7

Complex Surveys

We have already seen most of the components needed for selecting and computing estimates from a stratified multistage sample. Now let's put them all together. Section 7.1 shows how to select a stratified two-stage sample, and Section 7.3 shows how to compute estimates using examples from the National Health and Nutrition Examination Survey (NHANES).

The new features considered in this chapter are how to estimate quantiles (Section 7.2) and how to graph survey data (Sections 7.4 and 7.5). The code is in file `ch07.R` on the book website.

7.1 Selecting a Stratified Two-Stage Sample

In the following example, we select a sample of classes from the small population considered in Section 6.2 of SDA, after first dividing the classes into three strata based on their sizes. Stratum 1 contains the two large classes, stratum 2 contains six medium-sized classes, and stratum 3 contains the seven smallest classes. The code specifies drawing two primary sampling units (psus) without replacement (srswor) from each stratum, and drawing three students without replacement from each selected class. Of course other allocations of psus to strata can be used, as described in Chapter 3. The psus are arranged in strata, so the only new feature here is to add the stratification information in the function *mstage* from the `sampling` package (Tillé and Matei, 2021).

We use the *classeslong* data frame that we created in Section 1.5, where variable *class* gives the psu number and variable *studentid* gives the student identifier (numbered from 1 to *class_ size*) within each class.

```
# data(classeslong)
# define strata
classeslong$strat<-rep(3,nrow(classeslong))
classeslong$strat[classeslong$class_size > 40]<-2
classeslong$strat[classeslong$class_size > 70]<-1
# table(classeslong$class,classeslong$strat)
# order data by stratum
classeslong2<-classeslong[order(classeslong$strat),]
# check the stratum construction
table(classeslong2$strat,classeslong2$class_size)
##
##      15  20  22  24  26  33  34  44  46  54  63  76 100
##   1   0   0   0   0   0   0   0   0   0   0   0  76 100
##   2   0   0   0   0   0   0   0  88  92  54  63   0   0
##   3  15  20  22  24  26  33  34   0   0   0   0   0   0
nrow(classeslong2) # number of students in population
## [1] 647
```

DOI: 10.1201/9781003228196-7

To select a stratified cluster sample using *mstage*, we first define *numberselect* consisting of information on stratum sizes, number of psus selected from each stratum, and number of secondary sampling units (ssus; here, students) selected within each sampled psu as follows:

```
numberselect<-list(table(classeslong2$strat),rep(2,3),rep(3,6))
```

Next, use the *mstage* function to select a stratified two-stage cluster sample.

```
mstage(classeslong2,stage=list("stratified","cluster","stratified"),
               varnames=list("strat","class","studentid"),
               size=numberselect, method=list("","srswor","srswor"))
```

The *stage* argument for this example is a list with three components. The first component, `"stratified"`, defines the stratification for the psus, but nothing is selected at this stage (the first component of *method* is blank). Then, an SRS of psus (variable *class*) is selected within each stratum (this is described by the second components of *stage*, *varnames*, and *method*). At this point, the sample consists of the selected psus from the 3 strata. We then use `"stratified"` again to select an SRS of students from each sampled psu.

The result from *mstage* is a list with three components corresponding to the components of *stage* in the function argument. The third component is the final sample, saved in *sample3*. The variable *Prob*, computed by function *mstage*, is the final selection probability, and we calculate *finalweight* as its reciprocal.

```
# select a stratified two stage cluster sample,
# stratum: strat
# psu: class
# ssu: studentid
# number of psus selected n =2, size=rep(n=2,3 strata) (srswor)
# number of students selected m_i =3 size=rep(m_i= 3,6 classes) (srswor)
numberselect<-list(table(classeslong2$strat),rep(2,3),rep(3,6))
numberselect
## [[1]]
##
##   1   2   3
## 176 297 174
##
## [[2]]
## [1] 2 2 2
##
## [[3]]
## [1] 3 3 3 3 3 3
# select a stratified two-stage cluster sample
set.seed(75745)
tempid<-mstage(classeslong2,stage=list("stratified","cluster","stratified"),
               varnames=list("strat","class","studentid"),
               size=numberselect, method=list("","srswor","srswor"))
# get data
sample3<-getdata(classeslong2,tempid)[[3]]  #3rd stage
sample3$finalweight<-1/sample3$Prob
# check sum of weights, should be close to number of students in population
# (but not exactly equal, since psus not selected with prob proportional to M_i)
sum(sample3$finalweight)
## [1] 624
sample3 # print the sample
##       class class_size studentid strat ID_unit Prob_ 3 _stage       Prob
## 5.31      5         76        32     1      32     0.03947368 0.03947368
```

```
## 5.42      5          76         43      1       43     0.03947368 0.03947368
## 5.61      5          76         62      1       62     0.03947368 0.03947368
## 14.9     14         100         10      1       86     0.03000000 0.03000000
## 14.37    14         100         38      1      114     0.03000000 0.03000000
## 14.79    14         100         80      1      156     0.03000000 0.03000000
## 8.34      8          44         35      2      318     0.06818182 0.02272727
## 8.39      8          44         40      2      323     0.06818182 0.02272727
## 8.28      8          44         29      2      312     0.06818182 0.02272727
## 9.26      9          54         27      2      354     0.05555556 0.01851852
## 9.28      9          54         29      2      356     0.05555556 0.01851852
## 9.38      9          54         39      2      366     0.05555556 0.01851852
## 7.17      7          20         18      3      572     0.15000000 0.04285714
## 7.5       7          20          6      3      560     0.15000000 0.04285714
## 7.10      7          20         11      3      565     0.15000000 0.04285714
## 12.6     12          24          7      3      615     0.12500000 0.03571429
## 12.9     12          24         10      3      618     0.12500000 0.03571429
## 12.16    12          24         17      3      625     0.12500000 0.03571429
##       finalweight
## 5.31     25.33333
## 5.42     25.33333
## 5.61     25.33333
## 14.9     33.33333
## 14.37    33.33333
## 14.79    33.33333
## 8.34     44.00000
## 8.39     44.00000
## 8.28     44.00000
## 9.26     54.00000
## 9.28     54.00000
## 9.38     54.00000
## 7.17     23.33333
## 7.5      23.33333
## 7.10     23.33333
## 12.6     28.00000
## 12.9     28.00000
## 12.16    28.00000
```

You can also look at the selection probabilities for each stage if desired. The following extracts the other stages in *sample1* and *sample2*. The first stage defines the stratification, so `Prob_ 1 _stage` is 1. Stratum 1 contains two psus, so the function selects each with certainty. Classes are selected from stratum 2 with probability of 2/6, and classes are selected from stratum 3 with probability of 2/7. These are reported as `Prob_ 2 _stage` of *sample2*. At the third stage, `Prob_ 3 _stage` is calculated as $3/class\_size$. The final probability *Prob* is calculated as `1*Prob_ 2 _stage*Prob_ 3 _stage`.

```
sample1<-getdata(classeslong2,tempid)[[1]]  #1st stage
sample2<-getdata(classeslong2,tempid)[[2]]  #2nd stage
names(sample1)
## [1] "class"          "class_size"     "studentid"      "strat"
## [5] "ID_unit"        "Prob_ 1 _stage" "Stratum"
table(sample1$`Prob_ 1 _stage`)
##
##   1
## 647
table(sample2$strat,sample2$`Prob_ 2 _stage`) # Selection probs for psus in strata
```

```
##
##     0.285714285714286 0.333333333333333   1
##   1                 0                 0 176
##   2                 0                98   0
##   3                44                 0   0
table(sample3$class,sample3$`Prob_ 3 _stage`) # Selection probs for ssus in psus
##
##      0.03 0.0394736842105263 0.0555555555555556 0.0681818181818182 0.125 0.15
##   5     0                  3                  0                  0     0    0
##   7     0                  0                  0                  0     0    3
##   8     0                  0                  0                  3     0    0
##   9     0                  0                  3                  0     0    0
##   12    0                  0                  0                  0     3    0
##   14    3                  0                  0                  0     0    0
```

For more complicated designs, you may want to select the sample at each stage separately, as illustrated in Section 6.2. For example, you can use function *UPsampford* to select a sample of psus from each stratum, then select the sample at the subsequent stages.

