1 THE JOURNEY STARTS WITH R

LEARNING OBJECTIVES

1.1 Understand the fundamental structure and functionality of the R software environment.

1.2 Gain knowledge about the basics of data structures in R, such as vector, matrix, data frame, and list.

1.3 Appreciate the role and benefits of R packages in statistical computation and data analysis.

1.4 Understand the utility of CRAN as the primary source for R packages and the processes involved in their publishing and maintenance.

1.5 Grasp the concept of CRAN task views, especially the Spatial task view, and their role in organizing and identifying relevant R packages.

1.6 Recognize the significance and functions of various data types and data structures in R.

1.7 Appreciate the importance of the "learning by doing" approach in mastering R and its packages.

1.1 WHAT IS R, AND WHY SHOULD WE USE R?

Right about twenty-two years (2002) ago when I was still a very green PhD student at the University of Wisconsin—Milwaukee, I was facing a rather serious problem. I had my eyes on performing spatial analysis with my regional development dataset. In particular, I wanted to calculate the spatial autocorrelation index (the Moran's *I*, as many of you might have already guessed) with my county-level per capital GDP data. I tried everything that my then skill set could muster, including manually inputting the spatial weight matrix on an Excel sheet and performing matrix operation within Excel. It did get the work done, but I was very unhappy about it. Not the result, which turned out to be what I wanted, but the procedure. I was not happy that to calculate one statistical index, I had to spend an entire day just inputting data. Frankly speaking, I was quite desperate when I did the manual input but also quite depressed after I got the result. As you can imagine, what if the definition of the spatial weight matrix changed? Do I have to take another day to input the matrix again?

I did not believe that this was the only plausible way to perform such a task. Mind you, I did not think it was a simple task—far from it. I thought it a wonderful task that could explain whether data collected over geographical units are self-correlated. The excitement for a trained geographer was purely thrilling. However, the way to get results seemed too unintuitive. The next natural thing to do was a Google search (not as advanced as it is today but more focused). I learned I could do similar functions in SPSS, or I could purchase a software package that costed $450, or—on the first or second page of the search—use a package called `spdep` in a platform called R to do the calculations.

I had never heard of R until that moment. As a graduate student, I could hardly afford to purchase a package, so I decided to give the package `spdep` and platform R a try—it was free, after all, and if I was not successful, I was not losing much (this from a guy who spent a day inputting 1s and 0s into an Excel sheet).

This was when an entirely new world, and a wonderful one, opened its door to me. For about the next week and half, I indulged in the world of R, testing everything that was written on the freely available and incredibly detailed manual *An Introduction to R*, until my mind was full of R symbols and matrix operations and could not take it anymore. As much as I might doubt it, I started to be able to comprehend the simple and elegant way R executes its command and routines. More importantly, with the help from the author of the package `spdep`, Dr. Roger Bivand (huge thanks to Dr. Bivand here), I was able to generate weight matrix, calculate Moran's index, test the significance of Moran's index under different distribution assumptions, calculate local Moran's index, generate Moran's scatterplot, produce the LISA maps, and finally, start to comprehend and utilize spatial regression models for my own research and dissertation with the `spdep` package.

This was truly a dream come true for a desperate doctoral student who had pretty much all the data available and all the methods laid out, but who was only waiting for a way to implement them. R and the package `spdep` made everything possible, and a doctoral dissertation that utilized spatial analysis with regional development datasets was the result. This was only the start of my journey, though.

Following that initial enlightenment, I started to engage more and more into the working mechanisms of R and how to best use R and its ever-increasing package repository to solve my problems. I will never claim to be an R expert but remain an avid R-user. And henceforth, I have learned how to delve into *R*s amazingly simple and elegant language structure to decipher the math, mechanism, and algorithm of a particular function that I might be interested in. Under the guidance and teaching of Dr. Roger Bivand, I finished a few functions for the later R package "`spgwr`." Through the effective communications in R-sig-geo discussion forums, I learned how to debug and customize functions in R to suit my needs. As a trained urban geographer interested in computer programming but never trained for it, I find it amazingly effective (though not necessarily efficient) to use R and customize R to work the way I want. R is a perfect place for social scientists who are interested in computer programing—but not necessarily trained for it—to satisfy their needs for data analysis, including simple tasks of correlation analysis to more sophisticated ones, such as generalized additive models, spatially varying coefficient process models, and the like.

This book is written for social scientists like me, who are not afraid of exploring the wonderland of R and commanding R to work the way that they want. So, back to the question: What is R, and why should you use R? My take from my experience over almost two decades follows:

1. R is a programming language.

Without going all the way back to how R was born and how it is related to the statistical computing language S/S+, suffice it to say that R is a fully functional programing language. It follows set rules when implementing your computing and analyzing algorithms. It works well with commonly defined data structure and data types. Since I am not a programmer myself, I will not delve too much into the field that I am not most familiar with. Tutorials and learning materials for programing in R are abundant—either for free or for a fee. From a user's perspective, I often feel compelled to code in R when I have a statistical computing idea in my mind that I know the mathematics behind and which I can lay out the algorithm for in a clear step-by-step way. I am not, however, claiming to be a trained programmer and actively seeking to code in R. R is very accommodating for self-trained half-cooked programmers like me so we can accomplish our projects. This feature of R is especially useful to the majority of social scientists who need a convenient tool to accomplish their needs without deviating too much into the ocean of programming.

2. R is also a COMPLETE bundle of statistical analysis software.

During my teaching and researching career, when data analysis has come into play for any of my projects, I have experienced using quite a series of statistical analysis software packages, ranging from SPSS®, SAS®, MATLAB®, S+®, STATA®, and quite a few other packages. These are wonderful packages for common statistical analyses and satisfy most of the needs for data analyses. The problem with these packages, however, is that they satisfy "most" of the needs. If there is a function that is not embedded into the original distribution of the software, a researcher has either to write an extension or wait for the vendor to release an update.

Writing in a specific software package requires the researcher to not only know how to use the software (most social scientists are trained to use the software) but also how to code in the software environment (which is often a daunting task for quite a few scholars). Moreover, even when a scholars has invested time and money (we will come back to this point later) to learn how to code in one software package, it often takes an enormous amount of effort (if not impossible) for other scholars who are not familiar with the software package to use it. This prevents effective communication among scholars and could lead to repetitive implementations of certain algorithms in different packages.

R, on the other hand, provides a unifying platform for any scholars who are interested in implementing their algorithms to do so with ease. As of when this chapter was written, there are 19,391 packages (April 20, 2023) available to address almost all aspects of data analyses. However, the readers are cautioned not to take this number for its face value because it changes pretty much every few weeks, if not more frequently. By the time you read this book, you will see a much larger number and be amazed at how it grows. It might just be me or the cohort of scholars I work with, but every time I have a data analysis task in mind, my first action is to search the repository of R packages to see if there are packages that have already implemented the specific analytical approach or that provide similar functions so I can potentially modify it to suit my own needs. In the past 20 years, I have yet to find a time that R failed me. That is why I call R a "COMPLETE" statistical analysis software.

3. R is a code interpreter.

One of the things that interests me and encourages me to continue to learn and explore more about R is that you can usually get what you intended to do immediately from the

R interface. I feel this is critical for a first-time data analysts who has excellent domain knowledge and ideas but rather limited data analysis experiences. For instance, if you input in the R interface (the ">" symbol is the R interface prompt) 5 + 7, you immediately get the result 12 as in the following example:

```
5 + 7
[1] 12
```

While this is a simple example, this applies to more sophisticated operations, such as matrix manipulation, which is critical for statistical analysis. For instance, now I am defining two matrices, **A** and **B**, in R:

```
A <- matrix (seq (1,9), nrow = 3, ncol = 3)
B <- matrix (seq (1, 27, by = 3), nrow = 3, ncol = 3)
```

Both matrices are defined using the keyword "matrix" (R is case-sensitive, so please be mindful when writing in R) with several parameters. The first parameters "seq (1, 9)" and "seq (1, 27, by = 3)" provide data to be filled in the matrices. The parameters "nrow =" and "ncol =" are intuitive, indicating the "number of rows" and "number of columns" for the matrices. Here I defined two square matrices with the dimension of 3 by 3. If I want to see what A and B are, I can simply type the names in R and get the results instantly:

```
A
      [,1] [,2] [,3]
[1,]     1    4    7
[2,]     2    5    8
[3,]     3    6    9
B
      [,1] [,2] [,3]
[1,]     1   10   19
[2,]     4   13   22
[3,]     7   16   25
```

With these two matrices, we can perform all sorts of matrix manipulations in R directly. A few examples include the following:

Matrix multiplication:

```
A%*%B
      [,1] [,2] [,3]
[1,]    66  174  282
[2,]    78  213  348
[3,]    90  252  414
```

Element-wise multiplication:

```
A*B
     [,1] [,2] [,3]
[1,]    1   40  133
[2,]    8   65  176
[3,]   21   96  225
```

Matrix inversion (or not able to be inverted):

```
solve(A)
Error in solve.default(A) :
  Lapack routine dgesv: system is exactly singular: U[3,3] = 0
solve(B)
Error in solve.default(B) :
  system is computationally singular: reciprocal condition
number = 3.36431e-18
```

Both matrices here cannot be inverted by using the matrix inversion operator "solve()," because as the error messages report (as any seasoned programmer will tell you, the output error messages often contain nearly all the answers to your questions about "what goes wrong here?" Please read the error message carefully), these matrices are singular (meaning their determinants are exactly or computationally zero) and can be easily verified with the "det()" operator in R:

```
det(A)
[1] 0
det(B)
[1] 4.796163e-14 # this is basically the computationally
recognized 0.
```

As you can see so far, R as a code interpreter is intuitive, responsive, and as friendly as a machine language could be to let you know what you want to know or let you know why what you want to know cannot be produced. I spent much time during my first few months working with R simply enjoying the elegance of the program and how I could transform an idea into the results that I sought. To social scientists who might be shy from seemingly intimidating computer programs and "coding," R provides a fairly smooth start.

4. R is (most importantly) an open-source software platform.

Many might argue from a narrow perspective that there is no such thing as free lunch (from a broader perspective, that might still be true), since "being free" might suggests an "externality" that is not effective in any interactive system. Fortunately, this is not the case for R. I will not spend too much time engaging in the debate between what is better, the open-source software that does not have a traditional team of "tech support" behind it or the private, proprietary software that provides shining "tech support" 24/7. The lessons I learned from learning and applying and developing and teaching R, is that "being free" is not only more accessible to young minds that usually lack the necessary resources to afford for-fee software, but more importantly, way more productive since the user base and creative minds behind the software are many times that of any private, proprietary software packages. Now, let us journey to the point of getting R and really familiarize yourselves with R.

1.2 GETTING AND FAMILIARIZING YOURSELVES WITH R

Without giving you too much redundant information (I am quite sure you will be able to obtain the following information from any R-related book or a simple internet search), you can obtain the source code, the binary installation file, the packages, the manuals, and much useful information about R from this website: cran.r-project.org.

Depending on the operating system you are using, you will download the appropriate copy of R, and the installation to your computer shall be rather straightforward. I am a Windows user (probably the least tech savvy user of the three systems listed on the website), and installing R is as simple as downloading the installation file and clicking my way through. Because I am not an R expert, I will not discuss how to install R step by step. There are many free resources available online with easy access that provide instructions.

Once you've downloaded R, I recommend you get to know the command "`install.packages`:."

```
?install.packages
```

And then immediately install the helper package "`installr.`"

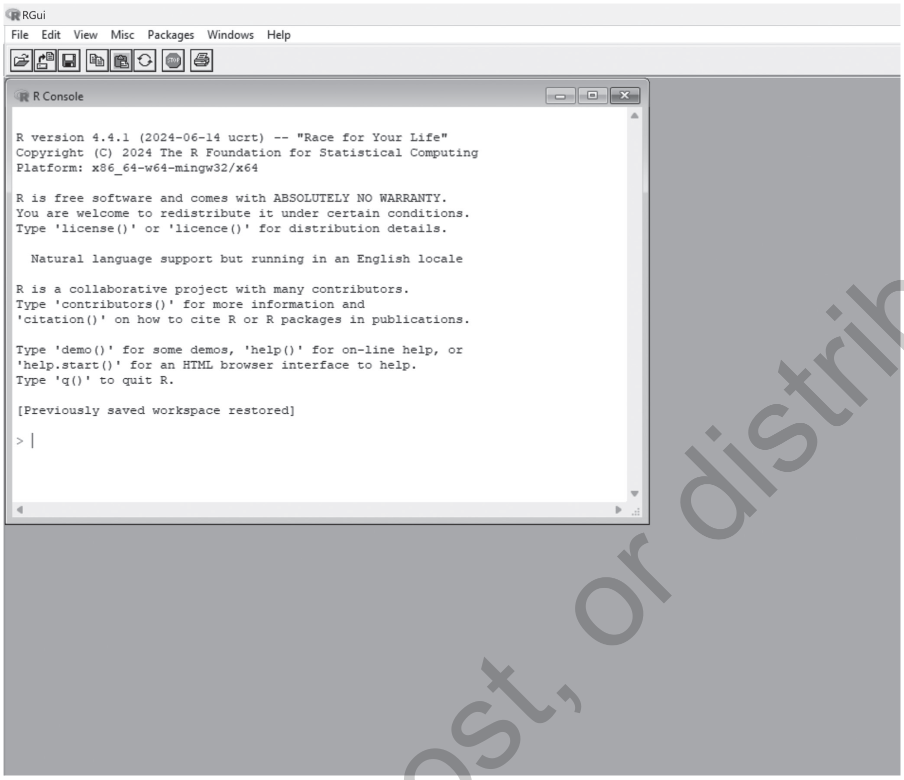
```
install.packages("installr", dependencies = T)
```

The "`dependencies = T`" argument is a useful one so that the related packages (dependencies) will get installed as well. When I am installing any package from the repository, I always indicate "`dependencies = T`." This helper package is useful when you need to update R from within the R interface instead of going to cran.r-project.org to download it again. This is convenient because R updates every two to three months. Such frequent updating will need some automatic manager so that you can focus your precious time on more brainpower-demanding tasks, such as exploring and analyzing your data.

Once you have downloaded and installed R, you can then run the interface; the image shown in Figure 1.1 is likely what you will see on your screen:

The screen is simple enough but contains essential information describing what R is, what license R is under, how to cite R properly in your own academic endeavors, and the `help()` and `help.start()` routines that can get you started. I recommend anyone (especially R new-comers but also veteran R users) try `help.start()` to get the core manuals, references, and other miscellaneous material. The `help()` function or similarly the "`?`" operator are probably the most important facilitators when using R.

FIGURE 1.1 ■ The Generic R Interface



```
R version 4.4.1 (2024-06-14 ucrt) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64

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

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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

[Previously saved workspace restored]

> |
```

The universe of R is vast, the functions are abundant, and the updates are frequent. Even if one is very familiar with using R, it is helpful to use the `help()` function and/or "`?`" operator to gain more detailed information on a particular package or function. The use of these two is fairly straightforward; you either include what you intended to find help with within the parentheses of `help()` or immediately after the "`?`" (one of the reasons that I tend to use "`?`" operator more than `help()`). For instance (and more often than not), if one attempts to use an analytical function to analyze the data but is not 100% sure which arguments are best for such or simply would like to try a variety of options, one can use `help(function name)` or "`?function name`" to gain the necessary information. The `help()` and "`?`" operator can also be used for getting more information about packages. One thing I need to caution you about is that if the author of the package did not write a help document or vignette about the package under the package's name, these two facilitators will not yield useful information. To obtain more information about packages, Google the packages' name or go directly to cran.r-project.org to find the packages (https://cran.r-project.org/web/packages/index.html).

Before I move on to introducing two useful companions of R, I would like to introduce two functions that will get any newcomers up to speed with R. The first is the `library()` function. This function is used to load installed packages. The power of R lies on its ever-growing pool of analytical packages that deal with practically all fields of statistical computation and analyses (if there are new methods that are not implemented in R, those methods might very well be implemented shortly, much faster than in any commercially licensed software packages). Apparently, it is not practical to load all the packages into R's working memory when R starts, nor is it efficient and needed. After all, one analytical task often will only involve a limited number of packages. Loading too many packages at the same time will likely clutter R's operation. Hence when R starts, it will only automatically load basic packages (the "`base`," "`stats`," "`graphics`," "`grDevices`," "`utlis`," "`datasets`," and "`methods`" that can be used for basic analysis and computation). If you attempt to analyze spatial data, the powerful spdep package and its relevant packages will need to be loaded into R's working memory using the `library()` function:

```
library(spdep)
```

This will automatically load spdep and any other packages that are related to `spdep` (the "`sp`," "`spData`," and "`sf`" packages) into R's working memory.

If a package is no longer needed, one can use the `detach` ("`package:packagename`") to unload it from the memory. Any related packages, however, will not be automatically unloaded; you will have to do this manually.

