Latent Variable Modeling Using R

A Step-by-Step Guide

A. Alexander Beaujean

ROUTLEDGE

Latent Variable Modeling Using R

This step-by-step guide is written for **R** and latent variable model (LVM) novices. Utilizing a path model approach and focusing on the *lavaan* package, this book is designed to help readers quickly understand LVMs and their analysis in **R**. The author reviews the reasoning behind the syntax selected and provides examples that demonstrate how to analyze data for a variety of LVMs. Featuring examples applicable to psychology, education, business, and other social and health sciences, minimal text is devoted to theoretical underpinnings. The material is presented without the use of matrix algebra. As a whole the book prepares readers to write about and interpret LVM results they obtain in **R**.

Each chapter features background information, boldfaced key terms defined in the glossary, detailed interpretations of **R** output, descriptions of how to write the analysis of results for publication, a summary, **R** based practice exercises (with solutions included in the back of the book), and references and related readings. Margin notes help readers better understand LVMs and write their own **R** syntax. Examples using data from published work across a variety of disciplines demonstrate how to use **R** syntax for analyzing and interpreting results. **R** functions, syntax, and the corresponding results appear in gray boxes to help readers quickly locate this material. A unique index helps readers quickly locate **R** functions, packages, and datasets. The book and accompanying website (`http://blogs.baylor.edu/rlatentvariable`) provide all of the data for the book's examples and exercises as well as **R** syntax so readers can replicate the analyses. The book reviews how to enter the data into **R**, specify the LVMs, and obtain and interpret the estimated parameter values.

Intended as a supplementary text for graduate and/or advanced undergraduate courses on latent variable modeling, factor analysis, structural equation modeling, item response theory, measurement, or multivariate statistics taught in psychology, education, human development, business, economics, and social and health sciences, this book also appeals to researchers in these fields. Prerequisites include familiarity with basic statistical concepts, but knowledge of **R** is *not* assumed.

A. Alexander Beaujean is an Associate Professor in Educational Psychology at Baylor University.

This page intentionally left blank

Latent Variable Modeling Using R

A Step-by-Step Guide

A. Alexander Beaujean

Routledge
Taylor & Francis Group
NEW YORK AND LONDON

First published 2014
by Routledge
711 Third Avenue, New York, NY 10017

and by Routledge
27 Church Road, Hove, East Sussex BN3 2FA

Routledge is an imprint of the Taylor & Francis Group, an informa business

Library of Congress Cataloging in Publication Data
A catalog record has been requested.

ISBN: 978-1-84872-698-7 (hbk)
ISBN: 978-1-84872-699-4 (pbk)
ISBN: 978-1-315-86978-0 (ebk)

Typeset in Latin Modern Roman
by A. Alexander Beaujean

Contents

Author Biography

A. Alexander Beaujean received PhDs in School Psychology and Educational Psychology from the University of Missouri. His research interests are in individual differences, especially their measurement and influence on life outcomes. He is currently an associate professor at Baylor University in the Educational Psychology Department, where he teaches courses on psychological assessment, educational and psychological measurement, and multiple regression. His scholarship has won awards from the American Academy of Health Behavior, American Psychological Association, Mensa, and the Society for Applied Multivariate Research.

Preface

The use of latent variable models has seen a tremendous amount of growth in the past 30 years across a variety of academic disciplines, including the sciences, clinical professions, business, and even the humanities. Part of the reason for this growth is the increasing availability of software to estimate these models' parameters. Traditionally, most of this software has either been too expensive or too complicated for anyone without access to the resources of a large business or university. This trend is rapidly changing, however, and there are now free programs that can conduct a latent variable analysis with only a modicum of knowledge about statistical programming.

This book is designed to introduce **R**, a free statistical program, and show how to use it for latent variable modeling. Thus, the book's two aims are to help readers:

1. understand the basics of the **R** language, and
2. use **R** to analyze a variety of useful latent variable models.

