For the first part of this presentation see:
http://www.peteraldhous.com/CAR/Aldhous_CAR2011_RforStats.pdf
This presentation is available at:
http://jacobfenton.s3.amazonaws.com/automate-r.pdf

R for statistics: automate your analysis

Jacob Fenton
CAR Director
Investigative Reporting Workshop,
American University

What's this about?

- R does great things with stats and graphics. But sometimes you don't have 1 dataset—you have 2,000. Must feed the newsapps beast.
- R is itself a programming language. You could just write a loop, of course.
- Data management across platforms is hard.
- 'Reproducibility' is key: editors want to rerun month-long analysis with different assumptions. The day before deadline. Dude, where's my temp2.txt file?
- Integrating R with the rest of your data 'system' may be easier if you reuse the same tools: databases to keep data and scripting languages to 'do' stuff . That way you don't have to remember R's for loop* syntax!

  * for (x in c(1:10)) print((x))

Disclaimer

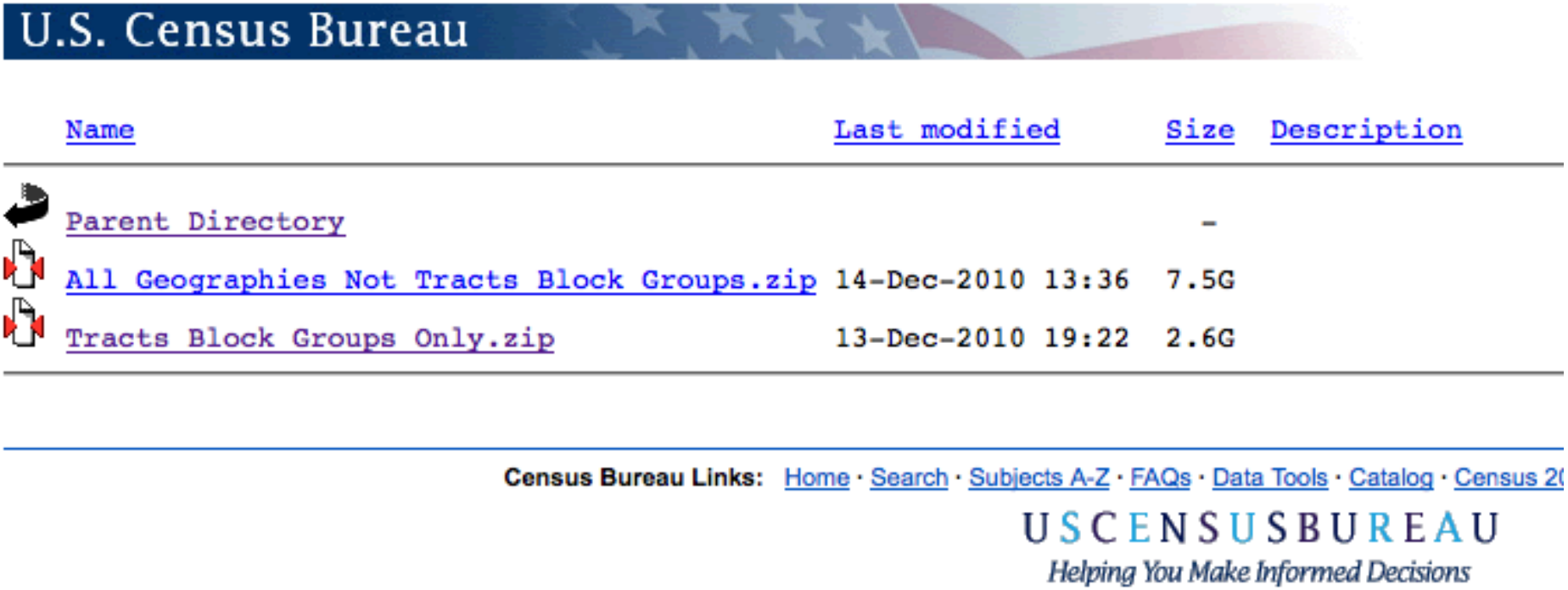
- Remember Peter's slide about running with scissors! This isn't an example of wise use of statistics—it's an example of integrating R with a complex dataset I cooked up for this talk.
- I'm gonna whip through an example process to get to the automation.

Danger: 95% confidence = 5% wrong

- Running enough a test at the 95% confidence interval will, about 5% of the time, produce ‘wrong’ answers.
- Consult an expert.

Example: ACS 5-year tract estimates

- Complex. But available in handy 2.6 Gig zipfile via ftp: http://www2.census.gov/acs2009_5yr/summaryfile/ !



