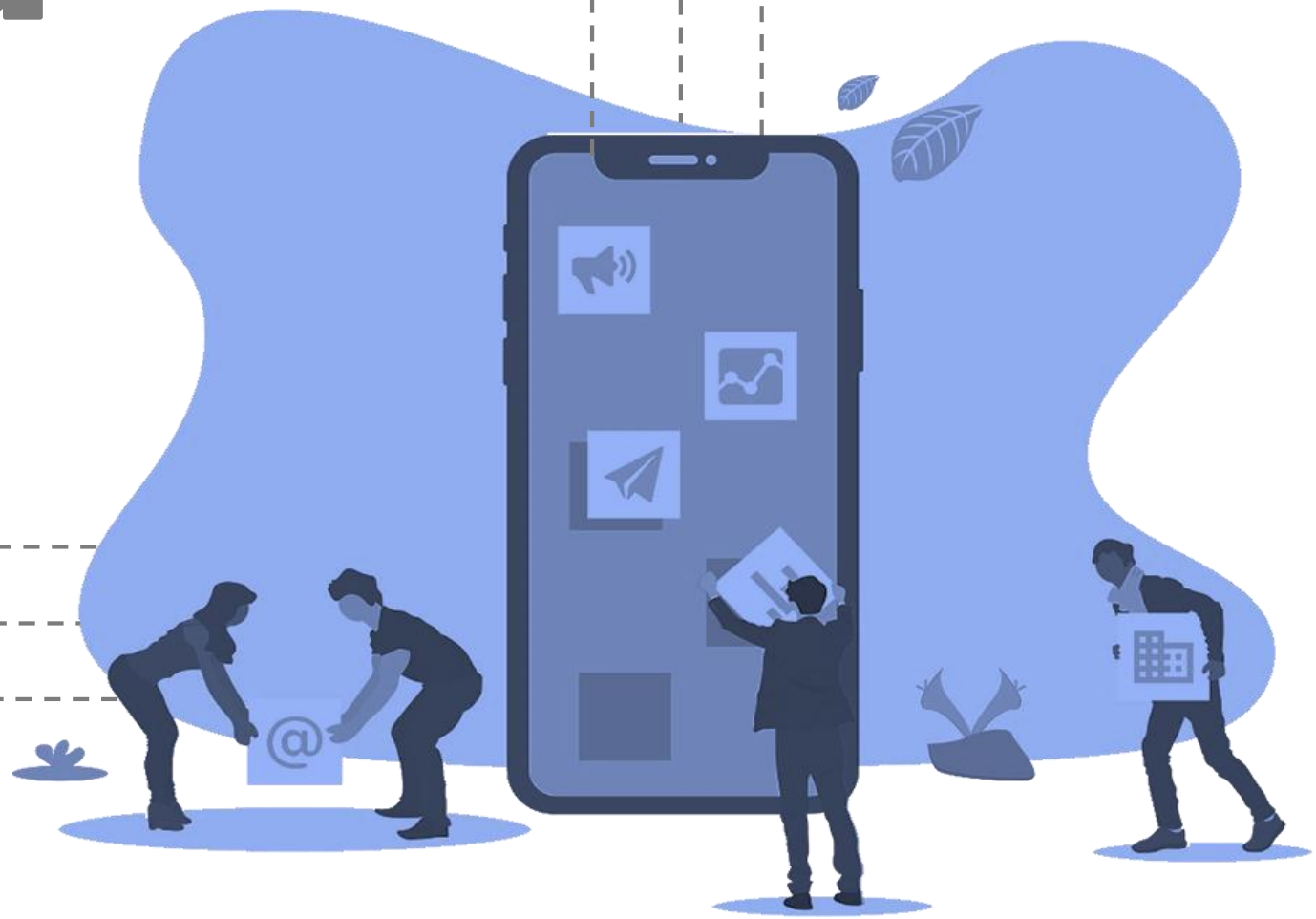
The Data Science Track

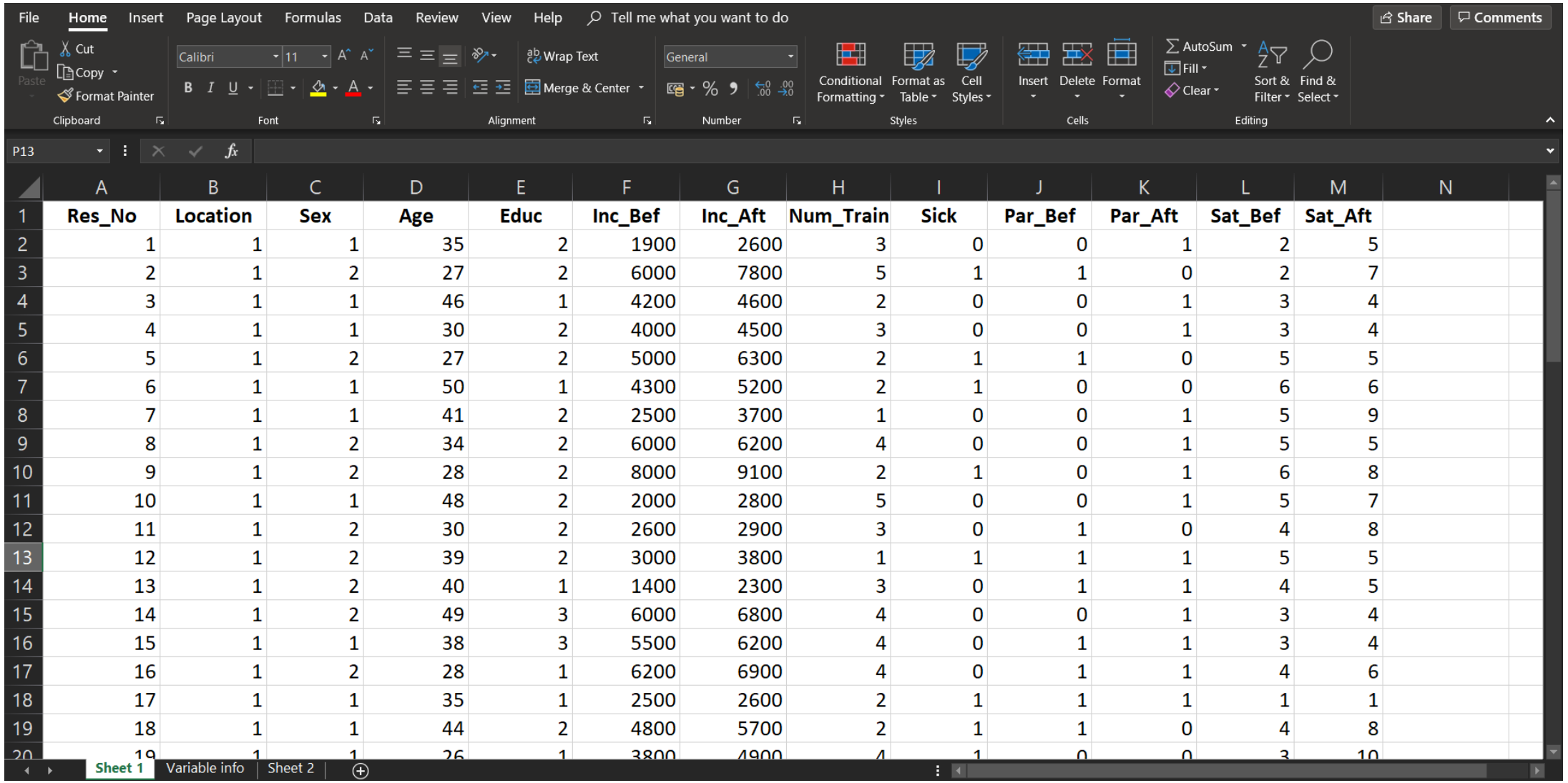


Prepared By: R. Daynolo

INTRODUCTION TO DATA SCIENCE WITH R

18. Raw and Processed Data

WHAT YOU WISH DATA LOOKED LIKE



Res_No	Location	Sex	Age	Educ	Inc_Bef	Inc_Aft	Num_Train	Sick	Par_Bef	Par_Aft	Sat_Bef	Sat_Aft
1	1	1	35	2	1900	2600	3	0	0	1	2	5
2	1	2	27	2	6000	7800	5	1	1	0	2	7
3	1	1	46	1	4200	4600	2	0	0	1	3	4
4	1	1	30	2	4000	4500	3	0	0	1	3	4
5	1	2	27	2	5000	6300	2	1	1	0	5	5
6	1	1	50	1	4300	5200	2	1	0	0	6	6
7	1	1	41	2	2500	3700	1	0	0	1	5	9
8	1	2	34	2	6000	6200	4	0	0	1	5	5
9	1	2	28	2	8000	9100	2	1	0	1	6	8
10	1	1	48	2	2000	2800	5	0	0	1	5	7
11	1	2	30	2	2600	2900	3	0	1	0	4	8
12	1	2	39	2	3000	3800	1	1	1	1	5	5
13	1	2	40	1	1400	2300	3	0	1	1	4	5
14	1	2	49	3	6000	6800	4	0	0	1	3	4
15	1	1	38	3	5500	6200	4	0	1	1	3	4
16	1	2	28	1	6200	6900	4	0	1	1	4	6
17	1	1	35	1	2500	2600	2	1	1	1	1	1
18	1	1	44	2	4800	5700	2	1	1	0	4	8
19	1	1	26	1	3800	4900	4	1	0	0	3	10

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

WHAT DOES DATA REALLY LOOK LIKE?

```
@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-EAS121:4:100:1783:1611#0/1
GGGTGGGCATTTCCACTCGCAGTATGGGTTGCCGCACGACAGGCAGCGGTCAGCCTGCGCTTTGGCCTGGCCTTCGGAAA
+HWI-EAS121:4:100:1783:1611#0/1
a``^\__`_```^a``a`^a_^__]a_]\]`a______`_^^`]X]_]XTV_\]]NX_XVX]]_TTTTG[VTHPN]VFDZ
@HWI-EAS121:4:100:1783:322#0/1
CGTTTATGTTTTTGAATATGTCTTATCTTAACGGTTATATTTTAGATGTTGGTCTTATTCTAACGGTCATATATTTTCTA
+HWI-EAS121:4:100:1783:322#0/1
abaa`^aaaaabbbaabaabbbbbb`bbbb_bbbbbbbb`bbbaV^_a``a``]``aT]a__V\]]_]^a`]a_abbaV__
@HWI-EAS121:4:100:1783:1394#0/1
GGGTCTTTATTGGTCTGGTGATCCCCCATATTCTCCGGTTGTGTGGTTTAACCGATCATCGCGCATTACTTCCCGGCTGC
+HWI-EAS121:4:100:1783:1394#0/1
```[aa\b^^[]aabbb][`a_abbb`a``bbbbbabaabaaaab_VZa_^___bab_X`[a\HV_[_]_[^_X\T_VQQ
@HWI-EAS121:4:100:1783:207#0/1
CCCTGGGAGATCGGAAGAGCGGTTCAGCAGGAATGCCGAGACCGATCTCGTATGCCGTCTTCTGCTTGAAAAAAAAAACA
+HWI-EAS121:4:100:1783:207#0/1
abba`Xa\^\\`aa]ba__bba[a_O_a`aa`aa`a]^V]X_a^YS\R_\H_[]\ZTDUZZUSOPX]]POP\GS\WSHHD
@HWI-EAS121:4:100:1783:455#0/1
GGGTAATTCAGGGACAATGTAATGGCTGCACAAAAAAATACATCTTTCATGTTCCATTGCACCATTGACAAATACATATT
+HWI-EAS121:4:100:1783:455#0/1
```

http://brianknaus.com/software/srtoolbox/s_4_1_sequence80.txt


```

WHAT DOES DATA REALLY LOOK LIKE?

