R nyelv – bevezetés, változók, struktúrák

Vadon Viktória

2024/2025/I. félév

Tartalomjegyzék

1. Első lépések **2**
1.1. R nyelv . . . 2
1.2. Telepítés . . . 2
1.3. Források . . . 3
1.4. Futtatás, felhasználói felület . . . 3
1.5. R Console ablak . . . 3
1.6. .R script fájlok . . . 4
1.7. Hogyan mentsünk? . . . 4
1.8. Segítség, súgó . . . 5
1.9. Nyalánkságok . . . 5
1.9.1. Memóriakezelés . . . 5
1.9.2. Könyvtárak, search path . . . 6

2. Szintaxis **6**
2.1. Szintaxis alapok . . . 6
2.2. Parancsként működő speciális karakterek . . . 6
2.3. Értékadás . . . 7
2.4. Típus: mode VS class . . . 7
2.5. Objektum attribútumai . . . 7

3. Mode; alapvető változótípusok **8**
3.1. Mode . . . 8
3.2. Típus ellenőrzése, módosítása . . . 8
3.3. Alapműveletek . . . 9

4. Class; adatstruktúrák **9**
4.1. Class . . . 9
4.2. Vector . . . 9
4.2.1. Elemek elérése . . . 10
4.2.2. Factor . . . 11
4.3. Array, matrix class-ok . . . 12
4.3.1. Mátrix attribútumai, műveletei . . . 13
4.4. List . . . 14
4.5. Data frame . . . 14

1. Első lépések

1.1. R nyelv

- ingyenes, nyílt forráskódú
- főleg valószínűségszámítás, statisztika, adatelemzési területeken használják
  - jól tud dolgozni táblázattal, mátrix-szal, adattáblával
  - fejlett grafikus képességek!
  - beépített függvények statisztikai mutatókra, adatelemzési műveletekre
  - statisztikai tesztek beépítve
  - valószínűségi eloszlások különböző függvényei beépítve, valószínűségi változók generálása adott eloszlásokból, stb.
- objektum alapú
  - minden változót objektumként tárol és a memóriában tart (ld. memóriakezelés)
  - sok beépített függvény outputja komplex adatstruktúra/objektum, amiből adattagok külön lekérhetők, vagy további függvényekkel elemezhető tovább
  - persze definiálhatók nem csak változók, hanem saját függvény objektumok is (ld. később)
  - különböző attribútumok tartoz(hat)nak hozzájuk
- interpretált
  - R Console-ban párbeszéd, munkafüzet formájában dolgozunk (mint egy parancssor)
  - script fájlok írhatók: gyakorlatilag egymás után futtatandó parancsok egymásutánja (mint egy shell script)

1.2. Telepítés

- a telepítendő R project hozza magával az R nyelv motort/interpretert és a fejlesztői környezet is
- letöltés:

  - R Project homepage > balra Download alatt CRAN – rendszertől függően kövessük a linkeket és utasításokat
  - mirror: letöltési szerver – gyors letöltéshez válasszunk földrajzilag közelit, pl. Ausztria
  - Windows-on első, általános célú telepítés: Download R for Windows > base (Install R for the first time) > legfelső link az installer (az anyag készülésekor R-4.4.1 verzió)
- telepítsük értelemszerűen

1.3. Források

- R project weboldal > manuals
- R intro

1.4. Futtatás, felhasználói felület

- parancsikon: R betű + kék ellipszis, elindítja az R GUI-t
- munkamappa beállítása:
  - **R GUI-ban: File > Change dir...**
  - parancsikon tulajdonságainál, „Run in” („futtatás helye” ?)
  - tipp: almappázzunk projektenként, hogy a mentések elkülönüljenek – ld. később a mentéseknél, egyes mentett objektumok csak kiterjesztést kapnak, fájlnevet nem.
- GUI: grafikus környezet, felépítése ablakon belül ablakok
  - Windows-ból ismert módszerrel tudjuk kezelni őket: maximalizálni, minimalizálni, átrendezni, kattintással előtérbe hozni, nyújtani... + „Windows” nevű GUI menü-ből ablakok közti váltogatás, különböző tiling (mozaik) opciók, stb.
  - mindig egy ablakunk aktív (az van előtérben)
  - **aktív ablaktól függően változik az ikonsorunk is!**
  - **ablakok pl.:**
  - R Console: „parancssor”, párbeszéd R-rel az interpreteren keresztül
  - script fájlok
  - ábrák
  - szerkesztésre megnyitott adattábla
  - súgó, egyéb segédletek...