To achieve these aims, this book has some distinctive features that I highlight below.

Path Model Approach to Latent Variable Modeling

Based on teaching graduate students in education, psychology, and related disciplines, I have found that using path models tends to be an effective way to help the novice learn about latent variable models. Consequently, after introducing the **R** program in Chapter 1, I then introduce path models in Chapter 2 and continue to use these models throughout the book. While relying only on path models comes at the price of excluding their matrix representations, it comes with the benefit of increasing the readers' facility of using a model-based approach to translate their research hypotheses into data analysis–an important tool for both students and professionals.

Because of my emphasis on path models throughout the book, I mostly use the **R** package `lavaan` (and packages that work with `lavaan`) to fit the latent variable models. I purposefully did this as `lavaan` uses a path model approach to specify latent variable models. Thus, the chapter text and the **R** syntax complement each other.

Real World Perspective

Having worked with scholars from many disciplines, I know that data are not always well behaved and the syntax to analyze such data are not always easy to find. Consequently, the majority of the examples I use in this text come from published work that represent real data scholars have analyzed. This data comes from a variety of disciplines including education, medicine, psychology, and sociology.

Modern Methods

Because **R** is open-source software, it is continually being updated and improved. Thus, it can use modern techniques to analyze data. While I incorporate this modernity throughout

the book, it is particularly highlighted in the last four chapters as they contain topics that are not readily available from some other latent variable programs. For example, in Chapter 7 I discuss missing data, and demonstrate methods to determine missing data patterns as well as modern methods of handling missing data–including the use of auxiliary variables. Likewise, in Chapter 8 I demonstrate how to use Monte Carlo methods to determine the sample size needed for a prospective study.

Intended Audience

This book can be used as a supplementary text alongside a more theoretical textbook in graduate courses on latent variable modeling. In addition, this book can also be used as a supplementary text in graduate or advanced undergraduate courses that survey latent variable models or courses that review LVMs such as item response theory, measurement, or multivariate statistics taught in a variety of disciplines such as psychology, education, human development, business, economics, and other social and health sciences. Third, professionals and researchers already using latent variable models, but unfamiliar with **R**, will find this book a useful tool for learning some important features of the **R** language.

I used examples from a variety of disciplines to make the context accessible to readers from many different backgrounds, such as business, economics, education, health sciences, human development, psychology, and social science. As the only prerequisite for the text is some familiarity with statistical concepts, both **R** novices and experts should find the text accessible.

Learning Tools

There are some key features in this text to help readers use its material.

Chapter Structure

Every chapter except the first follows the same structure. They all start with some background information, then I work through one or two examples in step-by-step detail, explicitly showing **R** syntax needed for the analyses and interpreting the output. I end each chapter describing how to write the results from that chapter's content for use in a report or publication, as well as providing practice exercises and references/suggested readings. Some of the exercises follow directly from the in-text examples, while others are designed to extend the chapter's content. Most of the exercises require only the use of sample statistics to fit the latent variable model, which I provide in the book. For the exercises that require raw data, I have the files on the book's website at `http://blogs.baylor.edu/rlatentvariable`.

Glossary and Indexes

At the end of the book there are two reader-centered items. The first is a glossary of terms that are likely new and unfamiliar to the latent variable modeling novice. The second are the indices. In addition to the author and subject indices, I also placed three **R** indexes. The first one contains **R** functions, while the second and third contain **R** packages and datasets, respectively. I separated these out purposefully so that the readers do not have to scour the entire index if they forget a **R** function, package, or dataset name.

Text Formatting

This is a hint!

- In the margins I periodically place hints, suggestions, and information that I have found useful. These notes are designed to help readers as they write the **R** syntax for their own models as well as understand some of the complexities involved with latent variable models.

Term

- Every time I introduce a key term, I use **boldface** and place the term in the margin. This should help readers find the areas of interest quickly when they use the book to create their own latent variable models. These terms are then defined in the end of text glossary.

