

Not So Short Introduction to R

A Practical Learning Journey from First Contact to Reproducible Data Systems

Daniel James

June 7, 2026

Table of contents

Not So Short Introduction to R	3
Dedication	3
About This Book	3
Who This Book Is For	3
What This Book Is Trying to Solve	4
How to Read This Book	4
What You Will Need	5
What You Will Learn to Do	5
What Makes This Book Different	5
Book Roadmap	6
Optional Setup Check	6
Introduction: The Day R Became Less Strange	7
Why This Book Exists	10
What This Book Will Help You Do	11
Part 1: Foundations	13
—	14
Before the First Command: Why R Changes the Way Data Work Thinks	15
Opening Scene: Leo Before the Prompt	15
The Problem Was Never Leo	16
What R Is	16
Why R Exists	17
Why R Feels Different from Excel	17
The Shift from Manual Work to Reproducible Work	18
The First Lightbulb Moment	19
What Leo Is Expected to Know at This Point	19
Chapter Outcome	20
Transition to Chapter 2	20
Understanding the R Ecosystem	21
The Engine, the Cockpit, and the Flight Plan	21
Opening Scene: The Engine Is Not the Whole Journey	21
What Changed After Chapter 1	21
The First Map of the R Ecosystem	22
R as the Engine	22
Before R: Understanding Your Computer Workspace	22
Installing R First	23
Leo Meets the R Console	23
The First Commands	24
What These Commands Teach	25
Where the Console Starts to Feel Limited	25
The Question That Changes the Workflow	26
RStudio as the Working Environment	26
Integrated Development Environments	26
Installing RStudio	27

First Look at the RStudio Workspace	27
Running the Same Commands in a Better Workflow	27
Save the Script	28
Packages as Specialist Extensions	29
A Note About the Dataset Waiting Ahead	29
A Tiny Glimpse of Analysis	30
Momentum Win	30
Chapter Outcome	30
Transition to Chapter 3	31
Data Before Types: Atoms, Containers, and the First Shape of R Thinking	32
Opening Scene: The Question Beneath the Code	32
What Changed After Chapter 2	32
The Material Beneath Every Command	33
Data Comes in Pieces	33
Values Need Somewhere to Live	34
Names Help R Find Values	35
From Single Values to Containers	35
A Small Visual Map	36
Special Values Are Still Signals	37
Predict Before You Run	37
Missing, Impossible, Infinite, and Empty	38
Why Inspection Comes Before Trust	39
Why This Matters Before Data Types	40
What Leo Is Expected to Know at This Point	40
Chapter Outcome	41
Transition to Chapter 4	41
Why Values Behave Differently: Data Types and the Hidden Identity of Data	42
Opening Scene: The Value That Looks Right but Fails	42
What Changed After Chapter 3	43
The First Confusion	43
The Hidden Difference	44
The Core Idea: Type Is Behaviour	45
A Small Example from Everyday Life	45
Seeing Types in Action	46
Predict Before You Run	46
Three Behaviours That Reveal Type	47
Behaviour 1: Calculation	47
Behaviour 2: Combination	47
Behaviour 3: Conflict	47
Naming the Types	48
Numeric Values	49
Integer Values	49
Character Values	50
Logical Values	51
Dates as Time-Aware Values	52
Factors as Category-Aware Values	53
Short Recap: What the Types Are Doing	53
Why Type Matters in Real Work	54
The Silent Failure Problem	54
Inspect Before You Trust	55

The First Rule of Reliable Analysis	56
The Mental Shift	57
Leo's Notebook	57
What Comes Next	57
Chapter Outcome	58
Key Terms	58
Practice Lab	59
Task 1: Compare what looks similar	59
Task 2: Inspect truth values	59
Task 3: Inspect dates and categories	60
Task 4: Write the lesson in plain language	60
How Data Holds Together: Structures, Containers, and the Shape of R Work	61
Opening Scene: When One Value Becomes Many	61
What Changed After Chapter 4	61
Why Structure Matters	62
Containers Become Structures	63
A Small Visual Map of Structures	63
The Two Behaviours Every Beginner Should Notice	64
Homogeneous Structures: When Sameness Matters	64
The First Homogeneous Structure: Vector	65
Vectors can hold different kinds of uniform data	65
When R Forces Values to Fit	67
The Rectangular Homogeneous Structure: Matrix	68
Why Homogeneous Structures Exist	69
Heterogeneous Structures: When Difference Matters	70
The First Heterogeneous Structure: List	70
Why Heterogeneous Structures Exist	72
Data Frames: Where Real Analysis Begins	72
Inspecting Structure	74
Seeing the Pattern Clearly	75
Structure Determines What You Can Do	76
Accessing Structured Data	76
Connecting to <code>df_r_migration</code>	77
A Short Recap	77
Leo's Notebook	78
Practice Lab	78
Task 1: Create vectors of different types	78
Task 2: Create a date vector and a factor vector	79
Task 3: Compare a vector and a matrix	79
Task 4: Create a learner list	79
Task 5: Create a small data frame	80
Key Terms	80
What Comes Next	81
Chapter Outcome	81
Part 2: Language of R	82
Syntax as Structure of Meaning: How R Reads Code	83
Opening Scene: Code Is Not a Pile of Symbols	83
What Changed After Chapter 5	84

R as a Language	85
What Is an Expression?	86
Expression and Assignment Are Not the Same Thing	87
Every Expression Returns Something	88
Assignment: Giving a Result a Name	89
A Simple Line of R	90
Operators Need Complete Expressions	91
Function Calls as Structured Expressions	91
Parentheses Are Not Decoration	92
Evaluation: How R Works Through Code	93
The Inside-Out Rule	94
Why Nested Code Confuses Beginners	95
Common Beginner Mistake: Reading Code as Magic	96
Try It Yourself	97
Short Recap	98
Leo's Notebook	98
Glossary	99
Chapter Outcome	99
Transition to Chapter 7	99
Logic in R: How Code Makes Judgments	101
Opening Scene: When Code Has to Decide	101
What Changed After Chapter 6	102
Logical Values: TRUE and FALSE	102
From Calculation to Judgment	103
Comparison Operators: Asking Questions	104
Predict Before You Run	105
Assignment Is Not Comparison	106
Logical Operators: Combining Conditions	106
& Versus &&, and Versus 	107
if: A Single Decision, Not a Filter	108
Vectorised Logic: One Rule, Many Values	110
Predict Before You Run: One Answer or Many?	110
Using Logical Results to Select Values	110
Logic with Missing Values	111
A Small Repair Win: From Wrong Test to Reliable Check	112
Logic in df_r_migration	112
Other Operator Families Leo Will Meet Later	113
How R Thinks: Logic Errors Are Different from Syntax Errors	113
Common Beginner Mistakes	114
What Leo Is Expected to Know at This Point	114
Chapter Outcome	115
Leo's Notebook	115
Transition to Chapter 8	115
Built-in Functions: R's Ready-Made Action System	117
Opening Scene: When R Already Knows the Action	117
What Changed After Chapter 7	117
From Judgment to Action	118
The Shape of a Function Call	119
How R Thinks: Inputs, Action, Output	120
Arguments: Instructions Inside the Action	120

