Andreas Krause, Melvin Olson

The Basics of S and S-PLUS

with fully worked out exercises and solutions

June 13, 1997

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Preface

S-Plus is a powerful tool for interactive data examination, creation of graphs, and implementation of customized routines based on the S language of AT&T Bell Labs (Lucent Technologies). The modern concept together with its flexibility makes it appealing to data analysts from all scientific fields. Among its many strengths are its powerful but flexible graphics routines, and its facility for writing functions whereby you may either modify an existing function, or create one of your own. Most standard statistical techniques and many non-standard ones besides are already programmed in S-Plus, making exploratory and formal statistical analysis very easy.

Our book was written to give a solid but quick introduction to the S-Plus environment. If you have never used S-Plus before, this book will get you up to speed quickly, as it is full of examples and insightful remarks on how S-Plus works. Nevertheless, we start from first principles and make few assumptions about prior knowledge of computing and statistics.

Although our book is primarily intended for the S-Plus novice, the material covered extends to more advanced topics and contains many hints beneficial to those people who already have some knowledge of S-Plus or are not aware of the many enhancements that have been made to the system over the past several years.

The structure of our book, with its detailed exercises and solutions at the end of each chapter, reflects its origin from lecture notes and homework exercises. This is an obvious way to use the book in the future. However, the exercises were kept as an integral part of the book to encourage active thinking on the part of the reader, as opposed to the more passive learning obtained by merely reading it. Comparison of your solutions to ours is strongly encouraged, as there is no single or correct approach to most problems.

The best way to use the book is interactively. That is, you should read the book while seated at your computer and repeat all of the examples as you read them. For getting an overview, reading all the text material should take no more than a day or two, while working through the examples will take some additional days. If used as a basis for lectures, typically one chapter forms the material for a 90-minute lecture, and the exercises repeat and reinforce the information contained in that chapter.

Using this book to lay the foundation, the reader is thus enabled to use S-Plus with all its basic capabilities. With the groundwork in place, it is then possible to tackle the really sophisticated functions and to write your own. If a specific problem is not covered in the book, you should be able to find the answer in one of the voluminous manuals on S-Plus.

This book has undergone many revisions and reflects not only our own experiences, but also those of the many people who influenced it by giving comments and proposing alternative ways to approach the subject. Students of the statistics department at the University of Dortmund, as well as students of the economics department of the University of Basel, were most helpful.

Many more people influenced the various stages of this manuscript and contributed directly or indirectly. They are far too numerous to mention. We would like to thank all of them. Special thanks go to Axel Benner (Heidelberg), Tom Cook (Wisconsin), Peter Hartmann (St. Gallen), Thomas T. Kötter (Berlin), and Tony Rossini (South Carolina).

We also owe a big thanks to the S-Plus developers and distributors. Doug Martin, Brian Clark, and Charlie Roosen at the Data Analysis Products Division of MathSoft, Inc. (formerly Statistical Sciences) in Seattle, and Reinhard Sy at GraS (Berlin), provided generous support and an early beta version of the latest S-Plus release.

The editors at Springer-Verlag (New York), John Kimmel and Martin Gilchrist, provided very helpful comments during a continous good cooperation.

And, last but not least, it is important to mention that this book would not have been possible without the tolerance and support of our wives, Michaela Jahn and Sabine Olson. We dedicate this book to them.

Basel and Geneva
May 1997

Andreas Krause and Melvin Olson

Contents

List of Figures

List of Tables

6. Exploring Data

In the preceding chapters, we have laid the foundation for understanding the concepts and ideas of the S-PLUS system. We explored basic ideas and how to use S-PLUS for performing calculations, and we have seen how data can be generated, stored, and accessed. Furthermore, we also looked at how data can be displayed graphically. All this will be useful as we explore real data sets and learn how to use the existing functionality of S-PLUS in this chapter. We will explore data sets that come with S-PLUS, specifically the Barley and Geyser data sets.

Rather than presenting a list of available statistical functions, we will go through a typical data analysis as a way of introducing the more useful and common commands and the kind of output we'll encounter. We chose to use S-PLUS data sets so you can follow along with the analysis we present and complete the exercises at the end of this chapter.

6.1 Univariate Data Exploration

In this section we will explore the different variables contained in the Barley data set. We will analyze the variables in one dimension, or in other words, univariate. The analysis of the dependence between the variables and the exploration of higher dimensional structure follows later on. We divide the data analysis into the two categories, "descriptive" and "graphical" exploration.

The Barley data set. The Barley data are measurements of yield in bushels per acre at different sites. The analysis comprises 6 sites planting 10 different varieties of barley in two successive years, 1931 and 1932. The data set therefore contains 120 measurements of barley yield. Our main goal will be to determine reasons for the different barley yields given by the different variable constellations, like the 1931 harvest of the fifth variety on site 4, and the 1932 harvest of the seventh variety at the same site.

6.1.1 Descriptive Statistics

Exploratory data analysis (EDA) is an approach to investigating data that stresses the need to know more about the structure and information inherent in the data. The methods used with this approach are referred to as descriptive, as opposed to confirmatory. Descriptive simply means that simple summaries are used to describe the data: their shapes, sizes, relationships, and so on. Examples of descriptive statistics are means, medians, standard deviations, histograms, scatterplots, ranges, and so on.

Given the basic information about the Barley data, the following analysis is intended to gain more information and structural knowledge about the numbers we have.

A typical place to begin is of course looking at the data. If the data set is small, we can easily look at it by simply printing it out. Our data set is almost too large to look at. If you type

```
> barley
```

at the S-PLUS prompt, you will get a display of 120 lines and 4 columns. We randomly pick some rows of the data matrix.

```
> barley[c(2, 17, 64, 70, 82, 98, 118), ]
         yield            variety  year         site
  2   48.86667          Manchuria  1931       Waseca
 17   29.66667           Svansota  1931 Grand Rapids
 64   32.96667          Manchuria  1932    Crookston
 70   26.16667            Glabron  1932    Crookston
 82   32.06666             Velvet  1932    Crookston
 98   44.70000            No. 462  1932       Waseca
118   35.90000  Wisconsin No. 38   1932    Crookston
```

This already shows us what the data look like. The `yield` is the number of bushels per acre, as we know from the data description (have another look at the manual or the online help pages to learn more about it). It is a decimal, and not always an integer. The number of bushels harvested was probably divided by the area of the corresponding plot of barley. The second variable, `variety`, contains a string describing the name of the variety of barley. The `year` is either 1931 or 1932, denoting the year of planting and harvesting. Finally, the `site` variable contains the name of the site from which the data originates.