Developer Use cases Products Docs More Apply Apps

include_entities	optional	The `entities` node will not be included when set to `false`.	`false`

Example Requests

```
$ curl --request GET
 --url 'https://api.twitter.com/1.1/search/tweets.json?q=nasa&result_type=popular'
 --header 'authorization: OAuth oauth_consumer_key="consumer-key-for-app",
 oauth_nonce="generated-nonce", oauth_signature="generated-signature",
 oauth_signature_method="HMAC-SHA1", oauth_timestamp="generated-timestamp",
 oauth_token="access-token-for-authed-user", oauth_version="1.0"'
```

```
$ twurl /1.1/search/tweets.json?q=nasa&result_type=popular
```

Example Response


```
{
    "statuses": [
        {
            "created_at": "Sun Feb 25 18:11:01 +0000 2018",
            "id": 967824267948773377,
            "id_str": "967824267948773377",
            "text": "From pilot to astronaut, Robert H. Lawrence was the first African-American to be selected as an
astronaut by any na… https://t.co/FjPEWnh804",
            "truncated": true,
            "entities": {
                "hashtags": [],
                "symbols": [],
                "user_mentions": [],
                "urls": [
                    {
```


WHAT DOES DATA REALLY LOOK LIKE?

---------- ALLERGIES ----------

ast Updated: 01 Dec 2011 @ 0851

llergy Name: TRIMETHOPRIM
ocation: DAYT29
ate Entered: 09 Mar 2011
eaction:
llergy Type: DRUG
A Drug Class: ANTI-INFECTIVES,OTHER
bserved/Historical: HISTORICAL
omments: The reaction to this allergy was MILD (NO SQUELAE)

llergy Name: TRAMADOL
ocation: DAYT29
ate Entered: 09 Mar 2011
eaction: URINARY RETENTION
llergy Type: DRUG
A Drug Class: NON-OPIOID ANALGESICS
bserved/Historical: HISTORICAL
omments: gradually worsening difficulty emptying bladder

---------- MEDICATION HISTORY ----------

Last Updated: 11 Apr 2011 @ 1737

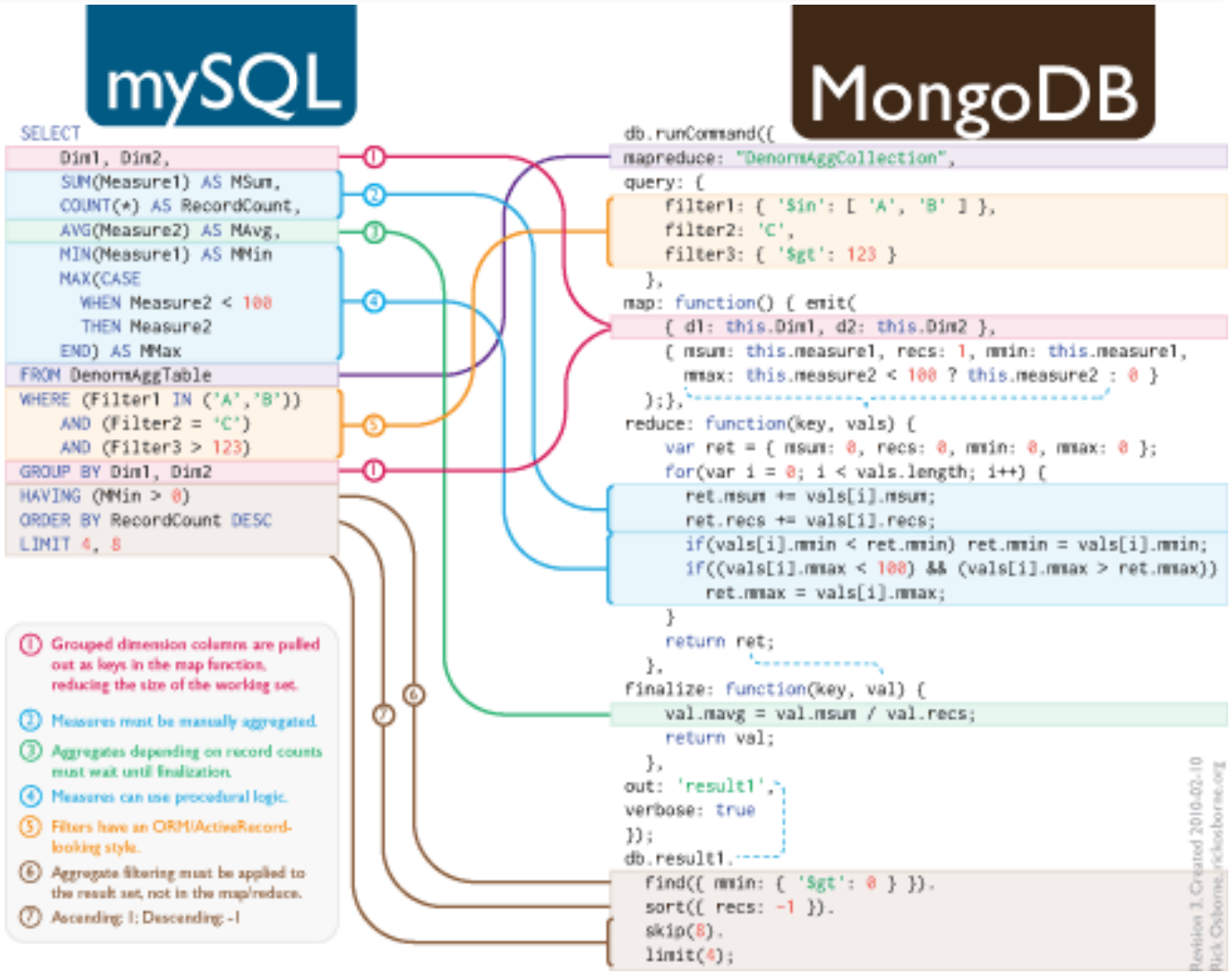
Medication: AMLODIPINE BESYLATE 10MG TAB
Instructions: TAKE ONE TABLET BY MOUTH TAKE ONE-HALF TABLET FOR
GRAPEFRUIT JUICE--
Status: Active
Refills Remaining: 3
Last Filled On: 20 Aug 2010
Initially Ordered On: 13 Aug 2010
Quantity: 45
Days Supply: 90
Pharmacy: DAYTON
Prescription Number: 2718953

Medication: IBUPROFEN 600MG TAB
Instructions: TAKE ONE TABLET BY MOUTH FOUR TIMES A DAY WITH FOO
Status: Active
Refills Remaining: 3
Last Filled On: 20 Aug 2010
Initially Ordered On: 01 Jul 2010

http://healthdesignchallenge.com/

WHERE IS DATA?

Infographic visualizing the relationship between SQL and MapReduce on MongoDB



https://rickosborne.org/blog/2010/02/infographic-migrating-from-sql-to-mapreduce-with-mongodb/

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

WHERE IS DATA?

Developer Use cases Products Docs More Apply Apps

include_entities	optional	The `entities` node will not be included when set to `false`.	`false`

Example Requests

```
$ curl --request GET
 --url 'https://api.twitter.com/1.1/search/tweets.json?q=nasa&result_type=popular'
 --header 'authorization: OAuth oauth_consumer_key="consumer-key-for-app",
 oauth_nonce="generated-nonce", oauth_signature="generated-signature",
 oauth_signature_method="HMAC-SHA1", oauth_timestamp="generated-timestamp",
 oauth_token="access-token-for-authed-user", oauth_version="1.0"'
```

```
$ twurl /1.1/search/tweets.json?q=nasa&result_type=popular
```

Example Response


```
{
    "statuses": [
        {
            "created_at": "Sun Feb 25 18:11:01 +0000 2018",
            "id": 967824267948773377,
            "id_str": "967824267948773377",
            "text": "From pilot to astronaut, Robert H. Lawrence was the first African-American to be selected as an
astronaut by any na… https://t.co/FjPEWnh804",
            "truncated": true,
            "entities": {
                "hashtags": [],
                "symbols": [],
                "user_mentions": [],
                "urls": [
                    {
```


WHERE IS DATA?

https://data.gov.ph/

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

THE GOAL


Raw data
Processing script
Tidy data
Data analysis
Data communication


ANALYSIS
STRUCTURE
ALGORITHM
001011
101110
PROCESS
</>
PROGRAMMING
SOLVING
KNOWLEDGE

DEFINITION OF DATA

"Data are values of qualitative or quantitative variables, belonging to a set of items."

Set of items: Population; the set of objects you are interested in

Variables: A measurement or characteristic of an item

- Qualitative: Country of origin, sex, treatment
- Quantitative: Height, weight, blood pressure

RAW VERSUS PROCESSED DATA

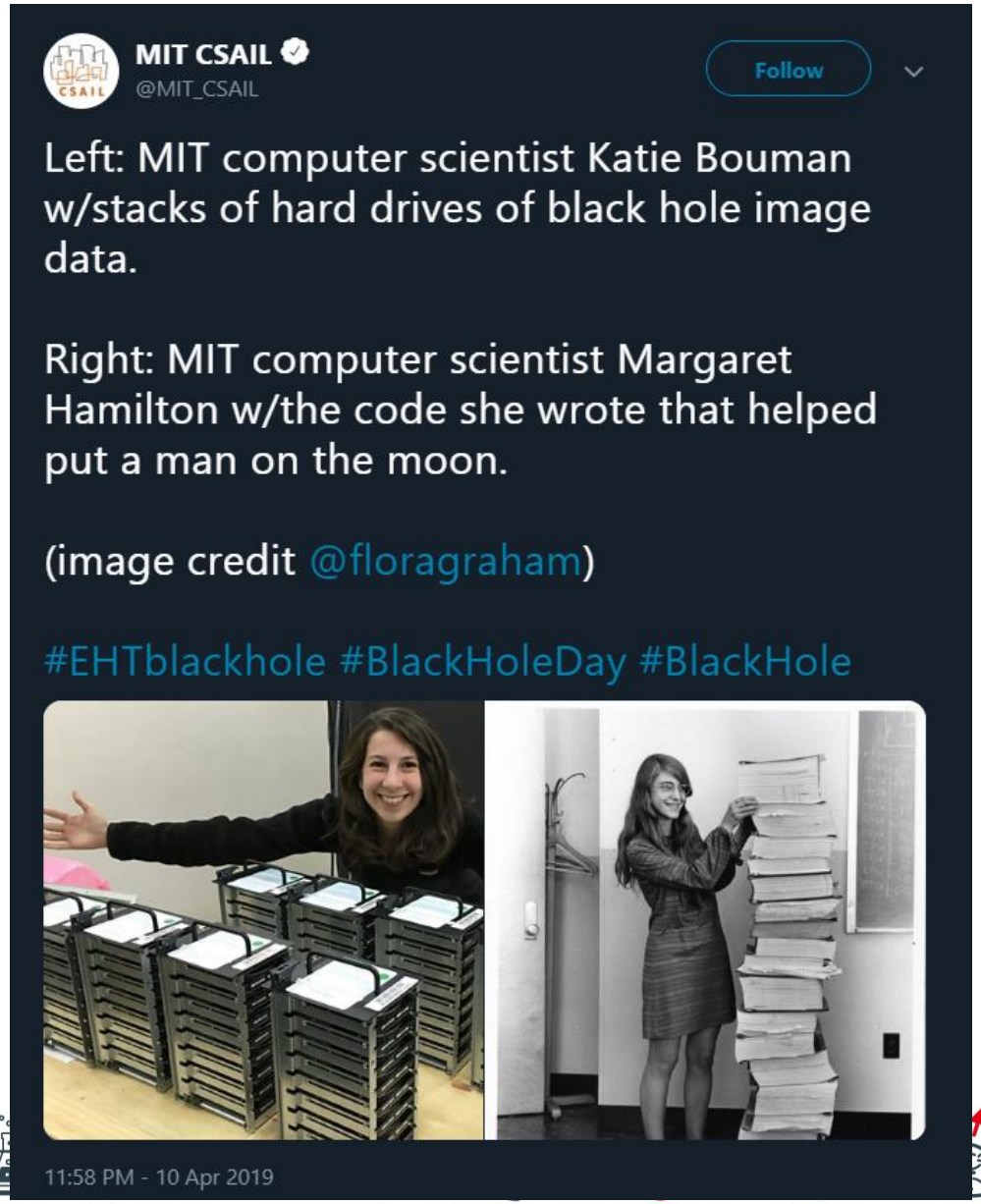
Raw data

- The original source of the data
- Often hard to use for data analyses
- Data analysis includes processing
- Raw data may only need to be processed once

