semana_1

Luis Ambrocio

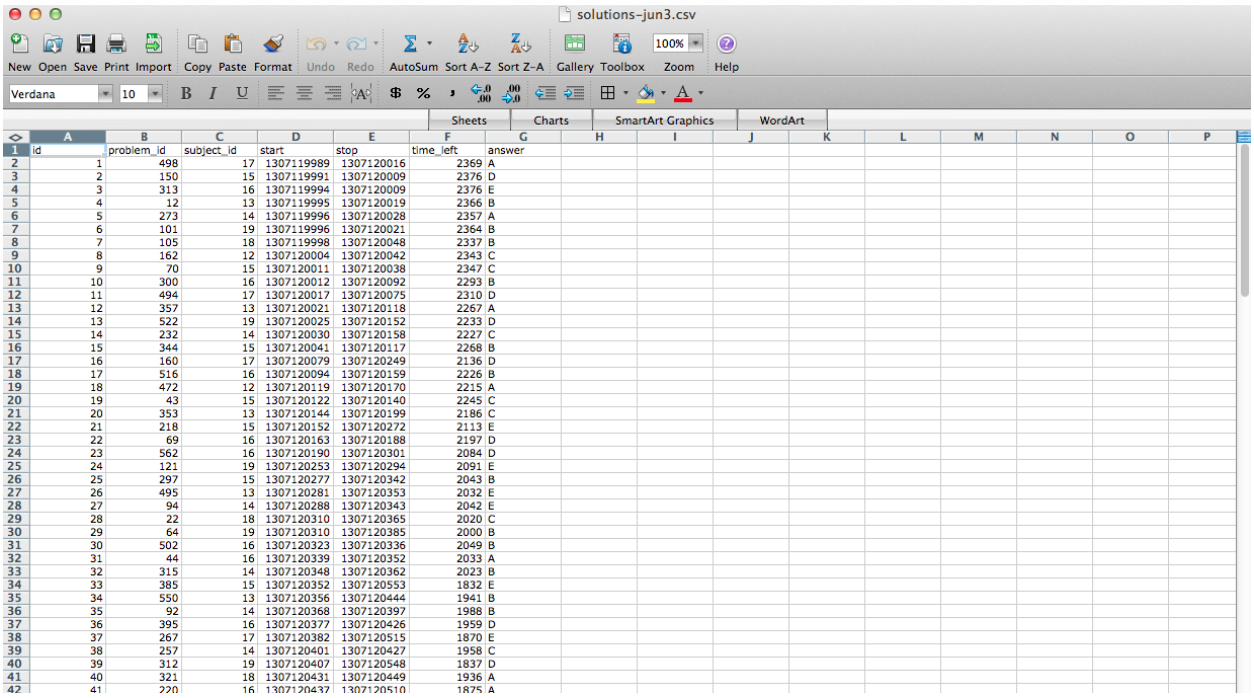
Contents

Motivacion y prerequisitos

Sobre este curso

- Este curso cubre las ideas básicas para preparar los datos para su análisis.
  - Encontrar y extraer datos sin procesar
  - Principios de ordenación de datos y cómo hacer que los datos estén ordenados
  - Implementación práctica a través de una variedad de paquetes R