`example.function()`

- Every time I discuss a **R** function or package, I use a `truetype font`. I attach parentheses to the **R** functions [e.g., `example.function()`], and place the name in the margin anytime I introduce a new function or go into substantial detail about it. This will help readers find the these functions quickly when using the book to write their own **R** syntax and analyze their own data.
- I placed all my **R** syntax in a gray box on the page, with resulting output given in the same gray box with two pound symbols ## on the left.

```
R syntax

##  Results
```

Book Contents

In Chapter 1, I introduce the **R** program, and discuss how to acquire it, input/import data, and execute some simple functions. The subsequent chapters follow a sequence found in many latent variable textbooks. Chapter 2 introduces path models, while Chapter 3 extends the path models to include latent variables. In Chapter 4 I discuss how to analyze a latent variable model with data from more than one group (including twin data), while in Chapter 5 I discuss how to analyze a latent variable model with data from more than one time period.

The last four chapters are unique for an applied latent variable modeling book. In Chapter 6, I discuss how to handle dichotomous variables, using both the traditional latent variable model perspective as well as an item response theory (IRT) perspective. Further, using a worked example, I show to convert the results from one type of analysis to the other. I devote the entirety of Chapter 7 to fitting a latent variable model with missing data. I discuss types of missing data, methods to determine missing data patterns, and modern methods of handling missing data–including the use of auxiliary variables.

In Chapter 8 I demonstrate how to determine a study's sample size using Monte Carlo simulation. This is not the typical method most textbooks discuss concerning sample size planning, but I chose to focus on this method as it can be used with a wide range of statistical models as well as account for missing data. In the last chapter, Chapter 9, I focus on latent variable models with different levels (i.e., hierarchical models). I include fitting both higher-order models as well as bi-factor models.

After the last chapter, I placed three appendices. Appendix A is about measures of model fit. I do not emphasize the use of any particular model fit index in the book, but in this ap-

pendix I present a variety of common fit indices, including their formulae and interpretation. The second appendix covers a different area. Throughout this book, I mostly use the `lavaan` package. There are other **R** packages that will fit latent variable models, but it has been my experience that it is confusing to learn multiple programs concurrently, as there is a tendency to mix the syntax. Thus, in Appendix B, I provide syntax for other **R** latent variable models packages for readers wishing know how they compare to `lavaan`. Appendix C contains answers (mostly **R** syntax) for each chapter's exercises, although I do suggest trying the exercises yourself before looking at the answers!

While I included as much content as I could, due to space considerations I had to exclude two au courant areas in latent variable modeling. The first area concerns models with a categorical latent variable (i.e., latent class, latent profile). There are **R** packages available for their estimation (e.g., `poLCA`, `mclust`) and the interested reader should read their documentation for more information. The second area is Bayesian estimation. With the integration of winBUGS and JAGS with **R** (e.g., `R2WinBUGS`, `R2jags`), Bayesian estimation of latent variable model is more accessible to **R** users than ever before. Using Bayesian estimation, however, requires much more information about the process of parameter estimation than I provide in this text.

Website

There is a companion website for this book at `http://blogs.baylor.edu/rlatentvariable`. It includes raw data files, **R** syntax for the book examples in a copy-and-paste format, links to related websites with helpful information about **R** and latent variable models, as well as supplemental chapters on creating latent variable model diagrams, LISREL notation, and bootstrapping.

Acknowledgments

I am indebted to many individuals for their help with this book. In particular, I want to thank the individuals who have provided feedback on previous drafts of this text: Danielle Fearon (Baylor University), Darrell Hull (University of North Texas), Grant Morgan (Baylor University), Sonia Parker (Baylor University), Terrill Saxon (Baylor University), Yanyan Sheng (Southern Illinois University-Carbondale), Kara Styck (University of Texas-San Antonio), Phil Wood (University of Missouri), as well as all the students in my latent variable and multiple regression courses.

