Andrzej Gałecki
Tomasz Burzykowski

Linear Mixed Effects Models Using R

A Step-by-step Approach

January 31, 2012

Springer

Moim bliskim
Violi, Martuni, Samancie, Arturkowi, i Pawełkowi
Moim Rodzicom i Nauczycielom
Dekadentom
– A.T.G.

Moim najbliższym i przyjaciołom
– T.B.

In memory of Tom Ten Have

Preface

Linear mixed-effects model (LMMs) are powerful modeling tools that allow for the analysis of datasets with complex, hierarchical structures. Intensive research in the past decade has led to a better understanding of their properties. The growing body of literature, including recent monographs, has considerably increased their popularity among applied researchers. There are several statistical software packages containing routines for LMMs. These include, for instance, SAS, SPSS, STATA, S+, and R. The major advantage of R is that it is a freely available, dynamically developing, open-source environment for statistical computing and graphics.

The goal of our book is to provide a description of tools available for fitting LMMs in R. The description is accompanied by a presentation of the most important theoretical concepts of LMMs. Additionally, examples of applications from various areas illustrate the main features of both theory and software. The presented material should allow readers to obtain a basic understanding of LMMs and to apply them in practice. In particular, theoretical concepts and their practical implementation in R are presented in the context of simpler, more familiar classes of models like, for example the classical linear regression model. Based on these concepts, more advanced classes of models, such as models with correlated residual errors, are introduced. In this way, we incrementally set the stage for LMMs, so that the exposition of the theory and R tools for these models becomes simpler and clearer. This structure naturally corresponds to the object-oriented programming concept, according to which R functions/methods for simpler models are also applicable to the more complex ones.

We assume that readers are familiar with intermediate linear algebra, calculus, and the basic theory of statistical inference and linear modeling. Thus, the intended audience for this book are graduate students of statistics and applied researchers in other fields.

Our exposition of the theory of various classes of models presented in the book focuses on concepts, which are implemented in the functions available in R. Readers interested in a more detailed description of the theory are referred to appropriate theoretical monograph books, which we indicate in the text.

There are a large number of R packages, that can be used to fit LMMs. Rather then attempting to briefly describe all of these packages, we focus mainly on two of them namely, **nlme** and **lme4**. In this way, we can provide a more detailed account of the tools offered by the two packages, which include a wide variety of functions for model fitting, diagnostics, inference, etc. The package **nlme** includes functions which allow for the fitting of a wide range of linear models and LMMs. Moreover, it has been available for many years and its code has been stable for some time now. Thus, it is a well-established R tool. On the other hand, the more recent package **lme4** offers an efficient computational implementation and an enhanced syntax, though at the cost of a more restricted choice of LMMs.

All classes of linear models presented in the book are illustrated using data from a particular dataset. In this way, the differences between the various classes of models, as well as differences in the R software, can be clearly delineated. LMMs, which are the main focus of the book, are also illustrated using three additional datasets, which extend the presentation of various aspects of the models and R functions. We have decided to include the direct output of R commands in the text. In this way, readers who would like to repeat the analyses conducted in the book, can directly check their own output. However, in order to avoid the risk of incompatibility with updated versions of the software, the results of the analyses have also been summarized in the form of edited tables.

To further support those readers who are interested in actively using the material presented in the book, we have developed the package **nlmeU**. It contains all the datasets and R code used in the book. The package is downloadable at `http://www-personal.umich.edu/~agalecki/`.

We hope that our book, which aims to provide a state-of-the-art description of the details of implementing of LMMs in R, will support a widespread use of the models by applied researchers in a variety of fields including biostatistics, public health, psychometrics, educational measurement, and sociology.

When working on the text, we were receiving considerable assistance and valuable comments from many people. We would like to acknowledge Geert Molenberghs (Hasselt University and the Catholic University of Leuven), Geert Verbeke (Catholic University of Leuven), José Pinheiro (Novartis AG), Paul Murrell (Auckland University), Przemysław Biecek (Warsaw University), Fabian Scheipl (Ludwig Maximilian University of Munich), Jeffrey Halter (University of Michigan), Shu Chen (University of Michigan), Marta Gałecka (Weill Cor-

nell Medical College) and members of the R-sig-ME discussion group led by Douglas Bates (University of Wisconsin-Madison) and Ben Bolker (McMaster University), for their comments and discussions at various stages during the preparation of the book. We are grateful to John Kimmel for encouraging us to consider writing the book and to Marc Strauss from Springer for their editorial assistance and patience. We also gratefully acknowledge financial support from the Claude Pepper Center grants AG08808 and AG024824 from the National Institute of Aging and from the IAP research Network P6/03 of the Belgian Government (Belgian Science Policy).

Ann Arbor, Michigan, USA — *Andrzej T. Gałecki*
Diepenbeek, Belgium, and Warszawa, Poland — *Tomasz Burzykowski*
January, 2012

Contents

List of Tables

I Introduction

II Linear Models for Independent Observations

List of Figures

I Introduction

II Linear Models for Independent Observations

III Linear Fixed-effects Models for Correlated Data

IV Linear Mixed-effects Models

List of R sessions

Part I

Introduction

1

Introduction

1.1 The aim of the book

Linear mixed-effects models (LMMs) are an important class of statistical models that can be used to analyze correlated data. Such data include *clustered observations*, *repeated measurements*, *longitudinal measurements*, *multivariate observations*, etc.

The aim of our book is to help readers in fitting LMMs using R software. R (`www.r-project.org`) is a language and an environment aimed at facilitating implementation of statistical methodology and graphics. It is an open-source software, which can be freely downloaded and used under the GNU general Public License. In particular, users can define and share their own functions, which implement various methods and extend the functionality of R. This feature makes R a very useful platform for propagating the knowledge and use of statistical methods.

We believe that, by describing selected tools available in R for fitting LMMs, we can promote the broader application of the models. To help readers less familiar with this class of linear models (LMs), we include in our book a description of the most important theoretical concepts and features of LMMs. Moreover, we present examples of applications of the models to real-life datasets from various areas to illustrate the main features of both theory and software.

1.2 Implementation of linear mixed-effects models in R

There are many packages in R, which contain functions that allow fitting various forms of LMMs. The list includes, but is not limited to, packages **amer**,

arm, **gamm**, **gamm4**, **GLMMarp**, **glmmAK**, **glmmBUGS**, **heavy**, **HGLMMM**, **lme4**, **lmec**, **lmm**, **longRPart**, **MASS**, **MCMCglmm**, **nlme**, **PSM**, and **pedigreemm**. On the one hand, it would seem that the list is rich enough to allow for a widespread use of LMMs. On the other hand, the number of available packages leads to difficulty evaluating their relative merits choosing of the most suitable package.

It is virtually impossible to describe the contents of all of the packages mentioned above. To facilitate and promote the use of LMMs in practice, it might be more useful to provide details for a few of them, so that they could be used as a starting point. Therefore, we decided to focus on the packages **nlme** and **lme4**, for several reasons. First, they contain the functions `lme()` and `lmer()`, respectively, which are specifically designed for fitting a broad range of LMMs. Second, they include many tools useful for applications such as model diagnostics. Finally, many other packages, which add new LMM classes or functionalities, depend on and are built around **nlme** and/or **lme4**. Examples include, but are not limited to, packages **amer**, **gamm**, **gamm4**, or **RLRsim**.

In fact, we focus more on the package **nlme** than on **lme4**. The main reason is that the former has already been around for some time. Thus, its code is stable. On the other hand, the package **lme4** is still under development and its code may be subject to changes in the near future. Thus, there is a risk that the features we might want to describe in the context of the current version of the package, may be modified in the later versions.

An important feature, that distinguishes R from many other existing statistical software packages implementing LMMs, is that it incorporates several concepts of an *object-oriented* (O-O) programming, such as *classes* of *objects* and *methods* operating on those classes. There are two O-O systems that have been implemented in R, namely, S3 and S4. They incorporate the O-O concepts to different degree, with S3 being a less formal, and S4 being a more stringent implementation. In both systems, the O-O concepts are implemented by defining special type of functions called *generic* functions. When such a function is applied to an object, it dispatches an appropriate *method* based on object's class. The system S3 has been used in the package **nlme**, while S4 has been used in the package **lme4**.

The O-O programming approach is very attractive in the context of statistical modeling because models can often be broken down into separable (autonomous) components such as data, mean structure, variance function, etc. Moreover, components defined for one type of model can also be used as building blocks for a different type of model.

1.3 The structure of the book

As it was mentioned in the previous section, an inherent feature of the O-O programming approach is that concepts and methods used for simpler objects or models are applicable to more complex objects or models. For this reason, in our book we chose an incremental build-up of the knowledge about the implementation of LMMs in the functions from packages **nlme** and **lme4**. In particular, in the first step, we decided to introduce theoretical concepts and their practical implementation in the R code in the context of simpler classes of LMs, like the classical linear regression model. The concepts are then carried over to more advanced classes of models, including LMMs. This step-by-step approach offers a couple of advantages. First, we believe that it makes the exposition of the theory and R tools for LMMs simpler and clearer. In particular, the presentation of the key concepts in the context of a simpler model makes them easier to explain and understand. Second, the step-by-step approach is helpful in the use of other R packages, which rely on classes of objects defined in the **nlme** and/or **lme4** packages.

As a result of this conceptual approach, we divided our book into four parts. Part I contains the introduction to the datasets used in the book. Parts II, III, and IV focus on different classes of LMs of increasing complexity. The structure of the three parts is, to a large extent, similar. First, a brief review of the main concepts and theory of a particular class of models is presented. Special attention is paid to the presentation of the link between similar concepts used for different classes. Then, the details of how to implement the particular class of models in the packages **nlme** and/or **lme4** are described. The idea is to present the key concepts in the context of simpler models, in order to enhance the understanding of them and facilitate their use for the more complex models. Finally, in each part, the particular class of LMs and the corresponding R tools are illustrated by analyzing real-life datasets.

In a bit more detail, the content of the four parts is as follows:

Chapter 2 of Part I contains a description of four case studies, which are used to illustrate various classes of LMs and of the corresponding R tools. On the other hand, Chap. 3 contains results of exploratory analyses of the datasets. The results are used in later chapters to support model-based analyses. Note that one of the case studies, the Age-related Macular Degeneration (ARMD) clinical trial, is used repeatedly for the illustration of all classes of LMs. We believe that in this way the differences between the models concerning, e.g., the underlying assumptions, may become easier to appreciate.

Part II focuses on LMs for independent observations. In Chap. 4, we recall the main concepts of the theory of the classical LMs with homoscedastic residual errors. Then, in Chap. 5, we present the tools available in R to fit such models.

This allows us to present the fundamental concepts used in R for statistical model-building, like *model formula*, *model frame*, etc. The concepts are briefly illustrated in Chap. 6 using the data from the ARMD trial.

Subsequently, we turn our attention to models with heteroscedastic residual errors. In Chap. 7, we review the basic elements of the theory. Chapter 8 presents the function `gls()` from the package **nlme**, which can be used to fit the models. In particular, the important concept of the *variance function* is introduced in the chapter. The use of the function `gls()` is illustrated using data from the ARMD trial in Chap. 9.

In Part III, we consider general LMs, i.e., LMs for correlated observations. In Chap. 10, we recall the basic elements of the theory of the models. In particular, we explain how the concepts used in the theory of the LMs with heteroscedastic residual errors for independent observations, presented in Chap. 7, are extended to the case of models for correlated observations. In Chap. 11, we describe additional features of the function `gls()`, which allow the use of it for fitting general LMs. In particular, we introduce the key concept of the *correlation structure*. The use of the function `gls()` is illustrated in Chap. 12 using the data from the ARMD trial.

