# Chapter 2
# Case Studies

## 2.1 Introduction

In this chapter, we introduce the case studies that will be used to illustrate the models and R code described in the book.

The case studies come from different application domains; however, they share a few features. For instance, in all of them the study and/or sampling design generates the observations that are *grouped* according to the levels of one or more *grouping* factors. More specifically, the levels of grouping factors, i.e., subjects, schools, etc., are assumed to be randomly selected from a population being studied. This means that observations within a particular group are likely to be correlated. The correlation should be taken into account in the analysis. Also, in each case there is one (or more) continuous measurement, which is treated as the dependent variable in the models considered in this book.

In particular, we consider the following datasets:

- *Age-Related Macular Degeneration (ARMD) Trial*: A clinical trial comparing several doses of interferon-$\alpha$ and placebo in patients with ARMD. Visual acuity of patients participating in the trial was measured at baseline and at four post-randomization timepoints. The resulting data are an example of *longitudinal data* with observations grouped by subjects. We describe the related datasets in more detail in Sect. 2.2.
- *Progressive Resistance Training (PRT) Trial*: A clinical trial comparing low- and high-intensity training for improving the muscle power in elderly people. For each participant, characteristics of two types of muscle fibers were measured at two occasions, pre- and post-training. The resulting data are an example of *clustered* data, with observations grouped by subjects. We present more detailed information about the dataset in Sect. 2.3.
- *Study of Instructional Improvement (SII)*: An educational study aimed at assessing improvement in mathematics grades of first-grade pupils, as compared to their kindergarten achievements. It included pupils from randomly selected

A. Gałecki and T. Burzykowski, *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*, Springer Texts in Statistics, DOI 10.1007/978-1-4614-3900-4_2,

classes in randomly selected elementary schools. The dataset is an example of *hierarchical* data, with observations (pupils' scores) grouped within classes, which are themselves grouped in schools. We refer to Sect. 2.4 for more details about the data.

- *Flemish Community Attainment-Targets (FCAT) Study*: An educational study, in which elementary school graduates were evaluated with respect to reading comprehension in Dutch. Pupils from randomly selected schools were assessed for a set of nine attainment targets. The dataset is an example of *grouped* data, for which the grouping factors are *crossed*. We describe the dataset in more detail in Sect. 2.5.

The data from the ARMD study will be used throughout the book to illustrate various classes of LMs and corresponding R tools. The remaining case studies will be used in Part IV only, to illustrate R functions for fitting LMMs.

For each of the aforementioned case studies there is one or more datasets included into the package **nlmeU**, which accompanies this book. In the next sections of this chapter, we use the R syntax to describe the contents of these datasets. Results of exploratory analyses of the case studies are presented in Chap. 3. Note that, unlike in the other parts of the book, we are not discussing the code in much detail, as the data-processing functionalities are not the main focus of our book. The readers interested in the functionalities are referred to the monograph by Dalgaard (2008).

The R language is not particularly suited for data entry. Typically, researchers use raw data created using other software. Data are then stored in external files, e.g., in the .csv format, read into R, and prepared for the analysis. To emulate this situation, we assume, for the purpose of this chapter, that the data are stored in a .csv-format file in the "C:\temp" directory.

## 2.2 Age-Related Macular Degeneration Trial

The ARMD data arise from a randomized multi-center clinical trial comparing an experimental treatment (interferon-$\alpha$) *versus* placebo for patients diagnosed with ARMD. The full results of this trial have been reported by Pharmacological Therapy for Macular Degeneration Study Group (1997). We focus on the comparison between placebo and the highest dose (6 million units daily) of interferon-$\alpha$.

Patients with macular degeneration progressively lose vision. In the trial, visual acuity of each of 240 patients was assessed at baseline and at four post-randomization timepoints, i.e., at 4, 12, 24, and 52 weeks. Visual acuity was evaluated based on patient's ability to read lines of letters on standardized vision charts. The charts display lines of five letters of decreasing size, which the patient must read from top (largest letters) to bottom (smallest letters). Each line with at least four letters correctly read is called one "line of vision." In our analyses, we will focus on the visual acuity defined as the total number of *letters* correctly read.

Another possible approach would be to consider visual acuity measured by the number of *lines* correctly read. Note that the two approaches are closely linked, as each line of vision contains five letters.

It follows that, for each of 240 patients, we have *longitudinal data* in the form of up to five visual acuity measurements collected at different, but common to all patients, timepoints. These data will be useful to illustrate the use of LMMs for continuous, longitudinal data. We will also use them to present other classes of LMs considered in our book.

### *2.2.1 Raw Data*

We assume that the raw ARMD data are stored in the "C:\temp" directory in a .csv-format file named `armd240.data.csv`. In what follows, we also assume that our goal is to verify the contents of the data and prepare them for analysis in R.

In Panel R2.1, the data are loaded into R using the `read.csv()` function and are stored in the data frame object `armd240.data`. Note that this data frame is not included in the **nlmeU** package.

The number of rows (records) and columns (variables) in the object `armd240.data` is obtained using the function `dim()`. The data frame contains 240 observations and 9 variables. The names of the variables are displayed using the `names()` function. All the variables are of class *integer*. By applying the function `str()`, we get a summary description of variables in the `armd240.data` data. In particular, for each variable, we get its class and a listing of the first few values.

The variable `subject` contains patients' identifiers. Treatment identifiers are contained in the variable `treat`. Variables `visual0`, `visual4`, `visual12`, `visual24`, and `visual52` store visual acuity measurements obtained at baseline and week 4, 12, 24, and 52, respectively. Variables `lesion` and `line0` contain additional information, which will not be used for analysis in our book.

Finally, at the bottom of Panel R2.1, we list the first three rows of the data frame `armd240.data` with the help of the `head()` function. To avoid splitting lines of the output and to make the latter more transparent, we shorten variables' names using the `abbreviate()` function. After printing the contents of the first three rows and before proceeding further, we reinstate the original names. Note that we apply a similar sequence of R commands in many other R panels across the book to simplify the displayed output.

Based on the output, we note that the data frame contains one record for each patient. The record includes all information obtained for the patient. In particular, each record contains five variables with visual acuity measurements, which are, essentially, of the same format. This type of data storage, with one record per subject, is called the "wide" format. An alternative is the "long" format with multiple records per subject. We will discuss the formats in the next section.

**R2.1** *ARMD Trial*: Loading raw data from a .csv-format file into the `armd240.data` object and checking their contents

```
> dataDir <- file.path("C:", "temp")       # Data directory
> fp <-                                    # File path
+   file.path(dataDir, "armd240.data.csv")
> armd240.data <-                          # Read data
+   read.csv(fp, header = TRUE)
> dim(armd240.data)                        # No. of rows and cols
  [1] 240   9
> (nms <- names(armd240.data))             # Variables' names
  [1] "subject"  "treat"    "lesion"   "line0"    "visual0"
  [6] "visual4"  "visual12" "visual24" "visual52"
> unique(sapply(armd240.data, class))      # Variables' classes
  [1] "integer"
> str(armd240.data)                        # Data structure
  'data.frame':   240 obs. of  9 variables:
   $ subject : int  1 2 3 4 5 6 7 8 9 10 ...
   $ treat   : int  2 2 1 1 2 2 1 1 2 1 ...
   $ lesion  : int  3 1 4 2 1 3 1 3 2 1 ...
   $ line0   : int  12 13 8 13 14 12 13 8 12 10 ...
   $ visual0 : int  59 65 40 67 70 59 64 39 59 49 ...
   $ visual4 : int  55 70 40 64 NA 53 68 37 58 51 ...
   $ visual12: int  45 65 37 64 NA 52 74 43 49 71 ...
   $ visual24: int  NA 65 17 64 NA 53 72 37 54 71 ...
   $ visual52: int  NA 55 NA 68 NA 42 65 37 58 NA ...
> names(armd240.data) <- abbreviate(nms)   # Variables' names shortened
> head(armd240.data, 3)                    # First 3 records
    sbjc tret lesn lin0 vsl0 vsl4 vs12 vs24 vs52
  1    1    2    3   12   59   55   45   NA   NA
  2    2    2    1   13   65   70   65   65   55
  3    3    1    4    8   40   40   37   17   NA
> names(armd240.data) <- nms               # Variables' names reinstated
```

