THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

Complex sampling and R.

Thomas Lumley

Statistics
University of Auckland

JSM — San Diego — 2011–7–29

Outline

1. In which we review basic concepts and attempt to match terminology
2. In which we meet R and describe surveys to it.
3. Mid-morning break: in which refreshment is supplied
4. In which we make pictures
5. Lunch break: in which we are sent out to forage
6. In which lines are drawn
7. Mid-afternoon break: in which refreshment is supplied
8. In which adjustments are made

Outline: Session 1

In which we review basic concepts and attempt to match terminology — survey design features: strata, clusters, unequal probabilities — variance estimation for totals: Horvitz-Thompson estimator, replicate weights — variance estimation for complex statistics: influence functions, replicate weights

In which we meet R and describe surveys to it the 'survey' package and multistage stratified samples — the 'survey' package and replicate weights — connecting to a relational database — some basic descriptive statistics

Outline: Session 2

In which we make pictures — visualizing categorical data with graphs of estimated population tables — boxplots and histograms — scatterplots for large, weighted data sets: bubble plots, hexagonal binning, transparency — scatterplot smoothers and density estimators — conditioning plots. — biplots for principal components.

Outline: Session 3

In which lines are drawn — comparisons between model-based and design-based regression models: assumptions, goals, generalizability — sampling-weighted least squares as a criterion — when is it safe to ignore weights? Is it ever preferable to ignore weights? — worked example from NHANES — worked examples of other forms of regression for survey data – generalized linear models, in particular, logistic regression

Outline: Session 4

In which adjustments are made - regression models for prediction lead to the regression estimator of a total — calibration allows the same process to be carried out by adjusting weights — how to do calibration when the goal of inference is regression coefficients — application to two-phase subsampling of existing cohorts, with Cox proportional hazards model

Complex sampling

To reduce cost, most large surveys do not use iid sampling

- Stratified sampling: fix the number sampled from a population stratum, to make the sample more representative
- Cluster sampling: recruit participants in clusters, eg, to reduce travel time between interviews
- Unequal probability: oversample important subgroups, eg, ethnic minorities, high-poverty neighbourhoods,...

Goal is inference about the finite population or the process that created it, not about the sampling process.

Basic estimation ideas

Individuals are sampled with known probabilities π_i from a population of size N to end up with a sample of size n. The 'population' can be a full population or an existing sample such as a cohort.

We write $R_i = 1$ if individual i is sampled, $R_i = 0$ otherwise

The design-based inference problem is to estimate what any statistic of interest would be if data from the whole population were available.

For most of today we will pretend that π_i really is known, ignoring non-response.

Basic estimation ideas

For a population total this is easy: an unbiased estimator of

$$T_X = \sum_{i=1}^{N} x_i$$

is the Horvitz–Thompson estimator

$$\hat{T}_X = \sum_{i:R_i=1} \frac{1}{\pi_i} X_i$$

Standard errors follow from formulas for the variance of a sum: main complication is that we do need to know $\text{cov}[R_i, R_j]$, which depends on the pairwise sampling probabilities π_{ij}

Basic estimation ideas

Other statistics follow from sums: if the statistic on the whole population would solve the estimating equation

$$\bar{U}(\theta) = \sum_{i=1}^{N} U_i(\theta) = 0$$

then a design-based estimate will solve

$$\widehat{\bar{U}(\theta)} = \sum_{i:R_i=1} \frac{1}{\pi_i} U_i(\theta) = 0$$

If the statistic on the whole population would maximize

$$\ell(\theta) = \sum_{i=1}^{N} \ell_i(\theta)$$

then a design-based estimate will maximize

$$\widehat{\ell(\theta)} = \sum_{i:R_i=1} \frac{1}{\pi_i} \ell_i(\theta)$$

Standard errors

Standard errors for other statistics come from the delta method or from resampling.

$$\begin{aligned}\text{var}[\hat{\theta}] &\approx \left(\frac{\partial\hat{\theta}}{\partial\hat{\tilde{U}}(\theta)}\right)^T \text{var}\left[\hat{\tilde{U}}(\theta)\right]\left(\frac{\partial\hat{\theta}}{\partial\hat{\tilde{U}}(\theta)}\right)\\ &\approx \left(\frac{\partial\hat{\tilde{U}}(\theta)}{\partial\theta}\right)^{-1}\Bigg|_{\theta=\hat{\theta}} \text{var}\left[\hat{\tilde{U}}(\hat{\theta})\right]\Big|_{\theta=\hat{\theta}} \left(\frac{\partial\hat{\tilde{U}}(\theta)}{\partial\theta}\right)^{-1}\Bigg|_{\theta=\hat{\theta}}\end{aligned}$$

where the middle term comes from the variance of a total, and the outer terms are typically byproducts of the optimisation process.

Standard errors

Linearisation estimators are very similar to 'sandwich' or 'HAC' estimators for regression models.

They can also be interpreted nicely with influence functions.

In the population, for least-squares regression

$$\hat{\beta} - \beta = \sum_{i=1}^{N} (X^T X)^{-1} x_i (y_i - \mu_i)$$

so $\hat{\beta}$ is approximately a population total of influence functions.

Applying the formula for variance of a sum to the influence functions gives the linearization estimator.

Resampling (replicate weights)

A simple bootstrap or jackknife won't work because the observations are not exchangable, but similar things do work.

- BRR: split the data into two independent halves, compute $(\widehat{\theta}_A - \widehat{\theta}_B)^2$, average over lots of splits.
- cluster jackknife: leave out one cluster (primary sampling unit) at a time
- cluster bootstrap: Sample $m-1$ from m PSUs with replacement (Rao & Wu)
- multistage bootstrap: works with large sampling fractions at multiple stages (Preston)

Resampling (replicate weights)

All the resampling estimators can be written as

$$K \sum_{i=1}^{M} r_i(\widehat{\theta}_i - \widehat{\theta})^2$$

or

$$K \sum_{i=1}^{M} r_i(\widehat{\theta}_i - \overline{\widehat{\theta}})^2$$

The choice of centering doesn't matter asymptotically or for totals. R defaults to the second version, but can optionally use the first.

[The jackknife estimators don't work for quantiles and similar non-smooth statistics, but bootstrap and BRR do]

Quantiles

Quantiles are tricky

- Too many definitions
- Not a smooth estimating function

We use Woodruff's method: estimate a confidence interval for $P(Y >$ quantile) and transform into a confidence interval for the quantile. Divide length by 2×1.96 to get SE estimate.

We use two definitions of quantile: interpolation between order statistics, and interpolation between distinct order statistics (?matches SUDAAN).

Domains

A domain or subpopulation (eg women) typically has random sample size; not correct simply to subset the data.

Domain total:

$$\sum_{i:R_i=1} \frac{1}{\pi_i} D_i y_i$$

where D_i is the indicator that observation i is in the domain

Similarly, for estimating functions, solve

$$\sum_{i:R_i=1} \frac{1}{\pi_i} D_i U_i(\theta) = 0$$

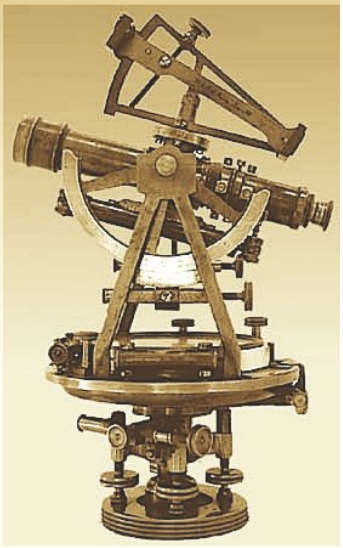
Some ambiguity with constraints: eg in binomial regression with log link, should the constraint $\exp(x_i\beta) \leq 1$ be enforced for i not in the domain?

The survey package

← →

R Survey package

Survey analysis in R



This is the homepage for the "survey" package, which provides facilities in R for analyzing data from complex surveys. The current version is 3.24. A much earlier version (2.2) was published in Journal of Statistical Software

A port of an older version of the package (version 3.6-8) to S-PLUS 8.0 is available from CSAN (thanks to Patrick Aboyoun at Insightful).

Features:

- Means, totals, ratios, quantiles, contingency tables, regression models, loglinear models, survival curves, for the whole sample and for domains.
- Variances by Taylor linearization or by replicate weights (BRR, jackknife, bootstrap, or user-supplied)
- Multistage sampling with or without replacement.
- PPS sampling with or without replacement: Horvitz-Thompson and Yates-Grundy estimators and a range of approximations.
- Post-stratification, generalized raking/calibration, GREG estimation, trimming of weights.
- Two-phase designs. Estimated weights for augmented IPW estimators.
- Graphics
- Support for using multiply imputed data
- Database-backed design objects for large data sets (now with replicate weights, too)
- Experimental support for parallel processing on multicore computers.
- Multivariate analysis: principal components, factor analysis (experimental).
- Likelihood ratio (Rao-Scott) tests for glms, Cox models, loglinear models.

The NEWS file gives a history of features and bug fixes.

Comparison shopping:
Alan Zaslavsky keeps a comprehensive list of survey analysis software for the ASA Section on Survey Research Methods.

User-generated ratings and reviews of this package (and others) at crantastic.

http://faculty.washington.edu/tlumley/survey/

← →

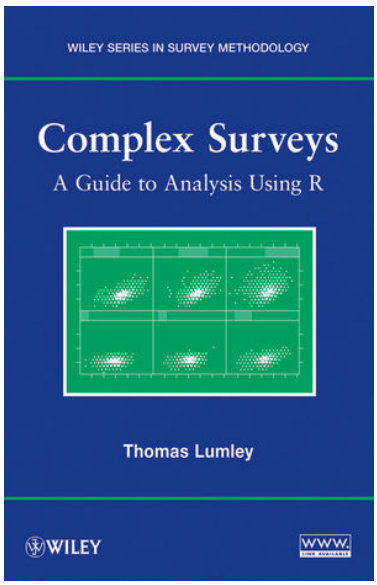
R Survey package

Version 3.28-2 is current, containing approximately 12000 lines of interpreted R code.

Version 2.3 was published in Journal of Statistical Software.

Major changes since then: finite population corrections for multistage sampling and PPS sampling, calibration and generalized raking, tests of independence in contingency tables, better tables of results, simple two-phase designs, loglinear models, ordinal regression models, Rao-Scott likelihood ratio tests for regression models, survival curves, more resampling methods, database-backed designs for large surveys

The book of the package



Design principles

- Ease of maintenance and debugging by code reuse
- Speed and memory use not initially a priority: don't optimize until there are real use-cases to profile.
- Rapid release, so that bugs and other infelicities can be found and fixed.
- Emphasize features that look like biostatistics (regression, calibration, survival analysis, exploratory graphics)

Intended market

- Methods research (because of programming features of R)
- Teaching (because of cost, use of R in other courses)
- Secondary analysis of national surveys (regression features, R is familiar to non-survey statisticians)
- Two-phase designs in epidemiology

Objects and Formulas

Collections of related information should be kept together in an object.

For surveys this means the data and the survey meta-data.

Instead of the `data=` argument in most non-survey R functions there is a `design=` argument

The way to specify variables from a data frame or object in R is a formula

```
~a + b + I(c < 5*d)
```

The survey package always uses formulas to specify variables in a survey data set.

Describing surveys to R

- `svydesign()` creates a survey design object from design variables
- `svrepdesign()` creates a survey design object from replicate weights
- `twophase()` creates a two-phase design object from design variables

and `as.svrepdesign()` creates replicate weights for an existing design object

Describing surveys to R

Stratified independent sample (without replacement) of California schools

```
data(api)
dstrat <- svydesign(id=~1,strata=~stype, weights=~pw,
     data=apistrat, fpc=~fpc)
```

- `stype` is a factor variable for elementary/middle/high school
- `fpc` is a numeric variable giving the number of schools in each stratum. If omitted we assume sampling with replacement
- `id=~1` specifies independent sampling.
- `apistrat` is the data frame with all the data.
- `pw` contains sampling weights ($1/\pi_i$). These could be omitted since they can be computed from the population size.

Note that all the variables are in `apistrat` and are specified as formulas.

Describing surveys to R

```
> dstrat
Stratified Independent Sampling design
svydesign(id = ~1, strata = ~stype, weights = ~pw, data = apistrat,
fpc = ~fpc)
> summary(dstrat)
Stratified Independent Sampling design
svydesign(id = ~1, strata = ~stype, weights = ~pw, data = apistrat,
fpc = ~fpc)
Probabilities:
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.02262 0.02262 0.03587 0.04014 0.05339 0.06623
Stratum Sizes:
E H M
obs 100 50 50
design.PSU 100 50 50
actual.PSU 100 50 50
Population stratum sizes (PSUs):
E M H
4421 1018 755
Data variables:
[1] "cds" "stype" "name" "sname" "snum" "dname"
[7] "dnum" "cname" "cnum" "flag" "pcttest" "api00"
...
```

Describing surveys to R

Cluster sample of school districts, using all schools within a district.

```
dclus1 <- svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
```

- `dnum` is a (numeric) identifier for school district
- No stratification, so no `strata=` argument

```
> summary(dclus1)
1 - level Cluster Sampling design
With (15) clusters.
svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)
Probabilities:
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.02954 0.02954 0.02954 0.02954 0.02954 0.02954
Population size (PSUs): 757
Data variables:
[1] "cds" "stype" "name" "sname" "snum" "dname"
[7] "dnum" "cname" "cnum" "flag" "pcttest" "api00"
...
```


Describing surveys to R

Two-stage sample: 40 school districts and up to 5 schools from each

```
dclus2 <- svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)
```

- `dnum` identifies school district, `snum` identifies school
- `fpc1` is the number of school districts in population,`fpc2` is number of schools in the district.
- Weights are computed from `fpc1` and `fpc2`

```
> summary(dclus2)
2 - level Cluster Sampling design
With (40, 126) clusters.
svydesign(id = ~dnum + snum, fpc = ~fpc1 + fpc2, data = apiclus2)
Probabilities:
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.003669 0.037740 0.052840 0.042390 0.052840 0.052840
Population size (PSUs): 757
Data variables:
[1] "cds" "stype" "name" "sname" "snum" "dname"
[7] "dnum" "cname" "cnum" "flag" "pcttest" "api00"
...
```

Replicate weights

California Health Interview Survey has 80 sets of replicate weights instead of design information

```
chis_adult <- read.dta("adult.dta")
chis <- svrepdesign(variables=chis_adult[,1:418],
     repweights=chis_adult[,420:499],
     weights=chis_adult[,419], combined.weights=TRUE,
     type="other", scale=1, rscales=1)
```

`scale` and `rscales` are the K and r_i from page 13

Can also specify, eg,`repweights="rakedw[1-9]"` for all variables beginning `raked` followed by a non-zero digit

Replicate weights

For JK1, JKn, BRR, and bootstrap, R knows the correct K and r_i.

Otherwise use `type="other"` and specify them.

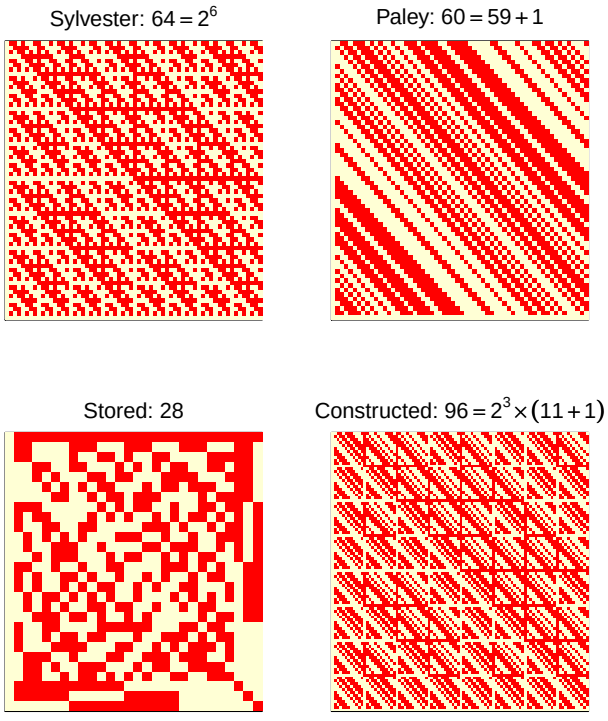
Help page `?svrepdesign` has notes for JK2 and for ACS data.

Replicate weights

Can also create replicate weights with `as.svrepdesign()`, using jackknife, three types of bootstrap, or BRR.

```
## one-stage cluster sample
dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
## convert to JK1 jackknife
rclus1<-as.svrepdesign(dclus1)
## convert to bootstrap
bclus1<-as.svrepdesign(dclus1, type="subbootstrap", replicates=100)

## two-stage sample with multistage bootstrap
dclus2<-svydesign(id = ~dnum + snum, fpc = ~fpc1 + fpc2,
              data = apiclus2)
mrbclus2<-as.svrepdesign(dclus2, type="mrb",replicates=100)
```


BRR: R knows Hadamard matrices



Describing surveys: database-based

```
library(RSQLite)
brfss <- svydesign(id=~X_PSU, strata=~X_STATE, weight=~X_FINALWT,
  data="brfss",  dbtype="SQLite", dbname="brfss07.db",
  nest=TRUE)
```

The `data` argument is the name of a database table or view, `dbtype` and `dbname` specify the database.

Only the design meta-data are loaded into the design object. Other variables are temporarily loaded as needed when an analysis is run.

Can use any database with an ODBC, JDBC, or R-DBI interface: anything on Windows, SQLite, Postgres, Oracle.

