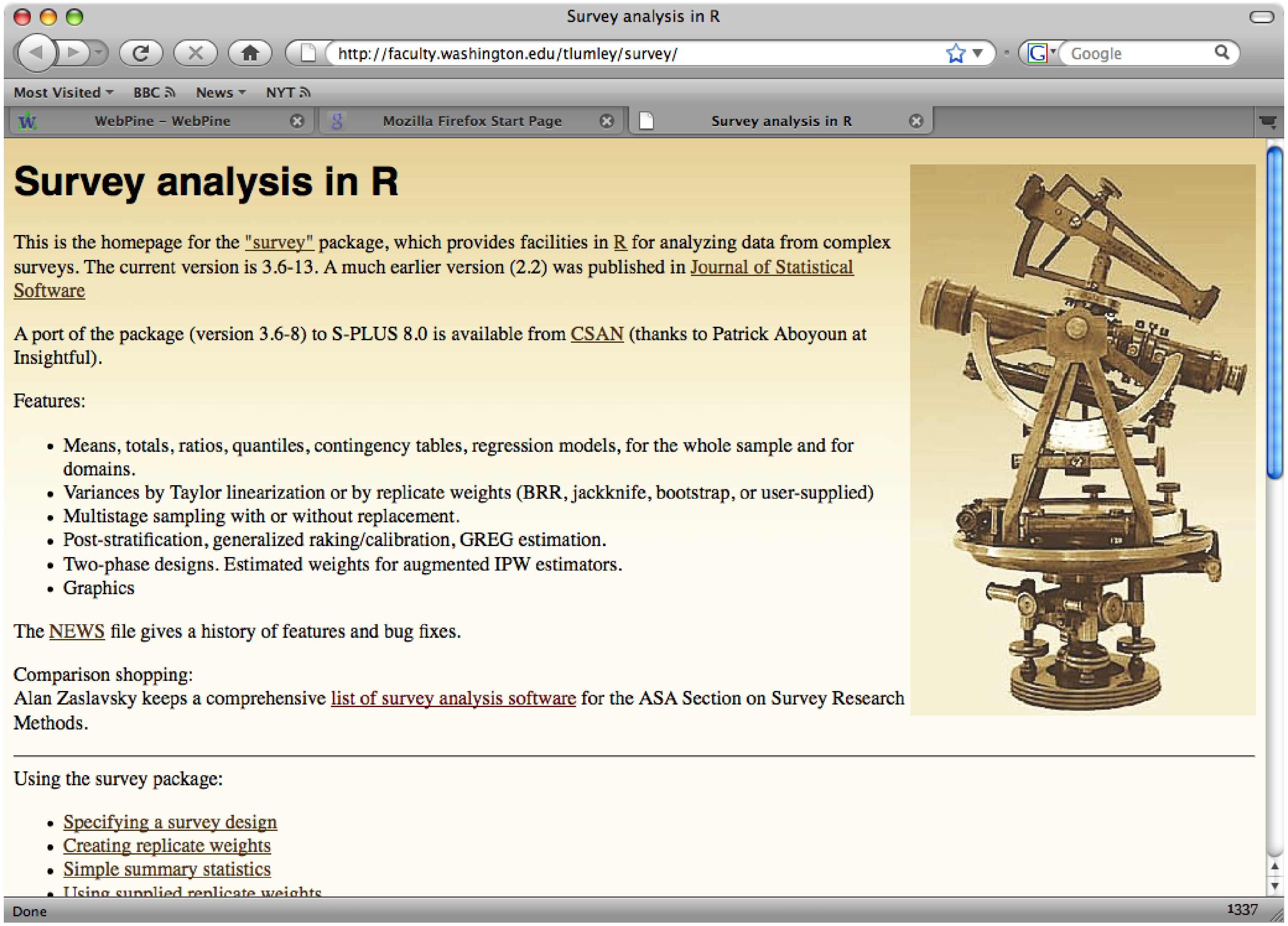
Complex sampling and R.

Thomas Lumley

Biostatistics
University of Washington

Census Bureau — 2009–9–10

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 3.6-13. A much earlier version (2.2) was published in Journal of Statistical Software

A port 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, 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.
- Post-stratification, generalized raking/calibration, GREG estimation.
- Two-phase designs. Estimated weights for augmented IPW estimators.
- Graphics

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.

Using the survey package:

- Specifying a survey design
- Creating replicate weights
- Simple summary statistics
- Using supplied replicate weights

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

R Survey package

Version 3.16 is current, containing approximately 9000 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.

A book, *Complex Surveys: a guide to analysis using R*, is in press at Wiley.

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

Outline

- Describing survey designs: `svydesign()`
- Database-backed designs
- Summary statistics: mean, total, quantiles, design effect
- Tables of summary statistics, domain estimation.
- Contigency tables: `svychisq()`, `svyloglin()`
- Graphics: histograms, hexbin scatterplots, smoothers.
- Regression modelling: `svyglm()`, `svyolr()`,
- Calibration
- Multiply-imputed data

Not on agenda: two-phase subsampling designs, PPS without replacement designs.

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.

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.

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.

Basic estimation ideas

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

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

is

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

Standard errors follow from formulas for the variance of a sum: main complication is that we do need to know $\mathrm{cov}[R_i, R_j]$.

Basic estimation ideas

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

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

then a design-based estimate will solve

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

Standard errors come from the delta method or from resampling (actually reweighting).

Theoretical details can become tricky, especially if $U_i()$ is not a function of just one observation.

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
- `fpc`1 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

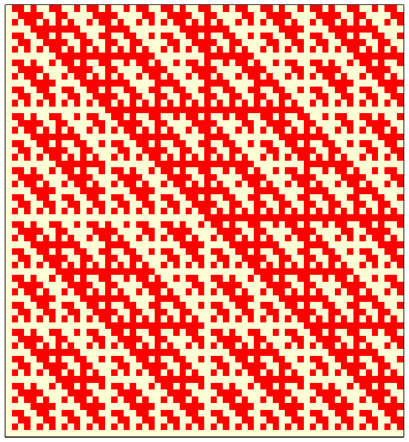
California Health Interview Survey has 50 sets of replicate weights instead of design information

```
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)
```

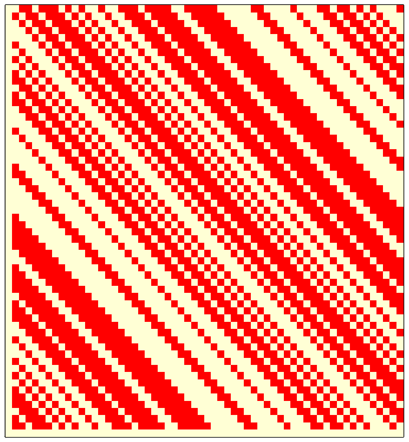
Can also create replicate weights with `as.svrepdesign()`, using jackknife, bootstrap, or BRR.

