Getting and cleaning data

Jeff Leek, PhD

# Motivation and pre-requisites

## About this course

* This course covers the basic ideas behind getting data ready for analysis
* Finding and extracting raw data
* Tidy data principles and how to make data tidy
* Practical implementation through a range of R packages
* What this course depends on
* The Data Scientist's Toolbox
* R Programming
* What would be useful
* Exploratory analysis
* Reporting Data and Reproducible Research

## What you wish data looked like

## What does data really look like?

[<http://brianknaus.com/software/srtoolbox/s_4_1_sequence80.txt>](http://brianknaus.com/software/srtoolbox/s_4_1_sequence80.txt)

## What does data really look like?

[<https://dev.twitter.com/docs/api/1/get/blocks/blocking>](https://dev.twitter.com/docs/api/1/get/blocks/blocking)

## What does data really look like?

[<http://blue-button.github.com/challenge/>](http://blue-button.github.com/challenge/)

## Where is data?

[<http://rickosborne.org/blog/2010/02/infographic-migrating-from-sql-to-mapreduce-with-mongodb/>](http://rickosborne.org/blog/2010/02/infographic-migrating-from-sql-to-mapreduce-with-mongodb/)

## Where is data?

[<https://dev.twitter.com/docs/api/1/get/blocks/blocking>](https://dev.twitter.com/docs/api/1/get/blocks/blocking)

## Where is data?

[<https://data.baltimorecity.gov/>](https://data.baltimorecity.gov/)

## The goal of this course

Raw data -> Processing script -> tidy data -> data analysis -> data communication

# Raw and processed data

## Definition of data

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

[<http://en.wikipedia.org/wiki/Data>](http://en.wikipedia.org/wiki/Data)

## Definition of data

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

[<http://en.wikipedia.org/wiki/Data>](http://en.wikipedia.org/wiki/Data)

**Set of items**: Sometimes called the population; the set of objects you are interested in

## Definition of data

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

[<http://en.wikipedia.org/wiki/Data>](http://en.wikipedia.org/wiki/Data)

**Variables**: A measurement or characteristic of an item.

## Definition of data

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

[<http://en.wikipedia.org/wiki/Data>](http://en.wikipedia.org/wiki/Data)

**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

[<http://en.wikipedia.org/wiki/Raw_data>](http://en.wikipedia.org/wiki/Raw_data)

**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

[<http://en.wikipedia.org/wiki/Computer_data_processing>](http://en.wikipedia.org/wiki/Computer_data_processing)

## An example of a processing pipeline

[<http://www.illumina.com.cn/support/sequencing/sequencing_instruments/hiseq_1000.asp>](http://www.illumina.com.cn/support/sequencing/sequencing_instruments/hiseq_1000.asp)

## An example of a processing pipeline

[<http://www.cbcb.umd.edu/~hcorrada/CMSC858B/lectures/lect22_seqIntro/seqIntro.pdf>](http://www.cbcb.umd.edu/~hcorrada/CMSC858B/lectures/lect22_seqIntro/seqIntro.pdf)

# The components of tidy data

## 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

[<https://github.com/jtleek/datasharing>](https://github.com/jtleek/datasharing)

## 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.

[<https://github.com/jtleek/datasharing>](https://github.com/jtleek/datasharing)

## 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 a thorough description of how you collected the data.
* There must be a section called "Code book" that describes each variable and its units.

[<https://github.com/jtleek/datasharing>](https://github.com/jtleek/datasharing)

## The instruction list

* Ideally a computer script (in R :-), but I suppose Python is ok too...)
* 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:

1. Step 1 - take the raw file, run version 3.1.2 of summarize software with parameters a=1, b=2, c=3
2. Step 2 - run the software separately for each sample
3. Step 3 - take column three of outputfile.out for each sample and that is the corresponding row in the output data set

[<https://github.com/jtleek/datasharing>](https://github.com/jtleek/datasharing)

## Why is the instruction list important?

[<http://www.colbertnation.com/the-colbert-report-videos/425748/april-23-2013/austerity-s-spreadsheet-error>](http://www.colbertnation.com/the-colbert-report-videos/425748/april-23-2013/austerity-s-spreadsheet-error)

# Downloading files

## Get/set your working directory

* A basic 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/jtleek/data/")
* Important difference in Windows setwd("C:\\Users\\Andrew\\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 it 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-delimited, csv, and other files

## Example - Baltimore camera data

[<https://data.baltimorecity.gov/Transportation/Baltimore-Fixed-Speed-Cameras/dz54-2aru>](https://data.baltimorecity.gov/Transportation/Baltimore-Fixed-Speed-Cameras/dz54-2aru)

## Example - Baltimore camera data

[<https://data.baltimorecity.gov/Transportation/Baltimore-Fixed-Speed-Cameras/dz54-2aru>](https://data.baltimorecity.gov/Transportation/Baltimore-Fixed-Speed-Cameras/dz54-2aru)

## Download a file from the web

fileUrl <- "https://data.baltimorecity.gov/api/views/dz54-2aru/rows.csv?accessType=DOWNLOAD"  
download.file(fileUrl,destfile="./data/cameras.csv",method="curl")

## Warning: running command 'curl  
## "https://data.baltimorecity.gov/api/views/dz54-2aru/rows.csv?accessType=DOWNLOAD"  
## -o "./data/cameras.csv"' had status 127

## Warning in download.file(fileUrl, destfile = "./data/cameras.csv", method  
## = "curl"): download had nonzero exit status

list.files("./data")

## [1] "cameras.csv"

dateDownloaded <- date()  
dateDownloaded

## [1] "Tue Jan 06 20:11:46 2015"

## 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

## Example - Baltimore camera data

[<https://data.baltimorecity.gov/Transportation/Baltimore-Fixed-Speed-Cameras/dz54-2aru>](https://data.baltimorecity.gov/Transportation/Baltimore-Fixed-Speed-Cameras/dz54-2aru)

## Download the file to load

if (!file.exists("data")) {  
 dir.create("data")  
}  
fileUrl <- "https://data.baltimorecity.gov/api/views/dz54-2aru/rows.csv?accessType=DOWNLOAD"  
download.file(fileUrl, destfile = "cameras.csv", method = "curl")  
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()*

## Baltimore example

cameraData <- read.table("./data/cameras.csv")

## Error: line 1 did not have 13 elements

head(cameraData)

## Error: object 'cameraData' not found

## Example: Baltimore camera data

cameraData <- read.table("./data/cameras.csv", sep = ",", header = TRUE)  
head(cameraData)

## address direction street crossStreet  
## 1 S CATON AVE & BENSON AVE N/B Caton Ave Benson Ave  
## 2 S CATON AVE & BENSON AVE S/B Caton Ave Benson Ave  
## 3 WILKENS AVE & PINE HEIGHTS AVE E/B Wilkens Ave Pine Heights  
## 4 THE ALAMEDA & E 33RD ST S/B The Alameda 33rd St  
## 5 E 33RD ST & THE ALAMEDA E/B E 33rd The Alameda  
## 6 ERDMAN AVE & N MACON ST E/B Erdman Macon St  
## intersection Location.1  
## 1 Caton Ave & Benson Ave (39.2693779962, -76.6688185297)  
## 2 Caton Ave & Benson Ave (39.2693157898, -76.6689698176)  
## 3 Wilkens Ave & Pine Heights (39.2720252302, -76.676960806)  
## 4 The Alameda & 33rd St (39.3285013141, -76.5953545714)  
## 5 E 33rd & The Alameda (39.3283410623, -76.5953594625)  
## 6 Erdman & Macon St (39.3068045671, -76.5593167803)

## Example: Baltimore camera data

read.csv sets *sep=","* and *header=TRUE*

cameraData <- read.csv("./data/cameras.csv")  
head(cameraData)

## address direction street crossStreet  
## 1 S CATON AVE & BENSON AVE N/B Caton Ave Benson Ave  
## 2 S CATON AVE & BENSON AVE S/B Caton Ave Benson Ave  
## 3 WILKENS AVE & PINE HEIGHTS AVE E/B Wilkens Ave Pine Heights  
## 4 THE ALAMEDA & E 33RD ST S/B The Alameda 33rd St  
## 5 E 33RD ST & THE ALAMEDA E/B E 33rd The Alameda  
## 6 ERDMAN AVE & N MACON ST E/B Erdman Macon St  
## intersection Location.1  
## 1 Caton Ave & Benson Ave (39.2693779962, -76.6688185297)  
## 2 Caton Ave & Benson Ave (39.2693157898, -76.6689698176)  
## 3 Wilkens Ave & Pine Heights (39.2720252302, -76.676960806)  
## 4 The Alameda & 33rd St (39.3285013141, -76.5953545714)  
## 5 E 33rd & The Alameda (39.3283410623, -76.5953594625)  
## 6 Erdman & Macon St (39.3068045671, -76.5593167803)

## 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

*In my experience, the biggest trouble with reading flat files are quotation marks ` or " placed in data values, setting quote="" often resolves these*.

# Reading Excel files

## Excel files

*Still probably the most widely used format for sharing data*

[<http://office.microsoft.com/en-us/excel/>](http://office.microsoft.com/en-us/excel/)

## Example - Baltimore camera data

[<https://data.baltimorecity.gov/Transportation/Baltimore-Fixed-Speed-Cameras/dz54-2aru>](https://data.baltimorecity.gov/Transportation/Baltimore-Fixed-Speed-Cameras/dz54-2aru)

## Download the file to load

if(!file.exists("data")){dir.create("data")}  
fileUrl <- "https://data.baltimorecity.gov/api/views/dz54-2aru/rows.xlsx?accessType=DOWNLOAD"  
download.file(fileUrl,destfile="./data/cameras.xlsx",method="curl")  
dateDownloaded <- date()

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

library(xlsx)  
cameraData <- read.xlsx("./data/cameras.xlsx",sheetIndex=1,header=TRUE)  
head(cameraData)

address direction street crossStreet intersection  
1 S CATON AVE & BENSON AVE N/B Caton Ave Benson Ave Caton Ave & Benson Ave  
2 S CATON AVE & BENSON AVE S/B Caton Ave Benson Ave Caton Ave & Benson Ave  
3 WILKENS AVE & PINE HEIGHTS AVE E/B Wilkens Ave Pine Heights Wilkens Ave & Pine Heights  
4 THE ALAMEDA & E 33RD ST S/B The Alameda 33rd St The Alameda & 33rd St  
5 E 33RD ST & THE ALAMEDA E/B E 33rd The Alameda E 33rd & The Alameda  
6 ERDMAN AVE & N MACON ST E/B Erdman Macon St Erdman & Macon St  
 Location.1  
1 (39.2693779962, -76.6688185297)  
2 (39.2693157898, -76.6689698176)  
3 (39.2720252302, -76.676960806)  
4 (39.3285013141, -76.5953545714)  
5 (39.3283410623, -76.5953594625)  
6 (39.3068045671, -76.5593167803)

## Reading specific rows and columns

colIndex <- 2:3  
rowIndex <- 1:4  
cameraDataSubset <- read.xlsx("./data/cameras.xlsx",sheetIndex=1,  
 colIndex=colIndex,rowIndex=rowIndex)  
cameraDataSubset

direction street  
1 N/B Caton Ave  
2 S/B Caton Ave  
3 E/B Wilkens Ave

## 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](http://cran.r-project.org/web/packages/XLConnect/index.html) package has more options for writing and manipulating Excel files
* The [XLConnect vignette](http://cran.r-project.org/web/packages/XLConnect/vignettes/XLConnect.pdf) is a good place to start for that package
* In general it is advised to store your data in either a database or in comma separated files (.csv) or tab separated files (.tab/.txt) as they are easier to distribute.

