José C. Pinheiro
Douglas M. Bates

Mixed-Effects Models in S and S-PLUS

With 172 Illustrations

Springer

José C. Pinheiro
Department of Biostatistics
Novartis Pharmaceuticals
One Health Plaza
East Hanover, NJ 07936-1080
USA
jose.pinheiro@pharma.novartis.com

Douglas M. Bates
Department of Statistics
University of Wisconsin
Madison, WI 53706-1685
USA
bates@stat.wisc.edu

Library of Congress Cataloging-in-Publication Data
Pinheiro, José C.
Mixed-effects models in S and S-PLUS / José C. Pinheiro, Douglas M. Bates
p. cm. — (Statistics and computing)
Includes bibliographical references and index.
ISBN 0-387-98957-9 (alk. paper)
I. Bates, Douglas M. II. Title. III. Series.
QA76.73.S15P56 2000
005.13'3—dc21 99-053566

Printed on acid-free paper.

Printed in the United States of America. (HAM)

9 8 7 6 5 SPIN 10995662

Springer Verlag is a part of *Springer Science+Business Media*

springeronline.com

3

Describing the Structure of Grouped Data

As illustrated by the examples in Chapter 1, we will be modeling data from experiments or studies in which the observations are grouped according to one or more nested classifications. Often this classification is by "Subject" or some similar experimental unit. *Repeated measures* data, *longitudinal* data, and *growth curve* data are examples of this general class of grouped data.

A common and versatile way of organizing data in S is as data.frame objects. These are in the form of tables where each row corresponds to an observation and each column corresponds to one of the variables being observed. We extend the data.frame class to the class of groupedData objects, which are data frames with additional information about the grouping of the observations and, possibly, other special roles of some variables.

In this chapter we describe creating, summarizing and displaying groupedData objects with a single level of grouping or with multiple levels of grouping.

3.1 The Display Formula and Its Components

A groupedData object contains the data values themselves, stored as a data frame, and a formula that designates special roles for some of the variables in the data frame. The most important of the special roles is that of a *grouping factor* that divides the observations into the distinct groups of

observations. The formula also designates a *response* and, when available, a *primary covariate*. It is given as

```
response ~ primary | grouping
```

where `response` is an expression for the response, `primary` is an expression for the primary covariate, and `grouping` is an expression for the grouping factor. Most often these expressions are simply the name of a variable in the data frame, but they could also be functions of one or more variables. For example, `log(conc)` would be a legitimate expression for the response if `conc` is one of the variables in the data frame.

The `formula` function extracts the formula from a grouped data object. Applied to some of the data sets used in Chapter 1 it produces

```
> formula( Rail )
travel ~ 1 | Rail
> formula( ergoStool )
effort ~ Type | Subject
> formula( Machines )
score ~ Machine | Worker
> formula( Orthodont )
distance ~ age | Subject
> formula( Pixel )
pixel ~ day | Dog/Side
> formula( Oats )
yield ~ nitro | Block
```

Notice that there is no primary covariate in the `Rail` data so we use the constant expression `1` in that position in the formula. In data with multiple, nested grouping factors, such as the `Pixel` data, the grouping factors are separated by "`/`". Factors that are nested within other factors appear further to the right so an expression like `Dog/Side` indicates that `Side` is nested within `Dog`.

The formula of a grouped data object has the same pattern as the formula used in a call to a trellis graphics function, such as `xyplot`. This is intentional. Because such a formula is available with the data, the `plot` method for objects in the groupedData class can produce an informative trellis display from the object alone. It may, in fact, be best to think of the formula stored with the data as a *display formula* for the data because it provides a meaningful default graphical display method for the data.

The `formula` function shown above is an example of an *extractor* function for this class. It returns some property of the object—the display formula in this case—without requiring the user to be aware of how that property is stored. We provide other extractor functions for each of the components of the display formula. The `getGroups` extractor returns the value of the grouping factor. A companion function, `getGroupsFormula`, returns the formula that is evaluated to produce the grouping factor. The extractors for the other components of the display formula are `getResponse` and `getCovariate`,

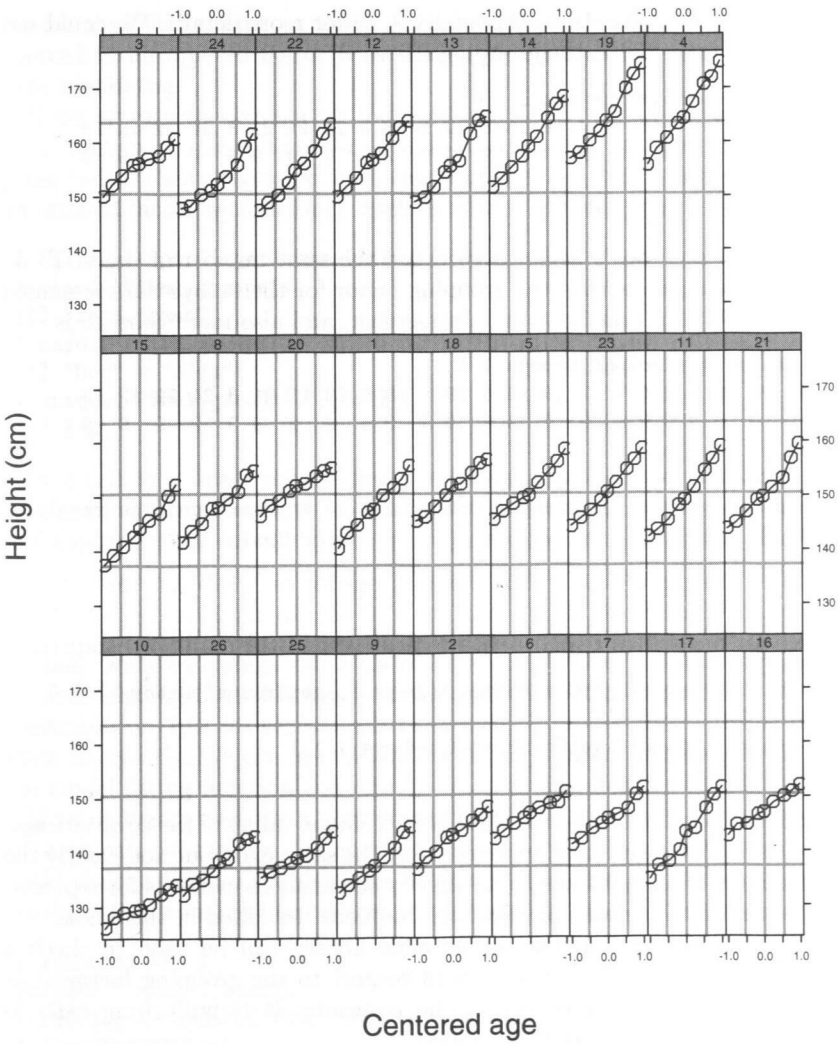

FIGURE 3.1. Heights of 26 boys from Oxford, England, each measured on nine occasions. The ages have been centered and are in an arbitrary unit.

which return numeric vectors or factors, and `getResponseFormula` and `getCovariateFormula`, which return formulas.

It is safer to use these extractor functions instead of checking the display formula for the object and extracting a variable from the object. For example, suppose we wish to check for balance in the `Oxboys` data, shown in Figure 3.1, and consisting of the heights at different ages of several boys

from Oxford, England (see Appendix A.19 for more detail). We could use the `table` function on the grouping factor,

```
> table( Oxboys$Subject )
 10 26 25 9 2 6 7 17 16 15 8 20 1 18 5 23 11 21 3 24 22 12 13
  9  9  9 9 9 9 9  9  9  9 9  9 9  9 9  9  9  9 9  9  9  9  9
 14 19 4
  9  9 9
```

to check if each boy's height is recorded the same number of times. To do this, we must know that the grouping factor for the `Oxboys` data is named `Subject`. A form that is easier to remember, and also more general, is

```
> table( getGroups( Oxboys ) )
 10 26 25 9 2 6 7 17 16 15 8 20 1 18 5 23 11 21 3 24 22 12 13
  9  9  9 9 9 9 9  9  9  9 9  9 9  9 9  9  9  9 9  9  9  9  9
 14 19 4
  9  9 9
> unique( table( getGroups( Oxboys ) ) )  # a more concise result
[1] 9
```

Because there are exactly nine observations for each subject, the data are balanced with respect to the number of observations. However, if we also check for balance in the covariate values, we find they are not balanced.

```
> unique( table( getCovariate( Oxboys ), getGroups( Oxboys ) ) )
[1] 1 0
> length( unique( getCovariate( Oxboys ) ) )
[1] 16
```

Further checking reveals that there are 16 unique values of the covariate `age`. The boys are measured at approximately the same ages, but not exactly the same ages. This imbalance could affect some analysis methods for repeated measures data. It does not affect the methods described in this book.

The `isBalanced` function in the `nlme` library can be used to check a groupedData object for balance with respect to the grouping factor(s) or with respect to the groups and the covariate. It is built from calls to `getGroups` and `table` like those above.

When applied to data with multiple, nested grouping factors, the `getGroups` extractor takes an optional argument `level`. Levels are counted from the outside inward so, in the `Pixel` data where the grouping is `Dog/Side`, `Dog` is the first level and `Side` is the second level. When the argument `level` specifies a single level the result is returned as a vector

```
> unique( getGroups(Pixel, level = 1) )
 [1] 1  2  3  4  5  6  7  8  9  10
> unique( getGroups(Pixel, level = 2) )
 [1] 1/R  2/R  3/R  4/R  5/R  6/R  7/R  8/R  9/R  10/R 1/L  ...

 1/R < 2/R < 3/R < 4/R < 5/R < ...
```

Notice that the groups at `level = 2`, the "`Side` within `Dog`" factor, are coerced to an ordered factor with distinct levels for each combination of `Side` within `Dog`.