Predict Before You Run	121
Inspect the Output Before You Trust It	121
A Beginner Core of Built-in Functions	122
Combining and Measuring Values	122
Summarising Values	123
Working with Missing Values	124
Checking and Converting Values	124
Working with Text	125
Working with Dates	126
Simple Formatting and Sampling	127
A Small Momentum Win	128
A Practical Map of Built-in Function Families	129
Reference Preview: Functions for Later	130
Functions, Operators, and Language Constructs Are Not the Same	130
Built-in Functions Are Not the Whole R Ecosystem	131
How to Ask for Help	131
Common Beginner Mistakes	132
What Leo Is Expected to Know at This Point	133
A Small Practice Lab	133
Chapter Outcome	134
Transition to Chapter 9	134
User-Defined Functions: Naming Actions of Your Own	135
Opening Scene: When Repetition Becomes a Signal	135
What Changed After Chapter 8	136
From Ready-Made Actions to Named Thinking	136
Functions as Named Thinking	137
Keep Beginner Functions Small	138
The Shape of a User-Defined Function	139
How R Thinks: Running a Function Leo Created	139
Predict Before You Run	140
Turning Repetition into a Function	141
A Small Momentum Win	141
Inputs, Body, and Output	142
Predict Before You Run: Text Cleaning	143
Testing a Function Before Trusting It	143
Simple Input Checks	144
The New Responsibility: When Leo Names an Action	145
When a Function Is Trying to Do Too Much	145
A Small Repair Moment	146
Default Arguments	147
Naming Functions Clearly	147
Common Beginner Mistakes	148
Mistake 1: Defining a Function but Forgetting to Call It	148
Mistake 2: Using an Outside Object by Accident	148
Mistake 3: Making the Function Too Big	149
Mistake 4: Forgetting to Inspect the Output	149
What Leo Is Expected to Know at This Point	149
A Small Practice Lab	150
Task 1: Predict the Output	150
Task 2: Name the Repeated Rule	150

Task 3: Test Before Trust	150
Task 4: Keep It Small	150
Chapter Outcome	151
From Naming Actions to Checking Behaviour	151
What Comes Next	152
Debugging R: A Practical Framework for Reading and Fixing Errors	153
Opening Scene: When Working Code Stops Working	153
What Changed After Chapter 9	154
The Lightbulb: Errors Are Signals	154
Errors, Warnings, and Messages Are Not the Same	155
The Debugging Framework	156
Step 1: Read the Message Slowly	157
Step 2: Locate the Failing Line	157
Step 3: Debug Missing Objects	157
Step 4: Check Spelling and Case	159
Step 5: Debug Assignment Problems	159
Step 6: Debug Syntax Errors	160
Step 7: Debug Function Calls and Arguments	161
Step 8: Debug Data Type Problems	161
Step 9: Debug Structure Problems	162
Step 10: Debug Data Frame Column Names	163
Step 11: Debug Package and Namespace Problems	164
Step 12: Debug Pipes	164
Step 13: Debug User-Defined Functions	165
Step 14: Run the Smallest Failing Code	166
Step 15: Fix One Thing at a Time	167
Common R Errors and First Questions	167
A Closer Look: <code>object of type 'closure' is not subsettable</code>	168
When to Use <code>traceback()</code>	169
Debugging Checklist	169
Cheat Sheet: Common Errors and First Moves	169
Try It Yourself	170
A Small Practice Lab	171
Short Recap	172
Leo's Notebook	173
Glossary	173
Chapter Outcome	174
Closing Part 2: Leo Can Repair the Conversation	174
What Comes Next	175
Part 3: Data Workflows: From Isolated Commands to Reproducible Analysis	176
Packages and the R Ecosystem in Practice: From Tools to Reproducible Workflows	177
Opening Scene: When R Becomes Bigger Than Base R	177
Part 3 Begins: From Commands to Workflows	178
Why Packages Exist	179
Packages as Specialist Toolboxes	180
Base R, Recommended Packages, and External Packages	181
Installing a Package	182
Loading a Package	182

Installing Is Not Loading	183
A Responsible Package Section	184
The Tidyverse as a Package Family	185
Namespaces: Where Functions Live	185
Function Conflicts	186
Using <code>package::function()</code> Without Loading Everything	187
Package Documentation	187
Package Versions	188
Updating and Removing Packages	189
A Package Workflow for This Book	189
Common Beginner Mistakes	191
Mistake 1: Installing a package every time a script runs	191
Mistake 2: Loading a package before it is installed	191
Mistake 3: Forgetting quotation marks in <code>install.packages()</code>	191
Mistake 4: Putting quotation marks inside <code>library()</code> unnecessarily	192
Mistake 5: Ignoring conflict messages	192
Mistake 6: Loading many packages without knowing why	192
Try It Yourself	192
A Small Practice Lab	193
Short Recap	194
Leo's Notebook	194
Glossary	195
Chapter Outcome	195
What Comes Next	196
Importing Data: From Files on Disk to Objects R Can Work With	197
Opening Scene: The File Leo Can See but R Cannot Yet Use	197
What Changed After Chapter 11	198
The Part 3 Workflow So Far	198
The Official Dataset for This Book	198
File, Path, and Object Are Not the Same Thing	200
A Small Visual Map of Importing	201
Project-Relative Paths	201
Importing the CSV File	202
A Base R Import Option	202
Seeing What Came In	203
Inspecting the Official Dataset	203
Import Messages Are Part of the Evidence	204
Protecting the Raw Data	204
Common Beginner Mistakes	205
Mistake 1: Confusing the file name with the object name	205
Mistake 2: Forgetting quotation marks around the path	205
Mistake 3: Using an absolute path	205
Mistake 4: Importing from the Downloads folder	206
Mistake 5: Assuming import means clean	206
Mistake 6: Overwriting the raw file	206
What Leo Is Expected to Know at This Point	206
Chapter Outcome	206
Transition to Chapter 13	207
Cleaning and Preparing Data: Making Evidence Trustworthy	208
Opening Scene: When Import Reveals the Work	208