We can determine this information in a more structured format by using the `summary` function, which summarizes each variable on its own.

```
> summary(barley)
         yield           variety      year              site
     Min.:14.43    Svansota:12  1932:60  Grand Rapids:20
  1st Qu.:26.87     No. 462:12  1931:60       Duluth :20
   Median:32.87   Manchuria:12              Univ Farm :20
     Mean:34.42     No. 475:12                 Morris :20
  3rd Qu.:41.40      Velvet:12              Crookston :20
     Max.:65.77    Peatland:12                 Waseca :20
                    Glabron:12
                    No. 457:12
                  Wisc No.38:12
                      Trebi:12
```

In fact, the `summary` function shows per default up to seven different values for factor variables, such that entering the command as above gives a slightly compressed output. To get exactly what you see above, we used `summary(barley, maxsum=11)`. The argument `maxsum` is the parameter setting the maximum number of rows displayed in the `summary` function.

Now, we investigate the data further. To have easier access to the variables contained in `barley`, we attach the data set.

```
> attach (barley)
```

This gives us direct access to the variables `yield`, `variety`, `year`, and `site`.

How about having a look at our main variable `yield`? Let us determine what the distribution looks like by using the stem and leaf display.

```
> stem (yield)
      N = 120 Median = 32.86667
      Quartiles = 26.85, 41.46666

      Decimal point is 1 place to the right of the colon
      1 : 4
      1 : 579
      2 : 0011122223333
      2 : 5556666666677777778899999999
      3 : 000000111222222233333344444
      3 : 5555667777888899
      4 : 000112223334444
      4 : 567777779999
      5 : 00
      5 : 5889
      6 : 4
      6 : 6
```

The data are ordered and categorized by a base times a power of ten. The stem and leaf display tells us that the data are put into categories 10, 20, 30, .., 60, because “the decimal point is 1 place to the right of the colon.” The lefthand side displays the category (or interval), and the righthand side of the colon has one digit per observation, displayed by the first digit following the category digit. We can immediately see that the smallest values are 14, 15, 17, and 19 (rounded, of course); and by far the largest values are 64 and 66 bushels per acre. Most of the yields are in the high twenties and low thirties, and the distribution looks rather symmetric, having a longer tail toward the larger values (in other words, it is right-skewed).

The stem and leaf plot can be viewed as a sort of sideways histogram from which we can see that the data approximately exhibits the bell-shape typical of a Normal or Gaussian distribution with no extreme values or outliers. As with the histogram, the display sometimes strongly depends on the categorization layout chosen. For glimpsing the distribution and detecting extreme values in the set, the above display is sufficient. However, if we want to determine whether the distribution is approximately bell-shaped like a Normal distribution, we would need to display it with different categorizations or use more sophisticated techniques.

If we want to apply the `stem` display to the other variables of barley, we would get an error message.

```
> stem (variety)
      Error in Ops.ordered(x, 0):
      "*" not meaningful for ordered factors
      Dumped
```

What we realize here is that S-PLUS behaves somewhat intelligently. It “knows” that `variety` is a factor variable with values like “Svansota” and “Manchuria” and that a stem and leaf display for such a variable does not make much sense. Therefore, S-PLUS simply refuses to do the desired display and tells us that our attempt is not meaningful for factors; in fact, the error message comes from an operation inside the `stem` function.

If we want to investigate the skewness of the `yield` variable further, we could use the `quantile` function and calculate the quantiles at 10, 20, 30, ..., 90%.

```
> quantile (yield, seq (0.1, 0.9, by=0.1))
              10%        20%    30%        40%        50%
         22.49667      26.08  28.09   29.94667   32.86667
              60%        70%    80%        90%
         35.13333   38.97333  43.32   47.45666
```

This shows us that the 10% quantile (22.5) is about 10 units away from the median (32.9), whereas the 90% quantile is 47.5, being about 15 bushels per acre away from the median. For a symmetric distribution, the two quantiles would be about the same distance away from the median. A measure of the data spread is the interquartile distance, the difference between the 25% and the 75% quantile.

```
> quantile (yield, c(0.25, 0.75))
             25%    75%
          26.875   41.4
```

This tells us that approximately 50% of the data lie within the interval of 26.9 to 41.4 bushels per acre. Taking the median 32.87, we are not so convinced any more about the symmetry of the distribution of the `yield` variable.

Another measure of spread is the variance, or the standard deviation. S-Plus does not have a built-in function for calculating the standard deviation, so you have to calculate it by taking the square root of the variance.

```
> sqrt (var (yield))
      10.33471
```

We can also use what we have learned about selecting subsets from data. We can calculate some interesting statistics by conditioning on certain values. Let us check if the barley harvest was very different in 1931 and 1932. Selecting only the values of `yield` that were obtained in 1931, we obtain

```
> summary (yield[year==1931])
       Min.  1st Qu.  Median   Mean  3rd Qu.   Max.
       19.7    29.09    34.2  37.08    43.85  65.77
```

and for 1932, we obtain

```
> summary (yield[year==1932])
       Min.  1st Qu.  Median   Mean  3rd Qu.   Max.
      14.43    25.48   30.98  31.76     37.8  58.17
```

As we can see, all the numbers are larger for 1931. The 1932 harvest seems to have been much worse than the 1931 yield, measured in bushels per acre.

We could determine the "most fruitful" sites for both years to see if they were the same in both. A good idea is to select the sites above the 90% quantile for both sites. We get the 90% quantile for both years as

```
> quantile (yield[year==1931], 0.9)
              90%
         49.90334

> quantile (yield[year==1932], 0.9)
              90%
            44.28
```

and retrieve all rows of the Barley data set where the yield in 1931 lies above the 90% quantile.

```
> barley[yield > 49.90334 & year==1931, ]
            yield     variety  year    site
      8  55.20000     Glabron  1931  Waseca
     20  50.23333      Velvet  1931  Waseca
     26  63.83330       Trebi  1931  Waseca
     32  58.10000     No. 457  1931  Waseca
     38  65.76670     No. 462  1931  Waseca
     56  58.80000  Wisc No.38  1931  Waseca
```

```
> barley[yield > 44.28 & year==1932, ]
            yield     variety  year    site
     86  49.23330       Trebi  1932  Waseca
     87  46.63333       Trebi  1932  Morris
     98  44.70000     No. 462  1932  Waseca
     99  47.00000     No. 462  1932  Morris
    116  58.16667  Wisc No.38  1932  Waseca
    117  47.16667  Wisc No.38  1932  Morris
```

Remember what we have learned about selecting data from matrices? In the example above, we query if `yield` is larger than 44.28 and `year` is equal to 1932, giving us a vector of `TRUE/FALSE` elements. These are used as row indices for the matrix `barley`, and the column index is omitted to get all columns. All the rows of `barley` are returned where it is `TRUE` that `yield` is larger than 44.28 and `year` is equal to 1932.

