Chapter 2
Quantitative Data

2.1 Introduction

This chapter covers some basic numerical and graphical summaries of data. Different numerical summaries and graphical displays would be appropriate for different types of data. A variable may be classified as one of the following types,

- Quantitative (numeric or integer)
- Ordinal (ordered, like integers)
- Qualitative (categorical, nominal, or factor)

and a data frame may contain several variables of possibly different types. There may be some structure to the data, such as in time series data, which has a time index. In this chapter we present examples of selected numerical and graphical summaries for various types of data. Chapter 3 covers summaries of categorical data in more detail. A natural continuation of Chapters 2 and 3 might be Chapter 5, "Exploratory Data Analysis."

2.2 Bivariate Data: Two Quantitative Variables

Our first example is a bivariate data set with two numeric variables, the body and brain size of mammals. We use it to illustrate some basic statistics, graphics, and operations on the data.

2.2.1 Exploring the data

Body and brain size of mammals

There are many data sets included with the R distribution. A list of the available data sets can be displayed with the `data()` command. `MASS` [50] is one of the *recommended packages* that is bundled with the base R package, so it should already be installed with R. To use the data sets or functions in `MASS` one first loads `MASS` by the command

```
> library(MASS)  #load the package
> data()         #display available datasets
```

After the `MASS` package is loaded, the data sets in `MASS` will be included in the list of available datasets generated by the `data()` command.

Example 2.1 (mammals). In the result of the `data()` command, under the heading *"Data sets in package MASS:"* there is a data set named `mammals`. The command

```
> ?mammals
```

displays information about the `mammals` data. This data contains brain size and body size for 62 mammals. Typing the name of the data set causes the data to be printed at the console. It is rather long, so here we just display the first few observations using `head`.

```
> head(mammals)
                   body brain
Arctic fox        3.385  44.5
Owl monkey        0.480  15.5
Mountain beaver   1.350   8.1
Cow             465.000 423.0
Grey wolf        36.330 119.5
Goat             27.660 115.0
```

This data consists of two numeric variables, `body` and `brain`.

R_x 2.1 *In the display above it is not obvious whether `mammals` is a matrix or a data frame. One way to check whether we have a matrix or a data frame is:*

```
> is.matrix(mammals)
[1] FALSE
> is.data.frame(mammals)
[1] TRUE
```

One could convert `mammals` to a matrix by `as.matrix(mammals)` if a matrix would be required in an analysis.

Some basic statistics and plots

The `summary` method computes a five number summary and mean for each numeric variable in the data frame.

```
> summary(mammals)
      body              brain        
 Min.   :   0.005   Min.   :   0.14  
 1st Qu.:   0.600   1st Qu.:   4.25  
 Median :   3.342   Median :  17.25  
 Mean   : 198.790   Mean   : 283.13  
 3rd Qu.:  48.203   3rd Qu.: 166.00  
 Max.   :6654.000   Max.   :5712.00  
```

If there were any missing data, these would appear in the summary as `NA`. The five number summaries are difficult to interpret because there are some extremely large observations (the means exceed not only the medians but also the third quartiles). This is clear if we view the five number summaries as side-by-side boxplots.

```
> boxplot(mammals)
```

The boxplots are shown in Figure 2.1(a). A scatterplot helps to visualize the relation between two quantitative variables. For a data frame with exactly two numeric variables like `mammals`, a scatterplot is obtained with the default arguments of `plot`.

```
> plot(mammals)
```

The scatterplot, shown in Figure 2.2(a), is not as informative as it could be because of the scale. A log transformation may produce a more informative plot. We display the scatterplot for the full `mammals` data set (on log-log scale) in Figure 2.2(b). (The `log` function computes the natural logarithm.)

```
> plot(log(mammals$body), log(mammals$brain),
+   xlab="log(body)", ylab="log(brain)")
```

In the second `plot` command we have also added descriptive labels for the axes.

℞ **2.2** *We have seen that `mammals` consists of two numeric variables, `body` and `brain`, so the operation `log(mammals)` applies the natural logarithm function to the two numeric variables, returning a data frame of two numeric variables (log(body), log(brain)). This is a convenient shortcut, but of course it would not work if the data frame contained variables for which the logarithm is undefined.*

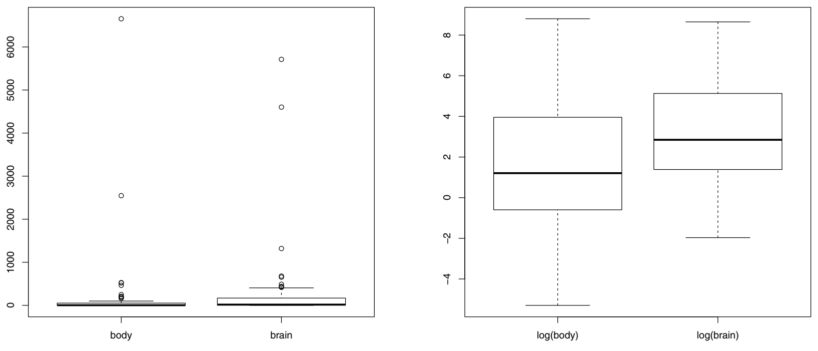
The summaries for the logarithms of body size and brain size are

```
> summary(log(mammals))
      body             brain       
 Min.   :-5.2983   Min.   :-1.966  
 1st Qu.:-0.5203   1st Qu.: 1.442  
 Median : 1.2066   Median : 2.848
 Mean   : 1.3375   Mean   : 3.140
 3rd Qu.: 3.8639   3rd Qu.: 5.111
 Max.   : 8.8030   Max.   : 8.650
```

and the corresponding side-by-side boxplots of the transformed data shown in Figure 2.1(b) are obtained by

```
boxplot(log(mammals), names=c("log(body)", "log(brain)"))
```

The default labels on the boxplots in Figure 2.1(b) would have been the variable names ("body", "brain"), so we added more descriptive labels to the plot with `names`.



(a) (b)

Fig. 2.1 Box plots of mammals brain size vs body size on (a) original and (b) log-log scale in Example 2.1.

2.2.2 Correlation and regression line

In Figure 2.2(b) we can now observe a linear trend; logarithms of body and brain size are positively correlated. We can compute the correlation matrix of the data on the log-log scale by

```
> cor(log(mammals))
          body     brain
body  1.0000000 0.9595748
brain 0.9595748 1.0000000
```

or compute the correlation coefficient by

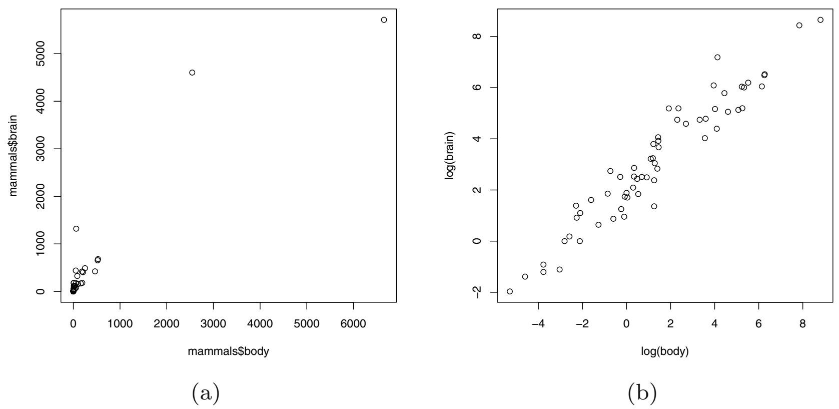


Fig. 2.2 Scatterplots of mammals brain size vs body size on (a) original and (b) log-log scale in Example 2.1.

```
> cor(log(mammals$body), log(mammals$brain))
[1] 0.9595748
```

Simple linear regression is covered in Chapter 7. However, the code to add a fitted straight line to the log-log plot is quite simple. The `lm` function returns the coefficients of the line, and the line can be added to the log-log plot by the `abline` function (see Figure 2.3.)

```
> plot(log(mammals$body), log(mammals$brain),
+   xlab="log(body)", ylab="log(brain)")
> x = log(mammals$body); y = log(mammals$brain)
> abline(lm(y ~ x))
```

℞ 2.3 *A fitted line was added to the scatterplot in Figure 2.3 by the code* `abline(lm(y ~ x))`*.* `lm` *is a function and* `y ~ x` *is a* `formula`*. This is one of many examples where the* `formula` *syntax is used in this book. A* `formula` *can be recognized by the tilde operator* $\sim$*, which connects a dependent variable on the left and predictor variable(s) on the right. Formulas will appear as arguments of some types of plot functions, and in the model specification argument for regression and analysis of variance model fitting. See the* `boxplot` *function that is used to produce Figure 2.4 below for an example of a formula argument in a plotting function.*

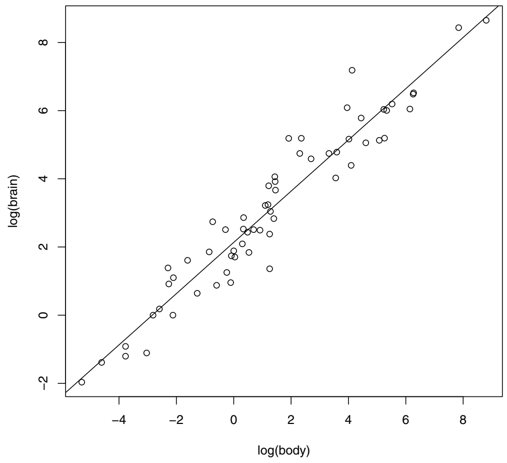


Fig. 2.3 Scatterplot on log-log scale with fitted line in Example 2.1.