I also wish to thank the people at Routledge/Taylor & Francis, especially Senior Editor Debra Riegert. While I am responsible for any errors remaining in the text, the book is much better as a result of their input.

I wish to thank Yves Rosseel and Sunthud Pornprasertmanit for answering my questions about their **R** packages, and Mori Jamshidian for the advanced material concerning the `MissMech` package. In addition, thanks to the Law School Admissions Council for allowing me to use some example *Figure Classification* items in the text, and to Craig Enders for allowing the use of his *Eating Attitudes Test* data.

Finally, I owe much to my family: Christine, Susana, and Byron Limbers for their help and support while I wrote the book, Susanna and Aleisa for being my little co-authors, and William and Lela Beaujean for their support that allowed me to learn about latent variable models in the first place.

A. Alexander Beaujean
Waco, Texas

1 | Introduction to R

Chapter Contents

1.1 Background

R is an open-source statistical software programming language and environment for statistical computing. It is currently maintained by the **R** Development Core Team (an international team of volunteer developers), and the **R** web page (also known as Comprehensive **R** Archive Network [CRAN]) is `http://www.r-project.org`. This is the main site for **R** information and obtaining the software.

Since **R** is syntax-based, as opposed to using a point-and-click interface, it may appear too complex for a non-specialist, but this really is not the case. Using syntax allows **R** a level of ease and flexibility not available with other programs. Take, for example, the process of analyzing a multiple regression model. While point-and-click type software can provide quick results for a single analysis, to analyze different models (e.g., using different predictor sets) or use the information from the regression for another analysis (e.g, make a scatterplot with a line of best fit, check model assumptions), it often takes many point-and-click iterations to produce the desired results. Moreover, if you have to stop your analysis and return to it days or weeks later, it can be hard to remember what you previously accomplished with the analysis or even the point-and-click sequences used to obtain the previous results. With **R**, though, many of these problems are not an issue. As **R** can store the results from the regression into *objects*, you can specify the parts of the regression results that need to be extracted for subsequent analysis. Furthermore, you can analyze multiple models and have **R** display their coefficients in a single window instead of opening many results windows, as many point-and-click programs would produce. Because these multiple models were analyzed

```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Figure 1.1 Typical on-screen text when starting **R**.

using syntax, if you save the syntax in an external file, then you can return to the analysis months later and exactly reproduce the previous results by simply pasting the syntax back into **R**.

1.1.1 Installing R

R can be run under Windows, Mac, and Unix-type operating systems. To download **R**, go to `http://www.r-project.org/` and select the *CRAN* hyperlink. This opens a list of places (mirrors) from which to download the program. Select a hyperlink from a mirror in your country, which loads a page with hyperlinks to download **R** for your operating system (select the precompiled binary distribution).

There are some graphical user interfaces (GUIs) for **R** developed by third parties. A partial list can be found at **R** Wiki (`http://rwiki.sciviews.org/doku.php?id=guis:projects`) and CRAN (`http://www.r-project.org/GUI`). There are also many text editors that are either designed to interact with **R**, or can be modified to do so. Typing R TEXT EDITOR (or something similar) into an Internet search engine will bring up many different options as well as people's opinions about them.

1.1.2 Starting R

When initially starting **R** in interactive mode (as opposed to batch mode), the screen looks something like Figure 1.1. The `>` symbol is called the prompt. It is not typed; instead, it is used to indicate where to type. When writing syntax in **R** directly, type in all commands at the `>` prompt. If a command is too long to fit on a single line, a `+` is used for the continuation prompt.

If you type `>`, **R** interprets it as "greater than."

1.1.3 Functions

R stores variables, data, functions, results, etc, in the computer's active memory in the form of named *objects*. The user can then do actions on these objects with *operators* (arithmetic, logical, comparison) and *functions* (which are themselves objects). Much of **R**'s functionality comes from applying functions to data or other objects. **R** functions are a set of instructions that take input, compute the desired value(s), and return the result. **R** comes pre-loaded with a set of commonly used functions, but there are many additional ones to add by loading

packages with the desired functions, or by writing a function. To use functions: (a) give the function's name followed by parentheses; (b) in the parentheses, give the necessary values for the function's argument(s).

1.1.3.1 Some Useful Functions

Below are helpful **R** functions that I find myself using repeatedly.

- *Comment.* This is not really a function, but in **R** anything after the `#` sign is assumed to be a comment and **R** ignores it. Comments are extremely helpful, as annotating **R** syntax can save a lot of future time and effort. `# (Comment)`
- *Assign.* Another symbol that most **R** users will encounter frequently is the left arrow, `<-`, which is **R**'s standard assignment operator (another option is using `=`, but it is better to reserve using `=` for defining values for arguments). The `<-` is **R**'s way of assigning whatever is on the right of the arrow to the object on the left of the arrow. `<- (Assign)`
- *Concatenate.* The *concatenate* function, `c()`, concatenates the arguments included in the function. Using `c()` in conjunction with `<-` assigns the concatenated objects into a new object. For example, to make a dataset of 5 observations with the values 4, 5 3, 6, 9, and name it `newData`, I would use the following syntax: `c()`

