Zelig Documentation

Release 5.0

The Zelig Team

May 12, 2014

Contents:

CHAPTER

ONE

INTRODUCTION

1.1 What Zelig and R do

Zelig [1] is an easy-to-use program that can estimate and help interpret the results of an enormous and growing range of statistical models. It literally is "everyone's statistical software" because Zelig's unified framework incorporates everyone else's (R) code. We also hope it will become "everyone's statistical software" for applications, and we have designed Zelig so that anyone can use it or add their models to it.

When you are using Zelig, you are also using R, a powerful statistical software language. You do not need to learn R separately, however, since this manual introduces you to R through Zelig, which simplifies R and reduces the amount of programming knowledge you need to get started. Because so many individuals contribute different packages to R (each with their own syntax and documentation), estimating a statistical model can be a frustrating experience. Users need to know which package contains the model, find the modeling command within the package, and refer to the manual page for the model-specific arguments. In contrast, Zelig users can skip these start-up costs and move directly to data analyses. Using Zelig's unified command syntax, you gain the convenience of a packaged program, without losing any of the power of R's underlying statistical procedures.

In addition to generalizing R packages and making existing methods easier to use, Zelig includes infrastructure that can improve all existing methods and R programs. Even if you know R, using Zelig greatly simplifies your work. It mimics the popular Clarify program for Stata (and thus the suggestions of King, Tomz, and Wittenberg, 2000) by translating the raw output of existing statistical procedures into quantities that are of direct interest to researchers. Instead of trying to interpret coefficients parameterized for modeling convenience, Zelig makes it easy to compute quantities of real interest: probabilities, predicted values, expected values, first differences, and risk ratios, along with confidence intervals, standard errors, or full posterior (or sampling) densities for all quantities. Zelig extends Clarify by seamlessly integrating an option for bootstrapping into the simulation of quantities of interest. It also integrates a full suite of nonparametric matching methods as a preprocessing step to improve the performance of any parametric model for causal inference (see MatchIt). For missing data, Zelig accepts multiply imputed datasets created by Amelia (see King, Honaker, Joseph, and Scheve, 2001) and other programs, allowing users to analyze them as if they were a single, fully observed dataset. Zelig outputs replication data sets so that you (and if you wish, anyone else) will always be able to replicate the results of your analyses (see King, 1995). Several powerful Zelig commands also make running multiple analyses and recoding variables simple.

Using R in combination with Zelig has several advantages over commercial statistical software. R and Zelig are part of the open source movement, which is roughly based on the principles of science. That is, anyone who adds functionality to open source software or wishes to redistribute it (legally) must provide the software accompanied by its source free of charge.As specified in the [http://www.gnu.org/copyleft/llGNU General Public License]. If you find a bug in open source software and post a note to the appropriate mailing list, a solution you can use will likely be posted quickly by one of the thousands of people using the program all over the world. Since you can see the source code, you might even be able to fix it yourself. In contrast, if something goes wrong with commercial software, you have to wait for

[1] Zelig is named after a Woody Allen movie about a man who had the strange ability to become the physical reflection of anyone he met — Scottish, African-American, Indian, Chinese, thin, obese, medical doctor, Hassidic rabbi, anything — and thus to fit well in any situation.

the programmers at the company to fix it (and speaking with them is probably out of the question), and wait for a new version to be released.

We find that Zelig makes students and colleagues more amenable to using R, since the startup costs are lower, and since the manual and software are relatively self-contained. This manual even includes an appendix devoted to the basics of advanced R programming, although you will not need it to run most procedures in Zelig. A large and growing fraction of the world's quantitative methodologists and statisticians are moving to R, and the base of programs available for R is quickly surpassing all alternatives. In addition to built-in functions, R is a complete programming language, which allows you to design new functions to suit your needs. R has the dual advantage that you do not need to understand how to program to use it, but if it turns out that you want to do something more complicated, you do not need to learn another program. In addition, methodologists all over the world add new functions all the time, so if the function you need wasn't there yesterday, it may be available today.

1.2 Getting Help

You may find documentation for Zelig on-line (and hence must be on-line to access it). If you are unable to connect to the Internet, we recommend that you print the pdf version of this document for your reference.

If you are on-line, you may access comprehensive help files for Zelig commands and for each of the models. For example, load the Zelig library and then type at the R prompt:

```
help.zelig(command)  # For help with all zelig commands.
help.zelig(logit)  # For help with the logit model.
```

In addition, help.zelig() searches the manual pages for R in addition to the Zelig specific pages. On certain rare occasions, the name of the help topic in Zelig and in R are identical. In these cases, help.zelig() will return the Zelig help page by default. If you wish to access the R help page, you should use help(topic).

In addition, built-in examples with sample data and plots are available for each model. For example, type demo(logit) to view the demo for the logit model. Commented code for each model is available under the examples section of each model reference page.

Please direct inquiries and problems about Zelig to our listserv at zelig@lists.gking.harvard.edu. We suggest you subscribe to this mailing list while learning and using Zelig: go to https://lists.gking.harvard.edu/mailman/listinfo/zelig. (You can choose to receive email in digest form, so that you will never receive more than one message per day.) You can also browse or search our [https://lists.gking.harvard.edu/pipermail/zelig/‖archive] of previous messages before posting your query.

