R Chapter 1

We set apart the R code in a different font, and preface it with a ">" symbol. This is the symbol that appears on the R console after which you type in the command. Don't type in the > symbol itself. After you are done typing in the code for a particular command, press *Return* (i.e., the Enter key). More information on R, including scripts that go along with many of the examples in each chapter is available on the book's website www.ArtofStat.com.

- **Data Entry:** To create a single list of observations of, say, numbers of hours watching TV, use the `c()` command, as in `> tvhours <- c(3, 1, 0, 0, 2, 1, 3)` for entering seven observations. The `c` stands for concatenate. To see the result of your entry, type `> tvhours` and press *Return*. For reading in descriptions instead of numbers, such as seven observations on happiness, use `> happiness <- c('very', 'not', 'very', 'pretty', 'very', 'not', 'pretty')`. You can use single quotes (as in 'very') or double quotes (as in "very") to enclose each description.

  For reading in more complex data, it is best to create a data file as discussed in Section 1.3 with a spreadsheet program (i.e., Excel or Google Sheets) and save it in .csv format. Then, in R, use the `read.csv()` function to read in the data file. To automatically open a dialogue that lets you select the file, use `> mydata <- read.csv(file.choose())`. Then, to see the result, type `> mydata` and press *Return*. Here is how your screen should look like when entering the above commands:

```
> tvhours <- c(3, 1, 0, 0, 2, 1, 3)
> tvhours
[1] 3 1 0 0 2 1 3
> happiness <- c('very', 'not', 'very', 'pretty',
 'very', 'not', 'pretty')
> happiness
[1] "very"   "not"    "very"   "pretty"
[5] "very"   "not"    "pretty"
> mydata <- read.csv(file.choose())
> mydata
   Student Gender Ethnicity Age TV_hours
1        1      m         h  32        3
2        2      f         h  23       15
3        3      f         w  27        0
```

- **Random Samples:** The R command `sample()` generates random numbers. For instance, `> sample(30, 5)` provides five numbers randomly selected from the numbers 1 through 30.

R Chapter 2

Check out the sample R scripts posted on www.ArtofStat.com (go to the tab titled "R Code"), which replicate the entire analysis from Examples 2 through 10. Here is just a short summary.

- **Frequency Tables and Bar and Pie Charts:** To create a frequency table when data are already available in summarized form, use the `data.frame` command to create two columns, as in:

```
> favcol <- c('red', 'blue', 'green')

> frequency <- c(25, 17, 14)
> mytab <- data.frame(favcol, frequency)
```

Then, get percentages for each category through

```
> mytab$percent <- 100*mytab$frequency/
  sum(mytab$frequency)
> mytab
  favcol frequency  percent
1    red        25 44.64286
2   blue        17 30.35714
3  green        14 25.00000
```

You can obtain a pie chart with

```
> pie(mytab$frequency, labels=mytab$favcol)
```

and a bar graph with

```
> barplot(mytab$frequency, names.arg=
mytab$ favcol)
```

If you prefer to plot percentages, use `mytab$percent` instead.

If you have the individual observations, as in

```
> favcol <- c('blue', 'red', 'red', 'blue',
'green', 'red', 'blue', 'red', 'red', 'green')
```

you can get a frequency table like this

```
> mytab <- as.data.frame(table(favcol))
> colnames(mytab) <- c('favcol', 'frequency')
> mytab$percent <- 100*mytab$frequency/
sum(mytab$frequency)
```

and continue with plots as before.

- **Summary Statistics, Histograms, and Box Plots:** If `> tvhours = c(3, 1, 0, 0, 2, 1, 3)` then `> summary(tvhours)` gives the 5-number summary and the mean. `mean, median, sd, IQR` are individual R commands used as in `> mean(tvhours)`. `> hist(tvhours)` and `> boxplot(tvhours)` give the corresponding plots.

R Chapter 3

Check out the sample R scripts posted on ArtofStat.com (go to the tab titled R Code), which replicate the analysis from many examples in this chapter. Here is just a short summary.