BRR: R knows Hadamard matrices

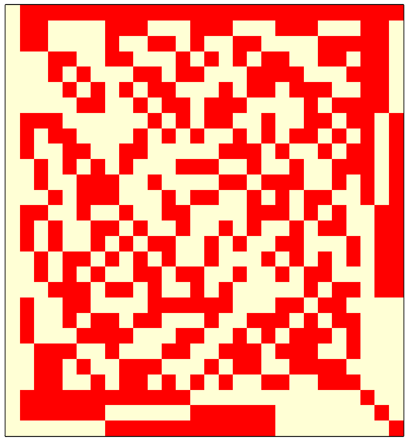
Sylvester: $64 = 2^6$



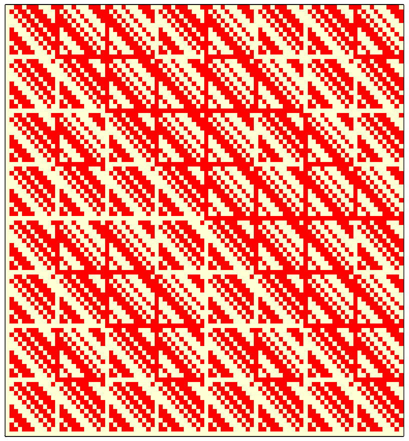
Paley: $60 = 59 + 1$



Stored: 28



Constructed: $96 = 2^3 \times (11 + 1)$



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 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) is also available.

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
```

Graphics

When complex sampling designs are analyzed with regression models it is more important to have good exploratory analysis methods.

Problems with survey data

- large data
- unequal weights.

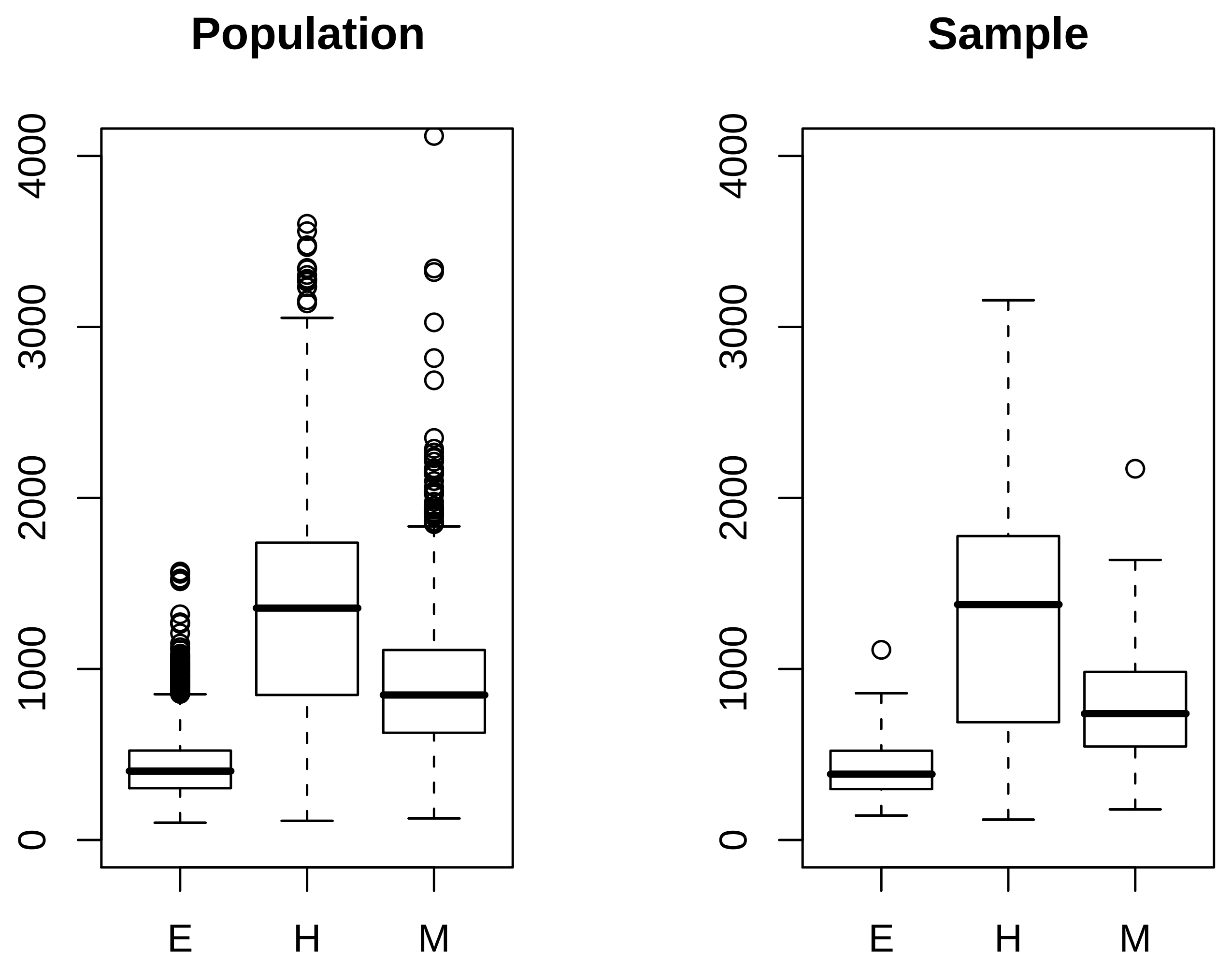
Estimating populations

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

We can substitute the survey estimates: boxplots use quantiles, histograms and barplots need tables. eg boxplot of enrollment by school type (Elementary/Middle/High)