If we extract the groups for multiple levels the result is returned as a data frame with one column for each level. Any inner grouping factors are preserved in their original form in this frame rather than being coerced to an ordered factor with distinct levels as above. For example,

```
> Pixel.groups <- getGroups( Pixel, level = 1:2 )
> class( Pixel.groups )
[1] "data.frame"
> names( Pixel.groups )
[1] "Dog"   "Side"
> unique( Pixel.groups[["Side"]] )
[1] R L
```

In a call to a linear mixed-effects modeling function, `lme` or `lmList`, or to a nonlinear mixed-effects modeling function, `nlsList` or `nlme` (discussed in Chapter 8), the default values for the response, for a covariate, and for the grouping factor are obtained from the formula stored with the data. These are, however, only the default values. They can be overridden with an explicit model formula. For example, during the course of model building we may wish to change our idea of what constitutes the response, say by transforming from a measure of concentration to the logarithm of the concentration. It is not necessary to change the formula stored with the data when doing this. We can use an explicit model formula as an argument to the model fitting function and override the formula stored with the data.

If we do decide to make a permanent change in the formula of a groupedData object, we can use the `update` function to do this. For example, we could change the covariate in the `PBG` data (discussed in §3.2.1 and Appendix A.21) from `dose` to `log(dose)` by updating the object.

```
> formula( PBG )
deltaBP ~ dose | Rabbit
> PBG.log <- update( PBG, formula = deltaBP ~ log(dose) | Rabbit )
> formula(PBG.log)
deltaBP ~ log(dose) | Rabbit
> unique( getCovariate(PBG.log) )
[1] 1.8326 2.5257 3.2189 3.9120 4.6052 5.2983
> unique( getCovariate(PBG) )
[1]   6.25  12.50  25.00  50.00 100.00 200.00
```

3.2 Constructing groupedData Objects

Constructing a groupedData object requires the data to be available as a data frame. There are several ways that data can be imported into S and

formed into a data frame. One of the simplest ways is using the `read.table` function on data stored in an external file (Venables and Ripley, 1999, §2.4).

For example, if the Oxford boys' height data are stored in a text file named `oxboys.dat` of the form

```
Subject     age height
      1 -1.0000 140.50
      1 -0.7479 143.40
      1 -0.4630 144.80
      1 -0.1643 147.10
      1 -0.0027 147.70
      1  0.2466 150.20
      1  0.5562 151.70
      1  0.7781 153.30
      1  0.9945 155.80
      2 -1.0000 136.90
          ...
     26 -0.0027 138.40
     26  0.2466 138.90
     26  0.5562 141.80
     26  0.7781 142.60
     26  1.0055 143.10
```

we can create a data frame with

```
> Oxboys.frm <- read.table( "oxboys.dat", header = TRUE )
> class( Oxboys.frm )        # check the class of the result
[1] "data.frame"
> dim( Oxboys.frm )          # check the dimensions
[1] 234 3
```

The argument `header = TRUE` in the call to `read.table` indicates that the first line of the file is to be used to create the names for the variables in the frame.

The result of `read.table` is of the data.frame class. It has two dimensions: the number of rows (cases) and the number of columns (variables).

A function to create objects of a given class is called a *constructor* for that class. The primary constructor function for a class is often given the same name as the class itself. Thus the default constructor for the groupedData class is the `groupedData` function. Its required arguments are a formula and a data frame. Optional arguments include `labels`, where display labels for the response and the primary covariate can be given, and `units`, where the units of these variables can be given. The default axis labels for data plots are constructed by pasting together components of `labels` and `units`. The reason for separating the units from the rest of the display label is to permit propagation of the units to derived quantities such as the residuals from a fitted model.

For example, reading the Oxboys data from a file, converting it to a groupedData object, and establishing default labels could be accomplished in a single call of

```
> Oxboys <- groupedData( height ~ age | Subject,
+    data = read.table("oxboys.dat", header = TRUE),
+    labels = list(x = "Centered age", y = "Height"),
+    units = list(y = "(cm)") )
> Oxboys                  # display the object
Grouped Data: height ~ age | Subject
    Subject     age height
  1       1 -1.0000 140.50
  2       1 -0.7479 143.40
  3       1 -0.4630 144.80
...
234      26  1.0055 143.10
```

By default the groupedData constructor also converts the grouping factor (the factor Subject in the Oxboys data) to an ordered factor. The order is determined by applying a summary function to the response within each group. The default summary function is max, the maximum.

When the grouping factor has been converted to an ordered factor, the panels in the trellis plots are arranged in that order. The order of the panels is from left to right across the rows, starting with the bottom row. In the default ordering described above the maximum value of the response in each panel will increase across the rows starting from the lower left panel. Most of the data plots in this book are ordered in this way.

Conversion of the grouping factor to an ordered factor is done only if the expression for the grouping factor is simply the name of a variable in the data frame. The conversion does not cause the rows of the data frame themselves to be reordered; it merely changes the class of the grouping factor and attaches the ordering to it. One way to examine the ordering is by requesting the unique values of the grouping factor

```
> unique( getGroups( Oxboys ) )
 [1] 1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
[21] 21 22 23 24 25 26

 10 < 26 < 25 < 9 < 2 < 6 < 7 < 17 < 16 < 15 < 8 < 20 < 1 < 18 ...
```

The first value of Subject in the data is 1, but the value of Subject with the smallest maximum height is 10 so this subject's data occupy the lower left panel in Figure 3.1.

The labels and units arguments are optional. We recommend using them because this makes it easier to create more informative trellis plots.

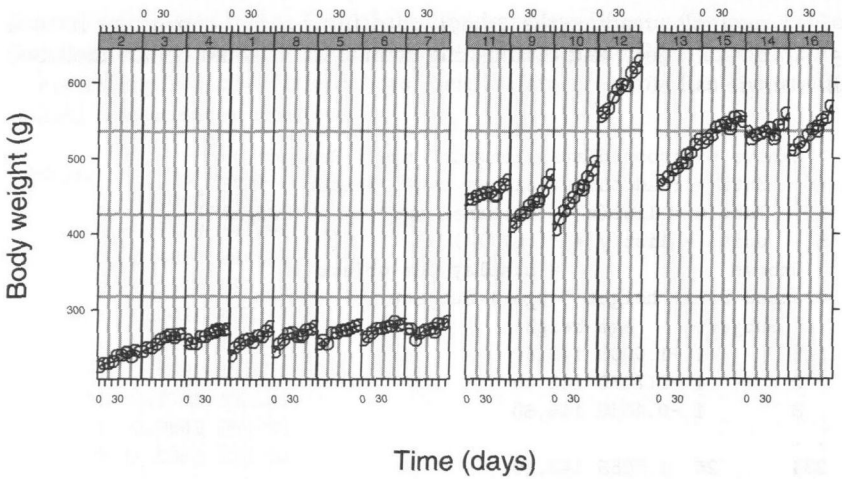

FIGURE 3.2. Weight versus time of rats on three different diets. The first group of eight rats is on the control diet. There were four rats each in the two experimental diets.

3.2.1 Roles of Other Experimental or Blocking Factors

Although the display formula can be used to designate a response, a covariate, and a grouping factor, these may not be sufficient to describe the structure of the experiment or study completely. Many experiments impose additional structure on the data. For example, the observations in the orthodontic data, described in §1.4.1, are grouped according to the subject on which the measurements were made. In most analyses of this type of data we would want to include the subject's sex as an explanatory factor. Because `Sex` is a characteristic of the subject, it is invariant within the observations for a single subject. A factor that is invariant within the groups determined by the grouping factor is said to be *outer* to the grouping factor.

When outer factors are present they can, and should, be designated as such when constructing a groupedData object using; for example,

```
outer = ~ Sex
```

Multiple outer factors can be specified by separating their names with "*" in the formula.

Outer factors can be characteristics of the subject (more generally, of the "experimental unit") such as `Sex` in this example. They can also be experimental factors that are applied to this level of experimental unit. For example, the `BodyWeight` data, shown in Figure 3.2 and described in Appendix A.3, contain measurements of the weights of 16 rats over time. Eight of the rats were given a control diet, four rats were given one exper-

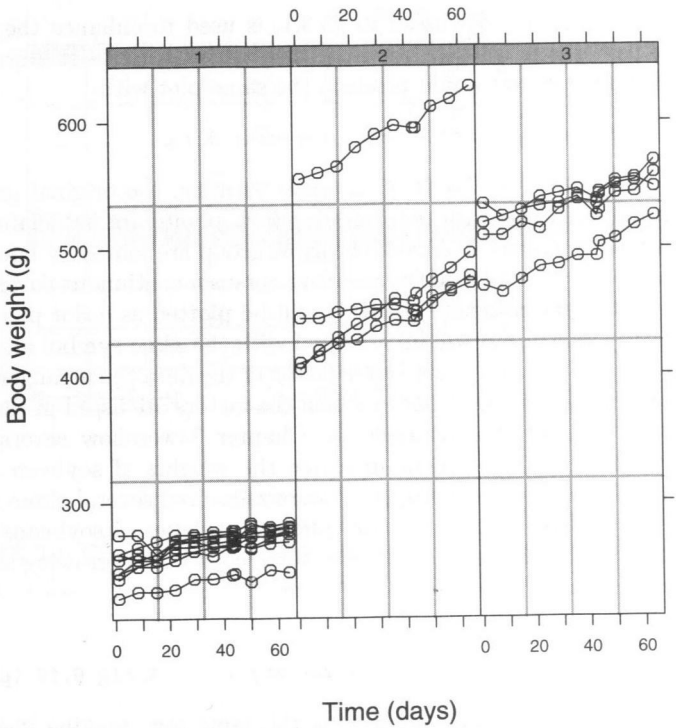

FIGURE 3.3. Weight of rats versus time for three different diets.

imental diet, and four rats were given another experimental diet. The `Diet` factor is an experimental factor that is outer to the grouping factor `Rat`.

One benefit of specifying outer factors to the constructor is that the constructor will modify the way in which the groups are ordered. Reordering of the groups is permitted only within the same level of an outer factor (or within the same combination of levels of outer factors, when there is more than one). This ensures that groups at the same levels of all outer factors will be plotted in a set of adjacent panels.

