Computational Analysis of Educational Data: A Field Guide Using R

Wei Wang, Mete Akcaoglu, Joshua Rosenberg, Shaun Kellogg

2026-01-12

Table of contents

Preface

Preface

Welcome to *Computational Analysis of Educational Data: A Field Guide Using R*!

Why this book?

Conventionally, as social scientists, we start the research process by generating research questions based on our previous knowledge and theories in the field. This is the conventional model of the order of operations in the process, and it is still the normative view of how it should work. However, it is also epistemologically possible that our observations of the world can guide our research questions. The process doesn't always proceed in the orderly linear fashion suggested by the conventional model.

We are limited in how we see the world: we don't know what we don't know. For example, for someone who does not know that Twitter posts can be collected and analyzed as data to capture the state of the world, studying Twitter data will most likely not become a topic for a research question. Once you start seeing what can be data in the world, it starts shaping our ideas of what is "researchable". Here is a simple model that we propose that shows the reciprocal nature of the research process, which is at the core of our aims in this book:

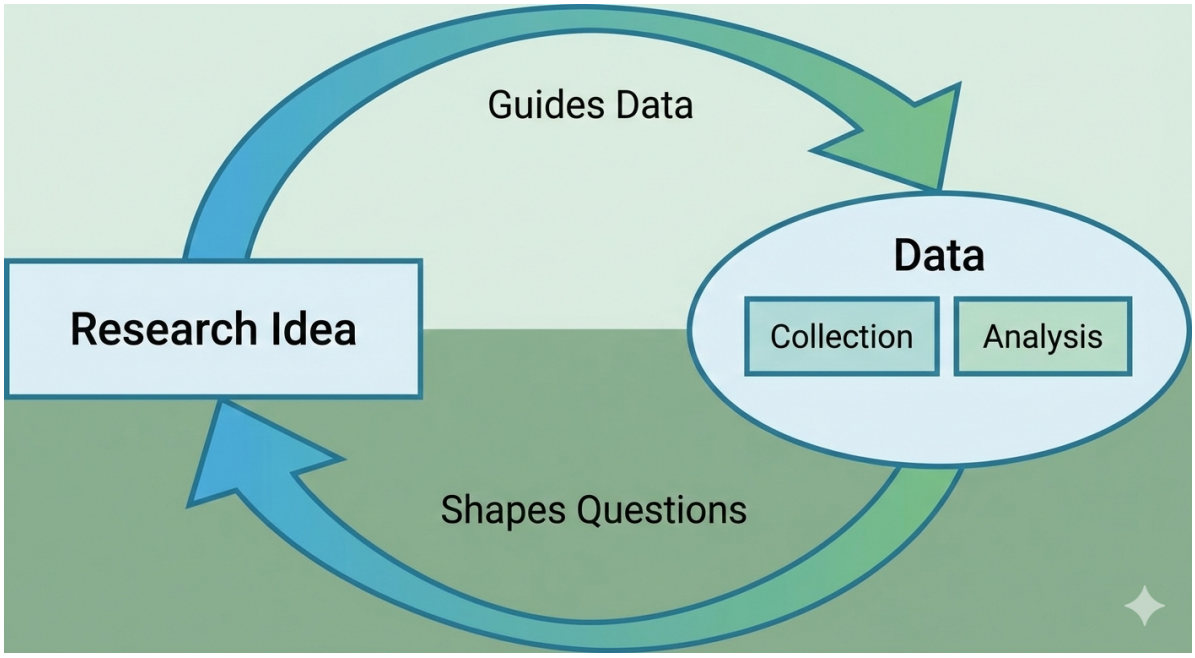


Figure 1: Figure 1. Research - data relationship (Generated using Gemini Pro based on an earlier drawing by the authors)

We are writing this book because in the past few years what we described above has started happening for us. We have published work on Twitter data, Social Network analyses, natural language processing, and machine learning that was only possible after we learned what kind of data already existed around us. We thought other social scientists may benefit from a resource that not only tells them what is available as data but also guides them through concrete examples of going through this research journey along with us. We hope that along this journey you will develop your own research questions, and maybe even replicate some of the studies we imagined in this book.

The second most important aspect of this book that does not exist out in the wild is the "field guide" process where we work through the research design and reporting process. To do that, for each new computational research method, we follow this process:

1. Start with what makes good data for that analysis (and how to capture it)
2. See what the data looks like (what it **has to **have, and what it can have)
3. Formulate sample research questions based on the resources provided by the data and the type of analysis.
4. Go through the analyses in R
5. Provide a sample write up for Methods

6. Provide a sample write up for Results

Who is this book for?

This book will be beginner-friendly but not for the absolute beginner. We will dedicate the initial chapters to take you to resources that will help you get started. But, to make sense of this book, you should have basic research design knowledge, basic statistical knowledge, and a basic understanding of R and RStudio. At the same time, this book will not be for experts or expertise. There are already many great resources that delve into the topics that we cover (e.g., Silge's book on using text data for machine learning).

We imagine that this book can become a part of doctoral coursework for future researchers, opening the doors for new ways to look at the world for research and data. Likewise, senior and junior academics/researchers would benefit greatly from this book to help them expand their research agendas.

For researchers like ourselves, we think this book can serve as a fun summer reading to rejuvenate and get excited about new research. At the same time, we also envision this book as a guidebook to keep on the side and frequently refer to, as researchers write up their work using these new methods.

This book will provide new ways to look at the world and formulate RQs. It will guide through the research process for each new method (including, friendly data organization tips, template for writing up and sharing this. We hope that you join us in this journey and this work will help open up new doors/embark on a journey of using computational research methods.

Organization of the books

The book is organized around four sections. Within these sections, there are specific chapters, which serve as field guide "entries" or "cases". While the section overviews (the first chapter within each section) introduce the techniques or methodologies used in the rest of the section's chapters, the chapters are intended to address a specific, narrow case, where we provide a sample write-up for researchers in writing their research questions, methods, results (and discussions) sections based on the analyses.

Getting Started

Positron and RStudio

You need an environment for writing and running R code. We're going to focus on **Positron**, Posit's newer IDE built on the foundation of VS Code. If you're already comfortable with RStudio, everything in this chapter applies there too—the fundamental concepts are identical, and the R code in this book works the same way in both.

Why Positron?

Positron represents where computational social science workflows are heading: multi-language support (R and Python in the same environment), modern editor features inherited from VS Code, and a more extensible architecture. If you're coming from VS Code, Positron will feel immediately familiar. If you're new to both, you're learning a tool that's designed for the future of data science.

That said, **RStudio remains great**. It's mature, widely documented, and has a loyal community. Some researchers prefer its more opinionated interface—everything you need is visible and organized for R-first workflows. If you're already using RStudio and prefer it, nothing in this book requires switching.

The instructions below work in both environments unless otherwise noted. When we say "Positron," assume the same applies to RStudio.

The Four Essential Panes

Both Positron and RStudio organize your workspace into four main areas (though the layout is customizable):

- **Source/Editor** (top-left): Where you write and edit your code files (.R scripts, .qmd documents)
- **Console** (bottom-left): Where R actually executes code and shows results in real-time
- **Environment/Variables** (top-right): Shows the objects (data, functions, etc.) currently loaded in memory
- **Files/Plots/Help** (bottom-right): Navigate your project files, view visualizations, and access documentation