1.3 How to Cite Zelig

To cite Zelig as a whole, please reference these two sources:

- Kosuke Imai, Gary King, and Olivia Lau. 2007. "Zelig: Everyone's Statistical Software", http://GKing.harvard.edu/zelig.
- Imai, Kosuke, Gary King, and Olivia Lau. (2008). "Toward A Common Framework for Statistical Analysis and Development." Journal of Computational and Graphical Statistics, Vol. 17, No. 4 (December), pp. 892-913.

To refer to a particular Zelig model, please refer to the "how to cite" portion at the end of each model documentation section.

```
x <- 1:10
y <- rnorm(10)
plot(x, y)
```

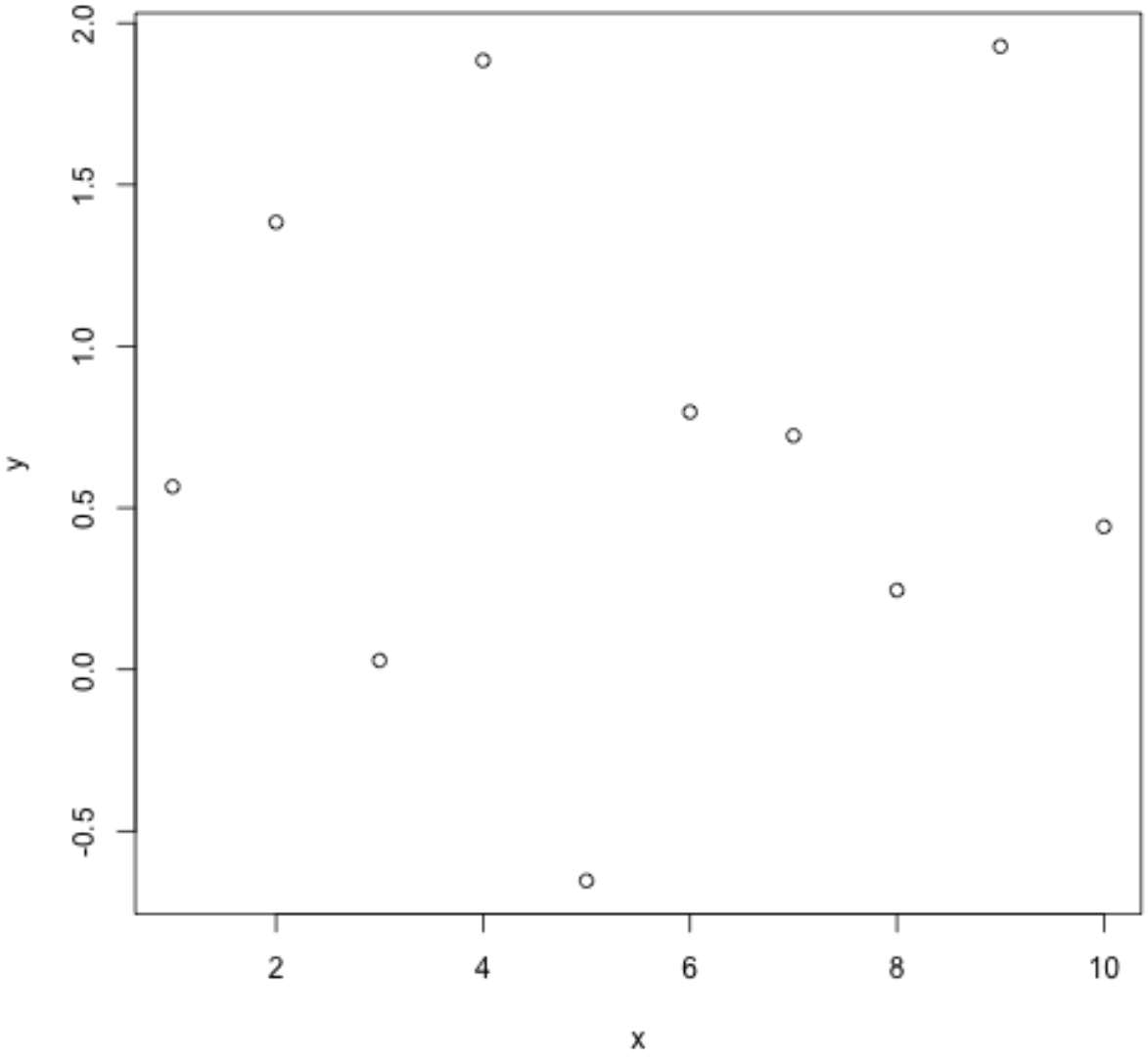


Figure 1.1: plot of chunk unnamed-chunk-2

```
print(y)
```

```
##  [1]  0.56612  1.38396  0.02811  1.88444 -0.65357  0.79660  0.72366
##  [8]  0.24537  1.92806  0.44184
```

```
2 + 3
```

```
## [1] 5
```

CHAPTER

TWO

INSTALLATION

To use Zelig, you must install the statistical program R (if it is not already installed), the Zelig package, and some R libraries (coda, MCMCpack, sandwich, VGAM, and zoo).

