4

Ratio and Regression Estimation

Ratio and regression estimation both use auxiliary information to increase the precision of survey estimates. This chapter shows how to incorporate auxiliary information into survey data analyses using R. The code in this chapter is in file `ch04.R` on the book website.

4.1 Ratio Estimation

Examples 4.2 and 4.3 of SDA. The *svyratio* function in the `survey` package (Lumley, 2020) computes ratios from survey data. Let's see how it works for Examples 4.2 and 4.3 of SDA. As the correlation coefficient between variables *acres87* and *acres92* is 0.995806. *acres87* would be an excellent auxiliary variable for ratio estimation. The code and output to estimate the ratio $\bar{y}_{\mathcal{U}}/\bar{x}_{\mathcal{U}}$, where $\bar{y}_{\mathcal{U}}$ is the population mean of *acres92* and $\bar{x}_{\mathcal{U}}$ is the population mean of *acres87*, are given in the following.

```
data(agsrs)
n<-nrow(agsrs) #300
agsrs$sampwt <- rep(3078/n,n)
agdsrs <- svydesign(id = ~1, weights=~sampwt, fpc=rep(3078,300), data = agsrs)
agdsrs
## Independent Sampling design
## svydesign(id = ~1, weights = ~sampwt, fpc = rep(3078, 300), data = agsrs)
# correlatIon of acres87 and acres92
cor(agsrs$acres87,agsrs$acres92)
## [1] 0.995806
# estimate the ratio acres92/acres87
sratio<-svyratio(numerator = ~acres92, denominator = ~acres87,design = agdsrs)
sratio
## Ratio estimator: svyratio.survey.design2(numerator = ~acres92,
##     denominator = ~acres87, design = agdsrs)
## Ratios=
##           acres87
## acres92 0.9865652
## SEs=
##             acres87
## acres92 0.005750473
confint(sratio, df=degf(agdsrs))
##                       2.5 %    97.5 %
## acres92/acres87 0.9752487 0.9978818
```

The sample in *agsrs* is an SRS, so we specify the survey design object in *svydesign* exactly as we did in Chapter 2. The only new feature is the *svyratio* function, which calculates the ratio $\hat{B} = \bar{y}/\bar{x}$ and its standard error.

DOI: 10.1201/9781003228196-4

Now that we have estimated the ratio from the data, we can use the *predict* function to obtain the ratio estimates of the population mean and total of y. The population total of x is $t_x = 964{,}470{,}625$ and the population mean of x is $\bar{x}_{\mathcal{U}} = t_x/N$. Note that the value of t_x came from the official U.S. Census of Agriculture statistics for 1987 (U.S. Bureau of the Census, 1995). This is greater than the sum of all x values in data set *agpop* because some counties in the population have missing values for *acres87*, as we saw in Section 1.7.

```
# provide the population total of x
xpoptotal <- 964470625
# Ratio estimate of population total
predict(sratio,total=xpoptotal)
## $total
##           acres87
## acres92 951513191
##
## $se
##         acres87
## acres92 5546162
# Ratio estimate of population mean
predict(sratio,total=xpoptotal/3078)
## $total
##          acres87
## acres92 309133.6
##
## $se
##          acres87
## acres92 1801.872
```

Examples 4.2 and 4.3 of SDA also explore the scatterplot of *acres92* versus *acres87*. Because all of the weights are the same (=3078/300), we can use the base R function *plot* to display the data in Figure 4.1 (see Chapter 7 for how to draw scatterplots for samples with unequal weights).

We scale the x and y variables so that the plot shows millions of acres instead of acres, and specify the axis labels in the *xlab* and *ylab* arguments. The function *abline* draws the line through the origin with slope $\hat{B}$.

```
par(las=1) # make tick mark labels horizontal (optional)
plot(x=agsrs$acres87/1e6,y=agsrs$acres92/1e6,
  xlab="Millions of Acres Devoted to Farms (1987)",
  ylab = "Millions of Acres Devoted to Farms (1992)",
  main = "Acres Devoted to Farms in 1987 and 1992")
# draw line through origin with slope Bhat
abline(0,coef(sratio))
```

Example 4.5 of SDA. Variables of interest in this example are the number of woody seedlings in pig-protected areas under each of ten sampled oak trees in 1992 (*seed92*) and 1994 (*seed94*) on Santa Cruz Island, California. The code below draws the scatterplot (shown in Figure 4.4 of SDA and not reproduced here) of *seed94* versus *seed92*. It also calculates the correlation of the two variables.

```
#scatterplot of seed92 and seed94
data(santacruz)
plot(santacruz$seed92,santacruz$seed94,
     main="Number of seedlings in 1994 and 1992",
     xlab="Number of seedlings in 1992",ylab="Number of seedlings in 1994")
```