The plot method for the groupedData class allows an optional argument `outer` that can be either a logical value or a formula. When this argument is used the panels are determined by the factor or combination of factors given in the `outer` formula. A logical value of `TRUE` or `T` can be used instead of a formula, indicating that the outer formula stored with the data should be used to determine the panels in the plot. For example, we can get a stronger visual comparison of the differences between the diets for the `BodyWeight` data with

```
> plot( BodyWeight, outer = ~ Diet, aspect = 3 )        # Figure 3.3
```

The `aspect` argument, discussed in §3.3.1, is used to enhance the visualization of patterns in the plot. Because this `outer` formula was stored with the `BodyWeight` data we would produce the same plot with

```
> plot( BodyWeight, outer = TRUE, aspect = 3 )
```

In plots grouped by the levels of an `outer` formula, the original grouping factor stored with the data determines which points are associated with each other in the panels. Points in the same group are joined by lines when each panel is a scatter plot of the response versus a continuous covariate. If there is no primary covariate the data will be plotted as a dot plot where points in the same group will be rendered with the same symbol and color.

When there is more than one outer factor in the data the arrangement of the panels depends on the order in which the factors are listed in the `outer` formula for the plot. For example, in Chapter 7 we show several plots of the results of an experiment in which the weights of soybean plants, grown in different experimental plots, were measured several times during the growing season. There were two different varieties of soybeans in the experiment, which was carried out over three consecutive growing seasons. The grouping factor is `Plot` and the outer factors are `Variety` and `Year`. The plot produced by

```
> plot( Soybean, outer = ~ Year * Variety )      # Fig 6.10 (p. 288)
```

arranges panels of the same `Variety` on the same row, making it easy to compare the results for each variety across years. Conversely, the plot produced by

```
> plot( Soybean, outer = ~ Variety * Year )
```

(not shown) arranges panels of the same `Year` on the same row, making it easy to compare varieties within each year.

When specifying outer factors in the constructor or in a plot call we should ensure that they are indeed constant or *invariant* within each level of the grouping factor. The `gsummary` function with the optional argument `invariantsOnly = TRUE` allows us to check this. It returns the values of only those variables that are invariant within each level of the grouping factor. These values are returned as a data frame with one row for each level of the grouping factor. For the `BodyWeight` data we get

```
> gsummary( BodyWeight, invar = TRUE )
   Rat Diet
2    2    1
3    3    1
4    4    1
1    1    1
8    8    1
5    5    1
```

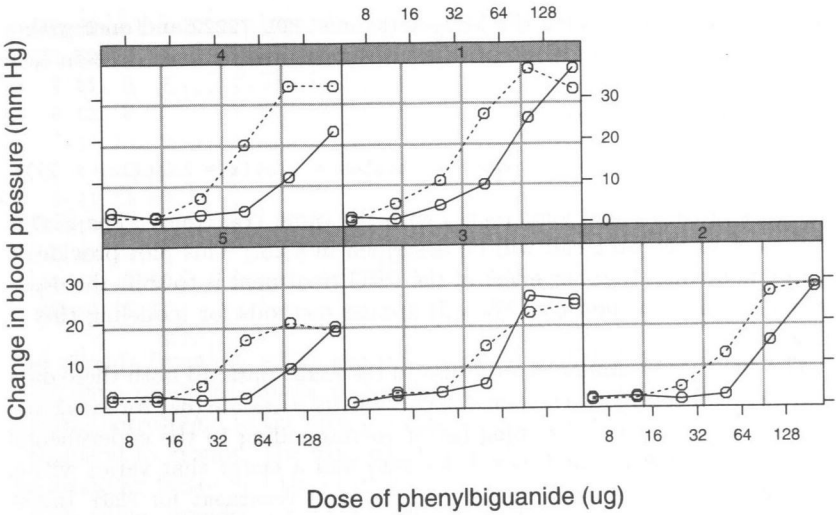

FIGURE 3.4. Change in blood pressure versus dose of phenylbiguanide (PBG) for five rabbits. Each rabbit was exposed to increasing doses of PBG, once after treatment with the H_5^3-agonist MDL 72222 and once after treatment with a placebo.

```
 6   6   1
 7   7   1
11  11   2
 9   9   2
10  10   2
12  12   2
13  13   3
15  15   3
14  14   3
16  16   3
```

indicating that `Diet` is an invariant of the grouping structure determined by the `Rat` factor. Notice that the grouping factor itself, `Rat` in this case, must be one of the invariants.

An outer factor is a characteristic of the experimental unit or an indicator of a treatment applied to the entire unit. In some experiments one level of a treatment may be applied to the experimental unit, say `Subject`, for some of the observations, then another level applied for other observations. If such an `inner` factor is distinct from the primary covariate, we may want to indicate it somehow on a data plot. An example of an inner factor is provided by the phenylbiguanide (PBG) data, shown in Figure 3.4 and described in Appendix A.21. These data are from a *cross-over* trial where each experimental animal was exposed to increasing doses of PBG under

two treatments; once with the MD_5-antagonist MDL 72222 and once with a placebo. The change in blood pressure was measured for each dose on each occasion.

In Figure 3.4, produced by

```
> plot( PBG, inner = ~ Treatment, scales = list(x = list(log = 2)))
```

the lines in each panel joint points with the same Treatment. (The scales argument to the plot call will be described in §3.3.) This plot provides a strong indication that the effect of the PBG treatment is to shift the dose–response curve to the right. We will discuss methods for modeling this in Chapter 7.

The PBG data is similar in structure to the Pixel data. In both these data sets there is a continuous response, a continuous covariate (day for Pixel and dose for PBG), a major grouping factor corresponding to the experimental animal (Dog for Pixel and Rabbit for PBG) and a factor that varies within this major grouping factor (Side for Pixel and Treatment for PBG). In the case of the Pixel data we used nested grouping factors to represent this structure. In the case of the PBG data we used a single grouping factor with an inner treatment factor. These two structures are quite similar—in fact, for the purposes of plotting the data, they are essentially equivalent. The choice of one structure or the other is more an indication of how we think the inner factor should be modeled. For the Pixel data we modeled the effect of the Side factor as a random effect because the Side just represents a random selection of lymph node for each Dog. In other experiments there may be important physiological differences between the left side and the right side of the animal so we would model it as a fixed effect. In the PBG data the Treatment factor is a factor with fixed, repeatable levels, and we model it as a fixed effect.

If we decide that an inner factor should be modeled as a random effect we should specify it as part of a nested grouping structure. If it should be a fixed effect we specify it as an inner factor. These choices can be overridden when constructing models.

3.2.2 Constructors for Balanced Data

The ergoStool data is an example of a data set that is balanced both with respect to the grouping factor and with respect to the primary covariate. That is, each Subject has the same number of observations on each Subject and each subject uses each stool type the same number of times.

It can be convenient to represent a balanced, unreplicated set of responses as a matrix. For example,

```
> ergoStool.mat <- asTable( ergoStool )
```

```
> ergoStool.mat
  T1 T2 T3 T4
8  7 11  8  7
5  8 11  8  7
4  7 11 10  9
9  9 13 10  8
6  9 11 11 10
3  7 14 13  9
7  8 12 12 11
1 12 15 12 10
2 10 14 13 12
```

The `asTable` function, which can only be used with balanced and unreplicated data, produces a table of the responses in the form of a matrix where columns correspond to the unique values of the primary covariate and rows correspond to groups. The `dimnames` of the matrix are the unique levels of the grouping factor and the covariate.

This table provides a compact representation of balanced data. Often the data from a balanced experiment are provided in the form of a table like this. The `balancedGrouped` function convert data from a table like this to a groupedData object.

```
> ergoStool.new <- balancedGrouped( effort ~ Type | Subject,
+                                   data = ergoStool.mat )
Warning messages:
  4 missing values generated coercing from character to numeric
       in: as.numeric(dn[[1]])
> ergoStool.new
Grouped Data: effort ~ Type | Subject
   Type Subject effort
1    T1       8      7
2    T2       8     11
3    T3       8      8
4    T4       8      7
5    T1       5      8
...
36   T4       2     12
```

The formula given as the first argument to `balancedGrouped` is used to assign names to the response, the primary covariate, and the grouping factor. The data values are extracted from the matrix itself and from its `dimnames` attribute. The matrix should be arranged so each row contains the data from one group. The covariate values, corresponding to the different columns, can be the levels of a factor or of a continous covariate. If the column names can all be converted to numeric values, the covariate is assumed to be continuous and is coerced to a numeric vector. Otherwise it is left as a factor. The process of checking for numeric values in these names is what generates the warning message in the previous example.

It is a common practice to label the levels of a factor like the `Type` factor as `1, 2, ...`, which would result in its being coerced to a numeric variable. Unless this is detected and the numeric variable is explicitly converted to a factor, models fit to such data will be nonsensical. It is always a good idea to check that the variables in a groupedData object have the expected classes. We describe how to do this in §3.4.

The optional `labels` and `units` arguments can be used with `balancedGrouped` just as in the `groupedData` constructor.

As seen in the example, the `balancedGrouped` constructor produces an object like any other groupedData object. The matrix of response values is converted to a vector and both the primary covariate and the grouping factor are expanded to have the same length as the response vector. Later manipulations of the object or plots created from the object do not rely on its having been generated from balanced data. This is intentional. Although there is a certain amount of redundancy in storing multiple copies of the same covariate or grouping factor values, it is offset by the flexibility of the data.frame structure in dealing with missing data and with general, unbalanced data.

Many methods for the analysis of longitudinal data or repeated measurements data depend on having balanced data. The analysis techniques described in this book do not.

3.3 Controlling Trellis Graphics Presentations of Grouped Data

Trellis graphics presentations of grouped data allow easy evaluation of the behavior of the response with respect to the primary covariate within each group. They also allow comparisons between groups. Because they are so effective at illustrating both within-group and between-group behavior, they are the default plot method for groupedData objects.

The defaults chosen in the trellis graphics library and in the `plot` method for groupedData objects will usually provide an informative and visually appealing plot. Sometimes, however, the default plot can be made even more informative by adjusting one or two of the trellis graphics parameters. In this section we describe some of the trellis parameters that are helpful in enhancing plots of grouped data. For a full discussion of the trellis graphics parameters and controls see Becker, Cleveland and Shyu (1996) or the online documentation for the `trellis` library.

3.3.1 Layout of the Trellis Plot

A trellis plot consists of one or more *panels* arranged in a regular array on one or more pages. In the default data plot for a groupedData object

with a continuous primary covariate, there is one panel for each level of the grouping factor. The horizonal axis in the panel is the primary covariate, the vertical axis is the response, and the data are represented both as points and by a connecting line. If the primary covariate is a factor, such as in the `Machine` data, or if there is no primary covariate, such as in the `Rail` data, the plot is a *dotplot* with one row for each level of the grouping factor. In this case the response is on the horizontal axis.