Note: In this document, > denotes the R prompt.

2.1 If You Are New to R

If you are new to R, we recommend that you read the following section on installation procedures as well as the overview of R syntax and usage in Sref{a:R}.

This distribution works on a variety of platforms, including Windows (see Section [ss:win]), MacOSX (see Section [ss:osx]), and Linux (see Section [ss:unix]). Alternatively, you may access R from your PC using a terminal window or an X-windows tunnel to a Linux or Unix server (see Section [ss:unix]). Most servers have R installed; if not, contact your network administrator.

There are advantages and disadvantages to each type of installation. On a personal computer, R is easier to install and launch. Using R remotely on a server requires a bit more set-up, but does not tie up your local CPU, and allows you to take advantage of the server's speed.

2.2 If You Know R

We recommend that you launch R and type

```
install.packages("Zelig")
```

then proceed to ??Sref{overview}??. For Windows R, you may edit the Rprofile file to load Zelig automatically at launch (after which you will no longer need to type library(Zelig) at startup). Simply add the line:

```
options(defaultPackages = c(getOption("defaultPackages"), "Zelig"))
```

2.3 Windows

2.3.1 Installing R

Go to the Comprehensive R Archive Network website (http://www.r-project.org) and download the latest installer for Windows at http://cran.us.r-project.org/bin/windows/base/ Double-click the .exe file to launch the R installer. We recommend that you accept the default installation options if this your first installation.

Installing Zelig

Once R is installed, you must install the Zelig package:

```
install.packages("Zelig")
```

Zelig will load the optional libraries whenever their functions are needed; it is not necessary to load any package other than Zelig at startup.

1. Alternatively, you may use the drop down menus to install Zelig.

1. Download the latest release of Zelig from CRAN. Store the .zip files in your R program directory. For example, the default R program directory is C:Program FilesRR-rvers.Note that when updating R to the latest release, the installer does not delete previous versions from your C:Program FilesRdirectory. In this example, the subdirectory R-rversstores R version fullvers. Thus, if you have a different version of R installed, you should change the last part of the R program directory file path accordingly.

2. Start R. From the drop-down menus, select the "Packages" menu and then the "Install Files from Local Zip Files" option.

3. A window will pop up, allowing you to select one of the downloaded files for installation. There is no need to unzip the files prior to installation. Repeat and select the other downloaded file for installation.

4. At the R prompt, type library(Zelig) to load the functionality described in this manual. Note that Zelig will automatically load the other libraries as necessary.

5. An additional recommended but optional step is to set up R to load Zelig automatically at launch. (If you skip this step, you must type library(Zelig) at the beginning of every R session.) To automate this process, edit the Rprofile file located in the R program subdirectory (in our example). Using a text editor such as Windows notepad, add the following line to the Rprofile file:

```
options(defaultPackages = c(getOption("defaultPackages"), "Zelig"))
```

Zelig is distributed under the GNU General Public License. After installation, the source code is located in your R library directory, which is by default C:Program FilesRR-rverslibraryZelig.

2.3.2 Updating Zelig

There are two ways to update Zelig.

1. We recommend that you periodically update Zelig at the R prompt by typing:

```
update.packages()
```

2. Alternatively, you may use the procedure outlined in [win.zelig] to periodically update Zelig. Simply download the latest .zip file and follow the steps.

2.4 Mac OS X

2.4.1 Installing R

If you are using MacOS X, you may install the latest version of R (fullvers at this time) from the CRAN website http://cran.us.r-project.org/bin/macosx/.

2.4.2 Installing Zelig

Once R is installed, you must install the Zelig package:

```
install.packages("Zelig")
```

Alternatively, you may use the drop down menus to install Zelig.

1. Download the latest release of Zelig from CRAN. Save the .tgz files in a convenient place.
2. Start R. From the drop-down menus, select the "Packages" menu and then the "Install Files from Local Files" option.
3. A window will pop up, allowing you to select the downloaded file for installation. There is no need to unzip the file prior to installation.

2.5 Version Compatibility

In addition to R itself, Zelig also depends on several R packages maintained by other development teams. Although we make every effort to keep the latest version of Zelig up-to-date with the latest version of those packages, there may occasionally be incompatibilities. See [table.compat] in the Appendix for a list of packages tested to be compatibile with a given Zelig release. You may obtain older versions of most packages at http://www.r-project.org.

2.6 UNIX and Linux

2.6.1 Installing R

Type R at the terminal prompt (which we denote as % in this section) to see if R is available. (Typing q() will enable you to quit.) If it is installed, proceed to the next section. If it is not installed and you are not the administrator, contact that individual, kindly request that they install R on the server, and continue to the next section. If you have administrator privileges, you may download the latest release at the CRAN website. Although installation varies according to your Linux distribution, we provide an example for Red Hat Linux 9.0 as a guide.