The second function is the `setwd()` function. This function sets up the working directory where all the input data and output results will be stored. For Windows users, other than using quotation marks to include the working directory, the separation marks between folders and subfolder must also be changed from "\" to either "/" or "\\." I have observed quite a few times in my career that data analysts are often eager to jump directly to analyzing their data and producing the results on screen, and when happy about them, saving the outputs into the default location of R (depending on which operating system one is using, the default location varies from simply the local document folder to some rather mysterious locations that are often non-intuitive). Even when the default working locations of R are easy to locate, it is not advisable to clutter outputs, saved R images (that include all the intermediate results from one R session), and other relevant graphs, in one place. Instead, for each project, create a working space designed for that project only. Then, each time R starts, use the `setwd()` to guide the analysis to that location on the computer:

```
setwd("C:\\mainfolder\\subfolder1\\subfolder2")
```

1.3 THE TWO COMPANIONS OF R

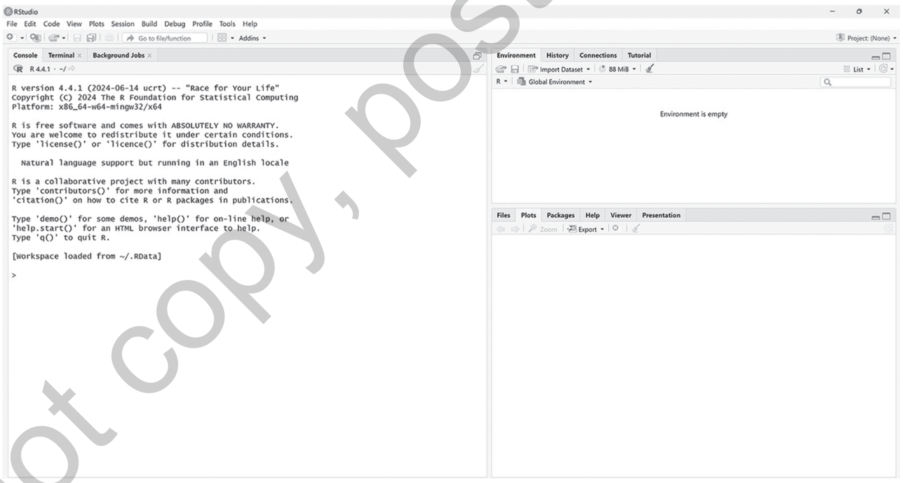
1.3.1 RStudio

By now, I believe you have found out that R is primarily a command-line based analytical platform. Although at first glance, it might seem the command-line driving analysis is somewhat tedious and last-century technology. For data analysts, however, the feeling of using command line for data analysis is actually very important. This is because it gives the analyst both a strong sense of control and actual control over the data analysis. This is vastly different from using commercial software packages when most of the time, the user provides inputs and the black box commercial software produces outputs. The process seems convenient and efficient at first, yet the black box-like commercial software takes away the actual control over data analysis. As a true data analyst, one is supposed to understand how the analysis is conducted and detect any potential deviation during the process of data analysis or, if needed, add extra steps to provide potential alternatives. With the commercial software packages, since the process is hidden from the analyst, the analyst eventually becomes a "slave" instead of "master" of the data analysis software package—one can only do what the software allows to do. With open-source software packages such as R, the analyst regains control over data analysis, becomes the true master of the data analysis process, and will be able to produce the most accurate and robust analytical results that fit the analyst's needs. Still, the generic R interface is a tad on the basic side of conducting data analysis. Some semi-graphic user interface might prove to facilitate the process of data analysis without taking away the control over data analysis.

There were efforts in R's vast package repository that attempt to provide a more user-friendly graphic interface to facilitate R's learning and data analyzing. Third-party software shells are also developed to integrate R into a better graphic user interface. Among the many efforts, I highly recommend the RStudio software that implements a convenient semi-graphic user interface but retains the actual feel of using a command-line analytical flowchart to conduct data analysis using R.

The RStudio software is also a free-of-charge software package (for personal use) that can be acquired from https://rstudio.com/. If you intend to use RStudio for commercial purposes or other advanced functionalities, there is a premium charge for it. As a scholar and educator, I never find the chance to wade into those waters, so no advice there. Again, for Windows users, installing the software package is pretty much a series of clicks away. As shown in Figure 1.2, the interface of RStudio looks like the interface of the generic R interface (which retains the feeling of using R as primarily a command line driving data analysis tool):

FIGURE 1.2 ■ The RStudio Interface



```
R version 4.4.1 (2024-06-14 ucrt) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64

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

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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

[Workspace loaded from ~/.RData]

>
```

Your screen might be slightly different from mine, but the general structure is similar. On the left side panel, it shows a similar analytical window as in the generic R interface, with three tabs: Console, Terminal, and Background Jobs. The Console tab is where most of the analytical activities will be conducted. The Terminal tab is like a mini command line window that one can navigate through different folders as when one is using Windows' own terminal shell (that can be brought up by using the "cmd" command). The Background Jobs tab allows one to run local jobs using R scripts stored on the local drive. On the top right side of the interface, there are three tabs. The Environment tab shows all the currently generated variables, user-defined functions, and other relevant intermediate results that one can easily review. The History tab stores all the recent commands inputted in the console. The Connection tab allows connections to outside database management systems. The lower-right panel contains all the utilities that are useful for viewing of contents of the default workspace (Files), data visualization (Plots), navigating (and loading) installed packages (Packages), and providing help documents (Help).

Using RStudio has a few distinctive advantages over using the generic R interface. First, RStudio allows text wrap within the console even when the Enter key is keyed in while the generic R interface would not allow such operation. In the generic R console, a long line of commands will be inputted as a single line, and if the width of the console is not enough, the line will automatically roll to the right (up to the Version 3.6.2), which makes the early part of the line invisible and increases the potential risk of missing parentheses or other critical operators (which happens quite a lot, especially if the line is long). In RStudio, inputs in the console will automatically be wrapped once the line reaches the end of the console border. If parentheses and other critical operators are missing, keying in the Enter key will not do anything, which might make you think something is wrong. So you may correct the mistake before the keyed-in command finishes and reports an error message. In addition, in RStudio, parentheses, brackets, quotation marks, and other operators that require beginning and ending pairs are usually automatically added when the beginning part is input, which reduces the risks of mistakes due to missing essential elements.

Second, the elements that are stored in R's working memory are directly viewable from the upper-right panel under the Environment tab. This is important because it helps the data analyst to examine the intermediate analytical results and check to see whether the analysis goes the way they intended. In the generic R console, I have to use the `ls()` command quite a lot to view what I have produced. This could become rather tedious and error-prone after a relatively sophisticated project that involves multiple steps and crosses many days.

Third, RStudio provides a straightforward helping panel on the lower-right side of the interface. In the generic R interface, if you type `help(something_that_needs_help)` or `?something_that_needs_help`, you will be brought out of R's interface and a browser window will open presenting the help information (or letting you know no such information is available). Switching back and forth between the R console and the helping page could quickly become distracting, especially when one explores new packages and new functionalities. In RStudio, the help information displays on the lower-right panel, and checking back and forth between the new packages and functionalities becomes much easier.

1.3.2 Notepad++

Prior to 2009, most of my R work was limited to running a little bit of analysis here and there. Even though I sometimes did have to write a few lines of scripts to streamline the process, it never occurred to me (as it was not entirely necessary) that I could save those R scripts and reuse them later. When I helped Dr. Bivand write some of the functions in the package `spgwr`, I used a simple text editor along with Windows (the generic Notepad). In 2009, I started to play around with the idea of developing a geographically weighted panel regression analysis, and I quickly realized that the generic Notepad was not much help when I had to write over 100 lines of scripts and codes at a time.

A quick Google search and communication among veteran R users led me to Notepad++. Again, this is another open source (freely available) software package from the What is Notepad++ website, https://notepad-plus-plus.org/. Without repeating what is available on Notepad++'s website, my main take and reasons for recommending this text editor follow.

First, it has all the functionalities that I (as a data analyst, not a programmer) need. The color coding of different elements of the script is especially useful. Automatic indentation and using collapsible lines on the left-hand side next to the line numbers to indicate code blocks have saved me quite a lot of troubles.

Second, the most appealing feature of Notepad++ is that this little text editor can automatically save unnamed scripts even after one closes the editor. Although I am not recommending leaving your hard-worked scripts unnamed and unsaved, accidents do happen. This feature of Notepad++ has saved me from losing my work more than once and is especially useful for new data analysts.

1.4 BASIC OPERATIONS IN R

1.4.1 R the Calculator

I always recommend when one finished downloading and installing R to use the calculator function of R. It is fun and straightforward and shows the fundamental ideal of "computation" in its most generic meaning. R can perform all the arithmetic operations, like when working with a calculator. This is also a great way to get familiarize with the computation operators in R: addition, subtraction, multiplication, and division:

```
5 + 3
[1] 8
5 - 3
[1] 2
5 * 3
[1] 15
5 / 3
[1] 1.666667
```

Square root and power operation:

```
sqrt(5)
[1] 2.236068
5^3
[1] 125
```

Integer division and the remainder of a division (the typical "mod" operation):

```
5 %/% 3
[1] 1
5 %% 3
[1] 2
```

Logarithm (natural base and 10 base) transformation:

```
log(5)
[1] 1.609438
log10(5)
[1] 0.69897
```

Exponential operation:

```
exp(5)
[1] 148.4132
```

Trigonometric operation (based on the predefined constant "pi"):

```
pi
[1] 3.141593
sin(pi/2)
[1] 1
cos(pi/2)
[1] 6.123032e-17
tan(pi/2)
[1] 1.633124e+16
```

Matrix operation (reiteration). First, to make sure that one can reproduce the numbers in the example, set a random number generator seed (`set.seed(123)`). Second, for demonstration purposes, one can define two 5 by 5 square matrices with random elements drawn from two standard normal distributions (with zero mean and 1 standard deviation). To reproduce the example in the lines, one must follow the order of the three lines that follow:

```
  set.seed(123) # set the seed to ensure reproducibility of the
script
  A<-matrix(rnorm(25), nrow = 5, ncol = 5)
  B<-matrix(rnorm(25), nrow = 5, ncol = 5)
  A
              [,1]       [,2]       [,3]       [,4]       [,5]
  [1,] -0.56047565  1.7150650  1.2240818  1.7869131 -1.0678237
  [2,] -0.23017749  0.4609162  0.3598138  0.4978505 -0.2179749
  [3,]  1.55870831 -1.2650612  0.4007715 -1.9666172 -1.0260044
  [4,]  0.07050839 -0.6868529  0.1106827  0.7013559 -0.7288912
  [5,]  0.12928774 -0.4456620 -0.5558411 -0.4727914 -0.6250393
```

```
B
            [,1]       [,2]        [,3]       [,4]        [,5]
[1,] -1.6866933  0.4264642  0.68864025 -0.6947070 -1.12310858
[2,]  0.8377870 -0.2950715  0.55391765 -0.2079173 -0.40288484
[3,]  0.1533731  0.8951257 -0.06191171 -1.2653964 -0.46665535
[4,] -1.1381369  0.8781335 -0.30596266  2.1689560  0.77996512
[5,]  1.2538149  0.8215811 -0.38047100  1.2079620 -0.08336907
```

Matrix multiplication:

```
A %*% B
             [,1]       [,2]         [,3]       [,4]        [,5]
[1,] -0.80265415  1.0424620  0.347800935  1.0696715  0.85003039
[2,] -0.01034799  0.3460080  0.005132924  0.4252767  0.31138749
[3,] -2.67558729 -1.1731406  1.339913456 -6.8318326 -2.87630259
[4,] -2.38952046  0.3488558 -0.276024304  0.5945056  0.75368346
[5,] -0.92227187 -1.2396033  0.259051616 -1.0742844 -0.02291934
```

Matrix inversion:

```
solve(A)
             [,1]       [,2]        [,3]        [,4]       [,5]
[1,]  3.11605881 -12.469418  0.01086633  0.15308383 -1.1713049
[2,]  1.35021038  -4.081299 -0.09416523 -0.72691374  0.1188552
[3,] -1.48604525   6.396487  0.45271425  0.12418361 -0.5798717
[4,]  1.28003477  -5.218414 -0.28611889  0.63586338 -0.6388145
[5,]  0.03511204  -1.410267 -0.11677971 -0.04144929 -0.9280410
```

Element-wise matrix multiplication and division (both matrices must have the same dimension):

```
A * B
            [,1]       [,2]        [,3]       [,4]        [,5]
[1,]  0.9453505  0.7314139  0.84295200 -1.2413810  1.19928197
[2,] -0.1928397 -0.1360032  0.19930723 -0.1035117  0.08781879
[3,]  0.2390640 -1.1323888 -0.02481245  2.4885502  0.47879047
[4,] -0.0802482 -0.6031485 -0.03386478  1.5212101 -0.56850973
[5,]  0.1621029 -0.3661474  0.21148143 -0.5711141  0.05210894
```

```
A / B
            [,1]       [,2]       [,3]       [,4]       [,5]
[1,]  0.33229256  4.0215917  1.7775345 -2.5721825  0.9507751
[2,] -0.27474463 -1.5620493  0.6495800 -2.3944642  0.5410353
[3,] 10.16285211 -1.4132778 -6.4732737  1.5541511  2.1986343
[4,] -0.06195071 -0.7821736 -0.3617524  0.3233611 -0.9345177
[5,]  0.10311549 -0.5424443  1.4609290 -0.3913959  7.4972564
```

Matrix transposition:

```
t(A)
           [,1]       [,2]       [,3]        [,4]       [,5]
[1,] -0.5604756 -0.2301775  1.5587083  0.07050839  0.1292877
[2,]  1.7150650  0.4609162 -1.2650612 -0.68685285 -0.4456620
[3,]  1.2240818  0.3598138  0.4007715  0.11068272 -0.5558411
[4,]  1.7869131  0.4978505 -1.9666172  0.70135590 -0.4727914
[5,] -1.0678237 -0.2179749 -1.0260044 -0.72889123 -0.6250393
```

The determinants of the matrices:

```
det(A)
[1] -0.3867446
det(B)
[1] -3.808855
```

I would encourage you to explore the calculator side of R to get a strong feeling of R's computation capability and limits (using large numbers or large matrices). It is also a good start to familiarize yourself with potential error messages that you are bound to encounter in later R scripting/coding/programming. Familiarity with those error messages will serve you well in the long run.

1.4.2 R the Script Interpreter

After experimenting with the calculator capability of R, I hope you also gained the distinctive feeling that R is an immediate script interpreter. In other words, when you input a line or multiple lines of scripts, R will interpret these scripts and produce corresponding responses. One distinctive advantage is that you can see immediately what your scripts are doing. Is it correct? Does it do what you want? If there are errors, what might cause those errors? Such an immediate response is important, especially for data analysts who are not trained as programmers.

Apparently, as a script/code interpreter, one drawback is that R may take excruciatingly long if the scripts are long or the objects involved in analyses (such as matrices) contain large numbers or multiple loops that requires many repeated computations. Still, as computers are becoming more powerful and parallel computing is a very real thing in the current computational technology arsenal, computational bottlenecks will become less of a concern because R is a code interpreter. In any scenarios, the advantage of immediate responses of a script interpreter gives data analysts more confidence to move forward.

1.4.3 R the Debugger

R is a complete kit for serving as the calculator, interpreting the scripts, performing statistical analysis, and providing an environment and platform for coding. Again, not assuming from a programmer's perspective, most of the time I am simply using packages to finish my tasks. There are occasions (and more such occasions when one uses R for long) when I have to modify the contents of a specific function to suit my own needs (fortunately that's entirely legal for most R packages). This is what I will be doing and what I always recommend even newcomers do.

Read the help page of the function carefully. Understand the meanings and usages of each parameter as much as possible. Then simply type the function's name in R (without the parentheses) and copy the output to Notepad++.

Modify the codes and assign the modified function to a new name (so it doesn't conflict with the original copy). If, however, something was not correctly modified or edited or if functions/parameters are not appropriately assigned (a "bug" was created) during your customization, a useful function `traceback()` can be used to pinpoint where exactly the problem is. For instance, considering the following script:

```
my_function <- function(x) {
  y <- x + 1
  z <- log(y)
  return(z)
}

# Intentionally causing an error
Print(my_function("a"))
```

You will get an error message:

```
    Error in h(simpleError(msg, call)) :
      error in evaluating the argument 'x' in selecting a method
for function 'print': object 'a' not found
```

While the error message is simple enough, using `traceback()` could provide a more organized way to find where the error is from:

```
traceback ()
4: h(simpleError(msg, call))
3: .handleSimpleError(function (cond)
      .Internal(C_tryCatchHelper(addr, 1L, cond)), "object 'a'
not found",
         base::quote(NULL)) at #2
   2: my_function(a)
   1: print(my_function(a))
```

The output list traces all the way to where the error message was produced: The function `print()` (1:) calls the function `my_function()`, (2:) within the print() function, the internal error that "`object 'a' not found`" produces the error, (3:) and then the internal function `h(simpleError(msg, call))` produces the error message that we saw (4:).

