R BASICS

OBJECTIVES OF THIS CHAPTER

This chapter begins with an introduction to the basics of R. The first section shows novice R users the interface, menus, and R toolbar. It introduces the RStudio and R Commander, the graphical user interfaces (GUIs), which make the use of R easier. It also introduces objects in R and R functions and arguments and shows how to install R and add-on packages, create script files, and import data. The second section shows readers several commonly used data types and various data management techniques, such as how to select cases and variables, create and recode variables, label categories for factors, and label variables. Further, it introduces `tidyverse` and how to use both the `dplyr` package in `tidyverse` and the `sjmisc` package for data management. The third section introduces basic graphic functions and the `ggplot2` package. After reading this chapter, you should be able to:

- Install, start, and exit R.
- Open existing data files.
- Enter commands, create a script file, and save output.
- Select cases and variables, create new variables, recode and label variables, and label values for categorical variables.
- Draw different types of graphs.

Do not copy, post, or distribute

1.1 INTRODUCTION TO R

R is a programming language. It was developed by Ross Ihaka and Robert Gentleman in the 1990s based on the S programming language. After several years of development, R version 1.0.0 was officially released in 2000. R is now developed by the R Development

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

Core Team. It can be run on almost all operating systems across Windows, Mac, Linux, and Unix. R has different names. It is also called a program, a package, a system, or an environment. In this book, we treat R as a general-purpose statistical package and a programming language. Just like IBM SPSS, SAS, and Stata, R is also a general-purpose statistical tool for data management, data analysis, and graphing.

R is a powerful tool for data management, graphics, and data analyses. It is capable of conducting various statistical analyses, from basic statistical analyses to more complex models, such as generalized linear models, generalized additive models, multivariate analyses, time series, survival analysis, propensity score analysis, multilevel modeling, structural equation modeling, cluster analysis, machine learning, and Bayesian statistics. As an R user, you will be amazed at the capacities of R for statistical analysis. If I claim that R can do any statistical analysis, it may sound like I am exaggerating. However, if you name a modern statistical method, it is very likely that you can find an R package or function which has been developed for that method.

R has extensive programming capabilities. You can write packages with new functions which can be shared with other users in the R community. With the contributions of experts from various fields, new statistical techniques can be quickly implemented in R. For example, the user-written package `ordinal` was developed for ordinal regression models, the `VGAM` package was developed for generalized linear and additive models, and the `lme4` package was developed for mixed-effects models or multilevel models. We can install these user-written packages using the `install.packages()` function. After installation, these add-on packages can be executed in the same way as the base package in R. So, when functions of interest are unavailable in the base package in R, users can search online to see if they have been developed by other users.

R provides more than one solution for your statistical analysis. You may find multiple packages with similar functions for a statistical method. For example, you can use either the `clm()` function in the `ordinal` package or the `vglm()` function in the `VGAM` package to fit ordinal logistic regression models. There are also several packages developed for multinomial logistic regression models, such as the `vglm()` function in `VGAM`, the `multimon()` function in `nnet`, and the `mlogit()` function in `mlogit`. Therefore, you have a variety of choices to solve your problems. In addition, the results can be cross-validated if multiple packages are used for the same analysis.

R is an open-source programming language. This means that the source code is freely available to use. You can also modify the source code, create your own, and share it with your colleagues or in the R community.

R is free. It is free to download, install, and use. The benefit of having a free statistical package is self-evident.

1.1.1 Installing, Starting, and Exiting R

To install R on your computer, you need to download it from the Comprehensive R Archive Network (CRAN) at http://cran.r-project.org/. The CRAN is the online network for storing the R software, add-on packages, and documentation. Choose the

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

link for the R installation file for Windows, Linux, or macOS operating systems. At the time of writing, the R version is R 4.1.0 (R Core Team, 2021). The installation of R is just as easy as the installation of any other program on these operating systems. For example, to install R on Windows, run the installation file by right clicking it and then select Run as Administrator. Then follow the steps to install it.

You can install 32-bit, 64-bit, or both versions on your computer. The advantage of the 64-bit version is that it can handle large data as long as your computer memory allows for it. Install 64-bit version if your computer supports 64-bit. To update R, you simply download and install the latest version following the procedures introduced above. You can either uninstall or keep the old version on your computer. In this book, we focus on R for Windows.

R can be started in two ways. First, you can run R by double-clicking the icon on the desktop. Second, you can start it by clicking the **Start** Menu on Windows, **All App**, and then **R**.

To exit R, you can either type the command `q()` and press **Enter** or use the pull-down menu. To use the menu, go to **File** and then click on **Exit**.

R is mainly a command-driven statistical package. To execute a command, you need to enter it on the command line and press the **Enter** key. Why do we still need to type commands in R instead of using point-and-click menus in other statistical packages? There is nothing wrong with using the graphic user interface (GUI) pull-down menus, but you may find that it will be more efficient to type commands. There are three reasons for this efficiency. First, it saves you time. For example, to run a simple regression analysis with a dependent variable `y` and an independent variable `x` in the `data1` dataset, you simply type:

```
lm(y ~ x, data = data1)
```

Second, it helps reproducible research. You can save all your commands to a script file so that you can replicate your analysis easily and share it with others. In addition, you can edit your script file for other analyses. For example, if we have three independent variables in a linear regression analysis, then we can modify the previous command as follows:

```
lm(y ~ x1 + x2 + x3, data = data1)
```

Third, it is extensible with add-on packages. You can use a variety of packages and functions in R which are not readily available on the GUI pull-down menus in other statistical packages. Due to R's extensive programming capacity, the number of add-on packages increases tremendously each year. You have more opportunities to find the right package to get your job done. In addition, you can write your own functions or edit the existing functions in R to make your work more efficient.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

1.1.2 R at First Sight: R Console, Menus, and Toolbar

R Console

Once you start R, you will see the R Console windows (Figure 1.1).

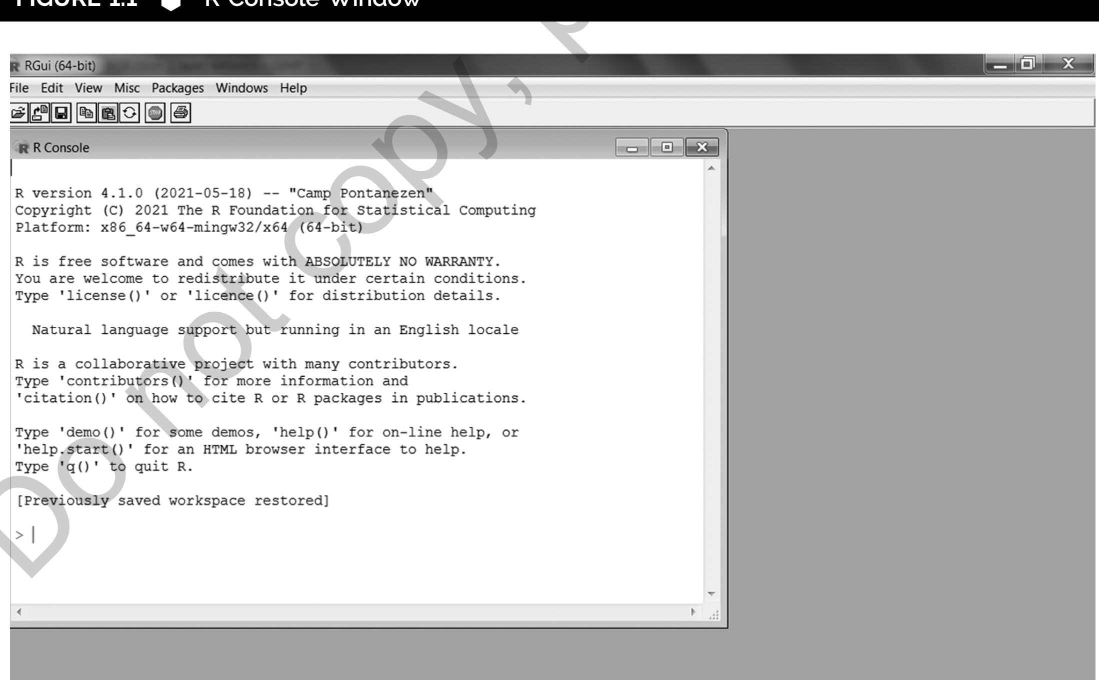
In the R Console window, you need to type a command next to the > symbol and press the **Enter** key to execute it. You can copy a command from a Notepad and paste it here. You can also edit a command before execution. You can only execute one command at a time in this window, whereas you can run a series of commands via the script file which will be explained later in the chapter. R is interactive. After you execute a command, the output is displayed below. The more commands you type, the more results you get. The output can be copied and pasted into a text file or a Word document. It can also be saved into a text file, which will be explained in more detail next.

R Menus

R has seven pull-down main menus, including File, Edit, View, Misc, Packages, Windows, and Help. These menus provide basic tools and features that can be used by pointing and clicking.

File menu: The purpose of this menu is to help you with files. Options in this menu help you open and save a script file, load and save a workspace, load and save a history file, change directory, print results, and exit R.

FIGURE 1.1 ● R Console Window



Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

Edit menu: The Edit menu helps you copy texts from the R Console window and paste them in the same or another location. It also provides the Data Editor and the GUI Preference Editor.

View menu: The View menu shows you the tool bar and the status bar.

Misc menu: The Misc menu helps you stop the current computation, stop all computations, list objects, remove objects, and list search path.

Packages menu: The Packages menu helps you install, update, and load packages, select CRAN mirrors, and select repositories.

Windows menu: This menu helps you organize the R Console window. Several options include Cascade, Tile Horizontally, and Tile Vertically.

Help menu: This menu provides FAQ on R, R manuals in PDF, help on R functions, html help, search help, search for words in help list archives and documentation, and the links for R project and CRAN home pages, respectively.

Please note that R does not provide menus for data management, graphs, and statistics. If you need a GUI for such functions, you need to install an add-on package, the R Commander package (i.e., `Rcmdr`), which was developed by John Fox and his colleagues. An introduction to this package is provided in a later section.

R Toolbar

The toolbar, located below the main menus, comprises a set of icons. It helps you quickly access the most frequently used features. Familiarizing yourself with these icons will make the use of R more efficient.

These icons include Open Scripts, Load Workspace, Save Workspace, Copy, Paste, Copy and Paste, Stop Current Computation, and Print. Table 1.1 shows the icons of the toolbar, their titles, and their functions.

1.1.3 RStudio

RStudio (RStudio Team, 2020) is a free, open-source integrated development environment (IDE) for R. An IDE basically is a program that makes programming easier. RStudio includes an R console, a text editor, and tools for workspace, history, plots, packages, and help. RStudio not only makes R look fancy, but also provides a variety of tools which make R more user-friendly and more convenient.

You can download RStudio at https://rstudio.com/. At the website, go to Products, click on RStudio, and then choose the open-source edition. Follow the steps to install it on your

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

TABLE 1.1 ⬢ Icons of the Toolbar, Their Titles, and Their Functions

Icon	Icon Title	Functions
	Open Scripts	Opens an existing script file (i.e., .R file)
	Load Workspace	Opens an existing workspace file (i.e, .RData file)
	Save Workspace	Saves the current workspace file (i.e, .RData file) to your computer
	Copy	Copies the selected text
	Paste	Pastes the copied text
	Copy and Paste	Copies and pastes the selected text to the current command line
STOP	Stop Current Computation	Stops executing the current program
	Print	Prints your R output

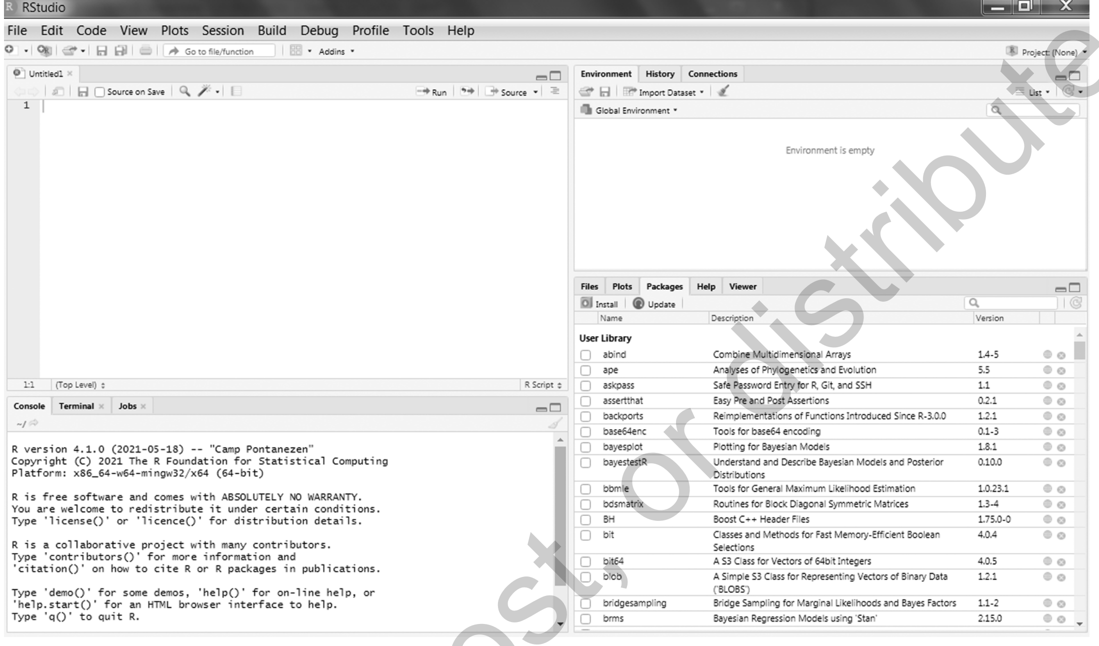
computer. You need to install R first and then install RStudio. At the time of this writing, the RStudio version is RStudio 1.4.1717-3. Figure 1.2 shows the screenshot of RStudio.