1. Log in as root.
2. Download the appropriate binary file for Red Hat 9 from CRAN.
3. Type the following command at the terminal prompt:

2.6.2 Installing Zelig

Before installing Zelig, you need to create a local R library directory. If you have done so already, you can skip to Section [unix.zelig]. If not, you must do so before proceeding because most users do not have authorization to install programs globally. Suppose we want the directory to be ~/.R/library. At the terminal prompt in your home directory, type:

```
mkdir ~/.R ~/.R/library
```

Now you are ready to install Zelig.

There are two ways to proceed.

1. Recommended procedure:

1. Open the ~/.Renviron file (or create it if it does not exist) and add the following line:

RLIBS = ” /.R/library”

You only need to perform this step once.

1. Start R. At the R prompt, type:

```
install.packages("Zelig")
```

2. Finally, create a .Rprofile file in your home directory, containing the line:

```
library(Zelig)
```

This will load Zelig every time you start R.

After installation, the source code is located in your R library directory. If you followed the example above, this is /.R/library/Zelig/.

Updating Zelig

There are two ways to update Zelig.

1. We recommend that you start R and, at the R prompt, type:

```
update.packages()
```

2. Alternatively, you may remove an old version by command by typing:

   R CMD REMOVE Zelig

Now you are ready to install Zelig.

CHAPTER

THREE

DATA ANALYSIS COMMANDS

3.1 Command Syntax

Once R is installed, you only need to know a few basic elements to get started. It's important to remember that R, like any spoken language, has rules for proper syntax. Unlike English, however, the rules for intelligible R are small in number and quite precise (see Section [s:syntax]).

3.1.1 Getting Started

1. To start R under Linux or Unix, type R at the terminal prompt or M-x R under ESS.
2. The R prompt is >.
3. Type commands and hit enter to execute. (No additional characters, such as semicolons or commas, are necessary at the end of lines.)
4. To quit from R, type `q()` and press enter.
5. The # character makes R ignore the rest of the line, and is used in this document to comment R code.
6. We highly recommend that you make a separate working directory or folder for each project.
7. Each R session has a workspace, or working memory, to store the objects that you create or input. These objects may be:

1. values, which include numerical, integer, character, and logical values;
2. data structures made up of variables (vectors), matrices, and data frames; or
3. functions that perform the desired tasks on user-specified values or data structures.

After starting R, you may at any time use Zelig's built-in help function to access on-line help for any command. To see help for all Zelig commands, type help.zelig(command), which will take you to the help page for all Zelig commands. For help with a specific Zelig or R command substitute the name of the command for the generic command. For example, type help.zelig(logit) to view help for the logit model.

3.1.2 Details

Zelig uses the syntax of R, which has several essential elements:

1. R is case sensitive. Zelig, the package or library, is not the same as zelig, the command.
2. R functions accept user-defined arguments: while some arguments are required, other optional arguments modify the function's default behavior. Enclose arguments in parentheses and separate multiple arguments with commas. For example, `print(x)` or `print(x, digits = 2)` prints the contents of the object x using

the default number of digits or rounds to two digits to the right of the decimal point, respectively. You may nest commands as long as each has its own set of parentheses: `log(sqrt(5))` takes the square root of 5 and then takes the natural log.

3. The <- operator takes the output of the function on the right and saves them in the named object on the left. For example, `z.out <- zelig(\dots)` stores the output from `zelig()` as the object z.out in your working memory. You may use z.out as an argument in other functions, view the output by typing z.out at the R prompt, or save z.out to a file using the procedures described in Section Saving Data.

4. You may name your objects anything, within a few constraints:

- You may only use letters (in upper or lower case) and periods to punctuate your variable names.
- You may not use any special characters (aside from the period) or spaces to punctuate your variable names.
- Names cannot begin with numbers. For example, R will not let you save an object as 1997.election but will let you save election.1997.

5. Use the `names()` command to see the contents of R objects, and the $ operator to extract elements from R objects. For example:

```
# Run least squares regression and save the output in working memory:
z.out <- zelig(y ~ x1 + x2, model = "ls", data = mydata)
# See what's in the R object:
names(z.out)
## [1] 'coefficients' 'residuals' 'effects' 'rank' Extract and display the
## coefficients in z.out:
z.out$coefficients
```

6. a frame is a rectangular matrix with n rows and k columns. Each column represents a variable and each row an observation. Each variable may have a different class. (See Section ??variable.classes?? for a list of classes.) You may refer to specific variables from a data frame using, for example, `data$variable`.

7. A list is a combination of different data structures. For example, `z.out` contains both coefficients (a vector) and data (a data frame). Use `names()` to view the elements available within a list, and the `$` operator to refer to an element in a list.

For a more comprehensive introduction, including ways to manipulate these data structures, please refer to Chapter [a:R].

3.1.3 Loading Data

Datasets in Zelig are stored in "data frames." In this section, we explain the standard ways to load data from disk into memory, how to handle special cases, and how to verify that the data you loaded is what you think it is.

Standard Ways to Load Data

Make sure that the data file is saved in your working directory. You can check to see what your working directory is by starting R, and typing getwd(). If you wish to use a different directory as your starting directory, use setwd("dirpath"), where "dirpath" is the full directory path of the directory you would like to use as your working directory.