What Changed After Chapter 12	208
The Part 3 Workflow So Far	209
Raw Evidence and Working Copy	210
A Small Visual Map of Cleaning	210
Before Cleaning: Inspect the Mess	211
Cleaning Is Not Transformation	212
Start with Column Names and Structure	212
Missing Values and Missing-Like Values	213
Text Fields: Spaces, Capitalisation, and Consistency	214
Logical Fields: Many Encodings for the Same Idea	216
Dates: Recognise Before Repairing	217
Numeric Fields: Values That Need Questioning	218
Duplicates and Repeated Records	220
A Cleaning Audit Trail	221
Saving or Naming the Prepared Data	221
Common Beginner Mistakes	222
Mistake 1: Cleaning the raw object directly	222
Mistake 2: Changing values before inspecting them	222
Mistake 3: Treating all repeated IDs as errors	223
Mistake 4: Replacing missing values too quickly	223
Mistake 5: Assuming dates are already dates	223
Mistake 6: Running one cleaning command and moving on	223
Mistake 7: Confusing cleaning with transformation	223
What Leo Is Expected to Know at This Point	223
Chapter Outcome	224
Transition to Chapter 14	224
Selecting, Filtering, and Transforming Data: Shaping Evidence for Analysis	225
Opening Scene: When Prepared Data Still Needs Shape	225
What Changed After Chapter 13	226
The Part 3 Workflow So Far	226
From Prepared Evidence to Analytical Shape	227
A Small Visual Map of Data Shaping	227
Selection, Filtering, and Transformation Are Different Moves	228
Base R and Tidyverse: Two Ways to Express the Same Move	229
Selecting Variables: Which Columns Matter?	229
Renaming for Clarity	230
Filtering Observations: Which Rows Answer the Question?	231
Filtering with Missing Values	232
Combining Conditions	232
Filtering with a Set of Values	233
Arranging Rows for Inspection	234
Transforming Variables: Creating Clearer Evidence	234
Creating a Score Gap	235
Creating Study Bands	235
Creating Tool Readiness	236
Transformation Is Not Cleaning	237
Building a Small Analysis-Ready Dataset	237
Checking After Every Move	240
Common Beginner Mistakes	240
Mistake 1: Selecting columns before understanding the question	240

Mistake 2: Treating selection as deletion	240
Mistake 3: Filtering before checking missing values	240
Mistake 4: Using <code>&&</code> instead of <code>&</code> for row-wise filtering	241
Mistake 5: Confusing assignment with comparison	241
Mistake 6: Creating new variables without checking them	241
Mistake 7: Overwriting the prepared dataset too early	241
Mistake 8: Summarising too early	241
What Leo Is Expected to Know at This Point	241
Chapter Outcome	242
Transition to Chapter 15	242
Summarising and Grouping Data: Turning Shaped Rows into Summary Evidence	243
Opening Scene: When Many Rows Need to Speak Together	243
What Changed After Chapter 14	244
The Part 3 Workflow So Far	244
From Shaped Rows to Meaningful Summary Evidence	245
A Small Visual Map of Summarising	245
Summarising and Grouping Are Different Moves	246
Summary Questions Come Before Summary Code	247
Counting Rows	247
Summarising One Variable	248
Missing Values in Summaries	249
Counting Categories	250
Grouping: Changing the Level of the Question	250
Grouped Summaries with Base R	251
Grouped Summaries with Tidyverse	251
Two-Group Comparisons	252
Proportions and Logical Summaries	253
Grouped Proportions	253
Summary Tables Need Inspection	254
Summary Objects for Chapter 16	255
Common Beginner Mistakes	256
Mistake 1: Summarising before asking a question	256
Mistake 2: Treating <code>na.rm = TRUE</code> as a solution	256
Mistake 3: Grouping by too many variables too soon	256
Mistake 4: Forgetting that grouping changes the level of analysis	256
Mistake 5: Confusing filtering with grouping	256
Mistake 6: Trusting a summary because it looks clean	256
Mistake 7: Trying to visualise before understanding the summary	256
What Leo Is Expected to Know at This Point	257
Chapter Outcome	257
Closing Part 3	257
Saving <code>df_analysis</code> for Part 4	258
Transition to Chapter 16	259
Part 4: Seeing and Communicating Insight	260
Data Visualization: Turning Patterns into Pictures	261
Opening Scene: When the Table Stops Being Enough	261
Data Handoff from Part 3	262
Why Visualization Comes After Preparation	263

Data Frames, Tidy Data, and Why Plots Need Structure	263
A First Look at Base R Graphics	264
Other Visualization Tools Leo May Encounter	267
The Grammar of Graphics: The Idea Behind ggplot2	268
The First ggplot2 Plot: Study Hours and Project Score	269
Aesthetic Mappings with aes()	270
Mapping Versus Setting	272
Geometric Layers: The Marks on the Page	274
Bar Charts: Counts and Prepared Values	275
Histograms: Bins and Distributions	278
Lines: Change Across an Ordered Variable	280
Continuous and Discrete Variables	282
Grouping: When One Pattern Is Really Several Patterns	282
Statistical Transformations: Counts, Bins, Means, and Summaries	284
Multi-Layer Rendering Order	286
Basic Faceting: Small Multiples Without Losing the Main Plot	287
Themes: Controlling Non-Data Appearance	290
Scales and Coordinate Systems	291
Saving and Exporting Plots with ggsave()	293
A Brief Look at Interactive Visualization	294
Common Plot Problems and How to Diagnose Them	295
What Leo Can Now Do	296
Chapter Outcome	297
Chapter Design Principle	298
Clarity and Control: Designing Visualisations People Can Understand	299
Opening Scene: When a Correct Plot Still Fails	299
What Changed After Chapter 16	300
Why Good Charts Fail	300
The Clarity-Control Framework	302
Scales: Directing the Reader's Eye	303
Colour as Information	305
Themes: Taking Control of Appearance	306
Controlling Text	308
Controlling Backgrounds, Borders and Gridlines	309
Legends That Help Instead of Confuse	309
Overplotting: When Too Much Data Hides Patterns	310
Problem scenario: 10,000, 50,000, and 100,000 rows	311
Label Collisions and Dense Annotations	313
Mathematical Annotation and Rich Text	314
Aspect Ratios and Plot Geometry	316
Zooming Without Lying	317
Building Multi-Panel Visualisations	318
Introducing Interactivity	319
Practical Diagnostic Guide	320
Clarity and Control Checklist	320
What Not to Do	321
Do not polish a weak question	321
Do not use colour as decoration	321
Do not hide inconvenient values	322
Do not label everything	322

