Chapter 1

Introduction: Principles and Concepts

This chapter presents some of the concepts and principles that recur throughout the book. We begin with the two guiding principles: the mission to explore and the responsibility to be trustworthy (Sections 1.1 and 1.2). With these as guidelines, we then introduce some concepts for programming with R (Section 1.3, page 4) and add some justification for our emphasis on that system (Section 1.4, page 9).

1.1 Exploration: The Mission

The first principle I propose is that our *Mission*, as users and creators of software for data analysis, is to enable the best and most thorough exploration of data possible. That means that users of the software must be ale to ask the meaningful questions about their applications, quickly and flexibly.

Notice that speed here is human speed, measured in clock time. It's the time that the actual computations take, but usually more importantly, it's also the time required to formulate the question and to organize the data in a way to answer it. This is the exploration, and software for data analysis makes it possible. A wide range of techniques is needed to access and transform data, to make predictions or summaries, to communicate results to others, and to deal with ongoing processes.

Whenever we consider techniques for these and other requirements in the chapters that follow, the first principle we will try to apply is the *Mission*:

How can these techniques help people to carry out this specific kind of exploration?

Ensuring that software for data analysis exists for such purposes is an important, exciting, and challenging activity. Later chapters examine how we can select and develop software using R and other systems.

The importance, excitement, and challenge all come from the central role that data and computing have come to play in modern society. Science, business and many other areas of society continually rely on understanding data, and that understanding frequently involves large and complicated data processes.

A few examples current as the book is written can suggest the flavor:

- Many ambitious projects are underway or proposed to deploy *sensor networks*, that is, coordinated networks of devices to record a variety of measurements in an ongoing program. The data resulting is essential to understand environmental quality, the mechanisms of weather and climate, and the future of biodiversity in the earth's ecosystems. In both scale and diversity, the challenge is unprecedented, and will require merging techniques from many disciplines.

- Astronomy and cosmology are undergoing profound changes as a result of large-scale digital mappings enabled by both satellite and ground recording of huge quantities of data. The scale of data collected allows questions to be addressed in an overall sense that before could only be examined in a few, local regions.

- Much business activity is now carried out largely through distributed, computerized processes that both generate large and complex streams of data and also offer through such data an unprecedented opportunity to understand one's business quantitatively. Telecommunications in North America, for example, generates databases with conceptually billions of records. To explore and understand such data has great attraction for the business (and for society), but is enormously challenging.

These and many other possible examples illustrate the importance of what John Tukey long ago characterized as "the peaceful collision of computing and data analysis". Progress on any of these examples will require the ability to explore the data, flexibly and in a reasonable time frame.

1.2 Trustworthy Software: The Prime Directive

Exploration is our mission; we and those who use our software want to find new paths to understand the data and the underlying processes. The mission is, indeed, to boldly go where no one has gone before. But, we need boldness to be balanced by our responsibility. We have a responsibility for the results of data analysis that provides a key compensating principle.

The complexity of the data processes and of the computations applied to them mean that those who receive the results of modern data analysis have limited opportunity to verify the results by direct observation. Users of the analysis have no option but to trust the analysis, and by extension the software that produced it. Both the data analyst and the software provider therefore have a strong responsibility to produce a result that is trustworthy, and, if possible, one that can be *shown* to be trustworthy.

This is the second principle: the computations and the software for data analysis should be trustworthy: they should do what they claim, and be seen to do so. Neither those who view the results of data analysis nor, in many cases, the statisticians performing the analysis can directly validate extensive computations on large and complicated data processes. Ironically, the steadily increasing computer power applied to data analysis often distances the results further from direct checking by the recipient. The many computational steps between original data source and displayed results must all be truthful, or the effect of the analysis may be worthless, if not pernicious. This places an obligation on all creators of software to program in such a way that the computations can be understood and trusted. This obligation I label the *Prime Directive*.

Note that the directive in no sense discourages exploratory or approximate methods. As John Tukey often remarked, better an approximate answer to the right question than an exact answer to the wrong question. We should seek answers boldly, but always explaining the nature of the method applied, in an open and understandable format, supported by as much evidence of its quality as can be produced. As we will see, a number of more technically specific choices can help us satisfy this obligation.