7.2 Estimating Quantiles

Quantiles are estimated using the empirical cumulative distribution function (cdf) $\hat{F}(y)$, which is the sum of the weights for the sample observations having $y_i \leq y$ divided by the sum of all of the weights. Because $\hat{F}(y)$ has jumps at the distinct values of y in the sample, however, for many values of q there is no value of y in the sample that has $\hat{F}(y)$ exactly equal to q. Multiple definitions for population and sample quantiles have been proposed (Hyndman and Fan, 1996; Wang, 2021).

The *svyquantile* function in the `survey` package (Lumley, 2020) calculates several estimates of quantiles and their standard errors. With `ties="discrete"`, the empirical cdf $\hat{F}$ is used directly, with jumps at the values of y in the sample. With `ties="rounded"` an interpolated cdf is used (see Exercise 7.19 of SDA). We usually prefer interpolated quantiles, as they smooth out an empirical cdf that has large jumps.

The quantiles $\tilde{\theta}_q$ are calculated by requesting the desired quantile values in the *svyquantile* function. Request the 0.25, 0.5, 0.75, and 0.90 quantiles, for example, by typing `quantiles =c(0.25, 0.50, 0.75, 0.90)`.

Example 7.5 of SDA. The following code requests quantiles and CIs for the height values in the SRS *htsrs*. Of course, for a complex design, one would include stratification and clustering information in the *svydesign* function.

```
data(htsrs)
dhtsrs<-svydesign(id = ~1,weights=rep(2000/200,200),fpc=rep(2000,200), data=htsrs)
# cdf treated as step function, gives values in Table 7.1 of SDA
svyquantile(~height, dhtsrs, quantiles=c(0.25,0.5,0.75,0.9), ties = "discrete")
##        0.25 0.5 0.75 0.9
## height  160 169  176 184
# interpolated quantiles (usually preferred method)
svyquantile(~height, dhtsrs, quantiles=c(0.25,0.5,0.75,0.9), ties = "rounded")
##         0.25    0.5 0.75   0.9
## height 159.7 168.75  176 183.4
```

The *svyquantile* function will also calculate confidence intervals for quantiles if you request `ci=TRUE`. The default method is `interval.type="Wald"`, which calculates the Woodruff (1952) interval presented in Section 9.5 of SDA.

Examples 7.6 and 9.12 of SDA. Here we calculate confidence intervals for the interpolated quantiles in the *htstrat* data.

```
data(htstrat)
popsize_recode <- c('F' = 1000, 'M' = 1000)
# create a new variable popsize for population size
htstrat$popsize<-popsize_recode[htstrat$gender]
head(as.data.frame(htstrat))
##    rn height gender popsize
## 1 201    166      F    1000
## 2 965    163      F    1000
## 3 490    166      F    1000
## 4 249    155      F    1000
## 5 260    154      F    1000
## 6 324    160      F    1000
# design object
# svydesign calculates the weights here from the fpc argument
dhtstrat<-svydesign(id = ~1, strata = ~gender, fpc = ~popsize,
          data = htstrat)
# ties = "discrete" gives values in Table 7.1 of SDA
svyquantile(~height, dhtstrat, c(0.25,0.5,0.75,0.9), ties = "discrete")
##        0.25 0.5 0.75 0.9
## height  161 168  177 182
# ties = "rounded" gives values in Example 9.12 of SDA
svyquantile(~height, dhtstrat, c(0.25,0.5,0.75,0.9), ties = "rounded",
            ci=TRUE, interval.type = "Wald")
## $quantiles
##            0.25      0.5    0.75   0.9
## height 160.7143 167.5556 176.625 181.5
##
## $CIs
## , , height
##
##            0.25      0.5     0.75      0.9
## (lower 159.3556 165.8078 173.3572 178.7176
## upper) 162.0247 170.0942 178.5439 190.1679
```

7.3 Computing Estimates from Stratified Multistage Samples

We have seen all the building blocks for computing the estimates from any survey. Now let's put them all together using the data from the National Health and Nutrition Examination Survey (NHANES, Centers for Disease Control and Prevention, 2017). The Centers for Disease Control and Prevention produce online tutorials for analyzing NHANES data. These, and sample code and tips for analyzing NHANES data using R, can be found at `https://wwwn.cdc.gov/nchs/nhanes/tutorials/`.

Example 7.9 of SDA. In this example, we look at statistics about body mass index (BMI, variable *bmxbmi*) for adults age 20 and over. We will compute these estimates using *svymean* and *svyquantile*.

One of the statistics to be calculated is the proportion of adults having BMI greater than 30, so we define categorical variable *bmi30* to equal 1 if the person's BMI is greater than 30 and 0 if it is less than or equal to 30. Note that the variable *bmxbmi* has missing values, so we set *bmi30* to be missing if *bmxbmi* is missing. In the data file `nhanes.csv`, missing values are coded by −9. We coded all the missing values −9 as "NA" in the R data set *nhanes* in the `SDAResources` package.

We also need to define a variable giving the domain of interest to be analyzed: adults age 20 and over who have data for *bmxbmi*. We define *age20d*=1 if *ridageyr*≥ 20 and *bmxbmi* is not missing, and 0 otherwise. This excludes the observations with missing values from the domain of interest, and ensures that standard errors for the domain are calculated correctly.

```
data(nhanes)
nrow(nhanes) #9971
## [1] 9971
names(nhanes)
##  [1] "sdmvstra" "sdmvpsu"  "wtint2yr" "wtmec2yr" "ridstatr" "ridageyr"
##  [7] "ridagemn" "riagendr" "ridreth3" "dmdeduc2" "dmdfmsiz" "indfmpir"
## [13] "bmxwt"    "bmxht"    "bmxbmi"   "bmxwaist" "bmxleg"   "bmxarml"
## [19] "bmxarmc"  "bmdavsad" "lbxtc"    "bpxpls"   "sbp"      "dbp"
## [25] "bpread"
# count number of observations with missing value for ridageyr, bmxbmi
sum(is.na(nhanes$ridageyr)) # ridageyr gives age in years
## [1] 0
sum(is.na(nhanes$bmxbmi))   # bmxbmi gives BMI
## [1] 1215
# define age20d and bmi30
nhanes$age20d<-rep(0,nrow(nhanes))
nhanes$age20d[nhanes$ridageyr >=20 & !is.na(nhanes$bmxbmi)]<-1
nhanes$bmi30<-nhanes$bmxbmi
nhanes$bmi30[nhanes$bmxbmi>30]<-1
nhanes$bmi30[nhanes$bmxbmi<=30]<-0
nhanes$bmi30<-factor(nhanes$bmi30) # set bmi30 as a categorical variable
# check missing value counts for new variables
sum(is.na(nhanes$age20d))
## [1] 0
sum(is.na(nhanes$bmi30))
## [1] 1215
sum(nhanes$age20d) # how many records in domain?
## [1] 5406
head(nhanes)
##   sdmvstra sdmvpsu  wtint2yr  wtmec2yr ridstatr ridageyr ridagemn riagendr
## 1      125       1 134671.37 135629.51        2       62       NA        1
## 2      125       1  24328.56  25282.43        2       53       NA        1
## 3      131       1  12400.01  12575.84        2       78       NA        1
## 4      131       1 102718.00 102078.63        2       56       NA        2
## 5      126       2  17627.67  18234.74        2       42       NA        2
## 6      128       1  11252.31  10878.68        2       72       NA        2
##   ridreth3 dmdeduc2 dmdfmsiz indfmpir bmxwt bmxht bmxbmi bmxwaist bmxleg
## 1        3        5        2     4.39  94.8 184.5   27.8    101.1   43.3
## 2        3        3        1     1.32  90.4 171.4   30.8    107.9   38.0
```

```
## 3        3        3        2     1.51  83.4 170.1   28.8    116.5   35.6
## 4        3        5        1     5.00 109.8 160.9   42.4    110.1   38.5
## 5        4        4        5     1.23  55.2 164.9   20.3     80.4   37.4
## 6        1        2        5     2.82  64.4 150.0   28.6     92.9   34.4
##   bmxarml bmxarmc bmdavsad lbxtc bpxpls sbp dbp bpread age20d bmi30
## 1    43.6    35.9     22.8   173     76 120  63      2      1     0
## 2    40.0    33.2     27.3   265     72 137  85      2      1     1
## 3    37.0    31.0     26.6   229     56 134  45      2      1     0
## 4    37.7    38.3     25.1   174     78 135  69      2      1     1
## 5    36.0    27.2       NA   204     76 106  55      2      1     0
## 6    33.5    31.4     23.1   190     64 121  59      2      1     0
```

Next, we use *svydesign* to describe the design information and use *subset* to select the domain of adults with data for BMI for analysis.

```
# stratified cluster design
d0709 <- svydesign(id = ~sdmvpsu, strata=~sdmvstra, weights=~wtmec2yr,
                   nest=TRUE, data = nhanes)
# domain estimation, age20+
d0709sub<-subset(d0709, age20d ==1)
d0709sub
## Stratified 1 - level Cluster Sampling design (with replacement)
## With (30) clusters.
## subset(d0709, age20d == 1)
```

Let's look at the features used in the *svydesign* function to describe the design.