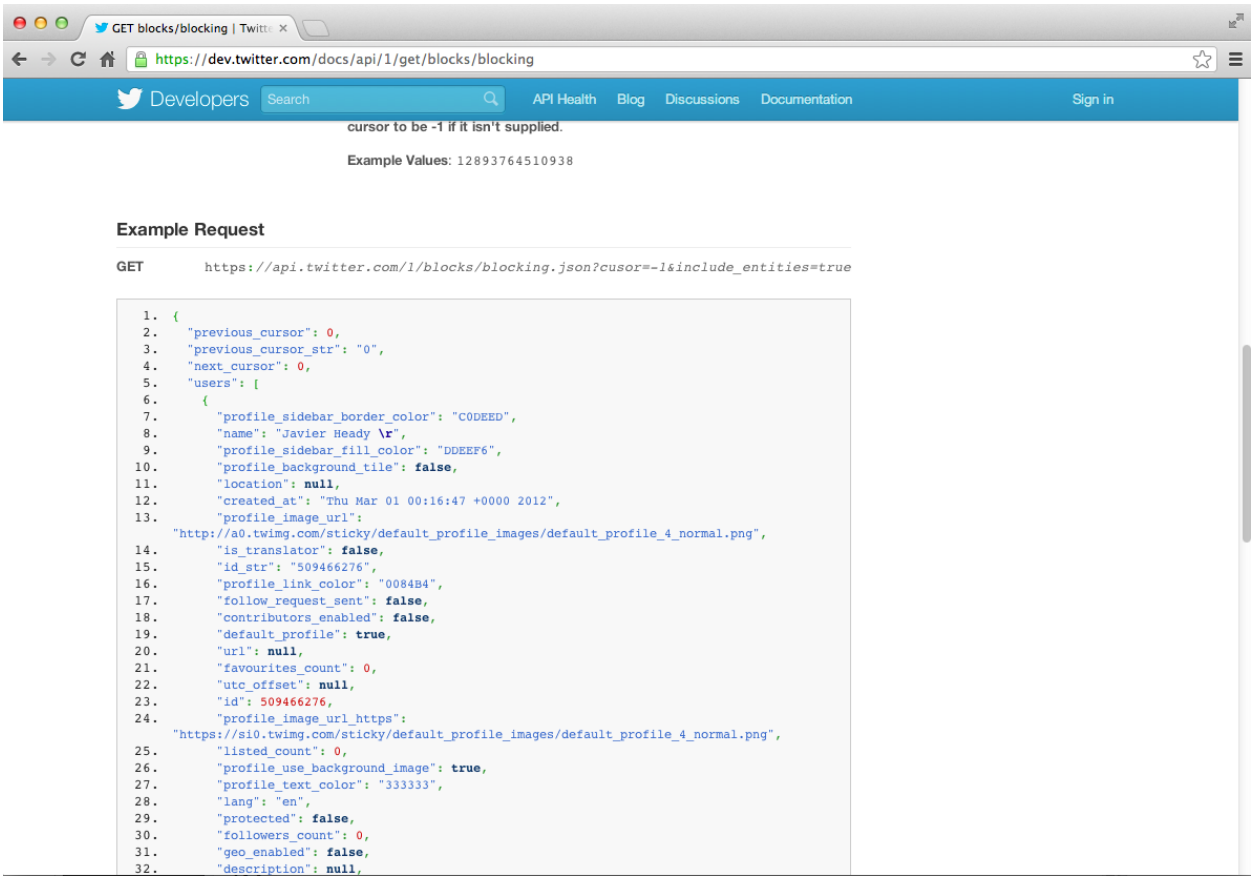
Cómo desea que se vean los datos

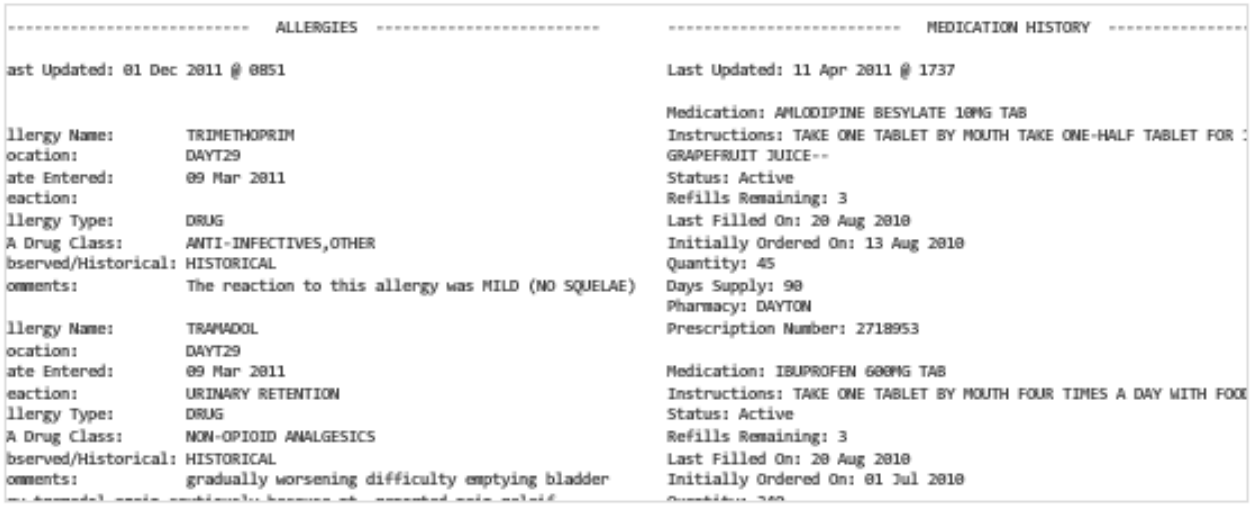


	A	B	C	D	E	F	G
1	id	problem_id	subject_id	start	stop	time_left	answer
2	1	498	17	1307119989	1307120016	2369	A
3	2	150	15	1307119991	1307120009	2376	D
4	3	313	16	1307119994	1307120009	2376	E
5	4	12	13	1307119995	1307120019	2366	B
6	5	273	14	1307119996	1307120028	2357	A
7	6	101	19	1307119996	1307120021	2364	B
8	7	105	18	1307119998	1307120048	2337	B
9	8	162	12	1307120004	1307120042	2343	C
10	9	70	15	1307120011	1307120038	2347	C
11	10	300	16	1307120012	1307120092	2293	B
12	11	494	17	1307120017	1307120075	2310	D
13	12	357	13	1307120021	1307120118	2267	A
14	13	522	19	1307120025	1307120152	2233	D
15	14	232	14	1307120030	1307120158	2227	C
16	15	344	15	1307120041	1307120117	2268	B
17	16	160	17	1307120079	1307120249	2136	D
18	17	516	16	1307120094	1307120159	2226	B
19	18	472	12	1307120119	1307120170	2215	A
20	19	43	15	1307120122	1307120140	2245	C
21	20	353	13	1307120144	1307120199	2186	C
22	21	218	15	1307120152	1307120272	2113	E
23	22	69	16	1307120163	1307120188	2197	D
24	23	562	16	1307120190	1307120301	2084	D
25	24	121	19	1307120253	1307120294	2091	E
26	25	297	15	1307120277	1307120342	2043	B
27	26	495	13	1307120281	1307120353	2032	E
28	27	94	14	1307120288	1307120343	2042	E
29	28	22	18	1307120310	1307120365	2020	C
30	29	64	19	1307120310	1307120385	2000	B
31	30	502	16	1307120323	1307120336	2049	B
32	31	44	16	1307120339	1307120352	2033	A
33	32	315	14	1307120348	1307120362	2023	B
34	33	385	15	1307120352	1307120553	1832	E
35	34	550	13	1307120356	1307120444	1941	B
36	35	92	14	1307120368	1307120397	1988	B
37	36	395	16	1307120377	1307120426	1959	D
38	37	267	17	1307120382	1307120515	1870	E
39	38	257	14	1307120401	1307120427	1958	C
40	39	312	19	1307120407	1307120548	1837	D
41	40	321	18	1307120431	1307120449	1936	A
42	41	220	16	1307120437	1307120510	1875	A