2.2.3 Analysis of bivariate data by group

IQ of twins

Example 2.2 (IQ of twins separated near birth). The data file "twinIQ.txt" contains IQ data on identical twins that were separated near birth. The data in the file is also available as a data set in the `UsingR` package [51] as `twins`. There are 27 observations on 3 variables:

Foster	IQ for twin raised with foster parents
Biological	IQ for twin raised with biological parents
Social	Social status of biological parents

The data set, which is shown in Table 2.1, can be imported using

```
> twins = read.table("c:/Rx/twinIQ.txt", header=TRUE)
```

and we display the first few observations with

```
> head(twins)
  Foster Biological Social
1     82         82   high
2     80         90   high
3     88         91   high
4    108        115   high
5    116        115   high
6    117        129   high
```

Next, we compute appropriate numerical summaries for each variable using `summary`.

```
> summary(twins)
     Foster          Biological        Social  
 Min.   : 63.00   Min.   : 68.0   high  : 7  
 1st Qu.: 84.50   1st Qu.: 83.5   low   :14  
 Median : 94.00   Median : 94.0   middle: 6  
 Mean   : 95.11   Mean   : 95.3              
 3rd Qu.:107.50   3rd Qu.:104.5              
 Max.   :132.00   Max.   :131.0              
```

The `summary` function displays five number summaries and means for the two IQ scores, which are numeric, and a frequency table for the factor, `Social`. The five number summaries of the IQ scores are very similar.

Table 2.1 IQ of twins separated near birth. The data is given in three columns in the file "twinIQ.txt".

Foster	Biological	Social	Foster	Biological	Social	Foster	Biological	Social
82	82	high	71	78	middle	63	68	low
80	90	high	75	79	middle	77	73	low
88	91	high	93	82	middle	86	81	low
108	115	high	95	97	middle	83	85	low
116	115	high	88	100	middle	93	87	low
117	129	high	111	107	middle	97	87	low
132	131	high				87	93	low
						94	94	low
						96	95	low
						112	97	low
						113	97	low
						106	103	low
						107	106	low
						98	111	low

We can display side-by-side boxplots of the difference in IQ scores by social status for comparison, using the formula `Foster - Biological` $\sim$ `Social`.

```
> boxplot(Foster - Biological ~ Social, twins)
```

The boxplot shown in Figure 2.4 suggests that there could be differences in IQ for twins raised separately, but it is not clear whether the differences are significant. Another way to view this data is in a scatterplot, with the social status indicated by plotting character or color. This type of plot can be displayed by creating an integer variable for `Social` and using it to select the plotting characters (`pch`) and colors (`col`).

```
> status = as.integer(Social)
> status
 [1] 1 1 1 1 1 1 1 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2
> plot(Foster ~ Biological, data=twins, pch=status, col=status)
```

The scatterplot is shown in Figure 2.5. Note that the levels of the factor `Social` are converted in alphabetical order: high=1, low=2, middle=3. To add the legend to the plot we used

```
> legend("topleft", c("high","low","middle"),
+   pch=1:3, col=1:3, inset=.02)
```

(On a color display, the symbols for *high*, *low*, and *medium* appear in colors black, red, and green, respectively.) To add the line Foster=Biological to the plot, we used `abline` with intercept 0 and slope 1:

```
> abline(0, 1)
```

Figure 2.5 does not reveal any dramatic differences by social status, although the high social status group may correspond to higher IQ scores for twins with their biological parents.

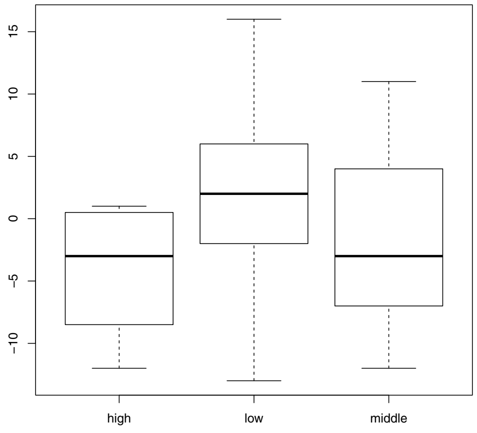


Fig. 2.4 Boxplots of the differences in twins IQ scores (Foster-Biological) in Example 2.2.

2.2.4 Conditional plots

Continuing with the `twins` data, we illustrate a basic conditional plot displayed with the `coplot` function. Instead of displaying the data in different colors or plotting characters, `coplot` displays several scatterplots, all on the same scale. We set this up with a formula `y` $\sim$ `x | a`, which indicates that the plots of `y` vs `x` should be conditional on variable `a`.

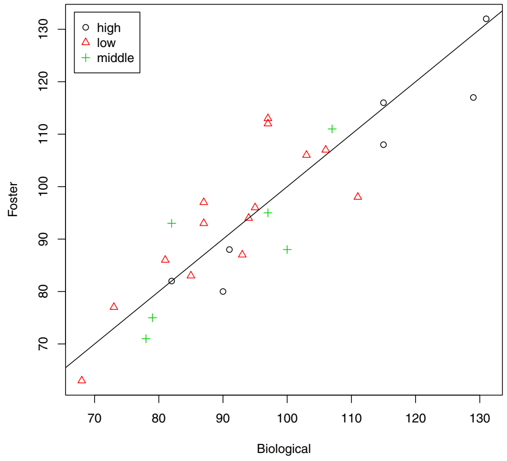


Fig. 2.5 Scatterplot of twins IQ in Example 2.2.

```
> coplot(Foster ~ Biological|Social, data=twins)
```

The coplot is shown in Figure 2.6. The order of the plots is from the bottom and from the left (corresponding to increasing values of `a`, typically). In our coplot, Figure 2.6, that order is: high (lower left), low (lower right), middle (top left), because they are in alphabetical order.

Another version of this type of conditional plot is provided by the function `xyplot` in the `lattice` package. The `lattice` package is included in the R distribution so it should already be installed. To use the function `xyplot` we first load the `lattice` package using `library`. The basic syntax for `xyplot` in this example is

```
    xyplot(Foster ~ Biological|Social, data=twins)
```

The above command displays a conditional plot (not shown) that is similar to the one in Figure 2.7, but with the default plotting character of an open blue circle. To obtain Figure 2.7 we used the following syntax that specifies a solid circle (`pch=20`) in the color black (`col=1`).

```
> library(lattice)
> xyplot(Foster ~ Biological|Social, data=twins, pch=20, col=1)
```

Neither of the conditional plots in Figures 2.6 or 2.7 reveal an obvious pattern or dependence on the conditioning variable social status.

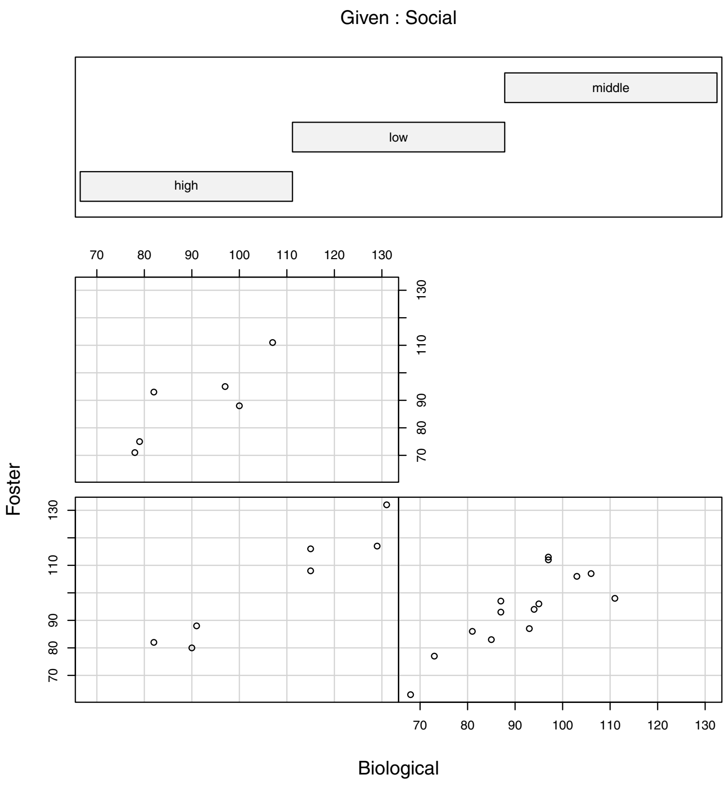


Fig. 2.6 Conditional plot (`coplot`) of twins IQ by social status of biological parents, in Example 2.2.