```
svyboxplot(enroll~stype, design=srs)
```

Estimating populations



R does maps

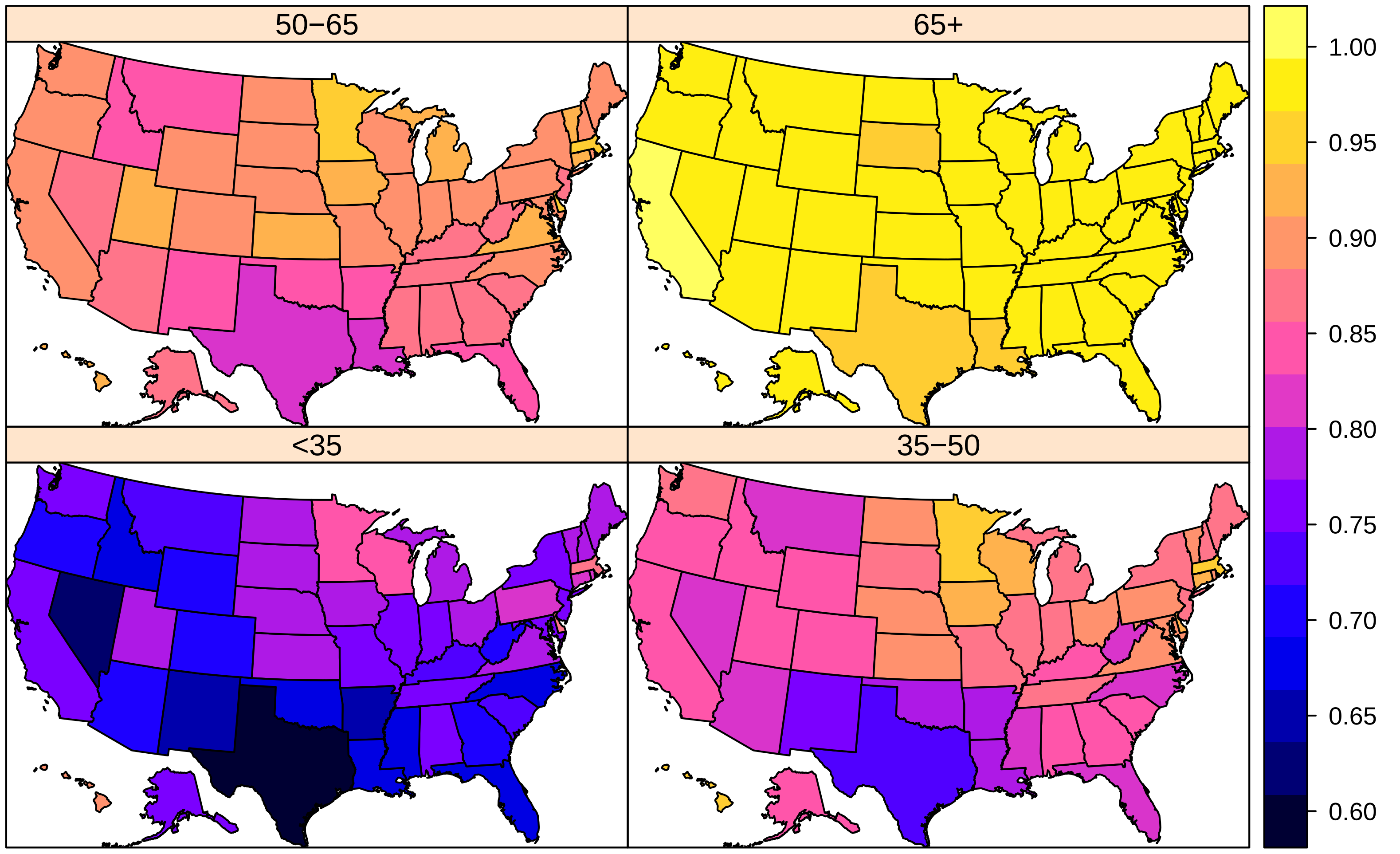
`maptools` and `sp` packages handle geographical data (eg ArcGIS shapefiles)

Estimate regional summaries with `svyby()`, merge output with GIS data, and then draw a map.

Example: health insurance by age and state, from BRFSS 2007.

GIS data for state outlines also come from BRFSS web site.

Insurance coverage



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.