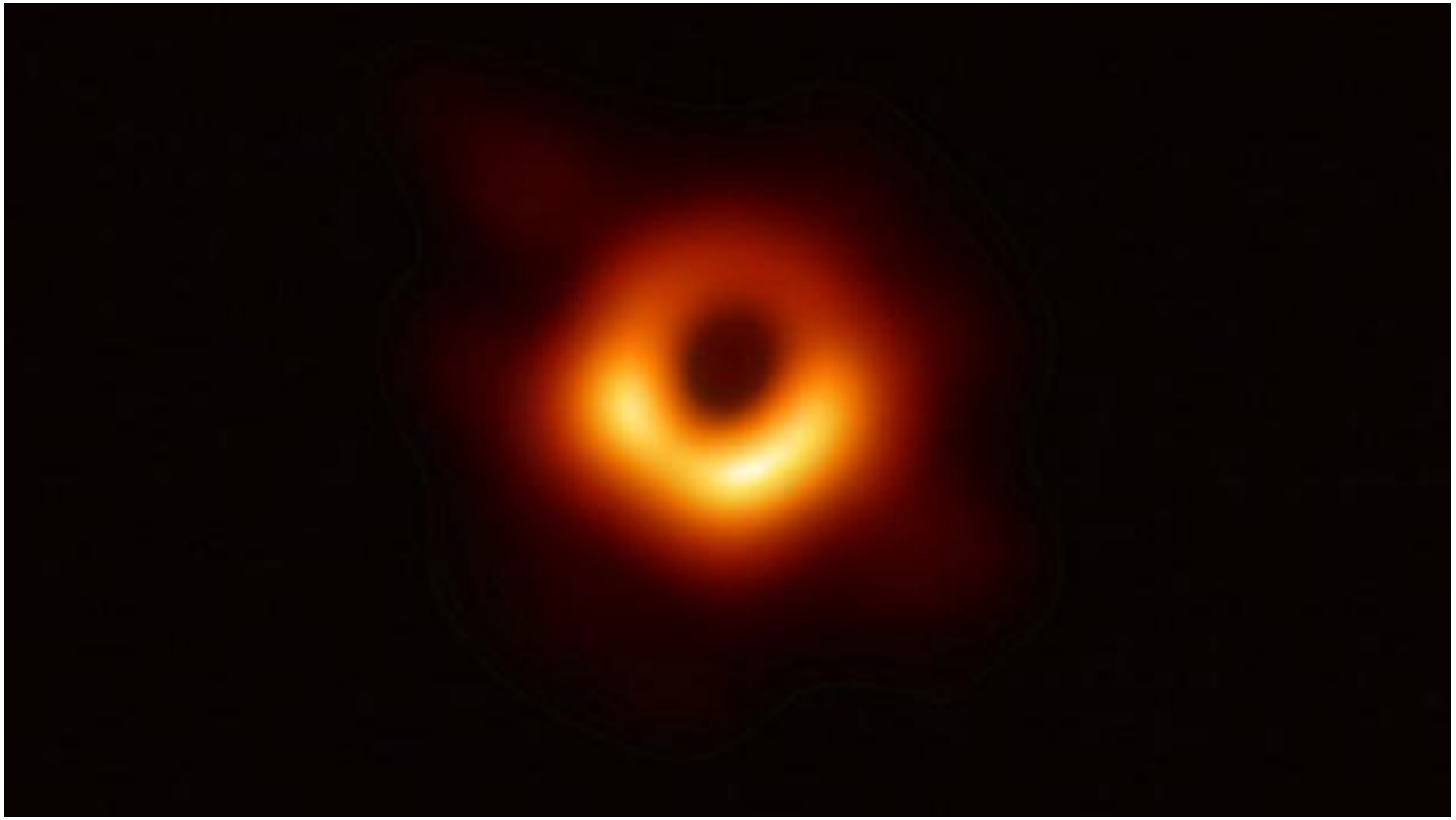
Processed data

- Data that is ready for analysis
- Processing can include merging, subsetting, transforming, etc.
- There may be standards for processing
- All steps should be recorded

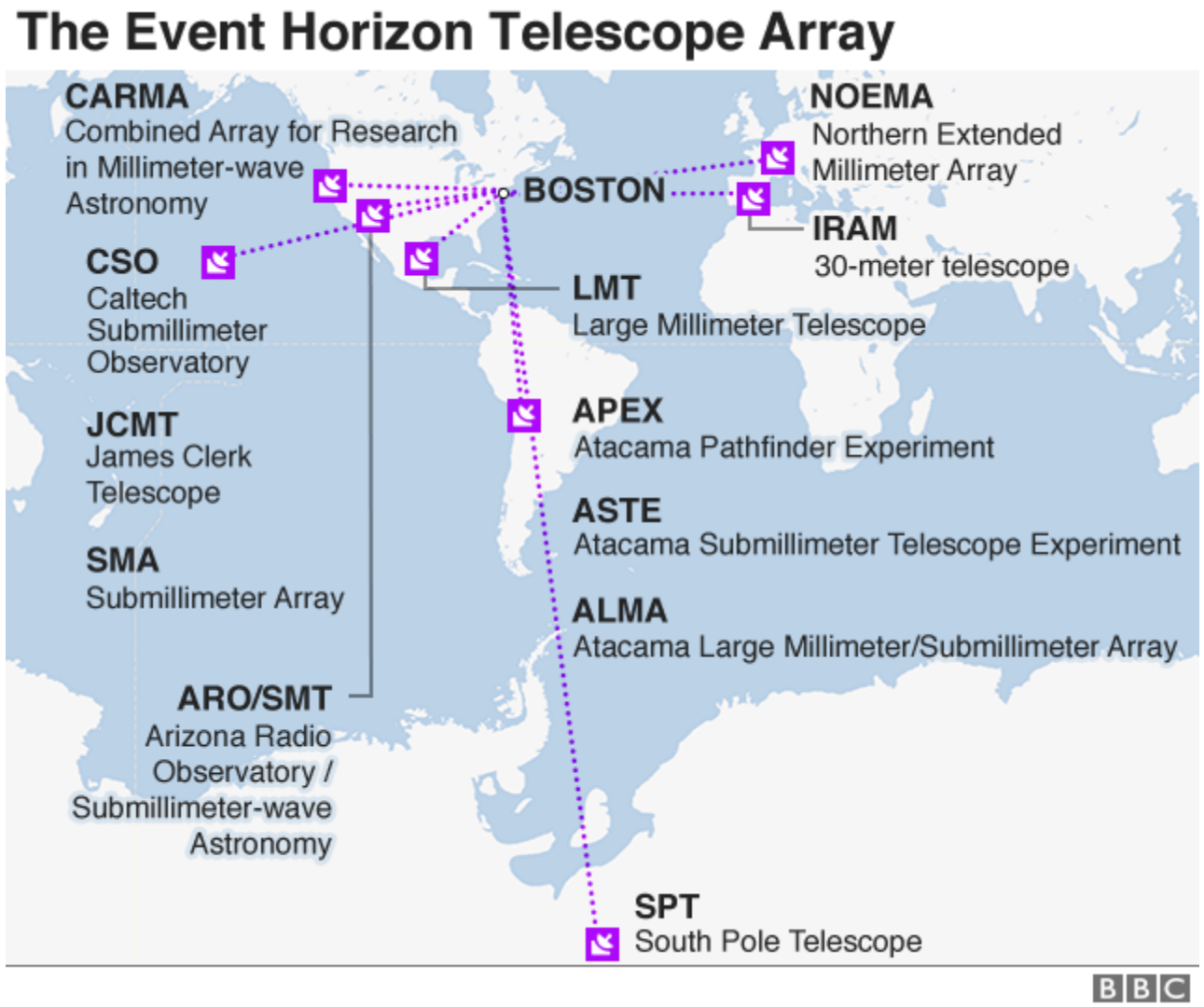
RAW VERSUS PROCESSED DATA



RAW VERSUS PROCESSED DATA



RAW VERSUS PROCESSED DATA



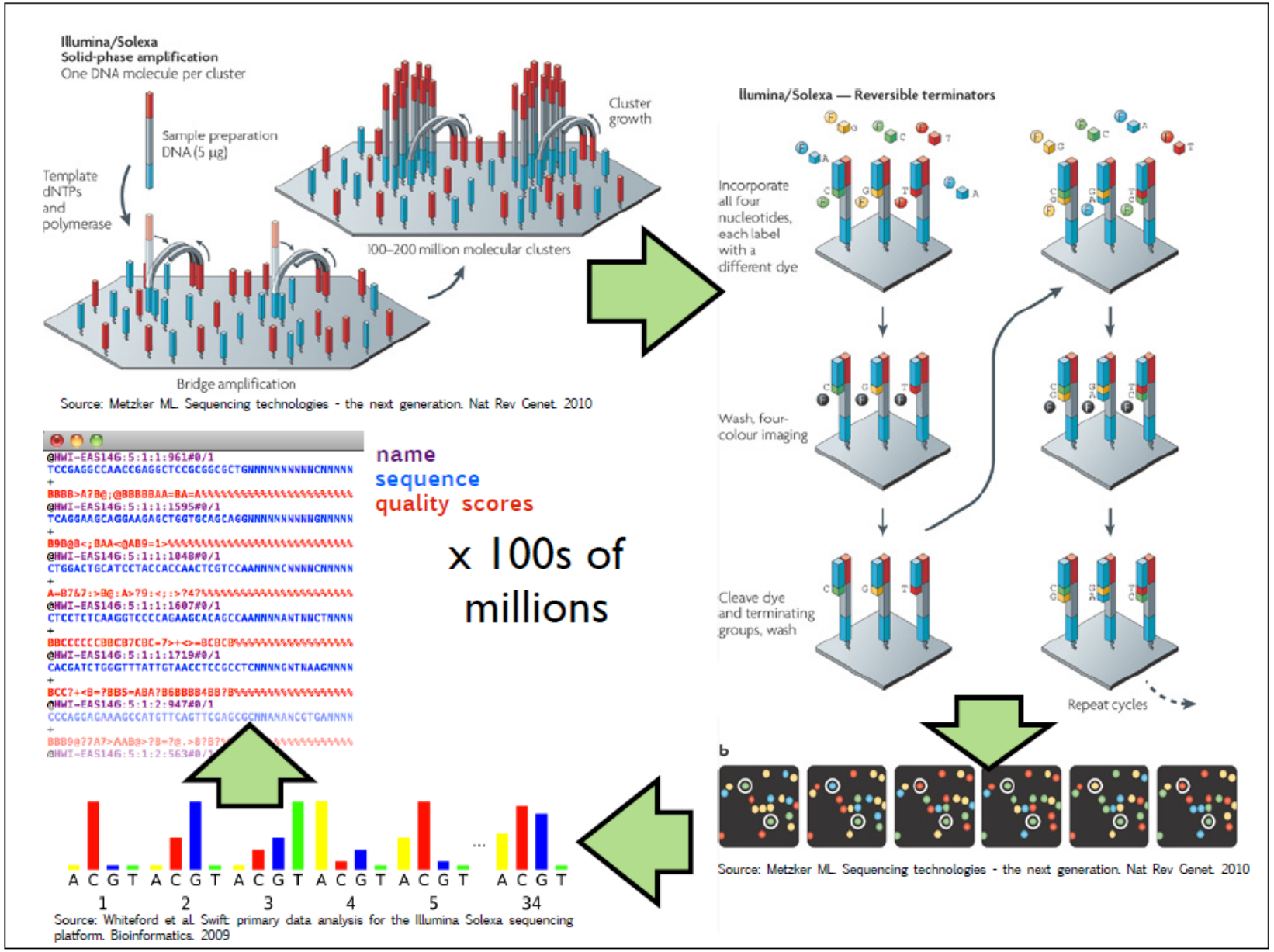
ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

