Complex survey samples

Thomas Lumley

R Core Development Team

useR 2006

Survey package

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

Version 3.5 is current, containing approximately 6000 lines of interpreted R code. (cf 250,000 lines of Fortran for VPLX)

Version 2.3 was published in Journal of Statistical Software. Major changes since then are finite population corrections for multistage sampling, calibration and generalized raking, tests of independence in contingency tables, better tables of results, simple two-phase designs.

Other relevant packages: pps, sampling, sampfling, rake.

Wishlist

Many features of the survey package result from requests from unsatisfied users.

For new methods the most important information is a reference that gives sufficient detail for implementation. A data set is nice but not critical.

For slowness/memory problems a data set is vital so I can actually measure the time and memory use. Optimization without profiling is like estimating population totals from a convenience sample

Overview

- Describing survey designs: `svydesign()`
- Replicate weights: `svrepdesign()`, `as.svrepdesign`
- Summary statistics: mean, total, quantiles, design effect
- Tables of summary statistics, domain estimation.
- Graphics: histograms, hexbin scatterplots.
- Regression modelling: `svyglm()`
- Calibration of weights: `postStratify()`, `calibrate()`

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.

Weights and probabilities

The basic estimation idea is that individuals are sampled with known probabilities π_i, so that the population total for a variable can be estimated by

$$T = \sum_{i=1}^{n} \frac{1}{\pi_i} X_i$$

Other statistics follow from this: if the statistic on the whole population would solve

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

then we solve

$$\sum_{i=1}^{n} \frac{1}{\pi_i} U_i(\theta) = 0$$

Standard errors

Standard errors for totals follow from elementary formulas for the variance of a sum.

Standard errors for more complicated statistics come from the delta method (propagation of errors, first-order Taylor expansion)

Another approach extends the idea of jackknife or bootstrap resampling: evaluate the statistic on a lot of slightly different weights and use the variability between these to estimate the variance. (replicat{e,ion} weights)

Technical details are in the code: `svyrecvar` and `svrVar`, references are in the help pages. Note that simple estimators such as ratios often have a choice of standard error estimators.

Types of designs

The calculations are correct for multistage stratified random sampling with or without replacements.

Taylor expansion is correct for unequal probability sampling with replacement (eg PPS with replacement). I am not confident that the replicate weights are correct in this case, but they are probably not bad.

I am thinking about how to use general Horvitz–Thompson estimators, probably based on sparse matrices. It isn't trivial.

Describing survey designs

Stratified independent sample (without replacement) of schools

```
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.

Describing survey designs

```
> 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 survey designs

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

```
> 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 survey designs

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

- Jackknife, leaving out one PSU (JK1, JKn)
- Bootstrap of PSUs within strata (best with large strata).
- Half-sample for designs with 2 PSUs/stratum (BRR)

Rather than leaving out PSUs we actually set the weight to zero.

Plackett–Burman designs

BRR splits the same in halves so that all PSUs appear in 50% of the half-samples and all pairs of PSUs from different strata appear together in 25% of the half-samples. [full orthogonal balance, Plackett–Burman design, Hadamard matrix]

Under this condition the standard error of the population mean or total is the same as if all 2^{nstrata} half-samples were used.

The number of half-samples has to be a multiple of 4, greater than the number of strata. Constructing the half-samples is easy if the number is a power of 2. The survey package also knows how to generate sets of $2^k(p+1)$ half-samples where p is a prime and $p+1$ is a multiple of 4. This gets close to the minimum possible number of half-samples in most cases.

Prespecified replicate weights

`svrepdesign` creates an object using prespecified replicate weights.

Some survey institutions prefer using replicate weights rather strata/cluster information for confidentiality reasons, or because it is easier to handle calibration/post-stratification.

Unless the replicate weights are a type that R knows about (BRR, JK1, JKn, Fay, bootstrap) it is also necessary to specify how to scale the variance: From `survey:::svrVar`

```
meantheta <- mean(thetas[rscales > 0])
v <- sum((thetas - meantheta)^2 * rscales) * scale
```

Here `scale` is a single number and `rscales` is a vector. For JKn `scale` is the finite population correction and `rscales` is $n/n-1$ for the stratum.

Constructing replicates weights

`as.svrepdesign` converts a `svydesign` to replicate weights. Default is jackknife.