The output above reveals that Waseca seemed to have an incredible harvest in 1931 compared to the other sites. 1932 wasn't that bad for Waseca either, but Morris also had a good year. It turns out that both sites had their big yield in 1932 for the same varieties, Trebi and No. 462. This might lead to further investigation of the surprising observations we just made, but let's continue using more of the S-PLUS functionality.

A useful tool for doing analyses like the ones we just did is the `by` function. It applies functions to data by first splitting the data up into subcategories. We can now calculate the summaries for the Barley data by year, using the `by` function.

```
> by (barley, year, summary)
```

This provides a summary of the whole data set, but in the form of two summary tables, one for 1931 and one for 1932. If we want S-PLUS to display more than just seven lines (because we have ten varieties, as mentioned above), we can add the `maxsum=11` setting as an additional parameter to the `by` function. Similar to the call to `summary` above, where we entered

```
> summary (barley, maxsum=11)
```

we can now enter

```
> by (barley, year, summary, maxsum=11)
```

to obtain the same formatting. In the same way, you could calculate the summary statistics or other figures for each site or each variety separately, without using a loop over the sites or varieties.

You have seen that a combination of only a handful of functions offers many ways of looking at a data set. Table 6.1 summarizes the S-PLUS functions and some of their parameters we have just seen. In addition, it contains a few more functions that might be helpful in other applications.

Table 6.1. Univariate descriptive statistics functions

S-PLUS Function	Description
`quantile`	Quantiles of data
`mean`	(Optionally trimmed) mean of data
`median`	Median of data
`stem`	Stem and leaf display
`var`	Variance of data or, if two variables are supplied, covariance matrix
`by`	Applies a function to data split by indices
`summary`	Summary statistics of an object
`apply, lapply, sapply`	Calculations on rows or columns of matrices and arrays (`apply`), and on components of lists (`lapply`, `sapply`)
`aggregate`	Aggregate data by performing a function (like mean) on subsets

Note If you want to use the `by` function to calculate the mean value of the yield for each year, you would encounter a little difficulty.

```
> by (yield, year, mean)
      Error in as.double:
      Cannot coerce mode list to double: .Data = list(..
      Dumped
```

The problem arises because the `mean` function is not object-oriented. In detail, `by` passes the `barley` elements - which are internally stored as lists - to `mean`, which in turn does not know what to do with them. Entering

```
> mean (list (123))
```

does not work either. A quick solution is to convert the list structure by using `unlist` before calculating the mean. We write a little function

```
> newmean <- function (x) { mean (unlist (x)) }
```

and use our function `newmean` instead.

```
> by (yield, year, newmean)
```

◁

6.1.2 Graphical Exploration

We have gained a basic impression and some insights into the Barley data set structure by studying some summary figures. The next step will lead us into graphical analyses. We will examine some displays of one variable at a time, but as most of the graphics functions reveal their possibilities in a multivariate context, we limit our univariate examination to some selected displays. Table 6.2 lists many of the graphical functions available in S-PLUS.

Table 6.2. Univariate graphical data exploration functions

S-PLUS Function	Description
`boxplot`	Boxplot display
`density`	Returns a density estimate for the distribution of the data. Use `plot(density(x))` to see it graphically
`dotchart`	Dot chart
`hist`	Histogram display
`identify`	Identifies the data point next to the point clicked on in the graphics window
`pie`	Pie chart
`plot`	Standard plot, depending on the data
`qqnorm`	Quantile-quantile plot for the Normal distribution
`qqplot`	Quantile-quantile plot to check how well two data sets overlay
`scatter.smooth`	Plot of the data plus a smoothed regression line

Note: Most of the graphics functions have a long list of typically optional parameters. It is of course possible to use the functions only in their elementary form (like `pie(x)`), but if you want to add colors, modify the labels, explode one of the pies in the pie chart, or have different breakpoints for the histogram, you need to know what you can do. The `boxplot` function, for example, currently has 27 optional parameters. Have a look at the documentation to learn more about all the possibilities offered.

Figure 6.1 shows a simple plot of the four variables in the Barley data set. We split the screen into a 2x2 layout and plot each variable.

```
> par (mfrow=c(2, 2))              # 2x2 layout
> plot (yield)
> plot (variety)
> plot (year)
> plot (site)[1]
```

[1] The variable labels are actually too long to fit. They were modified by the following commands (we will discuss the use of control characters in Section 8.4):

```
> levels (barley[[4]])[1] <- "Grand\nRapids"
> levels (barley[[4]])[3] <- "Univ. Farm"
```

Note that the `plot` function behaves differently for different types of data. The yield data are simply plotted against an index vector from 1 to 120 (the number of observations), and the factor variables `variety`, `year`, and `site` get a bar for each of their possible values. The length of the bar is determined by the number of values falling into this category. In this case, all values in the three variables have the same number of observations, showing again that we are dealing with a designed experiment - each combination of variables occurs exactly once.

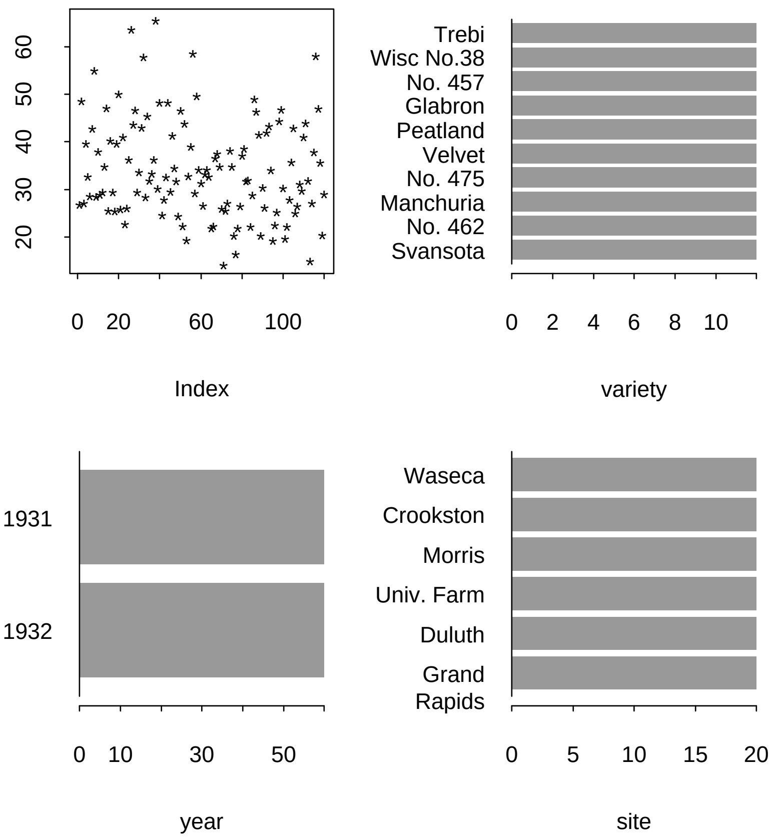

Figure 6.1. A display of the four variables `yield`, `variety`, `year`, and `site` in the `barley` data set.

