8

Nonresponse

We have already seen most of the features needed to select survey samples and to compute estimates of means, totals, and quantiles using the sampling weights and survey design.

Nonresponse affects almost all surveys, however. Even samples of inanimate objects such as audit records may have missing accounts or missing items for accounts. This chapter looks at how R deals with missing data and points you to resources for calibrating survey data and performing imputations in R. The code is in file `ch08.R` on the book website.

8.1 How R Functions Treat Missing Data

We saw in Section 1.7 that R uses the value NA to denote missing data. If a data set uses a different code for missing values, such as -99, these must be converted to NA before analyzing the data so that R recognizes the values as missing. Otherwise, functions will treat these observations as though $y = -99$ and include them in calculations, sometimes with embarrassing results as when the estimated mean age is a negative number.

Data set `impute.csv` contains the data from Table 8.4 of SDA; it codes the missing values by -99. In the R data set *impute* in the `SDAResources` package, these missing values have been converted to NA. Variables *education*, *crime*, and *violcrime* all have missing values.

Most functions in R have a default method for how missing values are treated. The *mean* function, for example, returns NA if you attempt to find the mean of a vector that has missing values; if you want it to calculate the mean of the non-missing values, include argument `na.rm=TRUE`. The following shows how some of the base R functions and the *svymean* function treat the missing data in variables *crime* and *violcrime*, which take on values 1, 0, and NA.

```
data(impute)
impute$crime
##  [1]  0  1  0  1  1  0  1  0  0 NA  0  0  1  1  0  0  0  0 NA  0
is.na(impute$crime) # vector with TRUE for missing values
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
# identify the rows with no missing values in columns 5-6
impute$cc<-complete.cases(impute[,5:6])
impute$cc
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE
## [13] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
mean(impute$crime)  # returns NA
## [1] NA
mean(impute$crime,na.rm=TRUE) # calculates mean of non-missing values
## [1] 0.3333333
```

DOI: 10.1201/9781003228196-8

```
table(impute$crime,impute$violcrime) # excludes values missing in either variable
##
##      0  1
##   0 11  0
##   1  1  3
table(impute$crime,impute$violcrime,useNA="ifany") # counts NAs as category in table
##
##         0  1 <NA>
##   0    11  0    1
##   1     1  3    2
##   <NA>  1  0    1
# input design information, use relative weights of 1 for comparison with above
dimpute <- svydesign(id = ~1, weights = rep(1,20), data = impute)
dimpute
## Independent Sampling design (with replacement)
## svydesign(id = ~1, weights = rep(1, 20), data = impute)
# calculate survey mean and se
svymean(~crime,dimpute) # returns NA
##       mean SE
## crime   NA NA
svymean(~crime, dimpute, na.rm=TRUE)
##          mean    SE
## crime 0.33333 0.114
svytable(~violcrime+crime,dimpute)
##          crime
## violcrime  0  1
##         0 11  1
##         1  0  3
```

When `na.rm=TRUE` is included, the *mean* and *svymean* functions estimate the mean using the observations having non-missing values.

In a complex survey, however, omitting the missing values can affect the standard error calculations. A better option, for complex surveys where the missing values are not imputed, is to use the *subset* function to define the subset of observations with non-missing values on which the analysis is to be performed—treating the non-missing values as a domain, as seen in Section 7.3. Remember, though, that an analysis of complete cases alone can be subject to nonresponse bias. You may want to explore the sensitivity of your estimates to the assumptions about the missing values.

8.2 Poststratification and Raking

Unit nonresponse occurs when a unit selected for the sample provides no data. In some data sets, the nonrespondent unit may be represented in the data set, but with missing values for all survey responses. In others, nonrespondent units are missing entirely from the data set.

The most common method for trying to compensate for potential effects of nonresponse is to weight the data. The sampling frame may contain information that can be used in weighting class adjustments, or information known about the population from an external source may be used to poststratify or rake the data as described in Section 8.6 of SDA.

We have already seen the *postStratify* function from the `survey` package (Lumley, 2020) in Chapter 4. The package also has a function to rake the weights to marginal counts in a table. It takes the form

```
rake(design.object, sample.margins, population.margins)
```

where *sample.margins* is a list of formulas describing the sample margins and *population.margins* is a list giving the population counts for each raking variable.

Here is a simple example, using the data in Section 8.6.2 of SDA. The entries in the data frame *rakewtsum* are the sums of weights for persons in the sample falling into each cross-classification of gender by race. For this example, we assume that the data are from a sample of size 500 where each person has weight 6 and construct data frame *rakedf* that has 500 records listing the race and gender of each person (in most applications of raking, you will already have the data frame and will not need this step).