- The *weights* argument in *svydesign* specifies the variable containing the final weights. The data set contains two weight variables: *wtint2yr* gives the weight for the set of persons with interview data, and *wtmec2yr* gives the weight for the subset of interviewed persons who had a medical examination. BMI is measured in the medical examination, so the appropriate weight variable to use is *wtmec2yr*.
- The *strata* and *id* arguments are used exactly as in Chapters 3 and 5, except now we include both of them. The `strata=~sdmvstra` argument says that *sdmvstra* is the variable giving the stratum membership. The psus specified in `id=~sdmvpsu` are the first-stage sampling units.
- In the NHANES data, the two psus in each stratum are labeled as '1' and '2'. The `nest=TRUE` argument says that psu labels are nested within strata—that is, multiple strata have the same psu labels. Typing `nest=TRUE` ensures that psu 1 in stratum 1 is recognized as being a different psu than psu 1 in stratum 2.

  When there is one stratum, the results are the same if you have `nest=TRUE`, `nest=FALSE`, or simply omit the *nest* argument.
- No *fpc* argument is included in *svydesign*. With complex samples such as NHANES, we usually want to calculate the with-replacement variance, which requires only psu-level information.

The *subset* function to define domains has been discussed in Chapter 4. This specifies that estimates are desired for the domain of persons age 20 and older having data for BMI (with *age20d*=1), and carries the stratification and clustering information from the full design over for analyzing the subset. If you just created a subset of the data consisting of the observations having $ridageyr \geq 20$, in some instances (for example, when some psus have

no members of the domain), the standard errors would be incorrect; by using the *subset* function, the correct standard errors are calculated.

Note that we exclude adults with missing values of *bmxbmi* from the domain of interest with *age20d*=1. The estimates are computed from the adults who have data. If the domain contained missing values, we would need to include `na.rm=TRUE` in the *svymean* function to be able to calculate statistics.

Functions *svymean* and *svyquantile* are then applied to calculate the estimated mean and quantiles of BMI and the proportion in each category of *bmi30*. The confidence intervals for quantiles differ slightly from those in SDA, which were calculated using SAS software under a slightly different algorithm. Adding `deff=TRUE` to *svymean* requests the design effect (deff) for each statistic.

```
# Request means and design effects
nhmeans<-svymean(~bmxbmi+bmi30, d0709sub, deff=TRUE)
degf(d0709sub)
## [1] 15
nhmeans
##             mean        SE   DEff
## bmxbmi 29.389101  0.253197 7.1248
## bmi300  0.607775  0.015856 5.7003
## bmi301  0.392225  0.015856 5.7003
confint(nhmeans,df=degf(d0709sub))
##             2.5 %     97.5 %
## bmxbmi 28.8494243 29.9287768
## bmi300  0.5739798  0.6415707
## bmi301  0.3584293  0.4260202
# Find quantiles
svyquantile(~bmxbmi, d0709sub, quantiles=c(0.05,0.25,0.5,0.75,0.95),
            ties = "rounded",ci=TRUE, interval.type="Wald")
## $quantiles
##            0.05     0.25     0.5     0.75     0.95
## bmxbmi 20.29893 24.35349 28.2349 33.06615 42.64092
##
## $CIs
## , , bmxbmi
##
##            0.05     0.25      0.5     0.75     0.95
## (lower 19.83403 23.92667 27.55465 32.35400 41.91584
## upper) 20.70609 24.84391 28.91359 33.64129 43.47766
```

The mean BMI for adults age 20 and over is 29.389 (using design object *d0709sub*), with 95% confidence interval [28.849, 29.929]. The estimated proportion of adults age 20 and over who have BMI > 30 (*bmi30*=1) is 0.392225 with 95% confidence interval [0.3584293, 0.4260202]. The confidence interval is calculated using a t distribution with 15 degrees of freedom (number of psus minus number of strata).

7.4 Univariate Plots from Complex Surveys

The *svyhist* and *svyboxplot* functions, along with *svydesign*, produce histograms and boxplots that incorporate the weights, as described in Chapter 7 of SDA.

Examples 7.10, 7.11, and 7.12 of SDA. These examples consider data in *htstrat*, a disproportional stratified sample of 160 women and 40 men.

Histograms and smoothed density estimates. Figure 7.1 shows the difference between a histogram constructed without the weights (left panel) and one constructed with the weights (right panel). Each histogram is overlaid with a smoothed density estimate.

```
data(htstrat)
# set graphics parameters, 1*2 plots, axis labels horizontal
par(mfrow=c(1,2),las=1,mar=c(2.1,4.1,2.1,0.3))
# Histogram overlaid with kernel density curve (without weight information)
# Displays the sample values, but does not estimate population histogram
# freq=FALSE changes the vertical axis to density
# breaks tell how many breakpoints to use
hist(htstrat$height,main="Without weights", xlab = "Height (cm)",
     breaks = 10, col="gray90", freq=FALSE, xlim=c(140,200), ylim=c(0,0.045))
# overlaid with kernel density curve
lines(density(htstrat$height),lty=1,lwd=2)

# Histogram (with weight information)
# create survey design object, weights calculated from fpc here
d0710 <- svydesign(id = ~1, strata = ~gender, fpc = c(rep(1000,160),rep(1000,40)),
                data = htstrat)
d0710
## Stratified Independent Sampling design
## svydesign(id = ~1, strata = ~gender, fpc = c(rep(1000, 160),
##     rep(1000, 40)), data = htstrat)
svyhist(~height,d0710, main="With weights",xlab = "Height (cm)",
        breaks = 10, col="gray90", freq=FALSE,xlim=c(140,200), ylim=c(0,0.045))
dens1<-svysmooth(~height,d0710,bandwidth=5)
lines(dens1,lwd=2) # draw the density line
```

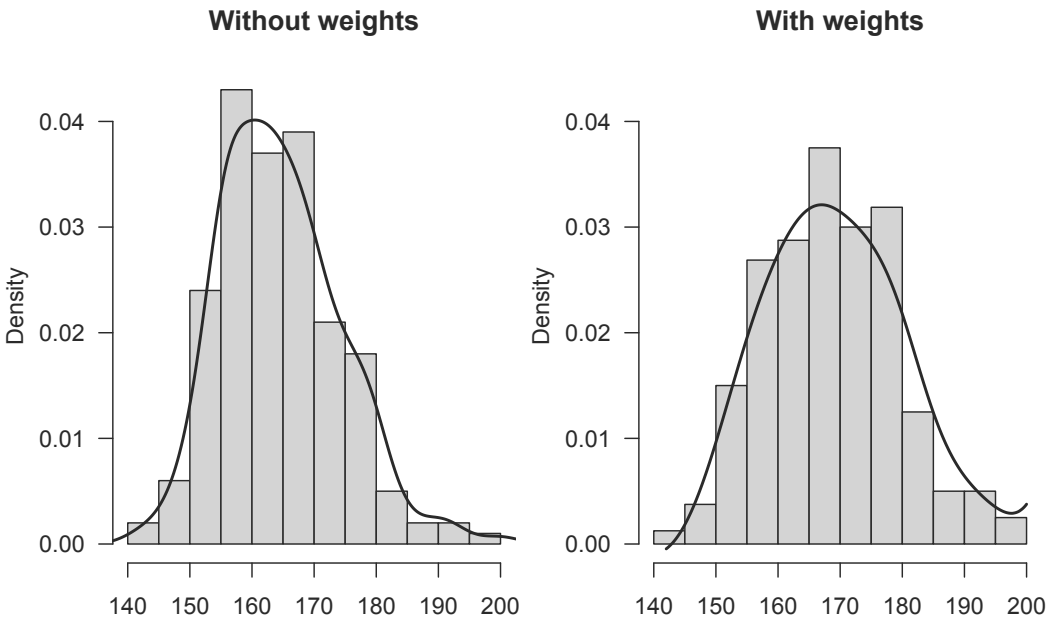

FIGURE 7.1: Histograms constructed without and with weights.