## 2.2.2 *Data for Analysis*

In this section, we describe auxiliary data frames, namely, `armd.wide`, `armd0`, and `armd`, which were derived from `armd240.data` for the purpose of analyses of the ARMD data that will be presented later in the book. The data frames are included in the package **nlmeU**. In what follows, we present the structure, contents, and for illustration purposes, how the data were created.

#### 2.2.2.1 *Data in the "Wide" Format: The Data Frame* `armd.wide`

Panel R2.2 presents the structure and the contents of the `armd.wide` data frame.

Note that the data are loaded into R using the `data()` function, without the need for attaching the package **nlmeU**. The data frame contains 10 variables. In particular, it includes variables `visual0`, `visual4`, `visual12`, `visual24`, `visual52`, `lesion`, and `line0`, which are exactly the same as those in the `armd240.data`. In contrast to the `armd240.data` data frame, it contains three factors: `subject`, `treat.f`, and `miss.pat`. The first two contain patient's identifier and treatment. They are constructed from the corresponding numeric variables available in `armd240.data`. The factor `miss.pat` is a new variable and contains a missing-pattern identifier, i.e., a character string that indicates which of the four post-randomization measurements of visual acuity are missing for a particular patient. The missing values are marked by `X`. Thus, for instance, for the patient with the `subject` identifier equal to 1, the pattern is equal to `--XX`, because there is no information about visual acuity at weeks 24 and 52. On the other hand, for the patient with the `subject` identifier equal to 6, there are no missing visual acuity

**R2.2** *ARMD Trial*: The structure and contents of data frame `armd.wide` stored in the "wide" format

```
> data(armd.wide, package = "nlmeU")          # armd.wide loaded
> str(armd.wide)                              # Structure of data
  'data.frame':   240 obs. of  10 variables:
   $ subject : Factor w/ 240 levels "1","2","3","4",..: 1 2 3 4 5 6 ...
  ...     [snip]
   $ treat.f : Factor w/ 2 levels "Placebo","Active": 2 2 1 1 2 2 1 ...
   $ miss.pat: Factor w/ 9 levels "----","---X",..: 4 1 2 1 9 1 1 1 ...
> head(armd.wide)                             # First few records
    subject lesion line0 visual0 visual4 visual12 visual24
  1       1      3    12      59      55       45       NA
  ...     [snip]
  6       6      3    12      59      53       52       53
    visual52 treat.f miss.pat
  1       NA  Active     --XX
  ...     [snip]
  6       42  Active     ----
> (facs <- sapply(armd.wide, is.factor))      # Factors indicated
   subject   lesion    line0  visual0  visual4 visual12 visual24
      TRUE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
  visual52  treat.f miss.pat
     FALSE     TRUE     TRUE
> names(facs[facs == TRUE])                   # Factor names displayed
  [1] "subject"  "treat.f"  "miss.pat"
```

measurements, and hence the value of the `miss.pat` factor is equal to `----`. At the bottom of Panel R2.2, we demonstrate how to extract the names of the factors from a data frame.

Panel R2.3 presents the syntax used to create factors `treat.f` and `miss.pat` in the `armd.wide` data frame. The former is constructed in Panel R2.3a from the variable `treat` from the data frame `armd240.data` using the function `factor()`. The factor `treat.f` has two levels, `Placebo` and `Active`, which correspond to the values of 1 and 2, respectively, of `treat`.

The factor `miss.pat` is constructed in Panel R2.3b with the help of the function `missPat()` included in the **nlmeU** package. The function returns a character vector of length equal to the number of rows of the matrix created by column-wise concatenation of the vectors given as arguments to the function. The elements of the resulting vector indicate the occurrence of missing values in the rows of the matrix. In particular, the elements are character strings of the length equal to the number of the columns (vectors). As shown in Panel R2.2, the strings contain characters "`-`" and "`X`", where the former indicates a nonmissing value in the corresponding column of the matrix, while the latter indicates a missing value. Thus, application

**R2.3** *ARMD Trial*: Construction of factors `treat.f` and `miss.pat` in the data frame `armd.wide`. The data frame `armd240.data` was created in Panel R2.1

**(a)** *Factor `treat.f`*

```
> attach(armd240.data)                    # Attach data
> treat.f <-                              # Factor created
+   factor(treat, labels = c("Placebo", "Active"))
> levels(treat.f)                         # (1) Placebo, (2) Active
  [1] "Placebo" "Active"
> str(treat.f)
   Factor w/ 2 levels "Placebo","Active": 2 2 1 1 2 2 1 1 2 1 ...
```

**(b)** *Factor `misspat`*

```
> miss.pat <-                             # Missing patterns
+   nlmeU:::missPat(visual4, visual12, visual24, visual52)
> length(miss.pat)                        # Vector length
  [1] 240
> mode(miss.pat)                          # Vector mode
  [1] "character"
> miss.pat                                # Vector contents
    [1] "--XX" "----" "---X" "----" "XXXX" "----" "----" "----"
  ...     [snip]
  [233] "----" "----" "----" "----" "----" "----" "----" "----"
> detach(armd240.data)                   # Detach armd240.data
```

of the function to variables visual4, visual12, visual24, and visual52 from the data frame armd240.data results in a character vector of length 240 with strings containing four characters as the elements. The elements of the resulting miss.pat vector indicate that, for instance, for the first patient in the data frame armd240.data visual acuity measurements at week 24 and 52 were missing, while for the fifth patient, no visual acuity measurements were obtained at any post-randomization visit.

Note that we used the nlmeU:::missPat() syntax, which allowed us to invoke the missPat() function without attaching the **nlmeU** package.

#### *2.2.2.2 Data in the "Long" Format: The Data Frame* armd0

In addition to the armd.wide data stored in the "wide" format, we will need data in the "longitudinal" (or "long") format. In the latter format, for each patient, there are multiple records containing visual acuity measurements for separate visits. An example of data in "long" format is stored in the data frame armd0. It was obtained from the armd.wide data using functions melt() and cast() from the package **reshape** (Wickham, 2007).

Panel R2.4 presents the contents and structure of the data frame armd0. The data frame includes eight variables and 1,107 records. The contents of variables subject, treat.f, and miss.pat are the same as in armd.wide, while visual0 contains the value of the visual acuity measurement at baseline. Note that the values of these four variables are repeated across the multiple records corresponding to a particular patient. On the other hand, the records differ with respect to the values of variables time.f, time, tp, and visual. The first three of those four variables are different forms of an indicator of the visit time, while visual contains the value of the visual acuity measurement at the particular visit. We note that having three variables representing time visits is not mandatory, but we created them to simplify the syntax used for analyses in later chapters.