```
rclus1 <- as.svrepdesign(dclus1)
bclus1 <- as.svrepdesign(dclus1, type="bootstrap", replicates=100)
```

Bootstrap and jackknife replicate weights incorporate the finite-sampling correction, BRR doesn't.

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 return 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
>        svyquantile(~api00, dclus1, c(.25,.5,.75))
        0.25 0.5  0.75
api00 551.75 652 717.5
>        svyquantile(~api00, dclus1, c(.25,.5,.75),ci=TRUE)
$quantiles
        0.25 0.5  0.75
api00 551.75 652 717.5

$CIs
, , api00

           0.25      0.5     0.75
(lower 493.2835 564.3250 696.0000
upper) 622.6495 710.8375 761.1355
```

Summary statistics

```
>         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
```

Ratio estimators

Estimating the ratio of population means/totals: `svyratio` takes two formulas specifying numerator and denominator variables.

```
>         svyratio(~api.stu, ~enroll, dclus1)
Ratio estimator: svyratio.survey.design2(~api.stu, ~enroll, dclus1)
Ratios=
           enroll
api.stu 0.8497087
SEs=
             enroll
api.stu 0.008386297
```

Ratio estimators

Ratio estimation of population total uses `predict`

```
> sep<-svyratio(~api.stu,~enroll, dstrat,separate=TRUE)
> com<-svyratio(~api.stu, ~enroll, dstrat)
> stratum.totals<-list(E=1877350, H=1013824, M=920298)
> predict(sep, total=stratum.totals)
$total
         enroll
api.stu 3190022

$se
          enroll
api.stu 29756.44

>  predict(com, total=3811472)
$total
         enroll
api.stu 3190038

$se
          enroll
api.stu 29565.98
```

Some details

- `svyratio` uses the Taylor expansion std error, which has larger unconditional but small conditional bias than the main alternative.
- `svyquantile` by default inverts a confidence interval for the cumulative probability, a slower and more accurate option is to invert a score test.
- Design effects can be calculated compared to with-replacement (DEFT) or without-replacement (DEFF) designs, without-replacement is the default.

Domain estimation

The correct standard error estimate for a subpopulation that isn't 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 the ratio estimator and regression estimator derivations. Some more documentation is in the `domain` vignette.

Note: subsets of design objects are not necessary smaller than the whole objects.

Pretty 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 (mostly requests from ine.pt).

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
M     M       611
>  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)
  stype statistics.api99 statistics.api00 se.api99 se.api00 DEff.api99
E     E         607.7917         648.8681 22.81660 22.36241   5.895734
H     H         595.7143         618.5714 41.76400 38.02025   2.211866
M     M         608.6000         631.4400 32.56064 31.60947   2.226990
  DEff.api00
E   6.583674
H   2.228259
M   2.163900
```

Computing over subgroups

```
>  svyby(~api99+api00, ~stype+sch.wide, dclus1, svymean, keep.var=FALSE)
      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
```

Domain and ratio estimators

Ratio estimators of domain means agree with the result from subsetting the design object:

```
> dstrat<-update(dstrat,imp=as.numeric(comp.imp=="Yes"))
> svyratio(~I(api.stu*imp),~imp,dstrat)
Ratio estimator: svyratio.survey.design2(~I(api.stu * imp), ~imp, dstrat)
Ratios=
                      imp
I(api.stu * imp) 439.9305
SEs=
                      imp
I(api.stu * imp) 19.24367
> svymean(~api.stu, subset(dstrat, comp.imp=="Yes"))
          mean     SE
api.stu 439.93 19.244
```

Formatting

`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
```

