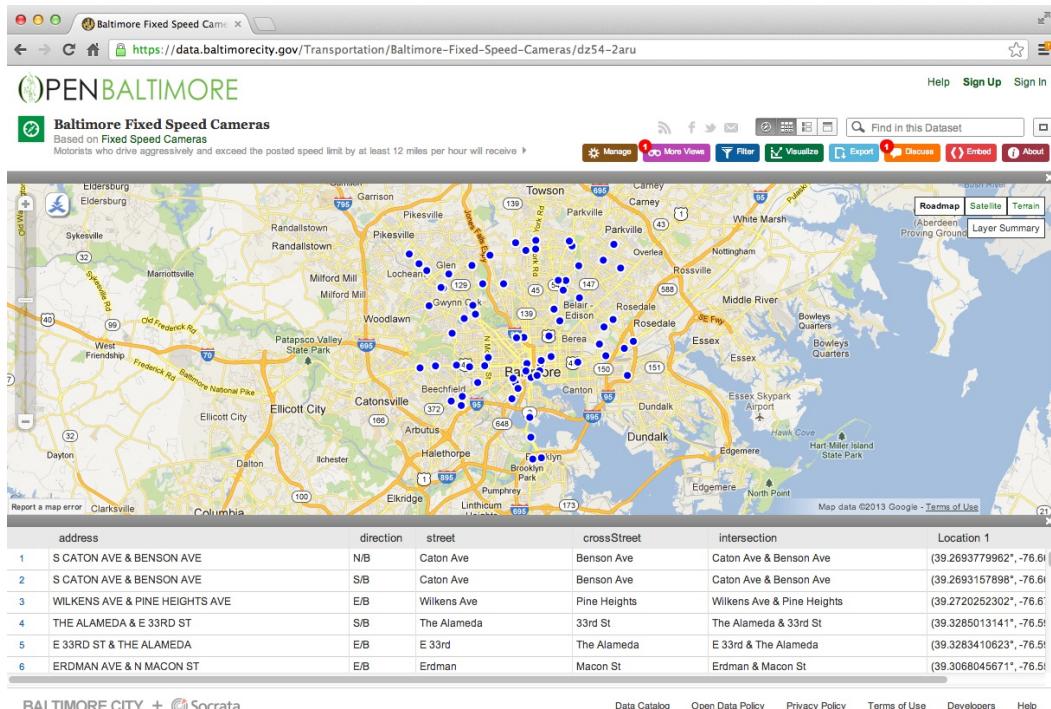




Editing text variables

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

Example - Baltimore camera data



BALTIMORE CITY + Socrata

Data Catalog Open Data Policy Privacy Policy Terms of Use Developers Help

<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
```

Quick aside - lists

```
mylist[1]
```

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

```
mylist$letters
```

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

```
mylist[[1]]
```

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

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

The screenshot shows a web browser displaying a PLOS ONE article. The URL in the address bar is www.plosone.org/article/info:doi/10.1371/journal.pone.0026895. The page features a banner for 'Simplify your research with automatic and continuous dosing'. The main navigation includes 'Articles', 'For Authors', 'About Us', 'Search', and 'sign in'. Below the navigation, the article is identified as 'OPEN ACCESS' and 'PEER-REVIEWED'. The title of the research article is 'Cooperation between Referees and Authors Increases Peer Review Accuracy' by Jeffrey T. Leek, Margaret A. Taub, and Fernando J. Pineda. The article has 6,497 views, 2 citations, 61 academic bookmarks, and 108 social shares. The interface includes tabs for 'Article', 'About the Authors', 'Metrics', 'Comments', and 'Related Content', along with buttons for 'Download', 'Print', and 'Share'. The 'Comments' section is expanded, showing a diagram illustrating the peer review process and a table of comments.

<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)
```

	<code>id</code>	<code>solution_id</code>	<code>reviewer_id</code>	<code>start</code>	<code>stop</code>	<code>time_left</code>	<code>accept</code>
1	1	3	27	1304095698	1304095758	1754	1
2	2	4	22	1304095188	1304095206	2306	1

```
head(solutions, 2)
```

