ce este furnizat odată cu tidyverse.

Pentru exemplificare ne vom folosi de un set de date nou în care avem evidența notelor la diverse discipline pentru studenți:

```
#creăm vectorii/atributele ce vor fi în data frameul stud
nume<-c("Ana","Bob", "Andrei", "Lucia", "Alex")
informatica<-c(10,10,9,5,7)
management<-c(8,7,7,9,9)
statistica<-c(9,7,7,5,6)
stud<-data.frame(nume,informatica,management,statistica)

#folosim gather() pentru a grupa toate disciplinele în coloana disciplina iar valorile acestora în coloana nume
#primul argument al fct gather() e numele setului de date, apoi numele coloanei noi în care vor fi adunate coloanele existente și numele coloanei în care se vor aduna valorile urmat de numele coloanelor adunate: de ex folosind indecsi 2:4 sau excluse: -1 sau -nume
note_stud<- gather(stud, discipline, note, 2:4)
```

#echivalent cu
note_stud<- gather(stud, discipline, note, -nume)
#vizualizăm noul set de date
note_stud
#putem să calculăm o medie a tuturor notelor de la toate disciplinele
mean(note_stud$note)

folosim spread() pentru operația inversă: pornind de la setul de date note_stud dorim să creăm o coloană pentru fiecare disciplină cu notele corespunzătoare
#primul argument al fct spread() e numele setului de date, apoi numele coloanei pe care dorim să o extindem pe mai multe coloane noi și numele coloanei de unde vor fi preluate valorile corespunzătoare; obținem un set de date asemănător cu stud cu deosebirea că de această dată rândurile sunt ordonate după nume
spread(note_stud, discipline, note)

Comparații mulțimi date

Putem folosi operatori precum >, >=,<,<=, ==,!= pentru a compara element cu element valorile dintre vectori și rezultatul va fi un nou vector ce conține valorile logice True sau False: vectorii comparați ar trebui să fie de același tip însă nu neapărat de aceeași dimensiune (în caz în care nu avem aceeași dimensiune la vectori se aplică "reciclarea valorilor" adică vectorul de dimensiune mai mică este completat cu valorile lui până ajunge la dimensiunea vectorului mai mare).

Pe lângă aceștia avem câteva funcții ce realizează operații pe mulțimi: **union**() – reuniunea mulțimilor, **intersect**() – intersecția mulțimilor și **setdiff**() – scădere dintre mulțimi

#comparăm secvențele de numere 1:3 cu cea 1:5
1:3<1:5 #FALSE FALSE FALSE TRUE TRUE
#fie un nou vector cu nume studenți pornind de la cel anterior
nume2=union(nume,c("Diana", "Matei"))
#comparăm elementele celor 2 vectori
nume2==nume
#scăderea dintre mulțimi
setdiff(nume2,nume)

#operațile pe mulțimi pot fi realizate și cu valorile coloanelor dintr-un set de date
stud$informatica<stud$management #FALSE FALSE FALSE TRUE TRUE