I. R alapismeretek

Tartalomjegyzék

1. Bevezetés 2
1.1. R interface-ek: RGui, RStudio 2
1.2. Munkafolyamat indítása, RData állomány létrehozása és mentése 2
1.3. Objektumok listázása és törlése 5
2. Értékadás 7
2.1. Egyelemű objektumok létrehozása 7
2.2. Vektorok 10
2.3. Mátrixok, tömbök 14
3. Adatgenerálás 20
4. Adatok fájlba íratása 22
5. R csomagok (R packages) 23
6. Felhasznált irodalom 23

1. Bevezetés

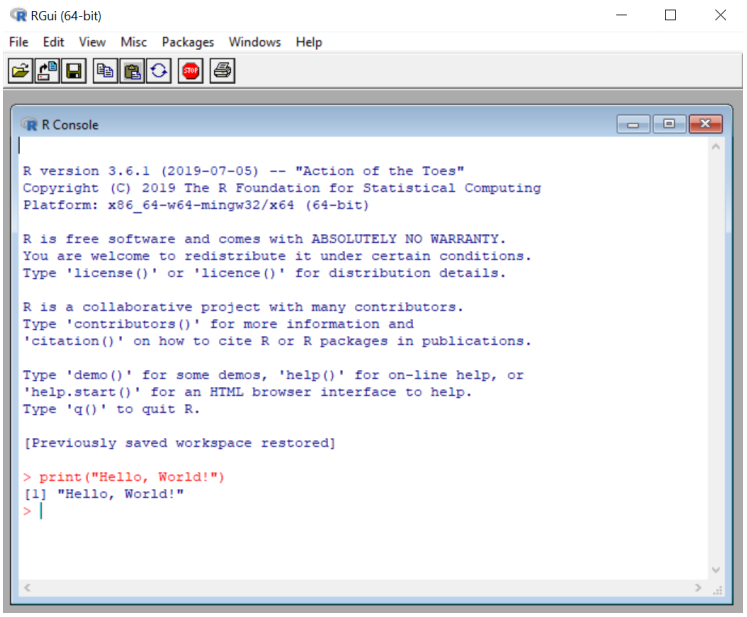
1.1. R interface-ek: RGui, RStudio

Az R programozási nyelv többek között regressziós modellek illesztésére, trendelemzésre, hipotézisvizsgálatra és többváltozós statisztikai vizsgálatokra, így faktor- és klaszteranalízisre használható. Az R rendelkezik grafikus megjelenítő eszközzel, ezáltal eredményeinket számos diagramtípus közül választva ábrázolhatjuk. Az R az évek folyamán nemcsak statisztikai, hanem más vizsgálatokra is alkalmassá vált, például gyökkeresési algoritmusok is elérhetők numerikus integráláshoz. Az alábbi jegyzetben idősor- és mezőelemzésre használjuk az R-t.

Az R programnyelvet 1993-ban alkották meg statisztikai feladatok elvégzésére, illetve az eredmények grafikus megjelenítésére. A programnyelv **nyílt forráskódú**, jól dokumentált. A legfrissebb stabil kiadása a 2019. december 12-én megjelent 3.6.2, verzió, amely Windows, Linux és Mac OS X operációs rendszer alatt a Comprehensive R Archive Network (CRAN) honlapjáról tölthető le (Windows alatt a keresendő állomány a *Download R for Windows* oldalon: R-3.6.2-win.exe). Ebben a jegyzetben a Windows alatt telepített R működésével foglalkozunk. A 64-bites verzió telepítését követően az R-t az RGui.exe állomány futtatásával indíthatjuk el, amely az Asztalon vagy a …\3.6.2\bin\x64 almappában található.

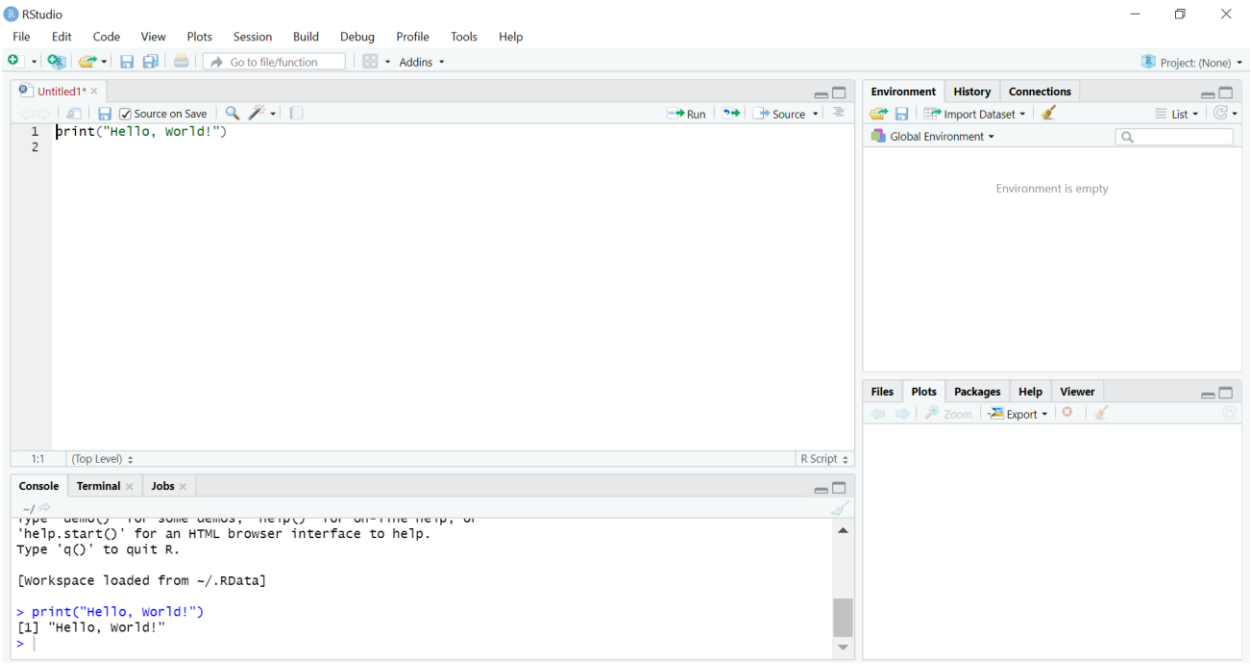
Az R-ben a kódok kétféleképpen értelmezhetők: **parancssori és script mód**ban. Előbbi esetben az R grafikus felhasználói felületének (graphical user interface, GUI) terminál ablakába (R Console) írjuk be a parancsokat és hajtatjuk végre azokat. Utóbbi esetben a script fájlban kerül tárolásra, amely a terminál ablakban hívható meg (*1. ábra*).