como los datos realmente se ven

```
@HWI-EAS121:4:100:1783:550#0/1
CGTTACGAGATCGGAAGAGCGGTTCAGCAGGAATGCCGAGACGGATCTCGTATGCGGTCTGCTGCGTGACAAGACAGGGG
+HWI-EAS121:4:100:1783:550#0/1
aaaaa`b_aa`aa`YaX]aZ`aZM^Z]YRa]YSG[[ZREQLHESDHNDDHNMEEDDMPENITKFLFEEDDDHEJQMEDDD
@HWI-EAS121:4:100:1611#0/1
GGGTGGGCATTTCCACTCGCAGTATGGGTTGCCGCACGACAGGCAGCGGTCAGCCTGCGCTTTGGCCTGGCCTTCGGAAA
+HWI-EAS121:4:100:1611#0/1
a``^\__`_```^a``a`^a_^__]a_]\]`a_______`_^^`]X]_]XTV_\]]NX_XVX]]_TTTTG[VTHPN]VFDZ
@HWI-EAS121:4:100:322#0/1
CGTTTATGTTTTTGAATATGTCTTATCTTAACGGTTATATTTTAGATGTTGGTCTTATTCTAACGGTCATATATTTTCTA
+HWI-EAS121:4:100:322#0/1
abaa`^aaaaabbbaabababbbbbb`bbbb_bbbbbbbb`bbbaV^_a``a``]``aT]a__V\]]_]^a`]a_abbaV__
@HWI-EAS121:4:100:1394#0/1
GGGTCTTTATTGGTCTGGTGATCCCCCATATTCTCCGGTTGTGTGGTTTAACCGATCATCGCGCATTACTTCCCGGCTGC
+HWI-EAS121:4:100:1394#0/1
```[aa\b^^[]aabbb][`a_abbb`a``bbbbbabaabaaaab_VZa_^___bab_X`[a\HV_[_]_[^_X\T_VQQ
@HWI-EAS121:4:100:207#0/1
CCCTGGGAGATCGGAAGAGCGGTTCAGCAGGAATGCCGAGACCGATCTCGTATGCCGTCTTCTGCTTGAAAAAAAAAACA
+HWI-EAS121:4:100:207#0/1
abba`Xa\^\\`aa]ba__bba[a_O_a`aa`aa`a]^V]X_a^YS\R_\H_[]\ZTDUZZUSOPX]]POP\GS\WSHHD
@HWI-EAS121:4:100:455#0/1
GGGTAATTCAGGGACAATGTAATGGCTGCACAAAAAAATACATCTTTCATGTTCCATTGCACCATTGACAAATACATATT
+HWI-EAS121:4:100:455#0/1
abb_babbabaabbbbbbbbbbbbbbba\`b`\abbbabbbbabbbbbbaabbbbb`bb`ab_O_bab_Q_bbabaa_a
```
```





la meta de este curso

Datos brutos -> Script de procesamiento -> datos ordenados-> análisis de datos -> comunicación de datos

datos crudos y procesados

Datos crudos * La fuente original de los datos * A menudo es difícil de usar para análisis de datos. * Análisis de datos *incluye* procesamiento * Es posible que los datos sin procesar solo necesiten procesarse una

vez

http://en.wikipedia.org/wiki/Raw_data

Datos procesados * Datos que están listos para analizar * El procesamiento puede incluir fusión, subconjunto, transformación, etc. * Puede haber estándares para el procesamiento. * Todos los pasos deben registrarse

http://en.wikipedia.org/wiki/Computer_data_processing

Los componentes de los datos ordenados

Las cuatro cosas que debes tener

1. Los datos brutos.
2. Un conjunto de datos ordenado
3. Un libro de códigos que describe cada variable y sus valores en el ordenado conjunto de datos.
4. Una receta explícita y exacta que usaste para ir de 1 -> 2,3.