# Reading XML

## 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

[<http://en.wikipedia.org/wiki/XML>](http://en.wikipedia.org/wiki/XML)

## 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>

[<http://en.wikipedia.org/wiki/XML>](http://en.wikipedia.org/wiki/XML)

## Example XML file

[<http://www.w3schools.com/xml/simple.xml>](http://www.w3schools.com/xml/simple.xml)

## Read the file into R

library(XML)  
fileUrl <- "http://www.w3schools.com/xml/simple.xml"  
doc <- xmlTreeParse(fileUrl,useInternal=TRUE)  
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 document

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>

## Programatically 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'

Information from: [<http://www.stat.berkeley.edu/~statcur/Workshop2/Presentations/XML.pdf>](http://www.stat.berkeley.edu/~statcur/Workshop2/Presentations/XML.pdf)

## Get the items on the menu and prices

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

[1] "Belgian Waffles" "Strawberry Belgian Waffles" "Berry-Berry Belgian Waffles"  
[4] "French Toast" "Homestyle Breakfast"

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

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

## Another example

[<http://espn.go.com/nfl/team/_/name/bal/baltimore-ravens>](http://espn.go.com/nfl/team/_/name/bal/baltimore-ravens)

## Viewing the source

[<http://espn.go.com/nfl/team/_/name/bal/baltimore-ravens>](http://espn.go.com/nfl/team/_/name/bal/baltimore-ravens)

## Extract content by attributes

fileUrl <- "http://espn.go.com/nfl/team/\_/name/bal/baltimore-ravens"  
doc <- htmlTreeParse(fileUrl,useInternal=TRUE)  
scores <- xpathSApply(doc,"//li[@class='score']",xmlValue)  
teams <- xpathSApply(doc,"//li[@class='team-name']",xmlValue)  
scores

[1] "49-27" "14-6" "30-9" "23-20" "26-23" "19-17" "19-16" "24-18"   
 [9] "20-17 OT" "23-20 OT" "19-3" "22-20" "29-26" "18-16" "41-7" "34-17"

teams

[1] "Denver" "Cleveland" "Houston" "Buffalo" "Miami" "Green Bay"   
 [7] "Pittsburgh" "Cleveland" "Cincinnati" "Chicago" "New York" "Pittsburgh"   
[13] "Minnesota" "Detroit" "New England" "Cincinnati"

## Notes and further resources

* Official XML tutorials [short](http://www.omegahat.org/RSXML/shortIntro.pdf), [long](http://www.omegahat.org/RSXML/Tour.pdf)
* [An outstanding guide to the XML package](http://www.stat.berkeley.edu/~statcur/Workshop2/Presentations/XML.pdf)

# Reading JSON

## JSON

* Javascript Object Notation
* Lightweight data storage
* Common format for data from application programming interfaces (APIs)
* 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 (unorderd, comma separated collection of key:value pairs in curley brackets *{}*)

[<http://en.wikipedia.org/wiki/JSON>](http://en.wikipedia.org/wiki/JSON)

## Example JSON file

## Reading data from JSON {jsonlite package}

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

[1] "id" "name" "full\_name" "owner"   
 [5] "private" "html\_url" "description" "fork"   
 [9] "url" "forks\_url" "keys\_url" "collaborators\_url"  
[13] "teams\_url" "hooks\_url" "issue\_events\_url" "events\_url"   
[17] "assignees\_url" "branches\_url" "tags\_url" "blobs\_url"   
[21] "git\_tags\_url" "git\_refs\_url" "trees\_url" "statuses\_url"   
[25] "languages\_url" "stargazers\_url" "contributors\_url" "subscribers\_url"   
[29] "subscription\_url" "commits\_url" "git\_commits\_url" "comments\_url"   
[33] "issue\_comment\_url" "contents\_url" "compare\_url" "merges\_url"   
[37] "archive\_url" "downloads\_url" "issues\_url" "pulls\_url"   
[41] "milestones\_url" "notifications\_url" "labels\_url" "releases\_url"   
[45] "created\_at" "updated\_at" "pushed\_at" "git\_url"   
[49] "ssh\_url" "clone\_url" "svn\_url" "homepage"   
[53] "size" "stargazers\_count" "watchers\_count" "language"   
[57] "has\_issues" "has\_downloads" "has\_wiki" "forks\_count"   
[61] "mirror\_url" "open\_issues\_count" "forks" "open\_issues"   
[65] "watchers" "default\_branch" "master\_branch"

jsonData$name

[1] "ballgown" "dataanalysis" "datascientist" "datasharing" "derfinder"   
 [6] "derfinder-1" "DSM" "EDA-Project" "futureofstats" "googleCite"   
[11] "graduate" "healthvis" "jhsph753" "jhsph753and4" "leekasso"   
[16] "modules" "rdsmGeneSig" "reviews" "rfitbit" "rpackages"   
[21] "sva" "swfdr" "talks" "testrepository" "tornado"   
[26] "tsp-devel" "tspreg"

## Nested objects in JSON

names(jsonData$owner)

[1] "login" "id" "avatar\_url" "gravatar\_id"   
 [5] "url" "html\_url" "followers\_url" "following\_url"   
 [9] "gists\_url" "starred\_url" "subscriptions\_url" "organizations\_url"   
[13] "repos\_url" "events\_url" "received\_events\_url" "type"   
[17] "site\_admin"

jsonData$owner$login

[1] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"  
[11] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"  
[21] "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek" "jtleek"

## Writing data frames to JSON

myjson <- toJSON(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" : 5.9,  
 "Sepal.Width" : 3,  
 "Petal.Length" : 5.1,  
 "Petal.Width" : 1.8,  
 "Species" : "virginica"  
 }  
]

[<http://www.r-bloggers.com/new-package-jsonlite-a-smarter-json-encoderdecoder/>](http://www.r-bloggers.com/new-package-jsonlite-a-smarter-json-encoderdecoder/)

## Convert back to JSON

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

[<http://www.r-bloggers.com/new-package-jsonlite-a-smarter-json-encoderdecoder/>](http://www.r-bloggers.com/new-package-jsonlite-a-smarter-json-encoderdecoder/)

## Further resources

* [<http://www.json.org/>](http://www.json.org/)
* A good tutorial on jsonlite - [<http://www.r-bloggers.com/new-package-jsonlite-a-smarter-json-encoderdecoder/>](http://www.r-bloggers.com/new-package-jsonlite-a-smarter-json-encoderdecoder/)
* [jsonlite vignette](http://cran.r-project.org/web/packages/jsonlite/vignettes/json-mapping.pdf)

# Using data.table

## data.table

* Inherets from data.frame
* All functions that accept data.frame work on data.table
* Written in C so it is much faster
* Much, much faster at subsetting, group, and updating

## Create data tables just like data frames

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 0.8528 a -0.97110  
2 1.1736 a -0.69167  
3 -0.7739 a 0.06864

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

x y z  
1: 0.41032 a 0.10602  
2: -1.74969 a -0.65800  
3: -0.08614 a 0.05627

## See all the data tables in memory

tables()

NAME NROW MB COLS KEY  
[1,] DT 9 1 x,y,z   
[2,] DT1 4 1 x,y x   
[3,] DT2 3 1 x,z x   
Total: 3MB

## Subsetting rows

DT[2,]

x y z  
1: -1.75 a -0.658

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

x y z  
1: 0.41032 a 0.10602  
2: -1.74969 a -0.65800  
3: -0.08614 a 0.05627

## Subsetting rows

DT[c(2,3)]

x y z  
1: -1.74969 a -0.65800  
2: -0.08614 a 0.05627

## Subsetting columns!?

DT[,c(2,3)]

[1] 2 3

## Column subsetting in data.table

* The subsetting function is modified for data.table
* The argument you pass after the comma is called an "expression"
* In R an expression is a collection of statements enclosed in curley brackets

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

[1] 10

print(k)

[1] 5

## Calculating values for variables with expressions

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

V1 V2  
1: 0.477 -3.307

DT[,table(y)]

y  
a b c   
3 3 3

## Adding new columns

DT[,w:=z^2]

x y z w  
1: 0.41032 a 0.10602 0.011239  
2: -1.74969 a -0.65800 0.432964  
3: -0.08614 a 0.05627 0.003167  
4: 2.17432 b 0.44756 0.200306  
5: 1.98301 b -0.63096 0.398111  
6: 0.10673 b 0.10226 0.010458  
7: 0.20678 c -0.69943 0.489205  
8: -0.95958 c -0.78335 0.613640  
9: 2.20682 c -1.24722 1.555565

## Adding new columns

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

x y z w  
1: 0.41032 2 0.10602 0.011239  
2: -1.74969 2 -0.65800 0.432964  
3: -0.08614 2 0.05627 0.003167  
4: 2.17432 2 0.44756 0.200306  
5: 1.98301 2 -0.63096 0.398111  
6: 0.10673 2 0.10226 0.010458  
7: 0.20678 2 -0.69943 0.489205  
8: -0.95958 2 -0.78335 0.613640  
9: 2.20682 2 -1.24722 1.555565

## Careful

head(DT,n=3)

x y z w  
1: 0.41032 2 0.10602 0.011239  
2: -1.74969 2 -0.65800 0.432964  
3: -0.08614 2 0.05627 0.003167

head(DT2,n=3)

x y z w  
1: 0.41032 2 0.10602 0.011239  
2: -1.74969 2 -0.65800 0.432964  
3: -0.08614 2 0.05627 0.003167

## Multiple operations

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

x y z w m  
1: 0.41032 2 0.10602 0.011239 2.464  
2: -1.74969 2 -0.65800 0.432964 1.374  
3: -0.08614 2 0.05627 0.003167 2.313  
4: 2.17432 2 0.44756 0.200306 2.930  
5: 1.98301 2 -0.63096 0.398111 2.667  
6: 0.10673 2 0.10226 0.010458 2.381  
7: 0.20678 2 -0.69943 0.489205 2.172  
8: -0.95958 2 -0.78335 0.613640 1.704  
9: 2.20682 2 -1.24722 1.555565 2.575