2.3 Multivariate Data: Several Quantitative Variables

Example 2.3 (Brain size and intelligence).

Data from a study comparing brain size and intelligence is available on the DASL web site [12]. Willerman et al. [56] collected a sample of 40 students' IQ and brain size measured by MRI. There are 8 variables:

Variable	Description
Gender	Male or Female
FSIQ	Full Scale IQ scores based on four Wechsler (1981) subtests
VIQ	Verbal IQ scores based on four Wechsler (1981) subtests
PIQ	Performance IQ scores based on four Wechsler (1981) subtests
Weight	Body weight in pounds
Height	Height in inches
MRI_Count	total pixel Count from the 18 MRI scans

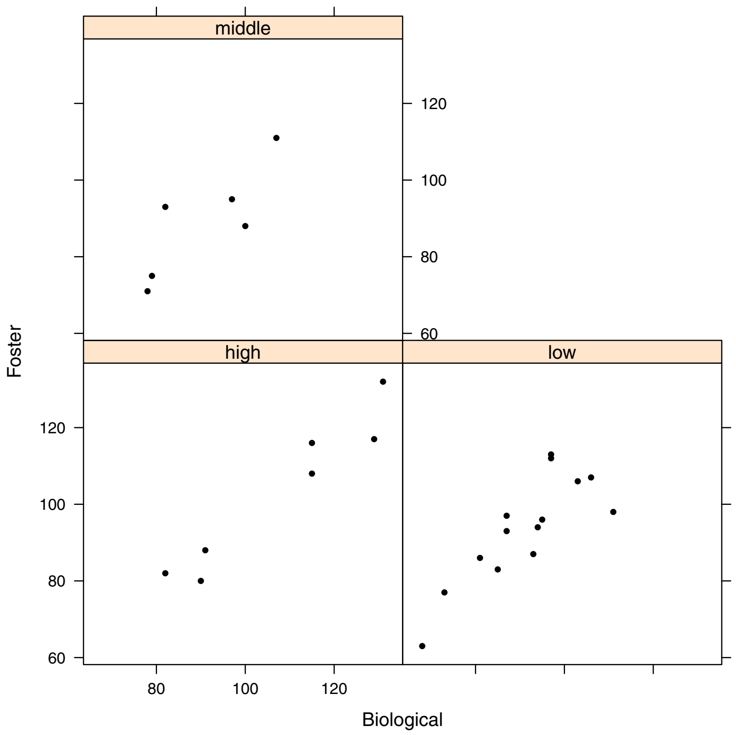


Fig. 2.7 Conditional scatterplot (using `xyplot` in the `lattice` package) of twins IQ by social status of biological parents, in Example 2.2.

2.3.1 Exploring the data

After reading in the data using `read.table` we use the `summary` function to display summary statistics.

```
> brain = read.table("brainsize.txt", header=TRUE)
> summary(brain)
    Gender        FSIQ             VIQ              PIQ
 Female:20   Min.   : 77.00   Min.   : 71.0   Min.   : 72.00
 Male  :20   1st Qu.: 89.75   1st Qu.: 90.0   1st Qu.: 88.25
             Median :116.50   Median :113.0   Median :115.00
             Mean   :113.45   Mean   :112.3   Mean   :111.03
             3rd Qu.:135.50   3rd Qu.:129.8   3rd Qu.:128.00
             Max.   :144.00   Max.   :150.0   Max.   :150.00
```

```
     Weight           Height          MRI_Count
 Min.   :106.0   Min.   :62.00   Min.   : 790619
 1st Qu.:135.2   1st Qu.:66.00   1st Qu.: 855919
 Median :146.5   Median :68.00   Median : 905399
 Mean   :151.1   Mean   :68.53   Mean   : 908755
 3rd Qu.:172.0   3rd Qu.:70.50   3rd Qu.: 950078
 Max.   :192.0   Max.   :77.00   Max.   :1079549
 NA's   :  2.0  NA's   : 1.00
```

The summary function displays appropriate summary statistics for each type of variable; a table is displayed for the categorical variable Gender, and the mean and quartiles are displayed for each numerical type of variable. The summaries show that there are some missing values of Weight and Height.

2.3.2 Missing values

There are several options for computing statistics for data with missing values. Many basic statistics functions such as mean, sd, or cor return missing values when missing values are present in the data. For example,

```
> mean(brain$Weight)
[1] NA
```

The help topic for a particular function explains what options are available. For the mean the optional argument is na.rm, which is FALSE by default. Setting it to TRUE allows the computation of the mean with the missing value(s) removed.

```
> mean(brain$Weight, na.rm=TRUE)
[1] 151.0526
```

This result is the same as the mean reported by summary on page 53.

2.3.3 Summarize by group

There are 20 males and 20 females in this data set, and on average males have larger bodies than females. Larger body size may be related to larger brain size, as we saw in Example 2.1. It may be informative to display statistics separately for males and females, which is easily done using the by function. The basic syntax for the by function is by(data, INDICES, FUN, ...) where the dots indicate possible additional arguments to the function named by FUN. This provides a place for our na.rm=TRUE argument to the mean function. For data we want to include all but the first variable using either brain[, -1] or brain[, 2:7]. This syntax, which omits the row index, indicates that we want all rows of data.

```
> by(data=brain[, -1], INDICES=brain$Gender, FUN=mean, na.rm=TRUE)
brain$Gender: Female
      FSIQ        VIQ        PIQ     Weight     Height  MRI_Count
   111.900    109.450    110.450    137.200     65.765 862654.600
------------------------------------------------------------
brain$Gender: Male
      FSIQ        VIQ        PIQ     Weight     Height    MRI_Count
 115.00000  115.25000  111.60000  166.44444   71.43158 954855.40000
```

As expected, the average weight, height, and MRI count is larger for males than for females.

A way to visualize the MRI counts by gender is to use a different color and/or plotting symbol in a scatterplot. The `plot` command is simpler if we first `attach` the data frame so that the variables can be referenced directly by name. A plot of `MRI_Count` by `Weight` is obtained by

```
> attach(brain)
> gender = as.integer(Gender)  #need integer for plot symbol, color
> plot(Weight, MRI_Count, pch=gender, col=gender)
```

It is helpful to add a legend to identify the symbol and color for each gender. The levels of `Gender` are numbered in alphabetical order when `gender` is created, so 1 indicates “Female” and 2 indicates “Male”.

```
> legend("topleft", c("Female", "Male"), pch=1:2, col=1:2, inset=.02)
```

The plot with the legend is shown in Figure 2.8. In this plot it is easy to see an overall pattern that `MRI_Count` increases with `Weight` and that `MRI_Count` for males tend to be larger than `MRI_Count` for females.

2.3.4 Summarize pairs of variables

A pairs plot displays a scatterplot for each pair of quantitative variables. We want to display scatterplots for all pairs excluding the first variable (gender) in the data frame.

```
> pairs(brain[, 2:7])
```

In the pairs plot shown in Figure 2.9, each of the pairs of IQ plots have points that are clearly clustered in two groups. It appears that perhaps a group of lower IQ and higher IQ individuals were selected for this study. Consulting the online documentation[1] for this data, we find that: “With prior approval of the University’s research review board, students selected for MRI were required to obtain prorated full-scale IQs of greater than 130 or less than 103, and were equally divided by sex and IQ classification.”

[1] http://lib.stat.cmu.edu/DASL/Stories/BrainSizeandIntelligence.html

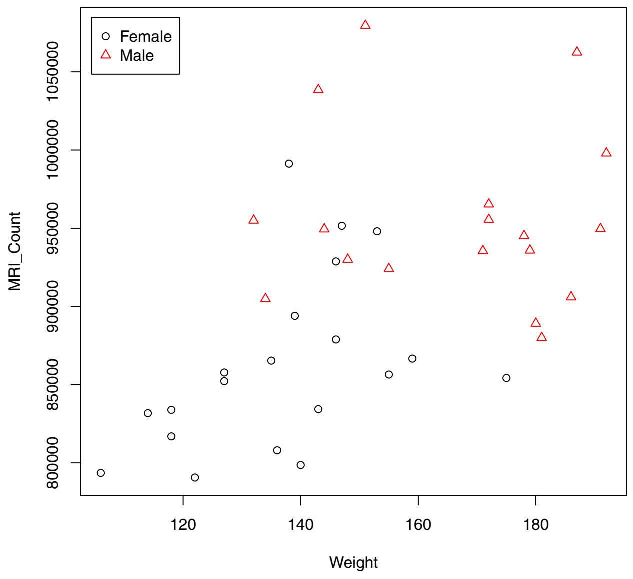


Fig. 2.8 Scatterplot of MRI count vs weight in Example 2.3.