Los datos brutos

- El extraño archivo binario que escupe su máquina de medición
- El archivo de Excel sin formato con 10 hojas de trabajo que le envió la empresa con la que contrató
- Los datos JSON complicados que obtuviste al raspar la API de Twitter
- Los números ingresados a mano que recolectó mirando a través de un microscopio

Sabes que los datos sin procesar están en el formato correcto si

1. No ejecuté ningún software en los datos.
2. No manipuló ninguno de los números de los datos.
3. No eliminó ningún dato del conjunto de datos.
4. No resumió los datos de ninguna manera

los datos ordenados

1. Cada variable que mida debe estar en una columna.
2. Cada observación diferente de esa variable debe estar en una fila diferente
3. Debe haber una tabla para cada "tipo" de variable.
4. Si tiene varias tablas, deben incluir una columna en la tabla que permita vincularlas

Algunos otros consejos importantes

- Incluya una fila en la parte superior de cada archivo con nombres de variables.
- Hacer que los nombres de las variables sean legibles por humanos AgeAtDiagnosis en lugar de AgeDx
- En general, los datos deben guardarse en un archivo por tabla.

El libro de códigos

1. Información sobre las variables (¡incluidas las unidades!) En el conjunto de datos no contenidas en los datos ordenados
2. Información sobre las elecciones resumidas que hizo
3. Información sobre el diseño del estudio experimental que utilizó

Algunos otros consejos importantes

- Un formato común para este documento es un archivo de texto / Word.
- Debe haber una sección llamada "Diseño del estudio" que tenga una descripción detallada de cómo recopiló los datos.
- Debe haber una sección llamada "Libro de códigos" que describa cada variable y sus unidades.

La lista de instrucciones

- Idealmente un script de computadora (en R :-), pero supongo que Python también está bien . . .)
- La entrada para el script son los datos sin procesar
- La salida son los datos procesados y ordenados.
- No hay parámetros para el script.

En algunos casos, no será posible escribir todos los pasos. En ese caso, debe proporcionar instrucciones como:

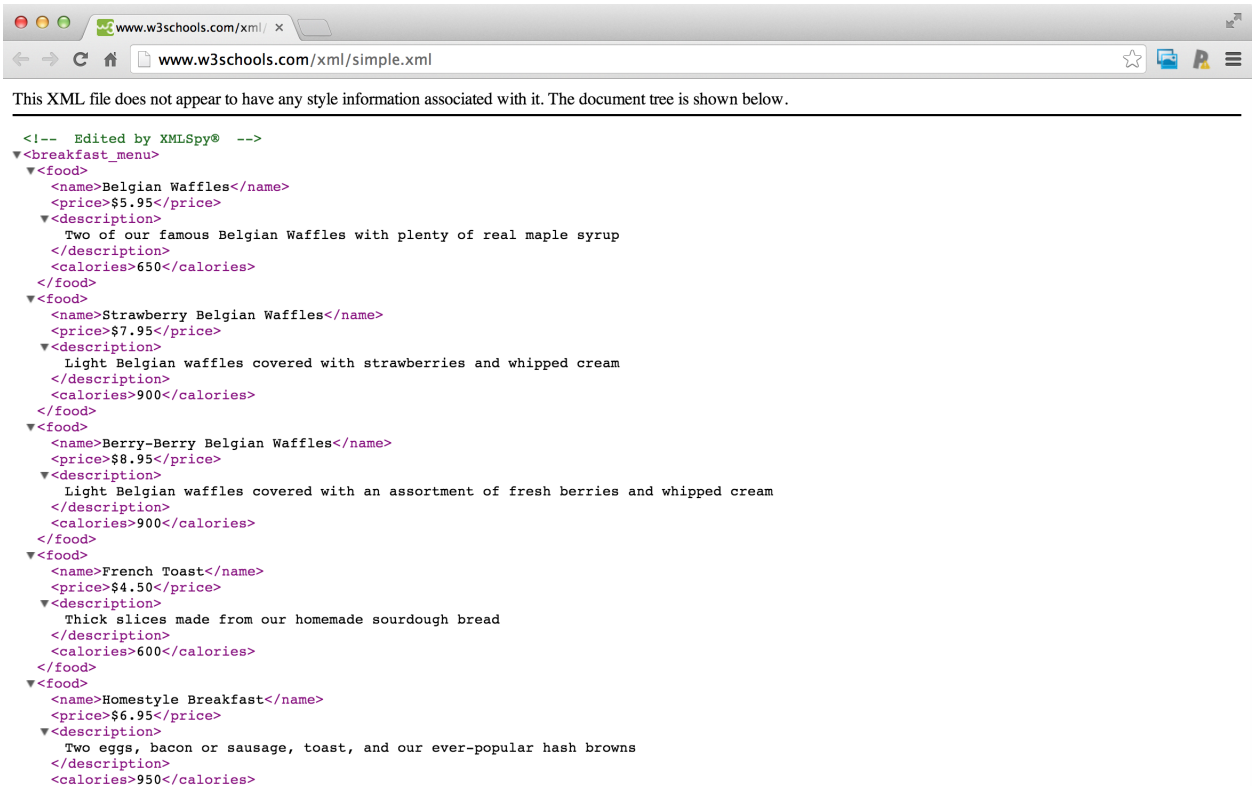
1. Paso 1: tome el archivo sin procesar, ejecute la versión 3.1.2 del software de resumen con los parámetros a = 1, b = 2, c = 3
2. Paso 2: ejecute el software por separado para cada muestra
3. Paso 3: tome la columna tres de outputfile.out para cada muestra y esa es la fila correspondiente en el conjunto de datos de salida