1.5. R Console ablak

- bevitel, végrehajtás enter-rel, azonnali hatás (és válasz, ha releváns)
- History: fel-le nyílbillentyűkkel előhívhatók korábbi parancsok, hogy megspóroljuk az újragépelést – ezek szerkeszthetők (jobbra-balra nyíl, Del, Backspace, gépeléssel

beillesztés)
- nem muszáj egyetlen sorban befejezni egy parancsot/függvényhívást – ha látja, hogy szintaktikailag nem teljes a parancs, pl. nem zártunk be zárójeleket, akkor megváltozik a sor eleji szimbólum >-ról +-ra, ezzel jelzi, hogy várja a folytatást

1.6. .R script fájlok

- kb. „shell script"
- cél: egymás után futtatandó parancs-sorozat, procedúra mentése későbbre
- tartalmazhat: változók definícióit, parancsok sorozatát és/vagy függvény definíció(ka)t is
- órai munkához javasolt Console-on „kísérletezni" a szintaxissal, de érdemi munkát, feladatok megoldását .R script-be tenni és menteni
- futtatás: aktuális sor vagy kijelölés(!) futtatható: ikonsorból, vagy jobb egérgombos menüből „Run line or selection'; gyorsbillentyű: **Ctrl+R**
  - ha nem akarunk egerezni, a mozgáshoz, kijelöléshez **néhány hasznos billentyűparancs** (Windows alatt):
  - **minden kijelölése: Ctrl+A**
  - több sor kijelölése: Shift + gyaloglás – a fix pont a kezdő pozíció! – visszafelé gyaloglással szűkíthető is a kijelölés
  - gyalogláshoz: nyílbillentyűk, Ctrl+jobbra-balra: szavanként lépkedés, Home/End: sor eleje/vége, Ctrl+Home/End: fájl eleje/vége
  - pl. Shift+Ctrl+Home: aktuális pozíciótól fájl elejéig ugrás+kijelölés
- teljes script fájl futtatása Console-ból: `source("fajlnev.R")` – ha nem a munkamappában van a fájl, akkor `source("eleresi_ut/fajlnev.R")`
- script futtatásakor is Console-ra kerül az output!

1.7. Hogyan mentsünk?

- több menthető dolog van!!
- workspace mentés: memóriában lévő objektumok (változók, általunk definiált függvények, stb.) (.RData) – alapértelmezett mentés, Ctrl+S ezt hozza elő, mentés nélküli kilépéskor ezt javasolja
- history: parancs-előzmények (.Rhistory) – ld. fentebb az R Console-nál – akkor érdemes, ha szeretnénk folytatni a munkát; .RData-t is mentsük vele!
- .RData, .Rhistory alapértelmezében fájlnév nélkül készül – ezért érdemes projektenként almappázni!
- **.R script fájlok** – újrahasznosítható kód, pl. procedúrák, stb. – ezzel utólag is visszakövethető a munka, kiadott parancsok (de az output nem – az majd újrafuttatással generálható újra)
- **Console log** (R-rel folytatott párbeszéd): ha a Console aktív, File > Save to file – .txt formában – minden lefutott input és hozzá tartozó output – ismerkedés

során érdemes ezt is menteni, hogy vissza tudjuk nézni a szintaxist, válaszokat, stb.
- generált ábrák: ha az ábra az aktív ablak, File > Save as > ... – válasszuk ki az almenüt a kívánt fájltípus, kiterjesztés szerint!
- korábbi mentés megnyitása: típustól függően: file > load script, load workspace, stb. – Console log-ot sehogy sem tudjuk visszatölteni! de pl. ha .R scriptből futottak a parancsok, újra lehet futtatni és generálni a korábbi outputokat[1]

1.8. Segítség, súgó

- a súgó böngészőben, de localhost-ról, internet nélkül is fut! az aktuális R verzióval együtt települ, és így mindig a telepített verzióra vonatkozik
- súgó kezdőlap: Console-on futtassuk `help.start()` – ezen belül kereshetünk, kattintással navigálhatunk, stb.
- adott témához, parancshoz súgó:
  - `help(kulcsszó)` – ha tudjuk az objektum pontos nevét, ezzel megnyitja az oldalát/dokumentációját
  - `??kulcsszó`, `help.search(kulcsszó)` (néha `"kulcsszó"` formában, azaz dupla idézőjelek közé zárva kell neki) – *keresés* a súgóban
- javaslat: nem kell memorizálni a szintaxist, (néha a függvény pontos nevét sem), használjuk bátran a súgót!