- The upper left panel is the R script editor, an enhanced text-editor with highlighting. You can type R commands and create a script file.
- The lower left panel is the R console. The output is displayed after you execute R commands in the R script editor. You can also type and execute R commands in the R console.
- The upper right panel contains the workspace and history of commands. The workspace displays the existing and temporary datasets and other objects which you create. The history tab shows the history of R commands.
- The lower right panel shows the tabs for files, plots, packages, help, and viewer. The Files tab helps you see file directories and manage files; the Plots tab displays the graphs you create; the Packages tab displays the installed packages; the Help tab provides documentation and other help files or pages for R functions and packages; and the Viewer tab does not provide much information, but it can be useful for a particular package to display local web content.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 1.2 ● Panels of RStudio



1.1.4 R Commander

R is a programming language, which does not have a well-built GUI system itself. The R Commander package (i.e., `Rcmdr`), which was developed by John Fox and his colleagues (Fox, 2005, 2017), provides a GUI for R. It is a menu system for reading data and conducting statistical analysis, which facilitates users to learn about the program. Users can use point-and-click menus to familiarize themselves with the features of importing data, recoding variables, making graphics, and choosing a variety of methods for statistical analysis. It is useful for novices or users who are familiar with other statistical packages such as IBM SPSS. It serves as a nice transitional tool to ease the anxiety of importing data and conducting data analysis without programming.

To install the `Rcmdr` package, use the following command.

```
install.packages("Rcmdr", dependencies = TRUE)
```

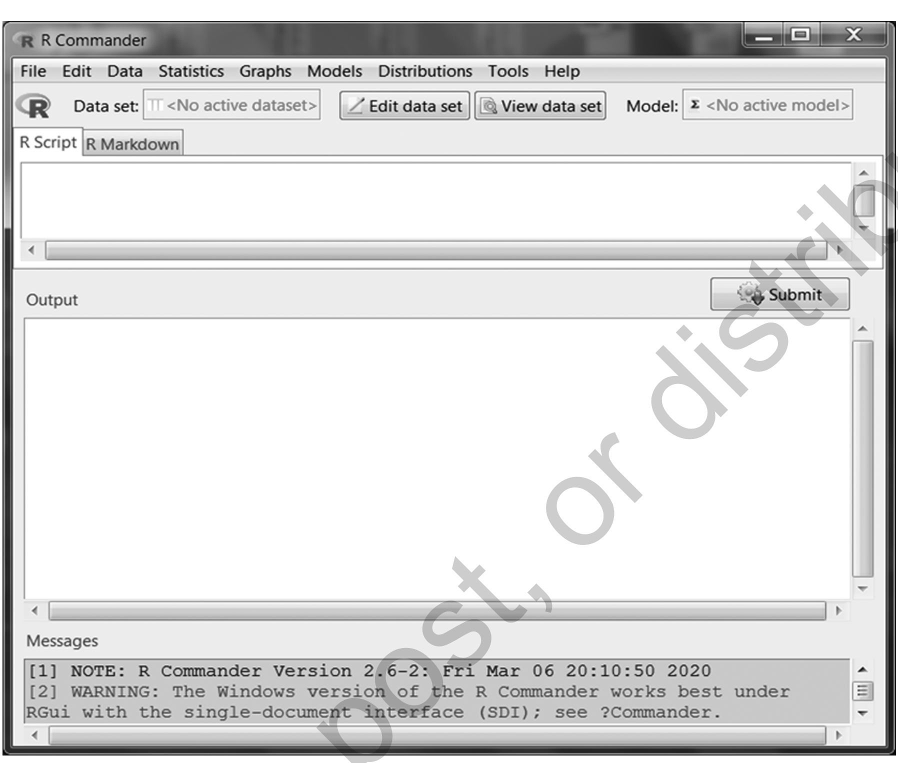
Choose the mirror close to you and then install it. To use the package, you need to load it using the `library(Rcmdr)` command. The screenshot is displayed in Figure 1.3.

The R Commander window provides a script file window at the top, an output window in the middle, and a messages window at the bottom. Just like IBM SPSS, you can load a dataset and then choose the statistical method of interest. You can also use menus to complete data management tasks, such as creating, deleting, and recoding variables, adding

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 1.3 ● Screenshot of R Commander



observations, and converting variables to factors. When you perform an analysis using the R Commander package, it displays the corresponding command in the script file window and the output in the output window. You can then save the command for future use.

There is nothing wrong with using the GUI pull-down menus to get the job done. As you become an experienced R user however, you will find that it will be more efficient to type commands and you will have a wide range of choices of packages.

1.1.5 R Base Package and Add-on Packages

The R base package is on your computer once R is installed. It contains many built-in functions for statistical analysis and graphing, such as `mean()` for means, `lm()` for linear regression models, and `glm()` for generalized linear models. The add-on packages are the user-written packages, also known as the third-party packages. They complement the base package by extending the capabilities of R. They are free and can easily be installed. They are normally stored on the CRAN.

You need to install the package of interest either by using the `install.packages()` function with the name of the package placed in quotation marks or by using

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

the pull-down menu in RStudio or R. For example, the `ordinal` package is the user-written package for ordinal regression models. To install it by typing the command in R, you need to type: `install.packages("ordinal")`. Choose the mirror close to you and then follow steps to install it. If you want to install the package in RStudio, you type the `install.packages("ordinal")` command in the script editor window and run the command by highlighting it. You only need to install it once. To see the installed packages in RStudio, click on the **Packages** tab in the lower right panel.

You also need to load it after the installation by using the `library(ordinal)` command. It needs to be loaded again if you restart the R session. An error message will be displayed if you load a package which has not been installed. As of the writing, there are more than 15,000 packages available on the CRAN. The rapidly increasing add-on packages promote R's popularity.

1.1.6 Objects in R

R is an object-oriented language. In other words, everything in R can be treated as an object. Therefore, a data frame, a graph, a function, and a fitted model are all objects in R. For novices, this seems a confusing concept if no examples are provided. To clarify, let us see an example. We assign the value 5 to an object named x with the following command.

```
x <- 5
```

The assignment operator, <-, which is composed of a less than sign (<) immediately followed by a hyphen (-), is used to assign the value to the object. The value being assigned to the object is on the right-hand side of the assignment operator and the object name is on the left-hand side. We can also use the equal sign, =, as the assignment operator.

We can assign a linear model to an object named `mod` using the following command.

```
mod <- lm(y ~ x)
```

In each of the two examples, the object has a name and content. Typing the object name shows the content of the object. Although you can use any name you like, it is always good practice to have a meaningful and concise object name. Having a meaningful name helps you understand the object. Also, it is preferable to have a concise name rather than a lengthy one. An object name can be a combination of letters, numbers, and dots. There are several basic rules for naming objects.

1. Object names should not start with a number. They should start with a letter or dot. For example, 1.mod is not a correct object name, but mod.1 is.
2. Object names allow letters, numbers, underscore characters (_), and dots (.). You should avoid space and special characters such as @, #, $, %, and &. For example, you can have an object named `mod.1` or `mod_1`, but not `mod$1` since $ is an operator for accessing a variable.
3. R is case-sensitive. This rule also applies to object names, so the object `mod.1` is different from `Mod.1`.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

1.1.7 Functions and Arguments in R

Functions in R are just like commands or procedures in other statistical packages. They are simply a set of instructions to perform a specific task. The form of an existing function in R is the function name with an argument or a set of arguments within the parentheses. In a programming language, arguments are inputs or parameters in a function, which are like options in other statistical packages. Let us see an example of a built-in function, `mean()`. To compute the mean of the variable x1, we use the `mean(x1)` command. In the command, `mean()` is the function for means and `x1` is the argument or input. Executing the command returns the output. When there are multiple arguments, they are separated by commas. For example, in the `mean(x1, na.rm = TRUE)` command, there are two arguments. One argument is the variable object and the other argument `na.rm = TRUE` is used to remove missing values. In a programming language, when executing a code in a function, we also say that we call a function. A call to that function just means that we execute that function. In the two examples above, we call the `mean()` function with the parentheses containing the arguments.

If you would like to create a new function, you can use the `function()` function. You need to assign the function a new name and have a body of statements. The body of a function is always enclosed by curly braces.

1.1.8 R Script Files

If you use the Console window to enter commands, you can enter one piece at a time. An R script file is a text file containing a list of R commands. R script files can help you put a list of commands in one file and execute them together as a batch. Using script-files helps you to organize commands, keep a record, and understand what you have done when you need them in the future. It is also helpful when you collaborate with other researchers on a research project. Your colleagues can simply replicate the analyses using the script files you provide, and they can modify them for new analyses. It will save you time when you need to replicate your statistical analyses. If you are an instructor, these script files are also helpful when preparing your instruction and for grading students' assignments.

To create a new script file in R, go to **File** and then click on **New Script**. You will see a new Untitled-R Editor window. To open an existed script in R, go to **File** and then click on **Open Script**. To run the script, you can highlight the commands, right click it and choose **Run line or selection**. You can also click on **Select all** and then run all the commands at once.

To save your script file in R, go to File and then click on Save as. Save the script with an ".R" extension.

RStudio has a highlighting text editor for script files, also known as a script editor. To create a new script file in RStudio, go to **File**, **New File**, and then click on **New Script**. One advantage to using the script editor in RStudio is that RStudio automatically loads the latest script when you open the program. Therefore, if you forget to save the script file in the last session, you can still do so. To run the script in RStudio, highlight the commands and click on **Run** at the upper right corner of the script editor.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

In this book, all the R commands are saved as R scripts files in RStudio. The R scripts are provided at the end of each chapter. They are also available online to download so you can replicate all the analysis.

You should save the script file to the current working directory. A working directory is the place where R scripts, data files, output, and graphs are stored. Although it is not required to set a working directory to run R, it is cumbersome to type the full path with the file name to access that file each time. By setting the working directory, you can access the file folder directly without providing the full file path. Use the `setwd(" ")` function with the full path in the quotation marks to set the working directory. For example, the `setwd("C:/CDA")` command sets the existing folder named CDA in the C drive as the working directory. You need to create the folder first before setting is as the working directory. To check your current working directory, type the command `getwd()`.

A good habit when you create a script file is to have comments in the file. Comments in an R script file are lines beginning with a pound sign or a hash symbol (#). The comments may include time, project name, who wrote it, and for what purposes. The clearer your comments are, the better your documentation will be. The comments immediately following # in the script files will not be executed but will be displayed in your output.

1.1.9 How to Open an Existing Dataset via the Command Line or the Menus in RStudio

A dataset is called a data frame in R. A data frame is a rectangular-shaped dataset with variables in columns and observations in rows. The data frame in R is similar to the dataset in an Excel spreadsheet or statistical packages such as IBM SPSS, SAS, and Stata. R has other data structures, such as vectors, lists, matrices, and factors, which will be introduced in the following section.

If you work on a research project using R, the first question you may ask is "How can I open or import an existing IBM SPSS, SAS, Stata, or Excel dataset into R?" For novices, importing an existing dataset into R can be a daunting task or frustrating experience. Since R is a programming language, you cannot double click the dataset to open it. If you are not using the R Commander package (i.e., `Rcmdr`), you cannot open the dataset via the R pull-down menu, either. One solution is to use the `foreign` package which has functions to read SAS, SPSS, Stata, Systat, and other formats of data into R. The other solution is to open the dataset via the pull-down menus in RStudio. We will introduce both methods next.

The `foreign` package is now part of the R base installation, so you do not need to install it. However, if you would like to update it to the latest version, you can use the `install.packages("foreign")` command to install it. To use this package, we use the `library(foreign)` command to load it first.

Importing a Stata Dataset

In the following example, we would like to directly open the GSS:2016 dataset, which is a Stata data file with the .dta extension format. We use the `read.dta()` function with the file name enclosed in quotation marks. The basic syntax is as follows.

```
read.dta(file = " ")
```

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

To read a Stata 13 or higher dataset, use the `read.dta13()` function in the `readstata13` package. You need to install the `readstata13` package first by typing `install.packages("readstata13")` since it is a user-written package. After installation, load the package by typing `library(readstata13)`.

Importing a SPSS Dataset

To read a SPSS dataset with the .sav extension, use the `read.spss()` function.

