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

```
sample_units
## [1]  5 14  6 14  6
mysample<-classes[sample_units,]
mysample
##      class class_size
## 5        5         76
## 14      14        100
## 6        6         63
## 14.1    14        100
## 6.1      6         63
# calculate ExpectedHits and sampling weights
mysample$ExpectedHits<-5*mysample$class_size/sum(classes$class_size)
mysample$SamplingWeight<-1/mysample$ExpectedHits
mysample$psuid<-row.names(mysample)
mysample
##      class class_size ExpectedHits SamplingWeight psuid
## 5        5         76    0.5873261       1.702632     5
## 14      14        100    0.7727975       1.294000    14
## 6        6         63    0.4868624       2.053968     6
## 14.1    14        100    0.7727975       1.294000  14.1
## 6.1      6         63    0.4868624       2.053968   6.1
# check sum of sampling weights
sum(mysample$SamplingWeight)
## [1] 8.398568
```

- Note that classes 6 and 14 both appear twice in the sample. When collecting data in one stage, each student within classes 6 and 14 must be included twice. Otherwise, estimates will be biased. If collecting data in two stages, you would take two independent subsamples from class 6 and two independent subsamples from class 14.

  When analyzing the data, make sure you use different psu names for the multiple instances of psus that appear more than once. For this example, you might want to use `row.names(mysample)` as the psu identifier, since it gives a unique name to each sampled psu.

- After selecting the sample, we need to calculate the sampling weights. The *ExpectedHits* variable gives $n\psi_i$, where n is the sample size and ψ_i is the draw-by-draw selection probability that is proportional to *class_ size* (note that the values of ψ_i sum to 1). This is the number of times we expect the unit to be in the sample. For example, class 5 has *ExpectedHits* = 5*76/647 = 0.5873261.

  Then, the *SamplingWeight* is $1/ExpectedHits = 1/(n\psi_i)$ for with-replacement sampling.

- The sum of the sampling weights for the sample is an unbiased estimator of N, and for large samples it should be close to N. For this example, however, the weight sum has a large standard error because the sample size is so small (see Exercise 6.45 of SDA).

6.1.2 Sampling without Replacement

There are several functions in the `sampling` package that will select unequal-probability samples without replacement. When the list of units in the sampling frame is in random order, systematic sampling is likely to produce a sample that behaves like an SRS without replacement. The *cluster* function in the `sampling` package can select a pps sample using systematic sampling.

The following code shows how the *cluster* function is called.

```
set.seed(330582)
cluster(data=classes, clustername=c("class"), size=5, method="systematic",
        pik=classes$class_size,description=TRUE)
## Number of selected clusters: 5
## Number of units in the population and number of selected units: 15 5
##   class ID_unit      Prob
## 1     1       1 0.3400309
## 2     5       5 0.5873261
## 3     8       8 0.3400309
## 4    11      11 0.3554869
## 5    14      14 0.7727975
```

The following arguments of the *cluster* function are used:

- `pik` is the vector of inclusion probabilities (or a vector of relative unit sizes that can be used to compute the probabilities). In this example, *pik* is *class_ size*, the auxiliary variable that gives the size of each class.
- `size=5` requests a sample of 5 units.
- `method="systematic"` describes the method used to select the sample. The *cluster* function can also be called with methods `"srswor"` (simple random sampling without replacement), `"srswr"` (simple random sampling with replacement), or `"poisson"` (Poisson sampling). The *pik* argument is not needed with methods srswor and srswr.
- `description=TRUE` asks the function to print the number of population and sampled units.

The *cluster* function creates variable *Prob* that includes the final inclusion probabilities for the units in the sample. You can compute sampling weights as $1/Prob$.

Other functions are also available for selecting unequal-probability samples. Table 6.1 lists sample selection functions in the `sampling` package that correspond to methods discussed in Chapters 5 and 6 of SDA. Tillé (2006) describes these and additional methods in the `sampling` package (the functions that select unequal-probability samples have names that begin with *UP*) for selecting samples. Another resource is the `pps` package (Gambino, 2021), which contains several functions for selecting unequal-probability samples.

You may want to consider writing your own function or using a different software package if none of the methods implemented in R meet your sample selection needs. The SURVEYSELECT procedure in SAS software provides additional options for selecting without-replacement probability samples, including the Hanurav–Vijayan method (Hanurav, 1967; Vijayan, 1968); see SAS Institute Inc. (2021) and Lohr (2022) for details.

6.2 Selecting a Two-Stage Cluster Sample

There are several ways to select a two-stage cluster sample in R. The *mstage* function from the `sampling` package will select both stages at once for simple random, systematic, or Poisson sampling. Alternatively, you can select the units at each stage separately: First select the psus, then select a sample of ssus from the selected psus.