1.9. Nyalánkságok

1.9.1. Memóriakezelés

- minden változó (és felhasználó által definiált függvények is) egy-egy objektum
- változók újradefiniálhatók, felülírhatók
- ha tervezzük menteni a workspace-t (memóriakép), érdemes előtte feltakarítani a fölösleges változókat!
- vagy csak azért is, hogy ne lassítsuk le a gépet, és saját magunkat se zavarjuk össze
- `ls()` vagy `objects()` parancs listázza az épp létező objektumok neveit
- `rm(valtozo1, valtozo2)` vagy `remove(valtozo1, valtozo2)` – változók listája, vesszővel elválasztva – végleg törli őket, felszabadítja a memóriát
  - nincs parancs minden objektum törlésére! az R ezzel akadályozza meg, hogy valamit véletlenül töröljünk. de a memória teljes törléséhez újraindíthatjuk a GUI-t, workspace mentése nélkül – csak ne felejtsük el menteni előtte, amit akartunk, pl. Console log-ot, .R scripteket!

[1] Ha véletlen generálás volt benne, akkor más véletlen számokat kapunk.

1.9.2. Könyvtárak, search path

- package = könyvtár, csomag
- telepített könyvtárak listája felugró ablakban: `library()`
- betöltés: `library(csomag.neve)`
  - tulajdonképpen a search path-hoz adja hozzá
- search path: keresési útvonal
  - gyakorlatilag az elérhető névterek, amiken belül keresi az objektumokat (változókat, függvényeket)
  - egy listában tárolja ezeket a névtereket, prioritási sorrendben
  - gondoljunk rá úgy, mint a betöltött csomagok listájára; a csomagokban definiált objektumok közvetlenül elérhetők

2. Szintaxis

2.1. Szintaxis alapok

- változó-és függvénynevekben megengedett karakterek:
  - `a-zA-Z0-9._` – angol abécé kis-és nagybetűi, számok, pont, alulvonás
  - érzékeny kis-és nagybetűkre!
  - konvenció: több szóból álló függvénynév, változónév esetén ponttal választjuk el a szavakat (ahol sok más nyelv alulvonást használna)
- függvény argumentumai
  - van „overload”, ugyanaz a függvény különböző argumentumokkal is hívható
  - argumentumok egy pár kerek zárójelben
  - `fuggveny.neve(arg1,arg2,nev1=ertek1,nev2=ertek2)`
  - argumentumok átadása: **vegyesen** pozíció és név szerint!
  - az első néhány fontos, kötelező argumentum az elejére, itt fontos a sorrend! – de ők is megadhatók név szerint is, akkor keverhető a sorrend.
  - utána opcionális argumentumok név szerint, tetszőleges sorrendben

2.2. Parancsként működő speciális karakterek

- komment: `#` – szinte bárhol használható, sor végéig érvényes
- `()` – kerek zárójel: függvény argumentuma(i)
- többdimenziós változó, mint vektor, mátrix stb. elemeinek elérése `[]`, `[[]]` – később visszatérünk a különbségre
- névvel ellátott „adattag” elérése: `$` – `objektum$adattag.neve` (ami Javascriptben a pont lenne)
- csoport létrehozása `{}`
- `;`, pontosvessző: egy sorban több parancs is kiadható, pontosvesszővel elválasztva – nem rejti el az outputot, azaz ha külön-külön van outputjuk, külön-külön kiírásra

kerül egymás alá!

2.3. Értékadás

- értékadás: x-be tároljuk el az expr (kifejezés) értékét:
  - `x <- expr`
  - `expr -> x`
  - a nyíl, bármelyik irányban is, erősen preferált!!
  - de működik az egyenlőségjel is: `x = expr` (szimpla egyenlőségjel!) (bár szentségtörésnek számít, magyar billentyűzeten egyszerűbb begépelni...)
- output VS értékadás:
  - ha az input egy `expr`, az output a kiszámolt értéke
  - ideiglenes változó: `.Last.value` (figyeljünk, mert 1 nagybetű csak...)-ban tárolja a legutóbbi Console output eredményét (ha tovább szeretnénk vele számolni, vagy utólag jövünk rá, hogy mégis el szeretnénk tárolni – utóbbira megoldás még a nyíllal előcsaljuk és szerkesztjük a parancs-előzményt, de ha véletlen generálás történt, akkor újrasorsol)
  - ha az input értékadás `x <- expr`, nincs Console output (az output-ot gyakorlatilag átirányítottuk a változóba)
  - ha az input `x` változónév, az output a benne tárolt érték/objektum

2.4. Típus: mode VS class

- `mode`: alapvető („egydimenziós") változótípus (ld. lentebb), mint numeric, character, logical, (ld. lentebb)
- `class`: osztály
  - objektumorientált programozásból ismert, az osztályhoz definiálhatók osztályspecifikus függvények, specializált kiíratás, stb.
  - pl. matrix, array, data.frame (ld. később)
  - sokszor egy összetett típus/struktúra, ami egyéb adattípusokból építkezik

