Statisztikai Szemle

Közzététel: 2020. július 9.

A tanulmány címe:

Adatforrások használata R-ben

Szerzők:

ABALIGETI GALLUSZ, a Pécsi Tudományegyetem PhD-hallgatója
E-mail: gabaligeti@gmail.com

GYIMESI ANDRÁS, a Pécsi Tudományegyetem PhD-hallgatója
E-mail: gyimesi.andras@ktk.pte.hu

KEHL DÁNIEL, a Pécsi Tudományegyetem adjunktusa
E-mail: kehld@ktk.pte.hu

DOI: https://doi.org/10.20311/stat2020.7.hu0858

Abaligeti Gallusz – Gyimesi András – Kehl Dániel

Adatforrások használata R-ben*

Data collection through R

ABALIGETI GALLUSZ, a Pécsi Tudományegyetem
PhD-hallgatója
E-mail: gabaligeti@gmail.com

GYIMESI ANDRÁS, a Pécsi Tudományegyetem
PhD-hallgatója
E-mail: gyimesi.andras@ktk.pte.hu

KEHL DÁNIEL, a Pécsi Tudományegyetem
adjunktusa
E-mail: kehld@ktk.pte.hu

A tanulmány célja, hogy olyan módszereket mutasson be, melyek segítségével statisztikai elemzésre kész adatállományok gyűjthetők. A szerzők a bemutatott adatforrásokat 3 kategóriára osztják: adatbázisokra, REST API végpontokra és weblapokra. Az adatgyűjtéshez alkalmazott eszköz az ingyenes, open source R nyelv és annak kiegészítő csomagjai, melyek segítségével mindhárom adatforrástípusból kinyerhetők adatok. Jelen írás adatelemzésre nem vállalkozik, az egyes adatállományok esetén inkább a potenciális lehetőségekre utal. A módszerek lehetséges használatára mutat be példákat a makroökonómia, a közösségi média, valamint a sport területéről. Ezeken a területeken tipikusan nagy mennyiségű publikusan elérhető adat található a világhálón, ám az ismertetett eljárások más területeken folytatott tudományos kutatásokban is alkalmazhatók. Az egyre népszerűbb akadémiai szemlélet, a megismételhető kutatás (reproducible research) egy példáját nyújtják a szerzők azzal, hogy elérhetővé teszik a cikkhez tartozó forráskódot. Amennyiben az alkalmazott csomagok lényeges elemei változnak, a tanulmány frissített verziója elérhető lesz (a cikk és mellékletei a következő linken találhatók: http://search.ksh.hu/#/year/2020?c=s#07).

TÁRGYSZÓ: megismételhető kutatás, adatgyűjtés, R

This study provides an overview of three methods which can be used to retrieve data in R, a free open source programming language, and its associated packages, specifically developed for such purposes. The methods are introduced in relation to databases, RESTful API endpoints, and web pages. The authors showcase examples from various fields, such as macroeconomics, social media, and sports, where vast amounts of data are available online; however, data analysis is not in

* A kutatás az Emberi Erőforrás Fejlesztési Operatív Program, EFOP-3.6.2-16-2017-00003: „Sport-, Rekreációs- és Egészséggazdasági Kooperációs Kutatóhálózat létrehozása” című projektjének támogatásával készült.

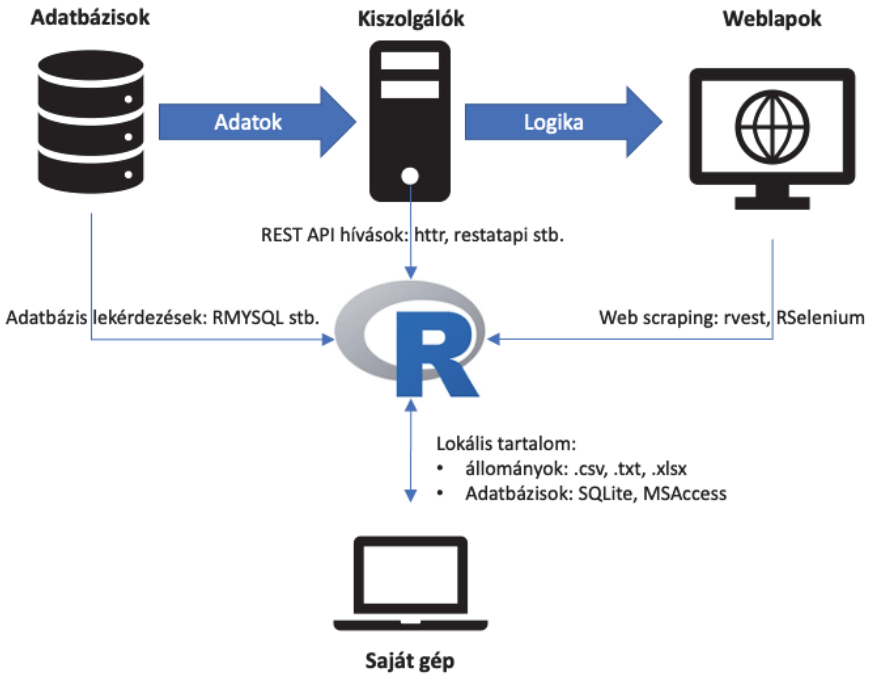
the focus of the study. They offer an example of reproducible research by making the relevant scripts available. Would there be any substantial changes to the syntax used by the corresponding packages, the authors will keep the scripts up to date.

Keyword: reproducible research, data collection, R

Az R környezet, egyre növekvő népszerűsége miatt, ma már szinte mindenki által ismert, a *Statisztikai Szemlé*ben is több tanulmányt olvashattunk, melyek kifejezetten az R használatára épülnek (*Daróczi* [2016], *Daróczi–Tóth* [2013], *Hajdu* [2018]). Jelen írásunkban egy logikai csoportosítás alapján gyűjtünk össze és mutatunk be olyan megoldásokat, R csomagokat, melyek valami módon az adatgyűjtés témaköréhez kapcsolódnak. A csoportosítás alapját az összegyűjtendő adatok helye és feldolgozottsága képezi. Nem kívánjuk az összes megemlített csomagot részletesen bemutatni, csupán néhány fontosabb példát ismertetünk, de a web scraping (weben elérhető adathalmaz másolása a helyi számítógépre) témakörével részletesebben is foglalkozunk. A tanulmány jellegéből adódóan tehát sok R csomagot megemlítünk, de csak azokat hivatkozzuk formálisan, amelyeket nagyobb mértékben felhasználunk a kódban. Az R környezetben az adattáblákat általában dataframe objektumokban tároljuk, a kinyert adatokat többnyire ilyen formátumban kapjuk meg. A példák során néhány esetben a bemutathatóság érdekében csupán a kinyert adatok töredékét vonultatjuk fel, az ilyen adatmanipulációra a **`dplyr`** csomag (*Wickham et al.* [2020]) függvényeit fogjuk használni. Érdekességképp említjük meg, hogy jelen tanulmány nem a szokásos szövegszerkesztőkben, hanem R-ben íródott, az ún. RMarkdown (Rmd) formátumban, amiből Word, HTML vagy akár pdf is generálható. A formátum előnye ezen kívül, hogy az R kódok végeredménye (ábra, táblázat) is könnyedén és dinamikusan ágyazható a dokumentumba. Ez a módszer alkalmas az elemzések és a tanulmány megszövegezésének összekapcsolására, ami fontos a kutatás megismételhetősége szempontjából (*Xie* [2015]). *Gandrud* [2016] részletesen bemutatja az eljárás alkalmazását és egyéb eszközöket, amellyel egy kutatás reprodukálhatóvá tehető bárki számára.

Az 1. ábra a tanulmányban követett csoportosítási logikát foglalja össze.

1. ábra. Adatforrások típusai
(Types of data sources)



Az 1. ábrán azokat a legfontosabb adatforrásokat szerepeltettük, melyeket R-rel viszonylag egyszerű kiaknázni, azaz léteznek erre előre megírt csomagok (ezekről lesz szó a további fejezetekben), melyekkel csatlakozhatunk az adatforrásokhoz és valamilyen formátumban, bizonyos paraméterekkel szabályozva, adatokat tudunk letölteni az R munkaterületére. Az ábra felső sorában a – valamilyen – hálózatról (intra-/internet) elérhető forrásokat gyűjtöttük össze: adatbázisok, REST API szolgáltatások[1] és weblapok. Fontos kiemelni, hogy balról jobbra nő az adatok feldolgozottsága, és ezzel párhuzamosan csökken az olvasás sebessége. Az ábra alsó részén a helyi fájlok elérését szimbolizáltuk.

Hogy egy kicsit más szemszögből is tudjunk mindezekre tekinteni, egy táblázatos összefoglalót is készítettünk az adatforrások típusairól. Arra helyeztük a hangsúlyt, hogy az adatforrások egyes típusai miben különböznek, ha lokálisan vagy ha hálózaton keresztül kívánjuk őket elérni.

[1] Ezeket a legegyszerűbb formázás nélküli weblapként elképzelni, melyek csupán a nyers adattartalmat szolgáltatják.