Formatting

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
```

Tests for two-way tables

`svychisq` does four variations on the Pearson χ^2 test

- First- and second-order Rao–Scott corrections: first order (`statistic="Chisq"`) corrects the mean of Pearson's X^2, second order (`statistic="F"`) corrects the variance as well.

- Wald-type tests that all the interaction parameters in a saturated log-linear model are zero. Original Koch et al proposal is `statistic="Wald"`, modified version for small numbers of PSUs is `statistic="adjWald"`.

Also works with replicate weights, where $n_{PSU} - n_{strata}$ is replaced by the rank of the matrix of weights minus 1.

Tests for two-way tables

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

        Pearson's X^2: Rao & Scott adjustment

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

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

        Design-based Wald test of association

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

Graphics

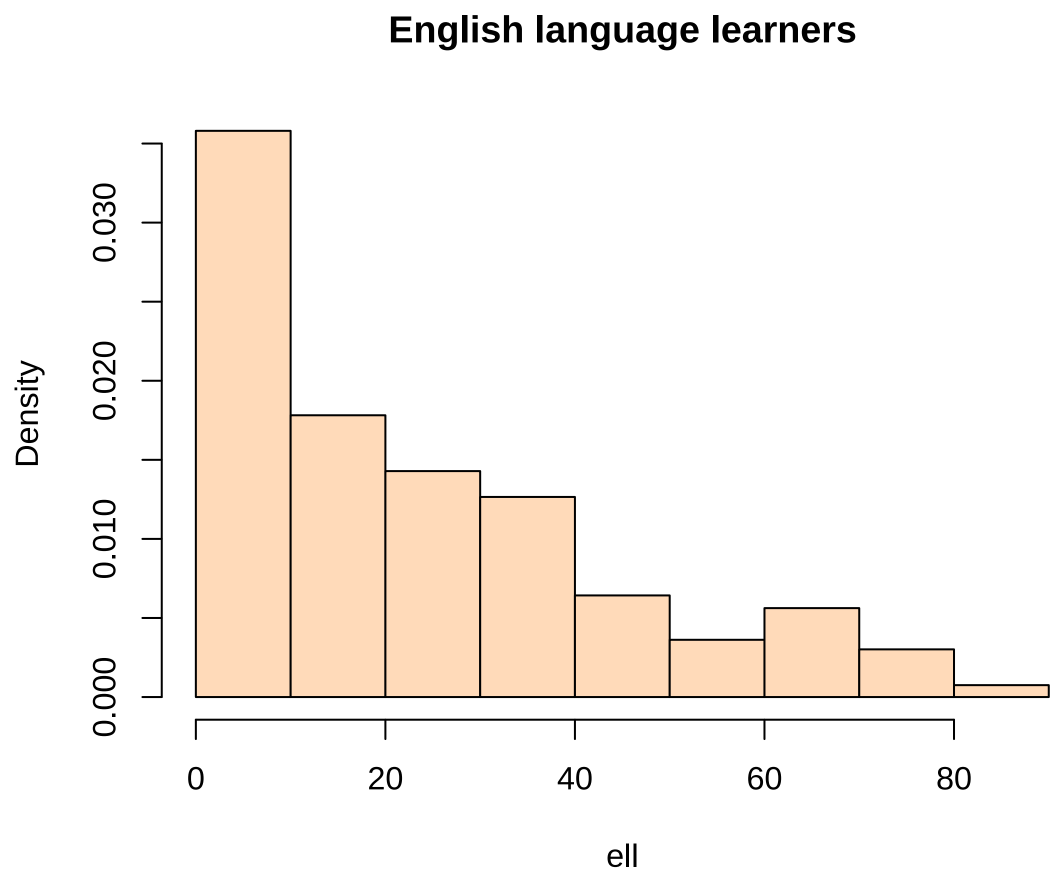
Two difficulties in graphics: large sample size, sampling weights.

- 'Bubble' plots with circle area proportional to weight
- histograms, boxplots using sample weights (could add smoothers here)
- Hexagonal binning plots, estimating population number over areas of plot
- Subsampling: generate a simple random sample with replacement from an estimated population and plot it.

`svyplot` does bubble plots, hexbin, subsampling. `svyhist` and `svyboxplot` do histograms and boxplots.

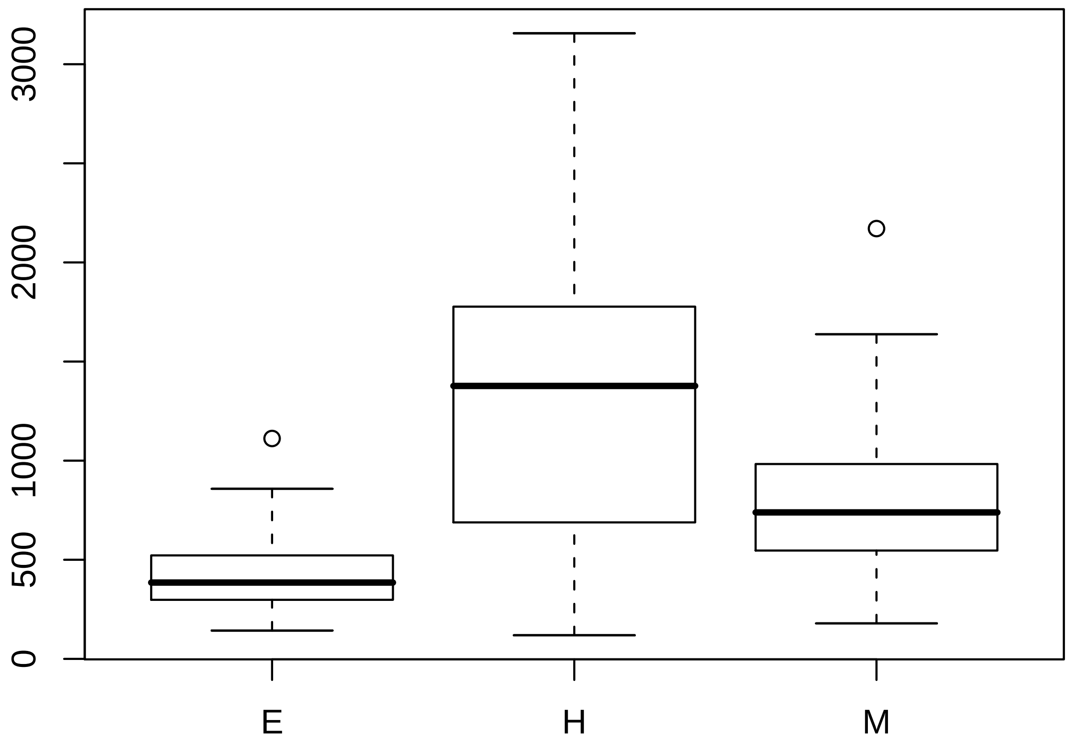
Graphics

