File handling

Tamas Kadlecsik

February 21, 2016

read.table(path/to/file, sep="", header=F, quote = "\"'", comment.char="#")

`path/to/file` – Elérési utak

DESCLAIMER: A jegyzetben a függvényparaméter, és függvény argumentum szavakat látszólag szinonimaként használom, azonban ez nem teljesen igaz. A következő órán kiderül, hogy ez miért nem teljsen igaz. Egyelőre viszont nyugodtan gondoljuk azt, hogy paraméter = argumentum.

R-ben dolgozva mindig van egy munkamappánk. Ezt a getwd() függvény meghívásával tudjuk lekérni, beállítani pedig a setwd("path/to/dir") függvénnyel.

Ha például én alapesetben meghívom a getwd()-t a konzolban nekem a

Unix alapú rendeszereken (Linux, OS X):

```
getwd()
```

```
## [1] "/media/nazgul/ext4/Dropbox/Prog/Education/AlapozoBioinformatika/3_file_handling"
```

```
setwd("/home/nazgul/")
getwd()
```

```
## [1] "/home/nazgul"
```

Windowson:

```
setwd("D:/Education/AlapozoBioinformatika")
```

Ez azért fontos, mert fájlokat elérhetünk relatív, és abszolút utakkal. Fájl beolvasása a read.table() függvénnyel történik. Amennyiben a csak a fájl nevét adjuk meg, például

```
read.table("toy.csv")
```

akkor a fájlt beállított munkamappában keresi ami jelen esetben a `/home/nazgul/` vagy `D:/Education/bioinfo`

ha a munkamappán belül van még almappánk, például `lecture1` akkor a

```
read.table("lecture1/toy.csv")
```

Paranccsal tudjuk elérni. Annyiban különbözik a relatív úttól az abszolút út, hogy Windows-on a meghajtó betűjelével kezdődik (`C:`, `D:`), míg unixon `/`-rel.

Tehát egy abszolút útvonal példa windowson:

`D:/Education/AlapozoBioinformatika/toy.csv`

Linux:

`/home/nazgul/AlapozoBioinformatika/toy.csv`

Ha nekünk a munkamappától eltérő helyről kell egy fájl azt célszerű abszolút úttal megadni.

sep=""

```
read.table(path/to/file, sep="", header=F)
```

Boncolgassuk kicsit a fájlbeolvasást. A fájl elérést kivesézték, nézzük a maradék két paramétert.

Az R be tud olvasni excel fájlokat is, azonban ez kicsit macerásabb (kell hozzá egy könyvtár, amikről csak később lesz szó). Azonban a leggyakrabban úgyis .csv-ket, .tsv-ket, .delim-eket, használunk. Ezek szimplán annyit jelentenek, hogy az egyes cellák adott karakterekkel vannak elválasztva. Az elválasztó karakterek leggyakrabban a vessző, pontosvessző, tab. A vesszőt használják angol nyelvterületen, mivel ott nem okoz gondot a tizedes pont. Ezzel szemben, ahol tizedes vesszőt használnak, ott értelemszerűen ez zavarra adhat okot, hiszen nem egyértelmű hogy pl a 17,26 most tizenhét egész huszonhat, vagy 17, és a mellette lévő cella 26.

A tabbal bezzeg sehol sincs gond, és még olvashatóbb is a kimenet. Azzal egyedül az a bibi, hogy az excel közvetlenül `.csv` kiterjesztésű fájlokat hajlandó megnyitni. Ezeket pedig `.tsv`-nek illik elnevezni, vagy .delim-nek. Igen gyakran dolgozunk gigás állományokkal, amiket szenvedős megnyitni, ezért a megfelelő kiterjesztés sokat tud segíteni. (Természetesen a legritkább esetben kapjuk megfelelő kiterjesztéssel a fájlt, aminek következtében a genealógiai megjegyzések, azon belül is egy konkrét női felmenőre vonatkozók igen gyakoriak az informatikai laborokban.)

A `sep=","`, `sep=";"`, és a `sep="\t"` paraméter éppen ezért a cellákat elválasztó karakterre vonatkozik. Ez lehetne akár `sep="a"` is, ez esetben a fájlunkat minden a betűnél fogja oszlopokra darabolni az R, aminek kevés értelme van. A `\t` a tab jele, vagyis, ha szemünkkel tabot látunk, ott a programnyelvek `\t`-t, csupán úgy írták meg az adott szövegszerkesztőt, hogy nagy lukként jelenítse meg.

