5

Cluster Sampling with Equal Probabilities

This chapter shows how to use R to compute estimates from one- and two-stage cluster samples when an SRS is selected at each stage. Chapter 6 will tell how to select a one-stage or two-stage cluster sample with equal or unequal probabilities—the syntax is similar for both, and deferring the sample selection examples to Chapter 6 allows us to look at the general case. The code in this chapter is in file `ch05.R` on the book website.

5.1 Estimates from One-Stage Cluster Samples

Example 5.2 of SDA. The following code and output produces estimates of the population mean and total for the GPA data, using functions from the `survey` package (Lumley, 2020). Variable *suite* identifies the clusters in the data, and variable *wt* is the sampling weight for the persons in the sample, defined as $100/5 = 20$ for every person.

```
data(gpa)
# define one-stage cluster design
# note that id is suite instead of individual student as we take an SRS of suites
dgpa<-svydesign(id=~suite,weights=~wt,fpc=~rep(100,20),data=gpa)
dgpa
## 1 - level Cluster Sampling design
## With (5) clusters.
## svydesign(id = ~suite, weights = ~wt, fpc = ~rep(100, 20), data = gpa)
# estimate mean and se
gpamean<-svymean(~gpa,dgpa)
gpamean
##      mean     SE
## gpa 2.826 0.1637
degf(dgpa)
## [1] 4
# n=5, t-approximation is suggested for CI
confint(gpamean,level=.95,df=4) # use t-approximation
##        2.5 %   97.5 %
## gpa 2.371593 3.280407
# confint(gpamean,level=.95) # uses normal approximation, if desired (for large n)
# estimate total and se (if desired)
gpatotal<-svytotal(~gpa,dgpa)
gpatotal
##      total     SE
## gpa 1130.4 65.466
confint(gpatotal,level=.95,df=4)
##        2.5 %   97.5 %
## gpa 948.6374 1312.163
```

DOI: 10.1201/9781003228196-5

The following features of the code and output deal with the cluster sampling:

- The `id=~suite` argument in *svydesign* tells all functions that use the design object that variable *suite* contains the primary sampling unit (psu) identifiers. If you omit the psu variable and instead use `id=~`, the data will be (incorrectly) analyzed as an SRS.
- The `weights=~wt` argument says that variable *wt* contains the sampling weights for the observation units (here, the students). If `weights=` is omitted but `fpc=` is supplied, selection probabilities are calculated from the population sizes, assuming an SRS of psus. That results in the same weights for this survey, but we recommend including the `weights=` argument as routine practice because most surveys have some type of adjustment that causes the final weights to differ from the sampling weights. (There is one exception, and that is for the without-replacement variance calculations discussed in Section 5.2.)
- The `fpc=~rep(100,20)` argument indicates that the total number of psus in the population is 100. When the *fpc* argument is included, functions use the finite population correction (fpc) when calculating variances. Omit `fpc=` if you do not want an fpc (but then make sure you include the `weights=` argument).
- Typing `dgpa` shows that this is a "1 - level Cluster Sampling design With (5) clusters". The *svydesign* function recognizes this as a one-stage design because one clustering variable is included in the *id* argument. This is the only indication in the output that the clustering was used in the analysis. Otherwise, the form of the statistics output is the same as for simple random or stratified sampling. Always check that the number of clusters in the design object equals the number of psus in your sample.
- The degrees of freedom (df), from function *degf*, equals the number of psus minus 1.
- The *svymean* and *svytotal* functions produce standard errors (SEs) for estimated population means and totals that account for the clustering in the design.