## plyr like operations

DT[,a:=x>0]

x y z w m a  
1: 0.41032 2 0.10602 0.011239 2.464 TRUE  
2: -1.74969 2 -0.65800 0.432964 1.374 FALSE  
3: -0.08614 2 0.05627 0.003167 2.313 FALSE  
4: 2.17432 2 0.44756 0.200306 2.930 TRUE  
5: 1.98301 2 -0.63096 0.398111 2.667 TRUE  
6: 0.10673 2 0.10226 0.010458 2.381 TRUE  
7: 0.20678 2 -0.69943 0.489205 2.172 TRUE  
8: -0.95958 2 -0.78335 0.613640 1.704 FALSE  
9: 2.20682 2 -1.24722 1.555565 2.575 TRUE

## plyr like operations

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

x y z w m a b  
1: 0.41032 2 0.10602 0.011239 2.464 TRUE 1.6255  
2: -1.74969 2 -0.65800 0.432964 1.374 FALSE -0.5819  
3: -0.08614 2 0.05627 0.003167 2.313 FALSE -0.5819  
4: 2.17432 2 0.44756 0.200306 2.930 TRUE 1.6255  
5: 1.98301 2 -0.63096 0.398111 2.667 TRUE 1.6255  
6: 0.10673 2 0.10226 0.010458 2.381 TRUE 1.6255  
7: 0.20678 2 -0.69943 0.489205 2.172 TRUE 1.6255  
8: -0.95958 2 -0.78335 0.613640 1.704 FALSE -0.5819  
9: 2.20682 2 -1.24722 1.555565 2.575 TRUE 1.6255

## Special variables

.N An integer, length 1, containing the number of elements of a factor level

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

x N  
1: a 33387  
2: c 33201  
3: b 33412

## Keys

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

x y  
 1: a 0.25959  
 2: a 0.91751  
 3: a -0.72232  
 4: a -0.80828  
 5: a -0.14135  
 6: a 2.25701  
 7: a -2.37955  
 8: a -0.45425  
 9: a -0.06007  
 10: a 0.86090  
 11: a -1.78466  
 12: a -0.13074  
 13: a -0.36984  
 14: a -0.18066  
 15: a -1.04973  
 16: a 0.37832  
 17: a -1.37079  
 18: a -0.31612  
  
  
100: a 0.38016  
 x y

## Joins

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

## Fast reading

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   
 0.312 0.015 0.326

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

user system elapsed   
 5.702 0.048 5.755

## Summary and further reading

* The latest development version contains new functions like melt and dcast for data.tables
* [<https://r-forge.r-project.org/scm/viewvc.php/pkg/NEWS?view=markup&root=datatable>](https://r-forge.r-project.org/scm/viewvc.php/pkg/NEWS?view=markup&root=datatable)
* Here is a list of differences between data.table and data.frame
* [<http://stackoverflow.com/questions/13618488/what-you-can-do-with-data-frame-that-you-cant-in-data-table>](http://stackoverflow.com/questions/13618488/what-you-can-do-with-data-frame-that-you-cant-in-data-table)
* Notes based on Raphael Gottardo's notes [<https://github.com/raphg/Biostat-578/blob/master/Advanced_data_manipulation.Rpres>](https://github.com/raphg/Biostat-578/blob/master/Advanced_data_manipulation.Rpres), who got them from Kevin Ushey.

# Reading mySQL

## mySQL

* Free and widely used open source database software
* Widely used in internet based applications
* Data are structured in
* Databases
* Tables within databases
* Fields within tables
* Each row is called a record

[<http://en.wikipedia.org/wiki/MySQL>](http://en.wikipedia.org/wiki/MySQL) [<http://www.mysql.com/>](http://www.mysql.com/)

## Example structure

[<http://dev.mysql.com/doc/employee/en/sakila-structure.html>](http://dev.mysql.com/doc/employee/en/sakila-structure.html)

## Step 1 - Install MySQL

[<http://dev.mysql.com/doc/refman/5.7/en/installing.html>](http://dev.mysql.com/doc/refman/5.7/en/installing.html)

## Step 2 - Install RMySQL

* On a Mac: install.packages("RMySQL")
* On Windows:
* Official instructions - [<http://biostat.mc.vanderbilt.edu/wiki/Main/RMySQL>](http://biostat.mc.vanderbilt.edu/wiki/Main/RMySQL) (may be useful for Mac/UNIX users as well)
* Potentially useful guide - [<http://www.ahschulz.de/2013/07/23/installing-rmysql-under-windows/>](http://www.ahschulz.de/2013/07/23/installing-rmysql-under-windows/)

## Example - UCSC database

[<http://genome.ucsc.edu/>](http://genome.ucsc.edu/)

## UCSC MySQL

[<http://genome.ucsc.edu/goldenPath/help/mysql.html>](http://genome.ucsc.edu/goldenPath/help/mysql.html)

## Connecting and listing databases

ucscDb <- dbConnect(MySQL(),user="genome",   
 host="genome-mysql.cse.ucsc.edu")  
result <- dbGetQuery(ucscDb,"show databases;"); dbDisconnect(ucscDb);

[1] TRUE

result

Database  
1 information\_schema  
2 ailMel1  
3 allMis1  
4 anoCar1  
5 anoCar2  
6 anoGam1  
7 apiMel1  
8 apiMel2  
9 aplCal1  
10 bosTau2  
11 bosTau3  
  
180 xenTro3

## Connecting to hg19 and listing tables

hg19 <- dbConnect(MySQL(),user="genome", db="hg19",  
 host="genome-mysql.cse.ucsc.edu")  
allTables <- dbListTables(hg19)  
length(allTables)

[1] 10949

allTables[1:5]

[1] "HInv" "HInvGeneMrna" "acembly" "acemblyClass" "acemblyPep"

## Get dimensions of a specific table

dbListFields(hg19,"affyU133Plus2")

[1] "bin" "matches" "misMatches" "repMatches" "nCount" "qNumInsert"   
 [7] "qBaseInsert" "tNumInsert" "tBaseInsert" "strand" "qName" "qSize"   
[13] "qStart" "qEnd" "tName" "tSize" "tStart" "tEnd"   
[19] "blockCount" "blockSizes" "qStarts" "tStarts"

dbGetQuery(hg19, "select count(\*) from affyU133Plus2")

count(\*)  
1 58463

## Read from the table

affyData <- dbReadTable(hg19, "affyU133Plus2")  
head(affyData)

bin matches misMatches repMatches nCount qNumInsert qBaseInsert tNumInsert tBaseInsert strand  
1 585 530 4 0 23 3 41 3 898 -  
2 585 3355 17 0 109 9 67 9 11621 -  
3 585 4156 14 0 83 16 18 2 93 -  
4 585 4667 9 0 68 21 42 3 5743 -  
5 585 5180 14 0 167 10 38 1 29 -  
6 585 468 5 0 14 0 0 0 0 -  
 qName qSize qStart qEnd tName tSize tStart tEnd blockCount  
1 225995\_x\_at 637 5 603 chr1 249250621 14361 15816 5  
2 225035\_x\_at 3635 0 3548 chr1 249250621 14381 29483 17  
3 226340\_x\_at 4318 3 4274 chr1 249250621 14399 18745 18  
4 1557034\_s\_at 4834 48 4834 chr1 249250621 14406 24893 23  
5 231811\_at 5399 0 5399 chr1 249250621 19688 25078 11  
6 236841\_at 487 0 487 chr1 249250621 27542 28029 1

## Select a specific subset

query <- dbSendQuery(hg19, "select \* from affyU133Plus2 where misMatches between 1 and 3")  
affyMis <- fetch(query); quantile(affyMis$misMatches)

0% 25% 50% 75% 100%   
 1 1 2 2 3

affyMisSmall <- fetch(query,n=10); dbClearResult(query);

[1] TRUE

dim(affyMisSmall)

[1] 10 22

## Don't forget to close the connection!

dbDisconnect(hg19)

[1] TRUE

## Further resources

* RMySQL vignette [<http://cran.r-project.org/web/packages/RMySQL/RMySQL.pdf>](http://cran.r-project.org/web/packages/RMySQL/RMySQL.pdf)
* List of commands [<http://www.pantz.org/software/mysql/mysqlcommands.html>](http://www.pantz.org/software/mysql/mysqlcommands.html)
* **Do not, do not, delete, add or join things from ensembl. Only select.**
* In general be careful with mysql commands
* A nice blog post summarizing some other commands [<http://www.r-bloggers.com/mysql-and-r/>](http://www.r-bloggers.com/mysql-and-r/)

# Reading HDF5

## HDF5

* Used for storing large data sets
* Supports storing a range of data types
* Heirarchical data format
* *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

[<http://www.hdfgroup.org/>](http://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 [<http://bioconductor.org/>](http://bioconductor.org/), primarily used for genomics but also has good "big data" packages
* Can be used to interface with hdf5 data sets.
* This lecture is modeled very closely on the rhdf5 tutorial that can be found here [<http://www.bioconductor.org/packages/release/bioc/vignettes/rhdf5/inst/doc/rhdf5.pdf>](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,nr=5,nc=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

## Notes and further resources

* hdf5 can be used to optimize reading/writing from disc in R
* The rhdf5 tutorial:
* [<http://www.bioconductor.org/packages/release/bioc/vignettes/rhdf5/inst/doc/rhdf5.pdf>](http://www.bioconductor.org/packages/release/bioc/vignettes/rhdf5/inst/doc/rhdf5.pdf)
* The HDF group has informaton on HDF5 in general [<http://www.hdfgroup.org/HDF5/>](http://www.hdfgroup.org/HDF5/)

# Reading data from the web

## Webscraping

**Webscraping**: Programatically extracting data from the HTML code of websites.

* It can be a great way to get data [How Netflix reverse engineered Hollywood](http://www.theatlantic.com/technology/archive/2014/01/how-netflix-reverse-engineered-hollywood/282679/)
* Many websites have information you may want to programaticaly read
* In some cases this is against the terms of service for the website
* Attempting to read too many pages too quickly can get your IP address blocked

[<http://en.wikipedia.org/wiki/Web_scraping>](http://en.wikipedia.org/wiki/Web_scraping)

## Example: Google scholar

[<http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en>](http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en)

## Getting data off webpages - readLines()

con = url("http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en")  
htmlCode = readLines(con)  
close(con)  
htmlCode

(returns HTML code)

## Parsing with XML

library(XML)  
url <- "http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en"  
html <- htmlTreeParse(url, useInternalNodes=T)  
  
xpathSApply(html, "//title", xmlValue)