Adatforrások helyük és típusuk szerint
(Data sources by location and type)

Típus	Lokális adatok	Hálózaton elérhető adatok
Fájl	.csv, .txt, .xlsx	URL
Adatbázis	SQLite, MSAccess	MySQL, SQL Server, PostgreSQL
API	nem jellemző	REST
HTML	nem jellemző	www

A tanulmány további fejezeteiben a táblázat egyes celláit fogjuk részletesen bemutatni, valamint példákat és javaslatokat adunk az alkalmazható csomagokra, illetve függvényekre.

1. Lokális adatok

Sok esetben a szükséges adatok a helyi számítógépen találhatók, különböző táblázatos (csv, txt), vagy valamilyen egyéb (statisztikai) szoftver (EViews, Excel, SPSS, Stata, SAS stb.) által készített formátumban. Az előbbi fájlok beolvasása egyszerűbb esetekben a **`read.table`** függvénnyel, illetve annak speciális változataival történhet. Az olyan csv fájlok esetén, ahol vesszővel jelölik a tizedesjegyet (ilyen például Magyarország is), célszerű a **`read.csv2`** parancs alkalmazása, amely alapbeállításként a vesszőt használja tizedes jelölőként és a cellák elhatárolására a pontosvesszőt (a beolvasó fájlok esetén gyakori a 2 végződés alkalmazása az olyan országok felhasználói számára, ahol nem az angolszász jelöléseket követik). A beolvasandó fájlok méretének növekedésével egyre inkább ajánlott a **`readr`** (*Wickham et al.* [2018]) vagy a **`data.table`** (*Dowle–Srinivasan* [2019]) csomagban található alternatív beolvasó algoritmusokat választani, melyek akár 10-15-ször gyorsabbak lehetnek. Az Excel formátum továbbra is meghatározó, annak ellenére, hogy nem statisztikai szoftverről van szó. Az ilyen fájlok beolvasására több R csomag is alkalmas, mi a **`readxl`** csomagot (*Wickham–Bryan* [2019]) emelnénk ki ezek közül.

Más statisztikai szoftverekből általában lehetséges az említett hordozható adatokat exportálni, de természetesen lehetőség van azokat közvetlenül is beolvasni, a **`haven`** csomag (*Wickham–Miller* [2020]) SPSS, Stata és SAS fájlok olvasására és írására is alkalmas.

2. Adatbázis-kapcsolatok

Felidézve az 1. ábrát, kijelenthetjük, hogy ideális esetben minden adat valamilyen közvetlen adatbázis-kapcsolaton keresztül lenne elérhető. Ez a valóságban természetesen nem így van, elsősorban biztonsági megfontolások miatt. Egy esetben jellemző ennek az adatforrásnak a használata: a szervezeten belüli itraneten keresztül történő csatlakozás során. A másik oka az adatbázis-elérés korlátozásának, hogy a webalkalmazások nagy részét böngészhető weboldalak kiszolgálására tervezik, ezért a mögöttük álló adatbázisokat nem célszerű mással (jelen esetben ad hoc lekérdezésekkel) terhelni.

Az adatbázisok két nagy csoportra oszthatók: relációsra és nemrelációsra. A két csoport között az a fő különbség, hogy míg a relációs sémák esetében a belső konzisztenciát biztosító szabályok (relációk) sok számítási kapacitást igényelnek, addig a lazább szabályokkal rendelkező nemrelációs adatbázisok – éppen ezért – nagyobb írási/olvasási sebességet tesznek lehetővé. Cikkünkben az első csoportba tartozó megoldásokról ejtünk szót, hiszen ezek a technológiák már kiforrottabbak, stabilabbak. Ennek ellenére a jövőben számíthatunk a nemrelációs adatbázisok, valamint a velük kapcsolatot teremtő R csomagok elterjedésére is.

A relációs adatbázisok precíz matematikai definíciója helyett két fontos tulajdonságát emelnénk ki:

> – egyrészt olyan táblák összességéről van szó, amelyeknek oszlopai szigorúan típusosak (dátum, egész szám, szöveg stb.);
>
> – másrészt a táblák oszlopai között lehetőség van kapcsolatok definiálására, amivel az oszlopok értékkészletét korlátozhatjuk más táblák adattartalmára.

A relációs adatbázisokból az SQL nyelv segítségével tudunk adatokat lekérdezni. A teljesség igénye nélkül a legelterjedtebb ingyenes, relációs adatbázis szoftverek a MySQL, PostgreSQL vagy az SQLite (ez utóbbi főleg lokális felhasználásra), kereskedelmi forgalomban pedig a MS SQL Server, illetve az Oracle megoldásai a legnépszerűbbek.

Ahhoz, hogy R-en (vagy bármilyen más kliensen) keresztül kényelmesen tudjunk egy adatbázist elérni és annak lekérdezéseket küldeni, szükség van a kapcsolatot felépítő eszközre: általános célú adatbázis-kapcsolódásra alkalmas csomag a **`DBI`** (*R Special Interest Group on Databases (R-SIG-DB)–Wickham–Müller* [2019]). Funkciója egy tolmácséra hasonlít, aki fordít az R és az adatbázis között, ugyanakkor alapállapotában nem ismer egyetlen nyelvet sem. Ahhoz, hogy megértse az adatbázist, fel kell paraméterezni a megfelelő szótárral, ezt hívjuk adatbázis-meghajtónak.

2.1. Adatbázis-meghajtók

A legnépszerűbb relációs adatbázisokhoz csomag formájában készülnek meghajtók: **RMySQL** (*Ooms et al*. [2020]), **RSQlite** (*Müller et al.* [2020]), **RPostgres** (*Wickham et al*. [2019]b). Azon adatbázisokhoz, amelyekhez dedikált csomag nem készül, az **RODBC** (*Ripley–Lapsley* [2019]) csomag használható, ami az operációs rendszer általános adatbázis-meghajtóját, az ODBC-t alkalmazza. Fontos tudni, hogy ez utóbbi esetén is kell egy ODBC kompatibilis meghajtó, azonban ezt az operációs rendszer szintjén szükséges telepíteni. Ennek illusztrálására vegyünk példának egy fiktív adatszolgáltatót, melynek MySQL adatbázisa a www.data.com címmel rendelkező szerveren található. Ha ennek a szolgáltatónak a MACRO nevű, makroadatokat tartalmazó adatbázisához kívánnánk csatlakozni R segítségével, azt a következő kódrészlettel tehetjük meg:

```
con <- DBI::dbConnect(RMySQL::MySQL(),
                      host = "www.data.com",
                      user = "scientist",
                      password = "statszemle"
                      dbname = "MACRO")
```

Ezzel felépült a kapcsolat a távoli adatbázis felé, amit a **con** változóban tárolunk el. Tegyük fel, hogy az adatbázis GDP (gross domestic product – bruttó hazai termék) nevű táblájából Magyarország tavalyi GDP-jét szeretnénk lekérdezni különböző devizanemekben, akkor azt a **dbSendQuery** függvény segítségével a következő SQL lekérdezés elküldésével tehetjük meg:

```
res <- dbSendQuery(con, "SELECT currency, value FROM GDP WHERE country =
'hun' and year = 2019")
dbFetch(res)
```

Végül a **dbFetch** függvény feladata lesz, hogy dataframe-et készítsen a lekérdezés eredményéből (és ezt írja ki a konzolra).

Külön érdemes kiemelni a MS SQL Serverhez történő csatlakozást, ez az ODBC kapcsolaton kívül a **RevoScaleR** csomaggal is elvégezhető anélkül, hogy a **DBI**-t használnánk. Ennél azonban jóval távolabbra mutató eszközről van szó, ugyanis a 2017-es verziótól indulva az SQL Server képes az adatbázisban R kódot futtatni és – más módszerek mellett – ezzel a csomaggal „kérhetjük meg” az adatbázist az R kód futtatására. Jelen tanulmány terjedelmét jóval meghaladná ennek a technológiának a további részletezése, de könnyen belátható, hogy az adatok mennyiségének növekedése miatt sokszor egyszerűbb/gyorsabb a feldolgozó motorokat az adatokhoz vinni, mint fordítva.