Investigating the yields for the different sites by year a little further, we create histograms for each combination of `site` and `year`. As there are six sites in the data sets, each having two yields for 1931 and 1932, we obtain 6x2=12 histograms. They show ten values for the ten different varieties planted.

Figure 6.2 shows the histogram display by `year` and `site`.

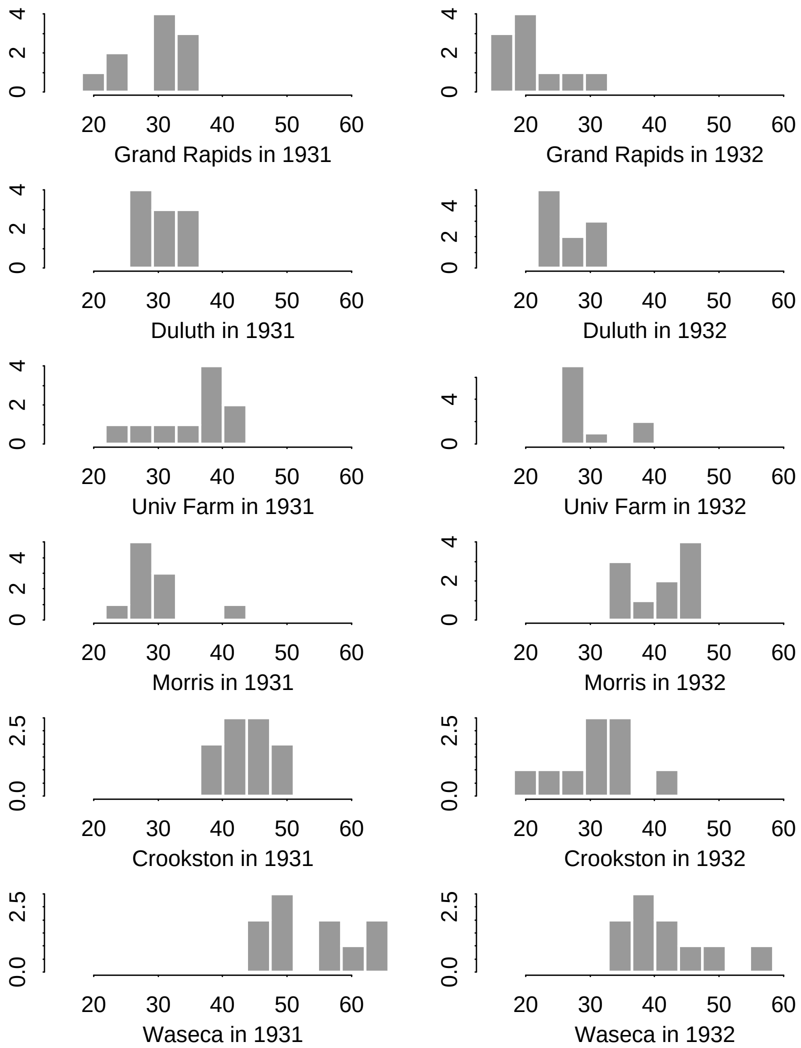

Figure 6.2. The barley yield by `site` and `year`.

The figure was obtained by entering

```
> par (mfcol=c(6, 2))                # multiple figure layout
> yrange <- range (yield)
> limits <- seq (min (yield), max (yield), length=15)
> hist (yield[site=="Grand Rapids" & year==1931],
+ xlim=yrange, breaks=limits, xlab="Grand Rapids in 1931")
```

```
> hist (yield[site=="Duluth" & year==1931], xlim=yrange,
+ breaks=limits, xlab="Duluth in 1931")
> hist (yield[site=="Univ Farm" & year==1931],
+ xlim=yrange, breaks=limits, xlab="Univ Farm in 1931")
> ...
# etc. for the other subsets [2]
```

In this way, we are able to review our data set in a concise form. The histograms can be compared column-wise, observing the yields for the different sites in 1931 and 1932, respectively. If they are viewed row-wise, we can see the different yields for the two years for a given site. More on this follows in Section 6.2.2 about multivariate graphical data analysis.

6.2 Multivariate Data Exploration

In the following section we will analyze the relationships between variables by examining them together or by conditioning one variable on the value of another. We saw how the barley yield differs when we condition on the year 1931 only, or on 1932. In this section, we will explore the relationship in more detail.

Note that we focus on data exploration by using exploratory techniques. Deterministic techniques like multivariate scaling and principal components analysis are left aside for the moment. If you know how these methods work, it will not be difficult to find the appropriate S-Plus functions in the manual and apply them to a specific data set.

So let's continue examining the Barley data.

6.2.1 Descriptive Statistics

As before in the case of a univariate setting, let us put together some descriptive measures. Table 6.3 shows functions useful for multivariate data. The functions in Table 6.3 are used in the following analyses.

To see how a variable is distributed in two dimensions, S-Plus offers the `table` and the `hist2d` function. We can determine how often different combinations of `year` and `site` occur by entering

[2] To be honest, we did not type in these long commands. If you know how to use `for` loops and `paste`, then you might understand how this works. If not, wait until Section 8.1.

```
> xrange <- range (yield)
> limits <- seq (min (yield), max (yield), length=15)
> for (ye in rev (levels (year)))
+  for (si in levels (site))
+   hist (yield[site==si & year==ye], xlim=xrange,
+    breaks=limits, xlab="", xlab=paste(si, "in", ye))
```

Table 6.3. Multivariate descriptive statistics functions

S-PLUS Function	Description
`hist2d`	Tabulation of two-dimensional continuous data by counting the number of occurrences in intervals
`table`	Tabulation of discrete data of any dimension. Counts the number of data equal to the cell values
`crosstabs`	Creation of a contingency table from factor data
`by`	Applying a function to a data set by conditioning on the values of indices
`split`	Split up a data set and get a list with one component per group value
`var`	Calculate the variance or the covariance, if a matrix with more than one column is supplied

```
> table (year, site)
       Grand          Univ
      Rapids  Duluth  Farm  Morris  Crookston  Waseca
  1932    10      10    10      10         10      10
  1931    10      10    10      10         10      10
```