We call *svydesign* and *svyhist* to create a histogram that uses the survey information. The only difference between *svyhist* and *hist* is that *svyhist* includes the design object so that the histogram accounts for the weights.

```
svyhist(~height, design object, main=" ", xlab = " ", ylab= "")
```

In *hist* and *svyhist*, the *breaks* argument controls the number of bins, and `freq=FALSE` changes the vertical axis to density instead of frequency. The *density* function calculates a kernel density curve for unweighted data, and *svysmooth* performs density estimation using the weighted data. We allowed the bandwidth to be chosen automatically by the *density* function and specified a bandwidth of 5 in the *svysmooth* function (if you omit the *bandwidth* argument, it will be chosen automatically).

In Figure 7.1, the unweighted plot on the left displays the values in the sample but, because this is a disproportionally allocated sample, it does not estimate the histogram that would be obtained if we measured everyone in the population. The distribution appears skewed, reflecting the underrepresentation of men (who have greater average height) in the sample. The histogram and density estimate in the plot on the right incorporate the survey weights and thus can be interpreted as estimates of the histogram and density that would be obtained if the entire population were measured.

Boxplots. The *svyboxplot* function creates boxplots that include the weights; it is the survey analog of the *boxplot* function. Figure 7.2 displays boxplots for the full sample and separately for each gender.

```
par(mfrow=c(1,2),las=1,mar=c(2.1,4.1,2.1,0.3))
# boxplot (with weight information)
svyboxplot(height~1,d0710,ylab="Height",xlab=" ", main="Full sample")
svyboxplot(height~gender,d0710,ylab="Height",xlab="Gender",
           main="Separately by gender")
```

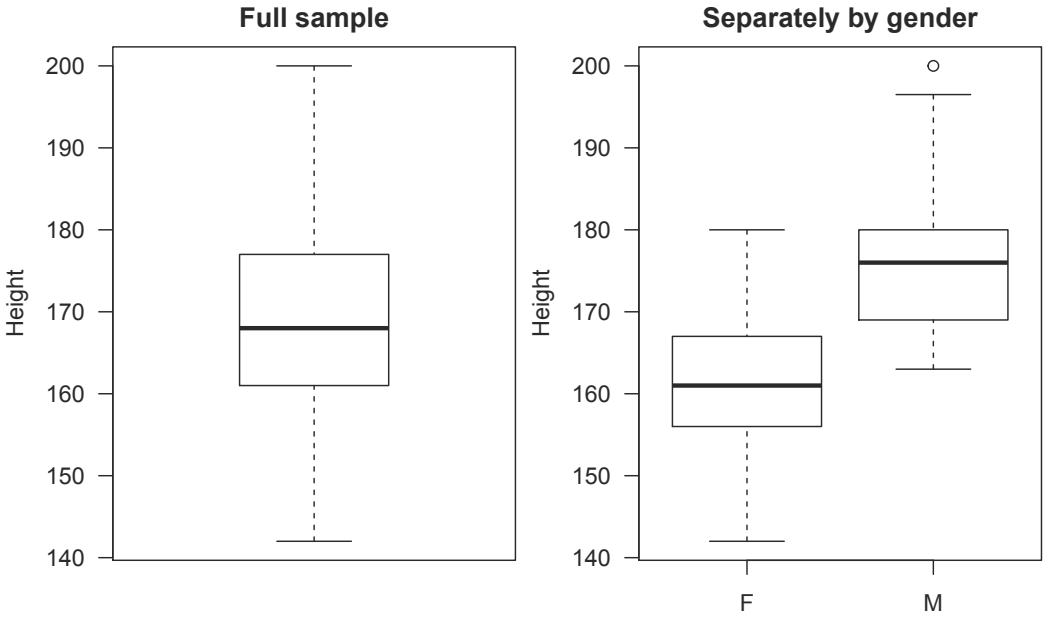

FIGURE 7.2: Boxplots of height for the full sample, and separately by gender.

Histograms for domains. What if you want to draw a histogram for just one domain from a complex survey? You can do that by using the *subset* function to redefine the object from *svydesign*. Figure 7.3 gives a histogram of BMI for adults age 20 and over, using the survey weights in the *nhanes* data. The smoothed density function is superimposed.

```
# Restore graphics settings
par(mfrow=c(1,1),las=1,mar=c(5.1, 4.1, 4.1, 2.1))
svyhist(~bmxbmi,d0709sub, main="Histogram of body mass index for adults",
        breaks = 30, col="gray90",xlab = "Body Mass Index (kg/m^2)")
dens2<-svysmooth(~bmxbmi,d0709sub)
lines(dens2,lwd=2)
```

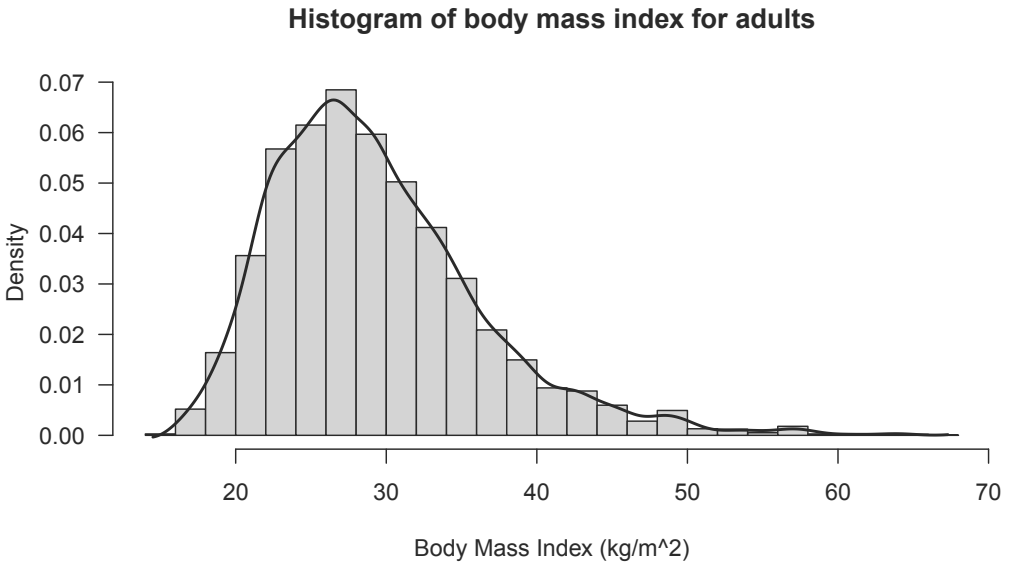

FIGURE 7.3: Histogram and density estimate for adults age 20 and over.

7.5 Scatterplots from Complex Surveys

Most of the plot types described in Section 7.6 of SDA can be drawn using functions *plot*, *svyplot*, *svyboxplot*, or *svycdf*.

Unweighted plots. The R functions *plot*, *hist*, and *boxplot* draw scatterplots, histograms, boxplots, and more. They do not incorporate the survey weights. In general, extra data preparation is needed to persuade these functions to draw graphs that estimate the population.

In some instances, however, you may want to examine the unweighted data. You may want to see how the weights affect the regression relationship between x and y, or to identify unusual observations in the data. For that reason, we include an unweighted scatterplot of the NHANES data.