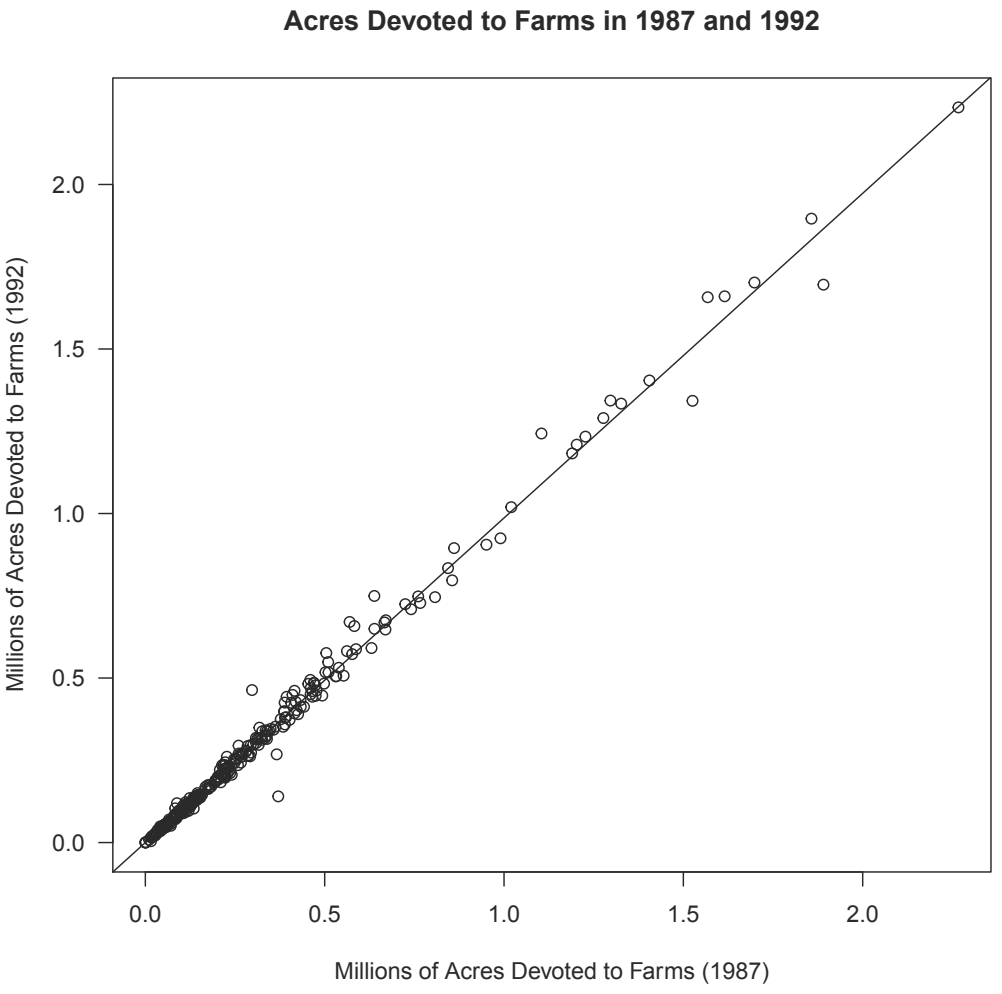

FIGURE 4.1: Scatterplot of acres devoted to farms in 1987 and 1992 (data *agsrs*).

```
cor(santacruz$seed92,santacruz$seed94)
## [1] 0.6106537
```

Now let's calculate the ratio of the number of seedlings in 1994 to the number of seedlings in 1992. Because the number of trees in the population is unknown, we define *sampwt* to be 1 for each observation, and we omit the *fpc* argument in the *svydesign* function (alternatively, one could set the population size to be a large number for the weights and fpc).

```
nrow(santacruz) #10
## [1] 10
santacruz$sampwt <- rep(1,nrow(santacruz))
design0405 <- svydesign(ids = ~1, weights = ~sampwt, data = santacruz)
design0405
## Independent Sampling design (with replacement)
## svydesign(ids = ~1, weights = ~sampwt, data = santacruz)
#Ratio estimation using number of seedlings of 1992 as auxiliary variable
```

```
sratio3<-svyratio(~seed94, ~seed92,design = design0405)
sratio3
## Ratio estimator: svyratio.survey.design2(~seed94, ~seed92, design = design0405)
## Ratios=
##           seed92
## seed94 0.2961165
## SEs=
##           seed92
## seed94 0.1152622
confint(sratio3, df=10-1)
##                     2.5 %    97.5 %
## seed94/seed92 0.03537532 0.5568577
```