Tracts: A few educational attainment tables

- B15002: SEX BY EDUCATIONAL ATTAINMENT FOR THE POPULATION 25 YEARS AND OVER --- Universe: Population 25 years and over
- B20004: MEDIAN EARNINGS IN THE PAST 12 MONTHS (IN 2009 INFLATION-ADJUSTED DOLLARS) BY SEX BY EDUCATIONAL ATTAINMENT FOR THE POPULATION 25 YEARS AND OVER --- Universe: Population 25 years and over with earnings
- Note the slight universe mismatch. Sigh.

Process Slide: Pull data, import to R

- My process is pull data into a database from raw summary files, muck with it in sql, then dump it to a text file: copy nicar_demo to ‘/nicar_demo.txt' with text delimiter = e'\t' null as '';
- I’m using a suite of tools for extracting ACS census data from the summary files directly into a database
- For this example, I computed tract-wise rates for residents with less than a hs diploma or a ba or higher, for both men and women, as well as the ratio of men to women, and the ratio of men’s median income to women’s. It’s not the best stat, but it works for a demo.

Quickie Slide: import to R, add names

It's good to confirm we imported the number of rows we expected—it's easy to do so with nrow()

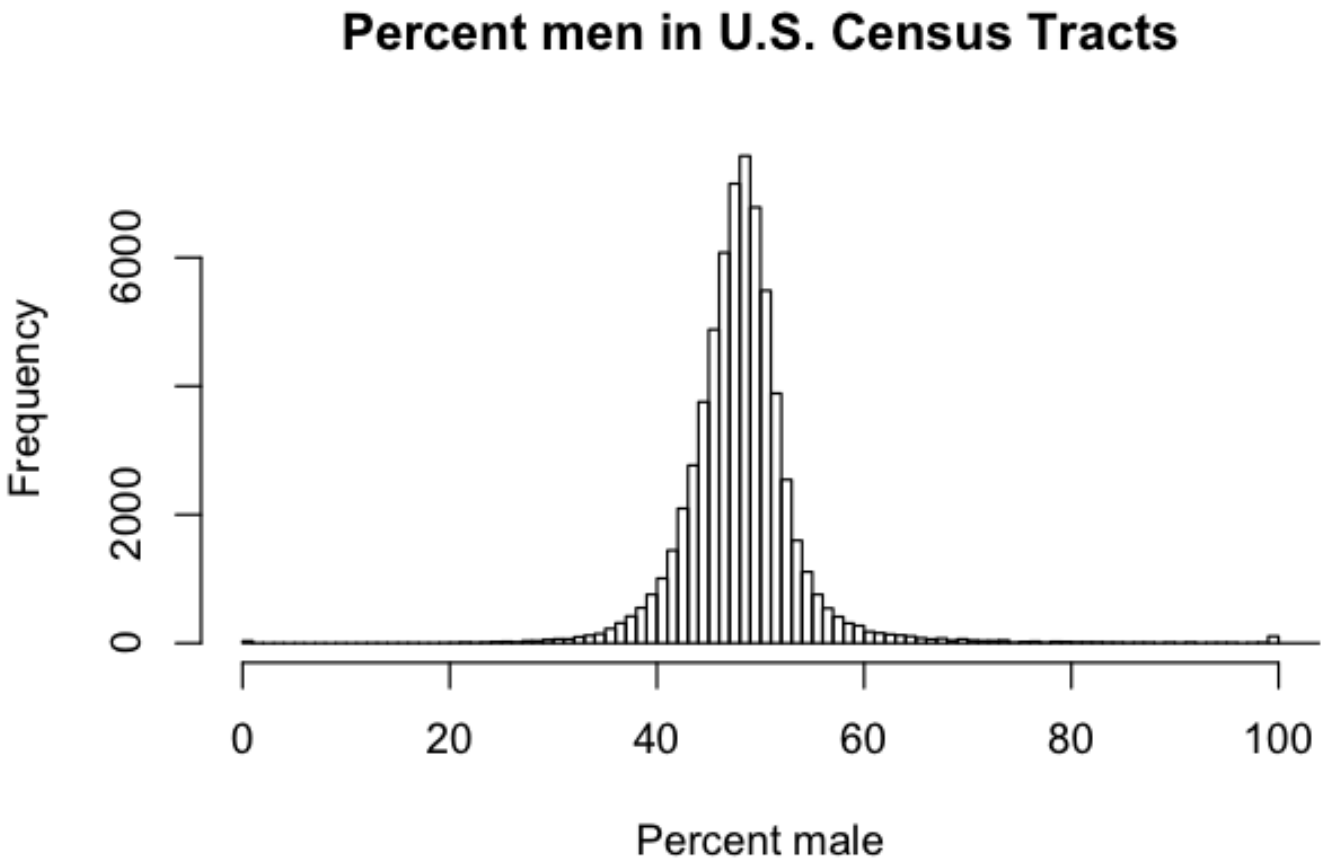
```
> a <- read.table('/Users/jacobfenton/IRW/django_sites/1.1.1/
broadband/workshop/acs_09_5yr/formatted/nicar_demo.txt',
header=FALSE, sep="\t", quote='');
> nrow(a)
[1] 65461
> names(a) <- c('geo_name', 'state', 'county', 'total',
'less_than_hs_grad_rate', 'less_than_hs_grad_rate_f',
'less_than_hs_grad_rate_m', 'ba_plus_rate', 'ba_plus_f_rate',
'ba_plus_m_rate', 'fraction_male', 'mi', 'mi_lthsg', 'mi_male',
'mi_female', 'mi_fmratio')
```