- **Contingency Tables and Bar Graphs:** To read in a 2×2 contingency table, create a matrix and give it row and column names:

```
> counts <- c(29, 98, 19485, 7086)
> mytable <- matrix(counts, nrow=2, ncol=2,
byrow=TRUE, dimnames=list('Food Type'
= c('Organic', 'Conventional'), 'Pesticides'
= c('Present', 'Absent')))
> mytable
             Pesticides
Food Type      Present Absent
  Organic           29     98
  Conventional   19485   7086
```

Then, find the conditional proportions for pesticide status through

```
> cond.props <- prop.table(mytable,1)
> cond.props
             Pesticides
Food Type        Present    Absent
  Organic      0.2283465 0.7716535
  Conventional 0.7333183 0.2666817
```

Plot the ones for the presence of pesticides in a bar graph like this:

```
> barplot(cond.props[,1], xlab='Food Type',
ylab='Proportion', ylim=c(0,1), main=
'Proportion of Food Samples \n with Pesticide
Present', col=c('green4', 'orange2'))
```

- **Scatterplots:** To create a scatterplot in R, provide the *x*- and *y*-variable and use the `plot()` command, illustrated here with the Subway Sandwich data:

```
> protein <- c(19, 36, 16, 19, 20, 23, 25,
29, 18, 23, 24, 25, 19, 18, 9)
> cost <- c(3.89, 4.99, 3.89, 4.39, 3.99,
4.39, 5.09, 4.89, 3.99, 4.89, 4.89, 4.89,
4.39, 4.49, 3.89)

> plot(protein, cost, xlab='Protein Content
(g)', ylab='Cost of Sandwich', main=
'Scatterplot of 15 Sandwiches', col='red')
```

- **Correlation Coefficient *r* and r^2:** To obtain 'the correlation coefficient *r*, use the `cor()` command. Square it to get r^2:

```
> cor(protein, cost)
[1] 0.7824487
> cor(protein, cost)^2
[1] 0.6122259
```

- **Intercept and Slope for Regression Equation:** To obtain the intercept and slope for the linear regression equation, and to overlay the regression line over the scatterplot constructed above, use:

```
> lin.reg <- lm(cost ~ protein)
> lin.reg
Coefficients:
(Intercept)      protein
    3.23228      0.05717
> abline(lin.reg, col='blue')
```

- **Loading .csv Data File into R and Fitting Regression Model:** Now, we illustrate loading a .csv data file (the Internet Use data) into R, creating a scatterplot, overlaying the regression line, and obtaining the intercept and slope. (*Note:* Any text after the # symbol is a comment. It is not part of the R command and only serves to explain what the command is doing. You don't have to type it into R.)

```
> myfile <- file.choose()   #point to where
your .csv file is located
> InternetUse <- read.csv(myfile)  #reads in
the file
> colnames(InternetUse)  #to see the names of
variables in dataset
> attach(InternetUse)  #so you can refer to
variable names
> plot(Internet.Penetration,
Facebook.Penetration, col='red')
>  lin.reg  <-  lm(Facebook.Penetration  ~
Internet.Penetration)
> lin.reg
Coefficients:
         (Intercept)  Internet.Penetration
              7.8980                0.4389
> abline(lin.reg, col='blue')
```

R Chapter 4

- **Random Samples:** The R command `sample` generates random numbers. For instance, `> sample(30, 5)` provides five numbers randomly selected from the numbers 1 through 30. For sampling with replacement, use `> sample (30, 5, replace=TRUE)`.

R Chapter 5

■ **Random Samples:** The R command `sample()` generates random numbers. For instance, `> sample(30, 5)` provides five numbers randomly selected from the numbers 1 through 30. For sampling with replacement, use `> sample(30, 5, replace=TRUE)`. The simple command `> sample(30)` provides a random permutation of the first 30 numbers.

- **Normal Distribution:** The R command `pnorm` finds probabilities under the normal curve. From Example 7,

```
> pnorm(230, mean=330, sd=80)
[1] 0.1056498
```

tells us that the probability of falling below 230 in a normal distribution with mean 330 and standard deviation 80 is 0.1056, or 10.56%. The command `> pnorm(430, mean=330, sd=80, lower=FALSE)` gives the probability of falling above 430. To get the probability of falling within 230 and 430, use `> pnorm(430, mean=330, sd=80) − pnorm(230, mean=330, sd=80)`, or, for the answer to the question in Example 10, `> pnorm(90, mean=83, sd=5) − pnorm(80, mean=83, sd=5)`.

