Advanced Data Analysis in R

Survey Analysis in R

Michael DeWitt

2018-02-09 (Updated 2019-03-12)

Survey Analysis in R

What makes survey analysis different?

Survey analysis is *design based*

Often we talk about *probability or random samples*

These concepts make inferences really nice

Properties of Design Based Surveys

A quick refresher[1]

1. Every individual in the population must have a non-zero probability of ending up in the sample (π_i)

[1] Lumley (2010)

Properties of Design Based Surveys

A quick refresher[1]

1. Every individual in the population must have a non-zero probability of ending up in the sample (π_i)
2. The probability of π_i must be known for every individual in who does end up in the sample

[1] Lumley (2010)

Properties of Design Based Surveys

A quick refresher[1]

1. Every individual in the population must have a non-zero probability of ending up in the sample (π_i)
2. The probability of π_i must be known for every individual in who does end up in the sample
3. Every pair of individuals in the sample must have a non-zero probability of both ending up in the sample ($\pi_{i,j}$ for the pair of individuals (i, j))

[1] Lumley (2010)

Properties of Design Based Surveys

A quick refresher[1]

1. Every individual in the population must have a non-zero probability of ending up in the sample (π_i)
2. The probability of π_i must be known for every individual in who does end up in the sample
3. Every pair of individuals in the sample must have a non-zero probability of both ending up in the sample ($\pi_{i,j}$ for the pair of individuals (i, j))
4. The probability $\pi_{i,j}$ must be known for every pair that does end up in the sample

[1] Lumley (2010)

Introducing the `survey` package

A little about `survey`

Thomas Lumley developed the `survey` package

Initially a port of STATA's `svy` functions following a similar syntax

Can perform typical types of design based analysis

- Simple Random
- Stratified
- Clusters
- Multi-stage
- Repeated Measures

A little about `survey`

Perform post-survey corrections
- post-stratification
- raking (iterative proportional fitting)
- calibration

And more. . . !

Diving into the software. . .

Describing your model

The primary argument in `survey` is the `svydesign` function

```
library(survey)

svydesign(ids = to specify clusters (~1 otherwise),
          probs = Sampling Probabilities if available,
          strata = Strata membership if available,
          fpc = Finite Population Values,
          data = Your Data Frame,
          nest = T/F if there is nesting within your strata,
          weights = Sampling Weights if available,
          pps = Probability Proportional to Size)
```

Quick Note On `survey`

Many of the functions in `survey` utilise R "formula notation"

Indicates the tilde "$\sim$" must be used (e.g. `~cluster`)

But Let's Try An Example

Let's try an example with the `api` data set that is part of the `survey` package

This data set represents California Academic Performance Index

```
library(survey)
library(dplyr)
data(api)
```

Let's Inspect the Data

```
head(apisrs)
##                  cds stype            name
## 1039 15739081534155     H  McFarland High
## 1124 19642126066716     E Stowers (Cecil
## 2868 30664493030640     H Brea-Olinda Hig
## 1273 19644516012744     E Alameda Element
## 4926 40688096043293     E Sunnyside Eleme
## 2463 19734456014278     E Los Molinos Ele
##                               sname snum
## 1039                 McFarland High 1039
## 1124 Stowers (Cecil B.) Elementary 1124
## 2868               Brea-Olinda High 2868
## 1273             Alameda Elementary 1273
## 4926          Sunnyside Elementary 4926
## 2463        Los Molinos Elementary 2463
##                         dname dnum           cname
## 1039        McFarland Unified  432            Kern
```

Specifying the Survey Object (SRS)

This is a simple random sample with finite population correct (since we know the population)

```
(svy_api_srs <- svydesign(ids = ~1,
                          fpc = ~fpc,
                          data = apisrs))
## Independent Sampling design
## svydesign(ids = ~1, fpc = ~fpc, data = apisrs)
```

Trying With A Different Survey Design (Stratified)

In this case we have a stratified random sample (different school types)

```
(svy_api_strat <- svydesign(ids= ~1,
                                strata = ~stype,
                                fpc = ~fpc,
                                data = apistrat))
## Stratified Independent Sampling design
## svydesign(ids = ~1, strata = ~stype, fpc = ~fpc, data = apist
```

Trying With A Different Survey Design (Cluster)

Two stage cluster sampling 40 school districts then five schools within each district

- Stage 1 district cluster with population `fpc1`
- Stage 2 district cluster with population `fpc2`

```
(svy_api_cluster <- svydesign(ids= ~dnum+snum,
                              fpc = ~fpc1+fpc2,
                              data = apiclus2))
## 2 - level Cluster Sampling design
## With (40, 126) clusters.
## svydesign(ids = ~dnum + snum, fpc = ~fpc1 + fpc2, data = apic
```

Analysis with svy objects

Correct Estimates

`survey` applies correct calculations given the survey design

```
svymean(~api00, svy_api_cluster)
##          mean     SE
## api00 670.81 30.099
```

vs

```
cbind(mean(apiclus2[["api00"]]),
      sd(apiclus2[["api00"]]))
##          [,1]     [,2]
## [1,] 703.8095 134.1507
```

Survey Functions

Functions in the `survey` package begin with the `svy` prefix

Utilise the formula notation

```
svyquantile(x = ~api99+api00,
            svy_api_srs,
            quantile= c(0.25,.75))
##       0.25 0.75
## api99  513  738
## api00  544  752
```

Calculating Contrasts

You can add contrasts with svycontrast

Say I wanted to look at the ratio of my high school score to my elementary school score

```
# Mean
mean_score <- svyby( ~api99, ~stype, svymean,
                        design = svy_api_cluster)
# Contrast ratio use `quote`
svycontrast(mean_score, quote(H/E))
##             nlcon     SE
## contrast 0.90614 0.0507
```