TABLE 6.1
Some functions for selecting a probability sample in the `sampling` package.

Function	Description
UPbrewer(pik)	Select an unequal-probability sample without replacement containing 2 psus per stratum using Brewer's (1963, 1975) method. The *pik* argument contains the desired inclusion probabilities π_i. Note that there is no argument for the sample size. For all of the UP sample selection methods, the sample size is assumed to be implicit in the *pik* vector because $\sum_{k=1}^{N} \pi_k = n$. The function `inclusionprobabilities(size,n)` will compute *pik* from a vector *size* of positive numbers and desired sample size n.
UPpoisson(pik)	Select a sample (of variable size) using Poisson sampling. The *pik* argument contains the desired inclusion probabilities π_k for each unit, and these should be between 0 and 1.
UPsystematic(pik)	Select an unequal-probability sample via systematic sampling.
UPsampford(pik)	Select an unequal-probability sample using Sampford's (1967) method, an extension of Brewer's method that allows drawing more than 2 psus per stratum.
srswor(n,N)	Select an SRS of size n without replacement from a population of size N.
srswr(n,N)	Select an SRS of size n with replacement from a population of size N.

Example 6.11 of SDA: Selection with *mstage* function. In Chapter 1, we expanded data `classes` to a long format that includes student information. Let's redo that here.

```
# create data frame classeslong
data(classes)
classeslong<-classes[rep(1:nrow(classes),times=classes$class_size),]
classeslong$studentid <- sequence(classes$class_size)
nrow(classeslong)
## [1] 647
table(classeslong$class) # check class sizes
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
##  44  33  26  22  76  63  20  44  54  34  46  24  46 100  15
head(classeslong)
##     class class_size studentid
## 1       1         44         1
## 1.1     1         44         2
## 1.2     1         44         3
## 1.3     1         44         4
## 1.4     1         44         5
## 1.5     1         44         6
```

We now use the *mstage* function to select a pps systematic sample of 5 classes and take an SRS without replacement of 4 students from each class. Each class in the psu sample can therefore be considered as a stratum for sample selection purposes, and an independent sample of size 4 is taken from each stratum. We call the function as

```
mstage(classeslong,stage=list("cluster","stratified"),
            varnames=list("class","studentid"),
            size=numberselect, method=list("systematic","srswor"),pik=prob)
```

In the function *mstage*, sampling specifications for the different stages are given in list objects. Lists in R allow you to combine structures of different type; here, the lists consist of vectors that have different lengths. If there are four stages of sampling, each list will have four components, each in the order of the stages of sampling. This example is for two-stage sampling, so each list has two components.

- The two stages are denoted by `stage=list("cluster","stratified")`, and the corresponding list naming the stratification or clustering variables at the stages is `varnames=list("class","studentid")`.

  For this example, a cluster sample of psus, identified by *class*, is desired for the first stage of sampling. After the psus are selected, the frame for the second stage of sampling consists of the listing of ssus for the sample of 5 psus. An SRS is selected from each of those 5 psus, so the method used is `"stratified"`.

- The desired sample sizes are given in the *size* argument as a list containing two levels. Here, we set *size* equal to the list `numberselect<-list(5,rep(4,5))`. We want to select a sample of 5 psus and then a subsample of 4 ssus from each sampled psu.

- The sampling methods at the two stages, systematic and simple random sampling respectively, are denoted by `method=list("systematic","srswor")`.

- The selection probabilities are given in `pik=prob`, where

  ```
  prob<-list(classes$class_size/647,4/classeslong$class_size)
  or
  prob<-list(classes$class_size/647)
  # srswor is with probability of 4/classeslong$class_size by default
  # since ssu sample size of 4 is supplied in numberselect argument
  ```

As always, you can set the seed to any integer you like; this allows you to re-create the sample later. Note that the sample in this book has different psus than the sample in SDA, which was selected using SAS software (SAS Institute Inc., 2021).

```
# select a two-stage cluster sample, psu: class, ssu: studentid
# number of psus selected: n = 5 (pps systematic)
# number of students selected: m_i = 4 (srs without replacement)
# problist<-list(classes$class_size/647) # same results as next command
problist<-list(classes$class_size/647,4/classeslong$class_size) #selection prob
problist[[1]]  # extract the first object in the list. This is pps, size M_i/M
##  [1] 0.06800618 0.05100464 0.04018547 0.03400309 0.11746522 0.09737249
##  [7] 0.03091190 0.06800618 0.08346213 0.05255023 0.07109737 0.03709428
## [13] 0.07109737 0.15455951 0.02318393
problist[[2]][1:5] # first 5 values in second object in list
## [1] 0.09090909 0.09090909 0.09090909 0.09090909 0.09090909
# number of psus and ssus
n<-5
numberselect<-list(n,rep(4,n))
numberselect
## [[1]]
## [1] 5
##
## [[2]]
```

```
## [1] 4 4 4 4 4
# two-stage sampling
set.seed(75745)
tempid<-mstage(classeslong,stage=list("cluster","stratified"),
               varnames=list("class","studentid"),
               size=numberselect, method=list("systematic","srswor"),pik=problist)
```

The output *tempid* contains two objects

`sample1<-getdata(classeslong,tempid)[[1]]`
`sample2<-getdata(classeslong,tempid)[[2]]`.