[1] "Jeff Leek - Google Scholar Citations"

xpathSApply(html, "//td[@id='col-citedby']", xmlValue)

[1] "Cited by" "397" "259" "237" "172" "138" "125" "122"   
 [9] "109" "101" "34" "26" "26" "24" "19" "13"   
[17] "12" "10" "10" "7" "6"

## GET from the httr package

library(httr); html2 = GET(url)  
content2 = content(html2,as="text")  
parsedHtml = htmlParse(content2,asText=TRUE)  
xpathSApply(parsedHtml, "//title", xmlValue)

[1] "Jeff Leek - Google Scholar Citations"

## Accessing websites with passwords

pg1 = GET("http://httpbin.org/basic-auth/user/passwd")  
pg1

Response [http://httpbin.org/basic-auth/user/passwd]  
 Status: 401  
 Content-type:

[<http://cran.r-project.org/web/packages/httr/httr.pdf>](http://cran.r-project.org/web/packages/httr/httr.pdf)

## Accessing websites with passwords

pg2 = GET("http://httpbin.org/basic-auth/user/passwd",  
 authenticate("user","passwd"))  
pg2

Response [http://httpbin.org/basic-auth/user/passwd]  
 Status: 200  
 Content-type: application/json  
{  
 "authenticated": true,  
 "user": "user"  
}

names(pg2)

[1] "url" "handle" "status\_code" "headers" "cookies" "content"   
[7] "times" "config"

[<http://cran.r-project.org/web/packages/httr/httr.pdf>](http://cran.r-project.org/web/packages/httr/httr.pdf)

## Using handles

google = handle("http://google.com")  
pg1 = GET(handle=google,path="/")  
pg2 = GET(handle=google,path="search")

[<http://cran.r-project.org/web/packages/httr/httr.pdf>](http://cran.r-project.org/web/packages/httr/httr.pdf)

## Notes and further resources

* R Bloggers has a number of examples of web scraping [<http://www.r-bloggers.com/?s=Web+Scraping>](http://www.r-bloggers.com/?s=Web+Scraping)
* The httr help file has useful examples [<http://cran.r-project.org/web/packages/httr/httr.pdf>](http://cran.r-project.org/web/packages/httr/httr.pdf)
* See later lectures on APIs

# Reading data from APIs

## Application programming interfaces

[<https://dev.twitter.com/docs/api/1/get/blocks/blocking>](https://dev.twitter.com/docs/api/1/get/blocks/blocking)

## Creating an application

[<https://dev.twitter.com/apps>](https://dev.twitter.com/appsmyapp%20%3c-%20oauth_app(%22twitter%22,%20key%20=%20%22TYrWFPkFAkn4G5BbkWINYw%22))

## Creating an application

## Creating an application

## 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]

created\_at id id\_str  
1 Mon Jan 13 05:18:04 +0000 2014 4.225984e+17 422598398940684288  
 text  
1 Now that P. Norvig's regex golf IPython notebook hit Slashdot, let's see if our traffic spike tops the previous one: http://t.co/Vc6JhZXOo8

## How did I know what url to use?

[<https://dev.twitter.com/docs/api/1.1/get/search/tweets>](https://dev.twitter.com/docs/api/1.1/get/search/tweets)

## In general look at the documentation

[<https://dev.twitter.com/docs/api/1.1/overview>](https://dev.twitter.com/docs/api/1.1/overview)

## 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, Githb, etc.

# Subsetting and sorting

## Subsetting - quick review

set.seed(13435)  
X <- data.frame("var1"=sample(1:5),"var2"=sample(6:10),"var3"=sample(11:15))  
X <- X[sample(1:5),]; X$var2[c(1,3)] = NA  
X

var1 var2 var3  
1 2 NA 15  
4 1 10 11  
2 3 NA 12  
3 5 6 14  
5 4 9 13

## Subsetting - quick review

X[,1]

[1] 2 1 3 5 4

X[,"var1"]

[1] 2 1 3 5 4

X[1:2,"var2"]

[1] NA 10

## Logicals ands and ors

X[(X$var1 <= 3 & X$var3 > 11),]

var1 var2 var3  
1 2 NA 15  
2 3 NA 12

X[(X$var1 <= 3 | X$var3 > 15),]

var1 var2 var3  
1 2 NA 15  
4 1 10 11  
2 3 NA 12

## Dealing with missing values

X[which(X$var2 > 8),]

var1 var2 var3  
4 1 10 11  
5 4 9 13

## Sorting

sort(X$var1)

[1] 1 2 3 4 5

sort(X$var1,decreasing=TRUE)

[1] 5 4 3 2 1

sort(X$var2,na.last=TRUE)

[1] 6 9 10 NA NA

## Ordering

X[order(X$var1),]

var1 var2 var3  
4 1 10 11  
1 2 NA 15  
2 3 NA 12  
5 4 9 13  
3 5 6 14

## Ordering

X[order(X$var1,X$var3),]

var1 var2 var3  
4 1 10 11  
1 2 NA 15  
2 3 NA 12  
5 4 9 13  
3 5 6 14

## Ordering with plyr

library(plyr)  
arrange(X,var1)

var1 var2 var3  
1 1 10 11  
2 2 NA 15  
3 3 NA 12  
4 4 9 13  
5 5 6 14

arrange(X,desc(var1))

var1 var2 var3  
1 5 6 14  
2 4 9 13  
3 3 NA 12  
4 2 NA 15  
5 1 10 11

## Adding rows and columns

X$var4 <- rnorm(5)  
X

var1 var2 var3 var4  
1 2 NA 15 0.18760  
4 1 10 11 1.78698  
2 3 NA 12 0.49669  
3 5 6 14 0.06318  
5 4 9 13 -0.53613

## Adding rows and columns

Y <- cbind(X,rnorm(5))  
Y

var1 var2 var3 var4 rnorm(5)  
1 2 NA 15 0.18760 0.62578  
4 1 10 11 1.78698 -2.45084  
2 3 NA 12 0.49669 0.08909  
3 5 6 14 0.06318 0.47839  
5 4 9 13 -0.53613 1.00053

## Notes and further resources

* R programming in the Data Science Track
* Andrew Jaffe's lecture notes [<http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf>](http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf)

# Summarizing data

## Example data set

[<https://data.baltimorecity.gov/Community/Restaurants/k5ry-ef3g>](https://data.baltimorecity.gov/Community/Restaurants/k5ry-ef3g)

## Getting the data from the web

if(!file.exists("./data")){dir.create("./data")}  
fileUrl <- "https://data.baltimorecity.gov/api/views/k5ry-ef3g/rows.csv?accessType=DOWNLOAD"  
download.file(fileUrl,destfile="./data/restaurants.csv",method="curl")  
restData <- read.csv("./data/restaurants.csv")

## Look at a bit of the data

head(restData,n=3)

name zipCode neighborhood councilDistrict policeDistrict Location.1  
1 410 21206 Frankford 2 NORTHEASTERN 4509 BELAIR ROAD\nBaltimore, MD\n  
2 1919 21231 Fells Point 1 SOUTHEASTERN 1919 FLEET ST\nBaltimore, MD\n  
3 SAUTE 21224 Canton 1 SOUTHEASTERN 2844 HUDSON ST\nBaltimore, MD\n

tail(restData,n=3)

name zipCode neighborhood councilDistrict policeDistrict  
1325 ZINK'S CAF\u0090 21213 Belair-Edison 13 NORTHEASTERN  
1326 ZISSIMOS BAR 21211 Hampden 7 NORTHERN  
1327 ZORBAS 21224 Greektown 2 SOUTHEASTERN  
 Location.1  
1325 3300 LAWNVIEW AVE\nBaltimore, MD\n  
1326 1023 36TH ST\nBaltimore, MD\n  
1327 4710 EASTERN Ave\nBaltimore, MD\n

## Make summary

summary(restData)

name zipCode neighborhood councilDistrict  
 MCDONALD'S : 8 Min. :-21226 Downtown :128 Min. : 1.00   
 POPEYES FAMOUS FRIED CHICKEN: 7 1st Qu.: 21202 Fells Point : 91 1st Qu.: 2.00   
 SUBWAY : 6 Median : 21218 Inner Harbor: 89 Median : 9.00   
 KENTUCKY FRIED CHICKEN : 5 Mean : 21185 Canton : 81 Mean : 7.19   
 BURGER KING : 4 3rd Qu.: 21226 Federal Hill: 42 3rd Qu.:11.00   
 DUNKIN DONUTS : 4 Max. : 21287 Mount Vernon: 33 Max. :14.00   
 (Other) :1293 (Other) :863   
 policeDistrict Location.1   
 SOUTHEASTERN:385 1101 RUSSELL ST\nBaltimore, MD\n: 9   
 CENTRAL :288 201 PRATT ST\nBaltimore, MD\n : 8   
 SOUTHERN :213 2400 BOSTON ST\nBaltimore, MD\n : 8   
 NORTHERN :157 300 LIGHT ST\nBaltimore, MD\n : 5   
 NORTHEASTERN: 72 300 CHARLES ST\nBaltimore, MD\n : 4   
 EASTERN : 67 301 LIGHT ST\nBaltimore, MD\n : 4   
 (Other) :145 (Other) :1289

## More in depth information

str(restData)

'data.frame': 1327 obs. of 6 variables:  
 $ name : Factor w/ 1277 levels "#1 CHINESE KITCHEN",..: 9 3 992 1 2 4 5 6 7 8 ...  
 $ zipCode : int 21206 21231 21224 21211 21223 21218 21205 21211 21205 21231 ...  
 $ neighborhood : Factor w/ 173 levels "Abell","Arlington",..: 53 52 18 66 104 33 98 133 98 157 ...  
 $ councilDistrict: int 2 1 1 14 9 14 13 7 13 1 ...  
 $ policeDistrict : Factor w/ 9 levels "CENTRAL","EASTERN",..: 3 6 6 4 8 3 6 4 6 6 ...  
 $ Location.1 : Factor w/ 1210 levels "1 BIDDLE ST\nBaltimore, MD\n",..: 835 334 554 755 492 537 505 530 507 569 ...