2.2. Absztrakt SQL

Az SQL nem egy szigorúan követett szabvány, inkább az egyes gyártók által használt nyelvjárásokról beszélhetünk. Emiatt egy komplex lekérdezés „átfordítása" két különböző dialektikájú adatbázis-kezelő (például Oracle PL/SQL és Microsoft T-SQL) között sok nehézséget okozhat. Az efféle problémák kiküszöbölésére készülnek az SQL absztrakcióját célzó könyvtárak a népszerű programnyelvekhez. R-ben ez a csomag a **dbplyr** (*Wickham–Ruiz* [2020]), ami a Tidy univerzumból (*Wickham et al*. [2019a]) megismert adatmanipulációs szintaktikát fordítja át olyan SQL dialektusra, amit az adott kapcsolat előír neki. Amennyiben tehát ismerjük a dataframe-ek transzformációs technikáit a **dplyr** csomagból, úgy ezeket minden további nélkül alkalmazhatjuk adatbázis-lekérdezések elkészítéséhez. Korábbi példánkhoz visszatérve (és abból a **con** változót felhasználva), a következő módon kérdezhetjük le a GDP-adatokat:

```
tbl_gdp <- tbl(con, "GDP")
tbl_gdp %>%
  select(currency, value) %>%
  filter(country == "hun" & year == 2019) %>%
  collect()
```

Felhívjuk a figyelmet a pipe (**%>%**) sorozat végén a **collect()** parancsra. A **dbplyr** ún. lusta módszerrel értékeli ki a számára megadott lekérdezést, tehát amíg nincs szükség konkrétan a lekérdezés eredményére, addig nem küldi el a lekérdezést az adatbázis felé. A **collect()** függvény meghívásával tudjuk utasítani az R-t, hogy ténylegesen töltse be a workspace-be a lekérdezés tartalmát.

Az adatbázisokkal foglalkozó rész zárásaként mindezek fényében azt javasoljuk, hogy R felhasználóként a **DBI** + adatbázis-meghajtó + **dbplyr** hármas segítségével nyerjünk ki adatokat relációs adatbázisokból.

3. REST interfészek elérése

A REST (representational state transfer) egy kliensszerver-alapú hálózati architektúra, melyben a kliensek kéréseket küldenek a szerver felé, amely ezekre bizonyos szabályokat követve válaszol. Ennek az architektúrának egy megvalósítása a HTTP (hyper text transfer protokoll), amit már ismerhetünk, hiszen az internet világa is ezt használja. Amennyiben egy szerver ismeri ezt a protokollt, akkor be-

szélhetünk arról, hogy rendelkezik REST interfésszel. A továbbiakban az ilyen típusú kiszolgálókról lesz szó.

A HTTP ún. „metódusok" segítségével valósítja meg a REST architektúrát. Ezek a metódusok alkalmasak arra, hogy a távoli szerverekről adatot kérdezzünk le, töröljünk, módosítsunk vagy hozzunk létre.[2] Cikkünk szempontjából az adatok (vagy más néven erőforrások) lekérdezése a legfontosabb, ezt a GET metódus segítségével tehetjük meg. Térjünk vissza a képzeletbeli adatszolgáltatóhoz, amelynek szervere (www.data.com) rendelkezik REST interfésszel. A korábbiakhoz hasonlóan Magyarország tavalyi makrogazdasági adatai közül a GDP-t szeretnénk lekérdezni, ezt a következő jellegű kéréssel[3] tehetjük meg:

GET http://www.data.com/macro/gdp?country=HUN&year=2019,

ahol

– a macro/gdp az erőforrás (adat) típusára utal,
– a ?country=HUN&year=2019 pedig a keresési feltételekre.

Természetesen elképzelhető, hogy a keresés más mezőnevekkel történik, de egy tipikus REST interfész ilyen formában fogad be kéréseket. A válasz attól függ, hogyan specifikálták a szerveren futó programot, de jellemzően így jelenik meg:

{ "HUF": 47462712, "EUR": 143826.4, "USD": 163664.5 },

vagyis különböző devizákban visszakapjuk a kért adatpontot, ún. JSON szövegformátumban (*Ooms* [2014]), azaz kapcsos zárójelek közötti kulcsérték-párokban. Az eredmények értelmezése a lekérdező feladata, ehhez a segítséget az interfész specifikációjában találja meg. Innen derül ki az is, hogy a példában kiírt értékek millióban értendők.

A valóságban ezek a kérések és a szervertől visszakapott válaszok rendkívül bonyolultak lehetnek és ebben is segítséget nyújt – sok más programnyelv mellett – az R, pontosabban bizonyos csomagjai. Ezek tehát egyrészt segítenek megszerkeszteni a kérést, általában könnyebben paraméterezhető függvényeken keresztül, másrészt átalakítják a választ olyan formátumba (tipikusan listába vagy dataframe-be) amit rögtön felhasználhatunk további elemzésre.

Általában a REST interfészek elérésére a **`httr`** csomagot (*Wickham* [2019]) ajánljuk. Számunkra a legfontosabb függvény a **`GET`**, ahogy a neve alapján sejtjük,

[2] Ezek a CRUD: create, read, update, delete műveletek.

[3] Természetesen egy „valódi" HTTP kérés ennél sokkal bonyolultabb, a GET ott a kérés típusát definiálja sok más paraméter mellett.

ennek a segítségével tudunk GET típusú kéréseket létrehozni. A korábbi példánkban szereplő kérést a következő paranccsal tudjuk elküldeni:

```
res <- GET("http://www.data.com/",
           path = "macro/gdp",
           query = list(country = "hun",
                        year = 2019))
```

Ezen felül rengeteg paramétere van a **GET** függvénynek, a teljesség igénye nélkül emeljük ki, hogy egyszerű HTTP autentikációra (azaz azonosításra) is alkalmas, illetve cookie kezelését is be lehet állítani vele.

A **httr** csomag tehát egy általánosan használható eszköz REST interfészek elérése. Vannak azonban annyira széles körben elfogadott interfészek (statisztikai szolgálatok, közösségimédia-oldalak vagy akár a Google egyes eszközei), hogy az R közösség konkrétan egy-egy ilyen interfészhez is fejlesztett dedikált csomagot. A továbbiakban ezekből mutatunk be néhányat.

3.1. Hivatalos statisztikai szolgálatok

A legtöbb nagy nemzetközi szervezet gyűjt, tárol és elérhetővé tesz adatokat. Hagyományosan ez csv, vagy Excel formátumban letölthető adatokat jelent a kutatók számára, amit később a használni kívánt statisztikai szoftverbe importálnak. Aki már gyűjtött több adatforrásból információt, annak nem ismeretlenek a potenciálisan felmerülő problémák (konverziós nehézségek, tizedes jelölő, eltérő országnevek stb.). Amennyiben az elemzést többször meg kell ismételni, esetleg adatrevíziók miatt újabb adatok állnak rendelkezésre, az egész folyamatot újra kell kezdeni. A hasonló problémák leküzdésére fejlesztettek a nagy nemzetközi szervezetek és sok hivatalos statisztikai szolgálat is olyan interfészeket, melyekkel automatizáltan lehet adatokat kinyerni az adatbázisaikból.

A **WDI** csomag (*Arel-Bundock* [2019]) a Világbank adatbázisából tölt le adatokat API segítségével, csupán néhány függvényből áll, a két legfontosabb a **WDIsearch**, amely segít az egyes idősorok kódjának megkeresésében, illetve a **WDI**, amely ténylegesen a letöltést végzi. Amennyiben például GDP-adatokat keresünk, alkalmazhatjuk a **WDIsearch("gdp")** függvényt, ami minden (többszáz) olyan adatsort visszaad, amely nevében szerepel a keresett kifejezés (de kereshetünk a leírásban vagy forrás alapján is). Az argumentum értékének reguláris kifejezéseket (regular expression – regex) is megadhatunk, így például kereshetünk olyan adatsorokat, melyeknek nevében szerepelnek a "gdp", "const" és "$" karakterek, ebben a sorrendben. Mivel a dollárjel egyben egy regex kifejezés is, ezért jeleznünk kell az

R számára, hogy ténylegesen a dollárjelre gondoltunk a két **** karakterrel (a **short = FALSE** argumentum segítségével a függvény a forrást és az adatok bővebb leírását is megadná). A függvény eredménye egy mátrix, amelynek most csak az első öt sorát és második oszlopát jelenítjük meg.

```
WDIsearch("gdp.*const.*\\$")[1:5, 2]
## [1] "GDP per capita, PPP (constant 2011 international $) "
## [2] "GDP (constant 2005 $)"
## [3] "GDP per capita, PPP (constant 1987 international $)"
## [4] "GDP per capita, PPP (constant 2011 international $)"
## [5] "GDP per capita (constant 2010 US$)"
```

A kutatáshoz szükséges adatsorok kódjának kiválasztása után a **WDI** parancs segít ezeket közvetlenül az R-be tölteni. Az alapbeállítás az, hogy minden országra letöltjük az adatokat, de természetesen szűkíthető is ez a kör az országok ISO-2 karakter kódjai segítségével. A következő parancs két indikátort tölt le, 1990-től kezdődően és a világ összes országára, illetve képzett egységekre is (például arab világ, világ összesen, EU stb.). Az eredmény első 10 sorát mutatja be a kódrészlet:

```
WDI(indicator = c("NY.GDP.PCAP.PP.KD", "SL.GDP.PCAP.EM.KD"), start =
1990) %>% head(10)
##    iso2c    country year NY.GDP.PCAP.PP.KD SL.GDP.PCAP.EM.KD
## 1     1A Arab World 1990          9630.924                NA
## 2     1A Arab World 1991          9590.123          39114.24
## 3     1A Arab World 1992          9876.652          40452.26
## 4     1A Arab World 1993          9871.500          40466.42
## 5     1A Arab World 1994          9903.001          40132.65
## 6     1A Arab World 1995          9856.896          39793.47
## 7     1A Arab World 1996         10078.841          40212.85
## 8     1A Arab World 1997         10251.053          40574.58
## 9     1A Arab World 1998         10560.338          41730.72
## 10    1A Arab World 1999         10538.357          41071.09
```