	<code>id</code>	<code>problem_id</code>	<code>subject_id</code>	<code>start</code>	<code>stop</code>	<code>time_left</code>	<code>answer</code>
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
```

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

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

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 “nuclear” 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 I'll be proud of him [which I am].

lol if you ever say "nuclear" people immediately think
DEATH by radiation LOL

Literals

The literal “Obama” 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 concedes to Obama but will her followers listen??

Are we sure Chelsea didn'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 “Obama”? or sentences that end in the word “Clinton”, or “clinton” or “clinto”?

Regular Expressions

We need a way to express

- whitespace word boundaries
- sets of literals
- the beginning and end of a line
- alternatives (“war” or “peace”) 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  
I'm listening to Bush - Hurricane (Album Version)
```

Character Classes with []

```
^[Ii] am
```

will match

i am so angry at my boyfriend i can'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 doesn'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 “” 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

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

More Metacharacters

“.” 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 “pipe” in the context of regular expressions; instead it translates to “or”; 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.
```

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!

'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 habit  
bad coordination today  
good, because 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 “.” as a literal period; to do that, we had to “escape” 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 “any number, including none, of the item” and + means “at least one of the item”

```
(.*)
```

will match the lines

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

More metacharacters: * and +

The * and + signs are metacharacters used to indicate repetition; * means “any number, including none, of the item” and + means “at least one of the item”