Finally, Part IV is devoted to LMMs. Chapter 13 reviews the fundamental elements of the theory of LMMs. In the presentation, we demonstrate the links between the concepts used in the theory of LMMs with those developed in the theory of general LMs (Chap. 10). We believe that by pointing to the links, the exposition of the fundamentals of the LMM theory becomes more transparent and easier to follow.

In Chap. 14, we describe the features of the function `lme()` from the package **nlme**. This function is the primary tool in the package used to fit LMMs. In particular, we describe in detail the representation of positive-definite matrices, which are instrumental in the implementation of the routines that allow fitting LMMs. Note that the concepts of the variance function and correlation structure, introduced in Chaps. 8 and 11, respectively, are also important for the understanding of the use of the function `lme()`.

In Chap. 15, we present the capabilities of the function `lmer()` from the package **lme4**. In many aspects, the function is used similarly to `lme()`, but there are important differences, which we discuss. The basic capabilities of both of the functions are illustrated by application of LMMs to the analysis of the ARMD trial data in Chap. 16. More details on the use of the function `lme()` are provided in Chaps. 17, 18, and 19, in which we analyze using LMMs, the data from the Progressive Resistance Training (PRT) study, the Study of Instructional Improvement (SII), and the Flemish Community Attainment-targets (FCAT) study, respectively. Finally, in Chap. 20, we present somewhat more advanced material on the additional R tools for LMMs, including

the methods for power calculations, influence diagnostics, and a new class of positive-definite matrices. The latter can be used to construct LMMs with random effects having a variance-covariance matrix defined as a Kronecker product of two or more matrices. Note that the newly-defined class is used in the analysis presented in Chap. 17.

Table 1.1 summarizes the successive classes of LMs, described in our book, together with the concepts introduced in the context of the particular class. The classes are identified by the assumptions made about the random part of the model.

==== Table 1.1 about here =====

Table 1.1: Classes of linear models with the corresponding components (building blocks) presented in the book. The R classes refer to the package **nlme**.

Linear model			Model component	
Class (residual errors)	Theory	Syntax	Name	R class
Homoscedastic, indep.	Ch. 4	Ch. 5	Data	*data.frame*
			Mean structure	*formula*
Heteroscedastic, indep.	Ch. 7	Ch. 8	Variance structure	*varFunc*
Correlated	Ch. 10	Ch. 11	Correlation structure	*corStruct*
Mixed-effects (LMM)	Ch. 13	CH. 14	Random-effects structure	*reStruct*

Our book contains 67 figures, 47 tables, and 187 panels with R code.

Finally, we would like to outline the scope of the contents of the book:

- The book is aimed primarily at providing explanations and help with respect to the tools available in R for fitting LMMs. Thus, we do not provide a comprehensive account of the methodology of LMMs. Instead, we limit ourselves to the main concepts and techniques, which have been implemented in the functions `lme()` and `lmer()` from the packages **nlme** and **lme4**, respectively, and which are important to the understanding of the use of the functions. A detailed exposition of the methodology of LMMs can be found in books by, for example, Searle et al. (1992), Davidian & Giltinan (1995), Vonesh & Chinchilli (1997), Pinheiro & Bates (2000), Verbeke & Molenberghs (2000), Demidenko (2004), Fitzmaurice et al. (2004), or West et al. (2007).

- In our exposition of methodology, we focus on the likelihood-based estimation methods, as they are primarily used in `lme()` and `lmer()`. Thus, we do not discuss, e.g., Bayesian approaches to the estimation of LMMs.

- We try to describe the use of various functions, which are available in the packages **nlme** and **lme4**, in sufficient detail. In our presentation, we focus on the main, or most often used, arguments of the functions. For a detailed description of all of the arguments, we refer the readers to R's help system.

- It is worth keeping in mind that, in many instances, the same task can be performed in R in several different ways. To some extent, the choice between the different methods is a matter of individual preference. In our description of the R code, we present methods, which we find to be the most useful. If alternative solutions are possible, we may mention them, but we are not aiming to be exhaustive.

- The analyses of the case studies aim principally to illustrate various linear models and the possibility of fitting the models in R. While we try to conduct as meaningful analyses as possible, they are not necessarily performed in the most optimal way with respect to, e.g., the model-building strategy. Thus, their results should not be treated as our contribution to the subject-matter discussion related to the examples. However, whenever possible or useful, we make an attempt to provide quantitative and/or qualitative interpretation of the results. We also try to formulate practical recommendations or guidance regarding model-building strategies, model diagnostics, etc. As mentioned earlier, however, the book is not meant to serve as a complete monograph on statistical modeling. Thus, we limit ourselves to providing recommendations or guidance for the topics which appear to be of interest in the context of the analyzed case studies.

1.4 Technical notes

The book is aimed at helping readers in fitting LMMs in R. We do assume that the reader has a basic knowledge of R. An introduction to R can be found in the book by Dalgaard (2008). A more advanced exposition is presented by Venables & Ripley (2010).

To allow readers to apply the R code presented in the book, we created the R package **nlmeU**. The package contains all the datasets and the code we used in the text. It also includes additional R functions, which we have developed.

We tried to use short lines of the R code to keep matters simple, transparent, and easy to generalize. To facilitate locating the code, we placed it in *panels*. The panels are numbered consecutively in each chapter and referred to, for

example as, R2.3, where "2" gives the number of the chapter and "3" is the consecutive number of the panel within the chapter. Each panel was given a caption explaining the contents. In some cases, the contents of a panel were logically split into different subpanels. The subpanels are then marked by consecutive letters and referred to by adding the appropriate letter to panel's number, for example, R2.3a or R2.3b. Tables and figures are numbered in a similar fashion.

Only in rare instances a few lines of R code were introduced directly into the text. In all these cases (as in the examples given later in this section), the code was written using the `true type` font and placed in separate lines marked with ">", mimicking R's command-window style.

To limit the volume of the output presented in the panels, in some cases we skipped a part of it. These interventions are indicated by the ... [*snip*] string.

The R functions are referred to in the text as `function()`, e.g., `lme()`. Functions' arguments and objects are marked using the same font, e.g., `argument` and `object`. For the R classes, we use italic, e.g., *lme* class.

For the proper execution of the R code used in the book, the following packages are required: **lattice**, **lme4**, **nlme**, **Matrix**, **plyr**, **reshape**, **RLRsim**, **splines**, **WWGbook**. Additionally, **nlmeU** is needed. Packages **lattice**, **nlme**, **Matrix**, and **splines** come with basic distribution of R and do not need to be installed. The remaining packages can be installed using the following code:

```
> pckgs  <-
+  c("lme4", "nlmeU", "plyr", "reshape", "RLRsim", "WWGbook",
+    "ellipse")
> install.packages(pckgs)
```

There are additional utility functions, namely, `Sweave()` (Leisch, 2002) and `xtable()` in **Sweave** and **xtable** (Dahl, 2009) packages, respectively, which are not needed to execute the code presented in the book, but were extensively used by us when preparing this manuscript.

It is worth noting that there are functions, which bear the same name in the packages **nlme** and **lme4**, but which have different definitions. To avoid unintentional masking of the functions, the packages should not be loaded simultaneously. Instead, it is recommended to switch between the packages. For example, when using **nlme** in a hypothetical R session, we load the package by using the `library()` or `require()` functions and execute statements as needed. Then, before switching to **lme4**, it is mandatory to detach the **nlme** package by using the `detach()` function. We also note that the `conflicts()`

function, included for illustration below, is very useful to identify names' conflicts:

```
> library(nlme)                 # Load package
> conflicts(detail = TRUE)      # Identifies names' conflicts
 ... statements omitted
> detach(package:nlme)          # Detach package
```

A similar approach should be applied when using the package **lme4**:

```
> library(lme4)
 ... statements omitted
> detach("package:lme4")
> detach("package:Matrix")      # Recommended
```

Note that detaching **Matrix** is less critical, but recommended.

In the examples above, we refer to the packages **nlme** and **lme4**. However, to avoid unintentional masking of objects, the same strategy may also be necessary for other packages, which may cause function names' conflicts.

When creating figures we used `"CMRoman"` and `"CMSans"` Computer Modern font families available in **cmrutils** package. These fonts are based on CM-Super font LaTeXpackage (Volovich & Lemberg, 1999) and CMSYASE fonts (Murrell & Ripley, 2006). The full syntax needed to create figures presented in the book is often extensive. In many cases, we decided to present a shortened version of the code. A full version is available in the **nlmeU** package.

Finally, the R scripts in our book were executed by using `R version 2.14.1 (2012-12-22)` under the Windows-XP operating system. We used the following global options:

```
> options(width = 65, digits = 5, show.signif.stars = FALSE)
```

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-α and placebo in patients with ARMD. Visual acuity of patients participating in the trial was measured at baseline and at four post-randomization time points. 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 a 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 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 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, for example, 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 (ARMD) Trial

The Age-related Macular Degeneration data arise from a randomized multi-center clinical trial comparing an experimental treatment (interferon-α) 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-α.

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 assume that our goal is to verify the content of the data.

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.

==== RSession R2.1 about here ====

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 `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

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
```

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.

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 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.

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.

==== RSession R2.2 around here ====

Note that the data are loaded into `R` using the `data()` function, without the need for loading 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 `armd240.data`. In contrast to `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 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

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"
```

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, repectively, of `treat`.

==== RSession R2.3 around here ====

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 the length equal to the number of rows of the matrix created by column-wise contactenation 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 non-missing value in the corresponding column of the matrix, while the latter indicates a missing value. Thus, application 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

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
```

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 `nlmeU:::missPat()` syntax, which allowed us to invoke the `missPat()` function without loading the **nlmeU** package.

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).

==== RSession R2.4 about here ====

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 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 ...
```

Panel R2.4 presents the contents and structure of the data frame `armd0`. The data frame includes eight variables and 1107 records. The contents of variables `subject`, `treat.f`, and `miss.pat`, is 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 proof useful when creating `armd` data 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 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., 1107, 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.

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.

=== Place Panel R2.5 about here =====

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, i.e., the records, for which `time>0`, from the object `armd0`. By removing the baseline measurements, we reduce the number of records from 1107 (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

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
```

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 (PRT) Study

The Progressive Resistance Training (PRT) data originate from a randomized trial aimed at devising evidence-based methods for improving and measuring the mobility and muscle power of elderly men and women in the 70+ age category. 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 data set 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 particular characteristic of muscle fibers for each subject correspond to a 2×2 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 (Chapt. 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.

=== Place Panel R2.6 about here ==========

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

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
```

`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` 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 2471 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` data set that will be used for analysis.

=== Place Panel R2.7 about here ==========

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. To this aim, 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

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 1 ...
   $ fiber.f: Factor w/ 2 levels "Type 1","Type 2": 1 1 2 1 2 1 1 1 ...
```

`train.pre.pos`, it includes the factors `fiber.f` and `occ.f`. Also, a subject's identifier `id` is stored as a factor.

=== Place Panel R2.8 about here =====

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
```

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 2471 records and nine variables. The contents of the first six rows of the data frame `prt` is displayed with the help of the `head()` function.

2.4 The Study of Instructional Improvement (SII) Project

The Study of Instructional Improvement (SII) was carried out to assess the math achievement-scores of first- and third-grade pupils in randomly selected classrooms from a national U.S. sample of elementary schools (Hill et al., 2005). The data set 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 1190 records and 12 variables.

