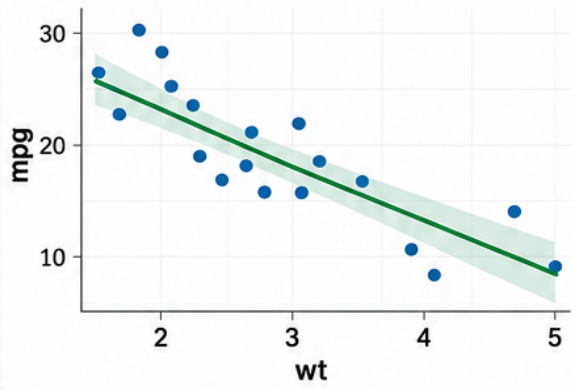
R PROGRAMMING COOKBOOK

FROM FIRST STEPS TO ADVANCED DATA SCIENCE

- R FUNDAMENTALS
- TIDYVERSE WORKFLOWS
- STATISTICAL ANALYSIS
- MACHINE LEARNING
- DATA VISUALIZATION
- REPRODUCIBLE RESEARCH
- WEB APPLICATIONS
- PACKAGE DEVELOPMENT
- PRODUCTION DEPLOYMENT

```r
library(tidyverse)
ggplot(mtcars,
  aes(wt, mpg)) +
  geom_point(color = "#2C7FB8") +
  geom_smooth(method = "lm",
    se = FALSE, color = "#1B9E77") +
  theme_minimal()
```



YURI LEBEDEV

R Programming Cookbook

From First Steps to Advanced Data Science

Steve Publications

This book is available at https://leanpub.com/rprogrammingcookbook

This version was published on 2026-08-03

Contents

From First Steps to Advanced Data Science

This book takes you from zero programming experience to confident R practitioner through clear explanations, practical examples and fully worked demonstrations. You will learn the language fundamentals, modern tidyverse workflows, statistical analysis, machine learning, data visualization, reproducible research, web applications, package development and production deployment. Every concept is explained step by step with complete, runnable code you can use immediately in your own projects.

Introduction

In 1993, two statisticians at the University of Auckland named Ross Ihaka and Robert Gentleman created a free programming language they called R. They chose the name partly as a nod to S, the commercial statistical language that inspired their work, and partly because it was short enough to type quickly. Neither could have predicted what would follow. Three decades later, R has become one of the most widely used tools in data science, statistics, bioinformatics, finance, social science research and many other fields. More than twenty thousand packages exist on CRAN alone, the central repository for R extensions, contributed by thousands of developers around the world.

You might be asking why you should learn R when Python, SQL and other languages also handle data analysis. The honest answer is that you do not need to choose just one language. Different tools excel at different tasks. R deserves your attention because it was designed from the ground up by statisticians for statistical computing, and that heritage shows in everything from its built-in probability distributions to its approach to data visualization. When you write a linear regression in R, you are using functions developed by people whose entire careers were devoted to making statistical methods correct, fast and accessible. That depth of domain expertise is hard to replicate.

At the same time, modern R is much more than a statistics calculator. It is a complete programming language with its own object-oriented systems, functional programming tools, metaprogramming capabilities and web application frameworks. The tidyverse ecosystem, created primarily by Hadley Wickham and his collaborators, has transformed how most practitioners interact with data by providing consistent, readable and powerful tools for data manipulation, visualization and modeling. Meanwhile, projects like tidymodels bring modern machine learning workflows into R with the same design philosophy.

This book is organized to take you on a deliberate journey from absolute beginner to advanced practitioner. We begin with installation and your very first program, then build up through core language concepts like variables, data types, control flow and functions. You will learn every major data structure in R: vectors, matrices, arrays, lists, factors, data frames and tibbles. From there we move into the tidyverse ecosystem that powers modern R workflows,

including dplyr for data manipulation, tidyr for reshaping, purrr for functional programming and ggplot2 for visualization.

The middle sections of the book cover statistical analysis and machine learning. You will learn how to perform hypothesis tests, fit regression models, evaluate their assumptions and interpret results. The machine learning chapters introduce supervised and unsupervised learning using tidymodels, covering classification, regression, clustering and model tuning. These chapters assume no prior statistics background but develop the concepts rigorously enough that you can apply them in research or production settings.