After setting your working directory, load data using one of the following methods:

1. If your dataset is in a tab- or space-delimited .txt file, use read.table("mydata.txt")

1. If your dataset is a comma separated table, use read.csv("mydata.csv").

2. To import SPSS, Stata, and other data files, use the foreign package, which automatically preserves field characteristics for each variable. Thus, variables classed as dates in Stata are automatically translated into values in the date class for R. For example:

```
library(foreign)  # Load the foreign package.
stata.data <- read.dta("mydata.dta")  # For Stata data.
spss.data <- read.spss("mydata.sav", to.data.frame = TRUE)  # For SPSS.
```

2. To load data in R format, use load("mydata.RData").

1. For sample data sets included with R packages such as Zelig, you may use the data() command, which is a shortcut for loading data from the sample data directories. Because the locations of these directories vary by installation, it is extremely difficult to locate sample data sets and use one of the three preceding methods; data() searches all of the currently used packages and loads sample data automatically. For example:

```
library(Zelig)  # Loads the Zelig library.
data(turnout)  # Loads the turnout data.
```

Special Cases When Loading Data

These procedures apply to any of the above read commands:

1. If your file uses the first row to identify variable names, you should use the option header = TRUE to import those field names. For example,

```
read.csv("mydata.csv", header = TRUE)
```

will read the words in the first row as the variable names and the subsequent rows (each with the same number of values as the first) as observations for each of those variables. If you have additional characters on the last line of the file or fewer values in one of the rows, you need to edit the file before attempting to read the data.

2. The R missing value code is NA. If this value is in your data, R will recognize your missing values as such. If you have instead used a place-holder value (such as -9) to represent missing data, you need to tell R this on loading the data:

```
read.table("mydata.tab", header = TRUE, na.strings = "-9")
```

Note: You must enclose your place holder values in quotes.

3. Unlike Windows, the file extension in R does not determine the default method for dealing with the file. For example, if your data is tab-delimited, but saved as a .sav file, read.table("mydata.sav") will load your data into R.

Verifying You Loaded The Data Correctly

Whichever method you use, try the names(), dim(), and summary() commands to verify that the data was properly loaded. For example,

```
data <- read.csv("mydata.csv", header = TRUE)  # Read the data.
dim(data)  # Displays the dimensions of the data frame
## [1] 16000 8 # in rows then columns.
data[1:10, ]  # Display rows 1-10 and all columns.
names(data)  # Check the variable names.
## [1] 'V1' 'V2' 'V3' These values indicate that the variables weren't named,
## and took default values.
names(data) <- c("income", "educate", "year")  # Assign variable names.
summary(data)  # Returning a summary for each variable.
```

In this case, the summary() command will return the maximum, minimum, mean, median, first and third quartiles, as well as the number of missing values for each variable.

3.1.4 Saving Data

Use save() to write data or any object to a file in your working directory. For example,

```
## Saves 'mydata' to 'mydata.RData' in your working directory.
save(mydata, file = "mydata.RData")
## Saves your entire workspace to the default '.RData' file.
save.image()
```

R will also prompt you to save your workspace when you use the q() command to quit. When you start R again, it will load the previously saved workspace. Restarting R will not, however, load previously used packages. You must remember to load Zelig at the beginning of every R session.

Alternatively, you can recall individually saved objects from .RData files using the load() command. For example,

```
load("mydata.RData")
```

loads the objects saved in the mydata.RData file. You may save a data frame, a data frame and associated functions, or other R objects to file.

3.2 Variables

3.2.1 Classes of Variables

R variables come in several types. Certain Zelig models require dependent variables of a certain class of variable. (These are documented under the manual pages for each model.) Use class(variable) to determine the class of a variable or class(data$variable) for a variable within a data frame.

Types of Variables

For all types of variable (vectors), you may use the c() command to "concatenate" elements into a vector, the : operator to generate a sequence of integer values, the seq() command to generate a sequence of non-integer values, or the rep() function to repeat a value to a specified length. In addition, you may use the <- operator to save variables (or any other objects) to the workspace. For example:

```
## Creates 'logic' (5 T/F values).
logic <- c(TRUE, FALSE, TRUE, TRUE, TRUE)
## All integers between 10 and 20.
var1 <- 10:20
## Sequence from 5 to 10 by intervals of 0.5.
var2 <- seq(from = 5, to = 10, by = 0.5)
## 20 'NA' values.
var3 <- rep(NA, length = 20)
## 15 '1's followed by 15 '0's.
var4 <- c(rep(1, 15), rep(0, 15))
```

For the seq() command, you may alternatively specify length instead of by to create a variable with a specific number (denoted by the length argument) of evenly spaced elements.

1. Numeric variables are real numbers and the default variable class for most dataset values. You can perform any type of math or logical operation on numeric values. If var1 and var2 are numeric variables, we can compute

```
var3 <- log(var2) - 2 * var1  # Create 'var3' using math operations.
```

2. Inf (infinity), -Inf (negative infinity), NA (missing value), and NaN (not a number) are special numeric values on which most math operations will fail. (Logical operations will work, however.) Use as.numeric() to transform variables into numeric variables. Integers are a special class of numeric variable.