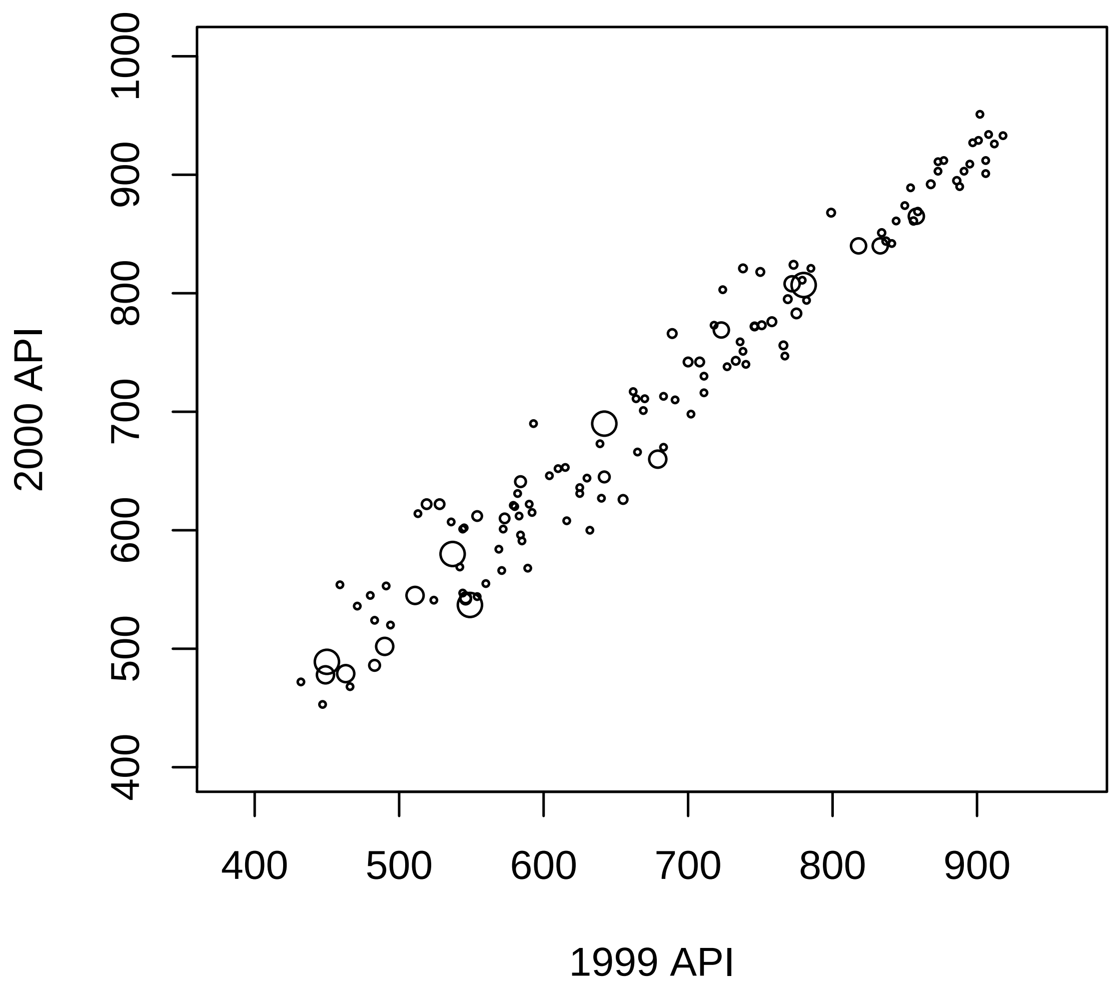
Another is transparent color: opacity for each point is proportional to sampling weight, giving a density estimation effect.

Bubble plots work well with small data sets, badly with large data sets.

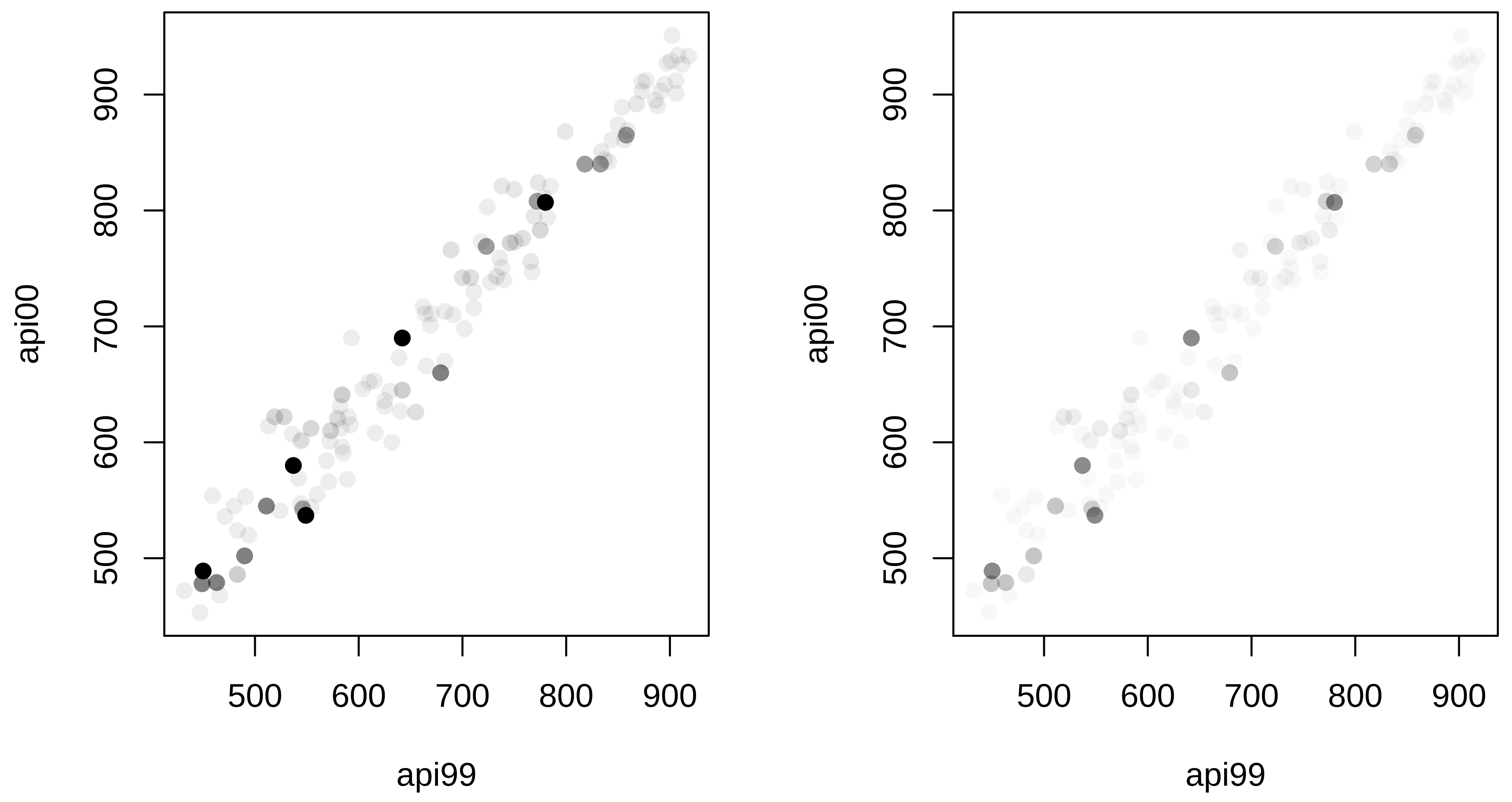
Transparency works well with large data sets, badly with small data sets.

Transparency allows color.

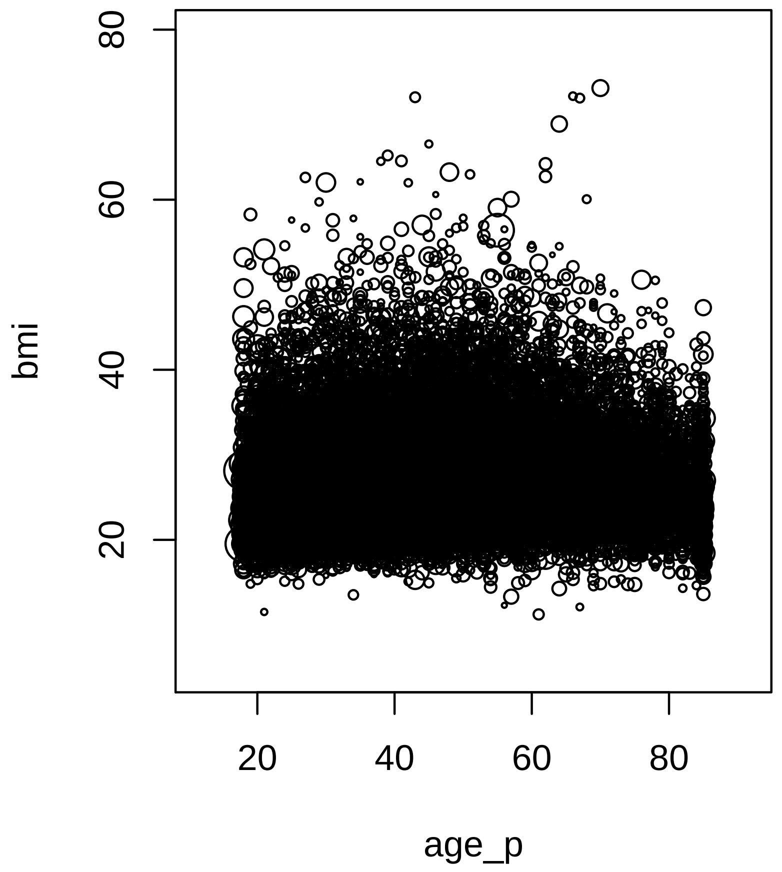
Scatterplots

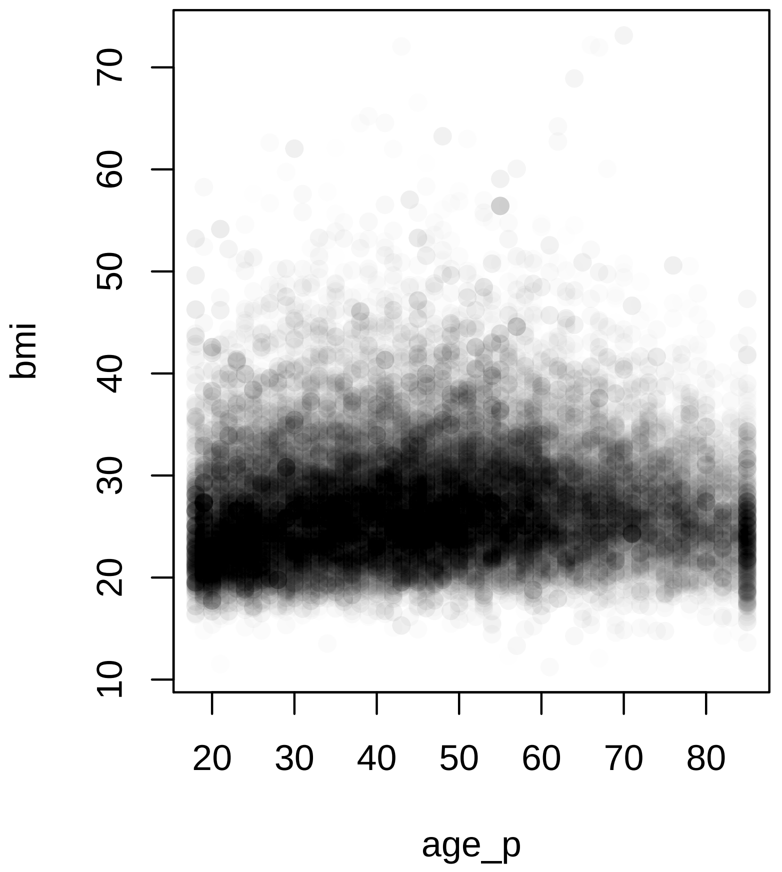


Scatterplots



Scatterplots





Binning and smoothing