1. ábra. Az RGui 3.6.1 verziójának terminálablaka.



A munkavégzést megkönnyíti, ha az R **integrált fejlesztői környezet**ét (integrated development environment, IDE), az RStudiót használjuk, amelyben osztott képernyőn látjuk a terminál ablakot, a scriptet, a létrehozott **objektumok**at (például vektorokat, tömböket) és az elkészült ábrákat, illetve egyéb információkat. Ingyenes verziója az RStudio honlapjáról tölthető le (a keresendő állomány: RStudio-1.2.5033.exe). Fontos kiemelnünk, hogy az RStudio használatának előfeltétele az RGui telepítése. Ezt követően a ...\RStudio\bin almappában található rstudio.exe állománnyal indíthatjuk el (*2. ábra*).

2. ábra. Az RStudio IDE 1.2.5033 verziójának alapértelmezett felépítése. (A négypaneles nézet a View menü Panes/Show All Panes pontjában állítható vissza.)



1.2. Munkafolyamat indítása, RData állomány létrehozása és mentése

A **munkafolyamat** (session) az RStudio indításakor kezdődik és a leállításával végződik. A létrehozott objektumok együttese a **munkaterület** (workspace), amelyhez **munkakönyvtár** (working directory) tartozik. A továbbiakban a munkavégzés mappájaként hivatkozunk erre. A munkafolyamat során különböző **objektumok**at hozunk létre, például számokat, vektorokat, tömböket, valamint bonyolultabb egységeket, például lineáris modelleket, hipotézisvizsgálatok eredményét tartalmazó objektumokat.

Az RStudio felülete alapértelmezetten négy részre osztott. Először a bal oldalon található két ablakról lesz szó. A bal alsó, **Console ablak**ban az ENTER-rel jóváhagyott kódok egyesével kerülnek értelmezésre és végrehajtásra az R parancsértelmezője által. Az eredmény szintén ebben az ablakban jelenik meg.

Az R-ben **beépített** függvények állnak rendelkezésre a számítások végrehajtására, azonban számos kiegészítő csomag (package) is letölthető, amelyek különböző függvényeket tartalmaznak. A függvények szintaxisa: `függvénynév(paraméter)`. Például tartalom képernyőre való kiíratása a `print` függvénnyel történhet, amelynek gömbölyű zárójelekkel határolt argumentumába írhatjuk a megjelenítendő szöveget. Az 1. és a 2. ábrán a *Hello, World!* szöveg került kiíratásra a `print("Hello, World!")` utasítással.

Léteznek paraméter nélkül használható függvények. Ilyen esetben a két gömbölyű zárójel közötti részt üresen hagyjuk. Például az R programozási nyelv használatára való hivatkozás módja a `citation()` utasítással kérhető le. Másik példa az R lezárására használható `q()` vagy `quit()` függvény.

Az **R súgórendszere** a `help.start()` utasítással indítható el. Egy adott függvényre vonatkozó súgórészlet a függvények neve elé kérdőjel írásával, vagy a `help()` argumentumába a kérdéses függvény nevének beillesztésével jeleníthető meg. Például az összegzést végző `sum` függvényről információ a `?sum` vagy a `help(sum)` utasítással kérhető le. A keresett függvény dokumentációja az RStudio jobb alsó ablakában, a **Help** fülön jelenik meg. Ez tartalmazza többek között az adott függvény leírását, szintaxisát és példát az alkalmazására.

A telepített RGui *`R/R.3.5.2/Doc/Manual`* mappájában érhető el az R részletes dokumentációja.

Utasítások sorozatát célszerű a bal felső, **Script ablak**ba írnunk és scriptként mentenünk, hiszen így később is fel tudjuk használni. Script bármilyen szövegszerkesztővel írható, amelyből a vágólapra másolhatók az utasítások és beilleszthetők a Script ablakba. Az utasításainkhoz fűzött magyarázatainkat, azaz kommentjeinket egy vagy több kettőskereszt (#) karakterrel vezetjük be. A scriptbeli utasítások az azokat tartalmazó sorok kijelölését követően, a `Run` feliratra kattintva futtathatók le. Script megnyitásához és mentéséhez célszerű az RStudio menüsorát használnunk. Új script a `File/New File/Script` menüpontban hozható létre. Mentésére a `File/Save` menüpontban kerülhet sor. (A mentéskor célszerű UTF-8 kódolást használni.) Az R kiterjesztésű állomány szövegszerkesztővel (pl. Notepad-del) is megnyitható, szerkeszthető.

A munkafolyamat során létrehozott objektumok törlődnek, ha a munkaterületet nem mentjük. **Objektumainkat bináris, RData kiterjesztésű állományba menthetjük** a munkavégzés mappájába a `save.image("név.Rdata")` utasítással. Beolvasása a `load("név.Rdata")` utasítással lehetséges. A létrehozott fájl hordozható, vagyis másik gépen szintén megnyitható, ha rendelkezésünkre áll az R.

A Console ablakban a munkafolyamat alatt kiadott utasítások lapozhatók a felfelé és lefelé mutató nyilakkal. A visszahívott parancsok szerkeszthetők is, a sorokban a balra és jobbra mutató nyilakkal pozícionálhatjuk a kurzort. A teljes parancs végrehajtása ENTER-rel történhet, ehhez nem szükséges a sor végére navigálni. Az utasítások **Rhistory** kiterjesztésű szöveges

állományban menthetők a `savehistory("név.Rhistory")` paranccsal. Betöltése a `loadhistory("név.Rhistory")` paranccsal történhet.

Kerülendő, hogy a létrehozott objektumainkat a munkavégzés alapértelmezett mappájában található .RData elnevezésű állományba mentsük, ugyanis a RStudio újbóli elindításakor ezek automatikusan betöltésre kerülnek, függetlenül attól, hogy szükségünk van ezekre az új munkafolyamat során vagy sem.

A **munkavégzés mappájának neve, elérési útja megjeleníthető** a `getwd()` utasítással. A munkafolyamat során létrehozott állományok (például png ábrák, txt fájlok) ebbe a mappába fognak mentődni.