You can verify the calculations using the formulas given in Section 5.2 of SDA if desired.

```
# you can also calculate SEs by direct formula
suitesum<-tapply(gpa$gpa,gpa$suite,sum)  #sum gpa for each suite
# variability comes from among the suites
st2<-var(suitesum)
st2
## [1] 2.25568
# SE of t-hat, formula (5.3) of SDA
vthat <-100^2*(1-5/100)*st2/5
sqrt(vthat)
## [1] 65.46596
# SE of ybar, formula (5.6) of SDA
sqrt(vthat)/(4*100)
## [1] 0.1636649
```

The variability *st2* is coming from the suite totals, and the fpc $(1 - 5/100)$ is applied to calculate the variance of $\hat{t}$. The SE of $\hat{t}$ is 65.46596, which is the same as calculated by *svytotal*.

The procedure for calculating estimates from one-stage cluster samples is exactly the same when the psus have unequal sizes.

Example 5.6 of SDA. Data set *algebra* has 12 classes (clusters) with unequal sizes selected from 187 classes. The syntax for analyzing the data with unequal-sized clusters is exactly

the same as for Example 5.2. Again, note that we use the number of clusters minus 1 (= 11) as the df.

```
data(algebra)
algebra$sampwt<-rep(187/12,299)
# define one-stage cluster design
dalg<-svydesign(id=~class,weights=~sampwt,fpc=~rep(187,299), data=algebra)
dalg
## 1 - level Cluster Sampling design
## With (12) clusters.
## svydesign(id = ~class, weights = ~sampwt, fpc = ~rep(187, 299),
##     data = algebra)
# estimate mean and se
svymean(~score,dalg)
##         mean     SE
## score 62.569 1.4916
# n=12, t-distribution is suggested for CI
degf(dalg)
## [1] 11
confint(svymean(~score,dalg),level=.95,df=11) #use t-approximation
##          2.5 %  97.5 %
## score 59.28562 65.8515
# estimate total and se if desired
svytotal(~score,dalg)
##        total    SE
## score 291533 19893
confint(svytotal(~score,dalg),level=.95,df=11)
##          2.5 %   97.5 %
## score 247749.4 335316.6
```

5.2 Estimates from Multi-Stage Cluster Samples

Calculating estimates from a multi-stage cluster sample is similar. The clustering structure is specified for the design object in the *svydesign* function, and then all functions called with that design object account for the clustering in the variance calculations.

There are several ways to estimate variances using the `survey` package. Let's start with Example 5.8 of SDA, where we calculate the with-replacement variance, and then discuss the issues involved for calculating variances for without-replacement samples using the schools data in Example 5.7 of SDA.

Example 5.8 of SDA. The *coots* data come from Arnold's (1991) work on egg size and volume of American coot eggs in Minnedosa, Manitoba, with a sample of 184 clutches (nests of eggs). Variable *csize* gives the number of eggs in the clutches. Two eggs (secondary sampling unit, ssu) are randomly selected from each clutch (psu). Since we do not have information on the total number of psus N, we use the relative weights *relwt* defined by *csize/2* to calculate the mean volume of eggs and its standard error.

```
data(coots)
# Want to estimate the mean egg volume
nrow(coots) #368
## [1] 368
```

```
coots$ssu<-rep(1:2,184) # index of ssu
coots$relwt<-coots$csize/2
head(coots)
##   clutch csize length breadth    volume tmt ssu relwt
## 1      1    13  44.30   31.10 3.7957569   1   1   6.5
## 2      1    13  45.90   32.70 3.9328497   1   2   6.5
## 3      2    13  49.20   34.40 4.2156036   1   1   6.5
## 4      2    13  48.70   32.70 4.1727621   1   2   6.5
## 5      3     6  51.05   34.25 0.9317646   0   1   3.0
## 6      3     6  49.35   34.40 0.9007362   0   2   3.0
dcoots<-svydesign(id=~clutch+ssu,weights=~relwt,data=coots)
dcoots
## 2 - level Cluster Sampling design (with replacement)
## With (184, 368) clusters.
## svydesign(id = ~clutch + ssu, weights = ~relwt, data = coots)
svymean(~volume,dcoots)  #ratio estimator
##          mean    SE
## volume 2.4908 0.061
confint(svymean(~volume,dcoots),level=.95,df=183)
##           2.5 %   97.5 %
## volume 2.370423 2.611134
# now only include psu information, results are the same
dcoots2<-svydesign(id=~clutch,weights=~relwt,data=coots)
dcoots2
## 1 - level Cluster Sampling design (with replacement)
## With (184) clusters.
## svydesign(id = ~clutch, weights = ~relwt, data = coots)
svymean(~volume,dcoots2)
##          mean    SE
## volume 2.4908 0.061
```