Summarize the data

- Summary() will summarize the data.
- “Variance is interesting” – Phil Meyer

```
 ba_plus_rate      ba_plus_f_rate     ba_plus_m_rate     fraction_male          mi        
Min.   :  0.0000   Min.   :  0.0000   Min.   :  0.0000   Min.   :  0.0000   Min.   :  2499  
1st Qu.:  0.1254   1st Qu.:  0.1289   1st Qu.:  0.1183   1st Qu.:  0.4547   1st Qu.: 26246  
Median :  0.2086   Median :  0.2097   Median :  0.2104   Median :  0.4804   Median : 31990  
Mean   :  0.2600   Mean   :  0.2548   Mean   :  0.2670   Mean   :  0.4814   Mean   : 34791  
3rd Qu.:  0.3544   3rd Qu.:  0.3445   3rd Qu.:  0.3721   3rd Qu.:  0.5039   3rd Qu.: 40708  
Max.   :  1.0000   Max.   :  1.0000   Max.   :  1.0000   Max.   :  1.0000   Max.   :198650  
NA's   :398.0000   NA's   :487.0000   NA's   :430.0000   NA's   :398.0000   NA's   :   522
```

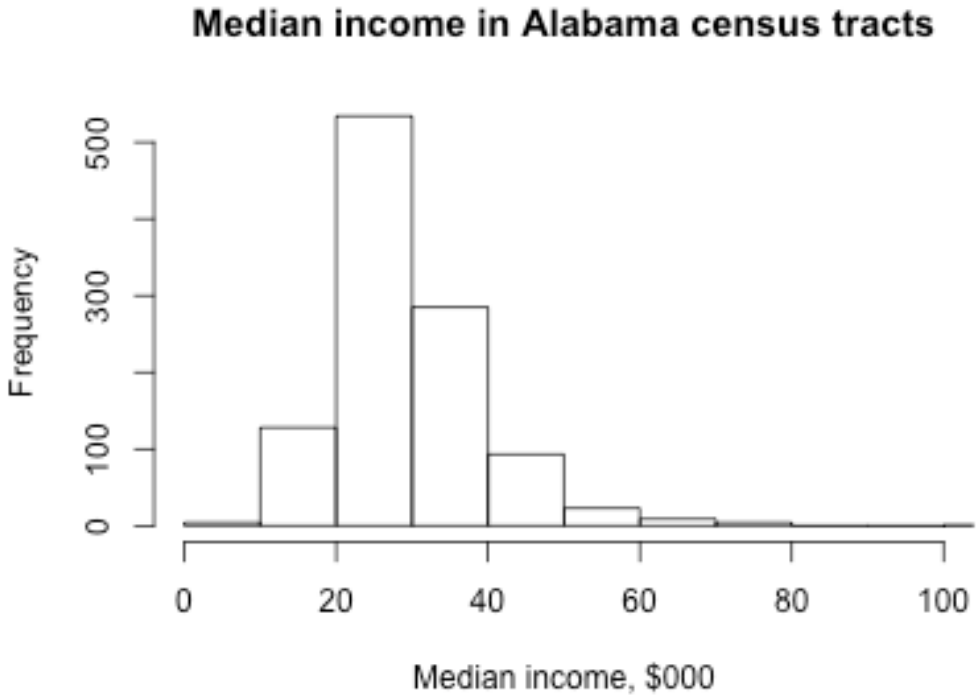
Tight normal distribution—not as interesting



A simple R graph

```
> Alabama <- subset(a,a$state=="1")
> nrow(Alabama)
[1] 1082
> hist(Alabama$mi/1000,breaks=c(10*0:10,1000), xlim=c(0,100), freq=TRUE,
xlab="Median income, $000", main="Median income in Alabama Census
Tracts")
```

For more complex graphics, try using ggplot2 and or the site Peter mentioned



Print charts en masse

- There are many ways to do this, but as an example I'm scripting the process with a python library called RPy2 and a script called write_graphs.py. To make the script work, you need to set file paths correctly on your own machine. Python makes adding state names much less painful.