Do not confuse interactivity with clarity	322
Try It Yourself	322
Practice 1: Fix a crowded scatter plot	322
Practice 2: Improve scale labels	322
Practice 3: Repair a legend	322
Practice 4: Use <code>ggrepel</code>	322
Practice 5: Decide whether interactivity is necessary	323
A Small Practice Lab	323
Common Beginner Mistakes	323
Mistake 1: Treating the default plot as the final plot	323
Mistake 2: Adding more design before naming the problem	323
Mistake 3: Using scale limits without noticing what they remove	323
Mistake 4: Choosing colours before deciding what colour means	323
Mistake 5: Making legends carry too much work	323
Mistake 6: Labeling every visible point	323
Mistake 7: Using interactivity to avoid editing	323
Mistake 8: Forgetting that refinement is still analysis	324
Short Recap	324
Leo's Notebook	324
Glossary	324
Chapter Outcome	325
What Comes Next	325
Communication from Plots to Story: Designing Visuals People Can Understand	344
Opening Scene: When Clear Plots Still Feel Unfinished	344
What Changed After Chapter 17	344
From Exploratory Plot to Explanatory Visual	345
The Explanatory Shift	346
A Before-and-After Redesign	347
Visual Hierarchy: Giving the Reader a Scanning Path	347
Semantic Text Layers: Title, Subtitle, Tag, and Caption	349
Typographic Architecture	350
Rich Text Formatting with <code>ggtext</code>	351
Dynamic Data Highlighting	352
Direct Visual Annotation	353
Categorical Narrative Ordering	354
Multi-Plot Compositions	355
Clarity and Control Problems in R Visualization	356
Accessible and Inclusive Design	358
Responsible Claims: Description, Interpretation, and Claim	359
Production-Grade Exporting	360
A Practical Story Workflow	361
A Weak Explanation Revised	363
What a Data Story Is and Is Not	363
Common Beginner Mistakes	364
Mistake 1: Starting with decoration	364
Mistake 2: Keeping the exploratory title	364
Mistake 3: Letting the legend do too much work	364
Mistake 4: Highlighting everything	364
Mistake 5: Using category order accidentally	364
Mistake 6: Forgetting the caption	364

Mistake 7: Exporting only at the end	364
Mistake 8: Overclaiming because the plot looks convincing	364
Try It Yourself	365
Practice 1: Rewrite a title	365
Practice 2: Add a subtitle	365
Practice 3: Replace a legend	365
Practice 4: Reorder categories	365
Practice 5: Write alt text	365
Practice 6: Choose an export setting	365
A Small Practice Lab	366
Short Recap	366
Leo’s Notebook	366
Glossary	366
Chapter Outcome	367
What Comes Next	368

Part 5: Reproducible, Collaborative, and Scalable Workflows 380

Reproducible Reports with Quarto: Turning Analysis into a Document 381

Opening Scene: When the Plot Is Right but the Report Is Fragile	381
What Changed After Part 4	382
Why Copy-Paste Reporting Breaks Trust	382
What Quarto Changes	383
File, Code, Output, and Explanation in One Place	384
The Anatomy of a Quarto Document	385
YAML: The Document’s Identity Card	386
Markdown: Writing the Explanation	386
Code Chunks: Where Analysis Runs	387
Chunk Options: Controlling What Readers See	388
Tables in Quarto	389
Figures in Quarto	390
Cross-Referencing Tables and Figures	391
Hiding Code Without Hiding Evidence	392
Rendering to HTML, PDF, and Word	392
Common Beginner Mistakes	394
Mistake 1: Treating Quarto as Word with code pasted in	394
Mistake 2: Forgetting to render after changes	394
Mistake 3: Using absolute machine-specific paths	394
Mistake 4: Hiding all code without explaining the method	394
Mistake 5: Creating figures or tables without captions	394
Mistake 6: Failing to cross-reference evidence in prose	394
Mistake 7: Letting chunk options obscure the analysis	394
Mistake 8: Mixing raw and processed data without explanation	394
Mistake 9: Editing rendered output manually	395
Try It Yourself	395
Practice 1: Build a one-page report	395
Practice 2: Add a captioned table	395
Practice 3: Add a captioned figure	395
Practice 4: Hide setup but not evidence	395
Practice 5: Render twice	395
Practice 6: Check the claims	395

A Small Practice Lab	395
Short Recap	396
Leo's Notebook	397
Glossary	397
Chapter Outcome	398
What Comes Next	398
Data Connections: Moving Beyond Local Files	399
Opening Scene: When the Data Is Not Waiting in the Folder	399
What Changed After Chapter 19	400
Why Local Files Are Not the Whole Data World	400
Local Files and Connected Data	401
Different Doors Into Data	401
Reading Data from a URL	403
A Safer Pattern: Connect, Inspect, Save Deliberately	404
A Small Workflow Map	405
Online Spreadsheets as Shared Data Sources	407
APIs Without the Mystery	407
Databases Without the Intimidation	408
Credentials, Privacy, and Access	409
Connected Data and Reproducibility	409
Where Connected Data Belongs in the Project	411
Live Connection or Saved Snapshot?	411
Common Beginner Mistakes	412
Try It Yourself	412
A Small Practice Lab	413
Short Recap	413
Leo's Notebook	413
Glossary	414
Chapter Outcome	415
What Comes Next	415
Collaboration and Version Control: Making Work Traceable	420
Opening Scene: The Folder Called Final	420
Why Professional Projects Need a History	421
What Changes After Chapter 20	422
Version Control as Project Memory	423
Git as a Tracking System, Not a Magic Cloud Folder	424
GitHub as a Remote Collaboration Space	424
What Git Tracks	425
.gitignore as a Safety Tool	426
How Version Control Fits the Project Route	428
The Version-Control Workflow	429
Collaboration Without Chaos	429
What Version Control Does Not Solve	430
Common Beginner Mistakes	431
Try It Yourself	432
A Small Practice Lab	432
Short Recap	433
Leo's Notebook	434
Glossary	434
Chapter Outcome	435