The pairs plot in Figure 2.9 reveals that some variables such as the IQ scores (FSIQ, VIQ, PIQ) have positive correlation. A table of Pearson correlation coefficients can be displayed using the `cor` function; here we round the table to two decimal places.

```
> round(cor(brain[, 2:7]), 2)
          FSIQ  VIQ  PIQ Weight Height MRI_Count
FSIQ      1.00 0.95 0.93     NA     NA      0.36
VIQ       0.95 1.00 0.78     NA     NA      0.34
PIQ       0.93 0.78 1.00     NA     NA      0.39
Weight      NA   NA   NA      1     NA        NA
Height      NA   NA   NA     NA      1        NA
MRI_Count 0.36 0.34 0.39     NA     NA      1.00
```

There are strong positive correlations between each of the IQ scores, but many of the correlations could not be computed due to the missing values in the data.

For computing covariances and correlations for data with missing values, there are several options. Here is one possible option that can be specified by the `use` argument in `cor`: if `use="pairwise.complete.obs"` then the correlation or covariance between each pair of variables is computed using all complete pairs of observations on those variables.

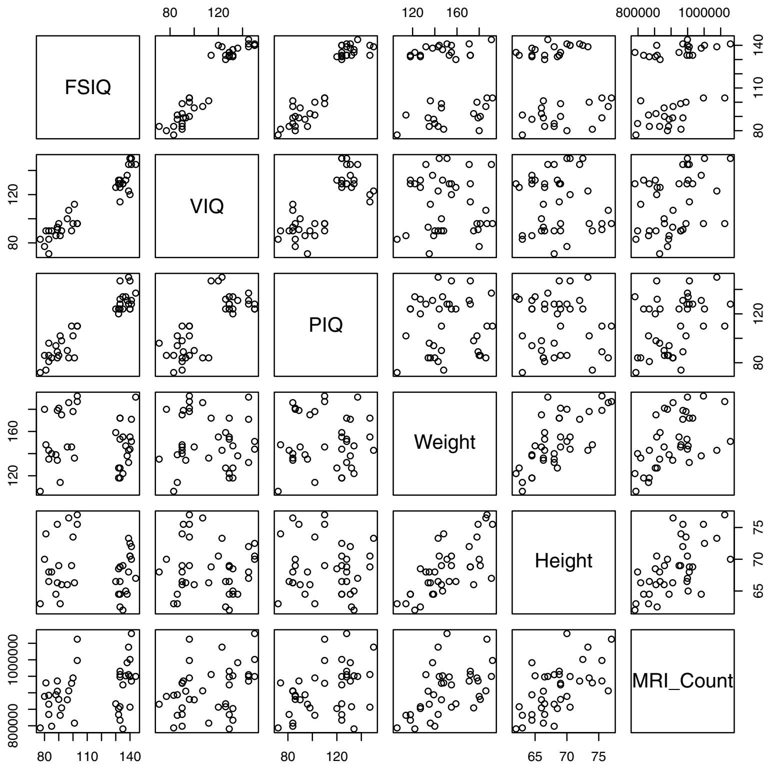


Fig. 2.9 Pairs plot of brain size and IQ measurements in Example 2.3.

```
> round(cor(brain[, 2:7], use="pairwise.complete.obs"), 2)
           FSIQ   VIQ   PIQ Weight Height MRI_Count
FSIQ       1.00  0.95  0.93  -0.05  -0.09      0.36
VIQ        0.95  1.00  0.78  -0.08  -0.07      0.34
PIQ        0.93  0.78  1.00   0.00  -0.08      0.39
Weight    -0.05 -0.08  0.00   1.00   0.70      0.51
Height    -0.09 -0.07 -0.08   0.70   1.00      0.60
MRI_Count  0.36  0.34  0.39   0.51   0.60      1.00
```

(For another approach see the function `complete.cases`.)

The pairs plot (Figure 2.9) and the correlation matrix suggest a mild positive association ($r = 0.36$) between brain size and IQ. However, the MRI count is also correlated with body size (weight and height). If we control for body size as measured by say, weight,

```
> mri = MRI_Count / Weight
> cor(FSIQ, mri, use="pairwise.complete.obs")
[1] 0.2353080
```

the sample correlation of `mri` with `FSIQ` ($r = 0.235$) is smaller than the correlation of `MRI_Count` with `FSIQ` ($r = 0.36$).

One could test whether the correlation is significant using the correlation test `cor.test`. Before adjusting for body size, we have

```
> cor.test(FSIQ, MRI_Count)

        Pearson's product-moment correlation

data:  FSIQ and MRI_Count
t = 2.3608, df = 38, p-value = 0.02347
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.05191544 0.60207414
sample estimates:
     cor
0.357641
```

The correlation test is significant at $\alpha = 0.05$. However, if we adjust for body size (using the transformed variable `mri`), the p-value is not significant at $\alpha = 0.05$.

```
> cor.test(FSIQ, mri)$p.value
[1] 0.1549858
```

For most statistical tests in R, the p-value of the test can be extracted like the example above.

2.3.5 Identifying missing values

This data frame (`brain`) has some missing values. The summary on page 53 indicates that there are two missing heights and one missing weight in the data. To identify these observations, we can use `which` and `is.na`.

```
> which(is.na(brain), arr.ind=TRUE)
     row col
[1,]   2   5
[2,]  21   5
[3,]  21   6
```

When using `which` on an object like a data frame or matrix we need `arr.ind=TRUE` to get the row and column numbers. Observations 2 and 21 have missing data in column 5 (height), and observation 21 has missing data in column 6 (weight). The missing observations are in rows 2 and 21, which we can extract by

```
> brain[c(2, 21), ]
   Gender FSIQ VIQ PIQ Weight Height MRI_Count
2    Male  140 150 124     NA   72.5   1001121
21   Male   83  83  86     NA     NA    892420
```

For example, one could replace missing values with the sample mean as follows.

```
> brain[2, 5] = mean(brain$Weight, na.rm=TRUE)
> brain[21, 5:6] = c(mean(brain$Weight, na.rm=TRUE),
+   mean(brain$Height, na.rm=TRUE))
```

The updated rows 2 and 21 in the data frame are

```
> brain[c(2, 22), ]
   Gender FSIQ VIQ PIQ   Weight Height MRI_Count
2    Male  140 150 124 151.0526   72.5   1001121
22   Male   97 107  84 186.0000   76.5    905940
```

2.4 Time Series Data

Example 2.4 (New Haven temperatures). The R data set `nhtemp` contains the average yearly temperatures in degrees Farenheit for New Haven, Connecticut, from 1912 to 1971. This is an example of a *time series*. The temperature variable is indexed by year.

```
> nhtemp
Time Series:
Start = 1912
End = 1971
Frequency = 1
 [1] 49.9 52.3 49.4 51.1 49.4 47.9 49.8 50.9 49.3 51.9 50.8 49.6
[13] 49.3 50.6 48.4 50.7 50.9 50.6 51.5 52.8 51.8 51.1 49.8 50.2
[25] 50.4 51.6 51.8 50.9 48.8 51.7 51.0 50.6 51.7 51.5 52.1 51.3
[37] 51.0 54.0 51.4 52.7 53.1 54.6 52.0 52.0 50.9 52.6 50.2 52.6
[49] 51.6 51.9 50.5 50.9 51.7 51.4 51.7 50.8 51.9 51.8 51.9 53.0
```

To visualize the pattern of temperatures over the years 1912 to 1971, we can easily display a time series plot using the `plot` function. For time series data, `plot` displays a line plot with time on the horizontal axis. The time plot is shown in Figure 2.10.

```
> plot(nhtemp)
```

One may be interested in identifying any trend in mean annual temperatures over the years represented by this data. One way to visualize possible trends is by fitting a smooth curve. One method of fitting a smooth curve is provided by the `lowess` function, which is based on locally-weighted polynomial regression. Below we plot the data again, this time including a more descriptive label for temperature, and add a horizontal reference line through the grand mean using `abline`. The smooth curve is added to the current plot with `lines`. (See Figure 2.11.)

```
> plot(nhtemp, ylab="Mean annual temperatures")
> abline(h = mean(nhtemp))
> lines(lowess(nhtemp))
```

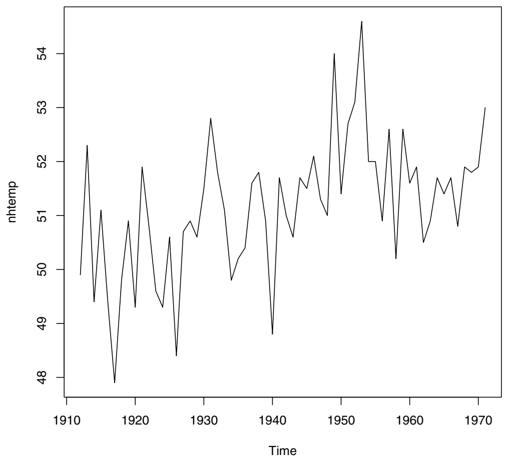


Fig. 2.10 Time plot of mean annual temperatures in New Haven, Connecticut.