Az Eurostat adatbázisához több csomag segítségével is kapcsolódhatunk, mi most ezek közül a **restatapi** csomagot (*Mészáros* [2020]) említjük meg, ami jelenleg is fejlesztés alatt áll, SDMX és XML alapon. A csomag funkciója nagyon hasonló, mint a **WDI** esetén, kereső és letöltő függvények alkotják. A szűrési lehetőségek széles körűek, dátumra, frekvenciára, területi egységre is könnyedén filterezhetünk. A következő példák néhány adatsor letöltésére vonatkoznak különböző szűrések mellett, az adatok tartalmát a felettük található megjegyzések mutatják.

```
# GDP and main components (output, expenditure and income)
dt1 <- get_eurostat_data("nama_10_gdp")
# Production of cow's milk on farms by NUTS 2 regions
dt2 <- get_eurostat_data("agr_r_milkpr",
                         date_filter = 2008,
                         keep_flags = TRUE)
# Employment by A*10 industry breakdowns
dt3 <- get_eurostat_data("nama_10_a10_e",
                         filters = c("Annual", "EU28", "Belgium",
                                     "AT", "Total", "EMP_DC", "person"),
                         date_filter = c(2002, 2008, 2013:2018))
```

A REST API-k egyre népszerűbbek, a teljesség igénye nélkül néhány további példa nemzetközi és nemzeti statisztikai adatbázisokra:

– **tradestatistics** (Open Trade Statistics),
– **cbsodataR** (Statistics Netherlands),
– **tidyqwi** (Convenient API for Accessing United States Census Bureau's Quarterly Workforce Indicator).

3.2. Közösségimédia-csomagok

Az elmúlt bő egy évtized egyik nagy trendje a közösségi média megjelenése, térnyerése volt világszerte. A felhasználók rengeteg adatot (kapcsolatok, követés, vélemények, képek, videók, hozzászólások stb.) állítottak elő. A legtöbb ilyen közösségi médiacég lehetővé tette, hogy – bizonyos keretek között – ezekhez az adatokhoz hozzáférjenek a felhasználók. Korábban a felhasználói adatok könnyedén gyűjthetők voltak kutatási célokra, akár R csomagokon keresztül is. A hozzáférés módja azonban változott az elmúlt időszakban, az adatok köre jelentősen szűkült a 2010-es évek közepétől kezdve, majd a Facebook és a Cambridge Analytica körüli botrány következtében az elérhető adatok köre drasztikusan csökkent. Elképzelhető, hogy ez a tendencia tovább fog folytatódni a jövőben. Ennek megfelelően a Facebookhoz és az Instagramhoz kapcsolódó korábbi R csomagokat ma már nem fejlesztik tovább, egyrészt azért, mert a kapcsolódó API-k gyorsan változtak, másrészt a funkciók korlátozása miatt gyakorlatilag kutatási célokra (is) alkalmatlanná váltak.

A Twitter hazánkban ugyan kevésbé, de például a tengerentúlon kifejezetten népszerű, az elmúlt években a politikai életbe is berobbant az Egyesül Államok elnökének köszönhetően. Másodpercenként több ezer, azaz naponta több száz millió rövid üzenet (tweet) keletkezik. A beszélgetések szövegeit, illetve a felhasználók

kapcsolódásait lehet elemezni, de az alapvető műveletek is elvégezhetők R-en keresztül: követés, szöveges üzenet posztolása stb. Több csomag is készült az API-hoz, a korábbi **twitteR** csomag helyett jelenleg az **rtweet** (*Kearney* [2019]) az, amely az aktív fejlesztés miatt a legjobb funkcionalitással rendelkezik (a cikk írásának időpontjában a legújabb frissítés néhány napos). A csomag feladata a tweetek gyűjtése az API-n keresztül, az adatokból készíthetők szöveges elemzések, illetve az említések és megosztások alapján (akár dinamikus) kapcsolati hálózatok is, amiben a **twinetverse** csomaggyűjtemény segíthet (*Coene* [2020]). A Twitter esetében is szűkültek az ingyenesen elérhető adatok, például csak egy hétnél frissebb tweetek érhetők el, illetve egyszerre csak korlátozott mennyiségű üzenet tölthető le, a limit azonban 15 percenként újraindul. Az API használatához elegendő felhasználónévvel és jelszóval rendelkezni.

Az egyik alapvető függvény a **search_tweets**, amely egy vagy több keresőszóra ad vissza olyan tweeteket, melyek az adott kifejezést tartalmazzák. A kívánt tweetek száma megadható, amit azonban korlátoz a feltételeknek eleget tevő tweetek száma, illetve a már említett korlátozások is, azonban a **retryonratelimit** argumentum segítségével akár milliónyi tweetet is letölthetünk, azonban ennek a futási ideje akár napokat is igényelhet. A letölteni kívánt üzenetek tovább szűrhetők azok típusa vagy földrajzi származása szerint.

A következő sor az FC Barcelona felhasználót megemlítő tweeteket éri el az elmúlt néhány napból, a lekérdezés eredménye természetesen napról napra változik. A találatok mindegyikéről 90 változó áll rendelkezésre, a továbbiakban csak a két legfontosabb változó első két megfigyelését mutatjuk be.

```
search_tweets("#fcbarcelona filter:mentions", n = 500, include_rts =
FALSE) %>%
  select(screen_name, text) %>%
  head(2)

## # A tibble: 2 x 2
##   screen_name text
##   <chr>       <chr>
## 1 al_bedwawi1985 "<U+00BF>Qué haces, Josep Maria Bartomeu? Te gusta
madrid\n\n@barca_a…
## 2 abidex_abidex20 "@goal @romeoagresti #fcBarcelona board and
confusion they ar…
```

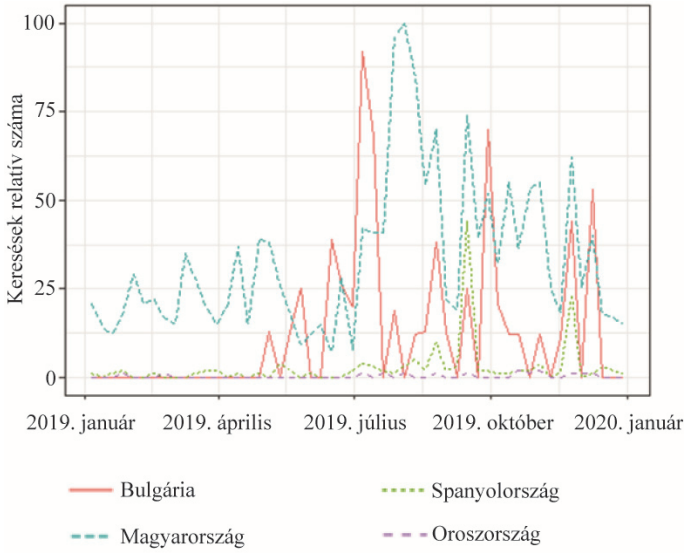
A legfrissebb tweetek letöltéséhez készült a **stream_tweets** függvény, amely előre adott ideig figyeli és tölti le a tweeteket, ebben az esetben szintén van lehetőség szűkítésre kulcsszavas, felhasználói vagy földrajzi hely alapján.

3.3. Google-szolgáltatások

A Google rengeteg szolgáltatást nyújt, melyek közül több is alkalmas arra, hogy adatokat gyűjtsünk. A Google Trends szolgáltatás a Google keresőjéhez kapcsolódik, egyes kereső kifejezések gyakoriságát vizsgálhatjuk vele. A szolgáltatás online is elérhető, illetve az eredmények csv formátumban letölthetők, de a **gtrendsR** csomaggal (*Massicotte–Eddelbuettel* [2020]) R-be is beolvashatók, ábrázolhatók, tovább elemezhetők. A csomag használatához nem szükséges regisztráció. A következő rövid példa a Ferencvárosi TC 2019-es keresési adatait gyűjti le (lásd a 2. ábrát) az Európa Liga H csoportjában érdekelt négy csapat országára vonatkozóan.

```
EuroLiga <- gtrends("Ferencvaros",
                    geo = c("HU", "ES", "BG", "RU"),
                    time = "2019-01-01 2019-12-31")
```

2. ábra. Google-keresések intenzitása négy érintett országban
(Intensity of Google searches in four countries)



A Vision szolgáltatás alapvetően képek elemzésére alkalmas. Használatához rendelkeznünk kell (bizonyos kereteken belül ingyenes) autentikációval, az igénylés a Google Cloud Platformon történhet. A magyar fejlesztők által készített **googleCloudVisionR** csomag (*Koncz et al.* [2020]) gyakorlatilag egyetlen felhasz-

nálóknak szánt függvényt tartalmaz, melynek neve **gcv_get_image_annotations**. A feldolgozandó képek elérési útján kívül a vizsgálati módot kell beállítanunk:

– **LABEL_DETECTION**: felcímkézés, az algoritmus lehetséges kulcsszavakat ajánl a képhez;
– **FACE_DETECTION**: arcfelismerés, az arcok helyével a képen és a legvalószínűbb érzelmi állapot;
– **TEXT_DETECTION/DOCUMENT_TEXT_DETECTION**: szövegfelismerés;
– **LOGO_DETECTION**: cégek, szervezetek logóit ismeri fel az algoritmus;
– **LANDMARK_DETECTION**: nevezetességek, látnivalók felismerését végzi az algoritmus.