## Quantiles of quantitative variables

quantile(restData$councilDistrict,na.rm=TRUE)

0% 25% 50% 75% 100%   
 1 2 9 11 14

quantile(restData$councilDistrict,probs=c(0.5,0.75,0.9))

50% 75% 90%   
 9 11 12

## Make table

table(restData$zipCode,useNA="ifany")

-21226 21201 21202 21205 21206 21207 21208 21209 21210 21211 21212 21213 21214 21215   
 1 136 201 27 30 4 1 8 23 41 28 31 17 54   
 21216 21217 21218 21220 21222 21223 21224 21225 21226 21227 21229 21230 21231 21234   
 10 32 69 1 7 56 199 19 18 4 13 156 127 7   
 21237 21239 21251 21287   
 1 3 2 1

## Make table

table(restData$councilDistrict,restData$zipCode)

-21226 21201 21202 21205 21206 21207 21208 21209 21210 21211 21212 21213 21214 21215 21216  
 1 0 0 37 0 0 0 0 0 0 0 0 2 0 0 0  
 2 0 0 0 3 27 0 0 0 0 0 0 0 0 0 0  
 3 0 0 0 0 0 0 0 0 0 0 0 2 17 0 0  
 4 0 0 0 0 0 0 0 0 0 0 27 0 0 0 0  
 5 0 0 0 0 0 3 0 6 0 0 0 0 0 31 0  
 6 0 0 0 0 0 0 0 1 19 0 0 0 0 15 1  
  
 21217 21218 21220 21222 21223 21224 21225 21226 21227 21229 21230 21231 21234 21237 21239  
 1 0 0 0 7 0 140 1 0 0 0 1 124 0 0 0  
 21251 21287  
 1 0 0  
 2 0 0  
 3 2 0  
 4 0 0  
 5 0 0  
 6 0 0  
 7 0 0  
 8 0 0  
 9 0 0  
 10 0 0  
 11 0 0  
 12 0 0  
 13 0 1  
 14 0 0

## Check for missing values

sum(is.na(restData$councilDistrict))

[1] 0

any(is.na(restData$councilDistrict))

[1] FALSE

all(restData$zipCode > 0)

[1] FALSE

## Row and column sums

colSums(is.na(restData))

name zipCode neighborhood councilDistrict policeDistrict Location.1   
 0 0 0 0 0 0

all(colSums(is.na(restData))==0)

[1] TRUE

## Values with specific characteristics

table(restData$zipCode %in% c("21212"))

FALSE TRUE   
 1299 28

table(restData$zipCode %in% c("21212","21213"))

FALSE TRUE   
 1268 59

## Values with specific characteristics

restData[restData$zipCode %in% c("21212","21213"),]

name zipCode neighborhood councilDistrict  
29 BAY ATLANTIC CLUB 21212 Downtown 11  
39 BERMUDA BAR 21213 Broadway East 12  
92 ATWATER'S 21212 Chinquapin Park-Belvedere 4  
111 BALTIMORE ESTONIAN SOCIETY 21213 South Clifton Park 12  
187 CAFE ZEN 21212 Rosebank 4  
220 CERIELLO FINE FOODS 21212 Chinquapin Park-Belvedere 4  
266 CLIFTON PARK GOLF COURSE SNACK BAR 21213 Darley Park 14  
276 CLUB HOUSE BAR & GRILL 21213 Orangeville Industrial Area 13  
289 CLUBHOUSE BAR & GRILL 21213 Orangeville Industrial Area 13  
291 COCKY LOU'S 21213 Broadway East 12  
362 DREAM TAVERN, CARRIBEAN U.S.A. 21213 Broadway East 13  
373 DUNKIN DONUTS 21212 Homeland 4  
383 EASTSIDE SPORTS SOCIAL CLUB 21213 Broadway East 13  
417 FIELDS OLD TRAIL 21212 Mid-Govans 4  
475 GRAND CRU 21212 Chinquapin Park-Belvedere 4  
545 RANDY'S BAR 21213 Broadway East 12  
604 MURPHY'S NEIGHBORHOOD BAR & GRILL 21212 Mid-Govans 4  
616 NEOPOL 21212 Chinquapin Park-Belvedere 4  
620 NEW CLUB THUNDERBIRD INC. 21213 Middle East 13  
626 NEW MAYFIELD, INC. 21213 Belair-Edison 13  
678 IKAN SEAFOOD 21212 Chinquapin Park-Belvedere 4  
711 KAY-CEE CLUB 21212 Homeland 4  
763 LA'RAE 21213 Oliver 12  
777 LEMONGRASS BALTIMORE 21213 Little Italy 1  
779 LEN'S SANDWICH SHOP 21213 Broadway East 12  
 policeDistrict Location.1  
29 CENTRAL 206 REDWOOD ST\nBaltimore, MD\n  
39 EASTERN 1801 NORTH AVE\nBaltimore, MD\n  
92 NORTHERN 529 BELVEDERE AVE\nBaltimore, MD\n  
111 EASTERN 1932 BELAIR RD\nBaltimore, MD\n  
187 NORTHERN 438 BELVEDERE AVE\nBaltimore, MD\n  
220 NORTHERN 529 BELVEDERE AVE\nBaltimore, MD\n  
266 NORTHEASTERN 2701 ST LO DR\nBaltimore, MD\n  
276 EASTERN 4217 ERDMAN AVE\nBaltimore, MD\n  
289 EASTERN 4217 ERDMAN AVE\nBaltimore, MD\n  
291 EASTERN 2101 NORTH AVE\nBaltimore, MD\n  
362 EASTERN 2300 LAFAYETTE AVE\nBaltimore, MD\n  
373 NORTHERN 5422 YORK RD\nBaltimore, MD\n  
383 EASTERN 1203 COLLINGTON AVE\nBaltimore, MD\n

## Cross tabs

data(UCBAdmissions)  
DF = as.data.frame(UCBAdmissions)  
summary(DF)

Admit Gender Dept Freq   
 Admitted:12 Male :12 A:4 Min. : 8   
 Rejected:12 Female:12 B:4 1st Qu.: 80   
 C:4 Median :170   
 D:4 Mean :189   
 E:4 3rd Qu.:302   
 F:4 Max. :512

## Cross tabs

xt <- xtabs(Freq ~ Gender + Admit,data=DF)  
xt

Admit  
Gender Admitted Rejected  
 Male 1198 1493  
 Female 557 1278

## Flat tables

warpbreaks$replicate <- rep(1:9, len = 54)  
xt = xtabs(breaks ~.,data=warpbreaks)  
xt

, , replicate = 1  
  
 tension  
wool L M H  
 A 26 18 36  
 B 27 42 20  
  
, , replicate = 2  
  
 tension  
wool L M H  
 A 30 21 21  
 B 14 26 21  
  
, , replicate = 3  
  
 tension  
wool L M H  
 A 54 29 24  
 B 29 19 24  
  
, , replicate = 4  
  
 tension  
wool L M H  
 A 25 17 18  
 B 19 16 17  
  
, , replicate = 5  
  
 tension  
wool L M H  
 A 70 12 10  
 B 29 39 13  
  
, , replicate = 6  
  
 tension  
wool L M H  
 A 52 18 43  
 B 31 28 15  
  
, , replicate = 7  
  
 tension  
wool L M H  
 A 51 35 28  
 B 41 21 15  
  
, , replicate = 8  
  
 tension  
wool L M H  
 A 26 30 15  
 B 20 39 16  
  
, , replicate = 9  
  
 tension  
wool L M H  
 A 67 36 26  
 B 44 29 28

## Flat tables

ftable(xt)

replicate 1 2 3 4 5 6 7 8 9  
wool tension   
A L 26 30 54 25 70 52 51 26 67  
 M 18 21 29 17 12 18 35 30 36  
 H 36 21 24 18 10 43 28 15 26  
B L 27 14 29 19 29 31 41 20 44  
 M 42 26 19 16 39 28 21 39 29  
 H 20 21 24 17 13 15 15 16 28

## Size of a data set

fakeData = rnorm(1e5)  
object.size(fakeData)

800040 bytes

print(object.size(fakeData),units="Mb")

0.8 Mb

# Creating new variables

## Why create new variables?

* Often the raw data won't have a value you are looking for
* You will need to transform the data to get the values you would like
* Usually you will add those values to the data frames you are working with
* Common variables to create
* Missingness indicators
* "Cutting up" quantitative variables
* Applying transforms

## Example data set

[<https://data.baltimorecity.gov/Community/Restaurants/k5ry-ef3g>](https://data.baltimorecity.gov/Community/Restaurants/k5ry-ef3g)

## Getting the data from the web

if(!file.exists("./data")){dir.create("./data")}  
fileUrl <- "https://data.baltimorecity.gov/api/views/k5ry-ef3g/rows.csv?accessType=DOWNLOAD"  
download.file(fileUrl,destfile="./data/restaurants.csv",method="curl")  
restData <- read.csv("./data/restaurants.csv")

## Creating sequences

*Sometimes you need an index for your data set*

s1 <- seq(1,10,by=2) ; s1

[1] 1 3 5 7 9

s2 <- seq(1,10,length=3); s2

[1] 1.0 5.5 10.0

x <- c(1,3,8,25,100); seq(along = x)

[1] 1 2 3 4 5

## Subsetting variables

restData$nearMe = restData$neighborhood %in% c("Roland Park", "Homeland")  
table(restData$nearMe)

FALSE TRUE   
 1314 13

## Creating binary variables

restData$zipWrong = ifelse(restData$zipCode < 0, TRUE, FALSE)  
table(restData$zipWrong,restData$zipCode < 0)

FALSE TRUE  
 FALSE 1326 0  
 TRUE 0 1

## Creating categorical variables

restData$zipGroups = cut(restData$zipCode,breaks=quantile(restData$zipCode))  
table(restData$zipGroups)

(-2.123e+04,2.12e+04] (2.12e+04,2.122e+04] (2.122e+04,2.123e+04] (2.123e+04,2.129e+04]   
 337 375 282 332

table(restData$zipGroups,restData$zipCode)

-21226 21201 21202 21205 21206 21207 21208 21209 21210 21211 21212 21213  
 (-2.123e+04,2.12e+04] 0 136 201 0 0 0 0 0 0 0 0 0  
 (2.12e+04,2.122e+04] 0 0 0 27 30 4 1 8 23 41 28 31  
 (2.122e+04,2.123e+04] 0 0 0 0 0 0 0 0 0 0 0 0  
 (2.123e+04,2.129e+04] 0 0 0 0 0 0 0 0 0 0 0 0  
   
 21214 21215 21216 21217 21218 21220 21222 21223 21224 21225 21226 21227  
 (-2.123e+04,2.12e+04] 0 0 0 0 0 0 0 0 0 0 0 0  
 (2.12e+04,2.122e+04] 17 54 10 32 69 0 0 0 0 0 0 0  
 (2.122e+04,2.123e+04] 0 0 0 0 0 1 7 56 199 19 0 0  
 (2.123e+04,2.129e+04] 0 0 0 0 0 0 0 0 0 0 18 4  
   
 21229 21230 21231 21234 21237 21239 21251 21287  
 (-2.123e+04,2.12e+04] 0 0 0 0 0 0 0 0  
 (2.12e+04,2.122e+04] 0 0 0 0 0 0 0 0  
 (2.122e+04,2.123e+04] 0 0 0 0 0 0 0 0  
 (2.123e+04,2.129e+04] 13 156 127 7 1 3 2 1