The following shows the code to produce the unweighted scatterplot of y variable body mass index (*bmxbmi*) versus x variable age (*ridageyr*) shown in Figure 7.4. This plot is for all ages, not just the subset of adults for which we computed summary statistics in Section 7.3. Numerous options are available for customizing the plot; we chose to use the default setting `type="p"` for the points and used plotting symbol '+' (`pch=3`), making the symbols small so more of them are displayed on the plot (`cex=0.5`). The optional *xlab* and *ylab* arguments allow customizing the axis labels. You can also set the minimum and maximum values for each axis using *xlim* and *ylim*.

```
# scatterplot without weights
par(las=1) # make tick mark labels horizontal
plot(nhanes$ridageyr,nhanes$bmxbmi,xlab="Age (years)",ylab="Body Mass Index",
    main="Scatterplot without weights",pch=3,cex=0.5,
    ylim=c(10,70),xlim=c(0,80))
```

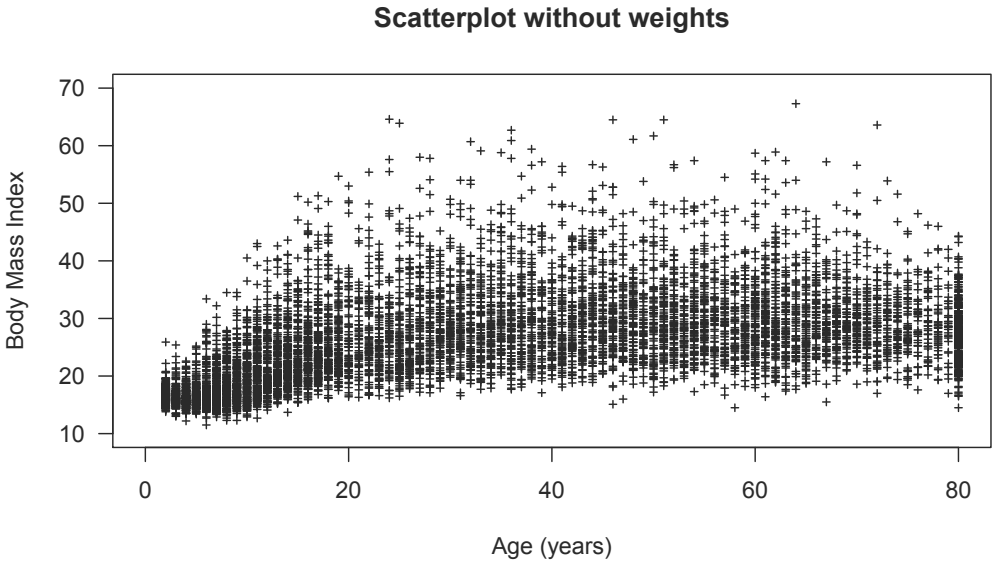

FIGURE 7.4: Scatterplot of BMI versus age (unweighted).

The unweighted scatterplot shows the relationship between x and y in the sample. If the sample is self-weighting, this scatterplot also estimates the relationship between x and y in the population. If the sample is not self-weighting, however, the relationship between x and y in the population may be different, and one of the other plots in this section should be used if you want to visualize the relationship between x and y in the population.

Although we do not recommend unweighted plots for non-self-weighting samples in general, sometimes, for small data sets, they are useful for seeing whether the relationship between x and y is the same with and without the weights (see Section 11.4 of SDA). The NHANES data, however, have so many data points that it is difficult to see patterns from the unweighted scatterplot in Figure 7.4.

Plot subsample of data. We can also select a subsample of the data that is approximately self-weighting, and draw the scatterplot of that subsample using the *plot* function. The plot of the subsample, displayed in Figure 7.5, then estimates the scatterplot that would be drawn from the population.

This subsample was selected with probability proportional to the weights and with replacement, using the *sample* function. Any method that will select a sample with probability proportional to the weights can be used, however, including the *UP* functions described in Table 6.1. If desired, you can draw multiple plots with different subsamples.

```
# select subsample with probability proportional to weights
set.seed(2847654)
subsamp<-sample(1:nrow(nhanes),500,replace=TRUE,prob=nhanes$wtmec2yr)
par(las=1) # make tick mark labels horizontal
plot(nhanes$ridageyr[subsamp],nhanes$bmxbmi[subsamp],
     xlab="Age (years)",ylab="Body Mass Index",
     main="Scatterplot of pps subsample",pch=3,cex=0.5,
     ylim=c(10,70),xlim=c(0,80))
```

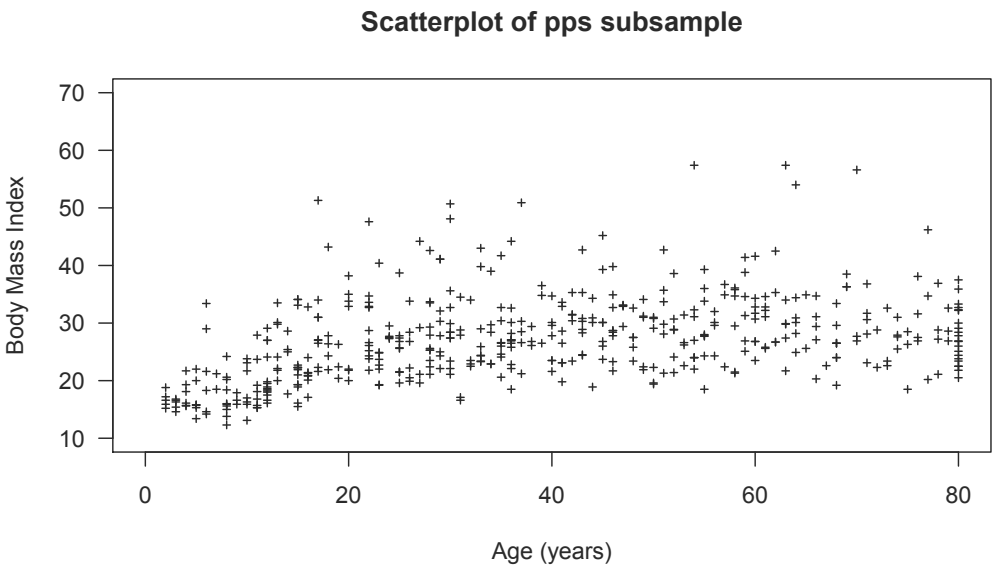

FIGURE 7.5: Scatterplot of BMI versus age (self-weighting subsample).

Bubble plots. If the sample is not self-weighting, a bubble plot displays the estimated shape of the population data. Bubble plots can be constructed using individual (x, y) values, or using bins. We will show how to use *svyplot* to create bubble plots of the individual (x, y) values.

The *svyplot* function is essentially the same as the *plot* function, except it adds the survey design object so that the weights are incorporated into the plots. You can use the same arguments to label the axes (*xlab, ylab*), set the extent of the plotting region (*xlim, ylim*), title the plot (*main*), and perform other formatting as in the *plot* function.

The *svyplot* function will construct a variety of types of plots that incorporate the weights, including bubble plots, "transparent" plots where the opacity of points is proportional to their weights, and binned hexagonal scatterplots.