4.2 Regression Estimation

Example 4.7 of SDA. Function *svyglm* calculates regression coefficients and regression estimators from survey data. It is the survey analog of the R function *glm*, which fits generalized linear models.

```
data(deadtrees)
head(deadtrees)
##   photo field
## 1    10    15
## 2    12    14
## 3     7     9
## 4    13    14
## 5    13     8
## 6     6     5
nrow(deadtrees) # 25
## [1] 25
# Fit with survey regression
dtree<- svydesign(id = ~1, weight=rep(4,25), fpc=rep(100,25), data = deadtrees)
myfit1 <- svyglm(field~photo, design=dtree)
summary(myfit1) # displays regression coefficients
##
## Call:
## svyglm(formula = field ~ photo, design = dtree)
##
## Survey design:
## svydesign(id = ~1, weight = rep(4, 25), fpc = rep(100, 25), data = deadtrees)
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.0593     1.3930   3.632   0.0014 **
## photo         0.6133     0.1259   4.870 6.44e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 5.548341)
##
## Number of Fisher Scoring iterations: 2
confint(myfit1,df=23) # df = 25-2
```

```
##                 2.5 %    97.5 %
## (Intercept) 2.3291420 7.7894421
## photo       0.3664593 0.8600894
# Regression estimate of population mean field trees
newdata <- data.frame(photo=11.3)
predict(myfit1, newdata)
##     link    SE
## 1 11.989 0.418
confint(predict(myfit1, newdata),df=23)
##      2.5 %   97.5 %
## 1 11.12455 12.85404
# Estimate total field tree, add population size in total= argument
newdata2 <- data.frame(photo=1130)
predict(myfit1, newdata2, total=100)
##     link     SE
## 1 1198.9 41.802
confint(predict(myfit1, newdata2,total=100),df=23)
##      2.5 %   97.5 %
## 1 1112.455 1285.404
```

The regression estimation uses the following functions:

svydesign As before, use *svydesign* to describe the design of the survey, which, in this case, is an SRS with sample size $n = 25$ and population size $N = 100$. Each observation has a weight of $100/25 = 4$.

svyglm The function `svyglm(field~photo, design=dtree)` tells which variables to analyze in the regression statement. The dependent (y) variable is before the ~ sign, and the independent (x) variables follow it. In this example, the dependent variable is *field* and there is one independent variable, *photo*. The *design* argument tells the name of the survey design object (here, *dtree*) to use in calculations.

summary The *summary* function gives the estimates of the regression coefficients, their associated standard errors, and the t statistic and p-value for testing whether each regression parameter equals 0 (the standard errors and tests will be discussed in Chapter 11).

confint As before, the *confint* function requests confidence limits for the regression parameters. You can specify the degrees of freedom with the *df* argument if desired: here the df equal the sample size minus 2: $25 - 2 = 23$. If you omit the *df* argument, the function uses the normal distribution to produce confidence intervals.

predict The *predict* function allows you to obtain estimates for predicted values from the estimated regression equation. For regression estimation of the mean, we first define a new data frame with `photo = 11.3` because we want to calculate the predicted value of the regression function at $\bar{x}_{\mathcal{U}} = 11.3$. The statement `predict(myfit1, newdata)` gives $\hat{B}_0 * (1) + \hat{B}_1 * (11.3)$. Regression estimation of the total multiplies each of these by the population size N (here, $N = 100$) by including the argument `total=100` in the *predict* function (called as `predict(myfit1, newdata2, total=100)`), which estimates $\hat{B}_0 * (100) + \hat{B}_1 * (1130)$, where $t_x = 1130$.

Note that the output gives slightly different standard errors and confidence intervals for the regression estimates of the mean and total than SDA because the R functions use a slightly different (although asymptotically equivalent) formula to calculate the standard error. See Section 11.6 of SDA for a discussion of the two variance estimates used.

4.3 Domain Estimation

A domain is a subset of the population for which estimates are desired. Because estimated domain means and totals are ratio estimates, they can be calculated with the *svyratio* function. It is usually easier, however, to compute them using *subset* or *svyby*.