```
svyhist(~ell, dstrat, main = "English language learners",
        col = "peachpuff")
```



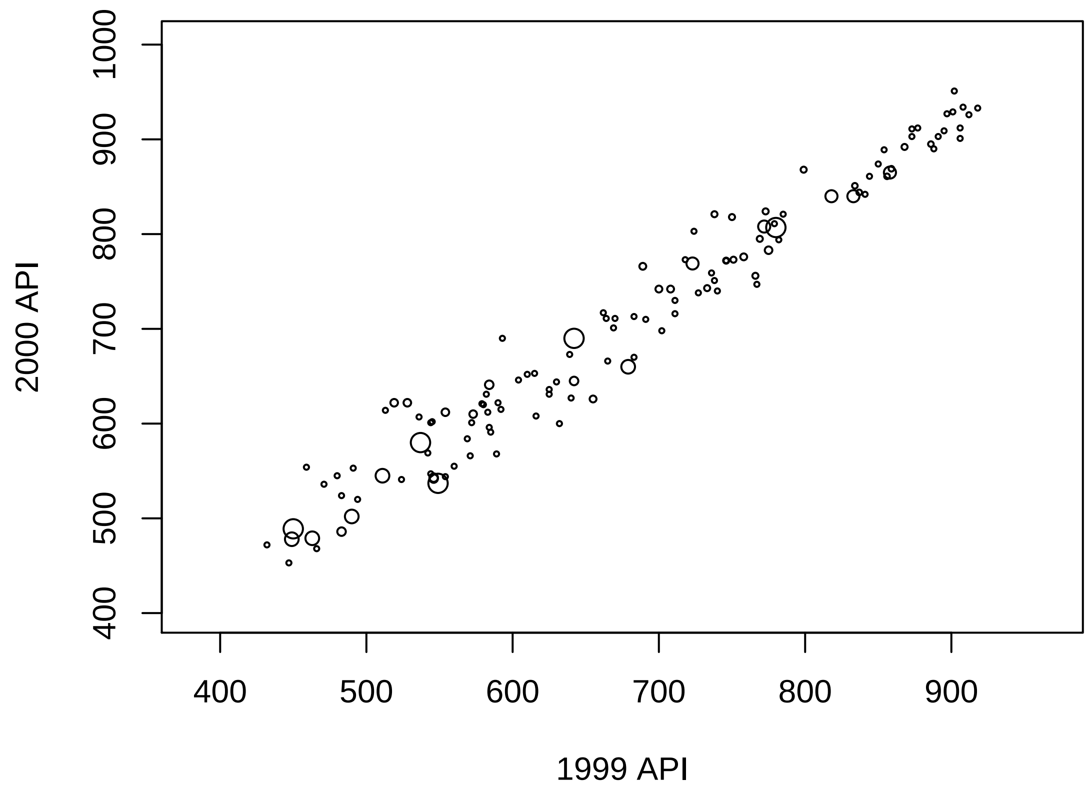
Graphics