Readers who have seen the *Star Trek*® television series[1] may recognize the term "prime directive". Captains Kirk, Picard, and Janeway and their crews were bound by a directive which (slightly paraphrased) was: Do nothing to interfere with the natural course of a new civilization. Do not distort

[1] Actually, at least five series, from "The Original" in 1966 through "Enterprise", not counting the animated version, plus many films. See `startrek.com` and the many reruns if this is a gap in your cultural background.

the development. Our directive is not to distort the message of the data, and to provide computations whose content can be trusted and understood.

The prime directive of the space explorers, notice, was not their *mission* but rather an important safeguard to apply in pursuing that mission. Their mission was to explore, to "boldly go where no one has gone before", and all that. That's really our mission too: to explore how software can add new abilities for data analysis. And our own prime directive, likewise, is an important caution and guiding principle as we create the software to support our mission.

Here, then, are two motivating principles: the mission, which is bold exploration; and the prime directive, trustworthy software. We will examine in the rest of the book how to select and program software for data analysis, with these principles as guides. A few aspects of R will prove to be especially relevant; let's examine those next.

1.3 Concepts for Programming with R

The software and the programming techniques to be discussed in later chapters tend to share some concepts that make them helpful for data analysis. Exploiting these concepts will often benefit both the effectiveness of programming and the quality of the results. Each of the concepts arises naturally in later chapters, but it's worth outlining them together here for an overall picture of our strategy in programming for data analysis.

Functional Programming

Software in R is written in a *functional style* that helps both to understand the intent and to ensure that the implementation corresponds to that intent. Computations are organized around functions, which can encapsulate specific, meaningful computational results, with implementations that can be examined for their correctness. The style derives from a more formal theory of *functional programming* that restricts the computations to obtain well-defined or even formally verifiable results. Clearly, programming in a fully functional manner would contribute to trustworthy software. The S language does not enforce a strict functional programming approach, but does carry over some of the flavor, particularly when you make some effort to emphasize simple functional definitions with minimal use of non-functional computations.

As the scope of the software expands, much of the benefit from functional style can be retained by using *functional methods* to deal with varied types

of data, within the general goal defined by the generic function.

Classes and Methods

The natural complement to functional style in programming is the definition of classes of objects. Where functions should clearly encapsulate the actions in our analysis, classes should encapsulate the nature of the objects used and returned by calls to functions. The duality between function calls and objects is a recurrent theme of programming with R. In the design of new classes, we seek to capture an underlying concept of what the objects mean. The relevant techniques combine directly specifying the contents (the slots), relating the new class to existing classes (the inheritance), and expressing how objects should be created and validated (methods for initializing and validating).

Method definitions knit together functions and classes. Well-designed methods extend the generic definition of what a function does to provide a specific computational method when the argument or arguments come from specified classes, or inherit from those classes. In contrast to methods that are solely class-based, as in common object-oriented programming languages such as C++ or Java, methods in R are part of a rich but complex network of functional and object-based computation.

The ability to define classes and methods in fact is itself a major advantage in adhering to the *Prime Directive*. It gives us a way to isolate and define formally what information certain objects should contain and how those objects should behave when functions are applied to them.

Data Frames

Trustworthy data analysis depends first on trust in the data being analyzed. Not so much that the data must be perfect, which is impossible in nearly any application and in any case beyond our control, but rather that trust in the analysis depends on trust in the relation between the data as we use it and the data as it has entered the process and then has been recorded, organized and transformed.

In serious modern applications, the data usually comes from a process external to the analysis, whether generated by scientific observations, commercial transactions or any of many other human activities. To access the data for analysis by well-defined and trustworthy computations, we will benefit from having a description, or model, for the data that corresponds to its natural home (often in DBMS or spreadsheet software), but can also be

a meaningful basis for data as used in the analysis. Transformations and restructuring will often be needed, but these should be understandable and defensible.

The model we will emphasize is the *data frame*, essentially a formulation of the traditional view of observations and variables. The data frame has a long history in the S language but modern techniques for classes and methods allow us to extend the use of the concept. Particularly useful techniques arise from using the data frame concept both within R, for model-fitting, data visualization, and other computations, and also for effective communication with other systems. Spreadsheets and relational database software both relate naturally to this model; by using it along with unambiguous mechanisms for interfacing with such software, the meaning and structure of the data can be preserved. Not all applications suit this approach by any means, but the general data frame model provides a valuable basis for trustworthy organization and treatment of many sources of data.