The numerical variable time provides the actual week, at which a particular visual acuity measurement was taken. The variable time.f is a corresponding ordered factor, with levels Baseline, 4wks, 12wks, 24wks, and 52wks. Finally, tp is a numerical variable, which indicates the position of the particular measurement visit in the sequence of the five possible measurements. Thus, for instance, tp=0 for the baseline measurement and tp=4 for the fourth post-randomization measurement at week 52.

Interestingly enough, visual acuity measures taken at baseline are stored both in visual0 and in selected rows of the visual variables. This structure will prove useful when creating the armd data frame containing rows with post-randomization visual acuity measures, while keeping baseline values.

The "long" format is preferable for storing longitudinal data over the "wide" format. We note that storing of the visual acuity measurements in the data frame armd.wide requires the use of six variables, i.e., subject and the five variables containing the values of the measurements. On the other hand, storing the same

**R2.4** *ARMD Trial*: The structure and contents of the data frame `armd0` stored in the "long" format

```
> data(armd0, package = "nlmeU")        # From nlmeU package
> dim(armd0)                            # No. of rows and cols
  [1] 1107    8
> head(armd0)                           # First six records
    subject treat.f visual0 miss.pat   time.f time visual tp
  1       1  Active      59     --XX Baseline    0     59  0
  2       1  Active      59     --XX     4wks    4     55  1
  3       1  Active      59     --XX    12wks   12     45  2
  4       2  Active      65     ---- Baseline    0     65  0
  5       2  Active      65     ----     4wks    4     70  1
  6       2  Active      65     ----    12wks   12     65  2
> names(armd0)                          # Variables' names
  [1] "subject"  "treat.f"  "visual0"  "miss.pat" "time.f"
  [6] "time"     "visual"   "tp"
> str(armd0)                            # Data structure
  'data.frame': 1107 obs. of  8 variables:
   $ subject : Factor w/ 240 levels "1","2","3","4",..: 1 1 1 2 2 2 ...
   $ treat.f : Factor w/ 2 levels "Placebo","Active": 2 2 2 2 2 2 ...
   $ visual0 : int  59 59 59 65 65 65 65 65 40 40 ...
   $ miss.pat: Factor w/ 9 levels "----","---X",..: 4 4 4 1 1 1 1 1 ...
   $ time.f  : Ord.factor w/ 5 levels "Baseline"<"4wks"<..: 1 2 3 1 ...
   $ time    : num  0 4 12 0 4 12 24 52 0 4 ...
   $ visual  : int  59 55 45 65 70 65 65 55 40 40 ...
   $ tp      : num  0 1 2 0 1 2 3 4 0 1 ...
```

information in the data frame `armd0` requires only three variables, i.e., `subject`, `time`, and `visual`. Of course, this is achieved at the cost of including more rows in the `armd0` data frame, i.e., 1,107, as compared to 240 records in `armd.wide`.

We also note that variables, with values invariant within subjects, such as `treat.f`, `visual0`, are referred to as *time-fixed*. In contrast, `time`, `tp`, and `visual` are called *time-varying*. This distinction will have important implications for the specification of the models and interpretation of the results.

#### *2.2.2.3 Subsetting Data in the "Long" Format: The Data Frame* `armd`

Data frame `armd` was also stored in a "long" format and was created from the `armd0` data frame by omitting records corresponding to the baseline visual acuity measurements.

Panel R2.5 presents the syntax used to create the data frame `armd`. In particular, the function `subset()` is used to remove the baseline measurements, by selecting

**R2.5** *ARMD Trial*: Creation of the data frame `armd` from `armd0`

```
> auxDt <- subset(armd0, time > 0)          # Post-baseline measures
> dim(auxDt)                                # No. of rows & cols
  [1] 867   8
> levels(auxDt$time.f)                      # Levels of treat.f
  [1] "Baseline" "4wks"     "12wks"    "24wks"    "52wks"
> armd <- droplevels(auxDt)                 # Drop unused levels
> levels(armd$time.f)                       # Baseline level dropped
  [1] "4wks"  "12wks" "24wks" "52wks"
> armd <-                                   # Data modified
+   within(armd,
+          {
+            contrasts(time.f) <-           # Contrasts assigned
+              contr.poly(4, scores = c(4, 12, 24, 52))
+          })
> head(armd)                                # First six records
    subject treat.f visual0 miss.pat time.f time visual tp
  2       1  Active      59      --XX   4wks    4     55  1
  3       1  Active      59      --XX  12wks   12     45  2
  5       2  Active      65      ----   4wks    4     70  1
  6       2  Active      65      ----  12wks   12     65  2
  7       2  Active      65      ----  24wks   24     65  3
  8       2  Active      65      ----  52wks   52     55  4
```

only the records, for which `time>0`, from the object `armd0`. By removing the baseline measurements, we reduce the number of records from 1,107 (see Panel R2.4) to 867.

While subsetting the data, care needs to be taken regarding the levels of the `time.f` and, potentially, other factors. In the data frame `armd0`, the factor had five levels. In Panel R2.5, we extract the factor `time.f` from the auxiliary data frame `auxDt`. Note that, in the data frame, the level `Baseline` is not used in any of the rows. For many functions in R it would not be a problem, but sometimes the presence of an unused level in the definition of a factor may lead to unexpected results. Therefore, it is prudent to drop the unused level from the definition of the `time.f` factor, by applying the function `droplevels()`. It is worth noting that, using the `droplevels()` function, the number of levels of the factors `subject` and `miss.pat` is also affected (not shown).

After modifying the aforementioned factors, we store the resulting data in the data frame `armd`. We also assign orthogonal polynomial contrasts to the factor `time.f` using syntax of the form "`contrasts(`*factor*`)<-`*contr.function*". We will revisit the issue of assigning contrasts to a factor in Panel R5.9 (Sect. 5.3.2).

The display of the first six records of `armd` in Panel R2.5 confirms that the data do not include the records corresponding to the baseline measurements of visual acuity.

Of course, the information about the values of the measurements is still available in the variable `visual0`.

Both data frames `armd0` and `armd`, introduced in this section, are stored in "long" format. The `armd0` will be primarily used for exploratory data analyses (Sect. 3.2). On the other hand, `armd` will be the primary data frame used for the analyses throughout the entire book.

## 2.3 Progressive Resistance Training Study

The PRT data originate from a randomized trial aimed for devising evidence-based methods for improving and measuring the mobility and muscle power of elderly men and women in the 70+ age category (Claflin *et al.*, 2011). The working hypothesis was that a 12-week program of PRT would increase: (a) the power output of the overall musculature associated with movements of the ankles, knees, and hips; (b) the cross-sectional area and the force and power of permeabilized single fibers obtained from the *vastus lateralis* muscle; and (c) the ability of young and elderly men and women to safely arrest standardized falls. The training consisted of repeated leg extensions by shortening contractions of the leg extensor muscles against a resistance that was increased as the subject trained using a specially designed apparatus.

In the trial, healthy young (21–30 years) and older (65–80 years) male and female subjects were randomized between a "high" and "low" intensity of a 12-week PRT intervention. Randomization was stratified by age group (young or old) and sex. In total, the dataset used in our book includes 63 subjects.

For each subject, multiple measurements characterizing two types of muscle fibers were obtained before and after the 12-week PRT. The resulting data are thus an example of *clustered* data. In particular, the measurements for a given characteristic of muscle fibers for each subject correspond to a $2 \times 2$ factorial design, with fiber type (1, 2) and occasion (pre-training, post-training) as the two design factors, which has important implications for the data analysis (Chap. 17).