=== Place Panel R2.9 about here =====

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:

- 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;

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 ...
```

- `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.

=== Place Panel R2.10 about here =====

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 ...
```

Essentially, the data frame `SIIdata` contains all the variables from `classroom`. However, the 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`.

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 in 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 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).

=== Place Panel R2.11 about here =====

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"
```

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 data set, 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.

=== Place Panel R2.12 about here==========

To this aim, 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

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"
```

in the argument `form` (more information on the use of formulae in R will be provided in Chap. 5).

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 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.

== Place Panel R2.13 about here==========

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"
```

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`. 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 to identify grouping factor(s) for which given covariate is invariant, i.e. is inner to, 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 Sects. 14.7 and Panel R18.5 in Sect. 18.2.2).

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 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 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, for example, 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 re-visit this issue in Chap. 15.

2.5 The Flemish Community Attainment-targets (FCAT) Study

The Flemish Community Attainment-targets (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 about here =====

Table 2.1: *FCAT Study*: Attainment targets for reading comprehension in Dutch. Based on Janssen et al. (2000).

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 before 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 4851 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.

=== Place Panel R2.14 about here ==========

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 ...
```

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 fame includes scores for nine targets for each of 539 pupils. Note that $9 \times 539 = 4851$, 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.

=== Place Panel R2.15 about here ==========

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 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.

=== Place Panel R2.16 about here ==========

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

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
```

`tab1>0`, we check that all counts of the table are non-zero. 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 exactly 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.

==== Table 2.2 about here =====

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`	1107 × 8
	`armd`	R2.5	`subject`	867 × 8
PRT Trial	`prt.subjects`	R2.7a	*none*	63 × 5
	`prt.fiber`	R2.7b	`id`	2471 × 5
	`prt`	R2.8	`id`	2471 × 9
SII Project	`SIIdata`	R2.10	`classid` nested ...	1190 × 12
			... in `schoolid`	
FCAT Study	`fcat`	R2.16	`id` crossed ...	4851 × 3
			... with `target`	

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. Preferable way of storing this type of data is to use "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 ARMD Trial the `armd.wide` data frame stores data in the "wide" format. Data frames `armd` and `armd0` store data in a "long" format and reflect a 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).

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 of 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. 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 on 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 *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, 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 data set 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 data sets. 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.

3

Data Exploration

3.1 Introduction

In this chapter, we present the results of exploratory analyses of the case studies introduced in Chap. 2. The results will serve as a basis for building LMs for the data in the next parts of the book.

While exploring the case-study data, we also illustrate the use of selected functions and graphical tools, which are commonly used to perform these tasks. Note, however, that, unlike in the other parts of the book, we are not discussing the functions and tools in much detail. The readers interested in the functionalities are referred to the monograph by Venables & Ripley (2010).

3.2 ARMD Trial: Visual acuity

In the ARMD data, we are mainly interested in the effect of treatment on the visual acuity measurements. Thus, in Fig. 3.1, we first take a look at the measurements by plotting them against time for several selected patients from both treatment groups. More specifically, we selected every 10-th patient from each group.

==== Fig.3.1 about here =====
(EPS: Figs/Figarmdprofiles See: Figarmdprofiles)

Based on the plots shown in Fig. 3.1, several observations can be made:

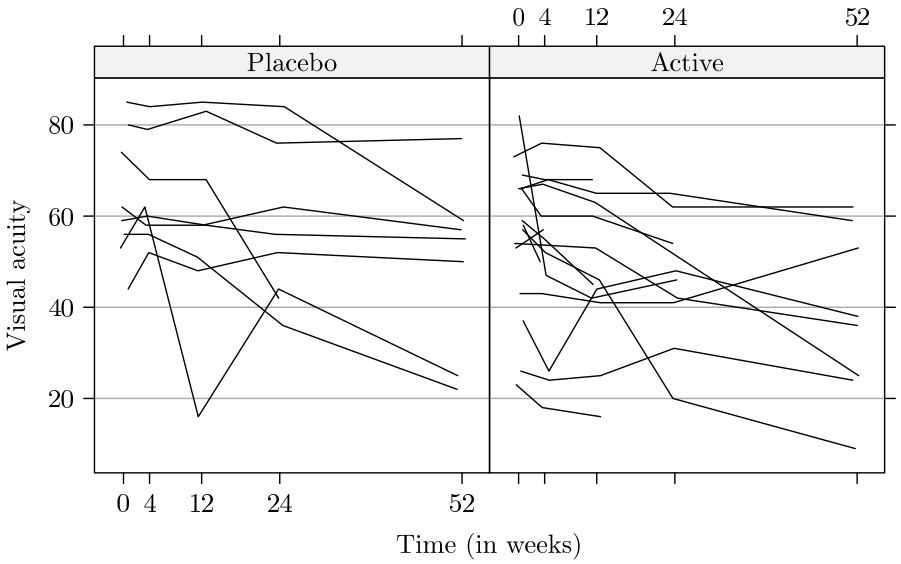


Fig. 3.1: *ARMD Trial*: Visual-acuity profiles for selected patients ("spaghetti plot").

- In general, visual acuity tends to decrease in time. This is in agreement with the remark made in Sect. 2.2 that patients with ARMD progressively loose vision.
- For some patients, a linear decrease of visual acuity over time can be observed, but there are also patients for whom individual profiles strongly deviate from a linear trend.
- Visual acuity measurements adjacent in time are fairly well correlated, with the correlation decreasing with an increasing distance in time.
- Visual acuity at baseline seems to, at least partially, determine the overall level of the post-randomization measurements.
- There are patients for whom several measurements are missing.

These observations will be taken into account when constructing models for the data.

=== Place Panel R3.1 about here =====

The syntax used to create Fig. 3.1 is shown in Panel R3.1. First, we load data to be used for exploration from the **nlmeU** package. Note that the code used to create figure employs the function `xyplot()` from the package **lattice**

R3.1: *ARMD Trial*: Syntax for the plot of visual-acuity profiles for selected patients in Figure 3.1.

```
> data(armd.wide, armd0, package = "nlmeU")       # Data loaded
> library(lattice)
> armd0.subset <-                                 # Subset
+   subset(armd0, as.numeric(subject) %in% seq(1, 240, 10))
> xy1 <-                                          # Draft plot
+   xyplot(visual ~ jitter(time) | treat.f,
+          groups = subject,
+          data = armd0.subset,
+          type = "l", lty = 1)
> update(xy1,                                     # Fig. 3.1
+        xlab = "Time (in weeks)",
+        ylab = "Visual acuity",
+        grid = "h")
> detach(package:lattice)
```

(Sarkar, 2008). The function is applied to the subset of the data frame `armd0` (Sect. 2.2.2). The formula used in the syntax indicates that the variables `visual` and `time` are to be used on the y- and x-axis, respectively. These variables are plotted against each other in separate panels for different values of the `treat.f` factor. Within each panel, data points are grouped for each subject and connected using solid lines. The function `jitter()` is used to add a small amount of noise to the variable `time`, thereby reducing the number of overlapping points.

In the next subsections, we explore particular features of the ARMD data in more detail.

3.2.1 Patterns of missing data

First, we check the number and patterns of missing visual acuity measurements. To this aim, we use the data frame `armd.wide`. As mentioned in Sect. 2.2.2, the data frame contains the factor `miss.pat` that indicates which of the four post-randomization measurements are missing for a particular patient. For example, the pattern `--X-` indicates that the only missing measurement was at the third post-randomization timepoint, i.e., at 24 weeks.

In Panel R3.2, we use three different methods to tabulate the number of patients with different levels of the factor `miss.pat`. From the displayed results we can conclude that, for instance, there were 188 patients, for whom all four post-randomization visual-acuity measurements were obtained. On the other hand, there were six patients, for whom the four measurements were missing.

==== RSession R3.2 around here ====

R3.2: *ARMD Trial*: Inspecting missing-data patterns in `armd.wide` data for the post-randomization visual-acuity measurements using three different methods.

```
> table(armd.wide$miss.pat)
  ---- ---X --X- --XX -XX- -XXX X--- X-XX XXXX
   188   24    4    8    1    6    2    1    6
> with(armd.wide, table(miss.pat))
  miss.pat
  ---- ---X --X- --XX -XX- -XXX X--- X-XX XXXX
   188   24    4    8    1    6    2    1    6
> xtabs(~miss.pat, armd.wide)
  miss.pat
  ---- ---X --X- --XX -XX- -XXX X--- X-XX XXXX
   188   24    4    8    1    6    2    1    6
```

It is also worth noting that there are eight ($= 4+1+2+1$) patients with four different non-monotone missing-data patterns, i.e., with intermittent missing visual-acuity measurements. When modeling data with such patterns, extra care is needed when specifying variance-covariance structures. We will come back to this issue in Sect. 11.4.2.

3.2.2 Mean-value profiles

In this section, we investigate the number of missing values and calculate the sample means of visual acuity measurements for different visits and treatment groups. To this aim, in Panel R3.3 we use the "long"-format data frame `armd0`, which was described in Sect. 2.2.2.

==== RSession R3.3 around here ====

To calculate counts of missing values in Panel R3.3a, we use the function `tapply()`. In general, this function is used to apply a selected function to each (non-empty) group of values defined by a unique combination of the levels of one or more factors. In our case, the selected function, specified in the `FUN` argument, checks the length of the vector created by selecting non-missing values from the vector passed as an argument to the function. Using `tapply()` functin, we apply it to the variable `visual` within the groups defined by combinations of the levels of factors `time.f` and `treat.f`. As a result, we

R3.3: *ARMD Trial*: Sample means and medians for visual acuity by time and treatment.

(a) *Counts of non-missing visual-acuity measurements.*

```
> attach(armd0)
> flst <- list(time.f, treat.f)                    # "By" factors
> (tN <-                                           # Counts
+   tapply(visual, flst,
+          FUN = function(x) length(x[!is.na(x)])))
           Placebo Active
  Baseline     119    121
  4wks         117    114
  12wks        117    110
  24wks        112    102
  52wks        105     90
```

(b) *Sample means and medians of visual-acuity measurements.*