Open Source Software

Turning to the general characteristics of the languages and systems available, note that many of those discussed in this book are *open-source* software systems; for example, R, Perl, Python, many of the database systems, and the Linux operating system. These systems all provide access to source code sufficient to generate a working version of the software. The arrangement is not equivalent to "public-domain" software, by which people usually mean essentially unrestricted use and copying. Instead, most open-source systems come with a copyright, usually held by a related group or foundation, and with a license restricting the use and modification of the software. There are several versions of license, the best known being the Gnu Public License and its variants (see `gnu.org/copyleft/gpl.html`), the famous GPL. R is distributed under a version of this license (see the `"COPYING"` file in the home directory of R). A variety of other licenses exists; those accepted by the *Open Source Initiative* are described at `opensource.org/licenses`.

Distinctions among open-source licenses generate a good deal of heat in some discussions, often centered on what effect the license has on the usability of the software for commercial purposes. For our focus, particularly for the concern with trustworthy software for data analysis, these issues are not directly relevant. The popularity of open-source systems certainly owes a lot to their being thought of as "free", but for our goal of trustworthy software, this is also not the essential property. Two other characteristics contribute more. First, the simple openness itself allows any sufficiently

competent observer to enquire fully about what is actually being computed. There are no intrinsic limitations to the validation of the software, in the sense that it is all there. Admittedly, only a minority of users are likely to delve very far into the details of the software, but some do. The ability to examine and critique every part of the software makes for an open-ended scope for verifying the results.

Second, open-source systems demonstrably generate a spirit of community among contributors and active users. User groups, e-mail lists, chat rooms and other socializing mechanisms abound, with vigorous discussion and controversy, but also with a great deal of effort devoted to testing and extension of the systems. The active and demanding community is a key to trustworthy software, as well as to making useful tools readily available.

Algorithms and Interfaces

R is explicitly seen as built on a set of routines accessed by an interface, in particular by making use of computations in C or Fortran. User-written extensions can make use of such interfaces, but the core of R is itself built on them as well. Aside from routines that implement R-dependent techniques, there are many basic computations for numerical results, data manipulation, simulation, and other specific computational tasks. These implementations we can term *algorithms*. Many of the core computations on which the R software depends are now implemented by collections of such software that are widely used and tested. The algorithm collections have a long history, often predating the larger-scale open-source systems. It's an important concept in programming with R to seek out such algorithms and make them part of a new computation. You should be able to import the trust built up in the non-R implementation to make your own software more trustworthy.

Major collections on a large scale and many smaller, specialized algorithms have been written, generally in the form of subroutines in Fortran, C, and a few other general programming languages. Thirty-plus years ago, when I was writing *Computational Methods for Data Analysis*, those who wanted to do innovative data analysis often had to work directly from such routines for numerical computations or simulation, among other topics. That book expected readers to search out the routines and install them in the readers' own computing environment, with many details left unspecified.

An important and perhaps under-appreciated contribution of R and other systems has been to embed high-quality algorithms for many computations in the system itself, automatically available to users. For example, key parts of the LAPACK collection of computations for numerical linear algebra

are included in R, providing a basis for fitting linear models and for other matrix computations. Other routines in the collection may not be included, perhaps because they apply to special datatypes or computations not often encountered. These routines can still be used with R in nearly all cases, by writing an interface to the routine (see Chapter 11).

Similarly, the internal code for pseudo-random number generation includes most of the well-regarded and thoroughly tested algorithms for this purpose. Other tasks, such as sorting and searching, also use quality algorithms. Open-source systems provide an advantage when incorporating such algorithms, because alert users can examine in detail the support for computations. In the case of R, users do indeed question and debate the behavior of the system, sometimes at great length, but overall to the benefit of our trust in programming with R.

The best of the algorithm collections offer another important boost for trustworthy software in that the software may have been used in a wide variety of applications, including some where quality of results is critically important. Collections such as LAPACK are among the best-tested substantial software projects in existence, and not only by users of higher-level systems. Their adaptability to a wide range of situations is also a frequent benefit.