The final sections address professional R development. You will learn to create reproducible documents with Quarto, build interactive web applications with Shiny, work with databases and APIs, write efficient parallel code, develop and publish your own packages, and follow best practices for code style, testing and documentation. Throughout the book you will find complete, working examples that demonstrate real-world patterns you can adapt to your own problems.

A few notes about how to use this book. The code examples are designed to be copy-paste runnable with current versions of R and its packages. If you see a code block, try running it yourself, then experiment by changing values or parameters. The best way to learn R is by writing R code, not just reading about it. The chapters build on each other in sequence, so I recommend working through them in order rather than jumping around. Later chapters assume familiarity with concepts from earlier ones.

This book targets the latest stable R ecosystem as of mid-2026, including R 4.6.x and current versions of tidyverse packages. Where older approaches are still widely used or encountered in legacy code, I will mention them alongside modern alternatives. My goal is that when you finish this book, you will not just understand how to use R but also why things work the way they do, giving you the foundation to continue learning and adapting as the ecosystem evolves.

Let us begin.

Chapter 1: Getting Started with R and RStudio

Before writing a single line of code, you need R installed on your computer and an environment where you can work comfortably. This chapter walks through the setup process, introduces the tools you will use throughout this book, and gets you writing and running your first R programs.

What Is R and Why It Matters

R is a programming language designed for statistical computing, data analysis and graphics. Unlike general-purpose languages like Python or Java that happen to support data analysis through libraries, R was built from the ground up by statisticians for statistical work. That distinction matters in practice. The core R installation includes dozens of probability distributions, hypothesis tests, regression functions, and time series methods that you can call directly without loading any additional packages.

The language has grown far beyond its statistical roots. Today R is used across many domains:

- Academic research: R dominates in fields like biostatistics, ecology, psychology and epidemiology. Many scientific journals prefer or require R for statistical analysis because of its reproducibility tools and transparent methodology.
- Business analytics: Companies use R for customer segmentation, forecasting, A/B testing, and reporting dashboards.
- Finance: Quantitative analysts rely on R for risk modeling, portfolio optimization, and time series analysis.
- Bioinformatics: R is the standard language in genomics and computational biology, with Bioconductor providing thousands of specialized packages.
- Machine learning: While Python has more visibility in deep learning, R competes strongly in traditional machine learning through tidymodels and other frameworks.

What makes R particularly powerful is its package ecosystem. The Comprehensive R Archive Network, known as CRAN, hosts over twenty thousand contributed packages. These are not just utilities but complete, documented, peer-reviewed software tools maintained by experts. Need to analyze survival data? There are several dedicated packages. Want to scrape web data? Multiple options exist. The community has likely already solved problems similar to yours.

R is also open source and free, distributed under the GNU General Public License. Anyone can download it, modify it, and redistribute it. This openness has fueled rapid innovation and broad adoption across academia, industry and government.

Installing R and RStudio

You need two things: R itself, which is the programming language runtime, and an integrated development environment, or IDE, where you write and run your code. The most popular choice for R is RStudio, now owned by Posit. While you can use other editors, RStudio provides features specifically designed for R that make the learning experience much smoother.

The current stable release of R as of mid-2026 is version 4.6.x. You should always install the latest stable version to benefit from bug fixes, performance improvements and new language features.

To install R, visit the official CRAN website at https://cran.r-project.org/. The page automatically detects your operating system and provides download links. Click on Download R for your platform:

- Windows users: click “Download R for Windows” then “base” and download the installer for the latest version. Run the downloaded .exe file and follow the installation prompts. Accept the defaults unless you have specific requirements.
- macOS users: click “Download R for macOS” and download the .pkg installer for your system. Run it and follow the prompts. Note that recent versions of R require macOS 10.15 Catalina or later.
- Linux users: follow the instructions for your distribution. For Debian-based systems like Ubuntu, add the CRAN repository to your sources list and install via apt. For RHEL-based systems, use yum or dnf with the appropriate repository.

After R is installed, download and install RStudio Desktop from https://posit.co/download/rstudio-desktop/. Choose the free Open Source version unless your organization has a Posit Workbench license. The installer bundles RStudio with support for current R versions. When you launch RStudio, it will use whichever R installation it finds on your system.

Verify that everything is working by opening RStudio and checking the version information. In the Console pane, type:

```
R.version.string
```

You should see output like "R version 4.6.1 (2026-06-24)". If you see a different version but it is 4.4 or higher, you are in good shape. Versions older than 4.3 may have compatibility issues with newer packages and should be upgraded.