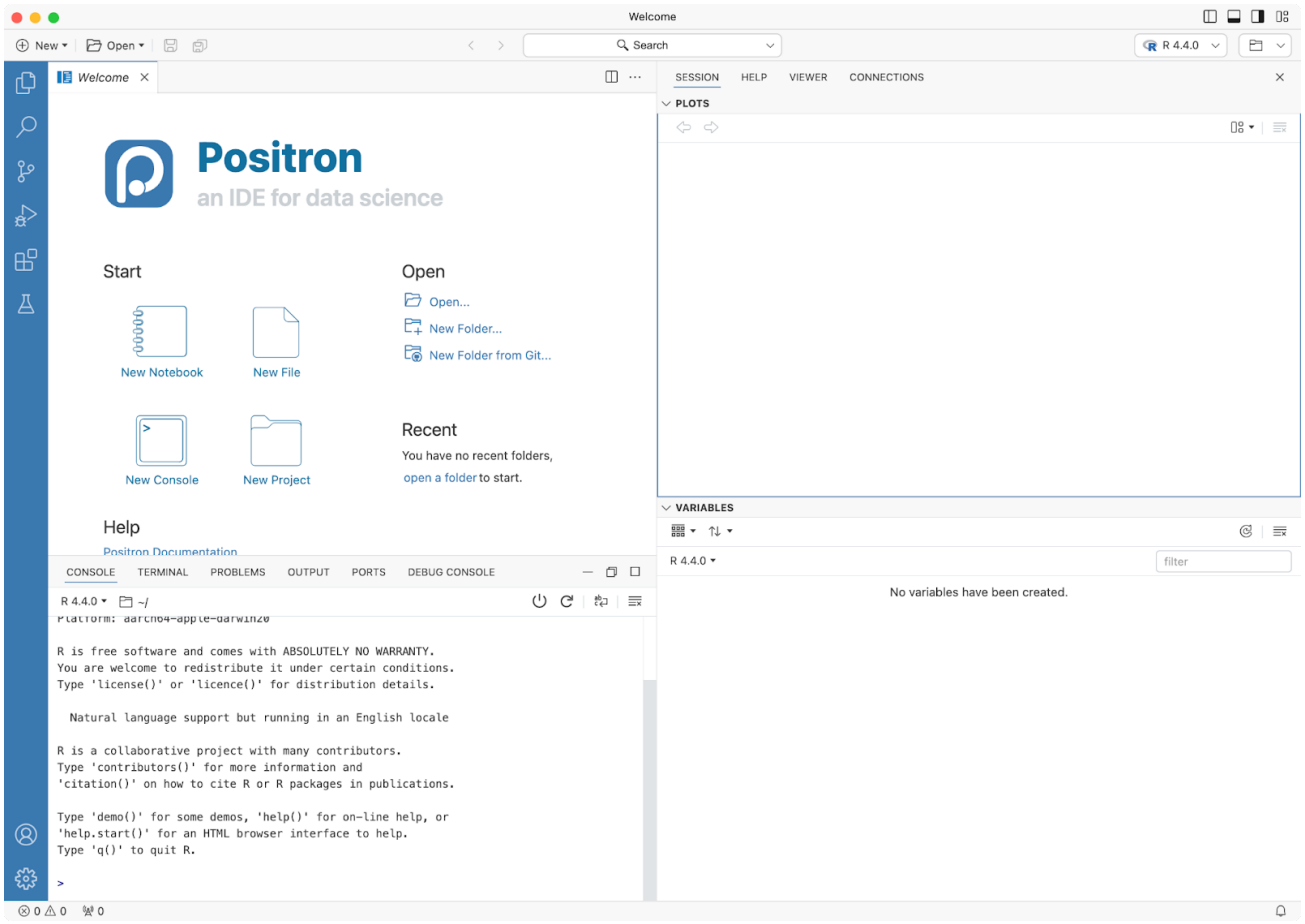


Figure 1: Positron Interface

Understanding Each Pane

Source/Editor Pane: This is where you spend most of your time writing code. The editor provides syntax highlighting (different colors for functions, strings, and comments), autocomplete suggestions, and the ability to work with multiple files in tabs. You can split the pane to view two files side-by-side, and for Quarto documents, you can preview the rendered output alongside your code.

Console Pane: When you run code, it executes here and displays results immediately. Press the up arrow to cycle through previously run commands—useful when you want to re-run or modify something. The prompt `>` means R is ready for input; a `+` means R is waiting for you to complete a command (often because you forgot a closing parenthesis). You can clear the console output with Cmd/Ctrl + L, though this doesn't delete your objects from memory.

Environment/Variables Pane: This shows every object currently loaded in R—datasets, variables, functions you've created. Click on a dataset name to open it in a spreadsheet-style viewer. The display also shows object types (data frame, list, numeric vector) and basic information like the number of rows and columns. You can remove objects using the broom icon, which clears your workspace—useful when starting fresh.

Files/Plots/Help Pane: The Files tab lets you navigate your project directory, create new folders, and open files. The Plots tab shows your visualizations with forward/back arrows to review previous plots and an Export button to save images. The Help tab displays R documentation when you run commands like `?mean` or search for functions. In RStudio, you'll also see a Packages tab for installing and loading packages; Positron handles packages differently but with similar functionality.

The defaults work fine for now. As you get comfortable, you might adjust panel sizes or move things around, but the essential logic stays the same: code on the left, context on the right.

R Projects

Always work in an R Project. This is the single most important RStudio habit for reproducibility.

An R Project is a folder that contains all files for a particular analysis—data, scripts, outputs—and sets your working directory automatically. This means your code will work on any computer without changing file paths.

To create a new project:

1. File → New Project
2. Choose "New Directory" or "Existing Directory"
3. Name your project (e.g., "my-first-css-analysis")
4. Click "Create Project"

Notice the `.Rproj` file in your folder? That's your project file. Double-click it to open Positron or RStudio with the correct working directory already set.

Essential Keyboard Shortcuts

Keyboard shortcuts can speed up your workflow considerably. You don't need to memorize them all at once—start with a few that feel natural, and others will follow as you get more comfortable.

Action	Windows/Linux	macOS
Run current line or selection	Ctrl + Enter	Cmd + Enter
New document	Ctrl + Shift + N	Cmd + Shift + N
Save file	Ctrl + S	Cmd + S
Find in file	Ctrl + F	Cmd + F
Comment/uncomment code	Ctrl + Shift + C	Cmd + Shift + C
Insert pipe operator (`|>`)	Ctrl + Shift + M	Cmd + Shift + M
Insert assignment operator (`<-`)	Alt + -	Option + -
Restart R session	Ctrl + Shift + F10	Cmd + Shift + 0

The run command (Ctrl/Cmd + Enter) is the one you'll use most—it executes the line where your cursor is, or if you've selected multiple lines, it runs all of them. The comment shortcut is helpful when testing code: you can quickly comment out a section to see if removing it fixes an error.

Positron and RStudio have slightly different shortcuts for a few operations, particularly around navigating panes and restarting R. To see the full list specific to your environment, go to Help → Keyboard Shortcuts or press Alt/Option + Shift + K.

RStudio Keybindings in Positron

If you're coming from RStudio and want to keep using familiar shortcuts, install the **RStudio Keymap** extension in Positron. Open the Extensions panel (View → Extensions), search for "RStudio Keymap," and click Install. This maps common RStudio shortcuts to their Positron equivalents so you don't have to retrain your muscle memory.

Extensions and Customization

Positron is built on VS Code, which means most VS Code extensions work in Positron. This gives you access to a large ecosystem of tools beyond what's built into the base IDE.

To access extensions, open the Extensions panel—usually visible in the left sidebar, or accessible via View → Extensions. From there, you can search for extensions by name or browse categories.

Useful Extensions for R Work

While Positron comes with solid R support out of the box, these extensions can enhance your workflow:

- **Rainbow CSV**: Colors columns in CSV files to make them easier to read
- **GitLens**: Shows detailed Git information inline (who changed what, when, and why)
- **Markdown Preview Enhanced**: Better previews for Markdown documents with extended syntax support

To install an extension, search for it by name in the Extensions panel, click on it, and press Install. Most extensions work immediately without restarting Positron.

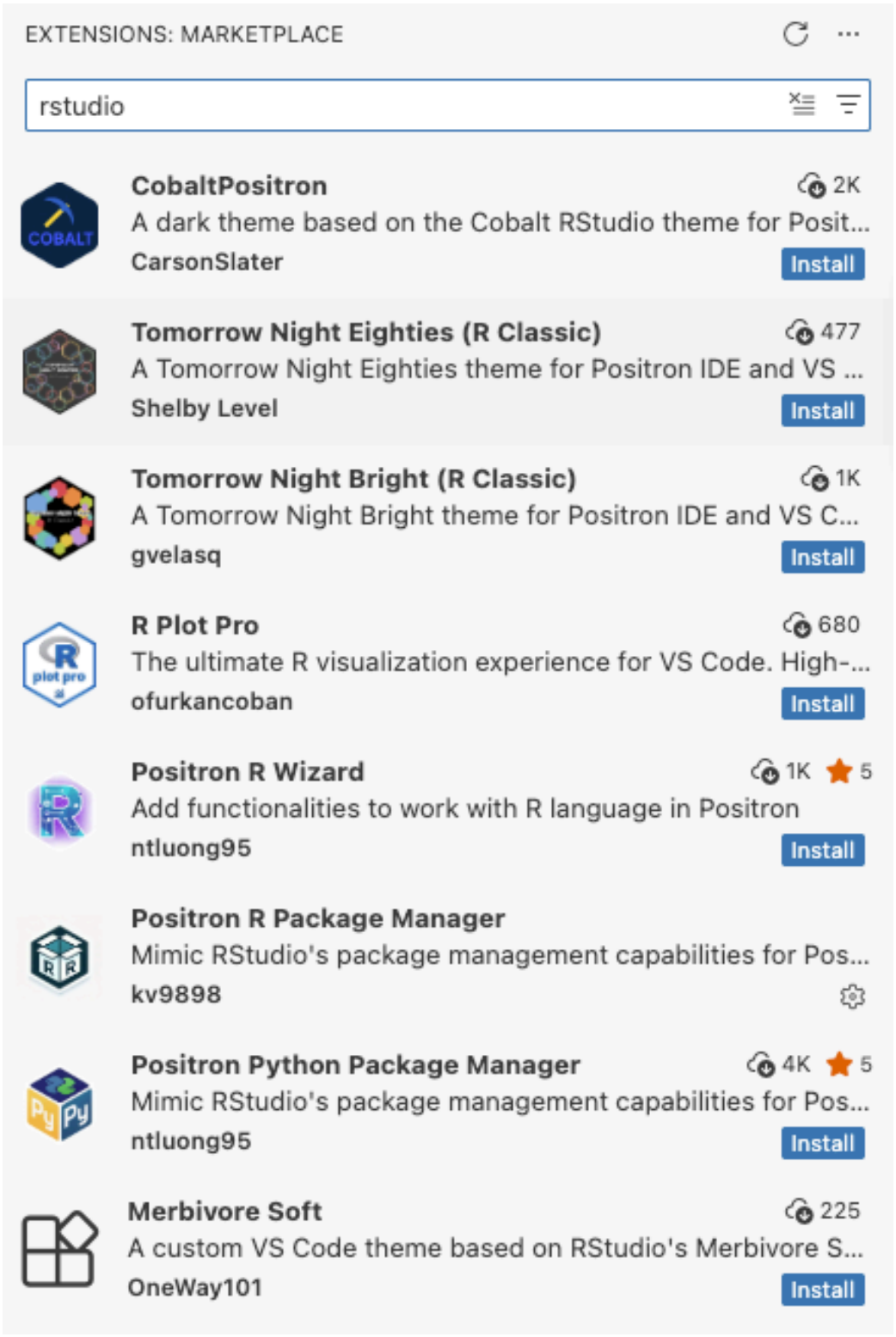


Figure 2: Extensions marketplace showing R-related extensions

RStudio's Approach

RStudio uses "addins" instead of extensions. These are R packages that add functionality to the IDE interface. The ecosystem is smaller than VS Code's, but focused specifically on R workflows. If you're using RStudio, you can find addins through the Addins menu once you've installed packages that provide them.