In *svydesign*, the two stages of the cluster sampling are given as `id=~clutch+ssu`. The formula lists the psus first and then the ssus. When the with-replacement variance is calculated, however, as is done here, you need only specify the psus—the point and variance estimates are the same whether you specify just the psus or you specify all stages of sampling. The *weight* argument must be included for this design because the weights are unequal.

Note that the confidence interval uses a t critical value with 183 df (number of psus minus 1). Also note that *svydesign* does not contain the *fpc* argument. This is because the total number of clutches in the population, N, is unknown. As a result, the *svymean* does not use an fpc when calculating estimates. In general, we recommend omitting the *fpc* argument for multi-stage cluster sampling even when N is known, and the remainder of this section discusses this issue.

Variance estimation for without-replacement two-stage cluster samples. Here are two options for estimating the variance of estimated means and totals in without-replacement two-stage sampling, where an SRS is selected at each stage.

Option 1. Calculate the with-replacement variance (recommended). As shown in Sections 5.3 and 6.6 of SDA, the estimated variability among estimated psu totals, s_t^2, also includes variability from the subsequent stages of sampling. If you estimate the with-replacement variance (at the psu level), the variance estimator incorporates *all* the variability from subsequent stages of sampling. The expected value of the with-replacement variance estimator is larger than the true variance of the without-replacement sample, but the difference is small if the sampling fraction at the psu level, n/N, is small.

Chapter 6 of SDA outlines additional benefits of ignoring the fpcs when the psus are selected with unequal probabilities.

To estimate the with-replacement variance for a multi-stage cluster sample, simply call the *svydesign* function as:

```
svydesign(id=~psuid,weights=~weightvariable,data=dataset)
```

where the *id* formula consists only of the variable giving the psu membership. Do not include the *fpc* argument when calling *svydesign*. Then *svytotal*, *svymean*, and other functions will calculate the with-replacement variance.

Option 2. Calculate the without-replacement variance. When an SRS or stratified random sample is taken at all stages of sampling, you can specify all stages of sampling in the *svydesign* function and calculate without-replacement variances. For a two-stage sample, call the *svydesign* function as:

`svydesign(id=~psuid+ssuid,fpc=~psufpc+ssufpc,data=dataset)`.

The *id* formula gives the variable identifying the psu membership followed by the variable identifying the ssu membership. The *fpc* formula has *psufpc*, the variable giving the population number of psus (= `rep(N,nrow(dataset))`), followed by *ssufpc*, the variable giving the values of the psu size for each observation (= M_i for ssus in psu i).

No *weights* argument is included. When the *weights* argument is omitted, it is assumed that an SRS is taken at both stages, and the inclusion probabilities are calculated from the population sizes given in *fpc* and the sample sizes in the data set. Thus, the weights are assumed to be $(NM_i)/(nm_i)$, where the values of n and m_i are counted from the data set and the values of N and M_i are given in the *fpc* arguments.

If there are more than two stages of sampling, and an exactly unbiased estimate of the variance is desired, you need to include terms for all stages of sampling in the *id* and *fpc* arguments of the *svydesign* function. If a survey has three stages, the without-replacement variance estimate requires knowledge of the psu membership, population size, and sample size; the ssu membership, population size, and sample size; and the tertiary sampling unit membership, population size, and sample size.

It can be complicated to keep track of all this information. In addition, calculations are done under the assumption that the final weights are the same as the sampling weights (computed as the inverse of the inclusion probabilities)—that is, there are no nonresponse adjustments or other modifications of the sampling weights.