BRFSS 2007 is about as large as a 1Gb laptop can handle with this approach: 430,000 records.

Summary statistics

`svymean`, `svytotal`, `svyratio`, `svyvar`, `svyquantile`

All take a formula and design object as arguments, return an object with `coef`, `vcov`, `SE`, `cv` methods.

Mean and total on factor variables give tables of cell means/totals.

Mean and total have `deff` argument for design effects and the returned object has a `deff` method.

```
> svymean(~api00, dclus1, deff=TRUE)
mean SE DEff
api00 644.169 23.542 9.3459
> svymean(~factor(stype),dclus1)
mean SE
factor(stype)E 0.786885 0.0463
factor(stype)H 0.076503 0.0268
factor(stype)M 0.136612 0.0296
```

Summary statistics

```
> svymean(~interaction(stype, comp.imp), dclus1)
mean SE
interaction(stype, comp.imp)E.No 0.174863 0.0260
interaction(stype, comp.imp)H.No 0.038251 0.0161
interaction(stype, comp.imp)M.No 0.060109 0.0246
interaction(stype, comp.imp)E.Yes 0.612022 0.0417
interaction(stype, comp.imp)H.Yes 0.038251 0.0161
interaction(stype, comp.imp)M.Yes 0.076503 0.0217
> svyvar(~api00, dclus1)
variance SE
api00 11183 1386.4
> svytotal(~enroll, dclus1, deff=TRUE)
total SE DEff
enroll 3404940 932235 31.311
```


Summary statistics

```
> mns <- svymean(~api00+api99,dclus1)
> mns
       mean    SE
api00 644.17 23.542
api99 606.98 24.225
> coef(mns)
 api00     api99
644.1694 606.9781
> SE(mns)
 api00      api99
23.54224 24.22504
> vcov(mns)
      api00     api99
api00 554.2371 565.7856
api99 565.7856 586.8526
> cv(mns)
api00        api99
0.03654666 0.03991090
```

Domain estimation

The correct standard error estimate for a subpopulation that isnt a stratum is not just obtained by pretending that the subpopulation was a designed survey of its own.

However, the `subset` function and `"["` method for survey design objects handle all these details automagically, so you can ignore this problem.

The package test suite (`tests/domain.R`) verifies that subpopulation means agree with derivations from ratio estimators and regression estimator derivations. Some more documentation is in the domain vignette.

Note: subsets of design objects don't necessarily use less memory than the whole objects.

Prettier tables

Two main types

- totals or proportions cross-classified by multiple factors
- arbitrary statistics in subgroups

Computing over subgroups

`svyby` computes a statistic for subgroups specified by a set of factor variables:

```
> svyby(~api99, ~stype, dclus1, svymean)
stype statistics.api99 se.api99
E E 607.7917 22.81660
H H 595.7143 41.76400
M M 608.6000 32.56064
```

`~api99` is the variable to be analysed, `~stype` is the subgroup variable, `dclus1` is the design object, `svymean` is the statistic to compute.

Lots of options for eg what variance summaries to present

Computing over subgroups

```
> svyby(~api99, ~stype, dclus1, svyquantile, quantiles=0.5,ci=TRUE)
  stype statistics.quantiles     statistics.CIs       se      var
E     E                  615 525.6174, 674.1479 37.89113 1435.738
H     H                  593 428.4810, 701.0065 69.52309  4833.46
M     M                  611 527.5797, 675.2395 37.66903 1418.955

> svyby(~api99, list(school.type=apiclus1$stype), dclus1, svymean)
  school.type statistics.api99 se.api99
E           E         607.7917 22.81660
H           H         595.7143 41.76400
M           M         608.6000 32.56064

> svyby(~api99+api00, ~stype, dclus1, svymean, deff=TRUE)
> svyby(~api99+api00, ~stype, dclus1, svymean, deff=TRUE)
  stype statistics.api99 statistics.api00 se.api99 se.api00
E     E         607.7917         648.8681 22.81660 22.36241
H     H         595.7143         618.5714 41.76400 38.02025
M     M         608.6000         631.4400 32.56064 31.60947
  DEff.api99 DEff.api00
E   5.895734   6.583674
H   2.211866   2.228259
M   2.226990   2.163900
```

Computing over subgroups

```
      stype sch.wide statistic.api99 statistic.api00
E.No      E       No        601.6667        596.3333
H.No      H       No        662.0000        659.3333
M.No      M       No        611.3750        606.3750
E.Yes     E      Yes        608.3485        653.6439
H.Yes     H      Yes        577.6364        607.4545
M.Yes     M      Yes        607.2941        643.2353
```


Computing over subgroups

```
> (a<-svyby(~enroll, ~stype, rclus1, svytotal, deff=TRUE,
+ vartype=c("se","cv","cvpct","var")))
  stype statistics.enroll       se cv.enroll cv%.enroll          var       DEff
E     E         2109717.1 631349.4 0.2992578   29.92578 398602047550 125.039075
H     H          535594.9 226716.6 0.4232987   42.32987  51400414315   4.645816
M     M          759628.1 213635.5 0.2812369   28.12369  45640120138  13.014932

> deff(a)
[1] 125.039075 4.645816 13.014932
> SE(a)
[1] 631349.4 226716.6 213635.5
> cv(a)
[1] 0.2992578 0.4232987 0.2812369
> coef(a)
[1] 2109717.1 535594.9 759628.1

> svyby(~api00,~comp.imp+sch.wide,design=dclus1,svymean,
+ drop.empty.groups=FALSE)
        comp.imp sch.wide statistics.api00 se.api00
No.No         No       No         608.0435 28.98769
Yes.No       Yes       No               NA       NA
No.Yes        No      Yes         654.0741 32.66871
Yes.Yes      Yes      Yes         648.4060 22.47502
```

Functions of estimates

svycontrast computes linear and nonlinear combinations of estimated statistics (in the same object).

```
> a <- svytotal(~api00 + enroll + api99, dclus1)
> svycontrast(a, list(avg = c(0.5, 0, 0.5), diff = c(1,
0, -1)))
     contrast     SE
avg   3874804 873276
diff   230363  54921
> svycontrast(a, list(avg = c(api00 = 0.5, api99 = 0.5),
diff = c(api00 = 1, api99 = -1)))
     contrast     SE
avg   3874804 873276
diff   230363  54921
```


Functions of estimates

```
> svycontrast(a, quote(api00/api99))
          nlcon     SE
contrast 1.0613 0.0062
> svyratio(~api00, ~api99, dclus1)
Ratio estimator: svyratio.survey.design2(~api00, ~api99, dclus1)
Ratios=
         api99
api00 1.061273
SEs=
            api99
api00 0.006230831
```

confint() works on most objects to give confidence intervals.

Crosstabs

svyby or svymean and svytotal with interaction will produce the numbers, but the formatting is not pretty.

ftable provides formatting:

```
> d<-svyby(~api99 + api00, ~stype + sch.wide, rclus1, svymean,
         keep.var=TRUE, vartype=c("se","cvpct"))
> round(ftable(d),1)
           sch.wide          No                                 Yes
             statistics.api99 statistics.api00 statistics.api99 statistics.api00
stype
E     svymean          601.7           596.3           608.3           653.6
      SE                70.0            64.5            23.7            22.4
      cv%               11.6            10.8             3.9             3.4
H     svymean          662.0           659.3           577.6           607.5
      SE                40.9            37.8            57.4            54.0
      cv%                6.2             5.7             9.9             8.9
M     svymean          611.4           606.4           607.3           643.2
      SE                48.2            48.3            49.5            49.3
      cv%                7.9             8.0             8.2             7.7
```


Crosstabs

svyby knows enough to structure the table without help. For other analyses more information is needed

```
> a<-svymean(~interaction(stype,comp.imp), design=dclus1, deff=TRUE)
> b<-ftable(a, rownames=list(stype=c("E","H","M"),comp.imp=c("No","Yes")))
> round(100*b,1)
              stype     E     H     M
comp.imp
No       mean         17.5   3.8   6.0
         SE            2.6   1.6   2.5
         Deff         87.8 131.7 200.4
Yes      mean         61.2   3.8   7.7
         SE            4.2   1.6   2.2
         Deff        137.2 131.4 124.7
```

Testing in tables

`svychisq` does four variations on the Pearson χ^2 test: corrections to the mean or mean and variance of X^2 (Rao and Scott) and two Wald-type tests (Koch et al).

The exact asymptotic distribution of the Rao–Scott tests (linear combination of χ^2_1) and a saddlepoint approximation to it are also available.

```
>   svychisq(~sch.wide+stype, dclus1)

Pearson's X^2: Rao & Scott adjustment

data:  svychisq(~sch.wide + stype, dclus1)
F = 5.1934, ndf = 1.495, ddf = 20.925, p-value = 0.02175
```


Testing in tables

```
>   svychisq(~sch.wide+stype, dclus1, statistic="adjWald")

Design-based Wald test of association

data:  svychisq(~sch.wide + stype, dclus1, statistic = "adjWald")
F = 2.2296, ndf = 2, ddf = 13, p-value = 0.1471
```

Learning to draw

← →

Graphs

Common difficulties with graphics based on survey data include

- Weights
- Large data sets
- Maps

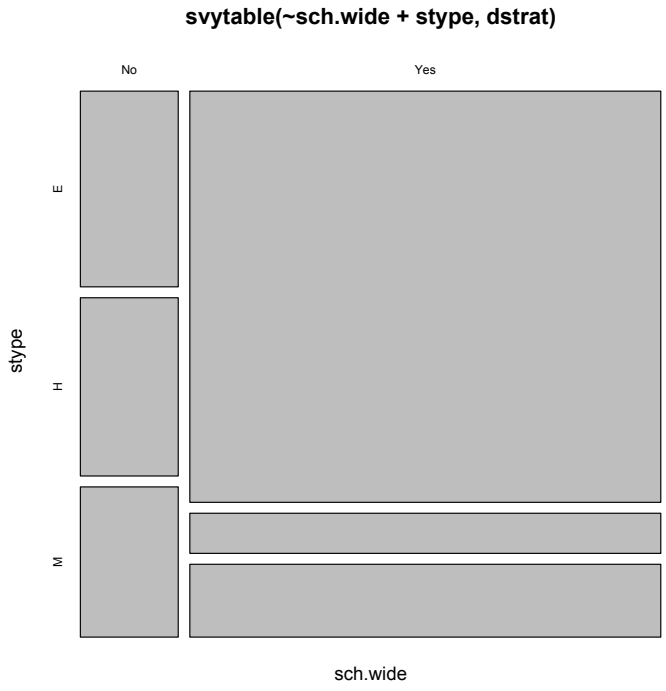
The goal is to create a graph in which we can see patterns when they are there and not see them when they aren't there.

Displaying a table

The default is a mosaic plot

```
plot(svytable(~sch.wide+stype, dstrat))
```

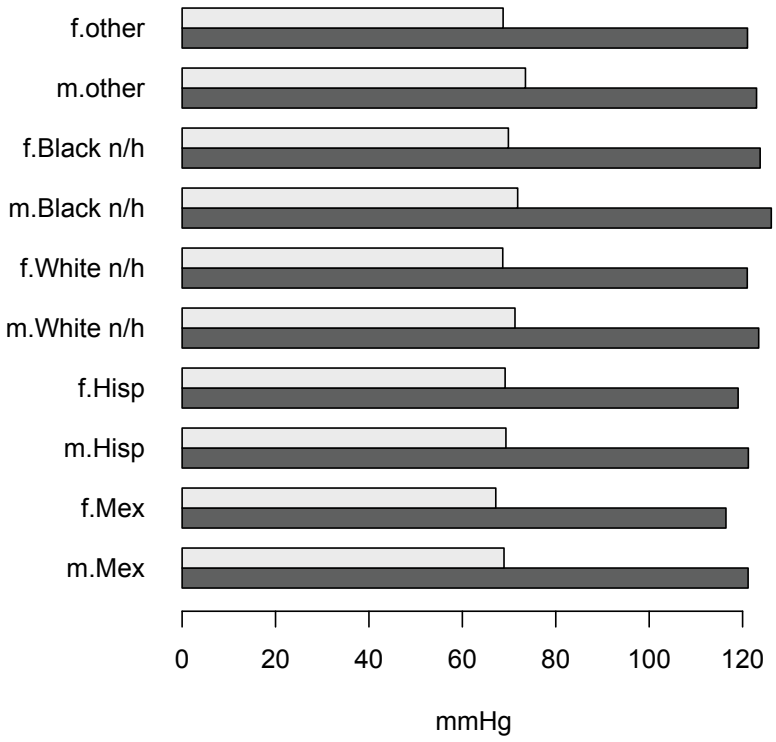

Displaying a table



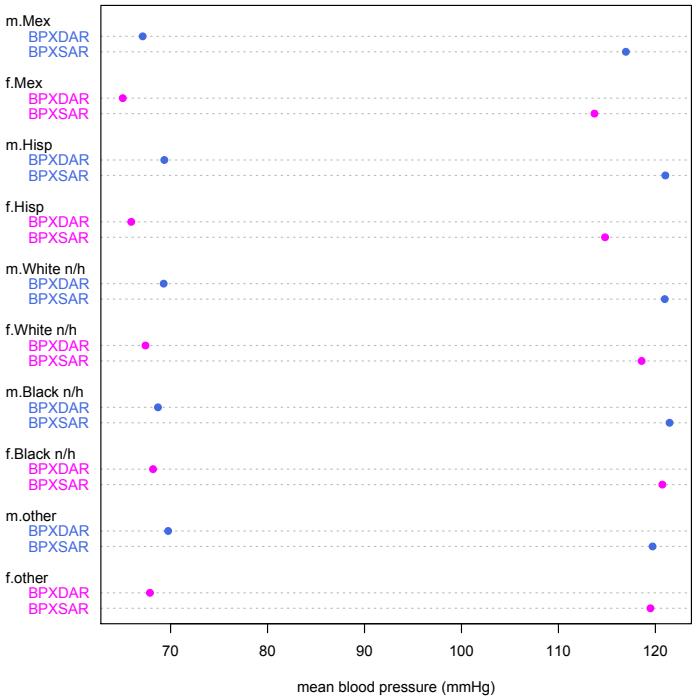
Displaying a table

```
nhanes<-update(nhanes, sex=factor(RIAGENDR,labels=c("m","f")),
    raceeth=factor(RIDRETH1,
        labels=c("Mex","Hisp","White n/h","Black n/h","other")) )
mns <- svyby(~BPXSAR+BPXDAR, ~sex+raceeth, svymean, design=nhanes,na.rm=TRUE)

dotchart(mns,cex=0.8,col=rep(c("royalblue","magenta"),each=4),
    xlab="mean blood pressure (mmHg)")

par(mar=c(4,10,1,1),las=1)
barplot(mns,horiz=TRUE,xlab="mmHg")
```

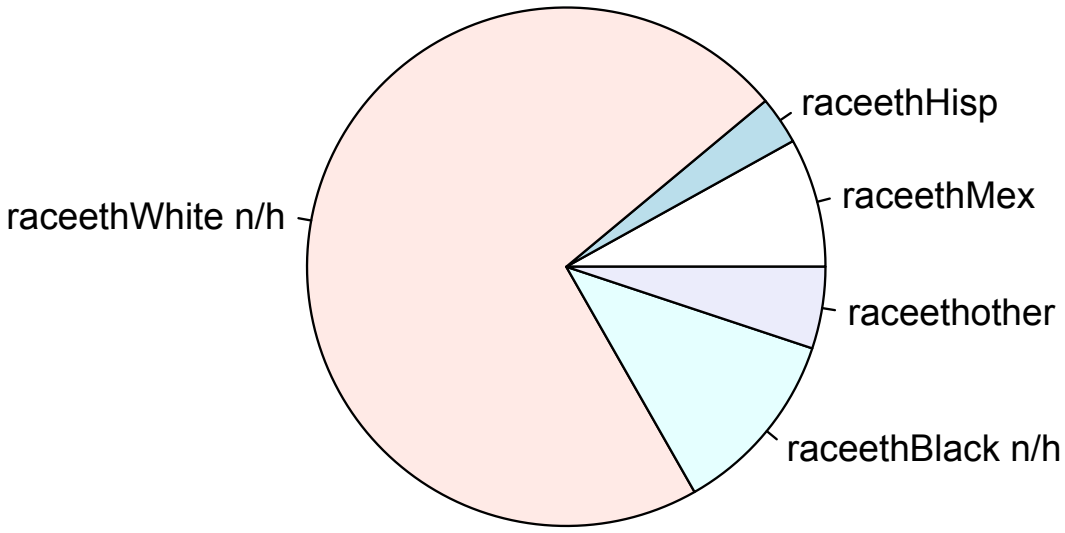

Displaying a table



Displaying a table



Pie charts

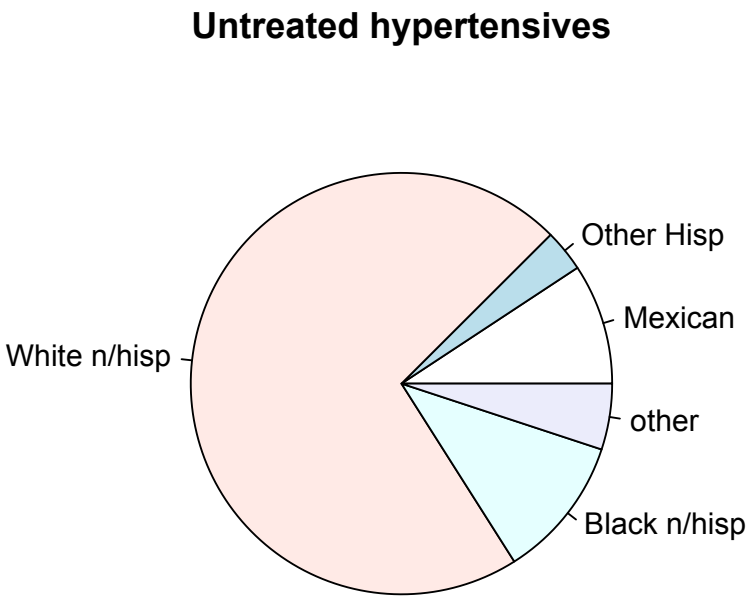



```
pie(svymean(~raceeth,nhanes,na.rm=TRUE))
```