When we need more detailed investigation and want to test out a few possible solutions to the bugs, we can then use `debug()` function to enter debugging mode of the newly created function. As a script interpreter, once you enter the debugging mode, each and every line within the function will be executed one by one. The intermediate results that usually are not visible to the user will now be viewable.

In computer programming, debugging is most used to find out whether or not there are software design issues or "bugs" that might prevent the software from doing what it is programmed to do. Debugging in the data analysis field can also be used to track errors and mistakes that are not immediately obvious (like typos) per the error message reported by the R console.

Long story short, I have been using the debugging function in R almost constantly to find out where something might go wrong. The usual steps I take following a reported error message in R follow:

- First, I will read the error message immediately to understand what exactly is going on. If the error message is clear enough (a typo, an exact or computational singular matrix for inversion operation, non-conformable matrix operation, etc.), I will correct the inputs to make the analysis proceed.
- Second, if the error message is too simple to decipher what exactly is going on, I will read the error message again to decide how many possible functions/routines are involved in producing the error message in addition to the one that generates the error.
- Third, I will then call the `debug()` function to include all possible functions/routines (one by one) that might produce the error message. With this step, the advantage of RStudio is obvious. Unlike the generic R console, which shows the codes of the function/routine immediately after the debugging process starts and then only show the lines where the current debugging process is, the RStudio will replace the upper left panel of the window with the entire code block of the function/routine that is being debugged and highlight the lines where the debugging process is at. This contextual viewing mechanism makes the debugging process more connected and meaningful than just showing the entire block of code once.
- Fourth, once the debugging process is ongoing, for each step, I will carefully examine the newly produced intermediate results and variables to see if anything

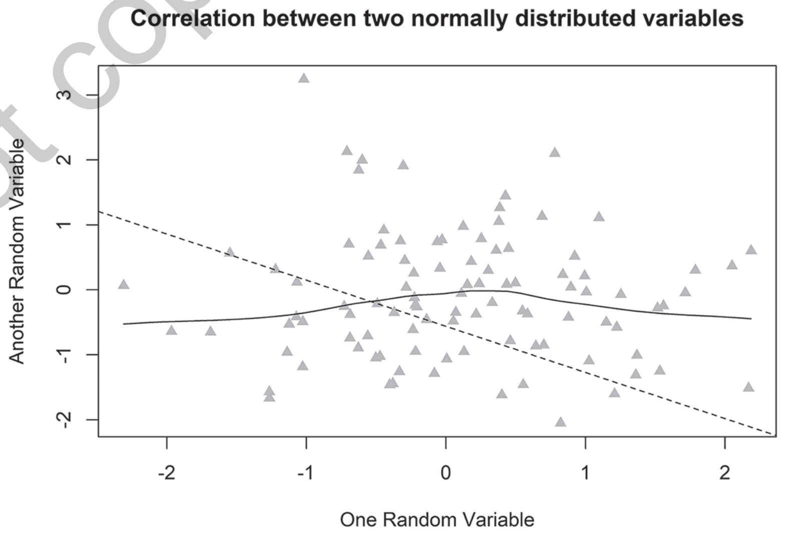
is out of the ordinary. This is especially important when debugging reaches the place where the error might be produced (it might take a few patient tries to get to the exact place where the error is being produced). Immediately before the step where the error is being produced, I will examine all the intermediate results and try to change a few things to see if I can bypass the error message. If I can, then I've found the problem, and a corresponding solution is on the horizon. If I cannot, then different analytical strategies might be needed—changing the inputs or finding an alternative analytical function are potential solutions. Still, as a data analyst, performing trial and error is a must and is a fairly easy process in R. It might come to the point where all easy alternatives (changing inputs, finding alternative functions) are not enough. R then provides the opportunity to change the code of the function/routine if that change is theoretically and methodologically justified.

- Fifth, once the debugging process is finished, the `undebug()` function must be called to tell R the debugging ends there. If the `undebug()` function is not called, R will automatically enter another round of debugging if the same function/ routine is called again.

1.4.4 R the Graph/Map Maker

Figure 1.3 is a graph produced in R with some relatively simple generic plot functions. First, two normally distributed random variables are generated. These two variables are put in a scatterplot, and a smooth line is added to show the relationship (or lack of relationship) of the two variables (the smooth line shall be close to a horizontal line since these are two randomly generated variables that have no inherent relationships at all). The scatter points are added as blue triangles. Then a trend line between these two variables is added as a red dash line in the graph.

FIGURE 1.3 ■ A Simple Scatterplot Produced in R



These are the scripts used to reproduce the preceding scatterplot between two normally distributed variables:

```
   jpeg("figure1-3.jpg", units = "in", width = 7, height = 5,
res=600)
   set.seed(123)
   x<-rnorm(100)
   y<-rnorm(100)
   scatter.smooth(x, y, xlab = "One Random Variable", ylab =
"Another Random Variable", main = "Correlation between two
normally distributed variables", col = "grey", pch = 17) # You
can set the col parameter to different colors to suit your own
needs
   abline(x, y, col = "black", lty = 2)
   dev.off()
```

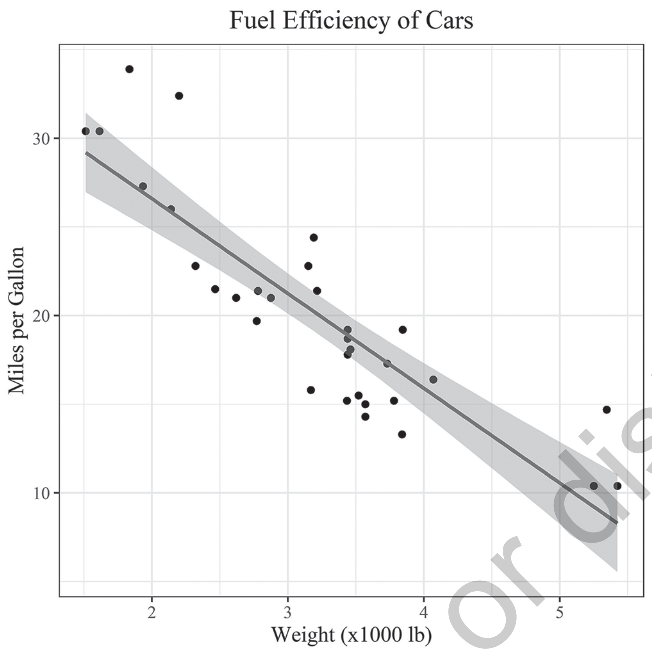
The `jpeg()` and `dev.off()` functions open and close the graphic device that will receive the graph. The line `set.seed(123)` sets a reproducible "seed" (The number 123 could be any number of your choice; as long as it remains the same, the results, especially from the native computer's random number generator, will remain the same.). The lines `x < rnorm(100)` and `y <- rnorm(100)` generate two normally distributed random variables, and the `scatter.smooth()` function produce the scatterplots with the smooth solid line, while the `abline()` function generates the straight dashed line in the graph.

The graph might not look like much compared to other commercially available software packages. It is produced by the generic plotting functions in R. What I have shown here, however, contains only very limited operations on the graphs that can be produced by R. Currently, the most popular graphic-producing package in R is the ggplot2 package by H. Wickham. The graphs produced by ggplot2 are more highly enhanced than the ones that can be produced by the generic plotting functions. Figure 1.4 is an example of a graph produced by ggplot2.

We produced the preceding graph using the example dataset that comes with ggplot2, the *mtcars* that contains the fuel efficiency data. The actual scripts used to produce the graph follow:

```
   windowsFonts(FontStyle=windowsFont("Times New Roman")) # this
sets the graph fonts to be "Times New Roman"
   jpeg("figure1-4.jpg", units = "in", width = 5, height = 5,
res=600)
   ggplot(mtcars, aes(x=wt, y=mpg)) + geom_point()+
       ggtitle("Fuel Efficiency of Cars") +
       xlab("Weight (x1000 lb)") + ylab("Miles per Gallon") +
       theme_bw()+
       theme(text=element_text(family="FontStyle", size=12)) +
stat_smooth(method = "lm") +
       theme(plot.title = element_text(hjust = 0.5))
   dev.off()
```

FIGURE 1.4 ■ Illustration of a ggplot2 Produced Graph



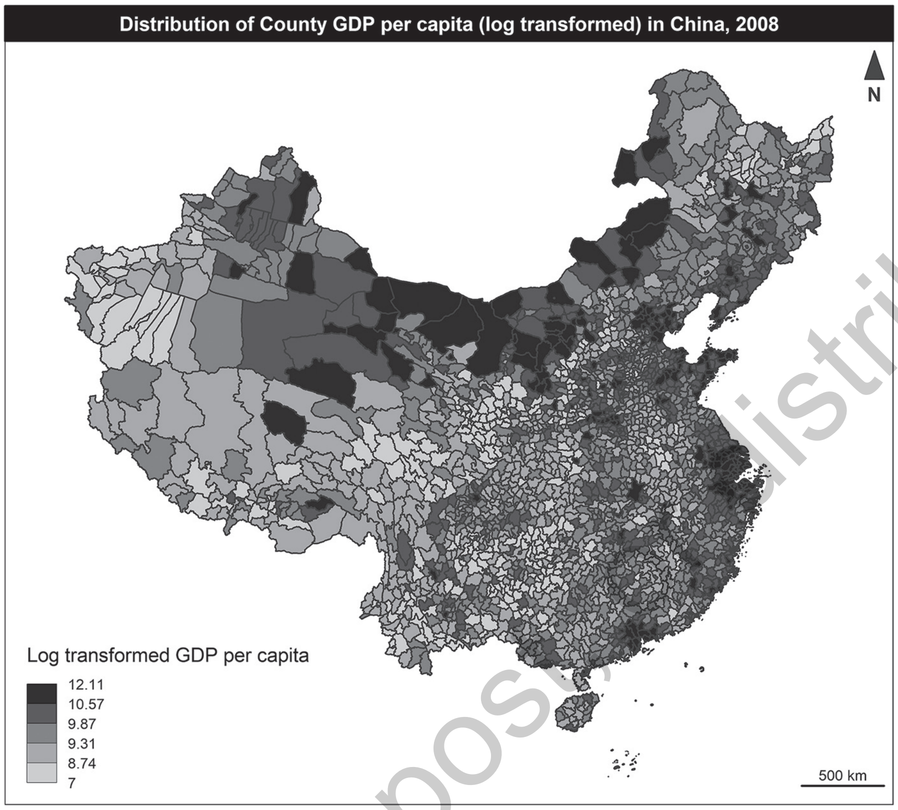
Again, the `jepg()` and `dev.off()` functions open and close the graphic device. However, in addition to using the generic plot functions, by using `windowsFonts()` to assign the font style, we are able to further fine-tune the graphs to our own needs (we can do this with generic plot functions as well, but this involves more steps than using ggplot2 package). Alternatively, when using ggplot2 package to produce a graph, you can save the graph to a graph object and use the ggsave() function to store the graph:

```
   p.out <- ggplot(mtcars, aes(x=wt, y=mpg)) + geom_point ()+
        ggtitle("Fuel Efficiency of Cars") +
        xlab("Weight (x1000 lb)") + ylab("Miles per Gallon") +
        theme_bw()+
        theme(text=element_text(family="FontStyle", size=12)) +
stat_smooth(method = "lm", col = "black", lty = 2) +
        theme(plot.title = element_text(hjust = 0.5))
   ggsave("Figure1-4.jpeg", p.out, width = 5, height = 5,
dpi=600)
```

There are quite a few intuitive tutorials and manuals online that can guide you through even the most daunting graphic tasks using ggplot2. I won't go into further detail in this book, but I strongly encourage you to explore ggplot2 package on your own.

Creating maps follows similar graphic strategies and uses the same graphic devices. The package ggplot2 is just one of many packages you can use to create maps. Among these packages, I found the "cartography" package works very well with the spatial data that social scientists often encounter. The package is also quite intuitive for one who is familiar with standard GIS map-making software, such as ArcGIS and QGIS. Figure 1.5

FIGURE 1.5 ■ An Illustrative Map of County Level Log Transformed GDP Per Capita in 2008, Mainland China


Distribution of County GDP per capita (log transformed) in China, 2008
N
Log transformed GDP per capita
12.11
10.57
9.87
9.31
8.74
7
500 km

is a simple example of an illustrative map produced by R showing the log transformed GDP per capita at county level in 2008 of mainland China.

The scripts used to create the map are simple and self-explanatory for anyone who is familiar with mapmaking in standard GIS software packages:

```
   #First, load the package "rgdal" that we used to read a
standard shapefile that contains both geography and attributes,
and the "cartography" package that we will use to create the map
(note that the "rgdal" is retired for better and more intuitive
spatial object):
   library(rgdal)
   library(cartography)
   #The workspace must be set to where the shapefile is stored
using the setwd() function as mentioned earlier, then read the
shapefile ("county_prj.shp") using the readOGR() function:
   cnct<- readOGR("county_prj.shp")
   #Setting up the graphic device using the jpeg() function and
close it off using the dev.off() function as before:
   jpeg("figure1-6.jpg", units = "in", width = 7.25, height = 7,
quality = 100, res = 600)
```

```
#Using the choroLayer() function to generate the map
choroLayer(spdf = cnct, var = "lnGDP", nclass = 5, col =
carto.pal(pal1 = "grey.pal", n1 = 5), method = "fisher-jenks",
legend.title.txt = "Log transformed GDP per capita", legend.
frame = FALSE, legend.pos = "bottomleft", legend.values.rnd = 2)
#Using the layoutLayer() function to overlay the title, map
scale, north arrow onto the map.
layoutLayer(title = "Distribution of County GDP per capita
(log transformed) in mainland China, 2008", posscale =
"bottomright", north = TRUE, postitle = "center")
dev.off()
```

The preceding script block gives users satisfactory control over how a choropleth map is made using the `choroLayer()` function. Once the shapefile is read into an R object (`cnct`), the `choroLayer()` function will take that as its first parameter (`spdf = cnct`). The variable that needs mapping is one of the attributes stored in the shapefile (`lnGDP`, the log transformed GDP per capita at county level in 2008, China). The "`col`" parameter controls what type of color scheme (palette) will be used. The "`grey.pal`" is an internally defined color palette that produces a grey-scale color scheme that is usually preferred for academic publications. The "method" parameter controls the classification method used to put the continuous values into different categories. The "`fisher-jenks`" method is one of the most commonly use methods in mapmaking. The other control parameters control the text, position, number format, and frame of the legend.

The auxiliary function `layoutLayer()` adds essential map elements, the title of the map, the scale bar, and the north arrows to the map. The values of both `choroLayer()` and `layoutLayer()` are usually self-explanatory enough so that the map maker can create a nice map without too much trouble. The document and function descriptions for the cartography package are also excellent guides for any map makers, and we encourage the potential map maker to read those documents and descriptions carefully to fully appreciate the power R brings to data analysis and graphic making.