Importing a SAS Dataset

To read a SAS permanent dataset, use the `read.ssd()` function; to read a SAS transport format, use the `read.xport()` function.

We can also use the `read_sas()` function in the `haven` package to open a SAS dataset. You need to install the `haven` package first by typing `install.packages("haven")` and then load the package by typing `library(haven)`.

Importing an Excel Dataset

To read an Excel dataset with the .csv extension, use the `read.table()` function or the `read.csv()` function.

To read an Excel dataset with the .xls extension, use the `read.xls()` function in the `gdata` package. You need to use `install.packages("gdata")` to install the package first and then load it with the `library(gdata)` command.

Importing a Delimited Text Dataset

To read an ASCII text dataset with the .txt extension, use the `read.table()` function.

R has its own data file format which has the .RData extension. You can use the `save()` function to save the data. For example, in the `save(data1, file = "data1.RData")` command, `data1` is the data object name and `file = "data1.RData"` specifies the data format.

Importing a Dataset Through the Menus in RStudio

In addition to typing the commands in the command line, it is also easy and handy to import an existing dataset through the menu system in RStudio with the following steps:

- First, go to **File** on the RStudio main menu and then click on **Import Dataset**. Select **From Text**, **From Excel**, **From SPSS**, **From SAS**, or **From Stata** according to your data file format and then click on it. You may be prompted to install the required packages. After you click on **Yes**, RStudio will automatically install them for you (Figure 1.4).
- Second, in the pop-up window, click on **Browse** to locate the dataset on your computer, select it, and click on **Open** (Figure 1.5).

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 1.4 ● Screenshot of Selecting a Dataset via RStudio

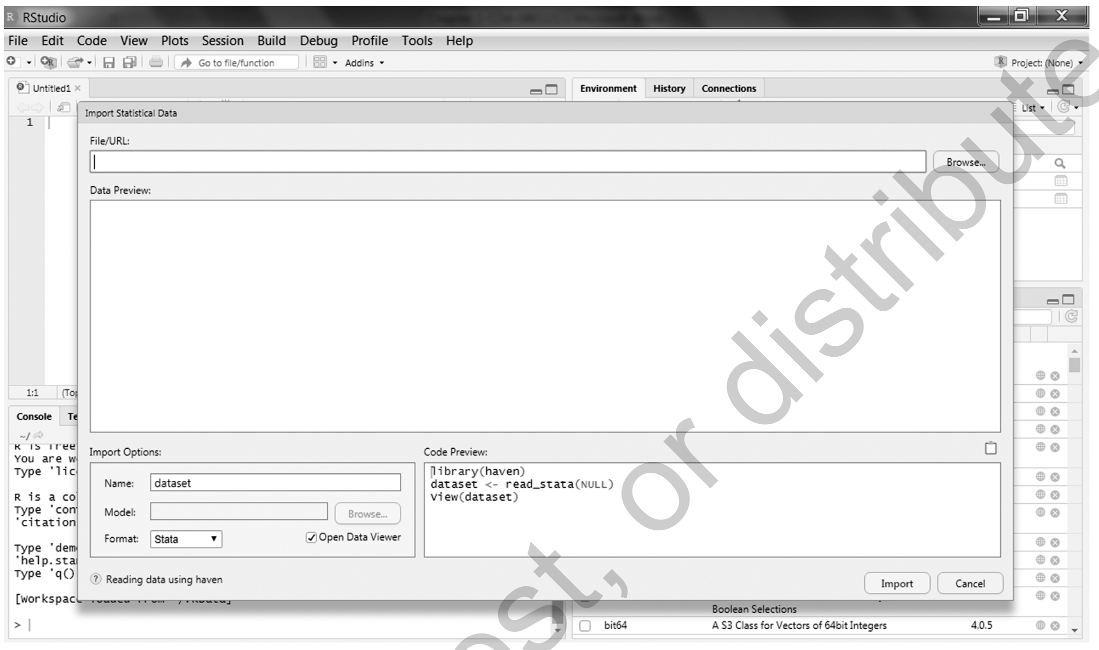
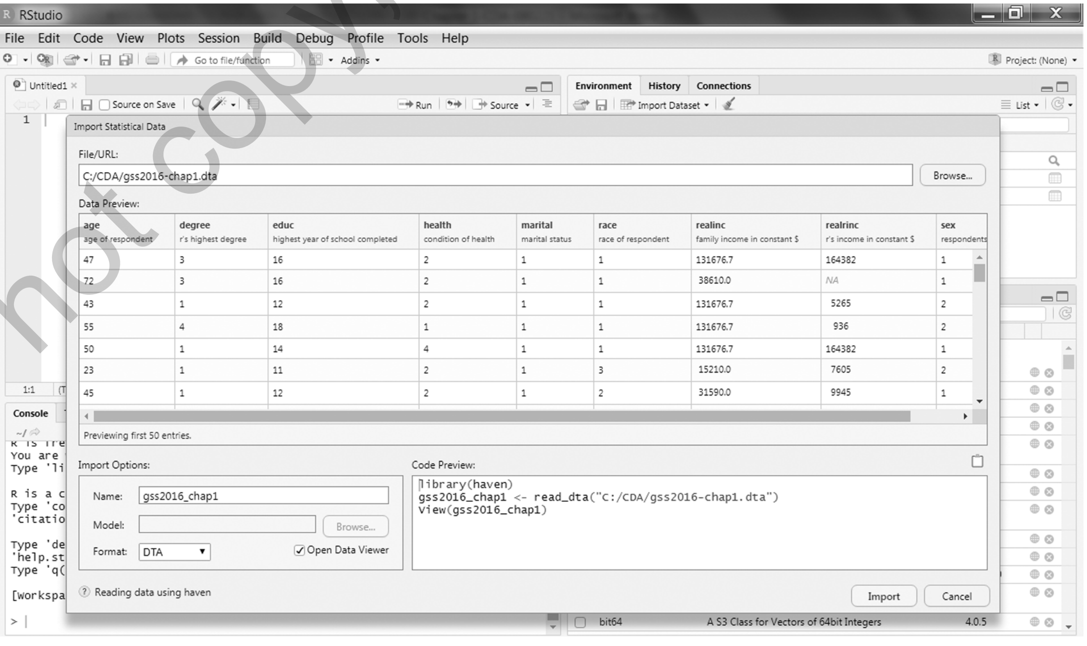


FIGURE 1.5 ● Screenshot of Opening a Dataset via RStudio



Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

- Third, once previewing the dataset and clicking on **Import**, you will see the imported dataset shown in the upper left panel and the R command echoed in the lower left R console.

1.1.10 How to Save R Output

When you execute a command in the R Console window, R will display the output following the command. There are several ways to save the output. Three ways are introduced as follows. First, a simple way is to copy and paste it. You can copy the selected output as text and paste it into a Word document or a text file. If you paste it into a Word document, then you may specify the font as `Courier New` and set the font size to 9 or smaller to show the results properly. Second, we can use the `sink()` function to save the results in a plain text format. This function needs to be used before we conduct statistical analysis, so we can save all of the results at the beginning of your R session. For example, the `sink("filename.txt")` command saves the output to a file named `filename.txt` in the current working directory. If you are uncertain of your working directory, type `getwd()` in the **Command** window. You can use the `setwd(" ")` function with the path in the quotation marks to change the working directory. Please note that output will not be shown in the R console when we use the `sink()` function. Third, you may use the `knitr` package (Xie, 2015, 2021) to save the R code, the notes, the output, and graphs in one document which can be a Word document, a pdf file, a presentation, or a web page. This package is useful for reproducible research since the R code, the analysis, and the reporting are well documented and easily replicated.

1.1.11 What If I Have a Question? How Do I Get Help?

We have many questions when we use R. Thankfully, R provides rich resources for users in different ways.

First, R itself provides help with the documentation of packages and functions. You can use the `help()` function to display the documentation for a package or a function. For example, the `help(lm)` command provides the documentation for linear regression models. A shortcut is to type `?` followed by the name of a package or function. For example, using `?lm` is the same as using `help(lm)` to obtain the documentation for linear regression models.

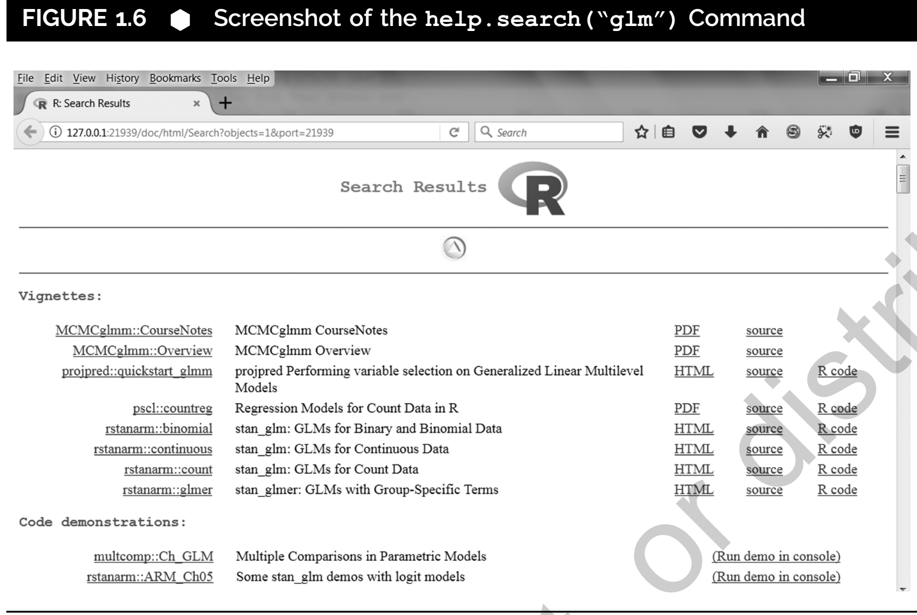
Second, R has online help and search facilities, which provide a wide range of help. If you have a question for a particular command, you can either click on **Help** from the menu or type `help.search()` with the function name enclosed in quotation marks within the parentheses or type `??` followed by the function name. For example, if you are looking for the help files for the `glm()` function for generalized linear models, you could type `help.search("glm")`. The screenshot shown in Figure 1.6 shows the help page for the `glm()` function.

The same screenshot can be reached if you use the pull-down menu. Go to **Help** on the menu, click on **Search help**, and then enter `glm` in the box. You can see the search results after clicking on **OK**.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 1.6 ● Screenshot of the `help.search("glm")` Command



Third, if you would like to get a more detailed introduction to a package, use the `help(package = " ")` command with the package function name enclosed in quotation marks. To look for help files for a function in a package, include both the function name and the package name in parentheses following the `help()` function. For example, `help(vglm, package = "VGAM")` command displays the documentation for the `vglm()` function.

Fourth, to search the web for help, use the `RSiteSearch(" ")` function with the key word enclosed in quotation marks. This function provides a keyword search.

Fifth, go to http://www.r-project.org/ to access the R manuals. R has provided complete PDF documentation. Users can also have access to these manuals directly from the menu or the installation folder within R.

Finally, you can get help from the following two websites:

- R seek: Search engine for R topics at https://rseek.org
- https://stackoverflow.com provides a general discussion forum for R users.

1.2 R DATA STRUCTURES: VECTORS, MATRICES, DATA FRAMES, AND LISTS

R includes several commonly used data structures, such as vectors, factors, matrices, data frames, and lists. We briefly discuss each data structure or data type as follows.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

1.2.1 Vector

Numeric Vector

A vector is a sequence of data elements of the same type. There are several different types of vectors, including numeric, character, complex, and logical vectors. In this section, we introduce two basic types of vectors, numeric and character vectors, which are a list of numeric values and characters or strings, respectively. We use the `c()` function, the concatenation function, to combine a list of numeric values or characters. For example, we would like to create a vector for a numeric variable named `age` using the function. The commands are as follows.

```
age <- c(47, 72, 43, 55, 50, 23, 45, 71, 86, 33)

age
```

In the first command, the `c()` function is used to combine the ages of 10 people and the created vector is assigned to an object named `age`. The second command is the vector name `age`. By typing the vector name, we get all the values of the vector. The output is as follows.

```
> # Use c() to create vectors
> age <- c(47, 72, 43, 55, 50, 23, 45, 71, 86, 33)
> age
  [1] 47 72 43 55 50 23 45 71 86 33
```

In the output, the first line shows the `c()` function with 10 values within the parentheses and the assigned object name `age`. The second line shows the vector name `age`. The third line shows the 10 values for `age`. `[1]` means that this line begins with the first value.

Character Vector

We can also use the `c()` function to combine a list of characters or strings for a character vector. For example, we want to create a vector for a character variable named `gender` by using the `c()` function. The characters are placed in quotation marks in the command as follows.

```
gender <-
c("male", "male", "female", "female", "male", "female", "male", "male", "female",
"female")
```

```
> gender <-
c("male", "male", "female", "female", "male", "female", "male", "male", "female", "female")
```

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

The command gender lists all the characters of the vector. The output is as follows.

```
> gender
 [1] "male"   "male"   "female" "female" "male"   "female" "male"   "male"
 [9] "female" "female"
```

In the output, [1] means that this first line begins with the first character value and [9] means that the second line begins with the ninth value.