Ritkább esetben megesik, hogy a cellákat egy vagy több space-el választják el egymástól ez esetben:

```
spe=" " # 1 space idézőjelek között
spe="  " # 2 space idézőjelek között
spe="   " # 3 space idézőjelek között
spe="    " # 4 space idézőjelek között
```

Bizonyos kódszerekesztők tab leütésnél 2 vagy 4 szóközt illesztenek be, így előfordulhat hogy ami tabbal elválasztottnak tűnik az valójában szóközszeparált. A tab vs 2 space vs 4 space évtizedekre visszanyúló vita a programozók között, nem is igazán van objektív válasz rá, így ez a helyzet valószínűleg az idők végéezetéig velünk marad.

Ha a `sep=""`, vagyis semmi, akkor az egész fájl egyetlen cellaként kerül beolvasásra, aminek szintén ritkán van értelme.

Azért `sep=""`-ként hivatkozunk a paraméterre, mivel ha nem adunk meg semmit, akkor ez az alapértelmezett értéke, értsd:

```
read.table(path/to/file, sep="")
==
read.table(path/to/file)
```

Vagyis a fenti két függvényhívás ugyanazt adja eredményül.

header=F

```
read.table(path/to/file, sep="", header=F)
```

A táblázatainknak általában az első sora a fejléc, ahol nem az értékek vannak, hanem azok nevei. Az órai példában ez a „name, age, eye color” sor volt, ha a header paramétert nem állítjuk be, akkor alapértelmezetten FALSE, vagyis nem veszi az első sor-t fejlécnek. Ezért fájlbeolvasáskor ezt TRUE-ra kell állítanunk. Hogy miért nem TRUE az alapértelmezett érték, azt ne tőlem kérdezzétek...

quote="\"'"

Gyakran a szöveges cellákat "" vagy ''-ek közé teszik. Főleg akkor fontos ez, ha szóköze a cellaszeparátor, ugyanis ez esetben nem egyértelmű, hogy pl

```
Name City
Annie New York
```

esetén

```
Name | City
Annie | New York
```

vagy

```
Name City Annie New | York
```

a helyes beolvasás. Azonban ha a fájlban

```
"Name" "City" "Annie" "New York"
```

Szerepel ez a probléma már semmiképp sem merülhet föl

Az escape \ operátorral már találkoztunk a tab \t karakternél. Az escape-elés azt jelenti, hogy a stringben leírt karakter valami mást jelent, mint escape-elés nélkül. Vagyis:

```
Annie\tNew York
```

esetén a leírt t betűt ne egysezű t-nek értelmezze az interpreter, hanem tab-nak.

A "\"" esetén szólunk az interpreternek, hogy a második " nem lezárja a stringet, hanem egyszerű " charakterként érelmezendő. Hiszen a "" önmagában üres stringet jelöl. Ugyanez a helyzet a '\''-el is.

Tehát a quote="\"'" argumentum azt adja meg a függvénynek, hogy a beolvasás során a "" és '' közé zárt karakterek összefüggő szöveges cellának értelmezendők.

Amennyiben nem használunk idézőjeleket jelölést, olykor beolvasás közben megzavarodhat az interpreter és bizonyos sorokat egyetlen cellának olvas be. Ilyenkor érdemes a quote="" vagyis “Üres string a szövegjelelő” == “Nince szövegjelölő” argumentummal meghívni a függvényt.

Itt is igaz, hogy ha kihadjuk az argumentumot akkor az alapértelmezett quote="\"'" argumntummal hívódik meg.

comment.char

Ez a parmaéter adja meg, hogy milyen karaktert értelmezzen az interpreter komment jelölőnek. Ha a megadott karakterrel találkozik, akkor abbahagyja az adott sor beolvasását és ugrik a következőre.

Alapértelmezésben az R-ben használatos #-ot veszi komment karakternek, így ha a beolvasandó táblázatban valahol szerelpel hashmark és nem kommentet jelöl akkor érdemes comment.char=""-el hívni a függvényt.

Most akkor mi van?

Ha beütjük a consolba, hogy

```
?read.table
```

megkapjuk a függvény teljes paraméter listáját. Látszik, hogy jóval több van neki, mint amiket most itt tárgyaltunk. Az esetek döntő többségében elegendő a

```
my.data <- read.table("D:/Education/AlapozoBioinformatika/toy.csv", header = T, sep=',')
```

(Windows)

```
my.data <- read.table("/home/nazgul/AlapozoBioinformatika/toy.csv", header = T, sep=',')
```

(Unix)

kinézetű hívás.

Ha valamiért meggajdulna és elkezdene több sort egyetlen cellába beolvasni leggyakrabban nem megfelően szeparáltak a szöveges cellák, ezt a `quote` argumentum beállításával tudjuk orvoslni, vagyis:

```
my.data <- read.table("D:/Education/AlapozoBioinformatika/toy.csv", header=T, sep=',', quote="")
```

Ha pedig szerepel valahol a táblázatban `#`, ami például ID-k esetén igen gyakori, akkor a `comment.char` paramétert kell üres stringre állítani:

```
my.data <- read.table("D:/Education/AlapozoBioinformatika/toy.csv", header=T, sep=',', comment.char="")
```

Ha mindkettő igaz:

```
my.data <- read.table("D:/Education/AlapozoBioinformatika/toy.csv", header=T, sep=',', comment.char="",
```

Az elnevezett argumentumok (`header=`, `sep=`, `quote=`, `comment.char=`) sorrendje természetesen nem számít, azonban a `path/to/file`-nak mindig a legelső argumentumnak kell lennie. Az elnevezett argumentumokból bármelyiket ki is hagyhatjuk.

