Chapter 4
Presentation Graphics

4.1 Introduction

One of the most attractive aspects of the R system is its capability to produce state-of-the-art statistical graphics. All of the chapters of this book illustrate the application of a variety of R functions for graphing quantitative and categorical data for one or several dimensions. The purpose of this chapter is to describe methods for adjusting the attributes of the graph and for interacting with the graph that will enable the user to produce a publication-level graphical display. We focus on the methods that we have found useful in our own work.

The book by Murrell [37] provides a good description of the traditional graphics system. Sarkar [43] and Wickham [52] provide overviews respectively of the `lattice` and `ggplot2` packages for producing graphics that will be introduced at the end of this chapter.

Example 4.1 (Home run hitting in baseball history).

We begin with an interesting graph that helps us understand the history of professional baseball in the United States. Major League Baseball has been in existence since 1871 and counts of particular baseball events, such as the number of hits, doubles, and home runs, have been recorded for all of the years. One of the most exciting events in a baseball game is a home run (a batted ball typically hit out of the ballpark) and one may be interested in exploring the pattern of home run hitting over the years.

The data file "batting.history.txt" (extracted from the `baseball-reference.com` web site) contains various measures of hitting for each season of professional baseball. We read in the dataset and use the `attach` function to make the variables available for use.

```
> hitting.data = read.table("batting.history.txt", header=TRUE,
+   sep="\t")
> attach(hitting.data)
```

J. Albert and M. Rizzo, *R by Example*, Use R, DOI 10.1007/978-1-4614-1365-3_4,

The variables `Year` and `HR` contain respectively the baseball season and number of home runs per game per team. We construct a time series plot of home runs against season using the `plot` function in Figure 4.1.

```
> plot(Year, HR)
```

Generally we see that home run hitting has increased over time, although there are more subtle patterns of change that we'll notice in the following sections.

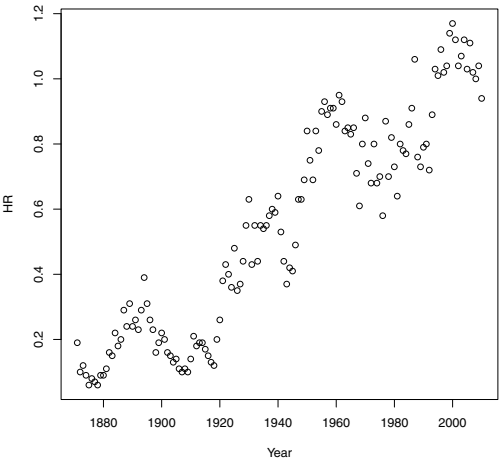

Fig. 4.1 Display of the average number of home runs hit per game per team through all of the years of Major League Baseball.

4.2 Labeling the Axes and Adding a Title

To communicate a statistical graphic, it is important that the horizontal and vertical scales are given descriptive labels. In addition, it is helpful to give a graphic a good title so the reader understands the purpose of the graphical display. In this example, we think that "Season" is a better label than "Year" for the horizontal axis, and we want to more precisely define what HR means on the vertical axis. We add these labels using the `xlab` and `ylab` arguments to `plot` and we create a graph title using the `main` argument. Using these labels, we create a better display as shown in Figure 4.2.

```
> plot(Year, HR, xlab="Season", ylab="Avg HR Hit Per Team Per Game",
+   main="Home Run Hitting in the MLB Across Seasons")
```

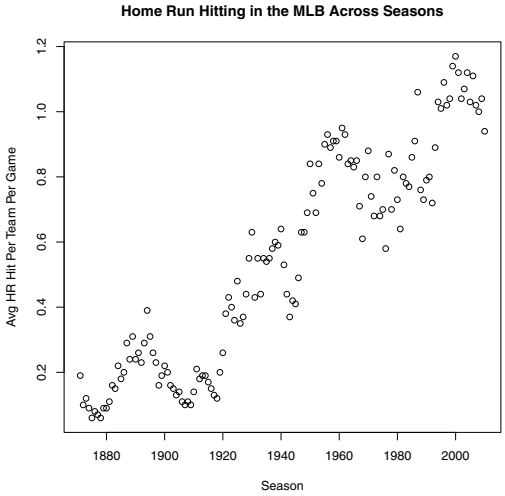

Fig. 4.2 Home run hitting graph with axis labels and title added.

R$_x$ 4.1 *By default, a high-level graphics function (such as* `plot`*) will overwrite the previous graph. One can avoid overwriting graphs by opening a new graphics window using the* `windows` *function on a Windows computer or* `quartz` *function on a Macintosh. Alternatively, after a graph is created, choose Recording from the History menu for a Windows interface. Then one can view different graphs by choosing "Next" and "Previous" from the History menu.*

4.3 Changing the Plot Type and Plotting Symbol

By default, the `plot` function produces a scatterplot of individually plotted points. Using the `type` argument option, one can connect the points with lines using `type = "l"`, show both points and connecting lines using `type = "b"`, or choose to show no points at all using `type = "n"`. In Figure 4.3, we illustrate the use of the `type = "b"` option which may be appropriate for some time series data.

```
> plot(Year, HR, xlab="Season", type="b",
+   ylab="Avg. Home Runs Hit by a Team in a Game",
+   main="Home Run Hitting in the MLB Across Seasons")
```

Using the `pch` argument in `plot`, one can specify the symbol of the plotting character. To see what symbols are available, Figure 4.4 draws all of the different plotting symbols together with the corresponding values of `pch`. The

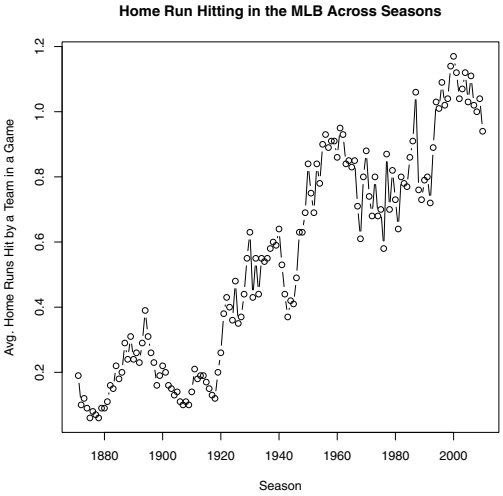

Fig. 4.3 Home run hitting graph using `type = b` option.

following R code to produce Figure 4.4 illustrates how one can modify the usual plotting axes. In the `plot` function, the horizontal and vertical domains of the plotting region are controlled using the `xlim` and `ylim` arguments. The `type = "n"` option indicates that nothing is plotted, and the `xaxt = "n"` and `yaxt = "n"` arguments indicate that no axes are to be drawn. The labels on the x and y axes are suppressed using the `xlab = ""` and `ylab = ""` arguments. The different plotting symbols are overlaid on the current plot using the `points` function. The `row` and `col` vectors indicate the plotting position of the points, the `pch` argument indicate the plotting symbols to be drawn, and the `cex = 3` argument magnifies the symbols to three times their usual size. Last, the `text` function allows one to display text on the current graph. Here we use the `text` function to overlay the numbers 0 through 20 next to the corresponding plotting symbols. These numbers are drawn 50% larger than their usual size (`cex = 1.5`) and they are positioned to the right of the symbol (`pos = 4`) about two character spaces away (`offset = 2`). The `title` function allows one to add a title after the graph has been drawn.

```
> row = rep(1:3, each=7)
> col = rep(1:7, times=3)
> plot(2, 3, xlim=c(.5,3.5), ylim=c(.5,7.5),
+    type="n", xaxt = "n", yaxt = "n", xlab="", ylab="")
> points(row, col, pch=0:20, cex=3)
> text(row, col, 0:20, pos=4, offset=2, cex=1.5)
> title("Plotting Symbols with the pch Argument")
```

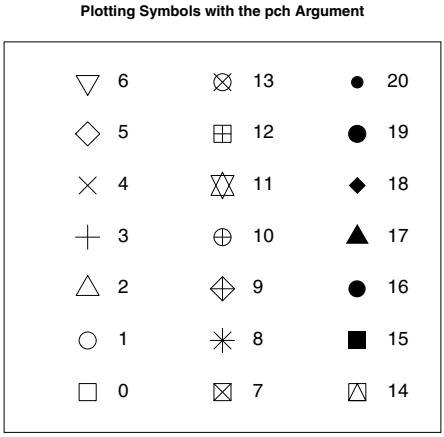

Fig. 4.4 Display of 21 different shapes of plotting points using the `pch` argument option. The number next to the symbol shape is the corresponding value of `pch`.