EXAMPLE OF A PROCESSING PIPELINE

Illumina sequencing platform

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

EXAMPLE OF A PROCESSING PIPELINE



THE FOUR THINGS YOU SHOULD HAVE

1. The raw data
2. A tidy data set
3. A code book describing each variable and its values in the tidy data set
4. An explicit and exact recipe you used to go from 1 → 2,3

THE RAW DATA

- The strange binary file your measurement machine spits out
- The unformatted Excel file with 10 worksheets the company you contracted with sent you
- The complicated JSON data you got from scraping the Twitter API
- The hand-entered numbers you collected looking through a microscope

You know the raw data is in the right format if you

1. Ran no software on the data
2. Did not manipulate any of the numbers in the data
3. You did not remove any data from the data set
4. You did not summarize the data in any way

THE TIDY DATA

1. Each variable you measure should be in one column
2. Each different observation of that variable should be in a different row
3. There should be one table for each “kind” of variable
4. If you have multiple tables, they should include a column in the table that allows them to be linked

Some other important tips

- include a row at the top of each file with variable names
- Make variable names human readable: AgeAtDiagnosis instead of AgeDx
- In general data should be saved in one file per table

THE CODE BOOK

1. Information about the variables (including units) in the data set not contained in the tidy data
2. Information about the summary choices you made
3. Information about the experimental study design you used

Some other important tips

- a common format for this document is a Word/Text file.
- there should be a section called “Study design” that has thorough description of how you collected the data
- there must be a section called “Code book” that describes each variable and its units

THE INSTRUCTION LIST

- Ideally a computer script (in R or any language that you want)
- The input for the script is the raw data
- The output is the processed, tidy data
- There are no parameters to the script

In some cases it will not be possible to script every step. In that case you should provide instructions like:

- Step 1 – take the raw file, run version 3.4.1 of summarize software with parameters a=1, b=2, c=3
- Step 2 – run the software separately for each sample
- Step 3 – take column three of `outputfile.out` for each sample and that is the corresponding row in the output data set

INTRODUCTION TO DATA SCIENCE WITH R

19. Downloading and Reading Data

GET/SET YOUR WORKING DIRECTORY

- A component of working with data is knowing your working directory
- The two main commands are `getwd()` and `setwd()`.
- Be aware of relative versus absolute paths
  - Relative – `setwd("./data"), setwd("../")`
  - Absolute – `setwd("/Users/username/data/")`
- Important difference in Windows `setwd("C:\Users\User\Downloads\)`

CHECKING FOR AND CREATING DIRECTORIES

- `file.exists("directoryName")` will check to see if the directory exists
- `dir.create("directoryName")` will create a directory if t doesn't exist
- Here is an example checking for a "data" directory and creating it if it doesn't exist

```
if(!file.exists("data")){
    dir.create("data")
}
```

GETTING DATA FROM THE INTERNET

`download.file()`

- Downloads a file from the internet
- Even if you could do this by hand, helps with reproducibility
- Important parameters are `url`, `destfile`, `method`
- Useful for downloading tab-limited, csv, and other files

https://stat.ethz.ch/R-manual/R-devel/library/utils/html/download.file.html

GETTING DATA FROM THE INTERNET

Home / Datasets / Distribution of EVP Participants per year: 2007-2012

Distribution of EVP Participants per year: 2007-2012

EVP Participants are Filipinos going to the US with J1 Visa or Trainee Visa. This information is designed to process the applicaction and registration of these training applicants. The system will enable the EVP secretariat to tap a particular participant with their specific skills and e xpertise for development project in the country. It can also generate statistics on the deployment of Filipino trainees to the U.S. The system also acts a monitoring system for the approval or non-approval of "No Objection Statement" (NOS) in case the EVP participants request not to return to the Philippines after their contract expires. They need to submit the necessary documentary requirements and obtain from EVP Committee its approval. Application for NOS for a particular EVP participant is inputted using the EVPIS together with the completed list of requirements.

Distribution of EVP Participants per year

Data is from 2007-2012

Commission on Filipinos Overseas

The Commission on Filipinos Overseas (CFO) is an agency of the Philippine Government under the Office of the President tasked to promote and uphold the interests, rights and welfare of overseas Filipinos and strengthen their ties with the Motherland.

License

License Not Specified

Data and Resources

CSV **Distribution of EVP Participants per year: 2007 - 2012**

Preview Download

Field	Value
Publisher	Commission on Filipinos Overseas

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

GETTING DATA FROM THE INTERNET

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

GETTING DATA FROM THE INTERNET

```
> fileUrl <- "https://data.gov.ph/sites/default/files/migrated/EVPJ1registrants20072012.csv"
> download.file(fileUrl, destfile = "C:/Users/User/Downloads/Stat197-4/data.csv")
trying URL 'https://data.gov.ph/sites/default/files/migrated/EVPJ1registrants20072012.csv'
Content type 'text/csv' length 409 bytes
downloaded 409 bytes

> list.files("C:/Users/User/Downloads/Stat197-4/")
 [1] "1. The Data Scientists' Toolbox.pdf"
 [2] "2. Command Line Interface.pdf"
 [3] "3. Creating Repos and Basic Git Commands.pdf"
 [4] "4. R Programming Overview.pdf"
 [5] "5. R Programming - Objects.pdf"
 [6] "6. R Programming - Control Structures.pdf"
 [7] "7. R Programming - Functions.pdf"
 [8] "8. Loop Functions.pdf"
 [9] "data.csv"
[10] "README.md"
> dateDownloaded <- date()
> dateDownloaded
[1] "Thu Apr 11 21:29:12 2019"
```

SOME NOTES ABOUT `download.file()`

- If the url starts with *`http`* you can use `download.file()`
- If the url starts with *`https`* on Windows you may be ok
- If the url starts with *`https`* on Mac you may need to set `method = "curl"`
- If the file is big, this might take a while
- Be sure to record when you downloaded.

READING LOCAL FLAT FILES

```
if(!file.exists("data")){
    dir.create("data")
}

fileUrl <- "https://data.gov.ph/sites/default/files/migrated/EVPJ1registrants20072012.csv"
download.file(fileUrl, destfile = "C:/Users/User/Downloads/Stat197-4/data/data.csv")
dateDownloaded <- date()
```

LOADING FLAT FILES – `read.table()`

- This is the main function for reading data into R
- Flexible and robust but requires more parameters
- Reads the data into RAM – big data can cause problems
- Important parameters `file, header, sep, row.names, nrows`
- Related: `read.csv(), read.csv2`

```
> read.csv
function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",
    fill = TRUE, comment.char = "", ...)
read.table(file = file, header = header, sep = sep, quote = quote,
    dec = dec, fill = fill, comment.char = comment.char, ...)
<bytecode: 0x000000000bee5df8>
<environment: namespace:utils>
> read.csv2
function (file, header = TRUE, sep = ";", quote = "\"", dec = ",",
    fill = TRUE, comment.char = "", ...)
read.table(file = file, header = header, sep = sep, quote = quote,
    dec = dec, fill = fill, comment.char = comment.char, ...)
<bytecode: 0x000000000a21b4b0>
<environment: namespace:utils>
```

LOADING FLAT FILES – read.table()

```
> mydata <- read.table("C:/Users/User/Downloads/Stat197-4/data/data.csv")
> head(mydata)
                                        V1
1 month,2007,2008,2009,2010,2011,2012
2           January,28,47,15,17,72,174
3     February,150,364,283,310,375,841
4        March,362,297,546,477,654,771
5            April,35,54,36,88,251,305
6              May,88,66,67,77,171,244
```

LOADING FLAT FILES – read.table()

```
> mydata <- read.table("C:/Users/User/Downloads/Stat197-4/data/data.csv",
+                      sep = ",", header = TRUE)
> head(mydata)
     month X2007 X2008 X2009 X2010 X2011 X2012
1  January    28    47    15    17    72   174
2 February   150   364   283   310   375   841
3    March   362   297   546   477   654   771
4    April    35    54    36    88   251   305
5      May    88    66    67    77   171   244
6     June    81    69    38    66    84    99
```

LOADING FLAT FILES – read.table()

```
> mydata <- read.csv("C:/Users/User/Downloads/Stat197-4/data/data.csv")
> head(mydata)
      month X2007 X2008 X2009 X2010 X2011 X2012
1   January    28    47    15    17    72   174
2  February   150   364   283   310   375   841
3     March   362   297   546   477   654   771
4     April    35    54    36    88   251   305
5       May    88    66    67    77   171   244
6      June    81    69    38    66    84    99
```