Here, *sample1* contains the classes selected at stage 1 along with the selection probabilities *Prob_ 1 _stage*, and *sample2* contains the selected ssus and the second-stage sampling probabilities, as well as the final selection probabilities *Prob* (which equals the product of *Prob_ 1 _stage* and *Prob_ 2 _stage*). We only need *sample2* but also show *sample1* so you can see the first-stage probabilities.

```
# get data
sample1<-getdata(classeslong,tempid)[[1]]
# sample 1 contains the ssus of the 5 psus chosen at the first stage
# Prob_ 1 _stage has the first-stage selection probabilities
head(sample1)
##      class_size studentid class ID_unit Prob_ 1 _stage
## 4.21         22        22     4     125      0.1700155
## 4.20         22        21     4     124      0.1700155
## 4.6          22         7     4     110      0.1700155
## 4            22         1     4     104      0.1700155
## 4.7          22         8     4     111      0.1700155
## 4.8          22         9     4     112      0.1700155
nrow(sample1)
## [1] 285
table(sample1$class) # lists the psus selected in the first stage
##
##   4   6   9  13  14
##  22  63  54  46 100
sample2<-getdata(classeslong,tempid)[[2]]
# sample 2 contains the final sample
# Prob_ 2 _stage has the second-stage selection probabilities
# Prob has the final selection probabilities
head(sample2)
##      class class_size studentid ID_unit Prob_ 2 _stage      Prob
## 4.21     4         22        22     125     0.18181818 0.0309119
## 4.7      4         22         8     111     0.18181818 0.0309119
## 4.5      4         22         6     109     0.18181818 0.0309119
## 4.19     4         22        20     123     0.18181818 0.0309119
## 6.48     6         63        49     250     0.06349206 0.0309119
## 6.53     6         63        54     255     0.06349206 0.0309119
nrow(sample2) # sample of 20 ssus altogether
## [1] 20
table(sample2$class) # 4 ssus selected from each psu
##
##  4  6  9 13 14
##  4  4  4  4  4
# calculate final weight = 1/Prob
sample2$finalweight<-1/sample2$Prob
```

```
# check that sum of final sampling weights equals population size
sum(sample2$finalweight)
## [1] 647
sample2[,c(1,2,3,6,7)] # print variables from final sample
##       class class_size studentid      Prob finalweight
## 4.21      4         22        22 0.0309119       32.35
## 4.7       4         22         8 0.0309119       32.35
## 4.5       4         22         6 0.0309119       32.35
## 4.19      4         22        20 0.0309119       32.35
## 6.48      6         63        49 0.0309119       32.35
## 6.53      6         63        54 0.0309119       32.35
## 6.23      6         63        24 0.0309119       32.35
## 6.33      6         63        34 0.0309119       32.35
## 9.50      9         54        51 0.0309119       32.35
## 9.29      9         54        30 0.0309119       32.35
## 9.31      9         54        32 0.0309119       32.35
## 9.36      9         54        37 0.0309119       32.35
## 13.10    13         46        11 0.0309119       32.35
## 13       13         46         1 0.0309119       32.35
## 13.45    13         46        46 0.0309119       32.35
## 13.39    13         46        40 0.0309119       32.35
## 14.4     14        100         5 0.0309119       32.35
## 14.78    14        100        79 0.0309119       32.35
## 14.98    14        100        99 0.0309119       32.35
## 14.63    14        100        64 0.0309119       32.35
```

The final sampling weight *finalweight* is the reciprocal of final selection probability. The psus were selected with probabilities $5M_i/647$ and the ssus for psu i were selected with probability $4/M_i$, so the final weight is $[647/(5M_i)](M_i/4) = 647/20 = 32.35$ for each ssu in the sample.

Example 6.11 of SDA: Selection in two steps. Another option, if you want to use a method other than systematic sampling to select the psus with unequal probabilities, is to select the first-stage units and the second-stage units in separate steps. This is often more convenient for populations where the psu sizes M_i are known only for units in the sample. For example, if nursing homes are psus, you may have to find out the value of M_i directly from each home and thus would know these values only after the first-stage sample is selected.