A következő példában az URL-en megtalálható ábrára futtattuk a felcímkézést. Az eredmények, illetve az egyes címkékhez tartozó Google által javasolt leírások és a hozzájuk tartozó pontértékek a kód alatt láthatók.

```
gcv_get_image_annotations(
  imagePaths =
"https://as01.epimg.net/futbol/imagenes/2019/07/30/champions/1564499509_
188310_1564499659_noticia_normal.jpg",
  feature = "LABEL_DETECTION",
  maxNumResults = 5) %>% select(description, score)
```

```
##          description     score
## 1:      Championship 0.8959382
## 2:              Team 0.8852716
## 3:           Product 0.8828333
## 4:               Fan 0.8494906
## 5: Competition event 0.8162445
```

4. Web scraping

A web scraping elnevezés a tágabb data scraping fogalomból ered, amely összefoglaló neve mindazon technikáknak, amellyel egy számítógépes program által létrehozott, ám ember által is értelmezhető adathalmazt, egy másik program segítségével nyerünk ki. A legkézenfekvőbb platform, ahol tömegesen érhetők el ilyen külső programok (és más emberek) által létrehozott adathalmazok, az a világháló,

így érdemes a web scraping kategóriáját külön tárgyalni. A web scraping lényegében a weben elérhető bármilyen adathalmaz másolását jelenti a helyi számítógépre. Bár ennek számos formája, technikája és célja lehet, jelen tanulmányban olyan módszereket mutatunk be, amelyek tipikusan kutatási célú statisztikai elemzésre kész adatállományok gyűjtésére alkalmasak.

A weboldalak HTML és XHTML nyelven írt kódokból épülnek fel, mely kódok a tényleges információn kívül rengeteg formázási, a böngésző számára szükséges tördelési előírást tartalmaz, hiszen a végfelhasználói tipikusan emberek. Ennek következménye, hogy a nyers információ kinyerése egyáltalán nem triviális (pláne nem optimális) egy számítógépes program számára. A letöltések egyszerűsítésére és optimalizálására szolgának az előző fejezetekben már tárgyalt API eszközök. Ebben a fejezetben olyan esetekre mutatunk példákat, amikor ilyen API vagy egyéb egyszerűsítő eszközök nem állnak rendelkezésünkre. Az R programcsomag előnye a web scraping feladatok ellátásában, hogy egyaránt hatékonyan elvégezhető vele a nyers adathalmaz előkészítése és későbbi elemzése is. Az **`xml2`** és az **`rvest`** csomag tipikus scraping feladatokra alkalmazható, a csomagok több függvényét is bemutatjuk a későbbiekben. Fontos megjegyezni, hogy az R programcsomag mellett számos más eszköz is alkalmazható, *Mitchell* [2018] a Python programnyelv, *Glez-Peña et al.* [2014] pedig egyéb eszközök használatát mutatja be a web scraping feladatokban.

Az egyszeri adatletöltés fő lépései a következők (*Munzert et al*. [2014]):

1. Az információ azonosítása. Fontos meghatároznunk, hogy pontosan milyen adatokra van szükségünk és ezek hol helyezkednek el. Lényeges kérdés, hogy a kívánt információ egy vagy esetleg több webcímen (URL) érhető el. Amikor az interneten böngészünk, általában nem foglalkozunk az URL-ekkel, azonban ha több webcímről szeretnénk adatot letölteni, fontos kérdés, hogy miben különbözik ezeknek az URL-je, mivel ennek segítségével tudjuk letölteni a weboldalakat. Az is lényeges, hogy milyen formában, mennyire strukturáltan érhető el az információ az adott weboldalon. Ehhez hasznos megvizsgálni a weboldal HTML kódját, amelyhez a későbbiekben támpontokat nyújtunk.

2. A stratégia megválasztása. Ha azonosítottuk, hogy milyen adatokat szeretnénk letölteni, valamint ezek hol és milyen formában helyezkednek el, akkor kiválasztjuk a megfelelő módszert azok kinyerésére. Ehhez a következőkben különböző szcenáriókat mutatunk be.

3. Weboldal(ak) letöltése. A weboldal HTML dokumentumának letöltéséhez konstruálunk egy letöltő függvényt. Egyetlen webcím esetén ez egyszerűen elvégezhető az **`xml2`** csomag **`read_html`** függvé-

nyével, melynek az adott URL-t kell megadnunk szöveges formában. Több webcím esetén ennél bonyolultabb a dolgunk, ám a **read_html** függvényre ilyen esetben is szükségünk van.

4. Információk kinyerése. Ha eltároltuk a weboldal HTML dokumentumát, már csak technikai kérdés a szükséges információk kinyerése. Ehhez kínál különböző függvényeket az **rvest** csomag. Ilyenek például a **html_nodes**, a **html_table** vagy a **html_text** függvények, melyek alkalmazását részletesen bemutatjuk.

5. Adatelőkészítés. Előfordulhat, hogy a kívánt információt más formában kapjuk meg, mint szeretnénk és szükség van még további lépésekre, hogy statisztikai elemzésre alkalmas adatállományt kapjunk. Ehhez használhatunk egyéb függvényeket, melyek közül néhány tipikus esetet mutatunk be. Gyakran szükség van karaktermanipulációra, amire kiválóan alkalmas a **stringr** csomag.

4.1. Adatok letöltése táblázatból

Mivel az adatok már a weboldalakon is gyakran táblázatba rendezve jelennek meg, az **rvest** csomag egyszerűsített megoldást kínál ezek kinyerésére. A **html_table** függvénynek meg kell adnunk egy HTML dokumentumot vagy ennek egy részletét (HTML node vagy HTML csomópont), és egy listát kapunk vissza, amelynek annyi eleme lesz, amennyi táblázat található az adott weboldalon. A listának minden eleme egy táblázat dataframe formában. A függvény a **<table>** taggel ellátott elemeket azonosítja táblázatként, így győződjünk meg róla, hogy a kívánt adatokat tartalmazó táblázat valóban ilyen. A következő példában a 2018-as FIFA világbajnokság Wikipedia oldaláról töltjük le a játékoskereteket tartalmazó táblázatokat, és egy dataframebe rendezzük őket.

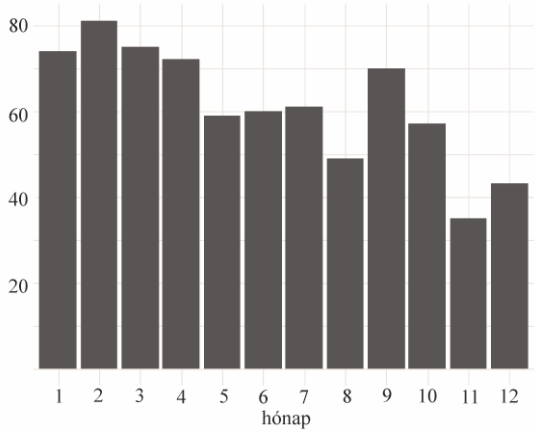
```
mytable <-
read_html("https://en.wikipedia.org/wiki/2018_FIFA_World_Cup_squads")
%>%
  html_table() %>%
  magrittr::extract(1:32) %>%
  bind_rows() %>%
  mutate(
    mob = substr(`Date of birth (age)`, 7, 8) %>%
      as.numeric()    )
```

Első lépésként azonosítjuk a webcímet, majd a **read_html** függvénnyel beolvassuk a HTML dokumentumot. Ezután a **html_table** függvénnyel kinyerjük az

összes táblázatot. A függvény egy 39 elemű listát ad vissza, ám nekünk csak az első 32-re, a játékoskereteket tartalmazó táblázatokra van szükségünk, ezért a **magrittr** csomag **extract** függvényét használjuk. Tegyük fel, hogy egy adatállományban szeretnénk látni az összes játékos születési hónapját. Ehhez további adatelőkészítési lépésekre van szükség. Mivel a listában szereplő táblázatok egyforma oszlopokat tartalmaznak, egyforma változónevekkel, így a **bind_rows** függvénnyel összekapcsolhatjuk őket egy közös dataframebe. Ezután létrehozunk egy új változót, amely a játékosok születési hónapját számformátumban adja meg. Ehhez a **substr** függvénnyel kinyerjük a születési dátumot szöveges formában tartalmazó változóból a megfelelő karaktereket, és azt számformátumra konvertáljuk. A 3. ábra az elemzésre kész adatok alapján a játékosok születési hónap szerinti megoszlását mutatja. Ezek alapján elemezhető az a gyakran vizsgált jelenség, hogy az év elején születettek nagyobb valószínűséggel válnak klasszis sportolóvá (*Musch–Grondin* [2001]).

```
mytable %>%
  ggplot(aes(x = factor(mob))) +
  geom_bar(stat="count") + xlab("hónapok") + ylab(element_blank()) +
  theme_minimal() +
  theme(text = element_text(size = 8, family = "serif"))
```