Customization

Both Positron and RStudio let you customize themes, font size, and panel arrangements. The defaults work well to start, but if you find yourself squinting at text or want a different color scheme, explore Preferences/Settings → Appearance. Changes are a matter of personal preference rather than workflow improvement, so don't feel pressure to customize immediately.

What Makes Positron Stand Out

Beyond the basics, Positron offers several features that become valuable as your projects grow in complexity:

Multi-language support: You can work with R and Python in the same project without switching between different environments. If you later need to use Python libraries or call Python code from R (common in machine learning workflows), Positron handles both languages natively.

Terminal integration: The built-in terminal in Positron has better shell support than RStudio's terminal. This matters when you need to run command-line tools, manage Docker containers, or work with system utilities alongside your R code.

Remote development: Through SSH extensions, you can connect to remote servers and work on them as if they were local. Your code runs on the server, but Positron's interface stays responsive on your machine. This is useful when analyzing large datasets that don't fit on your laptop or when you need more computing power.

Performance: The editor itself is noticeably faster with large files—syntax highlighting, scrolling, and searching all feel snappier. This doesn't matter much for typical scripts, but when you're working with thousand-line codebases or large Quarto documents, the difference shows.

Modern editor features: IntelliSense provides context-aware code completion (it knows what functions are available on specific objects), bracket matching highlights matching parentheses and braces automatically, and a minimap gives you a bird's-eye view for navigating long files.

These features are there when you need them. For now, just know they exist—they're part of why Positron represents where these tools are heading.

Version Control with Git and GitHub

Git is how computational social scientists version, share, and collaborate on their work. GitHub is the most popular platform for hosting Git repositories. If you've never used version control before, this section will get you set up with a workflow that integrates directly into your IDE.

Why Git Matters

Without version control, you end up with files like `analysis_final.R`, `analysis_final2.R`, `analysis_ACTUALLY_final.R`. Git tracks every change you make, so you can:

- Go back to any previous version without keeping multiple copies
- Experiment freely—if something breaks, you can always revert
- Collaborate with others without emailing files back and forth
- Share your work publicly for transparency and reproducibility

Most computational social science projects live in Git repositories, and employers expect researchers to be comfortable with basic version control.

Git Integration

Once Git is configured, you'll see version control features built into your environment:

- **In RStudio**: A "Git" tab appears in the Environment pane (top-right)
- **In Positron**: Git integration works through the Source Control panel (often on the left sidebar)

Either way, you can stage changes, write commit messages, and push to GitHub without leaving your editor. This is far friendlier than memorizing terminal commands.

Setting Up Git and GitHub: The Path of Least Resistance

Git authentication is notoriously confusing. We're going to use an R-native approach that minimizes pain: **Personal Access Tokens (PAT) via the `usethis` package**. This keeps you in familiar territory and works reliably for the kind of work you'll do in this book.

Step 1: Install Required Packages

```
install.packages("usethis")
install.packages("gitcreds")
```

Step 2: Configure Your Git Identity

Tell Git who you are (one-time setup):

```
usethis::use_git_config(
  user.name = "Jane Doe",           # Your actual name
  user.email = "jane@example.com"   # Email associated with GitHub
)
```

Important: Use the same email address you'll use (or already use) for your GitHub account.

Step 3: Create a GitHub Personal Access Token

```
usethis::create_github_token()
```

This opens GitHub in your browser. You'll need to:

1. Log in to GitHub if you haven't already
2. Leave the default scopes checked (at minimum: "repo" and "workflow")
3. Set expiration to 90 days (good security practice, though you'll regenerate when it expires)
4. Click "Generate token" at the bottom
5. **Copy the token immediately**—GitHub only shows it once

The token looks like: `ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`

Step 4: Store Your Token in R

```
gitcreds::gitcreds_set()
```

When prompted, paste your token and press Enter. R stores it securely for future use.

Step 5: Verify Everything Worked

```
usethis::git_sitrep()
```

This diagnostic report should show: - Your name and email - "Personal access token for 'https://github.com': '<discovered>' "

If so, you're all set. If not, see troubleshooting below.

Alternative: SSH Keys (For the Long Haul)

SSH keys can be convenient—set them up once and never think about authentication again. But they're more complex to configure initially.

Interested? See Happy Git with R, Chapters 10-12. For now, the PAT method will serve you well.

Using Git in Practice

We'll introduce Git commands as you need them throughout the book. For now, know that your IDE gives you a graphical interface for the most common operations:

- **Pull**: Download changes from GitHub (always do this before starting work)
- **Commit**: Save a snapshot of your changes with a descriptive message
- **Push**: Upload your commits to GitHub

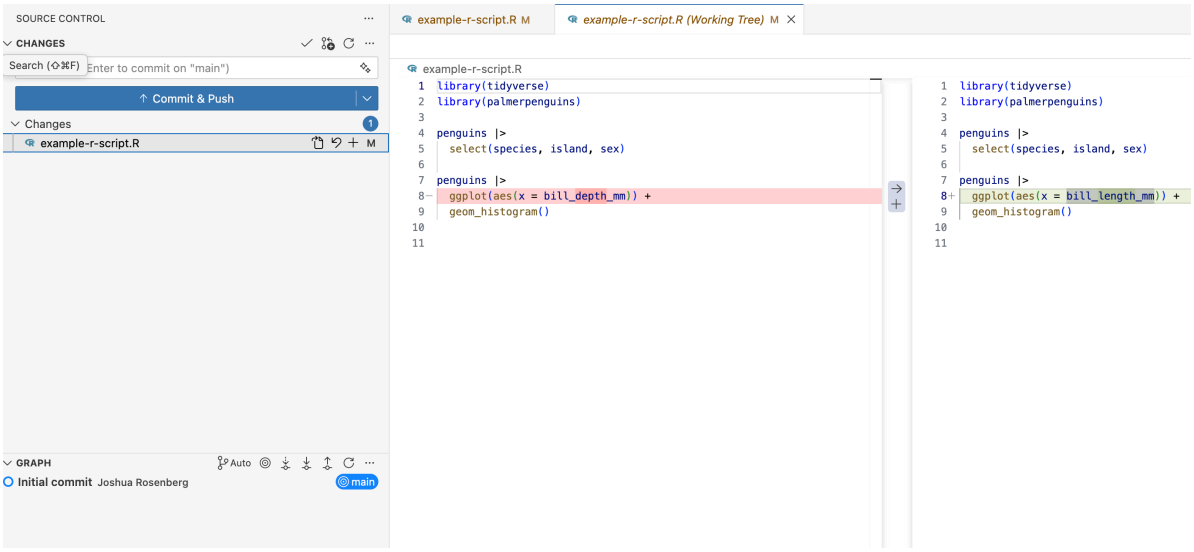


Figure 3: Source Control panel showing staged changes

You can also use terminal commands (`git status`, `git add`, `git commit`, `git push`) if you prefer, but the IDE interface handles 90% of everyday tasks.

The key habit: commit often with clear messages ("Add data cleaning script" rather than "changes"), and push regularly so your work is backed up and others can see your progress.

A Brief Note on Branches

So far we've discussed the basic workflow on a single branch (usually called `main`). Branches let you create parallel versions of your code for experimentation or collaboration without affecting your main work.

Think of branches as separate workspaces: you can try a new analysis approach on a feature branch, and if it works out, merge it back into main. If it doesn't work, you can simply delete the branch—your main code remains untouched.

When to use branches:

- **Experimenting with new approaches**: Try different analytical methods without risking your working code
- **Collaborating on specific features**: Each person works on their own branch, then merges when ready
- **Maintaining different versions**: Keep a stable main branch while developing updates separately

Your IDE makes branch management visual. In Positron, the Source Control panel shows your current branch and lets you create, switch, and merge branches through menus. In RStudio, the Git pane has a branch dropdown for the same operations.

For now, working on the main branch is sufficient. We'll cover branching in more detail when you need it for collaboration or when your projects become more complex.

Going Further

This chapter gave you a working environment. When you're ready to explore more:

- Positron Documentation: Official docs for Positron features
- RStudio User Guide: Comprehensive guide to all RStudio capabilities
- Happy Git and GitHub for the useR: Everything Git-related for R users
- GitHub Skills: Interactive tutorials for Git and GitHub

Advanced features worth exploring later:

- **Code snippets**: Type abbreviations that expand to code templates
- **Debugger**: Step through code line-by-line when things break
- **Live Share** (Positron/VS Code): Collaborate in real-time by sharing your workspace with others

R

We're not going to teach you RStudio comprehensively because excellent resources already exist for that purpose. We particularly recommend *R for Data Science*, *Data Science in Education Using R*, the RStudio primers, and *Happy Git and GitHub for the useR*.

Instead, this chapter covers just enough to reproduce our workflows and extend them to your own data science work. Think of this as the minimal viable setup for doing the work in this book.

Three Ways to Write R Code

Positron supports multiple file types for writing R code. You'll encounter all three in data science work:

R Scripts (.R)

Plain text files containing R code, executed line-by-line or all at once. Great for data processing pipelines, functions, and code you'll reuse.