```
> tMn <- tapply(visual, flst, FUN = mean)          # Sample means
> tMd <- tapply(visual, flst, FUN = median)        # Sample medians
> colnames(res  <- cbind(tN, tMn, tMd))            # Column names
  [1] "Placebo" "Active"  "Placebo" "Active"  "Placebo" "Active"
> nms1 <- rep(c("P", "A"), 3)
> nms2 <- rep(c("n", "Mean", "Mdn"), rep(2, 3))
> colnames(res) <- paste(nms1, nms2, sep = ":")    # New column names
> res
           P:n A:n P:Mean A:Mean P:Mdn A:Mdn
  Baseline 119 121 55.336 54.579  56.0  57.0
  4wks     117 114 53.966 50.912  54.0  52.0
  12wks    117 110 52.872 48.673  53.0  49.5
  24wks    112 102 49.330 45.461  50.5  45.0
  52wks    105  90 44.438 39.100  44.0  37.0
> detach(armd0)
```

obtain a matrix with the number of non-missing visual-acuity measurements for each visit and each treatment group. We store the matrix in the object `tN` for further use. The display of the matrix indicates that there were no missing measurements at baseline. On the other hand, at week 4, for instance, there were two and seven missing measurements in the placebo and active-treatment arms, respectively. In general, there are more missing measurements in the active-treatment group.

In Panel R3.3b, we use the function `tapply()` twice to compute the sample means and sample medians of visual acuity measurements for each combi-

nation of the levels of factors `time.f` and `treat.f`. We store the results in matrices `tMn` and `tMd`, respectively. We then create the matrix `res` by combining matrices `tN`, `tMn`, and `tMn` by columns. Finally, to improve the legibility of displays, we modify the names of the columns of `res`.

From the display of the matrix `res` we conclude that, on average, there was very little difference in visual acuity between the two treatment groups at baseline. This is expected in a randomized study. During the course of the study, the mean visual acuity decreased with time in both arms, which confirms the observation made based on the individual profiles presented in Fig. 3.1. It is worth noting that the mean value is consistently higher in the placebo group, which suggests lack of effect of interferon-α.

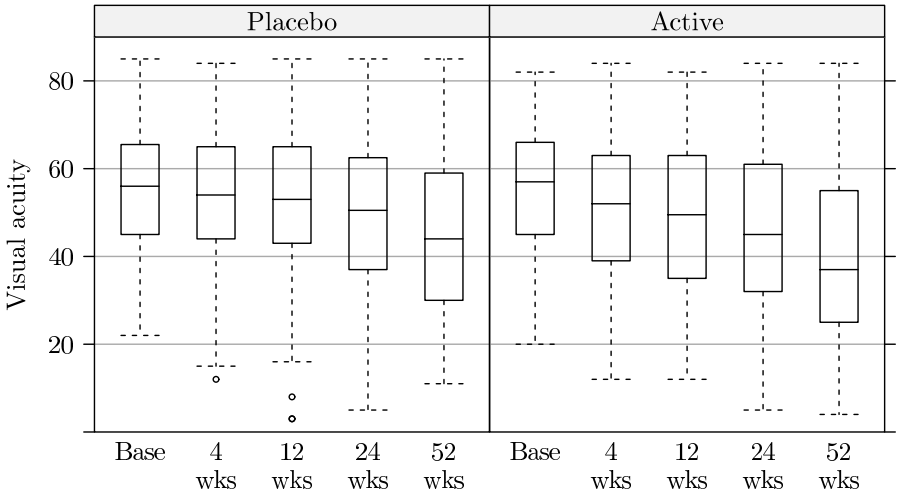


Fig. 3.2: *ARMD Trial*: Box-and-whiskers plots for visual acuity by treatment and time.

==== Fig.3.2 about here =====
(EPS:Figs/Figarmdmeans See: Figarmdmeans.eps)

Figure 3.2 presents box-and-whiskers plots of visual acuity for the five timepoints and the two treatment arms. The syntax to create the figure is shown in Panel R3.4. It uses the function `bwplot()` from the package **lattice**. Note that, we first create a draft of the plot, which we subsequently enhance by providing labels for the horizontal axis. In contrast to Fig. 3.1, measurements for all subjects at all timepoints are plotted. A disadvantage of the plot is that it does not reflect the longitudinal structure of the data.

=== Place Panel R3.4 about here =====

R3.4: *ARMD Trial*: Syntax for the box-and-whiskers plots in Fig. 3.2.

```
> library(lattice)
> bw1 <-                                              # Draft plot
+   bwplot(visual ~ time.f | treat.f,
+          data = armd0)
> xlims <- c("Base", "4\nwks", "12\nwks", "24\nwks", "52\nwks")
> update(bw1, xlim = xlims, pch = "|")                # Final plot
> detach(package:lattice)
```

The box-and-whiskers plots illustrate the patterns implied by the sample means and medians, presented in Panel R3.3b. The decrease of the mean values in time is clearly seen for both treatment groups. It is more pronounced for the active-treatment arm. As there was a slightly higher dropout in that arm, a possible explanation could be that patients, whose visual acuity improved, dropped out of the study. In such case, a faster progression of the disease in that treatment arm would be observed.

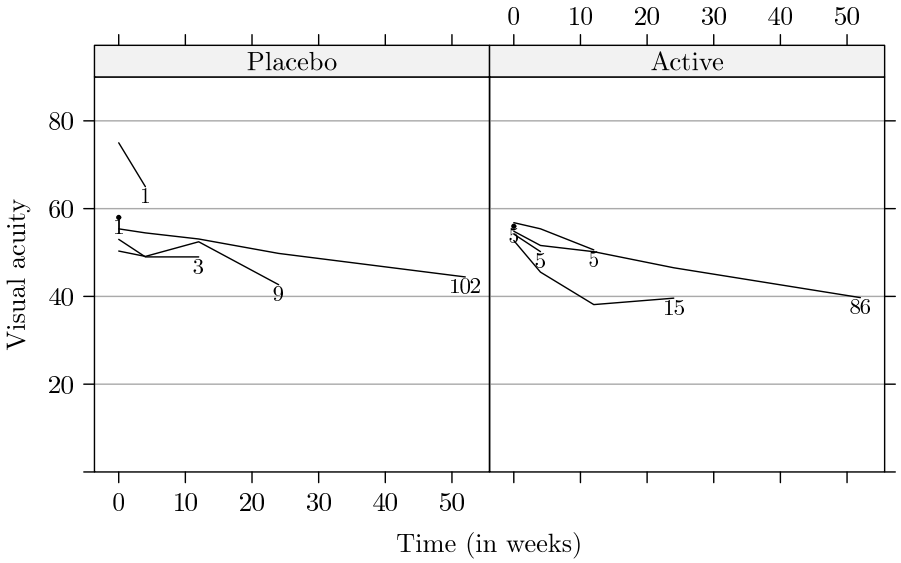


Fig. 3.3: *ARMD Trial*: Mean visual-acuity profiles by missing pattern and treatment (monotone missing-data patterns only).

==== Fig.3.3 about here =====
(EPS: Figs/Figarmdmissmeans.eps. Figarmdmissmeans.eps)

To check this possibility, we take a look at Fig. 3.3. It shows the mean values of visual acuity for patients with different monotone missing-data patterns. In addition, the number of subjects for each pattern is also given. We note that the number of subjects for the patterns with a larger number of missing values tends to be smaller. Note that, to save space, we do not present the syntax used to create the figure, as it is fairly complex.

The mean profiles, shown in Fig. 3.3, consistently decrease for the majority of the patterns. In general, they do not suggest an improvement in visual acuity before the drop off. Thus, they do not support the aforementioned explanation of a faster decrease of the mean visual acuity in the active-treatment arm.

In Panel R3.5, we present the syntax to investigate the number and form of monotone missing-data patterns for visual acuity. In particular, in Panel R3.5a, we create the data frame `armd.wide.mnt`, which contains data only for patients with monotone patterns. There are 232 such patients in total. Note that despite the fact that some patterns are not present in the data frame `armd.wide.mnt` they are still recognized as valid levels of the factor `miss.pat`. This might cause problems when using some R functions. Similarly to Panel R2.5, we could use the `droplevels()` function to remove the unused levels of the `miss.pat` variable. Instead, in Panel R3.5b, we modify the levels of the factor `miss.pat` in the `armd.wide.mnt` data with the help of the function `factor()`. Note that, instead of using the `levels` argument of the function, we could have used the argument `exclude` while indicating the levels to be excluded from the definition of the `miss.pat` factor.

==== RSession R3.5 around here ====

Finally, in Panel R3.5c, we use the function `tapply()` to obtain a matrix containing the number of patients for each monotone missing-data pattern and for each treatment arm. The displayed results indicate that the mean-value profiles for missing-data patterns with a larger number of missing values, shown in Fig. 3.3, are based on measurements for a small number of patients. Thus, the variability of these profiles is larger than for the patterns with a smaller number of missing values. Therefore, Fig. 3.3 should be interpreted with caution.

R3.5: *ARMD Trial*: The number of patients by treatment and missing-data pattern (monotone patterns only).

(a) *Subset of the data with monotone missing-data patterns.*

```
> mnt.pat<-                                    # Monotone patterns
+   c("----", "---X", "--XX", "-XXX", "XXXX")
> armd.wide.mnt <-                             # Data subset
+   subset(armd.wide, miss.pat %in% mnt.pat)
> dim(armd.wide.mnt)                           # Number of rows and cols
  [1] 232  10
> levels(armd.wide.mnt$miss.pat)               # Some levels not needed
  [1] "----" "---X" "--X-" "--XX" "-XX-" "-XXX" "X---" "X-XX"
  [9] "XXXX"
```

(b) *Removing unused levels from the* `miss.pat` *factor.*

```
> armd.wide.mnt1 <-
+   within(armd.wide.mnt,
+          {
+            miss.pat <- factor(miss.pat, levels=mnt.pat)
+          })
> levels(armd.wide.mnt1$miss.pat)
  [1] "----" "---X" "--XX" "-XXX" "XXXX"
```

(c) *The number of patients with different monotone missing-data patterns.*

```
> with(armd.wide.mnt1,
+      {
+        fl  <- list(treat.f, miss.pat)       # List of "by" factors
+        tapply(subject, fl, FUN=function(x) length(x[!is.na(x)]))
+      })
           ---- ---X --XX -XXX XXXX
  Placebo  102    9    3    1    1
  Active    86   15    5    5    5
```

3.2.3 Sample variances and correlations of visual-acuity measurements

Figure 3.4 shows a scatterplot matrix for the visual acuity measurements for those patients, for whom all post-randomization measurements are available. Scatterplots for corresponding pairs of variables are given below the diagonal. The size of the font for correlation coefficients reported above the diagonal is proportional to its value. We do not present the syntax for constructing the fig-

ure, as it is fairly complex. It can be observed that the measurements adjacent in time are strongly correlated. The correlation decreases with an increasing time gap. Worth noting is the fact that there is a substantial positive correlation between visual acuity at baseline and at the other post-randomization measurements. Thus, baseline values might be used to explain the overall variability of the post-randomization observations. This agrees with the observation made based on Fig. 3.1. It is worth noting that a scatterplot matrix of the type shown in Fig. 3.4 may not work well for longitudinal data with irregular time intervals.

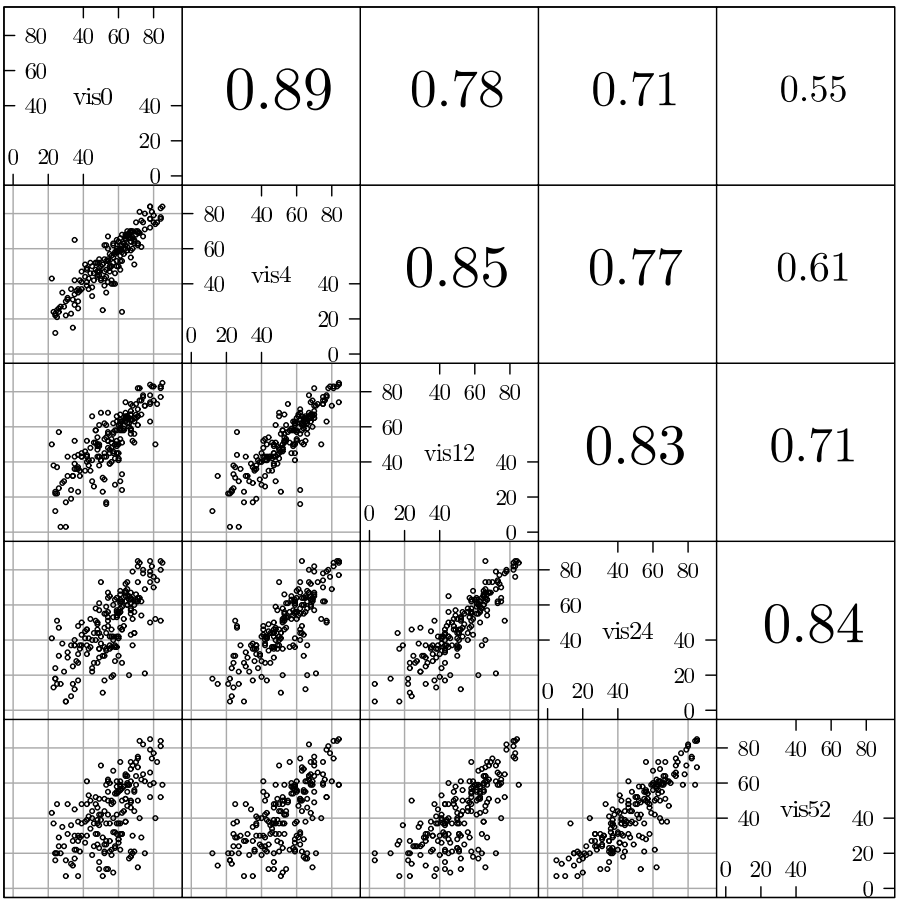


Fig. 3.4: *ARMD Trial*: Scatterplot matrix for visual acuity measurements. Scatterplots (below diagonal) and correlation coefficients (above diagonal) for complete cases only ($n = 188$).

==== Fig.3.4 about here =====
(EPS: Figs/Figarmdscatter.eps Figarmdscatter.eps)

=== Place Panel R3.6 about here ==========

R3.6: *ARMD Trial*: Variance-covariance and correlation matrices for visual-acuity measurements for complete cases only ($n = 188$).