The procedure to calculate estimates for domains is essentially the same as that to calculate estimates for the full sample, but you need to redefine the design for the domain with the *subset* function so that standard errors are calculated correctly. Type:

```
newdesign<-subset(original_design, domain)
```

Example 4.8 of SDA. The following code uses the *subset* function to request design information for each level of the variable *farmcat*, which is defined to equal "large" when *farms92* $\geq$ 600 and "small" otherwise.

```
agsrsnew<-agsrs #copy agsrs as agsrsnew, since we want to create a new column
# we calculated sampwt in the first code in this chapter
# define new variable farmcat
agsrsnew$farmcat<-rep("large",n)
agsrsnew$farmcat[agsrsnew$farms92 < 600] <- "small"
head(agsrsnew)
##              county state acres92 acres87 acres82 farms92 farms87 farms82
## 1     COFFEE COUNTY    AL  175209  179311  194509     760     842     944
## 2    COLBERT COUNTY    AL  138135  145104  161360     488     563     686
## 3      LAMAR COUNTY    AL   56102   59861   72334     299     362     447
## 4    MARENGO COUNTY    AL  199117  220526  231207     434     471     622
## 5     MARION COUNTY    AL   89228  105586  113618     566     658     748
## 6 TUSCALOOSA COUNTY    AL   96194  120542  134616     436     521     650
##   largef92 largef87 largef82 smallf92 smallf87 smallf82 region sampwt farmcat
## 1       29       28       21       57       47       66      S  10.26   large
## 2       37       41       42       12       44       47      S  10.26   small
## 3        4        4        3       16       20       30      S  10.26   small
## 4       48       66       62       14       11       28      S  10.26   small
## 5        7        9        9       11       23       27      S  10.26   small
## 6       20       17       23       18       32       29      S  10.26   small
dsrsnew <- svydesign(id = ~1, weights=~sampwt, fpc=rep(3078,300), data=agsrsnew)
# domain estimation for large farmcat with subset statement
dsub1<-subset(dsrsnew,farmcat=='large')   # design info for domain large farmcat
smean1<-svymean(~acres92,design=dsub1)
smean1
##           mean    SE
## acres92 316566 21553
df1<-sum(agsrsnew$farmcat=='large')-1 #calculate domain df if desired
df1
## [1] 128
confint(smean1, level=.95,df=df1) # CI
##            2.5 %   97.5 %
## acres92 273918.9 359212.4
stotal1<-svytotal(~acres92,design=dsub1)
stotal1
##             total       SE
## acres92 418987302 38938277
confint(stotal1, level=.95,df=df1)
```

```
##             2.5 %    97.5 %
## acres92 341941269 496033335
# domain estimation for small farmcat
dsub2<-subset(dsrsnew,farmcat=='small')  # design info for domain small farmcat
smean2<-svymean(~acres92,design=dsub2)
smean2
##           mean    SE
## acres92 283814 28852
df2<-sum(agsrsnew$farmcat=='small')-1 #calculate domain df if desired
confint(smean2, level=.95,df=df2) #CI
##            2.5 %   97.5 %
## acres92 226858.9 340768.5
stotal2<-svytotal(~acres92,design=dsub2)
stotal2
##             total       SE
## acres92 497939808 55919525
confint(stotal2, level=.95,df=df2)
##             2.5 %    97.5 %
## acres92 387553732 608325884
```

You can also calculate statistics for all domains defined by a factor variable at the same time, using the *svyby* function. Here, we estimate the population total and mean for both domains defined by *factor(farmcat)*. The first argument of *svyby* contains the variable(s) to analyze, and the second argument is the factor variable that defines the domains. The last argument gives the name of the function that is to be applied to each group in the *by* argument.

```
bothtot<-svyby(~acres92,by=~factor(farmcat),design=dsrsnew,svytotal)
bothtot
##       factor(farmcat)   acres92       se
## large           large 418987302 38938277
## small           small 497939808 55919525
confint(bothtot,level=.95)
##           2.5 %    97.5 %
## large 342669682 495304922
## small 388339553 607540062
bothmeans<-svyby(~acres92,by=~factor(farmcat),design=dsrsnew,svymean)
bothmeans
##       factor(farmcat)  acres92       se
## large           large 316565.7 21553.21
## small           small 283813.7 28852.24
confint(bothmeans,level=.95)
##          2.5 %   97.5 %
## large 274322.1 358809.2
## small 227264.4 340363.1
```

Note that confidence intervals here are slightly smaller than those given from the calculations with the *subset* function and in Example 4.8 of SDA. Because we did not specify the df in the *confint* function, it uses a normal distribution to calculate the intervals; the previous code, using the *subset* function, calculated the confidence intervals using a t distribution having $n_d - 1$ df, where n_d is the sample size of domain d.

Warning. In SRSs, you can calculate domain means and their standard errors by first forming a new, subsetted data set that consists of the observations in the domain and then calculating statistics on the subsetted data set. In complex surveys, however, that

method can result in incorrect standard errors (see Section 11.3 of SDA). To obtain correct statistics for domains, first define the survey design object using the function *svydesign* with the entire data set. Then use the function *subset* or *svyby* with the survey design object to obtain correct inferences for domains.