3. ábra. Születési hónapok gyakorisága
(Frequency of the months of birth)



4.2. Adatok letöltése csomópontokból

Találkozhatunk olyan esetekkel is, amikor a kívánt adatok nem táblázat formátumban jelennek meg a weboldalon, vagy nem hatékony a **html_table** függvény

használata. Az **rvest** csomag függvényeinek segítségével nemcsak **table**, hanem bármilyen más taggel ellátott elemet kinyerhetünk egy HTML dokumentumból. Miután azonosítottuk a webcímet és letöltöttük a weboldal HTML kódját, meg kell találnunk azt a csomópontot vagy csomópontok csoportját, amelyben a szükséges adatok megtalálhatók. Ehhez elengedhetetlen a weboldal HTML kódjának vizsgálata, amelyre bármelyik böngészőben lehetőségünk van. A szükséges csomópontot vagy csomópontok csoportját a **html_node**, illetve **html_nodes** függvények segítségével tudjuk kinyerni, amelyhez vagy egy XPath vagy CSS selectorral érhetünk el. Ebben a tanulmányban a XPath lekérdező nyelvvel nem foglalkozunk részletesen, a csomópontok eléréséhez a CSS selectort használjuk. Ha kinyertük a megfelelő csomóponto(ka)t, azokból több módon készíthetünk numerikus vagy szöveges adatokat, attól függően, hogy az adott csomópontnak melyik részére van szükségünk.

– Leggyakrabban az adott csomópont(ok) szöveges tartalmára van szükségünk, ilyenkor használhatjuk az **rvest** csomag **html_text** függvényét.

– Ha nem szöveges tartalomra, hanem a csomópont(ok) bármilyen más attribútumára van szükségünk, akkor a **html_attr** függvényt használhatjuk, amelyhez meg kell adnunk az attribútum nevét.

– A csomópont tag-jének a nevét (ilyen a korábbiakban említett **<table>** tag is) kinyerhetjük a **html_name** függvénnyel.

A következő példában bemutatjuk az említett módszerek alkalmazását a gyakorlatban. Tegyük fel, hogy a Manchester United labdarúgó csapatának az előző szezonban szerződtetett játékosairól gyűjtünk részletes adatokat a transfermarkt.com weboldalról. Ehhez azonosítjuk, hogy a weboldalon milyen URL-cím alatt és azon belül hol található meg a szükséges információ. Az alábbi kódrészletben a **html_nodes** függvény a megadott selector alapján letölti az összes olyan csomópontot, amely megfelel a kritériumnak. A függvényünk egy listát ad vissza, amely csomópontcsoportokat tartalmaz. Innen a már korábban használt **extract** függvény segítségével kinyerjük a megfelelő csoportot. Fontos megjegyezni, hogy bizonyos adatok ebben az esetben is táblázat formátumban érhetők el, azonban ha az előző fejezetben taglalt **html_table** függvényt használjuk ezen a weboldalon, akkor bizonyos adatokat nem tudunk kinyerni.

```
mynode <- read_html(
    "https://www.transfermarkt.com/manchester-
united/alletransfers/verein/985") %>%
  html_nodes(css ="div.large-6.columns > div") %>%
  magrittr::extract(3)
```

A példa további részében egy dataframe objektumot hozunk létre a szükséges adatokból, melynek minden sora egy-egy játékos és minden oszlopa egy változó. A **direction** változó a szezont és az átigazolás irányát tartalmazza. Ezt a fejlécből nyerjük ki, amely az első ** ** taggel ellátott csomópont a **mynode** HTML dokumentumon belül (ezt írja le a **div:first-of-type** selector). A szöveges tartalmat a **html_text** függvénnyel nyerjük ki. Ezután azonosítjuk azt a csomópontot, amelyben a játékos neve szerepel, majd a **html_nodes** függvény segítségével az összes ilyet kinyerjük. Ebből három különböző változóban eltároljuk a szöveges tartalmat (**html_text**), a játékos azonosítószámát, amelyet az **id** attribútum tartalmaz, valamint a hozzá tartozó hivatkozást a **href** attribútumból. A **transf** változó az átigazolás típusát és pénzbeli értékét tartalmazza.

```
mydata <- data.frame(
      direction = mynode %>% html_node("div:first-of-type") %>%
html_text(trim=TRUE),
      player_name = mynode %>% html_nodes("table > tbody > tr >
td:first-of-type > a") %>% html_text(),
      player_id = mynode %>% html_nodes("table > tbody > tr > td:first-
of-type > a") %>% html_attr("id"),
      player_ref = mynode %>% html_nodes("table > tbody > tr > td:first-
of-type > a") %>% html_attr("href"),
      transf = mynode %>% html_nodes("table > tbody > tr > td:last-of-
type") %>% html_text() %>% str_remove("€"),
      stringsAsFactors = FALSE)
mydata[1:5, c(1:3,5)]
##        direction       player_name player_id transf
## 1 Arrivals 19/20     Harry Maguire    177907 87.00m
## 2 Arrivals 19/20   Bruno Fernandes    240306 55.00m
## 3 Arrivals 19/20 Aaron Wan-Bissaka    477758 55.00m
## 4 Arrivals 19/20      Daniel James    319301 17.00m
## 5 Arrivals 19/20      Odion Ighalo     62121   Loan
```

4.3. Adatok letöltése ábra alapján

Bizonyos esetekben akár a weben található ábrák mögötti adatok letöltése is lehetséges. Legyen célunk *Lionel Messi* becsült piaci értékének, a becslés dátumának, illetve a csapat nevének letöltése a következő kódban látható címről. Az adatok táblázatos formában nem jelennek meg, az azonban az oldal forrásának vizsgálata alapján látható, hogy az ábra JSON formátumú adatokból épül fel. A **script** nevű csomópontok közül az utolsóban találjuk meg a szükséges információkat, amelyeket

a felesleges elemek eltávolítása után a könnyebben kezelhető dataframe formátumra alakítunk. Az így elkészített kód a következő fejezetben bemutatott technikával kombinálva akár a játékosok nagy száma esetén is képes begyűjteni a piaci értéküket.

```
messidata <- read_html(
  "https://www.transfermarkt.de/lionel-
messi/marktwertverlauf/spieler/28003") %>%
  html_nodes('script') %>%
  html_text() %>%
  magrittr::extract(length(.)) %>%
  str_extract("\\[\\{'y'.*\\}\\}\\]") %>%
  str_replace_all(pattern = "\'", replacement =  '\"') %>%
  encodeString() %>%
  fromJSON()

messidata %>% select(datum_mw, y, verein) %>% tail(5)
##      datum_mw         y           verein
## 30 30.05.2018 180000000 FC\\x20Barcelona
## 31 21.12.2018 160000000 FC\\x20Barcelona
## 32 11.06.2019 150000000 FC\\x20Barcelona
## 33 20.12.2019 140000000 FC\\x20Barcelona
## 34 08.04.2020 112000000 FC\\x20Barcelona
```

4.4. Adatok letöltése több hasonló weboldalról

Ha nagyobb adatállományokat szeretnénk kinyerni web scraping módszerrel, gyakran előfordul, hogy nem egy, hanem több webcím alatt találhatók meg a szükséges adatok. Ha az URL-címek hasonlók egymáshoz és azonosítani tudjuk, hogy milyen mintázat alapján térnek el, akkor a korábban bemutatott lépések ilyenkor is alkalmazhatók. Ekkor a letöltő függvényt meghívhatjuk egy ciklusban, amely végigmegy egy listán, és legenerálja a szükséges URL-címeket, valamint letölti a HTML dokumentumokat. A következő példában az angol első osztályú labdarúgó bajnokság végeredményeit szeretnénk letölteni 1992-től a 2018/2019-es szezonig bezárólag a worldfootball.net weboldalról. A különböző szezonok más-más URL-címek alatt érhetők el, azonban a címek csak az évszámokban térnek el, így a lekérdezésekhez használhatunk ciklust.

```
years <- 1992:2018
mynodes <- list()
for(i in 1:length(years)){
  mynodes[[i]] <- read_html(
    paste0("https://www.worldfootball.net/schedule/eng-premier-league-
",years[i], "-", years[i] + 1 ,"-spieltag/")) %>%
    html_node('#site > div.white > div.content > div > div:nth-child(7)
> div > table.standard_tabelle') %>%
    html_table()

  names(mynodes[[i]]) <- make.unique(names(mynodes[[i]]))
  mynodes[[i]] <- mynodes[[i]] %>%
    mutate(Team = Team.1,
           season = years[i]) %>%
    select(-Team.1)}
bind_rows(mynodes) %>% head(10)
##     #               Team M.  W  D  L goals Dif. Pt. season
## 1   1   Manchester United 42 24 12  6 67:31   36  84   1992
## 2   2         Aston Villa 42 21 11 10 57:40   17  74   1992
## 3   3        Norwich City 42 21  9 12 61:65   -4  72   1992
## 4   4    Blackburn Rovers 42 20 11 11 68:46   22  71   1992
## 5   5 Queens Park Rangers 42 17 12 13 63:55    8  63   1992
## 6   6        Liverpool FC 42 16 11 15 62:55    7  59   1992
## 7   7 Sheffield Wednesday 42 15 14 13 55:51    4  59   1992
## 8   8   Tottenham Hotspur 42 16 11 15 60:66   -6  59   1992
## 9   9     Manchester City 42 15 12 15 56:51    5  57   1992
## 10 10          Arsenal FC 42 15 11 16 40:38    2  56   1992
```