```
svyboxplot(enroll ~ stype, dstrat)
```



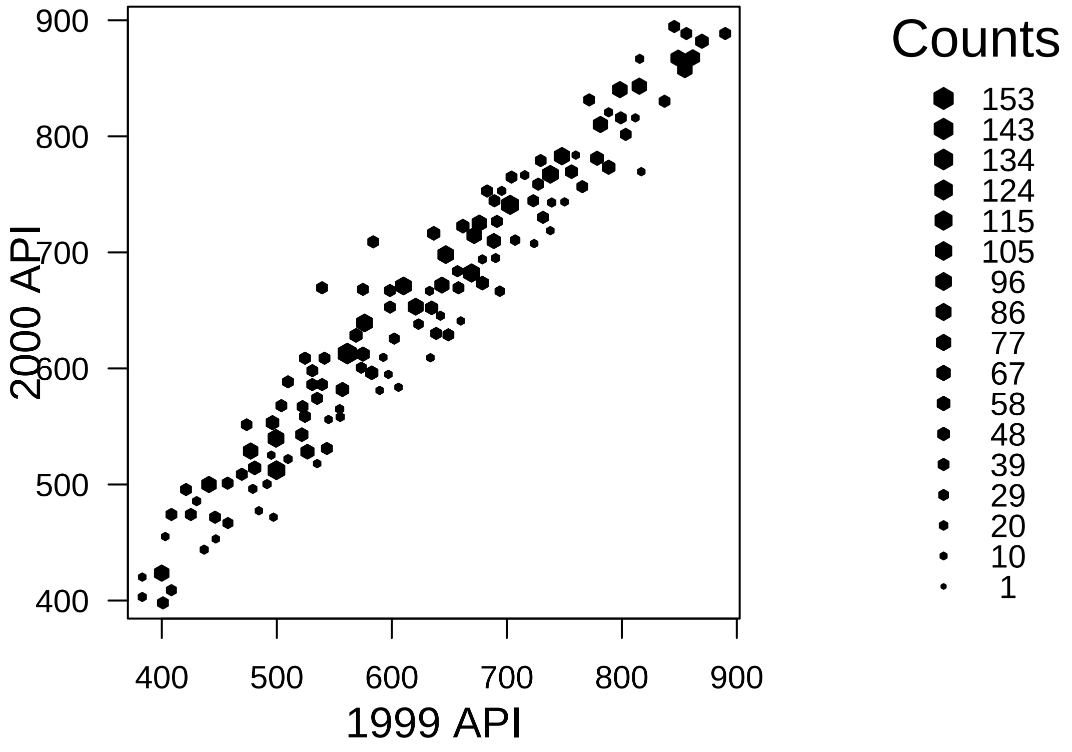
Graphics