4.4 Poststratification

Example 4.9 of SDA. The *postStratify* function computes poststratification weights and uses them to estimate population means and totals, along with their standard errors (discussed in Chapter 11 of SDA). Let's poststratify the SRS in *agsrs* by variable *region*.

```
data(agsrs)
dsrs <- svydesign(id = ~1, weights=rep(3078/300,300), fpc=rep(3078,300),
                  data = agsrs)
# Create a data frame that gives the population totals for the poststrata
pop.region <- data.frame(region=c("NC","NE","S","W"), Freq=c(1054,220,1382,422))
# create design information with poststratification
dsrsp<-postStratify(dsrs, ~region, pop.region)
summary(dsrsp)
## Independent Sampling design
## postStratify(dsrs, ~region, pop.region)
## Probabilities:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.09242 0.09407 0.09407 0.09771 0.10152 0.10909
## Population size (PSUs): 3078
## Data variables:
##  [1] "county"   "state"    "acres92"  "acres87"  "acres82"  "farms92"
##  [7] "farms87"  "farms82"  "largef92" "largef87" "largef82" "smallf92"
## [13] "smallf87" "smallf82" "region"
1/unique(dsrsp$prob)  # See the poststratified weight for each region
## [1] 10.630769 10.820513  9.850467  9.166667
svymean(~acres92, dsrsp)
##           mean    SE
## acres92 299778 17513
svytotal(~acres92, dsrsp)
##             total       SE
## acres92 922717031 53906392
```

The only new feature here is the *postStratify* function:

```
postStratify(design=dsrs, strata=region, population=pop.region)
```

The *postStratify* function tells R to construct poststratification weights, using poststrata in the variable *region*. The third argument is the name of the data frame (here, *pop.region*) that gives the population totals for the poststrata. The poststratified estimates of the population mean and total of *acres92*, when calculated with poststratified design object *dsrsp*, are reported together with standard errors when the *svymean* or *svytotal* function is called.

Note that the standard errors reported by the `survey` package differ slightly from those in Example 4.9 of SDA because a slightly different (although asymptotically equivalent) estimator for the variance is used (see Section 11.6 of SDA).

4.5 Ratio Estimation with Stratified Sampling

The *svyratio* function will compute either separate or combined ratio estimates. The default is combined ratio estimation, which calculates the ratio $\hat{\bar{y}}/\hat{\bar{x}}$, where $\hat{\bar{y}}$ is the estimate of the mean of y using the stratified design and $\hat{\bar{x}}$ is the estimated mean of x. All we need to do is to include the stratification information in the design structure formed by the *svydesign* function.

Combined ratio estimator. The following shows how to compute the ratio of *acres92* to *acres87* and the ratio estimator of the total for the stratified sample in *agstrat*, using the combined ratio estimator.

```
data(agstrat)
popsize_recode <- c('NC' = 1054, 'NE' = 220, 'S' = 1382, 'W' = 422)
agstrat$popsize <- popsize_recode[agstrat$region]
# input design information for agstrat
dstr <- svydesign(id = ~1, strata = ~region, fpc = ~popsize, weight = ~strwt,
                  data = agstrat)
# now compute the combined estimator of the ratio
combined<-svyratio(~ acres92,~acres87,design = dstr)
combined
## Ratio estimator: svyratio.survey.design2(~acres92, ~acres87, design = dstr)
## Ratios=
##           acres87
## acres92 0.9899971
## SEs=
##              acres87
## acres92 0.006187757
# we can get the combined ratio estimator of the population total
# with the predict function
predict(combined,total=964470625)
## $total
##           acres87
## acres92 954823130
##
## $se
##         acres87
## acres92 5967910
```

Separate ratio estimator. You can calculate ratios separately for each stratum by including `separate=TRUE` in the *svyratio* function.

```
separate<-svyratio(~acres92,~acres87,design = dstr,separate=TRUE)
separate
## Stratified ratio estimate: svyratio.survey.design2(~acres92, ~acres87,
##     design = dstr, separate = TRUE)
## Ratio estimator: Stratum == "NC"
## Ratios=
##           acres87
## acres92 0.9750666
## SEs=
##              acres87
## acres92 0.005483458
## Ratio estimator: Stratum == "NE"
```

```
## Ratios=
##           acres87
## acres92 0.8956073
## SEs=
##             acres87
## acres92 0.008853011
## Ratio estimator: Stratum == "S"
## Ratios=
##           acres87
## acres92 0.9935483
## SEs=
##            acres87
## acres92 0.01418835
## Ratio estimator: Stratum == "W"
## Ratios=
##          acres87
## acres92 1.011974
## SEs=
##            acres87
## acres92 0.01169809
#  Define the stratum totals for acres87 as a list:
stratum.xtotals <- list(NC=350474227,NE=22033421,S=280631939,W=311331038)
predict(separate,stratum.xtotals)
## $total
##           acres87
## acres92 955349448
##
## $se
##         acres87
## acres92 5731438
```

4.6 Model-Based Ratio and Regression Estimation

This section is optional and need only be read if covering Section 4.6 of SDA.