The process of incorporating quality algorithms in a user-oriented system such as R is ongoing. Users can and should seek out the best computations for their needs, and endeavor to make these available for their own use and, through packages, for others as well.

Incorporating algorithms in the sense of subroutines in C or Fortran is a special case of what we call *inter-system interfaces* in this book. The general concept is similar to that for algorithms. Many excellent software systems exist for a variety of purposes, including text-manipulation, spreadsheets, database management, and many others. Our approach to software for data analysis emphasizes R as the central system, for reasons outlined in the next section. In any case, most users will prefer to have a single home system for their data analysis.

That does not mean that we should or can absorb all computations directly into R. This book emphasizes the value of expressing computations in a natural way while making use of high-quality implementations in whatever system is suitable. A variety of techniques, explored in Chapter 12, allows us to retain a consistent approach in programming with R at the same time.

1.4 The R System and the S Language

This book includes computations in a variety of languages and systems, for tasks ranging from database management to text processing. Not all systems receive equal treatment, however. The central activity is data analysis, and the discussion is from the perspective that our data analysis is mainly expressed in R; when we examine computations, the results are seen from an interactive session with R. This view does not preclude computations done partly or entirely in other systems, and these computations may be complete in themselves. The data analysis that the software serves, however, is nearly always considered to be in R.

Chapter 2 covers the use of R broadly but briefly (if you have no experience with it, you might want to consult one of the introductory books or other sources mentioned on page vii in the preface). The present section give a brief summary of the system and relates it to the philosophy of the book.

R is an open-source software system, supported by a group of volunteers from many countries. The central control is in the hands of a group called R-core, with the active collaboration of a much larger group of contributors. The base system provides an interactive language for numerical computations, data management, graphics and a variety of related calculations. It can be installed on Windows, Mac OS X, and Linux operating systems, with a variety of graphical user interfaces. Most importantly, the base system is supported by well over a thousand packages on the central repository `cran.r-project.org` and in other collections.

R began as a research project of Ross Ihaka and Robert Gentleman in the 1990s, described in a paper in 1996 [17]. It has since expanded into software used to implement and communicate most new statistical techniques. The software in R implements a version of the S language, which was designed much earlier by a group of us at Bell Laboratories, described in a series of books ([1], [6], and [5] in the bibliography).

The S-Plus system also implements the S language. Many of the computations discussed in the book work in S-Plus as well, although there are important differences in the evaluation model, noted in later chapters. For more on the history of S, see Appendix A, page 475.

The majority of the software in R is itself written in the same language used for interacting with the system, a dialect of the S language. The language evolved in essentially its present form during the 1980s, with a generally functional style, in the sense used on page 4: The basic unit of programming is a function. Function calls usually compute an object that is a

function of the objects passed in as arguments, without side effects to those arguments. Subsequent evolution of the language introduced formal classes and methods, again in the sense discussed in the previous section. Methods are specializations of functions according to the class of one or more of the arguments. Classes define the content of objects, both directly and through inheritance. R has added a number of features to the language, while remaining largely compatible with S. All these topics are discussed in the present book, particularly in Chapters 3 for functions and basic programming, 9 for classes, and 10 for methods.

So why concentrate on R? Clearly, and not at all coincidentally, R reflects the same philosophy that evolved through the S language and the approach to data analysis at Bell Labs, and which largely led me to the concepts I'm proposing in this book. It is relevant that S began as a medium for statistics researchers to express their own computations, in support of research into data analysis and its applications. A direct connection leads from there to the large community that now uses R similarly to implement new ideas in statistics, resulting in the huge resource of R packages.

Added to the characteristics of the language is R's open-source nature, exposing the system to continual scrutiny by users. It includes some algorithms for numerical computations and simulation that likewise reflect modern, open-source computational standards in these fields. The LAPACK software for numerical linear algebra is an example, providing trustworthy computations to support statistical methods that depend on linear algebra.

Although there is plenty of room for improvement and for new ideas, I believe R currently represents the best medium for quality software in support of data analysis, and for the implementation of the principles espoused in the present book. From the perspective of our first development of S some thirty-plus years ago, it's a cause for much gratitude and not a little amazement.