Figure 7.6 shows a bubble plot for the NHANES data. The *inches* argument scales the bubbles; you may need to try several values until you find one that looks nice.

```
par(las=1) # make tick mark labels horizontal
svyplot(bmxbmi~ridageyr, design=d0709, style="bubble", inches=0.03,
        xlab="Age(years)",ylab="Body Mass Index",xlim=c(0,80),ylim=c(10,70),
        main="Weighted bubble plot of BMI versus age")
```

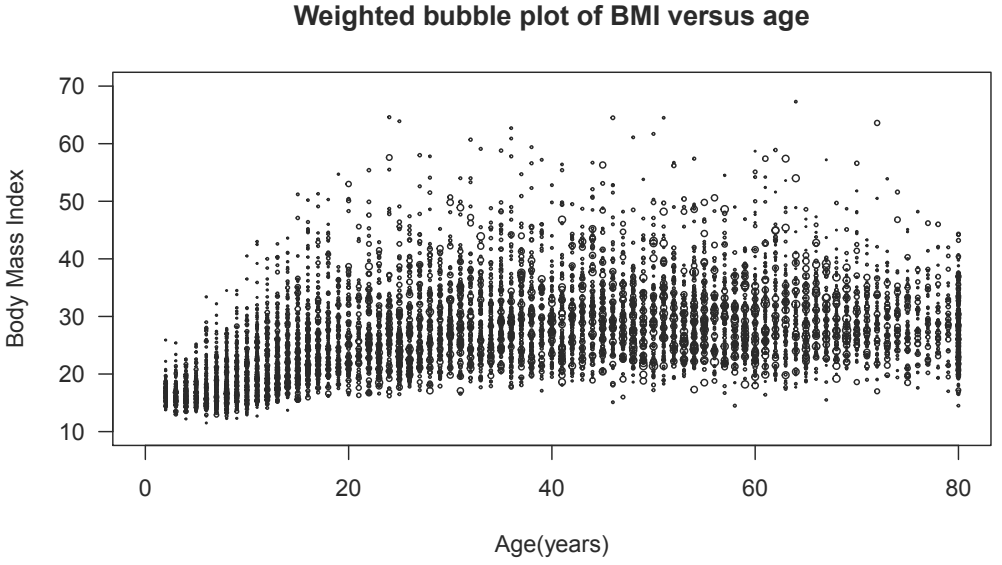

FIGURE 7.6: Bubble plot of BMI versus age (weighted).

Plot data for a domain. If we want to draw a plot for sample members in a specific domain, we can use the *subset* function to define a design object for the domain and then use *svyplot* with the subset design object.

Figure 7.7 shows a bubble plot of BMI versus age for the domain of non-Hispanic Asian Americans (having *ridreth3*=6).

```
# define subset
d0709subA<-subset(d0709, ridreth3==6)
par(las=1) # make tick mark labels horizontal
svyplot(bmxbmi~ridageyr, design=d0709subA, style="bubble",inches = 0.03,
        xlab="Age(years)",ylab="Body Mass Index",xlim=c(0,80),ylim=c(10,70),
        main="Weighted bubble plot of BMI versus age for Asian Americans")
```

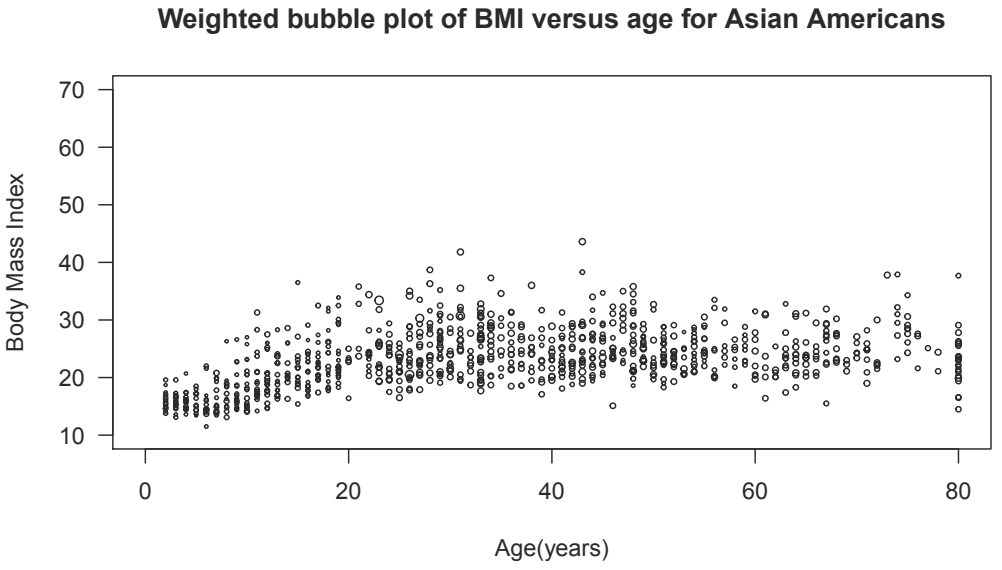

FIGURE 7.7: Bubble plot of BMI versus age for Asian Americans.

Note that for purposes of graphing data, it is also acceptable to consider just the subset of observations being graphed. This is because the scatterplots in this section incorporate the weights, but they do not use any other features of the survey design. Thus, if producing a subsampled graph for a domain that is similar to that in Figure 7.5, you could first create a subset of data consisting of the domain of interest, then select points with probabilities proportional to the weights, and then use the *plot* function to draw the scatterplot. With the *svyplot* function, however, it is easiest to create a subset design object for the domain being graphed.

Side-by-side boxplots. Boxplots will display the distributions of subgroups of the data. Figure 7.2 showed their use to display the distribution of height, separately for males and females.

They can also be used to display the bivariate relationship between two continuous variables. Simply partition the x variable into a categorical variable that defines different ranges of the variable. You can use the *round* function to round the x variable to the nearest multiple of a number. Alternatively, you can use the *cut* function to divide the range of x into intervals with user-supplied cutpoints and code each value x by the interval that contains it.

The easiest way to obtain side-by-side boxplots for survey data is through the *svyboxplot* function. Figure 7.8 shows boxplots of BMI by age groups that are formed by rounding the values of *ridageyr* to the nearest multiple of 5.

```
# include agegroup in the data frame
nhanes$agegroup<-5*round(nhanes$ridageyr/5)
d0709 <- svydesign(id = ~sdmvpsu, strata = ~ sdmvstra, nest=TRUE,
                   weights=~wtmec2yr, data = nhanes)
par(las=1) # make tick mark labels horizontal
svyboxplot(bmxbmi~factor(agegroup),d0709,ylab="Body mass index",xlab="Age Group",
           ylim=c(10,70),main="Side-by-side boxplots of BMI for age groups")
```

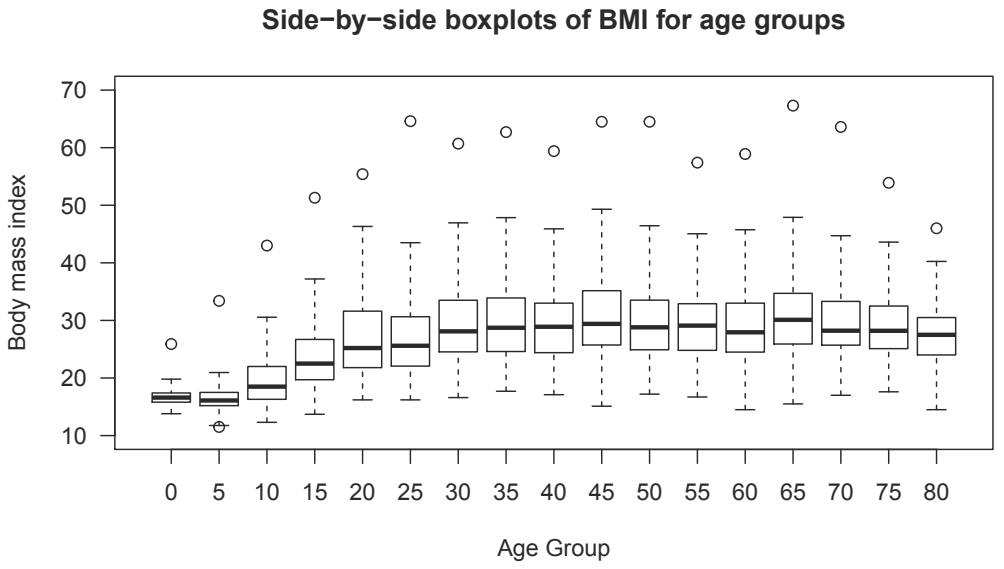

FIGURE 7.8: Boxplots of BMI by age groups.

Smoothed trend line for mean. The *svysmooth* function can carry out scatterplot smoothing and density estimation for weighted data. For the default method `"locpoly"`, the extra arguments are passed to *locpoly* from the `KernSmooth` package (Wand et al., 2020), which implements the smoothing methods described in Wand and Jones (1995). The default is local linear smoothing for the mean. If desired, the *bandwidth* argument can be included for a user-specified bandwidth.