3. Logical variables contain values of either TRUE or FALSE. R supports the following logical operators: ==, exactly equals; >, greater than; <, less than; >=, greater than or equals; <=, less than or equals; and !=, not

equals. The = symbol is not a logical operator. Refer to Section ??logical?? for more detail on logical operators. If var1 and var2 both have n observations, commands such as

```
var3 <- var1 < var2
var3 <- var1 == var2
```

create n

TRUE/FALSE observations such that the i

th observation in var3 evaluates whether the logical statement is true for the i th value of var1 with respect to the i th value of var2. Logical variables should usually be converted to integer values prior to analysis; use the as.integer() command.

4. Character variables are sets of text strings. Note that text strings are always enclosed in quotes to denote that the string is a value, not an object in the workspace or an argument for a function (neither of which take quotes). Variables of class character are not normally used in data analysis, but used as descriptive fields. If a character variable is used in a statistical operation, it must first be transformed into a factored variable.
5. Factor variables may contain values consisting of either integers or character strings. Use factor() or as.factor() to convert character or integer variables into factor variables. Factor variables separate unique values into levels. These levels may either be ordered or unordered. In practice, this means that including a factor variable among the explanatory variables is equivalent to creating dummy variables for each level. In addition, some models (ordinal logit, ordinal probit, and multinomial logit), require that the dependent variable be a factor variable.

3.2.2 Recoding Variables

Researchers spend a significant amount of time cleaning and recoding data prior to beginning their analyses. R has several procedures to facilitate the process.

Extracting, Replacing, and Generating New Variables

While it is not difficult to recode variables, the process is prone to human error. Thus, we recommend that before altering the data, you save your existing data frame using the procedures described in Section ??s:save??, that you only recode one variable at a time, and that you recode the variable outside the data frame and then return it to the data frame.

To extract the variable you wish to recode, type:

```
## Copies 'var1' from 'data', creating 'var'.
var <- data$var1
```

Do not sort the extracted variable or delete observations from it. If you do, the ith observation in var will no longer match the ith observation in data. To replace the variable or generate a new variable in the data frame, type:

```
## Replace 'var1' in 'data' with 'var'.
data$var1 <- var
## Generate 'new.var' in 'data' using 'var'.
data$new.var <- var
```

To remove a variable from a data frame (rather than replacing one variable with another):

```
data$var1 <- NULL
```

Alternatively, rather than recoding just specific values in variables, you may calculate new variables from existing variables. For example,

```
var3 <- var1 + 2 * var2
var3 <- log(var1)
```

After generating the new variables, use the assignment mechanism <- to insert the new variable into the data frame.

In addition to generating vectors of dummy variables, you may transform a vector into a matrix of dummy indicator variables. unique values (with n observations in the complete vector) into a ntimes k matrix.

Missing Data

To deal with missing values in some of your variables:

1. You may generate multiply imputed datasets using Amelia (or other programs).

2. You may omit missing values. Zelig models automatically apply list-wise deletion, so no action is required to run a model. To obtain the total number of observations or produce other summary statistics using the analytic dataset, you may manually omit incomplete observations. To do so, first create a data frame containing only the variables in your analysis. For example:

```
new.data <- cbind(data$dep.var, data$var1, data$var2, data$var3)
```

The cbind() command "column binds" variables into a data frame. (A similar command rbind() "row binds" observations with the same number of variables into a data frame.) To omit missing values from this new data frame:

```
new.data <- na.omit(new.data)
```

If you perform na.omit() on the full data frame, you risk deleting observations that are fully observed in your experimental variables, but missing values in other variables. Creating a new data frame containing only your experimental variables usually increases the number of observations retained after na.omit().