```
> visual.x <- subset(armd.wide, select = c(visual0:visual52))
> (varx <- var(visual.x, use = "complete.obs")) # Var-cov mtx
          visual0 visual4 visual12 visual24 visual52
  visual0  220.31  206.71   196.24   193.31   152.71
  visual4  206.71  246.22   224.79   221.27   179.23
  visual12 196.24  224.79   286.21   257.77   222.68
  visual24 193.31  221.27   257.77   334.45   285.23
  visual52 152.71  179.23   222.68   285.23   347.43
> print(cor(visual.x, use = "complete.obs"),    # Corr mtx
+       digits = 2)
          visual0 visual4 visual12 visual24 visual52
  visual0     1.00    0.89     0.78     0.71     0.55
  visual4     0.89    1.00     0.85     0.77     0.61
  visual12    0.78    0.85     1.00     0.83     0.71
  visual24    0.71    0.77     0.83     1.00     0.84
  visual52    0.55    0.61     0.71     0.84     1.00
> diag(varx)                  # Var-cov diagonal elements
   visual0  visual4 visual12 visual24 visual52
    220.31   246.22   286.21   334.45   347.43
> cov2cor(varx)               # Corr mtx (alternative way)
  ...      [snip]
```

In Panel R3.6, we provide the estimates of the variance-covariance and correlation matrices for visual acuity measurements. To this aim, we create the data frame `visual.x` from `armd.wide` by selecting only the five variables containing the measurements. We then apply functions `var()` and `cor()` to estimate the variance-covariance matrix and the correlation matrix, respectively. Note that, for both functions, we specify the argument `use = "complete.cases"`, which selects only those rows of the data frame `visual.x` that do not contain any missing values. In this way, the estimated matrices are assured to be positive semidefinite. An alternative (not shown) would be to specify `use = "pairwise.complete.obs"`. In that case, the elements of the matrices would be estimated using data for all patients with complete observations for the particular pair of visual acuity measurements. This could result in estimates

of variance-covariance or correlation matrices, which might not be positive semidefinite.

The variance-covariance matrix for visual acuity measurements is stored in the `varx` matrix. It indicates an increase of the variance of visual acuity measurements obtained at later timepoints. The estimated correlation matrix suggests a moderate to strong correlation of the measurements. We also observe that the correlation clearly decreases with the time gap, as already concluded from Fig. 3.4.

At the bottom of Panel R3.6 we demonstrate how to extract the diagonal elements of the matrix `varx` using the `diag()` function. We also present the use of the function `cov2cor()` to compute a correlation matrix corresponding to the variance-covariance. Note that we do not display the result of the use of the function, as it is exactly the same as the one obtained for the function `cor()`, already shown in Panel R3.6.

3.3 PRT Study: Muscle fiber specific-force

In the PRT study, we are primarily interested in the effect of the intensity of the training on the muscle fiber specific-force, measurements of which are contained in the variable `spec.fo` of the `prt` data frame (Sect. 2.3.2). In some analyses, we will also investigate the effect on the measurements of the isometric force, which are stored in the variable `iso.fo`.

First, however, we take a look at the information about subjects' characteristics, stored in the data frame `prt.subjects` (see Sect. 2.3.2). In Panel R3.7, we use the function `tapply()` to obtain summary statistics for the variable `bmi` for separate levels of the `prt.f` factor. The statistics are computed with the help of the `summary()` function. The displayed values of the statistics do not indicate any substantial differences in the distribution of BMI between subjects assigned to the low- or high-intensity training. Given that the assignment was randomized, this result is anticipated.

=== Place Panel R3.7 about here =====

For illustration purposes, we also obtain summary statistics for all variables in the `prt.subjects` data frame, except for `id`, with the help of the function `by()`. The function splits the data frame according to the levels of the factor `prt.f` and applies the function `summary()` to the two data frames resulting from the split. As a result, we obtain summary statistics for variables `prt.f`, `age.f`, `sex.f`, and `bmi` for the two training-intensity groups. From the displayed values of the statistics we conclude that there are no important differences in the distribution of sex and age-groups between the two inter-

R3.7: *PRT Trial*: Summary statistics for subjects' characteristics.

```
> data(prt.subjects, prt, package = "nlmeU") # Data loaded
> with(prt.subjects, tapply(bmi, prt.f, summary))
  $High
     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     18.4    22.9    24.8    25.1    28.2    31.0

  $Low
     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     19.0    23.1    24.8    24.7    26.3    32.3
> by(subset(prt.subjects, select = -id), prt.subjects$prt.f, summary)
  prt.subjects$prt.f: High
    prt.f      age.f       sex.f         bmi
   High:31   Young:15   Female:17   Min.   :18.4
   Low : 0   Old  :16   Male  :14   1st Qu.:22.9
                                    Median :24.8
                                    Mean   :25.1
                                    3rd Qu.:28.2
                                    Max.   :31.0
  ------------------------------------------------
  prt.subjects$prt.f: Low
    prt.f      age.f       sex.f         bmi
   High: 0   Young:15   Female:17   Min.   :19.0
   Low :32   Old  :17   Male  :15   1st Qu.:23.1
                                    Median :24.8
                                    Mean   :24.7
                                    3rd Qu.:26.3
                                    Max.   :32.3
```

vention groups. This is expected, given that the randomization was stratified by the two factors (see Sect. 2.3). Note that we should ignore the display for the factor `prt.f`, because it has been used for splitting the data.

=== Place Panel R3.8 about here =====

In Panel R3.8, we take a look at fiber measurements stored in the data frame `prt`. In particular, in Panel R3.8a, we check the number of non-missing measurements of the specific force per fiber type and occasion for selected subjects. To this aim, with the help of the function `tapply()`, we apply the function `length()` to the variable `spec.fo` for separate levels of the `id`, `fiber.f`, and `occ.f` factors. Note that, in the call to the function `tapply()`, we use a named list of the factors. The names of the components of the list are shortened versions of the factor names. In this way, we obtain a more legible display of the resulting array. In Panel R3.8a, we show the display for two subjects, `"5"` and

R3.8: *PRT Trial*: Extracting and summarizing the fiber-level information.

(a) *Number of fibers per type and occasion for the subjects "5" and "335".*

```
> fibL <-
+    with(prt,
+         tapply(spec.fo,
+                list(id = id, fiberF = fiber.f, occF = occ.f),
+                length))
> dimnms <- dimnames(fibL)
> names(dimnms)         # Shortened names displayed
  [1] "id"     "fiberF" "occF"
> fibL["5", , ]         # Number of fiber measurements for subject 5
         occF
  fiberF   Pre Pos
    Type 1  12  18
    Type 2   7   4
> fibL["335", , ]       # Number of fiber measurements for subject 335
         occF
  fiberF   Pre Pos
    Type 1  NA   8
    Type 2  14  11
```

(b) *Mean value of `spec.fo` by fiber type and occasion for subject "5".*

```
> fibM <-
+    with(prt,
+         tapply(spec.fo,
+                list(id = id, fiberF = fiber.f, occF = occ.f),
+                mean))
> fibM["5", , ]
         occF
  fiberF      Pre    Pos
    Type 1 132.59 129.96
    Type 2 145.74 147.95
```

"335". For the latter, we see that no measurements of the specific force were taken for type-1 fibers before the training.

In Panel R3.8b, we take a look at the mean value of the specific force per fiber type and occasion for selected subjects. To this aim, we use the function `tapply()` in a similar way as in Panel R3.8a, but in combination with the function `mean()`. In the panel, we display the mean values for the subject "5".

=== Place Panel R3.9 about here =====

R3.9: *PRT Trial*: Summarizing the fiber-level information with the help of functions `melt()` and `cast()` from the **reshape** package.

(a) *Pre-processing of the data (melting).*

```
> library(reshape)
> idvar <- c("id", "prt.f", "fiber.f", "occ.f")
> meas.var <- c("spec.fo", "iso.fo")
> prtM <-                                          # Melting data
+   melt(prt, id.var = idvar, measure.var = meas.var)
> dim(prtM)
  [1] 4942    6
> head(prtM, n = 4)                                # First four rows
    id prt.f fiber.f occ.f variable value
  1  5   Low  Type 1   Pre  spec.fo  83.5
  2  5   Low  Type 1   Pre  spec.fo 132.8
  3  5   Low  Type 2   Pre  spec.fo 161.1
  4  5   Low  Type 1   Pre  spec.fo 158.8
> tail(prtM, n = 4)                                # Last four rows
       id prt.f fiber.f occ.f variable value
  4939 520  High  Type 2   Pos   iso.fo 0.527
  4940 520  High  Type 1   Pos   iso.fo 0.615
  4941 520  High  Type 2   Pos   iso.fo 0.896
  4942 520  High  Type 2   Pos   iso.fo 0.830
```

(b) *Aggregating data (casting).*