A **paste0** függvény legenerálja a futóindextől függő URL-címet szöveges formában. Mindegyik címen ugyanazon csomópontban érhetők el a szükséges adatok, táblázatos formában, amelyet a **html_table** függvénnyel nyerhetünk ki. A ciklus egy listát ad vissza melynek elemei dataframek. Hogy elemzésre kész adatállományt kapjunk, ezeket össze kell csatolnunk, így szükség van még néhány lépésre. A kinyert táblázatokban duplikáltan szerepel a Team oszlopnév, ezért a feleslegest elhagyjuk, valamint generálunk egy **season** változót, amely a bajnokság kezdő évét tartalmazza minden szezonban. Végül a **dplyr** csomag **bind_rows** függvényével összekapcsoljuk a lista elemeit egy közös dataframebe.

4.5. Dinamikus oldalak scrapelése

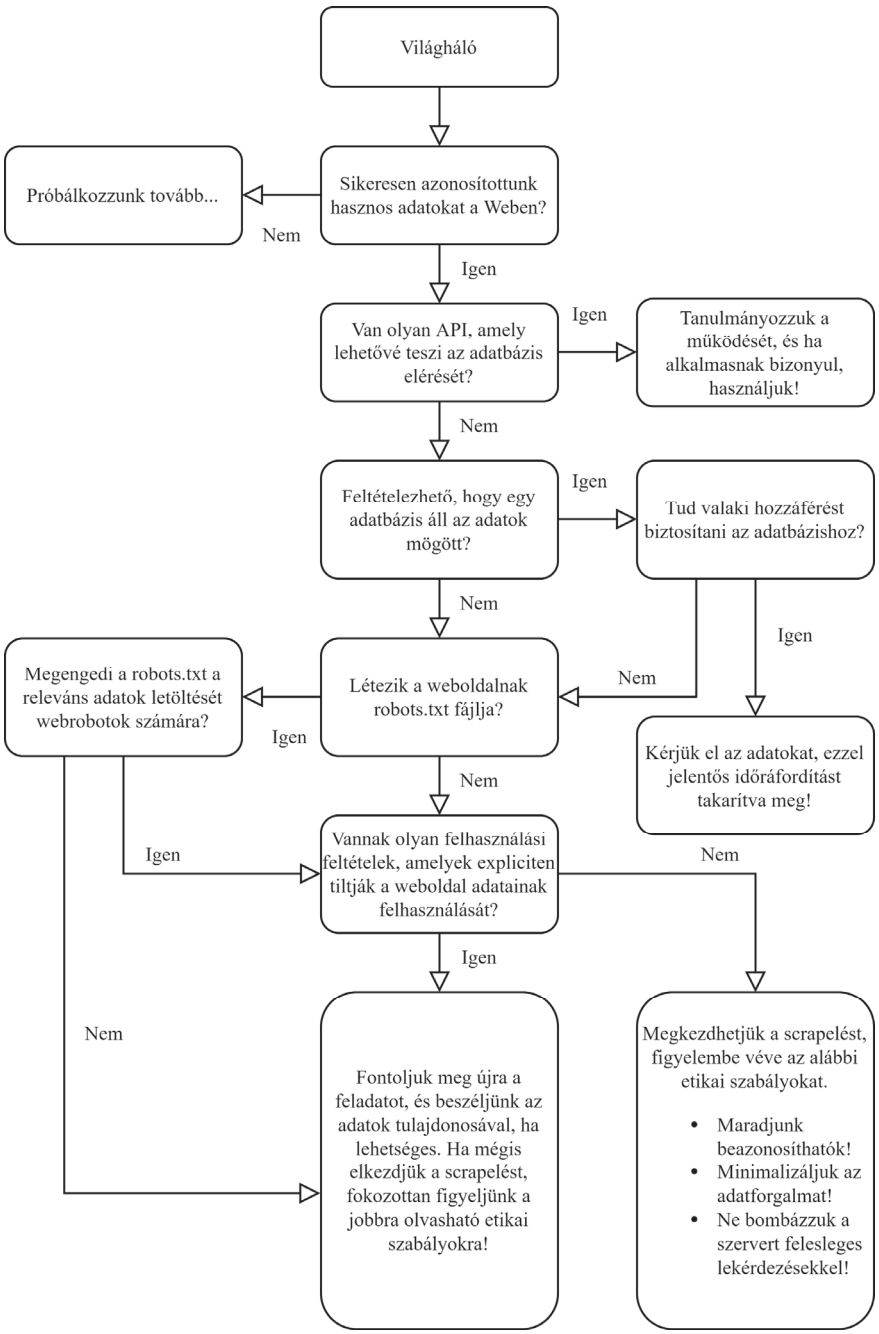
Sok esetben a weboldal technikai megoldása nem teszi lehetővé az URL-ek „végiglapozását", majd a kívánt adatok kinyerését, például azért nem, mert javascript segítségével jelenít meg adatokat dinamikusan, anélkül, hogy az URL megváltozna. Az ilyen oldalak automatikus scrapelésére is van lehetőség, de egy másik eszközt is alkalmaznunk kell. A Selenium (https://selenium.dev/) segítségével a böngésző működése automatizálható, elsősorban tesztelési feladatokhoz készült. A Selenium programozott parancsok segítségével képes tipikus eseményeket végrehajtani egy böngészőben (kattintás, legördülő menü megnyitása, kiválasztás, szöveg begépelése stb.), és az **`RSelenium`** csomag (*Harrison* [2020]) segítségével ez közvetlenül R-ből is megtehető. A következő példában a Selenium szerver elindítása után (mely az R jelenlegi verziójában Google Chrome böngésző esetén az alábbi parancssorral futtatható, azonban ez későbbi verziókban módosításra szorulhat) a rövid parancssor felkeresi a *Statisztikai Szemle* archívumát, kiválasztja a szerzők szerinti keresést, majd a keresőmezőbe beírja, hogy "Kovács" és a keresés gombra kattint. A példában a css selector segítségével történő azonosítást választottuk, az elemek **`#`** azonosítóját a weboldal vizsgálatával kereshetjük meg. Az eredményül kapott oldalról tetszőleges információ (szerzők neve, összefoglaló stb.) kinyerhető és tovább elemezhető, itt a **`res`** változóba egyszerűen a találatok számát tároltuk el (ami jelen sorok írásakor 269).

```
rD <- rsDriver(browser = c("chrome"),
               port = 4567L,
               geckover = NULL,
               chromever = "83.0.4103.39",
               iedrver = NULL,
               phantomver = NULL,
               verbose = FALSE)
remDr <- rD$client
remDr$navigate("https://www.ksh.hu/statszemle_archivum")
remDr$findElement(using = 'css', '#faAuthLink')$clickElement()
remDr$findElement(using = 'css',
'#faFindIt')$sendKeysToElement(list("Kovács"))
remDr$findElement(using = 'css', '#faSearchSubmit')$clickElement()
res <- remDr$findElement(using = 'css', '#faHitCount')$getElementText()
remDr$close()
```

4.6. Etikai kérdések

A web scraping óhatatlanul valamilyen információ lemásolásával jár, így joggal merül fel a kérdés, hogy milyen esetekben ütközik törvénybe. A legkézenfekvőbb és egyben leggyakoribb törvénysértés a szerzői jogok megsértése (*Dreyer–Stockton* [2013]). Általánosan megfogalmazható szabály, hogy mindig átláthatóan végezzük a publikus adatok letöltését, pontosan dokumentálva azok forrását. Ugyanakkor, mivel a törvény országonként és időben is folyamatosan változik, nehéz pontos meghatározást adni arra, hogy mi törvényes és mi nem (*Munzert et al.* [2014]). Számos példát találhatunk arra, amikor a bíróság web scraping tevékenységért elmarasztalt valakit. A legáltalánosabb eset, amikor egy vállalat letölti egy másik cég adatait, majd feldolgozva értékesíti ezt, például az eBay kontra Bidder's Edge eset (https://law.justia.com/cases/federal/district-courts/FSupp2/100/1058/2478126/).
Nagy volumenű, folyamatos adatletöltés céljából gyakran webrobotokat fejlesztenek, amelyek automatizáltan, nagy mennyiségben vadásszák az adatokat, és jelentős adatforgalmat generálnak, ami látható károkkal járhat a weboldal működtetője számára. Ehhez képest a tudományos célú egyedi web scraping feladatok kevesebb forgalmat generálnak, valamint nem kereskedelmi célt szolgálnak, így a jogi következmények kockázata is jóval alacsonyabb (*Munzert et al*. [2014]). Ez azonban nem jelenti azt, hogy amennyiben nem ütközik jogi akadályokba, bármit és bármilyen módszerrel letölthetünk. A „jól viselkedő" webrobotok is követik a weboldalak által megadott ún. robots.txt utasításait (például https://en.wikipedia.org/robots.txt), amelyekben webrobotonként szerepel, hogy milyen tartalmak engedélyezettek számára. A 4. ábra *Munzert et al*. [2014] alapján bemutatja az etikus web scraping lépéseit, amely követendő a hatékonyság, a jogszerűség és az etikett figyelembevételével.