Create one: File → New File → R Script

```
# This is an R script
# Lines starting with # are comments
data <- read.csv("mydata.csv")
summary(data)
```

Here's a more realistic example using student assessment data. Save this as `explore-data.R`:

```
# Example: Quick data exploration in an R script
library(tidyverse)

# Read the data
students <- read_csv("data/oulad-students-and-assessments.csv")
```

```r
# Quick summaries
summary(students$mean_weighted_score)
table(students$final_result)
```

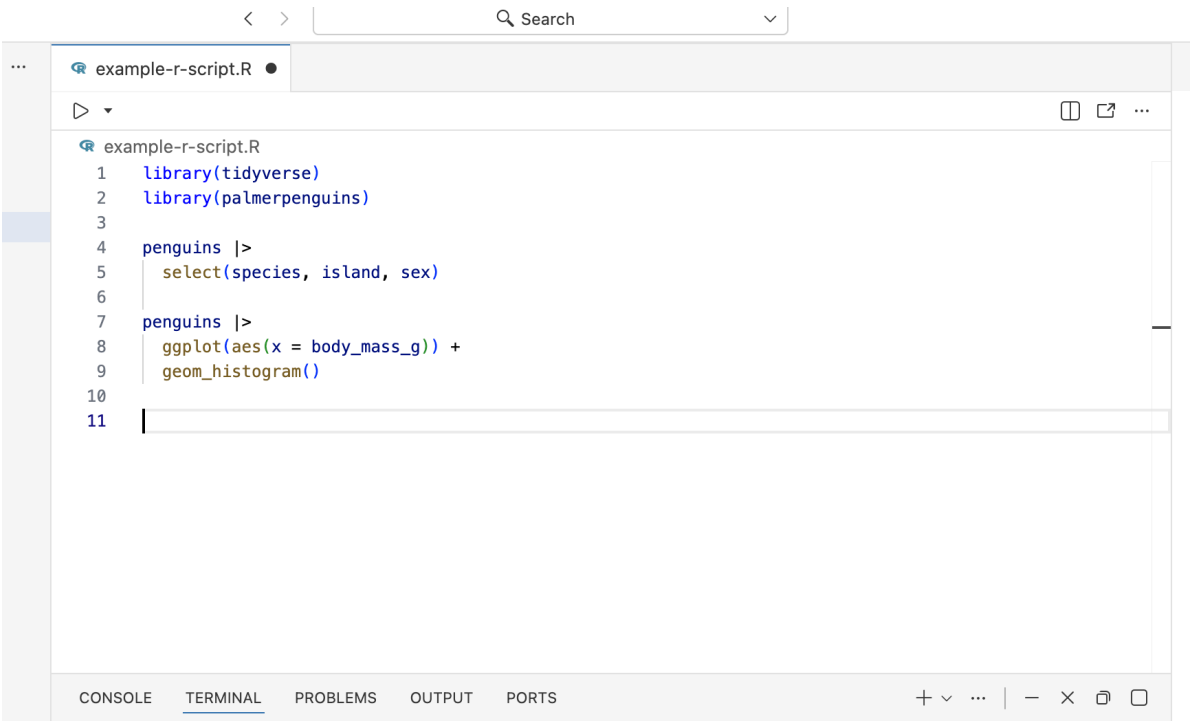


Figure 1: R script open in Source pane

This script loads the tidyverse package (more on this in the next chapter), reads a CSV file, and produces quick summaries. You can run it line-by-line using Cmd/Ctrl + Enter, or run the entire script at once. The output appears in the Console, but the script itself is saved for later use.

R Markdown (.Rmd)

Documents that weave together narrative text (in Markdown) and code chunks. When you "knit" an R Markdown file, it executes the code and generates a formatted document (HTML, PDF, or Word).

You should know R Markdown exists because you'll encounter it in older resources and projects, but we're not using it in this book.

Quarto (.qmd)

Quarto is the successor to R Markdown, with better multi-language support (R, Python, Julia) and more consistent syntax. It's what we'll use throughout this book.

Create one: File → New File → Quarto Document

The structure looks similar to R Markdown, but the rendering engine is more powerful:

````
---
title: "My Analysis"
format: html
---

## Introduction

This is regular text.
```{r}
# This is a code chunk
x <- 1:10
mean(x)
```
````

Click the "Render" button (or Ctrl/Cmd + Shift + K) to execute all code and generate your document.

Here's a more complete Quarto example analyzing student data:

````
---
title: "OULAD Student Analysis"
format: html
---

## Loading Data

First, we'll load the student assessment data from the Open University
Learning Analytics Dataset:

```{r}
library(tidyverse)
students <- read_csv("data/oulad-students-and-assessments.csv")
```
````

```
## Pass Rates by Gender

Let's examine how pass rates differ by gender:

```{r}
students |>
  group_by(gender, pass) |>
  count() |>
  pivot_wider(names_from = pass, values_from = n)
```

The table above shows the number of students who passed (1) and didn't
pass (0) for each gender. We can see that both groups have substantial
representation in the dataset.
```

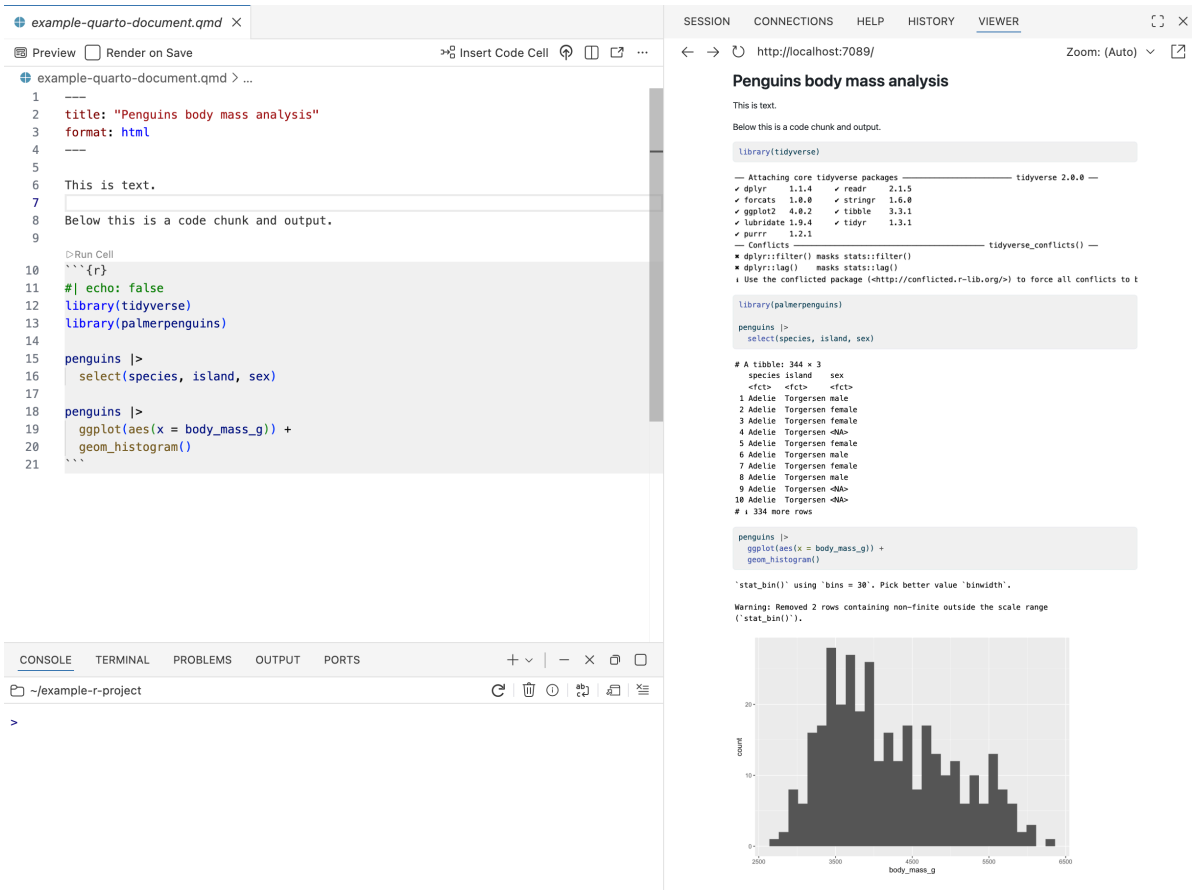


Figure 2: Rendered Quarto document showing code and output

When you render this document, Quarto executes each code chunk in order, captures the output, and weaves it together with your explanatory text. The result is an HTML document where readers see your code, your results, and your interpretation all together. This makes your analysis reproducible—anyone with your data can re-run your Quarto file and get the same results.

Key difference for beginners: Think of R scripts as "just code" and Quarto as "code + explanation + output" in one document. Use scripts for behind-the-scenes work, Quarto for analysis you want to share or understand later.

Controlling Chunk Behavior with Options

Quarto lets you control what appears in your rendered document using chunk options. These are specified at the top of each code chunk using `#|` syntax.

Common options include:

- `#| echo: false` — Hide the code, show only the output
- `#| eval: false` — Show the code but don't run it
- `#| warning: false` — Suppress warning messages
- `#| message: false` — Suppress messages (like package loading notifications)
- `#| fig-width: 8` and `#| fig-height: 6` — Control plot dimensions in inches

Here's an example showing how to hide code while displaying a plot:

````
```{r}
#| echo: false
#| message: false

library(tidyverse)
students <- read_csv("data/oulad-students-and-assessments.csv")

students |>
  ggplot(aes(x = final_result)) +
  geom_bar() +
  labs(title = "Distribution of Final Results")
```
````

````
This code will render and the output will be displayed, but the code itself *won't be displayed*.

▷Run Cell | Run Next Cell
```{r}
#| echo: false

library(tidyverse)
library(palmerpenguins)

penguins |>
  select(species, island, sex)
```

This code won't run (but it will display).

▷Run Cell | Run Above
```{r}
#| eval: false

penguins |>
  ggplot(aes(x = body_mass_g)) +
  geom_histogram()
```
````

Figure 3: Quarto document showing chunk options at top of code block

When rendered, readers see the plot but not the code that created it. This is useful for final reports where you want to emphasize results rather than implementation details.