## Easier cutting

library(Hmisc)  
restData$zipGroups = cut2(restData$zipCode,g=4)  
table(restData$zipGroups)

[-21226,21205) [ 21205,21220) [ 21220,21227) [ 21227,21287]   
 338 375 300 314

## Creating factor variables

restData$zcf <- factor(restData$zipCode)  
restData$zcf[1:10]

[1] 21206 21231 21224 21211 21223 21218 21205 21211 21205 21231  
32 Levels: -21226 21201 21202 21205 21206 21207 21208 21209 21210 21211 21212 21213 21214 ... 21287

class(restData$zcf)

[1] "factor"

## Levels of factor variables

yesno <- sample(c("yes","no"),size=10,replace=TRUE)  
yesnofac = factor(yesno,levels=c("yes","no"))  
relevel(yesnofac,ref="no")

[1] yes yes yes yes no yes yes yes no no   
Levels: no yes

as.numeric(yesnofac)

[1] 1 1 1 1 2 1 1 1 2 2

## Cutting produces factor variables

library(Hmisc)  
restData$zipGroups = cut2(restData$zipCode,g=4)  
table(restData$zipGroups)

[-21226,21205) [ 21205,21220) [ 21220,21227) [ 21227,21287]   
 338 375 300 314

## Using the mutate function

library(Hmisc); library(plyr)  
restData2 = mutate(restData,zipGroups=cut2(zipCode,g=4))  
table(restData2$zipGroups)

[-21226,21205) [ 21205,21220) [ 21220,21227) [ 21227,21287]   
 338 375 300 314

## Common transforms

* abs(x) absolute value
* sqrt(x) square root
* ceiling(x) ceiling(3.475) is 4
* floor(x) floor(3.475) is 3
* round(x,digits=n) round(3.475,digits=2) is 3.48
* signif(x,digits=n) signif(3.475,digits=2) is 3.5
* cos(x), sin(x) etc.
* log(x) natural logarithm
* log2(x), log10(x) other common logs
* exp(x) exponentiating x

[<http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf>](http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf) [<http://statmethods.net/management/functions.html>](http://statmethods.net/management/functions.html)

## Notes and further reading

* A tutorial from the developer of plyr - [<http://plyr.had.co.nz/09-user/>](http://plyr.had.co.nz/09-user/)
* Andrew Jaffe's R notes [<http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf>](http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf)
* A nice lecture on categorical and factor variables [<http://www.stat.berkeley.edu/classes/s133/factors.html>](http://www.stat.berkeley.edu/classes/s133/factors.html)

# Reshaping data

## The goal is tidy data

1. Each variable forms a column
2. Each observation forms a row
3. Each table/file stores data about one kind of observation (e.g. people/hospitals).

[<http://vita.had.co.nz/papers/tidy-data.pdf>](http://vita.had.co.nz/papers/tidy-data.pdf)

[Leek, Taub, and Pineda 2011 PLoS One](http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0026895)

## Start with reshaping

library(reshape2)  
head(mtcars)

mpg cyl disp hp drat wt qsec vs am gear carb  
Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4  
Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4  
Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1  
Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1  
Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2  
Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1

## Melting data frames

mtcars$carname <- rownames(mtcars)  
carMelt <- melt(mtcars,id=c("carname","gear","cyl"),measure.vars=c("mpg","hp"))  
head(carMelt,n=3)

carname gear cyl variable value  
1 Mazda RX4 4 6 mpg 21.0  
2 Mazda RX4 Wag 4 6 mpg 21.0  
3 Datsun 710 4 4 mpg 22.8

tail(carMelt,n=3)

carname gear cyl variable value  
62 Ferrari Dino 5 6 hp 175  
63 Maserati Bora 5 8 hp 335  
64 Volvo 142E 4 4 hp 109

[<http://www.statmethods.net/management/reshape.html>](http://www.statmethods.net/management/reshape.html)

## Casting data frames

cylData <- dcast(carMelt, cyl ~ variable)  
cylData

cyl mpg hp  
1 4 11 11  
2 6 7 7  
3 8 14 14

cylData <- dcast(carMelt, cyl ~ variable,mean)  
cylData

cyl mpg hp  
1 4 26.66 82.64  
2 6 19.74 122.29  
3 8 15.10 209.21

[<http://www.statmethods.net/management/reshape.html>](http://www.statmethods.net/management/reshape.html)

## Averaging values

head(InsectSprays)

count spray  
1 10 A  
2 7 A  
3 20 A  
4 14 A  
5 14 A  
6 12 A

tapply(InsectSprays$count,InsectSprays$spray,sum)

A B C D E F   
174 184 25 59 42 200

[<http://www.r-bloggers.com/a-quick-primer-on-split-apply-combine-problems/>](http://www.r-bloggers.com/a-quick-primer-on-split-apply-combine-problems/)

## Another way - split

spIns = split(InsectSprays$count,InsectSprays$spray)  
spIns

$A  
 [1] 10 7 20 14 14 12 10 23 17 20 14 13  
  
$B  
 [1] 11 17 21 11 16 14 17 17 19 21 7 13  
  
$C  
 [1] 0 1 7 2 3 1 2 1 3 0 1 4  
  
$D  
 [1] 3 5 12 6 4 3 5 5 5 5 2 4  
  
$E  
 [1] 3 5 3 5 3 6 1 1 3 2 6 4  
  
$F  
 [1] 11 9 15 22 15 16 13 10 26 26 24 13

## Another way - apply

sprCount = lapply(spIns,sum)  
sprCount

$A  
[1] 174  
  
$B  
[1] 184  
  
$C  
[1] 25  
  
$D  
[1] 59  
  
$E  
[1] 42  
  
$F  
[1] 200

## Another way - combine

unlist(sprCount)

A B C D E F   
174 184 25 59 42 200

sapply(spIns,sum)

A B C D E F   
174 184 25 59 42 200

## Another way - plyr package

ddply(InsectSprays,.(spray),summarize,sum=sum(count))

spray sum  
1 A 174  
2 B 184  
3 C 25  
4 D 59  
5 E 42  
6 F 200

## Creating a new variable

spraySums <- ddply(InsectSprays,.(spray),summarize,sum=ave(count,FUN=sum))  
dim(spraySums)

[1] 72 2

head(spraySums)

spray sum  
1 A 174  
2 A 174  
3 A 174  
4 A 174  
5 A 174  
6 A 174

## More information

* A tutorial from the developer of plyr - [<http://plyr.had.co.nz/09-user/>](http://plyr.had.co.nz/09-user/)
* A nice reshape tutorial [<http://www.slideshare.net/jeffreybreen/reshaping-data-in-r>](http://www.slideshare.net/jeffreybreen/reshaping-data-in-r)
* A good plyr primer - [<http://www.r-bloggers.com/a-quick-primer-on-split-apply-combine-problems/>](http://www.r-bloggers.com/a-quick-primer-on-split-apply-combine-problems/)
* See also the functions
* acast - for casting as multi-dimensional arrays
* arrange - for faster reordering without using order() commands
* mutate - adding new variables

# Merging data

## Peer review experiment data

[<http://www.plosone.org/article/info:doi/10.1371/journal.pone.0026895>](http://www.plosone.org/article/info:doi/10.1371/journal.pone.0026895)

## Peer review data

if(!file.exists("./data")){dir.create("./data")}  
fileUrl1 = "https://dl.dropboxusercontent.com/u/7710864/data/reviews-apr29.csv"  
fileUrl2 = "https://dl.dropboxusercontent.com/u/7710864/data/solutions-apr29.csv"  
download.file(fileUrl1,destfile="./data/reviews.csv",method="curl")  
download.file(fileUrl2,destfile="./data/solutions.csv",method="curl")  
reviews = read.csv("./data/reviews.csv"); solutions <- read.csv("./data/solutions.csv")  
head(reviews,2)

id solution\_id reviewer\_id start stop time\_left accept  
1 1 3 27 1304095698 1304095758 1754 1  
2 2 4 22 1304095188 1304095206 2306 1

head(solutions,2)

id problem\_id subject\_id start stop time\_left answer  
1 1 156 29 1304095119 1304095169 2343 B  
2 2 269 25 1304095119 1304095183 2329 C

## Merging data - merge()

* Merges data frames
* Important parameters: *x*,*y*,*by*,*by.x*,*by.y*,*all*

names(reviews)

[1] "id" "solution\_id" "reviewer\_id" "start" "stop" "time\_left"   
[7] "accept"

names(solutions)

[1] "id" "problem\_id" "subject\_id" "start" "stop" "time\_left" "answer"

## Merging data - merge()

mergedData = merge(reviews,solutions,by.x="solution\_id",by.y="id",all=TRUE)  
head(mergedData)

solution\_id id reviewer\_id start.x stop.x time\_left.x accept problem\_id subject\_id  
1 1 4 26 1304095267 1304095423 2089 1 156 29  
2 2 6 29 1304095471 1304095513 1999 1 269 25  
3 3 1 27 1304095698 1304095758 1754 1 34 22  
4 4 2 22 1304095188 1304095206 2306 1 19 23  
5 5 3 28 1304095276 1304095320 2192 1 605 26  
6 6 16 22 1304095303 1304095471 2041 1 384 27  
 start.y stop.y time\_left.y answer  
1 1304095119 1304095169 2343 B  
2 1304095119 1304095183 2329 C  
3 1304095127 1304095146 2366 C  
4 1304095127 1304095150 2362 D  
5 1304095127 1304095167 2345 A  
6 1304095131 1304095270 2242 C

## Default - merge all common column names

intersect(names(solutions),names(reviews))

[1] "id" "start" "stop" "time\_left"

mergedData2 = merge(reviews,solutions,all=TRUE)  
head(mergedData2)

id start stop time\_left solution\_id reviewer\_id accept problem\_id subject\_id answer  
1 1 1304095119 1304095169 2343 NA NA NA 156 29 B  
2 1 1304095698 1304095758 1754 3 27 1 NA NA <NA>  
3 2 1304095119 1304095183 2329 NA NA NA 269 25 C  
4 2 1304095188 1304095206 2306 4 22 1 NA NA <NA>  
5 3 1304095127 1304095146 2366 NA NA NA 34 22 C  
6 3 1304095276 1304095320 2192 5 28 1 NA NA <NA>

## Using join in the plyr package

*Faster, but less full featured - defaults to left join, see help file for more*

df1 = data.frame(id=sample(1:10),x=rnorm(10))  
df2 = data.frame(id=sample(1:10),y=rnorm(10))  
arrange(join(df1,df2),id)

id x y  
1 1 0.2514 0.2286  
2 2 0.1048 0.8395  
3 3 -0.1230 -1.1165  
4 4 1.5057 -0.1121  
5 5 -0.2505 1.2124  
6 6 0.4699 -1.6038  
7 7 0.4627 -0.8060  
8 8 -1.2629 -1.2848  
9 9 -0.9258 -0.8276  
10 10 2.8065 0.5794