Let's select a sample from data set *classes* in two stages. In this example we use function *UPsampford* (see Table 6.1) instead of systematic sampling to select 5 classes (psus) at the first stage of sample selection. We call

```
UPsampford(pik)
```

where *pik* is a vector of length N containing the desired inclusion probabilities. The function has no argument for the sample size; it is assumed that `sum(pik)=n`. The function *inclusionprobabilities* will compute *pik*, having sum n, from the desired sample size and a vector of positive numbers that gives the relative sizes of the units.

```
# select a cluster sample in two stages, psu: class, ssu: studentid
# number of psu selected n =5 (Sampford's method)
# first, convert the measure of size to a vector of probabilities
classes$stage1prob<-inclusionprobabilities(classes$class_size,5)
sum(classes$stage1prob)  # inclusion probabilities sum to n
## [1] 5
```

```
# select the psus
set.seed(29385739)
stage1.units<-UPsampford(classes$stage1prob)
stage1.sample<-getdata(classes,stage1.units)
stage1.sample
##    ID_unit class class_size stage1prob
## 1        1     1         44  0.3400309
## 3        3     3         26  0.2009274
## 7        7     7         20  0.1545595
## 13      13    13         46  0.3554869
## 14      14    14        100  0.7727975
```

The data frame *stage1.sample* contains the psus (classes) for the sample. Variable *stage1prob* contains the first-stage selection probabilities that we computed from the *class_size* variable. Now we can use the function to select the second-stage units. Since we have already formed the data frame *stage1.sample* that consists only of the sampled psus, the *strata* function provides a convenient way to select an SRS from each sampled psu.

To draw an SRS of students from each sampled psu, we first create data frame *stage1.long* with a data record for each student in the sampled classes, and then take an SRS of 4 students from each class.

```
# first-stage units are in stage1.sample
# now select the second-stage units (students)
# convert the psus in the sample to long format and assign student ids
npsu<-nrow(stage1.sample)
stage1.long<-stage1.sample[rep(1:npsu,times=stage1.sample$class_size),]
stage1.long$studentid<-sequence(stage1.sample$class_size)
head(stage1.long)
##     ID_unit class class_size stage1prob studentid
## 1         1     1         44  0.3400309         1
## 1.1       1     1         44  0.3400309         2
## 1.2       1     1         44  0.3400309         3
## 1.3       1     1         44  0.3400309         4
## 1.4       1     1         44  0.3400309         5
## 1.5       1     1         44  0.3400309         6
# use strata function to select 4 ssus from each psu
stage2.units<-strata(stage1.long, stratanames=c("class"),
                     size=rep(4,5), method="srswor")
nrow(stage2.units)
## [1] 20
# get the data for the second-stage sample
ssusample<-getdata(stage1.long,stage2.units)
head(ssusample)
##      class_size stage1prob studentid class ID_unit       Prob Stratum
## 1.3          44  0.3400309         4     1       4 0.09090909       1
## 1.13         44  0.3400309        14     1      14 0.09090909       1
## 1.21         44  0.3400309        22     1      22 0.09090909       1
## 1.26         44  0.3400309        27     1      27 0.09090909       1
## 3.11         26  0.2009274        12     3      56 0.15384615       2
## 3.18         26  0.2009274        19     3      63 0.15384615       2
```

The last step is computing the final selection probability, accounting for both stages of sampling, and the final sampling weight. In data frame *ssusample*, variable *stage1prob* contains the psu-level sampling probability (we defined this variable earlier and used it to select the psus) and variable *Prob* contains the ssu-level sampling probability (this is computed by the