Example 5.7 of SDA: With-replacement variance. Let's look at the with- and without-replacement variance calculations for the *schools* data. The following code calculates the with-replacement variance. Note that only the psu-level clustering is specified in the *id* argument and that the vector of student-level weights is provided. We can also estimate the proportion and the total number of students having *mathlevel*=2 by treating *mathlevel* as a factor variable.

```
data(schools)
head(schools)
##   schoolid gender math reading mathlevel readlevel  Mi finalwt
## 1        9      F   42      42         2         2 163  61.125
## 2        9      F   29      30         1         1 163  61.125
## 3        9      M   31      25         1         1 163  61.125
## 4        9      F   22      33         1         2 163  61.125
## 5        9      M   35      36         1         2 163  61.125
```

```
## 6        9      F   30      17         1         1 163  61.125
# calculate with-replacement variance; no fpc argument
# include psu variable in id; include weights
dschools<-svydesign(id=~schoolid,weights=~finalwt,data=schools)
# dschools tells you this is treated as a with-replacement sample
dschools
## 1 - level Cluster Sampling design (with replacement)
## With (10) clusters.
## svydesign(id = ~schoolid, weights = ~finalwt, data = schools)
mathmean<-svymean(~math,dschools)
mathmean
##        mean     SE
## math 33.123 1.7599
degf(dschools)
## [1] 9
# use t distribution for confidence intervals because there are only 10 psus
confint(mathmean,df=degf(dschools))
##         2.5 %  97.5 %
## math 29.14179 37.1041
# estimate proportion and total number of students with mathlevel=2
svymean(~factor(mathlevel),dschools)
##                         mean     SE
## factor(mathlevel)1 0.71231 0.0542
## factor(mathlevel)2 0.28769 0.0542
svytotal(~factor(mathlevel),dschools)
##                        total      SE
## factor(mathlevel)1 12303.4 2244.14
## factor(mathlevel)2  4969.1  676.26
```

Example 5.7 of SDA: Without-replacement variance. The *svymean* function will calculate without-replacement variances when simple or stratified random sampling is used at each stage. (As of this writing, it does not do so for all designs and thus will not compute the without-replacement variance for most of the unequal-probability samples that are discussed in Chapter 6.) To use it with the *schools* data, put both stages of clustering in the *id* argument and put both the psu and the ssu population sizes in the *fpc* argument. Do not include the *weights* argument.

```
# create a variable giving each student an id number
schools$studentid<-1:(nrow(schools))
# calculate without-replacement variance
# specify both stages of the sample in the id argument
# give both sets of population sizes in the fpc argument
# do not include the weight argument
dschoolwor<-svydesign(id=~schoolid+studentid,fpc=~rep(75,nrow(schools))+Mi,
                      data=schools)
dschoolwor
## 2 - level Cluster Sampling design
## With (10, 200) clusters.
## svydesign(id = ~schoolid + studentid, fpc = ~rep(75, nrow(schools)) +
##     Mi, data = schools)
mathmeanwor<-svymean(~math,dschoolwor)
mathmeanwor
##        mean     SE
## math 33.123 1.6605
confint(mathmeanwor,df=degf(dschoolwor))
##         2.5 %   97.5 %
```

```
## math 29.36667 36.87923
# estimate proportion and total number of students with mathlevel=2
svymean(~factor(mathlevel),dschoolwor)
##                         mean     SE
## factor(mathlevel)1 0.71231 0.0516
## factor(mathlevel)2 0.28769 0.0516
svytotal(~factor(mathlevel),dschoolwor)
##                        total      SE
## factor(mathlevel)1 12303.4 2097.83
## factor(mathlevel)2  4969.1  657.69
```

In the *schools* data, variable *Mi* gives the population number of students in each school. This information must be available in the data set to be able to calculate the without-replacement variance. The design object *dschoolwor* repeats that this is a "2-level Cluster Sampling Design" with 10 psus and 200 ssus.