Returning to our home run hitting graph, suppose we wish to use a solid circle plotting symbol (`pch = 19`) drawn 50% larger than default (`cex = 1.5`). Figure 4.5 shows the revised graph of the home run rates across seasons.

```
> plot(Year, HR, xlab="Season", cex=1.5, pch=19,
+   ylab="Avg. Home Runs Hit by a Team in a Game",
+   main="Home Run Hitting in the MLB Across Seasons")
```

4.4 Overlaying Lines and Line Types

To better understand the general pattern in a time series graph, it is helpful to apply a smoothing function to the data. A general, all-purpose *lowess* smoothing algorithm (Cleveland [10]) is implemented using the `lowess` function. In the following R code, the script `lowess(Year, HR)` will implement the lowess smoothing algorithm and the `lines` function overlays the smoothed points as a connected line on the current graph. One obtains the graph shown in Figure 4.6.

```
> plot(Year, HR, xlab="Season",
+   ylab="Avg. Home Runs Hit by a Team in a Game",
+   main="Home Run Hitting in the MLB Across Seasons")
> lines(lowess(Year, HR))
```

Looking at Figure 4.6, this smooth does not appear to be a good representation of the pattern of changes in the home run rates. The degree of

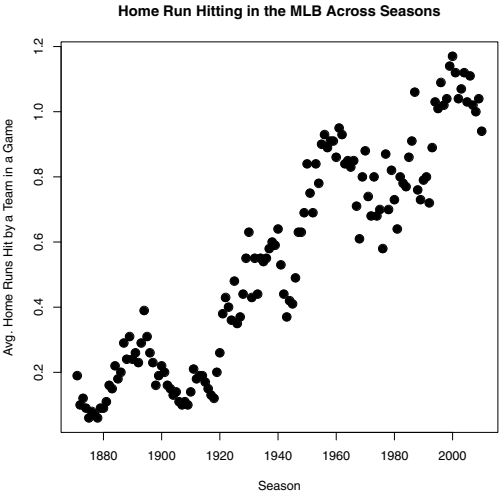

Fig. 4.5 Home run hitting graph with the choices `pch = 19` and `cex = 1.5` arguments in the `plot` function.

smoothness in the lowess algorithm is controlled by the smoother span parameter f, and it seems that the default choice $f = 2/3$ has oversmoothed the data. This comment suggests that one should try several smaller values for the smoothness parameter f and see the effect of these choices on the smooth on this particular dataset.

If one wishes to overlay several smoothing lines on the scatterplot, one should use different line patterns so that one can distinguish the different lines. One specifies the line style by the `lty` argument option. Six possible styles are numbered 1 through 6, that are equivalent to the strings "solid," "dashed," "dotted," "dotdash," "longdash," and "twodash." Figure 4.7 displays these six line styles using the following R code. As in the display of the point types, we first set up an empty graph with horizontal and vertical ranges from -2 to 2 and suppress the printing of the axes and the axes labels. We draw lines with the `abline` function with slope `b = 1` and intercepts `a` of $2, 1, 0, -1, -2, -3$ with corresponding line styles 1 through 6. We use the `legend` function so the reader can match the line style with the values of `lty`. In the `legend` function, we indicate that the legend box is to be drawn in the "topleft" portion of the graph, the legend labels are to be the six values "solid, "dashed," "dotted," "dotdash," "longdash," and "twodash," and the `lty` and the `lwd` parameters indicate the line type and thickness of the lines associated with the labels. In this example, we decide to use thicker lines (`lwd = 2`), so that the five line types are distinguishable.

```
> plot(0, 0, type="n", xlim=c(-2, 2),  ylim=c(-2, 2),
+    xaxt="n",  yaxt="n", xlab="", ylab="")
```

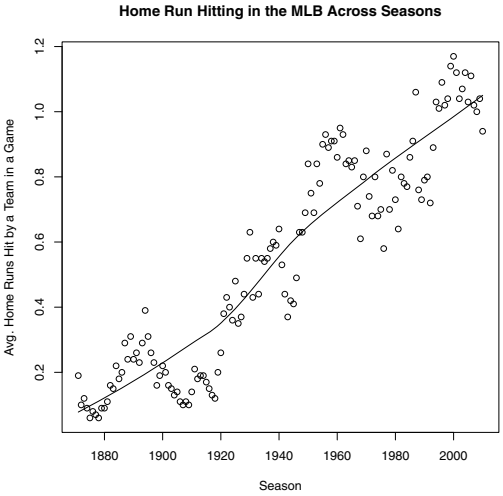

Fig. 4.6 Home run hitting graph with overlaying lowess fit using the default smoothing parameter.

```
> y = seq(2, -3, -1)
> for(j in 1:6)
+  abline(a=y[j], b=1, lty=j, lwd=2)
> legend("topleft", legend=c("solid", "dashed", "dotted",
+  "dotdash", "longdash", "twodash"), lty=1:6, lwd=2)
> title("Line Styles with the lty Argument")
```

Using the different line styles, we will redraw our home run graph with three lowess fits corresponding to the smoothing parameter values $f = 2/3$ (the default value), $f = 1/3$, and $f = 1/12$ in Figure 4.8. After constructing the basic graph by `plot`, we use the `lines` function three times to overlay the smooths on the graph using the line style values `lty ="solid"` (the default), `lty = "dashed"`, and `lty = "dotdash"`. We add a legend, so the reader will be able to connect the line style with the smoothness parameter value. Comparing the three smooths, it appears that the lowess smooth with the value $f = 1/12$ is the best match to the pattern of increase and decrease in the scatterplot. From this smoothed curve, we see that home run hitting increased from 1940 to 1960, decreased from 1960 to 1980, and increased again from 1980 to 2000.

```
> plot(Year, HR, xlab="Season",
+   ylab="Avg. Home Runs Hit by a Team in a Game",
+   main="Home Run Hitting in the MLB Across Seasons")
> lines(lowess(Year, HR), lwd=2)
> lines(lowess(Year, HR, f=1 / 3), lty="dashed", lwd=2)
> lines(lowess(Year, HR, f=1 / 12), lty="dotdash", lwd=2)
```

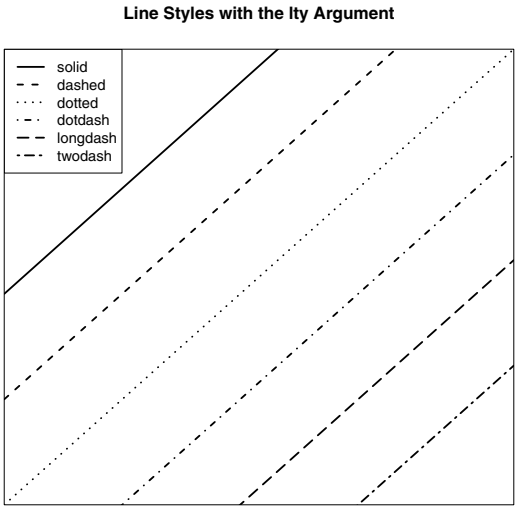

Fig. 4.7 Display of six different line styles using the `lty` argument option.

```
> legend("topleft", legend=c("f = 2/3", "f = 1/3",
+     "f = 1/12"), lty=c(1, 2, 4), lwd=2,  inset=0.05)
```