strata function and for this example equals $4/M_i$). Thus, the selection probability for each student in the sample is the product *stage1prob* × *Prob*. The final weight is the reciprocal of the final selection probability, which for this example equals 32.35 for all students in the sample because the first-stage sample was selected with probability proportional to M_i and the same subsample size (here, $m_i = 4$) was selected from each psu.

```
# compute the sampling weights
# stage1prob contains stage 1 sampling probability;
# Prob has stage 2 sampling probability
ssusample$finalprob<- ssusample$stage1prob*ssusample$Prob
ssusample$finalwt<-1/ssusample$finalprob
sum(ssusample$finalwt)  # check sum of weights
## [1] 647
# print selected columns of ssusample
print(ssusample[,c(1,2,3,4,6,8,9)],digits=4)
##       class_size stage1prob studentid class    Prob finalprob finalwt
## 1.3           44     0.3400         4     1 0.09091   0.03091   32.35
## 1.13          44     0.3400        14     1 0.09091   0.03091   32.35
## 1.21          44     0.3400        22     1 0.09091   0.03091   32.35
## 1.26          44     0.3400        27     1 0.09091   0.03091   32.35
## 3.11          26     0.2009        12     3 0.15385   0.03091   32.35
## 3.18          26     0.2009        19     3 0.15385   0.03091   32.35
## 3.19          26     0.2009        20     3 0.15385   0.03091   32.35
## 3.24          26     0.2009        25     3 0.15385   0.03091   32.35
## 7.11          20     0.1546        12     7 0.20000   0.03091   32.35
## 7.13          20     0.1546        14     7 0.20000   0.03091   32.35
## 7.18          20     0.1546        19     7 0.20000   0.03091   32.35
## 7.19          20     0.1546        20     7 0.20000   0.03091   32.35
## 13.16         46     0.3555        17    13 0.08696   0.03091   32.35
## 13.31         46     0.3555        32    13 0.08696   0.03091   32.35
## 13.34         46     0.3555        35    13 0.08696   0.03091   32.35
## 13.42         46     0.3555        43    13 0.08696   0.03091   32.35
## 14.1         100     0.7728         2    14 0.04000   0.03091   32.35
## 14.20        100     0.7728        21    14 0.04000   0.03091   32.35
## 14.35        100     0.7728        36    14 0.04000   0.03091   32.35
## 14.68        100     0.7728        69    14 0.04000   0.03091   32.35
```

6.3 Computing Estimates from an Unequal-Probability Sample

The syntax used to compute estimates from an unequal-probability cluster sample is largely the same as that used in Chapter 5 for equal-probability cluster samples. The *svymean* and *svytotal* functions of the `survey` package (Lumley, 2020) calculate estimates of means, totals, and proportions by using the formulas with survey weights. When the *fpc* argument is omitted from the *svydesign* function call, standard errors are calculated with the formulas for the with-replacement variance in Section 6.4 of SDA.

6.3.1 Estimates from with-Replacement Samples

Example 6.4 of SDA. This example shows how to calculate estimates when the cluster total t_i has already been found for each psu (or when the psus are also the observation units, that is, $M_i = 1$ for all psus). Since the summary statistic has already been calculated for

each psu, the *svydesign* function is called with `id=~1`. We only need to specify the unequal weights using the *weights* argument to calculate the estimates. Class 14 appears twice in the data since it was selected twice for the sample—we call it class 141 for the first appearance and class 142 for the second to distinguish them.

The mean calculated from *svymean* estimates $\bar{t}_U$, the population mean of the cluster totals t_i, which for this example is the total amount of time spent studying by students in class i. The average amount of time spent studying per student is estimated by the ratio $\hat{\bar{y}}_\psi = \hat{\bar{t}}_\psi / \hat{\bar{M}}_\psi$. The *svyratio* function can give the estimate $\hat{\bar{y}}_\psi$. (If the data set consists of the individual values y_{ij} instead of the summary statistics, then the mean $\hat{\bar{y}}_\psi$ can be estimated directly from *svymean*, as seen in the code for Example 6.6 of SDA.)

```
studystat <- data.frame(class = c(12, 141, 142, 5, 1),
                        Mi = c(24, 100, 100, 76, 44),
                        tothours=c(75,203,203,191,168))
studystat$wt<-647/(studystat$Mi*5)
sum(studystat$wt) # check weight sum, which estimates N=15 psus
## [1] 12.62321
# design for with-replacement sample, no fpc argument
d0604 <- svydesign(id = ~1, weights=~wt, data = studystat)
d0604
## Independent Sampling design (with replacement)
## svydesign(id = ~1, weights = ~wt, data = studystat)
# Ratio estimation using Mi as auxiliary variable
ratio0604<-svyratio(~tothours, ~Mi,design = d0604)
ratio0604
## Ratio estimator: svyratio.survey.design2(~tothours, ~Mi, design = d0604)
## Ratios=
##                Mi
## tothours 2.703268
## SEs=
##                 Mi
## tothours 0.3437741
confint(ratio0604, level=.95,df=4)
##                2.5 %   97.5 %
## tothours/Mi 1.748798 3.657738
# Can also estimate total hours studied for all students in population
svytotal(~tothours,d0604)
##          total     SE
## tothours  1749 222.42
```

The average amount of time a student spent studying statistics is estimated as 2.70 hours with an estimated standard error of 0.34 hours and a 95% confidence interval of [1.74, 3.66]. Note that 4 degrees of freedom (df; here, 1 less than the number of psus) are used for the confidence interval.

Example 6.6 of SDA. The estimates for a two-stage cluster sample with replacement are calculated exactly the same way as for a one-stage sample. For this example, we have data for the individual students in the psus so we enter those for each student.