You can also set options globally for the entire document by adding them to your YAML header. For example, to hide all code by default:

```yaml
---
title: "My Analysis"
format: html
execute:
  echo: false
---
```

Individual chunks can override global settings. For more options, see the Quarto documentation on execution options.

Console vs. Source: Exploration vs. Preservation

A common pattern in data science work:

- **Console**: Quick exploration, testing ideas, checking output. Code here disappears when you close Positron.
- **Source** (scripts/Quarto): Code you want to keep, reproduce, or share. This is your permanent record.

Here's what this looks like in practice. Try this in the Console for a quick calculation:

```
# In Console - just exploring
mean(c(85, 92, 78, 95, 88))
[1] 87.6
```

That's useful for a quick check, but if you want to preserve this process for later:

```
# In a script - keeping a record
test_scores <- c(85, 92, 78, 95, 88)
average_score <- mean(test_scores)
print(average_score)
```

The second approach creates a record you can return to, modify, and share. If you later need to change the scores or calculate something else, the script preserves your workflow.

Early on, you might type everything in the Console. That's fine for learning! But as soon as you do something you want to remember, put it in a script or Quarto file.

When You Get Stuck

Reading Error Messages

When something goes wrong, R prints an error message in the Console (usually in red). These messages often look cryptic at first, but they're trying to help:

```
Error in mean(x) : object 'x' not found
```

This tells you exactly what went wrong: R couldn't find an object called `x`. Either you misspelled it, or you haven't created it yet.

Getting Help on Functions

R has built-in documentation for every function:

```
?mean           # Opens help for the mean() function
??regression    # Searches all help files for "regression"
```

Help files include descriptions, arguments, examples, and related functions.

Where Computational Social Scientists Get Help

When you're truly stuck, the R community is helpful:

- RStudio Community: Welcoming forum for all R questions
- Stack Overflow: Searchable archive of millions of questions

Before asking a question, try searching for your error message. Someone has likely encountered it before. Of course, turning to AI can also help, as we describe next.

Large Language Models and AI Tools for Getting Unstuck

In Section 3, we cover using large language models as part of a research workflow. But LLMs are also useful for getting unstuck when you're learning and this is increasingly how working data scientists operate.

Two approaches work well:

1. **Ask LLMs directly**: ChatGPT, Claude, or your preferred model can explain error messages, suggest debugging approaches, and clarify confusing documentation. Copy your error message, describe what you were trying to do, and ask for help.
2. **Use AI tools directly in Positron**: AI coding assistants can work inside your editor to suggest code completions, explain what existing code does, and generate boilerplate. We'll cover how to set these up in the next section.

The best programmers don't memorize every function—they know how to find answers quickly. LLMs have become part of that toolkit, alongside documentation, Stack Overflow, and experimentation. Use them when you're stuck, but also pay attention to *why* solutions work. That's how you build intuition.

Using AI Tools Directly in Positron

Positron, being built on VS Code, supports AI assistant extensions that can help you write code more efficiently and understand errors more quickly.

Positron Assistant

Positron includes a built-in AI chat panel called Positron Assistant. You can open it from the sidebar (look for the chat icon) or with the keyboard shortcut Ctrl/Cmd + Shift + I. Positron Assistant is context-aware—it can see your open files, your console history, and the variables in your environment, so you can ask questions about your code without copying and pasting everything into an external tool.

For example, if you've just loaded a dataset and aren't sure how to reshape it, you can highlight the relevant code and ask the assistant to explain it, suggest next steps, or help you fix an error. You can also ask it to generate code from a plain-language description, like "make a bar chart of pass rates by gender." The assistant will write code that fits the context of your current session, using the packages you've already loaded and the data you're working with.

Positron Assistant connects to a language model provider that you configure in your settings—options include Anthropic (Claude), OpenAI, and others. Some providers require a paid API key; check Positron's documentation at https://positron.posit.co/ for current setup instructions. Because the assistant runs inside Positron rather than in a separate browser tab, it reduces the friction of switching between your code and a help tool, which makes it especially useful when you're learning.

When to Use Inline AI vs. External LLMs

Different tools excel at different tasks:

- **Inline AI (Copilot)**: Quick code completions, writing repetitive code, remembering function syntax, generating common patterns
- **External LLMs (ChatGPT/Claude)**: Complex explanations, understanding concepts, debugging logic errors, learning new approaches

Many programmers use both: Copilot for day-to-day coding, external LLMs when they're truly stuck or learning something new.

Important Caution

AI suggestions aren't always correct. Always review code before accepting it, especially for data analysis where subtle errors can lead to wrong conclusions. Use AI tools to speed up your work, but verify that the code does what you think it does.

Please see Section 1.4 for more information about the relationship between you, data, and LLMs.

Tidyverse

The tidyverse is a collection of R packages designed for data science with a shared philosophy: code should be readable, consistent, and focused on the data transformations you're actually trying to accomplish. If you've struggled with base R's sometimes cryptic syntax, the tidyverse will feel like a relief.

Most computational social science workflows rely on tidyverse packages for data manipulation, visualization, and modeling. Learning this ecosystem means learning the tools your collaborators use, the examples you'll find online, and the approaches that scale from quick exploration to publication-ready analysis.

What's in the Tidyverse?

The tidyverse is actually a meta-package that loads eight core packages when you run `library(tidyverse)`:

- **ggplot2**: Data visualization with a grammar of graphics
- **dplyr**: Data manipulation (filtering, selecting, summarizing, joining)
- **tidyr**: Reshaping data between wide and long formats
- **readr**: Reading rectangular data (CSV, TSV) efficiently
- **purrr**: Functional programming tools for iteration
- **tibble**: Modern reimagining of data frames
- **stringr**: String manipulation
- **forcats**: Working with categorical variables (factors)

Most of the work we do in this chapter relies heavily on `dplyr` and `ggplot2`, with the others playing supporting roles as needed.

Installing and Loading the Tidyverse

You install packages like any others in R.

Installation (one time):

```
install.packages("tidyverse")
```

Loading (at the start of each session or script):

```
library(tidyverse)
```

When you load the tidyverse, you'll see a message listing which packages were attached and any conflicts (functions from tidyverse packages that mask base R functions). The conflicts are normal and intentional—tidyverse functions are generally preferable for data science work.

Working with Real Data

Let's load actual data to see the tidyverse in action. We'll use student assessment data from the Open University Learning Analytics Dataset (OULAD), which contains information about student demographics and course outcomes.

```
# Read the data
students <- read_csv("data/oulad-students-and-assessments.csv")

# Take a quick look
glimpse(students)
```

```
Rows: 32,593
Columns: 17
$ code_module                <chr> "AAA", "AAA", "AAA", "AAA", "AAA", "AAA", "~
$ code_presentation          <chr> "2013J", "2013J", "2013J", "2013J", "2013J"~
$ id_student                 <dbl> 11391, 28400, 30268, 31604, 32885, 38053, 4~
$ gender                     <chr> "M", "F", "F", "F", "F", "M", "M", "F", "F"~
$ region                     <chr> "East Anglian Region", "Scotland", "North W~
$ highest_education          <chr> "HE Qualification", "HE Qualification", "A ~
$ imd_band                   <dbl> 10, 3, 4, 6, 6, 9, 4, 10, 8, NA, 8, 3, 7, 6~
$ age_band                   <chr> "55<=", "35-55", "35-55", "35-55", "0-35", ~
$ num_of_prev_attempts       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ studied_credits            <dbl> 240, 60, 60, 60, 60, 60, 60, 120, 90, 60, 6~
$ disability                 <chr> "N", "N", "Y", "N", "N", "N", "N", "N", "N"~
$ final_result               <chr> "Pass", "Pass", "Withdrawn", "Pass", "Pass"~
$ module_presentation_length <dbl> 268, 268, 268, 268, 268, 268, 268, 268, 268~
$ date_registration          <dbl> -159, -53, -92, -52, -176, -110, -67, -29, ~
$ date_unregistration        <dbl> NA, NA, 12, NA, NA, NA, NA, NA, NA, NA, NA,~
```

```
$ pass                    <dbl> 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ mean_weighted_score     <dbl> 780, 700, NA, 720, 690, 790, 700, 720, 720,~
```

The `glimpse()` function shows you the structure of your data: how many rows and columns, what type each column is (character, numeric, etc.), and the first few values. This dataset contains information about students (gender, age, region, disability status) along with their course performance (final_result, pass status, mean_weighted_score).

This is the dataset we'll use in the examples below. It represents the kind of educational data you'll work with in computational social science: thousands of observations with both categorical and continuous variables.