4.5 Using Different Colors for Points and Lines

We have focused on the selection of different point shapes and line styles, but it may be preferable to distinguish points and lines using different colors. R has a large number of color choices that are accessible by the `col` argument to graphics functions such as `plot` or `hist`. One can appreciate the large number of available plotting colors by simply typing

```
> colors()
  [1] "white"                "aliceblue"            "antiquewhite"
  [4] "antiquewhite1"        "antiquewhite2"        "antiquewhite3"
  [7] "antiquewhite4"        "aquamarine"           "aquamarine1"
 [10] "aquamarine2"          "aquamarine3"          "aquamarine4"
 [13] "azure"                "azure1"               "azure2"
 [16] "azure3"               "azure4"               "beige"
...
```

As a simple example, suppose you wish to graph the following ten points (1, 5), (2, 4), (3, 3), (4, 2), (5, 1), (6, 2), (7, 3), (8, 4), (9, 3), (10, 2) using

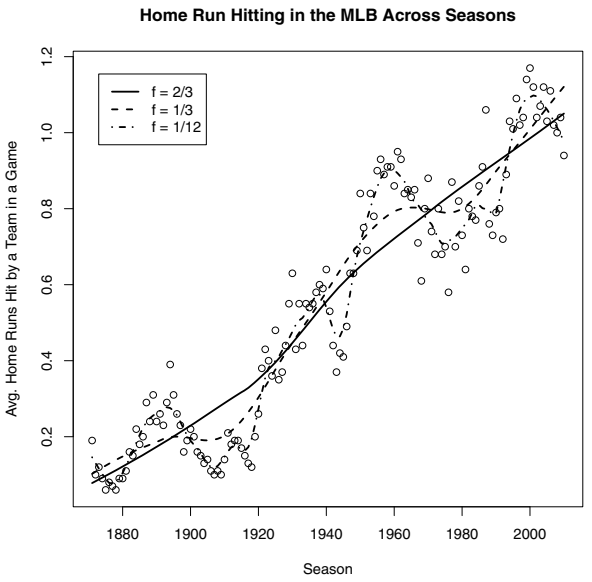

Fig. 4.8 Home run hitting graph with three lowess fits with three choices of the smoothing parameter.

a variety of colors. Try the following R code (output not shown) to see the display of colors. To make the colors more visible, the `plot` function uses the (`cx = 5, pch = 19`) arguments to draw large solid points.

```
> plot(1:10, c(5, 4, 3, 2, 1, 2, 3, 4, 3, 2),
+  pch=19, cex=5,
+  col=c("red", "blue", "green", "beige", "goldenrod",
+      "turquoise", "salmon", "purple", "pink", "seashell"))
```

One can also specify colors by numbers. By typing the following R code (output not shown), one can see the default mapping of the numbers 1 through 8 to colors.

```
> palette()
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```

The argument `col = 1` corresponds to black, `col = 2` corresponds to red, and so on. The following R code displays four lines drawn using the line styles `lty = 1`, ..., `lty = 4`. (The graph is not shown.)

```
> plot(0, 0, type="n", xlim=c(-2, 2), ylim=c(-2, 2),
+    xaxt="n", yaxt="n", xlab="", ylab="")
> y = c(-1, 1, 0, 50000)
```

```
> for (j in 1:4)
+  abline(a=0, b=y[j], lty=j, lwd=4)
```

If one adds the additional argument `col=j` to `abline`, one will see four lines drawn using contrasting line types and contrasting colors.

4.6 Changing the Format of Text

One has fine control over the style and placement of textual material placed on a graph. Specifically, one can choose the font family, style, and size of text through arguments of the `text` function. We illustrate some of these options through a simple example.

We begin by setting up an empty plot window where the horizontal limits are -1 and 6 and the vertical limits are -0.5 and 4. We place the text `"font = 1 (Default)"` at the coordinate point (2.5, 4). As the label suggests, this text is drawn using the default font family and size. (See the top line of text in the box in Figure 4.9.)

```
> plot(0, 0, type="n", xlim=c(-1, 6), ylim=c(-0.5, 4),
+    xaxt="n", yaxt ="n", xlab="", ylab="",
+    main="Font Choices Using, font, family and srt Arguments")
> text(2.5, 4, "font = 1 (Default)")
```

One can change the style of the text font using the `font` argument option. The values `font = 2`, `font = 3`, and `font = 4` correspond respectively to bold, italic, and bold-italic styles. Using the `srt` argument, one can rotate the text string by an angle (in degrees) in a counter-clockwise direction. We illustrate rotating the last string 20 degrees. (See the three lines of text on the left side of the box in Figure 4.9.)

```
> text(1, 3, "font = 2 (Bold)", font=2, cex=1.0)
> text(1, 2, "font = 3 (Italic)", font=3, cex=1.0)
> text(1, 1, "font = 4 (Bold Italic), srt = 20", font=4,
+  cex=1.0, srt=20)
```

One can control the size of the text using the `cex` option. In the R code, we use the `cex = 1.0` which corresponds to the default size. One can double the text size using `cex = 2` or decrease the text size using `cex = 0.5`.

The argument `family` is used to select the font family of the text. One can choose the alternative serif, sans, and mono families by the use of the `"serif"`, `"sans"`, and `"mono"` values for `family`. The following R code illustrates using `text` with these three common families and Figure 4.9 shows the results on the right hand side inside the box. The Hershey font families are also available. In the last line of code, we illustrate using the `"HersheyScript"` family, drawn at two and a half times the usual size using cex = 2.5, and drawn in red using `col = "red"`.

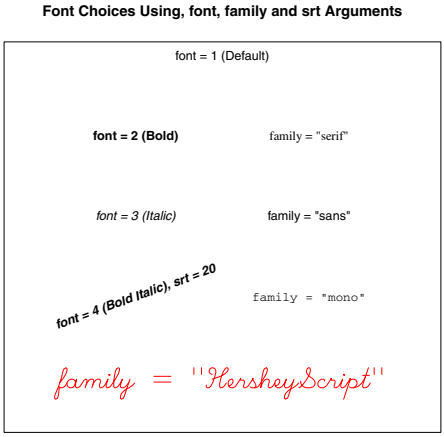

Fig. 4.9 Illustration of the use of different `font`, `family`, and `srt` arguments in the `text` function.

```
> text(4, 3, 'family="serif"', cex=1.0, family="serif")
> text(4, 2, 'family="sans"', cex=1.0, family="sans")
> text(4, 1, 'family="mono"', cex=1.0, family="mono")
> text(2.5, 0, 'family = "HersheyScript"', cex=2.5,
+   family="HersheyScript", col="red")
```

4.7 Interacting with the Graph

One helpful function is `identify` that allows a person to interact with a graph. Here we use `identify` to label interesting points in a scatterplot.

Example 4.2 (Home run hitting in baseball history (continued)).

We described the basic pattern in the home run hitting graph by a lowess smooth. We are next interested in examining the residuals, the vertical deviations of the points from the fitted points. In the following R script, we first save the lowess fit calculations in the variable `fit`. The component `fit$y` contains the fitted points and the residuals (actual HR values minus the fitted values) are computed and stored in the vector `Residual`. We construct a scatterplot of `Residual` against `Year` by the `plot` function and overlay a horizontal line at zero using the `abline` function.

```
> fit = lowess(Year, HR, f=1 / 12)
> Residual = HR - fit$y
> plot(Year, Residual)
> abline(h=0)
```

Looking at this graph (Figure 4.10), we notice two unusually large residuals, one positive and one negative. The `identify` function can be used to learn which seasons corresponded to these unusual residuals. This function has four arguments: the x and y plotting variables (`Year` and `Residual`), the number of points to identify (`n = 2`), and the vector of plotting labels to use (`Year`). After this function is executed, a crosshair appears when one moves the mouse over the graphics window. One identifies the points by clicking the mouse near the two points and the years corresponding to these points will appear. We see that the unusually large positive and large negative residuals correspond respectively to the 1987 and 1968 seasons. (The 1987 baseball season is notable in that players used steroids to help their hitting and the 1968 was a season where pitching was unusually dominant.) The output of `identify` is a vector with values 98 and 117 which are the row numbers of the data frame corresponding to these identified points.