Even with the relatively large sampling fractions in this example, the with- and without-replacement standard errors are similar. For variable *math*, the with-replacement standard error is 1.76, and the without-replacement standard error is 1.66.

In general, we recommend calculating the with-replacement variance (omitting the *fpc* argument) for multi-stage cluster sampling. It produces a variance estimate whose expectation is slightly larger than the true variance, but if n/N is small, the difference is negligible. If forced to choose between a standard error that is slightly too large and one that is too small, we usually prefer the former because a too-small standard error leads to claiming that estimates are more precise than they really are.

The most important thing to keep in mind for computing standard errors for cluster samples is that ssus in the same psu are usually more homogeneous than randomly selected ssus from the population. Thus, the essential feature for calculating standard errors is to capture that homogeneity by including the `id=~psuid` argument in *svydesign*. The issue of "to fpc or not to fpc" is minor compared with the effect of clustering.

5.3 Model-Based Design and Analysis for Cluster Samples

We often use models when designing a cluster sample, as shown in Section 5.4 of SDA. Data from a previous survey or pilot sample may be used to estimate the optimal subsampling or psu size. This often involves estimating the value of R^2 or R^2_a, which can be obtained from an Analysis of Variance (ANOVA) table.

Example 5.12 of SDA. The following shows how to derive an ANOVA table for the schools data.

```
# run lm with schoolid as a factor
fit5.12<-lm(math~factor(schoolid), data=schools)
# print ANOVA table
anova(fit5.12)
## Analysis of Variance Table
##
## Response: math
##                    Df  Sum Sq Mean Sq F value    Pr(>F)
## factor(schoolid)   9  7018.5  779.83  7.5834 1.785e-09 ***
```

```
## Residuals        190 19538.4  102.83
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
# extract the value of R-squared and adjusted R-squared
summary(fit5.12)$r.squared
## [1] 0.264281
summary(fit5.12)$adj.r.squared
## [1] 0.2294312
```

Example 5.14 of SDA. This example employs a random effects model, in which the school means are assumed to be normally distributed random variables with mean μ. In packages `nlme` (Pinheiro et al., 2021) and `lme4` (Bates et al., 2015, 2020), the *lme* (short for linear mixed effects) and *lmer* functions, respectively, calculate estimates from random effects models.

We use function *lme* for this example. For the one-way random effects model, the only fixed effect is the mean, so the *fixed* formula is `fixed=math~1`. Random effects are specified in the *random* argument so *factor(schoolid)*, the factor variable describing the psu membership, is placed behind the vertical bar in the *random* argument.

```
library(nlme)
fit5.14 <- lme(fixed=math~1,random=~1|factor(schoolid),data=schools)
summary(fit5.14)
## Linear mixed-effects model fit by REML
##   Data: schools
##        AIC      BIC    logLik
##   1516.259 1526.139 -755.1295
##
## Random effects:
##  Formula: ~1 | factor(schoolid)
##         (Intercept) Residual
## StdDev:    5.818064 10.14069
##
## Fixed effects:  math ~ 1
##             Value Std.Error  DF  t-value p-value
## (Intercept) 34.66  1.974628 190 17.55267       0
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -2.26713655 -0.74262324 -0.09451607  0.79142521  2.18576500
##
## Number of Observations: 200
## Number of Groups: 10
# extract the variance components
VarCorr(fit5.14)
## factor(schoolid) = pdLogChol(1)
##             Variance  StdDev
## (Intercept)  33.84987  5.818064
## Residual    102.83368 10.140694
```

A model-based analysis predicts the values of observations that are not observed in the data. For this data set, the unobserved values are the students who are not measured in the sampled schools, as well as the unsampled schools in the population. The estimated mean 34.66 from the output under `"Fixed effects: math ~ 1"` does not account for the population sizes of the schools, and gives a different estimate than in Example 5.7 of SDA.