Pie charts

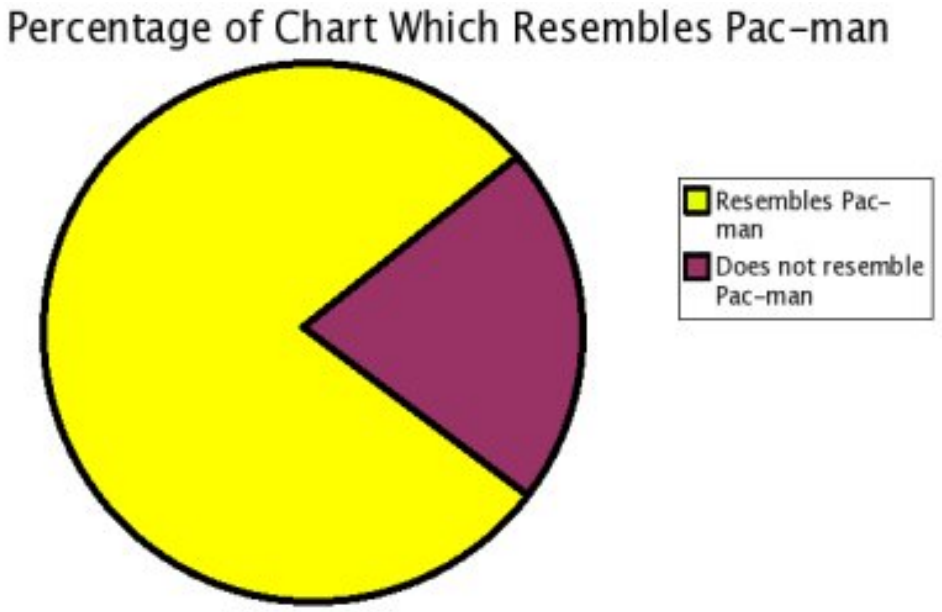
```
pie(svytotal(~raceeth,
     subset(nhanes,
        ((BPXDAR>140) | (BPXSAR>90)) & !(BPQ020==1 & BPQ050A==1)),
        na.rm=TRUE),
    labels=c("Mexican","Other Hisp","White n/hisp",
            "Black n/hisp","other"),
    main="Untreated hypertensives")
```


Pie charts



Pie charts

This is the only legitimate use for pie charts

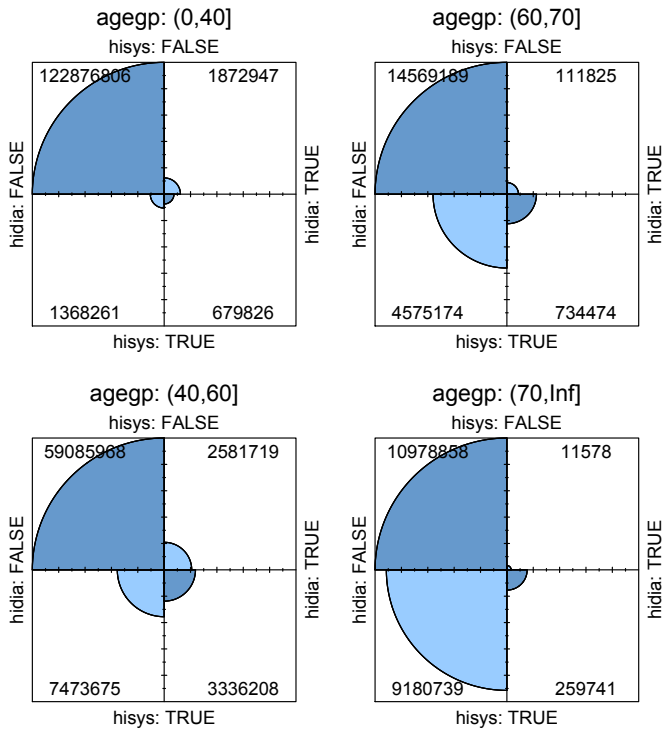


Comparing associations

A fourfold plot shows associations across a set of 2×2 tables.

```
dnhanes<-update(dnhanes, hisys= BPXSAR>140, hidia=BPXDAR>90)
dnhanes<-update(dnhanes, agegp=cut(RIDAGEYR,c(0,40,60,70,Inf)))
fourfoldplot(svytable(~hisys+hidia+agegp,dnhanes,round=1),
    std="ind.max")
```

Comparing associations



Estimating populations

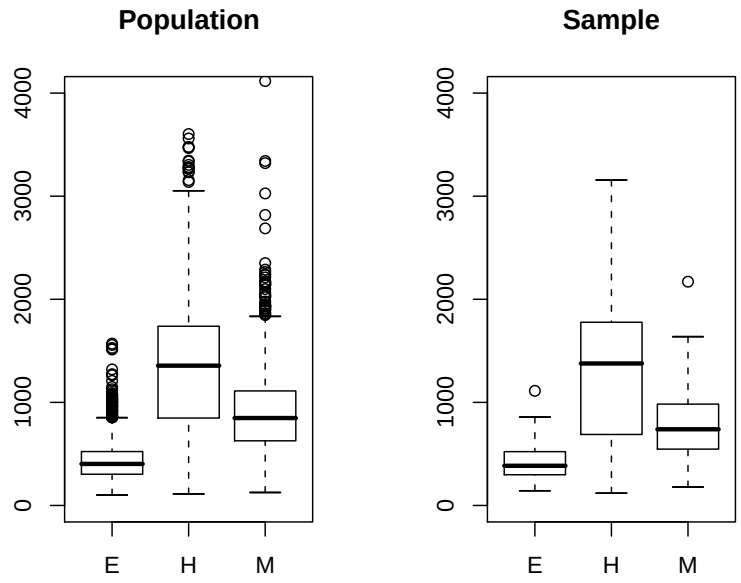
Boxplots and histograms display estimates of the population distribution function for a variable.

We can substitute the survey estimates: boxplots use quantiles, histograms need tables.

R: boxplot of enrollment by school type (Elementary/Middle/High)

```
boxplot(enroll~stype, apipop)
svyboxplot(enroll~stype, dstrat,all.outliers=TRUE,ylim=c(0,4000))
```

Estimating populations



Population on left, sample on right

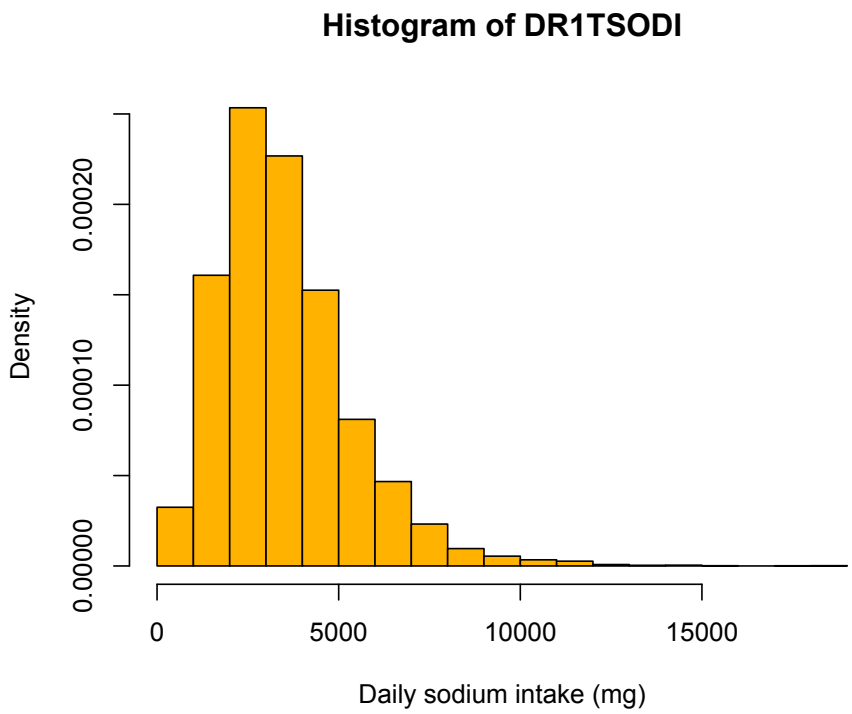
Histograms

Drawing a histograms involves dividing the sample into bins and counting the number in each bin. In survey data we use the estimated population number in each bin.

In R

```
svyhist(~DR1TSODI, nhanes, col="orange",
     xlab="Daily sodium intake (mg)")
```

Histograms



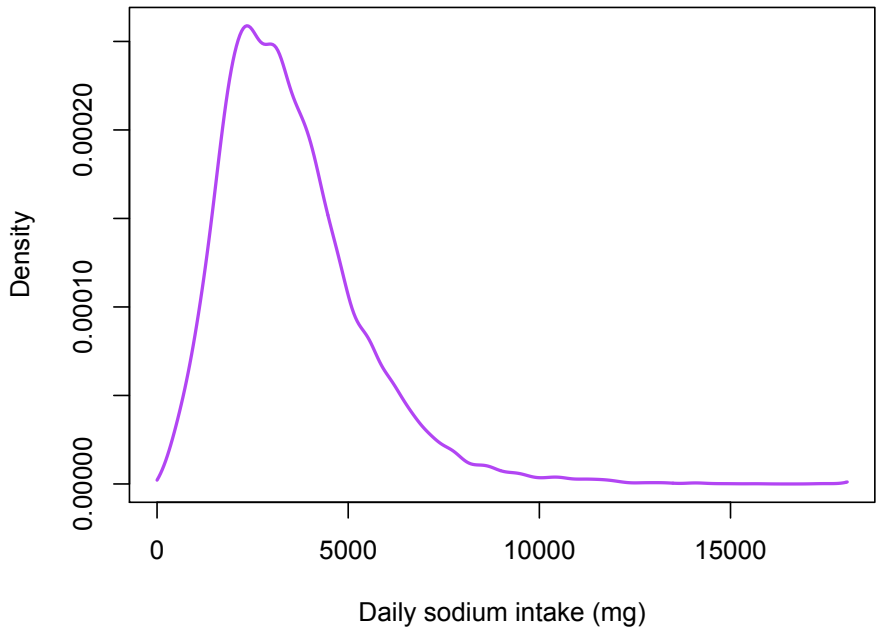
Density estimates

Kernel density estimates are smoother than histograms and give better estimates of continuous densities. They work by smearing out the weight of each observation over a small window, and then adding up the contributions from each point.

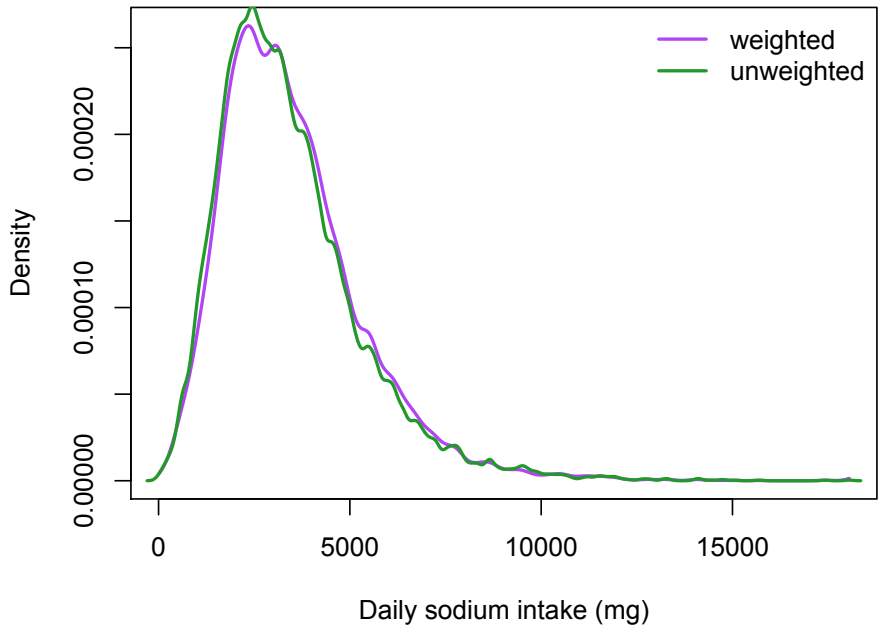
You can specify the 'bandwidth' of the smoother (usually by the Goldilocks method), or use the default that it works out for you.

```
plot(svysmooth(~DR1TSODI, nhanes,
    xlab="Daily sodium intake (mg)",bandwidth=250),
    col="purple",lwd=2,xlab="Daily sodium intake (mg)")
```

Density estimates



Effect of weighting



Effect of weighting

The impact is subtle, but ignoring the weights does underestimate sodium consumption:

```
> svymean(~I(DR1TSODI>3000),nhanes,na.rm=TRUE)
                              mean     SE
I(DR1TSODI > 3000)FALSE 0.44666 0.0087
I(DR1TSODI > 3000)TRUE  0.55334 0.0087
> mean(nhanesbp$DR1TSODI>3000,na.rm=TRUE)
[1] 0.5165171
```


Scatterplots

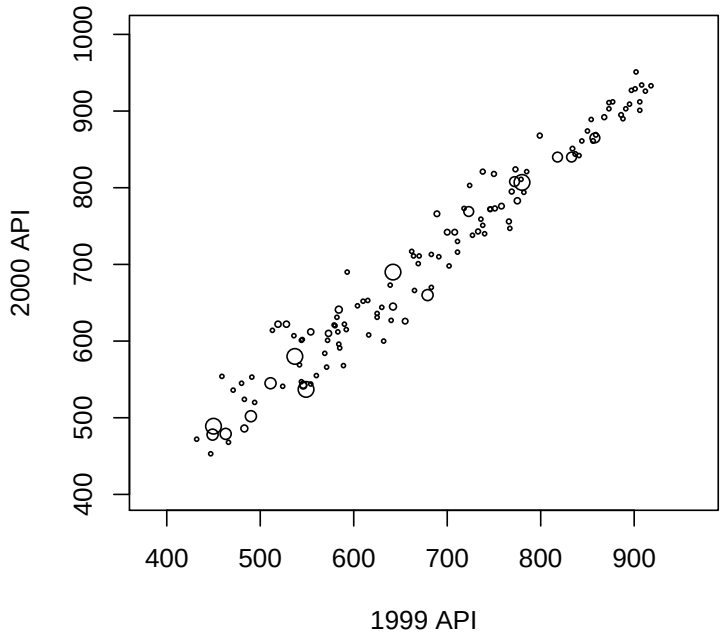
This approach doesn't work for scatterplots. We need to incorporate weights in the plotting symbols.

One approach is the bubble plot: radius or area of circle proportional to sampling weight.

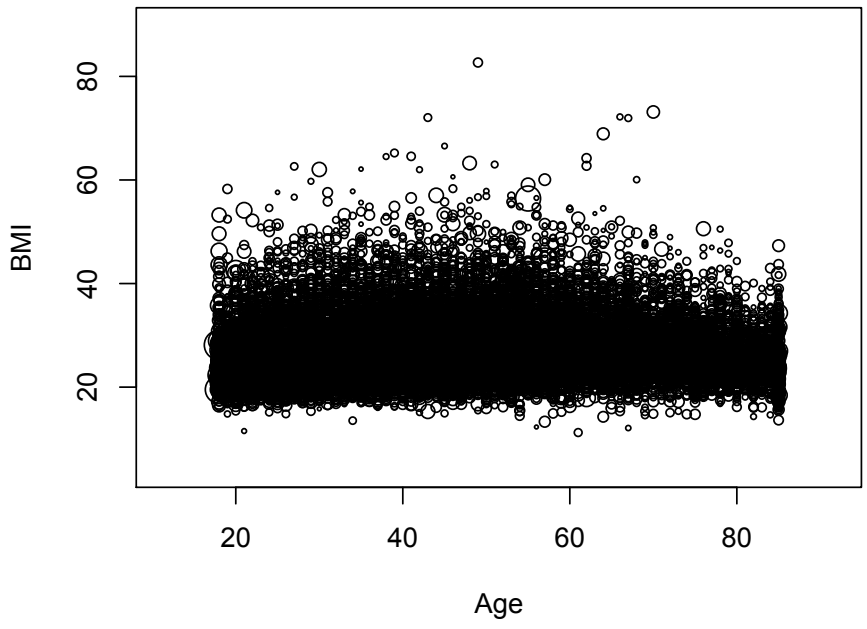
In R, `svyplot`

```
svyplot(api00~api99, design=stratrs, style="bubble")
```

Scatterplots



Scatterplots

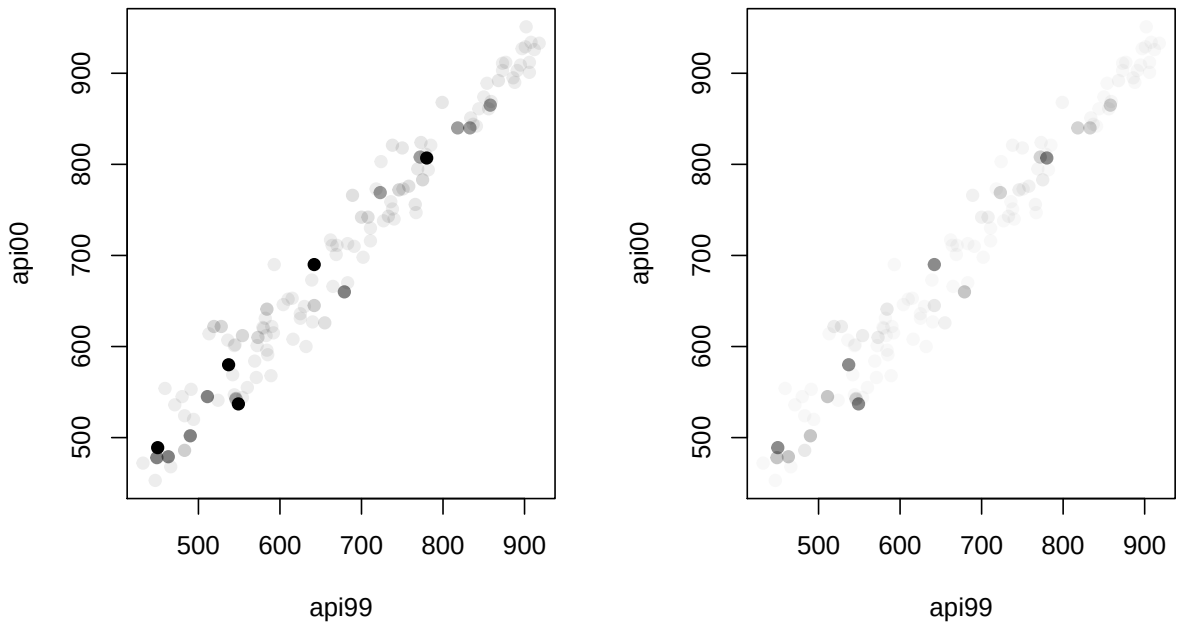


Scatterplots

Another approach is differential transparency: color weight proportional to sampling weight.