A **munkavégzés mappájában található állományok listázása** a `list.files()` utasítással lehetséges. Ha nincs állomány a mappában, akkor a `character(0)` eredményt kapjuk.

A `setwd` függvénnyel az **alapértelmezett mappától eltérő mappát választhatunk** a munkavégzés mappájaként. A függvény szintaxisa:
`setwd("elérési út")`

Az elérési útban a meghajtó azonosítóját, majd a mappák, almappák nevét vagy egy normál irányú perjel (/), vagy két fordított perjel (\\) választja el, azaz:
`meghajtó azonosítója:\\főmappanév\\almappanév` vagy
`meghajtó azonosítója:/főmappanév/almappanév`

Mivel az elérési út karakterlánc, ezért idézőjelek vagy aposztrófok közé teendő.

Ahogyan korábban említettük, a munkafolyamat befejezésére a `q()` vagy a `quit()` utasításokkal kerülhet sor. Az RStudio jobb felső sarkában a bezárás ikonra is kattinthatunk.

1.3. Objektumok listázása és törlése

A létrehozott objektumok az `ls()` vagy az `objects()` utasítással listázhatók. Ha nincs tárolt objektum, akkor a Console ablakban a `character(0)` szerepel. A létrehozott objektumok az RStudio jobb felső ablakának **Environment** fülén láthatók.

Az objektumok törlése az `rm` függvénnyel lehetséges, argumentumában felsorolva a törlendő objektumokat, vesszővel elválasztva. Az összes létrehozott objektum az `rm(list=ls())` paranccsal törölhető. (A törlést tehát két függvény egymásba ágyazásával értük el. A `list=ls()` paranccsal arra utasítottuk az R-t, hogy az objektumok neveit fűzze össze egy karakterlánc típusú vektorba, majd a törlést az `rm` függvénnyel hajtottuk végre.)

Megjegyzések:

- Az R-ben a sorok a > prompt karakterrel kezdődnek.
  Be nem fejezett utasításokra sortörést alkalmazva az új sor + jellel kezdődik > helyett.
- A szintaktikailag teljes utasítások sortöréssel végződnek.
- Az utasítás bevitele az Esc billentyűvel szakítható meg.
- Az R kis- és nagybetű érzékeny, így létrehozhatunk különböző értékű, `a` és `A` elnevezésű objektumot.
- A megnyitandó állományok nevében, elérési útjában található ékezetek, szóközök általában nem okoznak problémát, de ezek használata inkább kerülendő.
- Az R-ben a tizedesjel alapértelmezetten pont, nem vessző. (Állomány beolvasásakor ez átállítható, például a `read.table` függvény használata esetén.)
- A Console felület a Ctrl+l utasítással törölhető.
- A script több sora elé kettőskeresztet a sorok kijelölése után a Ctrl+Shift+c betűkombinációval tehetünk.

A továbbiakban egyszerű feladatokat oldunk meg, amelyek segítségével gyakoroljuk az értékadást és a létrehozott objektumokon néhány alapvető műveletet.

2. Értékadás

2.1. Egyelemű objektumok létrehozása

Az R-ben létrehozott objektumok lehetnek többek között egyeleműek vagy vektorértékűek – numerikus (numeric), szöveg (character) vagy logikai (logical) – mátrixok (matrix), tömbök (array), adatstruktúrák (data frame), listák. Utóbbiak abban különböznek a mátrixoktól és a tömböktől, hogy oszlopaikban eltérő típusú adatok (pl. számok, dátumok és szövegek) is szerepelhetnek.

Számos speciális típusú objektum létezik. Ilyen például a ncdf4 típusú objektum, amely NetCDF fájlokban tárolt többdimenziós tömbök meta adatait tartalmazhatja. A lista (list) típusú objektumokban különböző hosszúságú vektorok tárolhatók.

1. feladat: Hozzuk létre az üres `proba` mappát a Windows Intézőben és állítsuk be az R-ben a munkavégzés mappájaként!

- Jelenítsük meg a munkavégzés alapértelmezett mappájának nevét és helyét a `getwd()` parancs kiadásával!
- A munkavégzés mappájaként állítsuk be a `proba` mappát! Ehhez a `setwd` függvényt használjuk!
- Ellenőrizzük ismét a `getwd()` függvénnyel, hogy sikerült-e beállítani a `proba` mappát a munkavégzés mappájaként. Ellenőrizzük, hogy üres-e a mappa, vagyis listázzuk ki a mappában található állományokat a `list.files()` utasítással!
- Győződjünk meg arról az `ls()` függvénnyel, hogy nem állnak rendelkezésünkre objektumok.

Értékadás nélkül az R megjeleníti egy művelet eredményét, de nem tárolja el azt.

Alapvető műveletek:
Két szám összege: `x+y`
Két szám különbsége: `x-y`
Két szám szorzata: `x*y`
Két szám hányadosa: `x/y`
Hatványozás: Ha `x` a hatvány alapja és `y` a kitevője, akkor: `x**y` vagy `x^y`
A műveletek `()` zárójelekkel csoportosíthatók, így a műveleti sorrend megváltoztatható.

2. feladat: Adjunk össze két számot, például ötöt és nyolcat!

```
5+8
```

Az eredmény:

```
[1] 13
```

A parancs utáni sorban jelenik meg az eredmény. A szögletes zárójelekben található `[1]` arra utal, hogy az eredmény sorvektor (ebben az esetben egy komponensű vektor, azaz skalár).

Adjuk össze a két számot a `sum` függvénnyel is!

```
sum(5,8)
[1] 13
```

Az eredmény eltárolásához objektumba kell azt mentenünk. Értékadás az alábbi módokon lehetséges:

```
objektumnév <- kifejezés
kifejezés -> objektumnév
objektumnév = kifejezés
```

A jegyzetben rendszerint az első lehetőséget fogjuk választani. Egymás után több, azonos nevű objektumot létrehozva, az utóbbi felülírja az előbbit. Mivel az R kis- és nagybetű érzékeny, ugyanaz a név kis- és nagybetűvel írva különböző objektumokat fog jelölni.