Class 14 appears twice in the sample of psus in Example 6.4 of SDA. An independent set of students is selected for each appearance. To enable correct variance calculations, the first occurrence of class 14 is relabeled as class 141, and the second occurrence as class 142. These are counted as two separate psus in the estimation. If you labeled both as 14, then the *id* argument of *svydesign* would treat that as one psu with $m_i = 10$ instead of two psus of size 5.

The weight *studentwt* is calculated as the first-stage weight $M_0/(nM_i)$ times the second-stage weight M_i/m_i. The sample is self-weighting and the weight for each student simplifies to 647/25. For many problems, defining the weights is the trickiest part, and it is also the most important. Always check that the sum of the weights approximately (or exactly, in this case) equals the population size.

```
students <- data.frame(class = rep(studystat$class,each=5),
   popMi = rep(studystat$Mi,each=5),
   sampmi=rep(5,25),
   hours=c(2,3,2.5,3,1.5,2.5,2,3,0,0.5,3,0.5,1.5,2,3,1,2.5,3,5,2.5,4,4.5,3,2,5))
# The 'with' function allows us to calculate using variables from a data frame
# without having to type the data frame name for all of them
students$studentwt <- with(students,(647/(popMi*5)) * (popMi/sampmi))
# check the sum of the weights
sum(students$studentwt)
## [1] 647
# create the design object
d0606 <- svydesign(id = ~class, weights=~studentwt, data = students)
d0606
## 1 - level Cluster Sampling design (with replacement)
## With (5) clusters.
## svydesign(id = ~class, weights = ~studentwt, data = students)
# estimate mean and SE
svymean(~hours,d0606)
##       mean     SE
## hours  2.5 0.3606
degf(d0606)
## [1] 4
confint(svymean(~hours,d0606),level=.95,df=4) #use t-approximation
##          2.5 %   97.5 %
## hours 1.498938 3.501062
# estimate total and SE
svytotal(~hours,d0606)
##        total     SE
## hours 1617.5 233.28
confint(svytotal(~hours,d0606),level=.95,df=4)
##          2.5 %   97.5 %
## hours 969.8132 2265.187
```

In the *svydesign* function, we supply the weights (which, for this example, are the same for all students) but no *fpc* argument. We specify the psu membership in the *id* argument. This means that the variability is calculated at the first stage level using the pps with-replacement formulas, that is, the variability among $\hat{t}_i/\psi_i$. Note that 4 df (1 less than the number of psus) are used for the confidence interval.

6.3.2 Estimates from without-Replacement Samples

Even when an unequal-probability sample was selected without replacement, the with-replacement variance is commonly calculated for simplicity and stability. Use the *weights* argument to provide the sampling weights at the observation-unit level, and use the `id=~psuid` argument to provide the information on psu membership (recall that only the psu membership is needed to calculate the with-replacement variance).

Example 6.11 of SDA. This example analyzes the without-replacement unequal-probability sample the same way as for the sample in Example 6.6. Even though the sample was selected

without replacement, the with-replacement variance provides a good approximation.

```
data(classpps)
nrow(classpps)
## [1] 20
head(classpps)
##   class class_size finalweight hours
## 1     4         22       32.35   5.0
## 2     4         22       32.35   4.5
## 3     4         22       32.35   5.5
## 4     4         22       32.35   5.0
## 5    10         34       32.35   2.0
## 6    10         34       32.35   4.0
d0611 <- svydesign(ids = ~class, weights=~classpps$finalweight, data = classpps)
d0611
## 1 - level Cluster Sampling design (with replacement)
## With (5) clusters.
## svydesign(ids = ~class, weights = ~classpps$finalweight, data = classpps)
# estimate mean and SE
svymean(~hours,d0611)
##       mean     SE
## hours 3.45 0.4819
confint(svymean(~hours,d0611),level=.95,df=4) #use t-approximation
##          2.5 %   97.5 %
## hours 2.112147 4.787853
# estimate total and SE
svytotal(~hours,d0611)
##        total     SE
## hours 2232.2 311.76
confint(svytotal(~hours,d0611),level=.95,df=4)
##          2.5 %   97.5 %
## hours 1366.559 3097.741
```

Calculating the without-replacement variance for a one-stage sample. In general, we recommend calculating the with-replacement variance estimate and omitting the *fpc* argument from *svydesign* when unequal-probability sampling is used. Most of the replication methods for calculating variances in Chapter 9 also calculate with-replacement variances. You can skip the remainder of this section if the with-replacement variances work for your applications.

Functions in the `survey` package will calculate the without-replacement variance for some one-stage designs if you specify the inclusion probabilities in the *fpc* argument. (As of this writing, the package will not yet calculate without-replacement variances for two-stage designs—the situation where unequal-probability sampling is most commonly used.) The formulas for calculating the Horvitz-Thompson (HT), Sen-Yates-Grundy (SYG), and other without-replacement variance estimates require knowledge of the joint inclusion probabilities, so you must also supply those to the *svydesign* function.