### *2.3.1 Raw Data*

We assume that subjects' characteristics and experimental measurements are contained in external files named `prt.subjects.data.csv` and `prt.fiber.data.csv`, respectively.

In Panel R2.6, we present the syntax for loading and inspecting the two datasets. As can be seen from the output presented in Panel R2.6a, the file `prt.subjects.data.csv` contains information about 63 subjects, with one record per subject. It includes one character variable and five numeric variables, three of which are integer-valued. The variable `id` contains subjects' identifiers, `gender`

**R2.6** *PRT Trial*: Loading raw data from .csv files into objects `prt.subjects.data` and `prt.fiber.data`. The object `dataDir` was created in Panel R2.1

**(a)** *Loading and inspecting data from the `prt.subjects.data.csv` file*

```
> fp <- file.path(dataDir, "prt.subjects.data.csv")
> prt.subjects.data <- read.csv(fp, header = TRUE, as.is = TRUE)
> dim(prt.subjects.data)
  [1] 63  6
> names(prt.subjects.data)
  [1] "id"       "gender"   "ageGrp"   "trainGrp" "height"
  [6] "weight"
> str(prt.subjects.data)
  'data.frame':   63 obs. of  6 variables:
   $ id      : int  5 10 15 20 25 35 45 50 60 70 ...
   $ gender  : chr  "F" "F" "F" "F" ...
   $ ageGrp  : int  0 0 1 1 1 0 0 1 0 0 ...
   $ trainGrp: int  0 1 1 1 1 0 0 0 0 1 ...
   $ height  : num  1.56 1.71 1.67 1.55 1.69 1.69 1.72 1.61 1.71 ...
   $ weight  : num  61.9 66 70.9 62 79.1 74.5 89 68.9 62.9 68.1 ...
> head(prt.subjects.data, 4)
    id gender ageGrp trainGrp height weight
  1  5      F      0        0   1.56   61.9
  2 10      F      0        1   1.71   66.0
  3 15      F      1        1   1.67   70.9
  4 20      F      1        1   1.55   62.0
```

**(b)** *Loading and inspecting data from the `prt.fiber.data.csv` file*

```
> fp <- file.path(dataDir, "prt.fiber.data.csv")
> prt.fiber.data <- read.csv(fp, header = TRUE)
> str(prt.fiber.data)
  'data.frame':   2471 obs. of  5 variables:
   $ id           : int  5 5 5 5 5 5 5 5 5 5 ...
   $ fiber.type   : int  1 1 2 1 2 1 1 1 2 1 ...
   $ train.pre.pos: int  0 0 0 0 0 0 0 0 0 0 ...
   $ iso.fo       : num  0.265 0.518 0.491 0.718 0.16 0.41 0.371 ...
   $ spec.fo      : num  83.5 132.8 161.1 158.8 117.9 ...
> head(prt.fiber.data, 4)
    id fiber.type train.pre.pos iso.fo spec.fo
  1  5          1             0  0.265    83.5
  2  5          1             0  0.518   132.8
  3  5          2             0  0.491   161.1
  4  5          1             0  0.718   158.8
```

identifies sex, ageGrp indicates the age group, and trainGrp identifies the study group. Finally, height and weight contain the information of subjects' height and weight at baseline.

Note that the as.is argument used in the read.csv() function is set to TRUE. Consequently, it prevents the creation of a factor from a character variable. This applies to the gender variable, which is coded using the "F" and "M" characters.

The output in Panel R2.6b presents the contents of the file prt.fiber.data.csv. The file contains 2,471 records corresponding to individual muscle fibers. It includes five numeric variables, three of which are integer-valued. The variable id contains subjects' identifiers, fiber.type identifies the type of fiber, while train.pre.pos indicates whether the measurement was taken pre- or post-training. Finally, iso.fo and spec.fo contain the measured values of two characteristics of muscle fibers. These two variables will be treated as outcomes of interest in the analyses presented in Part IV of the book.

### 2.3.2 *Data for Analysis*

In Panels R2.7 and R2.8, we present the syntax used to create the prt dataset that will be used for analysis.

First, in Panel R2.7, we prepare data for merging. Specifically, in Panel R2.7a, we create the data frame prt.subjects, corresponding to prt.subjects.data, with several variables added and modified. Toward this end, we use the function within(), which applies all the modifications to the data frame prt.subjects.data. In particular, we replace the variable id by a corresponding factor. We also define the numeric variable bmi, which contains subject's body mass index (BMI), expressed in units of kg/m$^2$. Moreover, we create the factors sex.f, age.f, and prt.f, which correspond to the variables gender, ageGrp, and trainGrp, respectively. Finally, we remove the variables weight, height, trainGrp, ageGrp, and gender, and store the result as the data frame prt.subjects. The contents of the data frame is summarized using the str() function.

In Panel R2.7b, we create the data frame prt.fiber. It corresponds to prt.fiber.data, but instead of the variables fiber.type and train.pre.pos, it includes the factors fiber.f and occ.f. Also, a subject's identifier id is stored as a factor.

In Panel R2.8, we construct the data frame prt by merging the data frames prt.subjects and prt.fiber created in Panel R2.7. As a result, we obtain data stored in the "long" format with 2,471 records and nine variables. The contents of the first six rows of the data frame prt are displayed with the help of the head() function.

**R2.7** *PRT Trial*: Construction of the data frame `prt`. Creating data frames `prt.subjects` and `prt.fiber` containing subjects' and fiber measurements. Data frames `prt.subjects.data` and `prt.fiber.data` were created in Panel R2.6

**(a)** *Subjects' characteristics*

```
> prt.subjects <-
+   within(prt.subjects.data,
+          {
+           id <- factor(id)
+           bmi <- weight/(height^2)
+           sex.f <- factor(gender, labels = c("Female", "Male"))
+           age.f <- factor(ageGrp, labels = c("Young", "Old"))
+           prt.f <-
+             factor(trainGrp, levels = c("1", "0"),
+                    labels = c("High", "Low"))
+           gender <- ageGrp <- trainGrp <- height <- weight <- NULL
+          })
> str(prt.subjects)
  'data.frame':   63 obs. of  5 variables:
   $ id   : Factor w/ 63 levels "5","10","15",..: 1 2 3 4 5 6 7 8 9 ...
   $ prt.f: Factor w/ 2 levels "High","Low": 2 1 1 1 1 2 2 2 2 1 ...
   $ age.f: Factor w/ 2 levels "Young","Old": 1 1 2 2 2 1 1 2 1 1 ...
   $ sex.f: Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 2 1 2 2 ...
   $ bmi  : num  25.4 22.6 25.4 25.8 27.7 ...
```

**(b)** *Fiber measurements*

```
> prt.fiber  <-
+   within(prt.fiber.data,
+          {
+           id <- factor(id)
+           fiber.f <-
+             factor(fiber.type, labels = c("Type 1", "Type 2"))
+           occ.f <-
+             factor(train.pre.pos, labels = c("Pre", "Pos"))
+           fiber.type <- train.pre.pos <- NULL
+          })
> str(prt.fiber)
  'data.frame':   2471 obs. of  5 variables:
   $ id     : Factor w/ 63 levels "5","10","15",..: 1 1 1 1 1 1 1 1 ...
   $ iso.fo : num  0.265 0.518 0.491 0.718 0.16 0.41 0.371 0.792 ...
   $ spec.fo: num  83.5 132.8 161.1 158.8 117.9 ...
   $ occ.f  : Factor w/ 2 levels "Pre","Pos": 1 1 1 1 1 1 1 1 1 ...
   $ fiber.f: Factor w/ 2 levels "Type 1","Type 2": 1 1 2 1 2 1 1 ...
```