Example 4.11 of SDA. A model-based analysis of data from an SRS uses the same techniques taught in an introductory statistics class. Since the model-based analysis does not make use of the sampling weights, the *lm* or *glm* functions, which fit linear and generalized linear models for non-survey data, are used to fit the regression models and obtain the residuals. Here we use the *lm* function.

The format for fitting a regression model with *lm* is similar to *svyglm*, but with one important difference: the weights mean different things in the two functions. In the *svyglm* function, `weights=` tells how many population units are represented by each sample unit. In the *lm* function, the weight variable contains relative weights for a weighted least squares fit.

The model used is $Y_i = \beta x_i + \varepsilon_i$, with $V(\varepsilon_i) = \sigma^2 x_i$. The model has variance proportional to x_i, so obtaining the best linear unbiased estimates under this model would use a weight value proportional to the reciprocal of the variances. This is specified by defining the variable $recacr87 = 1/acres87$ when $acres87 > 0$ and $recacr87 =$ NA when $acres87 = 0$ (to avoid division by zero).

```
data(agsrs)
# define weights to use for weighted least squares analysis
agsrs$recacr87<-agsrs$acres87
agsrs$recacr87[agsrs$acres87!=0] <- 1/agsrs$acres87[agsrs$acres87!=0]
agsrs$recacr87[agsrs$acres87==0] <- NA
# fit weighted least squares model without intercept
fit<-lm(acres92~acres87-1,weights=recacr87,data=agsrs)
summary(fit)
##
## Call:
## lm(formula = acres92 ~ acres87 - 1, data = agsrs, weights = recacr87)
##
## Weighted Residuals:
##    Min     1Q Median     3Q    Max
## -369.9  -22.2   -5.8   10.8  311.7
##
## Coefficients:
##         Estimate Std. Error t value Pr(>|t|)
## acres87 0.986565   0.004844   203.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 46.1 on 298 degrees of freedom
##   (1 observation deleted due to missingness)
## Multiple R-squared:  0.9929,Adjusted R-squared:  0.9928
## F-statistic: 4.149e+04 on 1 and 298 DF,  p-value: < 2.2e-16
anova(fit)
## Analysis of Variance Table
##
## Response: acres92
##            Df   Sum Sq  Mean Sq F value    Pr(>F)
## acres87     1 88168461 88168461   41487 < 2.2e-16 ***
## Residuals 298   633307     2125
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
# find predicted value at population total for x
newdata3 <- data.frame(acres87=964470625)
predict(fit, newdata3, se.fit=TRUE)
## $fit
##         1
## 951513191
##
## $se.fit
## [1] 4671509
##
## $df
## [1] 298
##
## $residual.scale
## [1] 46.0998
```

The *weights* argument in *lm* specifies that a weighted least squares analysis is performed with weights *recacr87*, minimizing the weighted sum of squares $\sum_{i\in\mathcal{S}}(y_i - \beta x_i)^2/x_i$. The "$-1$" in `lm(acres92~acres87-1)` tells that the model is to be fit without an intercept. The *summary* function displays the regression coefficient $\hat{\beta} = 0.986565$ and the *anova* function

displays the ANOVA table. The model is fit to the 299 observations that have *acres87* > 0.

The *predict* function requests the predicted value from the regression model when *acres87* takes on the value $t_x = 964{,}470{,}625$, giving $\hat{t}_{yM} = \hat{\beta} t_x = 951{,}513{,}191$. The standard error, without the fpc, is $\sigma t_x / \sqrt{\sum_{i \in \mathcal{S}} x_i} = 4{,}671{,}509$.

Note that the sum of squares for error in the ANOVA table, 633,307, is the sum of squares of the weighted residuals, so the mean squared error in the ANOVA table gives $\hat{\sigma}^2 = 2125.19$.