We see that every site had ten plantings in each year. If we look at the three-dimensional table of `year`, `site`, and `variety`, we can see that every combination of `year`, `site`, and `variety` occurs exactly once, as we have ten different varieties. We shorten the output because of its excessive length.

```
> table (year, site, variety)
  ,,Svansota
       Grand          Univ
      Rapids  Duluth  Farm  Morris  Crookston  Waseca
  1932     1       1     1       1          1       1
  1931     1       1     1       1          1       1

  ,,No. 462
       Grand          Univ
      Rapids  Duluth  Farm  Morris  Crookston  Waseca
  1932     1       1     1       1          1       1
  1931     1       1     1       1          1       1
   ...
```

S-PLUS shows the third dimension by “walking through the dimensions.” It keeps the first value of the third dimension’s variable fixed (in this case, the Svansota variety), and shows a two-dimensional table for all data having Svansota as its variety. The next table is shown for the second value for `variety`, here No. 462, and so on, until all third dimension values are taken.

6.6 Exercises

Exercise 6.1

Generate 100 and 1000 random numbers, both samples from a Normal distribution with mean value 3 and variance 5. Draw histograms with bin width 0.5, 1, and 2 for each of the two samples.
Remember that all figures have exactly the same underlying distribution.
Plot them all in a single graphics window and label them accordingly. What is visible ?

Exercise 6.2

Create a vector containing three real numbers, one missing value (`NA`), and one infinite value (`Inf`). Calculate the mean and variance of the non-missing data. Figure out how the missing and infinite data are treated in a simple plot and histogram. What happens if you divide x by itself ? Can you explain this behavior ?

Exercise 6.3

Do an analysis of the Car data set provided with S-PLUS. The data set, stored under `car.all`, contains a set of cars plus a variety of their attributes, like fuel consumption, price, and many more.
Before starting the analysis, determine which car models are contained in the data and what the variables given in the data set are. Extract the fuel consumption (miles per gallon) and determine for each type of car (small, medium, compact, large, van, sporty) the model with the lowest and the highest fuel consumption.
On the basis of average fuel consumption and tank size, calculate the maximum travel distance without a gas refill.
Take the variables maximum travel distance, tank size, mileage, and horsepower, and display them graphically. Try to determine, using only graphics, the two variables with the strongest dependency.
Check your guess by calculating the correlation matrix for these variables.

Exercise 6.4

Test the S-PLUS random number generator with the "dice test." Simulate a number of die rolls and check if they all occur with approximately the same frequency. In other words, we need to simulate a die roll, where the six possible outcomes (1, 2, 3, 4, 5, and 6) occur with the same probability.
Calculate the test statistic $T = \sum_{i=1}^{6} \frac{(n_i - n/6)^2}{n/6}$ which is known to have a Chi-square distribution with 6 degrees of freedom. Use S-PLUS to calculate the critical value for the Chi-square distribution at the 5% level. Can you reject the hypothesis that all numbers occur with the same frequency ?
Repeat this exercise with different numbers of rolls of the die.

Exercise 6.5

In this exercise, we will use simulation to examine a statistical hypothesis test. We want to test the null hypothesis that the mean of a Normally distributed sample is 0, with known variance 1. Let the alternative be that the mean is not equal to 0 (two-sided).
We know that if we set the significance level to α (alpha), the test will reject the null hypothesis in α percent of the times, although the data actually come from a Normal distribution with mean 0 and variance 1. Prove this by generating 1000 samples of size 100, calculating for each sample whether or not the null hypothesis is rejected, and summarizing how often the null hypothesis was rejected.
The other influence factor for a test is the power, the probability of sticking with the null hypothesis although the alternative is correct. Calculate the power of the test for the alternative point hypothesis that the mean is equal to 0.1.

Exercise 6.6

Suppose we had forgotten the formula for computing the area of a circle. All is not lost. We can easily use the computer to approximate a circle's area by simulation. For convenience, we want to determine the area of the unit circle with radius 1. We name this unknown constant Pi (or π).
We know that the area covered by a square with the corner points (-1,1), (1,1), (1,-1) and (-1,-1) equals 4, as the length of each side is 2. This square completely contains the unit circle. If we had a random point somewhere in the square, we could calculate the probability that the point is also inside the unit circle. A random point Z is determined by two coordinates, say X and Y, which have independent uniform distributions on the interval [-1,1].
The probability of Z being inside the circle is given by the ratio of the circle's area and the square's area or, more mathematically, letting P=probability:

P(Z is inside the circle) = (area of the circle)/(area of the rectangle) = Pi/4.

Knowing this, we can estimate (the unknown) Pi by multiplying the estimated probability for Z being inside the circle, by 4.
What's left is to generate a large number of random points Z, calculate the proportion of points inside the circle, and multiply this by 4. This is our estimate of Pi.
Write an S-PLUS program for this problem and estimate Pi with 100, 1000, and 10,000 random points. Try to work with complete vectors only and avoid loops. How far is the approximation away from the Pi we (fortunately) happen to know ?
Hint: You need to know how to check if a point (x,y) is inside the circle. As the circle is defined by its origin and the radius, calculate how far the distance is from the circle's origin to the point. Use Pythagoras's rule to determine the distance between the middle of the circle and a point.

6.7 Solutions

Solution to Exercise 6.1

This exercise shows what different samples coming from the same distribution can look like. If you compare the graphs in the same column, the data are always exactly the same, but the parameter bin width for the histogram is changed. If you compare two graphs in the same row, the difference comes from the sample size (100 versus 1000), and, of course, from stochastic variation. The figure created is shown in Figure 6.15.