Essential dplyr Functions

The dplyr package provides functions for the most common data manipulation tasks. Here are the ones you'll use repeatedly.

filter() - Selecting Rows

Use `filter()` to keep only rows that meet certain conditions:

```
# Keep only students who passed
passed_students <- students |>
  filter(pass == 1)

# Multiple conditions with AND (comma means AND)
passed_females <- students |>
  filter(pass == 1, gender == "F")

# OR condition using |
good_outcomes <- students |>
  filter(pass == 1 | mean_weighted_score > 800)
```

This gives us 22,333 passed students from the original 32,593. Use `==` for equality tests, `>` and `<` for comparisons, and combine conditions with commas (AND) or `|` (OR).

select() - Choosing Columns

Use `select()` to pick which columns to keep or remove:

```r
# Select specific columns
student_basics <- students |>
  select(id_student, gender, age_band, final_result)

# Remove columns with minus sign
student_no_dates <- students |>
  select(-date_registration, -date_unregistration)

# Select by pattern
student_demographics <- students |>
  select(starts_with("id"), gender, region)
```

The `select()` function is useful when you have many columns but only need a few for analysis. Helper functions like `starts_with()`, `ends_with()`, and `contains()` make it easy to select groups of related columns.

mutate() - Creating or Modifying Columns

Use `mutate()` to add new columns or change existing ones:

```r
# Create new column
students <- students |>
  mutate(high_achiever = mean_weighted_score > 800)

# Modify existing column
students <- students |>
  mutate(age_band = factor(age_band))

# Multiple new columns at once
students <- students |>
  mutate(
    score_category = case_when(
      mean_weighted_score > 800 ~ "High",
      mean_weighted_score > 650 ~ "Medium",
      TRUE ~ "Low"
    )
  )
```

The `case_when()` function inside `mutate()` handles conditional logic: if mean_weighted_score is over 800, assign "High"; if over 650, assign "Medium"; otherwise (TRUE), assign "Low".

summarize() - Calculating Summaries

Use `summarize()` to collapse your data into summary statistics:

```
# Overall statistics
students |>
  summarize(
    mean_score = mean(mean_weighted_score, na.rm = TRUE),
    pass_rate = mean(pass, na.rm = TRUE),
    n_students = n()
  )
```

```
# A tibble: 1 x 3
  mean_score pass_rate n_students
       <dbl>     <dbl>      <int>
1       545.     0.379      32593
```

The `na.rm = TRUE` argument tells R to ignore missing values when calculating means. The `n()` function counts the number of rows.

group_by() - Operations by Group

Use `group_by()` before `summarize()` to calculate statistics for each group separately:

```
# Pass rates by gender
students |>
  group_by(gender) |>
  summarize(
    pass_rate = mean(pass, na.rm = TRUE),
    n = n()
  )
```

```
# A tibble: 2 x 3
  gender pass_rate     n
  <chr>      <dbl> <int>
1 F          0.390 14718
2 M          0.371 17875
```

```
# Multiple grouping variables
students |>
  group_by(gender, disability) |>
  summarize(mean_score = mean(mean_weighted_score, na.rm = TRUE))
```

```
# A tibble: 4 x 3
# Groups:   gender [2]
  gender disability mean_score
  <chr>  <chr>           <dbl>
1 F      N                481.
2 F      Y                456.
3 M      N                603.
4 M      Y                597.
```

After `group_by()`, subsequent operations happen separately for each group. This is one of the most powerful patterns in data analysis: split your data into groups, apply a function to each group, then combine the results.

arrange() - Sorting Rows

Use `arrange()` to reorder rows:

```
# Sort by score (ascending by default)
students |>
  arrange(mean_weighted_score)
```

```
# A tibble: 32,593 x 19
   code_module code_presentation id_student gender region      highest_education
   <chr>       <chr>                  <dbl> <chr>  <chr>       <chr>
 1 BBB         2013B                 521081 F      Yorkshire ~ Lower Than A Lev~
 2 BBB         2013B                 554986 F      London Reg~ Lower Than A Lev~
 3 BBB         2013B                2423078 M      London Reg~ A Level or Equiv~
 4 BBB         2013J                 467396 F      Wales       Lower Than A Lev~
 5 BBB         2013J                 559344 M      South Regi~ A Level or Equiv~
 6 BBB         2013J                 577965 F      Yorkshire ~ A Level or Equiv~
 7 BBB         2013J                 581016 F      South East~ Lower Than A Lev~
 8 BBB         2013J                 590867 M      London Reg~ A Level or Equiv~
 9 BBB         2013J                 591986 M      West Midla~ Lower Than A Lev~
10 BBB         2013J                 596988 F      West Midla~ Lower Than A Lev~
```

```
# i 32,583 more rows
# i 13 more variables: imd_band <dbl>, age_band <fct>,
#   num_of_prev_attempts <dbl>, studied_credits <dbl>, disability <chr>,
#   final_result <chr>, module_presentation_length <dbl>,
#   date_registration <dbl>, date_unregistration <dbl>, pass <dbl>,
#   mean_weighted_score <dbl>, high_achiever <lgl>, score_category <chr>
```

```
# Sort descending
students |>
  arrange(desc(mean_weighted_score))
```

```
# A tibble: 32,593 x 19
   code_module code_presentation id_student gender region      highest_education
   <chr>       <chr>                  <dbl> <chr>  <chr>       <chr>
 1 BBB         2013B                2280038 M      Yorkshire ~ Lower Than A Lev~
 2 BBB         2013B                 497088 F      South Regi~ A Level or Equiv~
 3 BBB         2014B                 633570 M      South East~ A Level or Equiv~
 4 BBB         2013J                 595570 F      South East~ A Level or Equiv~
 5 BBB         2014B                1626021 F      London Reg~ Lower Than A Lev~
 6 BBB         2014B                 555994 M      London Reg~ Lower Than A Lev~
 7 BBB         2013J                 595509 M      South Regi~ A Level or Equiv~
 8 BBB         2013B                 558903 F      East Midla~ HE Qualification
 9 BBB         2014B                  25997 F      London Reg~ A Level or Equiv~
10 FFF         2013B                 267602 M      East Midla~ A Level or Equiv~
# i 32,583 more rows
# i 13 more variables: imd_band <dbl>, age_band <fct>,
#   num_of_prev_attempts <dbl>, studied_credits <dbl>, disability <chr>,
#   final_result <chr>, module_presentation_length <dbl>,
#   date_registration <dbl>, date_unregistration <dbl>, pass <dbl>,
#   mean_weighted_score <dbl>, high_achiever <lgl>, score_category <chr>
```

```
# Multiple sort keys
students |>
  arrange(gender, desc(mean_weighted_score))
```

```
# A tibble: 32,593 x 19
   code_module code_presentation id_student gender region      highest_education
   <chr>       <chr>                  <dbl> <chr>  <chr>       <chr>
 1 BBB         2013B                 497088 F      South Regi~ A Level or Equiv~
```

```
 2 BBB         2013J                595570 F      South East~ A Level or Equiv~
 3 BBB         2014B               1626021 F      London Reg~ Lower Than A Lev~
 4 BBB         2013B                558903 F      East Midla~ HE Qualification
 5 BBB         2014B                 25997 F      London Reg~ A Level or Equiv~
 6 FFF         2013B               2387054 F      Yorkshire ~ Lower Than A Lev~
 7 FFF         2014J                397671 F      East Angli~ Lower Than A Lev~
 8 FFF         2013J               1948159 F      North Regi~ A Level or Equiv~
 9 FFF         2014B                506038 F      West Midla~ Lower Than A Lev~
10 FFF         2014J               1837138 F      East Midla~ A Level or Equiv~
# i 32,583 more rows
# i 13 more variables: imd_band <dbl>, age_band <fct>,
#   num_of_prev_attempts <dbl>, studied_credits <dbl>, disability <chr>,
#   final_result <chr>, module_presentation_length <dbl>,
#   date_registration <dbl>, date_unregistration <dbl>, pass <dbl>,
#   mean_weighted_score <dbl>, high_achiever <lgl>, score_category <chr>
```

This is useful when you want to see the highest or lowest values, or when you need data in a particular order for plotting or reporting.

count() - Quick Frequency Tables

Use `count()` as a shortcut for `group_by()` + `summarize(n = n())`:

```r
# Count final results
students |>
  count(final_result)
```

```
# A tibble: 4 x 2
  final_result     n
  <chr>        <int>
1 Distinction   3024
2 Fail          7052
3 Pass         12361
4 Withdrawn    10156
```

```r
# Count with percentages
students |>
  count(final_result) |>
  mutate(percent = n / sum(n) * 100)
```

```
# A tibble: 4 x 3
  final_result     n percent
  <chr>        <int>   <dbl>
1 Distinction   3024    9.28
2 Fail          7052   21.6
3 Pass         12361   37.9
4 Withdrawn    10156   31.2
```

The `count()` function is perfect for quickly understanding the distribution of categorical variables.

The Pipe Operator: |> and %>%

The tidyverse introduced a distinctive style: piping operations together to create readable data transformation pipelines. Instead of nesting functions inside each other, you "pipe" the output of one function into the next.

R now has a native pipe operator `|>` (introduced in R 4.1), while the tidyverse originally used `%>%` from the magrittr package. They work almost identically for most purposes.

Here's a concrete example using the dplyr functions we just learned:

```
# Without pipes (nested, hard to read)
summarize(
  group_by(
    filter(students, pass == 1),
    gender
  ),
  mean_score = mean(mean_weighted_score, na.rm = TRUE)
)
```

```
# With pipes (clear, step-by-step)
students |>
  filter(pass == 1) |>
  group_by(gender) |>
  summarize(mean_score = mean(mean_weighted_score, na.rm = TRUE))
```

```
# A tibble: 2 x 2
  gender mean_score
  <chr>       <dbl>
```

```
1 F            499.
2 M            638.
```

The piped version reads like instructions: "Take students, *then* keep only those who passed, *then* group by gender, *then* calculate mean scores." Each step is on its own line, making it easy to follow the logic.

We'll use the native pipe `|>` throughout this book because it's now built into R, but you'll encounter `%>%` in older code and documentation. For practical purposes, they're interchangeable in most tidyverse workflows.

How to read it: Think of `|>` as "then." The pipe takes the result from the left and passes it as the first argument to the function on the right.

Data Visualization with ggplot2

The ggplot2 package uses a "grammar of graphics" approach where you build plots in layers. You start with your data, specify how variables map to visual properties (x-axis, y-axis, colors), then add geometric objects like points, bars, or lines.