Hexagonal binning divides the plotting area into hexagons, counts the number in each hexagon [Carr et al, 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.

In R: `svyplot(bmi~age_p, design=nhis, style="hex")}`

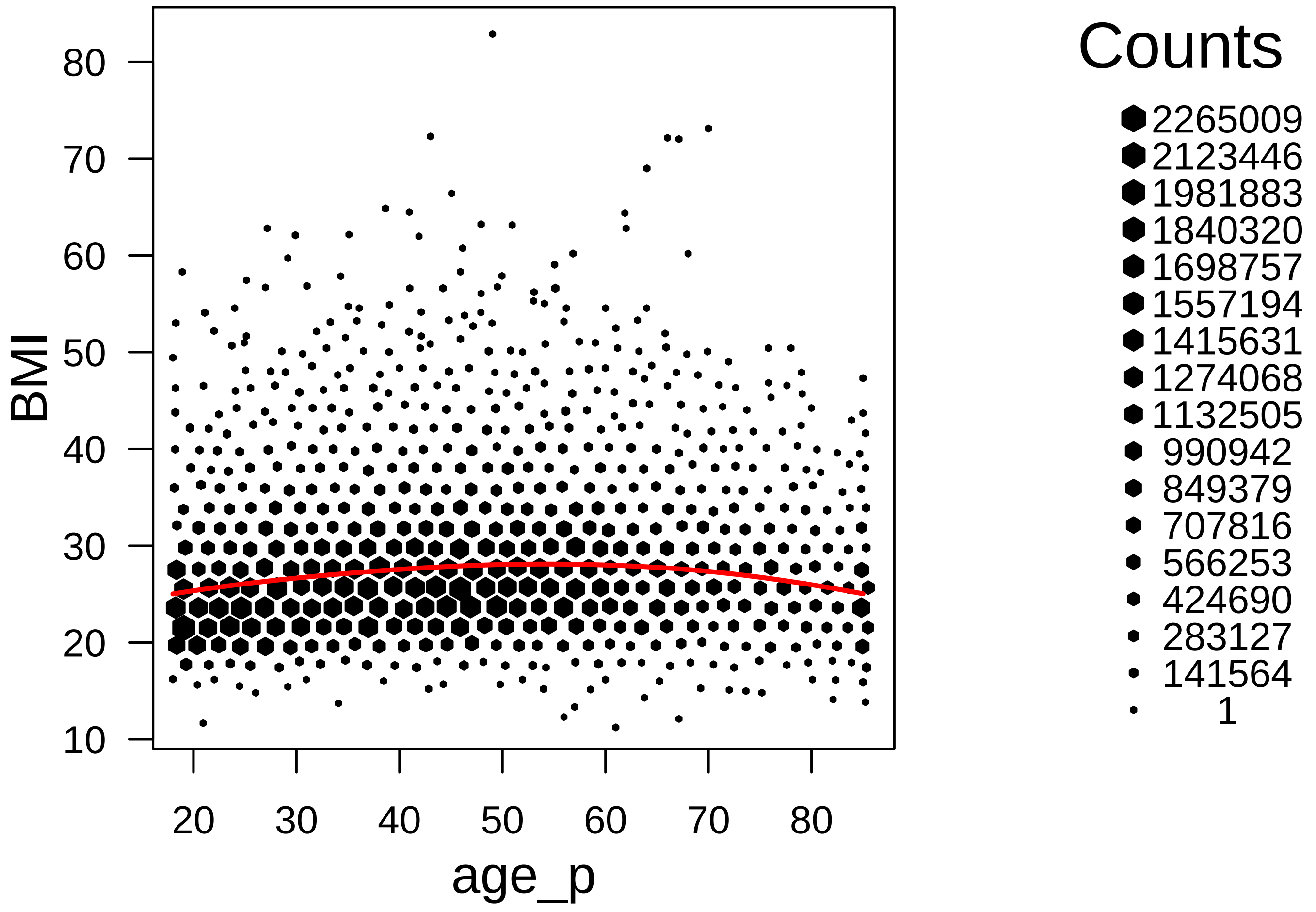
Binning and smoothing

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. `svysmooth()` has a binned kernel smoother for the mean and a quantile regression spline smoother for quantiles.

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