Alternatively, the map can also be produced by ggplot2 package:

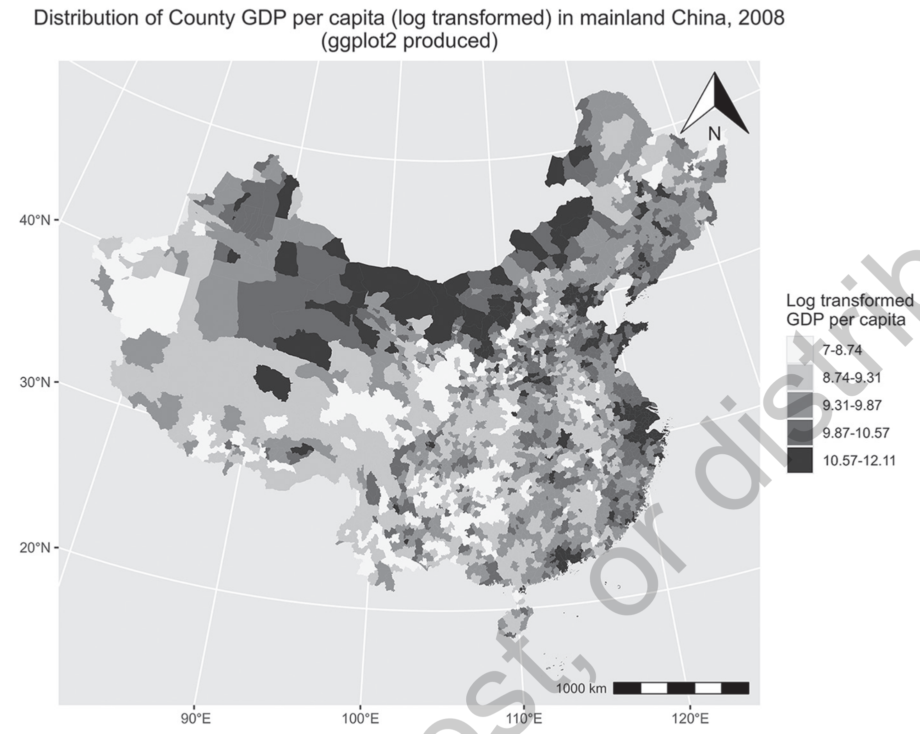
```
# Installing packages (if not installed)
if (!require(classInt)) install.packages("classInt")
if (!require(ggplot2)) install.packages("ggplot2")
if (!require(sf)) install.packages("sf")
if (!require(ggspatial)) install.packages("ggspatial")

# Loading packages
library(sf)
library(ggplot2)
library(classInt)
library(ggspatial)
```

```
  # Read the "county_prj.shp" as an sf object (instead of a
rgdal object for simplicity purposes):
  cnct <- st_read("county_prj.shp")
  # Then, use the classIntervals function from classInt package
to classify your "lnGDP" attribute into 5 classes using the
"fisher-jenks" method:
  # Calculating breaks using Fisher-Jenks
  breaks <- classInt::classIntervals(cnct$lnGDP, n=5,
style="fisher")$brks
  # Creating labels for these classes
  labels <- c()
  for (i in 1:(length(breaks) - 1)) {
   labels <- c(labels, paste0(round(breaks[i], 2), "-",
round(breaks[i+1], 2)))
  } # The labels object is for nicely looking labels for the
legend items.
  # You can then add this information as a new factor variable
in your sf object:
  cnct$lnGDP_class <- cut(cnct$lnGDP, breaks = breaks, labels =
labels, include.lowest = TRUE)
  # Create a palette of colors corresponding to each class and
reverse the vector
  colors <- RColorBrewer::brewer.pal(5, "Greys")
  # Create the plot
  map_plot <- ggplot(data = cnct) +
   geom_sf(aes(fill = lnGDP_class), color = NA) +
   scale_fill_manual(values = colors,
                     name = "Log transformed\nGDP per capita",
                     breaks = levels(cnct$lnGDP_class)) +
   labs(title = "Distribution of County GDP per capita (log
transformed) in mainland China, 2008") +
   theme(plot.title = element_text(hjust = 0.5), # centering
title
         legend.position = c(0.1, 0.1)) + # move legend to
bottom left
  annotation_scale(location = "br") +  # scale bar at the
bottom right
  annotation_north_arrow(location = "tr")  # north arrow at the
top right
```

The resultant map (Figure 1.6) looks somewhat different from the one produced from the `cartography` package (less control over the map elements, the north arrow can only be placed at "`tr`" top right or "`br`" bottom right) but is otherwise more appealing. However, since the longitudinal and latitudinal lines are added, the north arrow might not be needed, so the last line can be removed.

FIGURE 1.6 ■ An Illustrative Map of County-Level Log Transformed GDP Per Capita in 2008, Mainland China



The preceding script makes the transition from using the `sp` object that was produced by the "`rgdal`" package to the `sf` (simple feature) object that is a more recent development that follows the Simple Features standard by the Open Geospatial Consortium (OGC). This standard provides a consistent way to describe geometries and their relationships. The `sf` objects are based on data frames, making them more intuitive for users familiar with the data frame concept in R. On the other hand, the `sp` package in R has been a fundamental tool for handling spatial data. It introduced classes like `SpatialPoints`, `SpatialLines`, `SpatialPolygons`, and `SpatialGrid` to represent spatial data. These classes allowed for basic spatial operations, reading/writing spatial data, and interfacing with other spatial data formats. However, `sp` objects use slot-based S4 classes, which can be complex and less intuitive for new users.

On the contrary, `sf` objects use data frames with a special column for geometries, making them more consistent with typical R data manipulation workflows. For instance, we can simply use the following script to produce a simple feature (`sf`) object and actually "map" it (using the generic plot function). It will even show default choropleth color scheme based on the only available data – the "id."

```
   library(sf)
   # Create a simple feature (sf) object of 10 points, and
randomly generate their coordinates
   sf_obj <- st_as_sf(data.frame(
     x = rnorm(10),
     y = rnorm(10),
     id = 1:10),
     coords = c("x", "y")) # set up the coordinates
   print(sf_obj) # print the data frame view
   plot(sf_obj) # map the points with default id value to create
the color ramps
```

The `sf` package has extensive documentation and is actively maintained while the `sp` packages are retiring, ensuring users have access to the latest features and best practices in spatial data analysis. The transition from `sp` to `sf` objects in R represents a significant advancement in spatial data analysis. The `sf` objects provide a more intuitive, efficient, and consistent framework for handling spatial data, integrating seamlessly with modern data analysis workflows in R. The enhanced performance, improved visualization capabilities, and better support make sf the preferred choice for contemporary spatial data analysis.

1.5 THE R PACKAGES

As previously mentioned, one of the most appealing aspects of using R for statistical computation is the availability of various R packages. In general, each R package deals with one or one group of similar issues. For instance, the `spdep` and `spatialreg` packages (the `spatialreg` package was originally included in the `spdep` package but later separated to deal with spatial regression analysis) that this book will use intensively, by Dr. Roger Bivand and colleagues, state in the descriptions that `spdep` deals specifically with spatial data analysis, with detailed functions that deal with corresponding aspects of spatial data analysis, and `sptialreg` deals with spatial regression models. Since R packages published on the cran.r-project.org website will go through rigorous scrutiny by the CRAN moderators and a description of the package is an essential part of the built package, reading the description of the published R package usually will give readers sufficient information to decide whether the package meets the research needs.

Although it might be exaggerating to state that R covers nearly every aspect of statistical computation, the statement is probably not too far off. Wide coverage aside, the open-source feature of R determines that if new algorithms are proposed in the frontier of academic endeavors, the authors of such algorithms often will provide detailed instructions on how to implement such algorithms. Chances are that either the authors have already implemented those algorithms in R or the instructions are clear enough, and others can follow the instructions and implement those algorithms in R.

When I started to incorporate geographically weighted regression (GWR) analysis into my doctoral dissertation back in 2003, I gained help from two great sources: the *Geographically Weighted Regression* book published in 2002 (Fotheringham et al., 2003) and the prototype of GWR scripts written in R by one of the authors, Dr. Chris Brunsdon. The book was clearly written with all the proposed methods outlined in an easy-to-follow way. Dr. Chris Brunsdon's R scripts were also clear enough so I could follow through and make modifications based on the book's description. When I started to incorporate the R scripts to analyze my own data in late 2003 and early 2004, I was only less than half a year into learning R but already a 100% converted R user.

The benefit of making available and being able to utilize the most advanced statistical computational technologies relatively readily is twofold. On one hand, the algorithm proposers will be able to disseminate their research results to a wider audience and receive feedback and rigorous scrutiny quickly. Unlike commercial software packages, which often will only incorporate more mature and proven methods into the official releases and hence take years for a particular method to be incorporated (if incorporated at all), the open-source software packages can do so in a matter of weeks and receive feedback and scrutiny in an even shorter time. For the algorithm proposers, this is an excellent chance to spot any potential flaws in the proposed algorithm and make the algorithms even better. On the other hand, for the algorithm users, being able to apply the most advanced (newly available) algorithms in their research will produce potentially unprecedented research products and hence advance the scientific study of relevant disciplines quickly. These two benefits are apparently mutually supplementary. Better algorithms will attract more users, while more users will likely push the algorithms to become better. My own participation in contributing to the `spdep` and `spgwr` packages during the past decade is one such example, and I strongly encourage the readers of this book to explore the possibilities provided by R and R packages.

1.6 THE R TASK VIEWS AND SPATIAL TASK VIEW

As with any other software packages (commercial and open source alike), the packages are usually interrelated with one another to provide the most efficient computation and data analysis capability. For commercial software packages, such interrelationship will eventually lead to the packaging of relevant functions together to produce a software product. Common ones for statistical data analysis such as SPSS, EViews, SAS, ArcGIS and the like are good examples.

In R, to facilitate scholars finding relevant packages for a particular type of analysis, CRAN task views are created. The following CRAN task views description can be found at https://cran.r-project.org/web/views/:

> CRAN task views aim to provide some guidance which packages on CRAN are relevant for tasks related to a certain topic. They give a brief overview of the included packages and can be automatically installed using the ctv package. The views are intended to have a sharp focus so that it is sufficiently clear which packages should be included (or excluded) - and they are *not* meant to endorse the "best" packages for a given task. (CRAN Task Views, 2024)

Organizing the enormous and ever-increasing number of individually developed but centrally managed packages into task views is one excellent idea, especially for new data analysts since navigating the sheer number of packages can be a daunting task if one has only his or her own data and idea but little clue about what to use to execute the idea. Currently there are 40 task views listed on the CRAN website, ranging from Bayesian Inference to gRaphical Models in R (https://cran.r-project.org/web/views/). All the task views are maintained by volunteers, and users are the most active groups who can suggest new packages to be added to specific task views. Not surprisingly, many packages will appear in different task views due to their versatility in dealing with different topics.

During the past 10 plus years, packages that handle spatial data (or geographic data) have increased rapidly, thanks mainly to the development of the `sp` package by a group of R developers (Bivand, Pebesma, Gomez-Rubio, & Pebesma, 2008). The `sp` package extends R with fundamental classes and methods that are specifically designed to handle spatial data (Pebesma & Bivand, 2005). The spatial classes specify a structure of spatial data and define how such data are organized and stored. These include spatial points, lines and polygon classes that are commonly used in vector GIS and are familiar to social science researchers, and grid/raster classes that are more often used in natural science but see increasing application in social science as well. The fundamental methods define functions specialized for a particular data class. This increasing number of packages are now included in R's Spatial task view (https://cran.r-project.org/view=Spatial) maintained by Dr. Roger Bivand, professor in the Department of Economics, Norwegian School of Economics. The R-sig-geo mailing list (https://stat.ethz.ch/mailman/listinfo/R-SIG-Geo/) is also an excellent place to subscribe to for the best solutions, answers, and developments in spatial data analysis.

The Spatial task view includes packages that define classes for spatial data and metadata, packages that read and write spatial data, packages that handle spatial data, packages that visualize spatial data (mapping), and packages that analyze spatial data. In this book, I am trying to focus on spatial data reading/writing, handling, analyzing, and visualizing. Although when I first got into contact with the Spatial task view, I immediately installed the `ctv` package and then installed the entire task view (which took quite some time to do), I would not recommend that newcomers follow suit. As my working experiences with R grows, I realize that R's scalability and flexibility allows me to not include the entire task view in my daily data analysis. Though storing those packages on the computer seems like a safe bet, installing them when they are needed might prove to be more efficient. Still, the options are there, and the final decision is yours. Either way, I hope you will find the beauty of R for analyzing your spatial data as I did over a decade ago.

CONCLUSION

This first Chapter introduces you to the most basic side of the open-source computational statistic software/platform R and its ever-increasing number of packages that make it so versatile and ideal for many analytical tasks. This chapter is by no means a complete introduction to R and its functionalities. It rather serves as a door-opener for the data analysts to the wonderland of R. To enjoy what R can bring to you, you must be willing to sit down and work on it. Write a few scripts and see what R's responses are. Exploration is 90% of the journey. When I was asked the question, should I learn R? I always answered with my own question: Do you have a data analysis project at hand? If yes, then yes, you should learn R. The idea here is that R is best learned by doing. Not to diminish the importance of systematic training in using R, for the average social science data analyst, however, my experience suggests that "learning by doing" will often yield the most cost-effective learning experience. In addition to getting the analytical results, the (most likely) added benefits are that one will have the curiosity to pursue more of R's functionality and work more to know R.

REVIEW QUESTIONS

1. Discuss the primary functions and features of the R software environment.
2. Write an R script that creates a vector, matrix, data frame, and list. Provide examples for each, and explain the operations you perform on them.
3. Explain what R packages are. Demonstrate how to install and load an R package. Explain how you would use the package to perform a specific statistical computation or data analysis task.
4. Discuss the purpose and importance of CRAN as a source for R packages.
5. Explain the concept of CRAN task views and their significance in data analysis.
6. How does the Spatial task view aid in spatial data handling, visualization, and analysis? Please check the CRAN spatial task view for a detailed explanation.
7. Create a script that handles different data types in R (e.g., numeric, character, logical). Demonstrate how R manages these data types within vectors, matrices, data frames, and lists.
8. How does the wide coverage and open-source feature of R facilitate the development and application of new statistical algorithms?
9. Discuss the process by which R packages are made available on CRAN. How does this process ensure the quality of these packages?
10. Describe the impact of R's scalability and flexibility on its efficiency in data analysis.
11. How do algorithm proposers and users benefit from utilizing advanced statistical computational technologies that R offers?
12. How do R and its packages facilitate interdisciplinary research and advance scientific study?
13. Compare the incorporation of new packages in R with their incorporation in commercial software packages.

2 VERY BASIC CONCEPTS OF STATISTICAL DATA ANALYSIS

LEARNING OBJECTIVES

2.1 Understand the fundamental concepts of descriptive statistics and their application in analyzing and summarizing data.

2.2 Learn the principles and steps of hypothesis testing, including formulating null and alternative hypotheses, selecting significance levels, and interpreting *p* values.

2.3 Gain proficiency in conducting hypothesis tests using different statistical tests, such as *t* tests and chi-square tests in R.

2.4 Develop skills in interpreting the results of hypothesis tests and making informed decisions based on statistical evidence.

2.5 Understand the concepts of Type I and Type II errors and their implications in hypothesis testing.

2.6 Learn how to perform basic statistical analyses and hypothesis tests using R programming language.

2.7 Gain an understanding of foundational statistical concepts essential for further study, especially in spatial data science.

In this chapter, we aim to provide a gentle introduction to fundamental statistical terms and concepts for those who may need a refresher. Our goal is to make this content accessible and understandable to students with no prior knowledge of the subject. While this chapter is not a comprehensive guide to statistical analysis and probability theory, it will cover key concepts that will be used throughout the book. For a deeper understanding, we recommend pursuing formal training in statistics and probability.

The chapter begins with an overview of variables, variable distributions, and degrees of freedom. We will then delve into hypothesis testing, with an emphasis on understanding and interpreting *p values*, a concept frequently encountered in data analysis. Next, we will introduce essential tools for conducting exploratory data analysis, which will help you "listen" to your data and uncover meaningful insights. Finally, we will present an example of regression analysis, a topic that will be explored in greater detail in later chapters.

To ensure a practical, hands-on learning experience, all concepts and techniques will be demonstrated using the R programming language. We encourage you to practice along with the examples provided to gain valuable experience and develop your skills in spatial data analysis with R.

2.1 THE CONCEPTS OF VARIABLE, RANDOM VARIABLE AND VARIABLE DISTRIBUTION, AND DEGREES OF FREEDOM

2.1.1 Variable

In this section, we will introduce the concept of a variable and how to work with variables in R, providing hands-on examples for students with limited statistical training. Remember that hands-on experience is essential for learning.

A variable represents a specific characteristic of an object or an individual that can be measured or quantified. Examples of variables include age, gender, income, and temperature. In data analysis, variables are used to describe the data collected from observations or experiments. For example, the length of a river, the annual per capita GDP of a county, and the number of disease incidents are all variables. The term "variable" is used because the values it represents vary across different observed objects. In this sense, a variable can be thought of as a container that holds the varying values associated with specific objects in certain places.

For effective analysis, a variable must have an identifier to distinguish it from other variables. This identifier is called the variable's name (variable name). In R, a basic variable can be expressed as a vector using the concatenation function c():

```
   # Numeric variable
   A <- c(1, 2, 3, 4, 5)
   # Text variable:
   B <- c("This", "is", "an", "introduction", "to", "spatial",
"analysis", "using", "R", "book")
```

Once a variable has been assigned a name, you can access its elements using valid subscripts. First, determine the length of the variable vector:

```
   # Length of the vector:
   length(B)
```

Then, use any number from 1 to the length of the vector to extract the desired element. Note that in R, indexing starts at 1, unlike some other programming languages that use 0 as the starting index.

Variables are the building blocks for data analysis. Data analysis starts with understanding the data (through exploratory analysis) by using graphs, tables, and correlation analysis to examine relationships or co-variation among different variables. Analyzing data involves examining collections of variables that describe the various features of objects in a particular place. The process starts by forming a hypothesis about the characteristics and relationships among different aspects of these objects. Next, we observe the objects, identify variables of interest (i.e., the object's characteristics), and collect data using appropriate methods, such as surveys, field work, census statistics compilation, remote sensing image analysis, among many others. The collected variables are then organized into a suitable format, and appropriate methods are applied to test the hypothesis.