3. feladat: Tároljuk el a 10 és a 20 számot objektumként, majd jelenítsük meg a képernyőn!

```
objektum <- 10
OBJEKTUM <- 20
objektum
[1] 10
OBJEKTUM
[1] 20
```

Az objektumok, változók nevében szerepelhet pont, alulvonás és különleges karakter is, utóbbiak használata azonban kerülendő. Ha a név ponttal kezdődik, akkor a második karakter nem lehet szám. Névként nem használhatók az alábbi „foglalt” kifejezések, noha egy objektumnév részeként alkalmazhatók: *TRUE, FALSE, NULL, Inf, NaN, NA, NA_integer_, NA_real_, NA_complex_, NA_character_*. (A kis- és nagybetű érzékenység miatt kizárólag a teljes egyezés kerülendő az előbb felsorolt szavakkal.)

Beépített állandók: az R-ben kevés beépített változó érhető el. Ezek az alábbiak:

- `pi`: értéke: 3,141593
- `letters`: az angol abc kisbetűi
- `LETTERS`: az angol abc nagybetűi
- `month.name`: a hónapok nevei angolul
- `month.abb`: a hónapok hárombetűs rövidítése angolul

Egyéb konstansok például függvényekkel definiálhatók.

4. feladat: Adjuk meg az Euler-féle számot az `exp` függvénnyel (az exponenciális függvény 1 helyen felvett értéke), tároljuk `e` elnevezésű objektumként és írassuk ki a képernyőre!

```
e <- exp(1)
e
[1] 2.718282
```

5. feladat: Tároljuk `x` és `y` objektumként 1+2 és 2*5 eredményét, majd mentsük ezek hányadosát `z` objektumként, illetve jelenítsük meg a képernyőn!

```
x <- 1+2
y <- 2*5
z <- x/y
z
[1] 0.3
```

Előbbi esetben több utasítást adtunk meg sortöréssel, amelyeket az R egymás után értelmezett.

Több utasítás megadása esetén az utasítások **blokk**okba szervezhetők, kapcsos zárójelek használatával.

```
{x <- 1+2
y <- 2*5
z <- x/y
}
z
[1] 0.3
```

A `{` után a `>` prompt jel + jelre változik. Az utasítások blokkját `}` fejezi be. Az utasítások tehát ezt követően hajtódnak végre. Utasítások blokkokba szervezésére ciklusok esetén lehet szükségünk.

Az objektumok típusának, dimenziójának meghatározása:

Az objektumok típusa és dimenziója (kettő vagy annál több dimenziós objektum esetén) például a `class` és `dim` függvényekkel jeleníthető meg. Szintaxisuk:

```
class(objektumnév)
dim(objektumnév)
```

6. feladat: Ellenőrizzük az `x`, `y` és `z` objektumok típusát a `class` függvénnyel!

```
class(x)
[1] "numeric"

class(y)
[1] "numeric"

class(z)
[1] "numeric"
```

Bár `x`, `y` egész számok, míg z valós szám, mindhárom objektum típusa `numeric`.

Szöveg, illetve logikai értékű objektumok is létrehozhatók, amelyek típusa `character`, illetve `logical`.

7. feladat: Hozzuk létre az `abracim` objektumot `proba.png` és az `igaz` objektumot `TRUE` logikai értékkel, majd ellenőrizzük a típusukat!

```
abracim <- "proba.png"
igaz <- TRUE

class(abracim)
[1] "character"

class(igaz)
[1] "logical"
```

Karakterlánc mindig idézőjelek vagy aposztrófok között definiálandó!

Kiegészítés:
A maradékos osztás a `%%` operátorral, az egész osztás a `%/%` operátorral végezhető el. Például `9%%2` azt adja meg, hogyha kilencet osztjuk kettővel, akkor mennyi lesz a maradék, míg `9%/%2` azt adja meg, hogy hány egész számszor van meg kilencben a kettő.

2.2. Vektorok

Több érték vagy azonos típusú objektum „összefűzésére” (konkatenációjára) a `c` függvény használható, amelynek szintaxisa:
`c(érték1, érték2, érték3,...)`

A függvénnyel számokból például numerikus vektorok készíthetők.

Numerikus vektor:
8. feladat: Hozzunk létre két, egész számokból álló vektort és tároljuk ezeket objektumokként, majd ellenőrizzük a típusukat!

```
a <- c(1,2,3,4,5)
b <- c(5,4,3,2,1)
a
[1]  1  2  3  4  5
b
[1] 5  4  3  2  1

class(a)
[1] "numeric"
```

```
class(b)
[1] "numeric"
```

A numerikus vektorok típusa ugyanúgy `numeric`, mint ahogyan korábban skalár esetén láttuk.

Előbbi objektumok létrehozhatók a következő módon is, mivel a számsorozatok szabályosak, tagjaik egyenlő távolságra vannak egymástól.

```
a2 <- 1:5
b2 <- 5:1
a2
[1] 1 2 3 4 5
b2
[1] 5 4 3 2 1
```

Az elemek tehát kettőspont alkalmazásával sorolhatók fel.

Fűzzük össze az `a2` és a `b2` vektort a `c` függvény segítségével! (Ne tároljuk objektumként.)

```
c(a2,b2)
[1] 1 2 3 4 5 5 4 3 2 1
```

A vektor eltolható, nyújtható, zsugorítható:

9. feladat: Hozzunk létre a [0,9] intervallumon szabályos számsorozatot, az [1,10] intervallumon létrehozott számsorozat eltolásával és tároljuk objektumként! (Fontos: A `c` függvény mellett létrehozhatunk `c` nevű objektumot is. Egyidejű létezésük nem okoz problémát.)

```
c <- 1:10-1
c
[1] 0 1 2 3 4 5 6 7 8 9
```

Nyújtsuk a kétszeresére az `a` vektort és zsugorítsuk a felére a `b` vektort! Tároljuk ezeket is objektumokként!

```
d <- a*2
d
[1]  2  4  6  8 10
```

```
d2 <- b/2
d2
[1] 2.5 2.0 1.5 1.0 0.5
```

Alapműveletek a numerikus vektorokkal is végezhetők:

10. feladat: Végezzük el az alábbi műveleteket a vektorokkal!