```
> identify(Year, Residual, n=2, labels=Year)
[1]  98 117
```

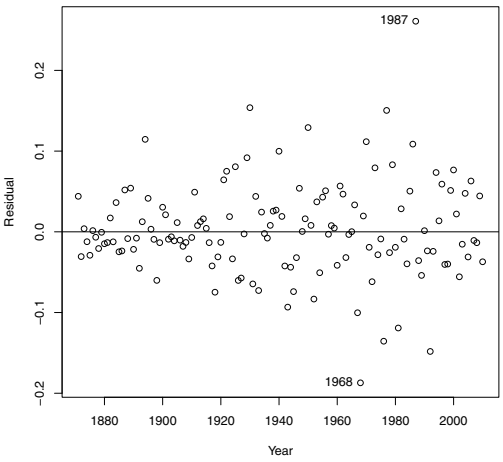

Fig. 4.10 Residual graph from lowess fit with two outliers identified.

4.8 Multiple Figures in a Window

Sometimes it is desirable to put several graphs in the same plot window. In the traditional graphics system, this can be accomplished using the `mfrow`

and `mfcol` parameter settings (the mnemonic is mf for "multiple figure"). Suppose one wishes to divide the window into six plotting regions in a grid with three rows and two columns. This will be set up by typing

```
> par(mfrow=c(3, 2))
```

Then a succession of six graphs, say using the `plot` function, will be displayed in the six plotting regions, where the the top regions are used first. If one wishes to restore the display back to a single plotting region, then type

```
> par(mfrow=c(1, 1))
```

In our example, suppose we wish to have two regions, where the home run rates and the lowess fit are displayed in the top region, and the residual graph is displayed in the bottom region. We begin using the `par` function with the `mfrow=c(2,1)` argument, and then follow with two `plot` functions displayed in the two regions (see Figure 4.11).

```
> par(mfrow=c(2, 1))
> plot(Year, HR, xlab="Season",
+   ylab="Avg HR Hit Per Team Per Game",
+   main="Home Run Hitting in the MLB Across Seasons")
> lines(fit, lwd=2)
> plot(Year, Residual, xlab="Season",
+   main="Residuals from Lowess Fit")
> abline(h=0)
```

4.9 Overlaying a Curve and Adding a Mathematical Expression

Example 4.3 (Normal approximation to binomial probabilities).

We use a new example to illustrate a few additional useful R graphics functions. It is well known that binomial probabilities can be closely approximated by the normal distribution. We wish to demonstrate this fact graphically by displaying a set of binomial probabilities and overlaying the approximating normal curve.

As our example, consider binomial probabilities for an experiment with sample size $n = 20$ and probability of success $p = 0.2$. We set up a grid of possible values for the binomial variable y, compute the corresponding probabilities using the R `dbinom` function, and store the probabilities in the vector `py`.

```
> n = 20; p = 0.2
> y = 0:20
> py = dbinom(y, size=n, prob=p)
```

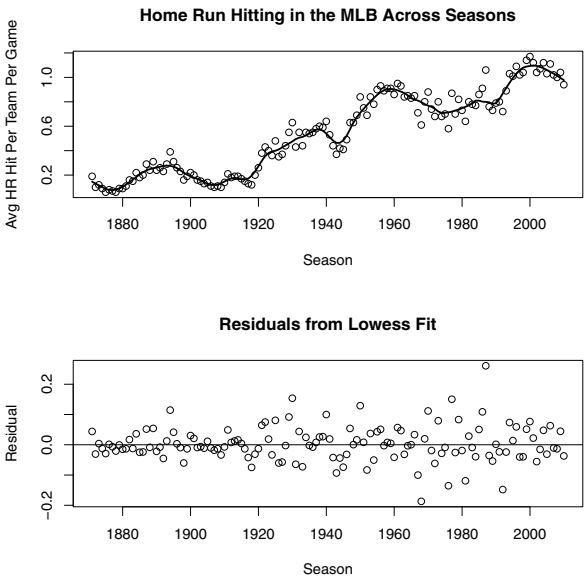

Fig. 4.11 Home run hitting graph with fit and residual plot.

We construct a *histogram* or vertical line style display using `plot` with the `type = "h"` argument option. We decide to plot thick lines with the `lwd=3` option, and limit the horizontal plotting range to (0, 15) by the `xlim` argument. Using `ylab`, we give a descriptive label for the vertical axis. The resulting display is shown in Figure 4.12.

```
> plot(y, py, type="h", lwd=3,
+   xlim=c(0, 15), ylab="Prob(y)")
```

Next we wish to overlay a matching normal curve to this plot. We compute the mean and standard deviation of the binomial distribution and use the `curve` function to overlay the normal curve. The function to be plotted by `curve` (in this case the `dnorm` normal density function) is always written as a function of x. Here x is a formal argument and does not reference a particular object in the workspace. Using the `add=TRUE` argument, we indicate that we wish to add this curve to the current plot and the `lwd` and `lty` arguments change the line thickness and line style of the displayed curve.

```
> mu = n * p; sigma = sqrt(n * p * (1 - p))
> curve(dnorm(x, mu, sigma), add=TRUE, lwd=2, lty=2)
```

To display the equation of the normal density on the graph, we can apply `text`, a general function for adding textual material to the current plot. The function `expression` can be used to add mathematical expressions. In the

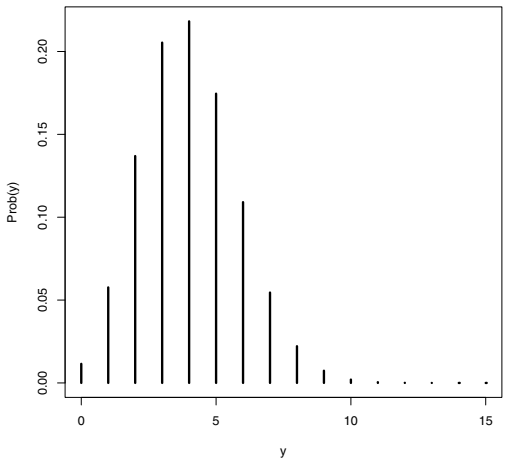

Fig. 4.12 Display of the binomial probability distribution for $n = 20$ and $p = 0.2$.

following R code, we indicate that we wish to place the text at the location (10, 0.15) on the plot, and use `expression` to enter in the normal density formula. The syntax inside `expression` is similar to Latex syntax – `frac` indicates a fraction, `sqrt` indicates a square root, and `sigma` refers to the Greek letter σ. The `paste` function is used to concatenate two character strings.

```
> text(10, 0.15, expression(paste(frac(1, sigma*sqrt(2*pi)), " ",
+            e^{frac(-(y-mu)^2, 2*sigma^2)})), cex = 1.5)
```

A descriptive title is added to the graph using the `title` function.

```
> title("Binomial probs with n=2, p=0.2, and matching normal curve")
```

The manual page giving a more complete description of the syntax for `expression` is found by typing

```
?plotmath
```

Last, we wish to draw a line that connects the normal density equation to the curve. This is conveniently done using the `locator` function. If we use `locator(2)`, a cross-hair will show on the graph and we choose two locations on the grid by clicking the mouse. The x and y locations of the chosen points are stored in the list `locs`. The `arrows` function draws a line between the two locations on the current display; the resulting display is shown in Figure 4.13.