It doesn't work well for small samples. In particular, the visual effect depends strongly on how the weights are scaled. The two graphs on the next slide scale the largest weight to solid black or 50% black and look fairly dissimilar.

126 schools

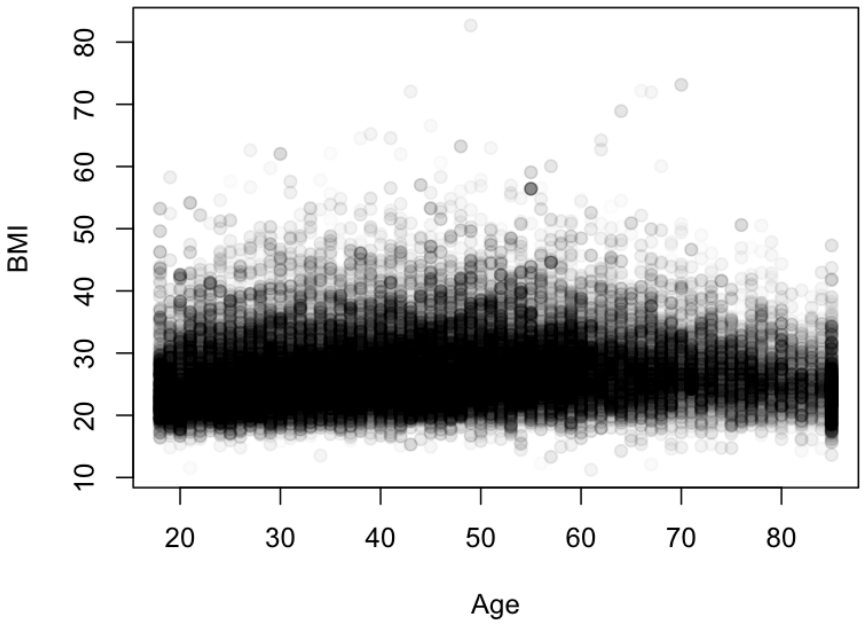


23,000 people

Transparency works better in large samples: the next slide show BMI vs age for 23,000 people from the National Health Interview Survey, where transparency works better than bubbles.

`svyplot(,type="transparent")` in R

23,000 people



Binning and smoothing

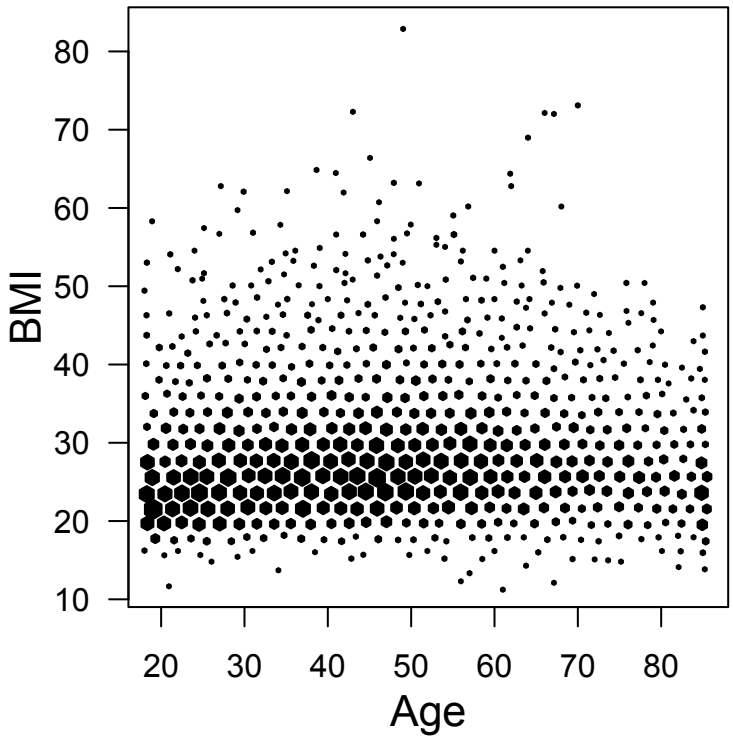
Hexagonal binning divides the plotting area into hexagons, counts the number in each hexagon [Carr, 1987, JASA]. In dense regions this has similar effect to transparency, but allows outliers to be seen, and produces much smaller graphics files.

For survey data we just add up the weight in each bin to get the estimated population numbers in each bin.

```
svyplot(bmi~age_p,style="hex",
    design=subset(nhis,bmi<90),
    xlab="Age",ylab="BMI",legend=0)
```

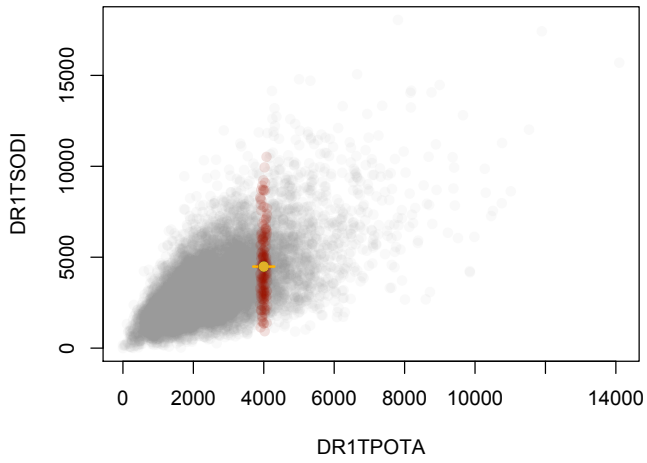
Currently doesn't allow colours to indicate groups; Auckland student Jie Fu Yu is working on this.

Binning and smoothing



Binning and smoothing

Scatterplot smoothers fit a smooth curve through the middle of the data. There are many variants, but they all work by estimating the mean value of Y using points in a small interval of values of x.



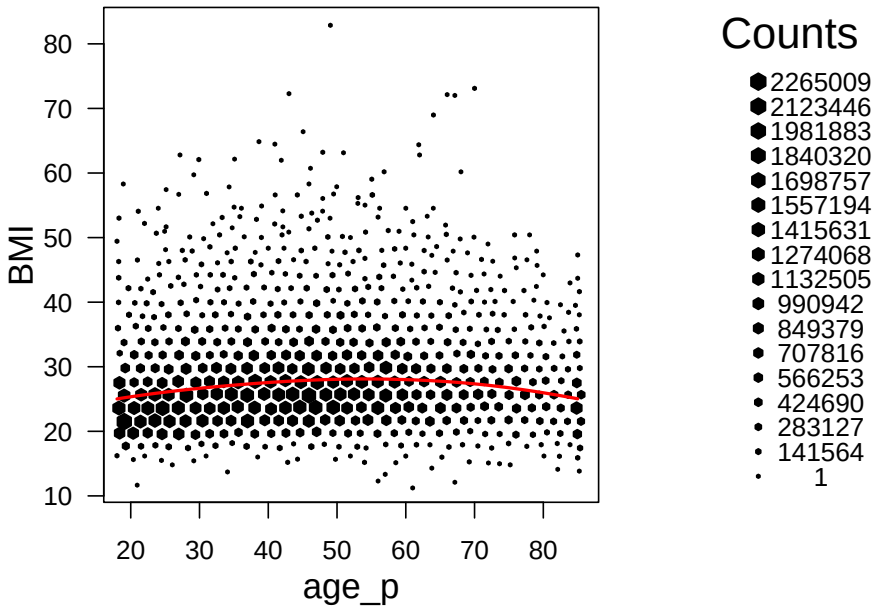
The important thing to tune is the width of the interval, which affects the smoothness of the curve

Binning and smoothing

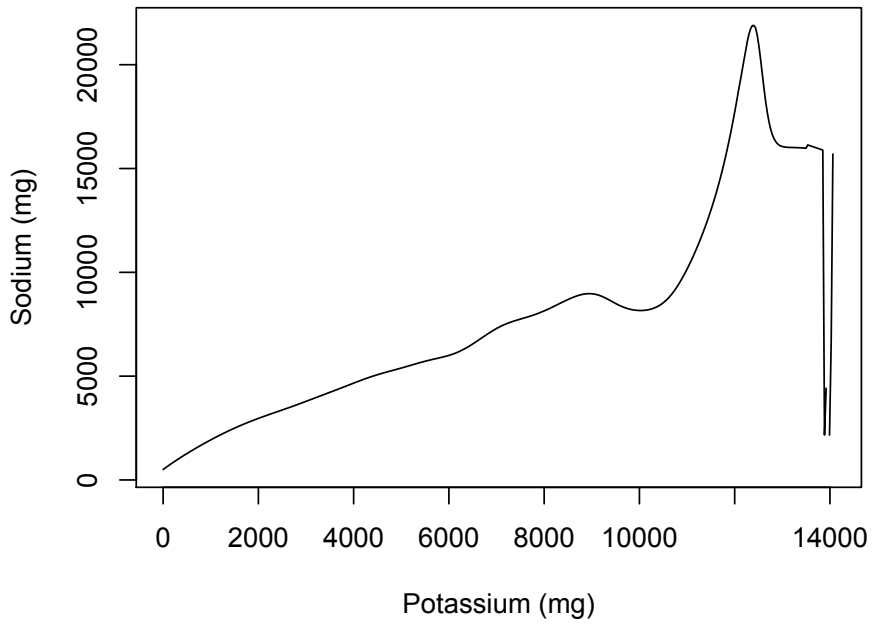
In R use `svysmooth` to create a smooth curve and then `plot` to draw it.

```
plot(svysmooth(DR1TSODI~DR1TPOTA,
    design=nhanes,bandwidth=500),
    xlab="Potassium (mg)",ylab="Sodium (mg)")plot(smth)
```

Binning and smoothing



Binning and smoothing



Quantile smoothers

For a continuous variable, the variation may be as important as the mean.

Plot a set of smooth quantile estimates to indicate the shape of the conditional distributions

Linear regression minimises $(Y-\mu)^2$ to give means.

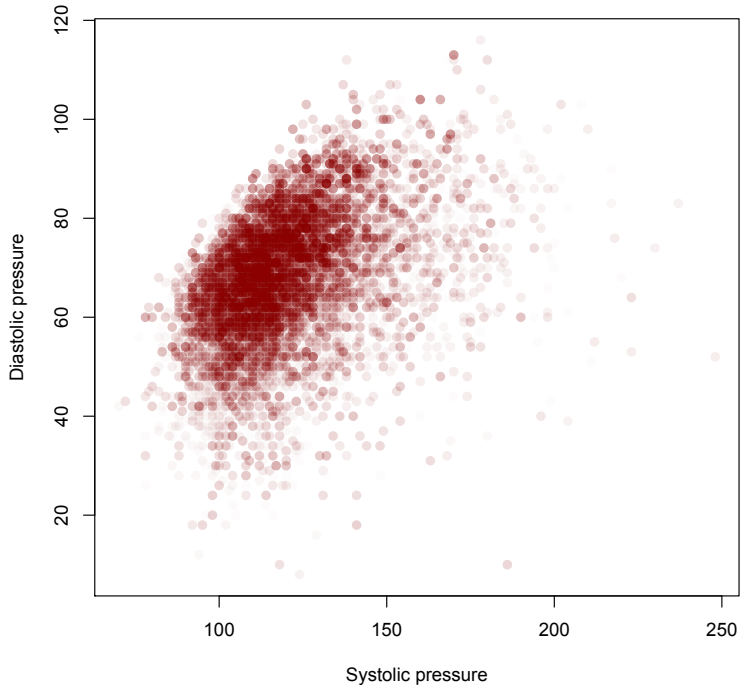
L_1 regression minimises $|Y-\mu| = (Y-\mu)^+ - (Y-\mu)^-$ to give medians

Quantile regression minimises $p(Y-\mu)^+ - (1-p)(Y-\mu)^-$, to give pth quantiles

Quantile smoothers

```
> library(foreign)
> demo<-read.xport("~/STATSdotcom/demo_c.xpt")
> bp<-read.xport("~/STATSdotcom/bpx_c.xpt")
> nhanes<-merge(demo,bp)
> dnhanes<-svydesign(id=~SDMVPSU,strata=~SDMVSTRA, weights=~WTMEC2YR,
   data=nhanes,nest=TRUE)
> svyplot(BPXDAR~BPXSAR,design=subset(dnhanes, BPXDAR>0), style="trans",
   pch=19, basecol="darkred",alpha=c(0,0.5), xlab="Systolic pressure",
   ylab="Diastolic pressure")
```

Quantile smoothers

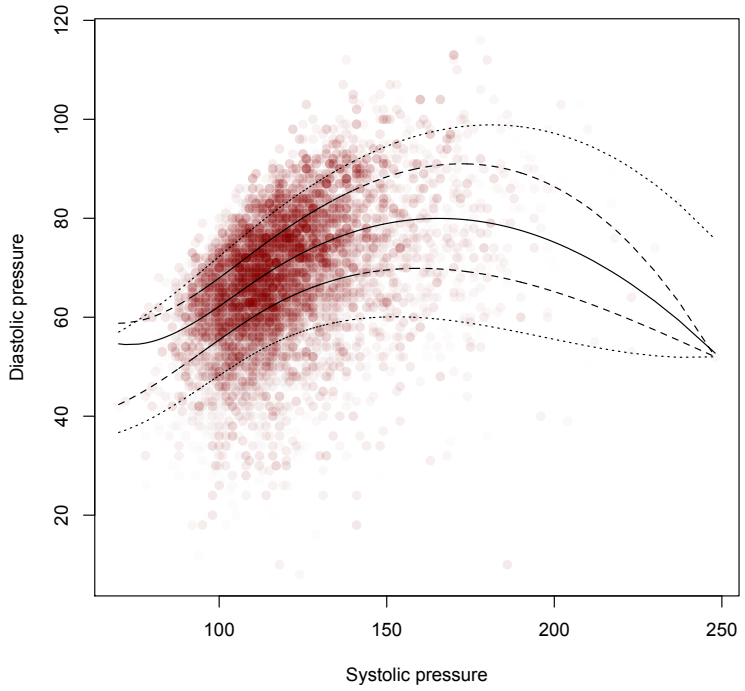


Quantile smoothers

```
> lines(svysmooth(BPXDAR~BPXSAR, design=subset(dnhanes, BPXDAR>0), df=4,
    method="quantreg", quantile=0.1),lty=3)
> lines(svysmooth(BPXDAR~BPXSAR, design=subset(dnhanes, BPXDAR>0), df=4,
    method="quantreg", quantile=0.25),lty=2)
> lines(svysmooth(BPXDAR~BPXSAR, design=subset(dnhanes, BPXDAR>0), df=4,
    method="quantreg", quantile=0.5))
> lines(svysmooth(BPXDAR~BPXSAR, design=subset(dnhanes, BPXDAR>0), df=4,
    method="quantreg", quantile=0.75),lty=2)
> lines(svysmooth(BPXDAR~BPXSAR, design=subset(dnhanes, BPXDAR>0), df=4,
    method="quantreg", quantile=0.9),lty=3)
```

[Note: should probably have truncated the highest values as well as the zeroes]

Quantile smoothers



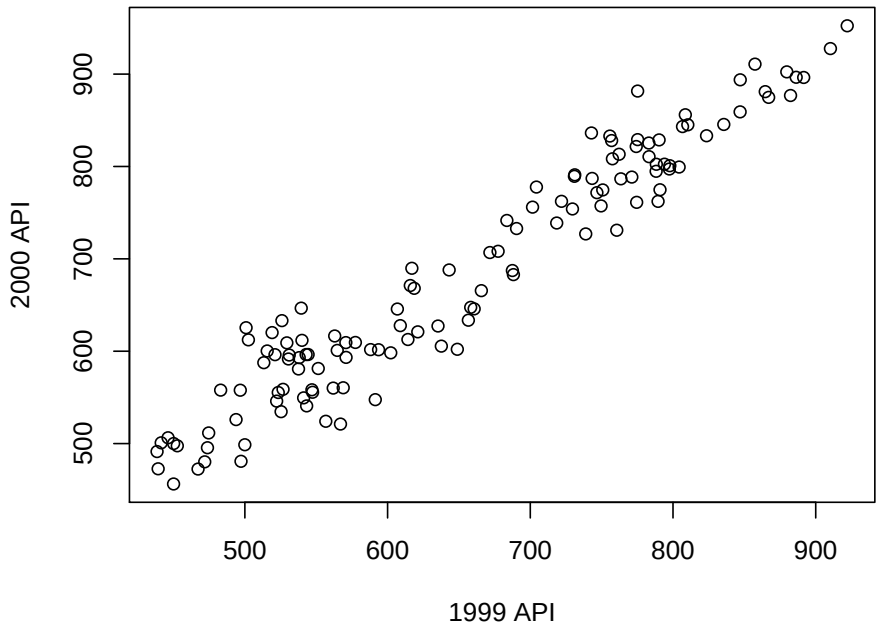
Synthetic data

Researchers are good at interpreting scatterplots of independent, equally-weighted observations, so we could estimate the population distribution and then draw a simple random sample (with replacement).