To find a percentile, for instance the 98th percentile for Mensa membership in Example 8, use

```
> qnorm(0.98, mean=100, sd=16)
[1] 132.86
```

to find that the required IQ score is 133. When you don't specify a mean and standard deviation, R automatically assumes a mean of 0 and standard deviation of 1, i.e., the standard normal distribution, so that `> pnorm(1.96)` returns 0.975 or `> qnorm(0.025)` returns −1.96. Finally, to draw a normal distribution in R, let

```
> mu <− 0
> sigma <− 1
> x <− mu + seq(-3.2, 3.2, 0.01)*sigma
> plot(x, dnorm(x, mean=mu, sd=sigma), type='l',
 ylab="")
```

You can set your own values for `mu` and `sigma` in the first two lines.

- **Binomial Distribution:** The R command `dbinom` finds probabilities for the binomial distribution, such as the probability of 2 matches $P(2) = 0.096$ when $n = 3$ (specified using `size`) and $p = 0.2$ (specified using `prob`) in Example 12:

```
> dbinom(2, size=3, prob=0.2)
[1] 0.096
```

To find the probability for the number of successes to be *less than or equal* to 2, use `> pbinom(2, size=3, prob=0.2)`. To draw the binomial distribution in R, such as in Figure 6.13 use

```
> n <- 10
> p <- 0.15

> plot(0:n,dbinom(0:n,size=n,prob=p),type='h',
ylab='Probability')
```

You can set your own values for `n` and `p` in the first two lines.

R Chapter 7

- **Bootstrap:** The `sample` command in R, together with the `replace=TRUE` option can draw samples with replacement, i.e., bootstrap samples. We illustrate with finding the bootstrap distribution for the median as in Activities 4 and 5, using the cereal sugar data.

```
> sugar <- c(11, 18, 5, 14, 12, 1, 10, 16,
0, 12, 14, 7, 9, 6, 3, 15, 4, 4, 3, 11)
> median(sugar) #observed median
[1] 9.5
> bootsamp1 <- sample(sugar, replace=TRUE) #
a bootstrap sample
> bootsamp1
[1] 9  4 11  4 11  0 12 11  3  7  9  7 15
6  5  6 16 11 16  4
> median(bootsamp1) #median of bootstrap sample
[1] 8

> bootsamp2 <- sample(sugar, replace=TRUE) #
another bootstrap sample
> median(bootsamp2) #median of bootstrap sample
[1] 6
> bootmedian <- c() #initializing
> for(i in 1:10000) bootmedian[i] <-
median(sample(sugar, replace=TRUE))
#creating 10,000 bootstrap samples
> head(bootmedian) #first 6 bootstrapped medians
[1]  9.5 12.0 11.5  6.5 11.0 10.0
> summary(bootmedian)
Min.  1st Qu.  Median  Mean  3rd Qu.  Max.
3.00     7.00    9.50  8.96    11.00 14.50
> sd(bootmedian) #bootstrap std. deviation
[1] 2.1797
```

We specified the sugar values and then illustrated how to create a bootstrap sample (`bootsamp1, bootsamp2`). The `for` loop generates a bootstrap sample 10,000 times and finds the median for each, saving it to `bootsmedian`—which we needed to initialize—for each iteration. After the for loop is completed, `bootmedian` contains the 10,000 medians generated. Summary statistics of the 10,000 generated bootstrap medians are shown via `summary` and `sd`. Type `> hist(bootmedian)` to obtain a histogram of the bootstrap distribution. To obtain the bootstrap distribution for the mean or standard deviation, simply replace the `median` command with `mean` or `sd` in the for loop. For carrying out the bootstrap with R for other statistics, such as the correlation, refer to the book's website for examples.