Írjunk scriptet!

Fájlokat fel tudunk dolgozni a konzolban is, de ezzel nem vagyunk sokkal beljebb, mintha excelben dolgoznánk, hiszen minden utasítást, minden fájlnál újra ki kell adni. Sőt ez esetben az excel még előnyösebb is talán.

Az R-rel – vagy bármilyen más programnyelvvel – történő adatfeldolgozás előnye, hogy eltehetjük a feldolgozásai műveleteinket későbbre egy ún. scriptfájlba. Ezt később bármikor meghívhatjuk, és ugyanaz a művelet fog lefutni minden alkalommal.

A scriptfájl valójában egy egyszerű szöveges fájl, ahová a konzolba kiadandó parancsokat írjuk egymás után. Rstudio-ban ezt a szövegszerkesztő „Source” gombjára kattintva tudjuk lefuttatni. Ezek után a parancsértelmező szépen sorban az első sortól az utolsóig lefuttatja az utasításainkat.

Az órán írt két scriptet angolul kommentelve megtaláljátok a honlapon. Azért angolul, mert ha ékezetes betű van a kódban, az R-ben be kell állítani a kódolást, és az plusz macera. Amúgy sem árt, ha megszokjuk, hogy programkódba csak angolul írunk, más nyelveken sokkal több szívás van az ékezetekkel.

print(x)

Eddig ha egy utasítás eredményét nem rendeltük változóhoz az automatikusan kiíródott a konzolra. Az egyetlen különbség a scriptfájl futtatása, és a konzolba pötyögés között, hogy scriptfájlban ez nem működik. Explicit ki kell jelentenünk, hogy ezt mi a konzolon szeretnénk látni. Ezt a print() függvénnyel tudjuk megtenni.

A print függvény argumentuma bármilyen változó, és bármilyen érvényes R utasítás lehet.

Leggyakrabban akkor használjuk, ha a program futását szeretnénk ellenőrizni. Debuggolás során (vagyis a scriptben történő hibakeresés kor) igen jó barátunk lesz ez a függvény.

FIGYELEM!!!

A programjaink futásának eredményét kitolhatjuk konzolra, de semmiképp ne kapcsolódjon össze a fejünkben a print() és egy függvény / program futásának eredménye. Programjaink futásának eredményét / eredményeit fájlba illik írni, erre szolgál a

`write.table(x, path/to/file, sep="", row.names=T, quote=T)`

Itt most 5 paraméter beállítására lesz szükségünk. `x` egy bármilyen valid R objektum lehet, vagyis vector, matrix, list, data.frame. A `path/to/file` és a `sep` természetesen ugyanúgy működik, mint a `read.table()` esetén. Azonban figyeljünk oda arra, hogy amennyiben már letéző fájl nevét adjuk meg, akkor kegyetlenül felülírja a régit!

A `sep`-et pedig megadhatjuk mi. Annak érdekében, hogy ha valaki másnak kell dolgoznia a fájlunkkal, adjunk neki beszédes nevet és tartsuk meg a konvenciókat. Ha lehet akkor vessző legyen a szeparátor, és `.csv` a kiterjesztés mivel úgyis tizedes pontot használ minden programnyelv.

Ha azt szeretnénk, hogy az excel is pontot használjon a vessző helyett, akkor windows-ban a tizedes jelet a "Vezérlőpult"->"Területi és Nyelvi Beállítások" fülön a "További beállítások" "Szám" fülén tudjuk átállítani.

A `row.names`-t érdemes F-re állítani, különben mindig beírja a sorneveket első oszlopként, ami kicsit zavaró. Ez igaz lesz akkor is, ha vektort írunk ki. Ha a `quote` paramétert nem állítjuk be, akkor a character típusú értékeinket idézőjelbe teszi. Ez adott esetben hasznos lehet, ha később excelben meg akarjuk nézni a táblázatunkat, és az adott számot mi csak egy jelölőnek, indexnek, faktornak, sorszámnak akarjuk használni, akkor kevésbé tévedünk el. Gyakorlatilag magunkra kényszerítjük, hogy megfelelőképpen használjuk az adott értéket. Egyéb esetben azonban zavaró, így én ki szoktam kapcsolni, de ez teljes mértékben egyéni preferencia kérdése.