Figure 7.9 shows the smoothed curve superimposed on the bubble plot in Figure 7.6. The color of the bubbles is changed to light gray for better visibility of the trend line.

```
# plot data bmxbmi~ridageyr
par(las=1) # make tick mark labels horizontal
svyplot(bmxbmi~ridageyr, design=d0709, style="bubble",basecol="gray",inches=0.03,
        xlab="Age(years)",ylab="Body Mass Index",xlim=c(0,80),ylim=c(10,70),
        main="Smoothed trend line with bubble plot of BMI versus age")
# plot smoothing trend line
# library(KernSmooth)  # install and load the package if not already done
smth<-svysmooth(bmxbmi~ridageyr,d0709)
lines(smth,lwd=2)
```

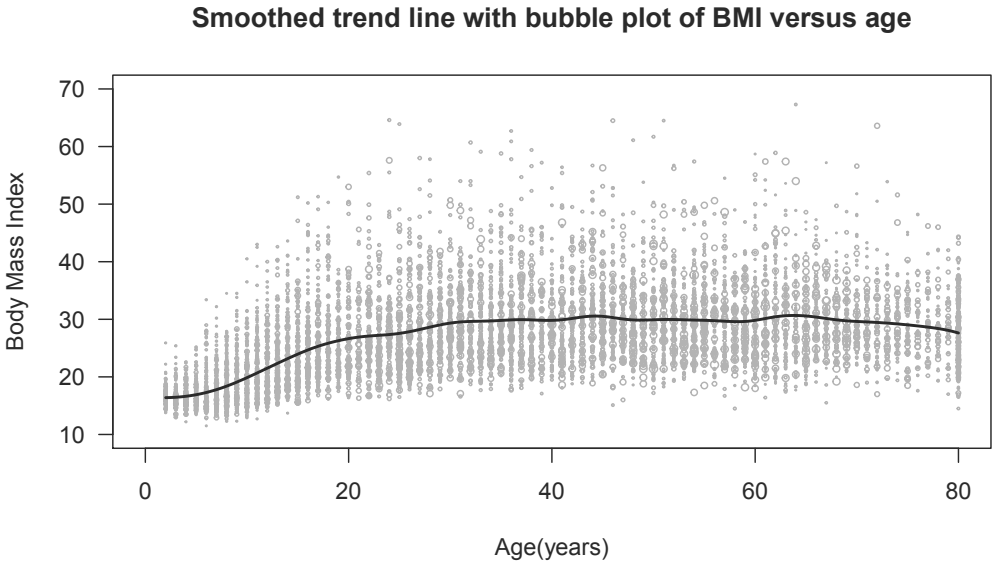

FIGURE 7.9: Smoothed trend line with bubble plot of BMI versus age.

Other smoothing methods may also be used to estimate trend lines. Figure 7.14 in SDA was created by calculating survey-weighted spline estimates (Zhang et al., 2015) of the trend line for a grid of x points and then connecting the dots with the *lines* function.

Smoothed trend lines for quantiles. To fit smoothed lines for quantiles, change to `method="quantreg"` in *svysmooth*. This smooths the regression quantiles from package `quantreg` (Koenker et al., 2021). Instead of fitting smoothed lines to the means of different groups, separate lines are drawn that estimate each conditional quantile (Koenker, 2005) specified in the *quantile* argument. You can think of this as "connecting the dots" of the quantiles shown in the side-by-side boxplots in Figure 7.8. We requested the quantiles corresponding to probabilities 0.05, 0.25, 0.5, 0.75, and 0.95 with the argument `taus=c(.05,.25,.5,.75,.95)`. This set of quantiles gives a good picture of the center (median) of the data, as well as the large and small values. Figure 7.10 includes the bubble plot and the smoothed line for each quantile.

```
# library(quantreg)) # install and load the package if not already done
# plot data bmxbmi~ridageyr
par(las=1) # make tick mark labels horizontal
svyplot(bmxbmi~ridageyr, design=d0709, style="bubble",basecol="gray",inches=0.03,
        xlab="Age (years)",ylab="Body Mass Index",xlim=c(0,80),ylim=c(10,70),
        main="Smoothed quantile trend lines")
# plot smoothed trend lines for quantiles
taus<-c(.05,.25,.5,.75,.95)
for (i in 1:length(taus)) {
 qsmth<-svysmooth(bmxbmi~ridageyr,d0709, quantile=taus[i],method="quantreg")
 lines(qsmth,lwd=1.2)
}
```

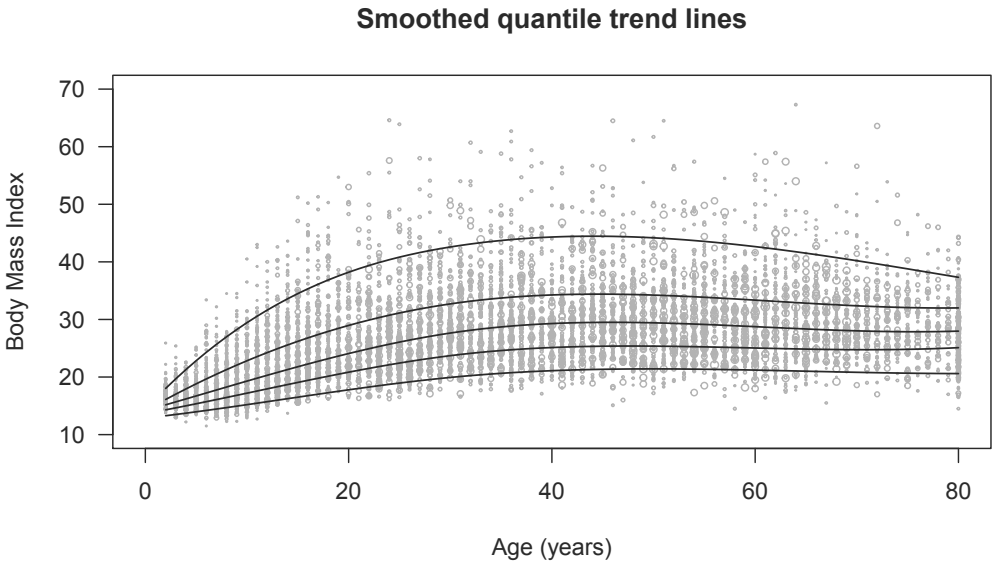

FIGURE 7.10: Smoothed quantile trend lines with bubble plot of BMI versus age.

Customizing graphs. Users who want to be able to do more customization of the graphs may want to write their own code. The graphs in SDA were produced with custom-written code (not using the *svyplot* function), and the file `ch07.R` includes code for using the `ggplot2` graphics visualization package (Wickham et al., 2020) to create more types of scatterplots with survey data.

7.6 Additional Code for Exercises

Some of the exercises in Chapter 7 of SDA ask you to construct an empirical cumulative distribution function (ecdf) or an empirical probability mass function (epmf). Any population characteristic can be estimated using these functions. There are functions in R that calculate commonly requested statistics such as means, totals, quantiles, and regression coefficients from survey data, so the survey analyst typically does not calculate or graph the ecdf. For continuous variables, it is usually more informative to create a histogram or a smoothed density estimate than to view the epmf, which may contain a large number of spikes.

The ecdf and epmf are useful concepts for learning about how survey weights work, however, because they illustrate how the sample is used to create a reconstruction of the population. And both are easy to calculate in R.

Example 7.5 of SDA. Function *emppmf* from package `SDAResources` calculates the empirical probability mass function for a variable with associated weights. Call the function as

```
emppmf(y,sampling.weight)
```

to return a list with component vectors *vals*, the distinct values of y, and *epmf*, the value of the epmf corresponding to each y value in *vals*.