**R2.8** *PRT Trial*: Construction of the data frame `prt` by merging `prt.subjects` with `prt.fiber` containing subjects' and fiber data. Data `prt.subjects` and `prt.fiber` were created in Panel R2.7

```
> prt <- merge(prt.subjects, prt.fiber, sort = FALSE)
> dim(prt)
  [1] 2471    9
> names(prt)
  [1] "id"      "prt.f"   "age.f"   "sex.f"   "bmi"     "iso.fo"
  [7] "spec.fo" "occ.f"   "fiber.f"
> head(prt)
    id prt.f age.f  sex.f    bmi iso.fo spec.fo occ.f fiber.f
  1  5   Low Young Female 25.436  0.265    83.5   Pre  Type 1
  2  5   Low Young Female 25.436  0.518   132.8   Pre  Type 1
  3  5   Low Young Female 25.436  0.491   161.1   Pre  Type 2
  4  5   Low Young Female 25.436  0.718   158.8   Pre  Type 1
  5  5   Low Young Female 25.436  0.160   117.9   Pre  Type 2
  6  5   Low Young Female 25.436  0.410    87.8   Pre  Type 1
```

## 2.4 The Study of Instructional Improvement Project

The SII was carried out to assess the math achievement scores of first- and third-grade pupils in randomly selected classrooms from a national US sample of elementary schools (Hill et al., 2005). The dataset includes results for 1,190 first-grade pupils sampled from 312 classrooms in 107 schools.

The SII data exhibit a *hierarchical* structure. That is, pupils are grouped in classes, which, in turn, are grouped within schools. This structure implies that, e.g., scores for pupils from the same class are likely correlated. The correlation should be taken into account in the analysis.

### *2.4.1 Raw Data*

As a starting point, we use the data frame `classroom`, which can be found in the **WWGbook** package.

In Panel R2.9, we investigate the structure and contents of the data frame. As it can be seen from the results of application of the `dim()` function, the data frame contains 1,190 records and 12 variables.

The names of the variables are listed with the help of the `names()` function. The contents of the variables, described on p. 118 of the book by West et al. (2007), are as follows:

**R2.9** *SII Project*: The structure and contents of the data frame `classroom` from the **WWGbook** package

```
> data(classroom, package = "WWGbook")
> dim(classroom)                          # Number of rows & variables
  [1] 1190   12
> names(classroom)                        # Variable names
   [1] "sex"       "minority" "mathkind" "mathgain" "ses"
   [6] "yearstea" "mathknow" "housepov" "mathprep" "classid"
  [11] "schoolid" "childid"
> classroom                               # Raw data
      sex minority mathkind mathgain   ses yearstea mathknow
  1     1        1      448       32  0.46        1       NA
  2     0        1      460      109 -0.27        1       NA
  3     1        1      511       56 -0.03        1       NA
  ...     [snip]
  1189  0        0      473       44 -0.03       25     0.50
  1190  1        0      453       69 -0.37       25     0.50
      housepov mathprep classid schoolid childid
  1      0.082     2.00     160        1       1
  2      0.082     2.00     160        1       2
  3      0.082     2.00     160        1       3
  ...     [snip]
  1189   0.177     2.00     239      107    1189
  1190   0.177     2.00     239      107    1190
> str(classroom)
  'data.frame':   1190 obs. of  12 variables:
   $ sex     : int  1 0 1 0 0 1 0 0 1 0 ...
   $ minority: int  1 1 1 1 1 1 1 1 1 1 ...
   $ mathkind: int  448 460 511 449 425 450 452 443 422 480 ...
   $ mathgain: int  32 109 56 83 53 65 51 66 88 -7 ...
   $ ses     : num  0.46 -0.27 -0.03 -0.38 -0.03 0.76 -0.03 0.2 0.64 ...
   $ yearstea: num  1 1 1 2 2 2 2 2 2 ...
   $ mathknow: num  NA NA NA -0.11 -0.11 -0.11 -0.11 -0.11 -0.11 ...
   $ housepov: num  0.082 0.082 0.082 0.082 0.082 0.082 0.082 0.082 ...
   $ mathprep: num  2 2 2 3.25 3.25 3.25 3.25 3.25 3.25 3.25 ...
   $ classid : int  160 160 160 217 217 217 217 217 217 217 ...
   $ schoolid: int  1 1 1 1 1 1 1 1 1 1 ...
   $ childid : int  1 2 3 4 5 6 7 8 9 10 ...
```

- School-level variables:
  - `schoolid`: school's ID number
  - `housepov`: % of households in the neighborhood of the school below the poverty level

- Classroom-level variables:
  - `classid`: classroom's ID number
  - `yearstea`: years of teacher's experience in teaching in the first grade
  - `mathprep`: the number of preparatory courses on the first-grade math contents and methods followed by the teacher
  - `mathknow`: teacher's knowledge of the first-grade math contents (higher values indicate a higher knowledge of the contents)
- Pupil-level variables:
  - `childid`: pupil's ID number
  - `mathgain`: pupil's gain in the math achievement score from the spring of kindergarten to the spring of first grade
  - `mathkind`: pupil's math score in the spring of the kindergarten year
  - `sex`: an indicator variable for sex
  - `minority`: an indicator variable for the minority status
  - `ses`: pupil's socioeconomic status

The outcome of interest is contained in the variable `mathgain`.

The abbreviated display of the contents of the `classroom` data frame shows that the data are stored with one record for each pupil. The output of the `str()` function indicates that the variables, contained in the data frame, are all either numeric or integer-valued. Note, however, that we do not have information about, e.g., the number of distinct levels of the integer-valued variables.

### *2.4.2 Data for Analysis*

In the analyses presented later in the book, we will be using the data frame `SIIdata`, which is included in the **nlmeU** package. It was constructed from the data frame `classroom` using the syntax shown in Panel R2.10.

Essentially, the data frame `SIIdata` contains all the variables from `classroom`. However, variables `sex`, `minority`, `schoolid`, `classid`, and `childid` are replaced by corresponding factors. Note that, in Panel R2.10, we illustrate various forms of the syntax for the function `factor()`, which can be used to create a factor. In this way, we can explain the process of construction of a factor in more detail.

For the variable `sex`, we explicitly use both the `levels` and `labels` arguments of the function `factor()`. In this way, we fully control the mapping of the values of the original variable to the factor levels and to their labels. In the syntax shown in Panel R2.10, the value `0` of the variable `sex` from the `classroom` data is considered the first level and is assigned the label `M`. On the other hand, the value `1` is considered the second level and is labeled `F`.