Function *rake* is then used to do raking adjustment on design object *drake* according to the population marginal totals of gender (*pop.gender*) and race (*pop.race*).

```
rakewtsum <- data.frame(gender=rep(c("F","M"),each=5),
                        race=rep(c("Black","White","Asian","NatAm","Other"),times=2),
                        wtsum=c(300,1200,60,30,30,150,1080,90,30,30))
rakewtsum # check data entry
##    gender  race wtsum
## 1       F Black   300
## 2       F White  1200
## 3       F Asian    60
## 4       F NatAm    30
## 5       F Other    30
## 6       M Black   150
## 7       M White  1080
## 8       M Asian    90
## 9       M NatAm    30
## 10      M Other    30
# Need data frame with individual records to use rake function
rakedf <- rakewtsum[rep(row.names(rakewtsum), rakewtsum[,3]/6), 1:2]
dim(rakedf)
## [1] 500   2
rakedf$wt <- rep(6,nrow(rakedf))
# Create the survey design object
drake <- svydesign(id=~1, weights=~wt, data=rakedf)
# Create data frames containing the marginal counts
pop.gender <- data.frame(gender=c("F","M"), Freq=c(1510,1490))
pop.race <- data.frame(race=c("Black","White","Asian","NatAm","Other"),
                       Freq=c(600,2120,150,100,30))
# Now create survey design object with raked weights
drake2 <- rake(drake, list(~gender,~race), list(pop.gender, pop.race))
drake2 # describes SRS with replacement
## Independent Sampling design (with replacement)
## rake(drake, list(~gender, ~race), list(pop.gender, pop.race))
# Look at first 10 entries in vector of raked weights
weights(drake2)[1:10]
##        1      1.1      1.2      1.3      1.4      1.5      1.6      1.7
## 7.511886 7.511886 7.511886 7.511886 7.511886 7.511886 7.511886 7.511886
##      1.8      1.9
## 7.511886 7.511886
```

```
# Look at sum of raked weights for raking cells
svytable(~gender+race, drake2)
##       race
## gender      Asian      Black      NatAm      Other      White
##      F   53.71714  375.59431   45.55940   13.66782 1021.46870
##      M   96.28286  224.40569   54.44060   16.33218 1098.53130
# Look at sum of raked weights for margins
svytotal(~factor(gender),drake2)
##                 total     SE
## factor(gender)F  1510 0.0028
## factor(gender)M  1490 0.0028
svytotal(~factor(race),drake2)
##                   total SE
## factor(race)Asian   150  0
## factor(race)Black   600  0
## factor(race)NatAm   100  0
## factor(race)Other    30  0
## factor(race)White  2120  0
```

After raking, the standard errors for the population counts for the raking categories equal 0 (approximately 0 for gender since the raking procedure stops when a preset tolerance is reached; you can change that tolerance if desired). After raking, there is no longer any sampling variability for the estimated population counts for each race category—the raking process has forced them to equal the population counts from the external source, which are assumed to be known exactly.

You can also see the raking iterations, if desired, by typing

```
rake(drake, list(~gender,~race), list(pop.gender, pop.race),
     control=list(verbose=TRUE)).
```

Other R functions for nonresponse weight adjustments. The `survey` package has several other functions for nonresponse weight adjustments. The *calibrate* function will perform calibration with general auxiliary information, as described in Section 11.6 of SDA. The *trimWeights* function will trim weights that are outside of user-defined bounds for the weights, and redistribute the trimmings among the untrimmed observations (this keeps the sum of the weights the same). Valliant et al. (2018) go through the detailed steps for constructing weights and describe additional resources for nonresponse weight adjustments using R.

8.3 Imputation

Many data sets have item nonresponse in addition to unit nonresponse. Item nonresponse occurs when a unit has responses for some of the items on the survey but missing values for other items. A person may answer questions about age, race, gender, and health outcomes but decline to answer questions about income. Or the data editing process may uncover logical inconsistencies in the data, such as a 6-year-old who is listed as being married. If the discrepancy cannot be resolved, both age and marital status may be recoded as missing for that person. In this section, we assume that the imputation is limited to missing items.