We will end up with duplicate points, so try adding random noise to them ('jittering'). In California API data, the median absolute year-to-year change was 25, so this is a reasonable noise level.

```
svyplot(api00~api99, design=dclus2, style="subsample",
          amount=list(x=25,y=25))
```

Synthetic data

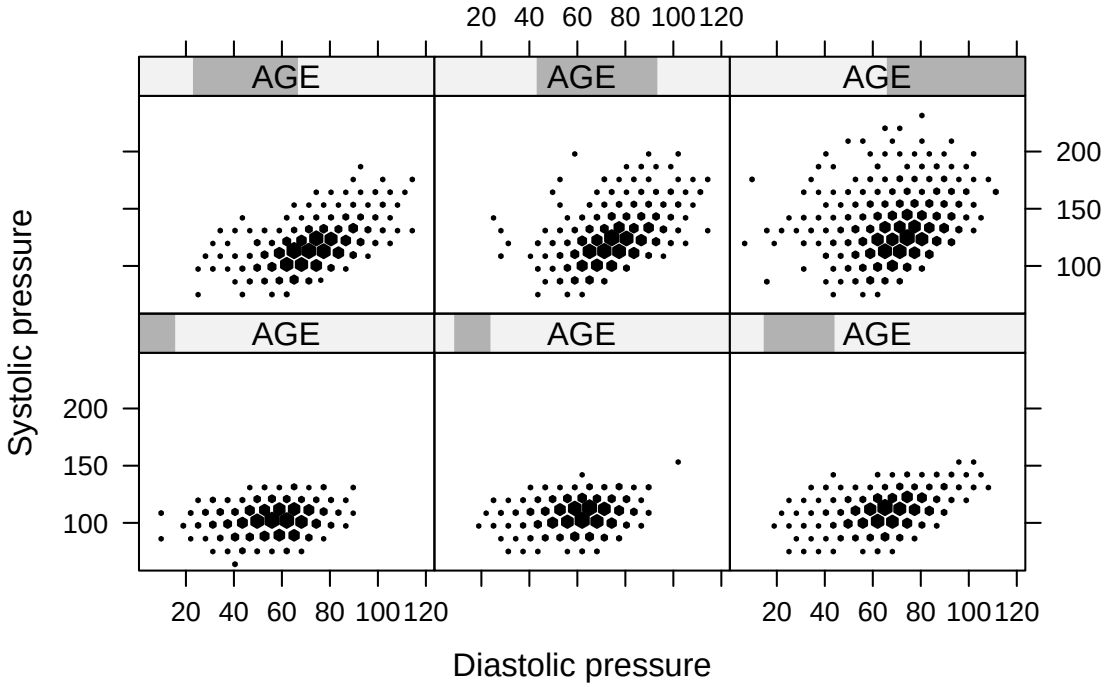


Conditioning plots

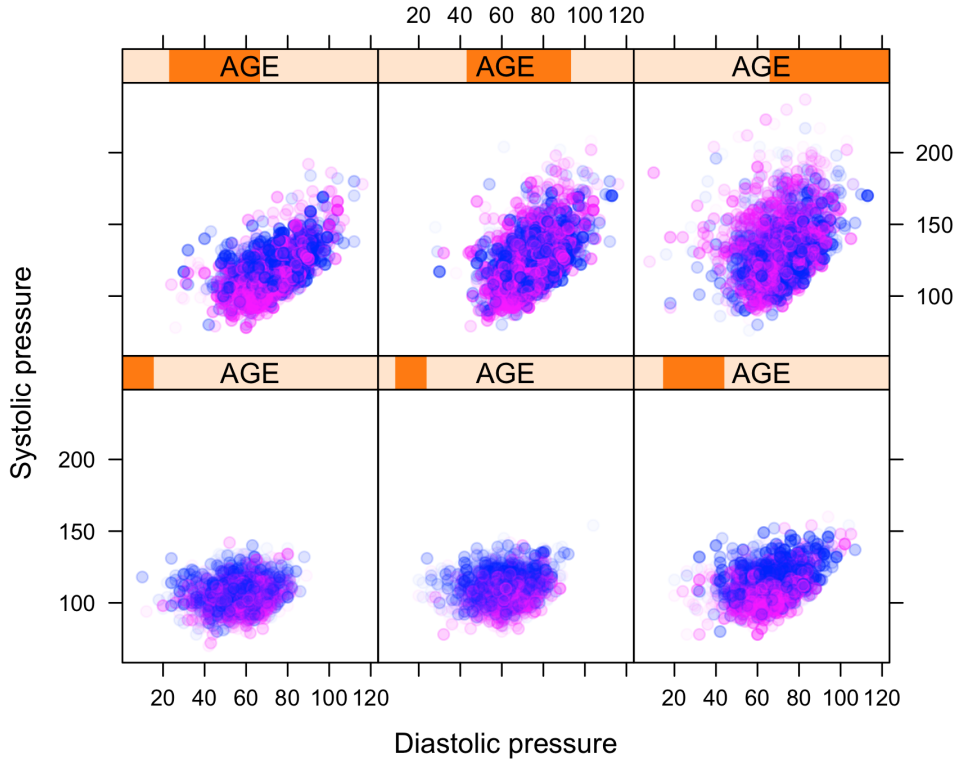
Three-dimensional 'false perspective' plots are useless, but 3-d and 4-d relationships can be shown by conditioning.

```
svycoplot(BPXSAR~BPXDAR|equal.count(RIDAGEMN), style="hex",
  design=subset(dhanes,BPXDAR>0), xbins=20,
  strip=strip.custom(var.name="AGE"),
  xlab="Diastolic pressure",ylab="Systolic pressure")

svycoplot(sysbp~diabp|agegp, style="transparent",
   basecol=function(d) c("magenta","royalblue")[d$sex]
   data=nhanes_design)
```

Conditioning plots



Conditioning plots

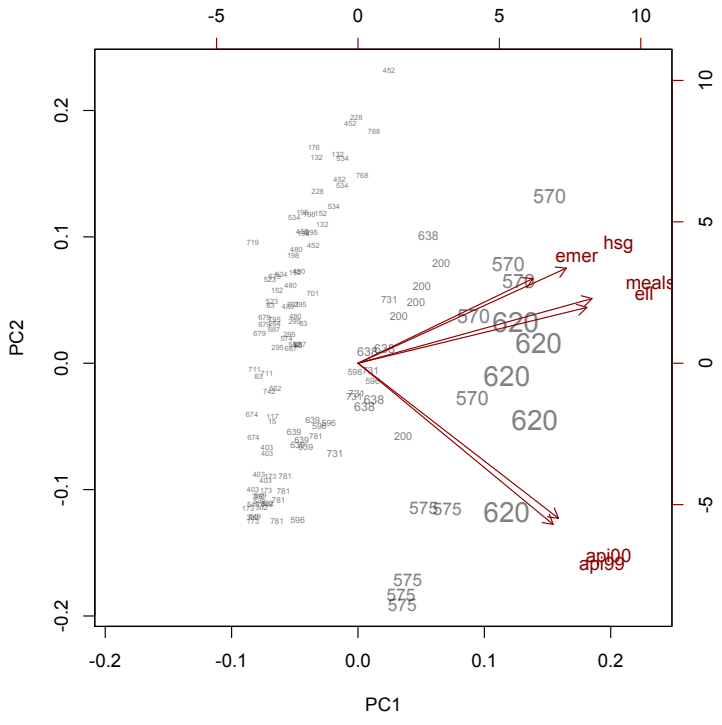


Biplots

Biplots show the first two left and right singular vectors of the data on the same graph, ie, project variables and observations on to a two-dimensional scatterplot.

```
> data(api)
> dclus2<-svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)
> pc <- svyprcomp(~api99+api00+ell+hsg+meals+emer, design=dclus2,
      scale=TRUE,scores=TRUE)

> biplot(pc, weight="scaled", max.cex=1.5, xlabs=~dnum)
```


Biplots

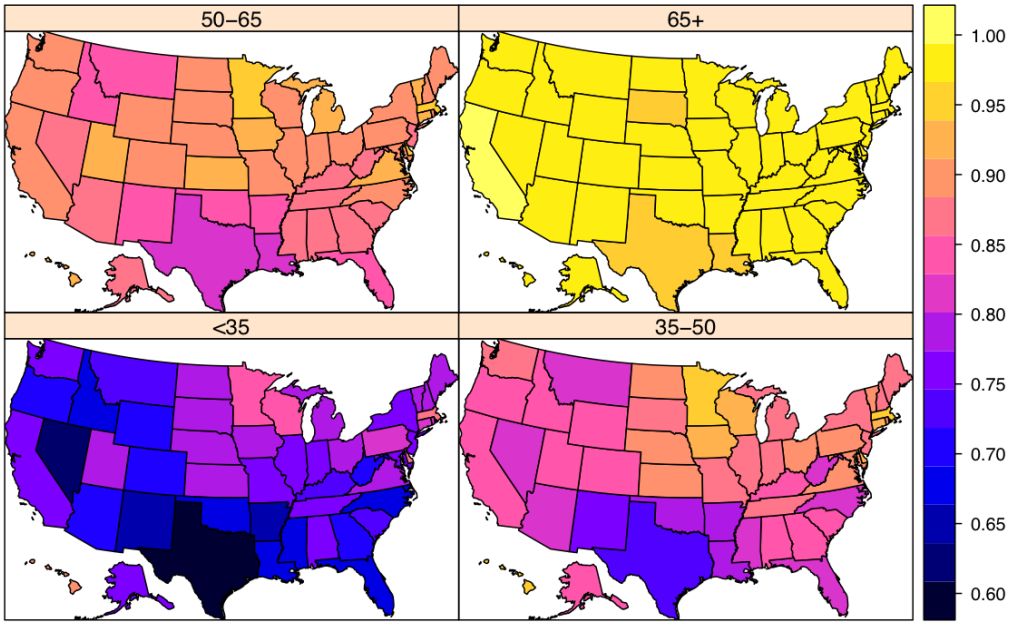


Maps

Maps are a form of graph for a table: compute estimates for each geographic region and then use them to color the map.

```
library(sp)
library(maptools)
## map data from BRFSS website
states<-readShapePoly("brfss_state_2007_download")
states<-states[order(states$ST_FIPS),]

brfss<-update(brfss, agegp=cut(AGE, c(0,35,50,65,Inf)))
hlth<-svyby(~I(HLTHPLAN==1), ~agegp+X_STATE, svymean,
    design=brfss)

hlthdata<-reshape(hlth[,c(1,2,4)],idvar="X_STATE",
    direction="wide",timevar="agegp")
names(hlthdata)[2:5]<-paste("age",1:4,sep="")

states@data<-merge(states,hlthdata,
    by.x="ST_FIPS",by.y="X_STATE",all=FALSE)
spplot(states,c("age1","age2","age3","age4"),
    names.attr=c("<35","35-50","50-65","65+"))
```


Maps



Regression modelling

← →

Available regression models

- `svyglm` for linear and generalized linear models (and regression estimator of total via `predict()`
- `svyolr` for proportional odds and other cumulative link models.
- `svycoxph` for Cox model
- `svyloglin` for loglinear models

others via `withReplicates()` or `svymle()`

What is being estimated

Design-based regression inference is estimating a population summary statistic, not a model parameter

eg, least-squares estimates the β that minimizes population residual sum of squares

$$\beta^* = \operatorname*{argmin}_{\beta} \sum_{i=1}^{N} (y_i - x_i\beta)^2$$

by

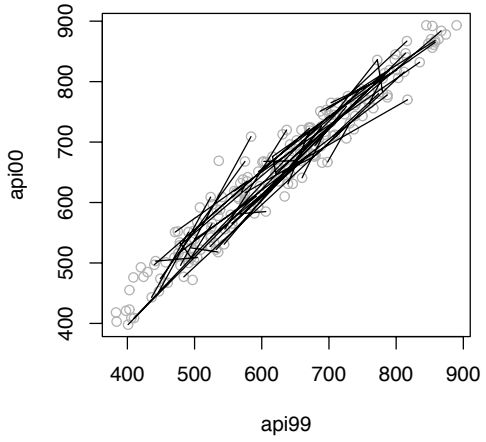
$$\widehat{\beta} = \operatorname*{argmin}_{\beta} \sum_{i:R_i=1} \frac{1}{\pi_i}(y_i - x_i\beta)^2$$

No assumptions are made about the distribution of Y: strictly, Y isn't even random

What is being estimated

For least squares, can define β without reference to a model

- For two points i,j, the best-fitting line just joins them
- For N points, the slope of the best-fitting line is an average over pairs of points, with weights $(x_i - x_j)^2$



What is being estimated

On the other hand, β^* is only a useful summary if the first-order linear trend is interesting: eg, not in a seasonal time series.

Generalizability to populations other than the measured one does rely on assumptions about the stability of the data-generating process

Depending on the application, more assumptions may be required: eg out-of-sample prediction requires accurate approximation to conditional distribution of $Y|X$.

Weights

If the sampling design variables were all correctly specified in the model, then $Y \perp R \,|\, X$, and the sampling is ignorable.

If the sampling is ignorable, ignoring it increases precision, potentially by a lot.

For large surveys, bias is likely more important than variance, so ignoring weights is risky.

And the design variables may not be appropriate for inclusion in the model

But for smaller surveys, ignoring weights may improve mean squared error noticeably.

Models for associations

Often want inference about relationships between variables.

Using data from NHANES: does a higher-sodium diet lead to higher blood pressure?

Model choice now needs to address **confounding.**

A fixed finite population doesnt have effects: we need to think of the population as the result of a data-generating process (or superpopulation)

Models for associations

The weighted least squares estimate gives an approximately unbiased, asymptotically Normal estimator of the least-squares linear approximation to the conditional mean in the data-generating process.

If the model includes a complete set of confounders for the effect of X on Y , the slope of the linear approximation to the mean of the process is an average causal effect of X on Y , and the weighted least squares estimator is approximately unbiased for this.

[Of course, you usually can't get a complete set of confounders]

Models for associations

Useful predictor variables can be divided into

- exposure of interest: this is the one we care about
- confounders: these are thought to affect the outcome and be associated with the exposure of interest, and not be affected by the exposure of interest
- precision variables: these are associated with the outcome and independent of the predictor of interest.

These decisions should be made in advance, not based on the data, in order for the standard errors and p-values to be correct.

Example

In NHANES, is there evidence that a high sodium diet leads to higher blood pressure?

Main variables

- BPXSAR, BPXDAR: systolic and diastolic BP
- BPQ030, BPQ040, BPQ050: Has your doctor ever told you?, Have you ever been prescribed? Are you currently taking?
- DR1TSODI, DR1TPOTA, DR1TKCAL: sodium, potassium, calories
- RIDRETH1, RIDAGEYR, RIAGENDR: race/ethnicity, age, gender
- SDMVPSU, SDMVSTRA,fouryearwt: (pseudo) PSU, stratum, sampling weight.