**R2.10** *SII Project*: Creation of the data frame `SIIdata` from the `classroom` data

```
> SIIdata <-
+   within(classroom,
+          {
+           sex <-                              # 0 -> 1(M), 1 -> 2(F)
+             factor(sex, levels = c(0, 1), labels = c("M", "F"))
+           minority <-                         # 0 -> 1(No), 1 -> 2(Yes)
+             factor(minority, labels = c("Mnrt:No", "Mnrt:Yes"))
+           schoolid <- factor(schoolid)
+           classid <- factor(classid)
+           childid <- factor(childid)
+           })
> str(SIIdata)
  'data.frame':   1190 obs. of  12 variables:
   $ sex     : Factor w/ 2 levels "M","F": 2 1 2 1 1 2 1 1 2 1 ...
   $ minority: Factor w/ 2 levels "Mnrt:No","Mnrt:Yes": 2 2 2 2 2 2 ...
  ...     [snip]
   $ classid : Factor w/ 312 levels "1","2","3","4",..: 160 160 160 ...
   $ schoolid: Factor w/ 107 levels "1","2","3","4",..: 1 1 1 1 1 1 ...
   $ childid : Factor w/ 1190 levels "1","2","3","4",..: 1 2 3 4 5 6 ...
```

It is worth noting that, in the printout of the structure of `SIIdata`, the variable `sex` is defined as a factor with two levels: `M` (first) and `F` (second). In the listing of the first values of the variable, obtained using the `str()` function, we only see the numerical representation (the ranks) of the levels, i.e., `1` or `2`. Thus, the information about the coding, `0` and `1`, of the original variable `sex` from the `classroom` data frame is lost. Of course, if needed, we could recover it based on the specified value of the `levels` argument.

For the variable `minority`, we only use the `labels` argument of the function `factor()`. Thus, by default, the `levels` argument is obtained by taking the unique values of the variable, i.e., `0` and `1`; representing them as characters "`0`" and "`1`", respectively; and then sorting them according to an increasing order of the numeric values of the variable. Thus, the assumed (ordered) levels are "`0`" (first) and "`1`" (second). Subsequently, the `labels` argument assigns the label `"Mnrt:No"` to the first level ("`0`") and `"Mnrt:Yes"` to the second level ("`1`"). In the printout of the structure of `SIIdata`, the listing of the first values of `minority` includes only the value `2`, i.e., the second level. Hence, we could conclude that, in the `classroom` data frame, the numeric value of `minority` for the first observations was equal to `1`, which is in agreement with the printout shown in Panel R2.9.

When converting variables `schoolid`, `childid`, and `classid` into factors, we use neither the `levels` nor `labels` argument. Thus, by default, the levels of the constructed factors are defined by taking the unique numeric values of each of the variables, representing the values as character strings, and sorting the strings in an

**R2.11** *SII Project*: Saving the `SIIdata` data in an external file

```
> rdaDir <- file.path("C:", "temp")          # Dir path
> fp <- file.path(rdaDir, "SIIdata.Rdata")   # External file path
> save(SIIdata, file = fp)                   # Save data
> file.exists(fp)
  [1] TRUE
> (load(fp))                                 # Load data
  [1] "SIIdata"
```

increasing order according to the numeric values. On the other hand, the labels are defined, by default, as equal to the (character) levels of the factor. Hence, for instance, for the variable `schoolid`, the ordered (character) levels are "1", "2", ..., "107", with the same sequence used to create the corresponding set of labels (see Panel R2.10).

For illustration purposes, in Panel R2.11, we present a syntax that allows saving data in an external file for later use and then loading them back from that file. It is recommended to perform these steps at the end of an R session. In our book, we do not have to do it, because the data are already saved in the **nlmeU** package.

### *2.4.3 Data Hierarchy*

In practice, we often want to verify whether identifying variables, contained in a dataset, were properly coded, so that they correctly reflect the intended data hierarchy. In this section, we present the R tools that can be used for this purpose. As an example, we use the data stored in the data frame `SIIdata`. In this way, we provide additional information about the structure of the data frame.

Toward this end, we create, in Panel R2.12, an auxiliary data frame `dtId`, which contains the school, class, and pupil identifiers from `SIIdata`. We then apply the function `duplicated()` to the auxiliary data frame. The function looks for duplicated rows in the data frame and returns a logical vector that indicates which rows are duplicates. By applying the function `any()` to the resulting logical vector, we check if any of the elements of the vector contains the logical value of `TRUE`. It turns out that there are no such elements, i.e., that there are no duplicated combinations of the three identifiers in the `SIIdata` data frame. This indicates that individual pupils in the data are uniquely identified by these variables, as intended.

Next, we apply the function `gsummary()` from the package **nlme**. The function provides a summary of variables, contained in a data frame, by groups of rows. In particular, the function can be used to determine whether there are variables that are invariant within the groups. Note that the groups are defined by the factors specified on the right-hand side of the formula specified in the argument `form` (more information on the use of formulae in R will be provided in Chap. 5).

**R2.12** *SII Project*: Investigation of the data hierarchy in the data frame `SIIdata`

```
> data(SIIdata, package = "nlmeU")          # Load data
> dtId <- subset(SIIdata, select = c(schoolid, classid, childid))
> names(dtId)                               # id names
  [1] "schoolid" "classid"  "childid"
> any(duplicated(dtId))                     # Any duplicate ids?
  [1] FALSE
> require(nlme)
> names(gsummary(dtId, form = ~childid, inv = TRUE))
  [1] "schoolid" "classid"  "childid"
> names(gsummary(dtId, form = ~classid, inv = TRUE))
  [1] "schoolid" "classid"
> names(gsummary(dtId, form = ~schoolid, inv = TRUE))
  [1] "schoolid"
```

We first apply the function `gsummary()` to the data frame `dtId`, with groups defined by `childid`. We also use the argument `inv = TRUE`. This means that only those variables, which are invariant within each group, are to be summarized. By applying the function `names()` to the data frame returned by the function `gsummary()`, we learn that, within the rows sharing the same value of `childid`, the values of variables `schoolid` and `classid` are also constant. In other words, variable `childid` is *inner* to both `classid` and `schoolid`. In particular, this implies that no pupil is present in more than one class or school. Hence, we can say that pupils are *nested* within both schools and classes. If some pupils were enrolled in, e.g., more than one class, then we could say that pupils were *crossed* with classes. In such case, the values of the `classid` identifier would not be constant within the groups defined by the levels of the `childid` variable.

Application of the function `gsummary()` to the data frame `dtId` with groups defined by `classid` allows us to conclude that, within the rows sharing the same value of `classid`, the values of `schoolid` are also constant. This confirms that, in the data, classes are coded as nested within schools. Equivalently, we can say that the variable `classid` is inner to `schoolid`.

Finally, there are no invariant identifiers within the groups of rows defined by the same value of `schoolid`, apart from `schoolid` itself.

In a similar fashion, in Panel R2.13, we use the function `gsummary()` to investigate, which covariates are defined at the school, class, or pupil level. In Panel R2.13a, we apply the function to the data frame `SIIdata`, with groups defined by `schoolid`. The displayed result of the function `names()` implies that the values of the variable `housepov` are constant (invariant) within the groups of rows with the same value of `schoolid`. Hence, `housepov` is the only school-level covariate, in accordance with the information given in Sect. 2.4.1.

In Panel R2.13b, we apply the function `gsummary()` with groups defined by `classid`. We store the names of invariant variables in the character vector `nms2a`.

**R2.13** *SII Project*: Identification of school-, class-, and pupil-level variables in the data frame `SIIdata`

**(a)** *School-level variables*

```
> (nms1 <-
+   names(gsummary(SIIdata,
+                  form = ~schoolid, # schoolid-specific
+                  inv = TRUE)))
  [1] "housepov" "schoolid"
```

**(b)** *Class-level variables*

```
> nms2a <-
+   names(gsummary(SIIdata,
+                  form = ~classid,  # classid- and schoolid-specific
+                  inv = TRUE))
> idx1  <- match(nms1, nms2a)
> (nms2 <- nms2a[-idx1])             # classid-specific
  [1] "yearstea" "mathknow" "mathprep" "classid"
```

**(c)** *Pupil-level variables*

```
> nms3a <-
+   names(gsummary(SIIdata,
+                  form = ~childid,  # All
+                  inv = TRUE))
> idx12 <- match(c(nms1, nms2), nms3a)
> nms3a[-idx12]                      # childid-specific
  [1] "sex"      "minority" "mathkind" "mathgain" "ses"
  [6] "childid"
```

We identify the names of variables, which are constant both at the school and class level, by matching the elements of vectors `nms1` and `nms2a`. After removing the matching elements from the vector `nms2a`, we store the result in the vector `nms2`. The latter vector contains the names of variables, which are invariant at the class level, namely, `yearstea`, `mathknow`, and `mathprep`.