Indexing a Vector

We can use indexing to access an individual element of a vector. Following the vector name, we use square brackets to place the position number. For example, the age[5] command references the fifth value of the age vector, which is 50. The output is as follows.

```
> # Index a vector
> age[5]
[1] 50
```

In the second example, the gender[3] command references the third value of the gender vector, which is "female." The output is as follows.

```
> gender[3]
[1] "female"
```

Factor

A factor is a vector for categories or levels. In other words, a factor or a factor variable is a categorical variable with multiple levels. It includes categories or levels which are internally coded as integer values with labels for the corresponding categories. For example, the factor variable health has four categories with labels, including 1 = poor health, 2 = fair health, 3 = good health, and 4 = excellent health. We use the factor() function to create a factor or categorical variable. In this example, we use the following command.

```
health <- factor(health, levels = c(1, 2, 3, 4), labels = c("poor health", "fair
health", "good health", "excellent health"))
```

In the command, health is the variable name, levels = c(1, 2, 3, 4) specifies the four categories, and labels = c("poor health", "fair health", "good health", "excellent health") specifies the labels with the characters enclosed in the quotation marks. The levels = argument is optional and can be omitted.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

A factor can be either a nominal or ordinal categorical variable. It is an ordinal variable when the categories are ordered. With the `ordered = TRUE` argument, we create an ordered factor as follows.

```
health <- factor(health, levels = c(1, 2, 3, 4), labels = c("poor health", "fair
health", "good health", "excellent health" ), ordered = TRUE)
```

1.2.2 Matrix

A matrix in R is a data structure with columns and rows in a two-dimensional rectangular layout with the same data type. Therefore, we cannot mix numeric and character data in a matrix. We use the `matrix()` function to create a matrix. For example, we use the `m1 <- matrix(1:8, nrow = 4, ncol = 2)` command to create the following 4 by 2 matrix which includes 4 rows and 2 columns. In the command, `1:8` represents the numbers from 1 to 8, which is a shortcut for the vector including the numbers from 1 to 8; `nrow = 4` specifies that the row number is 4; and `ncol = 2` specifies that the column number is 2. This matrix will be filled by columns since the `byrow=` argument is not specified. The resulting matrix is assigned to an object named `m1`.

```
> # Use matrix() to create matrices
> m1 <- matrix(1:8, nrow = 4, ncol = 2)
> m1
        [,1]    [,2]
[1,]       1       5
[2,]       2       6
[3,]       3       7
[4,]       4       8
```

If we would like the matrix to be filled by rows, we use the `byrow = TRUE` argument in the command. By running the `m2 <- matrix(1:8, nrow = 4, ncol = 2, byrow = TRUE )` command, we get the following output.

```
> m2 <- matrix(1:8, nrow = 4, ncol = 2, byrow = TRUE)
> m2
        [,1]    [,2]
[1,]       1       2
[2,]       3       4
[3,]       5       6
[4,]       7       8
```

Indexing a Matrix

We also use square brackets to reference elements in matrices. The row and column indices are enclosed in square brackets and are separated by a comma like this: [row, column]. For example, we use the `m1[2, 1]` command to access the element for the second row and the first column in the `m1` matrix. The output is as follows.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

```
> # Index a matrix
> m1[2, 1]
[1] 2
```

The m1[, 2] command accesses all the elements for the second column in the m1 matrix. When the row index is not specified, all the rows in the matrix are selected. The output is as follows.

```
> m1[ , 2]
[1] 5 6 7 8
```

When the column index is not specified, all the columns are selected. The m1[4,] command accesses all the elements for the fourth row in the m1 matrix.

```
> m1[4, ]
[1] 4 8
```

1.2.3 Data Frame

As introduced in the previous section, a data frame in R is a rectangular-shaped dataset with variables in columns and observations in rows. It is also referred to as a dataset in other statistical packages. A data frame can include different types of variables or vectors, such as numeric variables, character variables, and factor variables. We use the data.frame() function to create data frames. When we work on research projects, in most situations we work on data frames.

In the following example, we use the gss <- data.frame(age, gender) command to create a data frame named gss. In the command, the two variables age and gender are separated by a comma.

```
> # Use data.frame() to create a data frame
> age <- c(47, 72, 43, 55, 50, 23, 45, 71, 86, 33)
> gender <-
c("male", "male", "female", "female", "male", "female", "male", "male", "female", "female")
> gss <- data.frame(age, gender)
> gss
     age   gender
1     47     male
2     72     male
3     43   female
4     55   female
5     50     male
6     23   female
7     45     male
8     71     male
9     86   female
10    33   female
```

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

Several functions such as `str()`, `dim()`, and `names()` are useful to inspect a data frame. In this example, we use the `str()` function to display the structure of the `gss` dataset. The output shows that there are 10 observations and two variables in the dataset. The names and types of the two variables are shown next.

```
> str(gss)
'data.frame': 10 obs. of 2 variables:
$ age : num 47 72 43 55 50 23 45 71 86 33
$ gender: Factor w/ 2 levels "female","male": 2 2 1 1 2 1 2 2 1 1
```

We can also use the `dim()` function to display the dimension of the dataset.

```
> dim(gss)
[1] 10 2
```

The output also shows that there are 10 observations and two variables in the dataset. We use the `names()` function to display the names of the variables in the dataset.

```
> names(gss)
[1] "age"    "gender"
```

Accessing a Variable in a Data Frame

After we create or import a data frame, variables in the data frame are not automatically available to us for analysis. We can use the dollar sign (`$`) to access a variable from a data frame by placing `$` between the name of the data frame and the variable of interest. For example, to access `age` in the `gss` data frame, we use `gss$age`. The output is as follows.

```
> # Reference a variable in a data frame
> gss$age
  [1] 47 72 43 55 50 23 45 71 86 33
```

Useful Functions for Descriptive Statistics

We can use several useful functions to conduct basic descriptive statistics. These functions include the `mean()` function, the `sd()` function, the `min()` function, the `max()` function, and the `length()` function to obtain the mean, standard deviation, maximum, minimum, and number of observations of a variable, respectively. In the following example, we use these functions to compute descriptive statistics for `age`.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

```
> mean(gss$age)
[1] 52.5
> sd(gss$age)
[1] 19.10352
> min(gss$age)
[1] 23
> max(gss$age)
[1] 86
> length(gss$age)
[1] 10
```

To access a variable in a data frame, we can also use the `attach()` function to attach the data frame. In this way, we do not need to repeatedly type the name of the data frame followed by the `$` operator. If we attach more than one dataset with the same variable name, the variable in the first dataset will be masked or replaced by the second one or the newest one with the same name. Therefore, once completing the analysis on one dataset, it is a good practice to use the `detach()` function to detach it from the R session.

1.2.4 List

A list is simply a combination of objects. We can combine a vector, a matrix, and a data frame into a list. We use the `list()` function to create a list. In the following example, we use the `list.gss <- list(age, m1, gss)` command to create a list named `list.gss` with three components. In the command, there are three components separated by commas: `age` is the vector, `m1` is the matrix, and `gss` is the data frame. The output is as follows.

```
> # Use list() to create a list
> list.gss <- list(age, m1, gss)
> list.gss
[[1]]
 [1] 47 72 43 55 50 23 45 71 86 33


[[2]]
       [,1]    [,2]
[1,]      1       5
[2,]      2       6
[3,]      3       7
[4,]      4       8

[[3]]
     age   gender
1     47     male
2     72     male
3     43   female
4     55   female
5     50     male
6     23   female
7     45     male
8     71     male
9     86   female
10    33   female
```

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

We can add the names for the three components in the list. The output is as follows.

```
> list.gss2 <- list(age = age, matrix1 = m1, gssdata = gss)
> list.gss2
$`age`
  [1] 47 72 43 55 50 23 45 71 86 33


$matrix1
        [,1]    [,2]
[1,]       1       5
[2,]       2       6
[3,]       3       7
[4,]       4       8

$gssdata
      age    gender
1      47      male
2      72      male
3      43    female
4      55    female
5      50      male
6      23    female
7      45      male
8      71      male
9      86    female
10     33    female
```

1.3 DATA MANAGEMENT

Before conducting data analysis, we need to work on various data management tasks to make the data ready for analysis. This section introduces some commonly used functions for basic data management tasks, such as selecting variables and observations, creating a new variable, recoding a variable, dummy coding, reverse coding, labeling a variable, labeling values for a factor variable, dealing with missing values, combining data, reshaping data, and converting data types. We also briefly introduce the `dplyr` (Wickham et al., 2021) and `sjmisc` packages (Lüdecke, 2018a) and the pipe operator `%>%` for data management.

1.3.1 Selecting Variables

To select variables from a dataset, we first use the indexing method with square brackets following the name of the dataset. The basic syntax is `dataset[, c("var1", "var2", "var3")]`. In the command, we use the `c()` function to select variables with the variable names in quotation marks. In the square brackets, rows are not specified since all rows are selected.

The second method is to use the `subset()` function to select variables. The basic syntax of the function is `subset(dataset, select = c(var1, var2,`

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

var3)). In the command, we first specify the data frame named dataset. Then we use the select = c(var1, var2, var3) argument to select variables of interest.

The third method is to the use the select() function in the dplyr package. You need to install the package first by typing install.packages("dplyr") and then load it with the library(dplyr) function. The basic syntax is select(dataset, var1, var2, var3). In the command, we first specify the dataset and then specify the three variables which are separated by commas. The quotation marks are not needed for each variable. Examples of using the dplyr package are introduced in Section 1.4.

1.3.2 Selecting Observations

To select observations from a dataset, we again use the indexing method with square brackets. For example, to select the first 100 observations in a dataset, we use the generic command, dataset[c(1:100),]. In the command, we use the c() function to select observations from 1 to 100. In the square brackets, following the comma, columns or variables are not specified since all variables are selected. We can also specify conditions using the indexing method. For example, in the dataset [age > 50 & gender = = "female",] command, we select the observations for age larger than 50 and select observations for females. The & sign means that both conditions are met. The double equal sign "==" is different from the single equal sign "=". The former sign is one of the logical operators.

Another method to select observations is to use the subset() function. The generic structure of the function is subset(dataset, age > 50 & gender = = "female"). In the command, we first specify the dataset. Then we use the age > 50 & gender = = "female" argument to select observations.

The third method is to the use the filter() function in the dplyr package. You need to install the package first by typing install.packages("dplyr") and then load it with the library(dplyr) function. The syntax is filter (dataset, age > 50 & gender = = "female").

1.3.3 Selecting Observations and Variables

To select observations and variables simultaneously from a dataset, we use the same three methods introduced above. For example, with the indexing method, the dataset[age > 50 & gender = = "female", c("var1", "var2", "var3")] command selects the observations for age larger than 50 and observations for females for the three variables. The same results can be obtained using the subset(dataset, age > 50 & gender = = "female", select = c(var1, var2, var3)) command. In the command, we first specify the data frame, and then specify the conditions for selecting observations, and finally specify the selected variables.

If we use the dplyr package, we need to first use filter(dataset, age > 50 & gender = = "female") to select observations and then use the select(dataset, var1, var2, var3) command to select the three variables.

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

Do not copy, post, or distribute

1.3.4 Creating a New Variable

When you work on a research project, you often need to create new variables, recode an existing variable to a new variable, or combine several variables into one variable. There are also situations when you need to label a variable or label values for a categorical variable. In this section, we will briefly introduce commands to fulfill these tasks.

The assignment operator, `<-` , is used to create a new variable.

```
var2 <- var1
```

This command creates a variable `var2`, which is the same as the original variable `var1`.

```
var3 <- var1 + var2
```

This command creates a variable `var3`, which is the sum of the two variables `var1` and `var2`.

Another method is to the use the `mutate()` function in the `dplyr` package. If we would like to create a variable which is a sum of the two other variables, the basic syntax is `mutate(dataset, var3 = var1 + var2)`. In the command, the first argument is `dataset` and the new variable `var3` equals the sum of `var1` and `var2`.