```
newData <- c(4, 5, 3, 6, 9)
```

- *Help.* The `help()` function returns information about a function (or certain special words or characters). A shortcut for `help()` is a question mark, `?`. For example, the following two lines of syntax return the same results. `help()` `?`

```
help(mean)
?mean
```

The `help()` function returns a page that (at a minimum) describes the function, its arguments, and gives some examples of how to use it. Some help pages have much more detail than others. To just execute the example syntax for a function, use the `example()` function. `example()`

```
example(mean)


##
## mean> x <- c(0:10, 50)
##
## mean> xm <- mean(x)
##
## mean> c(xm, mean(x, trim = 0.10))
## [1] 8.8 5.5
```

To obtain help on an entire **R** package, use the `package` argument in the `help()` function.

```
help(package = psych)
```

If you do not know exactly what you need help with in **R**, search through **R**'s documentation using the `help.search()` function. The function's argument needs to be enclosed in quotation marks. For example, if I was interested in testing to see if a variable follows a normal distribution, I could type: `help.search()`

Topic	Package	Description
jarque.test	moments	Jarque-Bera test for normality

Figure 1.2 Example output from `help.search()` function. The results from this output indicate that in the `moments` package there is a function called `jarque.test()` that performs the Jarque-Bera test for normality.

```
help.search("normality")
```

The resulting output contains functions from packages that might be of interest, such as shown in Figure 1.2.

Another useful way to get help is to use the Rseek website (`http://www.rseek.org/`), which is a site that uses Google to help find **R** functions, lists, syntax, etc.

`help.start()`

If you find yourself totally lost on where to start asking for help, then type `help.start()` into **R**. The resulting output consists of many important documents useful for navigating **R**, as well as provides another search engine (*Search Engine & Keywords*) for **R** help materials.

1.1.3.2 Writing a Function

In **R**, if a function is not available to do the desired analysis or data manipulation, there is an option to write a new function using the `function()` function. The following syntax is an example of a function I wrote to calculate the arithmetic mean, called `ArithMean()`.

`function()`