R Chapter 8

- **Confidence Interval for a Proportion:** In R, using data from Example 3, type

```
> x <- 637
> n <- 1361
> phat <- x/n

> se <- sqrt(phat*(1-phat)/n)
> zscore <- qnorm(0.975)
> me <- zscore*se
> phat + c(-1,1)*me
```

pressing return after each, to obtain the 95% confidence interval and its components (e.g., to see the margin of error, type `me` and press return). Alternatively, you can use the `prop.test` function to obtain a confidence interval, with output (edited) shown below:

```
> prop.test(637, 1361, conf.level=0.95,
correct=FALSE)
95 percent confidence interval:
 0.4416559 0.4946004
sample estimates:
        p
0.4680382
```

This is actually the (Wilson) interval from Exercise 8.123, which for large n is the same as the one from Section 8.2. For a 99% confidence interval, use `conf.level=0.99`.

- **Confidence Interval for a Mean:** In R, using data from Example 9, type

```
> x <- c(540, 565, 570, 570, 580, 590, 590,
590, 595, 610, 620)
```

to read in the observations, then type

```
> n <- length(x)
> xbar <- mean(x)
> s <- sd(x)
> se <- s/sqrt(n)
> tscore <- qt(0.975, df=n-1)
> me <- tscore*se
> xbar + c(-1,1)*me
```

to obtain the confidence interval. More conveniently, the `t.test` function returns the same, with output (edited):

```
> x <- c(540, 565, 570, 570, 580, 590, 590,
590, 595, 610, 620)
> t.test(x, conf.level=0.95)
95 percent confidence interval:
 568.7583 598.5144
sample estimates:
mean of x
 583.6364
```

For a 99% confidence interval, use `conf.level=0.99`.

R Chapter 9

- **Test for a Proportion:** The `prop.test` function returns results of a significance test about a proportion. Using the data from Example 4,

```
> prop.test(637, 1353, p=0.5, alternative=
'two-sided', correct=FALSE)

data: 637 out of 1353, null probability 0.5
X-squared = 4.6127, df = 1, p-value = 0.03174
alternative hypothesis: true p is not equal
to 0.5
```

  Here, `p=0.5` specifies the value for the proportion under the null hypothesis. For the test in Example 3, you would use `p=1/3` and also `alternative='greater'` to specify a one-sided test. The output spells out the alternative hypothesis and shows that the P-value is equal to 0.03174 (0.032 rounded), but doesn't show the z test statistic value of -2.15. (Instead, it shows the square of it and labels it X-squared.)

- **Test for a Mean:** The `t.test` function returns results of a significance test about a mean. Illustrating with the small data set from Exercise 9.33, we get the following output:

```
> x <- c(2000, 1000, 3000, 2000)
> t.test(x, mu=1000, alternative='greater')

        One Sample t-test
data:  x
t = 2.4495, df = 3, p-value = 0.04586
alternative hypothesis: true mean is greater
than 1000
```

  Here, `mu=1000` specifies the value of the mean under the null hypothesis. The output spells out the alternative hypothesis tested and shows the value of the test statistic ($t = 2.4495$, or 2.45 rounded), the degrees of freedom ($df = 4 - 1 = 3$) and the P-value (0.04586).

R Chapter 10

- **Comparing two proportions:** The `prop.test` function returns a confidence interval and results of a significance test for the difference of two proportions. Using data from Example 4,

```
> prop.test(c(347, 327), c(11535, 14035),
correct=FALSE)
X-squared = 11.352, df = 1, p-value = 0.0007536
alternative hypothesis: two.sided
95 percent confidence interval:
 0.002790305 0.010776620
```

  returns the 95% confidence interval for $p_1 - p_2$ and the P-value for the two-sided test H_0: $p_1 - p_2 = 0$. (To change the confidence level or type of alternative hypothesis, insert `conf.level = 0.99` or `alternative = 'greater'`, respectively; see Chapters 8 and 9.) As mentioned in Chapter 9, the R output doesn't show the z test statistic value directly (here: $z = 3.37$) but rather its square $(3.37)^2 = 11.35$, labeled X-squared. The P-value is not influenced by this. For comparing proportions from two dependent samples, the `mcnemar.test` function returns results of a significance test, as in

```
> mcnemar.test(matrix(c(1921,  16,  58,  5),
nrow = 2), correct = FALSE)
```

  for data from Example 17. (Again, the function returns the square of the test statistic, $(4.88)^2 = 23.8$, but the P-value is not affected by this.)