The RStudio Interface

RStudio organizes its window into four main panes arranged in a grid. Understanding this layout will save you time and confusion as you work.

The Console pane, typically in the bottom-left by default, is where R executes your code and displays results. When you type R commands here and press Enter, they run immediately. This interactive mode is excellent for exploration, testing small snippets and learning the language. The console shows a prompt, usually >, indicating it is ready for input. If you see a + prompt instead, R is waiting for you to complete an expression, such as closing a parenthesis or quote.

The Source Editor pane in the top-left is where you write and save scripts. Scripts are files with the .R extension containing sequences of R commands. Unlike the console, which forgets everything when you close RStudio, scripts persist and can be re-run, shared and version-controlled. Professional R work happens primarily in scripts rather than the console.

The top-right pane contains tabs for Environment, History, Connections and more. The Environment tab shows all objects currently loaded in your R session: data frames, vectors, functions and their sizes. This is invaluable for tracking what you have created and how much memory you are using.

The History tab records commands you have run, which can help you recall previous work.

The bottom-right pane has tabs for Files, Plots, Packages, Help, Viewer and more. The Files tab provides a file browser for navigating your project directory. The Plots tab displays any graphs you have created and lets you zoom, export or clear them. The Packages tab shows which packages are installed and loaded, with checkboxes to load or unload them. The Help tab displays documentation when you look up functions.

You can rearrange these panes by dragging their edges or changing the layout through Tools, Global Options, Pane Layout. The default arrangement works well for most users, so resist the urge to customize until you have a clear reason.

Your First Program

Let us write some R code. Start by creating a new script. In RStudio, click File, New File, R Script, or use the keyboard shortcut Ctrl+Shift+N on Windows and Linux or Cmd+Shift+N on macOS. A blank editor window opens with the name “Untitled1”.

Type the following line:

```
print("Hello, R!")
```

Then press Ctrl+Enter or Cmd+Enter to run that line. You should see “Hello, R!” appear in the Console pane. Congratulations, you have written and executed your first R program.

The print() function takes an argument, in this case the text string “Hello, R!”, and displays it. Text in R is always enclosed in quotes, either double quotes or single quotes. The parentheses surround the arguments passed to the function. This pattern of function_name(arguments) is how you call almost every function in R.

Let us try something slightly more interesting. Add these lines below your first:

```
x <- 10
y <- 25
sum_result <- x + y
print(sum_result)
```

Run all three lines by selecting them and pressing Ctrl+Enter. The console shows 35. Here is what happened step by step:

The first line creates a variable named x and assigns it the value 10. The symbol <- is R's assignment operator. It takes the value on the right and stores it in the name on the left. You can read this as "x gets 10" or "10 goes into x".

The second line similarly creates y with value 25.

The third line computes x + y, which is 10 + 25 = 35, and stores that result in a new variable called sum_result.

The fourth line prints the value of sum_result.

Check the Environment pane. You should see x, y and sum_result listed with their values. These objects exist in your current R session until you remove them or restart R.

A note on assignment: R also accepts = as an assignment operator, so `x = 10` works the same as `x <- 10`. However, <- is the traditional R style and is preferred throughout this book and in most R codebases. The reason is that = has a second role in R: it can specify named arguments inside function calls, which can create ambiguity. Using <- exclusively for assignment keeps your intent clear.

Using R as a Calculator

Before moving to scripts, let us use the console interactively. R works perfectly as a calculator and this is a good way to get comfortable with the interface. Type each of these directly into the Console and press Enter:

```
2 + 3
10 * 4
100 / 7
2 ^ 8
17 %% 5
```

You should see results of 5, 40, approximately 14.29, 256, and 2 respectively. These are the basic arithmetic operators:

- `+` for addition
- `-` for subtraction
- `*` for multiplication
- `/` for division
- `^` for exponentiation
- `%%` for modulo, which gives the remainder after division

R also includes many mathematical functions built in. Try these:

```
sqrt(144)
log(100)
log10(1000)
exp(2)
abs(-7.3)
sin(pi / 2)
round(3.14159, digits = 2)
```

The results are 12, approximately 4.61 (natural log), 3, approximately 7.39, 7.3, 1, and 3.14. Notice how round() uses a named argument `digits = 2` to specify precision. Named arguments make code more readable because you do not need to remember the order of parameters.