```
[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; they 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 he's done.  
in my view, Bush doesn't need these debates..  
bush doesn't need the debates? maybe you are right  
That's what Bush supporters are doing about the debate.  
Felix, I don'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 “|”, but also can be used to “remember” text matched by the subexpression enclosed
- We refer to the matched text with \1, \2, 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 desastritoooooo
```

More metacharacters: (and) revisited

The * is “greedy” 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 “unfriendly” 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

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

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 = unabbreviated 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/>

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/>

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/>
- The lubridate vignette is the same content <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

Jeffrey Leek
Johns Hopkins Bloomberg School of Public Health

Open Government Sites

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

Gapminder

The screenshot shows a web browser window for the Gapminder website. The URL in the address bar is www.gapminder.org/data/. The page title is "Data in Gapminder World". The main content area displays a table of indicators, each with a name, provider, category, subcategory, download, view, and visualize links. The indicators listed are:

Indicator name	Data provider	Category	Subcategory	Download	View	Visualize
Adults with HIV (%; age 15-49)	Based on UNAIDS	Health	HIV			
Age at 1st marriage (women)	Various sources	Population				
Aged 15+ employment rate (%)	International Labour Organization	Work	Employment rate			
Aged 15+ labour force participation rate (%)	International Labour Organization	Work	Labour force participation			
Aged 15+ unemployment rate (%)	International Labour Organization	Work	Unemployment			
Aged 15-24 employment rate (%)	International Labour Organization	Work	Employment rate			

<http://www.gapminder.org/>

Survey data from the United States

The screenshot shows a web browser window with the URL www.asdfree.com. The page title is "analyze survey data for free". The navigation bar includes links for "about / faq", "main code repository", "latest releases", "rss", "ajdamico@gmail.com", "tutorials", and "r-bloggers". A sidebar on the left lists "AVAILABLE DATA" with links to various survey datasets like ACS, CPS, and HRS. The main content area features a section titled "analyze the health and retirement study (hrs) with r". It provides a detailed explanation of the HRS dataset, mentioning its longitudinal nature and the challenges of analyzing it. It also links to instructions on how to merge years and download the data. At the bottom, there's a form to enter an email address for updates.

analyze survey data for free

about / faq main code repository latest releases rss ajdamico@gmail.com tutorials
r-bloggers

reproducible survey analysis syntax from a website that's easy to type.

AVAILABLE DATA

- american community survey (acs)
- area resource file (arf)
- basic stand alone medicare claims public use files (bsapufs)
- behavioral risk factor surveillance system (brfss)
- consumer expenditure survey (ces)
- current population survey (cps)
- general social survey (gss)
- health and retirement study (hrs)
- medical expenditure panel survey (meps)
- national health and nutrition examination survey (nhanes)
- national health interview survey (nhis)
- national study on drug use and health (nsduh)

METHODS

- why and how to install monetdb with r on windows

enter your email address for updates:
www.asdfree.com/2013/01/analyze-health-and-retirement-study-hrs.html

1992 - 2010 download HRS microdata.

analyze the health and retirement study (hrs) with r

the hrs is the one and only longitudinal survey of american seniors. with a panel starting its third decade, the current pool of respondents includes older folks who have been interviewed every two years as far back as 1992. unlike cross-sectional or short panel surveys, respondents keep responding until, well, death do us part. paid for by the national institute on aging and administered by the university of michigan's institute for social research, if you apply for an interviewer job with them, i hope you like werther's original.

figuring out how to analyze this data set might trigger your *fight-or-flight* synapses if you just start clicking around on michigan's website. instead, read pages numbered 10-17 (pdf pages 12-19) of [this introduction pdf](#) and don't touch the data until you understand figure a-3 on that last page. if you start enjoying yourself, here's [the whole book](#). after that, it's time to [register](#) for access to the (free) data. keep your username and password handy, you'll need it for the top of the download automation r script. next, look at [this data flowchart](#); to get an idea of why the [data download](#) page is such a righteous jungle. but wait, good news: umich recently farmed out its data management to the [rand corporation](#), who promptly constructed a [giant consolidated file](#) with one record per respondent across the whole panel. oh so beautiful. the rand files make much of the older data and syntax examples obsolete, so when you come across stuff like [instructions on how to merge years](#), you can happily ignore them - rand has done it for you.

the health and retirement study only includes noninstitutionalized adults when new respondents get added to the panel (as they were in 1992, 1993, 1998, 2004, and 2010) but once they're in, they're in - respondents have a weight of zero for interview waves when they were nursing home residents; but they're still responding and will continue to contribute to your statistics so long as you're generalizing about a population from a previous wave (for example: it's possible to compute "among all americans who were 50+ years old in 1998, x% lived in nursing homes by 2010"). my source for that 411? [page 13 of the design doc](#). wicked. this new github repository contains five scripts:

<http://www.asdfree.com/>

Infochimps Marketplace

The screenshot shows the Infochimps Data Marketplace homepage. At the top, there's a navigation bar with links for Home, Solutions, Platform, Resources, Community, Company, Blog, and a prominent orange 'Request a Demo' button. Below the navigation is a search bar labeled 'Search for data' and a link to 'API Documentation'. On the right side of the header is a 'Log in' button. The main content area has a dark header with the text 'Data Marketplace'. Below it, a sub-header reads: 'With thousands of public and proprietary data sets, our data marketplace is a great resource of experimental and sample data for data scientists and developers.' There are two main sections: 'Geo APIs' (represented by a globe icon) and 'Social APIs' (represented by a speech bubble icon). Each section has a brief description and a 'Learn more' link. Below these sections is a 'Top Tags' section featuring a grid of small, light-colored rectangular tags with various data-related terms like 'locations', 'business', 'geo', 'population', 'census', 'government', 'demographics', 'america', etc.

<http://www.infochimps.com/marketplace>

What's in your data?

Participate in competitions

Kaggle is an arena where you can match your data science skills against a global cadre of experts in statistics, mathematics, and machine learning. Whether you're a world-class algorithm wizard competing for prize money or a novice looking to learn from the best, here's your chance to jump in and geek out, for fame, fortune, or fun.

[Join as a participant](#)

(Need convincing?)

Create a competition

Kaggle is a platform for data prediction competitions that allows organizations to post their data and have it scrutinized by the world's best data scientists. In exchange for a prize, winning competitors provide the algorithms that beat all other methods of solving a data crunching problem. Most data problems can be framed as a competition.

[Learn more about hosting](#)

<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>

More specialized collections

- [Stanford Large Newtork Data](#)
- [UCI Machine Learning](#)
- [KDD Nuggets Datasets](#)
- [CMU Statlib](#)
- [Gene expression omnibus](#)
- [ArXiv Data](#)
- [Public Data Sets on Amazon Web Services](#)

Some API's with R interfaces

- `twitter` and `twitteR` package
- `figshare` and `rfigshare`
- `PLoS` and `rplos`
- `rOpenSci`
- `Facebook` and `RFacebook`
- `Google maps` and `RGoogleMaps`

Load the dplyr package

This step is important!