For modeling, one often wishes to transform a time series so that the mean is stable over time. When a mean appears to have an approximately linear trend over time as in Figure 2.11, first differences often remove the trend. If $X_1, X_2, \ldots$ is the time series, then we can obtain the time series of first differences $X_2 - X_1, X_3 - X_2, \ldots$ using the `diff` function.

```
> diff(nhtemp)
Time Series:
Start = 1913
End = 1971
Frequency = 1
 [1]  2.4 -2.9  1.7 -1.7 -1.5  1.9  1.1 -1.6  2.6 -1.1 -1.2 -0.3
[13]  1.3 -2.2  2.3  0.2 -0.3  0.9  1.3 -1.0 -0.7 -1.3  0.4  0.2
[25]  1.2  0.2 -0.9 -2.1  2.9 -0.7 -0.4  1.1 -0.2  0.6 -0.8 -0.3
[37]  3.0 -2.6  1.3  0.4  1.5 -2.6  0.0 -1.1  1.7 -2.4  2.4 -1.0
[49]  0.3 -1.4  0.4  0.8 -0.3  0.3 -0.9  1.1 -0.1  0.1  1.1
```

A time plot for the differenced series of temperatures with a reference line through 0 and `lowess` curve is obtained by the code below and shown in Figure 2.12.

```
> d = diff(nhtemp)
> plot(d, ylab="First differences of mean annual temperatures")
> abline(h = 0, lty=3)
> lines(lowess(d))
```

Figure 2.12 suggests that the mean of the differenced series is stable over time; notice that the lowess curve (solid line) is nearly horizontal and very close to the dotted horizontal line through 0.

2.5 Integer Data: Draft Lottery

Example 2.5 (The 1970 Draft Lottery Data).

During the Vietnam War, the Selective Service System of the United States held a draft lottery on December 1, 1969 to determine the order of draft (induction) into the Army. The 366 birthdays (including leap year birthdays) were drawn to determine the order that eligible men would be called into service. Each birthday was matched with a number from 1 to 366 drawn from a barrel. The lowest numbers were drafted first. For an interesting discussion of statistical questions about this lottery see Fienberg [18] and Starr [46].[2]

The data and information about the lottery is available from the Selective Service System web site[3]. We converted it into a tab delimited file, which can be read into an R data frame by

```
> draftnums = read.table("draft-lottery.txt", header=TRUE)
```

This data frame is a table that contains the lottery numbers by day and month. The names of the variables are

```
> names(draftnums)
 [1] "Day" "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul"
 [9] "Aug" "Sep" "Oct" "Nov" "Dec"
```

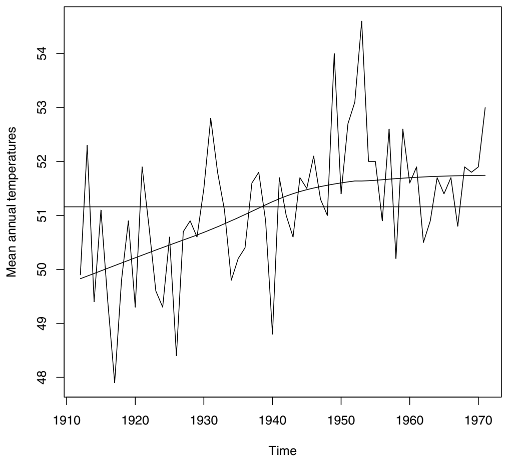


Fig. 2.11 Time plot of mean annual temperatures in New Haven, Connecticut. A horizontal reference line is added to identify the grand mean and a smooth curve is added using `lowess`.

[2] http://www.amstat.org/publications/jse/v5n2/datasets.starr.html

[3] http://www.sss.gov/LOTTER8.HTM

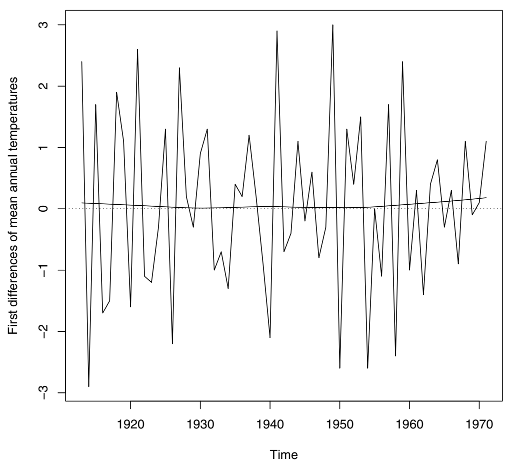


Fig. 2.12 Time plot of first differences of mean annual temperatures in New Haven, Connecticut. A smooth curve is added using `lowess`.

To find the draft number for a January 15 birthday, for example, we read the 15th observation of column "Jan",

```
> draftnums$Jan[15]
[1] 17
```

and find that the corresponding draft number is 17.

Assuming the numbers were drawn randomly, we might expect that the medians of draft numbers for each month were near the number $366/2 = 183$. To display a table of medians for the draft numbers by month we "apply" the median function to the months (columns). The `sapply` function is a 'user-friendly' version of `apply`. However, by default the `median` function returns a missing value if there are missing values in the data.

```
> months = draftnums[2:13]
> sapply(months, median)
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
211  NA 256  NA 226  NA 188 145  NA 201  NA 100
```

R_x **2.4** *The syntax* `draftnums[2:13]` *extracts the second through thirteenth variable from the data frame* `draftnums`. *It would be equivalent to use the syntax* `draftnums[, 2:13]` *or* `draftnums[, -1]`.

Our data has missing values for months with less than 31 days, so we use `na.rm=TRUE` in the `median` function. In `sapply`, the extra argument to `median` is simply listed after the name of the function.

```
> sapply(months, median, na.rm=TRUE)
  Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep
211.0 210.0 256.0 225.0 226.0 207.5 188.0 145.0 168.0
  Oct   Nov   Dec
201.0 131.5 100.0
```

The sample medians by month are less uniformly near 183 than one might expect by chance. A time plot of the medians by month can be obtained as follows.

```
> medians = sapply(months, median, na.rm=TRUE)
> plot(medians, type="b", xlab="month number")
```

In this plot we used `type="b"` to obtain both points and lines, and added a descriptive label "month number" on the horizontal axis. The plot (Figure 2.13) reveals an overall decreasing trend by month.

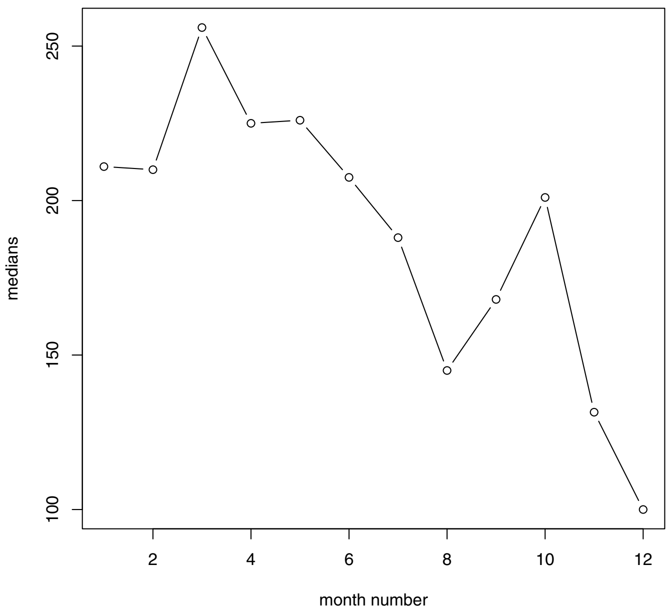


Fig. 2.13 Medians of 1970 draft lottery numbers by month of birthday.

℞ 2.5 *The result of* `sapply` *in Example 2.5 is a vector of medians in the order of the columns (months) January, . . . , December. Then Figure 2.13 is a plot of a vector of data. When the* `plot` *function is used with a single variable, a time plot of the data is displayed, with an index variable shown along the* *horizontal axis. The index values 1 through 12 in this case correspond to the month numbers.*

A boxplot by months of the draft numbers is helpful for comparing the distributions of numbers for birthdays in different months.

```
> months = draftnums[2:13]
> boxplot(months)
```

The parallel boxplots shown in Figure 2.14 look less uniformly distributed across months than we might expect due to chance. The December birthdays appear to have lower draft numbers than birthdays in some other months. In fact, the numbers in the last two months of the year seem to be lower than other months overall. For a possible explanation, more draft lottery data, and further discussion of the lottery, see Starr [46] and "The Vietnam Lotteries" at http://www.sss.gov/lotter1.htm.