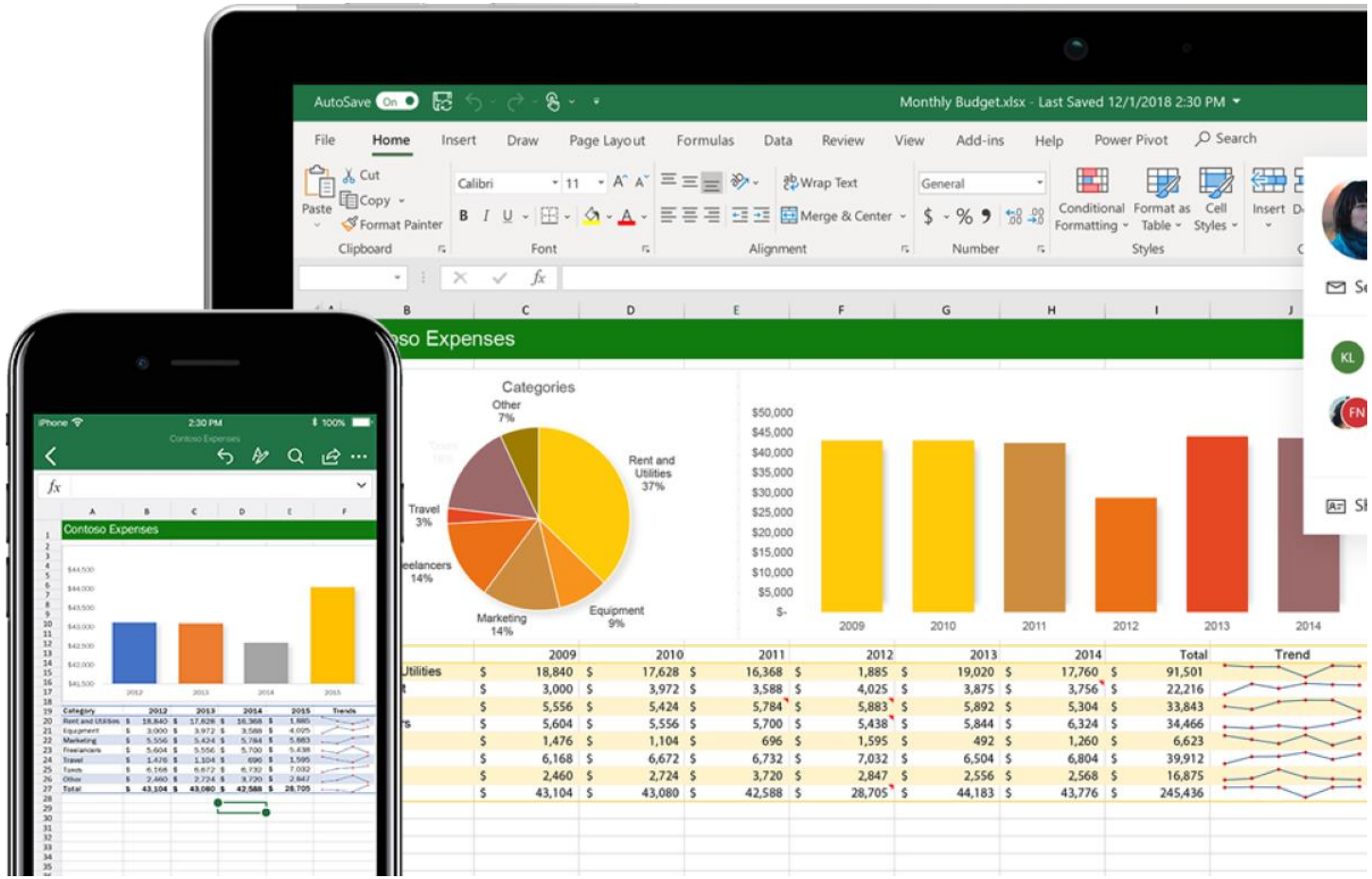
SOME MORE IMPORTANT PARAMETERS

- `quote` – you can tell R whether there are any quoted values; `quote = ""` means no quotes
- `na.strings` – set the character that represents a missing value
- `nrows` – how many rows to read of the file (e.g. `nrows = 10` reads 10 lines)
- `skip` – number of lines to skip before starting to read

Note: the biggest trouble with reading flat files are quotation marks ` and " placed in the data values, setting `quote = ""` often resolves these

READING EXCEL FILES

Still probably the most widely used format for sharing data



ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

read.xlsx(), read.xlsx2() {xlsx package}

The simplified formats of these two functions are:

```
read.xlsx(file, sheetIndex, header=TRUE, colClasses=NA)
read.xlsx2(file, sheetIndex, header=TRUE, colClasses="character")
```

read.xlsx(), read.xlsx2() {xlsx package}

```
library(xlsx)
mydata <- read.xlsx("C:/Users/User/Downloads/Stat197-4/physical.xlsx",
                    sheetIndex = 1, header = TRUE)
head(mydata)
```

	Res_No	Location	Sex	Age	Educ	Inc_Bef	Inc_Aft	Num_Train	Sick	Par_Bef	Par_Aft	Sat_Bef	Sat_Aft
1	1	1	1	35	2	1900	2600	3	0	0	1	2	5
2	2	1	2	27	2	6000	7800	5	1	1	0	2	7
3	3	1	1	46	1	4200	4600	2	0	0	1	3	4
4	4	1	1	30	2	4000	4500	3	0	0	1	3	4
5	5	1	2	27	2	5000	6300	2	1	1	0	5	5
6	6	1	1	50	1	4300	5200	2	1	0	0	6	6

READING SPECIFIC ROWS AND COLUMNS

```
colIndex <- 1:7
rowIndex <- 1:5
mydatasubset <- read.xlsx("C:/Users/User/Downloads/Stat197-4/physical.xlsx",
                          sheetIndex = 1, colIndex=colIndex,
                          rowIndex = rowIndex, header = TRUE)
mydatasubset
```

```
  Res_No Location Sex Age Educ Inc_Bef Inc_Aft
1      1        1   1  35    2    1900    2600
2      2        1   2  27    2    6000    7800
3      3        1   1  46    1    4200    4600
4      4        1   1  30    2    4000    4500
```

FURTHER NOTES

- The `write.xlsx` function will write out an Excel file with similar arguments.
- `read.xlsx2` is much faster than read.xlsx but for reading subsets of rows may be slightly unstable.
- The `XLConnect` package has more options for writing and manipulating Excel files
- The `XLConnect vignette` is a good place to start for that package
- In general, it is advised to store your data in either a database or in a comma separated file (.csv) or tab separated files (.tab or .txt) as they are easier to distribute

READING XML FILES

XML

- Extensible markup language
- Frequently used to store structured data
- Particularly widely used in internet applications
- Extracting XML is the basis for most web scraping
- Components:
    - **Markup** – labels that give the text structure
    - **Content** – the actual text of the document

TAGS, ELEMENTS AND ATTRIBUTES

- **Tags** correspond to general labels
  - Start tags `<section>`
  - End tags `</section>`
  - Empty tags `<line-break />`

- **Elements** are specific examples of tags
  - `<Greeting>` Hello, world `</Greeting>`

- **Attributes** are components of the label
  - `<img src="jeff.jpg" alt="instructor"/>`
  - `<step number="3">` Connect A to B. `</step>`

EXAMPLE XML FILE

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<breakfast_menu>
  ▼<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>
  ▼<food>
     <name>Strawberry Belgian Waffles</name>
     <price>$7.95</price>
    ▼<description>
       Light Belgian waffles covered with strawberries and whipped cream
     </description>
     <calories>900</calories>
   </food>
  ▼<food>
     <name>Berry-Berry Belgian Waffles</name>
     <price>$8.95</price>
    ▼<description>
       Light Belgian waffles covered with an assortment of fresh berries and whipped cream
     </description>
     <calories>900</calories>
   </food>
```

https://www.w3schools.com/xml/simple.xml

READ THE XML FILE INTO R

```
> library(XML)
> library(RCurl)
> fileUrl <- "https://www.w3schools.com/xml/simple.xml"
> xdata <- getURL(fileUrl)
> doc <- xmlParse(xdata)
> rootNode <- xmlRoot(doc)
> xmlName(rootNode)
[1] "breakfast_menu"
> names(rootNode)
   food    food    food    food    food
"food" "food" "food" "food" "food"
```

DIRECTLY ACCESS PARTS OF THE XML DOC

```
> 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>
```

PROGMATICALLY EXTRACT PARTS OF THE FILE

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

```
                                                                                        food
                          "Belgian Waffles$5.95Two of our famous Belgian Waffles with plenty of real maple syrup650"
                                                                                        food
                   "Strawberry Belgian Waffles$7.95Light Belgian waffles covered with strawberries and whipped cream900"
                                                                                        food
"Berry-Berry Belgian Waffles$8.95Light Belgian waffles covered with an assortment of fresh berries and whipped cream900"
                                                                                        food
                                             "French Toast$4.50Thick slices made from our homemade sourdough bread600"
                                                                                        food
                      "Homestyle Breakfast$6.95Two eggs, bacon or sausage, toast, and our ever-popular hash browns950"
```

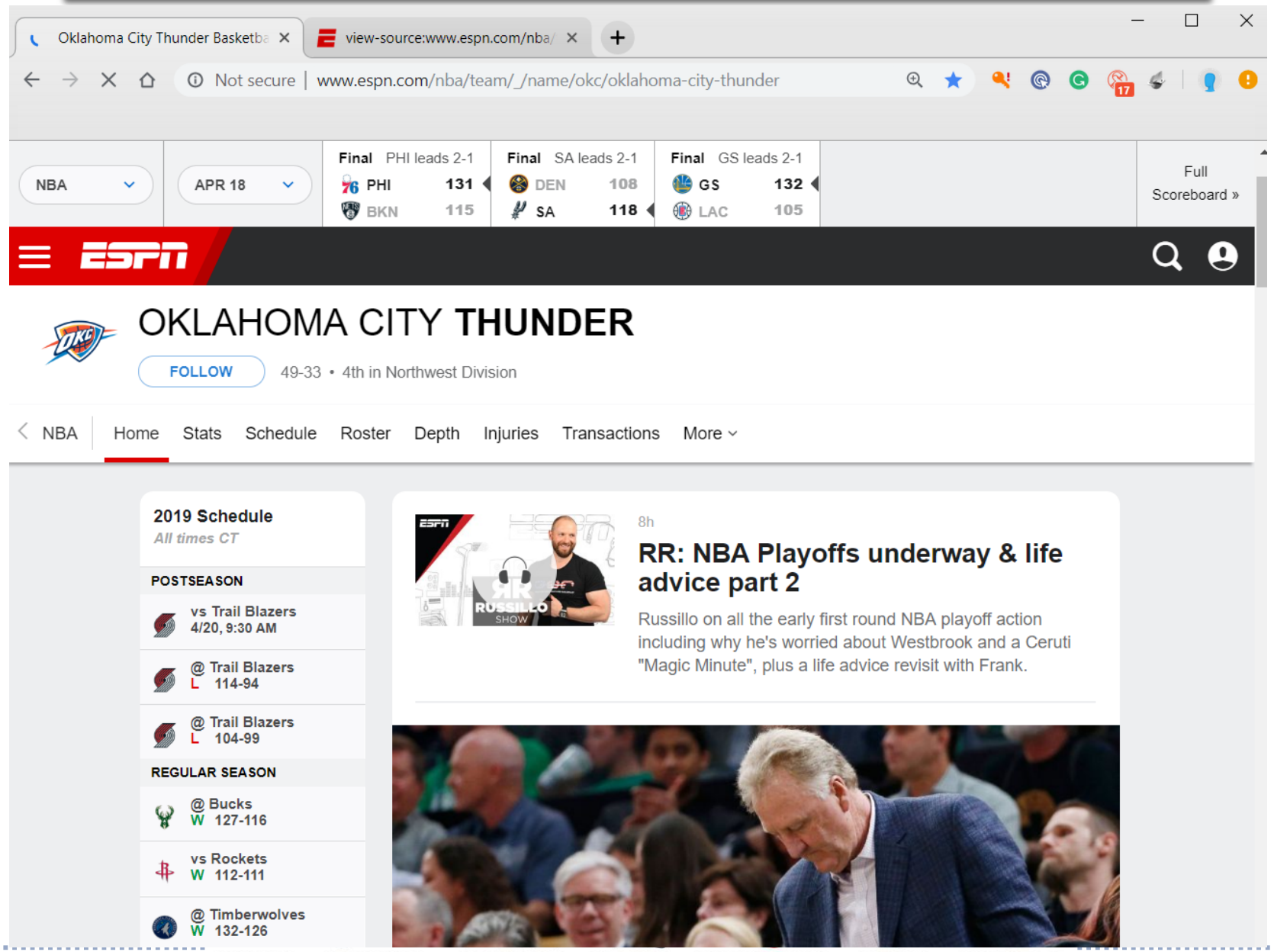
XPath

- `/node` - Top level node
- `//node` - Node at any level
- `node[@attr-name]` - Node with an attribute name
- `node[@attr-name='bob']` - Node with attribute name attr-name='bob'

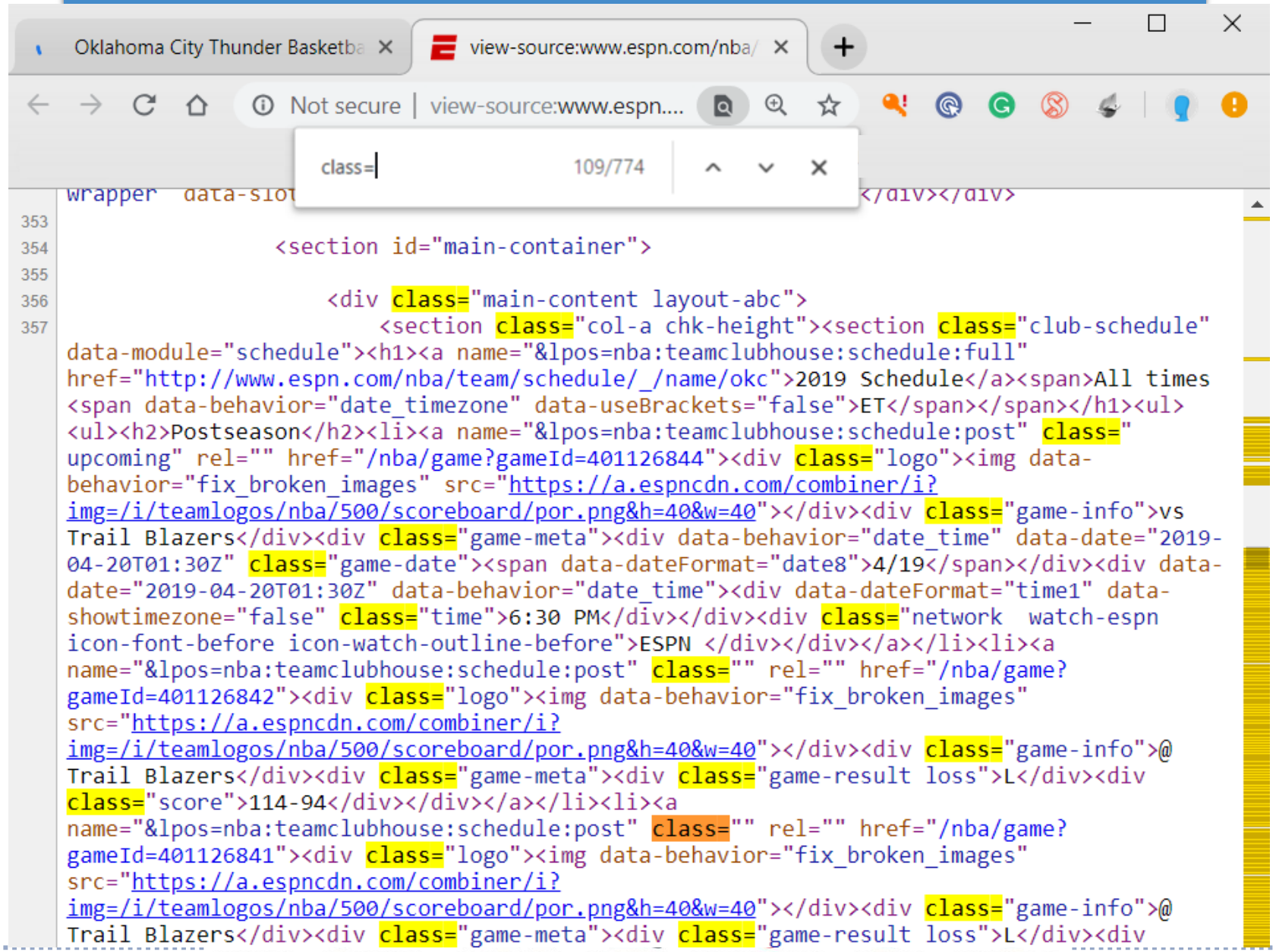
GET ITEMS ON THE MENU AND PRICES