Example

```
nhanesbp <- read.csv("nhanesbp.csv")
nhanes <- svydesign(id=~SDMVPSU, strata=~SDMVSTRA,
    weight=~fouryearwt, data=nhanesbp, nest=TRUE)
nhanes <- update(nhanes, trt = !is.na(BPQ050A) & BPQ050A==1)
summary(svyglm(BPXSAR~DR1TSODI+DR1TPOTA, design=nhanes))
summary(svyglm(BPXSAR~DR1TSODI+DR1TPOTA+RIAGENDR+factor(RIDRETH1)
      +I(RIDAGEYR/10), design=nhanes))
summary(svyglm(BPXSAR~DR1TSODI+DR1TPOTA+RIAGENDR+factor(RIDRETH1)
      +I(RIDAGEYR/10) +DR1TKCAL+trt, design=nhanes))
```


Example

```
Call:
svyglm(formula = BPXSAR ~ DR1TSODI + DR1TPOTA + RIAGENDR
    + factor(RIDRETH1) + I(RIDAGEYR/10), design = nhanes)
Survey design:
update(nhanes, trt = !is.na(BPQ050A) & BPQ050A == 1)

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)        1.066e+02  1.447e+00  73.720  < 2e-16 ***
DR1TSODI           6.076e-04  1.840e-04   3.303 0.003241 **
DR1TPOTA          -1.072e-03  1.997e-04  -5.366 2.18e-05 ***
RIAGENDR          -3.562e+00  4.370e-01  -8.150 4.33e-08 ***
factor(RIDRETH1)2 -8.226e-02  1.554e+00  -0.053 0.958262
factor(RIDRETH1)3 -1.274e+00  7.618e-01  -1.672 0.108657
factor(RIDRETH1)4  3.747e+00  8.353e-01   4.485 0.000184 ***
factor(RIDRETH1)5  9.914e-01  1.168e+00   0.849 0.405061
I(RIDAGEYR/10)     4.981e+00  1.278e-01  38.986  < 2e-16 ***
```

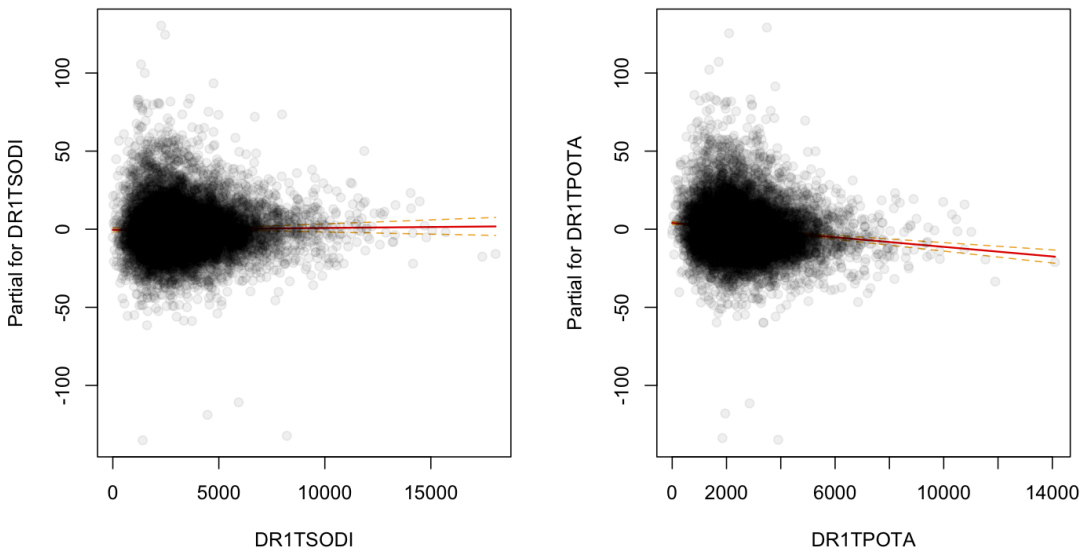
Results

	Adjustments		
	none	age, sex, race	cal,treat
Sodium (mmHg/g/day)	-0.57	0.61	0.12
(SE)	(0.17)	(0.18)	(0.20)
Potassium (mmHg/g/day)	0.06	-1.07	-1.53
(SE)	(0.25)	(0.20)	(0.18)

The associations are in the expected directions, but surprisingly weak (1g/day is a big difference in intake, 1 mmHg is a very small difference in BP)

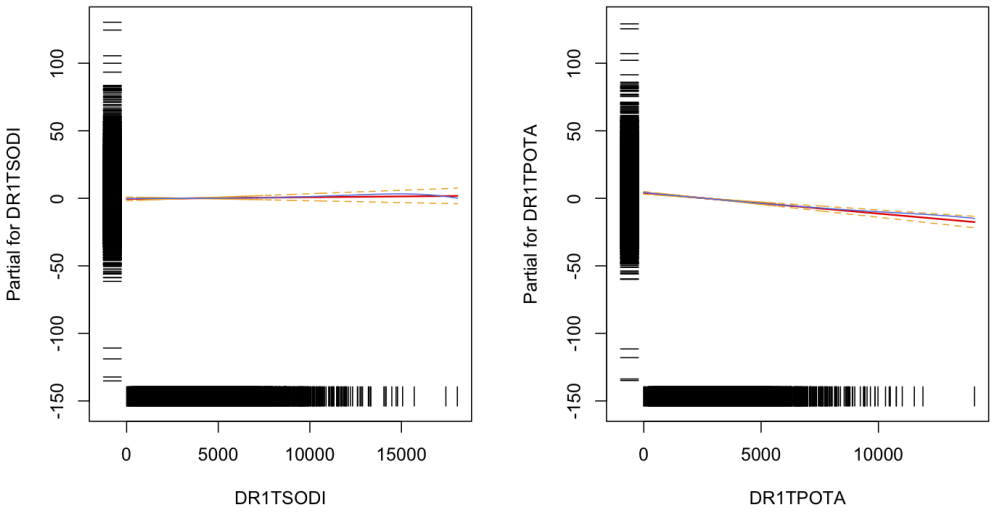
Model criticism

```
termplot(model,col.res="#00000010",partial=TRUE,se=TRUE,pch=19, terms=c(1,2))
```



Model criticism

```
termplot(model,smooth=make.panel.svysmooth(nhanes[-model$na.action,]), se=TRUE,
   terms=c(1,2), rug=TRUE, partial=TRUE,col.res=0, col.smth="royalblue",
   data=model.frame(nhanes[-model$na.action,]),lty.smth=1)
```



Model criticism

No sign of non-linearity.

Age might be nonlinear, but modelling age with splines doesn't change the coefficients we are interested in.

Perhaps an age interaction?

```
intmodel<-svyglm(BPXSAR~(DR1TSODI+DR1TPOTA)*RIDAGEYR
  +RIAGENDR*RIDAGEYR+ factor(RIDRETH1)+ns(RIDAGEYR,3)
   +DR1TKCAL+trt, design=nhanes)
```

Some evidence that the potassium association changes with age, but not for sodium

```
DR1TSODI:RIDAGEYR  1.424e-05  1.036e-05   1.376 0.189163
DR1TPOTA:RIDAGEYR -3.085e-05  1.191e-05  -2.590 0.020502 *
```

Model criticism

But the associations are still weak at all ages

```
> svycontrast(intmodel, quote(DR1TPOTA+30*`DR1TPOTA:RIDAGEYR`))
               nlcon    SE
contrast -0.00099654 2e-04
> svycontrast(intmodel, quote(DR1TPOTA+40*`DR1TPOTA:RIDAGEYR`))
              nlcon    SE
contrast -0.0013051 2e-04
> svycontrast(intmodel, quote(DR1TPOTA+50*`DR1TPOTA:RIDAGEYR`))
              nlcon    SE
contrast -0.0016136 2e-04
> svycontrast(intmodel, quote(DR1TPOTA+80*`DR1TPOTA:RIDAGEYR`))
              nlcon    SE
contrast -0.0025391 5e-04
```

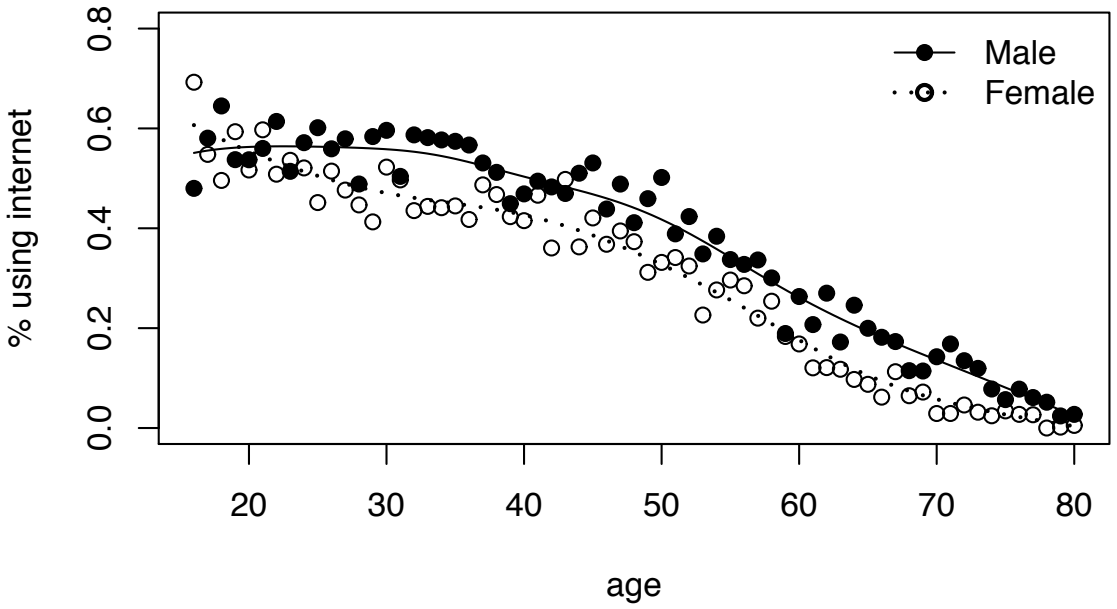
Measurement error is part of the problem: even good dietary data is bad.

Logistic regression model

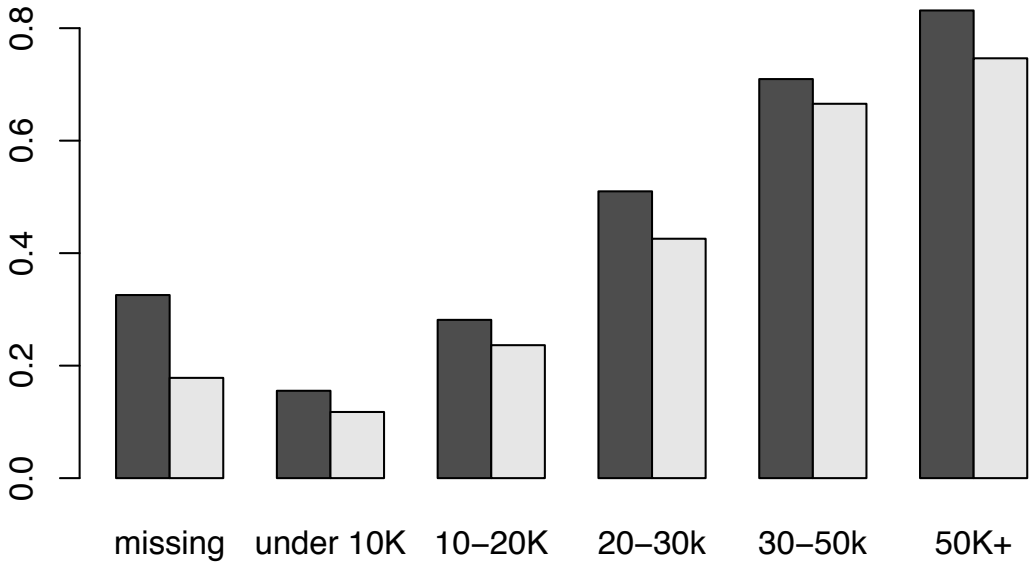
Internet use in Scotland (2001) by age, sex, and income.

```
shs<-svydesign(id=~psu, strata=~stratum, weight=~grosswt,
     data=shs_data)
bys<-svyby(~intuse,~age+sex,svymean,design=shs)
plot(svysmooth(intuse~age,
      design=subset(shs,sex=="male" & !is.na(age)),
    bandwidth=5),ylim=c(0,0.8),ylab="% using internet")
lines(svysmooth(intuse~age,
      design=subset(shs,sex=="female" & !is.na(age)),
    bandwidth=5),lwd=2,lty=3)
points(bys$age,bys$intuse,pch=ifelse(bys$sex=="male",19,1))
legend("topright",pch=c(19,1),lty=c(1,3),lwd=c(1,2),
    legend=c("Male","Female"),bty="n")
byinc<-svyby(~intuse, ~sex+groupinc, design=shs)
barplot(byinc)
```

Age (cohort) effect



Income



Code

```
> m<-svyglm(intuse~I(age-18)*sex,design=shs,
    family=quasibinomial())
> m2<-svyglm(intuse~(pmin(age,35)+pmax(age,35))*sex,
    design=shs,family=quasibinomial)
> summary(m)
svyglm(intuse ~ I(age - 18) * sex, design = shs,
   family = quasibinomial())
Survey design:
svydesign(id = ~psu, strata = ~stratum, weight = ~grosswt,
   data = ex2)
Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)           0.804113   0.047571  16.903  < 2e-16 ***
I(age - 18)          -0.044970   0.001382 -32.551  < 2e-16 ***
sexfemale            -0.116442   0.061748  -1.886   0.0594 .
I(age - 18):sexfemale -0.010145   0.001864  -5.444 5.33e-08 ***
```


Code

```
> summary(m2)
Call:
svyglm(intuse ~ (pmin(age, 35) + pmax(age, 35)) * sex,
    design = shs, family = quasibinomial)
Survey design:
svydesign(id = ~psu, strata = ~stratum, weight = ~grosswt,
   data = ex2)
Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)             2.152291   0.156772  13.729  < 2e-16 ***
pmin(age, 35)           0.014055   0.005456   2.576 0.010003 *
pmax(age, 35)          -0.063366   0.001925 -32.922  < 2e-16 ***
sexfemale               0.606718   0.211516   2.868 0.004133 **
pmin(age, 35):sexfemale -0.017155   0.007294  -2.352 0.018691 *
pmax(age, 35):sexfemale -0.009804   0.002587  -3.790 0.000151 ***
```

Code

```
> svycontrast(m2,
   quote(`pmin(age, 35)` +`pmin(age, 35):sexfemale`))
            nlcon     SE
contrast -0.0031 0.0049
> svycontrast(m2,
   quote(`pmax(age, 35)` + `pmax(age, 35):sexfemale`))
             nlcon     SE
contrast -0.07317 0.0018
```


Loglinear models

`svyloglin()` does loglinear models

```
a<-svyloglin(~backpain+neckpain+sex+sickleave, nhis)
a2<-update(a, ~.^2)
a3<-update(a,~.^3)
b1<-update(a,~.+(backpain*neckpain*sex)+sex*sickleave)
b2<-update(a,~.+(backpain+neckpain)*sex+sex*sickleave)
b3<-update(a,~.+backpain:neckpain+sex:backpain+sex:neckpain
    +sex:sickleave)
```

`anova()` method computes Rao–Scott working loglikelihood and working score tests, with the two Rao–Scott approximations for the p-value and the exact asymptotic distribution.

Loglinear models

```
> anova(a,a2)
Analysis of Deviance Table
 Model 1: y ~ backpain + neckpain + sex + sickleave
Model 2: y ~ backpain + neckpain + sex + sickleave + backpain:neckpain +
    backpain:sex + backpain:sickleave + neckpain:sex + neckpain:sickleave +
    sex:sickleave
Deviance= 3563.795 p= 0
Score= 4095.913 p= 0
> anova(a2,a3)
Analysis of Deviance Table
 Model 1: y ~ backpain + neckpain + sex + sickleave + backpain:neckpain +
    backpain:sex + backpain:sickleave + neckpain:sex + neckpain:sickleave +
    sex:sickleave
Model 2: y ~ backpain + neckpain + sex + sickleave + backpain:neckpain +
    backpain:sex + backpain:sickleave + neckpain:sex + neckpain:sickleave +
    sex:sickleave + backpain:neckpain:sex + backpain:neckpain:sickleave +
    backpain:sex:sickleave + neckpain:sex:sickleave
Deviance= 11.55851 p= 0.02115692
Score= 11.58258 p= 0.02094331
> print(anova(a2,a3),pval="saddlepoint")
[...snip...]
Deviance= 11.55851 p= 0.02065965
Score= 11.58258 p= 0.02044939
```


Other models

Can extend the modelling in two ways

- `svymle()` takes a loglikelihood and score function for a single observation and maximizes the weighted population loglikelihood estimate, inserting linear predictors for any parameters.

- `withReplicates()` runs arbitrary code, supplying each set of replicate weights, and combines the answers to give standard errors.

Appendix E of the book has a worked example for negative binomial regression.

Neither approach works for mixed models, which are difficult and currently not implemented.

Adjustment of weights

← →

Post-stratification and calibration

Post-stratification and calibration are ways to use auxiliary information on the population (or the phase-one sample) to improve precision.

They are closely related to the Augmented Inverse-Probability Weighted estimators of Jamie Robins and coworkers, but are easier to understand.

Auxiliary information

HT estimator is inefficient when some additional population data are available.

Suppose x_i is known for all i

Fit $y \sim x\beta$ by (probability-weighted) least squares to get $\widehat{\beta}$. Let r^2 be proportion of variation explained.

$$\widehat{T}_{reg} = \sum_{R_i=1} \frac{1}{\pi_i}(y_i - x_i\widehat{\beta}) + \sum_{i=1}^{N} x_i\widehat{\beta}$$

ie, HT estimator for sum of residuals, plus population sum of fitted values

Auxiliary information

Let β^* be true value of β (ie, least-squares fit to whole population).

Regression estimator

$$\widehat{T}_{reg} = \sum_{R_i=1} \frac{1}{\pi_i}(y_i - x_i\beta^*) + \left(\sum_{i=1}^{N} x_i\right)\beta^* + \sum_{i=1}^{N}\left(1 - \frac{R_i}{\pi_i}\right)x_i(\widehat{\beta} - \beta^*)$$

compare to HT estimator

$$\widehat{T} = \sum_{R_i=1} \frac{1}{\pi_i}(y_i - x_i\beta^*) + \left(\sum_{R_i=1} \frac{1}{\pi_i}x_i\right)\beta^*$$

Second term uses known vs observed total of x, third term is estimation error for β, of smaller order.

Auxiliary information

For large n, N and under conditions on moments and sampling schemes

$$\text{var}\left[\widehat{T}_{reg}\right] = (1-r^2)\,\text{var}\left[\widehat{T}\right] + O(N/\sqrt{n}) = \left(1 - r^2 + O(n^{-1/2})\right)\text{var}\left[\widehat{T}\right]$$

and the relative bias is $O(1/n)$

The lack of bias does not require any assumptions about $[Y|X]$

$\widehat{\beta}$ is consistent for the population least squares slope β, for which the mean residual is zero by construction.