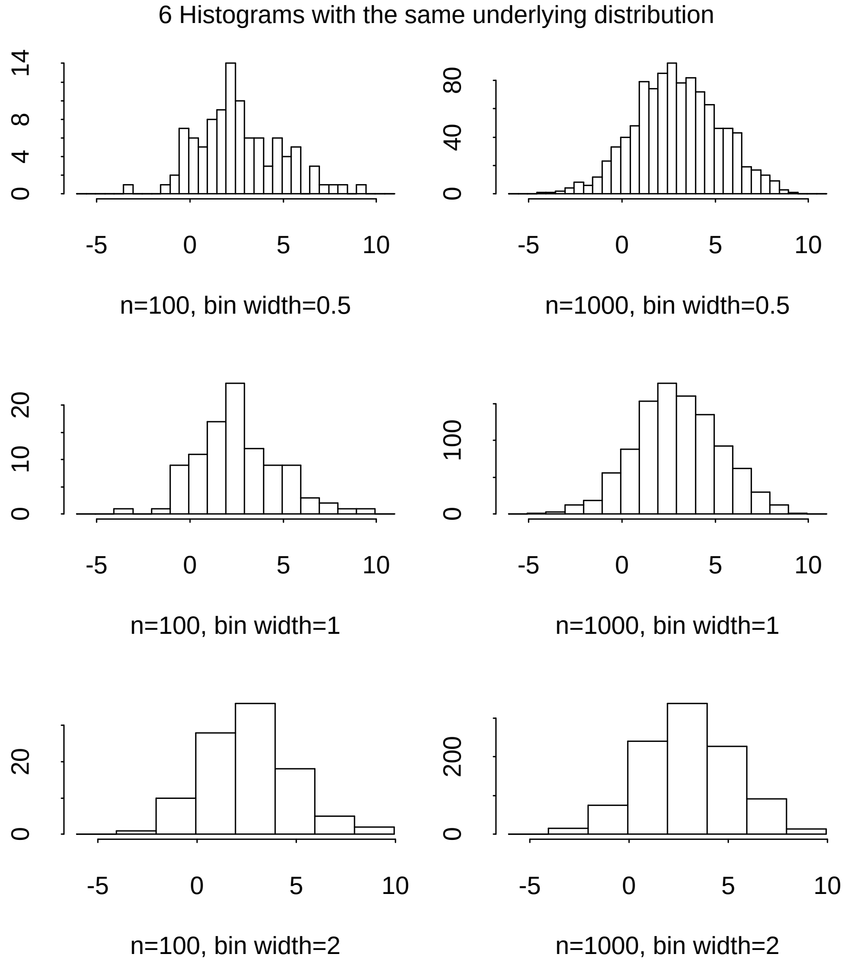

Figure 6.15. Different histogram displays of Normally distributed data.

We start by creating the two data sets.

```
> x <- rnorm (100, 3, sqrt (5))    # generate the data
> y <- rnorm (1000, 3, sqrt (5))
```

We determine the data range and divide the intervals into equally spaced subintervals ranging from the minimum to the maximum value, having different widths for the bins. Note that we need to calculate the *common* minimum and maximum over the two data sets to cover the whole range for both data sets. This generates three sequences of boundaries.

```
> min.xy <- min (x, y)             # determine the range for
> max.xy <- max (x, y)             #  the histogram bounds
> breaks1 <- seq (min.xy, max.xy, 0.5)
> breaks2 <- seq (min.xy, max.xy, 1)
> breaks3 <- seq (min.xy, max.xy, 2)
```

Now we are ready to actually do the histograms using the data we generated above. We change the character size and in particular, we set the histogram color as the background color, which is always the color number 0.

```
> par (mfrow=c(3,2), cex=0.6)      # the page layout
> hist (x,breaks=breaks1,xlab="n=100, bin width=0.5",col=0)
> hist (x,breaks=breaks2,xlab="n=100, bin width=1",  col=0)
> hist (x,breaks=breaks3,xlab="n=100, bin width=2",  col=0)
> hist (y,breaks=breaks1,xlab="n=1000,bin width=0.5",col=0)
> hist (y,breaks=breaks2,xlab="n=1000,bin width=1",  col=0)
> hist (y,breaks=breaks3,xlab="n=1000,bin width=2",  col=0)
```

Finally, we set a title on top of the layout figure using the `mtext` function. Experimenting with the position (the line) of the title shows that the line -1 is a good position to place the title. The side number 3 is the top of the figure, and setting `outer=T` puts the title on top of the overall figure instead of on top of the current (sub-)plot.

```
> title.string <-
+ "6 Histograms with the same underlying distribution"
> mtext (title.string, side=3, outer=T, line=-1)
```

Solution to Exercise 6.2

In this example, we will work with missing data and infinite values, to understand how they are treated and how you can actually handle them in data sets.
We begin by creating a vector by entering it directly from the keyboard.

```
> x <- c(1, 2, NA, 4, 1/0)
```

We could have entered `+Inf` or `-Inf` in the same way. Now let's try to calculate the mean of these data directly. This results in

```
> mean (x)
   NA
```

As we have learned in this section, we can specify a parameter for removing the missing data before doing the calculation:

```
> mean (x, na.rm=T)
   Inf
```

We see that S-PLUS removed the missing value, but not the infinite value. And, of course, the sum of data with an infinite value is infinite (it works vice versa with negative infinite data). To calculate the mean (and variance) over all *real* numbers, we need to remove the infinite value, too.

```
> y <- x[!is.na (x) & !is.inf (x)]
> mean (y)
   2.333333
> var (y)
   2.333333
```

Now let's divide x by itself:

```
> x/x
   1 1 NA 1 NA
```

Note that a missing value divided by anything else results in a missing value again, and infinity divided by infinity also results in a missing value.

Solution to Exercise 6.3

We do the Car data analysis interactively. The first thing we do is attach the data set `car.all`.

```
> attach (car.all)
```

To determine which variables are in the data set, the `names` function delivers the data set's column labels or, in other words, the variable names.

```
> names (car.all)
   "Length" "Wheel.base" "Width" "Height" "Front.Hd."
   "Rear.Hd" "Frt.Leg.Room" "Rear.Seating" "Frt.Shld"
   "RearShld" "Luggage" "Weight" "Tires" "Steering"
   "Turning" "Disp." "HP" "Trans1" "Gear.Ratio"
   "Eng.Rev" "Tank" "Model2" "Dist.n" "Tires2"
   "Pwr.Steer" ".empty." "Disp2" "HP.revs" "Trans2"
   "Gear2" "Eng.Rev2" "Price" "Country" "Reliability"
   "Mileage" "Type"
```

Since we have already attached the data set, we can access the variables just by their names. For example, entering `HP` returns the HP data for all the cars in the data set.

The names of the cars in the data set are returned by the names of the rows of the data matrix. Therefore, we enter

```
> dimnames (car.all)
```

to see the car names and the variable names, or just

```
> dimnames (car.all)[[1]]
```

to get only the first component of the list, the names of the cars. Now we save the names of the cars for later use.

```
> car.names <- dimnames(car.all)[[1]]
```

Next, we calculate the maximum travel distance for all cars.

```
> max.travel <- Mileage*Tank
```

To extract the categories and the car with the longest maximum travel distance for each category, we need to figure out what the categories are and use the `by` function to do the calculation by category. We first figure out that the variable's name for the category of car is `Type.` We can use the `summary` command to view the data structure.

```
> summary (Type)
    Compact  Large  Medium  Small  Sporty  Van  Na's
         19      7      26     22      21   10     6
```

We see that there are only seven cars classified as large in the data, six have no classification, and there are more than twenty cars in the groups Medium, Small, and Sporty.
Determining the cars with the highest fuel consumption (or the lowest mileage) in the different categories is done by using `min` in combination with `by` as follows.

```
> by (Mileage, Type, min, na.rm=T)
     Compact  Large  Medium  Small  Sporty  Van
          21     18      20     25      24   18
```

Note that we need to supply the parameter `na.rm=T` to the `min` function, as there are missing values in the data. The minimum will be missing if any of the data are missing.