The residuals produced by *lm* are $e_i = y_i - \hat{y}_i$. For a ratio model, the weighted residuals $e_{iw} = e_i / \sqrt{x_i}$ should be plotted instead of e_i, because if the model variance structure is correct, the e_{iw} should all have approximately equal variances and a plot of e_{iw} versus the predicted values or x_i will show no patterns.

```
# plot weighted residual versus predicted values
wresid<-fit$residuals*sqrt(fit$weights)
par(las=1)
plot(fit$fitted.values, wresid,
     main="Plot of weighted residuals versus predicted values",
     xlab="Predicted value from regression model",
     ylab="Weighted residuals")
```

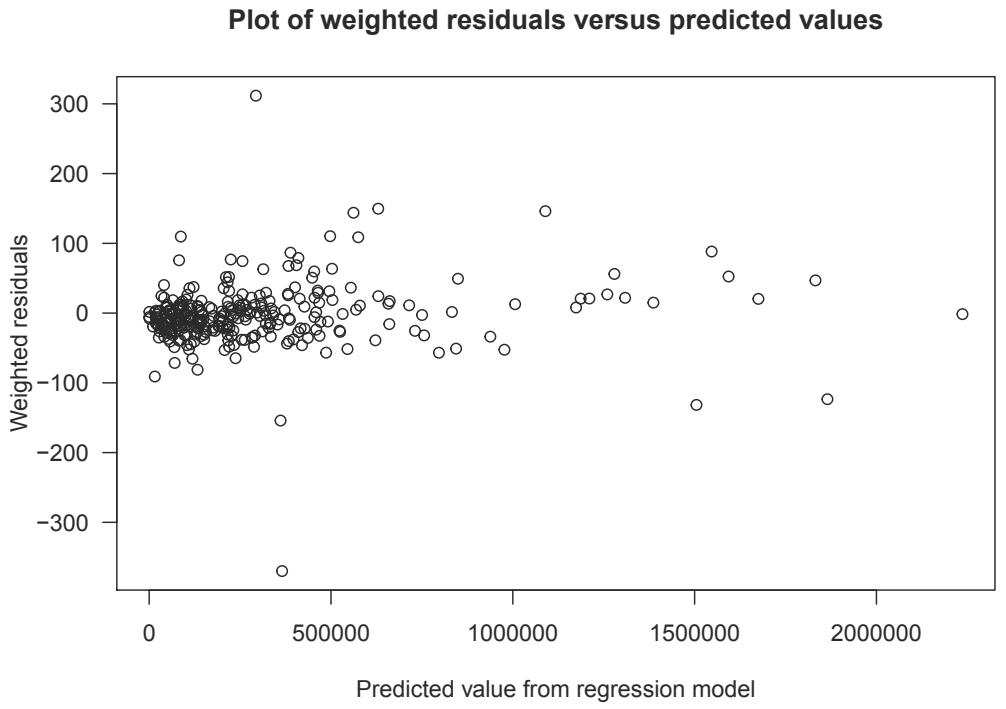

FIGURE 4.2: Plot of weighted residuals versus fitted values.

Figure 4.2 shows a couple of potential outliers, but no other indications that the model is inappropriate.

Example 4.12 of SDA. The *lm* function is also used to fit a regression model and to obtain residuals for the dead tree data from Example 4.7 of SDA.

```
data(deadtrees)
# Fit with lm
fit2 <- lm(field~photo, data=deadtrees)
summary(fit2)
##
## Call:
## lm(formula = field ~ photo, data = deadtrees)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.0319 -1.8053  0.1947  1.4212  3.8080
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.0593     1.7635   2.869 0.008676 **
## photo         0.6133     0.1601   3.832 0.000854 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.406 on 23 degrees of freedom
## Multiple R-squared:  0.3896,Adjusted R-squared:  0.3631
## F-statistic: 14.68 on 1 and 23 DF,  p-value: 0.0008538
# Estimate mean field trees
newdata <- data.frame(photo=11.3)
predict(fit2, newdata,se.fit=TRUE)
## $fit
##        1
## 11.98929
##
## $se.fit
## [1] 0.4941007
##
## $df
## [1] 23
##
## $residual.scale
## [1] 2.406153
```

Because the regression model is fit under the assumption that $V(\varepsilon_i) = \sigma^2$ for all observations, no *weights* argument is used in *lm*. The *predict* function gives the regression estimate of the population mean $\hat{\beta}_0 + \hat{\beta}_1 \bar{x}_{\mathcal{U}} = 11.9893$, and its standard error (without fpc) of 0.494. These are the values calculated in Example 4.12 of SDA. Typing `summary(fit2)` gives the regression coefficients, their standard errors, and other information about the fit.

We have applied both *lm* and *svyglm* to analyze the tree data from Example 4.7. Table 4.1 compares the estimates and standard errors from the two functions. All the point estimates are the same, but the standard errors from *svyglm* differ from those calculated by *lm*; Sections 4.6 and 11.4 of SDA discuss why that occurs.