Test data (large file!):
http://jacobfenton.s3.amazonaws.com/nicar-raleigh/nicar_demo.txt

Script:
http://jacobfenton.s3.amazonaws.com/nicar-raleigh/write_graphs.py

Sample output:
http://jacobfenton.s3.amazonaws.com/nicar-raleigh/graphs/NC.png

Disclosure: python

- I'm working with python's RPy2 library. Easy_install rpy2.
- It appears there's similar ruby code at rsruby: http://rubyforge.org/projects/rsruby/ . No idea if it works.

Simple python R interaction

```
import rpy2.robjects as robjects
##Write R code in python, and execute it in R.
rcode0="""
a <- read.table('nicar_demo.txt', header=FALSE,
   sep="\t", quote='');
"""
robjects.r(rcode0)
```

Correlation Matrix

- Transform the data into a matrix (turns out it's been a data frame this whole time…)

```
b=data.matrix(a)
Write the correlation table out
write.table(cor(b, use="complete.obs"),
    file="correlations.txt", sep="|", eol="\n",
    row.names=TRUE)
```

Correlation matrix excerpt

- Here's a few selected columns.

	<hs_grad_r	ba_+_r	frac_male	mi	mi_lthsg	mi_male	mi_female	mi_fmratio
<_hs_grad_r	1.00	-0.65	0.06	-0.62	-0.21	-0.60	-0.55	0.17
<_hs_grad_r_f	0.96	-0.61	0.00	-0.58	-0.22	-0.55	-0.53	0.13
<_hs_grad_r_m	0.80	-0.53	-0.33	-0.51	-0.20	-0.47	-0.46	0.10
ba_+_r	-0.65	1.00	0.01	0.78	0.23	0.74	0.73	-0.10
ba_+_f_r	-0.63	0.97	0.05	0.76	0.22	0.70	0.73	-0.05
ba_+_m_r	-0.64	0.98	-0.02	0.77	0.23	0.74	0.69	-0.13
frac_male	0.06	0.01	1.00	-0.01	0.00	-0.05	-0.01	0.11
mi	-0.62	0.78	-0.01	1.00	0.36	0.94	0.86	-0.20
mi_lthsg	-0.21	0.23	0.00	0.36	1.00	0.34	0.30	-0.10
mi_male	-0.60	0.74	-0.05	0.94	0.34	1.00	0.70	-0.38
mi_female	-0.55	0.73	-0.01	0.86	0.30	0.70	1.00	0.16
mi_fmratio	0.17	-0.10	0.11	-0.20	-0.10	-0.38	0.16	1.00

- Mi_fmratio is the median income of women divided by the median income of men. Frac_male is the fraction of men in a tract.

Test correlation uncertainty

```
> cor.test(a$mi, a$mi_fmratio, use="complete.obs")

    Pearson's product-moment correlation

data:  a$mi and a$mi_fmratio
t = -53.9061, df = 64655, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.2147560 -0.2000030
sample estimates:
      cor
-0.2073913
```

Run it by state

- We can pass data back from R to python too:

```
rcode3 = """cor.test(t1$mi, t1$mi_fmratio,
    use="complete.obs")"""
   result = robjects.r(rcode3)
```

This will allow us to compute correlation coefficients by state as well as their uncertainties*, and dump them to a text file.

See example script:
http://jacobfenton.s3.amazonaws.com/nicar-raleigh/write_state_correlations.py

* We left out initial uncertainties, so this isn't super meaningful…

Script results: meaningful?