Reweighting

Since $\widehat{\beta}$ is linear in y, we can write $x\widehat{\beta}$ as a linear function of y and so $\widehat{T}_{reg}$ is also a linear function of Y

$$\widehat{T}_{reg} = \sum_{R_i=1} w_i y_i = \sum_{R_i=1} \frac{g_i}{\pi_i} y_i$$

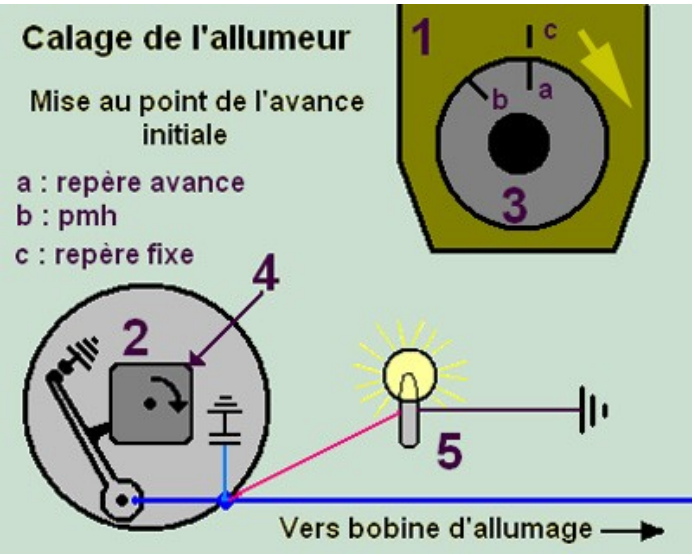
for some (ugly) w_i or g_i that depend only on the xs

For these weights

$$\sum_{i=1}^{N} x_i = \sum_{R_i=1} \frac{g_i}{\pi_i} x_i$$

$\widehat{T}_{reg}$ is an IPW estimator using weights that are 'calibrated' or 'tuned' (French: *calage*) so that the known population totals are estimated correctly.

Calage



Calibration

The general calibration problem: given a distance function $d(\cdot, \cdot)$, find calibration weights g_i minimizing

$$\sum_{R_i=1} d(g_i, 1)$$

subject to the calibration constraints

$$\sum_{i=1}^{N} x_i = \sum_{R_i=1} \frac{g_i}{\pi_i} x_i$$

Lagrange multiplier argument shows that $g_i = \eta(x_i\beta)$ for some $\eta()$, β; and γ can be computed by iteratively reweighted least squares.

For example, can choose $d(,)$ so that g_i are bounded below (and above).

[Deville *et al* JASA 1993; JNK Rao *et al*, Sankhya 2002; Lumley et al (Int Stat. Rev. 2011)]

Calibration

When the calibration model in x is saturated, the choice of $d(,)$ does not matter: calibration equates estimated and known category counts.

In this case calibration is also the same as estimating sampling probabilities with logistic regression, which also equates estimated and known counts.

Calibration to a saturated model gives the same analysis as pretending the sampling was stratified on these categories: post-stratification

Post-stratification is a much older method, and is computationally simpler, but calibration can make more use of auxiliary data.

Standard errors

Standard errors come from the regression formulation

$$\hat{T}_{reg} = \sum_{R_i=1} \frac{1}{\pi_i}(y_i - x_i\hat{\beta}) + \sum_{i=1}^{N} x_i\hat{\beta}$$

The variance of the second term is of smaller order and is ignored.

The variance of the first term is the usual Horvitz–Thompson variance estimator, applied to residuals from projecting y on the calibration variables.

Computing

`postStratify()` and `rake()` take marginal tables as input

`calibrate()` is more general, allowing continuous variables. It takes column totals of a design matrix as input (the latest version also allows lists of marginal tables, as in `rake`)

Post-stratification

Stratified sampling uses population data on the stratifying variable to add information to the survey, so that variation between strata doesn't show up in the standard error.

We often have population data that were not used to stratify

- The variable is useful for our analysis, but not for other analyses of the data.
- The variable does not appear on sampling lists, so can't be used to stratify.
- The sampling is clustered and the variable is defined for individuals.

Stratified sampling

Suppose we take a simple random sample of 100 schools from California, and end up with

```
> table(apisrs$stype)
 E  H  M
76  7 17
```

We could have obtained exactly the same data from a stratified random sample with stratum sizes $n_E = 76$, $n_H = 7$, $n_M = 17$.

This is true for any simple random sample: we can define a corresponding set of stratum sizes that would make the sample possible.

Conversely, any sample that could have occurred by stratified sampling could also have occured by simple random sampling.

Stratified sampling

We can slice up the set of all possible simple random samples into pieces corresponding to all possible sets of stratified random samples.

Every sample is in exactly one slice.

This mean we can convert a simple random sample into a stratified random sample by observing which slice it lies in, and adjusting the weights from N/n to N_h/n_h for that slice.

The adjustment of weights makes sure that we get exactly the right estimate for N_h, just as we would in a stratified sample.

In statistical terms, we condition on the observed n_h.

Efficiency gain

Post-stratification is not quite as good as stratification: Suppose the variance is the same in each stratum, so that sampling with equal π_i is optimal.

Our standard errors from the post-stratified sample are the same as if we had done a stratified random sample with $n_E = 76$, $n_H = 7$, $n_M = 17$.

But in a stratified random sample of 100 schools with equal π_i in each stratum we would choose $n_E = 71$, $n_H = 12$, $n_M = 17$, and get slightly smaller standard errors.

Efficiency gain

Also, the standard errors ignore the uncertainty that comes from having to estimate the sampling weights: N_h/n_h is random under post-stratification, fixed under stratification.

This uncertainty is negligible as long as n_h is not too small.

In contrast to stratification, post-stratification with very small strata is not efficient.

Example

```
> dsrs<-svydesign(id=~1,data=apisrs, fpc=~popsize)
> pop.types <- data.frame(stype=c("E","H","M"),
                Freq=c(4421,755,1018))
> dps<-postStratify(dsrs, strata=~stype, pop=pop.types)
> svymean(~enroll+stype, dsrs)
         mean      SE
enroll 586.24 38.9173
stypeE   0.76  0.0426
stypeH   0.07  0.0254
stypeM   0.17  0.0374
> svymean(~enroll+stype, dps)
            mean        SE
enroll 633.65869    34.864
stypeE   0.71376 1.780e-17
stypeH   0.12189 8.603e-18
stypeM   0.16435 6.002e-18
```


Simulation

```
one.sim<-function(){
srs_rows<-sample(6157,100) #no missing
dsrs<-svydesign(id=~1,fpc=~popsize,data=apipop[srs_rows,])
dps<-postStratify(dsrs, strata=~stype, pop=pop.types)

unstrat<-svytotal(~enroll, dsrs,na.rm=TRUE)
poststrat<-svytotal(~enroll, dps,na.rm=TRUE)
c(coef(unstrat),coef(poststrat))
}
manysim<-replicate(500,one.sim())
```

Simulation

```
> mean(manysim[,1]-3811472)
[1] -404056.3
> mean(manysim[,2]-3811472)
[1] -34256.03
```

```
> sd(manysim[,1])
[1] 94477.48
> sd(manysim[,2])
[1] 55961.21
```

Post-stratification gives more accurate estimates.

Clustering

If post-strata cut across clusters we can't just treat the data as a stratified multistage sample.

Post-stratification still works, but computations are done differently.

The variable is centered at the mean for the group, and the standard error is computed for the residuals.

$$Y_i = (Y_i - \bar{y}) + \bar{y}$$

This is valid when the post-strata are reasonably large, as the group mean has se^2 proportional to $1/n_h$, which can be neglected. It breaks down when n_h is too small.

Example:CA schools

Cluster sample of schools: all schools from 15 districts.

Can't stratify on school type: clusters contain multiple school types

Can post-stratify on school type

- gives exact estimates of numbers of schools of each type
- improves estimation for variables that differ by school type

Example:CA schools

```
> dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
> svymean(~api00, dclus1)
        mean     SE
api00 644.17 23.542
>  svytotal(~enroll, dclus1)
         total     SE
enroll 3404940 932235
> svytotal(~stype, dclus1)
         total      SE
stypeE 4873.97 1333.32
stypeH  473.86  158.70
stypeM  846.17  167.55

> pop.types <- data.frame(stype=c("E","H","M"), Freq=c(4421,755,1018))
> dclus1p<-postStratify(dclus1, ~stype, pop.types)
> summary(dclus1p)
1 - level Cluster Sampling design
With (15) clusters.
postStratify(dclus1, ~stype, pop.types)
Probabilities:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.01854 0.03257 0.03257 0.03040 0.03257 0.03257
```

Example:CA schools

```
> svytotal(~stype, dclus1p)
       total SE
stypeE  4421  0
stypeH   755  0
stypeM  1018  0
> svymean(~api00, dclus1p)
        mean     SE
api00 642.31 23.921
> svytotal(~enroll, dclus1p)
          total     SE
enroll 3680893 406293
```


Post-stratification and regression

The post-stratification estimate of a mean or total is exactly the regression estimator for a model that has indicator variables for all the groups.

The known population cell sizes for each group are used to compute the predicted values, and the variance of the residuals is used to compute the standard error estimate

```
> dclus1<-update(dclus1, middle=as.numeric(stype=="M"),
           high=as.numeric(stype=="H"))
> m<-svyglm(enroll~middle+high,design=dclus1)
> predict(m, total=6194,newdata=data.frame(middle=1018,high=755))
     link     SE
1 3680893 406293
> svytotal(~enroll, dclus1p)
         total     SE
enroll 3680893 406293
```

Post-stratification

Post-stratification requires the population joint distribution of all the post-stratum variables.

eg, post-stratifying on agegroup, race, sex requires counts for each age-race-sex combination.

- Post-strata may be too small for good estimation
- Joint distributions may not be available

Raking

Suppose we have population data on age and on income, but not jointly.

We could

- Post-stratify on age, ignoring income; or
- Post-stratify on income, ignoring age

Why not both: post-stratify on age, then on income?

Raking

Post-stratifying on income can affect the age totals and *vice versa*, but we can iterate until the estimates settle down.

If there are no zeros in the sample table, the estimates will converge. If there are zeros, the estimates may converge or may oscillate.

In the survey world this algorithm is called raking. In loglinear modelling, the same algorithm is called iterative proportional fitting.

Standard errors

For replicate weights the standard errors come from raking each set of replicates.

For standard errors based on the sampling design we iterate the procedure of centering the variables around the post-stratum mean.

Raking will typically give larger standard errors than post-stratification on the same variables: it uses less population information.

Computing

Using replicate weights that have already been raked does not require any special software.

The sampling weights in large surveys have often been raked, but the raking strata are not given. Analysing the data as if the weights were simply sampling weights is conservative..

R has a `rake` function, which works by calling the `postStratify` function iteratively.

SUDAAN can do raking, SPSS, SAS and Stata do not have it built in, but there are user-written macros available.

Example

`rclus1` is a cluster sample of 15 California school districts: 183 schools. Using jackknife weights (leave out one cluster).

```
> rclus1
Call: as.svrepdesign(dclus1)
Unstratified cluster jacknife (JK1) with 15 replicates.

> ## population marginal totals for each stratum
> pop.types <- data.frame(stype=c("E","H","M"),
            Freq=c(4421,755,1018))
> pop.schwide <- data.frame(sch.wide=c("No","Yes"),
           Freq=c(1072,5122))

> rclus1r <- rake(rclus1, list(~stype,~sch.wide),
                       list(pop.types, pop.schwide))
```

Example

```
> xtabs(~stype, apipop)
stype
   E    H    M
4421  755 1018
> svytable(~stype, rclus1r, round=TRUE)
stype
   E    H    M
4421  755 1018
> xtabs(~sch.wide, apipop)
sch.wide
  No  Yes
1072 5122
> svytable(~sch.wide, rclus1r, round=TRUE)
sch.wide
  No  Yes
1072 5122
```


Example

```
> ## joint totals don't correspond
> xtabs(~stype+sch.wide, apipop)
     sch.wide
stype   No  Yes
    E  472 3949
    H  334  421
    M  266  752

> svytable(~stype+sch.wide, rclus1r, round=TRUE)
     sch.wide
stype   No  Yes
    E  478 3943
    H  201  554
    M  393  625
```

Example

```
> ## but much closer than before raking
> svytable(~stype+sch.wide,rclus1,round=TRUE)
     sch.wide
stype   No  Yes
    E  406 4468
    H  102  372
    M  271  575
>## Some means
>   svymean(~comp.imp+enroll+stype, rclus1r)
                mean        SE
comp.impNo    0.31984    0.0167
comp.impYes   0.68016    0.0167
enroll      588.84405   74.8322
stypeE        0.71375 4.793e-05
stypeH        0.12189 1.465e-05
stypeM        0.16436 3.335e-05
```


Example

```
>   svymean(~comp.imp+enroll+stype, rclus1)
                  mean      SE
comp.impNo    0.273224  0.0303
comp.impYes   0.726776  0.0303
enroll      549.715847 50.4611
stypeE        0.786885  0.0514
stypeH        0.076503  0.0278
stypeM        0.136612  0.0332
```

More complex raking

It is possible to rake on any set of tables, even partly overlapping.

For example, suppose we have population tables of age by sex by region and age by ethnicity by region, and age by sex by ethnicity, but not age by sex by ethnicity by region.

We can still post-stratify iteratively on the three tables we have until the estimates converge.

More complex raking

```
pop.table <- xtabs(~stype+sch.wide,apipop)
pop.table2 <- xtabs(~stype+comp.imp,apipop)

rakeclus1r<-rake(dclus1,
                 sample=list(~stype+sch.wide, ~stype+comp.imp),
                 population=list(pop.table, pop.table2))
```

Raking and regression

Can approximately think of raking as a logistic regression model for the probability of being sampled. The sampling weights provide odds $\pi_i/(1-\pi_i)$, and raking modifies these to $g_i\pi_i/(1-\pi_i)$

Raking with just two marginal tables fits a logistic regression model with main effects and no interactions.

Post-stratification fits all the interactions.

Since the null model is true (π_i are the correct sampling probabilities), inference is valid for any model and we trade off the precision gain from adjusting the sampling to the precision loss from estimating more parameters.

More complicating raking procedures with overlapping tables correspond to more complicated logistic models with some interactions included.

Family Resources Survey

Data for Scotland, via PEAS at Napier University.

This is a set of 4695 observations of households in a cluster sample, clustered by postcode.

We can define a survey design and estimate the mean income for all households, households with children, and single-parent households.

The weights as supplied have already been raked, using council tax band and housing tenure type (rented, owned, etc), so the usual weighted analysis is conservative.

Raking the weights doesn't change the point estimates (because they are already raked) but allows R to take advantage of the increase in precision.

Family Resources Survey

```
> frs.des <- svydesign(id=~PSU, weights=~GROSS2, data=frs)
> svymean(~HHINC, subset(frs.des,  ADULTH==1 & DEPCHLDH>0),
+     deff=TRUE)
          mean       SE   DEff
HHINC 276.5555   8.4954 1.0101
> svymean(~HHINC, subset(frs.des,  DEPCHLDH>0),deff=TRUE)
         mean      SE   DEff
HHINC 611.205  15.599 1.7622
> svymean(~HHINC, frs.des,deff=TRUE)
         mean      SE   DEff
HHINC 483.091  10.639 2.9066
```


Family Resources Survey

```
> pop.ctband <- data.frame(CTBAND=1:9,
+         Freq=c(515672, 547548, 351599, 291425,
+                         266257, 147851, 87767, 9190, 19670))
> pop.tenure <- data.frame(TENURE=1:4,
+         Freq=c(1459205,493237, 128189, 156348))
> frs.raked <- rake(frs.des, sample=list(~CTBAND, ~TENURE),
+         population=list(pop.ctband, pop.tenure))
> svymean(~HHINC, frs.raked)
        mean     SE
HHINC 483.09 7.5781
> svymean(~HHINC, subset(frs.raked, DEPCHLDH>0))
        mean     SE
HHINC 611.21 12.541
> svymean(~HHINC, subset(frs.raked, DEPCHLDH>0 & ADULTH==1))
        mean     SE
HHINC 276.56 8.4417
```

Compared to regression

```
> frs.des <- update(frs.des,
+   ctband1=CTBAND==1, ctband2=CTBAND==2,
+   ctband3=CTBAND==3, ctband4=CTBAND==4,ctband5=CTBAND==5,
+   ctband6=CTBAND==6,ctband7=CTBAND==7, ctband8=CTBAND==8,
+   ctband9=CTBAND==9,
+   tenure1=TENURE==1, tenure2=TENURE==2, tenure3=TENURE==3,
+   tenure4=TENURE==4)
> m <- svyglm(HHINC~ctband2+ctband3+ctband4+ctband5+ctband6
+    +ctband7+ctband8+ctband9+tenure2+tenure3+tenure4,
+    design=frs.des)
> totals <- c(2236979, 547548, 351599, 291425, 266257, 147851,
+     87767, 9190, 19670, 493237, 128189, 156348)
> names(totals) <- c("(Intercept)", "ctband2", "ctband3",
+     "ctband4", "ctband5", "ctband6", "ctband7", "ctband8",
+     "ctband9", "tenure2", "tenure3", "tenure4")
> totals <- as.data.frame(t(totals))
```


Compared to regression

```
> totincome <- predict(m, newdata=totals, total= 2236979)
> svycontrast(totincome, 1/2236979)
         contrast     SE
contrast   483.09 7.5046
```

Calibration and regression

- Regression estimators of the population total
- Calibration of weights

Estimating totals

A fitted linear regression model

$$E[Y_i] = \hat{\alpha} + \hat{\beta} x_i$$

gives predictions for y when x is known.