2.5. Objektum attribútumai

- attribútum: objektumhoz tartozó tulajdonság – nem adattag, hanem metadata
- alábbi függvényekkel lekérhetők `fuggveny(obj)` formában:
- `mode()`
- `class()` – ha egyszerű adattípusról van szó, ugyanaz, mint a mode
- `length()`: hossz – többdimenziós objektum esetén a tárolt adattagok száma
- `attributes()`: egyéb attribútumok, név-érték „listában" – class-tól vagy adott objektumtól függően
  - pl. `names()`: adattagok nevei (magyarázat később)
  - pl. `dim()`: méret, dimenziók – mátrix, többdimenziós tömb esetén

- adott attribútum felülírása: `attr(obj,attr.neve) <- ...`, vagy pl. `attributum(obj) <- ...`

3. Mode; alapvető változótípusok

3.1. Mode

- emlékeztető: **mode: alapvető, „egydimenziós" változótípusok**
- numeric – szám
  - integer – egész
  - double – dupla pontosságú lebegőpontos
  - complex – komplex szám, pl. `2+3i`
- character – karakter, „string", karaktervektor
  - pl. `"abc"`, bevitel idézőjellel (vagy aposztrófffal)
  - C típusú string: `\0`-val végződő, lehet benne `\n`, `\t`, stb.
- logical
  - szokásos `TRUE`, `FALSE` (`T`, `F` formában is – ha nem írjuk felül)
- **néhány speciális érték:**
  - `Inf` – szimbolikus végtelen (lebegőpontosból ismert), numeric típus
  - `pi`: $\pi \approx 3.14...$
  - `NaN` – not a number, nem szám (lebegőpontos NaN, pl. 0/0, Inf-Inf), numeric típus
  - `NA` – not available, hiányzó adat, logical típus
  - `NULL` – szimbolikus semmi – mode-ja, class-a is NULL – hiányzó attribútum jelzése pl.

3.2. Típus ellenőrzése, módosítása

- **típus/osztály ellenőrzése**
  - lekérés: `mode()`, `class()` – Console-on szöveges választ kapunk
  - adott típusba, osztályba tartozik-e: pl. `is.numeric()`, `is.vector()` – Console-on válasz logical érték: TRUE/FALSE
  - **speciális...**
  - NA ellenőrzése: CSAK `is.na()` ! (ha relációs jellel próbáljuk ellenőrizni, hülyeségeket kapunk.)
  - `is.na()` eredménye TRUE NA-re és NaN-re is – a kettő megkülönböztetésére `is.nan()`, csak NaN-re TRUE
  - `is.null()`
- konverzió és hasonlók...
  - *helyi hatású* konverzió, casting/coersion: pl. `as.numeric()`, `as.vector()`
  - `unclass()`: class eltávolítása *ideiglenesen* – pl. osztályától megfosztott mátrixból csak az adatok vektora marad meg

3.3. Alapműveletek

- alapműveletek numeric értékeken:
  - +, -, *, /
  - exponens: `^` kalap, vagy ** két csillag
  - div, egész hányados: `%/%`
  - mod, egész osztás maradéka `%%`
  - néhány beépített matematikai függvény: sqrt(), sin(), cos(), tan(), log(), exp(), abs()
- relációk vizsgálata – tipikusan numeric értékek között – eredményük mindig logical TRUE/FALSE
  - <, >, <=, >= értelemszerűen
  - == egyenlő-e; != nem egyenlő-e
- logikai műveletek
  - logical TRUE/FALSE változókra (vagy kiértékelt relációkra – őket zárójelezzük, hogy egyértelmű legyen)
  - `&` és, `|` vagy – bináris!
  - xor() kizáró vagy
  - ! (felkiáltójel): tagadás
  - zárójelekkel csoportosítható
  - „többváltozós és/vagy”: all(), any(): igaz ha mind igaz, ill. ha legalább egy – lehet a hasában felsorolás, vagy logical-ökből álló vector (ld. később)
- stringek összefűzése: `paste(string1,string2,sep=" ")` – alapértelmezésben szóközt tesz közéjük, de lehet üres string `""`, stb.

4. Class; adatstruktúrák

4.1. Class

- class: komplex, többdimenziós adatstruktúra
- amikről beszélni fogunk: vector, factor, matrix/array, list, data.frame

4.2. Vector

- vector: vektor, *azonos mode-ú változók gyűjteménye*, egyetlen indexszel
  - ha definiáláskor több mode kerülne bele, automatikus coersion történik – egy olyan típusra, amibe minden belefér – pl. ha character keveredik numeric közé, minden character lesz
  - különböző mode-ú objektumok gyűjteménye `list` – ld. később
- legalapvetőbb, leggyakoribb – egyetlen változó is egy 1-dimenziós vektor
- létrehozás:
  - sokszor valós alkalmazásokhoz külső adatforrást használnak – erről később