```
> locs = locator(2)
> arrows(locs$x[1], locs$y[1], locs$x[2], locs$y[2])
```

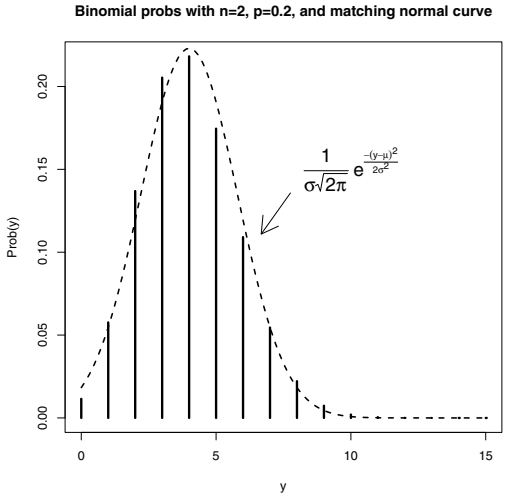

Fig. 4.13 Display of the binomial probability distribution for $n = 20$ and $p = 0.2$.with text and title added.

4.10 Multiple Plots and Varying the Graphical Parameters

We use a snowfall data example to illustrate constructing multiple plots using the `layout` function and modifying some of the graphical parameters using the `par` function.

The website http://goldensnowball.blogspot.com describes the Golden Snowball Award given to the city in New York State receiving the most snowfall during the winter season. The variables `snow.yr1` and `snow.yr2` contain the 2009-10 and 2010-11 snowfalls (in inches through February 24, 2011) for the cities Syracuse, Rochester, Buffalo, Binghamton, and Albany.

```
> snow.yr1 = c(85.9, 71.4, 68.8, 58.8, 34.4)
> snow.yr2 = c(150.9, 102.0, 86.2, 80.1, 63.8)
```

We wish to construct a scatterplot and a difference plot in the same plotting window. We use the `windows` function to open a new graphics device of size 5 inches by 7 inches. The `layout` divides this device into two regions; the `matrix` argument divides the device into two regions of a single column and the relative heights of the two regions are 6 and 4.

```
> windows(width=5, height=7)
> layout(matrix(c(1, 2), ncol=1), heights=c(6, 4))
```

With the default settings, the *figure region* is the entire region of the graphics device and the *plot region* is the region where the data is graphed. The

default location of the plot region can be displayed by showing the `plt` argument of `par`; the four values are represented in terms of fractions of the figure region. The region goes from 0.117 to 0.940 in the horizontal direction and 0.146 to 0.883 in the vertical direction.

```
> par("plt")
[1] 0.1171429 0.9400000 0.1459315 0.8826825
```

We wish to construct a scatterplot in the top region. We modify the `plt` and `xaxt` arguments to `par`. The values for `plt` result in a figure region with extra space on the right and less space below, and the `xact="n"` argument suppresses the plotting of the x axis. We use the `plot` function to construct a scatterplot of the two year snowfall amounts, use the `abline` function to overlay a line where the two snowfalls are equal, and label Syracuse's snowfall by the `text` function. The resulting display is shown in Figure 4.14.

```
> par(plt=c(0.20, 0.80, 0, 0.88), xaxt="n")
> plot(snow.yr1, snow.yr2, xlim=c(30, 100), ylim=c(30, 155),
+   ylab="2010-11 Snowfall (in)", pch=19,
+   main="Snowfall in Five New York Cities")
> abline(a=0, b=1)
> text(80, 145, "Syracuse")
```

The snowfall is measured in inches and we want to also display centimeters on the right axis to better communicate this graph to a non-American audience. We first save information about the current y tick mark locations (stored in the `yaxp` argument) in the variable `tm`. We use this information to compute the tick mark locations. Then we use the `axis` function to draw a right vertical axis; note that we compute the centimeter values by multiplying the inch values by 2.54 and rounding to the nearest 10 to make the tick labels more readable. Last, we use the `mtext` function to write a label on this new axis.

```
> tm = par("yaxp")
> ticmarks = seq(tm[1], tm[2], length=tm[3]+1)
> axis(4, at=ticmarks,
+     labels=as.character(round(2.54 * ticmarks, -1)))
> mtext("2010-11 Snowfall (cm)", side=4, line=3)
```

We next construct a scatterplot of the increase in snowfall (from 2009-10 to 2010-11) in the bottom region. The plot region is modified (using the `plt` argument) so that it matches the first graph horizontally and there is little space between the two regions. The `xaxt="s"` argument turns on the plotting of the x-axis. We plot the increase in snowfall against the 2009-10 amount and again identify the point corresponding to Syracuse. A right vertical axis corresponding to centimeters is again constructed using the same commands as in the top region.

```
> par(plt=c(0.20, 0.80, 0.35, 0.95), xaxt="s")
> plot(snow.yr1, snow.yr2 - snow.yr1, xlim=c(30, 100),
```

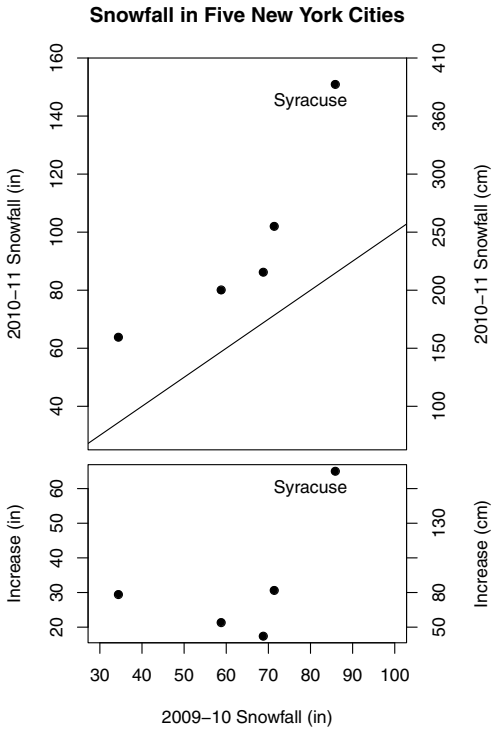

Fig. 4.14 Two graphs to display the snowfall of five New York cities for two consecutive seasons. This display uses the `layout` function to divide the graphics device into unequal areas, the `plt` argument to `par` adjusts the figure regions, and the `axis` function is used to add a second axis to the plots.

```
+  xlab="2009-10 Snowfall (in)", pch=19,
+  ylab="Increase (in)")
> text(80, 60, "Syracuse")
> tm=par("yaxp")
> ticmarks=seq(tm[1], tm[2], length=tm[3] + 1)
> axis(4, at=ticmarks,
+    labels=as.character(round(2.54 * ticmarks, -1)))
> mtext("Increase (cm)", side=4, line=3)
```

The message from this plot is that all five New York cities had more snow during the 2010-11 season. The bottom plot focuses on the change in snowfall; Syracuse had over 60 inches more snowfall in 2010-11 and the remaining cities had an additional 25 inches of snow.

4.11 Creating a Plot using Low-Level Functions

Many graphics commands in R are high-level functions that will take care of all aspects of graph construction including drawing the axes, plotting the tick marks and tick labels, and choosing the points and lines to be displayed. In some situations one wishes to create a special-purpose graph with finer control over the construction process. Here we illustrate a simple example of constructing a graph from scratch.

Example 4.4 (Graphing a circle with labels).

In Chapter 13, in our introduction of Markov Chains, in Figure 13.5, we constructed a graphical display of a circle on which six locations, numbered 1 through 6, are displayed. Since we don't want the plot axes to show, this is a good opportunity to illustrate some of the lower-level plotting functions in R.

To begin, one needs to open a new graphics frame – this is accomplished by the `plot.new` function.

```
> plot.new()
```

A coordinate system is set up using the `plot.window` function. The minimum and maximum values of the horizontal and vertical scales are controlled by the `xlim` and `ylim` arguments . The `pty` argument indicates the type of plot region – the choice `pty = "s"` generates a square plotting region.

```
> plot.window(xlim=c(-1.5, 1.5), ylim=c(-1.5, 1.5), pty="s")
```

Now that the coordinate system has been set up, it is convenient to use polar coordinates to graph a circle. One defines a vector of angles `theta` equally spaced from 0 to 2π and the coordinates of the circle are the cosine and sine functions evaluated at these angles. The function `lines` with arguments `x` and `y` adds a line graph to the current graph. (The work is displayed in Figure 4.15.)

```
> theta = seq(0, 2*pi, length=100)
> lines(cos(theta), sin(theta))
```

Next, suppose one wishes to draw large solid plotting points at six equally-spaced locations on the circle. Six angle values are defined and stored in the vector `theta`. The function `points` adds points to the current graph. The arguments `cex = 3` plots the symbols at three times the usual size, and `pch = 19` uses a solid dot plotting symbol.