If x is known for the whole population we can predict y for the whole population and estimate the total

$$\hat{T}_y = \sum_{i=1}^{N} \left(\hat{\alpha} + \hat{\beta} x_i \right)$$

In fact this simplifies to

$$\hat{T}_y = N\hat{\alpha} + \hat{\beta} \sum_{i=1}^{N} x_i$$

so we only need the population total of x, not the individual xs.

Calibration

Three basic types of calibration

- Linear (or regression) calibration: identical to regression estimator
- Raking: multiplicative model for weights, guarantees $g_i > 0$
- Logit calibration: logit link for weights, popular in Europe, provides upper and lower bounds for g_i

Calibration

Upper and lower bounds for g_i can also be specified for linear and raking calibration (these may not be achievable, but we try).

Trimming of weights is available for when the desired bounds can't be achieved.

Calibration to cluster totals is allowed, as is enforcing constant calibration adjustments within clusters (eg, to make sampling weights the same for all members of a household).

The user can specify other calibration loss functions (eg, mathematical statisticians keep asking about Hellinger distance).

Calibration

The `calibrate()` function takes three main arguments

- a survey design object
- a model formula describing the design matrix of auxiliary variables
- a vector giving the column sums of this design matrix in the population.

and additional arguments describing the type of calibration.

Calibration

```
> data(api)
> dclus1<-svydesign(id=~dnum, weights=~pw, data=apiclus1, fpc=~fpc)
> pop.totals<-c(`(Intercept)`=6194, stypeH=755, stypeM=1018)

> (dclus1g<-calibrate(dclus1, ~stype, pop.totals))
1 - level Cluster Sampling design
With (15) clusters.
calibrate(dclus1, ~stype, pop.totals)
> svymean(~api00, dclus1g)
        mean     SE
api00 642.31 23.921
>  svymean(~api00,dclus1)
        mean     SE
api00 644.17 23.542
```

Calibration

```
> svytotal(~enroll, dclus1g)
         total     SE
enroll 3680893 406293
> svytotal(~enroll,dclus1)
         total     SE
enroll 3404940 932235
> svytotal(~stype, dclus1g)
       total        SE
stypeE  4421 1.118e-12
stypeH   755 4.992e-13
stypeM  1018 1.193e-13
```


Calibration

```
> (dclus1g3 <- calibrate(dclus1, ~stype+api99,
                         c(pop.totals, api99=3914069)))
1 - level Cluster Sampling design
With (15) clusters.
calibrate(dclus1, ~stype + api99, c(pop.totals, api99 = 3914069))
> svymean(~api00, dclus1g3)
        mean     SE
api00 665.31 3.4418
> svytotal(~enroll, dclus1g3)
         total     SE
enroll 3638487 385524
> svytotal(~stype, dclus1g3)
       total        SE
stypeE  4421 1.179e-12
stypeH   755 4.504e-13
stypeM  1018 9.998e-14
```

Calibration

```
> range(weights(dclus1g3)/weights(dclus1))
[1] 0.4185925 1.8332949

> (dclus1g3b <- calibrate(dclus1, ~stype+api99,
         c(pop.totals, api99=3914069),bounds=c(0.6,1.6)))
1 - level Cluster Sampling design
With (15) clusters.
calibrate(dclus1, ~stype + api99, c(pop.totals, api99 = 3914069),
    bounds = c(0.6, 1.6))

> range(weights(dclus1g3b)/weights(dclus1))
[1] 0.6 1.6
```


Calibration

```
> svymean(~api00, dclus1g3b)
        mean     SE
api00 665.48 3.4184
> svytotal(~enroll, dclus1g3b)
         total     SE
enroll 3662213 378691
> svytotal(~stype, dclus1g3b)
       total        SE
stypeE  4421 1.346e-12
stypeH   755 4.139e-13
stypeM  1018 8.238e-14
```

Calibration

```
> (dclus1g3c <- calibrate(dclus1, ~stype+api99, c(pop.totals,
+     api99=3914069), calfun="raking"))
1 - level Cluster Sampling design
With (15) clusters.
calibrate(dclus1, ~stype + api99, c(pop.totals, api99 = 3914069),
    calfun = "raking")
> range(weights(dclus1g3c)/weights(dclus1))
[1] 0.5342314 1.9947612
> svymean(~api00, dclus1g3c)
        mean     SE
api00 665.39 3.4378
```


Calibration

```
> (dclus1g3d <- calibrate(dclus1, ~stype+api99, c(pop.totals,
+     api99=3914069), calfun="logit",bounds=c(0.5,2.5)))
1 - level Cluster Sampling design
With (15) clusters.
calibrate(dclus1, ~stype + api99, c(pop.totals, api99 = 3914069),
    calfun = "logit", bounds = c(0.5, 2.5))
> range(weights(dclus1g3d)/weights(dclus1))
[1] 0.5943692 1.9358791
> svymean(~api00, dclus1g3d)
        mean     SE
api00 665.43 3.4325
```

Raking-like syntax

```
pop.table <- xtabs(~stype+sch.wide,apipop)
pop.table2 <- xtabs(~stype+comp.imp,apipop)

rakeclus1r<-rake(dclus1,
                 sample=list(~stype+sch.wide, ~stype+comp.imp),
                 population=list(pop.table, pop.table2))
calclus1r<-calibrate(dclus1,
             formula=list(~stype+sch.wide, ~stype+comp.imp),
             population=list(pop.table, pop.table2),
             calfun="raking")
```


Trimming

```
> dclus1tr <- calibrate(dclus1, ~stype+api99,
        c(pop.totals, api99=3914069),bounds=c(0.5,2),
        trim=c(2/3,3/2))
37 weights were trimmed
> svymean(~stype+api00, design=dclus1tr)
            mean     SE
stypeE   0.72598 0.0000
stypeH   0.11057 0.0000
stypeM   0.16345 0.0000
api00  662.72890 3.4388
```

Also `trimWeights` to trim weights rather than calibration adjustments.

Types of calibration

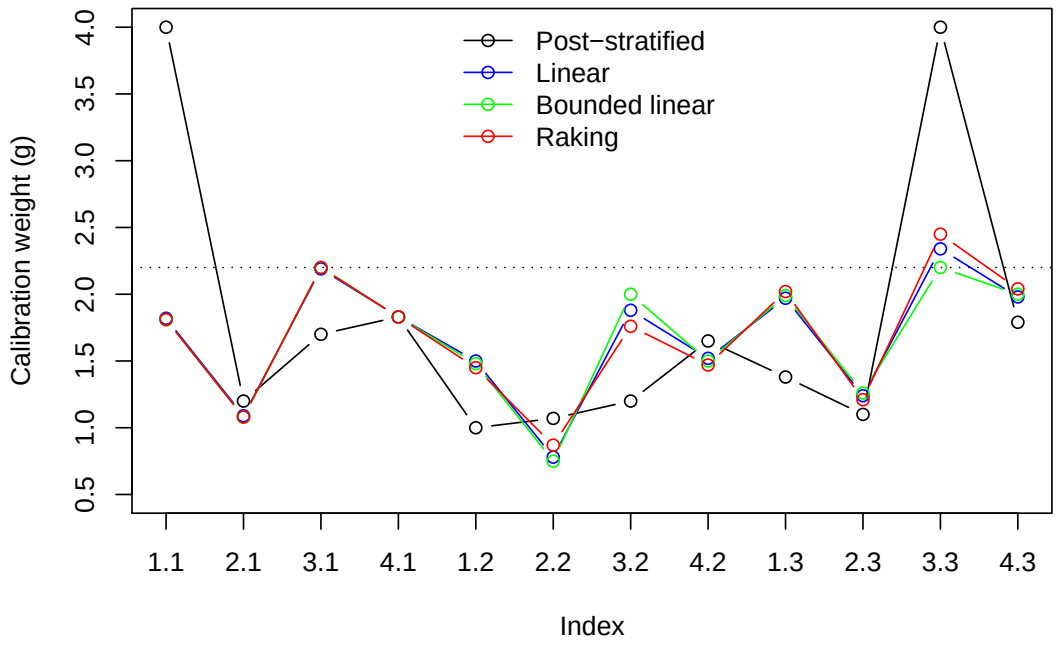
Post-stratification allows much more flexibility in weights, in small samples can result in very influential points, loss of efficiency.

Calibration allows for less flexibility (cf stratification vs regression for confounding)

Different calibration methods make less difference

Example from Kalton & Flores-Cervantes (J. Off. Stat, 2003): a 3×4 table of values.

Types of calibration



Calibration to estimate β

Calibration on raw variables doesn't help with estimating regression coefficients

Reason:

$$\widehat{\beta} = \beta + \sum_{i \in \text{sample}} (X^T W X)^{-1} x_i w_i (y_i - \mu_i)$$

and the raw variables x and y are nearly uncorrelated with $x_i w_i (y_i - \mu_i)$.

Need to calibrate on variables correlated with $x_i w_i (y_i - \mu_i)$, more generally, correlated with the influence function for the parameter of interest.

Influence functions

If we have another estimator for the same parameter, its influence functions must be correlated (Convolution Theorem)

So: use auxiliary variables z to get imputed $\widehat{x}_i$ for everyone in population, regress y on $\widehat{x}$, use the influence functions from that model.

More reasonable in two-phase cohort design: use z measured on a whole cohort to impute x measured on a subsample.

Influence functions

Extreme example: California schools.

We want to fit `api00~ell+mobility+emer`

Assume that the predictor variables and `api99` are available for the whole population.

Calibration on the predictor variables has little impact on precision; calibration on `api99` reduces standard error only for intercept.

Fit a population model `api99~ell+mobility+emer` and calibrate on **its influence functions** to get large gains in precision

Influence functions

```
> m0 <- svyglm(api00~ell+mobility+emer, clus1)
> var_cal <- calibrate(clus1, formula=~api99+ell+mobility+emer,
      pop=c(6194,3914069, 141685, 106054, 70366),
      bounds=c(0.1,10))
> m1<-svyglm(api00~ell+mobility+emer, design=var_cal)
> popmodel <- glm(api99~ell+mobility+emer, data=apipop,
     na.action=na.exclude)
> inffun <- dfbeta(popmodel)
> index <- match(apiclus1$snum, apipop$snum)
> clus1if <- update(clus1, ifint = inffun[index,1],
     ifell=inffun[index,2], ifmobility=inffun[index,3],
     ifemer=inffun[index,4])
> if_cal <- calibrate(clus1if,
    formula=~ifint+ifell+ifmobility+ifemer,
    pop=c(6194,0,0,0,0))
> m2<-svyglm(api00~ell+mobility+emer, design=if_cal)
```

Influence functions

```
> coef(summary(m0))
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 780.459500 30.0210123 25.997108 3.156974e-11
ell          -3.297892  0.4689026 -7.033215 2.173478e-05
mobility     -1.445370  0.7342887 -1.968395 7.473627e-02
emer         -1.814215  0.4233504 -4.285374 1.287085e-03

> coef(summary(m1))
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 785.408240 13.7640081 57.062466 5.912274e-15
ell          -3.273108  0.6242978 -5.242864 2.756024e-04
mobility     -1.464732  0.6651257 -2.202188 4.989506e-02
emer         -1.676541  0.3742041 -4.480284 9.309647e-04
```


Influence functions

```
> coef(summary(m2))
              Estimate Std. Error    t value     Pr(>|t|)
(Intercept) 790.631553  5.8409844 135.359298 4.480786e-19
ell          -3.260976  0.1300765 -25.069679 4.678967e-11
mobility     -1.405554  0.2247022  -6.255187 6.214930e-05
emer         -2.240431  0.2150534 -10.418024 4.902863e-07
```

More realistic example

Data from National Wilms' Tumor Study, via Norm Breslow.

Histology (cell weirdness) is hard to assess; NWTS central lab pathologist is better than anyone else.

Suppose we had all tumors assessed by local hospital lab and a subsample reassessed by central lab.

Use calibration to bring in information from outside the subsample (more model-robust than multiple imputation)

More realistic example

Two-phase sampling

- superpopulation sampling at phase 1, modelled as SRS with replacement
- stratified sampling without replacement at phase 2

`twophase()` function is similar to `svydesign`, but has two of everything, and a `subset` argument specifying which observations are in phase 2.

More realistic example

- Step 1: impute central-lab histology from local institutional histology and other variables
- Step 2: Fit the desired Cox model to the whole sample using just the imputed data
- Step 3: Extract the influence functions and calibrate using them (raking, to avoid negative weights)
- Step 4: Fit the model to the calibrated subsample

More realistic example

```
impmodel <- glm(histol~instit+I(age>10)+I(stage==4)*study,
  data=nwts, subset=in.subsample, family=binomial)
nwts$imphist <- predict(impmodel, newdata=nwts, type="response")

ifmodel <- coxph(Surv(trel,relaps)~imphist*age+I(stage>2)*tumdiam,
  data=nwts)
inffun <- resid(ifmodel, "dfbeta")
colnames(inffun) <- paste("if",1:6,sep="")

nwts_if <- cbind(nwts, inffun)
if_design <- twophase(id = list(~1, ~1), subset = ~in.subsample,
  strata = list(NULL, ~interaction(instit, relaps)),
  data = nwts_if)
if_cal <- calibrate(if_design,  phase=2,  calfun="raking"
  ~if1+if2+if3+if4+if5+if6+relaps*instit)
```

More realistic example

```
m1 <- svycoxph(Surv(trel, relaps)~histol*age+I(stage>2)*tumdiam,
  design=nwts_design)
m2 <- svycoxph(Surv(trel, relaps)~histol*age+I(stage>2)*tumdiam,
  design=if_cal)
```

Result: phase-two contribution to variance is eliminated for variables other than `histol`, reduced proportionally to imputation accuracy for `histol`.

Same gains as multiple imputation, but would be valid under model misspecification.

More realistic example

	Two-phase sample			Full data
	sampling weights	raked	direct imputation	
Coefficient estimate				
histology	1.808	2.113	2.108	1.932
age	0.055	0.101	0.101	0.096
stage > 2	1.411	1.435	1.432	1.389
tumor diameter	0.043	0.061	0.061	0.058
histology:age	-0.116	-0.159	-0.159	-0.144
stage > 2:diameter	-0.074	-0.084	-0.083	-0.079
Standard error				
histology	0.221	0.171	0.174	0.157
age	0.023	0.014	0.016	0.016
stage > 2	0.361	0.276	0.249	0.250
tumor diameter	0.021	0.016	0.014	0.014
histology:age	0.054	0.039	0.040	0.035
stage > 2:diameter	0.030	0.022	0.020	0.020

That's all, folks

Any (more) questions?

Bonus track: multiply-imputed data

Using multiply-imputed data

Multiple imputation of missing data: fit a model to the data, simulate multiple possibilities for the missing data from the predictive distribution of the model. (Rubin, 1978)

Do the same analysis to each completed data set

Point estimate is the average of the point estimates

Variance estimate is the average variance + variance between point estimates.

Simple approach is technically valid only for 'proper' imputations including posterior uncertainty in model parameters, which is inefficient. [Wang & Robins, Bka 1998]

Using multiply-imputed data

Need code to do repeated analysis, combine results.

- `imputationList()` wraps a list of data frames or database tables
- `svydesign()` can take an `imputationList` as the data argument to give a set of designs.
- `with(designs, expr)` runs `expr`, with `design=` each one of the designs in turn
- `MIcombine()` combines the results.

NHANES III imputations

```
> library(mitools)
> library(RSQLite)
> impdata <- imputationList(c("set1","set2","set3","set4","set5"),
           dbtype="SQLite", dbname="~/nhanes/imp.db")
> impdata
MI data with 5 datasets
Call: imputationList(c("set1", "set2", "set3", "set4", "set5"),
    dbtype = "SQLite", dbname = "~/nhanes/imp.db")
> designs <- svydesign(id=~SDPPSU6, strat=~SDPSTRA6,
    weight=~WTPFQX6, data=impdata, nest=TRUE)
> designs
DB-backed Multiple (5) imputations: svydesign(id = ~SDPPSU6,
    strat = ~SDPSTRA6, weight = ~WTPFQX6,
    data = impdata, nest = TRUE)
```


NHANES III imputations

```
> designs<-update(designs,
    age=ifelse(HSAGEU==1, HSAGEIR/12, HSAGEIR))
> designs<-update(designs,
    agegp=cut(age,c(20,40,60,Inf),right=FALSE))
> res <- with(subset(designs, age>=20),
    svyby(~BDPFNDMI, ~agegp+HSSEX, svymean))
> summary(MIcombine(res))
Multiple imputation results:
    with(subset(designs, age >= 20), svyby(~BDPFNDMI,
         ~agegp + HSSEX, svymean, design = .design))
    MIcombine.default(res)
              results          se    (lower    upper) missInfo
[20,40).1  0.9355049 0.003791945 0.9279172 0.9430926     28 %
[40,60).1  0.8400738 0.003813224 0.8325802 0.8475674     10 %
[60,Inf).1 0.7679224 0.004134875 0.7598032 0.7760416      8 %
[20,40).2  0.8531107 0.003158246 0.8468138 0.8594077     26 %
[40,60).2  0.7839377 0.003469386 0.7771144 0.7907610     11 %
[60,Inf).2 0.6454393 0.004117235 0.6370690 0.6538096     38 %
```