2.1.2 Random Variables

To use variables for data analysis and modeling, however, we also need to introduce the concept of random variables. A random variable is a type of variable whose possible values are numerical outcomes of a random phenomenon. Unlike regular variables that are fixed once observed, random variables are used to model and analyze data that are subject to randomness and uncertainty. There are two main types of random variables:

- **Discrete random variables:** These can take on a finite or countably infinite set of values. For example, the number of heads when flipping a coin three times is a discrete random variable, which can take the values 0, 1, 2, or 3.
- **Continuous random variables:** These can take on any value within a given range. For example, the exact height of students in a class is a continuous random variable, which can take any value within a certain interval.

Random variables are fundamental in statistical analysis because they allow us to make probabilistic statements about data. For example, we can calculate the probability that a random variable falls within a certain range or determine the likelihood of different outcomes.

In R, random variables can be simulated using functions from various probability distributions. For example:

```
   # Simulating a discrete random variable: Number of heads in
10 coin flips
   set.seed(123)
   num_heads <- rbinom(10, size = 1, prob = 0.5)
   print(num_heads)

   # Simulating a continuous random variable: Heights of
students
   set.seed(123)
   heights <- rnorm(100, mean = 170, sd = 10)
   hist(heights)
```

While regular variables hold observed data, when we analyze these data, especially for making predictions or inferences about a larger population, we treat these regular variables as random variables. This is because we want to do the following:

- **Model uncertainty:** By treating a regular variable as a random variable, we acknowledge that the observed data are just one possible outcome of a *random* process.
- **Make statistical inference:** By treating variables as random, we can estimate population parameters, test hypotheses, and calculate confidence

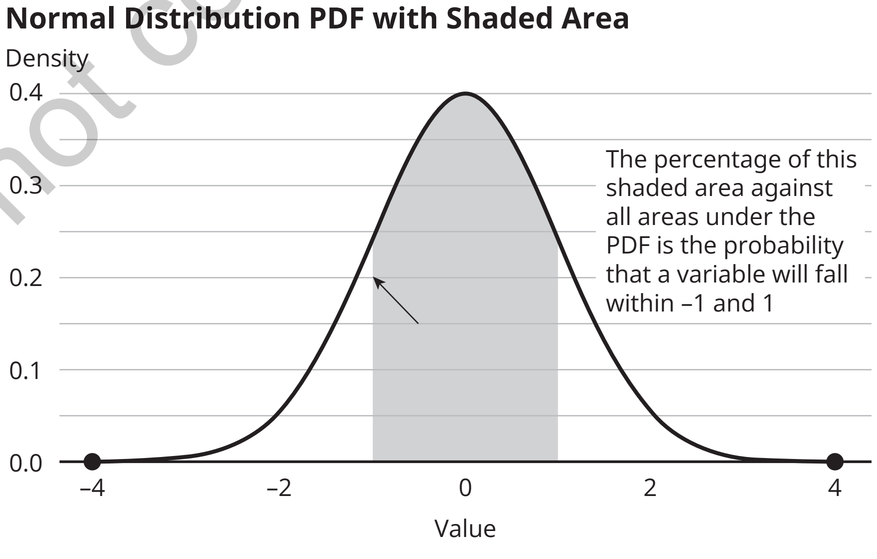
intervals—basically, answer this question: How *likely* is (or what is the *likelihood* of) the observed value if it is one of the results of a random experiment, which assumes all the possible outcomes are known.

- **Understand probability distributions:** This is because random variables are associated with probability distributions, which provide a framework for making probabilistic statements about the data. This is essential for predictive modeling and hypothesis testing.
- **Estimate sampling variability:** Treating regular variables as random variables helps us understand and quantify sampling variability. This is important because any given sample is just one of many possible samples that could be drawn from the population.

2.1.3 Variable Distribution

The concept of variable distribution is essential for data analysis. In simple terms, a distribution describes the range of possible values a variable can take and the likelihood (or probability) of each value occurring. All probabilities must add up to one. For discrete variables, the probability of a particular value occurring is often described by the frequency with which that value appears in the dataset for the variable. For continuous variables, the probability of a particular value occurring is described by the probability density function (PDF) rather than the raw frequency. A PDF assigns a probability density to each value within the continuous variable's range. A PDF is usually graphed as a curve in a coordinate system. Figure 2.1 plots a typical PDF of a standard normal distribution (mean zero and standard deviation 1). Since a continuous variable can take infinitely many values within its range, the probability of observing an exact value is technically zero. Instead, we consider the probability of the variable falling within a specific interval.

FIGURE 2.1 ■ A Probability Density Function (PDF) of the Standard Normal Distribution



To find the probability of a continuous variable falling within a given range, we calculate the area under the curve of the PDF within that interval and divide it over all the areas under the PDF. In practice, this often involves integrating the PDF over the specified range.

For example, consider a continuous variable following a normal distribution with a mean of 0 and a standard deviation of 1. To find the probability of the variable falling between -1 and 1, we would calculate the area under the curve of the normal distribution's PDF within this interval and divide it over all the areas under the normal distribution's PDF (Figure 2.1).

In R, we can compute the probability of a continuous variable falling within a specified range using the relevant cumulative distribution function (CDF). For a normal distribution, we would use the pnorm() function:

```
   # Calculate the probability of a normally distributed
variable with mean 0 and standard deviation 1 falling between -1
and 1
   lower_bound <- -1
   upper_bound <- 1
   mean <- 0
   sd <- 1
   probability <- pnorm(upper_bound, mean, sd) - pnorm(lower_
bound, mean, sd)
   probability
```

Understanding the concept of probability density functions and cumulative distribution functions for continuous variables is essential for selecting appropriate statistical tests, estimating parameters, and interpreting results in data analysis.

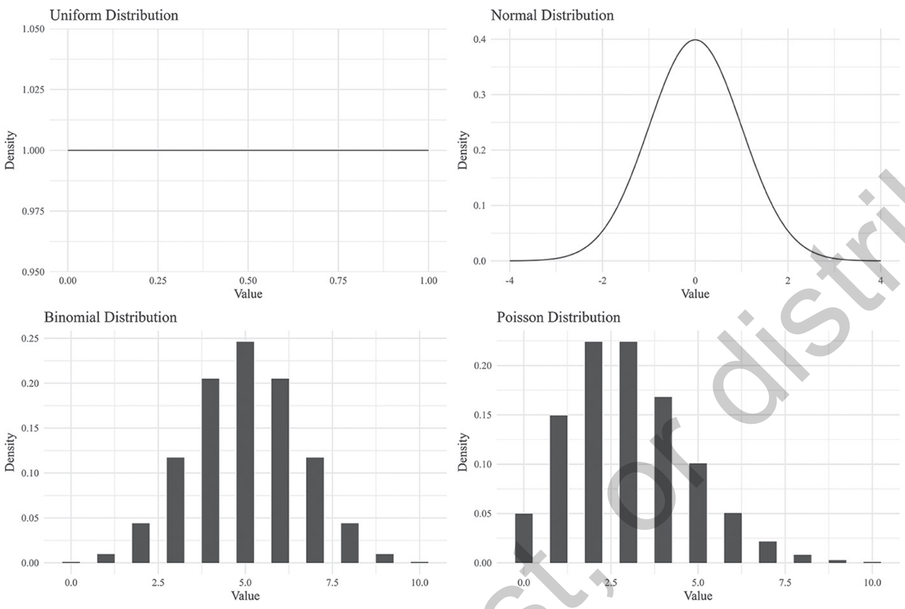
There are several types of variable distributions commonly encountered in data analysis, including continuous and discrete distributions. Continuous distributions describe variables that can take any value within a specified range (e.g., height, weight), while discrete distributions apply to variables with a limited number of distinct values (e.g., the number of children in a family).

Some common distributions include the following:

- **Normal distribution (Gaussian distribution):** A continuous distribution often found in nature, characterized by a symmetrical bell shape (Figure 2.1). Most values cluster around the mean, with fewer values appearing as we move further away from the mean in either direction.
- **Uniform distribution:** A continuous distribution where all values within the specified range have an equal probability of occurring.
- **Binomial distribution:** A discrete distribution representing the number of successes in a fixed number of Bernoulli trials (each with the same probability of success).
- **Poisson distribution:** A discrete distribution often used to model the number of events occurring within a fixed interval of time or space.

Now, let's consider some R examples to illustrate these distributions (all four distributions are illustrated in Figure 2.2).

FIGURE 2.2 ■ Illustrations for the Four Commonly Encountered Distributions



Normal distribution in R: In R, we can generate random samples from a normal distribution using the rnorm() function:

```
   # Generate 100 random samples from a normal distribution with
mean 0 and standard deviation 1
   samples <- rnorm(100, mean = 0, sd = 1)
   # Plot a histogram of the samples
   hist(samples, main = "Histogram of Random Samples from
Normal Distribution", xlab = "Value", ylab = "Frequency", col =
"lightblue", border = "black")
```

Uniform distribution in R: To generate random samples from a uniform distribution in R, we can use the runif() function:

```
   # Generate 100 random samples from a uniform distribution
between 0 and 1
   samples <- runif(100, min = 0, max = 1)

   # Plot a histogram of the samples
   hist(samples, main = "Histogram of Random Samples from
Uniform Distribution", xlab = "Value", ylab = "Frequency", col =
"lightgreen", border = "black")
```

Binomial distribution in R: We can generate random samples from a binomial distribution using the rbinom() function in R:

```
   # Generate 100 random samples from a binomial distribution
with 10 trials and a probability of success of 0.5
   samples <- rbinom(100, size = 10, prob = 0.5)
   # Plot a histogram of the samples
   hist(samples, main = "Histogram of Random Samples from
Binomial Distribution", xlab = "Value", ylab = "Frequency", col
= "lightcoral", border = "black")
```

Poisson distribution in R: Finally, we can generate random samples from a Poisson distribution using the rpois() function in R:

```
   # Generate 100 random samples from a Poisson distribution
with a mean (lambda) of 5
   samples <- rpois(100, lambda = 5)
   # Plot a histogram of the samples
   hist(samples, main = "Histogram of Random Samples from
Poisson Distribution", xlab = "Value", ylab = "Frequency", col =
"lightsalmon", border = "black")
```

Understanding variable distributions is crucial for selecting appropriate statistical tests and models, estimating parameters, and interpreting results.

2.1.4 Degrees of Freedom

Degrees of freedom is a key concept in statistical analysis and hypothesis testing that refers to the number of values in the final calculation of a statistic that are free to vary. It is crucial for determining the test statistic's distribution, calculating *p* values, and making inferences about a population based on a sample. In this discussion, we will explore the concept of degrees of freedom in different statistical scenarios, with R scripts and graphs to illustrate the concepts where necessary.

To better understand degrees of freedom, let's consider a simple example. Suppose you have a sample of five numbers, and you know that their mean is 10. If you were given the first four numbers (e.g., 8, 9, 12, and 11), you could easily calculate the fifth number (10) because the mean is fixed. In this case, there are four degrees of freedom because only four numbers are free to vary, while the fifth number is restricted by the known mean. Degrees of freedom can be thought of as the number of independent pieces of information that are used to estimate a parameter or compute a statistic. We will provide more examples to help deepen the understanding of degrees of freedom.

Mean and variance: Let's consider a sample of *n* data points ($x_1, x_2, \ldots, x_n$), where we want to compute the mean and the variance. The mean is calculated as the following:

$$mean = \left(x_1 + x_2 + \ldots + x_n\right)/n$$

When calculating the mean, no degrees of freedom are involved because the values of the data points are not restricted in any way. However, when we calculate the variance, there is a constraint: the mean value. The variance is calculated as follows:

$$variance = \frac{\sum (x_i - \bar{x})^2}{(n-1)}.$$

In this case, there are (n - 1) degrees of freedom. The "-1" is due to the fact that the mean has already been calculated, and therefore, one degree of freedom is "used up."

Linear regression: In the context of linear regression, degrees of freedom play a significant role in the estimation of parameters and the assessment of model fit. Consider a simple linear regression model with one predictor variable:

$$y = \beta_0 + \beta_1 * x + \varepsilon$$

In this model, we are trying to estimate the values of the intercept (β_0) and the slope (β_1) that minimize the sum of squared residuals. For a dataset with n observations, we have n pieces of information (the observed y values) to estimate two parameters (β_0 and β_1). Therefore, the degrees of freedom for the model are (n - 2).

The degrees of freedom are an essential concept in statistics that help us understand the constraints involved in estimating parameters and computing test statistics. As a general rule of thumb, we prefer there are more degrees of freedom when estimating parameters or computing test statistics because it helps us understand how much independent information is available for these tasks and adjust our calculations accordingly. As the degrees of freedom increase, the precision of our estimates improves. With more independent information, we can make more accurate estimates of the parameters in question. This is because the sample size is generally larger, which provides us with more data points to base our estimates on. As a result, our estimates will be less susceptible to random variation, and the overall uncertainty will be reduced. In addition, in some cases (especially regression analysis), having too few degrees of freedom can lead to overfitting, where a model captures random noise in the data rather than the underlying pattern. Overfitting occurs when the model is too complex **relative to** the available data, and it can lead to poor performance on new, unseen data. By ensuring an adequate number of degrees of freedom, we can minimize the risk of overfitting and improve the generalizability of our models.

2.2 THE CONCEPT OF HYPOTHESIS TESTING

Hypothesis testing is a fundamental aspect of scientific research and statistical analysis. It is a method used by researchers to make decisions and draw conclusions about populations based on sample data. The process involves formulating a hypothesis, collecting data, analyzing the data, and then interpreting the results to either accept or reject the hypothesis. Hypothesis testing helps researchers determine if their observations are due to random chance or if there is a genuine effect or relationship in the population.

In hypothesis testing, we start with two competing hypotheses: the null hypothesis and the alternative hypothesis. These hypotheses represent different possible explanations for the observed data, and our goal is to determine which hypothesis is more likely to be true.

2.2.1 The Null and Alternative Hypotheses

Null hypothesis (H_0): The null hypothesis, denoted as H_0, represents the "status quo" or the assumption that there is no effect or relationship between the variables being studied. It is a **baseline** hypothesis that we assume to be true unless there is sufficient evidence to suggest otherwise (in practice, the null hypothesis is often the one we aim to find evidence to reject). The null hypothesis typically states that there is no difference between groups or that the correlation between two variables is zero. For example, if we are studying the effect of urbanization on regional per capita GDP, the null hypothesis might state that the level of urbanization has no effect on regional per capita GDP.

Alternative hypothesis (H_1): The alternative hypothesis, denoted as H_1 or H_a, is the opposite of the null hypothesis. It represents the claim that there is an effect or relationship between the variables being studied. The alternative hypothesis may suggest that the difference between groups is not zero or the correlation between two variables is not zero. Continuing with the urbanization and regional per capital GDP example, the alternative hypothesis might state that the level of urbanization does have an effect on regional per capita GDP.

Once we have a clear understanding of the null and alternative hypothesis, we are ready to perform hypothesis testing. Hypothesis testing is the ultimate goal for statistical analysis so that we use our data, analyze the data with various estimating procedures, and judge from the results to inform whether we have sufficient evidence to reject or fail to reject the null hypothesis. But prior to introducing hypothesis testing, we need to understand the Type I and Type II errors and the very important concept of *p* value in statistical analysis and hypothesis testing.

2.2.2 The Type I and Type II Errors

When conducting hypothesis tests in statistics, we are often faced with the challenge of making decisions based on incomplete information. As a result, there is always a possibility of making incorrect conclusions (errors). Two common types of errors that can occur in this process are Type I and Type II errors. Understanding these errors is crucial for interpreting the results of hypothesis tests and making informed decisions.

Type I error: A **Type I** error occurs when **we reject the null hypothesis (H_0) when it is actually true**. In other words, we conclude that there is an effect or relationship between the variables being studied when, in reality, there isn't one. This is also known as a "**false positive**" or "**false alarm**."

To better understand Type I errors, let's consider an example. Imagine that a scholar is testing whether a certain environmental protection action (such as increasing the amount of environmental protection investment) is effective in increasing the regional economic performance in an area (often measured by per capita GDP or median household income). The null hypothesis (H_0) states that the action has no effect on the regional economic performance, while the alternative hypothesis (H_1) states that the action does have an effect. After collecting environmental protection and regional economic performance data over the study area, including multiple regional units, and performing a hypothesis test, the researchers find a statistically significant result and reject the null hypothesis. However, if the environmental protection action is actually ineffective and the observed effect was just due to random chance, the researchers have committed a Type I error.

The probability of making a Type I error is denoted by the Greek letter alpha (α) and is also known as the "significance level" of a hypothesis test. Researchers often choose a significance level, such as 0.05 or 0.01, to control the risk of making a Type I error. A smaller significance level means a lower probability of making a Type I error, but it also means that it will be more challenging to reject the null hypothesis when it is false.

Type II error: A Type II error occurs when we fail to reject the null hypothesis (H_0) when it is actually false. In other words, we conclude that there is no effect or relationship between the variables being studied when, in reality, there is one. This is also known as a "**false negative**" or "**missed opportunity**." Continuing with the environmental protection and regional economic performance example, suppose the environmental protection action is effective in affecting regional economic performance, but our data analysis results do not show a statistically significant effect. In this case, the researchers would fail to reject the null hypothesis, wrongly concluding that the environmental protection action is ineffective. This would be a Type II error.