- **Comparing two means:** The `t.test` function returns a confidence interval and results of a significance test for the difference of two means, where the user supplies the observations from the samples in the two groups. Using part of the small data set from Exercise 10.31 on the time spent on social media,

```
> males <- c(5, 9, 12, 12, 13, 15, 20)
> females <- c(5, 7, 10, 11, 12, 14, 16, 20,
20, 23, 40)
> t.test(males, females)
t = −1.1404, df = 15.256, p-value = 0.2717
alternative hypothesis: true difference in
means is not equal to 0
95 percent confidence interval:
 −11.167226   3.375018
```

  we obtain a 95% confidence interval for $\mu_1 - \mu_2$ and the P-value for the two-sided test H_0: $\mu_1 - \mu_2 = 0$. You cannot use the `t.test` function when only supplying summary statistics (sample size, mean, and st. dev.). Use the `var.equal=TRUE` option when you want to use the test assuming equal standard deviations. Use the `paired=TRUE` option to analyze paired data, as in (using data from Exercise 10.58):

```
> before <- c(5.08, 5.99, 5.32, 6.03, 5.44)
> after <- c(5.36, 5.98, 5.62, 6.26, 5.68)
> t.test(before, after, paired=TRUE)
  Paired t-test
t = -3.7155, df = 4, p-value = 0.02056
95 percent confidence interval:
  -0.36343134   -0.05256866
sample estimates:
mean of the differences
                 -0.208
```

R Chapter 11

- **Chi-squared distribution:** The `pchisq` function finds (lower-tail) probabilities under the chi-squared distribution, such that `> 1 − pchisq(2.86, df = 2)` returns the upper tail probability `0.239` from the chi-squared distribution with $df = 2$, which is the P-value from Example 5. `qchisq` returns percentiles, such as `> qchisq(0.95, df = 4)` returning `9.49`, the value mentioned in Table 11.6.
- **Chi-squared test:** You can get the X^2 statistic, its associated P-value, expected counts, and standardized residuals from the `chisq.test` function. This function expects a table as input, so if you have raw data, you first need to create a contingency table from them, using the `table` command. We illustrate with the raw data from Example 5, which you can find on the book's website, under the name "HeadacheRaw.csv." (If you already have the data in contingency table format, you can type it into R as shown in the R code for Chapter 3.)

```
> myfile <- file.choose() #path to file
> headache <- read.csv(myfile) #reads in file
> attach(headache) #can reference names
> mytable <- table(drug, outcome) #get table
> mytable
         outcome
drug      mild moderate severe
  active   486    113     16
  placebo  355     80      5
```

```
> mytest <- chisq.test(mytable)
> mytest
 Pearson's Chi-squared test
data:  mytable
X-squared = 2.8601, df = 2, p-value = 0.2393
```

This shows the X^2 statistic and P-value we obtained in Example 5. You can get the expected cell counts by typing `> mytest$expected`, and residuals and standardized residuals through `> mytest$residuals` and `> mytest$stdres`. To carry out the permutation test with 10,000 random permutations mentioned in Section 11.5, use `> chisq.test(mytable, simulate.p.value = TRUE, B = 10000)`.

R Chapter 12

- **Linear Regression:** The `lm` function fits linear regression models and was introduced in the R code for Chapter 3. We illustrate by loading the female athlete data set into R and fitting a linear regression model to replicate the output of Table 12.1:

```
> myfile <- file.choose() #path to file
> athletes <- read.csv(myfile) #reads in file
> head(athletes,3) #check column names
  Athlete BP60 maxBP LP200 maxLP
1       1   10    80    15   295
2       2   12    85    35   330
3       3   20    85    23   410
> myfit <- lm(maxBP ~ BP60, data=athletes)
#fits regression model
> summary(myfit) #replicates Table 12.1
Coefficients:
            Estimate Std. Error t value  Pr(>|t|)
(Intercept) 63.5369      1.9565  32.475   < 2e-16
BP60         1.4911      0.1497   9.958  6.48e-14
  Residual standard error: 8.003 on 55 degrees
    of freedom
  Multiple R-squared: 0.6432,
  Adjusted R-squared: 0.6368
  F-statistic: 99.17 on 1 and 55 DF,
  p-value: 6.481e-14
```

We recognize the columns of Table 12.1: Estimate, standard error, t statistic and P-value, labeled `Pr(>|t|)` in R. Because both P-values are extremely small, R prints them in scientific notation, e.g., `6.48e-14` for the slope, which stands for 6.48×10^{-14}. We also recognize the residual standard deviation `8.003` (R calls it the residual standard error), the r^2 statistic `0.6432`, and the F-statistic `99.17` and P-value `6.481e-14` from Table 12.6.