- Is this meaningful?
- Also, we left out uncertainties at the outset.
- http://jacobfenton.s3.amazonaws.com/nicar-raleigh/graphs/state_correlations.txt

A	B	C	D
##state by state correlation of census tracts			
state	correlation	correlation_mi	correlation_max
New Hampshir	-0.53	-0.61	-0.44
Rhode Island	-0.40	-0.50	-0.29
Utah	-0.39	-0.46	-0.31
Massachusetts	-0.39	-0.43	-0.34
Nevada	-0.38	-0.46	-0.30
Washington, D	-0.37	-0.49	-0.23
Connecticut	-0.36	-0.42	-0.30
Alaska	-0.35	-0.48	-0.21
Michigan	-0.33	-0.37	-0.30
Maryland	-0.32	-0.37	-0.27
New Jersey	-0.31	-0.35	-0.27
Florida	-0.31	-0.34	-0.28
Washington	-0.30	-0.34	-0.24
New Mexico	-0.29	-0.38	-0.20
Wisconsin	-0.29	-0.34	-0.24
Indiana	-0.29	-0.34	-0.24
Tennessee	-0.29	-0.34	-0.24
Hawaii	-0.28	-0.39	-0.17
Ohio	-0.28	-0.31	-0.24
Oregon	-0.28	-0.34	-0.21
North Dakota	-0.27	-0.39	-0.15
South Carolina	-0.27	-0.33	-0.21
Alabama	-0.26	-0.32	-0.21
Pennsylvania	-0.26	-0.29	-0.23
Missouri	-0.26	-0.31	-0.21
Oklahoma	-0.25	-0.31	-0.19
Delaware	-0.25	-0.38	-0.12
Wyoming	-0.25	-0.40	-0.08
Texas	-0.25	-0.27	-0.22
New York	-0.24	-0.27	-0.22
South Dakota	-0.24	-0.36	-0.11
Arkansas	-0.23	-0.30	-0.15

But wait, there's more

- This is just a very simple demo of RPy2. There's a lot more complex interactions available. It's possible to load data from a script directly. That's less useful for large data sets, but helpful in other contexts. For instance, train an algorithm on a sample data set, then feed data to it by script for classification. Etc.

Appendix (added after talk)

- R libraries that talk directly to databases!
- For mysql see: RmySQL: http://cran.r-project.org/web/packages/RMySQL/index.html
- For postgresql see RpgSQL: http://cran.r-project.org/web/packages/RpgSQL/index.html
- There's also RpostgreSQL, but as of this writing no binaries were known to be available…

Peter's email about mysql

- Hi all,
- This issue came up in the Q&A.
- Here's the documentation for the RMySQL package that allows you to connect R to a MySQL database: http://cran.r-project.org/web/packages/RMySQL/RMySQL.pdf
- And here's the equivalent for the RPostgreSQL package: http://cran.r-project.org/web/packages/RPostgreSQL/RPostgreSQL.pdf
- I've experimented with the former, and some sample code is below. Using MySQL Server 5.1, I had to replace the **libmysql.dll** file with an older version to sidestep an error message, but once I'd done this everything seemed to work OK. Some notes on this glitch here: http://tolstoy.newcastle.edu.au/R/e2/help/07/10/28442.html and here: http://www.stat.berkeley.edu/users/spector/s133/RMySQL_windows.html).
- Sample code (entries in square brackets need to be replaced with specific code for your database and queries, minus the brackets):
- **# Make the connection to the MySQL server**
- con <- dbConnect(MySQL(), user="[user]", password="[password]", dbname="[databasename]")
- **# show available databases**
- dbGetQuery(con, "SHOW DATABASES")
- **# select database**
- dbGetQuery(con, "USE [databasename]")
- **# show the tables in the database**
- dbGetQuery(con, "SHOW TABLES")
- **OR**
- dbListTables(con)
- **# List fields in table**
- dbListFields(con, "[tablename]")
- **# Run query**
- dbGetQuery(con, "[SQL query]")
- **# Close connection**
- dbDisconnect(con)
- This came up during a discussion of the main limitation of R: the fact that the entire session is stored in RAM, which can lead to memory problems when dealing with large datasets. Pulling in data from a database as required and removing objects from the session as soon as you're done with them is one approach to dealing with this limitation.
- If you're still hitting problems with R's memory barrier, you might want to experiment with commercial software developed to address the "big data" problem by Revolution Analytics, see: http://www.revolutionanalytics.com/products/enterprise-big-data.php.
- Cheers,
- Peter

Jacob's email about PostgreSQL

- FWIW, on Mac OS X with PostgreSQL I found it much easier to install RpgSQL (http://cran.r-project.org/web/packages/RpgSQL/index.html) rather than RPostgreSQL, because there are binaries available, so you don't have to build from source. According to the CRAN page there are no RPostgreSQL binaries available for windows either, but they may just be elsewhere...

  The process of installing RpgSQL is a bit annoying because it depends on the java postgres driver, but it turns out making that work is as simple as putting the .jar file (available here: http://jdbc.postgresql.org/download.html) in a place where R is looking for it...