CHAPTER

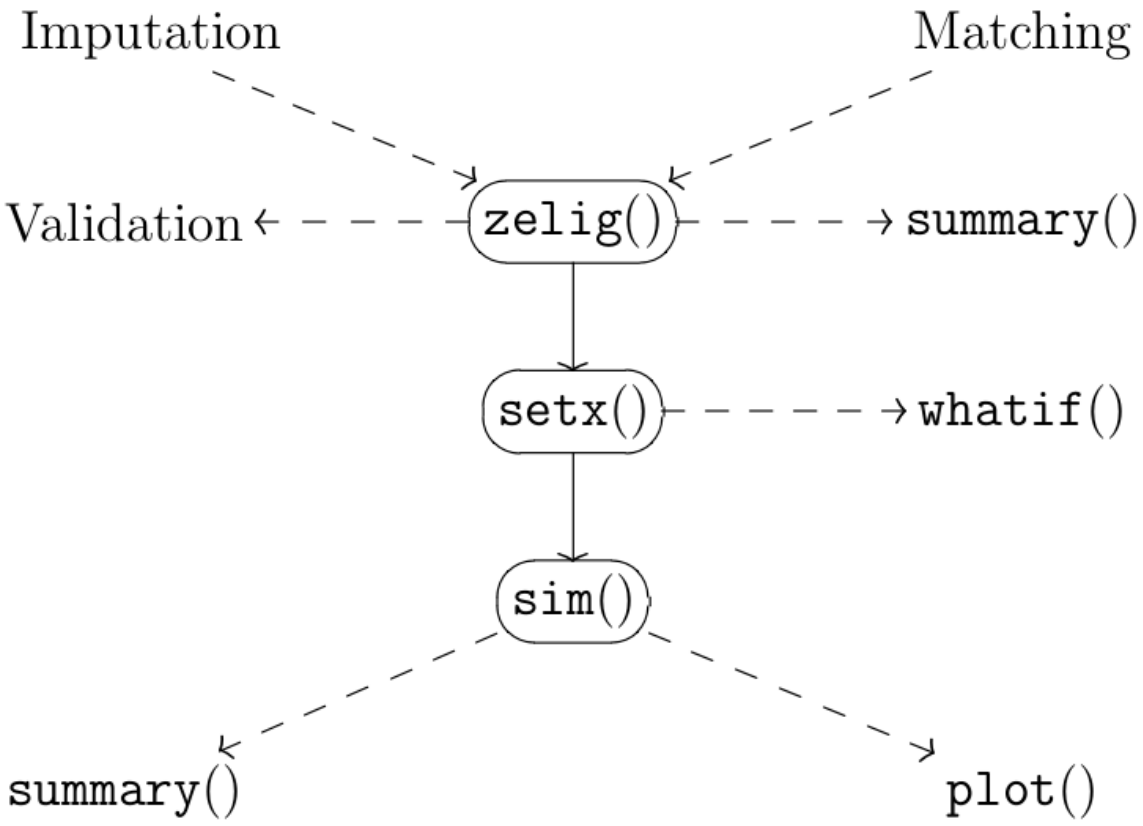
FOUR

STATISTICAL COMMANDS

4.1 Zelig Commands

4.1.1 Quick Overview

For any statistical model, Zelig does its work with a combination of three commands.



1. Use `zelig()` to run the chosen statistical model on a given data set, with a specific set of variables. For standard likelihood models, for example, this step estimates the coefficients, other model parameters, and a variance-covariance matrix. In addition, you may choose from a variety of options:

- Pre-process data: Prior to calling `zelig()`, you may choose from a variety of data pre-processing commands (matching or multiple imputation, for example) to make your statistical inferences more accurate.

- Summarize model: After calling `zelig()`, you may summarize the fitted model output using `summary()`.
- Validate model: After calling `zelig()`, you may choose to validate the fitted model. This can be done, for example, by using cross-validation procedures and diagnostics tools.

2. Use `setx()` to set each of the explanatory variables to chosen (actual or counterfactual) values in preparation for calculating quantities of interest. After calling `setx()`, you may use `WhatIf` to evaluate these choices by determining whether they involve interpolation (i.e., are inside the convex hull of the observed data) or extrapolation, as well as how far these counterfactuals are from the data. Counterfactuals chosen in `setx()` that involve extrapolation far from the data can generate considerably more model dependence (see (37), (39), (48)).

3. Use `sim()` to draw simulations of your quantity of interest (such as a predicted value, predicted probability, risk ratio, or first difference) from the model. (These simulations may be drawn using an asymptotic normal approximation (the default), bootstrapping, or other methods when available, such as directly from a Bayesian posterior.) After calling `sim()`, use any of the following to summarize the simulations:

- The `summary()` function gives a numerical display. For multiple `setx()` values, `summary()` lets you summarize simulations by choosing one or a subset of observations.
- If the `setx()` values consist of only one observation, `plot()` produces density plots for each quantity of interest.

Whenever possible, we use `z.out` as the `zelig()` output object, `x.out` as the `setx()` output object, and `s.out` as the `sim()` output object, but you may choose other names.

4.1.2 Examples

- Use the turnout data set included with Zelig to estimate a logit model of an individual's probability of voting as function of race and age. Simulate the predicted probability of voting for a white individual, with age held at its mean:

```
data(turnout)
z.out <- zelig(vote ~ race + age, model = "logit", data = turnout)
x.out <- setx(z.out, race = "white")
s.out <- sim(z.out, x = x.out)
summary(s.out)
```

- Compute a first difference and risk ratio, changing education from 12 to 16 years, with other variables held at their means in the data:

```
data(turnout)
z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout)
x.low <- setx(z.out, educate = 12)
x.high <- setx(z.out, educate = 16)
s.out <- sim(z.out, x = x.low, x1 = x.high)
summary(s.out)
# Numerical summary.
plot(s.out)
```

- Calculate expected values for every observation in your data set:

```
data(turnout)
z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout)
x.out <- setx(z.out, fn = NULL)
s.out <- sim(z.out, x = x.out)
summary(s.out)
```

- Use five multiply imputed data sets from (47) in an ordered logit model:

```
data(immi1, immi2, immi3, immi4, immi5)
z.out <- zelig(as.factor(ipip) ~ wage1992 + prtyid + ideol, model = "ologit",
    data = mi(immi1, immi2, immi3, immi4, immi5))
```