```
> theta=seq(0, 2*pi, length=7)[-7]
> points(cos(theta), sin(theta), cex=3, pch=19)
```

The points will be labelled outside of the circle using large numbers 1 through 6. To get precise control over the plotting locations, the `locator` function graphically select the locations. The `text` function places the `labels`

one through six at these locations and the labels are drawn at two and one half times the usual size by specifying `cex = 2.5`. The plot is completed by drawing a box around the display with the `box` function.

```
> pos = locator(6)
> text(pos, labels=1:6, cex=2.5)
> box()
```

Figure 4.15 shows the completed graph. Using these low-level plot functions,

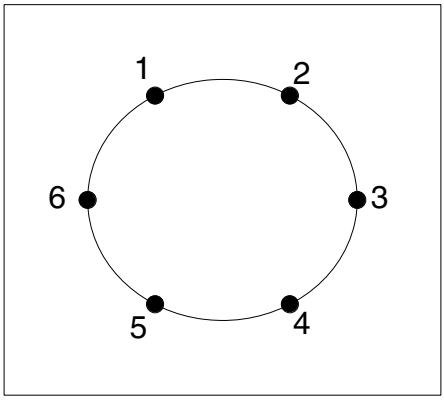

Fig. 4.15 Graphical display of a circle with labeled locations constructed using low-level graphics functions in R.

the user has much control over the axes display and characteristics of the drawn lines and points. Using these tools, it is possible in principle to create many new types of graphical displays.

4.12 Exporting a Graph to a Graphics File

After creating a graph with R, one often wishes to export the graph to a graphics format such as *pdf*, *gif*, or *jpeg*, so the graph can be included in a document or web page. There are several ways of exporting graphics that we will outline through an example.

Example 4.5 (Home run hitting in baseball history (continued)).

Suppose we wish to export the R graph in Figure 4.8 that displays the home run rates together with three lowess fits. In the Windows system, one

can simply save graphs using menu commands. After the graph has been constructed, select the graph and select the menu option "Save as" under the File menu. One selects the graphics format (metafile, postscript, pdf, png, bmp, or jpg) of interest and gives a name to the graphics file.

Alternatively, one can use functions to export graphs. For example, suppose we wish to save the same graph in the current working directory in a pdf format. In the R code, one first uses the function `pdf` to open the graphics device. The argument is the name of the saved graphics file, although one does not need to specify a name; the default name for saved files is "Rplots.pdf" that will overwrite any existing file with the same name. One next includes all of the functions for producing the graph. Then the `dev.off` function will close the graphics device and save the file.

```
> pdf("homerun.pdf")
> plot(Year, HR, xlab="Season",
+   ylab="Avg. Home Runs Hit by a Team in a Game",
+   main="Home Run Hitting in the MLB Across Seasons")
> lines(lowess(Year, HR))
> lines(lowess(Year, HR, f=1 / 3), lty=2)
> lines(lowess(Year, HR, f=1 / 6), lty=3)
> legend("topleft", legend=c("f = 2/3", "f = 1/3", "f = 1/6"), lty=c(1, 2, 3))
> dev.off()
```

Rx 4.2 *It is a common mistake to forget to type the* `dev.off()` *function at the end. If this function is not included, the graphics file will not be created.*

There are similar functions such as `postscript` and `jpeg` for saving the graph using alternative graphics formats. See `?Devices` for more details.

4.13 The `lattice` Package

Almost all of the graphs illustrated in this book are based on the traditional graphics system provided in the `graphics` package in R. Although this package provides much flexibility in producing a variety of statistical graphics, there are some alternative graphics systems available that extend and improve the traditional system. In this section we give a brief overview of the graphics available in the `lattice` package developed by Sarkar [43].

The `graphics` package contains a number of useful functions such as `plot`, `barplot`, `hist`, and `boxplot` for constructing statistical graphs. The `lattice` package contains a collection of plotting functions such as `xyplot`, `barchart`, `histogram`, and `bwplot` that produce similar types of graphs. By typing

```
> help(package=lattice)
```

one sees a display of functions and objects in the `lattice` package. Why should one use `lattice` functions instead of the traditional R graphing functions? First, there are several graphing functions in `lattice` that are not

available in the traditional system. Second, the default appearance of some of the `lattice` graphs is sometimes better than the default appearance of the traditional graphs. Last, the `lattice` package provides some attractive extensions of the basic R graphics system and these extensions will be the focus of this section.

Example 4.6 (Gas consumption of cars).

To illustrate the `lattice` graphics functions, consider the dataset `mtcars` which contains the fuel consumption and other measures of design and performance for a group of 32 cars. Suppose one wishes to construct a scatterplot of miles per gallon against the weight of the car (lbs divided by 1000). In the `lattice` package, the basic syntax of the scatterplot function is

```
xyplot(yvar ~ xvar, data)
```

where `data` is the data frame containing the two variables. In the R script, the `lattice` package is loaded (using the `library` function) and a scatterplot of `mpg` and `wt` is constructed. Axis labels and a title are added by the same arguments as the traditional graphics system. The resulting graph is in Figure 4.16.

```
> library(lattice)
> xyplot(mpg ~ wt, data=mtcars, xlab="Weight",
+   ylab="Mileage",
+   main="Scatterplot of Weight and Mileage for 32 Cars")
```

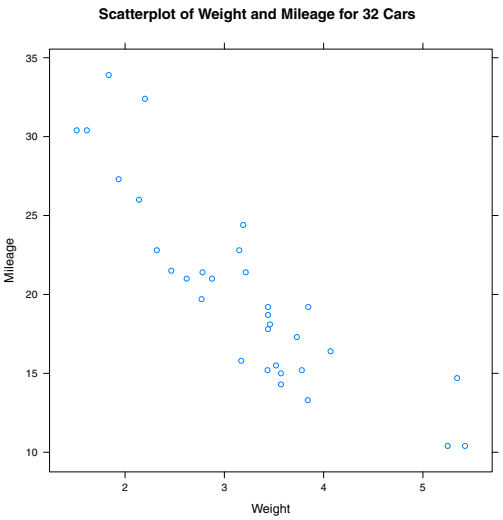

Fig. 4.16 Scatterplot of weight and mileage for a sample of cars using the `lattice` `xyplot` function.

From the scatterplot one understands that the mileage of a car depends on the number of cylinders in the engine. If one controls for the number of cylinders, is there still an association between mileage and weight? One nice extension offered by the `lattice` package is the availability of *conditional* graphs, that is, graphs conditional on the values of a third variable. In the case of a scatterplot, this conditional plot has the basic syntax

```
xyplot(yvar ~ xvar | cvar, data)
```

where `cvar` is a third variable with a limited number of values. In this example, the variable `cyl` is number of cylinders in the engine, and `xyplot` function is used to construct a separate scatterplot of mileage and weight for each possible number of cylinders. To help in visualizing the plotting symbols in Figure 4.17, solid plotting symbols (`pch = 19`) are drawn one and a half times larger than the usual size (`cex = 1.5`).