For numeric covariates the *aspect ratio* of each panel, which is the ratio of the physical size of the vertical axis to that of the horizontal axis, is determined by the 45-degree banking rule described in Cleveland (1994, §3.1). We have found that this rule produces appealing and informative aspect ratios in a wide variety of cases. If you wish to override this choice of aspect ratio, you can give an numerical value as the optional `aspect` argument in the call to `plot`. A value greater than 1.0 produces tall, narrow panels, while a value between 0.0 and 1.0 produces short, wide panels.

The arrangement of panels in rows and columns on the page is calculated to make the specified number of panels of the chosen aspect ratio fill as much as possible of the available area. This does not always create a good arrangement for comparing patterns across outer factors. For example, the grouping factor `Plant` in the `CO2` data, described in Appendix A.5, has twelve different levels. The plants themselves come from one of two types and have been subjected to one of two treatments. Because the aspect ratio chosen by the banking rule creates panels that are taller than they are wide, a more-or-less square plot area will be filled with three rows of four panels each (Figure 3.5).

For some combinations of grass type and treatment the panels are spread across more than one row in Figure 3.5. It would be better to keep these combinations on the same row so we can more easily compare treatments and grass types. That is, we would prefer the panels to be arranged in four rows of three or, perhaps, two rows of six. If we have four rows of three, we may wish to indicate visually that the lower two rows represent one type of plant (Québec) and the upper two rows represent the other type (Mississippi). We can do this by specifying a list as the optional `between` argument. A component named `x` in this list indicates the sizes of gaps to leave between columns while a component named `y` specifies gaps between rows. (When forming a `between` argument for the rows, remember that the trellis convention is to count from the bottom to the top, not from the top to the bottom.) The gaps are given in units of character heights. Generally a gap of 0.5 is sufficient to distinguish groups without using too much space.

An arrangement of the `CO2` data in two rows of six panels with a gap between the third and fourth columns, shown in Figure 3.6, is produced by

```
> plot(CO2, layout=c(6,2), between=list(x=c(0,0,0.5,0,0))) # Fig 3.6
```

Assembling the panels on a single page is effective when there is a small or moderate number of groups. If there is a large number of groups, the

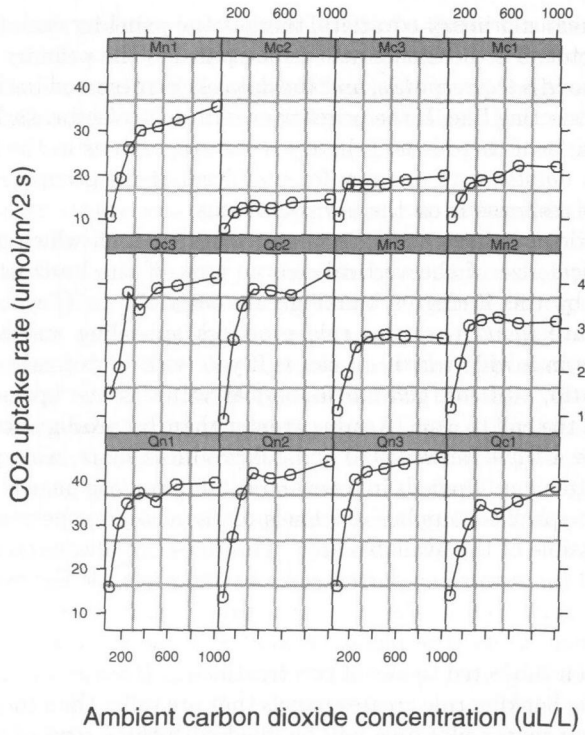

FIGURE 3.5. Carbon dioxide uptake versus ambient CO_2 concentration for *Echinochloa crus-galli* plants, six from Québec and six from Mississippi. Half the plants of each type were chilled overnight before the measurements were taken. The labels of each panel show the origin of the plant (Québec or Mississippi) and the treatment (chilled or nonchilled). This plot shows a default layout of the panels.

single page plot may result in the panels being too small to be informative. In these cases a third component can be added to the `layout` argument causing the plot to be spread over several pages.

If the number of groups in some level of an outer factor does not fit exactly into the rectangular array, an optional `skip` argument can be used to skip over selected panel locations.

The use of both of these arguments is illustrated in the code for the figures of the spruce tree growth data (Appendix A.28). There were four groves of trees; two exposed to an ozone-rich atmosphere, and two exposed to a normal atmosphere. In the first and second groves 27 trees were measured, but in the third and fourth groves only 12 and 13 trees were measured, respectively.

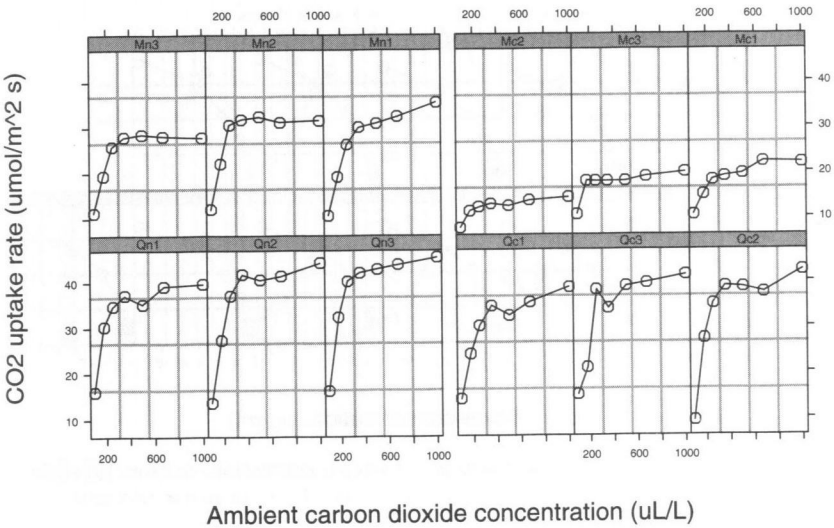

FIGURE 3.6. Carbon dioxide uptake versus ambient CO_2 concentration for *Echinochloa crus-galli* plants. This plot shows an alternative layout of the panels.

The three pages of figures (Figures A.8–A.10, pages 445–447) in an array of four rows of seven columns per page were created with

```
> plot( Spruce, layout = c(7, 4, 3),
+       skip = c(rep(FALSE, 27), TRUE, rep(FALSE, 27), TRUE,
+                rep(FALSE, 12), rep(TRUE, 2), rep(FALSE,13)) )
```

An alternative arrangement (not shown) of three pages in an array of three rows of nine columns per page can be created with

```
> plot( Spruce, layout = c(9, 3, 3),
+       skip = c(rep(FALSE, 66), TRUE, TRUE, rep(FALSE, 13)) )
```

On the first two pages of this plot the array would be filled. On the third page there would be a gap in the middle of the array to separate the panels corresponding to the two different groves of trees exposed to a normal atmosphere.

3.3.2 Modifying the Vertical and Horizontal Scales

It is often helpful to present quantities such as concentrations on a logarithmic scale. The optional `scales` argument for trellis graphics functions allows specification of logarithmic scales on either the vertical or horizontal axes. The axis annotations can be at powers of 10 or at powers of 2.

For example, the default plot of the DNase assay data shown in Figure 3.7 squeezes most of the data onto the left-hand side of the panel. (These data are described in more detail in Appendix A.7.)

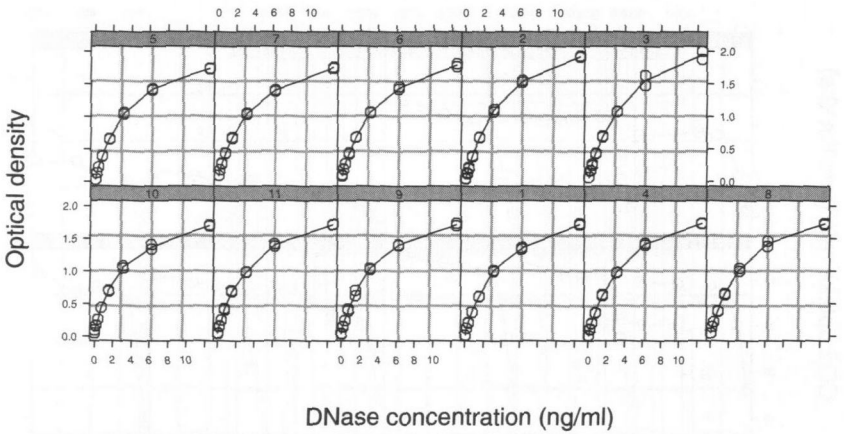

FIGURE 3.7. Optical density versus DNase concentration, arithmetic scale

Both the facts that the unique values of the DNase concentration are, for the most part, logarithmically spaced

```
> unique( getCovariate(DNase) )
[1]  0.048828  0.195312  0.390625  0.781250  1.562500  3.125000
[7]  6.250000 12.500000
> log( unique(getCovariate(DNase)), 2 )
[1] -4.35614 -2.35614 -1.35614 -0.35614  0.64386  1.64386  2.64386
[8]  3.64386
```

and the experimenters' desire to fit a logistic response function (described in Chapter 6 and Appendix C.7) to the logarithm of the concentration indicate that we should use a logarithmic scale on the horizontal axis. This change is incorporated in Figure 3.8, produced with

```
> plot( DNase, layout=c(6,2), scales = list(x=list(log=2)) )
```

3.3.3 Modifying the Panel Function

This is an advanced topic. You can consider skipping this section unless you want to modify the way the data are presented within each panel.