The probability of making a Type II error is denoted by the Greek letter beta (β), and the "power" of a hypothesis test is defined as 1 - β. The power represents the probability of correctly rejecting the null hypothesis when it is false (**so you have the power to make the right decision**). A higher power means a lower probability of making a Type II error. To increase the power of a hypothesis test, researchers can increase the sample size, use more sensitive measurement instruments, or choose a higher significance level (α). However, increasing the significance level will also increase the risk of making a Type I error.

Balancing Type I and Type II errors: In practice, researchers need to balance the risks of Type I and Type II errors. Reducing the risk of one type of error often increases the risk of the other. The appropriate balance depends on the specific context of the study and the potential consequences of each type of error. For example, in medical research, the consequences of a Type I error (falsely concluding that an ineffective treatment works) might be more severe than the consequences of a Type II error (failing to detect the effectiveness of a truly beneficial treatment). In such cases, lowering the Type I error will have higher priority than lowering the Type II error. On the other hand, let's consider a study examining the relationship between air pollution levels and the prevalence of respiratory illnesses in different regions of a city. The goal of the study is to determine whether areas with higher air pollution levels have a higher prevalence of respiratory illnesses among the population. In this context, a Type I error would occur if the researchers reject the null hypothesis when it is actually true, concluding that there is a relationship between air pollution levels and respiratory illnesses when there isn't one. As a result, resources may be allocated to address a nonexistent issue, and other factors contributing to respiratory illnesses may be overlooked. On the other hand, a Type II error would occur if the researchers fail to reject the null hypothesis when it is false, meaning that they do not find a significant relationship between air pollution levels and respiratory illnesses when there actually is one. In this case, the consequences of the Type II error can be more severe than a Type I error. This is because if a Type II error is made, the true relationship between air pollution levels and respiratory illnesses is not identified. Consequently, no action will be taken to address the air pollution problem in the affected regions, and the public health policies will not target the root cause of the increased prevalence of respiratory illnesses. This can lead to a continuous increase in respiratory illnesses and the associated healthcare costs, as well as a decline in the overall quality of life for the residents of the affected regions.

Type I and Type II errors are inherent risks in hypothesis testing. Understanding these errors is essential for interpreting the results of hypothesis tests and making informed decisions. Researchers must carefully consider the potential consequences of each type of error in the context of their study and weigh the risks accordingly.

When designing a study, researchers can take steps to minimize the risks of Type I and Type II errors. Some of these steps include the following:

- **Choosing an appropriate significance level (α):** Selecting a lower significance level will reduce the risk of making a Type I error but may increase the risk of a Type II error. The chosen significance level should reflect the potential consequences of each type of error in the context of the study.
- **Increasing sample size:** A larger sample size generally reduces the risk of both Type I and Type II errors by providing more data points and increasing the power of the hypothesis test. However, increasing sample size may also require additional time, effort, and resources.
- **Employing more sensitive measurement instruments:** Using more precise and accurate measurement tools can help to reduce measurement errors and improve the reliability of the data. This can, in turn, reduce the risks of both Type I and Type II errors.
- **Conducting additional studies:** If the results of a single study are inconclusive or the risks of Type I and Type II errors are high, researchers may consider conducting additional studies to gather more evidence and increase the confidence in their conclusions.

Ultimately, researchers need to balance the risks of Type I and Type II errors in the context of their specific study and the potential consequences of each type of error. By carefully considering these risks and taking steps to minimize them, researchers can improve the reliability and validity of their findings and make more informed decisions based on the results of their hypothesis tests.

2.2.3 The Probability Value: *p* Value

The *p* value, or probability value, is a fundamental concept in statistical hypothesis testing. It is the probability of observing a test statistic as extreme as or more extreme than the one obtained from the sample data, assuming that the null hypothesis is true. In other words, the *p* value measures the compatibility of the observed data with the null hypothesis. A smaller *p* value indicates stronger evidence against the null hypothesis, while a larger *p* value suggests that the observed data is consistent with the null hypothesis. We have heard students or even practitioners talking about the *p* value being the "probability that the null hypothesis is true"; this is an inappropriate indication. A hypothesis is a statement and is *hypothesized* to be always true. In the sense of the *p* value that measures the compatibility of the observed data with the null hypothesis, it might be more appropriate to call the *p* value the "probability of making a Type I error."

Interpreting the *p* value: To interpret the *p* value, it is compared to a predetermined significance level, commonly denoted by alpha (α). The significance level is a threshold chosen (quite arbitrarily) by the researcher, usually set at 0.05 or 0.01, which represents the probability of making a Type I error (i.e., rejecting the null hypothesis when it is actually true). If the *p* value is less than or equal to the significance level ($p \le \alpha$), the null hypothesis is rejected, and the researcher concludes that there is evidence supporting the alternative hypothesis. If the *p* value is greater than the significance level ($p > \alpha$), the null hypothesis is not rejected and the researcher concludes that there is insufficient evidence to support the alternative hypothesis.

Let's consider a spatial study that examines whether there is a significant difference in average annual rainfall between two regions, A and B. The null hypothesis (H_0) states that there is no difference in average annual rainfall between the two regions, while the alternative hypothesis (H_1) states that there is a difference. Suppose we have collected rainfall data from both regions over the past ten years. We can use R to perform a two-sample *t* test to compare the average annual rainfall in both regions and calculate the *p* value.

First, let's create some sample data for the two regions:

```
   # Sample data for regions A and B
   rainfall_region_A <- c(1200, 1300, 1100, 1250, 1350, 1275,
1150, 1325, 1230, 1290)
   rainfall_region_B <- c(1450, 1400, 1500, 1550, 1375, 1425,
1380, 1530, 1460, 1390)
   Next, we can perform the two-sample t-test and calculate the
p-value:
   # Perform two-sample t-test
   t_test_result <- t.test(rainfall_region_A, rainfall_region_B)
   # Extract p-value
   p_value <- t_test_result$p.value
```

Now, let's say we choose a significance level of 0.05. We can compare the *p* value to the significance level to make a decision:

```
   # Compare p-value to significance level (alpha)
   alpha <- 0.05
   if (p_value <= alpha) {
    cat("Reject the null hypothesis. There is evidence of a
significant difference in average annual rainfall between the
two regions.")
   } else {
    cat("Do not reject the null hypothesis. There is
insufficient evidence to support a significant difference in
average annual rainfall between the two regions.")
   }
```

In this example, the *p* value helps us determine whether the observed difference in average annual rainfall between the two regions is statistically significant or whether it could have occurred by chance alone. By comparing the *p* value to the chosen significance level, we can make an informed decision about the null and alternative hypotheses, guiding our conclusions and recommendations based on the results of the study.

2.2.4 Hypothesis Testing

Now we have a good understanding of the Type I and Type II errors and the meaning of the *p* value, we are now able to comprehend statistical hypothesis testing. Hypothesis testing is a fundamental concept in statistics used to determine whether there is enough evidence in a sample of data to infer that a certain condition holds true for an entire population. It involves making an initial assumption, the null hypothesis (H_0), and then determining whether the observed data provides enough evidence to reject this assumption in favor of an alternative hypothesis (H_1).

In statistical analysis, hypothesis testing often takes these following steps:

- **Formulate the null hypothesis (H_0):** The null hypothesis often represents the absence of an effect (relationship) or no difference between groups. For example, if we are studying whether property's distance to parks will have an impact on property values, the null hypothesis might be that there is no effect between the distance to the park and the values.
- **Formulate the alternative hypothesis (H_1):** The alternative hypothesis is the opposite of the null hypothesis, representing the presence of an effect or a difference between groups. In the distance to parks and property values example, the alternative hypothesis would be that there is an effect between property's distance to the park and the property values.
- **Choose a significance level (α):** The significance level is a threshold, usually set at 0.05 or 0.01, which represents the probability of making a Type I error (i.e., rejecting the null hypothesis when it is actually true). The significance level helps control the risk of drawing false conclusions.
- **Collect sample data and calculate a test statistic:** The test statistic measures the extent to which the observed data deviates from what would be expected under the null hypothesis. Different hypothesis tests use different test statistics, depending on the data and the research question.
- **Calculate the *p* value:** The *p* value is the probability of observing a test statistic as extreme as or more extreme than the one obtained from the sample data, assuming that the null hypothesis is true. A smaller *p* value indicates stronger evidence against the null hypothesis, while a larger *p* value suggests that the observed data is consistent with the null hypothesis.
- **Make a decision:** Compare the *p* value to the significance level (α). If the *p* value is less than or equal to α ($p \leq \alpha$), reject the null hypothesis and conclude that there is evidence supporting the alternative hypothesis. If the *p* value is greater than α ($p > \alpha$), do not reject the null hypothesis and conclude that there is insufficient evidence to support the alternative hypothesis.

Consider this example: Suppose a researcher wants to investigate whether there is a relationship between the distance to the nearest park and property prices in a city. Common sense is that properties closer to parks will have higher prices due to the increased desirability of living near green spaces. The null hypothesis will then be that there is no effect of properties' distance to parks on property values.

First, the researcher collects data on the distance from a random sample of 100 properties to the nearest park (in meters) and their corresponding property prices. Then, we can use a simple linear regression analysis in R to examine the relationship between the distance to the nearest park (independent variable) and property prices (dependent variable). Let's create some sample data:

```
   # Set seed for reproducibility
   set.seed(123)
   # Sample data for distance to the nearest park (in meters)
   distance_to_park <- runif(100, min = 50, max = 5000)
   # Generate property prices based on a simple linear
relationship with added noise
   property_prices <- 600000 - 0.1 * distance_to_park +
rnorm(100, mean = 0, sd = 50000)
   Next, we can perform the simple linear regression analysis:
   # Perform linear regression analysis
   model <- lm(property_prices ~ distance_to_park)
```

Now, we can extract the model summary and *p value* associated with the independent variable (distance_to_park):

```
   # Extract model summary
   model_summary <- summary(model)
   # Extract p-value for the independent variable
   p_value <- model_summary$coefficients[2, 4]
```

Let's say we choose a significance level of 0.05. We can compare the *p* value to the significance level to make a decision:

```
   # Compare p-value to significance level (alpha)
   alpha <- 0.05
   if (p_value <= alpha) {
    cat("Reject the null hypothesis. There  is evidence of a
significant relationship between distance to the nearest park
and property prices.")
   } else {
    cat("Do not reject the null hypothesis. There is
insufficient evidence to support a significant relationship
between distance to the nearest park and property prices.")
   }
```

In this example, the hypothesis test helps the researcher determine whether the observed relationship between distance to the nearest park and property values is statistically significant or whether it could have occurred by chance alone. By comparing the *p* value to the chosen significance level, the researcher can make an informed decision about the null and alternative hypotheses, guiding their conclusions and recommendations based on the results of the analysis.

2.3 EXPLORATORY DATA ANALYSIS

Exploratory data analysis (EDA) is a critical step in the data analysis process that involves the initial examination and visualization of data to gain insights, identify patterns, and reveal potential relationships or anomalies. It helps researchers understand the structure of the data and formulate hypotheses for further analysis. Here are some key aspects of EDA:

- **Descriptive statistics:** In EDA, researchers often start by calculating basic descriptive statistics for each variable. These include measures of central tendency (mean, median, mode), dispersion (range, variance, standard deviation), and shape (skewness, kurtosis). Descriptive statistics provide a summary of the data and help researchers identify trends, outliers, or possible data entry errors.
- **Data visualization:** Visualizing the data through various graphs and charts is an essential component of EDA. Common visualizations include histograms, boxplots, scatterplots, and bar charts. These visualizations can help researchers identify patterns, relationships between variables, and areas of interest for further investigation. For example, a researcher studying the relationship between customer satisfaction and product quality might create a scatterplot to see if there is a discernable pattern between the two variables. A histogram could be used to visualize the distribution of customer satisfaction scores, revealing potential issues or areas for improvement.
- **Data exploration:** EDA often involves examining the distribution and relationships among variables in the dataset. This can include investigating subgroups of the data, calculating summary statistics for different categories, or creating visualizations to compare different groups. For instance, a researcher investigating the impact of an educational intervention on student performance might compare the test scores of students who received the intervention to those who did not. This information can help identify the effectiveness of the intervention and inform future educational policies or practices.
- **Data transformation and normalization:** During EDA, researchers might discover that some variables have skewed distributions or extreme values that could impact subsequent analyses. In such cases, data transformation (e.g., logarithmic, square root) or normalization (e.g., min-max scaling, *z* score standardization) techniques can be applied to adjust the data and create a more suitable input for further analysis.
- **Correlation analysis:** EDA often involves examining the relationships between pairs of variables using correlation coefficients, such as Pearson's correlation or Spearman's rank correlation. These coefficients provide a measure of the strength and direction of the relationship between two variables, helping researchers identify potential predictors or confounding factors in their study.

EDA is a crucial and informative step in the data analysis process across various disciplines. It allows researchers to gain a deep understanding of the data, identify patterns and relationships, and formulate hypotheses for further investigation. By using a combination of descriptive statistics, data visualization, and data exploration techniques, researchers can effectively explore their data and make informed decisions about subsequent analyses and research questions.

2.3.1 Plotting Your Data: Histogram, Boxplot, Scatterplot, Trend Line

Visualizing your data using various types of plots is an essential part of exploratory data analysis. In many studies, different types of plots can help researchers identify patterns, relationships, and anomalies in the data. In this section, we will discuss the following types of plots with examples and provide R scripts to generate them.

Histogram: A histogram is a graphical representation of the distribution of a dataset. It is an estimate of the probability distribution of a continuous variable. A histogram can be used to visualize the distribution of variables such as temperature, precipitation, or pollution levels. Consider this example R script that generates a histogram for visualizing the distribution of annual rainfall in a region:

```
   # Sample data representing annual rainfall in mm
   rainfall <- c(800, 1200, 950, 1100, 900, 850, 1300, 1150,
1050, 1000)
   # Create a histogram
   hist(rainfall, main="Histogram of Annual Rainfall",
xlab="Rainfall (mm)", ylab="Frequency", col="lightblue",
border="black")
```

Boxplot: A boxplot is a standardized way of displaying the distribution of data based on a five-number summary ("minimum," first quartile (Q1), median, third quartile (Q3), and "maximum"). It can be used to detect outliers and understand the spread of the data. For example boxplots can be useful for comparing the distribution of variables such as air quality index (AQI) across multiple locations, as the following R script does for different cities:

```
   # Sample data representing AQI for three cities
   city1 <- c(35, 50, 65, 55, 40, 70, 80, 45, 60, 75)
   city2 <- c(55, 60, 75, 80, 90, 85, 95, 100, 110, 105)
   city3 <- c(30, 45, 60, 50, 55, 65, 75, 80, 70, 90)
   # Create a boxplot
   boxplot(city1, city2, city3, names=c("City 1", "City 2",
"City 3"), main="AQI Comparison", xlab="Cities", ylab="AQI",
col="lightblue")
```

Scatterplot: A scatterplot is a graph that displays the relationship between two variables using Cartesian coordinates. Scatterplots are commonly used to investigate the relationship between variables such as temperature and CO_2 emissions or population density and air pollution, as in the scatterplot generated by the R script that follows:

```
   # Sample data representing population density (people per sq.
km) and air pollution levels (PM2.5)
   population_density <- c(500, 700, 600, 800, 900, 400, 1000,
750, 850, 950)
   air_pollution <- c(35, 50, 40, 55, 65, 30, 70, 45, 60, 75)
   # Create a scatterplot
   plot(population_density, air_pollution, main="Population
Density vs. Air Pollution", xlab="Population Density (people/sq.
km)", ylab="Air Pollution (PM2.5)", pch=19, col="blue")
```

Trend lines: Trend lines are used to identify the general direction of the data and can be added to scatterplots to show the relationship between two variables more clearly. In many studies, trend lines can help in understanding the correlation between variables, such as the impact of urbanization on water pollution levels. Consider this R script that examines the relationship between population density and air pollution levels with a trend line:

```
   # Sample data representing population density (people per sq.
km) and air pollution levels (PM2.5)
   population_density <- c(500, 700, 600, 800, 900, 400, 1000,
750, 850, 950)
   air_pollution <- c(35, 50, 40, 55, 65, 30, 70, 45, 60, 75)

   # Create a scatterplot
   plot(population_density, air_pollution, main="Population
Density vs. Air Pollution", xlab="Population Density (people/sq.
km)", ylab="Air Pollution (PM2.5)", pch=19, col="blue")

   # Add a trend line
   trend_line <- lm(air_pollution ~ population_density)
   abline(trend_line, col="red", lwd=2)
```

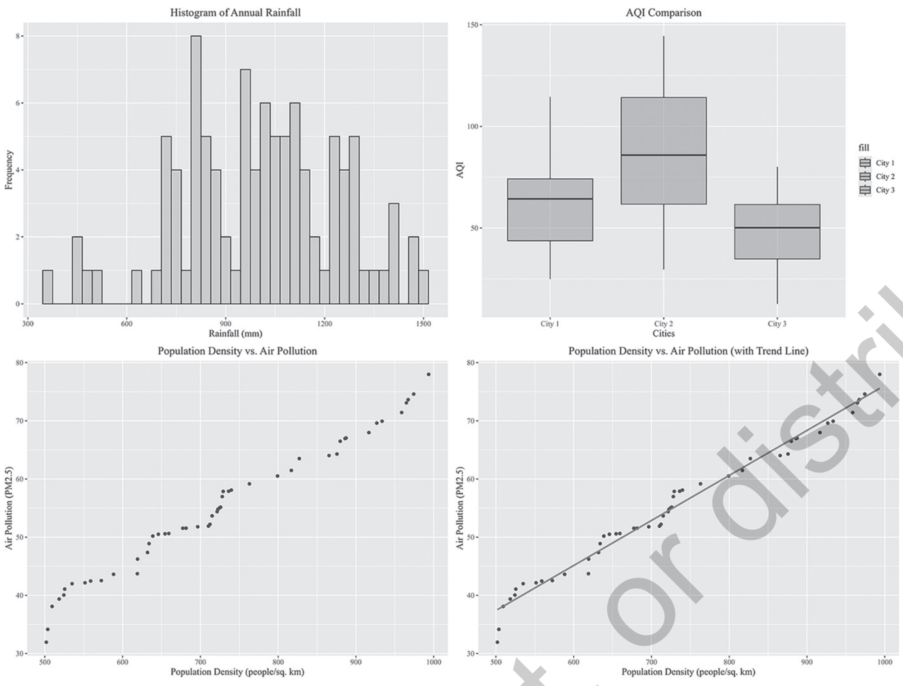
All four graphs are produced in Figure 2.3.