## If you have multiple data frames

df1 = data.frame(id=sample(1:10),x=rnorm(10))  
df2 = data.frame(id=sample(1:10),y=rnorm(10))  
df3 = data.frame(id=sample(1:10),z=rnorm(10))  
dfList = list(df1,df2,df3)  
join\_all(dfList)

id x y z  
1 6 0.39093 -0.16670 0.56523  
2 1 -1.90467 0.43811 -0.37449  
3 7 -1.48798 -0.85497 -0.69209  
4 10 -2.59440 0.39591 -0.36134  
5 3 -0.08539 0.08053 1.01247  
6 4 -1.63165 -0.13158 0.21927  
7 5 -0.50594 0.24256 -0.44003  
8 9 -0.85062 -2.08066 -0.96950  
9 2 -0.63767 -0.10069 0.09002  
10 8 1.20439 1.29138 -0.88586

## More on merging data

* The quick R data merging page - [<http://www.statmethods.net/management/merging.html>](http://www.statmethods.net/management/merging.html)
* plyr information - [<http://plyr.had.co.nz/>](http://plyr.had.co.nz/)
* Types of joins - [<http://en.wikipedia.org/wiki/Join_(SQL)>](http://en.wikipedia.org/wiki/Join_(SQL))

# Editing text variables

## Example - Baltimore camera data

[<https://data.baltimorecity.gov/Transportation/Baltimore-Fixed-Speed-Cameras/dz54-2aru>](https://data.baltimorecity.gov/Transportation/Baltimore-Fixed-Speed-Cameras/dz54-2aru)

## Fixing character vectors - tolower(), toupper()

if(!file.exists("./data")){dir.create("./data")}  
fileUrl <- "https://data.baltimorecity.gov/api/views/dz54-2aru/rows.csv?accessType=DOWNLOAD"  
download.file(fileUrl,destfile="./data/cameras.csv",method="curl")  
cameraData <- read.csv("./data/cameras.csv")  
names(cameraData)

[1] "address" "direction" "street" "crossStreet" "intersection" "Location.1"

tolower(names(cameraData))

[1] "address" "direction" "street" "crossstreet" "intersection" "location.1"

## Fixing character vectors - strsplit()

* Good for automatically splitting variable names
* Important parameters: *x*, *split*

splitNames = strsplit(names(cameraData),"\\.")  
splitNames[[5]]

[1] "intersection"

splitNames[[6]]

[1] "Location" "1"

## Quick aside - lists

mylist <- list(letters = c("A", "b", "c"), numbers = 1:3, matrix(1:25, ncol = 5))  
head(mylist)

$letters  
[1] "A" "b" "c"  
  
$numbers  
[1] 1 2 3  
  
[[3]]  
 [,1] [,2] [,3] [,4] [,5]  
[1,] 1 6 11 16 21  
[2,] 2 7 12 17 22  
[3,] 3 8 13 18 23  
[4,] 4 9 14 19 24  
[5,] 5 10 15 20 25

[<http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%203.pdf>](http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%203.pdf)

## Quick aside - lists

mylist[1]

$letters  
[1] "A" "b" "c"

mylist$letters

[1] "A" "b" "c"

mylist[[1]]

[1] "A" "b" "c"

[<http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%203.pdf>](http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%203.pdf)

## Fixing character vectors - sapply()

* Applies a function to each element in a vector or list
* Important parameters: *X*,*FUN*

splitNames[[6]][1]

[1] "Location"

firstElement <- function(x){x[1]}  
sapply(splitNames,firstElement)

[1] "address" "direction" "street" "crossStreet" "intersection" "Location"

## Peer review experiment data

[<http://www.plosone.org/article/info:doi/10.1371/journal.pone.0026895>](http://www.plosone.org/article/info:doi/10.1371/journal.pone.0026895)

## Peer review data

fileUrl1 <- "https://dl.dropboxusercontent.com/u/7710864/data/reviews-apr29.csv"  
fileUrl2 <- "https://dl.dropboxusercontent.com/u/7710864/data/solutions-apr29.csv"  
download.file(fileUrl1,destfile="./data/reviews.csv",method="curl")  
download.file(fileUrl2,destfile="./data/solutions.csv",method="curl")  
reviews <- read.csv("./data/reviews.csv"); solutions <- read.csv("./data/solutions.csv")  
head(reviews,2)

id solution\_id reviewer\_id start stop time\_left accept  
1 1 3 27 1304095698 1304095758 1754 1  
2 2 4 22 1304095188 1304095206 2306 1

head(solutions,2)

id problem\_id subject\_id start stop time\_left answer  
1 1 156 29 1304095119 1304095169 2343 B  
2 2 269 25 1304095119 1304095183 2329 C

## Fixing character vectors - sub()

* Important parameters: *pattern*, *replacement*, *x*

names(reviews)

[1] "id" "solution\_id" "reviewer\_id" "start" "stop" "time\_left"   
[7] "accept"

sub("\_","",names(reviews),)

[1] "id" "solutionid" "reviewerid" "start" "stop" "timeleft" "accept"

## Fixing character vectors - gsub()

testName <- "this\_is\_a\_test"  
sub("\_","",testName)

[1] "thisis\_a\_test"

gsub("\_","",testName)

[1] "thisisatest"

## Finding values - grep(),grepl()

grep("Alameda",cameraData$intersection)

[1] 4 5 36

table(grepl("Alameda",cameraData$intersection))

FALSE TRUE   
 77 3

cameraData2 <- cameraData[!grepl("Alameda",cameraData$intersection),]

## More on grep()

grep("Alameda",cameraData$intersection,value=TRUE)

[1] "The Alameda & 33rd St" "E 33rd & The Alameda" "Harford \n & The Alameda"

grep("JeffStreet",cameraData$intersection)

integer(0)

length(grep("JeffStreet",cameraData$intersection))

[1] 0

[<http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%203.pdf>](http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%203.pdf)

## More useful string functions

library(stringr)  
nchar("Jeffrey Leek")

[1] 12

substr("Jeffrey Leek",1,7)

[1] "Jeffrey"

paste("Jeffrey","Leek")

[1] "Jeffrey Leek"

## More useful string functions

paste0("Jeffrey","Leek")

[1] "JeffreyLeek"

str\_trim("Jeff ")

[1] "Jeff"

## Important points about text in data sets

* Names of variables should be
* All lower case when possible
* Descriptive (Diagnosis versus Dx)
* Not duplicated
* Not have underscores or dots or white spaces
* Variables with character values
* Should usually be made into factor variables (depends on application)
* Should be descriptive (use TRUE/FALSE instead of 0/1 and Male/Female versus 0/1 or M/F)

# Regular Expressions

## Regular expressions

* Regular expressions can be thought of as a combination of literals and *metacharacters*
* To draw an analogy with natural language, think of literal text forming the words of this language, and the metacharacters defining its grammar
* Regular expressions have a rich set of metacharacters

## Literals

Simplest pattern consists only of literals. The literal âœnuclearâ would match to the following lines:

Ooh. I just learned that to keep myself alive after a  
nuclear blast! All I have to do is milk some rats  
then drink the milk. Aweosme. :}  
  
Laozi says nuclear weapons are mas macho  
  
Chaos in a country that has nuclear weapons -- not good.  
  
my nephew is trying to teach me nuclear physics, or  
possibly just trying to show me how smart he is  
so Iâ™ll be proud of him [which I am].  
  
lol if you ever say "nuclear" people immediately think  
DEATH by radiation LOL

## Literals

The literal âœObamaâ would match to the following lines

Politics r dum. Not 2 long ago Clinton was sayin Obama  
was crap n now she sez vote 4 him n unite? WTF?  
Screw em both + Mcain. Go Ron Paul!  
  
Clinton conceeds to Obama but will her followers listen??  
  
Are we sure Chelsea didnâ™t vote for Obama?  
  
thinking ... Michelle Obama is terrific!  
  
jetlag..no sleep...early mornig to starbux..Ms. Obama  
was moving

## Regular Expressions

* Simplest pattern consists only of literals; a match occurs if the sequence of literals occurs anywhere in the text being tested
* What if we only want the word âœObamaâ? or sentences that end in the word âœClintonâ, or âœclintonâ or âœclintoâ?

## Regular Expressions

We need a way to express - whitespace word boundaries - sets of literals - the beginning and end of a line - alternatives (âœwarâ or âœpeaceâ) Metacharacters to the rescue!

## Metacharacters

Some metacharacters represent the start of a line

^i think

will match the lines

i think we all rule for participating  
i think i have been outed  
i think this will be quite fun actually  
i think i need to go to work  
i think i first saw zombo in 1999.

## Metacharacters

$ represents the end of a line

morning$

will match the lines

well they had something this morning  
then had to catch a tram home in the morning  
dog obedience school in the morning  
and yes happy birthday i forgot to say it earlier this morning  
I walked in the rain this morning  
good morning

## Character Classes with []

We can list a set of characters we will accept at a given point in the match

[Bb][Uu][Ss][Hh]

will match the lines

The democrats are playing, "Name the worst thing about Bush!"  
I smelled the desert creosote bush, brownies, BBQ chicken  
BBQ and bushwalking at Molonglo Gorge  
Bush TOLD you that North Korea is part of the Axis of Evil  
Iâ™m listening to Bush - Hurricane (Album Version)

## Character Classes with []

^[Ii] am

will match

i am so angry at my boyfriend i canâ™t even bear to  
look at him  
  
i am boycotting the apple store  
  
I am twittering from iPhone  
  
I am a very vengeful person when you ruin my sweetheart.  
  
I am so over this. I need food. Mmmm bacon...

## Character Classes with []

Similarly, you can specify a range of letters [a-z] or [a-zA-Z]; notice that the order doesnâ™t matter

^[0-9][a-zA-Z]

will match the lines

7th inning stretch  
2nd half soon to begin. OSU did just win something  
3am - cant sleep - too hot still.. :(  
5ft 7 sent from heaven  
1st sign of starvagtion

## Character Classes with []

When used at the beginning of a character class, the âœ^â is also a metacharacter and indicates matching characters NOT in the indicated class

[^?.]$

will match the lines

i like basketballs  
6 and 9  
dont worry... we all die anyway!  
Not in Baghdad  
helicopter under water? hmmm

# Regular Expressions II

## More Metacharacters

âœ.â is used to refer to any character. So

9.11

will match the lines

its stupid the post 9-11 rules  
if any 1 of us did 9/11 we would have been caught in days.  
NetBios: scanning ip 203.169.114.66  
Front Door 9:11:46 AM  
Sings: 0118999881999119725...3 !