R has several predefined constants available:

```
pi
exp(1)
sqrt(2)
```

These give you the mathematical constants pi, e and the square root of 2 with high precision. You can use these directly in calculations without defining them yourself.

Interactive exploration like this is one of R's greatest strengths for learning and data analysis. When you encounter a new function or want to test an idea, type it into the console and see what happens. R will either give you a result or an error message that often explains what went wrong.

Saving and Running Scripts

Interactive work in the console is great for exploration but terrible for reproducibility. When you close RStudio, everything in the console disappears unless you explicitly save it. Scripts solve this problem by recording your code in files that persist across sessions.

Save your current script by clicking File, Save As and giving it a descriptive name like `getting_started.R`. Choose a folder where you will keep your R projects. Good naming habits matter: use lowercase letters, underscores between words and the .R extension. Avoid spaces and special characters in file names.

Once saved, you can run individual lines with Ctrl+Enter or select multiple lines and run them together. You can also run the entire script by clicking the Source button in the editor toolbar or pressing Ctrl+Shift+Enter.

Let us create a more complete example script. Replace your current script contents with:

```
# Getting Started with R
# A simple demonstration script

# Define some values
population <- 8000000
growth_rate <- 0.015
years <- 10

# Calculate future population using compound growth formula
future_population <- population * (1 + growth_rate) ^ years

# Display results
cat("Current population:", population, "\n")
cat("Growth rate:", growth_rate * 100, "%\n")
cat("Years:", years, "\n")
cat("Future population:", round(future_population), "\n")
```

Run the entire script. The console output shows:

```
Current population: 8000000
Growth rate: 1.5 %
Years: 10
Future population: 9268341
```

This script demonstrates several important concepts. Lines starting with # are comments. R ignores everything after the # symbol on a line. Comments are essential for documenting what your code does, especially as scripts grow longer and more complex. Write comments as if you are explaining the code to another person, or to yourself six months from now when you have forgotten the details.

The cat() function is similar to print() but gives you more control over formatting. The `\n` sequences are newline characters that move the output to the next line. Without them, everything would appear on one long line.

Notice how the script flows logically: it defines inputs, performs a calculation, then displays results. This structure of input-process-output is a pattern you will see throughout data analysis work.

Projects and Working Directories

As your R work grows, you will accumulate scripts, data files, plots and other materials. Keeping these organized in projects makes everything easier to manage.

In RStudio, create a new project by clicking File, New Project. You can start with an empty project in a new directory. Give it a name like `r-learning` and choose a location on your computer. RStudio will create the folder and open it as your active project.

When you work within a project, RStudio sets that folder as your working directory. This is the default location where R looks for files when you read data or save outputs. You can check your current working directory with:

```
getwd()
```

And change it with:

```
setwd("/path/to/your/directory")
```

However, when using RStudio projects, you rarely need to set the working directory manually. The project system handles this automatically, and your scripts become portable because they reference files relative to the project root rather than absolute paths on your specific computer.

Good project organization habits include:

- Keeping related scripts, data, and outputs in one project folder
- Using subdirectories for different types of files: `data/` for raw data, `scripts/` for analysis code, `output/` for results and plots
- Committing projects to version control like Git so you can track changes and collaborate

These practices become increasingly important as your R skills advance and your projects grow in complexity.

Getting Help

Learning any programming language involves encountering problems and finding solutions. R has excellent built-in help systems that you should learn to use early.

To get documentation for a specific function, use the `?` operator:

```
?mean
?lm
?read.csv
```

This opens the Help pane showing the function's description, arguments, examples and related functions. The Examples section is particularly valuable because it shows real usage patterns you can adapt.

If you are not sure of the exact function name, use `??` to search:

```
??regression
??plot
```

This searches all loaded packages for topics matching your keyword.

You can also use the help() function:

```
help(mean)
help.start()
```

The help.start() command opens an HTML-based help browser that lets you browse documentation by package.

When you encounter an error, read the message carefully. R error messages have improved significantly in recent versions and often tell you exactly what went wrong. Common patterns include:

- "object 'x' not found": you referenced a variable that does not exist, possibly due to a typo
- "non-numeric argument to binary operator": you tried to do arithmetic on non-numeric data
- "subscript out of bounds": you tried to access an element beyond the length of a vector or list

If built-in help does not resolve your issue, the R community is large and helpful. Stack Overflow has hundreds of thousands of R questions answered by experts. The Posit Community forum at https://community.posit.co/ is another excellent resource. When asking for help, provide a minimal reproducible example: a small piece of code that demonstrates the problem without requiring access to your full dataset or environment.