Let's calculate some joint inclusion probabilities first and then use them in the *svydesign* function. Functions in the `sampling` package will calculate joint inclusion probabilities for some of the sample-selection methods; for example, function *UPsampfordpi2* will calculate the joint inclusion probabilities for a sample selected using Sampford's method. For some other sample-selection methods, however, the joint inclusion probabilities must be calculated directly.

Example 6.8 of SDA. We use the supermarket example to illustrate the calculation of joint inclusion probabilities when the sample size is 2. We first create a data frame of the supermarket population containing the store identifier, area, and revenue.

```
supermarket<-data.frame(store=c('A','B','C','D'),area=c(100,200,300,1000),
                        ti=c(11,20,24,245))
supermarket
##   store area  ti
## 1     A  100  11
## 2     B  200  20
## 3     C  300  24
## 4     D 1000 245
```

The draw-by-draw method was used to select two supermarkets for the sample, where the selection probability for draw 1 was proportional to store area. We can use that information to calculate π_i, the probability of store i being included in the sample, and π_{ik}, the joint probability that stores i and k are both included in the sample.

Here, we use matrix operations to calculate the probabilities by applying the formulas in Example 6.8 of SDA, noting that if $\mathbf{a}$ and $\mathbf{b}$ are two vectors, the (i, j) entry of $\mathbf{ab}^T$ is $a_i b_j$. The *apply* function sums the entries in each column.

```
supermarket$psi<-supermarket$area/sum(supermarket$area)
psii<-supermarket$area/sum(supermarket$area)
piik<- psii %*% t(psii/(1-psii)) + (psii/(1-psii)) %*% t(psii)
diag(piik)<-rep(0,4) # set the diagonal entries of the matrix equal to zero
piik  # joint inclusion probabilities
##            [,1]       [,2]       [,3]      [,4]
## [1,] 0.00000000 0.01726190 0.02692308 0.1458333
## [2,] 0.01726190 0.00000000 0.05563187 0.2976190
## [3,] 0.02692308 0.05563187 0.00000000 0.4567308
## [4,] 0.14583333 0.29761905 0.45673077 0.0000000
pii<-apply(piik,2,sum)
pii # inclusion probabilities
## [1] 0.1900183 0.3705128 0.5392857 0.9001832
```

The results show that $\pi_1 = 0.19$, $\pi_2 = 0.37$, $\pi_3 = 0.539$, and $\pi_4 = 0.90$. The joint inclusion probabilities are given in *piik*: for example, $\pi_{12} = 0.01726$. These are the numbers shown in Table 6.6 of SDA.

Now let's use the values of π_i and π_{ik} to calculate the HT and SYG variance estimates. Of course, since the supermarket sample has only two units, neither estimate will be very accurate, but it will serve to illustrate the methods.

Example 6.9 of SDA. Suppose supermarkets C and D were selected from the population in Example 6.8 of SDA. We will calculate the Horvitz–Thompson (HT) estimate for total revenue and the without-replacement HT and SYG variance estimates.

As always, we specify all the information about the design in the *svydesign* function. We tell the function that this is an unequal-probability sample without replacement through the *fpc* argument. Instead of giving the population sizes in the *fpc* argument, however, for pps sampling without replacement we specify `fpc=~pii`, the inclusion probability for each sampled unit.

We also use two other arguments to the *svydesign* function that we have not seen before. The *variance* argument tells whether to calculate the HT or SYG (the function calls this "YG") formula for the variance. We supply the joint inclusion probabilities in the *pps* argument,

after first placing π_i on the diagonal elements of the joint probabilities matrix and using the function *ppsmat* to get the joint probabilities in the form required by *svydesign*.

```
supermarket2<-supermarket[3:4,]
supermarket2$pii <- pii[3:4] # these are the unit inclusion probs when n=2
jointprob<-piik[3:4,3:4]  # joint probability matrix for stores C and D
diag(jointprob)<-supermarket2$pii # set diagonal entries equal to pii
jointprob
##           [,1]      [,2]
## [1,] 0.5392857 0.4567308
## [2,] 0.4567308 0.9001832
# Horvitz-Thompson type
dht<- svydesign(id=~1,  fpc=~pii, data=supermarket2,
                  pps=ppsmat(jointprob),variance="HT")
dht
## Sparse-matrix design object:
##  svydesign(id = ~1, fpc = ~pii, data = supermarket2, pps = ppsmat(jointprob),
##     variance = "HT")
svytotal(~ti,dht)
##     total     SE
## ti 316.67 82.358
# Sen-Yates-Grundy type
dsyg<- svydesign(id=~1,  fpc=~pii, data=supermarket2,
                  pps=ppsmat(jointprob),variance="YG")
dsyg
## Sparse-matrix design object:
##  svydesign(id = ~1, fpc = ~pii, data = supermarket2, pps = ppsmat(jointprob),
##     variance = "YG")
svytotal(~ti,dsyg)
##     total     SE
## ti 316.67 57.094
```