```
> xpathSApply(rootNode, "//name", xmlValue)
[1] "Belgian Waffles"
[2] "Strawberry Belgian Waffles"
[3] "Berry-Berry Belgian Waffles"
[4] "French Toast"
[5] "Homestyle Breakfast"
> xpathSApply(rootNode, "//price", xmlValue)
[1] "$5.95" "$7.95" "$8.95" "$4.50" "$6.95"
```

ANOTHER EXAMPLE



VIEWING THE SOURCE



EXTRACT CONTENT BY ATTRIBUTES

```
fileUrl <- "http://www.espn.com/nba/team/_/name/okc/oklahoma-city-thunder"
ydata <- getURL(fileUrl)
doc <- htmlTreeParse(ydata, useInternalNodes = TRUE)
node <- xmlRoot(doc)
game_meta <- xpathSApply(node, "//div[@class='game-meta']", xmlValue)
game_info <- xpathSApply(node, "//div[@class='game-info']", xmlValue)
score <- xpathSApply(node, "//div[@class='score']", xmlValue)
```

EXTRACT CONTENT BY ATTRIBUTES

```
> game_meta
 [1] "4/196:30 PMESPN "  "L114-94"   "L104-99"
 [4] "W127-116"          "W112-111"  "W132-126"
 [7] "W123-110"          "W119-103"  "L106-103"
[10] "L115-105"          "W107-99"   "L115-103"
[13] "W116-109"          "L123-114"  "L116-107"
[16] "L110-88"           "L108-106"  "W108-96"
[19] "W98-89"            "L118-110"  "W129-121"
[22] "L131-120"          "W99-95"    "L116-102"
[25] "L108-104"
```

EXTRACT CONTENT BY ATTRIBUTES

```
> game_info
 [1] "vs  Trail Blazers" "@  Trail Blazers"  "@  Trail Blazers"
 [4] "@  Bucks"          "vs  Rockets"       "@  Timberwolves"
 [7] "vs  Pistons"       "vs  Lakers"        "vs  Mavericks"
[10] "vs  Nuggets"       "vs  Pacers"        "@  Grizzlies"
[13] "@  Raptors"        "vs  Raptors"       "vs  Heat"
[16] "vs  Warriors"      "@  Pacers"         "vs  Nets"
[19] "@  Jazz"           "@  Clippers"       "@  Trail Blazers"
[22] "@  Timberwolves"   "vs  Grizzlies"     "@  Spurs"
[25] "vs  76ers"
```

EXTRACT CONTENT BY ATTRIBUTES

```
> score
 [1] "114-94"  "104-99"  "127-116" "112-111" "132-126"
 [6] "123-110" "119-103" "106-103" "115-105" "107-99"
[11] "115-103" "116-109" "123-114" "116-107" "110-88"
[16] "108-106" "108-96"  "98-89"   "118-110" "129-121"
[21] "131-120" "99-95"   "116-102" "108-104"
```

NOTE

- Extracting data from XML

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

- XML Tutorial

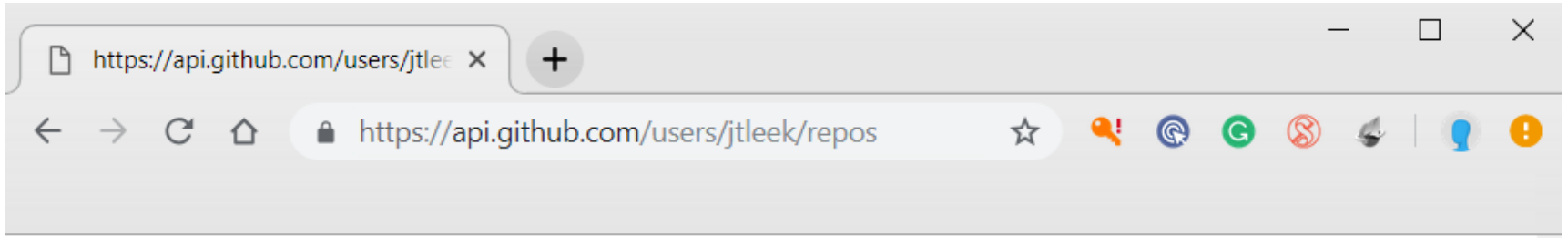
https://www.w3schools.com/xml/

READING JSON

JSON

- Javascript Object Nation
- Lightweight data storage
- Common format for data from application programming interfaces (API)
- Similar structure to XML but different syntax/format
- Data stored as
  - Numbers (double)
  - Strings (double quoted)
  - Boolean (true or false)
  - Array (ordered, comma separated enclosed in square brackets [])
  - Object (unordered, comma separated collection of key: value pairs in curly brackets {})

EXAMPLE JSON FILE




```
[
  {
    "id": 155565363,
    "node_id": "MDEwOlJlcG9zaXRvcnkxNTU1NjUzNjM=",
    "name": "2018",
    "full_name": "jtleek/2018",
    "private": false,
    "owner": {
      "login": "jtleek",
      "id": 1571674,
      "node_id": "MDQ6VXNlcjE1NzE2NzQ=",
      "avatar_url": "https://avatars2.githubusercontent.com/u/1571674?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/jtleek",
      "html_url": "https://github.com/jtleek",
      "followers_url": "https://api.github.com/users/jtleek/followers",
      "following_url": "https://api.github.com/users/jtleek/following{/other_user}",
      "gists_url": "https://api.github.com/users/jtleek/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/jtleek/starred{/owner}{/repo}",
      "subscriptions_url": "https://api.github.com/users/jtleek/subscriptions",
      "organizations_url": "https://api.github.com/users/jtleek/orgs",
      "repos_url": "https://api.github.com/users/jtleek/repos",
      "events_url": "https://api.github.com/users/jtleek/events{/privacy}",
      "received_events_url": "https://api.github.com/users/jtleek/received_events",
      "type": "User",
      "site_admin": false
    },
    "html_url": "https://github.com/jtleek/2018"
```

READING DATA FROM JSON (jsonlite)

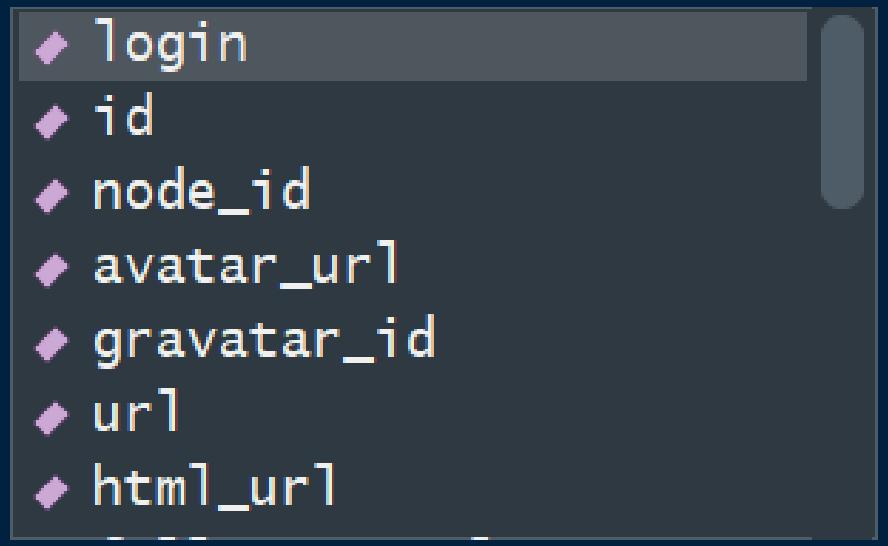
```
library(jsonlite)
jsonData <- fromJSON("https://api.github.com/users/jtleek/repos")
names(jsonData)
```

```
> 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"
```

NESTED OBJECTS IN 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$owner$
```



WRITING DATA FRAMES TO JSON

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

```
    "Sepal.Length": 5,
    "Sepal.Width": 3.4,
    "Petal.Length": 1.5,
    "Petal.Width": 0.2,
    "Species": "setosa"
  },
  {
    "Sepal.Length": 4.4,
    "Sepal.Width": 2.9,
    "Petal.Length": 1.4,
    "Petal.Width": 0.2,
    "Species": "setosa"
  },
  {
```