https://github.com/jtleek/datasharing

Leyendo XML

- Lenguaje de marcado extensible
- Se utiliza con frecuencia para almacenar datos estructurados.
- Particularmente utilizado en aplicaciones de Internet.
- La extracción de XML es la base de la mayoría de los web scraping.
- Componentes
  - Marcado: etiquetas que dan estructura al texto
  - Contenido: el texto real del documento
- etiquetas, elementos y atributos
- Las etiquetas corresponden a las etiquetas generales
  - Etiquetas de inicio `<sección>`
  - Etiquetas finales `</section>`
  - Etiquetas vacías `<line-break />`
- Los elementos son ejemplos específicos de etiquetas.
  - `<Saludo> Hola, mundo </Saludo>`
- Los atributos son componentes de la etiqueta
  - `<img src =" jeff.jpg "alt =" instructor "/>`
  - `<step number =" 3 "> Conecta A con B. </step>`

Ejemplo de un archivo XML



Leyendo el archivo en R

```
library(XML)
if(!file.exists("D:/luism/Descargas/simple.xml")){
fileUrl <- "http://www.w3schools.com/xml/simple.xml"
download.file(fileUrl, destfile = "D:/luism/Descargas/simple.xml")}

doc <- xmlTreeParse("D:/luism/Descargas/simple.xml", useInternalNodes = TRUE)

rootNode <- xmlRoot(doc)
xmlName(rootNode)
```

```
## [1] "breakfast_menu"
```

```
names(rootNode)
```

```
##   food   food   food   food   food
## "food" "food" "food" "food" "food"
```

Acceda directamente a partes del documento XML

```
rootNode[[1]]
```

```
## <food>
##   <name>Belgian Waffles</name>
##   <price>$5.95</price>
##   <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
##   <calories>650</calories>
## </food>
```

```
rootNode[[1]][[1]]
```

```
## <name>Belgian Waffles</name>
```

Extrae partes del archivo mediante programación

```
xmlSApply(rootNode,xmlValue)
```

```
##
##                               "Belgian Waffles$5.95Two of our famous Belgian Waffles with plenty of
##
##                     "Strawberry Belgian Waffles$7.95Light Belgian waffles covered with strawberries a
##
## "Berry-Berry Belgian Waffles$8.95Light Belgian waffles covered with an assortment of fresh berries a
##
##                                                 "French Toast$4.50Thick slices made from our homemade
##
##                          "Homestyle Breakfast$6.95Two eggs, bacon or sausage, toast, and our ever-popu
```

XPath

- */node* Nodo de nivel superior
- *//nodo* Nodo en cualquier nivel
- *node[@attr-name]* Nodo con un nombre de atributo
- *node[@attr-name = 'bob']* Nodo con nombre de atributo attr-name = 'bob'

Information from: http://www.stat.berkeley.edu/~statcur/Workshop2/Presentations/XML.pdf

```
xpathSApply(rootNode,"//name",xmlValue)
```

```
## [1] "Belgian Waffles"             "Strawberry Belgian Waffles"
## [3] "Berry-Berry Belgian Waffles" "French Toast"
## [5] "Homestyle Breakfast"
```

```
xpathSApply(rootNode,"//price",xmlValue)
```

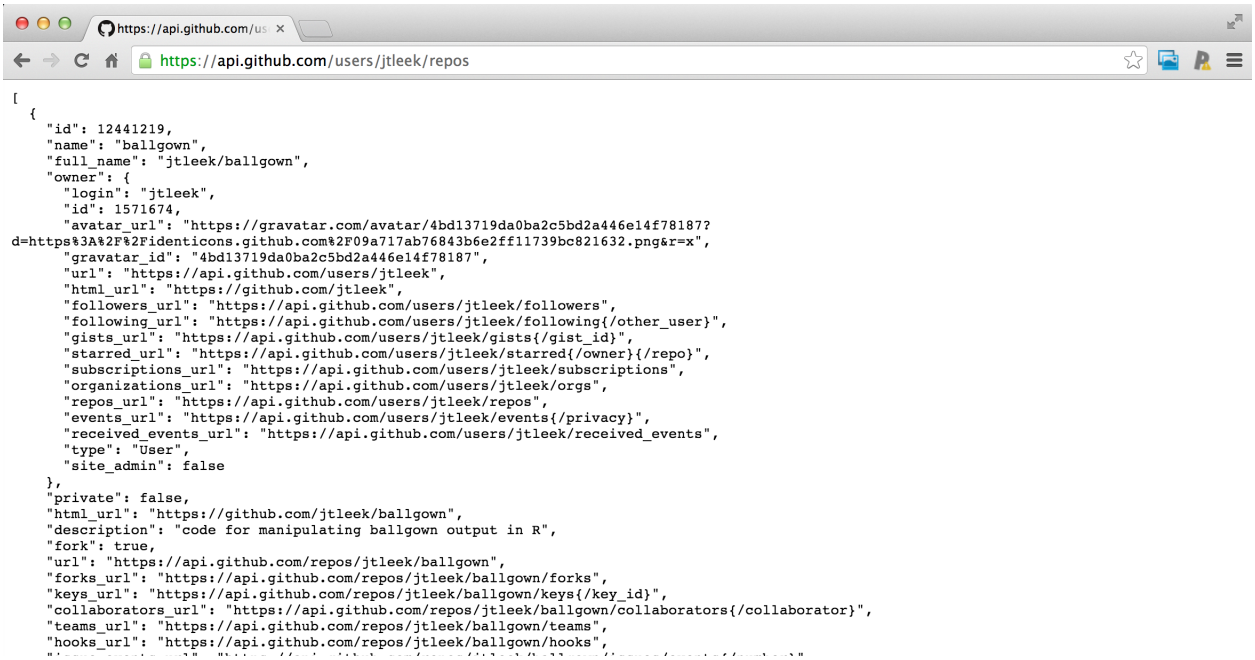
```
## [1] "$5.95" "$7.95" "$8.95" "$4.50" "$6.95"
```