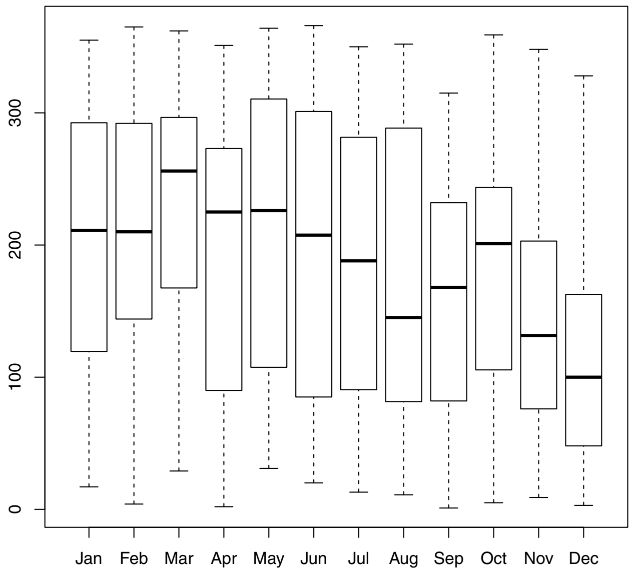


Fig. 2.14 Boxplots comparing 1970 draft lottery numbers by month of birthday.

2.6 Sample Means and the Central Limit Theorem

Example 2.6 (Sample means). The data frame randu contains 400 triples of successive random numbers that were generated using an algorithm called RANDU. In this example we investigate the distribution of the sample mean of each triple of numbers. We know that if the numbers are truly from a continuous Uniform(0, 1) distribution, their expected value is 1/2 and the variance is $1/12 = 0.08333$. First let us compute some basic sample statistics.

```
> mean(randu)
        x         y         z
0.5264293 0.4860531 0.4809547
> var(randu)
             x            y            z
x  0.081231885 -0.004057683  0.004637656
y -0.004057683  0.086270206 -0.005148432
z  0.004637656 -0.005148432  0.077860433
```

Here, because randu is a data frame with three variables, the means are reported separately for each of the three variables, and the var function computes a variance-covariance matrix for the three variables. Each of the three sample means is close to the uniform mean 1/2. The diagonal of the variance-covariance matrix has the three sample variances, which should be close to 0.08333 under the assumption that RANDU generates Uniform(0, 1) data.

```
> diag(var(randu))
         x          y          z
0.08123189 0.08627021 0.07786043
```

The off-diagonal elements in the variance-covariance matrix are the sample covariance, and theoretically the covariances should be zero: the numbers in columns x, y, and z should be uncorrelated if in fact the RANDU table represents independent and identically distributed (iid) numbers. The sample correlations are each close to zero in absolute value:

```
> cor(randu)
            x           y           z
x  1.00000000 -0.04847127  0.05831454
y -0.04847127  1.00000000 -0.06281830
z  0.05831454 -0.06281830  1.00000000
```

Remark 2.1. Although the randu data (x, y, z) have correlations close to zero, in fact there is a linear relation that can be observed if we view the data in a 3-D plot. Try displaying the following plot to see that a pattern can be observed in the data (it is not quite random).

```
    library(lattice)
    cloud(z ~ x + y, data=randu)
```

See Chapter 4 for the plot (Figure 4.19, page 125) and further discussion of the `cloud` function.

We are interested in the distribution of the row means. Each row is assumed to be a random sample of size 3 from the Uniform(0,1) distribution. We can extract the row means using `apply`:

```
> means = apply(randu, MARGIN=1, FUN=mean)
```

Here `MARGIN=1` specifies rows and `FUN=mean` names the function to be applied to the rows. Alternately one could use

```
    rowMeans(randu)
```

to obtain the vector of means. Now `means` is a vector of 400 sample means. We plot a frequency histogram of the sample means using `hist`.

```
> hist(means)
```

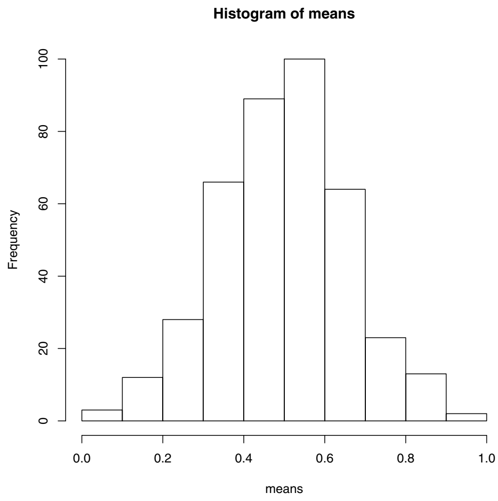


Fig. 2.15 Frequency histogram produced with the default arguments to the `hist` function, for the sample means in Example 2.6.

The histogram of means shown in Figure 2.15 is mound shaped and somewhat symmetric. According to the Central Limit Theorem, the distribution of the sample mean tends to normal as the sample size tends to infinity. To compare our histogram with a normal distribution, however, we need a probability histogram, not a frequency histogram. A probability histogram (not shown) can be displayed by

```
> hist(means, prob=TRUE)
```

A density estimate can be displayed by:

```
> plot(density(means))
```

The density estimate is shown in Figure 2.16. It looks somewhat bell-shaped like a normal distribution.

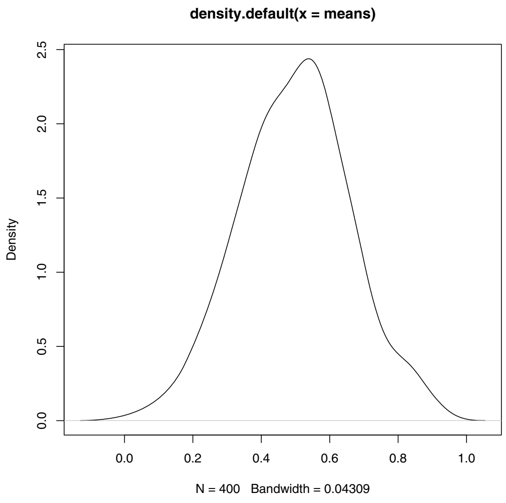


Fig. 2.16 Density estimate for the sample means in Example 2.6.

By default the truehist function in the MASS package displays a probability histogram. We show the result of truehist below in Figure 2.17. Suppose we want to add a normal density to this histogram. The mean should be 1/2 and the variance of the sample mean should be $\frac{1/12}{n} = \frac{1/12}{3} = 1/36$.

```
> truehist(means)
> curve(dnorm(x, 1/2, sd=sqrt(1/36)), add=TRUE)
```

From the histogram and normal density shown in Figure 2.17 one can observe that the distribution of sample means is approaching normality even with a sample size as small as three.

A normal-QQ plot compares the quantiles of a normal distribution with sample quantiles. When the sampled population is normal, the QQ plot should be close to a straight line. The plot and reference line are obtained by

```
> qqnorm(means)
> qqline(means)
```

The normal-QQ plot in Figure 2.18 is consistent with an approximately normal distribution of sample means.

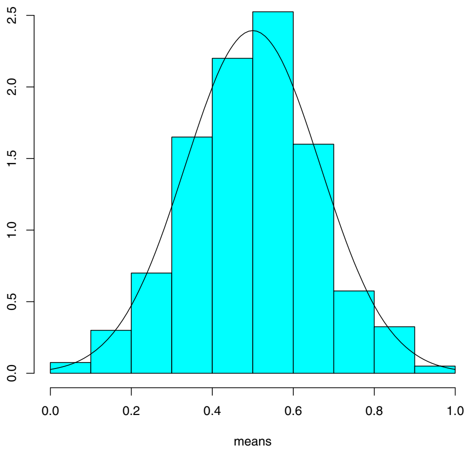


Fig. 2.17 Histogram produced with `truehist` in `MASS` package for the sample means in Example 2.6. The normal density is added with `curve`.

2.7 Special Topics

2.7.1 Adding a new variable

Example 2.7 (mammals, cont.). In Example 2.1, suppose that we wish to create two categories of mammals, large and small. Say a mammal is "large" if body weight is above the median. A factor variable can be added to the data frame to indicate whether the mammal's body weight is above or below the median. We compute the median body size and use the `ifelse` function to assign "large" or "small" levels.

```
> m = median(mammals$body)
> mammals$size = ifelse(mammals$body >= m, "large", "small")
```

It is easy to understand `ifelse`; if the condition is true it returns the first value "large" and otherwise the second value "small".

Rx 2.6 *The code*

```
    mammals$size = ifelse(mammals$body >=m, "large", "small")
```

assigns the "large" or "small" values to the variable `size` in the `mammals` data frame. Since the variable `size` does not yet exist in this data frame, a new variable `size` is created in this data frame.

We use `head` to display the first six rows of the data frame:

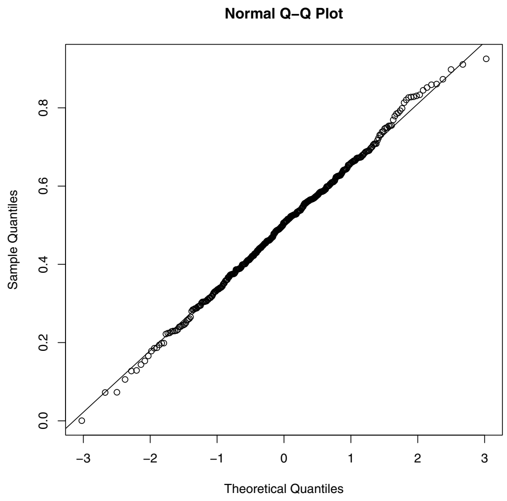


Fig. 2.18 Normal-QQ plot of the sample means in Example 2.6.

```
> head(mammals)
                   body brain  size
Arctic fox        3.385  44.5 large
Owl monkey        0.480  15.5 small
Mountain beaver   1.350   8.1 small
Cow             465.000 423.0 large
Grey wolf        36.330 119.5 large
Goat             27.660 115.0 large
```

The new variable `size` makes it easy to carry out an analysis for the data separately for large and small mammals. For example,

```
  subset(mammals, size=="large")
```

will return a data frame containing the large mammals only. The == operator is logical equality, not assignment.