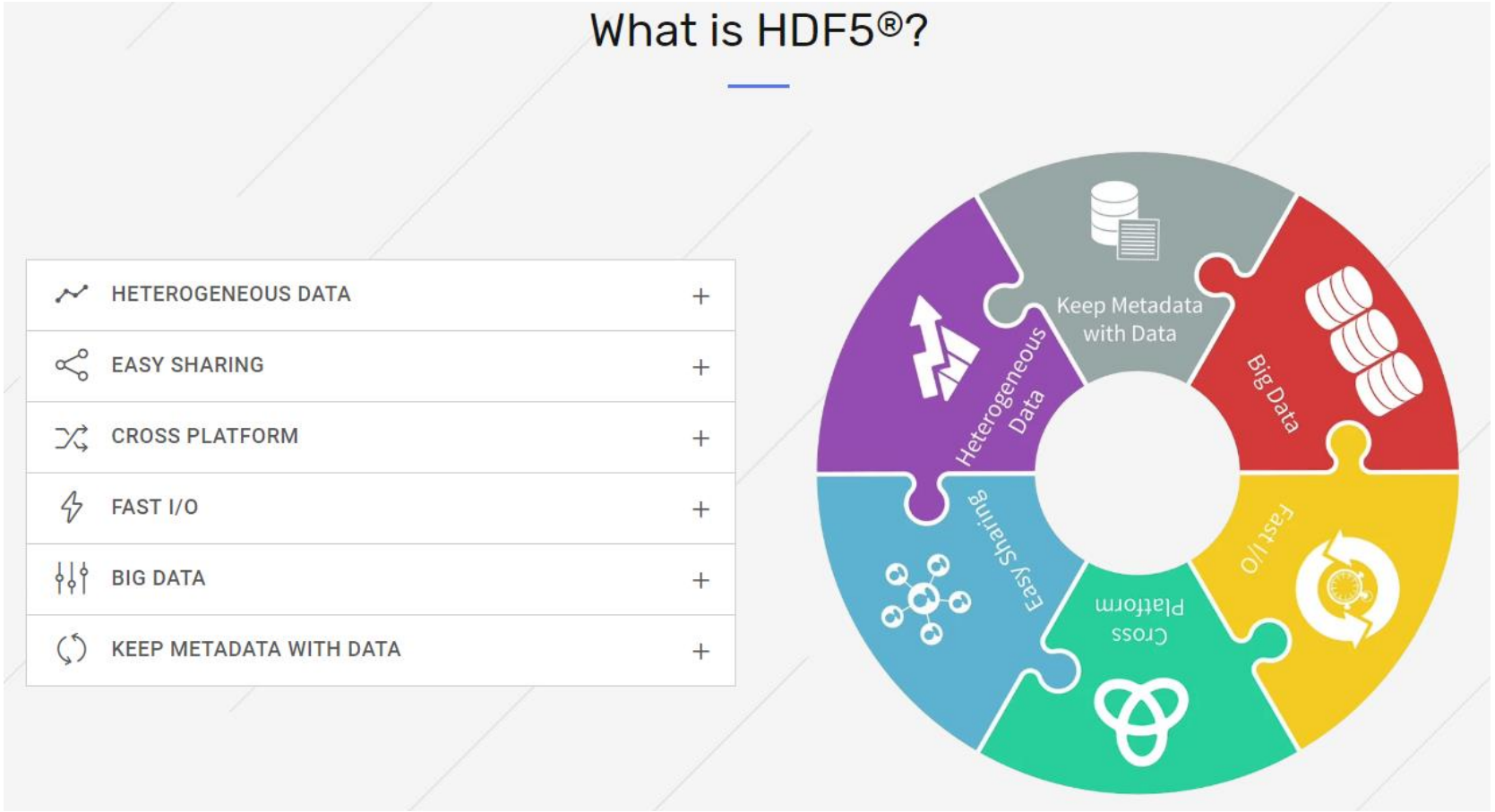
CONVERT BACK TO JSON

```
> myIris <- fromJSON(myjson)
> head(myIris)
  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
```

FURTHER NOTES

- http://json.org/
- A good tutorial on jsonlite:
  https://www.r-bloggers.com/new-package-jsonlite-a-smarter-json-encoderdecoder/
- jsonlite Vignette:
  https://cran.r-project.org/web/packages/jsonlite/vignettes/json-aaquickstart.html

READING HDF5



https://www.hdfgroup.org/

ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

READING HDF5

HDF5

- Hierarchical data format
- Used for storing large data sets
- Supports storing a range of data types
- *groups* containing zero or more data sets and metadata
  - Have a *group header* with group name and list of attributes
  - have a *group symbol table* with a list of objects in group
- *datasets* multidimensional array of data elements with metadata
  - Have a *header* with name, datatype, dataspace, and storage layout
  - Have a *data array* with the data

https://www.hdfgroup.org/

R HDF5 PACKAGE

```
source("http://bioconductor.org/biocLite.R")
biocLite("rhdf5")
```

```
> library(rhdf5)
> created <- h5createFile("example.h5")
> created
[1] TRUE
```

- This will install packages from Bioconductor (https://www.bioconductor.org/), primarily used for genomics but also has good “big data” packages
- Can be used to interface with HDF5 data sets
- rhdf5 tutorial:
  http://www.bioconductor.org/packages/release/bioc/vignettes/rhdf5/inst/doc/rhdf5.pdf

CREATE GROUPS

```
> created <- h5createGroup("example.h5","foo")
> created <- h5createGroup("example.h5","baa")
> created <- h5createGroup("example.h5","foo/foobaa")
> h5ls("example.h5")
   group    name       otype dclass dim
0      /     baa H5I_GROUP
1      /     foo H5I_GROUP
2   /foo foobaa H5I_GROUP
```

WRITE TO GROUPS

```
> A <- matrix(1:10, 5, 2)
> h5write(A, "example.h5", "foo/A")
> B <- array(seq(0.1, 2.0, by = 0.1), dim = c(5, 2, 2))
> attr(B, "scale") <- "liter"
> h5write(B, "example.h5", "foo/foobaa/B")
> h5ls("example.h5")
          group    name       otype  dclass       dim
0             /     baa   H5I_GROUP                  
1             /     foo   H5I_GROUP                  
2          /foo       A H5I_DATASET INTEGER     5 x 2
3          /foo foobaa   H5I_GROUP                  
4 /foo/foobaa       B H5I_DATASET   FLOAT 5 x 2 x 2
```

WRITE A DATA SET

```
df <- data.frame(1L:5L, seq(0, 1, length.out = 5),
        c("ab","cde", "fghi", "a", "s"),
        stringsAsFactors = FALSE)
h5write(df, "example.h5", "df")
h5ls("example.h5")
```

```
          group    name       otype    dclass       dim
0             /     baa   H5I_GROUP                   
1             /      df H5I_DATASET  COMPOUND         5
2             /     foo   H5I_GROUP                   
3          /foo       A H5I_DATASET   INTEGER     5 x 2
4          /foo  foobaa   H5I_GROUP                   
5 /foo/foobaa       B H5I_DATASET     FLOAT 5 x 2 x 2
```

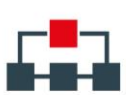
READING DATA

```
readA <- h5read("example.h5", "foo/A")
readB <- h5read("example.h5", "foo/foobaa/B")
readdf <- h5read("example.h5", "df")
```

```
> readA
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

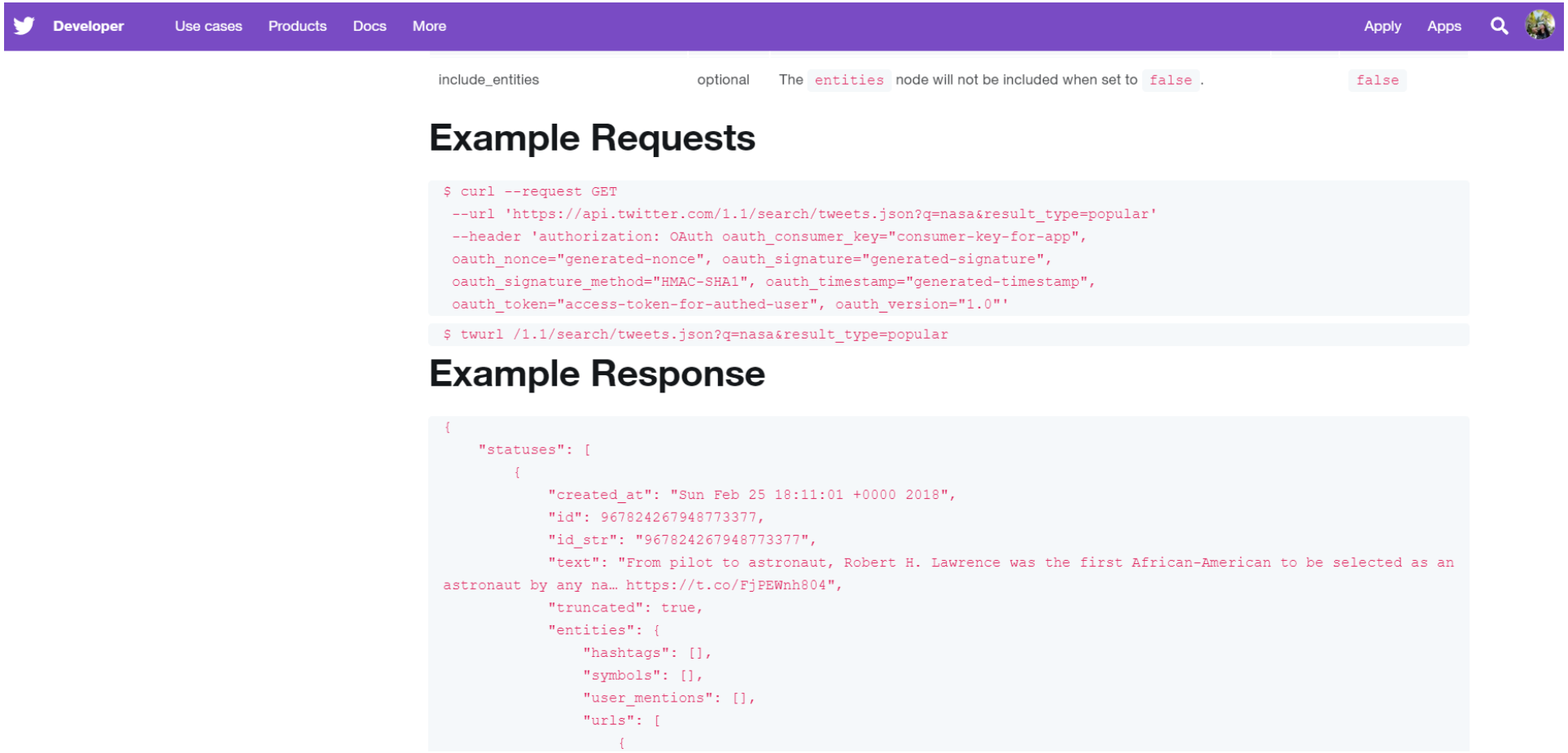
WRITING AND READING CHUNKS