Basic Structure

Every ggplot follows this template:

```
ggplot(data = DATA, aes(x = X_VAR, y = Y_VAR)) +
  geom_FUNCTION()
```

The `aes()` function (short for "aesthetics") maps your data columns to visual properties. The `geom_` functions specify what kind of plot to draw.

Bar Charts - Categorical Data

Use `geom_bar()` to visualize counts of categorical variables:

```
# Simple bar chart
students |>
  ggplot(aes(x = final_result)) +
  geom_bar()
```

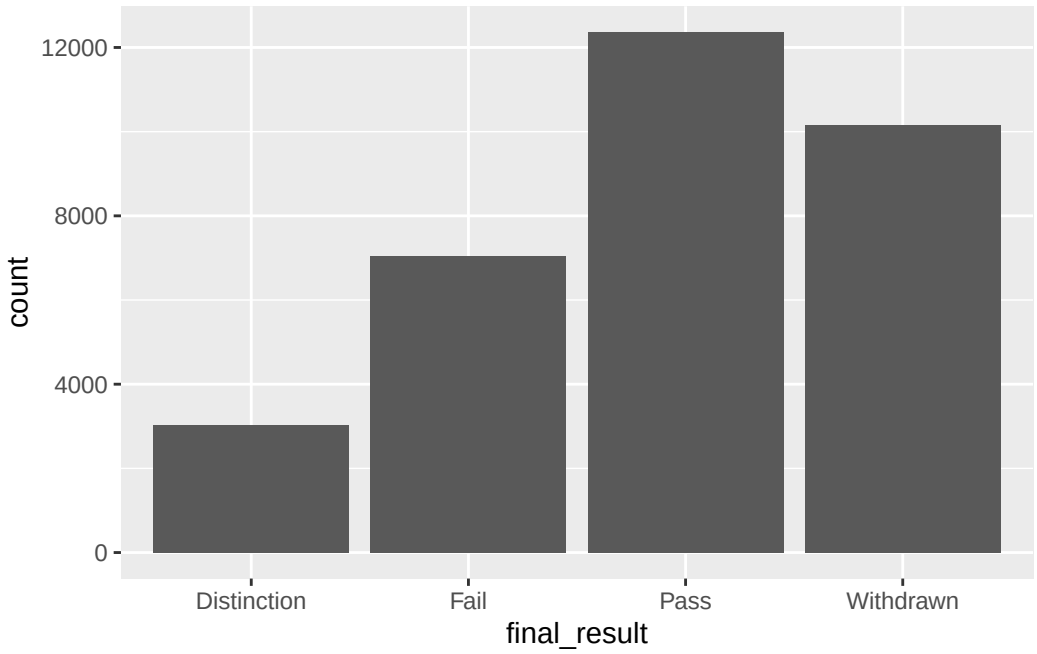


Figure 1: Distribution of student final results

This automatically counts how many students have each final result (Pass, Fail, Distinction, Withdrawn) and displays the counts as bars.

```
# Stacked bar chart by group
students |>
  ggplot(aes(x = gender, fill = final_result)) +
  geom_bar()
```

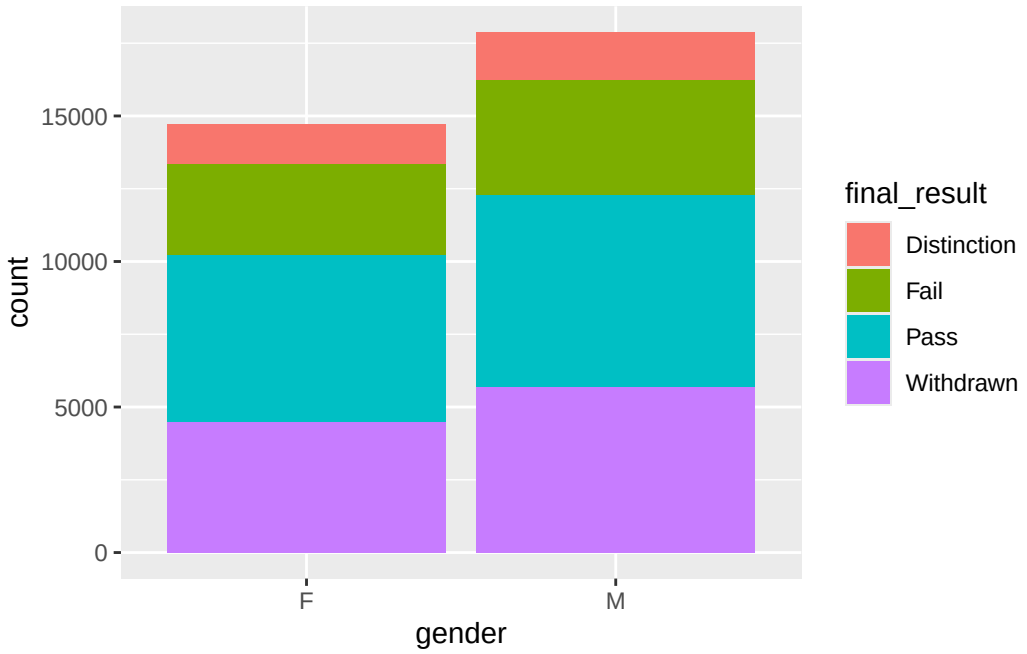


Figure 2: Stacked bar chart showing final results by gender

The `fill` aesthetic colors the bars by final_result, creating stacked bars that show how outcomes differ between genders.

```
# Side-by-side bars
students |>
  ggplot(aes(x = gender, fill = final_result)) +
  geom_bar(position = "dodge")
```

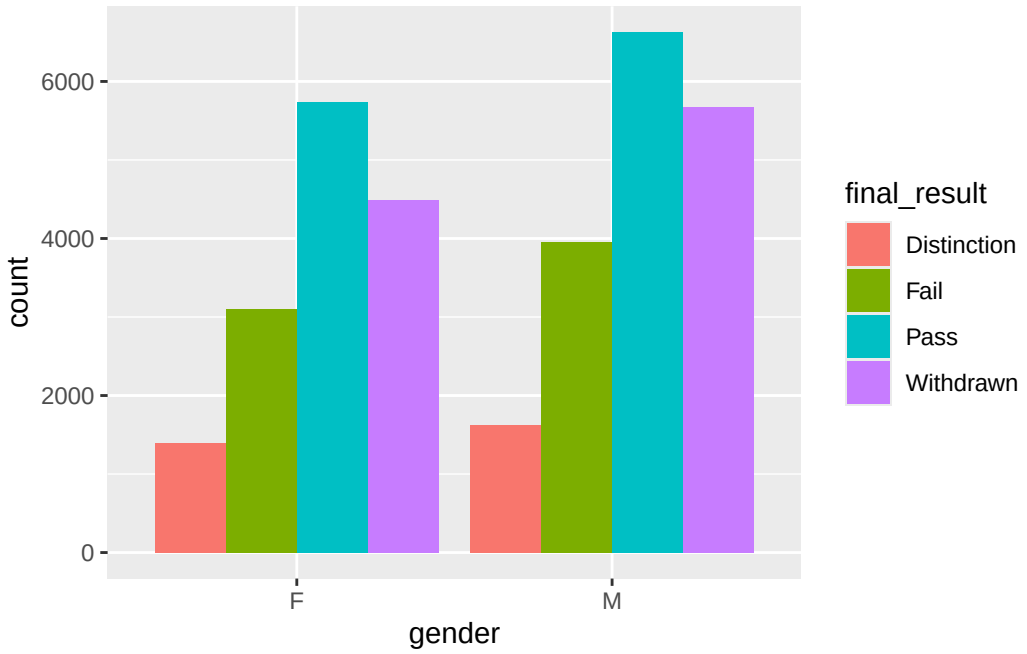


Figure 3: Side-by-side bar chart showing final results by gender

Setting `position = "dodge"` puts bars next to each other instead of stacking them, making it easier to compare counts directly.

Histograms - Distributions

Use `geom_histogram()` to see the distribution of continuous variables:

```
# Distribution of scores
students |>
  ggplot(aes(x = mean_weighted_score)) +
  geom_histogram(binwidth = 50)
```

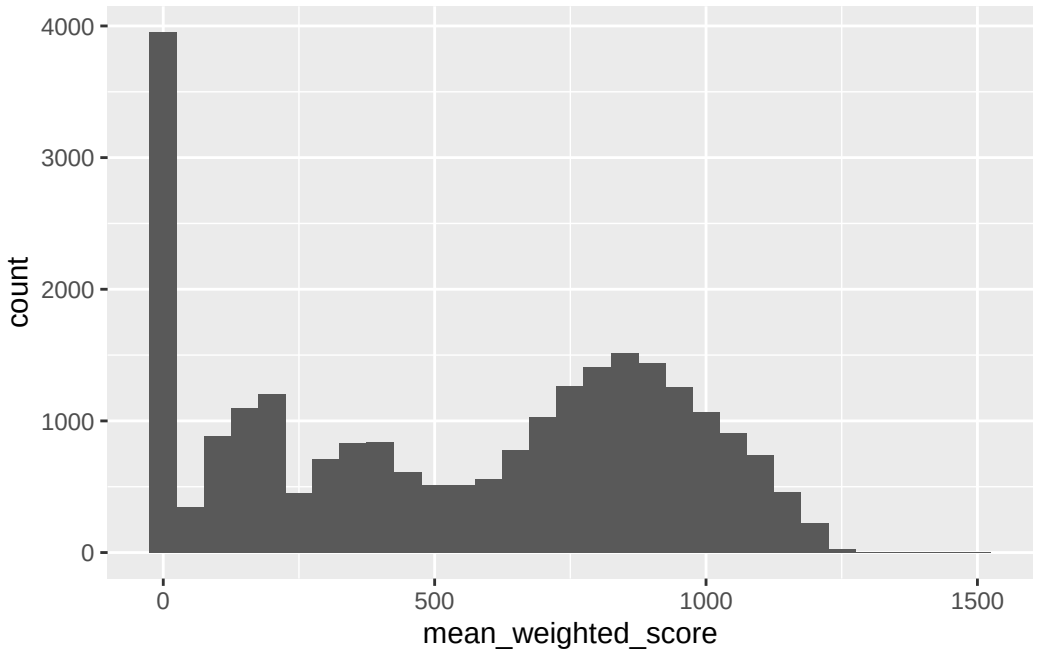


Figure 4: Distribution of mean weighted scores

The `binwidth` argument controls how wide each bar is. Smaller values show more detail, larger values show broader patterns.

```
# Separate distributions by group
students |>
  filter(!is.na(mean_weighted_score)) |>
  ggplot(aes(x = mean_weighted_score, fill = factor(pass))) +
  geom_histogram(binwidth = 50, position = "identity", alpha = 0.5)
```

Figure 5: Overlaid histograms of scores by pass status

Using `position = "identity"` overlays the histograms, and `alpha = 0.5` makes them semi-transparent so you can see both distributions.