```
svyplot(api00~api99, design=dclus2, style="bubble",
         xlab="1999 API",ylab="2000 API")
```

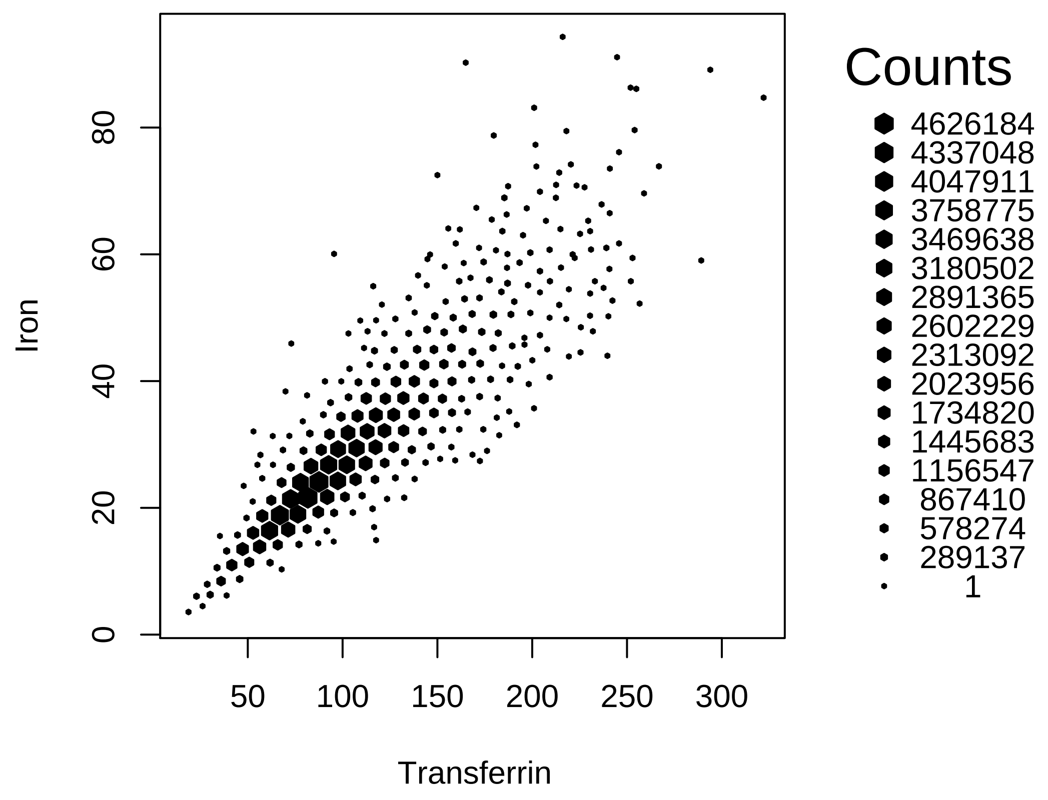


Graphics

```
svyplot(api00~api99, design=dstrat, style="hex",
   xlab="1999 API",ylab="2000 API")
```



Graphics


Counts
4626184
4337048
4047911
3758775
3469638
3180502
2891365
2602229
2313092
2023956
1734820
1445683
1156547
867410
578274
289137
1
Iron
0
20
40
60
80
50
100
150
200
250
300
Transferrin

Regression models

- `svyglm` for linear and generalized linear models

- `svycoxph` for Cox model (no std errors on survival curves yet)

Some other models, eg censored parametric regression models, could be fitted with `svymle` (`svydesign` objects only) or `withReplicates` (replicate weight designs only).

Regression and domain estimators

Academic Performance Index in schools with more or less than 20% "English language learners"

```
> svyby(~api00,~I(ell>20), dclus1,svymean)
      I(ell > 20) statistics.api00 se.api00
FALSE       FALSE         717.9661 15.53905
TRUE         TRUE         609.0565 25.55300

> summary(svyglm(api00~I(ell>20), dclus1))
Call:
svyglm(api00 ~ I(ell > 20), dclus1)

Survey design:
svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       717.97      15.54   46.20 8.33e-16 ***
I(ell > 20)TRUE  -108.91      19.48   -5.59 8.78e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 8577.366)
```

Regression and domain estimators

```
> summary(svyglm(api00~I(ell>20)+0, dclus1))

Call:
svyglm(api00 ~ I(ell > 20) + 0, dclus1)

Survey design:
svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
I(ell > 20)FALSE   717.97      15.54   46.20 8.33e-16 ***
I(ell > 20)TRUE    609.06      25.55   23.84 4.11e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 8577.366)

Number of Fisher Scoring iterations: 2
```

Logistic regression

Do school type and socioeconomic variables predict attaining school-wide performance target?