2.3.2 Listen to the Data

In today's world, we are experiencing an explosion of data availability. With advancements in technology, data collection, and storage capabilities, researchers and analysts have access to vast amounts of data from various domains. This abundance of data has created new opportunities for scientific inquiry, but it also poses challenges, particularly in situations where existing theories may not adequately explain the observed phenomena. In such instances, the concept of "listening to the data" becomes a crucial scientific practice.

"Listening to the data" is practically what EDA is for. It emphasizes the importance of being receptive to the insights and patterns that the data reveals rather than relying solely on established theories or preconceived expectations. This approach is especially valuable

FIGURE 2.3 ■ The Four Commonly Used EDA Graphs



when existing theories are insufficient to explain complex patterns or when new hypotheses need to be generated.

To effectively listen to the data, researchers must adopt a flexible, open-minded attitude, allowing the data to guide their inquiry. Here are some key practices for embracing this approach:

- Stay curious: Researchers should be willing to explore the data without preconceived notions, allowing unexpected patterns or relationships to emerge.
- Develop domain knowledge: Having a strong understanding of the subject matter can help researchers identify relevant variables, ask pertinent questions, and interpret the patterns they observe in the data.
- Iterate and refine: The process of listening to the data often involves iterating through multiple rounds of analysis, visualization, and hypothesis generation. Researchers should be prepared to refine their questions and hypotheses as new insights emerge from the data.
- Collaborate with diverse teams: Working with interdisciplinary teams can facilitate a more comprehensive understanding of the data, as team members bring different perspectives and expertise to the table.
- Leverage advanced analytics and machine learning: Modern analytical tools and techniques, such as machine learning algorithms and artificial intelligence, can help researchers uncover hidden patterns and structures within the data, enabling them to gain deeper insights.

By listening to the data, researchers reap numerous benefits, including the following:

- Discover new phenomena: Data-driven exploration can lead to the identification of previously unknown relationships, structures, or patterns, paving the way for groundbreaking discoveries.
- Enhance predictive capabilities: By uncovering complex relationships within the data, researchers can improve their ability to predict future events or trends, enabling more informed decision-making.
- Uncover hidden biases: Listening to the data can help researchers identify and address potential biases in their datasets, leading to more accurate and reliable results.
- Increase collaboration: Data-driven research encourages collaboration across disciplines, as researchers from diverse backgrounds work together to explore and interpret the data.
- Empower the scientific community: Listening to the data can democratize the scientific process, as researchers with varying levels of expertise can contribute to the analysis and interpretation of large datasets.

2.4 HAVE A TASTE OF REGRESSION ANALYSIS

In this chapter, we have discussed various aspects of basic statistics, laying the foundation for further exploration of more advanced techniques. One such technique, which holds great importance in the realm of data analysis, is regression analysis. This section aims to provide an accessible and intuitive understanding of regression analysis, placing it in the broader context of data analysis. We do not, however, intend to provide a comprehensive discussion of regression analysis but merely a fundamental conceptual build-up to prepare you for studies with spatial data in later chapters.

Regression analysis is a fundamental statistical method that allows us to examine the relationships between variables and make predictions based on these relationships. It plays a crucial role in various scientific disciplines, such as geography, environmental studies, and social sciences. By exploring regression analysis, we can build upon the foundational concepts introduced in this chapter and delve deeper into the world of data analysis.

The core concept of regression analysis is to model the relationship between a dependent variable (also referred to as the response or outcome variable) and one or more independent variables (also called predictors or explanatory variables). The dependent variable is the one we aim to predict or explain, while the independent variables are the factors that influence it. The primary goal is to establish a mathematical model that accurately represents the relationship between the dependent and independent variables, allowing for meaningful insights and accurate predictions.

One of the most basic forms of regression analysis is simple linear regression, which models the relationship between one dependent variable and one independent variable. The relationship is represented by a straight line, which is fitted to the data in such a way that the differences between the observed and predicted values of the dependent variable are minimized. Simple linear regression serves as an essential building block for understanding more complex types of regression analysis, such as multivariate linear regression. Multivariate linear regression extends simple linear regression to include multiple independent variables. This type of regression enables us to examine the combined effects of several predictors on the dependent variable and assess the relative importance of each predictor. In doing so, we can gain a more comprehensive understanding of the factors influencing the dependent variable.

As we move from basic statistics to more advanced analytical techniques like regression analysis, it is crucial to keep in mind the foundational concepts and principles we have learned thus far. These core ideas, such as understanding variables, distributions, hypothesis testing, and exploratory data analysis, serve as the groundwork upon which more sophisticated techniques are built.

When conducting regression analysis, it is essential to ensure that the data is accurate, reliable, and representative. The quality of the data has a significant impact on the validity of the analysis, and as we have discussed in this chapter, it is crucial to practice good data management and cleaning techniques.

2.4.1 The Most Basic Concepts of Regression Analysis

In this subsection, we will introduce some of the most fundamental concepts of regression analysis. The goal of regression analysis is to establish a mathematical model based on available data that can represent as accurately as possible the relationship between these variables, enabling us to make predictions and gain insights from observed information. To illustrate the regression analysis, we use simple regression as an illustration. Extension to multivariate regression is straightforward.

The intercept and slope: In a simple linear regression model, we aim to fit a straight line to the data points. The equation for a straight line is given by the following equation:

$$y = \beta_0 + \beta_1 * x + \varepsilon$$

Here, y is the dependent variable, x is the independent variable, β_0 is the intercept, β_1 is the slope, and ε is the residual (error term).

The intercept (β_0) represents the value of the dependent variable when the independent variable is equal to zero. For instance, if we were modeling the relationship between altitude (x) and temperature (y), the intercept would represent the estimated temperature at sea level (i.e., when altitude is zero). The slope (β_1) represents the change in the dependent variable corresponding to a one-unit change in the independent variable. In our altitude–temperature example, the slope indicates the change in temperature for every one-unit increase in altitude (e.g., for every 100 meters of ascent). A positive slope suggests that as the independent variable increases, so does the dependent variable, while a negative slope indicates an inverse relationship between the two variables.

Residuals: The residual (ε) in the regression equation represents the difference between the observed value of the dependent variable and the predicted value based on the fitted line. In other words, it measures the vertical distance between the actual data point and the corresponding point on the regression line. Residuals can be both positive and negative, depending on whether the observed value is above or below the regression line. Residuals play an important role in regression analysis, as they provide information about the variation in the dependent variable that is not explained by the independent variable(s).

In regression analysis, we often assume that the relationship between the dependent and independent variables can be modeled as a deterministic function plus a stochastic (random) component. The deterministic function represents the systematic relationship between the variables, while the stochastic component accounts for random variation that cannot be explained by the deterministic function alone. The residuals can be thought of as realizations of this stochastic component. They represent the part of the dependent variable's variation that remains unexplained after accounting for the effects of the independent variable(s). By examining the residuals, we can gain insights into the structure of the stochastic process and assess the adequacy of our regression model.

As the representation of the stochastic process, the residuals often are assumed to have some very important properties.

Independence: The residuals should be independent of each other. This means that the value of one residual should not depend on the value of another residual. If the residuals are not independent, it may indicate that some important information has not been captured by the independent variable(s) or that the underlying assumptions of the model are not met.

Normality: In many regression models, it is assumed that the residuals follow a normal distribution with a mean of zero and constant variance. This assumption allows us to perform hypothesis tests and construct confidence intervals. If the residuals do not appear to be normally distributed, it may be necessary to consider alternative regression models or transformations of the data.

Homoscedasticity: The residuals should exhibit constant variance across different observations of the independent variable(s). This property, known as *homoscedasticity*, is important for ensuring the validity of the model's statistical inference. If the residuals display non-constant variance (*heteroscedasticity*), it may indicate that the model does not adequately capture the relationship between the dependent and independent variables or that the data require transformation.

These important properties are the foundations for using data to estimate the unknown intercept and slope(s). By carefully examining the residuals and their properties, we can assess the quality of our regression model and identify any potential issues or areas for improvement. This, in turn, helps to ensure that our model provides an accurate representation of the relationship between the dependent and independent variables, allowing us to make reliable predictions and draw meaningful conclusions.

2.4.2 The Ordinary Least Squares Estimator

The ordinary least squares (OLS) estimator is a widely used method for fitting a linear regression model to data. The aim of linear regression is to find the best-fitting line that describes the relationship between the dependent variable (Y) and one or more independent variables (X). Y is a vector of dependent variables, with elements being y_i. X is a matrix of independent variables, with elements being x_{ij}. $i = 1, \ldots, n$, n is the number of observations, and $j = 1, \ldots, m$, m is the number of independent variables. The OLS estimator achieves this by minimizing the sum of the squared differences between the observed values of the dependent variable and the predicted values based on the fitted line.

The equation for a simple linear regression model, with one independent variable (j = 1), can be represented as

$$y_i = \beta_0 + \beta_1 * x_i + \varepsilon$$

where

y_i is the ith dependent variable,

x_i is the ith independent variable,

β_0 is the intercept (the value of y_i when $x_i = 0$),

β_1 is the slope (the change in y_i for a unit change in x_i),

ε is the residual (the difference between the observed value of y_i and the predicted value based on the model).

To estimate the coefficients (β_0 and β_1) using OLS, we minimize the sum of the squared residuals:

$$min\sum\left(y_i - \left(\beta_0 + \beta_1 * x_i\right)\right)^2$$

To achieve this, we compute the partial derivatives of the sum of squared residuals with respect to β_0 and β_1, and set them equal to zero. Solving these equations, we get the OLS estimators for β_0 and β_1:

$$\beta_1 = \frac{\sum\left(\left(x_i - \overline{X}\right)*\left(y_i - \overline{Y}\right)\right)}{\sum\left(x_i - \overline{X}\right)^2}.$$
$$\beta_0 = \overline{Y} - \beta_1 * \overline{X}$$

$\overline{X}$ and $\overline{Y}$ are the mean values of X and Y. This process is easily implemented in R with simulated data:

```
   # Generate some sample data
   set.seed(42) # So that you can repeat the script and obtain
the same results.
   X <- seq(0, 10, by = 0.1)
   Y <- 2 + 3 * X + rnorm(length(X), mean = 0, sd = 2)
   # Fit the OLS model
   model <- lm(Y ~ X)
   # Display the model coefficients
   summary(model)$coefficients
```

The estimated β_1 and β_0 will be very similar to the known values (2 and 3). You can also create a scatterplot of the data and add the fitted regression line:

```
  library(ggplot2)
  # Create a scatterplot of the data
  ggplot(data = data.frame(X, Y), aes(x = X, y = Y)) +
   geom_point() +
   geom_smooth(method = "lm", se = FALSE, linetype = "solid",
color = "blue", size = 1) +
   labs(x = "X", y = "Y") +
   theme(text = element_text(size = 12, family = "Times New
Roman"))
```

2.4.3 The Maximum Likelihood Estimator

Maximum likelihood estimation (MLE) is an alternative method to OLS for estimating the parameters of a statistical model. While OLS is mainly used for linear regression, MLE offers a more flexible approach that can be applied to a wide range of models, including those with non-normal error distributions, non-independent residuals, or non-linear relationships between variables.

The idea behind MLE is to find the parameter values that make the observed data as probable as possible. In other words, we want to maximize the likelihood or the probability of obtaining the data given the model's parameters. To achieve this, we first define a likelihood function, which represents the probability of the observed data given certain parameter values. We then seek the values of the parameters that maximize this function.

In a simple linear regression context, let's say our model is the following:

$$Y = \beta_0 + \beta_1 * X + \varepsilon$$

For better presentation, we use the matrix forms for both Y and X. Y is the vector of dependent variable, X is the matrix of independent variable, β_0 is the intercept, β_1 is the slope, and ε represents the error term. In OLS, we minimize the sum of squared residuals to estimate the parameters β_0 and β_1. However, in MLE, we estimate these parameters by maximizing the likelihood function. The likelihood function is given by assuming that the error term (ε) is normally distributed with a mean of 0 and a constant variance (σ^2). Under this assumption (normality assumption), the probability density function (PDF) of a normally distributed error term is given by the following equation:

$$f(\varepsilon) = \left(1/\left(\sigma * \sqrt{(2 * \pi)}\right)\right) * e^{(-(\varepsilon^2)/(2*\sigma^2))}$$

Since $\varepsilon = Y - (\beta_0 + \beta_1 * X)$, we can rewrite the PDF in terms of Y, X, β_0, and β_1 as follows:

$$f(Y) = \left(1/\left(\sigma * \sqrt{2 * \pi}\right)\right) * e^{(-(Y-(\beta_0+\beta_1*X))^2/(2*\sigma^2))}$$

Now, to construct the likelihood function $L(\beta_0, \beta_1)$, we multiply the probability densities for each data point (Y_i, X_i) in our sample, which is the joint likelihood of the observed data:

$$L(\beta_0,\beta_1) = \prod f(Y_i)$$

The likelihood function, $L(\beta_0,\beta_1)$, represents the joint probability of all the observed data points given the model's parameters (β_0 and β_1). The goal of MLE is to find the values of β_0 and β_1 that maximize this likelihood function. In practice, it's more common to work with the log-likelihood function, as it simplifies the calculations and avoids numerical issues by taking a logarithmic transformation of the multiplication, which transforms the multiplication to addition.

It is essential to note that for linear regression, when the errors are normally and independently distributed, MLE and OLS will provide the same estimates for the model parameters. However, MLE becomes particularly useful when the assumptions of OLS, such as normality of errors, *independence of the error*, are not met or when dealing with non-linear models like logistic regression.

MLE is a versatile and powerful method for estimating the parameters of a statistical model. While OLS is specifically designed for linear regression and assumes normally and independently distributed errors, MLE can be applied to various models, including those with non-normal error distributions or non-linear relationships. The choice between OLS and MLE depends on the specific assumptions of the problem and the nature of the data being analyzed.

2.4.4 Read and Interpret the Regression Outputs

Interpreting regression outputs is an essential skill when conducting data analysis. In this section, we will discuss how to interpret the results of a simple linear regression using R, including coefficients, *t* statistics, *p* values, *R* squared, and residual diagnostics.

Let's use an example to illustrate. We will generate a synthetic dataset with a linear relationship between two variables: the distance to the city center (X) and the price of a house (Y).

```
# Generate synthetic data
set.seed(123)
n <- 100
X <- seq(1, n, 1)
Y <- 300000 - 5000 * X + rnorm(n, mean = 0, sd = 10000)
# Fit a simple linear regression model
model <- lm(Y ~ X)
# Display the summary of the model
summary(model)
```

Now let's interpret the regression output.