Here we look at the stratified sample of heights in *htstrat*, where each female has a sampling weight of 1000/160 and each male has a sampling weight of 1000/40. It produces the estimated population proportion for each of the distinct values of *height*; the estimated proportion for value y is the sum of the weights for observations having $height = y$ divided by the sum of all the weights in the sample. Figure 7.11 plots the epmf for *height*, using `type="h"` to draw vertical lines.

```
# Empirical pmf for stratified sample of heights
# define sampling weight
htstrat$sampwt <- 1000/sum(htstrat$gender=="F")
htstrat$sampwt[htstrat$gender=="M"] <- 1000/sum(htstrat$gender=="M")
# use function emppmf to calculate pmf
strresult <- emppmf(htstrat$height,htstrat$sampwt)
# plot
par(las=1)
plot(strresult$vals, strresult$epmf,type="h",xlab="Height Value, y (cm)",
     ylab="Empirical pdf",lwd=1.2,
     main="Empirical pdf for stratified sample of heights (weighted)")
```

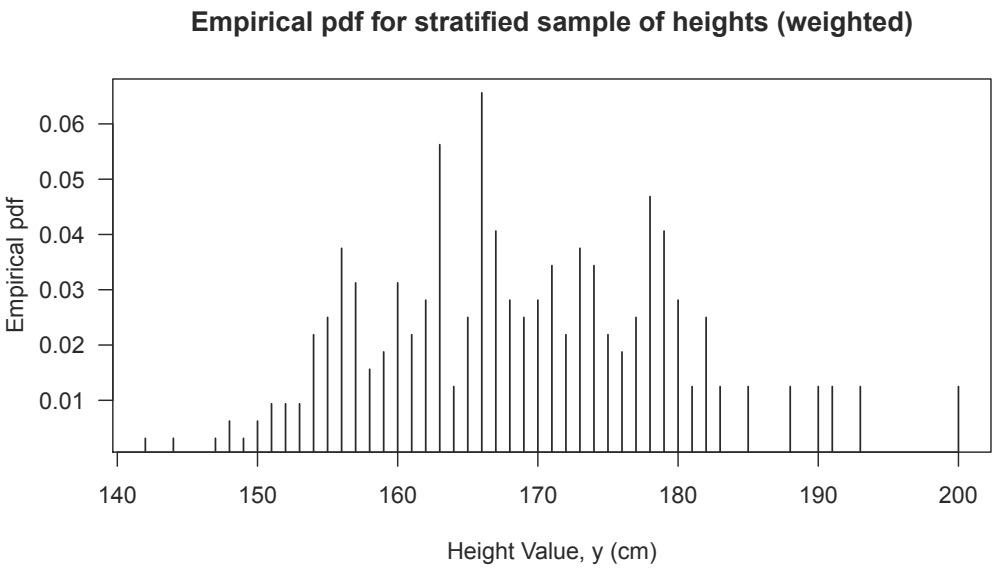

FIGURE 7.11: Empirical pmf for stratified sample of heights, using weights.

Function *ecdf* in base R will estimate the empirical cdf without weights, and function *svycdf* from the `survey` package will estimate the empirical cdf by incorporating survey weights. In contrast to *svyquantile*, *svycdf* does not interpolate but instead produces a right-continuous step function. Figure 7.12 shows the cdf of the population *htpop* (thick black) with the ecdf of sample *htstrat* without incorporating weights (medium red), and the ecdf of sample `htstrat` with weights (thin purple). We see that the medium red line is quite far off from the population cdf, while the thin purple line with weights is close to the population cdf.

```
# data(htstrat)
# Recall that
# d0710 <- svydesign(id = ~1, strata = ~gender, fpc = c(rep(1000,160),rep(1000,40)),
#                    data = htstrat)
cdf.weighted<-svycdf(~height, d0710)
cdf.weighted
## Weighted ECDFs: svycdf(~height, d0710)
## evaluate the function for height 144
cdf.weighted[[1]](144)
## [1] 0.00625
## compare to population and unweighted sample ecdfs.
cdf.pop<-ecdf(htpop$height)    # ecdf for population
cdf.samp<-ecdf(htstrat$height) # unweighted ecdf of sample
par(las=1,mar=c(5.1,4.1,2.1,2.1))
plot(cdf.pop, do.points = FALSE,
     xlab="Height value y",ylab="Empirical cdf",xlim=c(135,205),lwd=2.5,
     main="Empirical cdfs for population and sample")
lines(cdf.samp, col="red", do.points = FALSE, lwd=1.8)
lines(cdf.weighted[[1]], do.points = FALSE,  col ="purple",lwd=1)
legend("topleft", legend=c("Population", "Sample unweighted", "Sample weighted"),
       col=c("black", "red", "purple"),lwd=c(2.5,1.8,1),cex=0.8,bty="n")
```

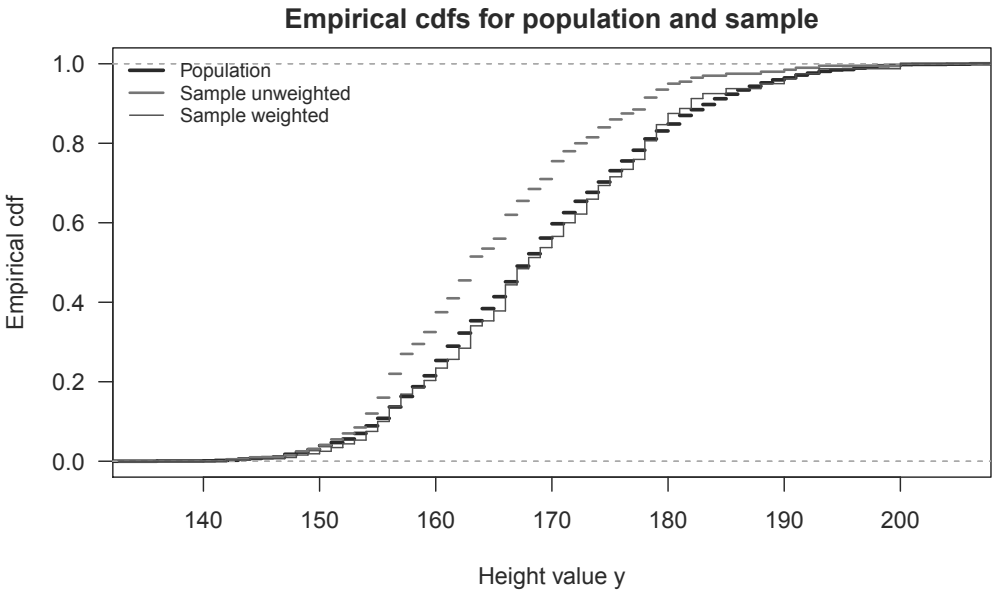

FIGURE 7.12: Empirical cdf of height for data *htpop*, and for data *htstrat* with and without weights.

7.7 Summary, Tips, and Warnings

Table 7.1 lists the major R functions used in this chapter.

Tips and Warnings

- For stratified multistage surveys, put the stratification variable in the *strata* argument of *svydesign*, and put the first-stage clustering variable in the *id* argument. Include the *weights* argument and do not include *fpc*. This will calculate the with-replacement variance approximations.
- If you want separate statistics for domains, first create the design object for the entire sample using the *svydesign* function, then use the *subset* or *svyby* function to calculate statistics for the domain of interest. This will ensure that standard errors for the domain statistics are calculated correctly. Do not subset the data first and then run *svydesign*—this can lead to incorrect standard error calculations.
- Incorporate the survey weights when constructing graphs if you want them to estimate the graphs that would be constructed if you had data from the entire population.

TABLE 7.1
Functions used for Chapter 7.

Function	Package	Usage
sample	base	Select a with-replacement sample with unequal probabilities
subset	base	Work with a subset of a vector, matrix, or data frame
confint	stats	Calculate confidence intervals, add df for t confidence interval
density	stats	Compute a kernel density estimate for self-weighting data
ecdf	stats	Calculate empirical cdf from self-weighting sample
par	graphics	Set graphics parameters
hist	graphics	Draw a histogram without weights
boxplot	graphics	Draw a boxplot without weights
plot	graphics	Draw a scatterplot without weights
mstage	sampling	Select a stratified multistage sample
svydesign	survey	Specify the survey design
svymean	survey	Calculate mean and standard error of mean
svyquantile	survey	Calculate quantiles and their confidence intervals
svyhist	survey	Draw a histogram of survey data, incorporating the weights
svyboxplot	survey	Draw boxplot of survey data, incorporating the weights
svyplot	survey	Draw scatterplot of survey data, incorporating the weights
svysmooth	survey	Estimate a smoothed density estimate or trend line from survey data
svycdf	survey	Calculate the empirical cdf from survey data
emppmf	SDAResources	Calculate the empirical probability mass function from survey data