To obtain the predicted mean of the response at given x-value of 11 and to obtain the confidence interval for the mean, use the `predict` function:

```
> predict(myfit, newdata=data.frame(BP60=11),
interval='confidence', se.fit=TRUE)
$fit
       fit      lwr      upr
1 79.93844 77.81405 82.06283

$se.fit
[1] 1.060051
```

This shows the predicted value `79.9`, the confidence interval's lower bound `77.8` and upper bound `82.1`, and the *se* of `1.06` for the construction of the confidence interval. Compare this to Table 12.5. The prediction interval is

```
> predict(myfit, newdata=data.frame(BP60=11),
interval='prediction')
       fit      lwr      upr
1 79.93844 63.75961 96.11727
```

which is also the one shown in Table 12.5 or the screenshot from the *Linear Regression* app.

You can obtain fitted values, residuals, and residual plots through:

```
> myfitted <- fitted(myfit) #fitted values for
observations in dataset
> myres <- residuals(myfit) #raw residuals
> mystdres <- rstandard(myfit) #standardized
residuals
> head(mystdres,4)
         1          2          3          4
 0.1957575  0.4501844 -1.0692080 -0.7602345
> hist(mystdres) #histogram of std. residuals
> boxplot(mystdres) #box plot of std. residuals
```

Finally, the ANOVA table in Table 12.6 is requested through:

```
> anova(myfit).
Analysis of Variance Table
Response: maxBP
          of Sum sq  Mean sq F value    Pr(>F)
BP60       1  6351.8 6351.8   99.161 6.481e-14
Residuals 55  3522.8   64.1
```

R Chapter 13

- **F Distribution:** The `pf` function finds (lower-tail) probabilities under the *F* distribution, such that `> 1 - pf(40.48, df1=3, df2=60)` returns the upper tail probability `0.000` from the *F* distribution that is reported as the P-value in Example 5. `qf` returns percentiles, such as `> qf(0.95, df1=3, df2=40)` returning `2.84`, as shown in Table 13.8.
- **Multiple Regression:** The `lm` function also fits multiple regression models. It was introduced in the R code for Chapter 3 and expanded on in the R code for Chapter 12. To illustrate multiple regression with two quantitative predictors, we load the Oregon house selling price data set and replicate Table 13.3 about the parameter estimates and their standard errors. (Below, some original output that R produces has been removed to focus on the essentials. We also created simple variable names in the Oregon .csv file for ease of presentation.)

```
> myfile <- file.choose() #path to file
> oregon <- read.csv(myfile) #reads in file
> head(oregon,3) #show the first three rows and column names
   Price Size Acres Lotsize Bedrooms Bathrooms
1 232500 1679  0.23 10018.8        3       1.5
2 470000 4494  0.52 22651.2        5       4.0
3 150000 2542  0.11  4791.6        4       0.0
> myfit <- lm(Price ~ Size + Bedrooms, data=oregon) #fits multiple regression model
> summary(myfit) #provides estimates, standard errors, P-values and R2, compare to Table 13.3
Coefficients:
             Estimate  Std. Error  t value   Pr(>|t|)
(Intercept) 60102.140   18622.905    3.227   0.00146
Size           62.983       4.753   13.250   < 2e-16
Bedrooms    15170.411    5329.806    2.846   0.00489

Residual standard error: 80270 on 197 degrees of freedom
Multiple R-squared:  0.5244, Adjusted R-squared:  0.5196
F-statistic: 108.6 on 2 and 197 DF,  p-value: < 2.2e-16
```