```
> summary(svyglm(sch.wide~stype+ell+mobility,dclus1,
      family=quasibinomial))

Call:
svyglm(sch.wide ~ stype + ell + mobility, dclus1,
      family = quasibinomial)

Survey design:
svydesign(id = ~dnum, weights = ~pw, data = apiclus1, fpc = ~fpc)

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.057e+00  4.064e-01   5.062 0.000491 ***
stypeH      -9.291e-01  6.886e-01  -1.349 0.207026
stypeM      -1.571e+00  6.167e-01  -2.547 0.029009 *
ell          1.209e-02  1.111e-02   1.089 0.301698
mobility    -8.177e-05  1.673e-02  -0.005 0.996195
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Tests

Likelihood ratio tests are not available, so multi-coefficient tests have to use the Wald method. `regTermTest()` packages this.

```
> model <- svyglm(sch.wide~stype+ell+mobility,dclus1, family=quasibinomial)

> regTermTest(model, ~stype)
Wald test for stype
 in svyglm(sch.wide ~ stype + ell + mobility, dclus1, family = quasibinomial)
Chisq =  6.638399  on  2  df: p= 0.036182

> regTermTest(model, ~ell+mobility)
Wald test for ell mobility
 in svyglm(sch.wide ~ stype + ell + mobility, dclus1, family = quasibinomial)
Chisq =  1.329623  on  2  df: p= 0.51437
```

Cox model

The Cox proportional hazards model is the most popular model for time-to-event in biostatistics. Here we use a two-phase design in sampling from a cohort that is in turn a simple random sample from a large population.

Event is relapse in Wilm's Tumour: `edrel` is observation time, `rel` is relapse indicator

`age` is known for everyone, but `histol` is determined from stored samples for everyone who relapses and a stratified random sample of others. [Case–cohort design]

We use `twophase()` to specify the two-phase study design: like `svydesign()` but two of everything.

Cox model

```
> library("survival")
Loading required package: splines
> data(nwtco)
> dcchs<-twophase(id=list(~seqno,~seqno), strata=list(NULL,~rel),
+               subset=~I(in.subcohort | rel), data=nwtco)
> svycoxph(Surv(edrel,rel)~factor(stage)+factor(histol)+I(age/12), design=dcchs)
Call:
svycoxph.survey.design(formula = Surv(edrel, rel) ~ factor(stage) +
    factor(histol) + I(age/12), design = dcchs)

                  coef exp(coef) se(coef)     z       p
factor(stage)2  0.6927      2.00    0.163  4.25 2.1e-05
factor(stage)3  0.6269      1.87    0.168  3.73 1.9e-04
factor(stage)4  1.2995      3.67    0.189  6.88 6.1e-12
factor(histol)2 1.4583      4.30    0.145 10.02 0.0e+00
I(age/12)       0.0461      1.05    0.023  2.00 4.5e-02
```

Calibration

Calibration adjusts the survey weights so that the estimated population total for a variable exactly matches the known true value.

Simplest version is post-stratification: adjust the weights so that a categorical variable matches the population counts. Recovers most of the information lost by not stratifying the sampling.

`postStratify` takes a design, a formula, and a data frame or table giving population totals.

Post-stratification

```
> svymean(~api00, dclus1)
        mean     SE
api00 644.17 26.329
> svytotal(~enroll, dclus1)
          mean     SE
enroll 3404940 932235
>
>      pop.types <- data.frame(stype=c("E","H","M"), Freq=c(4421,755,1018))
>    rclus1p<-postStratify(rclus1, ~stype, pop.types)
>      summary(rclus1p)
Call: postStratify(rclus1, ~stype, pop.types)
Unstratified cluster jacknife (JK1) with 15 replicates.
Variables:
 [1] "cds"        "stype"      "name"       "sname"      "snum"       "dname"
 [7] "dnum"       "cname"      "cnum"       "flag"       "pcttest"    "api00"
...
```

Post-stratification

```
>      svymean(~api00, rclus1p)
        mean    SE
api00 642.31 26.45
>      svytotal(~enroll, rclus1p)
          mean     SE
enroll 3680893 346014
> svytotal(~stype,rclus1p)
       mean         SE
stypeE 4421 4.685e-12
stypeH  755 1.719e-13
stypeM 1018 2.877e-13
```

Raking

With population totals for two categorical variables but not the joint distribution, alternately post-stratify on each one until convergence — raking.

rake takes a design, a list of formulas, and a list of data frames or tables giving population totals.

```
> 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))
> dclus1r<-rake(dclus1, list(~stype,~sch.wide), list(pop.types, pop.schwide))
>
> svymean(~api00, dclus1r)
        mean     SE
api00 641.23 23.739
> svytotal(~enroll, dclus1r)
         total     SE
enroll 3647300 399094
```

Calibration

Regression calibration adjusts weights to match the totals of multiple variables by least squares

Generalized raking extends to iterative least squares adjustment, including raking as a special case.