Example 8.9 of SDA. It is easy to perform simple imputation methods such as cell mean or regression imputation in R. We do not recommend these methods for most applications, since they do not preserve multivariate relationships and do not provide a means of accounting for the imputation when calculating standard errors. But they may be useful when there are, say, a handful of missing items in one of the weighting variables or if it is desired to have an initial imputation before moving on to one of the more advanced imputation methods.

Cell mean imputation is a special case of regression imputation, where the explanatory variables are the factors defining the cells. For cell mean imputation, we first define the cells that cross-classify the observations by gender and age group, then use the *tapply* function to calculate the mean age for the non-missing values in each cell. Note that we also calculate a matrix *impute.flag* that tells which observations were imputed.

```
##### cell mean imputation #####
impute$education
##  [1] 16 NA 11 NA 12 NA 20 12 13 10 12 12 11 16 14 11 14 10 12 10
impute.cm<-impute
# define matrix giving imputation flags, TRUE for each missing value
impute.flag<-is.na(impute)
# fit two-way model with interaction, omit NAs from model-fitting
edmodel<-lm(education~factor(gender)*factor(age>=35),
            data=impute.cm,na.action=na.omit)
# replace missing values with imputations from model
newdata<- impute[is.na(impute$education),]
impute.cm$education[is.na(impute$education)] <- predict(edmodel,newdata)
impute.cm$education
##  [1] 16.00 12.25 11.00 11.25 12.00 12.25 20.00 12.00 13.00 10.00 12.00 12.00
## [13] 11.00 16.00 14.00 11.00 14.00 10.00 12.00 10.00
```

The other variables can be imputed similarly. Although this simple method might suffice if there are only a handful of missing items to be imputed, it will not preserve multivariate relationships. If you want to perform multivariate analyses of imputed data, the imputation model must include the relationships you will be studying in the analysis model. This requires a much more complicated procedure than a simple cell mean or regression model. The imputer must decide on an imputation method and model(s), and decide how to incorporate survey design features into the imputation (see Little and Vartivarian, 2003; Reiter et al., 2006; Kott, 2012, for discussions of this issue).

For most applications, we recommend using one of the many contributed packages for R that perform imputation for survey data. We briefly describe four of them here; Yadav and Roychoudhury (2018) review additional packages.

- The `Hmisc` package (Harrell, 2021) contains functions for single and multiple imputation using additive regression and predictive mean matching.
- The `VIM` package (Kowarik and Templ, 2016; Templ et al., 2021) performs model-based, hot-deck, and nearest-neighbor imputations. The package also provides tools for exploring the missing data patterns and imputations through visualization.
- The `FHDI` package (Im et al., 2018; Cho et al., 2020) imputes multivariate missing data using the fractional hot deck imputation method described by Kim and Fuller (2004) and Kim (2011). In this method, multiple donors contribute to each imputation.
- The `mice` package (van Buuren et al., 2021) performs multivariate imputation by chained equations. This method relies on a sequence of regression models that predict the missing values for each response variable in turn. This is the package that we usually use when

imputing data. Azur et al. (2011) and van Buuren (2018) describe how to perform imputations using this method.

8.4 Summary, Tips, and Warnings

The `survey` package has functions for raking and poststratification that can be used to perform simple nonresponse adjustments of the weights.

Table 8.1 lists the major R functions used in this chapter.

TABLE 8.1
Functions used for Chapter 8.

Function	Package	Usage
is.na	base	Indicate which values are missing; the function returns TRUE if the value is missing and FALSE otherwise
subset	base	Work with a subset of a vector, matrix, or data frame
complete.cases	stats	Indicate which records have complete data; the function returns a vector with the value TRUE if the record has no missing values and FALSE if at least one item is missing
lm	stats	Fit a linear model to a data set (not using survey methods)
predict	stats	Obtain predicted values from a model object
svydesign	survey	Specify the survey design
svymean	survey	Calculate mean and standard error of mean
svytotal	survey	Calculate total and standard error of total
postStratify	survey	Adjust the sampling weights using poststratification
rake	survey	Carry out poststratification to table margins using raking

Tips and Warnings

- Check how missing values are coded before analyzing your data set, and recode the missing values to NA.
- The functions *postStratify* and *rake* in the `survey` package will perform poststratification adjustments to the weights.
- If the survey has item nonresponse and imputation is not used to fill in missing values, treat the observations with non-missing values as a domain using the *subset* function.
- Several R packages are available that perform imputation. Be aware, though, that imputed values are only as good as the model that produces them. Performing a good imputation requires a lot of expertise; the references in the For Further Reading section of Chapter 8 of SDA can help you get started. See Haziza (2009) for a summary of approaches.