2.7.2 Which observation is the maximum?

The variables `body` and `brain` can be referenced by `mammals$body` and `mammals$brain`. We can identify observation numbers using the `which` function. For example, here we identify the largest animals.

```
> which(mammals$body > 2000)
[1] 19 33

> mammals[c(19, 33), ]
```

```
                 body brain  size
Asian elephant   2547  4603 large
African elephant 6654  5712 large
```

The `which` function returned the row numbers of the mammals for which `mammals$body > 2000` is `TRUE`. Then we extracted these two rows from the data frame.

The maximum body size is

```
> max(mammals$body)
[1] 6654
```

Suppose that we want to identify the animal with the maximum body size, rather than the maximum value. A function that can be used to identify the largest mammal is `which.max`. It returns the index of the maximum, rather than the value of the maximum. The `which.min` function returns the index of the minimum. We can then use the results to extract the observations with the maximum or the minimum body size.

```
> which.max(mammals$body)
[1] 33

> mammals[33, ]
                 body brain  size
African elephant 6654  5712 large

> which.min(mammals$body)
[1] 14

> mammals[14, ]
                           body brain  size
Lesser short-tailed shrew 0.005  0.14 small
```

The African elephant has the greatest body size of 6654 kg, while the lesser short-tailed shrew has the smallest body size (0.005 kg) in this data set.

2.7.3 Sorting a data frame

Example 2.8 (Sorting mammals). Clearly the mammals are not listed in order of increasing body size. We could `sort` or `rank` the body size variable, but these functions do not help us to order the entire data frame according to body size. To list the mammals in a particular order, we first obtain the ordering using the `order` function. The `order` function will return a sequence of integers that sorts the data in the required order. To see how this works, let us take a small subset of the `mammals` data.

```
> x = mammals[1:5, ]   #the first five
> x
                   body brain  size
Arctic fox        3.385  44.5 large
```

```
Owl monkey         0.480  15.5 small
Mountain beaver    1.350   8.1 small
Cow              465.000 423.0 large
Grey wolf         36.330 119.5 large
```

We want to sort the observations by body size.

```
> o = order(x$body)
> o
[1] 2 3 1 5 4
```

The result of `order` saved in the vector `o` indicates that in order of increasing body size we require observations 2, 3, 1, 5, 4. Finally using the result from `order` we re-index the data. We use the `[row, column]` syntax with `o` for the row and leave the column blank to indicate that we want all columns.

```
> x[o, ]
                    body brain  size
Owl monkey         0.480  15.5 small
Mountain beaver    1.350   8.1 small
Arctic fox         3.385  44.5 large
Grey wolf         36.330 119.5 large
Cow              465.000 423.0 large
```

The code to sort the full mammals data frame by body size is similar:

```
> o = order(mammals$body)
> sorted.data = mammals[o, ]
```

We display the last three observations of the sorted data frame using `tail`, and find the three largest body sizes with their corresponding brain sizes.

```
> tail(sorted.data, 3)
                 body brain  size
Giraffe           529   680 large
Asian elephant   2547  4603 large
African elephant 6654  5712 large
```

2.7.4 Distances between points

In this section we discuss the `dist` function for computing distances between points. We return to the `mammals` data introduced in Example 2.1 and continued in Examples 2.7-2.8, which contains body size and brain size of 62 mammals and a categorical variable `size` that we created in Example 2.7.

Example 2.9 (Distances between points). The original `mammals` data (body, brain) is an example of a bivariate (two-dimensional) data set. Distances between observations are defined only for numeric variables, so we begin by first reloading the data to restore the `mammals` data frame to its original form.

```
> data(mammals)
```

Suppose that we are interested in annotating the plot in Figure 2.2(b) with line segments representing the distances between some of the observations. The Euclidean distance between points $x = (x_1, \dots, x_d)$ and $y = (y_1, \dots, y_d)$ in a d-dimensional space is given by

$$\|x - y\| = \sqrt{\sum_{k=1}^{d} (x_k - y_k)^2}.$$

In two dimensions, as we have here, the distance is just the length of the line segment connecting the two points (the length of the hypotenuse of a right triangle). The distance matrix of a sample contains the distance between the i^{th} and j^{th} observation in row i, column j. A triangular table of distances is returned by `dist`; because the distance matrix is symmetric with zeroes on the diagonal, the `dist` object stores and displays it in a compact way. We illustrate with a small subset of `mammals`.

```
> x = mammals[1:5, ]

> dist(x)
                Arctic fox Owl monkey Mountain beaver        Cow
Owl monkey       29.145137
Mountain beaver  36.456841   7.450966
Cow             596.951136 617.928054      622.184324
Grey wolf        81.916867 110.005557      116.762838 525.233490
```

By default, the `dist` function returns Euclidean distances, but `dist` can optionally compute other types of distances as specified by the `method` argument.

For many applications, we require the full distance matrix. The `as.matrix` function converts a distance object to a matrix.

```
> as.matrix(dist(x))
                Arctic fox Owl monkey Mountain beaver      Cow Grey wolf
Arctic fox         0.00000  29.145137       36.456841 596.9511  81.91687
Owl monkey        29.14514   0.000000        7.450966 617.9281 110.00556
Mountain beaver   36.45684   7.450966        0.000000 622.1843 116.76284
Cow              596.95114 617.928054      622.184324   0.0000 525.23349
Grey wolf         81.91687 110.005557      116.762838 525.2335   0.00000
```

The scatterplot for the full `mammals` data frame (Figure 2.19) was created with this command:

```
> plot(log(mammals$body), log(mammals$brain),
+     xlab="log(body)", ylab="log(brain)")
```

Next, to display a few of the distances, we add line segments corresponding to the distances (cow, wolf), (wolf, human) and (cow, human). First we extract these three observations from `mammals` and store them in `y`. The three points form a triangle, so it is easy to draw the segments between them using `polygon`.

```
> y = log(mammals[c("Grey wolf", "Cow", "Human"), ])
> polygon(y)
```

(To add the segments one at a time, one could use the `segments` function.) Labels are added using `text`. The labels will be centered at coordinates of the points in `y` if we use:

```
    text(y, rownames(y))
```

The placement of the text labels can be adjusted. Here we used

```
> text(y, rownames(y), adj=c(1, .5))
```

for better placement of the labels. Also see Example 4.2 for an interactive method of labeling points using the `identify` function.

From the plot shown in Figure 2.19 we see that if measuring by the Euclidean distances of logarithms of brain and body size, humans are somewhat closer to cows than to wolves. The actual distances are

```
> dist(y)
      Grey wolf      Cow
Cow    2.845566
Human  2.460818 2.314068
```

2.7.5 Quick look at cluster analysis

Example 2.10 (Cluster analysis of distances). Distance matrices are often computed for cluster analysis. Cluster analysis is often applied to reveal possible structure in data. Although an in depth discussion of cluster analysis is beyond the scope of this book, in this section we take a quick look at how to implement a simple cluster analysis. A function that implements hierarchical cluster analysis is `hclust`. For example,

```
> d = dist(log(mammals))
> h = hclust(d, method="complete")
```

performs a hierarchical cluster analysis based on *furthest neighbors.* Using `method="complete"`, beginning with the singletons (individual observations), the two clusters with the smallest maximum pairwise distance are joined at each step. Clearly the distances in Figure 2.19 show that "cow" is not the nearest point to "cat" or to "human". Another widely applied method is *Ward's minimum variance*, obtained with the squared distance matrix and `method=ward` in `hclust`. Several other popular clustering methods are also implemented in `hclust`. A good reference on hierarchical cluster analysis is Everitt, Landau, and Leese [15].

In this example we will work with the largest half of the mammals. We extract the larger half by the `subset` function:

```
> big = subset(mammals, subset=(body > median(body)))
```

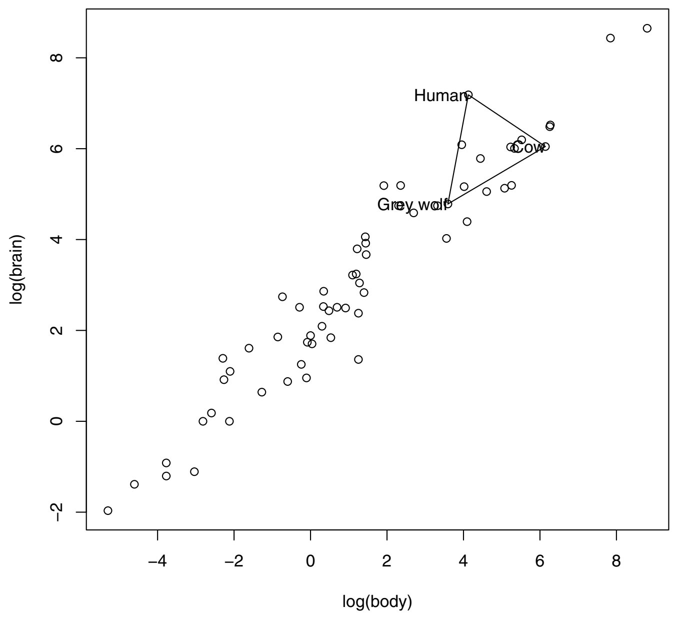


Fig. 2.19 Scatterplot with distances between three of the mammals labeled in Example 2.1.