Két vektor komponensenkénti összege, pl.

```
d+d2
[1] 4.5  6.0  7.5  9.0 10.5
```

Két vektor komponensenkénti különbsége, pl.

```
a-b
[1] -4 -2  0  2  4
```

Két vektor komponensenkénti szorzata, pl.

```
 d*d2
[1] 5 8 9 8 5
```

Megjegyzés: két, különböző hosszúságú vektor esetén a rövidebb vektor komponenseit újból felsorolja az R és ezekre hajtja végre a további műveleteket, pl.

```
a+c
[1]  1  3  5  7  9  6  8 10 12 14
```

Vektor szorzása skalárral, pl.

```
a*9
[1]  9 18 27 36 45
```

Skalárszorzás, pl.

```
skalarszorz <- a%*%b
skalarszorz
      [,1]
[1,]   35
```

Ellenőrizzük le a `skalarszorz` objektum típusát!

```
class(a%*%b)
[1] "matrix"
```

Az eredményül kapott egyelemű mátrix a `drop` vagy az `as.numeric` függvénnyel alakítható skalárrá. Alakítsuk skalárrá és jelenítsük meg az objektum típusát a `class` függvénnyel!

```
drop(skalarszorz)
[1] 35
```

```
class(drop(skalarszorz))
[1] "numeric"
```

Egy objektum, így vektor adott komponense vagy komponensei a `[]` zárójelek használatával jelölhető ki.

```
objektum[sorszám]
objektum[c(sorszám_1,sorszám_2)]
objektum[sorszám_1:sorszám_n]
```

Vektor adott komponense vagy komponensei ki is hagyhatók a felsorolásból:

```
objektum[-sorszám]
objektum[c(-sorszám_1,-sorszám_2)]
```

(Ezek természetesen új objektumokként is tárolhatók.)

11. feladat: Jelenítsük meg az `a` vektor 5. komponensét, majd a `b` vektor 1. és 4. komponensét, végül a c vektor 2. és 8. komponense közötti értékeket!

```
a[5]
[1] 5

b[c(1,4)]
[1] 5 2

c[2:8]
[1] 1 2 3 4 5 6 7
```

Nullvektor a `c` függvény alkalmazásával állítható elő:

```
null <- c()
null
NULL
```

Ha egy vektort adatokkal szeretnénk feltölteni, akkor célszerű előbb lenulláznunk a vektort, azaz:

```
null <- c(0)
```

Karakterek sorozatából álló vektor (string, karakterlánc):

12. feladat: Hozzunk létre karakterláncból álló vektort a `c` függvény segítségével!

```
gyumolcs <- c("alma","mandarin","szilva","narancs")
gyumolcs
[1] "alma" "mandarin" "szilva" "narancs"
```

Ennek a vektornak is megjeleníthető adott sorszámú komponense, illetve lekérhető a típusa:

```
class(gyumolcs)
[1] "character"
```

Logikai értékű vektor: TRUE és FALSE értékekből álló vektorok

13. feladat: Hozzuk létre a „hamis”, „igaz”, „igaz” logikai értékű vektort a `c` függvénnyel és tároljuk objektumként!

```
logikai <- c(FALSE,TRUE,TRUE)
logikai
[1] FALSE  TRUE  TRUE
```

Ha az előbb felsorolt értékeket idézőjelek közé helyezzük, akkor karakterlánc típusú vektort hozunk létre.

Faktorok:
Olyan speciális objektumok, amelyek osztályokba sorolják a vektorok komponenseit. A faktor tehát numerikus kódolású vektort jelent szövegértékű szintekkel.

Például varianciaanalízis (analysis of variance, ANOVA) esetén az összehasonlítandó csoportok megkülönböztetendők valamilyen tényező (faktor) alapján. Ilyen lehet a nem, a kor, az időszak stb. Tegyük fel, hogy januári, februári és márciusi adatokat csoportosítunk. A hónapok nevei karakterláncként állnak rendelkezésünkre a `honapok` objektumban. Faktor típusú változó az `as.factor` függvénnyel hozható létre.

```
honapok <- c("január", "február", "március", "január",
"február", "március","január", "február", "március")

honapok_faktor <- as.factor(honapok)

class(honapok_faktor)
[1] "factor"
```

Megjegyzés: az `as.factor` függvény tehát az `as.numeric` függvényhez hasonlóan a típuskonverziós függvények közé tartozik.

2.3. Mátrixok, tömbök

Egy mátrix vagy tömb lehet többek közt numerikus, karakterlánc, logikai, de egy tömbön belül többféle adattípus nem alkalmazható.

Létrehozásuk módja:
```
matrix(objektum vagy érték, nrow=sorok száma, ncol=oszlopok
száma, byrow = FALSE vagy TRUE)
```
vagy:
```
array(objektum vagy érték, dim=c(dim_1, dim_2, ..., dim_n))
```

Kétdimenziós tömbök:
14. feladat: Állítsuk elő az alábbi két mátrixot!
Az `A` mátrix legyen 10x10-es, és töltsük fel az 1, 2, …, 100 számokkal!
```
A <- array(1:100, dim=c(10,10))
A
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1   11   21   31   41   51   61   71   81    91
 [2,]    2   12   22   32   42   52   62   72   82    92
 [3,]    3   13   23   33   43   53   63   73   83    93
 [4,]    4   14   24   34   44   54   64   74   84    94
 [5,]    5   15   25   35   45   55   65   75   85    95
 [6,]    6   16   26   36   46   56   66   76   86    96
 [7,]    7   17   27   37   47   57   67   77   87    97
```

```
 [8,]    8   18   28   38   48   58   68   78   88    98
 [9,]    9   19   29   39   49   59   69   79   89    99
[10,]   10   20   30   40   50   60   70   80   90   100
```

A `B` mátrix legyen 10x10-es, és töltsük fel a 100,99, …, 1 elemekkel!

```
B <- matrix(100:1, nrow=10, ncol=10)
B
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]  100   90   80   70   60   50   40   30   20    10
 [2,]   99   89   79   69   59   49   39   29   19     9
 [3,]   98   88   78   68   58   48   38   28   18     8
 [4,]   97   87   77   67   57   47   37   27   17     7
 [5,]   96   86   76   66   56   46   36   26   16     6
 [6,]   95   85   75   65   55   45   35   25   15     5
 [7,]   94   84   74   64   54   44   34   24   14     4
 [8,]   93   83   73   63   53   43   33   23   13     3
 [9,]   92   82   72   62   52   42   32   22   12     2
[10,]   91   81   71   61   51   41   31   21   11     1
```

A mátrix alapértelmezetten oszloponként kerül feltöltésre az adatokkal. A `matrix` függvény `byrow=TRUE` paraméterével állítható be soronkénti feltöltés. Többek között mátrixok adatainak térképes ábrázolásakor fontos, hogy tudjuk, milyen rend szerint tölti fel az R a mátrixot adatokkal, hiszen ennek hiányában tévesen pozícionálhatjuk az értékeket.