All done by `calibrate()`

`calibrate()`

- `design` to be calibrated
- `formula` specifying calibration variables
- `population` vector specifying totals (as column totals of design matrix generated by `formula`)
- `aggregate.stage` optional level of sampling where weights must be constant within sampling units, or `aggregate.index` for replicate weight designs
- `calfun` is `"linear"`, `"logit"`, `"raking"`
- `bounds` are bounds for weights: optional except in `logit`

calibrate()

```
> pop.totals<-c(`(Intercept)`=6194, stypeH=755, stypeM=1018)
>
> help(calibrate)
>  (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 3.373e-14
stypeH   755 1.368e-14
stypeM  1018 2.609e-14
```

Calibration

Calibration the second phase of a two-phase design can give useful gains in information. Also useful as a way of handling missing data

In Wilm's Tumour example, calibrate on disease stage and local hospital histology classification

Calibration

```
> gcchs<-calibrate(dcchs, ~interaction(rel, instit, stage), phase=2)
> svycoxph(Surv(edrel,rel)~factor(stage)+factor(histol)+I(age/12),
     design=gcchs)
Call:
svycoxph.survey.design(formula = Surv(edrel, rel) ~ factor(stage) +
    factor(histol) + I(age/12), design = gcchs)


                  coef exp(coef) se(coef)     z       p
factor(stage)2  0.658      1.93   0.1352  4.86 1.1e-06
factor(stage)3  0.800      2.23   0.1356  5.90 3.6e-09
factor(stage)4  1.297      3.66   0.1522  8.52 0.0e+00
factor(histol)2 1.511      4.53   0.1287 11.74 0.0e+00
I(age/12)       0.037      1.04   0.0235  1.58 1.2e-01

Likelihood ratio test=NA  on 5 df, p=NA  n= 1154
```

Missing data

`estWeights` constructs a calibrated two-phase design from a data frame with missing data on some variables

```
>     data(airquality)
>
>      ## ignoring missingness, using model-based standard error
>      summary(lm(log(Ozone)~Temp+Wind, data=airquality))

Call:
lm(formula = log(Ozone) ~ Temp + Wind, data = airquality)

Residuals:
     Min       1Q   Median       3Q      Max
-2.34415 -0.25774  0.03003  0.35048  1.18640

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.531932   0.608901  -0.874  0.38419
Temp         0.057384   0.006455   8.889 1.13e-14 ***
Wind        -0.052534   0.017128  -3.067  0.00271 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Missing data

```
> ## Without covariates to predict missingness we get
> ## same point estimates, but different (sandwich) standard errors
> daq<-estWeights(airquality, formula=~1,subset=~I(!is.na(Ozone)))
> summary(svyglm(log(Ozone)~Temp+Wind,design=daq))

Call:
svyglm(log(Ozone) ~ Temp + Wind, design = daq)

Survey design:
estWeights(airquality, formula = ~1, subset = ~I(!is.na(Ozone)))

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.531932   0.833602  -0.638   0.5247
Temp         0.057384   0.008453   6.789 5.51e-10 ***
Wind        -0.052534   0.020330  -2.584   0.0110 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 0.3130098)

Number of Fisher Scoring iterations: 2
```

Missing data

```
>  ## Reweighting based on weather, month
>  d2aq<-estWeights(airquality, formula=~Temp+Wind+Month,
+                        subset=~I(!is.na(Ozone)))
>  summary(svyglm(log(Ozone)~Temp+Wind,design=d2aq))

Call:
svyglm(log(Ozone) ~ Temp + Wind, design = d2aq)

Survey design:
estWeights(airquality, formula = ~Temp + Wind + Month,
                     subset = ~I(!is.na(Ozone)))

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.577759   0.812118  -0.711   0.4783
Temp         0.057689   0.008213   7.024 1.72e-10 ***
Wind        -0.048750   0.019729  -2.471   0.0150 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 0.3232150)

Number of Fisher Scoring iterations: 2
```