We recognize the elements of Table 13.3 (parameter estimates, standard errors, *t* statistics and P-values, labeled `Pr(>|t|)`) in the output. We also recognize R^2 as `0.5244` (compare to Table 13.5), the residual standard deviation `80270` (R calls it the `Residual standard error`), and the *F* statistic `108.6` for the overall test with corresponding P-value `2.2e-16`, which is essentially 0. We can obtain 95% confidence intervals for each slope parameter through

```
> confint(myfit)
                2.5 %      97.5 %
Size          53.60892    72.35647
Bedrooms     4659.61107 25681.20995
```

We get the predicted price of the first home (see Example 2) and a confidence interval for it via

```
> predict(myfit, newdata=data.frame(Size=1679, Bedrooms=3), interval='confidence')
       fit         lwr         upr
    1 211361.3   197605.3   225117.3
```

You can obtain all fitted values, residuals, and residual plots (histogram and standardized residuals plotted against house size, see Examples 8 and 9) through:

```
> myfitted <- fitted(myfit)  #fitted values for all observations
> myres <- residuals(myfit) #raw residuals
> mystdres <- rstandard(myfit) #standardized residuals
> hist(mystdres) #histogram of std. residuals
> plot(mystdres ~ oregon$Size) #scatterplot of std. residuals vs size
```

- **Categorical Predictors:** To illustrate multiple regression with one categorical predictor, we consider the condition of the house, together with its size. (In our data set, condition had the two categories "Good" and "Bad," and we keep it that way to illustrate how R creates indicator variables automatically. However, in the .csv file, you can replace "Good" with 1 and "Bad" with 0 to create the indicator variables yourself.)

```
> oregon[c(1,2,11), c(1,2,9)] #show rows 1, 2 and 11 and columns 1, 2 and 9 of the oregon data
    Price Size Condition
1  232500 1679      Good
2  470000 4494      Good
11 357500 3189       Bad
> myfit1 <- lm(Price ~ Size + Condition, data=oregon)
> summary(myfit1)
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)   109197.911  19951.982   5.473 1.34e-07
Size             66.463      4.682  14.196  < 2e-16
ConditionGood -12926.940  17196.712  -0.752    0.453
```

This replicates the output from Table 13.11. By default, for categorical variables with two categories, R creates indicator variables following this rule: The category that comes alphabetically first (here: "Bad") is set to 0, the other (here: "Good") is set to 1.

- **Scatterplot and Correlation Matrix:** To obtain the scatterplot matrix, type `> pairs(oregon[, c(1,2,5)])`, where `c(1,2,5)` specifies the column numbers of the variables in the data set you want to plot (e.g., Bedroom is the 5th column in the `oregon` data set). This reproduces Figure 13.1. The correlation matrix is obtained in a similar fashion, using `> cor(oregon[, c(1,2,5)])`. This reproduces the correlation matrix shown in the margin at the beginning of Section 13.2.

R Chapter 14

- **F Distribution:** The `pf` function finds (lower-tail) probabilities under the *F* distribution, such that `> 1 - pf(6.43, df1=2, df2=12)` returns the upper tail probability `0.013` from the *F* distribution that is reported as the P-value in Example 3. `qf`

  returns percentiles, such as `> qf(0.95, df1=2, df2=12)` returning `3.88`, as shown in the table and graph in the margin of Example 3.

- **One-way Analysis of Variance:** To obtain the one-way ANOVA table (Table 14.2) for the telephone holding time data (Example 2), we can use R's `aov` function. (Note the way we enter the data.)

```
> y <- c(5, 1, 11, 2, 8, 0, 1, 4, 6, 3, 13,
9, 8, 15, 7) #all observations
> group <- rep(c('A', 'M', 'C'), times=c(5, 5,
5))
> myAnova <- aov(y ~ group) #creates ANOVA
table
> summary(myAnova)
Analysis of Variance Table
          Df Sum Sq Mean Sq F value  Pr(>F)
group      2  149.2    74.6   6.431 0.01264
Residuals 12  139.2    11.6
```

  This shows the sum of squares, mean squares, the resulting *F* statistic and the P-value of 0.013 we obtained in Example 3 (labeled `Pr(>F)`). With the data set up this way, you can get side-by-side box plots (see the one next to Example 4) through

  `> boxplot(y~group)`. The `times=c(5, 5, 5)` in the definition of `group` tells R to repeat each of the labels 5 times, because we had 5 observations in each group. If we had 3 observations in the first, 7 in the second, and 4 in the third group, we would need to use `times=c(3, 7, 4)`. Alternatively, you can always create a .csv file (with two columns, one for the group labels, the other for the observations), read it into R and provide it to the `aov` function, as we did in the R code for Linear Regression in Chapter 13.