We can compare the variance estimates from the two methods with the true without-replacement variance $V(\hat{t}_{\rm HT}) = 4383.6$ in SDA (which is known for this small example where the full population is known), with $\hat{V}_{\rm HT}(\hat{t}_{\rm HT}) = (82.358)^2 = 6782.8$, and $\hat{V}_{\rm SYG}(\hat{t}_{\rm HT}) = (57.094)^2 = 3259.8$. In most situations, the SYG variance estimate is preferred because it is more stable.

The *svydesign* function also provides some approximation methods to calculate without-replacement variance estimates for one-stage samples. Option `pps=HR(sum(piisq)/n)`, where *piisq* is the vector of squared inclusion probabilities and n is the number of psus selected, gives the Hartley and Rao (1962; see Exercise 6.36 in SDA) approximation to the variance.

Example 6.10 of SDA. Let's do one more example, to compare the with-replacement, HT, and SYG variance estimates calculated for the unequal-probability sample in data *agpps*, as well as the Hartley–Rao approximation. We would expect the with-replacement variance estimate to work well here because $n = 15$ is small relative to $N = 3078$.

```
data(agpps)
jtprobag<-as.matrix(agpps[,20:34])
diag(jtprobag)<-agpps$SelectionProb
# Horvitz-Thompson type
dhtag<- svydesign(id=~1,  fpc=~SelectionProb, data=agpps,
                  pps=ppsmat(jtprobag),variance="HT")
svytotal(~acres92,dhtag)
```

```
##             total       SE
## acres92 936291172 70466858
# Sen-Yates-Grundy type
dsygag<- svydesign(id=~1,  fpc=~SelectionProb, data=agpps,
                   pps=ppsmat(jtprobag),variance="YG")
svytotal(~acres92,dsygag)
##             total       SE
## acres92 936291172 11715201
# Hartley-Rao approximation
sumsqprob<-sum(agpps$SelectionProb^2)/nrow(agpps)
dHRag<-svydesign(id=~1, fpc=~SelectionProb, data=agpps,
                   pps=HR(sumsqprob),variance="YG")
svytotal(~acres92,dHRag)
##             total       SE
## acres92 936291172 12148234
# With-replacement variance
dwrag<-svydesign(id=~1, weights=~SamplingWeight, data=agpps)
svytotal(~acres92,dwrag)
##             total       SE
## acres92 936291172 12293009
```

Note that the with-replacement (12,293,009), SYG (11,715,201), and Hartley–Rao (12,148,234) standard errors are all similar to each other. The HT standard error (70,466,858) is larger and often less stable; in general, we recommend one of the other methods.

6.4 Summary, Tips, and Warnings

Several functions in the `sampling` package will select equal- and unequal-probability cluster samples; some of these are listed in Table 6.1. The *sample* function can be used to select with-replacement unequal-probability samples.

Table 6.2 lists the major R functions used in this chapter.

Tips and Warnings

- When selecting an unequal-probability sample, check the calculation of the selection probabilities to make sure these are roughly proportional to the unit sizes.
- The more complex the sampling plan, the more complicated the weight calculations. Check that the sum of the weights approximately equals the population size.
- For unequal-probability sampling, omitting the *fpc* argument in the *svydesign* function gives the with-replacement variance. In general, this is the approach that we recommend. If the without-replacement variance is desired, use the Sen-Yates-Grundy formula directly.

TABLE 6.2
Functions used for Chapter 6.

Function	Package	Usage
sample	base	Select a with-replacement sample with unequal probabilities
confint	stats	Calculate confidence intervals, add df for t confidence interval
apply	base	Apply a function to the rows or columns of a matrix
cluster	sampling	Select a cluster sample
strata	sampling	Select a stratified random sample (here used to select ssus from the sampled psus)
mstage	sampling	Select a multi-stage cluster sample
UPsampford	sampling	Select an unequal-probability sample of units using Sampford's method
inclusionprobabilities	sampling	Convert a vector of positive size measures to selection probabilities, for use in the UP selection functions
getdata	sampling	Extract the data after selecting a sample
svydesign	survey	Specify the survey design
svymean	survey	Calculate mean and standard error of mean
svyratio	survey	Calculate ratio and standard error of ratio
svytotal	survey	Calculate total and standard error of total