Finally, in Panel R2.13c, we look for pupil-level variables. The syntax is similar to the one used in R2.13b. As a result, we identify variables `sex`, `minority`, `mathkind`, `mathgain`, and `ses`, again consistent with variables listed in Sect. 2.4.1.

Considerations, presented in Panel R2.13, aimed at identifying grouping factor(s) for which a given covariate is invariant. The resulting conclusions have important implications for computations of the number of denominator degrees of freedom for the conditional $F$-tests applied to fixed effects in LMMs (see Sect. 14.7 and Panel R18.5 in Sect. 18.2.2).

#### 2.4.3.1 Explicit and Implicit Nesting

The `SIIdata` data frame is an example of data having *nested* structure. This structure, with classes being nested within schools, can be represented in the data in two different ways, depending on how the two relevant factors, namely, `schoolid` and `classid`, are coded.

First, we consider the case when the levels of `classid` are explicitly coded as *nested* within the levels of the `schoolid` grouping factor. This way of coding is referred to as *explicit nesting* and is consistent with that used in `SIIdata`, as shown in Panel R2.12. More specifically, the nesting was accomplished by using *different* levels of the `classid` factor for different levels of the `schoolid` factor. Consequently, the intended nested structure of data is explicitly reflected by the levels of the factors. This is the preferred and natural approach.

The nested structure could also be represented by using *crossed* grouping factors. Taking the `SIIdata` data as an example, we might consider the case when, by mistake or for any other reason, two different classrooms from two different schools would have *the same* code. In such a situation, and without any additional information about the study design, the factors would be incorrectly interpreted as (partially) crossed. To specify the intended nested structure, we would need to cross `schoolid` and `classid` factors using, e.g., the command `factor(schoolid:classid)`. The so-obtained grouping factor, together with `schoolid`, would specify the desired nested structure. Such an approach to data coding is referred to as *implicit nesting*.

Although the first way of representing the nested structure is simpler and more natural, it requires caution when coding the levels of grouping factors. The second approach is more inclusive, in the sense that it can be used both for crossed *and* nested factors.

We raise the issue of the different representations of nested data because it has important implications for a specification of an LMM. We will revisit this issue in Chap. 15.

## 2.5 The Flemish Community Attainment-Targets Study

The FCAT data results from an educational study, in which elementary-school graduates were evaluated with respect to reading comprehension in Dutch. The evaluation was based on a set of attainment targets, which were issued by the Flemish Community in Belgium. These attainment targets can be characterized by the text type and by the level of processing. We use data which consist of the responses of a group of 539 pupils from 15 schools who answered 57 items assumed to measure nine attainment targets. In Table 2.1, the nine attainment targets are described by the type of text and by the level of processing. In addition, we indicate the number of items that were used to measure each one of the targets.

**Table 2.1** *FCAT Study*: FCAT Study: Attainment targets for reading comprehension in Dutch. Based on Janssen et al. (2000). Reproduced with permission from the copyright owner

| Target | Text type | Level of processing | No. of items |
|---|---|---|---|
| 1 | Instructions | Retrieving | 4 |
| 2 | Articles in magazine | Retrieving | 6 |
| 3 | Study material | Structuring | 8 |
| 4 | Tasks in textbook | Structuring | 5 |
| 5 | Comics | Structuring | 9 |
| 6 | Stories, novels | Structuring | 6 |
| 7 | Poems | Structuring | 8 |
| 8 | Newspapers for children, textbooks, encyclopedias | Evaluating | 6 |
| 9 | Advertising material | Evaluating | 5 |

These data were analyzed previously by, e.g., Janssen et al. (2000) and Tibaldi et al. (2007). In our analyses we will use two types of outcomes. First, we will consider total target scores, i.e., the sum of all positive answers for a target. Second, we will consider average target scores, i.e., the sum of all positive answers for a category divided by the number of items within the target. In both cases, we will treat the outcome as a continuous variable.

### *2.5.1 Raw Data*

We assume that the raw data for the FCAT study are stored in an external file named `crossreg.data.csv`.

In Panel R2.14, we present the syntax for loading and inspecting the data. As seen from the output presented in the panel, the file `crossreg.data.csv` contains 4,851 records and three variables. The variable `id` contains pupils' identifiers, `target` identifies the attainment targets (see Table 2.1), and `scorec` provides the total target score for a particular pupil. Note that the data are stored using the "long" format, with multiple records per pupil.

In Panel R2.15, we investigate the contents of the `crossreg.data` data frame in more detail. In particular, by applying the function `unique()` to each of the three variables contained in the data frame, we conclude that there are 539 unique values for `id`, nine unique values for `target`, and 10 unique values for `scorec`. Thus, the data frame includes scores for nine targets for each of 539 pupils. Note that $9 \times 539 = 4{,}851$, i.e., the total number of records (rows). Because the maximum number of items for a target is nine (see Table 2.1), the variable `scorec` contains integer values between 0 and 9.

**R2.14** *FCAT Study*: Loading raw data from the .csv file into the object `crossreg.data`. The object `dataDir` was created in Panel R2.1

```
> fp <- file.path(dataDir, "crossreg.data.csv")
> crossreg.data <- read.csv(fp, header = TRUE)
> dim(crossreg.data)                        # No. of rows and columns
  [1] 4851    3
> names(crossreg.data)                      # Variable names
  [1] "target" "id"     "scorec"
> head(crossreg.data)                       # First six records
    target id scorec
  1      1  1      4
  2      2  1      6
  3      3  1      4
  4      4  1      1
  5      5  1      7
  6      6  1      6
> str(crossreg.data)                        # Data structure
  'data.frame':   4851 obs. of  3 variables:
   $ target: int  1 2 3 4 5 6 7 8 9 1 ...
   $ id    : int  1 1 1 1 1 1 1 1 1 2 ...
   $ scorec: int  4 6 4 1 7 6 6 5 5 3 ...
```

**R2.15** *FCAT Study*: Inspection of the contents of the raw data. The data frame `cressreg.data` was created in Panel R2.14

```
> unique(crossreg.data$target)          # Unique values for target
  [1] 1 2 3 4 5 6 7 8 9
> (unique(crossreg.data$id))            # Unique values for id
    [1]   1   2   3   4   5   6   7   8   9  10  11  12  13 14 15
  ...     [snip]
  [526] 526 527 528 529 530 531 532 533 534 535 536 537 538 539
> unique(crossreg.data$scorec)          # Unique values for scorec
   [1] 4 6 1 7 5 3 2 8 0 9
> summary(crossreg.data$scorec)         # Summary statistics for scorec
     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0.0     3.0     4.0     3.9     5.0     9.0
```

### 2.5.2 *Data for Analysis*

In the analyses presented later in the book, we will be using the data frame `fcat`, which is constructed based on the data frame `crossreg.data`. In Panel R2.16, we present the syntax used to create the `fcat` data and to investigate data grouping structure. First, in Panel R2.16a, we replace the variables `id` and `target` by corresponding factors. For the factor `target`, the labels given in parentheses indicate the number of items for a particular target.