## More Metacharacters: |

This does not mean âœpipeâ in the context of regular expressions; instead it translates to âœorâ; we can use it to combine two expressions, the subexpressions being called alternatives

flood|fire

will match the lines

is firewire like usb on none macs?  
the global flood makes sense within the context of the bible  
yeah ive had the fire on tonight  
... and the floods, hurricanes, killer heatwaves, rednecks, gun nuts, etc.  
ï¿¼

## More Metacharacters: |

We can include any number of alternatives...

flood|earthquake|hurricane|coldfire

will match the lines

Not a whole lot of hurricanes in the Arctic.  
We do have earthquakes nearly every day somewhere in our State  
hurricanes swirl in the other direction  
coldfire is STRAIGHT!  
â™cause we keep getting earthquakes

## More Metacharacters: |

The alternatives can be real expressions and not just literals

^[Gg]ood|[Bb]ad

will match the lines

good to hear some good knews from someone here  
Good afternoon fellow american infidels!  
good on you-what do you drive?  
Katie... guess they had bad experiences...  
my middle name is trouble, Miss Bad News

## More Metacharacters: ( and )

Subexpressions are often contained in parentheses to constrain the alternatives

^([Gg]ood|[Bb]ad)

will match the lines

bad habbit  
bad coordination today  
good, becuase there is nothing worse than a man in kinky underwear  
Badcop, its because people want to use drugs  
Good Monday Holiday  
Good riddance to Limey

## More Metacharacters: ?

The question mark indicates that the indicated expression is optional

[Gg]eorge( [Ww]\.)? [Bb]ush

will match the lines

i bet i can spell better than you and george bush combined  
BBC reported that President George W. Bush claimed God told him to invade I  
a bird in the hand is worth two george bushes

## One thing to note...

In the following

[Gg]eorge( [Ww]\.)? [Bb]ush

we wanted to match a âœ.â as a literal period; to do that, we had to âœescapeâ the metacharacter, preceding it with a backslash In general, we have to do this for any metacharacter we want to include in our match

## More metacharacters: \* and +

The \* and + signs are metacharacters used to indicate repetition; \* means âœany number, including none, of the itemâ and + means âœat least one of the itemâ

(.\*)

will match the lines

anyone wanna chat? (24, m, germany)  
hello, 20.m here... ( east area + drives + webcam )  
(he means older men)  
()

## More metacharacters: \* and +

The \* and + signs are metacharacters used to indicate repetition; \* means âœany number, including none, of the itemâ and + means âœat least one of the itemâ

[0-9]+ (.\*)[0-9]+

will match the lines

working as MP here 720 MP battallion, 42nd birgade  
so say 2 or 3 years at colleage and 4 at uni makes us 23 when and if we fin  
it went down on several occasions for like, 3 or 4 \*days\*  
Mmmm its time 4 me 2 go 2 bed

## More metacharacters: { and }

{ and } are referred to as interval quantifiers; the let us specify the minimum and maximum number of matches of an expression

[Bb]ush( +[^ ]+ +){1,5} debate

will match the lines

Bush has historically won all major debates heâ™s done.  
in my view, Bush doesnâ™t need these debates..  
bush doesnâ™t need the debates? maybe you are right  
Thatâ™s what Bush supporters are doing about the debate.  
Felix, I donâ™t disagree that Bush was poorly prepared for the debate.  
indeed, but still, Bush should have taken the debate more seriously.  
Keep repeating that Bush smirked and scowled during the debate

## More metacharacters: and

* m,n means at least m but not more than n matches
* m means exactly m matches
* m, means at least m matches

## More metacharacters: ( and ) revisited

* In most implementations of regular expressions, the parentheses not only limit the scope of alternatives divided by a âœ|â, but also can be used to âœrememberâ text matched by the subexpression enclosed
* We refer to the matched text with , , etc.

## More metacharacters: ( and ) revisited

So the expression

+([a-zA-Z]+) +\1 +

will match the lines

time for bed, night night twitter!  
blah blah blah blah  
my tattoo is so so itchy today  
i was standing all all alone against the world outside...  
hi anybody anybody at home  
estudiando css css css css.... que desastritooooo

## More metacharacters: ( and ) revisited

The \* is âœgreedyâ so it always matches the *longest* possible string that satisfies the regular expression. So

^s(.\*)s

matches

sitting at starbucks  
setting up mysql and rails  
studying stuff for the exams  
spaghetti with marshmallows  
stop fighting with crackers  
sore shoulders, stupid ergonomics

## More metacharacters: ( and ) revisited

The greediness of \* can be turned off with the ?, as in

^s(.\*?)s$

## Summary

* Regular expressions are used in many different languages; not unique to R.
* Regular expressions are composed of literals and metacharacters that represent sets or classes of characters/words
* Text processing via regular expressions is a very powerful way to extract data from âœunfriendlyâ sources (not all data comes as a CSV file)
* Used with the functions grep,grepl,sub,gsub and others that involve searching for text strings (Thanks to Mark Hansen for some material in this lecture.)

# Working with dates

## Starting simple

d1 = date()  
d1

[1] "Sun Jan 12 17:48:33 2014"

class(d1)

[1] "character"

## Date class

d2 = Sys.Date()  
d2

[1] "2014-01-12"

class(d2)

[1] "Date"

## Formatting dates

%d = day as number (0-31), %a = abbreviated weekday,%A = unabbreviated weekday, %m = month (00-12), %b = abbreviated month, %B = unabbrevidated month, %y = 2 digit year, %Y = four digit year

format(d2,"%a %b %d")

[1] "Sun Jan 12"

## Creating dates

x = c("1jan1960", "2jan1960", "31mar1960", "30jul1960"); z = as.Date(x, "%d%b%Y")  
z

[1] "1960-01-01" "1960-01-02" "1960-03-31" "1960-07-30"

z[1] - z[2]

Time difference of -1 days

as.numeric(z[1]-z[2])

[1] -1

## Converting to Julian

weekdays(d2)

[1] "Sunday"

months(d2)

[1] "January"

julian(d2)

[1] 16082  
attr(,"origin")  
[1] "1970-01-01"

## Lubridate

library(lubridate); ymd("20140108")

[1] "2014-01-08 UTC"

mdy("08/04/2013")

[1] "2013-08-04 UTC"

dmy("03-04-2013")

[1] "2013-04-03 UTC"

[<http://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/>](http://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/)

## Dealing with times

ymd\_hms("2011-08-03 10:15:03")

[1] "2011-08-03 10:15:03 UTC"

ymd\_hms("2011-08-03 10:15:03",tz="Pacific/Auckland")

[1] "2011-08-03 10:15:03 NZST"

?Sys.timezone

[<http://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/>](http://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/)

## Some functions have slightly different syntax

x = dmy(c("1jan2013", "2jan2013", "31mar2013", "30jul2013"))  
wday(x[1])

[1] 3

wday(x[1],label=TRUE)

[1] Tues  
Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat

## Notes and further resources

* More information in this nice lubridate tutorial [<http://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/>](http://www.r-statistics.com/2012/03/do-more-with-dates-and-times-in-r-with-lubridate-1-1-0/)
* The lubridate vignette is the same content [<http://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html>](http://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html)
* Ultimately you want your dates and times as class "Date" or the classes "POSIXct", "POSIXlt". For more information type ?POSIXlt

# Data resources

## Open Government Sites

* United Nations [<http://data.un.org/>](http://data.un.org/)
* U.S. [<http://www.data.gov/>](http://www.data.gov/)
* [List of cities/states with open data](http://simplystatistics.org/2012/01/02/list-of-cities-states-with-open-data-help-me-find/)
* United Kingdom [<http://data.gov.uk/>](http://data.gov.uk/)
* France [<http://www.data.gouv.fr/>](http://www.data.gouv.fr/)
* Ghana [<http://data.gov.gh/>](http://data.gov.gh/)
* Australia [<http://data.gov.au/>](http://data.gov.au/)
* Germany [<https://www.govdata.de/>](https://www.govdata.de/)
* Hong Kong [<http://www.gov.hk/en/theme/psi/datasets/>](http://www.gov.hk/en/theme/psi/datasets/)
* Japan [<http://www.data.go.jp/>](http://www.data.go.jp/)
* Many more [<http://www.data.gov/opendatasites>](http://www.data.gov/opendatasites)

## Gapminder

[<http://www.gapminder.org/>](http://www.gapminder.org/)

## Survey data from the United States

[<http://www.asdfree.com/>](http://www.asdfree.com/)

## Infochimps Marketplace

[<http://www.infochimps.com/marketplace>](http://www.infochimps.com/marketplace)

## Kaggle

[<http://www.kaggle.com/>](http://www.kaggle.com/)

## Collections by data scientists

* Hilary Mason <http://bitly.com/bundles/hmason/1>
* Peter Skomoroch <https://delicious.com/pskomoroch/dataset>
* Jeff Hammerbacher <http://www.quora.com/Jeff-Hammerbacher/Introduction-to-Data-Science-Data-Sets>
* Gregory Piatetsky-Shapiro <http://www.kdnuggets.com/gps.html>
* [<http://blog.mortardata.com/post/67652898761/6-dataset-lists-curated-by-data-scientists>](http://blog.mortardata.com/post/67652898761/6-dataset-lists-curated-by-data-scientists)

## More specialized collections

* [Stanford Large Network Data](http://snap.stanford.edu/data/)
* [UCI Machine Learning](http://archive.ics.uci.edu/ml/)
* [KDD Nugets Datasets](http://www.kdnuggets.com/datasets/index.html)
* [CMU Statlib](http://lib.stat.cmu.edu/datasets/)
* [Gene expression omnibus](http://www.ncbi.nlm.nih.gov/geo/)
* [ArXiv Data](http://arxiv.org/help/bulk_data)
* [Public Data Sets on Amazon Web Services](http://aws.amazon.com/publicdatasets/)

## Some API's with R interfaces

* [twitter](https://dev.twitter.com/) and [twitteR](http://cran.r-project.org/web/packages/twitteR/index.html) package
* [figshare](http://api.figshare.com/docs/intro.html) and [rfigshare](http://cran.r-project.org/web/packages/rfigshare/index.html)
* [PLoS](http://api.plos.org/) and [rplos](http://cran.r-project.org/web/packages/rplos/rplos.pdf)
* [rOpenSci](http://ropensci.org/packages/index.html)
* [Facebook](https://developers.facebook.com/) and [RFacebook](http://cran.r-project.org/web/packages/Rfacebook/)
* [Google maps](https://developers.google.com/maps/) and [RGoogleMaps](http://cran.r-project.org/web/packages/RgoogleMaps/index.html)