```
library(dplyr)

## 
## Attaching package: 'dplyr'
## 
## The following object is masked from 'package:stats':
## 
##     filter
## 
## The following objects are masked from 'package:base':
## 
##     intersect, setdiff, setequal, union
```

select

```
chicago <- readRDS("chicago.rds")
dim(chicago)
```

```
## [1] 6940     8
```

```
head(select(chicago, 1:5))
```

```
##   city tmpd    dptp        date pm25tmean2
## 1 chic 31.5 31.500 1987-01-01         NA
## 2 chic 33.0 29.875 1987-01-02         NA
## 3 chic 33.0 27.375 1987-01-03         NA
## 4 chic 29.0 28.625 1987-01-04         NA
## 5 chic 32.0 28.875 1987-01-05         NA
## 6 chic 40.0 35.125 1987-01-06         NA
```

select

```
names(chicago)[1:3]
```

```
## [1] "city" "tmpd" "dptp"
```

```
head(select(chicago, city:dptp))
```

```
##   city tmpd   dptp
## 1 chic 31.5 31.500
## 2 chic 33.0 29.875
## 3 chic 33.0 27.375
## 4 chic 29.0 28.625
## 5 chic 32.0 28.875
## 6 chic 40.0 35.125
```

select

In dplyr you can do

```
head(select(chicago, -(city:dptp)))
```

Equivalent base R

```
i <- match("city", names(chicago))
j <- match("dptp", names(chicago))
head(chicago[, -(i:j)])
```

filter

```
chic.f <- filter(chicago, pm25tmean2 > 30)
head(select(chic.f, 1:3, pm25tmean2), 10)
```

```
##      city tmpd dptp pm25tmean2
## 1    chic   23 21.9     38.10
## 2    chic   28 25.8     33.95
## 3    chic   55 51.3     39.40
## 4    chic   59 53.7     35.40
## 5    chic   57 52.0     33.30
## 6    chic   57 56.0     32.10
## 7    chic   75 65.8     56.50
## 8    chic   61 59.0     33.80
## 9    chic   73 60.3     30.30
## 10   chic   78 67.1     41.40
```

filter

```
chic.f <- filter(chicago, pm25tmean2 > 30 & tmpd > 80)
head(select(chic.f, 1:3, pm25tmean2, tmpd), 10)
```

```
##      city tmpd dptp pm25tmean2
## 1  chic   81 71.2    39.6000
## 2  chic   81 70.4    31.5000
## 3  chic   82 72.2    32.3000
## 4  chic   84 72.9    43.7000
## 5  chic   85 72.6    38.8375
## 6  chic   84 72.6    38.2000
## 7  chic   82 67.4    33.0000
## 8  chic   82 63.5    42.5000
## 9  chic   81 70.4    33.1000
## 10 chic   82 66.2    38.8500
```

arrange

Reordering rows of a data frame (while preserving corresponding order of other columns) is normally a pain to do in R.

```
chicago <- arrange(chicago, date)  
head(select(chicago, date, pm25tmean2), 3)
```

```
##           date pm25tmean2  
## 1 1987-01-01      NA  
## 2 1987-01-02      NA  
## 3 1987-01-03      NA
```

```
tail(select(chicago, date, pm25tmean2), 3)
```

```
##           date pm25tmean2  
## 6938 2005-12-29    7.45000  
## 6939 2005-12-30   15.05714  
## 6940 2005-12-31   15.00000
```

arrange

Columns can be arranged in descending order too.

```
chicago <- arrange(chicago, desc(date))  
head(select(chicago, date, pm25tmean2), 3)
```

```
##           date pm25tmean2  
## 1 2005-12-31    15.00000  
## 2 2005-12-30    15.05714  
## 3 2005-12-29     7.45000
```

```
tail(select(chicago, date, pm25tmean2), 3)
```

```
##           date pm25tmean2  
## 6938 1987-01-03        NA  
## 6939 1987-01-02        NA  
## 6940 1987-01-01        NA
```

rename

Renaming a variable in a data frame in R is surprisingly hard to do!

```
head(chicago[, 1:5], 3)
```

```
##   city tmpd dptp          date pm25tmean2
## 1 chic  35 30.1 2005-12-31 15.00000
## 2 chic  36 31.0 2005-12-30 15.05714
## 3 chic  35 29.4 2005-12-29  7.45000
```

```
chicago <- rename(chicago, dewpoint = dptp,
                    pm25 = pm25tmean2)
head(chicago[, 1:5], 3)
```

```
##   city tmpd dewpoint          date      pm25
## 1 chic  35    30.1 2005-12-31 15.00000
## 2 chic  36    31.0 2005-12-30 15.05714
## 3 chic  35    29.4 2005-12-29  7.45000
```

mutate

```
chicago <- mutate(chicago,  
                    pm25detrend=pm25-mean(pm25, na.rm=TRUE))  
head(select(chicago, pm25, pm25detrend))
```