  - véletlen számokból álló vektor generálása – szintén később
  - c(), combine/concatenate, összefűzés függvénnyel – tetszőleges számú argumentum, lehet benne helyben bepötyögött szám/kifejezés, változó, már létező vektor; pl. `vekt <- c(1,0,-3,pi)`
  - számtani sorozat generálása:
    * `a:b` : tartomány, kettőspont szintaxis: számok sorozta a-tól b-ig egyesével növekedve (vagy csökkenve)
    * **seq() függvény**
    * 4 argumentumból 3-at kell megadni:
    * from=a, vagy név nélküli 1. argumentum a : a-tól
    * to=b, vagy név nélküli 2. argumentum b : b-ig
    * by=d : növekmény d
    * length=n vagy len=n : összesen n elem legyen
  - ismétlés: rep()
    * `rep(vekt,times=2)`: kétszer egymás után fűz
    * `rep(vekt,each=2)`: elemenként kétszer ismétel
- műveletek, relációk vektorokkal
  - alapműveletek, egyváltozós függvények, relációs feltételvizsgálatok *elemenként*!
  - *különböző hosszúságú vektorokon is működik!* ilyenkor az eredmény olyan hosszú, mint a hosszabbik, a rövidebbiket pedig elkezdi ismételgetni az elejéről – ha nem egész számszor használta fel, akkor dob egy warning-ot, de akkor is kiszámolja.
- néhány vektor->skalár függvény:
  - min(), max()
  - length() : vektor hossza
  - sum() : összeg; prod() : szorzat ; mean() : átlag ; var() : (empirikus korrigált) szórásnégyzet
  - range(vekt) : tulajdonképpen c(min(vekt),max(vekt))
  - pmin(), pmax(): hasába több vektor, külön-külön min/max
- rendezés: sort(vekt) – rendezett másolat, eredetit nem írja felül

4.2.1. Elemek elérése

- elem lekérése: vekt[index]
- indexelés (matekosan) 1-től indul!
- felülírás: vekt[index] <- uj.ertek
- ha kiindexelünk, az eredmény NA[2]
- de nem létező index definiálható, bővül a vektor – hiányzó köztes értékeket NA-val tölti fel
- indexvektor használata vekt[indexvekt]

[2] kivéve: 0-ás index egy 0 hosszú, azonos mode-ú vektort add vissza

  - több elem is lekérhető egyszerre
  - ezzel az elemek sorrendje is felcserélhető, vagy ismétlődhet is elem ha az indexe többször szerepel
  - eredmény olyan hosszú, mint az indexvektor – lehet hosszabb, mint az eredeti vektor!
  - itt is használható tartomány (a:b-szintaxis), c(), seq(), stb.
  - minden *kivéve* az adott indexű elemek: vekt[-indexvekt]; vagy az indexvektoron belül negatív elemek
  - de: pozitív és negatív indexek nem keverhetők!
- kifejezett beszúrás, törlés nincs
  - törlés viszonylag egyszerű negatív indexeléssel
  - beszúráshoz tartományos indexelés, összefűzés, ...
  - majd a módosított vektorral felülírjuk az eredetit
  - vektor vége levágható a length() attribútum kisebbre állításával
- „szűrés logikai feltétel szerint"
  - indexbe egy logikai feltétel, eredmény: vektor azon elemei (eredeti sorrendben), amikre igaz a feltétel
  - pl. pozitív elemek kiválogatása: vekt[vekt>0]
  - bent a feltételben is a vektor nevét használjuk!
  - lehet összetett feltétel is – összekapcsoláshoz ld. a logikai műveleteket
- indexelés logical vektorral
  - „párba állítjuk" a vektorok elemeit, azokat hozzuk, akiknek TRUE a párja
  - ha a logical vektor rövidebb, akkor ciklikusan ismétli
  - igazából a fenti, feltétel szerinti válogatás hátterében is ez áll

4.2.2. Factor

- factor: vector-ra épülő class – egy kategorikus változó: véges sok különböző értéke lehet, az adatok csoportosítására szolgál
  - mégis hogyan értelmezzük ezt a csoportosítást?
  - **példa**: tegyük fel, hogy adott:
  - egy string vector, hallgatók neveivel
  - ugyanolyan hosszú numeric vector, *azonos sorrendben* a hallgatók magasságával
  - ugyanolyan hosszú string vector, *azonos sorrendben* a hallgatók nemével: F/N
  - itt a nemek vektora egy kategorikus változó, ami szerint érdemes lehet csoportosítani az adatokat
- mivel egy másik class, mint a vector, másképp viselkedik, pl. más a grafikus reprezentációja, extra attribútumok, stb.
- factor tulajdonságai:
  - lehet rendezett vagy rendezetlen – például érdemjegyek rendezett, de egy megyék szerinti, vagy férfi/nő csoportosítás nem