```
> prtC <- cast(prtM, fun.aggregate = mean)       # Casting data
> names(prtC)
  [1] "id"      "prt.f"   "fiber.f" "occ.f"   "spec.fo" "iso.fo"
> names(prtC)[5:6] <- c("spec.foMn", "iso.foMn") # Names modified
> head(prtC, n = 4)
    id prt.f fiber.f occ.f spec.foMn iso.foMn
  1  5   Low  Type 1   Pre    132.59  0.51500
  2  5   Low  Type 1   Pos    129.96  0.72289
  3  5   Low  Type 2   Pre    145.74  0.47057
  4  5   Low  Type 2   Pos    147.95  0.71175
```

In Panel R3.9, we illustrate how to summarize the fiber-level information using functions from the package **reshape**. First, in Panel R3.9a, we use the generic function `melt()` to prepare the data for further processing. More specifically, we apply the function to the data frame `prt` and we specify factors `id`, `prt.f`, `fiber.f`, and `occ.f` as "identifying variables." On the other hand, we indicate variables `spec.fo` and `iso.f` as "measured variables." In the resulting data frame, `prtM`, the values of the measured variables are "stacked" within the groups defined by the combinations of the levels of the identifying variables. The stacked values are stored in a single variable named, by default, `value`. They are identified by the levels of factor named, by default, `variable`, which contain the names of the measured variables.

The display, shown in Panel R3.9a, indicates that the number of records in the data frame `prtM` increases to 4942, as compared to 2471 records in the data frame `prt` (see Panel R2.7). The increase results from the stacking of the values of `spec.fo` and `iso.fo` in the variable `value`. The outcome of the process is further illustrated by the display of the first and last four rows of the data frame `prtM`.

In Panel R3.9b, we apply the function `cast()` to the data frame `prtM` to compute the mean values of the measured variables, i.e., `spec.fo` and `iso.fo`, within the groups defined by the combinations of the levels of the identifying variables. To indicate that we want to compute the mean values, we use the argument `fun.aggregate=mean`. The resulting data frame is stored in the object `prtC`. Before displaying the contents of the object, we modify the names of the two last variables, which contain the mean values of `spec.fo` and `iso.fo`. The display of the first four records of `prtC` shows the means per fiber type and occasion for the subject `"5"`. Note that, for `spec.fo`, the mean values correspond to the values reported at the end of Panel R3.8.

==== Fig.3.5 about here =====
(EPS: Figs/FigMLSexplore11 See: FigMLSexplore11.eps)

Figure 3.5 shows the pre- and post-training mean values of the specific force for all subjects separately for the two fiber types and training intensities. The figure was created using the function `dotplot()` from the package **lattice**. To increase interpretability of this figure, we ordered the subjects on y-axis within each study group by mean values of the pre-training `spec.fo` for type-1 fibers. If for a given subject like, e.g., "335", the pre-training measures were not available, the post-training measures were used instead. For brevity, we do not show the syntax used to create the figure.

Several observations can be made based on the figure:

- there is no clear effect of the training intensity;

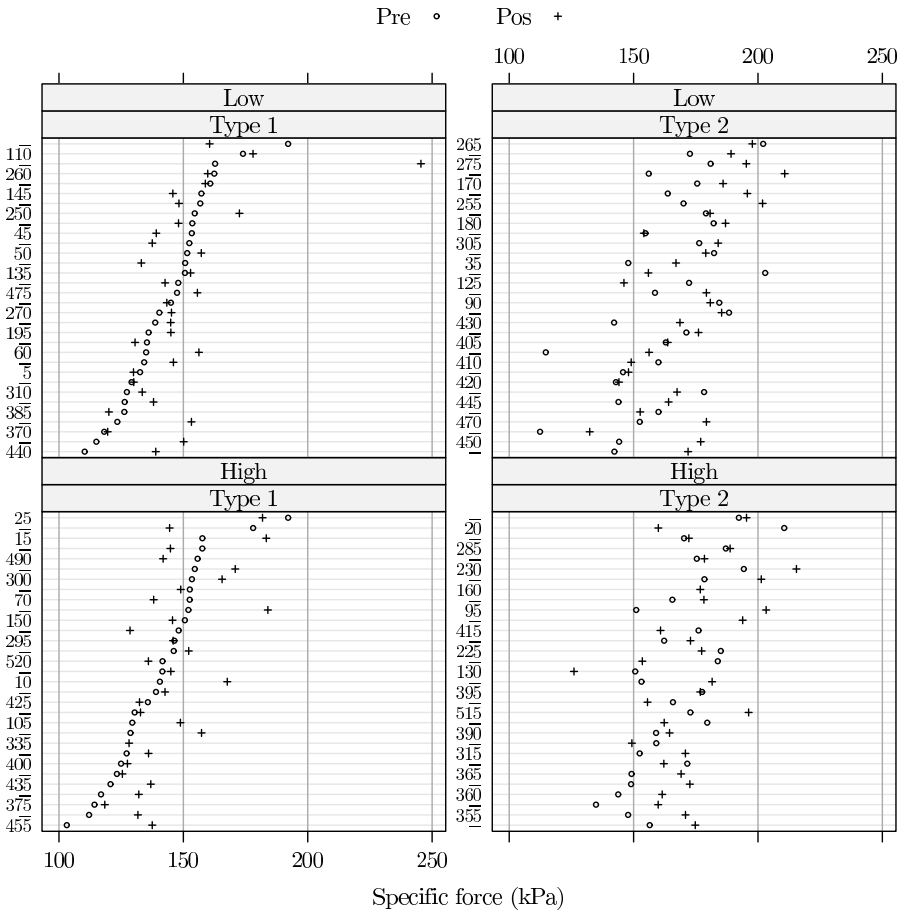


Fig. 3.5: Individual means for specific force by occasion, fiber type, and training intensity.

- in general, measurements of the specific force are higher for type-2 than for type-1 fibers;
- on average, post-training values are larger than pre-training measurements;
- for both types of fibers there is considerable variability between subjects with respect to the overall level of measurements and with respect to the magnitude of the post-pre differences;
- there is a correlation between the mean measurements observed for the same individual, as seen, e.g., from the similar pattern of measurements for both types.

These observations will be taken into account when modeling the data in Part IV of the book.

Note that the plot in the lower-left panel of Fig. 3.5 confirms the missing pre-training measurements for type-1 fibers for the subject `"335"`.

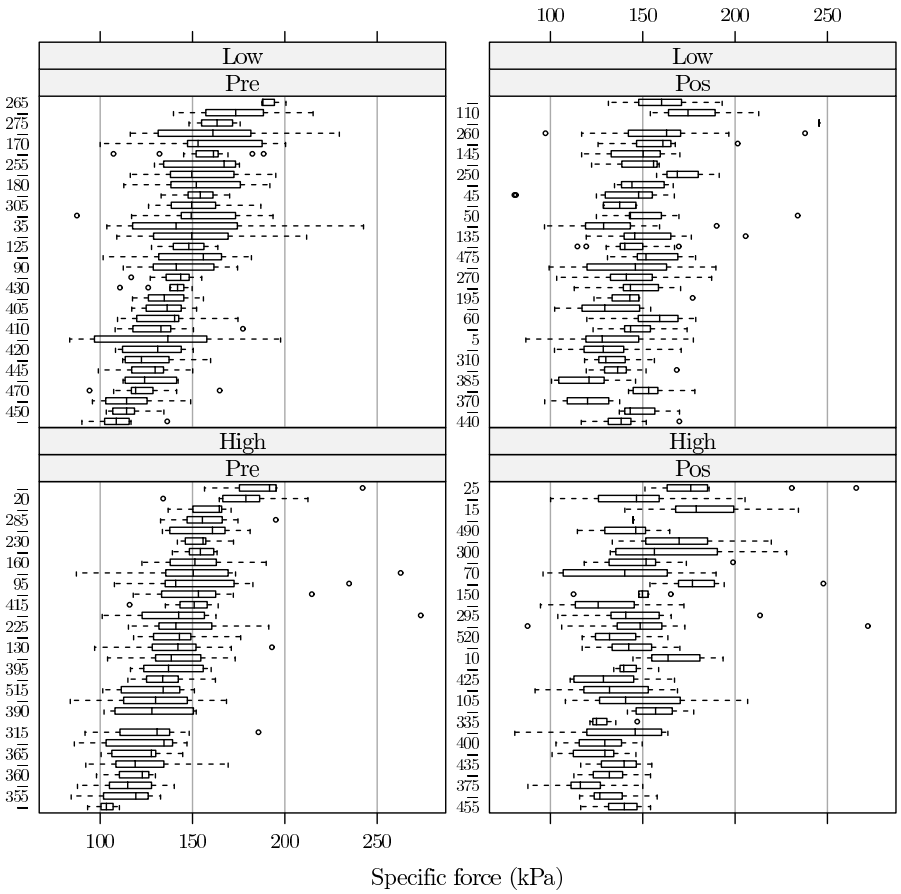


Fig. 3.6: *PRT Trial*: Subject-specific box-and-whiskers plots for the specific force by training intensity and measurement occasion (type-1 fibers only).

==== Fig.3.6 about here =====
(EPS: Figs/Figmlsbwhisk2.eps) See: Figmlsbwhisk2.eps)

Figure 3.6 presents information for the specific force for the type-1 fibers. More specifically, it shows box-and-whiskers plots for the individual measurements

of the specific force for the two measurement occasions and training intensities. All 63 subjects on the y-axis are ordered in the same way as in Fig. 3.5. Figure 3.6 was created using the function `bw()` from the package **lattice**. Note, however, that we do not present the detailed code. The plots suggest that the subject-specific variances of the pre-training measurements are somewhat smaller than the post-training ones. There is also a considerable variability between the subjects with respect to the variance of the measurements.

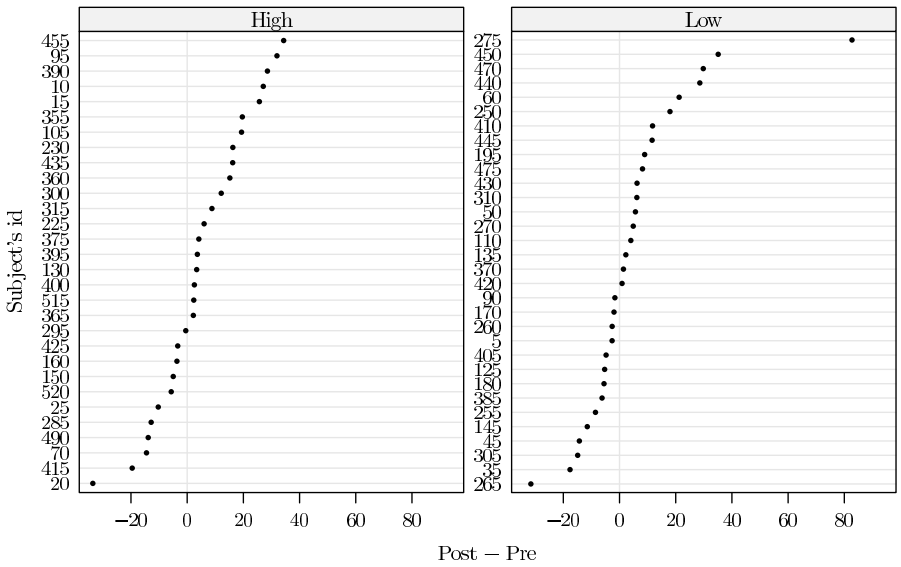


Fig. 3.7: *PRT Trial*: Individual pre-post differences of the mean values for the specific force, ordered by an increasing value, for the two training intensity groups (type-1 fibers only).

==== Fig.3.7 about here =====
(EPS: Figs/MLSexploreFig22.eps See: MLSxploreFig22.eps)

Figure 3.7 presents the individual pre-post differences of the mean values for the specific force for the type-1 fibers for the two training-intensity groups. The differences were ordered according to increasing values within each training group. To conserve space, we do not show the syntax used to create the figure. The plots indicate an outlying value of the difference for the subject `"275"` in the low-intensity training group.

3.4 SII Project: Gain in the math achievement-score

In this section, we conduct an exploratory analysis of the SII data that were described in Sect. 2.4. We focus on the measurements of the gain in the math achievement-score, stored in the variable `mathgain` (see Sect. 2.4.1). Given the hierarchical structure of the data, we divide the analysis into three parts, in which we look separately at the school, class-, and child-level data.

=== Place Panel R3.10 about here==========

R3.10: *SII Project*: The number of missing values for variables included in the `SIIdata` data frame.