4. ábra. Az etikus web scraping folyamata
(Process of ethical web scraping)



5. Összefoglalás

Tanulmányunkban olyan alternatív adatgyűjtési technikákat mutattunk be, melyek valamilyen szempontból túlmutatnak a megszokott módszereken. Az egyre szélesebb körben elérhető REST API szolgáltatások mellett részletesebben foglalkoztunk a web scraping megoldásokkal és az etikus scraping kérdésével. Az adatszerzési módszerekhez példákat adtunk az ingyenes R nyelv segítségével. A példák jó része a megfelelő csomagok telepítése után reprodukálható, néhány esetben azonban regisztráció vagy egyéb beállítás szükséges, amiben a szerzők szívesen segítenek. A tudományos kutatásban egyre nagyobb az igény az állítások statisztikai módszerekkel történő alátámasztására. Ennek azonban akadálya lehet, ha az adatok nem állnak rendelkezésre. A tanulmányban bemutatott módszerek alkalmazásával a szekunder adatforrásokból elérhető adatmennyiség kibővíthető, így a kutatásokban használt statisztikai elemzések minősége javítható, valamint több empirikus kutatás válik megvalósíthatóvá. A megismételhető kutatás szemléletéhez hozzá tartozik az adatgyűjtés módjának átláthatósága. Ha külső személyek is vissza tudják követni a felhasznált adatokat, az biztosítja az adatok validitását, ezáltal javítja a kutatás hitelességét. Az írásunkban bemutatott adatgyűjtési módszerek programkód segítségével működnek, így egyszerűen visszakövethetővé tehetők a forráskód közzétételével. Amennyiben egy kutatás a megismételhetőségre törekszik, javasoljuk ezen lépések alkalmazását.

Irodalom

DARÓCZI G. [2016]: Alkalmazott Statisztika? R! *Statisztikai Szemle*. 94. évf. 11–12. sz. 1108–1122. old.

DARÓCZI G. – TÓTH G. [2013]: Felhőtlen statisztika a felhőben. *Statisztikai Szemle*. 91. évf. 11. sz. 1118–1142. old.

DREYER, A. J. – STOCKTON, J. [2013]: Internet "data scraping": A primer for counseling clients. *New York Law Journal*. No. 7. pp. 1–3.

GANDRUD, C. [2016]: *Reproducible Research with R and R Studio. Second Edition*. CRC Press. Boca Raton.

GLEZ-PEÑA, D. – LOURENÇO, A. – LÓPEZ-FERNÁNDEZ, H. – REBOIRO-JATO, M. – FDEZ-RIVEROLA, F. [2014]: Web scraping technologies in an API world. *Briefings in Bioinformatics*. Vol. 15. Issue 5. pp. 788–797. https://doi.org/10.1093/bib/bbt026

HAJDU, O. [2018]: Többváltozós statisztikai R Open alkalmazások. *Statisztikai Szemle*. 96. évf. 10. sz. 1021–1047. old. https://doi.org/10.20311/stat2018.10.hu1021

KEARNEY, M. W. [2019]: rtweet: Collecting and analyzing Twitter data. *Journal of Open Source Software*. Vol. 4. No. 42. pp. 1829. https://doi.org/10.21105/joss.01829

MITCHELL, R. [2018]: *Web Scraping with Python: Collecting More Data from the Modern Web*. O'Reilly Media. Sebastopol.

MUNZERT, S. – RUBBA, C. – MEIßNER, P. – NYHUIS, D. [2014]: *Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining*. John Wiley & Sons. Hoboken.

MUSCH, J. – GRONDIN, S. [2001]: Unequal competition as an impediment to personal development: A review of the relative age effect in sport. *Developmental Review*. Vol. 21. No. 2. pp. 147–167. https://doi.org/10.1006/drev.2000.0516

WICKHAM, H. – AVERICK, M. – BRYAN, J. – CHANG, W. – MCGOWAN, L. D. – FRANÇOIS, R. – GROLEMUND, G. – HAYES, A. – HENRY, L. – HESTER, J. – KUHN, M. – PEDERSEN, T. L. – MILLER, E. – BACHE, S. M. – MÜLLER, K. – OOMS, J. – ROBINSON, D. – SEIDEL, D. P. – SPINU, V. –TAKAHASHI, K. – VAUGHAN, D. – WILKE, C. – WOO, K. – YUTANI, H. [2019a]: Welcome to the tidyverse. *Journal of Open Source Software.* Vol. 4. No. 43. pp. 1686. https://doi.org/10.21105/joss.01686

XIE, Y. [2015]: *Dynamic Documents with R and Knitr*. CRC Press. Boca Raton.

Internetes hivatkozások

AREL-BUNDOCK, V. [2019]: *WDI: World Development Indicators (World Bank)*. https://CRAN.R-project.org/package=WDI

COENE, J. [2020]: *twinetverse: Easily Install and Load Packages for 'Twitter' Network Analysis and Visualisation*. http://twinetverse.john-coene.com/

DOWLE, M. – SRINIVASAN, A. [2019]: *data.table: Extension of 'data.frame'*. https://CRAN.R-project.org/package=data.table

HARRISON, J. [2020]: *RSelenium: R Bindings for 'Selenium WebDriver'*. https://CRAN.R-project.org/package=RSelenium

KONCZ, T. – VARKOLY, B. – LUKACS, P. – KOCSIS, E. [2020]: *googleCloudVisionR: Access to the 'Google Cloud Vision' API for Image Recognition, OCR and Labeling*. https://CRAN.R-project.org/package=googleCloudVisionR

MASSICOTTE, P. – EDDELBUETTEL, D. [2020]: *gtrendsR: Perform and Display Google Trends Queries*. https://CRAN.R-project.org/package=gtrendsR

MÉSZÁROS, M. [2020]: *restatapi: Search and Retrieve Data from Eurostat Database*. https://CRAN.R-project.org/package=restatapi

MÜLLER, K. – WICKHAM, H. – JAMES, D. A. – FALCON, S. [2020]: *RSQLite: 'SQLite' Interface for R*. https://CRAN.R-project.org/package=RSQLite

OOMS, J. [2014]: *The jsonlite Package: A Practical and Consistent Mapping between JSON Data and R Objects. arXiv:1403.2805 [stat.CO]*. https://arxiv.org/abs/1403.2805

OOMS, J. – JAMES, D. – DEBROY, S. – WICKHAM, H. – HORNER, J. [2020]: *RMySQL: Database Interface and 'MySQL' Driver for R*. https://CRAN.R-project.org/package=RMySQL

R SPECIAL INTEREST GROUP ON DATABASES (R-SIG-DB) – WICKHAM, H. – MÜLLER, K. [2019]: *DBI: R Database Interface.* https://CRAN.R-project.org/package=DBI

RIPLEY, B. – LAPSLEY, M. [2019]: *RODBC: ODBC Database Access*. https://CRAN.R-project.org/package=RODBC

WICKHAM, H. [2019]: *httr: Tools for Working with URLs and HTTP*. https://CRAN.R-project.org/package=httr

WICKHAM, H. – BRYAN, J. [2019]: *readxl: Read Excel Files*. https://CRAN.R-project.org/package=readxl

WICKHAM, H. – FRANÇOIS, R. – HENRY, L. – MÜLLER, K. [2020]: *dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr

WICKHAM, H. – HESTER, J. – FRANÇOIS, R. [2018]: *readr: Read Rectangular Text Data*. https://CRAN.R-project.org/package=readr

WICKHAM, H. – MILLER, E. [2020]: *haven: Import and Export 'SPSS', 'Stata' and 'SAS' Files*. https://CRAN.R-project.org/package=haven

WICKHAM, H. – OOMS, J. – MÜLLER, K. [2019b]: *RPostgres: 'Rcpp' Interface to 'PostgreSQL'*. https://CRAN.R-project.org/package=RPostgres

WICKHAM, H. – RUIZ, E. [2020]: *dbplyr: A 'dplyr' Back End for Databases*. https://CRAN.R-project.org/package=dbplyr