What Comes Next	435
From Functions to Packages: Building Reusable R Tools	437
Opening Scene: When Useful Functions Become Hard to Control	437
What Changes After Chapter 21	438
Revisiting the Function Story Without Repeating It	439
Why Copying Functions Is Not a Workflow	440
From a Loose Function to a Small Toolkit	440
What a Package Really Is	442
The Smallest Useful Package Mental Model	442
Anatomy of a Beginner-Friendly R Package	443
The R/ Folder: Where Functions Live	444
DESCRIPTION: The Package Identity Card	445
NAMESPACE: What the Package Makes Available	446
Documentation: Explaining What the Tool Does	446
Examples: Showing the Tool in Motion	447
Tests: Teaching Leo to Trust His Tools	448
Package Development Without Overwhelm	449
A Small Package Sketch for the Book Workflow	450
When Not to Create a Package	450
Common Beginner Mistakes	451
A Small Practice Lab	451
Short Recap	452
Leo's Notebook	452
Glossary	453
Chapter Outcome	454
What Comes Next	454
Reliability and Automation: Making Workflows Repeatable	458
Opening Scene: Which Script Should I Run First?	458
The Lightbulb Idea	458
What Changes After Chapter 22	459
Reliability Before Automation	460
Why Manual Workflows Break	461
From Files to a Workflow Route	461
Script-Level Discipline	463
Workflow-Level Discipline	463
Checking Assumptions Before Running	464
Checking Folders Before Saving	465
Checking Columns Before Analysis	466
A Master Script as a First Automation Habit	466
Controlled Use of <code>source()</code>	468
Clean Sessions: The Reality Test	468
Output Discipline	469
Rendering as Part of the Workflow	470
Render-Safe Thinking	471
Small Automation Without Overengineering	472
Validation After Running	472
Logging Without Making the Project Heavy	473
The Reproducibility Checklist	474
When to Stop Automating	475
A Small Practice Lab	476

Common Beginner Mistakes	476
Short Recap	478
Leo's Notebook	478
Glossary	478
Chapter Outcome	479
What Comes Next	480
Performance and Scale: Making R Work Efficiently	481
Opening Scene: The Script Works, but It No Longer Feels Light	481
The Lightbulb Idea	481
Why Performance Belongs Near the End of the Book	482
Correctness Before Speed	483
What Scale Means in Practical R Work	485
A Diagnostic Map of Common Bottlenecks	486
Timing Code Without Becoming Obsessed with Timing	488
Vectorisation and the Danger of Row-by-Row Thinking	489
Runtime Growth: Why Small Examples Can Mislead	491
Memory, Copying, and Object Size	492
Efficient Pipelines Without Sacrificing Clarity	494
Larger-Than-Comfortable Data: Practical Options	495
Repetition Tools Are Not Magic Speed Buttons	498
When One Core Is Not Enough: A Careful Introduction to Parallel Computing	500
Performance as Workflow Design	502
A Small Applied Example Using the Learner Dataset	504
Reducing Repetition Without Hiding Meaning	506
Common Mistakes Leo Should Avoid	508
Try It Yourself	510
A Small Practice Lab	510
Short Recap	510
Leo's Notebook	511
Glossary	511
Chapter Outcome	512
What Comes Next	512
Final Integration Project: Building an End-to-End Analytical System	515
Opening Scene: The Last Question Is Not a Command	515
The Final Lightbulb Idea	515
Why Integration Is the Last Skill	516
The Final Project Brief	517
The Project Folder Structure	518
The End-to-End Workflow Map	519
Step 1: Prepare the Project Environment	520
Step 2: Import the Raw Learner Data	521
Step 3: Clean and Prepare the Evidence	521
Step 4: Create the Analysis-Ready Dataset	522
Step 5: Validate Assumptions and Outputs	523
Step 6: Summarise the Evidence	523
Step 7: Visualise the Findings	524
Step 8: Render the Final Quarto Report	525
Step 9: Preserve the Work with Version Control	525
Step 10: Review Performance and Scale	525
The Final Project Checklist	526

Common Integration Failures	527
Try It Yourself	528
Final Practice Lab	528
Leo's Final Notebook	529
Glossary	529
Book Outcome	530
Closing Reflection: From First Command to Analytical System	531
Author's Note	541
Why This Book Had to Exist	541
What I Hope Changed for You	542
The Companion Materials for This Book	543
For Teams, Teachers, Researchers, and Organizations	544
You Were Never the Problem	545
References	547

Not So Short Introduction to R

Dedication

For the learners who have carried data work by memory, patience, and trial and error, and who are ready to turn that effort into a clearer, repeatable way of working.

About This Book

This book is for people who want to learn R without being rushed past the part where confusion usually begins.

You may be a researcher, analyst, graduate student, lecturer, professional, or self-taught learner. You may already work with data in Excel, Google Sheets, SPSS, STATA, forms, dashboards, or exported files from different systems. You may have copied code from a tutorial or asked an AI tool for help, only to find that the code worked once but did not become a method you could fully explain.

You are not starting from nothing.

You already know that data work can be messy. You know that files arrive in different formats. You know that columns change names. You know that values look correct until they refuse to calculate. You know that a result is easier to trust when the path behind it is clear.

This book was built for that learner.

Not the person who wants to become a software engineer overnight.

Not the person looking for a thin list of shortcuts.

The serious beginner who wants to understand R properly, from first contact with the console to reproducible data systems.

Who This Book Is For

This book is written for learners whose work, study, or research depends on data, but who need a patient route into R.

You are likely a fit for this book if you:

- want to learn R from the ground up without being treated like you are unintelligent
- use Excel or other data tools but want a more repeatable workflow
- need to clean, transform, summarize, visualize, or report data
- have tried R before and felt lost after copying a few commands
- want to understand why code works, not only make it run once
- prefer a story-driven explanation that still respects technical accuracy
- want a practical path from first command to reproducible projects

You do not need a computer science background to begin.

You need patience, curiosity, and a willingness to let small ideas become clear before rushing into large ones.