A két, különböző függvénnyel létrehozott tömb mátrix típusú:

```
class(A)
[1] "matrix"
class(B)
[1] "matrix"
```

Ha a sorok és az oszlopok számának szorzata kisebb, mint a mátrix általunk definiált elemszáma, akkor a szükségesnél kevesebb elemet fog tartalmazni a mátrix vagy az adatsor csonkolódik. Például:

```
C <- matrix(1:100, nrow=2, ncol=5)
C
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Megjegyzés: Idősorok és mezők elemzése során a számítások eredményeit tömbben tárolhatjuk. A tömb szerkezetének pontos ismerete hiányában az adatok tévesen kerülhetnek ábrázolásra, téves következtetéseket vonhatunk le.

Mátrix elemei között is végezhetők alapvető műveletek:

15. feladat: Szorozzuk meg az `A` mátrix komponenseit egy skalárral, nevezetesen kettővel!

```
D <- matrix(2*1:100, nrow=10, ncol=10) # vagy
D <- 2*A
D
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    2   22   42   62   82  102  122  142  162   182
 [2,]    4   24   44   64   84  104  124  144  164   184
 [3,]    6   26   46   66   86  106  126  146  166   186
 [4,]    8   28   48   68   88  108  128  148  168   188
 [5,]   10   30   50   70   90  110  130  150  170   190
 [6,]   12   32   52   72   92  112  132  152  172   192
 [7,]   14   34   54   74   94  114  134  154  174   194
 [8,]   16   36   56   76   96  116  136  156  176   196
 [9,]   18   38   58   78   98  118  138  158  178   198
[10,]   20   40   60   80  100  120  140  160  180   200
```

Mátrixműveletek:

16. feladat: **Transzponáljuk az A mátrixot!**

```
t(A)
```

```
    [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    2    3    4    5    6    7    8    9    10
 [2,]   11   12   13   14   15   16   17   18   19    20
 [3,]   21   22   23   24   25   26   27   28   29    30
 [4,]   31   32   33   34   35   36   37   38   39    40
 [5,]   41   42   43   44   45   46   47   48   49    50
 [6,]   51   52   53   54   55   56   57   58   59    60
 [7,]   61   62   63   64   65   66   67   68   69    70
 [8,]   71   72   73   74   75   76   77   78   79    80
 [9,]   81   82   83   84   85   86   87   88   89    90
[10,]   91   92   93   94   95   96   97   98   99   100
```

17. feladat: **Adjuk meg `A` és a `B` mátrixszorzatát!**

```
A %*% B
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8] [,9] [,10]
 [1,] 43105 38505 33905 29305 24705 20105 15505 10905 6305  1705
 [2,] 44060 39360 34660 29960 25260 20560 15860 11160 6460  1760
 [3,] 45015 40215 35415 30615 25815 21015 16215 11415 6615  1815
 [4,] 45970 41070 36170 31270 26370 21470 16570 11670 6770  1870
 [5,] 46925 41925 36925 31925 26925 21925 16925 11925 6925  1925
 [6,] 47880 42780 37680 32580 27480 22380 17280 12180 7080  1980
 [7,] 48835 43635 38435 33235 28035 22835 17635 12435 7235  2035
 [8,] 49790 44490 39190 33890 28590 23290 17990 12690 7390  2090
 [9,] 50745 45345 39945 34545 29145 23745 18345 12945 7545  2145
[10,] 51700 46200 40700 35200 29700 24200 18700 13200 7700  2200
```

Mátrixszorzásra például főkomponens-elemzést követően, az eredeti adatsor főkomponensekkel való közelítésekor lehet szükségünk.

18. feladat: Invertáljuk a `solve` függvénnyel az alábbi módon definiált `E` mátrixot!

```
E <- matrix(c(4,3,2,1), nrow=2, ncol=2)
E
     [,1] [,2]
[1,]    4    2
[2,]    3    1

solve(E)
     [,1] [,2]
[1,] -0.5    1
[2,]  1.5   -2
```

Mátrix adott sora, oszlopa vagy eleme a `[]` zárójelekkel jeleníthető meg:

```
objektum[sorszám,]
objektum[,oszlopszám]
objektum[sorszám,oszlopszám]
```

Mátrix adott sora vagy oszlop a következőképpen hagyható ki a megjelenítésből:

```
objektum[-sorszám,]
objektum[,-oszlopszám]
```

Megjegyzés: Számítások elvégzése vagy adatábrázolás során szükség lehet a tömb adott részének kijelölésére.

19. feladat: Jelenítsük meg a `D` mátrix első oszlopát, első sorát, az első elemét, végül pedig a második, harmadik sorának és második, harmadik, negyedik oszlopának közös elemeit!

```
D[,1]
[1]  2  4  6  8 10 12 14 16 18 20

D[1,]
[1]   2  22  42  62  82 102 122 142 162 182

D[1,1]
[1] 2

D[2:3,2:4]
     [,1] [,2] [,3]
[1,]   24   44   64
[2,]   26   46   66
```

Vektorok, mátrixok összekapcsolása:
Vektorok mátrixokká fűzhetők össze, illetve vektor mátrixszal, valamint mátrix mátrixszal is összekapcsolható a `cbind` és az `rbind` függvényekkel.

Egy mátrix, vagy vektor típusú objektumhoz egy mátrix objektum kijelölt oszlopának kapcsolására példa:

```
objektum <- cbind(objektum1, objektum2[,oszlopszám])
```

Egy mátrix vagy vektor típusú objektumhoz Egy mátrix objektum kijelölt sorának kapcsolására példa:

```
objektum <- rbind(objektum1, objektum2[sorszám,])
```

20. feladat: Készítsünk új mátrixot az `A` mátrix első és a `B` mátrix második sorából!

```
F <- rbind(A[1,], B[2,])
F
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1   11   21   31   41   51   61   71   81    91
[2,]   99   89   79   69   59   49   39   29   19     9
```

Az `A` mátrix és a `B` mátrix sorainak azonos elemszámúnak kell lennie.

Többdimenziós tömbök:
Az `array` függvénnyel kettőnél több dimenziós tömböket is létrehozhatunk.

21. feladat: Hozzuk létre a `tomb` nevű háromdimenziós, 10x10x10-es tömböt és töltsük fel 1-től 1000-ig a természetes számokkal!