```
smth <- svysmooth(api00~api99, stratrs, bandwidth=40)
plot(smth)
```

Binning and smoothing



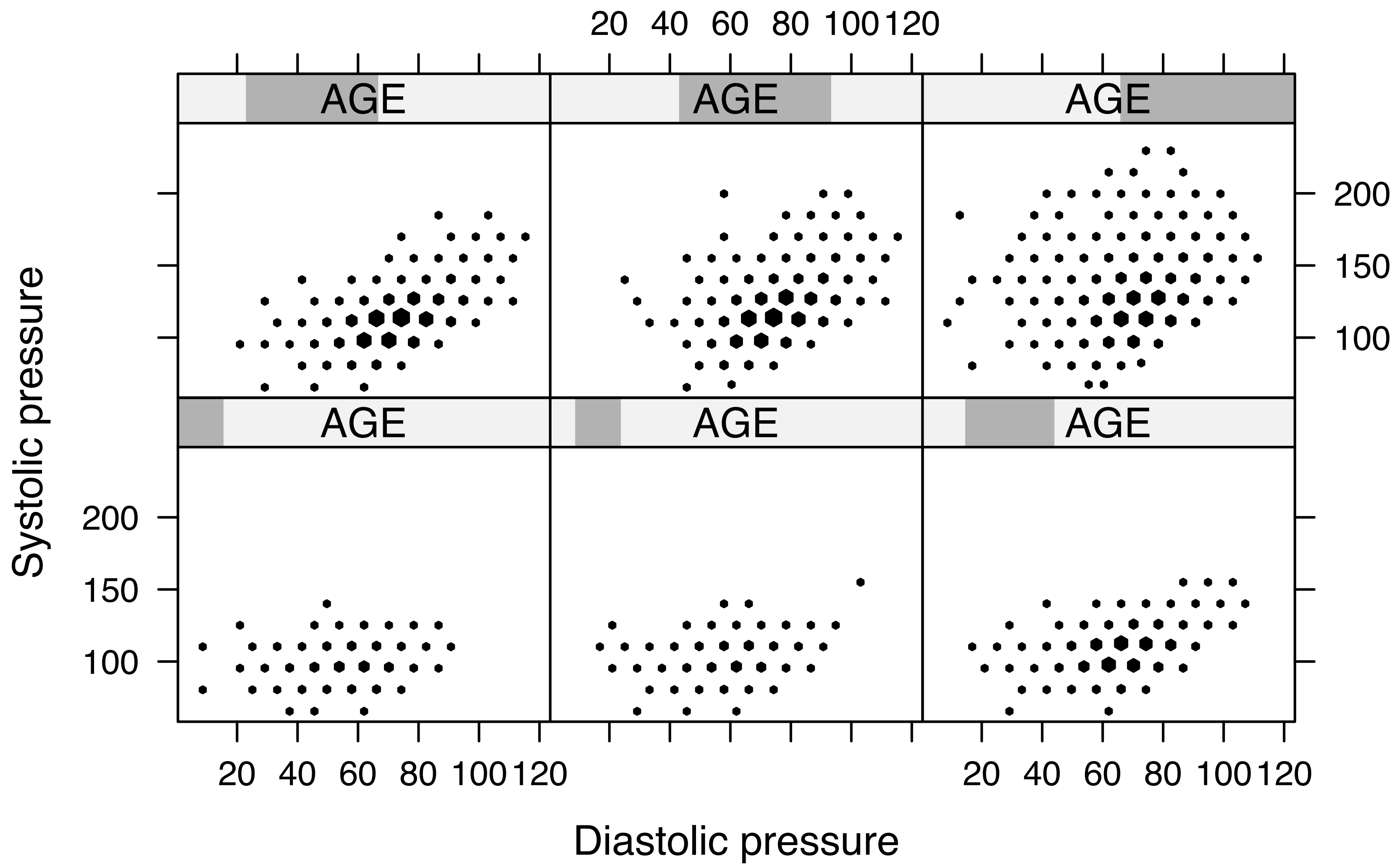
Conditioning plots

`svycoplot()` does conditioning plots, which can either use hexagonal binning or transparency

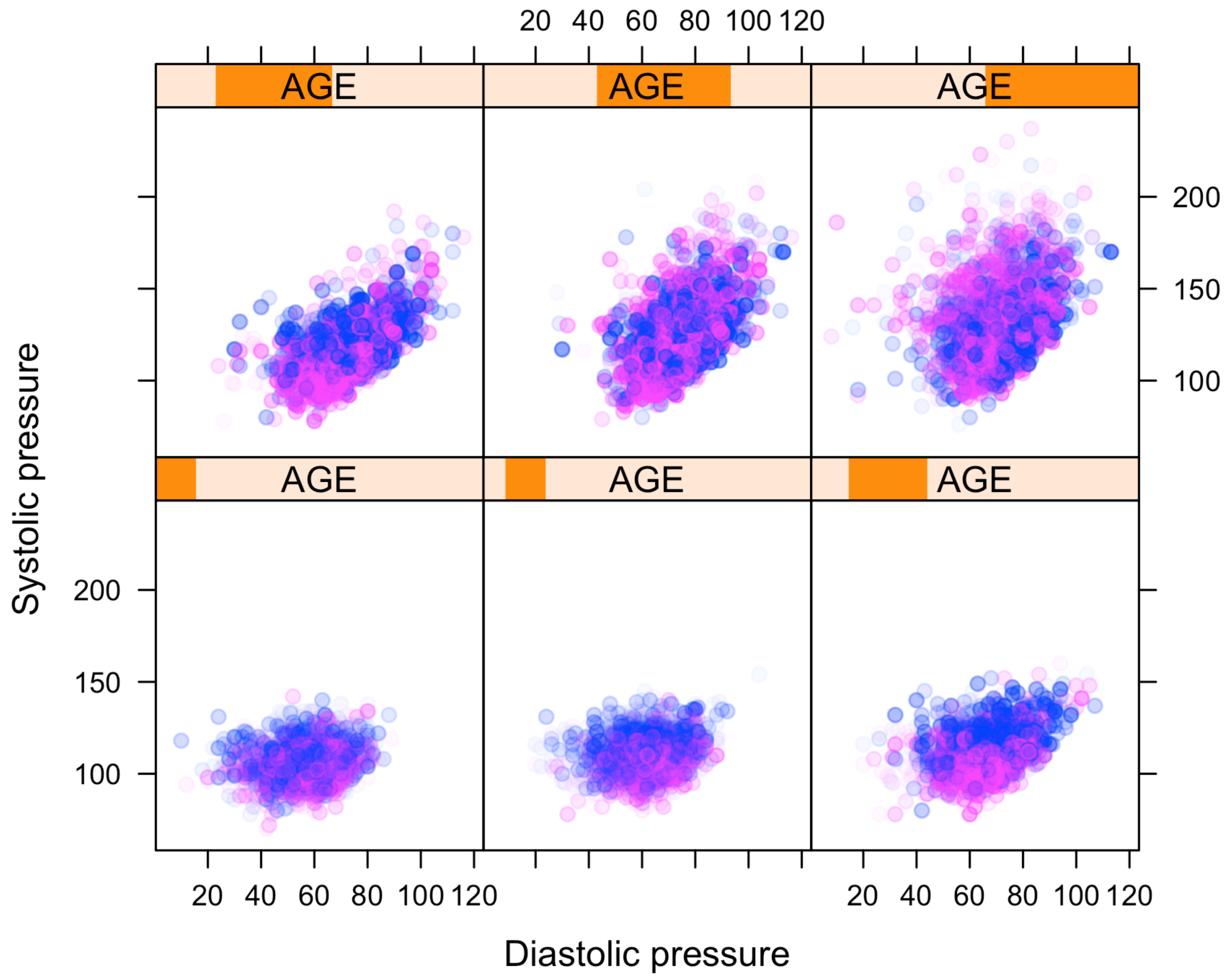
```
svycoplot(BPXSAR~BPXDAR|equal.count(RIDAGEMN), style="hex",
  design=subset(dhanes,BPXDAR>0), xbins=20,
  strip=strip.custom(var.name="AGE"),
  xlab="Diastolic pressure",ylab="Systolic pressure")