The residuals and predicted values from the model can be obtained by requesting `resid(fit5.14)` and `predict(fit5.14)`. You can also type `plot(fit5.14)` to obtain a plot of standardized residuals versus fitted values.

5.4 Additional Code for Exercises

Exercise 5.40 of SDA. The exercise uses the function *intervals_ex40*, available from the book website and R package `SDAResources` (Lu and Lohr, 2021). To run the function, load the `SDAResources` package and type

```
intervals_ex40(groupcorr=0, numintervals=100, groupsize=5, sampgroups=10,
               popgroups=5000, mu=0, sigma=1)
```

using the desired values for the arguments. The function call given above uses the default values of the arguments and will give the same results as running `intervals_ex40()`.

The arguments of function *intervals_ex40* are given in Table 5.1.

TABLE 5.1
Arguments of the *intervals_ex40* function.

Argument	Description
groupcorr	Desired intraclass correlation coefficient, must be between 0 and 1 (default is 0).
numintervals	Number of confidence intervals to be generated (default is 100).
groupsize	Number of observations, M, in each population cluster (default is 5).
sampgroups	Number of clusters to be sampled (default is 10).
popgroups	Number of clusters in population (default is 5000). This should be set to be at least 200 times as large as the value of *sampgroups* so that the fpc is negligible.
mu	Population mean (default is 0).
sigma	Population standard deviation (default is 1).

For the exercise, you are asked to generate 100 intervals of 50 observations each, taken in 10 clusters of size 5. This uses the default values of all arguments except for ICC. When running the function, you only need to specify the arguments that differ from the default values, so that you can generate 100 intervals with ICC = 0.3 by running the function with argument `groupcorr=0.3`.

```
set.seed(9231)
# generate intervals for cluster sample with groupcorr = 0.3
intervals_ex40(groupcorr = 0.3) # leave other parameters unchanged
##  Number_of_intervals       SRS_cover_prob   Cluster_cover_prob
##          100.0000000            0.8000000            0.9500000
##    SRS_mean_CI_width Cluster_mean_CI_width
##            0.5556272            0.9111856
##      Replicate mu sample_mean     srs_lci    srs_uci in_srs_ci SRS_CI_width
## [1,]         1  0 -0.07311322 -0.35806539 0.21183894         1    0.5699043
## [2,]         2  0  0.25038517 -0.05127437 0.55204471         1    0.6033191
## [3,]         3  0  0.03499401 -0.27232646 0.34231448         1    0.6146409
## [4,]         4  0 -0.18948478 -0.45038074 0.07141119         1    0.5217919
```

```
## [5,]         5  0  0.14058713 -0.08585203 0.36702629         1    0.4528783
##        clus_lci  clus_uci in_clu_ci clus_CI_width
## [1,] -0.6526766 0.5064502         1     1.1591268
## [2,] -0.2716028 0.7723731         1     1.0439759
## [3,] -0.4941334 0.5641214         1     1.0582548
## [4,] -0.5814274 0.2024579         1     0.7838853
## [5,] -0.2344528 0.5156271         1     0.7500799
```