```
> data(SIIdata, package = "nlmeU")
> sapply(SIIdata, FUN = function(x) any(is.na(x)))
       sex minority mathkind mathgain      ses yearstea mathknow
     FALSE    FALSE    FALSE    FALSE    FALSE    FALSE     TRUE
  housepov mathprep  classid schoolid  childid
     FALSE    FALSE    FALSE    FALSE    FALSE
> sum(as.numeric(is.na(SIIdata$mathknow)))
  [1] 109
> range(SIIdata$mathknow, na.rm = TRUE)
  [1] -2.50  2.61
```

First, however, we check whether the data frame `SIIdata` contains complete information for all variables for all pupils. To this aim, in Panel R3.10, we use the function `sapply()`. It applies the function, specified in the `FUN` argument, to each column (variable) of the data frame `SIIdata`. The latter function checks whether any value in a particular column is missing. The displayed results indicate that only the variable `mathknow` contains missing values. By applying the function `sum()` to the vector resulting from the transformation of a logical vector indicating the location of missing values in the variable `mathknow` to a numeric vector, we check that the variable contains 109 missing values. The non-missing values range from -2.50 to 2.61.

3.4.1 School-level data

In this section, we investigate the school-level data.

First, in Panel R3.11, we use the function `xtabs()` to tabulate the number of pupils per school. The result is stored in the array `schlN`. The display of

the array is difficult to interpret. By applying the function `range()`, we check that the number of pupils per school varied between 2 and 31. By applying the function `xtabs()` to the array `schlN`, we obtain the information about the number of schools with a particular number of pupils. For instance, there were two schools, for which data for only two pupils are included in the data frame `SIIdata`. On the other hand, there was only one school, for which data for 31 pupils were collected.

=== Place Panel R3.11 about here==========

R3.11: *SII Project*: Extracting the information about the number of pupils per school.

```
> (schlN <- xtabs(~schoolid, SIIdata))  # Number of pupils per school
schoolid
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
 11  10  14   6   6  12  14  16   6  18  31  27   9  15  13   6
...     [snip]
 97  98  99 100 101 102 103 104 105 106 107
  6   2  19  13  16  11   8   6  10   2  10
> range(schlN)
[1]  2 31
> xtabs(~schlN) # Distribution of the number of pupils over schools
schlN
 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
 2  4  6  5  8  5  9  9 10  7  7  6  3  5  4  2  2  3  1  2  2
24 27 31
 1  3  1
```

In Panel R3.12, we obtain the information about the mean value of variables `mathkind` and `mathgain` for each school (see Sect. 2.4.1). To this aim, with the help of the function `by()`, we apply the function `colMeans()` to the values of the two variables within the groups defined by the same level of the factor `schoolid`, i.e., within each school. Note that the resulting output has been abbreviated.

=== Place Panel R3.12 about here==========

Panel R3.13 shows the syntax for constructing the data frame `schlDt`, which contains the school-specific means of variables `mathgain`, `mathkind`, and `housepov`. In particular, in Panel R3.13a, we use functions `melt()` and `cast()` (for an explanation of the use of the functions, see the description of Panel R3.9) to create the data frame `cst1`, which contains the number of classes and children for each school. On the other hand, in Panel R3.13b,

R3.12: *SII Project*: Computation of the mean value of pupils' math scores for each school.

```
> attach(SIIdata)
> (mthgM <- by(cbind(mathgain, mathkind), schoolid, colMeans))
  INDICES: 1
  mathgain mathkind
    59.636  458.364
  ------------------------------------------------
  ...      [snip]
  ------------------------------------------------
  INDICES: 107
  mathgain mathkind
      48.2    464.2
> detach(SIIdata)
```

we use the functions to create the data frame `cst2` with the mean values of variables `mathgain`, `mathkind`, and `housepov` for each school. Finally, in Panel R3.13c, we merge the two data frames to create `schlDt`. Note that, after merging, we remove the two auxiliary data frames.

== Place Panel R3.13 about here==========

The data frame `schlDt` is used in Panel R3.14 to explore the school-specific mean values of variables `housepov` and `mathgain`. In particular, in Panel R3.14a, we use the function `summary()` to display the summary statistics for the mean values. On the other hand, in Panel R3.14b, we use the function `xyplot()` from the package **lattice** to construct scatterplots of the mean values of the variable `mathgain` *versus* variables `housepov` and `mthkMn`.

== Place Panel R3.14 about here==========

The scatterplots are shown in Fig. 3.8. The plot in Fig. 3.8a does not suggest a strong relationship between the school-specific mean values of `mathgain` and `housepov`. On the other hand, in Fig. 3.8b there is a strong negative relationship between the mean values of `mathgain` and `mathkind`: the larger the mean for the latter, the lower the mean for the former. The relationship suggests that the higher the teacher's knowledge of first-grade math contents, the lower the mean gain in the math achievement-score of pupils. Note that the plots in Fig. 3.8 should be interpreted with caution, as they show school-specific means, which were estimated based on different numbers of observations.

==== Fig. 3.8 about here==
(EPS: Figs/SiieploreFig50A-B))

R3.13: *SII Project*: Constructing a data frame with summary data for schools.

(a) *Creating a data frame with the number of classes and children for each school.*

```
> library(reshape)
> idvars <- c("schoolid")
> mvars <- c("classid", "childid")
> dtm1 <- melt(SIIdata, id.vars = idvars, measure.vars = mvars)
> names(cst1  <-
+   cast(dtm1,
+        fun.aggregate = function(el) length(unique(el))))
  [1] "schoolid" "classid"  "childid"
> names(cst1) <- c("schoolid", "clssn", "schlN")
```

(b) *Creating a data frame with the school-specific means of selected variables.*

```
> mvars <- c("mathgain", "mathkind", "housepov")
> dtm2 <- melt(SIIdata, id.vars = idvars, measure.vars = mvars)
> names(cst2 <- cast(dtm2, fun.aggregate = mean))
  [1] "schoolid" "mathgain" "mathkind" "housepov"
> names(cst2) <- c("schoolid", "mthgMn", "mthkMn", "housepov")
```

(c) *Merging the data frames created in parts (a) and (b) above.*

```
> (schlDt  <- merge(cst1, cst2, sort = FALSE))
      schoolid clssn schlN mthgMn mthkMn housepov
  1          1     2    11 59.636 458.36    0.082
  2          2     3    10 65.000 487.90    0.082
  3          3     4    14 88.857 469.14    0.086
  4          4     2     6 35.167 462.67    0.365
  ...     [snip]
  107      107     2    10 48.200 464.20    0.177
> rm(cst1, cst2)
```

3.4.2 Class-level data

In this section, we investigate the class-level data.

First, in Panel R3.15, we use the function `xtabs()` to tabulate the number of pupils per class. The result is stored in the array `clssN`. By applying the function `sum()` to the array, we check that the total number of pupils is 1190, in agreement with the information obtained, e.g., in Panel R2.10. With the help of the function `range()`, we find that the number of pupils per class varies between 1 and 10. By applying the function `xtabs()` to the array `clssN`, we

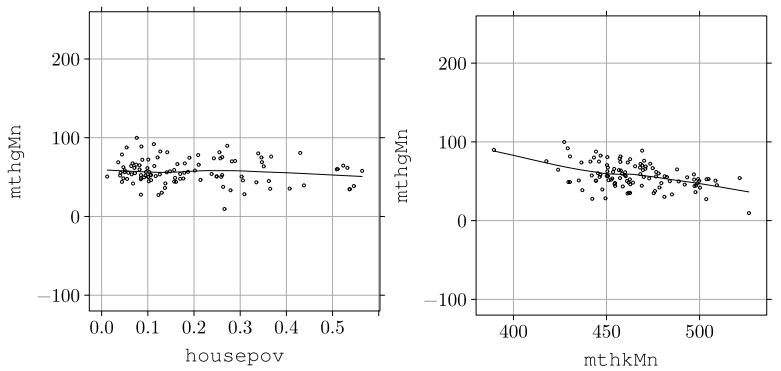
R3.14: *SII Project*: Exploring the school-level data. The data frame `schlDt` was created in Panel R3.13.

(a) *Summary statistics for the school-specific mean values of* ***housepov****.*

```
> summary(schlDt$housepov)
     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.0120  0.0855  0.1480  0.1940  0.2640  0.5640
```

(b) *Scatterplots of the school-specific mean values for* ***housepov*** *and* ***mathkind****.*

```
> library(lattice)
> xyplot(mthgMn ~ housepov,                          # Fig. 3.8a
+        schlDt, type = c("p", "smooth"), grid = TRUE)
> xyplot(mthgMn ~ mthkMn,                            # Fig. 3.8b
+        schlDt, type = c("p", "smooth"), grid = TRUE)
```



(a) *vs.* `housepov` (b) *vs.* `mthkMn`

Fig. 3.8: *SII Project*: Scatterplots of the school-specific mean values of the variable `mathgain` *versus* variables `housepov` and `mthkMn`.

obtain information about the number of classes with a particular number of pupils. The information is stored in the array `clssCnt`. The display of the array indicates that, for instance, there were 42 classes with only one pupil included in the data frame `SIIdata`. On the other hand, there were two classes, for which data for 10 pupils were collected. Finally, by applying the function `sum()` to the array `clssCnt`, we verify that the data frame `SIIdata` contains information about 312 classes.

=== Place Panel R3.15 about here==========

R3.15: *SII Project*: Extracting the information about the number of pupils per class.

```
> (clssN <- xtabs(~ classid, SIIdata))
  classid
    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
    5   3   3   6   1   5   1   4   3   2   4   5   9   4   1   6
  ...     [snip]
  305 306 307 308 309 310 311 312
    4   4   4   3   3   3   2   4
> sum(clssN)                          # Total number of pupils
  [1] 1190
> range(clssN)
  [1]  1 10
> (clssCnt <- xtabs(~clssN)) # Distribution of no. of pupils/classes
  clssN
   1  2  3  4  5  6  7  8  9 10
  42 53 53 61 39 31 14 13  4  2
> sum(clssCnt)                        # Total number of classes
  [1] 312
```

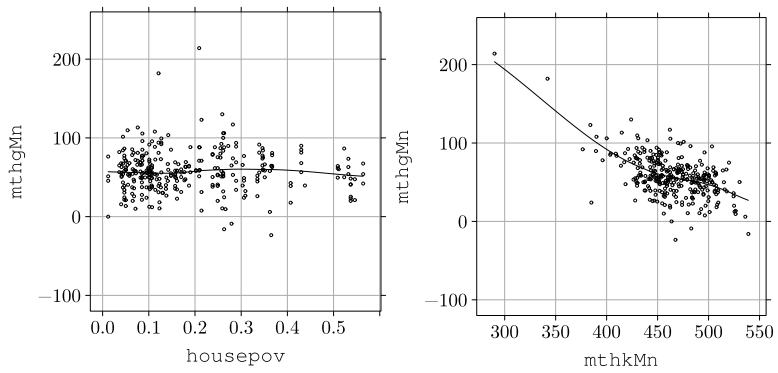
In Panel R3.16, we present an abbreviated printout of the contents of the data frame `clssDt`. The data frame contains the mean values of variables `mathgain` and `mathkind` for each class, together with the count of pupils, `clssN`. It also includes the values of the class-level variables `mathknow` and `mathprep` and the school-level variable `housepov`. The data frame was created using a syntax (not shown) similar to the one presented in Panel R3.13.

=== Place Panel R3.16 about here==========

R3.16: *SII Project*: Contents of the class-level data. The auxiliary data frame `clssDt` was created using a syntax similar to the one shown in Panel R3.13.