svycoplot(BPXSAR~BPXDAR|equal.count(RIDAGEMN), style="trans",
   design=subset(dhanes,BPXDAR>0),
   strip=strip.custom(var.name="AGE"),
   xlab="Diastolic pressure",ylab="Systolic pressure",
   basecol=function(d) ifelse(d$RIAGENDR==2,"magenta","blue"))
```

Conditioning plots



Conditioning plots



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 (no std errors on survival curves yet)

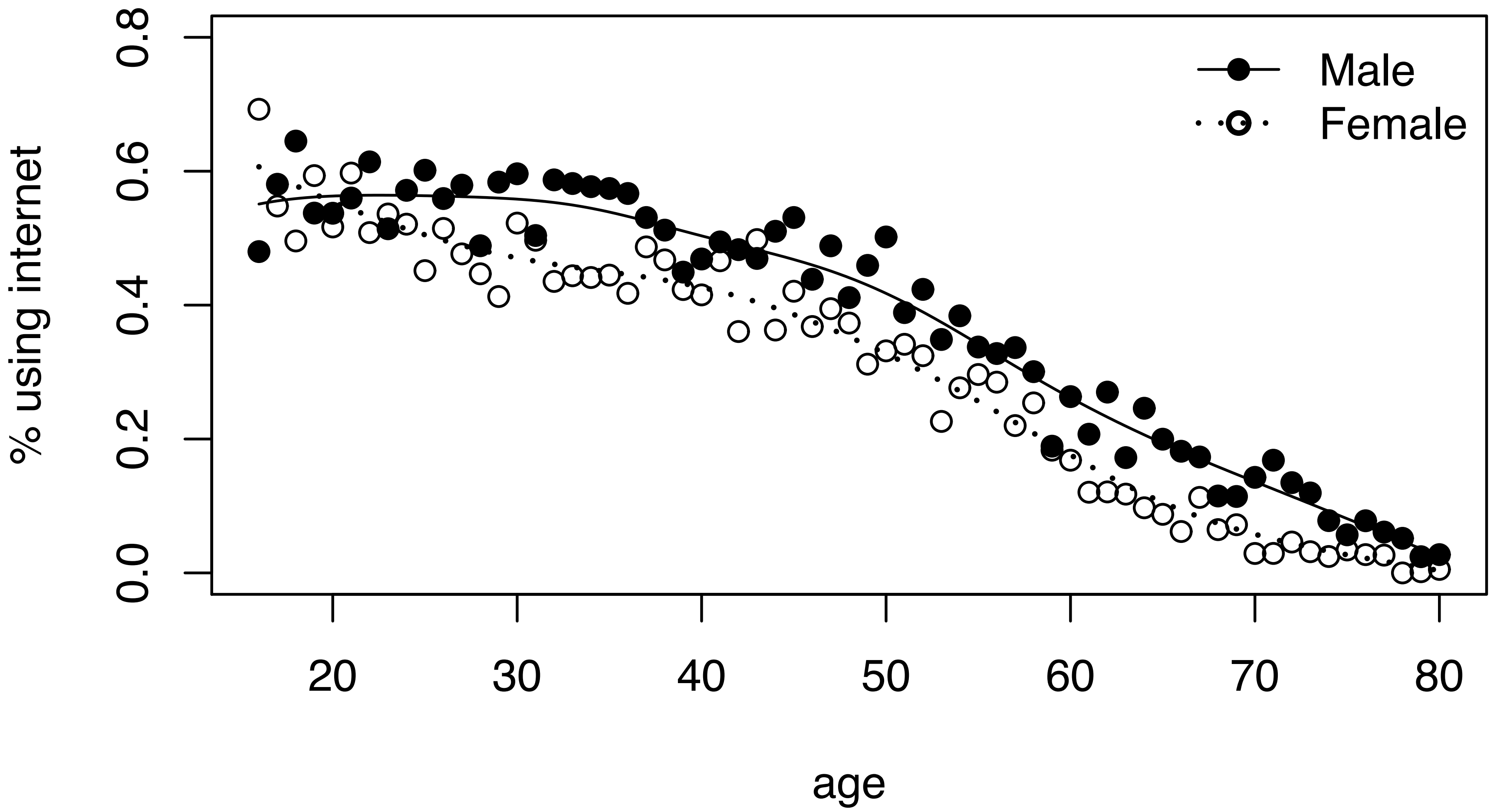
For these models the point estimates are the same for frequency weights, sampling weights, or precision weights, so the point estimation can just reuse the ordinary regression code.

Regression model

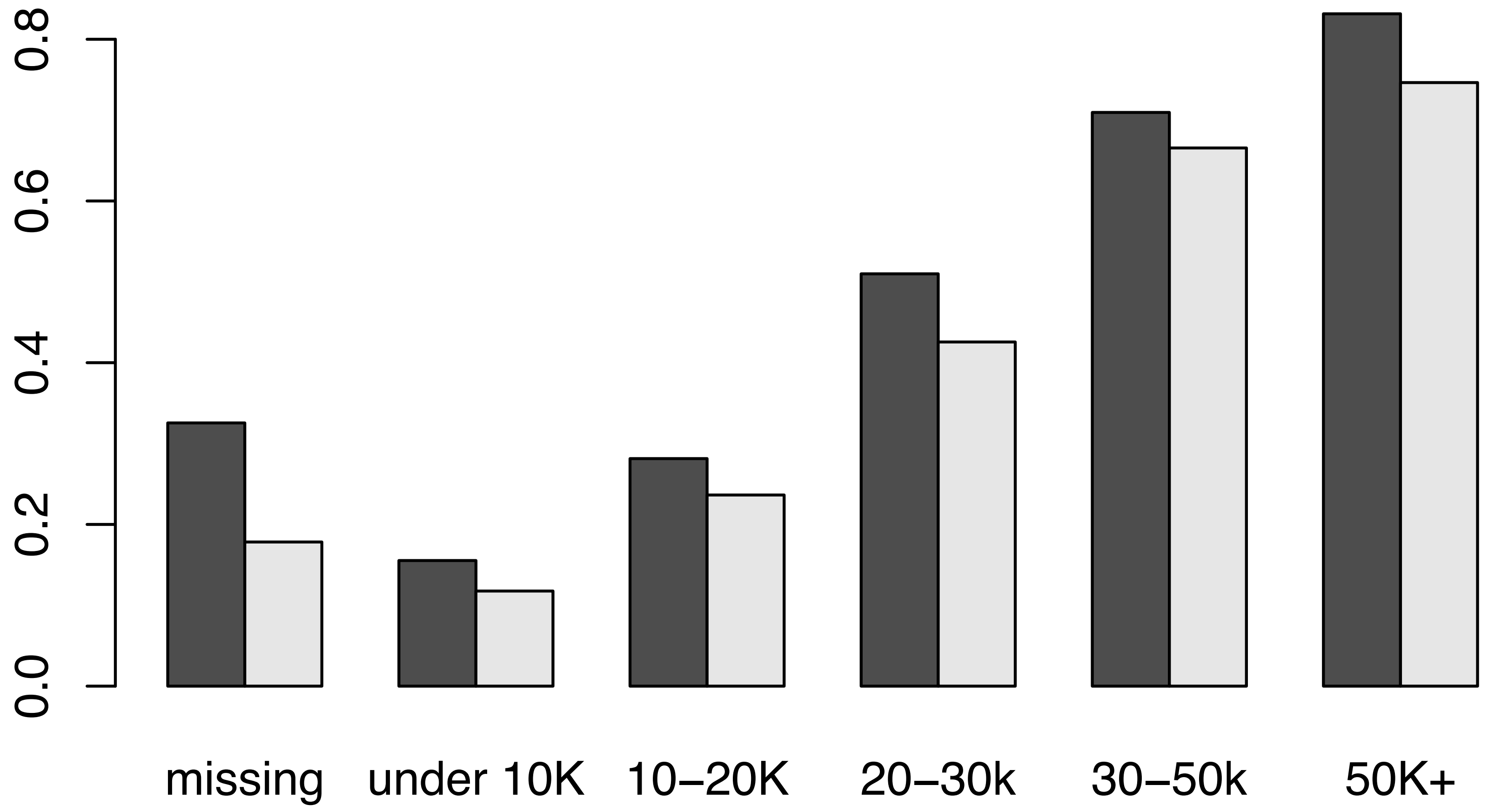
Internet use in Scotland (2001) by age, sex, and income.