Coefficients: The coefficients are the estimated values for the intercept (β_0) and slope (β_1) in the linear equation $Y = \beta_0 + \beta_1 * X$. In our example, the intercept represents the estimated price of a house at zero distance from the city center (you can interpret this as the houses are right next to the city center), while the slope indicates the change in house price per unit increase in distance. These values are essential for interpreting regression outputs and using the outputs for decision-making.

```
# Extract the coefficients
coef(model)
```

Standard error (std. error), *t* statistics, and *p* values: The *t* statistic is used to determine whether a given coefficient is significantly different from zero. The *t* statistic is the quotient of the estimate divided by the standard error. We usually want the standard error to be small compared to the estimates so that we have more confidence in the estimates (a smaller standard error means narrower estimating range that the estimated coefficient will fall in, hence more confidence). So a large *t* statistic (which will always be related to a small *p* value) suggests that the variable has a significant effect on the dependent variable. The *p* value is the probability of obtaining a *t* statistic at least as extreme as the observed one, given that the null hypothesis (no effect) is true (recall the *p* value is the probability of making a Type I error). If the *p* value is less than a predetermined significance level (e.g., 0.05), we will have a higher confidence to reject the null hypothesis and conclude that the variable is significant.

```
# Extract the t-statistics and p-values
summary(model)$coefficients[, "t value"]
summary(model)$coefficients[, "Pr(>|t|)"]
```

***R* squared:** The *R* squared (or coefficient of determination) measures the proportion of the total variation in the dependent variable that is explained by the regression model. A higher *R* squared indicates a better fit, but it does not guarantee a meaningful relationship or an appropriate model. It is important to consider other diagnostic measures and the context of the research question when interpreting *R*-squared values. In addition, the *R*-squared value can no longer be calculated if the residuals are not independently distributed, which is usually the case when data are collected over geographic spaces (geographic datasets). We will discuss this situation in more detail in later chapters.

```
# Extract the R-squared
summary(model)$r.squared
```

Residual diagnostics: Checking the residuals (observed values minus predicted values) can help assess the model's assumptions and detect potential problems. Common residual diagnostics include examining the distribution of residuals and plotting the residuals against the predicted values or independent variables. In *R*, you can use the following commands:

```
   # Histogram of residuals
   hist(residuals(model), main = "Histogram of Residuals", xlab
= "Residuals")
   # Scatterplot of residuals vs. predicted values
   plot(predict(model), residuals(model), main = "Residuals
vs. Predicted Values", xlab = "Predicted Values", ylab =
"Residuals")
   abline(h = 0, col = "red")
```

These graphs can help us evaluate whether the residuals are normally distributed and whether there is any pattern in the residuals, which might indicate problems with the model.

Interpreting regression analysis results is critical for many applications and will be crucial for later chapters when we introduce spatial regression analysis, eigenfunction-based spatial filtering regression analysis, and geographically weighted regression analysis.

2.5 PRACTICES IN R

In this section, we will summarize the concepts introduced so far and provide hands-on practices in R to facilitate your understanding of the fundamental statistical concepts and techniques used in data analysis. We will cover the following topics:

- Creating and manipulating variables
- Understanding variable distributions
- Exploratory data analysis
- Basic concepts of regression analysis
- Interpreting regression outputs
- Hypothesis testing
- Confidence intervals

Practice 1: Creating and Manipulating Variables

Purpose: Create a vector population representing the population of five cities in thousands: 120, 500, 300, 200, 800. Then, create a second vector area representing the area of each city in square kilometers: 50, 200, 120, 80, 400. Finally, calculate the population density for each city and store it in a vector called pop_density.

```
   population <- c(120, 500, 300, 200, 800)
   area <- c(50, 200, 120, 80, 400)
   pop_density <- population / area
   pop_density
```

In this practice, we create two vectors representing the population and area of five cities. By calculating the population density using element-wise division, we obtain a new vector representing the density of each city. This hands-on exercise provides a concrete example of how to manipulate and analyze data in R while also demonstrating the importance of understanding relationships between variables. Through this practice, you are expected to learn to create and manipulate vectors in R, perform mathematical operations on the data, and interpret the results. This foundational skill can be applied to various data analysis tasks in the future, as you will often need to perform calculations or transformations on your own data in order to answer research questions or reveal patterns.

Furthermore, this exercise highlights the importance of properly interpreting the results of data manipulation. By understanding the relationship between population, area, and population density, we can draw meaningful insights from the data and use it to inform decision-making in real-world contexts. As we progress in our data analysis journey, we can build upon this practice by exploring more advanced techniques, such as visualizing the data using plots or maps, performing statistical tests to determine the significance of observed relationships, or incorporating additional variables to gain a deeper understanding of the data. By mastering these skills, we will be better equipped to tackle complex data analysis tasks and draw meaningful conclusions from our findings.

Practice 2: Understanding Variable Distributions

Purpose: Create a vector random_numbers containing 100 random numbers from a normal distribution with a mean of 10 and a standard deviation of 2. Then, plot a histogram of the generated numbers.

```
   set.seed(42)
   random_numbers <- rnorm(100, mean = 10, sd = 2)
   hist(random_numbers, main = "Histogram of Random Numbers",
xlab = "Value", ylab = "Frequency", col = "lightblue", border =
"black")
```

In this practice, we generate a sample of 100 random numbers from a normal distribution using the rnorm() function. Our goal is to understand how random sampling works and to gain experience in visualizing the underlying distribution of our data. To accomplish this, we create a histogram that allows us to observe the distribution of these numbers and explore the characteristics of the normal distribution.

By working with a sample of random numbers, we develop an intuitive understanding of the concept of random sampling, which is a fundamental aspect of many statistical analyses. We have learned how to use R functions to generate random samples from various probability distributions, which can be useful in simulating data or testing the performance of different statistical methods.

The histogram we create in this practice serves as a visual representation of our data, enabling us to identify the shape, center, and spread of the distribution. This hands-on experience in data visualization helps us develop the skills necessary to create meaningful and informative plots that can be used to communicate our findings and support data-driven decision-making. By engaging in this practice, we not only gain valuable experience working with R but also enhance our understanding of key statistical concepts. As we progress in our data analysis journey, we can extend this practice by exploring other probability distributions, generating samples of different sizes, or incorporating more advanced visualization techniques. This foundational knowledge will be crucial in helping us tackle more complex data analysis tasks and draw accurate conclusions from our data.

Practice 3: Exploratory Data Analysis

Purpose: Using the population and area vectors created in Practice 1, create a scatterplot with population on the *x* axis and area on the *y* axis. Add a title and axis labels to the plot.

```
   plot(population, area, main = "Population vs. Area", xlab =
"Population (thousands)", ylab = "Area (sq. km)", pch = 19, col
= "blue")
```

In addition, we can apply the scatter.smooth() function to the same data, which will fit a smoothed curve to the data points in addition to the generated scatterplots, helping us detect any underlying trends or patterns more effectively.

Let's modify the previous practice by using the scatter.smooth() function:

```
   # Create a scatterplot with population on the x-axis and area
on the y-axis.
   # Add a smoothed line using the scatter.smooth() function.
   library("graphics")
   scatter.smooth(population, area, main = "Population vs.
Area", xlab = "Population (thousands)", ylab = "Area (sq. km)",
pch = 19, col = "blue")
```

By applying scatter.smooth() to our scatterplot, we can more easily identify potential relationships between population and area. The smoothed curve provides a visual representation of the possible trend in the data, which can help us make inferences about the overall association between the two variables. In this case, the curve may suggest that there is a certain relationship between population size and area, although it is crucial to note that this visualization is not a definitive measure of association.

In this practice, we showcase the significance of employing typical and important EDA techniques, including scatterplots and smoothed curves, to derive valuable insights from our data. By utilizing these visualization methods, we can delve deeper into the data to reveal underlying patterns and trends that might have been otherwise overlooked. These insights can help us make more informed decisions and develop better strategies in our analysis. As we explore our data with these visualization techniques, we gain a more comprehensive understanding of the relationships between variables, the presence of outliers, and the overall distribution of our data. This understanding is crucial in guiding our choice of statistical methods, identifying potential areas for further research, and generating hypotheses. Furthermore, these visualization techniques can help us communicate our findings more effectively to others, be it our colleagues, supervisors, or a broader audience. By presenting our data in a visually appealing and easily understandable manner, we can convey complex information and insights with clarity. This practice highlights the importance of leveraging EDA techniques like scatterplots and smoothed curves to enhance our data analysis capabilities. As we become more proficient in utilizing these visualization methods, we can better understand the intricacies of our data, ultimately leading to more robust and insightful research outcomes.

Practice 4: Basic Concepts of Regression Analysis and Interpreting Regression Outputs

Purpose: Perform a simple linear regression using the population and area vectors created in Practice 1. Store the results in a variable called regression_model and print a summary of the model.

```
regression_model <- lm(area ~ population)
summary(regression_model)
```

In this practice, we delve into the process of conducting a simple linear regression using the `lm()` function, a powerful tool in R that fits a linear model to our dataset. This is an essential step in understanding the relationship between variables and making predictions based on the fitted model. As we explore the output of the linear regression, we gain valuable insights into various aspects of the model, allowing us to evaluate its performance and make informed decisions regarding its suitability for our analysis.

The summary output generated by the `lm()` function provides a wealth of information about our linear model, such as the coefficients, residuals, and goodness-of-fit statistics. The coefficients indicate the estimated intercept and slope of the regression line, helping us understand the direction and magnitude of the relationship between our independent and dependent variables. By examining these coefficients, we can make inferences about the nature of the relationship and its potential implications for our research questions. Residuals, another key component of the summary output, represent the difference between the observed values and the values predicted by the model. Analyzing the residuals can help us identify any patterns or deviations that might suggest the presence of unexplained variation in our data or inadequacies in our model. A thorough examination of residuals can lead to a better understanding of the underlying structure of the data and inform potential improvements to our model. Goodness-of-fit statistics, such as *R* squared and adjusted *R* squared, are essential for evaluating the performance of our linear model. These statistics provide a measure of how well the model explains the variation in the data, allowing us to compare different models and select the most suitable one for our analysis.

This practice emphasizes the importance of conducting a simple linear regression using the lm() function in R and interpreting the resulting summary output. By carefully examining the coefficients, residuals, and goodness-of-fit statistics, we can gain a deeper understanding of our linear model and its implications for our research. This foundational knowledge equips us with the necessary skills to tackle more complex regression analyses and develop robust models that accurately capture the relationships in our data. Understanding linear regression and being able to interpret the outputs of the `lm()` function are crucial for later understanding of spatial data analysis that also utilizes regression techniques.

Practice 5: Hypothesis Testing

Purpose: Perform a *t* test to determine if there is a significant difference in population density between two groups of cities: Group A with populations of 100, 150, and 200 and Group B with populations of 700, 750, and 800. Use a significance level of 0.05.

```
group_a <- c(100, 150, 200)
group_b <- c(700, 750, 800)
t_test_result <- t.test(group_a, group_b)
t_test_result$p.value
```

In this practice, we engage in the process of performing a *t* test to compare the means of population in two distinct groups of cities using the `t.test()` function in R. The *t* test is a fundamental statistical tool that allows us to assess whether the means of two groups are significantly different from each other, providing valuable insights into potential patterns and relationships within our data. This practice highlights the importance of hypothesis testing and understanding the underlying assumptions in our statistical analyses.

By employing the `t.test()` function, we are able to carry out the comparison of the means and obtain the test results, which include essential information about the *t* statistic, degrees of freedom, and *p* value. The *p* value is a crucial element in hypothesis testing, as it helps us determine whether there is a statistically significant difference between the two groups at a specified significance level, typically set at 0.05 or 0.01 (recall this means the chances of making a Type I error—when the null hypothesis is right, but we reject it). Interpreting the *p* value is an essential skill in statistical analysis. A small *p* value (less than the chosen significance level) indicates that the observed difference between the means is unlikely to have occurred by chance alone, and we reject the null hypothesis in favor of the alternative hypothesis. Conversely, a large *p* value (greater than the significance level) suggests that there is insufficient evidence to reject the null hypothesis (because the chance of making a Type I error is not acceptable), and we cannot conclude that a significant difference between the means exists. Understanding the implications of the *p* value and its relationship to the significance level is vital in making informed decisions based on the *t* test results. This practice demonstrates the importance of conducting hypothesis tests, such as the *t* test, to uncover meaningful insights into our data and inform the direction of our research.

While this practice specifically emphasizes the value of using *t* tests to compare the means of two groups and highlights the crucial role of interpreting the *p* value in hypothesis testing, the logic applies to all hypothesis testing and the interpreting of *p* values. By developing a solid foundation in performing hypothesis testing in R and understanding the interpretation of the results, we are better equipped to carry out robust statistical analyses and draw meaningful conclusions from our data in various research contexts.

Practice 6: Confidence Intervals

Purpose: Aside from the conventional approach of using *p* values (which involves accepting or rejecting a null hypothesis based on a predetermined level of acceptable risk for committing a Type I error), there is an alternative and more flexible method for conducting hypothesis testing known as the calculation of confidence intervals. While both *p* values and confidence intervals serve a similar purpose in a numeric context, the presentation of a confidence interval offers greater clarity and adaptability. It explicitly communicates to the audience that there is a certain percentage of confidence that the estimated values or statistics will fall within the specified interval. A *p* value, on the other hand, conveys the level of confidence in avoiding a Type I error. However, this information is often implicit and can be challenging for students to grasp fully. The confidence interval approach provides a more intuitive and understandable representation of the uncertainty surrounding an estimate, making it easier for students and other audiences to interpret the results of hypothesis testing.

In this sense, calculating confidence intervals offers a more flexible and transparent alternative to using *p* values for hypothesis testing. By presenting confidence intervals, we can more effectively communicate the level of certainty associated with our estimates, making it easier for our audiences to understand and engage with the findings of a statistical analysis. This practice will calculate the 95% confidence interval for the mean population of the population vector created in Practice 1.

```
mean_population <- mean(population)
pop_sd <- sd(population)
n <- length(population)
error_margin <- qnorm(0.975) * (pop_sd / sqrt(n))
lower_bound <- mean_population - error_margin
upper_bound <- mean_population + error_margin
c(lower_bound, upper_bound)
```

In this practice, we calculate the 95% confidence interval for the mean population of the population vector. We first calculate the mean and standard deviation of the population and then use the `qnorm()` function to find the error margin. Finally, we calculate the lower and upper bounds of the confidence interval by using the error margin. The upper bound is the error margin above the mean, and the lower bound is the error margin below the mean.

Confidence interval also serves an explicit purpose in regression analysis when we are trying to gauge whether there is a statistical relationship between the dependent variable and the independent variable(s). When we obtain the estimates of the independent variables, the standard errors of the independent variables along with a specific level of confidence can be used to calculate the error margins, and we can calculate the confidence interval in this way as well. If the value zero is not included in the confidence interval under a certain percentage, then we can be certain percentage confident that there is a statistically significant relationship between the dependent variable and the independent variable. If, on the other hand, zero is included in the confidence interval, then we cannot exclude the likelihood within our set confidence that there is a chance that the dependent variable is not statistically related with the independent variable.

These practices cover various concepts and techniques in statistics and R programming. As you progress through your studies, continue to practice and explore these concepts using different datasets (especially your own datasets) and real-world examples to deepen your understanding and improve your skills.

This chapter only serves as a very rudimentary, gentle refreshing reminder of some of the most fundamental statistical concepts that we will be encountering frequently in later chapters. This chapter will by no means replace a good introductory statistical course that could build a foundation for further statistical analysis and exploring the wonderful world of data science, especially spatial data science.

REVIEW QUESTIONS

1. What is descriptive statistics, and why is it important in data analysis?
2. Explain the concept of a probability distribution and provide an example.
3. What is the purpose of a boxplot, and how is it useful in descriptive statistics?
4. Describe how to generate a histogram in R using a data vector.
5. What is regression analysis, and why is it important in data analysis?
6. Explain the significance of coefficients and residuals in a regression analysis output.
7. Conduct a simple linear regression in R using the following simulated data: population <- c(100, 200, 300, 400, 500); area <- c(50, 60, 80, 120, 200). Provide a summary of the regression model.
8. What is a *t* test, and when would you use it in data analysis?
9. Conduct a *t* test in R to determine if there is a significant difference between the following two groups of data: group1 <- c(100, 150, 200); group2 <- c(200, 250, 300). Interpret the result.
10. Explain the concept of a confidence interval and its significance in statistical analysis.
11. Calculate the 95% confidence interval for the mean of the following dataset in R: data <- c(100, 150, 200, 250, 300). Provide the result.
12. How does hypothesis testing relate to regression analysis and confidence intervals?
13. What are some fundamental statistical concepts that will be encountered frequently in later chapters of spatial data science study?
14. How do *p* values and confidence intervals differ in their approach to hypothesis testing?
15. Why is it important to understand these fundamental statistical concepts before delving deeper into spatial data science?
16. What is the significance level, and how does it affect the results of a *t* test?
17. What are the assumptions made when performing a *t* test?
18. What does it mean if zero is included in a confidence interval?
19. What is the importance of understanding and interpreting the output of the lm() function in R?
20. What insights can be gained from a thorough examination of residuals in a regression model?