Summary

In this chapter you installed R and RStudio, learned the interface layout, wrote your first programs, and explored R's interactive capabilities. You now have a working development environment and understand the basic workflow of writing scripts, running code, and getting help. These foundations will support everything that follows.

The next chapter dives into the core building blocks of R programming: variables, data types, operators and how R stores and manipulates data. Understanding these fundamentals is essential because every R program, no matter how complex, ultimately works with data through these basic mechanisms.

Chapter 2: Variables, Data Types, and Basic Operations

Every programming language needs ways to store data, distinguish between different kinds of data, and perform operations on that data. R is no exception. This chapter covers the foundational concepts of variables, data types, and operators that you will use in virtually every R program you write. Master these basics and the rest of the language becomes much easier to understand.

Variables and Assignment

A variable is a named container that holds a value. In R, you create variables using assignment, which stores a value under a name so you can refer to it later. The standard assignment operator in R is `<-`, read as "gets" or "is assigned".

```
age <- 30
name <- "Alice"
height <- 1.65
is_student <- FALSE
```

After these lines run, the variable `age` holds the number 30, `name` holds the text "Alice", and so on. You can use these variables anywhere in your code:

```
next_year_age <- age + 1
cat(name, "will be", next_year_age, "next year.\n")
```

This prints: "Alice will be 31 next year."

Variable names in R follow specific rules. They must start with a letter or a dot followed by a letter. They can contain letters, numbers, dots and underscores. Names are case-sensitive, so `Age` and `age` are different variables. Avoid using dots at the beginning of names unless you understand the convention, because R uses leading dots for certain internal functions and some packages reserve that pattern.

The tidyverse style guide, which is widely followed in modern R code, recommends using snake_case for variable and function names: lowercase letters with underscores between words. So `customer_age` is preferred over `CustomerAge`, `customer.age`, or `CUSTOMER_AGE`. This convention improves readability and consistency across projects.

Some names are reserved and cannot be used as variables because they are keywords in the language: `if`, `else`, `for`, `while`, `function`, `repeat`, `next`, `break`, `TRUE`, `FALSE`, `NULL`, `NA`, `Inf`, and `NaN`. Trying to assign to these produces an error.

You can also use = for assignment, particularly at the top level:

```
x = 5
y <- 5
identical(x, y)
```

Both create variables with the same value, and `identical()` confirms they are exactly the same. However, `<-` is preferred throughout this book and in most R codebases. The reason is that = has a secondary role in R as a named argument operator inside function calls, which can create ambiguity. Using `<-` exclusively for assignment makes your intent unambiguous.

To remove a variable from your environment, use rm():

```
rm(age, name, height)
```

This frees the memory and removes the names. To clear all objects, you can run `rm(list = ls())`, but be careful: this deletes everything in your current session.

Core Data Types

R is a dynamically typed language, meaning you do not declare the type of a variable when you create it. R infers the type from the value you assign. Understanding R's core data types is crucial because many operations behave differently depending on the types involved.

R has six basic atomic data types:

1. **Logical**: represents truth values, TRUE or FALSE
2. **Integer**: whole numbers without fractional parts
3. **Double**: numbers with decimal points, the default numeric type
4. **Complex**: numbers with real and imaginary parts
5. **Character**: text strings
6. **Raw**: binary data, rarely used directly

Let us examine each type:

Logical values represent boolean truth:

```
is_raining <- TRUE
has_umbrella <- FALSE
temperature_above_freezing <- 25 > 0
```

The last line evaluates the comparison `25 > 0`, which is true, so the result is TRUE. Logical values are fundamental to conditional logic and filtering data.

Integers are whole numbers. When you type a number without a decimal point, R treats it as a double by default, not an integer:

```
x <- 5
class(x)
y <- 5L
class(y)
```

The `L` suffix forces R to store the value as an integer. You will rarely need to explicitly use integers in everyday R programming because doubles handle most numeric work perfectly well. Integers matter mainly in specialized contexts like indexing large datasets where memory efficiency counts.