The presentation of the data within each panel is controlled by a *panel function*. If no primary covariate is available, or if the primary covariate is a categorical variable, the default panel function for the plot method for groupedData objects is `panel.dotplot`. When the primary covariate is numeric, the default panel function is

```
function(x, y) {
  panel.grid()
  panel.xyplot(x, y)
```

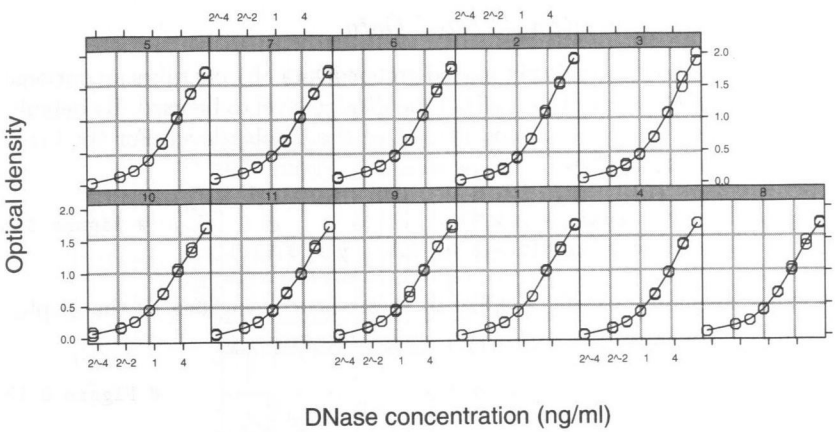

FIGURE 3.8. Optical density versus DNase concentration for eleven runs of an assay. The concentration is shown on a logarithmic scale.

```
  y.avg <- tapply(y, x, mean)        # average y for each distinct x
  xvals <- as.numeric(names(y.avg))
  ord <- order(xvals)
  panel.xyplot(xvals[ord], y.avg[ord], type = "l")
}
```

The first two lines of this function draw the background grid and place symbols at the data values. The actual symbol that is drawn is determined by the trellis device that is active when the plot is displayed. It is usually an open circle.

The last four lines of the panel function add a line through the data values. Some care needs to be taken when doing this. In the DNase assay data, for example, there are duplicate observations at each concentration in each run. Rather than "joining the dots," it makes more sense to draw the line through the average response in each set of replicates. In the default panel function `xvals` is defined to be the unique values of `x` and `y.avg` is calculated as the average of the `y` values at these distinct `x` values. Finally, the `xvals` vector is put into increasing order and the line drawn with the points in this order.

This panel function can be overridden with an explicit `panel` argument to the plot call if, for example, you want to omit the background grid. If you do override the default panel function, it would be a good idea to follow the general pattern of this function. In particular, you should draw the grid first (if you choose to do so) then add any points or lines. Also, be careful to handle replicates or unusual ordering of the (x, y) pairs gracefully.

3.3.4 *Plots of Multiply-Nested Data*

The `plot` method for multiply-nested groupedData objects takes an optional argument `displayLevel` that defines the *display* level to be used. By default, the innermost level of grouping is used as the display level. For the `Pixel` data, this is "`Side` within `Dog`", as shown in Figure 3.9.

```
> plot( Pixel, layout = c(4,5),                          # Figure 3.9
+       between = list(x = c(0, 0.5), y = 0.5) )
```

A more meaningful trellis display of these data, using `Dog` as the display level, is obtained with

```
> plot( Pixel, displayLevel = 1 )                        # Figure 3.10
```

When the display level for a multiply-nested groupedData object is smaller than the maximum grouping level, different actions can be taken with respect to the inner grouping levels. The default action is to preserve, as much as possible, the original structure of the data and use the combination of the inner grouping factors as a single inner factor, as in Figure 3.10. Alternatively, observations within a given value of the display level may be collapsed over some, or all inner levels by giving a `collapse` argument in the `plot` call. Another optional argument, `FUN`, can specify the summary function to be use when collapsing the data. This summary function should take a numeric vector as its argument and return a single numeric value. The default is the `mean` function.

Using `collapse` can help to reduce clutter in a plot. For example, the `Wafer` data, from an experiment in semiconductor manufacturing, give current intensity versus the applied voltage at eight sites on each of ten wafers (see Appendix A.30 for details). Differences between wafers and between sites within wafers are too subtle to be noticeable on a plot at `displayLevel = 2` that has separate panels for each `Site` within each `Wafer`. Displaying at the `Wafer` level with separate curves for each `Site` within each `Wafer`, as in Figure 3.11, produces a cluttered plot because the eight curves for each wafer nearly overlay each other. It is more informative to examine the mean curve for each wafer and the standard deviation about this mean curve, as in Figures 3.12 and 3.13

```
> plot( Wafer, display = 1, collapse = 1 )               # Fig. 3.12
> plot( Wafer, display = 1, collapse = 1,                # Fig. 3.13
+       FUN = function(x) sqrt(var(x)), layout = c(10,1) )
```

In general there is a trend for the standard deviation about the curve to be greater when the response is greater. This is a common occurrence. A less obvious effect is that the standard deviation is greater in the wafers with overall higher current intensity, even though the differences in the current intensity are small.

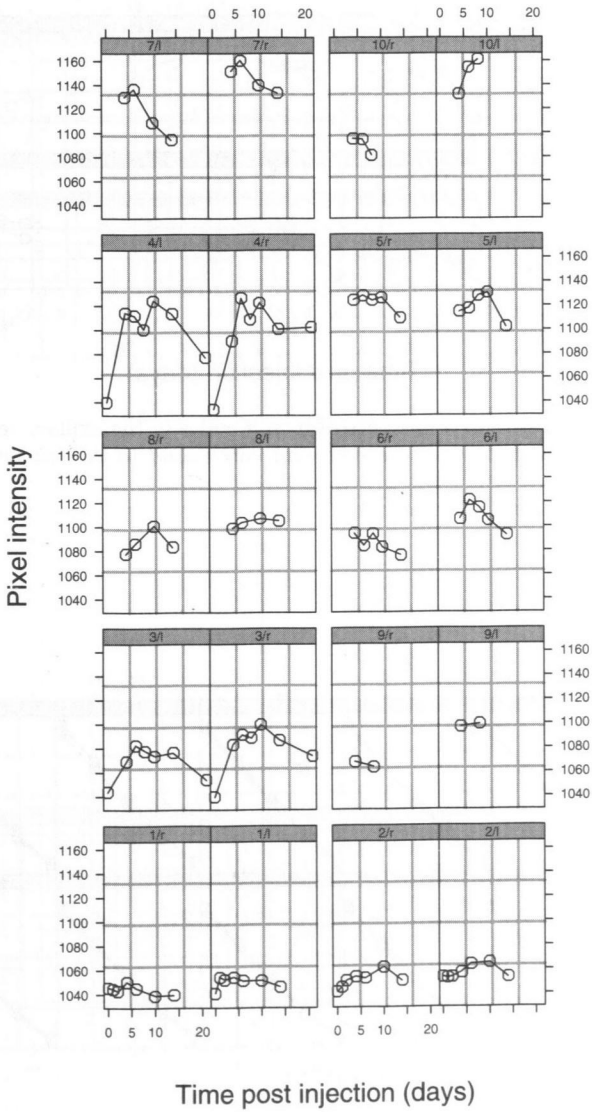

FIGURE 3.9. Mean pixel intensity of lymph nodes in the axillary region versus time by `Side` within `Dog`.

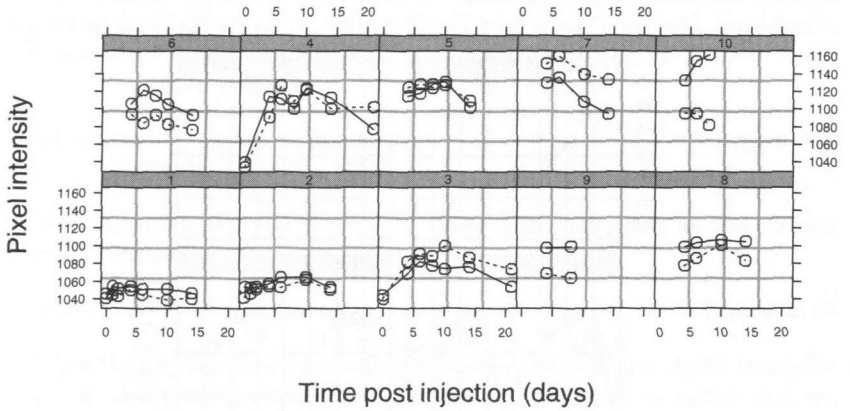

FIGURE 3.10. Mean pixel intensity of lymph nodes in the axillary region versus time by `Dog`. The two curves on each plot correspond to the left and the right sides of the `Dog`.

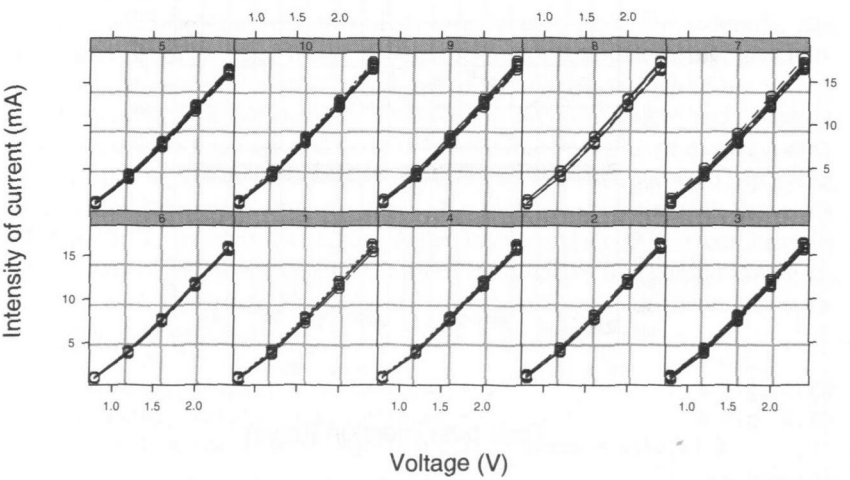

FIGURE 3.11. Current versus voltage for the `Wafer` data. The panels correspond to wafers. With each wafer the current was measured at eight different sites.