```
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
```

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 $\hat{\beta}$. Let r^2 be proportion of variation explained.

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

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

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

compare to HT estimator

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

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

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

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

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

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

Reweighting

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

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

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

For these weights

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

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

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]

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

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

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

R provides `calibrate()` for calibration (and `postStratify()` for post-stratification)

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

Computing

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

The user can specify other calibration loss functions (eg Hellinger distance).

Computing

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.

Computing

```
> 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
```

Computing

```
> 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
```

Computing

```
> (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
```

Computing

```
> 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
```

Computing

```
> 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
```

Computing

```
> (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
```

Computing

```
> (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
```

Types of calibration

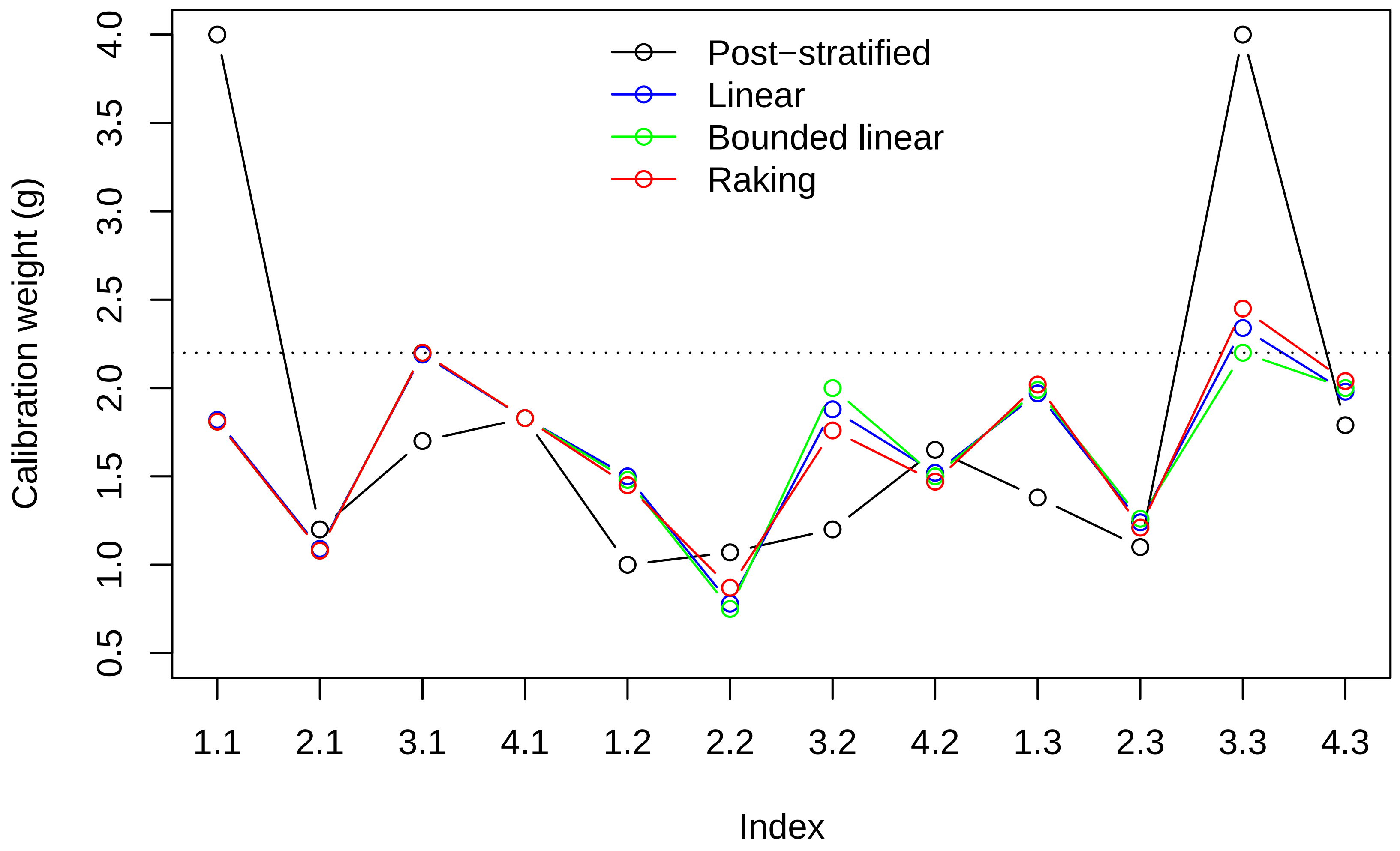
Post-stratification allows much more flexibility in weights, in small samples can result in very influential points, loss of efficiency.

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



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 %
```

That’s all, folks

Any questions?