Then we compute distances and the clustering for the large animals:

```
> d = dist(log(big))
> h = hclust(d, method="complete")
```

The result of `hclust` can be plotted in a tree diagram called a *dendrogram*.

```
> plot(h)
```

The dendrogram is shown in Figure 2.20. The lowest branches to be clustered are the "nearest" according to this algorithm, while nodes that join at a higher level are less alike. For example, we see that the two elephants cluster together early, but are not joined with other clusters until much later. Note that this analysis is based entirely on brain and body sizes, so clusters represent relative size of the animal in some sense.

Let's see which pair of animals are the closest according to this clustering algorithm (the first pair to be merged into a cluster). That would be the pair with the smallest distance. That pair will be identified by the values returned in `h$merge`.

```
> head(h$merge)
     [,1] [,2]
[1,]  -22  -28
[2,]  -25  -57
[3,]  -21  -29
[4,]   -8  -12
[5,]   -9  -62
[6,]  -41  -60
```

We see that the logarithms of the 22nd and 28th observations have the smallest distance and therefore were the first cluster formed. These observations are:

```
> rownames(mammals)[c(22, 28)]
[1] "Horse"   "Giraffe"
```

and their logarithms are

```
> log(mammals)[c(22, 28), ]
            body    brain
Horse   6.255750 6.484635
Giraffe 6.270988 6.522093
```

Note that the cluster analysis will be different if the distances are computed on the original `mammals` data rather than the logarithms of the data, or if a different clustering algorithm is applied.

Exercises

2.1 (*chickwts* data). The `chickwts` data are collected from an experiment to compare the effectiveness of various feed supplements on the growth rate of chickens (see `?chickwts`). The variables are `weight` gained by the chicks, and type of `feed`, a factor. Display side-by-side boxplots of the weights for the six different types of feeds, and interpret.

2.2 (*iris* data). The `iris` data gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of three species of iris. There are four numeric variables corresponding to the sepal and petal measurements and one factor, `Species`. Display a table of means by `Species` (means should be computed separately for each of the three `Species`).

2.3 (*mtcars* data). Display the `mtcars` data included with R and read the documentation using `?mtcars`. Display parallel boxplots of the quantitative variables. Display a pairs plot of the quantitative variables. Does the pairs plot reveal any possible relations between the variables?

2.4 (*mammals* data). Refer to Example 2.7. Create a new variable r equal to the ratio of brain size over body size. Using the full `mammals` data set, order

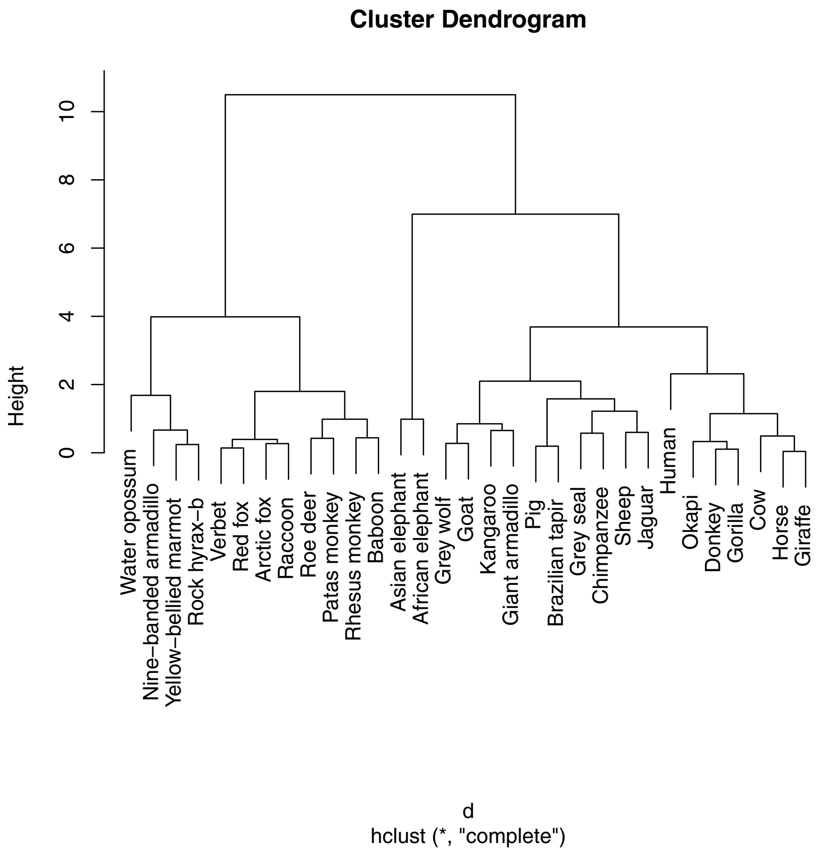


Fig. 2.20 Cluster dendrogram of log(mammals) data by nearest neighbor method in Example 2.1.

the `mammals` data by the ratio r. Which mammals have the largest ratios of brain size to body size? Which mammals have the smallest ratios? (Hint: use `head` and `tail` on the ordered data.)

2.5 (*mammals* data, continued). Refer to Exercise 2.5. Construct a scatterplot of the ratio $r = brain/body$ vs body size for the full `mammals` data set.

2.6 (*LakeHuron* data). The `LakeHuron` data contains annual measurements of the level, in feet, of Lake Huron from 1875 through 1972. Display a time plot of the data. Does the average level of the lake appear to be stable or changing with respect to time? Refer to Example 2.4 for a possible method of transforming this series so that the mean is stable, and plot the resulting series. Does the transformation help to stabilize the mean?

2.7 (Central Limit Theorem with simulated data). Refer to Example 2.6, where we computed sample means for each row of the `randu` data frame. Repeat the analysis, but instead of `randu`, create a matrix of random numbers using `runif`.

2.8 (Central Limit Theorem, continued). Refer to Example 2.6 and Exercise 2.7, where we computed sample means for each row of the data frame. Repeat the analysis in Exercise 2.7, but instead of sample size 3 generate a matrix that is 400 by 10 (sample size 10). Compare the histogram for sample size 3 and sample size 10. What does the Central Limit Theorem tell us about the distribution of the mean as sample size increases?

2.9 (1970 Vietnam draft lottery). What are some possible explanations for the apparent non-random patterns in the 1970 draft lottery numbers in Example 2.5? (See the references.)

2.10 ("Old Faithful" histogram). Use `hist` to display a *probability* histogram of the waiting times for the Old Faithful geyser in the *faithful* data set (see Example A.3). (Use the argument `prob=TRUE` or `freq=FALSE`.)

2.11 ("Old Faithful" density estimate). Use `hist` to display a *probability* histogram of the waiting times for the Old Faithful geyser in the *faithful* data set (see Example A.3) and add a `density` estimate using `lines`.

2.12 (Ordering the *mammals* data by brain size). Refer to Example 2.1. Using the full `mammals` data set, order the data by brain size. Which mammals have the largest brain sizes? Which mammals have the smallest brain sizes?

2.13 (*mammals* data on original scale). Refer to the `mammals` data in Example 2.7. Construct a scatterplot like Figure 2.19 on the original scale (Figure 2.19 is on the log-log scale.) Label the points and distances for cat, cow, and human. In this example, which plot is easier to interpret?

2.14 (*mammals* cluster analysis). Refer to Example 2.10. Repeat the cluster analysis using Ward's minimum variance method instead of nearest neighbor (complete) linkage. Ward's method is implemented in `hclust` with `method="ward"` when the first argument is the *squared* distance matrix. Display a dendrogram and compare the result with the dendrogram for the nearest neighbor method.

2.15 (Identifying groups or clusters). After cluster analysis, one is often interested in identifying groups or clusters in the data. In a hierarchical cluster analysis such as in Example 2.10, this corresponds to *cutting* the dendrogram (e.g. Figure 2.20) at a given level. The `cutree` function is an easy way to find the corresponding groups. For example, in Example 2.10, we saved the result of our complete-linkage clustering in an object `h`. To cut the tree to form five groups we use `cutree` with `k=5`:

```
g = cutree(h, 5)
```

Display `g` to see the labels of each observation. Summarize the group sizes using `table(g)`. There are three clusters that have only one mammal. Use `mammals[g > 2]` to identify which three mammals are singleton clusters.