```
> xyplot(mpg ~ wt | cyl, data=mtcars, pch=19, cex=1.5,
+   xlab="Weight", ylab="Mileage")
```

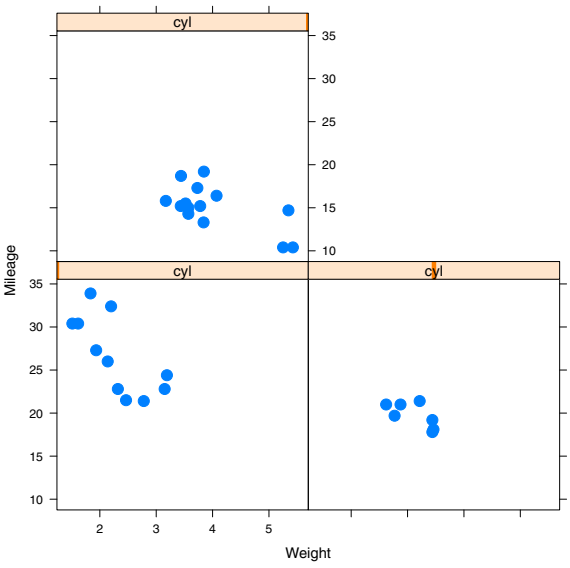

Fig. 4.17 Scatterplot of weight and mileage conditional on the number of cylinders using the `lattice` `xyplot` function.

Since the scaling is the same for all three plots, one can easily compare the three scatterplots. The panels in the lower left, lower right, and upper left sections of the figure correspond respectively to four, six, and eight cylinders.

For the 4-cylinder cars, there seems to be a relatively strong relationship between mileage and weight. In contrast, for 8-cylinder cars, the relationship between mileage and weight seems weaker.

The `lattice` package also allows for graphical displays that allow for comparisons between different data groups. Suppose that one is interested in comparing the weights of 4-cylinder, 6-cylinder, and 8-cylinder cars. A density plot is an estimate of the smooth distribution of a variable. One can construct density plots of different groups of data using the syntax

```
> densityplot(~ yvar, group=gvar, data)
```

where `yvar` is the continuous variable of interest, and `gvar` is the grouping variable. This type of graph can be used to compare the distributions of weights of the three types of cars. (See Figure 4.18.) The `auto.key` option will add a legend above the graph showing the line color of each group.

```
> densityplot(~ wt, groups=cyl, data=mtcars,
+   auto.key=list(space="top"))
```

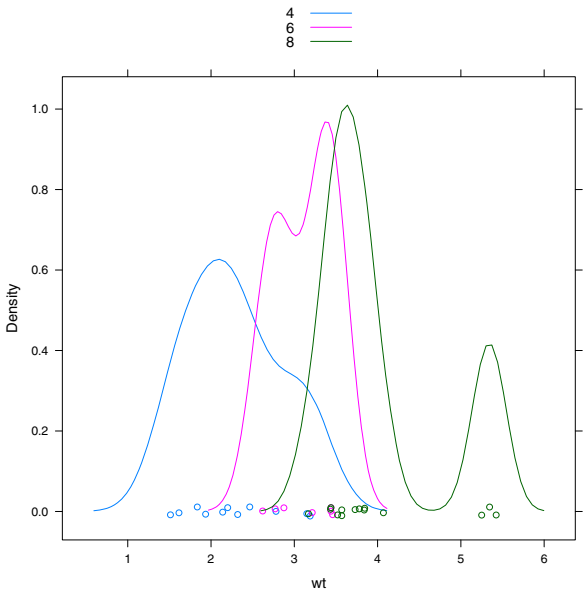

Fig. 4.18 Density plots of car weights for 4-cylinder, 6-cylinder, and 8-cylinder cars using the `lattice` `densityplot` function with the `groups` argument option.

Looking at this graph, one sees the variability of weights for each group of cars. Four-cylinder cars tend to weigh between 1000-4000 pounds and six-cylinder cars between 2000-4000 pounds. The eight-cylinder cars appear to

cluster in two groups, one group weighing 3000-4000 pounds and a second group between 5000-6000 pounds.

Example 4.7 (Sample means).

In Chapter 2, we considered collections of triples of random numbers generated using the the RANDU algorithm. Although this algorithm is supposed to produce uncorrelated triples, it has been well-documented that the simulated numbers have an association pattern. This can be easily shown using a a three-dimensional scatterplot drawn by the `lattice cloud` function.

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

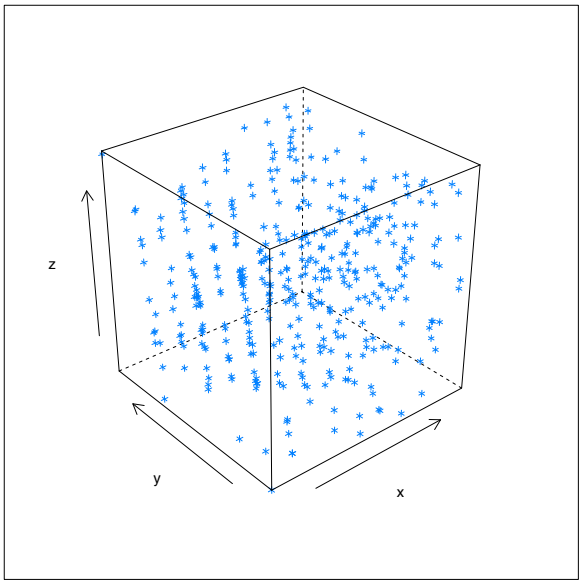

Fig. 4.19 Cloud plot of RANDU random number triples using the `lattice` package.

Looking carefully at Figure 4.19, one sees groups of points (x, y, z) that follow linear patterns of the type

$$z = ax + by$$

for constants a and b. The algorithm RANDU was used as the random number generation algorithm for the early personal computers, but currently alternative algorithms (without this dependence problem) are used for simulating random numbers in modern computers.

4.14 The ggplot2 Package

A second alternative to traditional graphics in R is the `ggplot2` package described in Wickham.[52] This system is based on a different way of thinking about graphics. Specifically, `ggplot2` is an R implementation of the "grammar of graphics" described in Wilkinson [55]. This alternative graphics system is described in the context of an interesting sports example.

Example 4.8 (World records in the mile run).

Professional athletes are continuously setting new world records in many sports such as running, jumping, and swimming, and it is interesting to explore the patterns of these world records over time. The data set "world.record.mile.csv" contains the world record times in the mile run for men and for women. Obviously, the world record times have decreased over time, but we are interested in learning about the pattern of decrease, and seeing if the pattern of decrease is different for men and women.

We read in the dataset using the `read.csv` function. We are interested in only the record times that were obtained in the year 1950 or latter; we use the `subset` function to create a new data frame `mile2` containing the times for these recent world records.

```
> mile = read.csv("world.record.mile.csv")
> mile2 = subset(mile, Year >= 1950)
```

Wilkinson [55] describes a statistical graphic as a combination of independent components. Variables in a dataset are assigned particular roles or *aesthetics*; the aesthetic for one variable might be the plotting position along the horizontal axis and the aesthetic for a second variable might be the color or shape of the plotting point. Once the aesthetics for a set of variables are defined, then one uses a *geometric object* or *geom* to construct a graph. Examples of geoms are points, lines, bars, histograms, and boxplots.

One thinks of constructing a graph by overlaying a set of layers on a grid. To construct a scatterplot, one overlays a set of point geoms on a Cartesian coordinate system. If one is interested in summarizing the pattern of points, then one might wish to overlay a smoothing curve on the current graph. A lowess smoother is an example of a *statistic*, a transformation of the data that you wish to graph.

There are many choices how the variables are mapped to the aesthetic properties. For example, variables may be graphed on a log scale, or one wishes to map a variable using a specific range of colors. These mappings are called *scales* and there are many choices for scales in the `ggplot2` package. The graphics system also allows for *position adjustments*, fine tune positioning of graphical objects such as jittering and stacking, and *faceting*, plotting subsets of the data on different panels.

For our example, we first load in the `ggplot2` package. The `ggplot` function is used to initialize the graph – the first argument is the name of the

data frame mile2 and the second argument ase() defines the aesthetics that map variables in the data frame to aspects of the graph. We indicate in the aes argument that Year is to be plotted along the horizontal (x) axis, seconds is to be plotted along the vertical (y) axis, and the variable Gender will be used as the color and shape aesthetics. We store this graph initialization information in the variable p.

```
> library(ggplot2)
> p = ggplot(mile2, aes(x = Year, y = seconds,
+  color = Gender, shape = Gender))
```

The function ggplot does not perform any plotting – it just defines the data frame and the aesthetics. We construct a plot by adding geom and statistic layers to this initial definition. A scatterplot is constructed using the geom_point function; the size = 4 arguments indicates that the points will be drawn twice the usual size of 2. Then we add a smoothing lowess curve using the geom_smooth function.