```
1 # Function to calculate the arithmetic mean
2 ArithMean <- function(x) {
3 Sx <- sum(x)
4 Mean<- Sx/length(x)
5 return(Mean)
6 }
7 example.data <- c(5,10,15)
8 ArithMean(example.data)
```

First, I told **R** that I wanted to define the function named `ArithMean()`, which only takes one argument, `x` (see line 2). The left brace, `{`, indicates where the text of the function is going to start and the right brace, `}`, indicates where the text of the function is going to end. After defining the function, I evaluated one call to it (line 8). Since the sum of the numbers in the vector `example.data` is 30 and the length of the vector (i.e., the number of elements) is 3, the call to the function returned the value 10.

In the `ArithMean()` function, `x` is the *formal argument*, whereas in the call to function, `example.data`, is the *actual argument*. The formal argument is a placeholder, but `example.data` is the value used in the computation. Sometimes **R** functions have default arguments, which are values that a function's argument(s) automatically initialize unless you specify a different value.

1.1.4 Packages

Using packages is a vital component to using **R**. With the initial download, **R** includes some base packages that provide the backbone functions of many statistical analysis, such as `mean()` and `var()`. These functions, however, may not do a particular analysis of interest.

`mean()`
`var()`

Thus, I can see if a contributed **R** package has a function for the needed analysis. These **R** packages usually consist of functions and example data that were written in the **R** language (although sometimes they are written in FORTRAN or C and then linked back into **R**).

The ability for users to contribute packages is extremely powerful, as there are many experts across a variety of fields who have contributed packages that contain functions to compute almost any statistical analysis. A list of **R** packages, along with a short description of what they do, can be found in CRAN, but it is very long and hard to navigate unless looking for a specific package by name. An alternative is to examine *CRAN task views* [`http://cran.r-project.org/web/views/`], which is designed to help users find packages associated with specific types of fields or analyses. For example, the *Psychometrics* view [`http://cran.r-project.org/web/views/Psychometrics.html`] has many packages dealing with item and test analysis.

`install.packages()`

To install a package, use the (oddly enough named!) `install.packages()` function, naming the package to install in quotation marks. For example, to install the *BaylorEdPsych* package, I use the following syntax:

```
install.packages("BaylorEdPsych", dep = TRUE)
```

The `dep = TRUE` argument tells **R** that, in addition to the package of interest, I also want to download any other package upon which the package of interest is dependent. Installing the dependent packages saves the time of having to download each required package separately. Packages only need to installed on a computer's hard disk once, but need to be loaded into **R**'s memory each time **R** is restarted and there is a need to use one of the package's functions. Load an already-installed package by using the `library()` function.

The interactive version of **R** provides an alternative method of installing packages via the *Packages & Data* menu.

`library()`

```
library(BaylorEdPsych)
```

Since **R** is case sensitive, `Install.packages("BaylorEdPsych", dep = TRUE)`, `install.packages("BaylorEdpsych", dep = TRUE)`, `install.Packages("BaylorEdPsych", dep = TRUE)`, or any other permutation will result in an error message.

1.1.5 Data Input

1.1.5.1 Concatenate

The easiest way to enter data into **R** is to directly type it using the `c()` function, and then assign it to an object. To verify that the data are in the object (in this case, a vector), just type the object's name.

```
newData <- c(4, 5, 3, 6, 9)
newData


## [1] 4 5 3 6 9
```

Once the data is in an **R** object, I can then apply functions to the object, e.g.,

```
mean(newData)
sum(newData)
```

1.1.5.2 Import Data from an External Source

Unless the dataset has one variable with a few observations (e.g., data from a textbook example), it is usually better to store the data in an external file and then import into **R**. Here are three suggestions when saving data in an external file.

- Code all missing values to `NA`, which is the default indicator in **R** for a missing value.
- Make sure the variable names do not have spaces. Use a period in lieu of a space (e.g., `first.name`).
- Save the data as a plain text file, using either tabs, spaces, or commas as delimiters. Typically when data storage programs export space- or tab-delimited files they append a `.txt` extension and use a `.csv` extension for comma-delimited files. Most spreadsheet and database programs can export data in at least one of these formats.

Variable names need to start with letters.

Besides a period, variable names should not have any other non-alphanumeric characters.

You can use other extensions for data files, such as `.dat`.

After the data are properly stored in the external file, the `read.table()` function can import the externally stored data. The main argument for this function is the external file's name and location. **R** requires the use of a forward slash or double backslash to indicate a file location. For example, say I stored a dataset in a file named `data.csv`, which I stored in a folder named *name* that is further nested in a folder named *file*. To import the data using any of the following syntaxes.

`read.table()`