```
tomb <- array(1:1000, dim=(c(10,10,10)))
```

Ellenőrizzük a típusát és dimenzióját!

```
class(tomb)
[1] "array"
```

```
dim(tomb)
[1] 10 10 10
```

A tömb 10 darab kétdimenziós táblázatból áll. Jelenítsük meg az 1. és a 10. táblázatot a következőképpen!

```
tomb[,,1]
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1   11   21   31   41   51   61   71   81    91
 [2,]    2   12   22   32   42   52   62   72   82    92
 [3,]    3   13   23   33   43   53   63   73   83    93
 [4,]    4   14   24   34   44   54   64   74   84    94
 [5,]    5   15   25   35   45   55   65   75   85    95
 [6,]    6   16   26   36   46   56   66   76   86    96
 [7,]    7   17   27   37   47   57   67   77   87    97
 [8,]    8   18   28   38   48   58   68   78   88    98
 [9,]    9   19   29   39   49   59   69   79   89    99
[10,]   10   20   30   40   50   60   70   80   90   100
```

```
tomb[,,10]
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]  901  911  921  931  941  951  961  971  981   991
 [2,]  902  912  922  932  942  952  962  972  982   992
 [3,]  903  913  923  933  943  953  963  973  983   993
 [4,]  904  914  924  934  944  954  964  974  984   994
 [5,]  905  915  925  935  945  955  965  975  985   995
 [6,]  906  916  926  936  946  956  966  976  986   996
 [7,]  907  917  927  937  947  957  967  977  987   997
 [8,]  908  918  928  938  948  958  968  978  988   998
 [9,]  909  919  929  939  949  959  969  979  989   999
[10,]  910  920  930  940  950  960  970  980  990  1000
```

Esetenként szükségünk lehet adott dimenziójú, üres tömb létrehozatalára. Ekkor az `array` függvény argumentumából a tömbbe töltendő adatokat elhagyjuk, azaz, pl. 10x10x10-es üres (`NA`-val kitöltött) tömb létrehozatalára a következő utasítás adandó ki:

```
tomb <- array(dim=(c(10,10,10)))
```

3. Adatgenerálás

A `seq` függvénnyel szabályos sorozatok hozhatók létre:

```
seq(from=sorozat kezdete, to=sorozat vége,
by=növekedés/csökkenés mértéke)
```

22. feladat: Hozzunk létre az [1,20] intervallumon kettesével növekvő számsorozatot!
Megjegyzés: A `from=`, `to=`, `by=` kódrészek kiírása nem szükséges, elegendő megadni a kezdeti és a végső értéket, valamint a növekedés vagy a csökkenés mértékét.

```
sor1 <- (seq(1,20,2))
sor1
[1]  1  3  5  7  9 11 13 15 17 19
```

Hozzunk létre a [20,1] intervallumon kettesével csökkenő számsorozatot!

```
sor2 <- (seq(20,1,-2))
sor2
 [1] 20 18 16 14 12 10  8  6  4  2
```

A sorozat tagjainak sorrendje a `rev` függvénnyel is megfordítható:

23. feladat: A `sor3` legyen a [100,0] intervallumon tízesével csökkenő számok sorozata!

```
sor3 <- rev(seq(0,100,10))
sor3
[1]  100  90  80  70  60  50  40  30  20  10   0
```

A `rep` függvény egy sorozat valahányszor történő ismétlésére alkalmazható:

```
rep(ismétlendő sorozat, times vagy each=ismétlődés száma)
```

24. feladat: Hozzunk létre az [1,5] intervallumon egyesével növekvő sorozatot, ismételjük meg kétszer a teljes sorozatot egymás után és írassuk ki a képernyőre!

```
sor4 <- 1:5
sor4
[1] 1 2 3 4 5
sor5 <- rep(sor4, times=2)
sor5
[1] 1 2 3 4 5 1 2 3 4 5
```

Előbbi két művelet egymásba ágyazva is elvégezhető:

```
sor4 <- rep(1:5, times=2)
sor4
[1] 1 2 3 4 5 1 2 3 4 5
```

Az `each` paraméterrel minden egyes elem egymás után is megismételhető. Például ismételjük meg a sor4 objektum minden elemét egymást követően kétszer!

```
rep(sor4, each=2)
[1] 1 1 2 2 3 3 4 4 5 5 1 1 2 2 3 3 4 4 5 5
```

A létrehozott objektumok hossza a `length` függvénnyel határozható meg.

25. feladat: Ellenőrizzük a tíz tagból álló `sor4` sorozat hosszát!

```
length(sor4)
[1] 10
```

Az R-ben létrehozhatók adott valószínűségi eloszlású véletlenszám-sorozatok is:

Az R-be épített eloszlások és szintaxisuk az alábbi címen érhetők el, általánosan a generálandó adatsor elemszáma és az eloszlás paraméterei adandók meg.

https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Distributions.html

26. feladat: Hozzuk létre az alábbi eloszlású számsorozatokat!

Geometriai eloszlású, 100 tagú számsorozat képzése, p= 0,5 valószínűséggel:

```
rgeom(100, p=1/2)
```

Poisson-eloszlású, 100 tagú számsorozat képzése, $\lambda = 5$ paraméterrel:

```
rpois(100, lambda=5)
```

Normális eloszlású, 200 tagú számsorozat képzése, 0 átlaggal és 1 szórással:

```
rnorm(200, mean=0, sd=1)
```

Egyenletes eloszlású, 200 tagú számsorozat képzése, a [0,1] zárt intervallumon:

```
runif(200, min=0, max=1)
```

A véletlen számsorozat reprodukálhatóságára az előző utasításokat megelőzően a `set.seed(azonosítószám)` parancs adandó ki.

Például a következő két utasítást adjuk ki háromszor!

```
set.seed(1)
runif(200, min=0, max=1)
```

4. Adatok fájlba íratása

Adatok fájlba való kiíratására alkalmas a `cat` függvény .

27. feladat: Írassuk ki fájlba, hogy „Hello, világ!”, két sorban!

```
cat("Hello", file="proba.txt", sep="\n")
cat(" világ!", file="proba.txt", append=TRUE)
```

Sortörés a `sep="\n"` paraméterrel, szóköz a `sep="\ "` paraméterrel adható meg:

Az `append=TRUE` paraméter nem kapcsolja össze a sorokat, csak az utolsó sor tartalma, a példa esetében a `"világ!"` szó belekerül az állományba. Az alapértelmezett beállítás: `append=FALSE`.

A `write.table` függvénnyel is kiírathatók az adatsorok fájlba.