Doubles are the standard numeric type in R, representing real numbers with decimal precision:

```
pi_approx <- 3.14159
temperature <- -5.2
scientific <- 6.022e23
```

R uses double-precision floating point representation, which gives about 15 to 17 significant digits of precision. This is sufficient for virtually all statistical and data analysis work, but you should be aware that floating point arithmetic can produce surprising results due to rounding:

```
0.1 + 0.2 == 0.3
```

This returns FALSE, not TRUE, because 0.1 and 0.2 cannot be represented exactly in binary floating point. The sum is very close to 0.3 but not exactly equal. For most practical purposes this does not matter, but it can cause issues with exact equality comparisons on computed values. Use approximate comparison functions like `all.equal()` when precision matters:

```
all.equal(0.1 + 0.2, 0.3)
```

Complex numbers support mathematical operations involving imaginary components:

```
z <- 3 + 4i
Re(z)
Im(z)
abs(z)
```

These give the real part (3), imaginary part (4), and magnitude (5). Complex numbers are used in signal processing, electrical engineering, and certain statistical methods, but most R users encounter them rarely.

Character strings represent text:

```
greeting <- "Hello, World!"
name <- 'Alice'
empty_string <- ""
```

R accepts both single and double quotes for strings, and they are equivalent. If your string contains the quote character you are using, escape it with a backslash or use the other quote type:

```
text1 <- "She said \"Hello\""
text2 <- 'She said "Hello"'
```

Both produce the same result.

Raw vectors hold binary data at the byte level and are primarily used internally for reading and writing files in binary format. You will rarely create raw vectors directly.

Special Values

Beyond the basic types, R defines several special values that appear frequently in data analysis:

```
NA
NaN
Inf
NULL
```

NA stands for "not available" and represents missing data. This is one of the most important concepts in R because real-world data almost always contains missing values. NA can appear in any type:

```
x <- c(1, 2, NA, 4)
y <- c("apple", NA, "cherry")
```

Most operations involving NA return NA as well, because the result is unknown when one input is missing:

```
10 + NA
mean(c(1, 2, NA, 4))
```

Both return NA. Many functions provide a way to handle missing values, typically through an `na.rm` argument:

```
mean(c(1, 2, NA, 4), na.rm = TRUE)
```

This returns 2.33, the mean of the non-missing values.

NaN stands for "not a number" and represents undefined mathematical results:

```
0 / 0
sqrt(-1)
```

NaN is a special case of NA. All NaN values are also NA, but not all NA values are NaN. Use `is.nan()` to test specifically for NaN and `is.na()` to test for any missing value:

```
is.na(NA)
is.na(NaN)
is.nan(NA)
is.nan(NaN)
```

Inf represents positive infinity, and `-Inf` represents negative infinity:

```
1 / 0
-1 / 0
Inf + 100
Inf * 2
Inf - Inf
```

The last expression returns NaN because infinity minus infinity is mathematically undefined.

NULL represents the absence of a value or object entirely, different from NA which represents a missing value within an existing object:

```
x <- NULL
length(x)
y <- c(1, 2, NULL, 4)
y
```

NULL has length zero and disappears when combined with other values in vectors. This makes it useful for representing optional function arguments or empty results.

Type Conversion and Coercion

Sometimes you need to convert a value from one type to another. R provides explicit conversion functions:

```
as.logical(1)
as.logical(0)
as.logical(NA)

as.integer(3.7)
as.integer("5")
as.integer(TRUE)

as.numeric(5L)
as.numeric("3.14")
as.numeric(TRUE)

as.character(42)
as.character(3.14)
as.character(TRUE)
as.character(as.Date("2024-01-15"))
```

These functions attempt conversion and return NA with a warning if the conversion does not make sense:

```
as.numeric("hello")
as.integer("three")
```

Both return NA because the strings cannot be interpreted as numbers.

R also performs implicit coercion automatically when you mix types in certain operations. This is called type coercion and follows a hierarchy: logical < integer < double < complex < character. When R encounters mixed types, it coerces everything to the "highest" type needed:

```
c(TRUE, FALSE, TRUE)
c(1L, 2L, 3L)
c(1, 2.5, 3)
c(1L, 2.5)
c(1, "hello")
c(TRUE, 5)
```

Notice how mixing a number with text converts the number to text, and mixing logical with numeric converts the logical to numeric (TRUE becomes 1, FALSE becomes 0). Understanding coercion helps you avoid subtle bugs where data silently changes type in unexpected ways.