```
# Windows
new.data <- read.table("C:\\file\\name\\data.csv", sep = ",")
new.data <- read.csv("C:\\file\\name\\data.csv")
new.data <- read.csv("C:/file/name/data.csv")

# Mac and Unix-type systems
new.data <- read.table("/Users/first_last/file/name/data.csv", sep = ",")
new.data <- read.csv("/Users/first_last/file/name/data.csv")
```

The `sep` argument tell the function how the variables are delimited in the data file. The default option is tab delimitation, so specifying `sep=","` is needed to indicate comma delimitation. The `read.csv()` function works just like the `read.table()` function, only it assumes the data are comma-delimited.

`read.csv()`

By default, the `read.table()` function returns a *data frame*. A data frame is a type of **R** object that stores variables as columns. Data frames are useful as they can store different types of variables, such as strings and numbers.

If the location of the data file is under many sublayers of folders, (or, more typically for me, I forget its exact location), I can search for file using the `file.choose()` function.

`file.choose()`

```
new.data <- read.table(file.choose(), header = TRUE, sep = ",")
```

This opens a dialog box that allows me to choose the file interactively.

1.1.5.3 Import Other Programs' Data

R can read in data from file formats other than plain text. Table 1.1 lists some packages and functions along with the data types they read.

Table 1.1 R Packages and Functions to Import/Read Various Data Formats.

Package/Function	File types
`read.fwf()`[a]	Fixed with format
`read.DIF()`[a]	Data Interchange Format
`xlsReadWrite`	Excel (.xls)
`gdata`	Excel (.xls)
`xlsx`	Excel (.xlsx)
`RODBC`	ODBC databases, e.g., MS SQLServer, MS Access, Oracle
`RMySQL`	MySQL
`RJDBC`	JDBC compliant databases
`RSQLite`	SQLite
`foreign`	Minitab, S3, SAS, SPSS, Stata, Systat, dBase, ARFF, DBF, REC, Octave

[a] Loaded in **R** by default.

1.1.5.4 Enter a Covariance Matrix as Data

In some situations there is no access to raw data, but there is access to a covariance (or correlation) matrix. Since such matrices are symmetric, I make use of the `lavaan` package's `lower2full` function. Using the `lower2full()` function requires entering the lower diagonal of the covariance matrix *by row*. The following syntax creates a correlation matrix named *example.cor* that consists of the correlations among four variables. In addition, I name the variables using the `rownames()` and `colnames()` functions.

`lower2full()`
`rownames()`
`colnames()`

Published articles seldom give raw data, but often provide covariances. Thus, it is important to be able to use a covariance matrix as data for an analysis.

```
# load lavaan
library(lavaan)
# input covariances
example.cor <- lower2full(c(1, 0.85, 1, 0.84, 0.61, 1, 0.68, 0.59, 0.41, 1))
# name the rows and columns
rownames(example.cor) <- colnames(example.cor) <- c("Var1", "Var2", "Var3", "Var4")
example.cor


##      Var1 Var2 Var3 Var4
## Var1 1.00 0.85 0.84 0.68
## Var2 0.85 1.00 0.61 0.59
## Var3 0.84 0.61 1.00 0.41
## Var4 0.68 0.59 0.41 1.00
```

1.1.5.5 Package Datasets

Many **R** packages include datasets that are used to demonstrate their functions. The `data()` function produces a list of the datasets.

`data()`

```
# list datasets available from currently loaded packages
data()
# list datasets in all the installed packages (whether loaded or not)
data(package = .packages(all.available = TRUE))
```

Table 1.2 Common Probability Distributions Defined in **R**.

Distribution	**R** syntax
Binomial	`binom`
χ^2	`chisq`
F	`f`
Normal	`norm`
Poisson	`pois`
Student's t	`t`

To load a dataset from a specific package, use the dataset's name as an argument for the `data()` function.