In the same fashion, we obtain the maximum values.

```
> by (Mileage, Type, max, na.rm=T)
     Compact  Large  Medium  Small  Sporty  Van
          27     23      23     37      33   20
```

You might have produced slightly different output, because we used the command `unclass(by(...))` to obtain this shortened output.

To analyze the four variables `HP`, `Tank`, `Mileage`, `max.travel`, we create a new data set containing only these data.

```
> car.subset <- data.frame (HP, Tank, Mileage, max.travel)
```

A first display of the data can be generated using `pairs`.

```
> pairs (car.subset)
```

The display of pairwise scatterplots is shown in Figure 6.16. We immediately

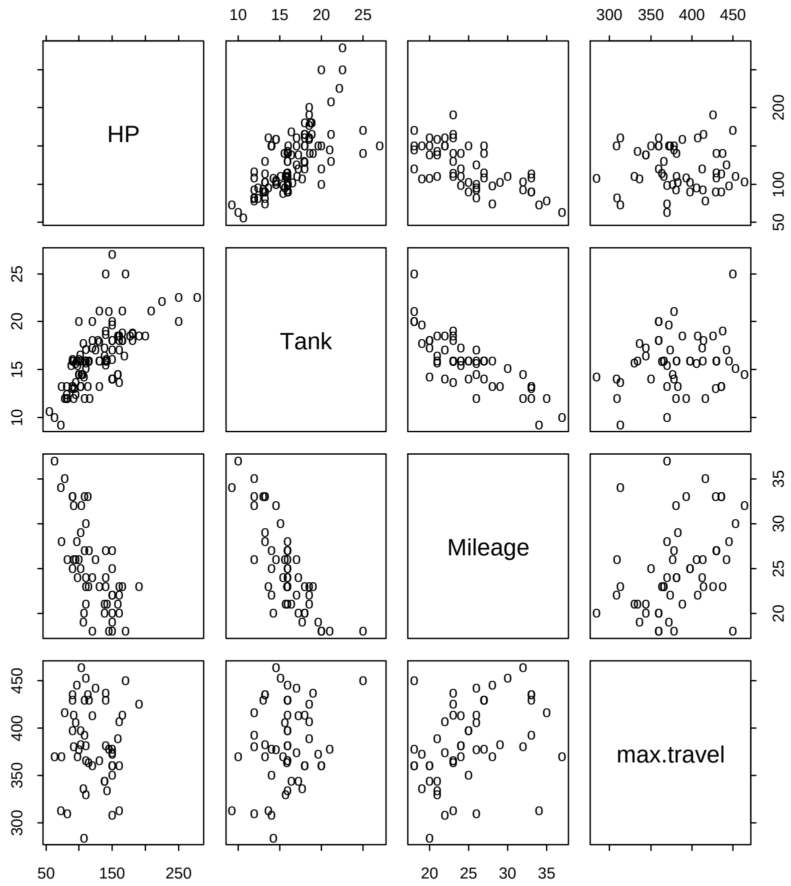

Figure 6.16. Some components of the Car data set (horse power, tank size, mileage, and maximum travel distance).

see strong relationships between many variables, like between HP and tank size or between tank size and mileage in a negative way: the smaller the mileage, the larger the tank, and vice versa.

Confirming this by numbers is a little harder to do because of the missing data contained in the Car data. We want to calculate the correlation between the variables using `cor`, or the variance/covariance matrix using `var`. Neither of the functions accept missing data, which is sensible. Therefore, we need to remove all car data with a missing value in one or more of the four variables. The function `is.na` is useful for checking if any data are missing, and if you remember that `TRUE` values are coded with 1 and `FALSE` values with 0, then it becomes clear what to do. We sum up the `TRUE/FALSE` values obtained from `is.na` and see if the sum is 0 (indicating that none of the data are missing) or greater than 0. The latter data are removed.

```
> problems <- apply (is.na (car.subset), 1, sum) != 0
> car.without.missings <- car.subset [!problems, ]
```

Now we can actually calculate the correlation matrix.

```
> cor (car.without.missings)
                   HP    Mileage       Tank  max.travel
        HP  1.0000001  -0.666153   0.676410  -0.0205656
   Mileage -0.6661530   1.000000  -0.801337   0.3727046
      Tank  0.6764099  -0.801337   1.000000   0.2121667
max.travel -0.0205656   0.372705   0.212167   1.0000000
```

The results of our first guess can be immediately confirmed, as the correlation between HP and tank size is 0.68. There is also a strong negative correlation between tank size and mileage, about -0.8.

Solution to Exercise 6.4

The dice test is done by generating a simulation of die rolls. There are several ways to simulate a die roll. A standard solution is to generate uniform random numbers on the interval from 0 to 6, cut off the digits behind the comma using `trunc`, and add 1 to obtain a sample of numbers from 1 to 6.
We show a different approach using the S-PLUS function `probsamp` to generate a sample from a set of values (sampling with replacement), and the function `table` will count the outcomes for us.

```
> n <- 6000
> rolls <- sample (1:6, n, replace=T)
> rolls.count <- table (rolls)
> rolls.count                       # the tabulated outcome
     1    2    3    4    5    6
   988 1028  988 1062  944  990
> test.stat <- sum ((rolls.count - n/6)^2 / (n/6))
> test.stat                         # the test statistic T
  8.152
```

```
> alpha <- c(0.9, 0.95, 0.99)
> qchisq (alpha, 6)                  # the critical values
    10.64464  12.59159  16.8119
```

As you can see, our test statistic is smaller than all three critical values of the Chi-square distribution, such that we cannot reject the hypothesis that we generated rolls with a fair set of dice.
In fact, the lowest alpha level at which the hypothesis could be rejected (the p-value) is calculated as

```
> 1 - pchisq(test.stat)
```

Solution to Exercise 6.5

You can simulate a statistical test's rejection percentage by generating data from the null hypothesis several times, and by calculating how often the test would reject the null hypothesis, although the data come from the distribution specified in the null hypothesis. In this exercise, we will only work on matrices and vectors, as this is an elegant and efficient way to program in S-PLUS.
We first generate 1000 samples of size 100 from the null hypothesis, being a Normal distribution with mean 0 and standard deviation 1. The samples are stored in a matrix, each row representing one sample.

```
> x <- matrix (rnorm (100000), 1000, 100)
```