Now, let us see an example using the General Social Survey 2016 (GSS 2016) dataset. I would like to create a new variable `realinc1`, which is the variable `realinc/10000`. We first load the `foreign` package, import the data with the `read.dta()` function, and then attach the dataset. The `realinc1 <- realinc/10000` command is used to create the new variable `realinc1`.

```
> # Remove all objects
> rm(list = ls(all = TRUE))
# Import the Stata data with the foreign package
> library(foreign)
> chp1 <- read.dta("C:/CDA/gss2016-chap1.dta")
> attach(chp1)
> # Create a new variable
> realinc1 <- realinc/10000
```

The second method is to use the `mutate()` function in the `dplyr` package. We first load the package and then use the `chp1.n <- mutate(chp1, realinc2 = realinc/10000)` command. In this command, `chp1` is the dataset and `realinc2` is the new variable, which equals `realinc/10000`.

```
> # Create a new variable using mutate()
> library(dplyr)
> chp1.n <- mutate(chp1, realinc2 = realinc/10000)
```

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

Do not copy, post, or distribute

1.3.5 Recoding a Variable

There are several ways to recode a variable. Three methods are introduced in this section. For example, we would like to create a new categorical variable SES, according to the family income realinc. The first method is to use the indexing method with conditions within square brackets ([ ]). We enter the following five commands:

```
> # Recode a variable
> chp1$SES <- NA
> chp1$SES[chp1$realinc >= 0 & chp1$realinc <= 11114] <- 1
> chp1$SES[chp1$realinc  >= 11115 & chp1$realinc <= 25739] <- 2
> chp1$SES[chp1$realinc  >= 25740 & chp1$realinc <= 38609] <- 3
> chp1$SES[chp1$realinc >= 38610] <- 4
> chp1$SES <- factor(chp1$SES)
```

The first command creates a new variable SES with NA for the missing values. The object chp1$SES refers to the variable SES in the dataset chp1. The second command creates the level 1 for SES if the value of realinc is equal to or greater than 0 and less than 11,114. The third command creates the level 2 for SES if the value of realinc is equal to or greater than 11,115 and less than 25,739. The fourth command creates the level 3 for SES if the value of realinc is equal to or greater than 25,740 and less than 38,609. The fifth command creates the level 4 for SES if the value of realinc is equal to or greater than 38,610.

We then use the factor() function to convert SES into a factor variable with the command chp1$SES <- factor(chp1$SES). The table() function is used to display the frequencies for the four categories.

```
> # Create a factor
> chp1$SES <- factor(chp1$SES)
> table(chp1$SES)
  1     2     3     4
394   439   307   565
```

To convert SES to an ordered factor variable with labels, we use the command chp1$SES <- factor(chp1$SES, labels = c("low SES", "low-middle SES", "upper-middle SES", "high SES"), ordered = TRUE). In the command, labels = c("low SES", "low-middle SES", "upper-middle SES", "high SES") specifies the labels and ordered = TRUE specifies that the variable is ordinal. The table() function is used to display the frequencies for the four categories.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

```
> # Create a factor with labels
> chp1$SES <- factor(chp1$SES, labels = c("low SES", "low-middle SES", "upper-middle
SES", "high SES"), ordered = TRUE)
> table(chp1$SES)

          low SES      low-middle SES      upper-middle SES      high SES
              394                 439                   307           565
```

The second method is to use the `recode()` function in the `car` package (Fox & Weisberg, 2019). You need to install the package first by typing `install.packages("car")` and then load it by using the `library(car)` function. Since the `car` package also contains a function with the same name as that in the `dplyr` package, we use `car::recode` to access the `recode()` function in the `car` package.

```
> # Recode a variable using recode() in the car package
> install.packages("car")
> library(car)
Loading required package: carData


Attaching package: 'car'


> chp1$SES2 <- car::recode(chp1$realinc, "0:11114 = 1; 11115:25739 = 2; 25740:
38609 = 3; 38610:hi = 4; else = NA")
```

In this command, `chp1$realinc` is the original variable `realinc` in the dataset `chp1` and `"0:11114 = 1; 11115:25739 = 2; 25740:38609 = 3; 38610:hi = 4; else = NA"` specifies how the ranges of the original values are coded to the new values. The recode specifications are separated by semicolons (;) and the argument is placed in the quotation marks. In the specifications, `hi` indicates the highest value and `else = NA` specifies the missing values. The recoded variable is named `chp1$SES2` which means `SES2` in the dataset `chp1`.

We again use the `factor()` function to convert `SES2` into a factor variable. The `table()` function is used to display the frequencies for the four categories.

```
> chp1$SES2 <- factor(chp1$SES2)
> table(chp1$SES2)


  1    2    3    4
394  439  307  565
```

We also convert `SES2` to an ordered factor variable with labels and use the `table()` function to display the frequencies for the four categories.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

```
> chp1$SES2 <- factor(chp1$SES, labels = c("low SES", "low-middle SES", "upper-
middle SES", "high SES"), ordered = TRUE)
> table(chp1$SES2)


        low SES     low-middle SES     upper-middle SES     high SES
            394                439                  307          565
```

The third method is to use the rec() function in the sjmisc package. You need to install the package first by typing install.packages("sjmisc") and then load it by using the library(sjmisc) command. In the rec() function, the first argument can be either a data frame or a variable. If the first argument is a data frame, then the rec() function creates a new data frame; if the first argument is a variable, then the function creates a new variable. In the following two examples, we first demonstrate how to recode a variable and create a new variable. We also demonstrate how to recode that variable and then create a new data frame.

In the following command, chp1$realinc is the original variable realinc in the dataset chp1, so the function will create a new variable. The rec = "0:11114 = 1; 11115: 25739 = 2; 25740:38609 = 3; 38610:hi = 4; else = NA" argument specifies how the ranges of the original values are coded to the new values with the rec= argument. Just like the recode() function in the car package, in the rec() function, the recode specifications are separated by semicolons (;) and the argument is placed in quotation marks. In the specifications, hi indicates the highest value and else = NA specifies the missing values. The recoded variable is named chp1$SES3 which means SES2 in the dataset chp1. The table() function displays the frequencies, and the results are the same as those using the first two methods.

```
> # Recode a variable using rec () in the sjmisc package: method 1
> # Install sjmisc by using install.packages("sjmisc")
> library(sjmisc)
> chp1$SES3 <- rec(chp1$realinc, rec = "0:11114 = 1; 11115: 25739 = 2; 25740:38609
= 3; 38610:hi = 4; else = NA")
> table(chp1$SES3)


  1     2     3     4
394   439   307   565
```

In the next command, with everything else the same, the first argument is the data frame chp1 and the second argument is the variable, so the function will create a new data frame. This data frame named chp1.re contains the recoded variable realinc_r only. The new variable name realinc_r is automatically assigned and is the combination of the original name realinc and the suffix _r. We use the add_columns() function in the sjmisc package to add the new data frame to the original

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

data frame. In the `chp1 <- add_columns(chp1.re, chp1, replace = FALSE)` command, `chp1.re` is the new data frame and `chp1` is the original data frame. The `replace = FALSE` argument specifies that both data frames are kept. The combined dataset is named `chp1`. The `table()` function displays the same frequencies as above.

```
> # Recode a variable using rec() in the sjmisc package: method 2
> chp1.re <- rec(chp1, realinc, rec = "0:11114 = 1; 11115:
  25739 = 2; 25740:38609 = 3; 38610:hi = 4; else = NA", append = FALSE)
> chp1 <- add_columns(chp1.re, chp1, replace = FALSE)
> table(chp1.re$realinc_r)


  1    2    3    4
394  439  307  565
```

1.3.6 Creating a Dummy or Binary Variable

We can also use the `rec()` function to create a dummy or binary variable. For example, let us create a variable `education` with a value of 1 for respondents' highest year of school completed greater than 13.79 years, and a value of 0 otherwise. The `table()` function is used to display the frequencies of the two categories.

```
> # Create a binary or dummy variable: method 1
> chp1$education <- rec(chp1$educ, rec = "min:13.79=0; 14:hi=1; else = NA")
> table(chp1$education)


  0    1
929  944
```

Another method to create a dummy variable is to use the `dicho()` function in the `sjmisc` package. In the `chp1$education2 <- dicho(chp1$educ, dich.by = 13.79, append = TRUE)` command, `chp1$educ` is the variable, the `dich.by = 13.79` argument splits the variable into two groups by the value of 13.79, and the `append = TRUE` argument adds the new variable to the data frame. The `table()` function displays the same frequencies as those above.

```
> # Create a binary or dummy variable: method 2 using dicho() in sjmisc
> chp1$education2 <- dicho(chp1$educ, dich.by = 13.79, append = TRUE)
> table(chp1$education2)


  0    1
929  944
```

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

1.3.7 Reverse Coding with rec()

You can use the rec() function to reverse categories of a variable. For example, a survey item uses a four-point scale of 1–4, with 1 = excellent and 4 = poor. You want to reverse the order and define poor to be 1 and strongly excellent to be 4. In the following example, we first recode the character labels to the numeric labels with excellent = 1 and poor = 4. The table() function displays the frequencies.

```
> # Reverse coding with rec()
> table(health)
health
excellent      good      fair      poor      dk      iap      na
      414       914       427       118       0        0       0
> chp1$health.n <- rec(chp1$health, rec = "excellent=1; good=2; fair=3; poor=4;
else = NA")
> table(chp1$health.n)


  1      2      3      4
414    914    427    118
```

For simplicity, we can also use the rec = "rev" argument in the rec() function for reverse coding. In the example, with the rec = "rev" argument, we reverse the order of the categories of the variable named health.rev. The table() function displays the frequencies. As we can see, the categories are reversed in the frequency table.

```
> chp1$health.rev <- rec(chp1$health.n, rec = "rev")
> table(chp1$health.rev)


  1      2      3      4
118    427    914    414
```

1.3.8 Labeling Values for Factor Variables

We can use the factor() function to recode a numeric variable to a factor variable. As introduced above, we can also use the rec() function in the sjmisc package to recode the values to the categories and then label them. The purpose of labeling values is to define the numeric values of a categorical variable. This will make your analysis easier and interpretation clearer. Once you label values of a categorical variable, the label will appear in your output when you conduct an analysis. In the following example, we use the str() function to display the structure of the recoded variable health.rev.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

```
> str(chp1$health.rev)
  num [1:1873]  3 3 3 4 1 3 3 3 1 4 ...
```

The output shows that `health.rev` is numeric with 1,873 observations. We can use the `factor()` function to convert the numeric variable to a factor with labels. In the command, `chp1$health.rev` refers to the variable `health.rev` in the dataset, `chp1`, `levels = c(1, 2, 3, 4)` specifies the values of the four categories, and `labels = c("poor", "fair", "good", "excellent")` specifies the labels. The resulting output by the `table()` function displays the frequencies of the four categories with labels.

```
> # Label a factor
> chp1$health.rev <- factor(chp1$health.rev, levels = c(1, 2, 3, 4), labels =
c("poor", "fair", "good", "excellent"))
> table(chp1$health.rev)


        poor      fair      good   excellent
         118       427       914         414
```

1.3.9 Labeling a Variable

Labeling shows the meaning of a variable. To label a variable, the first method is to use the `label()` function in the `Hmisc` package. You need to install the package first by typing `install.packages("Hmisc")` and then load it using the `library (Hmisc)` function. The basic syntax is `label(varname) <- "label text"`.

For example, if you want to label a variable `health.rev` with the text `recoded health status`, type the following command.

```
label(health.rev) <- "recoded health status"
```

1.3.10 The `row_sums()` and `row_means()` Functions in the `sjmisc` Package

We can use the `row_sums()` function in `sjmisc` package to create a new variable, which is a summation of several items in a survey. For example, if we would like to create a sum score for five variables, we use the following command:

```
row_sums(dataset, var1:var5, n = 5)
```

In the command, `dataset` is the name of the data frame, `var1:var5` specifies the five variables, and `n=5` specifies the minimum number of non-missing values per row. In this example, we specify `n = 5` since there are no missing values in these five

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

variables. This command creates a variable with a total row score of var1, var2, var3, var4, and var5.

We can also use the row_means() function in the sjmisc package to create a variable with a row mean score of a set of variables. The basic syntax is row_means(dataset, var1:var5, n=5).

1.3.11 How to Deal With Missing Values When Recoding Variables

Missing data need to be coded as NA so R can recognize them correctly. If datasets from other software packages are imported into R, missing data need to be coded correctly since the datasets may include user-defined missing values, such as 999, na, and iap. We can use the rec() function in the sjmisc package to recode these user-defined missing values to NA.

If we want to exclude missing values in a data frame, we can use the no.omit() function with the name of the data frame enclosed in parentheses and create a new data frame like this: data.new <- no.omit(dataname).

1.3.12 Other Useful Data Management Functions

The following commands will be briefly introduced, but examples using real data will be omitted here due to space limitations:

1. Combining data

   - The rbind() function can be used to add cases to the existing variables. When we have two datasets containing the same variables with the same variable types, this command can be applied to combine different cases into one dataset.

   - The cbind() can be used to add variables from two datasets without a common unique identification variable. If the two datasets have a common identification variable, we can use the merge() function. The generic structure of the merge() function is merge(data1, data2, by = "ID", all = TRUE, sort = TRUE). In the command, data1 and data2 are the two datasets, by = "ID" specifies the sorting ID, all = TRUE keeps the unmatched cases from both datasets, and sort = TRUE specifies that the ID variable should be sorted.

2. Reshaping data
   The reshape package (Wickham, 2007) is useful when we reorganize data into different forms. You need to install the package first by typing install.packages("reshape") and then load it with the library(reshape) function. The melt() function reorganizes the dataset in the long form, whereas the cast() function transforms the dataset in the wide form. For example, in longitudinal data analysis, a person-level

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

dataset is in the wide form, a multivariate layout with one record per individual; on the other hand, a person-period dataset is in the long form with multiple records for each individual, representing each time-point for data collection.