Notas y otros recursos

- Official XML tutorials short, long
- An outstanding guide to the XML package
- http://en.wikipedia.org/wiki/XML

Leyendo JSON

- Notación de objetos de Javascript(Javascript Object Notation)
- Almacenamiento de datos ligero
- Formato común para datos de interfaces de programación de aplicaciones (API)
- Estructura similar a XML pero diferente sintaxis / formato
- Datos almacenados como
  - Números (double)
  - Strings (entre comillas dobles)
  - Booleano (*verdadero* o *falso*)
  - Matriz (ordenada, separada por comas encerrada entre corchetes *[]*)
  - Objeto (desordenado, colección de claves separada por comas: pares de valores entre llaves *{}*)



leyendo datos:

```
library(jsonlite)
jsonData <- fromJSON("https://api.github.com/users/jtleek/repos")
names(jsonData)
```

```
##  [1] "id"               "node_id"          "name"
##  [4] "full_name"        "private"          "owner"
##  [7] "html_url"         "description"      "fork"
## [10] "url"              "forks_url"        "keys_url"
## [13] "collaborators_url" "teams_url"       "hooks_url"
## [16] "issue_events_url" "events_url"       "assignees_url"
## [19] "branches_url"     "tags_url"         "blobs_url"
## [22] "git_tags_url"     "git_refs_url"     "trees_url"
## [25] "statuses_url"     "languages_url"    "stargazers_url"
## [28] "contributors_url" "subscribers_url"  "subscription_url"
## [31] "commits_url"      "git_commits_url"  "comments_url"
## [34] "issue_comment_url" "contents_url"    "compare_url"
## [37] "merges_url"       "archive_url"      "downloads_url"
## [40] "issues_url"       "pulls_url"        "milestones_url"
## [43] "notifications_url" "labels_url"      "releases_url"
## [46] "deployments_url"  "created_at"       "updated_at"
## [49] "pushed_at"        "git_url"          "ssh_url"
## [52] "clone_url"        "svn_url"          "homepage"
## [55] "size"             "stargazers_count" "watchers_count"
## [58] "language"         "has_issues"       "has_projects"
## [61] "has_downloads"    "has_wiki"         "has_pages"
## [64] "forks_count"      "mirror_url"       "archived"
## [67] "disabled"         "open_issues_count" "license"
## [70] "forks"            "open_issues"      "watchers"
## [73] "default_branch"
```

```
jsonData$name
```

```
##  [1] "2018"                     "ads2020"
##  [3] "advdatasci"               "advdatasci-project"
##  [5] "advdatasci-swirl"         "advdatasci15"
##  [7] "advdatasci16"             "advdatasci_swirl"
##  [9] "ballgown"                 "big_course"
## [11] "bookdown-start"           "books"
## [13] "capitalIn21stCenturyinR" "careerplanning"
## [15] "coc"                      "courses"
## [17] "COVID-19"                 "crsra"
## [19] "cshlcg-labs"              "data"
## [21] "dataanalysis"             "datascientist"
## [23] "datasharing"              "datawomenontwitter"
## [25] "day1"                     "derfinder"
## [27] "derfinder-1"              "DSM"
## [29] "EDA-Project"              "escalatr"
```