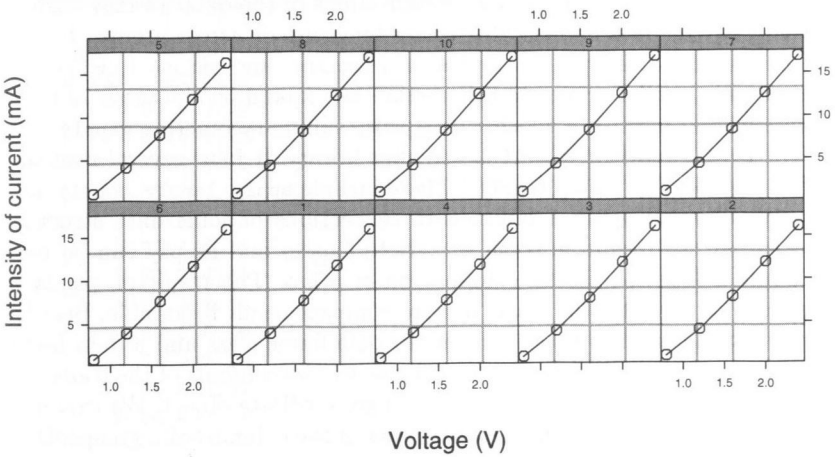

FIGURE 3.12. Mean current versus voltage at the wafer level for the `Wafer` data. Each point on each curve is the average of the current for that voltage at eight sites on the wafer.

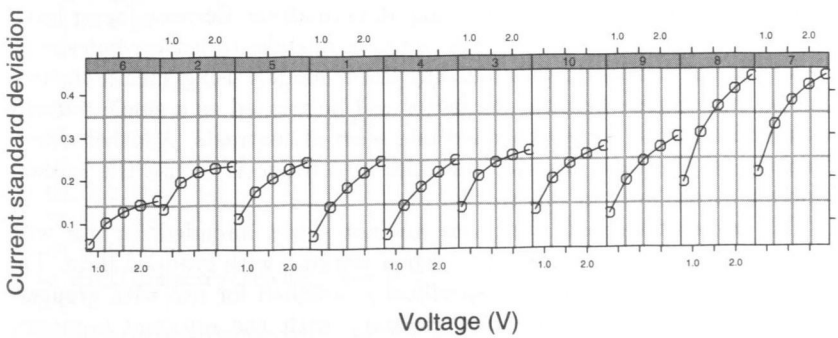

FIGURE 3.13. Standard deviation of current versus voltage at the wafer level for the `Wafer` data. Each point on each curve is the standard deviation of the current at that voltage at eight sites on the wafer.

3.4 Summaries

In addition to creating graphical presentations of the data we may wish to summarize the data numerically, either by group or across groups. In §3.1 we demonstated the use of the `table`, `gsummary`, and `unique` functions to summarize data by group. In this section we expand on the usage of these functions and introduce another groupwise summary function `gapply`.

The `gapply` function is part of the nlme library. It joins several standard S functions in the "apply family." These include `apply`, `lapply`, `tapply`, and `sapply`. They all apply a function to subsections of some data structure and gather the results in some way. Both `lapply` and `sapply` can be used to apply a function to the components of a list. The result of `lapply` is always a list; `sapply` will create a more compact result if possible. Because the groupedData class inherits from the data.frame class and a data frame can be treated as a list whose components are the columns of the frame, we can apply a function to the columns of a groupedData object. For example, we can use `sapply` to check the `data.class` of the columns of a groupedData object by

```
> sapply( ergoStool, data.class )
    effort      Type   Subject
 "numeric" "factor" "ordered"
```

We see that `effort`, the response, is a numeric variable; `Type`, the covariate, is a factor, and `Subject`, the grouping factor, is an ordered factor. We could replace `sapply` with `lapply` and get the same information, but the result would be returned as a list and would not print as compactly.

Checking the `data.class` of all variables in a data.frame or a groupedData object is an important first step in any data analysis. Because factor levels are often coded as integers, it is a common mistake to leave what should be a factor as a numeric variable. Any linear models using such a "factor" will be meaningless because the factor will be treated as a single numeric variable instead of being expanded into a set of contrasts. Another way of checking on the `data.class` of variables in a frame is to use the `summary` function shown later in this section.

The `table` and `unique` functions are not solely intended for use with grouped data, but often are useful when working with grouped data. The `gsummary` function, however, is specifically designed for use with groupedData objects. In §3.2.1 we used `gsummary` with the optional argument `invariantsOnly=TRUE` to extract only those variables in the groupedData object that are invariant within levels of the grouping factor. These could be experimental factors, like `Diet` in the `BodyWeight` data, or they could simply be additional characteristics of the groups, like `Sex` in the `Orthodont` data. The `Theoph` data are another example of a medical study where the grouping is by `Subject`. They are from a study of the pharmacokinetics of the drug theophylline, which is used to treat asthma. Each subject's weight

and dose of theophylline are given in the data. As this is a short-duration study (about 24 hours) and only one dose of theophylline was given, both `Wt` and `Dose` are invariants.

```
> gsummary( Theoph, inv = TRUE )
   Subject   Wt Dose
6        6 80.0 4.00
7        7 64.6 4.95
8        8 70.5 4.53
11      11 65.0 4.92
3        3 70.5 4.53
2        2 72.4 4.40
4        4 72.7 4.40
9        9 86.4 3.10
12      12 60.5 5.30
10      10 58.2 5.50
1        1 79.6 4.02
5        5 54.6 5.86
```

Sometimes it is distracting to have the grouping factor itself included as one of the invariants. The optional argument `omitGroupingFactor=TRUE` suppresses this. The combination of `omit = TRUE` and `inv = TRUE` can be used to check if there are any nontrivial invariants. The value returned will be `NULL` unless there are invariants other than the grouping factor.

```
> gsummary( Theoph, omit = TRUE, inv = TRUE )
     Wt Dose
6  80.0 4.00
7  64.6 4.95
8  70.5 4.53
11 65.0 4.92
3  70.5 4.53
2  72.4 4.40
4  72.7 4.40
9  86.4 3.10
12 60.5 5.30
10 58.2 5.50
1  79.6 4.02
5  54.6 5.86
> is.null(gsummary(Theoph, inv = T, omit = T)) # invariants present
[1] F
> is.null(gsummary(Oxboys, inv = T, omit = T)) # no invariants
[1] T
```

When the `invariantsOnly` argument is omitted or given the value `FALSE`, a summary of all the variables in the object is returned. The default summary is a "representative value" for each variable within each group: *numeric* variables are represented by their mean within each group and *non-numeric* variables (e.g. `factors`) by their modes (the most frequently occuring value)

within each group. When multiple modes are present, the first element of the sorted modes is returned.

```
> gsummary( Theoph )
   Subject   Wt Dose     time     conc
6        6 80.0 4.00 5.888182 3.525455
7        7 64.6 4.95 5.865455 3.910909
8        8 70.5 4.53 5.890000 4.271818
11      11 65.0 4.92 5.871818 4.510909
3        3 70.5 4.53 5.907273 5.086364
2        2 72.4 4.40 5.869091 4.823636
4        4 72.7 4.40 5.940000 4.940000
9        9 86.4 3.10 5.868182 4.893636
12      12 60.5 5.30 5.876364 5.410000
10      10 58.2 5.50 5.915455 5.930909
1        1 79.6 4.02 5.950000 6.439091
5        5 54.6 5.86 5.893636 5.782727
```

By giving a numeric summary function, which is a function that calculates a single numerical value from a numeric vector, as the argument `FUN`, we can produce other summaries. For example, we can check that the `Theoph` data are sorted according to increasing values of the maximum response with

```
> gsummary( Theoph, FUN = max, omit = TRUE )
     Wt Dose  time  conc
6  80.0 4.00 23.85  6.44
7  64.6 4.95 24.22  7.09
8  70.5 4.53 24.12  7.56
11 65.0 4.92 24.08  8.00
3  70.5 4.53 24.17  8.20
2  72.4 4.40 24.30  8.33
4  72.7 4.40 24.65  8.60
9  86.4 3.10 24.43  9.03
12 60.5 5.30 24.15  9.75
10 58.2 5.50 23.70 10.21
1  79.6 4.02 24.37 10.50
5  54.6 5.86 24.35 11.40
```

This ordering of the subjects does indeed give increasing maximum concentration of theophylline.

The `FUN` argument to `gsummary` is applied only to numeric variables in the grouped data object. Any non-numeric variables are represented by their modes within each group. A variable that is invariant within each group is represented by the (single) value that it assumes within each group. In other words, the value returned for each variable is determined according to:

- If the variable is an invariant, its value within each group is returned.

- If the variable is a factor (ordered or unordered) or of mode `character`, the mode for each group is returned.
- If the variable is numeric and not invariant, the summary function `FUN` is applied within each group and those values are returned.

When there are a large number of groups in the data, the result of `gsummary` may itself be so large as to be unwieldy. The `Quinidine` data, described in Appendix A.25, is from a study where thirteen different variables were recorded on 136 subjects.

```
> Quin.sum <- gsummary( Quinidine, omit = TRUE, FUN = mean )
> dim( Quin.sum )
[1] 136  13
```

Upon examining the first few rows

```
> Quin.sum[1:10, ]
        time conc dose interval Age Height  Weight      Race Smoke
109 30.2633   NA   NA       NA  70     67 58.000 Caucasian    no
 70  0.7500   NA   NA       NA  68     69 75.000 Caucasian    no
 23 52.0263   NA   NA       NA  75     72 108.000 Caucasian   yes
 92  8.8571   NA   NA       NA  68     72 65.000 Caucasian    yes
111 18.1638   NA   NA       NA  68     66 56.000     Latin    yes
  5 24.3750   NA   NA       NA  62     71 66.000 Caucasian    yes
 18 196.8438  NA   NA       NA  87     69 85.375 Caucasian    no
 24 31.2500   NA   NA       NA  55     69 89.000     Latin    no
  2 12.2000   NA   NA       NA  58     69 85.000     Latin    no
 88  4.7900   NA   NA       NA  85     72 77.000 Caucasian    no
    Ethanol    Heart Creatinine   glyco
109    none  No/Mild      >= 50 0.46000
 70  former  No/Mild      >= 50 1.15000
 23    none  No/Mild      >= 50 0.83875
 92  former  No/Mild      >= 50 1.27000
111  former  No/Mild      >= 50 1.23000
  5    none   Severe      >= 50 1.39000
 18    none  No/Mild       < 50 1.26000
 24  former  No/Mild      >= 50 0.57000
  2 current Moderate      >= 50 0.82000
 88    none Moderate      >= 50 0.61000
```

we see some unusual results. The summarized values of `conc`, `dose`, and `interval` are always recorded as missing values (`NA`). A less obvious peculiarity in the data is an apparent inconsistency in the numeric values of `height` and `weight`; for some reason `height` was recorded in inches while `weight` was recorded in kilograms.