3. Converting variable types
   A variable can be coded in either a numeric or a string format. A numeric variable deals with numbers, whereas a character or a string variable contains text data. The `as.numeric()` function can be used to convert a character variable into a numeric variable, whereas the `as.character()` function can be used to convert a numeric variable into a character variable. The `factor()` function can be used to convert a numeric variable or a character variable into a factor variable.

1.4 DATA MANAGEMENT WITH THE `tidyverse` AND `sjmisc` PACKAGES

The `tidyverse` (Wickham et al., 2019) is not one package, but a collection of several packages that make importing, cleaning, exploring, managing, and visualizing data in R much easier. It includes the following major packages:

- `readr` for importing datasets
- `tibble` for creating data frames
- `dplyr` for data management or data munging
- `tidyr` for creating tidy data
- `ggplot2` for visualizing data
- `purr` for functional programming
- `stringr` for string management
- `forcat` for handling factors

You can use the `install.packages("tidyverse")` command to install all the packages above and then use the `library(tidyverse)` command to load them. In this chapter, we mainly focus on the `dplyr` and `ggplot2` packages in `tidyverse`.

In the previous section, we discussed several useful functions in the `dplyr` and `sjmisc` packages for data management. Specifically, in the `dplyr` package, we introduced the `filter()` function for selecting observations, the `select()` function for selecting variables by their names, the `mutate()` function for creating new variables; in the `sjmisc` package, we introduced the `rec()` function for recoding variables, the `add_columns()` function for adding created variables to a data frame, and the `dicho()` function for creating dummy variables; we also introduced how to

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

use the row_sums() function and row_means() function to create a sum score and a mean score for a set of variables, respectively.

In the dplyr package (Wickham et al., 2021), there are other useful functions. For example, we can also use the arrange() function to sort a variable or a set of variables in a data frame, use the group_by() function to group data by a factor or categorical variable, and use the summarize() function to summarize data. Further, we can use the pipe operator %>% to follow each function which you would like to pipe. The pipe %>% means "then" when we execute these functions sequentially. In other words, you run a function and then conduct the second one by specify %>% between these two functions or steps. You continue to run functions as a chain with %>% until you complete the work. The strength of using pipes is to execute a series of functions in sequence without creating temporary data frames. A summary of the major functions in the dplyr package is as follows.

- filter() function for selecting observations
- select() function for selecting variables
- mutate() function for creating new variables
- arrange() function for sorting variables
- group_by() function for grouping data by a factor variable
- summarize() function for summarizing data

Please note that the summarize() function can be also written as summarise(). In addition, the pipe operator %>% is a part of the magrittr package, which is automatically loaded when you load the dplyr package or tidyverse.

In the sjmisc package (Lüdecke, 2018a), there are other useful functions for data management, which are summarized below.

- rec() function for recoding variables
- add_columns() function for adding variables as columns to a data frame
- dicho() function for creating dummy variables
- row_sums() function for creating a sum score for a set of variables
- row_means() function for creating a mean score for a set of variables
- to_factor() function for recoding a variable into a factor variable with labels
- var_rename() function for renaming variables
- to_long() function for transforming a data frame from wide format to long format

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

- `descr()` function for descriptive statistics
- `frq()` function for creating a frequency table for factor variables

Since all the functions in the sjmisc package work together with the dplyr package, using both packages makes the data management tasks easier. In addition, the pipe operator %>% works with both packages, which makes the commands more concise.

In the first example, we would like to select the two variables, sex and age, from the data frame chp1. We then would like to split the data by the grouping variable sex and conduct descriptive statistics of age by the two groups of sex. We use the select() and the group_by() function in the dplyr package and the descr() function in the sjmisc package with pipes. This single-line chp1 %>% select(sex,age) %>% group_by(sex) %>% descr(age) command tells R to run it as a chain. It reads "load dataset chp1, then select sex and age, then group the data by sex, and then describe age." The output is as follows.

```
> # Data management using the dplyr and sjmisc packages with the pipe operator %>%
> library(dplyr)
> library(sjmisc)
> chp1 %>% select(sex,age) %>% group_by(sex) %>% descr(age)

## Basic descriptive statistics

Grouped by: male

 var    type  label    n NA.prc  mean    sd    se   md trimmed     range  skew
 age integer    age  831      0 48.62 17.08  0.59   48   48.09 71 (18-89) 0.18

Grouped by: female

 var    type label     n NA.prc   mean    sd    se   md trimmed     range skew
 age integer    age 1042      0  51.02 17.67 0.55   52   50.62 71 (18-89) 0.12
```

In the second example, we would like to select the two variables, sex and health, from the data frame chp1. We then also would like to split the data by the grouping variable sex and create a frequency table of health grouped by sex. We use the select() and the group_by() function in the dplyr package and the frq() function in the sjmisc package with pipes. The chp1 %>% select(sex, health) %>% group_by(sex) %>% frq(health) command tells R to load the dataset, select sex and age, group the data by sex, and then create the frequency table for health. The output is as follows.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

```
> chp1 %>% select(sex,health) %>% group_by(sex) %>% frq(health)

health <categorical>
# grouped by: male
# total N=831  valid N=831  mean=2.15  sd=0.83

      Value    |     N    |    Raw %    |    Valid %    |    Cum. %
--------------------------------------------------------------------
excellent      |   180    |    21.66    |     21.66     |     21.66
good           |   402    |    48.38    |     48.38     |     70.04
fair           |   197    |    23.71    |     23.71     |     93.74
poor           |    52    |     6.26    |      6.26     |    100.00
Dk             |     0    |     0.00    |      0.00     |    100.00
Iap            |     0    |     0.00    |      0.00     |    100.00
Na             |     0    |     0.00    |      0.00     |    100.00
<NA>           |     0    |     0.00    |      <NA>     |      <NA>


health <categorical>
# grouped by: female
# total N=1042  valid N=1042  mean=2.12  sd=0.83

Value        |     N    |    Raw %    |    Valid %    |    Cum. %
------------------------------------------------------------------
excellent    |   234    |    22.46    |     22.46     |     22.46
good         |   512    |    49.14    |     49.14     |     71.59
fair         |   230    |    22.07    |     22.07     |     93.67
poor         |    66    |     6.33    |      6.33     |    100.00
Dk           |     0    |     0.00    |      0.00     |    100.00
Iap          |     0    |     0.00    |      0.00     |    100.00
Na           |     0    |     0.00    |      0.00     |    100.00
<NA>         |     0    |     0.00    |      <NA>     |      <NA>
```

1.5 GRAPHS

R is powerful in drawing various graphs. In this section, some basic functions in R will be introduced. We will focus on the `hist()` function for histograms, the `barplot()` function for bar charts, the `boxplot()` function for box plots, and the `plot()` function for scatterplots. In addition, the `ggplot2` package will be introduced.

For each type of graph, R offers rich options, which may seem complicated. You can save the commands into a script file so that the graphs can be easily reproduced or edited.

In the following sections, I will show you how to create basic graphs, such as histograms, bar charts, box plots, and scatterplots.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

1.5.1 Histograms

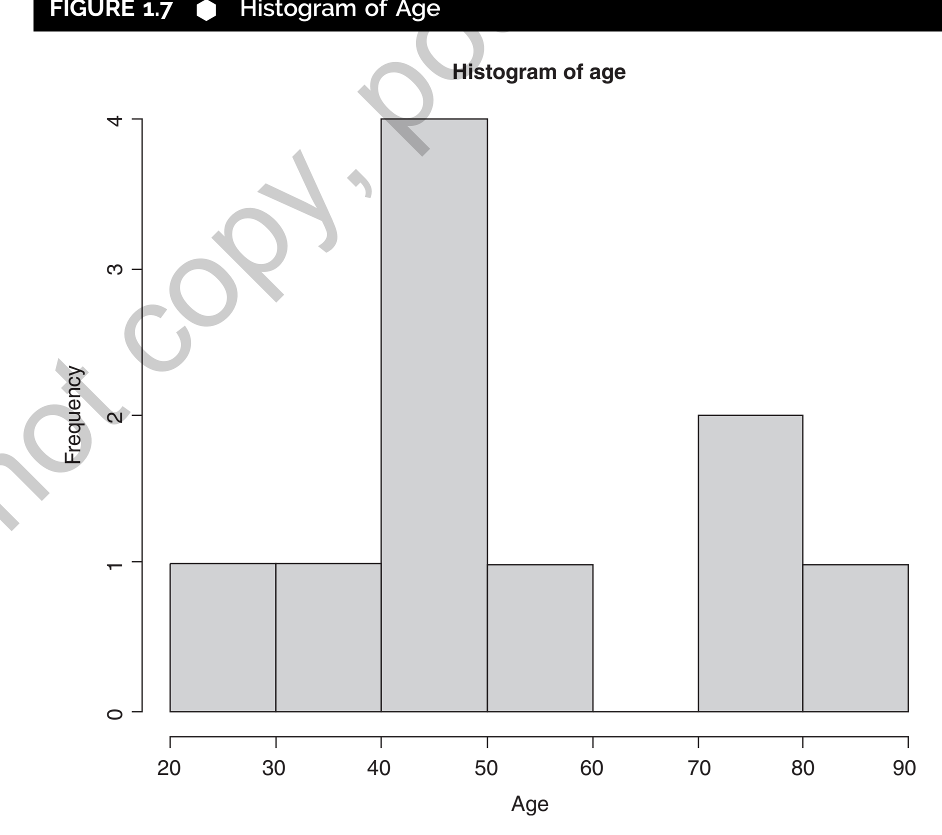
The histogram is one of the most frequently used graphs, and is used to present in a visual manner a frequency distribution of data. In a histogram, scores appear on a horizontal scale and frequency counts are displayed on a vertical scale. Histograms are normally used for continuous variables. They can also be used for ordinal variables if their underlying traits are continuous.

The function for histograms is `hist()`. For example, to see the distribution of the variable `age` using the `gss` data file, simply enter the `hist(age)` command. The histogram is shown in Figure 1.7.

```
> # Create a histogram
> hist(age)
```

We can customize the graph by adding more arguments or parameters to the `hist()` function. For example, we can use the `main=` argument to add a title for the graph, use the `xlab` and `ylab` arguments to provide labels for the x axis and y axis, respectively, use the `xlim` and `ylim` arguments to specify ranges of the axes, and use the `col=` argument to choose colors. We can also use the `breaks=`

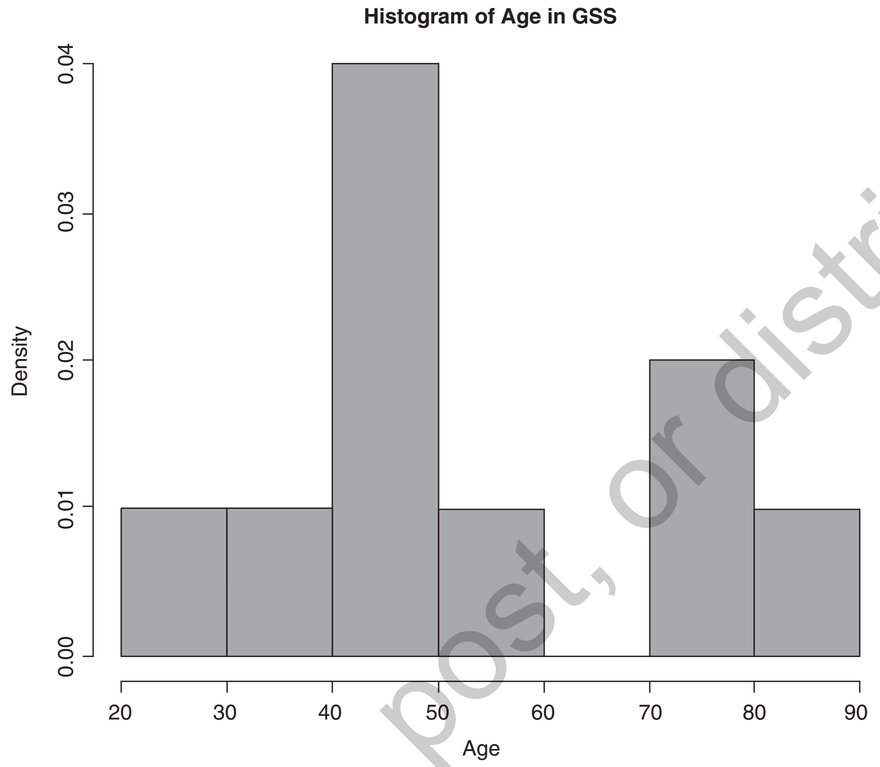
FIGURE 1.7 ● Histogram of Age



Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 1.8 ⬢ Histogram of Age in GSS



argument to specify the number of cells and use the `freq = FALSE` argument to request the percentages instead of the frequencies. The following is a command with customized arguments: `hist(age, main = "Histogram of Age in GSS", xlab = "Age", xlim = c(20,90), col = "lightblue", freq = FALSE)`. The updated histogram is shown in Figure 1.8.

```
> # Create a histogram with customized arguments
> hist(age, main="Histogram of Age in GSS",
+   xlab="Age", xlim=c(20,90), col="lightblue", freq=FALSE)
```

1.5.2 Bar Charts

Bar charts are normally used to display frequencies for categorical variables. The `barplot()` function is used for bar charts. For example, to draw a bar chart for a categorical variable `gender`, type the following command: `barplot(table(gender))`. In the command, we use the `table()` function to get the frequencies since `gender` is a factor or categorical variable. We will then see the following output and the graph shown in Figure 1.9.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 1.9 Bar Chart of Gender

```
> # Create a bar plot
> barplot(table(gender))
```

We can also customize the bar chart by using the `main=` argument to add a title for the graph, using the `xlab=` and `ylab=` arguments to provide labels for x axis and y axis, respectively, using the `names.arg=` argument to name bars, and using `col=` to specify colors.