Vectors: The Fundamental Data Structure

In R, the most basic data structure is the vector. A vector is an ordered sequence of elements that are all of the same type. Even a single value in R is technically a vector of length one:

```
x <- 5
length(x)
is.vector(x)
```

You create vectors using the c() function, which stands for "combine" or "concatenate":

```
numbers <- c(1, 2, 3, 4, 5)
names <- c("Alice", "Bob", "Charlie")
bools <- c(TRUE, TRUE, FALSE, TRUE)
mixed <- c(1, 2, "three", 4)
class(mixed)
```

The last vector shows coercion in action: mixing numbers and text converts everything to character.

Vectors are everywhere in R. Data frame columns are vectors. Function arguments often expect vectors. Understanding how to work with vectors is essential because R is designed around vectorized operations: functions that operate on entire vectors at once rather than requiring explicit loops.

You can access individual elements of a vector using square brackets with a numeric index. R uses 1-based indexing, meaning the first element is at position 1, not 0:

```
fruits <- c("apple", "banana", "cherry", "date", "elderberry")
fruits[1]
fruits[3]
fruits[5]
```

You can also select multiple elements by providing a vector of indices:

```
fruits[c(1, 3, 5)]
fruits[c(2, 4)]
```

Negative indices exclude those positions:

```
fruits[-1]
fruits[-c(1, 5)]
```

This returns all elements except the first, and all except the first and last.

You can use logical vectors to subset:

```
values <- c(10, 25, 3, 42, 17)
values > 20
values[values > 20]
```

The expression `values > 20` creates a logical vector indicating which elements are greater than 20, and using that as an index selects only the TRUE positions. This pattern, called logical subsetting, is one of R's most powerful and frequently used features.

If you try to access an index beyond the vector length, R returns NA:

```
fruits[10]
```

This behavior can be both helpful and dangerous. It helps because your code does not crash, but it can introduce silent errors if you are not careful. Always check vector lengths when indexing dynamically.

Working with Factors

Factors are R's way of representing categorical data: variables that take on a limited set of distinct values, like gender, education level, or product category. Factors look similar to character vectors but store the data more efficiently and carry information about the possible categories, called levels.

```
colors <- factor(c("red", "blue", "green", "red", "blue", "red"))
colors
levels(colors)
```

The output shows each value and indicates that there are three levels: blue, green and red. Notice that R sorts the levels alphabetically by default. You can specify the order explicitly:

```
sizes <- factor(c("M", "S", "L", "S", "M", "XL"),
                levels = c("S", "M", "L", "XL"))
sizes
levels(sizes)
```

Now the levels follow the logical size ordering rather than alphabetical order. This matters for plotting and modeling because R uses the level order to determine how categories are arranged and compared.

Factors are particularly important in statistical modeling. When you include a factor as a predictor in a regression model, R automatically creates dummy variables for each level, using the first level as the reference category. Understanding factors helps you control how your models interpret categorical variables.

The forcats package from tidyverse provides many useful functions for working with factors, including fct_relevel() to change level order, fct_infreq() to reorder by frequency, and fct_collapse() to combine levels. These tools make factor manipulation much easier than base R approaches.

One common pitfall with factors is that adding a new value not in the existing levels produces NA:

```
colors <- factor(c("red", "blue", "green"))
colors[4] <- "yellow"
colors
```

The fourth element becomes NA because “yellow” is not a recognized level. To add new levels, you must update the factor explicitly:

```
levels(colors) <- c(levels(colors), "yellow")
colors[4] <- "yellow"
```

Operators and Precedence

R provides a comprehensive set of operators for arithmetic, comparison, logical operations, and more. Understanding operator precedence is important because it determines the order in which expressions are evaluated.

Arithmetic operators:

```
10 + 3
10 - 3
10 * 3
10 / 3
10 ^ 3
10 %% 3
10 %/% 3
```

These give addition, subtraction, multiplication, division, exponentiation, modulo (remainder), and integer division. The last two are less commonly used but handy: `%%` gives the remainder and `%/%` gives the quotient without the fractional part.

Comparison operators return logical values:

```
5 == 5
5 != 3
10 > 7
10 >= 10
3 < 5
3 <= 2
```

Note that == tests for equality while <- or = performs assignment. Confusing these is a common beginner error. Also remember the floating point caveat: avoid using == to compare computed numeric values where rounding might be involved.