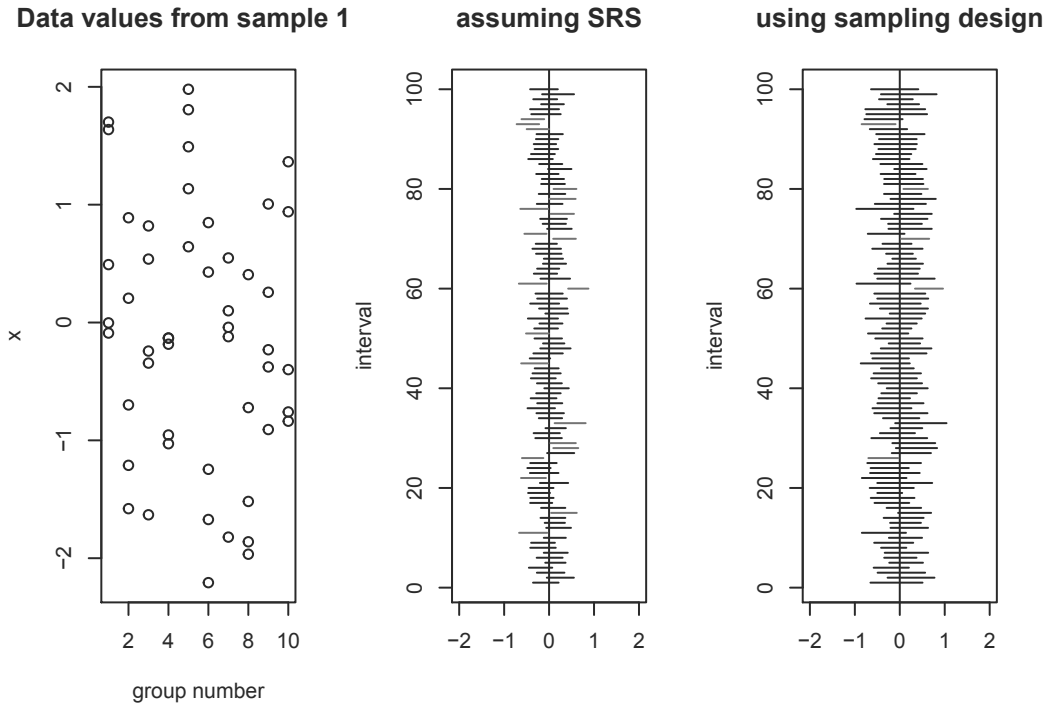

FIGURE 5.1: Interval estimates created assuming SRS and using clustering formulas.

We initialized the random number seed so that we could reproduce the intervals later, but if you are repeating the exercise, you may want to let the computer generate your starting seed (otherwise, you may get the same set of samples each time).

- Function *intervals_ex40* calculates two sets of interval estimates: a set that uses SRS formulas and hence has coverage probability (proportion of intervals that include the true population mean 0) `SRS_cover_prob` of 0.80 that is less than 0.95, and a second set that calculates the correct confidence intervals using the formulas for one-stage cluster sampling with coverage probability `Cluster_cover_prob` of 0.95.
- It also prints the average width of the interval estimates for the two methods: `SRS_mean_CI_width` of 0.5556272, which is less than `Cluster_mean_CI_width` of 0.9111856.
- The first five replicates and their summary statistics are printed, where `srs_lci` is the lower limit, and `srs_uci` is the upper limit from the SRS estimate. Similarly, `clus_lci` and `clus_uci` are the lower and upper confidence limits from the estimate calculated using the clustering. If desired, the function can be modified so that information from all replicates is stored in a data set.

- Three graphs are produced, similar to those in Figure 5.1. The first graph shows a scatterplot of the last simulated sample, the second graph shows the interval estimates produced for each sample if analyzed as an SRS, and the third shows the interval estimates produced for each sample when analyzed as a cluster sample. When the graph is produced in color, intervals that include the true value of the population mean are black, and those that do not include the true value are red.

  The estimated coverage probability for each procedure is the proportion of intervals that include the true population mean. In Figure 5.1, the estimated coverage probability of the procedure that (incorrectly) treats the data as an SRS is 0.80—substantially smaller than the nominal 0.95 probability of a confidence interval.

5.5 Summary, Tips, and Warnings

Table 5.2 lists the main R functions used in this chapter. We have seen most of these before; the main difference is how the survey design is specified in the *svydesign* function to indicate the clustering.

TABLE 5.2
Functions used for Chapter 5.