What This Book Is Trying to Solve

Many introductions to R start too quickly.

They begin with syntax, packages, or datasets before the learner understands what kind of system R is. The beginner is asked to run commands before understanding where the commands live, what objects are, why types matter, or how the work will be saved and repeated.

That creates a familiar problem.

The learner may produce an output without building understanding.

A command works, but the meaning is unclear.

An error appears, but the cause is hidden.

A package is loaded, but the learner does not know what R itself is doing.

This book slows down at the right places.

It treats R as a working system, not only a language of isolated commands. It begins with first contact, then builds the ideas that make later data work possible: objects, types, structures, syntax, logic, functions, packages, cleaning, transformation, visualization, communication, reproducibility, collaboration, and scale.

The aim is not to make R look easy by hiding its structure.

The aim is to make R learnable by revealing the structure in the right order.

How to Read This Book

Do not read this book like a reference manual.

Read it like a learning journey.

Each chapter follows Leo, a capable beginner who wants to understand R without pretending that the early confusion is not real. Leo's questions are part of the method. When he pauses, the book slows down. When he meets a problem, the book uses that problem to introduce the next idea.

You will get the most from this book if you use it in three passes.

First, read for orientation. Let the story and explanations give you the mental map.

Second, read with R open. Run the code examples slowly and notice what changes.

Third, read for transfer. Start connecting the ideas to your own data work, research task, class activity, or reporting problem.

You do not need to master everything at once.

You need one clear concept, then another.

What You Will Need

You do not need to be a programmer before you begin.

You do need a few practical things:

- a laptop or desktop computer
- R installed from CRAN
- RStudio installed as the working environment
- a willingness to type commands rather than only read them
- a folder where you can keep scripts, data, and outputs as the book develops
- patience with early errors, because they are part of learning a language

Later in the book, you will work with the teaching dataset `df_r_migration`. It will help you practise real data ideas without jumping between unrelated examples.

For now, the most important tool is attention.

What You Will Learn to Do

By the end of this book, you should be able to:

- understand what R is and why it matters for data work
- use RStudio as a practical working environment
- create, inspect, and reuse objects
- understand data types and data structures
- recognize why values behave differently in R
- use logic and functions with confidence
- install and use packages with purpose
- clean and transform messy data
- build clear summaries and visualizations
- communicate insight through charts and written explanation
- create reproducible reports with Quarto
- organize projects so that files, scripts, data, and outputs make sense
- begin thinking about collaboration, reliability, automation, performance, and scale

The focus throughout is practical understanding.

This is not a book about sounding technical.

It is a book about becoming capable with R in a way that can survive real work.

What Makes This Book Different

This book does not begin by throwing the learner into a full dataset and hoping the concepts become clear later.

It begins with first contact.

The early chapters give Leo the mental ground: what R is, how the R ecosystem is organized, what objects are, why data types matter, and how structures shape behaviour. Only after that

foundation does the book move into data cleaning, transformation, visualization, reporting, and reproducible systems.

The teaching style is deliberate.

Small examples appear before larger ones.

Friction appears before the tool that resolves it.

Concepts are named after Leo has felt why they matter.

The book also uses one main teaching dataset, `df_r_migration`, so that the learning journey stays coherent. Leo does not jump from fruit examples to car examples to unrelated toy data. The same dataset grows with him as the work becomes more realistic.

You are not only learning commands.

You are learning how R thinks about data.

Book Roadmap

The structure follows a gradual movement from first contact to reproducible data systems.

- First, Leo understands why R changes the way data work is recorded and repeated.
- Then, he meets the R ecosystem: R, the console, RStudio, packages, scripts, projects, and the wider working environment.
- Then, he learns the foundation of R data: objects, atoms, containers, types, and structures.
- Then, he learns how R reads and executes instructions through syntax, logic, and functions.
- Then, he moves into real data workflow: coercion, cleaning, and transformation.
- Then, he learns to see and communicate insight through visualization and data storytelling.
- Finally, he builds toward reproducible reports, data connections, collaboration, automation, performance, and an end-to-end project.

This is not a race toward advanced code.

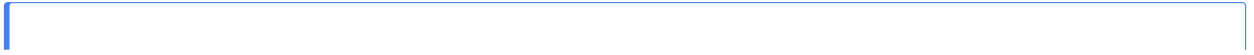
It is a steady movement toward control.

Optional Setup Check

The code blocks throughout the book are meant to be executable. This quick check confirms that your R session is working.

```
version$version.string  
#> [1] "R version 4.5.2 (2025-10-31 ucrt)"
```

Introduction: The Day R Became Less Strange



i Chapter Opening Story

You know the screen before you understand it.

Not the code. The silence.

The window opens with no spreadsheet grid, no ribbon, no familiar tabs, no rows running down the left side, no columns labelled A, B, C. There is no cell waiting for a value. No formula bar. No button that says sort, filter, clean, summarize, or chart.

Just a console.

A prompt waits on the screen.

```
>
```

It does not explain itself.

It does not ask what file you want to open.

It does not offer a tutorial.

It simply waits.

Leo looks at it longer than he expected.

He is not new to data. He has worked with spreadsheets. He has cleaned tables. He has copied values, checked formulas, dragged columns, fixed formats, removed duplicates, and wondered why the same file behaves differently on two different mornings.

Why This Book Exists

This book exists because many people are told to learn R before they are helped to understand R.

That is where the trouble often begins.

A beginner is handed syntax before meaning.

Functions before context.

Packages before the base system.

Data frames before data itself.

Plots before objects.

Projects before the learner understands where R is running.

The learner copies a line of code. It fails. They copy another line. It fails differently. They search online. They find three possible answers, none of which explains the original problem. Slowly, they begin to believe that R is only for people who were already technical before they began.

That belief is false.

R can be learned by serious beginners.

But it must be taught in the right order.

This book exists to give that order.

It does not treat the beginner as careless or incapable. It assumes the learner is new. That difference matters. A new learner does not need to be patronized. A new learner needs a path, patient explanation, honest friction, and enough repetition for ideas to settle.

Leo is the learner in this book.

He is intelligent.

He is curious.

He is cautious.

He is not stupid, and the book never treats him as if he is. He simply does not yet know how R works. His confusion is not a character flaw. It is part of the learning process.

Through Leo, this book teaches R as a movement from first contact to structured analytical work.