```
##          pm25 pm25detrend  
## 1 15.00000 -1.230958  
## 2 15.05714 -1.173815  
## 3  7.45000 -8.780958  
## 4 17.75000  1.519042  
## 5 23.56000  7.329042  
## 6  8.40000 -7.830958
```

group_by

Generating summary statistics by stratum

```
chicago <- mutate(chicago,
                    tempcat = factor(1 * (tmpd > 80),
                                     labels = c("cold", "hot"))

hotcold <- group_by(chicago, tempcat)
summarize(hotcold, pm25 = mean(pm25, na.rm = TRUE),
          o3 = max(o3tmean2),
          no2 = median(no2tmean2))

## Source: local data frame [3 x 4]

##    tempcat      pm25        o3       no2
## 1   cold  15.97807 66.587500 24.54924
## 2     hot  26.48118 62.969656 24.93870
## 3     NA  47.73750  9.416667 37.44444
```

group_by

Generating summary statistics by stratum

```
chicago <- mutate(chicago,
                    year = as.POSIXlt(date)$year + 1900)
years <- group_by(chicago, year)
summarize(years, pm25 = mean(pm25, na.rm = TRUE),
          o3 = max(o3tmean2, na.rm = TRUE),
          no2 = median(no2tmean2, na.rm = TRUE))
```

```
## Source: local data frame [19 x 4]
```

```
##
```

	year	pm25	o3	no2
## 1	1987	NaN	62.96966	23.49369
## 2	1988	NaN	61.67708	24.52296
## 3	1989	NaN	59.72727	26.14062
## 4	1990	NaN	52.22917	22.59583
## 5	1991	NaN	63.10417	21.38194
## 6	1992	NaN	50.82870	24.78921
## 7	1993	NaN	44.30093	25.76993

%>%

```
chicago %>% mutate(month = as.POSIXlt(date)$mon + 1)
  %>% group_by(month)
  %>% summarize(pm25 = mean(pm25, na.rm = TRUE),
    o3 = max(o3tmean2, na.rm = TRUE),
    no2 = median(no2tmean2, na.rm = TRUE))
```

Source: local data frame [12 x 4]

##

	month	pm25	o3	no2
## 1	1	17.76996	28.22222	25.35417
## 2	2	20.37513	37.37500	26.78034
## 3	3	17.40818	39.05000	26.76984
## 4	4	13.85879	47.94907	25.03125
## 5	5	14.07420	52.75000	24.22222
## 6	6	15.86461	66.58750	25.01140
## 7	7	16.57087	59.54167	22.38442
## 8	8	16.93380	53.96701	22.98333
## 9	9	15.91279	57.48864	24.47917

dplyr

Once you learn the dplyr “grammar” there are a few additional benefits

- ▶ dplyr can work with other data frame “backends”
- ▶ data.table for large fast tables
- ▶ SQL interface for relational databases via the DBI package

Managing Data Frames with dplyr

December 30, 2014

dplyr

The data frame is a key data structure in statistics and in R.

- ▶ There is one observation per row
- ▶ Each column represents a variable or measure or characteristic
- ▶ Primary implementation that you will use is the default R implementation
- ▶ Other implementations, particularly relational databases systems

dplyr

- ▶ Developed by Hadley Wickham of RStudio
- ▶ An optimized and distilled version of `plyr` package (also by Hadley)
- ▶ Does not provide any “new” functionality per se, but **greatly** simplifies existing functionality in R
- ▶ Provides a “grammar” (in particular, verbs) for data manipulation
- ▶ Is **very** fast, as many key operations are coded in C++

dplyr Verbs

- ▶ `select`: return a subset of the columns of a data frame
- ▶ `filter`: extract a subset of rows from a data frame based on logical conditions
- ▶ `arrange`: reorder rows of a data frame
- ▶ `rename`: rename variables in a data frame
- ▶ `mutate`: add new variables/columns or transform existing variables
- ▶ `summarise / summarize`: generate summary statistics of different variables in the data frame, possibly within strata

There is also a handy `print` method that prevents you from printing a lot of data to the console.

dplyr Properties

- ▶ The first argument is a data frame.
- ▶ The subsequent arguments describe what to do with it, and you can refer to columns in the data frame directly without using the \$ operator (just use the names).
- ▶ The result is a new data frame
- ▶ Data frames must be properly formatted and annotated for this to all be useful