Logical operators combine or negate logical values:

```
TRUE & TRUE
TRUE & FALSE
TRUE | FALSE
!TRUE
TRUE && TRUE
TRUE || FALSE
```

There are two versions of AND and OR. The single-character versions `&` and `|` are vectorized, meaning they operate element-wise on vectors:

```
c(TRUE, FALSE, TRUE) & c(TRUE, TRUE, FALSE)
c(1, 2, 3) > 1 & c(1, 2, 3) < 3
```

The double-character versions `&&` and `||` only look at the first element of each vector and short-circuit: they stop evaluating as soon as the result is determined. Use `&&` and `||` in if statements where you are testing single conditions, and `&` and `|` when working with vectors.

Operator precedence determines evaluation order. Higher precedence operators are evaluated first:

1. `^` (exponentiation)
2. Unary minus: `-x`
3. `:`, the sequence operator
4. `%any%`, special operators like `%%`
5. `*`, `/`
6. `+`, `-`
7. `<`, `<=`, `>`, `>=`
8. `==`, `!=`
9. `!`
10. `&`, `&&`
11. `|`, `||`
12. `->`, `<-`

When in doubt, use parentheses to make your intent explicit:

```
(2 + 3) * 4
2 + (3 * 4)
```

These give different results: 20 and 14 respectively. Parentheses improve both correctness and readability.

Summary

This chapter covered the building blocks of R programming. You learned how to create and name variables, the six atomic data types and their characteristics, special values like NA and NULL, type conversion and coercion rules, vectors as R's fundamental data structure, factors for categorical data, and the full range of operators with their precedence.

These concepts form the foundation for everything else in R. Every data structure you will encounter is built from these basic types. Every computation uses these operators. Every function takes arguments of these types. As you move forward, refer back to this chapter whenever you encounter type-related confusion or unexpected behavior. The next chapter builds on these basics by introducing control flow and functions, which let you make decisions in your code and organize it into reusable units.

Chapter 3: Control Flow and Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Conditional Logic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Vectorized Conditionals with ifelse

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Loop Alternatives with the apply Family

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Writing Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Function Arguments in Depth

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Scope and Environments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 4: Data Structures Deep Dive

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Lists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Matrices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Arrays

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Data Frames

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Tibbles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Choosing the Right Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 5: Strings, Dates, Times, and Regular Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Character Strings in R

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

String Manipulation Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

The stringr Package

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Dates and Times in Base R

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

The lubridate Package

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Regular Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 6: Data Import, Export, and File I/O

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Working with Directories and Files

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Reading Plain Text with Base R

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Modern Import with readr

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Reading Excel Files

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Other Formats: JSON, XML, and More

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

The Arrow Package for Large Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Best Practices for Data Loading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 7: The Tidyverse Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

What Is the Tidyverse

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Pipes and Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

dplyr for Data Manipulation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Joining and Reshaping with tidyr

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Functional Programming with purrr

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Combining Tools in Real Workflows

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 8: Data Visualization with ggplot2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Principles of Effective Visualization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

The Grammar of Graphics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Building Plots Step by Step

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Geometries and Statistical Transformations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Customizing Appearance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Advanced Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 9: Statistical Analysis with R

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Descriptive Statistics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Probability Distributions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Hypothesis Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Linear Regression

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Model Diagnostics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Reporting Statistical Results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 10: Machine Learning Foundations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Supervised vs Unsupervised Learning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Model Training Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Classification Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Regression Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Clustering and Dimensionality Reduction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Evaluation and Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 11: Reproducible Research with Quarto

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

What Is Reproducible Research

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Introduction to Quarto

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Writing Your First Quarto Document

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Code Chunk Options

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Creating Reports and Presentations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Parameterized Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Version Control Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 12: Error Handling, Debugging, and Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Understanding Errors and Warnings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Error Handling Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Debugging Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Profiling Code Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Optimization Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Memory Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 13: Object-Oriented Programming in R

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Why OOP in R

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

S3 Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

S4 Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

R6 Reference Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Choosing Between Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 14: Building and Sharing R Packages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Why Write a Package

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Setting Up a Package

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Writing Functions for Packages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Documentation with roxygen2

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Testing with testthat

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Building and Installing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Checking for Problems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Publishing Beyond CRAN

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 15: Web Applications, APIs, and Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Shiny Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Working with APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Database Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Parallel and High-Performance Computing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Deployment Options

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Chapter 16: Best Practices and Advanced Topics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Code Style and Standards

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Common Pitfalls and How to Avoid Them

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Functional Programming Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Metaprogramming with rlang

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Working with Large Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

The Future of R

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Building Your R Career

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

Conclusion: Your Journey Forward

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/rprogrammingcookbook.