Returning to the question of the missing values in `conc`, `dose`, and `interval`, the data from a single subject indicate the reason for this

```
> Quinidine[Quinidine[["Subject"]] == 3, 1:8]
   Subject    time conc dose interval Age Height Weight
17       3    0.00   NA  201       NA  67     69     69
18       3    8.00   NA  201       NA  67     69     69
19       3   16.00   NA  201       NA  67     69     69
20       3   24.00   NA  201       NA  67     69     69
21       3   32.00   NA  201       NA  67     69     69
22       3   41.25  2.4   NA       NA  67     69     69
23       3  104.00   NA  201        8  67     69     69
24       3  113.00  2.3   NA       NA  67     69     69
25       3 3865.00   NA  201        6  67     69     62
26       3 3873.00   NA  201       NA  67     69     62
27       3 3881.00   NA  201       NA  67     69     62
28       3 3889.00   NA  201       NA  67     69     62
29       3 3897.00   NA  201       NA  67     69     62
30       3 3900.00   NA   NA       NA  67     69     62
31       3 3905.00   NA  201       NA  67     69     62
32       3 3909.00  4.7   NA       NA  67     69     62
33       3 4073.00   NA  201        8  67     69     62
```

Each observation is either a record of a dosage or a record of a concentration measurement but never both. Thus, whenever `dose` is present, `conc` is missing and whenever `conc` is present, both `dose` and `interval` are missing. Because any subject in the experiment must have at least one dosage record and at least one concentration record, every subject has missing data in the `conc`, `dose`, and `interval` variables.

The default behavior of most summary functions in S is to return the value `NA` if the input vector contains any missing values. The `mean` function behaves like this and returns `NA` for every subject in each of these three variables. The behavior can be overridden in `mean` (and in several other summary functions) by giving the optional argument `na.rm = TRUE`. The default value of `FUN` in `gsummary` is `mean` with `na.rm = TRUE`.

```
> Quin.sum1 <- gsummary( Quinidine, omit = TRUE )
> Quin.sum1[1:10, 1:7]
       time   conc   dose interval    Age Height  Weight
 1   92.817 2.2000 268.71   6.0000 60.000     69 106.000
 2   12.200 1.2000 166.00       NA 58.000     69  85.000
 3 2090.015 3.1333 201.00   7.3333 67.000     69  65.294
 4  137.790 3.3667 236.39   7.3333 88.000     66  95.185
 5   24.375 0.7000 301.00       NA 62.000     71  66.000
 6    3.625 2.6000 166.00   6.0000 76.000     71  93.000
 7 1187.320 2.7833 256.22   6.0000 60.097     66  85.484
 8   20.019 2.5000 166.00       NA 52.000     71  75.000
 9   69.200 3.9500 498.00   6.0000 68.000     70  79.000
10 1717.261 3.1667 201.00   8.0000 73.154     69  79.462
```

Notice that there are still some `NA`'s in the groupwise summary for `interval`. For these subjects every value of `interval` was missing.

The function **summary** can be used with any data frame to summarize the columns according to their class. In particular, we can use it on `Quin.sum1` to obtain some summary statistics for each variable with each group (subject) counted only once.

```
> summary( Quin.sum1 )
      time                conc             dose          interval
 Min.   :   0.065   Min.   :0.50   Min.   : 83   Min.   : 5.00
 1st Qu.:  19.300   1st Qu.:1.70   1st Qu.:198   1st Qu.: 6.00
 Median :  47.200   Median :2.24   Median :201   Median : 6.00
 Mean   : 251.000   Mean   :2.36   Mean   :224   Mean   : 6.99
 3rd Qu.: 171.000   3rd Qu.:2.92   3rd Qu.:249   3rd Qu.: 8.00
 Max.   :5360.000   Max.   :5.77   Max.   :498   Max.   :12.00
                                                 NA's   :29.00
      Age             Height          Weight            Race
 Min.   :42.0   Min.   :60.0   Min.   : 41.0   Caucasian:91
 1st Qu.:61.0   1st Qu.:67.0   1st Qu.: 67.8   Latin     :35
 Median :66.0   Median :70.0   Median : 79.2   Black     :10
 Mean   :66.9   Mean   :69.6   Mean   : 79.2
 3rd Qu.:73.0   3rd Qu.:72.0   3rd Qu.: 88.2
 Max.   :92.0   Max.   :79.0   Max.   :119.0
  Smoke      Ethanol        Heart     Creatinine       glyco
 no :94   none   :90   No/Mild :55   < 50 : 36   Min.   :0.390
 yes:42   current:16   Moderate:40   >= 50:100   1st Qu.:0.885
          former :30   Severe  :41               Median :1.170
                                                 Mean   :1.210
                                                 3rd Qu.:1.450
                                                 Max.   :2.990
```

Contrast this with the result of

```
> summary( Quinidine )
...
      time             conc              dose          interval
 Min.   :   0   Min.   :   0.40   Min.   : 83   Min.   :   4.00
 1st Qu.:  16   1st Qu.:   1.60   1st Qu.:166   1st Qu.:   6.00
 Median :  60   Median :   2.30   Median :201   Median :   6.00
 Mean   : 373   Mean   :   2.45   Mean   :225   Mean   :   7.11
 3rd Qu.: 241   3rd Qu.:   3.00   3rd Qu.:249   3rd Qu.:   8.00
 Max.   :8100   Max.   :   9.40   Max.   :603   Max.   :  12.00
                NA's   :1110.00   NA's   :443   NA's   :1222.00
      Age             Height          Weight            Race
 Min.   :42.0   Min.   :60.0   Min.   : 41.0   Caucasian:968
 1st Qu.:60.0   1st Qu.:67.0   1st Qu.: 69.5   Latin     :384
 Median :66.0   Median :69.0   Median : 78.0   Black     :119
 Mean   :66.7   Mean   :69.2   Mean   : 79.7
 3rd Qu.:74.0   3rd Qu.:72.0   3rd Qu.: 89.0
 Max.   :92.0   Max.   :79.0   Max.   :119.0
  Smoke      Ethanol        Heart     Creatinine
 no :1024   none   :991   No/Mild :598   < 50 : 418
```

```
yes: 447    current:191    Moderate:375    >= 50:1053
            former :289    Severe  :498
     glyco
Min.   :0.39
1st Qu.:0.93
Median :1.23
Mean   :1.28
3rd Qu.:1.54
Max.   :3.16
```

The first summary tells us that there are 94 nonsmokers in the study and 42 smokers while the second tells us that there are 1024 total observations on the nonsmokers and 447 on the smokers.

Both summaries are useful to us in understanding these data. Because `NA`'s in `conc` and `dose` are mutually exclusive, we can see that there are at most 1110 dosage records and at most 443 concentration measurements. Because there are 136 subjects in the study, this means there are on average fewer than three concentration measurements per subject.

We can get an exact count of the number of concentrations by counting the number of nonmissing values in the entire `conc` variable. One way to do this is

```
> sum( ifelse(is.na(Quinidine[["conc"]]), 0, 1) )
[1] 361
```

This is equivalent to the somewhat more terse expression

```
> sum( !is.na(Quinidine[["conc"]]) )
[1] 361
```

because the logical values `TRUE` and `FALSE` are interpreted as 1 and 0 in arithmetic expressions.

A similar expression

```
> sum( !is.na(Quinidine[["dose"]]) )
[1] 1028
```

tells us that there are 1028 dosage events. The 164 rows that are neither dosage records nor concentration measurements are cases where values of other variables changed for the subject.

With only 361 concentration measurements for 136 subjects there are fewer than three concentration measurements per subject. To explore this further, we would like to determine the distribution of the number of concentration measurements per subject. We need to divide the data by subject (or "by group" in our general terminology) and count the number of nonmissing values in the `conc` variable for each subject. The function `gapply` is available to perform calculations such as this. It "applies" another function to some or all of the variables in a grouped data object by group.

```
> gapply( Quinidine, "conc", function(x) sum(!is.na(x)) )
 109 70 23 92 111 5 18 24 2 88 91 117 120 13 89 27 53 122 129 132
   1  1  3  1   1 2  3  1 1  1  1   3   2  1  3  1  1   1   2   3
 16 106 15 22 57 77 115 121 123 11 48 126 223 19 38 42 52 56 63 83
  1   1  1  1  3  1   4   1   1  2  2   2   6  1  1  2  1  1  4  1
 104 118 137 17 29 34 46 73 87 103 138 45 44 97 36 37 72 100 8 71
   2   2   1  1  1  1  3  2  2   1   2  3  7  2  2  3  1   3 1  5
 6 14 26 75 20 96 99 134 12 49 67 85 112 127 55 68 124 1 35 47 79
 1  3  1  3  2  3  2   1  1  3  3  1   3   3  6  3   1 2  2  5  3
 95 114 135 105 116 62 65 107 130 66 139 33 80 125 110 128 136 21
  3   2   2   1   3  4  7   4   3  1   3  3  2   1  11   2  11  2
 43 90 102 40 84 98 30 82 93 108 119 32 133 7 9 76 94 58 113 50 39
  1  1   2  2  6  2  1  3  4   1   3  1   2 6 2  6  5  1   2  3  2
 78 25 61 3 64 60 59 10 69 4 81 54 41 74 28 51
 10  2  2 3  4  4  3  6  2 6 11  4  3  3  4  6
```

The result can be a bit confusing when printed in this way. It is a named vector of the counts where the names are the values of the `Subject` variable. Thus, subjects 124 and 125 both had only a single concentration measurement. To obtain the distribution of the measurements, we apply the `table` function to this vector

```
> table( gapply(Quinidine, "conc", function(x) sum(!is.na(x))) )
  1  2  3 4 5 6 7 10 11
 46 33 31 9 3 8 2  1  3
```

We see that most of the subjects in the study have very few measurements of the response. A total of 110 out of the 136 subjects have fewer than four response measurements. This is not uncommon in such routine clinical data (data that are collected in the routine course of treatment of patients). A common consequence of having so few observations for most of the subjects is that the information gained from such a study is imprecise compared to that gained from a controlled experiment.