Next, we calculate the mean and the standard deviation for each sample separately. Note that this is done without looping.

```
> m <- apply (x, 1, mean)
```

Finally, we need to check whether the test statistic - the mean of the sample - is outside the interval of the $(\alpha/2, 1-\alpha/2)$ quantiles of the test statistic's distribution. Knowing that the variance of a mean of identically independently distributed samples is reduced by the square root of the number of samples, we obtain that the test statistic comes (given the hypothesis) from a N(0, $1/\sqrt{100}) = \mathcal{N}(0, 0.1)$ distribution. As the Normal distribution is symmetric around the mean, we can also check if the sample mean's absolute value is larger than the $1-\alpha/2$-quantile.

```
> z <- abs (m) > qnorm(0.975, 0, 0.1)
```

The result is a vector of `TRUE`/`FALSE` values, and summing them up yields the number of `TRUE` values, which in our case was 47. The mean value is the mean number of rejections over all 1000 samples, which should be around 0.05 (as we are testing on the 5% level).

```
> mean (z)
    0.047
```

You can also check what's in the z variable by tabulating the data.

```
> table (z)
    FALSE  TRUE
      953    47
```

To calculate the approximate power of the test when the data come from the alternative, or, in other words, the number of correct rejections of the null hypothesis, we generate samples from the alternative hypothesis and calculate how often the test rejects the null hypothesis.

```
> x <- matrix (rnorm (100000, 0.1, 1), 1000, 100)
> m <- apply (x, 1, mean)
> z <- abs(m) > qnorm(0.975,0,0.1)# number of rejections
> mean (z)
   0.185
```

We find that if the alternative hypothesis was that the mean is equal to 0.1, the Normal test rejects the wrong hypothesis correctly in only approximately 18.5% of the simulations. Of course the further the alternative is away from the null hypothesis, the better the test in terms of the power.
Another lesson from this is if the significance level (1 - alpha) were increased, the number of correct rejections of the null hypothesis would become even smaller, as the quantile of the Normal distribution on the righthand side of the comparison would become larger. In other words, the lower the type I error (rejecting the hypothesis although it is correct), the higher the type II error (not rejecting the hypothesis although it is incorrect).
A statistical test's power is typically illustrated by calculating many power values for different point alternatives and connecting the points by a curve to obtain the power curve. If you are interested, go ahead and calculate a power curve for this test. Guess what the curve looks like before actually calculating and plotting it.

Solution to Exercise 6.6

We show how to simulate 1000 two-dimensional random numbers in the square [-1,1] x [-1,1] and how to determine for each point if it is inside or outside the circle with radius 1 around the origin (0,0). Finally, we estimate π (Pi) from the proportion of points being inside the circle to the total number of points generated.

```
> n <- 1000
> x <- runif (n, -1, 1)
> y <- runif (n, -1, 1)
```

Determining if a point is inside the unit circle is equal to determining if the distance between the point and the origin of the circle (0,0) is less than 1, as we know that all points inside a circle are at maximum the radius away from the origin of the circle. Calculating the distance between two points in two dimensions is done by using Pythagoras' rule, and the distance between the points $x = (x_1, x_2)$ and $y = (y_1, y_2)$ is given as $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$. We omit the square root, as the inequality is untouched, because the radius on the righthand side is equal to 1.

```
> distance <- x^2 + y^2
> inside.circle <- sum (distance <= 1)
```

Now we estimate the unknown constant π according to the formula derived before.

```
> pi.estimate <- (4*inside.circle)/n
```

The constant `pi` is an S-PLUS constant, such that we can calculate how far we are away from the correct value of π (that we fortunately happen to know).

```
> away.from.pi <- pi - pi.estimate
```

Just to show that it is also possible to do the whole calculation on a single line, here is our one-liner:

```
> pi.est <- (length(runif(n,-1,1)^2+runif(n,-1,1)^2 <= 1)*4)/n
```

In the following table, we show what we obtained as estimates for different numbers of random values. Depending on the current sequence of random numbers, you might get different results. What you should note, nevertheless, is that the deviation from the true Pi gets smaller as n increases.

Table 6.9. Estimating Pi using a simulation technique. Results for different sample sizes

n	Estimate	Error
100	2.96	0.182
1000	3.088	0.0535
10000	3.1532	-0.0116
100,000	3.1366	0.00499

To visualize what we just simulated, plot the square and the circle and add the simulated points to it. The result might look like Figure 6.17. We draw the geometric figures rectangle and circle, and add the points to it. This might look like the following little piece of code.

```
> corners.x <- c(-1,-1, 1, 1, -1)
> corners.y <- c(-1, 1, 1,-1, -1)
> circle.seq <- seq (0, 2*pi, length=1000)
> plot(corners.x,corners.y,type="l",xlab="",ylab="",axes=F)
> axis(1)
> axis(2)
> lines (sin (circle.seq), cos(circle.seq))
> points (x, y)
> title ("Estimation of Pi using simulation")
```

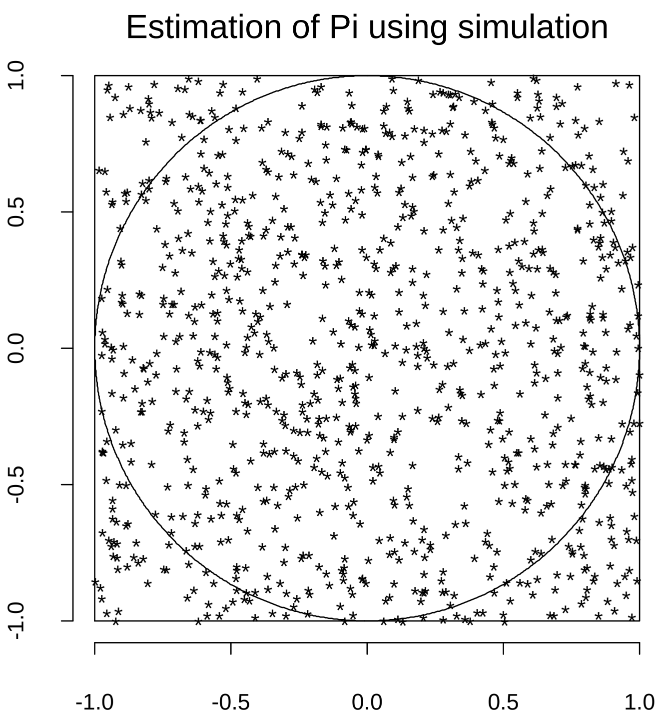

Figure 6.17. Estimating Pi using a simulation technique.