```
> clssDt
      classid housepov mathknow mathprep clssN   mthgMn mthkMn
  1         1    0.335    -0.72     2.50     5  47.8000 459.00
  2         2    0.303     0.58     3.00     3  65.6667 454.00
  3         3    0.040     0.85     2.75     3  15.6667 492.67
  4         4    0.339     1.08     5.33     6  91.5000 437.00
  ...     [snip]
  312     312    0.546    -1.37     2.00     4  47.5000 418.50
```

Figure 3.9 presents scatterplots of the class-specific means of the variable `mathgain` *versus* the values of the variable `housepov` and *versus* the class-specific means of the variable `mathkind`. The figure was created by a syntax similar to the one presented in Panel R3.14b based on the data from the data frame `clssDt`. Figure 3.9a does not suggest a strong relationship between the mean values of `mathgain` and `housepov`. On the other hand, as seen in Fig. 3.9b, there is a strong negative relationship between the mean values of `mathgain` and `mathkind`. These conclusions are similar to the ones drawn based on Fig. 3.8. As was the case for the latter figure, the plots in Fig. 3.9 should be interpreted with caution, as they show class-specific mean values estimated based on different numbers of observations.



(a) *vs.* `housepov` (b) *vs.* `mthkMn`

Fig. 3.9: *SII Project*: Scatterplots of the class-specific mean values of the variable `mathgain` *versus* variables `housepov` and `mthkMn`.

==== Fig. 3.9 about here ==
(EPS: Figs/SiieploreFig90A,B)

3.4.3 Pupil-level data

In this section, specifically in Panel R3.17, we explore the pupil-level data.

== Place Panel R3.17 about here==========

First, in Panel R3.17a, we construct an auxiliary data frame `auxDt` by merging data frames `SIIdata` and `clssDt`. Note that the latter contains the class-level data, including the means of variables `mathgain` and `mathkind` and the number of pupils (see Panel R3.16). Next, with the help of the function `with()`, we add a new factor, `clssF`, to `auxDt` and store the resulting data frame in

R3.17: *SII Project*: Exploring the pupil-level data. The data frame `clssDt` was created in Panel R3.16.

(a) *Adding the class-level data to the data frame* ***SIIdata***.

```
> auxDt <- merge(SIIdata, clssDt, sort = FALSE)
> auxDt2  <-
+   within(auxDt,
+          {
+           auxL  <- paste(classid, schoolid, sep = "\n:")
+           auxL1 <- paste(auxL, clssN, sep = "\n(")
+           auxL2 <- paste(auxL1, ")", sep = "")
+           clssF <-                              # Factor clssF created
+             factor(schoolid:classid, labels = unique(auxL2))
+          })
> tmpDt <- subset(auxDt2, select = c(classid, schoolid, clssN, clssF))
> head(tmpDt, 4)                                # First four records
  classid schoolid clssN        clssF
1     160        1     3 160\n:1\n(3)
2     160        1     3 160\n:1\n(3)
3     160        1     3 160\n:1\n(3)
4     217        1     8 217\n:1\n(8)
> tail(tmpDt, 4)                                # Last four records
     classid schoolid clssN          clssF
1187      96      107     8  96\n:107\n(8)
1188      96      107     8  96\n:107\n(8)
1189     239      107     2 239\n:107\n(2)
1190     239      107     2 239\n:107\n(2)
```

(b) *Scatterplots of the pupil-level data.*

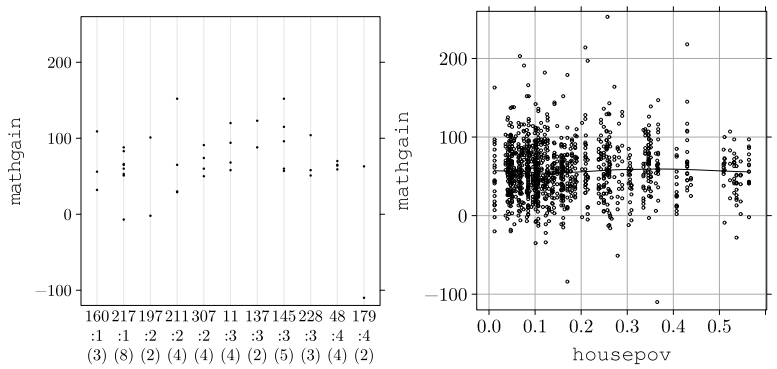
```
> library(lattice)
> dotplot(mathgain ~ clssF,                     # Fig. 3.10a
+         subset(auxDt2, schoolid %in% 1:4))
> xyplot(mathgain ~ housepov, SIIdata,          # Fig. 3.10b
+        type = c("p", "smooth"))
> detach(package:lattice)
```

the object `auxDt2`. The factor `clssF` combines the information about the class and the school for each pupil. The information is stored in a character string of the form: `classid\n:schoolid\n(clssN)`. The particular format of the string will prove useful in the construction of plots of the pupil-specific data. The format is illustrated in the display of the first and last four records of the data frame `auxDt2`. Note that we limit the display to variables `classid`, `schoolid`, `clssN`, and `clssF`.

In Panel R3.17b, we construct two plots of the pupil-level data. First, by applying the function `dotplot()` from the package **lattice** to the data frame `auxDt2`, we plot the values of the variable `mathgain` *versus* the levels of the factor `clssF` for the schools with `schoolid` between 1 and 4. Then, using the function `xyplot()` from the package **lattice**, we plot the values of the variable `mathgain` *versus* the values of the variable `houspov` for all pupils from the data frame `SIIdata`. The resulting plots are shown in Fig. 3.10.

The plot shown in Fig. 3.10a indicates considerable variability of the observed values of the gain in the math achievement-score even between the classes belonging to the same school. Note that the interpretation of the plot is much enhanced by the labels provided on the horizontal axis. The construction of the labels is facilitated by the chosen format of the levels of the factor `clssF`.

The plot shown in Fig. 3.10b indicates the lack of a relationship between the observed values of the gain in the math achievement-score for individual pupils and the values of the variable `housepov`. Note that a similar conclusion was drawn for the school- and class-specific mean values of `mathgain` based on Fig. 3.8a and 3.9a, respectively.



(a) *vs.* `clssF` (selected schools) (b) *vs.* `housepov`

Fig. 3.10: *SII Project*: Scatterplots of the observed values of `mathgain` for individual pupils *versus* the school/class indicator and the variable `housepov`.

==== Fig. 3.10 about here ==
(EPS: Figs/Fig.SiieploreFig400A,B.eps)

3.5 FCAT Study: Target score

The FCAT data set have a rather simple structure and contents (see Sect. 2.5). The main interest pertains to the distribution of the scores for the nine attainment targets, which are stored in the variable `scorec` of the data frame `fcat` (see Sect. 2.5.2). In Panel R3.18, we present syntax addressing this issue.

== Place Panel R3.18 about here==========

R3.18: *FCAT Study*: Summarizing the information about the total scores for attainment targets.

(a) *Summarizing scores for each child and attainment target.*

```
> data(fcat, package = "nlmeU")
> (scM <- with(fcat, tapply(scorec, list(id, target), mean)))
     T1(4) T2(6) T3(8) T4(5) T5(9) T6(6) T7(8) T8(6) T9(5)
1        4     6     4     1     7     6     6     5     5
2        3     4     6     2     7     4     6     3     3
...    [snip]
539      0     3     5     1     6     3     5     2     4
```

(b) *Histograms of scores for different attainment targets.*

```
> library(lattice)
> histogram(~scorec | target, data = fcat, # Fig. 3.11
+           breaks = NULL)
> detach(package:lattice)
```

In Panel R3.18a, we present how to obtain the mean value of the dependent variable for each combination of levels of the crossed factors, i.e. `id` and `target` in the `fcat` data frame. In particular, we use the function `tapply()` to apply the function `mean()` to the variable `scorec` for each combination of levels of the crossed factors. As a result, we obtain the matrix `scM`, which contains the mean value of the total score for each child and each attainment target. Obviously, in our case, there is only one observation for each child and target. Thus, by displaying a (abbreviated) summary of the matrix `scM`, we obtain, in fact, a tabulation of individual scores for all children.

In Panel R3.18b, we use the function `histogram()` from the package **lattice** to construct a histogram of the observed values of total scores for each attainment target. The resulting histograms are shown in Fig. 3.11. They clearly illustrate the differences in the measurement scale for different targets, which result from

the varying number of items per target (see Sect. 2.5). Some asymmetry of the distribution of the scores can also be observed.

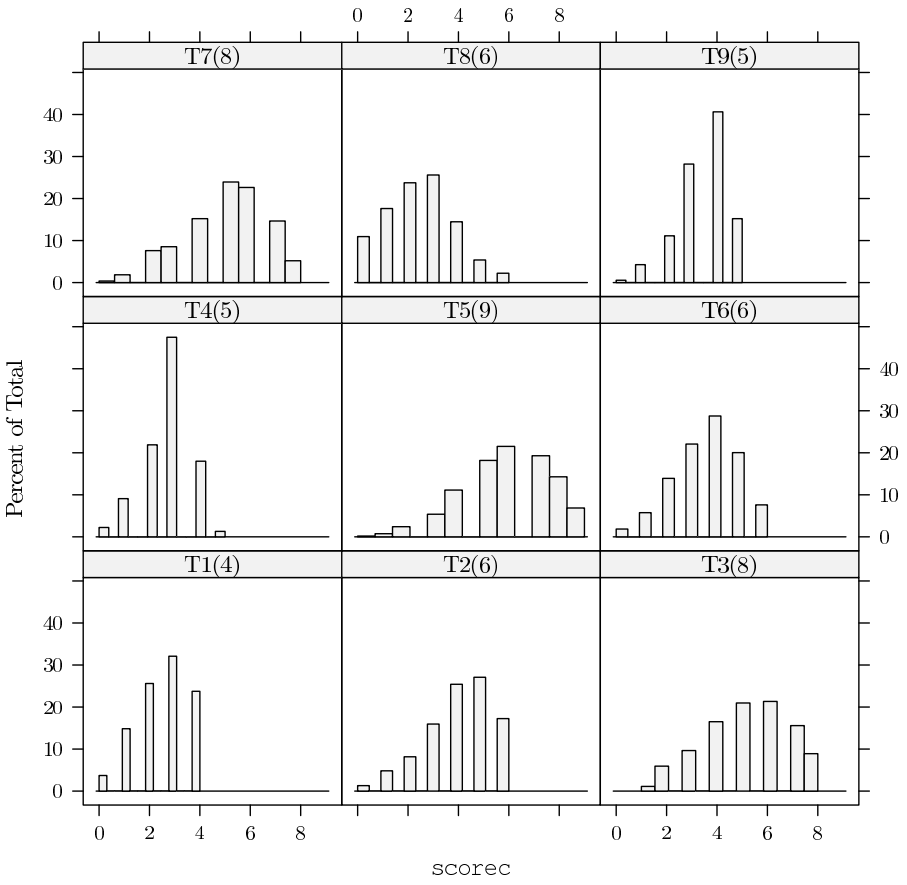


Fig. 3.11: Histograms of individual total scores for different attainment targets.

==== Fig.3.11 about here =====
(EPS: Figs/FigatotHist)===)

3.6 Chapter summary

In this chapter, we presented exploratory analyses of the four case studies introduced in Chap. 2. The results of the analyses will be used in the next parts of our book to build models for the case studies.

In parallel to the presentation of the results of the exploratory analyses, we introduced a range of R tools, which are useful for such analyses. For instance, functions `cast()` and `melt()` from the package **reshape** are very useful in transforming data involving aggregated summaries. The importance of using graphical displays is also worth highlighting. To this aim, the tools available in packages **graphics** (R Development Core Team, 2010) and **lattice** (Sarkar, 2008) are very helpful. The former package implements traditional graphical displays, whereas the latter offers displays based on a grid-graphics system (Murrell, 2005).

Due to space limitations, our presentation of the tools was neither exhaustive nor detailed. However, we hope that the syntax and its short description, which were provided in the chapter, can help the reader in finding appropriate methods applicable to a particular problem at hand.