```
> p + geom_point(size = 4) + geom_smooth()
```

Figure 4.20 displays the resulting graphical display. Since the variable Gender has been assigned to the color and shape aesthetics, note that the male and female record times are plotted using different colors and shapes and a legend is automatically drawn outside of the plotting region. Note that since Gender has a color aesthetic, the smoothing curves are graphed for each gender. The record times appear to be linearly decreasing as a function of year for both genders this time period, although the rate of decrease is much greater for the women times.

Although the "grammar" of the ggplot2 system may seem odd at first look, one can construct attractive and useful graphics using a limited amount of R code. The book Hadley [52] provides a comprehensive description of the ggplot2 package and many graphics examples are displayed on the accompanying website had.co.nz/ggplot2.

Exercises

4.1 (Speed and stopping distance). The data frame cars in the datasets package gives the speed (in mph) and stopping distance (in ft) for 50 cars.

a. Use the plot function to construct a scatterplot of speed (horizontal) against dist (vertical).
b. Revise the basic plot by labeling the horizontal axis with "Speed (mpg)" and the vertical axis with "Stopping Distance (ft)," Add a meaningful title to the plot.
c. Revise the plot by changing the plot symbol from the default open circles to red filled triangles (col="red", pch=17).

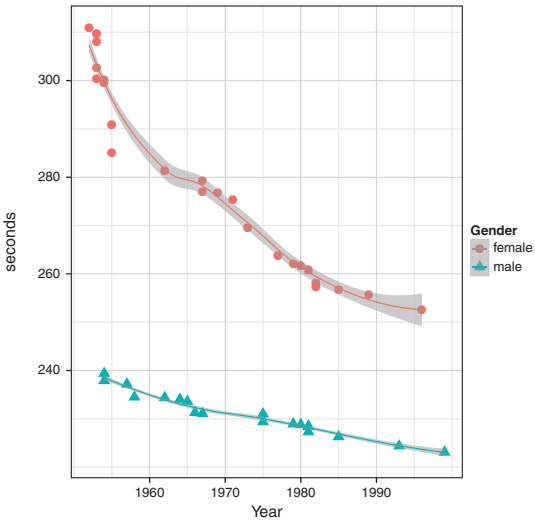

Fig. 4.20 Scatterplots of world record running times for the mile for the men and women using the `ggplot2` package. Smoothing lines are added to show the general pattern of decrease for each gender.

4.2 (Speed and stopping distance (continued)). Suppose that one wishes to compare linear and quadratic fits to the (`speed`, `dist`) observations. One can construct these two fits in R using the code

```
fit.linear = lm(dist ~ speed, data=cars)
fit.quadratic = lm(dist ~ speed + I(speed^2), data=cars)
```

a. Construct a scatterplot of speed and stopping distance.
b. Using the `abline` function with argument `fit.linear`, overlay the best line fit using line type "dotted" and using a line width of 2.
c. Using the `lines` function, overlay the quadratic fit using line type "long-dash" and a line width of 2.
d. Use a legend to show the line types of the linear and quadratic fits.
e. Redo parts (a) - (d) using two contrasting colors (say red and blue) for the two different fits.

4.3 (Speed and stopping distance (continued)).

a. Construct a residual plot for the linear fit by typing

```
plot(cars$speed, fit.linear$residual)
```

b. Add a blue, thick (`lwd=3`) horizontal line to the residual plot using the `abline` function.
c. There are two large positive residuals in this graph. By two applications of the `text` function, label each residual using the label "POS" in blue.

d. Label the one large negative residual in the graph with the label "NEG" in red.
e. Use the `identify` function to find the row numbers of two observations that have residuals close to zero.

4.4 (Multiple graphs). The dataset `mtcars` contains measurements of fuel consumption (variable `mpg`) and other design and performance characteristics for a group of 32 automobiles. Using the `mfrow` argument to `par`, construct scatterplots of each of the four variables `disp` (displacement), `wt` (weight), `hp` (horsepower), `drat` (rear axle ratio) with mileage (`mpg`) on a two by two array. Comparing the four graphs, which variable among displacement, weight, horsepower, and rear axle ratio has the strongest relationship with mileage?

4.5 (Drawing houses). The following function `house` plots an outline of a house centered about the point `(x, y)`:

```
house=function(x, y, ...){
  lines(c(x - 1, x + 1, x + 1, x - 1, x - 1),
     c(y - 1, y - 1, y + 1, y + 1, y - 1), ...)
  lines(c(x - 1, x, x + 1), c(y + 1, y + 2, y + 1), ...)
  lines(c(x - 0.3, x + 0.3, x + 0.3, x - 0.3, x - 0.3),
     c(y - 1, y - 1, y + 0.4, y + 0.4, y - 1), ...)
}
```

a. Read the function `house` into R.
b. Use the `plot.new` function to open a new plot window. Using the `plot.window` function, set up a coordinate system where the horizontal and vertical scales both range from 0 to 10.
c. Using three applications of the function `house`, draw three houses on the current plot window centered at the locations (1, 1), (4, 2), and (7, 6).
d. Using the `...` argument, one is able to pass along parameters that modify attributes of the `line` function. For example, if one was interested in drawing a red house using thick lines at the location (2, 7), one can type

   ```
   house(2, 7, col="red", lwd=3)
   ```

   Using the `col` and `lty` arguments, draw three additional houses on the current plot window at different locations, colors, and line types.
e. Draw a boundary box about the current plot window using the `box` function.

4.6 (Drawing beta density curves). Suppose one is interesting in displaying three members of the beta family of curves, where the beta density with shape parameters a and b (denoted by Beta(a,b)) is given by

$$f(y) = \frac{1}{B(a,b)} y^{a-1}(1-y)^{b-1},\ 0 < y < 1.$$

One can draw a single beta density, say with shape parameters $a = 5$ and $b = 2$, using the `curve` function:

```
curve(dbeta(x, 5, 2), from=0, to=1))
```

a. Use three applications of the `curve` function to display the Beta(2, 6), Beta(4, 4), and Beta(6, 2) densities on the same plot. (The `curve` function with the `add=TRUE` argument will add the curve to the current plot.)
b. Use the following R command to title the plot with the equation of the beta density.

```
title(expression(f(y)==frac(1,B(a,b))*y^{a-1}*(1-y)^{b-1}))
```

c. Using the `text` function, label each of the beta curves with the corresponding values of the shape parameters a and b.
d. Redraw the graph using different colors or line types for the three beta density curves.
e. Instead of using the `text` function, add a legend to the graph that shows the color or line type for each of the beta density curves.

4.7 (`lattice` graphics). The dataset `faithful` contains the duration of the eruption (in minutes) `eruptions` and the waiting time until the next eruption `waiting` (in minutes) for the Old Faithful geyser. One is interested in exploring the relationship between the two variables.

a. Create a factor variable `length` that is "short" if the eruption is smaller than 3.2 minutes, and "long" otherwise.

```
faithful$length = ifelse(faithful$eruptions < 3.2,
  "short", "long")
```

b. Using the `bwplot` function in the `lattice` package, construct parallel boxplots of the waiting times for the "short" and "long" eruptions.
c. Using the `densityplot` function, construct overlapping density plots of the waiting times of the "short" and "long" eruptions.

4.8 (`ggplot2` graphics). In Exercise 4.7, the waiting times for the Old Faithful geysers were compared for the short and long eruptions where the variable `length` in the `faithful` data frame defines the duration of the eruption.

a. Suppose a data frame `dframe` contains a numeric variable `num.var` and a factor `factor.var`. After the `ggplot2` package has been loaded, then the R commands

```
ggplot(dframe, aes(x = num.var, color = factor.var))
  + geom_density()
```

will construct overlapping density estimates of the variable `num.var` for each value of the factor `factor.var`. Use these commands to construct overlapping density estimates of the waiting times of the geysers with short and long eruptions.
b. With a data frame `dframe` containing a numeric variable `num.var` and a factor `factor.var`, the `ggplot2` syntax

```
ggplot(dframe, aes(y = num.var, x = factor.var))
  + geom_boxplot()
```

will construct parallel boxplots of the variable `num.var` for each value of the factor `factor.var`. Use these commands to construct parallel boxplots of the waiting times of the geysers with short and long eruptions.