```
library(BaylorEdPsych)
# load the MLBPitching2011 dataset from the BaylorEdPsych package
data(MLBPitching2011)
```

1.1.5.6 Simulate Data

In addition to importing data, **R** has the capability of simulating data from some well-known probability distributions. I have listed some of the common distributions in Table 1.2. To simulate data, use the r*dist*`()` function, where *dist* is the name of the distribution from Table 1.2. For example, the following syntax simulates 500 observations from a normal distribution with a mean of 0.0, and standard deviation of 1.0.

`rnorm()`

```
x <- rnorm(500, m=0, sd=1)
```

In Table 1.3, I give examples of other **R** functions that use probability distributions. The sequence function, `seq()`, can be very useful when simulating a probability density. It generates a sequence of numbers from a starting place to an ending place by a specified increment. I show an example of its use in the first row of Table 1.3. I discuss data simulation in more detail in Chapter 8 within the context of sample size planning.

`seq()`

1.1.6 Access a Variable Within a Dataset

Once a dataset is loaded into **R** and saved into an object, I can either do operations on the entire dataset or individual variables within the dataset. To reference a variable within a dataset, use the `$` operator.

`$`

Table 1.3 R Probability Distribution Functions.

Function	Description	Example
d*dist*	Density	`dnorm(seq(-3,3,.4))`
p*dist*	Cumulative probability	`pnorm(-1.96)`
q*dist(m)*	Quantile (value at the m percentile of the distribution)	`qnorm(0.025)`
r*dist(b)*	b random numbers from the distribution	`rnorm(100, m=0, sd=1)`

```
new.data$variable
```

An alternative to the `$` function is to attach a dataset using the `attach()` function. After attaching a dataset, I can directly access the variables within it.

`attach()`

```
attach(new.data)
variable
```

Using the `attach()` function can cause trouble, though, if the name of a dataset's variable is already being used by **R**. Specifically, if I attach two datasets to **R** and both have a variable with the same name, **R** automatically uses the variable from the most recently attached dataset. An alternative to using `attach()` is to wrap the syntax in the `with()` function. For example, say I have a dataset named *new.data*, and a variable in the dataset is named *age*. The following syntax calculates the mean of the *age* variable.

The `detach()` function detachs a dataset from **R**'s local memory.

`with()`

```
with(new.data, mean(age))
```

Some functions have an argument to specify the dataset to use, in which case there is no need to use the `with()` function or `$` notation. For example, the following syntax regresses the *age* variable on the *IQ* variable in the *new.data* dataset.[1]

```
lm(age ~ IQ, data = new.data)
```

1.1.7 Example: Entering Data and Accessing Variables

Say I have the following data stored in a tab-delimited file named `SampleData.txt`:

```
Age IQ Height Sex
18 100 65 F
21 110 68 M
45 103 65 M
54 120 69 M
```

I use the `read.table()` function to import the dataset, and then I store the imported data in an object named `example.data`. To import the data: (a) tell **R** where the data is located (within parentheses), and (b) tell **R** the name to assign the newly-created data frame. In addition, I specify that the *SampleData.txt* file has variable names (i.e., a header) using the `header=TRUE` argument.

```
example.data <- read.table(file = "SampleData.txt", header = TRUE)
```

To show (print) the data on screen, type the name of the **R** object.

```
example.data
```

```
##   Age  IQ Height Sex
## 1  18 100      6   F
## 2  21 110     68   M
## 3  45 103     65   M
## 4  54 120     69   M
```

If the dataset is large, use the `head()` function to examine the first 6 lines.

Make a habit of checking your data immediately after importing it to make sure it was read correctly.

To specify a specific variable within a dataset, use the `$` operator, the `attach()` function, or the `with()` function.

[1] The `lm()` function is the default regression function in **R**.