The journey begins with the raw R console because the learner needs to see the engine. Then the book allows the friction of that raw experience to appear. Only after that does RStudio become meaningful.

The same pattern runs through the whole book.

Exposure first.

Friction second.

Solution last.

That principle matters because tools make more sense when they arrive as answers to problems already felt.

RStudio matters after Leo experiences the limits of the console.

Packages matter after Leo sees that base R cannot comfortably do everything.

Data types matter after Leo sees that values which look similar may behave differently.

Coercion matters after Leo sees R silently change values to make them fit.

Pipelines matter after Leo feels the repetition of manual cleaning.

Visualization matters after Leo sees that tables can contain insight without making that insight visible.

Quarto matters after Leo understands that scripts alone do not fully communicate an analysis.

The book does not rush past confusion.

It uses confusion as the doorway into understanding.

What This Book Will Help You Do

By the end of this book, you will be able to move from isolated R commands to a complete, reproducible data workflow.

You will know what R is, why it exists, and why it is more than a calculator. You will know why R should be installed first and why RStudio becomes useful only after the raw console has shown its limits. You will know how to write commands, create objects, inspect values, and save code in a script instead of depending on memory.

You will learn to see R as a system.

R is the engine.

RStudio is the working environment.

Packages extend capability.

Scripts preserve thought.

Projects organize work.

Quarto turns analysis into a document that can be read, checked, rendered, and shared.

The emphasis here is understanding before speed.

This is not a trick-first book. It is not built around memorizing commands. It is built around the slow but powerful movement from confusion to control.

You will learn why R treats data the way it does.

You will learn that data begins as small pieces.

You will learn that those pieces have types.

You will learn that types affect behavior.

You will learn that values are placed into structures.

You will learn that structures determine what R can do easily and what it will resist.

You will learn why this works:

and why this does not behave the same way:

```
"10" + 5
```

You will learn why a column can look numeric and still refuse to calculate.

You will learn why missing values matter.

You will learn why NA, NaN, Inf, and NULL are not random annoyances but signals that must be understood.

Later, you will work with one teaching dataset: `df_r_migration`.

That dataset will carry the practical journey of the book. It includes learner-centred variables such as `learner_id`, `country`, `background`, `study_hours`, `confidence_score`, `project_score`, `first_script_completed`, `enrolment_date`, `first_login_date`, and `week`.

The book does not jump from one unrelated toy example to another.

One dataset grows with you.

At first, you will only hear about it. Then you will inspect it. Then you will clean it. Then you will transform it. Then you will visualize it. Then you will communicate insight from it. Finally, you will use it as part of a reproducible analytical system.

That is the real movement of this book.

From first contact to working structure.

From copied commands to understood code.

From scattered effort to reproducible workflow.

i PhD Insight Made Simple

Many beginners think confusion means they are not capable.

In R, confusion often means the learner has reached a hidden rule.

The task is not to pretend the confusion is not there. The task is to expose the rule behind it. Once Leo understands the rule, the problem becomes less personal and more technical. Technical problems can be inspected, tested, corrected, and practiced.

That is why this book slows down at the moments where many tutorials speed up.

! First Contact Application

Before you continue, write down your honest starting point with R.

Answer these five questions:

1. Have you installed R before, or is this your first time?
2. When you see the R console, what feels most unfamiliar?
3. What kind of data work do you currently do most often?
4. What part of that work feels repetitive, fragile, or hard to reproduce?
5. What would you like R to help you do with more confidence by the end of this book?

Do not solve anything yet.

Do not try to sound advanced.

Just name your starting point clearly. Later, this becomes your evidence of progress.

Part 1: Foundations

—

Before the First Command: Why R Changes the Way Data Work Thinks

Opening Scene: Leo Before the Prompt

The problem is not that Leo cannot work with data.

He can.

He has cleaned spreadsheets late at night. He has copied formulas across hundreds of rows. He has renamed workbooks carefully, then still wondered which version was correct. He has opened files called `final`, `final_updated`, and `use_this_one`, knowing that none of those names guarantees safety.

He has repaired formulas that worked yesterday and failed today because one column moved. He has copied tabs from one workbook to another. He has made manual corrections that were obvious at the time, then difficult to explain a week later. He has watched data travel from file to file until the final result looked neat, while the route behind it became harder to defend.

That is the pressure of spreadsheet-heavy work.

The finished output may look controlled. The chart may be clean. The summary may be accepted. But underneath the result sits a chain of actions that only Leo remembers: the filter he applied, the rows he removed, the lookup he corrected, the formula he extended, the duplicate he fixed by hand.

And memory is a fragile place to store a method.

So when Leo hears people talk about R as powerful, difficult, and important for serious data work, he is not sure what to believe. Some people describe R as if it belongs only to statisticians, programmers, or people who have been coding since childhood.

Leo is not afraid of learning.

He is cautious.

That caution is reasonable. A new tool can make an intelligent person feel slow at first. R will ask Leo to work differently. It will not begin with rows, columns, ribbons, or a familiar grid. It will begin with instructions.

This chapter prepares Leo for that first contact.

He will not master R here. He will not clean a dataset. He will not build a chart. He will not write a full analysis.

Before the first command, he needs to understand why the first command matters.

i Chapter Focus

This chapter is not about doing everything in R. It is about helping Leo understand what kind of tool R is, and why it changes the way data work is recorded, repeated, and defended.

The Problem Was Never Leo

Many people meet R after years of working with spreadsheets.

That history matters.

It means Leo is not arriving empty. He already understands more about data work than he may realize. He knows that data can be messy. He knows that categories can be inconsistent. He knows that dates can behave strangely. He knows that one small change in a source file can affect the whole report.

But he has learned those lessons inside tools that often hide the path behind the result.

A spreadsheet can show the final number without clearly showing every step that produced it. A workbook can contain formulas, hidden columns, copied sheets, manual corrections, and pasted values. Some of the logic is visible. Some of it is buried. Some of it exists only in the memory of the person who prepared the file.

That is not a personal weakness.

It is a workflow problem.

Leo's issue is not lack of ability. His issue is that his previous tools have allowed too many important steps to remain scattered, manual, or difficult to repeat.

R matters because it changes that relationship.

It gives Leo a way to write the work down.

Not just the answer.

The path.

What R Is

R is a programming language built for data work. It focuses on statistical computing, data analysis, and data visualization.

It was created in the early 1990s by Ross Ihaka and Robert Gentleman at the University of Auckland. The name “R” comes partly from the first letters of their names and partly from its relationship to the earlier S language.