Function	Package	Usage
tapply	base	Apply a function to each group of values; groups are defined by the second argument
confint	stats	Calculate confidence intervals, add df for t confidence interval
lm	stats	Fit a linear model to a data set (not using survey methods)
lme	nlme	Fit a random-effects or mixed-effects model to a data set (not using survey methods)
svydesign	survey	Specify the survey design
svymean	survey	Calculate mean and standard error of mean
svytotal	survey	Calculate total and standard error of total
intervals_ex40	SDAResources	Show differences between (incorrect) SRS formulas and (correct) cluster formulas applied to cluster samples

In the `survey` package, clusters are identified in the `id=` argument of the *svydesign* function. The general form of the *svydesign* function for a one-stage cluster sample (without stratification) is:

```
svydesign(id=~psuvar,weights=~weightvar,fpc=rep(N,nobs),data=dataset)
```

where *psuvar* is the name of the variable in *dataset* containing the psu identifiers. The variable *weightvar* contains the weights for the observation units in the data. If an fpc is desired (and often it is not), N is the number of psus in the population, and *nobs* is the number of psus in the sample.

For multi-stage cluster sampling (again without stratification), the following form of the *svydesign* function will calculate point estimates with the weights in *weightvar* and with-replacement standard errors (note the absence of the fpc argument):

`svydesign(id=~psuvar,weights=~weightvar,data=dataset)`.

After specifying the survey design, the *svymean* and *svytotal* functions are used exactly as in other chapters. The only difference for cluster sampling is that you must list the cluster variable(s) in the *svydesign* function.

Tips and Warnings

- Use the *id* argument to specify the clustering, and check that the number of clusters listed when you print the survey design object equals the number of psus in your sample.
- When calculating estimates for one-stage cluster samples, or for two-stage cluster samples using with-replacement variance estimates, include the *weights* argument when specifying the survey design. The weight variable should contain the final weights at the observation level. Check that the sum of the weights approximately equals the number of observation units in the population.
- In general, we recommend calculating with-replacement variances, but the `survey` package functions will also calculate without-replacement variances for the designs discussed in this chapter, where an SRS is taken at each stage of sampling. If you calculate the without-replacement variances for a two-stage cluster design, it is useful to check these by also calculating the with-replacement variance (they should be close).

6

Sampling with Unequal Probabilities

In this chapter, we discuss how to select a sample with equal or unequal probabilities, and how to compute estimates from an unequal probability sample. The code is in file `ch06.R` on the book website.

Let's start with sample selection. Section 6.1 shows how to select a one-stage cluster sample with equal or unequal probabilities, and Section 6.2 presents two methods for selecting a two-stage sample. We'll look at code for computing estimates for unequal-probability samples in Section 6.3.

6.1 Selecting a Sample with Unequal Probabilities

This section shows how to select a sample of primary sampling units (psus) with unequal probabilities with the *sample* function and with functions from the `sampling` package (Tillé and Matei, 2021). Subsampling all secondary sampling units (ssus) in the selected psus will give a one-stage cluster sample.

6.1.1 Sampling with Replacement

Example 6.2 of SDA. In Chapters 2 through 5, units, whether observation units or clusters, were selected with equal probabilities. In Example 6.2, 5 classes are sampled from 15 classes with probability proportional to size (pps) and with replacement.

Section 2.1 showed how to use the *sample* function to select a simple random sample, with or without replacement. It can also be used to select a with-replacement sample with unequal probabilities by including the optional *prob* argument. Call the function as

```
sample(1:N,n,replace=TRUE,prob=probvar)
```

where N is the number of psus in the population, n is the desired sample size of psus, and *probvar* is a vector of length N that gives the size measures or selection probabilities for the psus. In this example, psus are classes, and *class_size* gives the number of students in the class.

```
data(classes)
classes[1:2,]
##   class class_size
## 1     1         44
## 2     2         33
N<-nrow(classes)
set.seed(78065)
# select 5 classes with probability proportional to class size and with replacement
sample_units<-sample(1:N,5,replace=TRUE,prob=classes$class_size)
```

DOI: 10.1201/9781003228196-6