TABLE 4.1
Comparison of the estimates and standard errors for model `field~photo` from the *lm* and *svyglm* functions.

	Intercept		**Slope**		**Predicted Value,** $x = 11.3$	
	lm	svyglm	lm	svyglm	lm	svyglm
Estimate	5.0593	5.0593	0.6133	0.6133	11.989	11.989
SE	1.7635	1.3930	0.1601	0.1259	0.494	0.418

Requesting `plot(fit2)` produces a collection of residual and diagnostic plots from the *lm* model object. Figure 4.3 displays the plot of the residuals versus *photo* (the x variable), which shows no pattern.

```
# plot residuals versus predicted values
plot(deadtrees$photo, fit2$residuals,
    main="Plot of residuals versus photo values",
    xlab="Photo values (x variable)",
    ylab="Residuals")
```

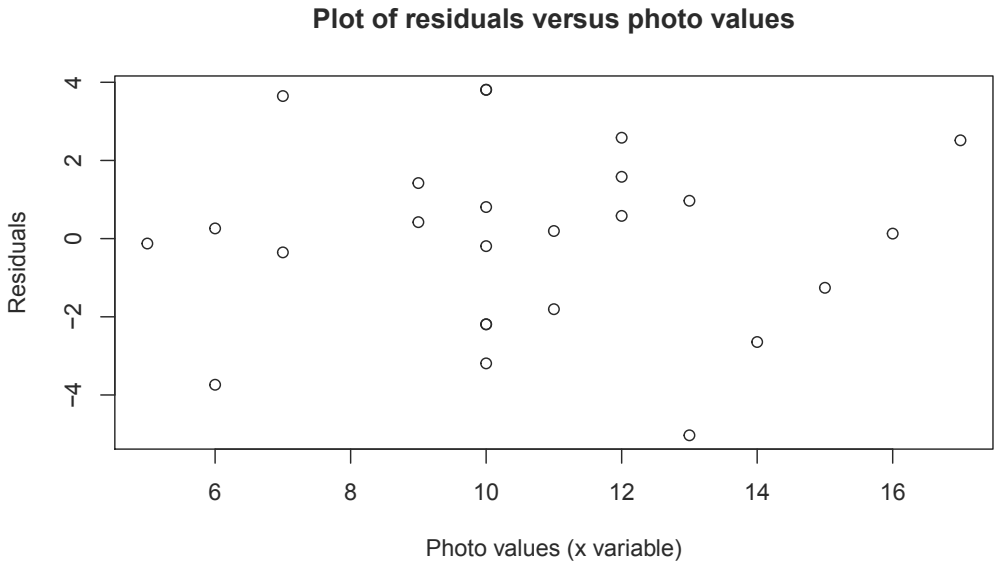

FIGURE 4.3: Plot of residuals versus x variable.

4.7 Summary, Tips, and Warnings

Table 4.2 lists the major functions used in this chapter to compute ratio and regression estimates, calculate statistics for domains, and create poststratification weights.

To calculate a ratio or ratio estimator for an SRS or stratified sample, use the *svyratio* function from the `survey` package. The function *svyglm* fits regression models to survey data.

Tips and Warnings

- Draw a scatterplot of your data when fitting a ratio or regression model, so you can see whether ratio or regression estimation is likely to improve efficiency.
- For domain estimation, first define a design object for the entire sample with the *svydesign* function. Then use the *subset* function to define a domain of interest, or calculate statistics for all domains with the *svyby* function. It may be tempting to calculate statistics for a subset of the sample by creating a data set containing only that subset, but doing that can result in incorrect standard errors for domain statistics.
- Poststratification can be done using the *postStratify* function.

TABLE 4.2
Functions used for Chapter 4.

Function	Package	Usage
subset	base	Work with a subset of a vector, matrix, or data frame
confint	stats	Calculate confidence intervals
cor	stats	Calculate the correlation of vectors (not using survey methods)
lm	stats	Fit a linear model to a data set (not using survey methods)
anova	stats	Compute an analysis of variance table from a model object
predict	stats	Obtain predicted values from a model object
plot	graphics	Draw a scatterplot of data
abline	graphics	Add a straight line to a plot
svydesign	survey	Specify the survey design
svymean	survey	Calculate mean and standard error of mean
svyratio	survey	Calculate a ratio or ratio estimate from survey data
svyglm	survey	Fit a regression model to survey data. The coefficients may then be used to calculate regression estimates
svytotal	survey	Calculate total and standard error of total
svyby	survey	Calculate statistics for subsets of a survey defined by a factor variable
postStratify	survey	Adjust the sampling weights using poststratification