For Leo, the simplest definition is enough:

R is a language for giving data-related instructions to a computer.

That definition matters because it separates R from many tools Leo may already know.

R is not only an application that opens on the screen. It is not only a calculator. It is not only a statistics program. It is a language Leo can use to express data work clearly and repeatably.

In a spreadsheet, Leo often works by clicking, dragging, filtering, copying, and pasting. In R, he writes instructions. Those instructions can be saved, inspected, corrected, rerun, and shared.

This is the beginning of a different relationship with data.

Leo is no longer only manipulating what he sees on the screen. He is beginning to describe what should happen to the data.

i Plain Definition

R is a language for working with data through written instructions. Those instructions may begin with simple commands, but they can grow into complete workflows for cleaning, analysis, visualization, and reporting.

Why R Exists

R exists to solve one core problem: working with data efficiently and reproducibly.

Data work becomes fragile when it depends only on memory, manual calculations, copied formulas, spreadsheet tabs, and file versions with names like `final`, `final_updated`, or `use_this_one`.

Those habits may get work done for a while. But as the work grows, the risk grows with it. More files. More columns. More people. More updates. More chances for the method to drift away from the result.

R gives Leo a way to describe his work as code.

With R, Leo can eventually:

- store data in structured objects
- clean and transform data
- run statistical analysis
- create visual outputs
- save the instructions behind the result
- repeat the same workflow later

The main idea is simple:

R does not only help Leo get an answer. It helps him preserve the path to the answer.

That path matters.

In serious data work, the result is only part of the story. Leo also needs to know how the result was produced. He needs to be able to return to the work, check it, revise it, and explain it to someone else.

R supports that kind of work because the steps can be written down.

This does not mean R removes every problem. No tool does. But R gives Leo a stronger way to control work that would otherwise depend too much on manual repetition.

Why R Feels Different from Excel

Leo is used to visible tools. In Excel, he can see rows, columns, cells, formulas, tabs, filters, charts, and menus. The workbook gives him a surface to touch. He can point to a cell, edit it, drag a formula, and see the change immediately.

R feels different because it begins with written instructions.

In a spreadsheet, Leo often sees the data before he writes the logic. In R, he often writes the logic that tells the computer what to do with the data.

That shift can feel strange at first. The screen may look plain. The instructions may feel strict. A missing quotation mark or a wrong capital letter may cause an error.

This does not mean Excel is bad or R is automatically better.

They solve different kinds of problems.

Excel is useful for quick inspection, manual editing, familiar business workflows, and situations where direct visual interaction is helpful.

R becomes useful when work needs to be repeated, checked, documented, scaled, or shared.

Excel Is Not the Enemy

Leo does not need to reject Excel in order to learn R.

The goal is to understand when spreadsheet work is enough and when a repeatable code-based workflow becomes safer, clearer, and easier to maintain.

The Shift from Manual Work to Reproducible Work

The deeper promise of R is reproducibility.

A spreadsheet workflow may depend on memory:

- which file was opened
- which rows were copied
- which formulas were dragged
- which filters were applied
- which chart was updated manually

Those steps may be familiar, but they can also become invisible. If something changes, Leo may have to reconstruct the process from memory. If someone else asks how the result was produced, he may have to explain a long chain of manual actions that were never fully recorded.

This is where R changes the nature of the work.

R encourages Leo to write those steps down as instructions.

A written instruction can be saved. A saved instruction can be rerun. A rerun instruction can be checked. A checked instruction can be corrected. A corrected instruction can be shared. A shared instruction can be understood by someone else.

The work becomes less dependent on what Leo remembers and more dependent on what the workflow records.

The shift is not from “non-technical” to “technical.”

The shift is from hidden steps to visible steps.

Leo is learning a clearer way to express work he already understands.

The Real Shift

The real shift is not from Excel to R.

The real shift is from work that depends on memory to work that can be repeated from saved instructions.

The First Lightbulb Moment

At some point, the reason for R becomes clearer.

R is not mainly about replacing Excel.

It is not mainly about looking more technical.

It is not mainly about writing clever code.

R matters because it makes the invisible steps of data work visible, repeatable, and defensible.

That is the lightbulb moment.

When Leo works only through manual actions, the result may survive, but the route can disappear.

When Leo writes the work as instructions, the route remains. It can be read. It can be questioned.

It can be improved. It can be run again.

This changes the kind of confidence Leo can have in his work.

He is no longer saying only:

Here is the answer.

He is moving toward being able to say:

Here is the answer, and here is the path that produced it.

That distinction matters in research. It matters in business. It matters in teaching. It matters anywhere results need to be trusted.

R gives Leo a way to make his data work accountable.

Not perfect.

Accountable.

What Leo Is Expected to Know at This Point

At this point, Leo does not need to know RStudio, packages, data frames, scripts, projects, or visualization.

He only needs to understand a few ideas:

- R is a language for data work
- R exists to make data work more repeatable
- R feels different because it uses written instructions
- R is useful when work needs to be checked, repeated, and improved

The hands-on part begins in Chapter 2.

This matters because beginners often feel pressure to understand everything immediately. Leo does not need that pressure here. He only needs enough orientation to make the first direct encounter with R meaningful.

Not Yet

Leo is not expected to understand RStudio, packages, data frames, projects, or reporting yet.

Those ideas will come later. For now, he only needs to understand why R is worth meeting.

Chapter Outcome

By the end of this chapter, Leo has:

- understood what R is
- understood why R exists
- seen why R differs from spreadsheet-only work
- understood the value of reproducible data work
- prepared mentally for his first direct interaction with R

Leo has not written code yet.

That is deliberate.

Before learning commands, he needed to understand the problem those commands will help solve. He now sees that R is not simply another software package to click through. It is a language for giving data-related instructions to a computer. It helps turn hidden manual steps into visible, repeatable work.

That understanding is enough for now.

The first command will make more sense because of it.

Transition to Chapter 2

Leo is now ready to meet R directly.

In the next chapter, the idea becomes contact. He will install R, open the console, and face the plain prompt where every R user eventually begins:

```
>
```

That small symbol will ask him for one thing.

An instruction.

Ready for First Contact

Chapter 1 gives Leo the reason for learning R.

Chapter 2 gives him the first direct experience of using it.

Leo has not typed anything yet.

But the system is waiting.