```
write.table(objektumnév, "fájlnév.txt", sep="", na="NA",
dec=",", col.names=TRUE vagy FALSE, row.names=TRUE vagy FALSE)
```

`objektumnév:` Az objektum, amit ki szeretnénk íratni.
`"fájlnév.txt":` A létrehozandó állomány neve.

Adott sor elemeit tagoló szimbólumot a `sep` paraméterrel lehet szabályozni.
`sep=""` – tagolás tabulátorral vagy szóközzel (alapértelmezett)
`sep=","` – tagolás vesszővel
`sep=";"` – tagolás pontosvesszővel

A tizedesjel a `dec` paraméterrel állítható be, alapértelmezetten pont.

Hibás vagy hiányzó értéket helyettesítő szöveg vagy szám az `na` paraméterrel állítható be. (A fenti esetben a létrehozott állományban `NA` jelölné ezeket az értékeket.)

`col.names:` Az oszlopok azonosítóit tartalmazza-e az állomány. Alapértelmezetten igen.
`row.names:` A sorok azonosítóit tartalmazza-e az állomány. Alapértelmezetten igen.

A fájl nemcsak txt kiterjesztésű lehet, hanem például prn vagy csv is. Utóbbi kiterjesztésű állomány létrehozatalára a `write.csv` függvény is alkalmazható, a `write.table` függvényhez hasonló szintaktikával.

28. feladat: Írassuk ki fájlba az `A` mátrixot, sorazonosítók nélkül, tabulátorral tagolva, oszlopazonosítókkal!

```
write.table(A, "adat.txt", sep="\t", row.names=FALSE)
```

(A `row.names=FALSE` nélkül több oszlopunk volna, mint oszlopazonosítónk.)
Az adatok vágólapra is másolhatók és táblázatkezelőbe illeszthetők.
Az állomány a munkavégzés mappájába mentődik.

5. R csomagok (R packages)

Az R-ben telepíthetők kiegészítő csomagok, azaz package-ek, amelyek az R-ben alapértelmezetten el nem érhető algoritmusokat, függvényeket tartalmaznak. A csomagok saját dokumentációval rendelkeznek, amelyek – használat esetén – feltüntetendők publikációink hivatkozáslistájában. A hivatkozás a legegyszerűbben a `citation` függvénnyel kérdezhető le, azonban célszerű a CRAN honlapján is leellenőrizni.

```
citation("csomagnév")
```

Telepítés:

A telepítés legegyszerűbb módja az RStudióban a `Tools/Install packages.../` menüpont `Packages` mezőjében a telepítendő csomagok felsorolása szóközzel vagy vesszővel. A csomagok legördülő menüből is kiválaszthatók.

A CRAN mirror megválasztásakor általában a földrajzilag legközelebbi hely kijelölése javasolt.

A csomagok a Console felületen is telepíthetők az alábbi módon:

```
install.packages("csomagnév")
```

A csomagok a CRAN honlapjáról letöltve (Windows alatt r-release zip fájlok), közvetlenül a saját gépünkre másolva is telepíthetők. Ez akkor lehet célszerű, ha nem a legújabb R verzióval dolgozunk és régebbi verziószámú csomagot kell telepítenünk. Utóbbiak a https://cran.r-project.org/src/contrib/Archive/ oldalon találhatók meg.

Ha a Windows-ban rendelkezünk rendszergazdai jogosultsággal, akkor a kitömörített mappa a telepített R programot tartalmazó könyvtár *Library* mappájába másolandó, majd az alábbi utasítással indítható a telepítés:

```
install.packages("csomagnév")
```

A csomagot tetszőleges mappába is telepíthetjük. Ekkor a telepítés és a betöltés módja:

```
install.packages("csomagnév", lib="/elérési út/")
library(csomagnév, lib.loc="/elérési út/")
```

Ha korábban már letöltöttük a csomagokat, akkor internetkapcsolat nélkül is telepíthetők:

```
install.packages("csomagnév.kiterjesztés", repos="NULL",
type=...)
```

`type="source"`: ha Linux-os, tar.gz kiterjesztésű állomány áll rendelkezésünkre,
`type="win.binary"`: ha Windows-os zip állományt töltöttünk le.

Internetkapcsolat nélküli telepítéskor problémát jelenthet, hogy számos csomagnak van további csomagfüggelme, amely csomagok beszerzéséről vagy frissítéséről szintén gondoskodnunk kell. Pl. a `fields` csomag nem működik a `maps` csomag nélkül. Ha van internetkapcsolatunk, akkor általában a függőségben lévő csomagok is telepítésre vagy frissítésre kerülnek, manuális beavatkozás nélkül.

A telepített csomagok a következő paranccsal nyithatók meg és csukhatók be:
`library(csomagnév)` vagy `require(csomagnév)`
`detach_package(csomagnév)`

Az R-ben alapértelmezetten megtalálható (pl. `stats` csomag), illetve a telepített csomagok az `installed.packages()` utasítással listázhatók ki. Az elérhető csomagok a `library()` utasítással is megtekinthetők.

Megjegyzés: A bevezetés során az általunk használandó csomagok és dokumentációjuk:
NetCDF állományok kezeléséhez:
ncdf4: https://cran.r-project.org/web/packages/ncdf4/index.html

Térképes ábrázoláshoz:
maps: https://cran.r-project.org/web/packages/maps/index.html
maptools: https://cran.r-project.org/web/packages/maptools/index.html (szükséges csomag: sp)

Az image.plot függvényt tartalmazó csomag:
fields: https://cran.r-project.org/web/packages/fields/index.html (szükséges: spam, maps)

Színskála készítéséhez:
RColorBrewer: https://cran.r-project.org/web/packages/RColorBrewer/index.html

6. Felhasznált irodalom

Solymosi Norbert: Bevezetés az R-nyelv és környezet használatába, jegyzet
https://cran.r-project.org/doc/contrib/Solymosi-Rjegyzet.pdf

Abari Kálmán: Bevezetés az R-be, jegyzet
http://psycho.unideb.hu/munkatarsak/abari_kalman/szamitastechnika_II/bevezetes_az_R_be_2008_04.pdf

R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.
A https://www.rdocumentation.org/ oldalon elérhető leírások a függvények paramétereiről.