The second argument to `gapply` is the name or index of the variable in the `groupedData` object to which we will apply the function. When only a single name or index is given, the value passed to the function to be applied is in the form of a vector. If more than one variable is to be passed, they are passed as a data.frame. If this argument is missing, all the variables in the `groupedData` object are passed as a data.frame so the effect is to select subsets of the rows corresponding to unique values of the grouping factor from the `groupedData` object.

To illustrate this, let us determine those subjects for whom there are records that are neither dosage records nor concentration records. For each subject we must determine whether there are any records with both the `conc` variable and the `dose` variable missing.

```
> changeRecords <- gapply( Quinidine, FUN = function(frm)
+     any(is.na(frm[["conc"]]) & is.na(frm[["dose"]])) )
```

```
> changeRecords
 109 70 23 92 111 5 18 24 2 88 91 117 120 13 89 27 53 122 129 132
   F  F  F  F   F F  T  F F  F  F   F   F  F  F  F  F  F   F   T
...
 78 25 61 3 64 60 59 10 69 4 81 54 41 74 28 51
  F  F  T T  T  F  F  T  F T  T  T  T  F  T  F
```

As we see, the printed representation as a named vector of length 136 is not easy to read. It is more informative if we convert the result to a vector of levels of the `Subject` factor for which there are records that are neither dosage records nor concentration records.

```
> sort( as.numeric( names(changeRecords)[changeRecords] ) )
  [1]   3   4   7  10  14  18  28  33  37  40  41  44  45  46  47
 [16]  48  50  54  55  61  62  63  64  65  71  75  76  77  79  80
 [31]  81  82  84  94  95  96  97  98 110 112 114 118 119 127 132
 [46] 133 135 136 139 223
```

Notice that subject 3 is one of those with such a "change" record. We printed some of the variables from this subject's data on page 123. We can see there that the record in question is row 30. If we look at this row and the adjacent rows

```
> Quinidine[29:31,]
Grouped Data: conc ~ time | Subject
   Subject time conc dose interval Age Height Weight      Race
29       3 3897   NA  201       NA  67     69     62 Caucasian
30       3 3900   NA   NA       NA  67     69     62 Caucasian
31       3 3905   NA  201       NA  67     69     62 Caucasian
   Smoke Ethanol    Heart Creatinine glyco
29   yes  former Moderate       < 50  1.71
30   yes  former Moderate       < 50  1.71
31   yes  former Moderate       < 50  1.71
```

we can see that there are no changes in any of the variables except for `time` so this row is redundant. We did not cull such redundant rows from the data set because we wanted to be able directly to compare results based on these data with analyses done by others.

The data for subject 4 provide a better example.

```
> Quinidine[Quinidine[["Subject"]] == 4, ]
Grouped Data: conc ~ time | Subject
   Subject   time conc dose interval Age Height Weight  Race Smoke
45       4   0.00   NA  332       NA  88     66    103 Black   yes
46       4   7.00   NA  332       NA  88     66    103 Black   yes
47       4  13.00   NA  332       NA  88     66    103 Black   yes
48       4  19.00   NA  332       NA  88     66    103 Black   yes
49       4  21.50  3.1   NA       NA  88     66    103 Black   yes
50       4  85.00   NA  249        6  88     66    103 Black   yes
51       4  91.00  5.8   NA       NA  88     66    103 Black   yes
```

```
52      4  91.08   NA  249       NA  88     66     103 Black   yes
53      4  97.00   NA  249       NA  88     66     103 Black   yes
54      4 103.00   NA  249       NA  88     66     103 Black   yes
55      4 105.00   NA   NA       NA  88     66      92 Black   yes
56      4 109.00   NA  249       NA  88     66      92 Black   yes
57      4 115.00   NA  249       NA  88     66      92 Black   yes
58      4 145.00   NA  166       NA  88     66      92 Black   yes
59      4 151.00   NA  166       NA  88     66      92 Black   yes
60      4 156.00  3.1   NA       NA  88     66      92 Black   yes
61      4 157.00   NA  166       NA  88     66      92 Black   yes
62      4 163.00   NA  166       NA  88     66      92 Black   yes
63      4 169.00   NA  166       NA  88     66      92 Black   yes
64      4 174.75   NA  201       NA  88     66      92 Black   yes
65      4 177.00   NA   NA       NA  88     66      92 Black   yes
66      4 181.50  3.1   NA       NA  88     66      92 Black   yes
67      4 245.00   NA  201        8  88     66      92 Black   yes
68      4 249.00   NA   NA       NA  88     66      86 Black   yes
69      4 252.50  3.2   NA       NA  88     66      86 Black   yes
70      4 317.00   NA  201        8  88     66      86 Black   yes
71      4 326.00  1.9   NA       NA  88     66      86 Black   yes
   Ethanol  Heart Creatinine glyco
45    none Severe      >= 50  1.48
46    none Severe      >= 50  1.48
47    none Severe      >= 50  1.48
48    none Severe      >= 50  1.48
49    none Severe      >= 50  1.48
50    none Severe      >= 50  1.61
51    none Severe      >= 50  1.61
52    none Severe      >= 50  1.61
53    none Severe      >= 50  1.61
54    none Severe      >= 50  1.61
55    none Severe      >= 50  1.61
56    none Severe      >= 50  1.61
57    none Severe      >= 50  1.61
58    none Severe      >= 50  1.88
59    none Severe      >= 50  1.88
60    none Severe      >= 50  1.88
61    none Severe      >= 50  1.88
62    none Severe      >= 50  1.88
63    none Severe      >= 50  1.88
64    none Severe      >= 50  1.88
65    none Severe      >= 50  1.68
66    none Severe      >= 50  1.68
67    none Severe      >= 50  1.87
68    none Severe      >= 50  1.87
69    none Severe      >= 50  1.87
70    none Severe      >= 50  1.83
71    none Severe      >= 50  1.83
```

Here rows 55, 65, and 68 are in fact "change rows." Both rows 55 and 68 record changes in the subject's weight. Row 65 records a change in the `glyco` variable (i.e., the serum concentration of alpha-1-acid glycoprotein).

3.5 Chapter Summary

In this chapter we have shown examples of constructing, summarizing, and graphically displaying groupedData objects. These objects include the data, stored as a data frame, and a formula that designates different variables as a response, a primary covariate, and as one or more grouping factors. Other variables can be designated as outer or inner factors relative to the grouping factors. Accessor or extractor functions are available to extract either the formula for these variables or the value of these variables.

Informative and visually appealing trellis graphics displays of the data can be quickly and easily generated from the information that is stored with the data. The regular data summary functions in S can be applied to the data as well as the `gsummary` and `gapply` functions that are especially designed for these data.

Informative plots and summaries of the data are very useful for the preliminary phase of the statistical analysis. Many important features of the data are identified at this stage, but usually one is interested in going a step further in the analysis and fitting parametric models, such as the linear mixed-effects models described in the next chapter.

Exercises

1. In Figure 3.6 (p. 113), the twelve panels in the plot of the `CO2` data were laid out as two rows of six columns. Create a plot of these data arranged as four rows of three columns. You should insert some space between the second and third rows to separate the panels for the Mississippi plants from those for the Québec plants.

2. Use the `outer` argument to the `plot` function to produce a plot of the `CO2` data similar to Figure 8.15 (p. 369) where each panel displays the data for all three plants at some combination of `Variety` and `Treatment`. Produce two such plots of these data, each laid out as two rows by two columns. One plot should have the rows determined by `Variety` and the columns by `Treatment`. This arrangement makes it easy to assess the effect of `Treatment` within a `Variety`. The other plot should have rows determined by `Treatment` and columns by `Variety`, allowing easy assessment of the effect of `Variety` within `Treatment`.

3. The `Dialyzer` data, described in Appendix A.6 and used in some examples in §5.4 and §8.3.3, consists of observations of ultrafiltration rates at different transmembrane pressures on different subjects. The `QB` variable, which indicates the blood flow rate used for the subject, is outer to the grouping factor `Subject`.

   (a) Check if these data are balanced with respect to the number of observations on each `Subject`.

   (b) Check if these data are balanced with respect to the number of observations and the values of the transmembrane pressure. Verify your result using the `isBalanced` function.

   (c) Produce a plot of the ultrafiltration rate versus transmembrane pressure by subject.

   (d) Check with `gsummary` that `QB` is invariant within each `Subject`. Determine which `Subjects` are at which `QB` levels.

   (e) Recreate the plot from part (c) arranging for the panels corresponding to subjects at a `QB` of 200 dl/min to be separated from those at a `QB` of 300 dl/min.

   (f) Use the `outer` argument to `plot` as `outer = TRUE` or `outer = ~QB` to create a plot of ultrafiltration rate versus transmembrane pressure by subject divided into two panels according to blood flow rate. Compare this plot to Figure 5.1 (p. 215). Note the change in the aspect ratio of the panels relative to the plots in parts (c) and (d).

   (g) Use the `aspect` argument in a plot like the last one to produce a plot similar to Figure 5.1.

   (h) Use `gsummary` or `gapply` to determine the maximum observed ultrafiltration rate by subject. Produce dotplots or boxplots of this maximum rate dividing the subjects according to blood flow rate (`QB`). Does the maximum ultrafiltration rate appear to be related to the blood flow rate?

4. In the `DNase` data, described in §3.3.2, there are two measurements of the optical density at each DNase concentration within each run. When such *replicate* observations are available, one can check the assumption of constant variance for the ϵ_{ij} in a linear mixed-effects model by plotting the logarithm of the standard deviation of the replicate measurements versus the logarithm of their average (Box, Hunter and Hunter, 1978, §7.8).

   (a) Determine the average optical density at each set of replicate observations. One way to do this would be to create a copy of the groupedData object `DNase` redefining the display formula to be `density ~1 | Run/conc`. Applying `gsummary` to this object will produce the average optical density for each pair of replicates.

(b) Determine the standard deviation of the optical density at each set of replicate observations. Note that some of these standard deviations should be zero because the replicate observations are equal. Due to numerical round-off some equal replicates may produce a standard deviation that is very small but not exactly zero. Use a boxplot of the logarithm of the standard deviations to check for these. You may wish to replace those small values with zero.

(c) Plot the logarithm of the standard deviation versus the logarithm of the average. Estimate the slope of a straight line fit to these points.