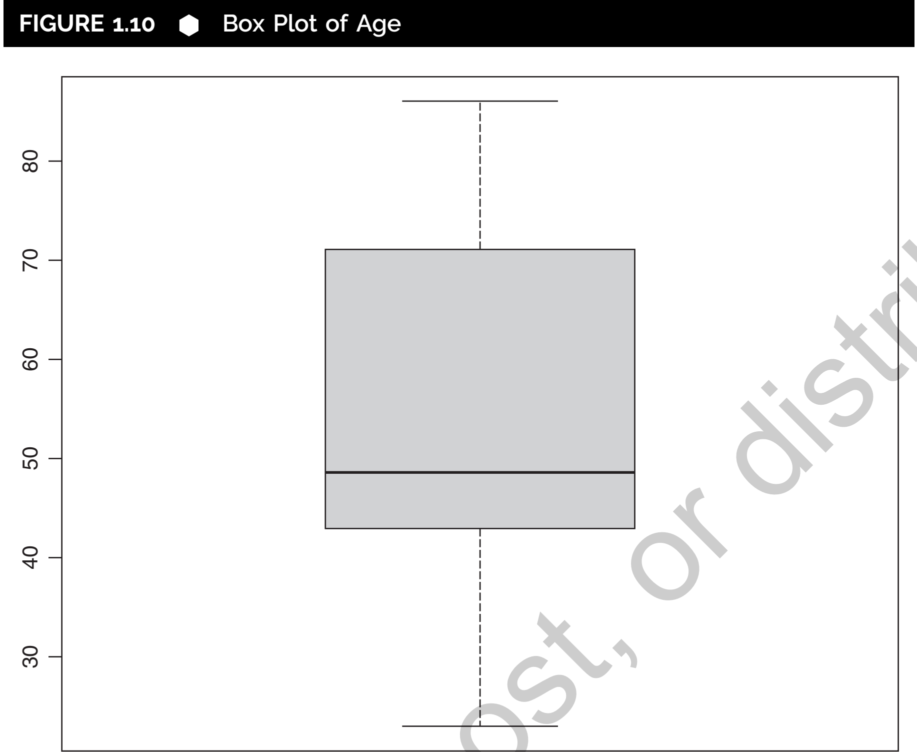
1.5.3 Box Plots

Box plots are useful for displaying a distribution and to identify outliers for a continuous variable. They display the 25th and 75th percentiles, median, whiskers, and outliers. In a box plot, the lower and upper ends of the box indicate the 25th and 75th percentiles, respectively. The width of the box indicates the interquartile range. The horizontal line in the box is the median, which is the 50th percentile. The lines below and above the box are whiskers, which indicate the spread of your data. Observations beyond the whiskers are shown as dots, which are outliers.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 1.10 ● Box Plot of Age



We use the `boxplot(age)` command to draw a box plot for `age` (Figure 1.10).

```
> # Create a box plot
> boxplot(age)
```

We can also customize the box plot by using the `main=` argument to add a title, using the `xlab=` and `ylab=` arguments to provide labels for the x axis and y axis, respectively, using `col=` to specify colors, and using the `horizontal = TRUE` argument to make the graph horizontally.

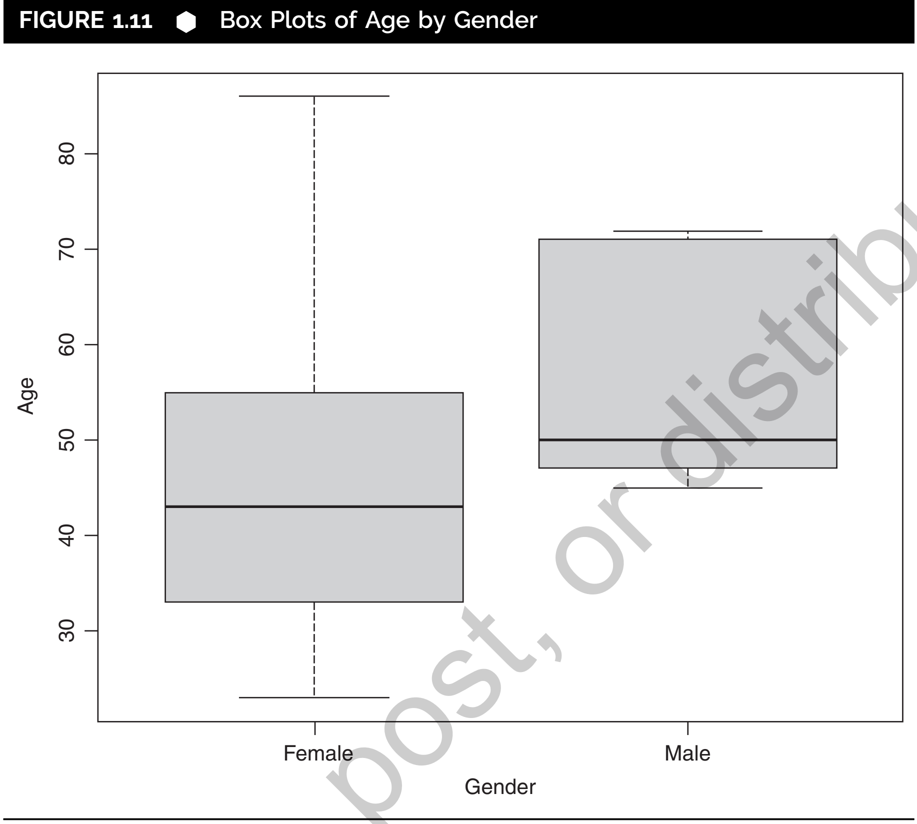
To display several box plots in one graph, we use the `boxplot()` function with the model equation for the variables enclosed in parentheses. For example, the `boxplot(age ~ gender)` command produces the boxplots for `age` by the grouping variable `gender` (Figure 1.11).

```
> # Create a box plot grouped by gender
> boxplot(age ~ gender)
```

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 1.11 Box Plots of Age by Gender



Do not copy, post, or distribute

1.5.4 Scatterplots

Scatterplots are used to show a relationship between two variables. They are two-dimensional graphs, with the x axis displaying values of one variable and the y axis displaying values for another variable.

To see a scatterplot of two variables `var1` and `var2`, type the following command:

```
plot(var1, var2)
```

This command tells R to draw a two-way scatterplot for `var1` and `var2`.

For example, we would like to draw a scatterplot for the pre- and post-math tests. We first use the `c()` function to create two variables and then use the `data.frame()` function to create a data frame named `math`. After attaching the dataset, we use the `plot()` function to draw the graph.

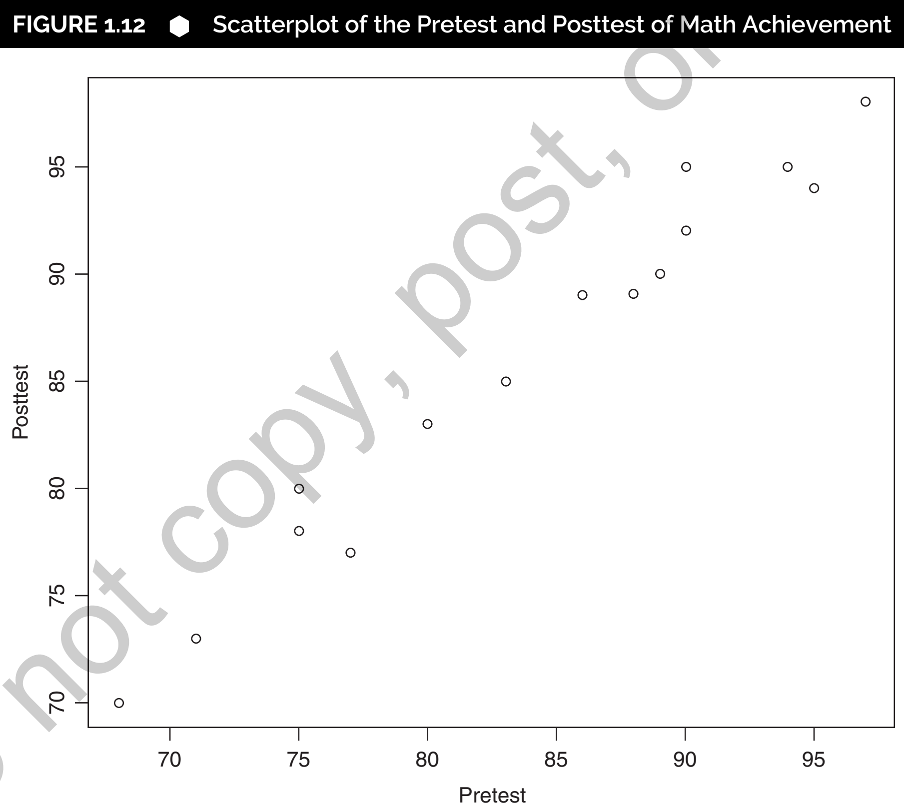
Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

```
> # Create a data frame for a scatterplot
> pretest <- c(80, 90, 68, 88, 75, 71, 83, 75, 95, 86, 77, 90, 97, 94, 89)
> posttest <- c(83, 95, 70, 89, 80, 73, 85, 78, 94, 89, 77, 92, 98, 95, 90)
> math <- data.frame(pretest,posttest)
> # Create a scatterplot
> plot(pretest, posttest)
```

The graph is displayed in Figure 1.12.

We can also customize the box plot by using the `main=` argument to add a title, using the `sub=` argument to add a subtitle, using the `xlab=` and `ylab=` arguments to provide labels for x axis and y axis, respectively, and using `col=` to specify colors.

FIGURE 1.12 ● Scatterplot of the Pretest and Posttest of Math Achievement



Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

We can also use the scatterplot(var1, var2) function in the car package to draw scatterplots. An example is not provided here.

1.5.5 Scatterplots with ggplots2

The ggplot2 package (Wickham, 2016), developed by Hadley Wickham, is a popular package for R graphing with a unified framework. It is powerful in providing a variety of types of graphs. It is flexible since you have a variety of options for customizing color, height, size, and shape of graphs. You can use the install.packages("ggplot2") command to install the package and then use the library(ggplot2) command to load it. This package is a part of tidyverse, so if you have installed tidyverse, you do not need to reinstall ggplot2.

The ggplot() function in the ggplot2 package includes three major elements: data, aesthetics (aes), and geometry (geometric objects).

1. Data: we need to specify a data frame.
2. Aesthetics (aes): this argument specifies the x and y lines, colors, point size, and point shape.
3. Geoms: this argument specifies many types of graphs. For example, we can specify histogram (geom_histogram), line (geom_line), box plot (geom_boxplot), bar (geom_bar), point (geom_point), and many other types of graphs.

In the following example, we use the ggplot(math, aes(x=pretest, y=posttest)) + geom_point() command to draw the scatterplot for the two variables, pretest and posttest. In the command, math is the dataset, aes(x=pretest, y=posttest)) specifies the x and y variables, and geom_point() specifies that the type of graph is a scatterplot with the point geom. The R code is shown below, and the graph is displayed in Figure 1.13.