```
> h5write(c(12, 13, 14), "example.h5", "foo/A", index=list(1:3, 1))
> h5read("example.h5", "foo/A")
     [,1] [,2]
[1,]   12    6
[2,]   13    7
[3,]   14    8
[4,]    4    9
[5,]    5   10
```

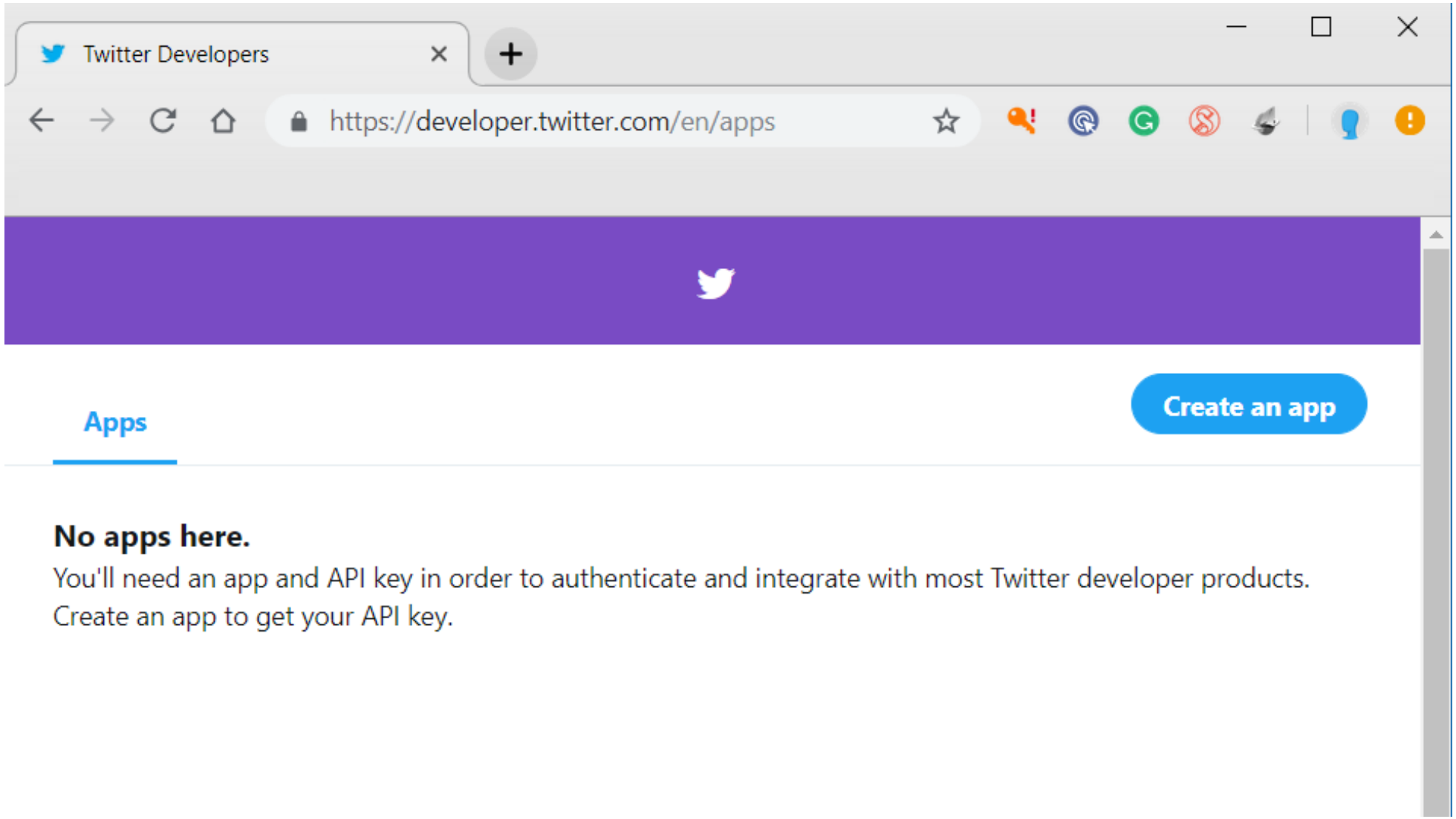


READING DATA FROM APIs

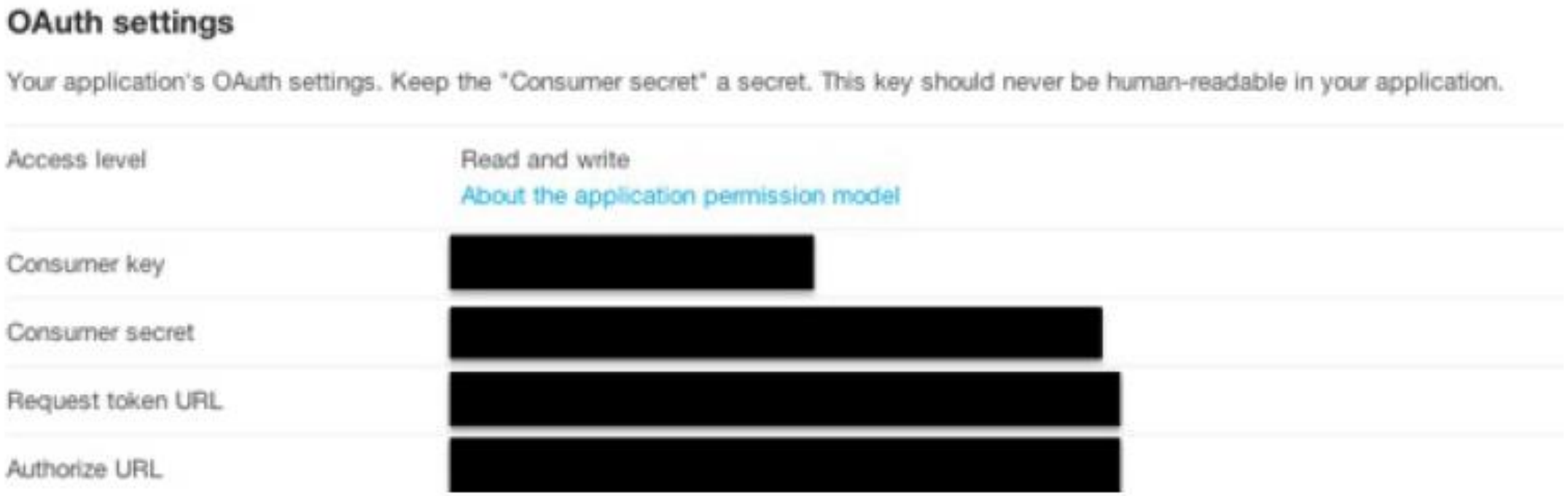
Application Programming Interfaces



CREATING AN APPLICATION



CREATING AN APPLICATION



ANALYSIS STRUCTURE ALGORITHM PROCESS PROGRAMMING SOLVING KNOWLEDGE

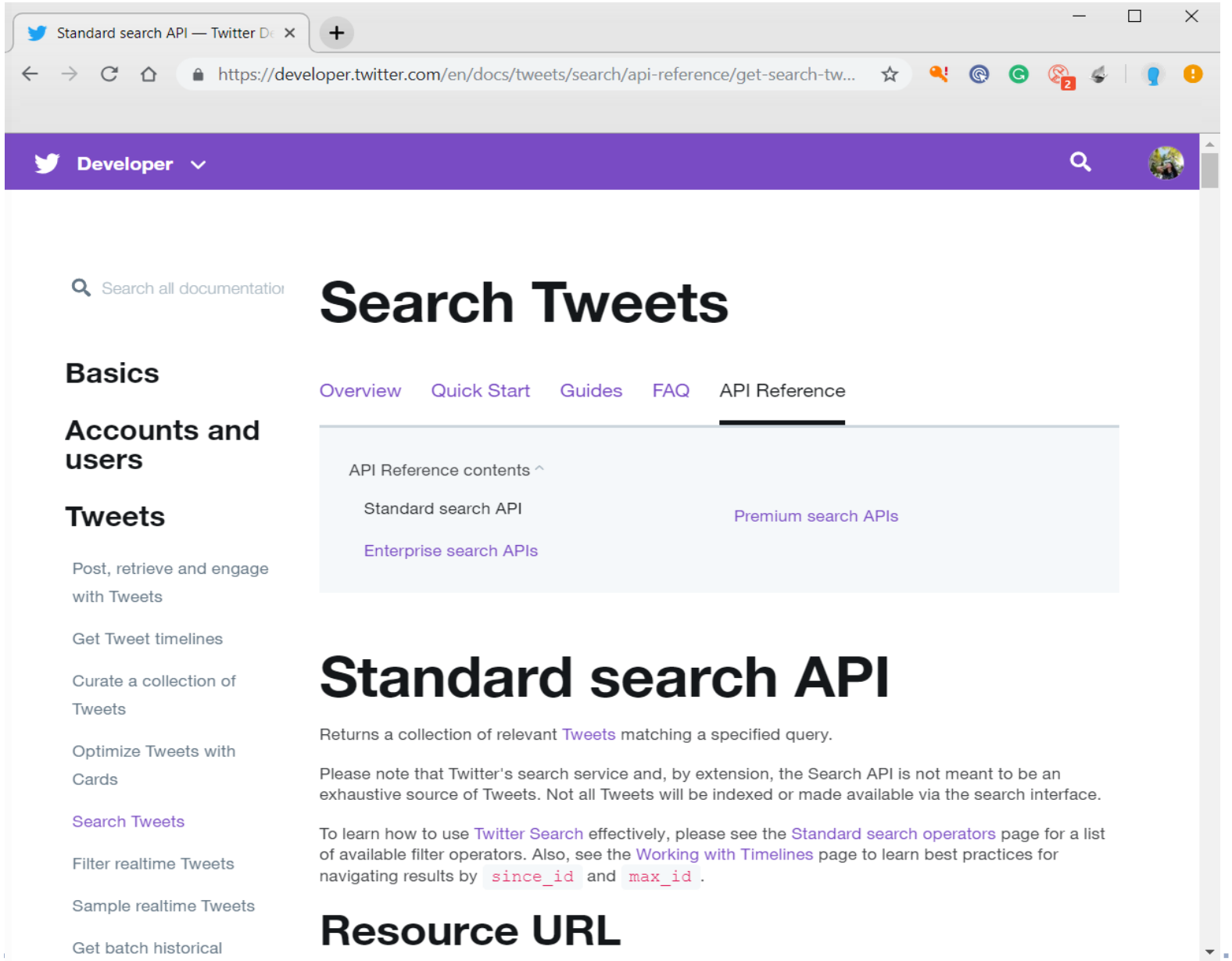
ACCESSING TWITTER FROM R

```
myapp = oauth_app("twitter",
                  key="yourConsumerKeyHere",secret="yourConsumerSecretHere")
sig = sign_oauth1.0(myapp,
                    token = "yourTokenHere",
                     token_secret = "yourTokenSecretHere")
homeTL = GET("https://api.twitter.com/1.1/statuses/home_timeline.json", sig)
```

CONVERTING THE JSON OBJECT

```
json1 = content(homeTL)
json2 = jsonlite::fromJSON(toJSON(json1))
json2[1,1:4]
```

In general, look at the documentation



In general, look at the documentation

- `httr` allows `GET, POST, PUT, DELETE` requests if you are authorized
- You can authenticate with a user name or a password
- Most modern APIs use something like `oauth`
- `httr` works well with Facebook, Google, Twitter, GitHub, etc.

READING FROM OTHER SOURCES

foreign package

- Loads data from Minitab, S, SAS, SPSS, Stata, Systat
- Basic Functions read.foo
  - read.arff (Weka)
  - read.dta (Stata)
  - read.mtp (Minitab)
  - read.octave (Octave)
  - read.spss (SPSS)
  - read.xport (SAS)

- See the help page for more details
  http://cran.r-project.org/web/packages/foreign/foreign.pdf

READING IMAGES

- jpeg
  https://cran.r-project.org/web/packages/jpeg/index.html
- Readbitmap
  https://cran.r-project.org/web/packages/readbitmap/index.html
- png
  https://cran.r-project.org/web/packages/png/index.html
- EBImage (Bioconductor)
  https://www.bioconductor.org/packages/release/bioc/manuals/EBImage/man/EBImage.pdf

READING GIS DATA

- rdgal
  https://cran.r-project.org/web/packages/rgdal/index.html
- rgeos
  https://cran.r-project.org/web/packages/rgeos/index.html
- raster
  https://cran.r-project.org/web/packages/raster/index.html

READING MUSIC DATA

- tuneR
  https://cran.r-project.org/web/packages/tuneR/index.html
- seewave
  http://rug.mnhn.fr/seewave/