Adding Contrasts to the data

Use the `update` function to add new calculated fields to your survey design object

```
(svy_api_cluster <- update(svy_api_cluster,
                           score_imp = api00/api99))
## 2 - level Cluster Sampling design
## With (40, 126) clusters.
## update(svy_api_cluster, score_imp = api00/api99)
```

Adding Contrasts to the data

Now we can easily perform our analysis

```
svyby(~score_imp, ~stype, svymean,
      design = svy_api_cluster)
##   stype score_imp          se
## E     E  1.057525 0.006591223
## H     H  1.005193 0.003781072
## M     M  1.017354 0.012393586
```

Performing Regressions

```
svyglm(score_imp~ meals + avg.ed, svy_api_cluster)
## 2 - level Cluster Sampling design
## With (40, 126) clusters.
## update(svy_api_cluster, score_imp = api00/api99)
##
## Call:  svyglm(formula = score_imp ~ meals + avg.ed, design =
##
## Coefficients:
## (Intercept)        meals       avg.ed
##   0.9742040    0.0007394    0.0103667
##
## Degrees of Freedom: 125 Total (i.e. Null);  37 Residual
## Null Deviance:       0.2624
## Residual Deviance: 0.2118    AIC: -391
```

Post-survey corrections

Motivating Example

All about survey error![2]

Non-response can bias our answers

Convenience samples suffer from response bias

[2](See Groves and Lyberg (2010))

Let's Make Some Fake Data

Initially use data from the `MASS` package (Venables and Ripley 2002)

```
df <- (MASS::survey) %>%
  na.omit()
```

Survey responses of 237 Statistics I students at the University of Adelaide

Let’s Examine Some Statistics

Let’s say we want to make inferences about a population using this survey.

But before we do that we want to check the population margins

```
prop.table(table(df$Sex))
##
## Female   Male
##    0.5    0.5
```

Creating Our Survey Design

First we create our `svydesign` object

```
survey_design_unweighted <- svydesign(ids = ~1, data =df)
```

Create Population Data

Then we create data sets to represent the population distribution

```
(gender_dist <- data.frame(
  Sex = c("Female", "Male"),
  Freq = round(nrow(df) * c(.55, .45),0)))
##      Sex Freq
## 1 Female   92
## 2   Male   76
```

Apply Post-stratification

We can then use the `postStratify` function and supply

- `svydesign` object
- The variable we want to post-stratify
- The population margins

```
(survey_design_weighted <- postStratify(
  survey_design_unweighted,
  ~Sex,
  gender_dist))
## Independent Sampling design (with replacement)
## postStratify(survey_design_unweighted, ~Sex, gender_dist)
```

Different Population Inferences

```
svymean(~Height, survey_design_unweighted)
##          mean     SE
## Height 172.48 0.7684
```

```
svymean(~Height, survey_design_weighted)
##          mean     SE
## Height 171.82 0.5384
```

More than one variable?

The actual proportion of left-handed peoples is 10%

```
prop.table(table(df$W.Hnd))
##
##       Left      Right
## 0.07142857 0.92857143
```

Set Up Additional Population Margins

Our 10% lefties...

```
(handed <- data.frame(
  W.Hnd = c("Left", "Right"),
  nrow(df) * c(.1, .9)))
##   W.Hnd nrow.df....c.0.1..0.9.
## 1  Left                   16.8
## 2 Right                  151.2
```

Enter Raking

Raking or iterative proportional fitting post-stratifies iteratively on the specified population margins until the new weights stabilise.

Useful when the joint distributions are not known

User must specify the threshold for weight stabilisation

Now Rake

We can implement raking with the `rake` function by supplying:

- Sample margins (variables to rake)
- Population margins

```
survey_design_rake <- rake(
  survey_design_unweighted,
  sample.margins = list(~Sex, ~W.Hnd),
  population.margins = list(gender_dist,handed))
```

Checking your weights

It is important to check your weights

Low representation in surveys leads to highly variable estimates

See this Tesler (2018)[3]

```
summary(weights(survey_design_rake))
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8706  0.8706  1.0654  1.0000  1.0654  1.5671
```

[3] (link here)[https://www.washingtonpost.com/news/monkey-cage/wp/2018/08/17/no-one-third-of-african-americans-dont-support-trump-not-even-close/?utm_term=.a45e7da91344)]

Trim The Weights

There are many methods of trimming weights

```
median_wt <- median(weights(survey_design_rake))
IQR_wt <- IQR(weights(survey_design_rake))

trimmed <-trimWeights(survey_design_rake,
            upper = median_wt + IQR_wt,
            lower = median_wt - IQR_wt)
```

Add the Weights to a data set

One trick is to add the survey weights to your data

```
df_with_wts <- df %>%
  add_column(wts = weights(trimmed))
```

But I have a zero. . .

Post-survey treatment with continuous indicators

References

Groves, R. M., and L. Lyberg. 2010. "Total Survey Error: Past, Present, and Future." *Public Opinion Quarterly* 74 (5): 849–79. https://doi.org/10.1093/poq/nfq065.

Lumley, Thomas. 2010. *Complex Surveys: A Guide to Analysis Using R*. John Wiley & Sons.

Tesler, Michael. 2018. "Analysis No, One-Third of African Americans Don't Support Trump. Not Even Close." *Washington Post*. https://www.washingtonpost.com/news/monkey-cage/wp/2018/08/17/no-one-third-of-african-americans-dont-support-trump-not-even-close/.

Venables, W. N., and B. D. Ripley. 2002. *Modern Applied Statistics with S*. Fourth. New York: Springer. http://www.stats.ox.ac.uk/pub/MASS4.