  - a rendezést sort() adja – növekvő vagy alfabetikus rendezés
  - levels() attribútum: „szintek” – a megengedett értékek
- *diszkrét* adatvektor konvertálása factor-rá: factor(vekt) – nem ír felül automatikusan!
  - ha rendezett factor-t szeretnénk: ordered(vekt)
  - megmarad az elemek eredeti sorrendje, de az azonos értékeket „azonosítja”
- fenti példában átlagmagasság számítása, külön-külön férfiakra és nőkre: tapply(magassag.vekt,nem.
  – output egy vektor, a nem.vekt lehetséges értékeivel indexelve
  - általában: tapply(data,factor,function), ahol feltesszük, a data és factor azonos hosszú vektorok, és azonos indexű elemeik összetartozó adatpárok
  - háttérben: tapply először a factor értékei szerint feldarabolja, szétválogatja data-t, több, akár különböző hosszúságú részvektorra – ezt az adatstruktúrát hívják néha ragged array-nek – és utána minden részvektorra alkalmazza a megadott függvényt
- folytonos adatok kategorizálása, diszkretizálása
  - statisztikai vizsgálatokhoz sokszor hasznos, ha egy folytonos adatot, mint pl. a fenti magasság, intervallumokra vágunk – pl. népesség csoportosítása nem pontos életkor, hanem korosztályok szerint
  - intervallumokra bontáshoz: cut(data.vekt,breaks), ahol:
  - vagy: breaks=n, pozitív egész, hány kategóriára vágjunk? – automatikusan számolt töréspontok
  - vagy: breaks=vekt, egy vector típusú változó, ami a töréspontokat tartalmazza – az első intervallum elejét és utolsó intervallum végét is beleértve! ha rosszul adjuk meg, a túl kicsi vagy túl nagy értékek nem kerülnek bele egyik kategóriába sem! hasznos lehet min(data), -Inf, stb. használata
  - a szintek nevei automatikusan a létrejött intervallumok
  - opcionális argumentum: `ordered_result=TRUE`, legyen rendezett a faktor
- gyakoriságtábla: table()
  - table(factor) – gyakoriságok, a lehetséges értékekkel címkézve – diszkrét vektorra is működik
  - table(factor1,factor2) – együttes gyakoriság-táblázat: feltéve, hogy a két factor azonos indexű tagjai itt is összetartozó adatpárok, pl. azonos sorrendben személyek szem- és hajszíne, egy táblázatot készít, megszámolva, melyik szem- és hajszín kombinációval hány személy van az adathalmazban

4.3. Array, matrix class-ok

- azonos mode-ú elemek tárolására alkalmasak, 2 vagy több dimenziós (véges) táblázatokba rendezve
- mátrix: 2-dim, 2 index
- array: lehet több dimenziós, több indexű is.
- dim() attribútum: méreteket tartalmazó vektor, pl. ha dim() értéke c(2,3), egy 2

soros, 3 oszlopos mátrixot jelent.

- mátrix létrehozása:
  - opció 1: létező vektorból coersion-nel, dim attribútum definiálásával: M <- vekt, dim(my.matrix) = c(m,n) – oszloponként tölti fel! – ez csak akkor működik, ha vekt mn hosszú
  - opció 2: példányosítás: M <- matrix(data,m,n), vagy matrix(data,c(m,n)) – data az adatvektor, m, n a sorok és oszlopok száma – itt is oszloponként tölt fel! – ha a vektor hossza nem megfelelő, akkor warning, de létrejön, levágja a végét, vagy ciklikusan ismétel
    - opcionális argumentum: byrow=TRUE: feltöltés soronként
- elemek elérése: több indexszel, M[i,j]
  - kihagyott index: vegyük mindet – sor M[i,], oszlop M[,j]
  - itt is működik c()-vel vagy :-szintaxissal több elem kiválasztása – több sor, több oszlop esetén a metszetükben lévő elemek részmátrixát kapjuk
  - ha csak egy indexet kap, vessző nélkül – az adatvektor adott indexű elemét kapjuk!
- blokkmátrix építése:
  - cbind(matrix1,matrix2,...): mátrixok oszlopvektorait másolja be balról jobbra – ugyanolyan hosszú sorokból, azaz ugyanannyi sorból kell állni a két mátrixnak, és egymás mellé kerülnek
  - rbind(): analóg módon a sorvektorokat másolja be fentről lefelé
  - függőlegesen és vízszintesen is több blokkból álló mátrix: fenti parancsok egymásba ágyazásával, vagy több lépésben
  - vigyázat: c() a mátrixokban tárolt adatok 1-dim vektorait fűzi össze, tipikusan nem ezt akarjuk!