Objetos anidados en JSON

```
names(jsonData$owner)
```

```
##  [1] "login"               "id"                  "node_id"
##  [4] "avatar_url"          "gravatar_id"         "url"
##  [7] "html_url"            "followers_url"       "following_url"
## [10] "gists_url"           "starred_url"         "subscriptions_url"
## [13] "organizations_url"   "repos_url"           "events_url"
## [16] "received_events_url" "type"                "site_admin"
```

```
jsonData$owner$login
```

```
##  [1] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"
##  [9] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"
## [17] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"
## [25] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"
```

escribir data frames en JSON

```
myjson <- toJSON(head(iris), pretty=TRUE)
cat(myjson)
```

```
## [
##   {
##     "Sepal.Length": 5.1,
##     "Sepal.Width": 3.5,
##     "Petal.Length": 1.4,
##     "Petal.Width": 0.2,
##     "Species": "setosa"
##   },
##   {
##     "Sepal.Length": 4.9,
##     "Sepal.Width": 3,
##     "Petal.Length": 1.4,
##     "Petal.Width": 0.2,
##     "Species": "setosa"
##   },
##   {
```

```
##     "Sepal.Length": 4.7,
##     "Sepal.Width": 3.2,
##     "Petal.Length": 1.3,
##     "Petal.Width": 0.2,
##     "Species": "setosa"
##   },
##   {
##     "Sepal.Length": 4.6,
##     "Sepal.Width": 3.1,
##     "Petal.Length": 1.5,
##     "Petal.Width": 0.2,
##     "Species": "setosa"
##   },
##   {
##     "Sepal.Length": 5,
##     "Sepal.Width": 3.6,
##     "Petal.Length": 1.4,
##     "Petal.Width": 0.2,
##     "Species": "setosa"
##   },
##   {
##     "Sepal.Length": 5.4,
##     "Sepal.Width": 3.9,
##     "Petal.Length": 1.7,
##     "Petal.Width": 0.4,
##     "Species": "setosa"
##   }
## ]
```

de JSON a dataframe

```
iris2 <- fromJSON(myjson)
head(iris2)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

otros recursos

- http://www.json.org/
- A good tutorial on jsonlite - http://www.r-bloggers.com/new-package-jsonlite-a-smarter-json-encoderdecoder/
- jsonlite vignette
- http://en.wikipedia.org/wiki/JSON

Usando data.table

- Hereda de dataframe
  - Todas las funciones que aceptan data.frame funcionan en data.table
- Escrito en C por lo que es mucho más rápido

- Mucho, mucho más rápido al crear subconjuntos, agrupar y actualizar

creacion de tablas

```
library(data.table)
DF = data.frame(x=rnorm(9),y=rep(c("a","b","c"),each=3),z=rnorm(9))
head(DF,3)
```

```
##           x y          z
## 1 0.1281210 a -1.1594222
## 2 0.1017796 a  0.6211235
## 3 1.3595822 a -0.2538872
```

```
DT = data.table(x=rnorm(9),y=rep(c("a","b","c"),each=3),z=rnorm(9))
head(DT,3)
```

```
##            x y          z
## 1:  2.318047 a -0.5138050
## 2:  1.817984 a  0.1147285
## 3: -1.233444 a -0.2324543
```

mirar todos los data.tables guardados en la memoria

```
tables()
```

```
##    NAME NROW NCOL MB  COLS KEY
## 1:   DT    9    3  0 x,y,z
## Total: 0MB
```

extraer subconjuntos

```
DT[2,]
```

```
##           x y         z
## 1: 1.817984 a 0.1147285
```

```
DT[DT$y=="a",]
```

```
##            x y          z
## 1:  2.318047 a -0.5138050
## 2:  1.817984 a  0.1147285
## 3: -1.233444 a -0.2324543
```

```
DT[c(2,3),]
```

```
##            x y          z
## 1:  1.817984 a  0.1147285
## 2: -1.233444 a -0.2324543
```

```
DT[,c(2,3)]
```

```
##    y           z
## 1: a -0.51380496
## 2: a  0.11472847
## 3: a -0.23245430
## 4: b -0.81122782
## 5: b -0.36909562
## 6: b  0.74440117
## 7: c -0.30469462
## 8: c  1.66394825
## 9: c -0.02804972
```

Subconjunto de columnas en data.table

- La función de subconjunto se modifica para data.table
- El argumento que pasa después de la coma se llama "expresión"
- En R, una expresión es una colección de declaraciones encerradas entre corchetes

ejemplos de declaraciones

```
{
  x = 1
  y = 2
}
k = {print(10); 5}
```

```
## [1] 10
```

```
print(k)
```

```
## [1] 5
```

Calcular valores para variables con expresiones

```
DT[,list(mean(x),sum(z))]
```

```
##          V1        V2
## 1: 0.314394 0.2637508
```

```
DT[,table(y)]
```

```
## y
## a b c
## 3 3 3
```

agregando nuevas columnas

```
DT[,w:=z^2]

DT2 <- DT
DT[, y:= 2]

# el cambio se hace desde la original a la copia
head(DT,n=3)
```

```
##            x y          z          w
## 1:  2.318047 2 -0.5138050 0.26399553
## 2:  1.817984 2  0.1147285 0.01316262
## 3: -1.233444 2 -0.2324543 0.05403500
```

```
head(DT2,n=3)
```

```
##            x y          z          w
## 1:  2.318047 2 -0.5138050 0.26399553
## 2:  1.817984 2  0.1147285 0.01316262
## 3: -1.233444 2 -0.2324543 0.05403500
```

multiples operaciones