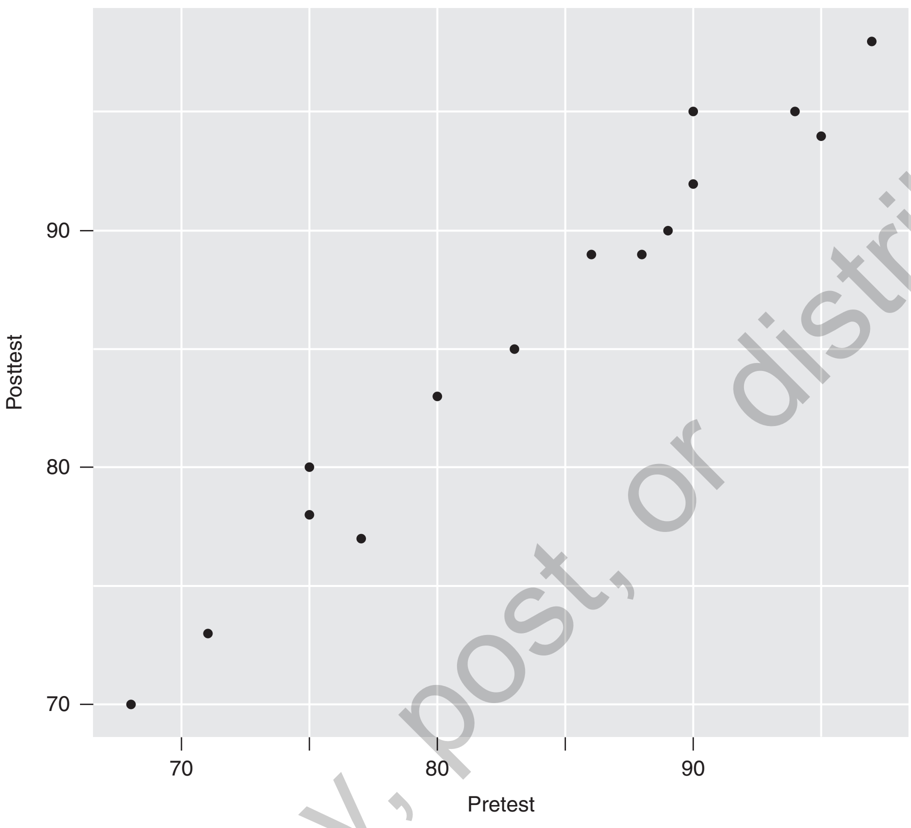
```
> # Create a scatterplot using ggplot() in ggplot2
> library(ggplot2)
> ggplot(math, aes(x=pretest, y=posttest)) + geom_point()
```

In addition to the point plot, we can also make a line plot with geom_line(). Further, to add a regression line to the graph, we use the ggplot(math, aes(x=pretest, y=posttest)) + geom_point() + stat_smooth(method=lm) command. In the command, the stat_smooth(method=lm) function specifies that the method of linear regression is used to fit the regression line. The stat_smooth()

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 1.13 ⬢ Scatterplot with the Point Goem in ggplot2



Do not copy, post, or distribute

function also produces a 95% confidence interval for the regression line. The R code is shown below, and the resulting graph is displayed in Figure 1.14.

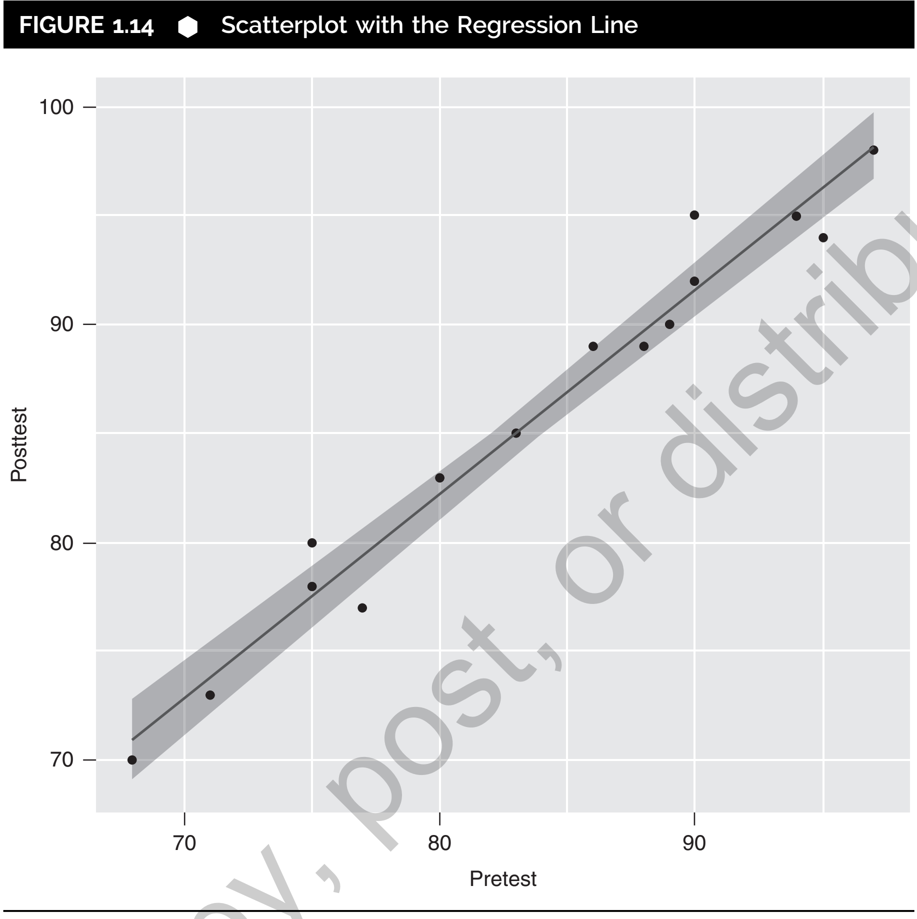
```
> ggplot(math, aes(x=pretest, y=posttest)) + geom_point() + stat_smooth(method=lm)
```

1.5.6 How to Save Graphs

R does not automatically save the graphs you create. To save your graph if you are running RStudio, you can either save it from the plot panel or enter the command. For example, if you would like to use the menu in the plot panel in RStudio, go to **Plots** and click on **Export**. You can choose to **Save as Image** or **Save as PDF**. Or, if you

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 1.14 ● Scatterplot with the Regression Line



would like to save a graph created by the `ggplot()` function, use the `ggsave()` function to save the file.

The created graph can be copied and pasted into a Word document, but this method would be tedious if you needed to save a lot of graphs all at one time.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

1.6 SUMMARY OF R COMMANDS IN THIS CHAPTER

```
# Chapter 1 R Script

# The following user-written packages need to be installed first by using install.packages(" ") and then
by loading it with library()
# library(dplyr)
# library(sjmisc)
# library(car)
# library(ggplot2)

# Section 1.2
# Use c() to create vectors
age <- c(47, 72, 43, 55, 50, 23, 45, 71, 86, 33)
age
gender <-
c("male", "male", "female", "female", "male", "female", "male", "male", "female", "female")
gender
# Index a vector
age[5]
gender[3]

# Use matrix() to create matrices
m1 <- matrix(1:8, nrow=4, ncol=2)
m1
m2 <- matrix(1:8, nrow=4, ncol=2, byrow=TRUE)
m2
# Index a matrix
m1[2, 1]
m1[4, ]
m1[ ,2]

# Use data.frame() to create a data frame
gss <- data.frame(age, gender)
gss
str(gss)
dim(gss)
names(gss)

# Reference a variable in a data frame
gss$age

# Several functions for descriptive statistics
mean(gss$age)
sd(gss$age)
min(gss$age)
max(gss$age)
length(gss$age)

# Use list() to create a list
list.gss <- list(age, m1, gss)
list.gss
list.gss2 <- list(age=age, matrix1=m1, gssdata=gss)
list.gss2
```

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

```
# Remove all objects
rm(list = ls(all=TRUE))

# Section 1.3
library(foreign)
chp1 <- read.dta("C:/CDA/gss2016-chap1.dta")
attach(chp1)
str(chp1)

# Create a new variable
realinc1 <- realinc/10000
# Create a new variable using mutate()
library(dplyr)
chp1.n <- mutate(chp1, realinc2=realinc/10000)
mean(realinc1, na.rm=TRUE)
mean(chp1.n$realinc2, na.rm=TRUE)

# Recode a variable
chp1$SES <- NA
chp1$SES[chp1$realinc >=0 & chp1$realinc <= 11114] <- 1
chp1$SES[chp1$realinc  >= 11115 & chp1$realinc <= 25739] <- 2
chp1$SES[chp1$realinc  >= 25740 & chp1$realinc <= 38609] <- 3
chp1$SES[chp1$realinc >= 38610] <- 4

# Create a factor
chp1$SES <- factor(chp1$SES)
table(chp1$SES)

# Create a factor with labels
chp1$SES <- factor(chp1$SES, labels = c("low SES", "low-middle SES", "upper-middle SES", "high SES"),
ordered = TRUE)
table(chp1$SES)

# Recode a variable using recode() in the car package
# install.packages("car")
library(car)
chp1$SES2 <- car::recode(chp1$realinc, "0:11114 = 1; 11115: 25739 = 2; 25740:38609 = 3; 38610:hi = 4;
else = NA")
chp1$SES2 <- factor(chp1$SES2)
chp1$SES2 <- factor(chp1$SES, labels = c("low SES", "low-middle SES", "upper-middle SES", "high SES"),
ordered = TRUE)
table(chp1$SES2)

# Recode a variable using rec () in the sjmisc package: method 1
library(sjmisc)
chp1$SES3 <- rec(chp1$realinc, rec = "0:11114 = 1; 11115: 25739 = 2; 25740:38609 = 3; 38610:hi = 4;
else = NA")
table(chp1$SES3)

# Recode a variable using rec () in the sjmisc package: method 2
chp1.re <- rec(chp1, realinc, rec = "0:11114 = 1; 11115:25739 = 2; 25740:38609 = 3; 38610:hi = 4; else
= NA", append = FALSE)
chp1 <- add_columns(chp1.re, chp1, replace = FALSE)
table(chp1.re$realinc_r)

# Create a binary or dummy variable: method 1
chp1$education <- rec(chp1$educ, rec = "min:13.79=0; 14:hi=1; else = NA")
table(chp1$education)
```

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

```
# Create a binary or dummy variable: method 2 using dicho() in sjmisc
chp1$education2 <- dicho(chp1$educ, dich.by = 13.79, append = TRUE)
table(chp1$education2)

# Reverse coding with rec()
table(health)
chp1$health.n <- rec(chp1$health, rec = "excellent=1; good=2; fair=3; poor=4; else = NA")
table(chp1$health.n)
chp1$health.rev <- rec(chp1$health.n, rec = "rev")
table(chp1$health.rev)
str(chp1$health.rev)

# Label a factor
chp1$health.rev <- factor(chp1$health.rev, levels=c(1, 2, 3, 4), labels = c("poor", "fair", "good",
"excellent"))
table(chp1$health.rev)
table(chp1$sex)

# Section 1.4
# Data management using the dplyr and sjmisc packages with the pipe operator %>%
library(dplyr)
library(sjmisc)
chp1 %>% select(sex,age) %>% group_by(sex) %>% descr(age)
chp1 %>% select(sex,health) %>% group_by(sex) %>% frq(health)

# Section 1.5
age <- c(47, 72, 43, 55, 50, 23, 45, 71, 86, 33)
gender <-
c("male", "male", "female", "female", "male", "female", "male", "male", "female", "female")
gss <- data.frame(age, gender)

# Create a histogram
hist(age)
# Create a histogram with customized arguments
hist(age, main="Histogram of Age in GSS",
    xlab="Age", xlim=c(20,90), col="lightblue", freq=FALSE)

# Create a bar plot
barplot(table(gender))

# Create a box plot
boxplot(age)
# Create a box plot grouped by gender
boxplot(age ~ gender)

# Create a data frame for a scatterplot
pretest <- c(80, 90, 68, 88, 75, 71, 83, 75, 95, 86, 77, 90, 97, 94, 89)
posttest <- c(83, 95, 70, 89, 80, 73, 85, 78, 94, 89, 77, 92, 98, 95, 90)
math <- data.frame(pretest,posttest)

# Create a scatterplot
plot(pretest, posttest)
# Create a scatterplot using ggplot() in ggplot2
library(ggplot2)
ggplot(math, aes(x=pretest, y=posttest)) + geom_point()
ggplot(math, aes(x=pretest, y=posttest)) + geom_line()
ggplot(math, aes(x=pretest, y=posttest)) + geom_point() + stat_smooth(method=lm)
```

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

Glossary

Data frame A rectangular-shaped dataset in R with variables in columns and observations in rows, also referred to as a dataset in other statistical packages.

Factor A categorical variable or a factor variable with multiple levels.

Functions A set of instructions to perform a specific task in R, just like commands or procedures in other statistical packages.

`ggplot2` An add-on package for R graphing with a unified framework, also a part of `tidyverse`.

Matrix A data structure in R with columns and rows in a two-dimensional rectangular layout with the same data type.

R A programming language and a general-purpose statistical tool for data management, data analysis, and graphing.

R Commander (i.e., `Rcmdr`) An add-on package providing a graphic user interface (GUI) system for R.

R script file A text file containing a list of R commands.

RStudio A free, open-source integrated development environment (IDE) for R that makes programming easier.

`sjmisc` An add-on package for data transformation and management.

`tidyverse` A collection of several add-on packages that make importing, cleaning, exploring, managing, and visualizing data in R easier.

Vector A sequence of data elements of the same type.

Exercises

Use the GSS 2016 data available at **https://edge.sagepub.com/liu1e** for the following problems.

1. Find the variable `happy` and recode it to a new variable `happyrev`. Recode the values of 1, 2, and 3 in `happy` into the values of 3, 2, and 1, respectively, for the new variable.
2. Label the new variable `happyrev`, `happiness`, and then label its values 1 "not too happy", 2 "pretty happy", and 3 "very happy".
3. Produce a histogram for `educ`.
4. Draw a scatterplot to explore the relationship between `educ` and `coninc`.

Do not copy, post, or distribute

Copyright ©2023 by SAGE Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.