4.3.1. Mátrix attribútumai, műveletei

- nrow(), ncol() – sorok, oszlopok száma
- skalárral szorzás, összeadás, kivonás elemenként
- transzponálás t()
- mátrixszorzás `%*%`
  - vector adattípus az R fejében flexibilisen lehet sor-vagy oszlopvektor is, nincs definiálva!
  - matrix és vector szorzásánál automatikusan értelmezi
  - de! két vector mátrix-szorzata nincs egyértelműen definiálva! egyértelműsítés:
    - ver 1: skalár szorzásra crossprod(vekt1,vekt2), diadikus szorzatra outer(vekt1,vekt2)
    - outer() függvény általánosabban: outer(vekt1,vekt2,function), ahol function egy kétváltozós függvény, az első változó vekt1 értékein iteráls, a második vekt2 értékein, és egy táblázatban adja vissza az értékpárokon számolt eredményeket; a fenti diadikus szorzat az alapértelmezéssel: outer(vekt1,vekt2,"*")

        * vector kényszerítése sor/oszlopmátrixszá: rbind(), cbind()
    - crossprod(x,y): $x^T y$, de effektívebb, mint `t(x) %*% y`; crossprod(x) rövidítés crossprod(x,x)-re
- diag() - trükkös, mert: diag(matrix) – diagonális elemek vector-ban, diag(vector) – diagonális mátrix, a vector elemeiből, diag(n) – nxn-es egységmátrix
- determináns: det()
- solve(A,b): $Ax = b$ LER megoldása – inverz számítás: solve(a), bár nem effektív
- sajátértékek: eigen(matrix) – output egy komplex objektum, values adattag a sajátértékeket, vectors adattag a sajátvektorokat tartalmazza, elérésük: `$`; pl. ev <- eigen(M), ev$values

4.4. List

- 1-dimenziós objektum, de különböző mode-ú elemeket is tartalmazhat.
- nem gyakori
- az egyes adattagok name attribútummal is elláthatók, de nem kötelező – hasonlít Python dictionary-hez, a name a key, az elem/adattag a value
- tartalmazhat újabb list-et is, vagy általánosságban bármilyen class-t
- létrehozás: list(név1=elem1,elem2.név.nélkül,...)
- elemek elérése: list[[index]], list[["név"]], list$név
    - ha az elem egy list, vagy vector, jöhet egy újabb [[]] vagy [] az elemei eléréséhez
    - a list elemire működik [] is, de! az name attribútummal együtt hozza az elemet – tovább nem indexelhető.
- elemek nevei: names() attribútum; az egész felülírható, vagy adott indexű eleme(i)
- list-ek is összefűzhetők c()-vel, az eredmény list

4.5. Data frame

- data.frame class, adattáblák tárolására szolgál
- a sorok összetartoznak, egy-egy kísérlet, vagy kutatási alany adatai, az oszlopok pedig egy-egy mérés eredménye
- korábbi példánkban: hallgatók soronként, első oszlop a név, második a magasság, harmadik a nem
- az oszlopok vector típusok, azaz azonos mode-ú adatokat tartalmaznak (kivétel: NA-k lehetnek benne); általában elnevezzük őket, names() attribútum – ha nem, az R generál nekik nevet
- a sorokat általában csak indexeljük, de row.names() attribútum segítségével elnevezhetjük őket
- indexelés, részek elérése:
    - oszlop: df["oszlop.neve"], df[oszlop.indexe] – névvel együtt kiemeli az oszlopot; gyakorlatilag egy kisebb data.frame-et ad vissza