- Use the nearest propensity score matching via MatchIt package, and then calculate the conditional average treatment effect of the job training program based on the linear regression model:

```
library(MatchIt)
data(lalonde)
m.out <- matchit(treat ~ re74 + re75 + educ + black + hispan + age, data = lalonde,
    method = "nearest")
m.data <- match.data(m.out)
z.out <- zelig(re78 ~ treat + distance + re74 + re75 + educ + black + hispan +
    age, data = m.data, model = "ls")
x.out0 <- setx(z.out, fn = NULL, treat = 0)
x.out1 <- setx(z.out, fn = NULL, treat = 1)
s.out <- sim(z.out, x = x.out0, x1 = x.out1)
summary(s.out)
```

- Validate the fitted model using the leave-one-out cross validation procedure and calculating the average squared prediction error via `boot` package. For example:

```
library(boot)
data(turnout)
z.out <- zelig(vote ~ race + educate, model = "logit", data = turnout, cite = F)
cv.out <- cv.glm(z.out, data = turnout, k = 11)
print(cv.out$delta)
```

4.1.3 Details

1. `z.out <- zelig(formula, model, data, by = NULL, ...)`

The `zelig()` command estimates a selected statistical model given the specified data. You may name the output object (`z.out` above) anything you desire. You must include three required arguments, in the following order:

1. `formula` takes the form `y ~ x1 + x2`, where `y` is the dependent variable and `x1` and `x2` are the explanatory variables, and `y`, `x1`, and `x2` are contained in the same dataset. The + symbol means "inclusion" not "addition." You may include interaction terms in the form of `x1*x2` without having to compute them in prior steps or include the main effects separately. For example, R treats the formula `y ~ x1*x2` as `y ~ x1 + x2 + x1*x2`. To prevent R from automatically including the separate main effect terms, use the `I()` function, thus: `y ~ I(x1 * x2)`.
2. `model` lets you choose which statistical model to run. You must put the name of the model in quotation marks, in the form `model = "ls"`, for example. See Section ??4.3?? for a list of currently supported models.
3. `data` specifies the data frame containing the variables called in the formula, in the form `data = mydata`. Alternatively, you may input multiply imputed datasets in the form `data = mi(data1, data2, ...)`.[1] If you are working with matched data created using `MatchIt`, you may create a data frame within the `zelig()` statement by using `data = match.data(...)`. In all cases, the data frame or MatchIt object must have been previously loaded into the working memory.
4. `by` (an optional argument which is by default `NULL`) allows you to choose a factor variable (see Section ??2??) in the data frame as a subsetting variable. For each of the unique strata defined in the by variable, `zelig()` does a separate run of the specified model. The variable chosen should not be in the formula, because there will be no variance in the by variable in the subsets. If you have one data set for all 191 countries in the UN, for

[1] Multiple imputation is a method of dealing with missing values in your data which is more powerful than the usual list-wise deletion approach. You can create multiply imputed datasets with a program such as Amelia; see King, Honaker, Joseph, Scheve (2000).

example, you may use the by option to run the same model 191 times, once on each country, all with a single `zelig()` statement. You may also use the `by` option to run models on `MatchIt` subclasses.

5. The output object, `z.out`, contains all of the options chosen, including the name of the data set. Because data sets may be large, Zelig does not store the full data set, but only the name of the dataset. Every time you use a Zelig function, it looks for the dataset with the appropriate name in working memory. (Thus, it is critical that you do not change the name of your data set, or perform any additional operations on your selected variables between calling `zelig()` and `setx()`, or between `setx()` and `sim()`.)

6. If you would like to view the regression output at this intermediate step, type `summary(z.out)` to return the coefficients, standard errors, t-statistics and p- values. We recommend instead that you calculate quantities of interest; creating `z.out` is only the first of three steps in this task.

2. `x.out <- setx(z.out, fn = list(numeric = mean, ordered = median, others = mode), data = NULL, cond = FALSE, ...)`

   The `setx()` command lets you choose values for the explanatory variables, with which `sim()` will simulate quantities of interest. There are two types of `setx()` procedures:

   - You may perform the usual unconditional prediction (by default, `cond = FALSE`),

     by explicitly choosing the values of each explanatory variable yourself or letting `setx()` compute them, either from the data used to create `z.out` or from a new data set specified in the optional data argument. You may also compute predictions for all observed values of your explanatory variables using `fn = NULL`.

   - Alternatively, for advanced uses, you may perform conditional prediction (`cond = TRUE`),

     which predicts certain quantities of interest by conditioning on the observed value of the dependent variable. In a simple linear regression model, this procedure is not particularly interesting, since the conditional prediction is merely the observed value of the dependent variable for that observation. However, conditional prediction is extremely useful for other models and methods, including the following:

     - In a matched sampling design, the sample average treatment effect for the treated can be estimated by computing the difference between the observed dependent variable for the treated group and their expected or predicted values of the dependent variable under no treatment (16).
     - With censored data, conditional prediction will ensure that all predicted values are greater than the censored observed values (28).
     - In ecological inference models, conditional prediction guarantees that the predicted values are on the tomography line and thus‘‘.

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*