- **Pairwise Comparisons of Means:** To obtain the multiplicity adjusted (Tukey) confidence intervals for all pairwise comparisons of means, such as the ones shown in Table 14.4, use

```
> TukeyHSD(myAnova)
  Tukey multiple comparisons of means
     95% family-wise confidence level
     diff    lwr   upr p adj
C-A   5.0  -0.75 10.75 0.091
M-A  -2.6  -8.35  3.15 0.472
M-C  -7.6 -13.35 -1.85 0.011
```

  This shows the estimated difference and lower and upper confidence bounds. You can visualize these intervals by typing: `> plot(TukeyHSD(myAnova))`

- **Regression and ANOVA:** To obtain the multiple linear regression output from Table 14.6 where R sets up the indicator variables automatically, use

```
> myfit <- lm(y ~ group)
> summary(myfit)
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    5.400      1.523   3.545  0.00403
groupC         5.000      2.154   2.321  0.03868
groupM        -2.600      2.154   -1.207 0.25068
```

  This shows that R uses a different way (compared to MINITAB or other software) to set up indicator variables. It sets the first indicator variable to 1 when the group is Classical (indicated by labeling it `groupC`) and 0 otherwise, and it sets the second indicator variable to 1 when the group was Muzak (indicated by labeling it `groupM`) and 0 otherwise. By default, R sorts the group labels alphabetically and with three labels uses the last two to set up indicator variables, making the first the reference category. Results do **not** depend on how indicator variables are set up, and you will find the same fitted means as given in part b of Example 7. You can get the ANOVA table shown in Table 14.6 via `> anova(myfit)`.

- **Two-way Analysis of Variance:** To replicate Table 14.10 in Example 9 for corn yield, load the data set cornyield.csv from the book's website into R. Using the relationship between regression and ANOVA, we can replicate Table 14.10 through

```
> myfile <- file.choose() #provide path
> corn <- read.csv(myfile) #load file
> head(corn, 3) #show first 3 rows
  fertilizer manure yield
1       high   high  13.7
2       high   high  15.8
3       high   high  13.9
> myfit <- lm(yield ~ fertilizer + manure,
data=corn) #main effects
> anova(myfit)


Analysis of Variance Table
           Df  Sum Sq  Mean Sq   F value  Pr(>F)
fertilizer  1  17.672  17.6720   6.3324  0.02219
manure      1  19.208  19.2080   6.8828  0.01779
Residuals  17  47.442   2.7907
```

  and the regression coefficients for the regression model (Example 10) through

```
> summary(myfit)
Coefficients:
             Estimate Std. Error  t value  Pr(>|t|)
(Intercept)   15.4900     0.6470   23.941  1.55e-14
fertilizerlow -1.8800     0.7471   -2.516    0.0222
manurelow     -1.9600     0.7471   -2.624    0.0178
```

This shows different values for the parameter estimates from the ones reported in Table 14.12, but it's because R sets up indicator variables differently. (For instance, it set the first indicator to 1 when fertilizer is low instead of high, which you can see by the label `fertilizerlow`.) Results do **not** depend on the way indicator variables are set up. Plugging in, you get the same results as in Example 10.7. To investigate the interaction (see Example 11), use

```
> myfit2 <- lm(yield ~ fertilizer + manure  +
fertilizer:manure, data=corn) #fits interaction
> anova(myfit2)
Analysis of Variance Table
                 Df Sum Sq  Mean Sq  F value   Pr(>F)
fertilizer        1 17.672   17.672   6.3683  0.02258
manure            1 19.208   19.208   6.9218  0.01816
fertilizer:manure 1  3.042    3.042   1.0962  0.31066
Residuals        16 44.400    2.775
```

This replicates Table 14.14.