```
DT[,m:= {tmp <- (x+z); log2(tmp+5)}]
```

operaciones similares a plyr

```
DT[,a:=x>0]
DT[,b:= mean(x+w),by=a]
```

Variables especiales

`.N` Un número entero, de longitud 1, que contiene el número de elementos de un nivel de factor

```
set.seed(123);
DT <- data.table(x=sample(letters[1:3], 1E5, TRUE))
DT[, .N, by=x]
```

```
##    x     N
## 1: c 33294
## 2: b 33305
## 3: a 33401
```

llaves

```
DT <- data.table(x=rep(c("a","b","c"),each=100), y=rnorm(300))
setkey(DT, x)
DT['a']
```

```
##       x           y
##   1: a  0.88631257
##   2: a  2.82858132
##   3: a  2.03145429
##   4: a  1.90675413
##   5: a  0.21490826
##   6: a -0.86273413
##   7: a -2.20493863
##   8: a  0.24105923
##   9: a  1.83832419
##  10: a  0.79205468
##  11: a  0.65053469
##  12: a -1.53912061
##  13: a -0.60830053
##  14: a  0.38195644
##  15: a -1.07500044
##  16: a  0.21994264
##  17: a -0.78288781
##  18: a -1.11003346
##  19: a -1.65871456
##  20: a -0.50147343
##  21: a  1.91636375
##  22: a  1.41236645
##  23: a  0.92260986
##  24: a  1.01106201
##  25: a  0.57213026
##  26: a -0.62843126
##  27: a -0.36316140
##  28: a -1.05858811
##  29: a -0.42935803
##  30: a  0.86941467
##  31: a -0.54001647
##  32: a -1.14647747
##  33: a -0.17151840
```

```
##  34: a -0.56368340
##  35: a -0.42994346
##  36: a -1.23723779
##  37: a  0.15901329
##  38: a -1.16711067
##  39: a -0.08111944
##  40: a -0.51667953
##  41: a  0.99540703
##  42: a  0.79752142
##  43: a  0.53895224
##  44: a -1.40405605
##  45: a  0.40144065
##  46: a -0.52432237
##  47: a -0.83952146
##  48: a  0.47556591
##  49: a -0.01194696
##  50: a  0.10319780
##  51: a -0.38575415
##  52: a  1.11726438
##  53: a -0.49961390
##  54: a -0.44735091
##  55: a -0.23784512
##  56: a -0.86939374
##  57: a  1.14887678
##  58: a  0.53864996
##  59: a -0.10680992
##  60: a  0.60053649
##  61: a -1.47499445
##  62: a  0.98126964
##  63: a -0.61118738
##  64: a  0.08938648
##  65: a -0.01327227
##  66: a -0.97219341
##  67: a -0.57946225
##  68: a  0.14963144
##  69: a  0.47640689
##  70: a  0.44729682
##  71: a -0.19180956
##  72: a  0.51712710
##  73: a  0.40338273
##  74: a  1.78411385
##  75: a  0.27775645
##  76: a  0.77394978
##  77: a -2.08081928
##  78: a -0.35920889
##  79: a -0.45932217
##  80: a  0.20181947
##  81: a  0.62401138
##  82: a -0.25722981
##  83: a  0.94414021
##  84: a  0.25074808
##  85: a -0.72784257
##  86: a  0.36881323
##  87: a  0.44415068
```

```
##  88: a -1.00535422
##  89: a -0.33152471
##  90: a -0.37039325
##  91: a -0.79701529
##  92: a  0.28148559
##  93: a  0.33307250
##  94: a  0.52690325
##  95: a -0.78168949
##  96: a -0.02793948
##  97: a -1.74492339
##  98: a  0.65284209
##  99: a -0.93830821
## 100: a  0.62753159
##      x           y
```

uniones

```
DT1 <- data.table(x=c('a', 'a', 'b', 'dt1'), y=1:4)
DT2 <- data.table(x=c('a', 'b', 'dt2'), z=5:7)
setkey(DT1, x); setkey(DT2, x)
merge(DT1, DT2)
```

```
##    x y z
## 1: a 1 5
## 2: a 2 5
## 3: b 3 6
```

una caracteristica es la lectura rapida

```
big_df <- data.frame(x=rnorm(1E6), y=rnorm(1E6))
file <- tempfile()
write.table(big_df, file=file, row.names=FALSE, col.names=TRUE, sep="\t", quote=FALSE)
system.time(fread(file))
```

```
##    user  system elapsed
##    0.22    0.03    0.11
```

```
system.time(read.table(file, header=TRUE, sep="\t"))
```

```
##    user  system elapsed
##    8.72    0.34    9.39
```

Resumen y lectura adicional

- La última versión de desarrollo contiene nuevas funciones como `melt` y`dcast` para data.tables
- https://r-forge.r-project.org/scm/viewvc.php/pkg/NEWS?view=markup&root=datatable
- Aquí hay una lista de diferencias entre data.table y data.frame
- http://stackoverflow.com/questions/13618488/what-you-can-do-with-data-frame-that-you-cant-in-data-table
- Notas basadas en las notas de Raphael Gottardo
- https://github.com/raphg/Biostat-578/blob/master/Advanced_data_manipulation.Rpres, quien los obtuvo de Kevin Ushey.