In Panel R2.16b, we cross-tabulate the factors `id` and `target` and store the resulting table in the object `tab1`. Given the large number of levels of the factor `id`, it is difficult to verify the values of the counts for all cells of the table. By applying the function `all()` to the result of the evaluation of expression `tab1>0`, we check that all counts of the table are nonzero. On the other hand, with the help of the `range()` function, we verify that all the counts are equal to 1. This indicates that, in the data frame `fcat`, the levels of the factor `target` are crossed with the levels of the factor `id`. Moreover, the data are balanced, in the sense that there is the same number of observations, namely, one observation for each combination of the levels of the two factors. Because all counts in the table are greater than zero, we can say that the factors are *fully crossed*.

## 2.6 Chapter Summary

In this chapter, we introduced four case studies, which will be used for illustration of LMs described in our book.

We started the presentation of each case study by describing study design and considering that raw data are stored in a .csv file. We chose this approach in an attempt to emulate a common situation of using external data files when analyzing data using R. In the next step, we prepared the data for analysis by creating the necessary variables and, in particular, factors. Including factors as part of data is a feature fairly unique to R. It affects how a given variable is treated by graphical and modeling functions. This approach is recommended, but not obligatory. In particular, creating factors can be deferred to a later time, when, e.g., *model formula* is specified. We will revisit this issue in Chap. 5.

The data frames, corresponding to the four case studies, are included in the package **nlmeU**. As with other packages, the list of datasets available in the package can be obtained by using the `data(package = "nlmeU")` command. For the reader's convenience, the datasets are summarized in Table 2.2. The table includes the information about the R-session panels, which present the syntax used to create the data frames, grouping factors, and number of rows and variables.

The four case studies introduced in this chapter are conducted by employing different study designs. All of them lead to grouped data defined by one or more nested or crossed grouping factors. The preferable way of storing this type of data

**R2.16** *FCAT Study*: Construction and inspection of the contents of the data frame `fcat`. The data frame `crossreg.data` was created in Panel R2.14

**(a)** *Construction of the data frame* `fcat`

```
> nItms <- c(4, 6, 8, 5, 9, 6, 8, 6, 5)        # See Table 2.1
> (lbls <- paste("T", 1:9, "(", nItms, ")", sep = ""))
 [1] "T1(4)" "T2(6)" "T3(8)" "T4(5)" "T5(9)" "T6(6)" "T7(8)"
 [8] "T8(6)" "T9(5)"
> fcat <-
+   within(crossreg.data,
+          {
+           id <- factor(id)
+           target <- factor(target, labels = lbls)
+          })
> str(fcat)
 'data.frame':   4851 obs. of  3 variables:
  $ target: Factor w/ 9 levels "T1(4)","T2(6)",..: 1 2 3 4 5 6 7 8 ...
  $ id    : Factor w/ 539 levels "1","2","3","4",..: 1 1 1 1 1 1 1 ...
  $ scorec: int  4 6 4 1 7 6 6 5 5 3 ...
```

**(b)** *Investigation of the data grouping structure*

```
> (tab1 <- xtabs(~ id + target, data = fcat))   # id by target table
     target
id    T1(4) T2(6) T3(8) T4(5) T5(9) T6(6) T7(8) T8(6) T9(5)
  1       1     1     1     1     1     1     1     1     1
  2       1     1     1     1     1     1     1     1     1
...     [snip]
  539     1     1     1     1     1     1     1     1     1
> all(tab1 > 0)                                 # All counts > 0?
 [1] TRUE
> range(tab1)                                   # Range of counts
 [1] 1 1
```

is to use the "long" format with multiple records per subject. Although this term is borrowed from the literature pertaining to longitudinal data, it is also used in the context of other grouped data. Below, we describe the key features of the data in each study.

In the ARMD trial, the `armd.wide` data frame stores data in the "wide" format. Data frames `armd` and `armd0` store data in the "long" format and reflect the hierarchical data structure defined by a single grouping factor, namely, `subject`. For this reason, and following the naming convention used in the **nlme** package, we will refer to the data structure in our book as data with a *single level of grouping*. Note that, more traditionally, these data are referred to as *two-level data* (West et al., 2007).

**Table 2.2** Data frames available in the **nlmeU** package

| Study | Data frame | R-panel | Grouping factors | Rows × vars |
|---|---|---|---|---|
| *ARMD Trial* | `armd.wide` | R2.2 | *None* | 240 × 10 |
| | `armd0` | R2.4 | `subject` | 1,107 ×8 |
| | `armd` | R2.5 | `subject` | 867 × 8 |
| *PRT Trial* | `prt.subjects` | R2.7a | *None* | 63 × 5 |
| | `prt.fiber` | R2.7b | `id` | 2,471 ×5 |
| | `prt` | R2.8 | `id` | 2,471 ×9 |
| *SII Project* | `SIIdata` | R2.10 | `classid` nested ... | 1,190 ×12 |
| | | | ... in `schoolid` | |
| *FCAT Study* | `fcat` | R2.16 | `id` crossed ... | 4,851 ×3 |
| | | | ... with `target` | |

The hierarchical structure of data contained in the data frame `SIIdata` is defined by two (nested) grouping factors, namely, `schoolid` and `classid`. Thus, in our book, this data structure will be referred to as data with *two levels of grouping*.

This naming convention works well for hierarchical data, i.e., for data with nested grouping factors. It is more problematic for structures with crossed factors. This is the case for the FCAT study, in which the data structure is defined by two crossed grouping factors, thus without a particular hierarchy.

As a result of data grouping, variables can be roughly divided into group- and measurement-specific categories. In the context of longitudinal data they are referred to as time-fixed and time-varying variables. The classification of the variables has important implications for the model specification.

To our knowledge, the *groupedData* class, defined in the **nlme** package, appears to be the only attempt to directly associate a hierarchical structure of the data with objects of the *data.frame* class. We do not describe this class in more detail, however, because it has some limitations. Also, its initial importance has diminished substantially over time. In fact, the data hierarchy is most often reflected indirectly by specifying the structure of the model fitted to the data. We will revisit this issue in Parts III and IV of our book.

When introducing the SII case study, we noted that the nested data structure can be specified by using two different approaches, namely, explicit and implicit nesting, depending on the coding of the levels of grouping factors. The choice of the approach is left to the researcher's discretion. The issue has important implications for the specification of LMMs, though, and it will be discussed in Chap. 15.

The different data structures of the cases studies presented in this chapter will allow us to present various aspects of LMMs in Part IV of the book. Additionally, the ARMD dataset will be used in the other parts to illustrate other classes of LMs and related R tools.

The main focus of this chapter was on the presentation of the data frames related to the case studies. In the presentation, we also introduced selected concepts related

to grouped data and R functions, which are useful for data transformation and inspection of the contents of datasets. By necessity, our introduction was very brief and fragmentary; a more in-depth discussion of those and other functions is beyond the scope of our book. The interested readers are referred to, e.g., the book by Dalgaard (2008) for a more thorough explanation of the subject.

http://www.springer.com/978-1-4614-3899-1

Linear Mixed-Effects Models Using R
A Step-by-Step Approach
Gałecki, A.; Burzykowski, T.
2013, XXXII, 542 p. 64 illus., Hardcover
ISBN: 978-1-4614-3899-1