  - `df$oszlop.neve` vagy df[[oszlop.indexe]] vagy df[,oszlop.indexe] szintaxissal „csupasz” adatvektor lekérése
  - elem elérése: df[sor.index,oszlop.index] – bármelyik lecserélhető "név"-re is
  - sor elérése: df[sor.indexe,], df["sor.neve",]
  - több sor és/vagy oszlop: c() vagy :-szintaxis; negatív indexszel kihagyás is működik
- data.frame létrehozása
  - alapvetően oszloponként építkezik, minden oszlop egy vector
  - sokszor: df <- data.frame(név1=vekt1, név2=vekt2,...)
  - pl. mátrixból coersion-nel: as.data.frame()
  - adatok forrása lehet bármi, ami oszlopokat vagy vector-okat tartalmaz: másik data.frame, vector, factor, mátrix, vector-okat tartalmazó list
  - több forrás is kombinálható, de a vector-ok hossza azonos kell legyen; pl. df <- data.frame(matrix,list.of.vectors,név=vekt,...)
  - maximális információ öröklődik: hozza magával a más class-ból örökölt vector-ok name attribútumait is
  - létrehozáskor is elnevezhetők a sorok, opcionális row.names= argumentum – lehet egy oszlop neve/indexe, ha az adatforrás része; vagy egy vector, numeric vagy string elemekkel
- data.frame módosítása
  - tetszőleges sor vagy oszlop felüldefiniálható – pl. egy oszlop konverziója factor-rá – az input legyen megfelelő méretű vector/list; sor esetén adott indexen megfelelő típusok!
  - definiálhatunk új sort vagy oszlopot a hiányzó index/név definiálásával
    - név szerint definiálva a következő helyre kerül
    - index szerint definiálva: sor esetén az üres sorokat NA-val tölti fel, oszlop esetén error, ha üres oszlop maradna!
  - adatok szerkesztése szabadon egy Excel-szerű, grafikus editorban: fix(df), edit(df) – különbség: fix az eredetit szerkeszti, míg edit egy másolatot szerkeszt és ad vissza
  - törlés: negatív indexszel kihagyás, eredeti felülírása
    - sorok törlésénél figyeljünk, mert a korábbi oszlop indexek most row.names()-szé válnak, a látott „index” már nem egyenlő a sorszámmal! a látott szám szerint df["regi.sorindex",] szerint érjük el az adott sort!
- változók elérése, névtér...
  - sokszor bosszantó `my.data.frame$oszlop.neve` formában elérni az oszlopokat
  - attach("my.data.frame") láthatóvá teszi a az oszlopokat mint változókat
    - vigyázzunk, ha oszlop.neve valami nagyon általános, mint "x", és létezik már ilyen néven változónk! ilyenkor kapunk warning-ot is.
  - de! oszlop módosításához, új oszlop létrehozásához továbbra is kell a $-szintaxis!
  - attach() működik pl. list-re is

  - ha végeztünk a munkával, detach(df)-fel rejtsük el újra a változókat! – hogy a további munkánál ne zavarjon be a három különböző x nevű változó.
  - magyarázat: search path, namespace...
    * namespace, névtér: objektumok nevei egyediek kell legyenek – de lehet egy általános x változóm, és egy (vagy több) data.frame-en belül x nevű oszlopom – ezek különböző névterek. gondoljunk rá úgy, hogy a data.frame-en belül az oszlop egy lokális változó; vagy hogy kifelé maskolja, elrejti a változóit.
    * a háttérben: attach() hozzáadja a data.frame-et, mint namespace-t a search path-hoz, avagy hogy hol keresi a változókat – ezért lesznek előtag nélkül is láthatóak az adattagjai – ha nincs névütközés
    * alapvetően az R az általános névtérben hoz létre változókat, ezért kell módosításhoz, új oszlop definiálásához az előtag továbbra is!
    * search() paranccsal listázhatjuk a search path-on lévő névtereket – betöltött csomagokat és adathalmazokat, és hogy melyik milyen indexen/pozíción van
    * ls(index) vagy ls(név) listázza az adott névtérben foglalt objektumokat
    * detach() is működik név vagy pozíció szerint – detach() argumentum nélkül a 2. pozícióról töröl, tipikusan a legutóbb attach()-elt adathalmaz van ott
- data.frame olvasás külső fájlból
  - read.table("forrásfájl.kit"), vagy ha nem a munkamappában van a fájl: read.table("relatív/elérési/út/forrásfájl.kit")
  - feltételezzük, hogy megfelelő formátumú a fájl:
    * adatok data.frame-ként elrendezve: soronként egy-egy adatrekord, oszloponként azonos típusú változók
    * első sor: oszlopok nevei
    * alapértelmezésben felteszi, hogy az első oszlop a sorok nevei, és első sor első eleme üres
    * ha nincsenek sornevek, ezt header=TRUE opcionális argumentummal jelezzük
    * formátum: tipikusan tabulátorral tagolt .csv, .txt
- minta adatforrások R-en belül
  - data() paranccsal listázhatjuk az elérhető adathalmazokat – a legtöbbje data.frame, de akad pl. több-dimenziós array is
  - csomagokból még több
    * teljes csomag betöltése nélkül, csak az adathalmaz betöltése: data(adathalmaz.neve,package="csomag.neve")
    * vagy csomag betöltése package("csomag.neve"), és használható az adathalmaz + a csomag teljes tartalma
  - **hogyan ismerkedjünk egy adathalmazzal elemzés előtt:**
  - ?adathalmaz.neve – megnyitja a súgóban a leírást – minta mérete, oszlopok típusai, jelentése

– head() paranccsal megnézhetjük az első pár sort, hogy intuitív képet kapjunk róla, mi van benne
– edit()-tel megnyithatjuk grafikus szerkesztőben
– adathalmaz.neve + enter: teljes adathalmaz kiírása Console-ra – nagy adathalmazoknál átláthatatlan lehet