Boxplots - Comparing Distributions

Use `geom_boxplot()` to compare distributions across groups:

```
# Scores by final result
students |>
  filter(!is.na(mean_weighted_score)) |>
  ggplot(aes(x = final_result, y = mean_weighted_score)) +
  geom_boxplot()
```

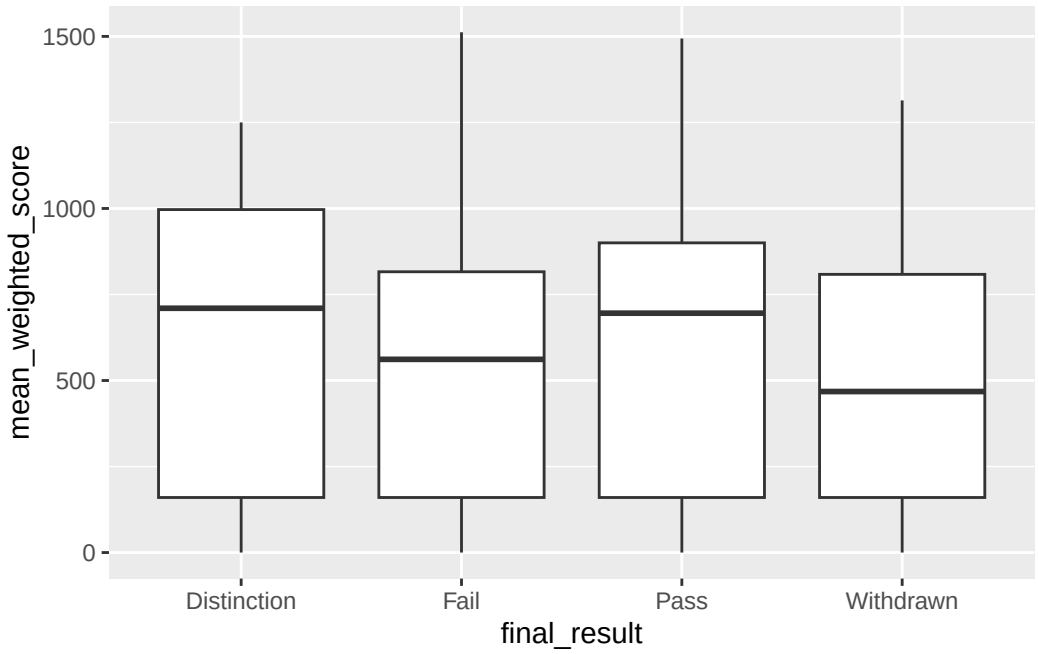


Figure 6: Score distributions by final result

Boxplots show the median (middle line), quartiles (box edges), and outliers (individual points). This makes it easy to see that Distinction students have higher median scores than Pass students, who score higher than those who Fail or Withdraw.

```
# With colors
students |>
  filter(!is.na(mean_weighted_score)) |>
  ggplot(aes(x = final_result, y = mean_weighted_score, fill =
  final_result)) +
  geom_boxplot()
```

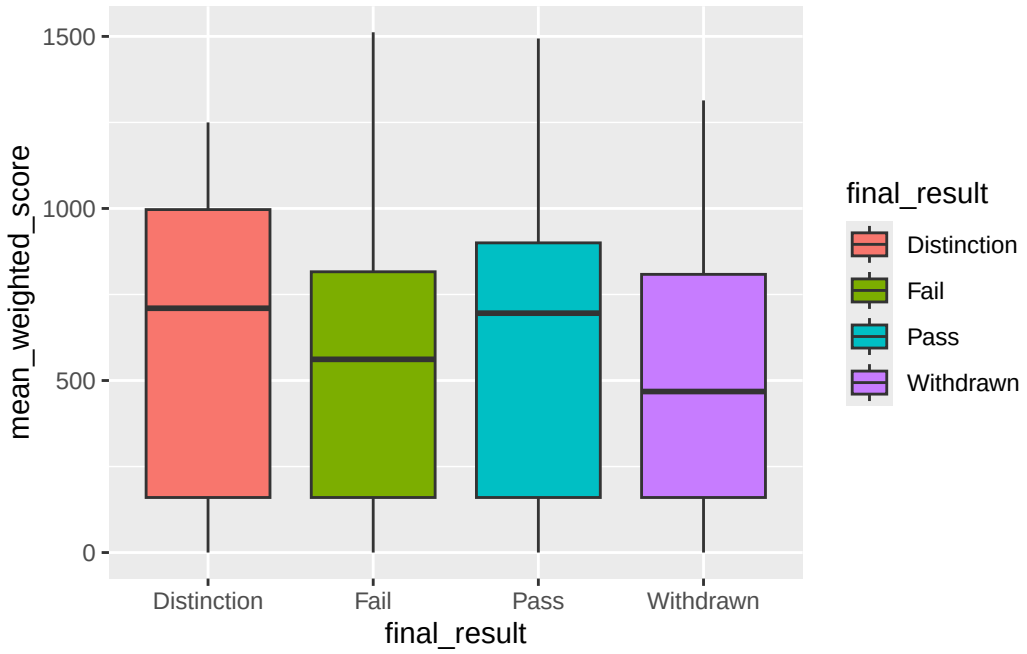


Figure 7: Score distributions by final result with color

Adding `fill` colors each boxplot by the category, making them easier to distinguish.

Scatter Plots - Relationships

Use `geom_point()` to explore relationships between two continuous variables:

```
# Registration timing vs. scores
students |>
  filter(!is.na(mean_weighted_score)) |>
  ggplot(aes(x = date_registration, y = mean_weighted_score)) +
  geom_point(alpha = 0.3)
```

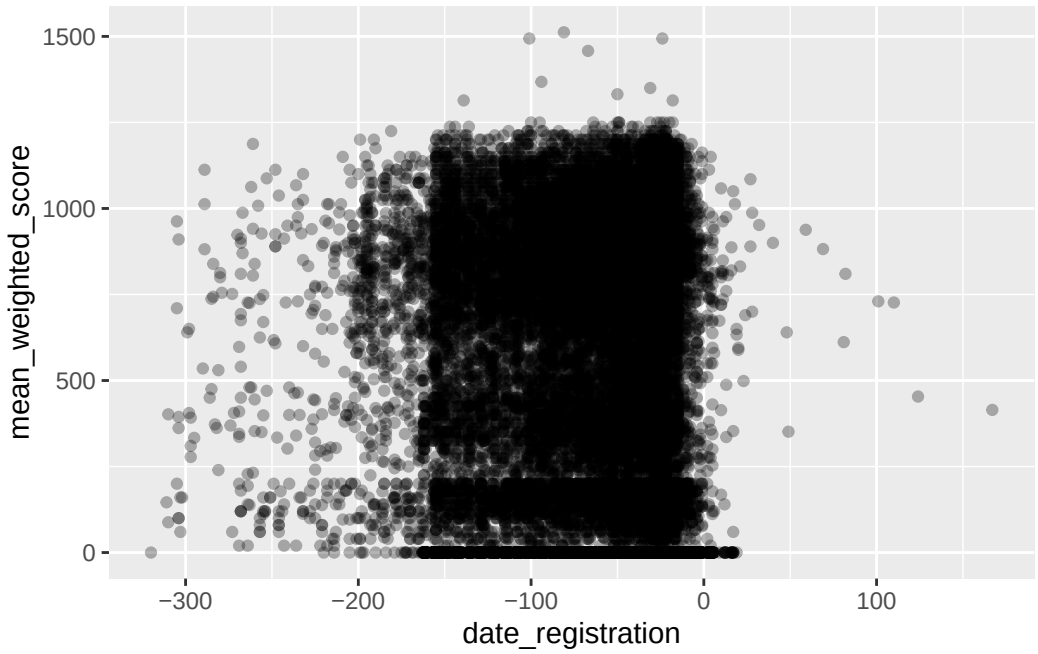


Figure 8: Registration timing versus mean weighted scores

The `alpha` parameter makes points semi-transparent, which helps when many points overlap. This plot would show whether students who register earlier tend to score higher or lower.

```
# Add a trend line
students |>
  filter(!is.na(mean_weighted_score)) |>
  ggplot(aes(x = date_registration, y = mean_weighted_score)) +
  geom_point(alpha = 0.3) +
  geom_smooth(method = "lm")
```

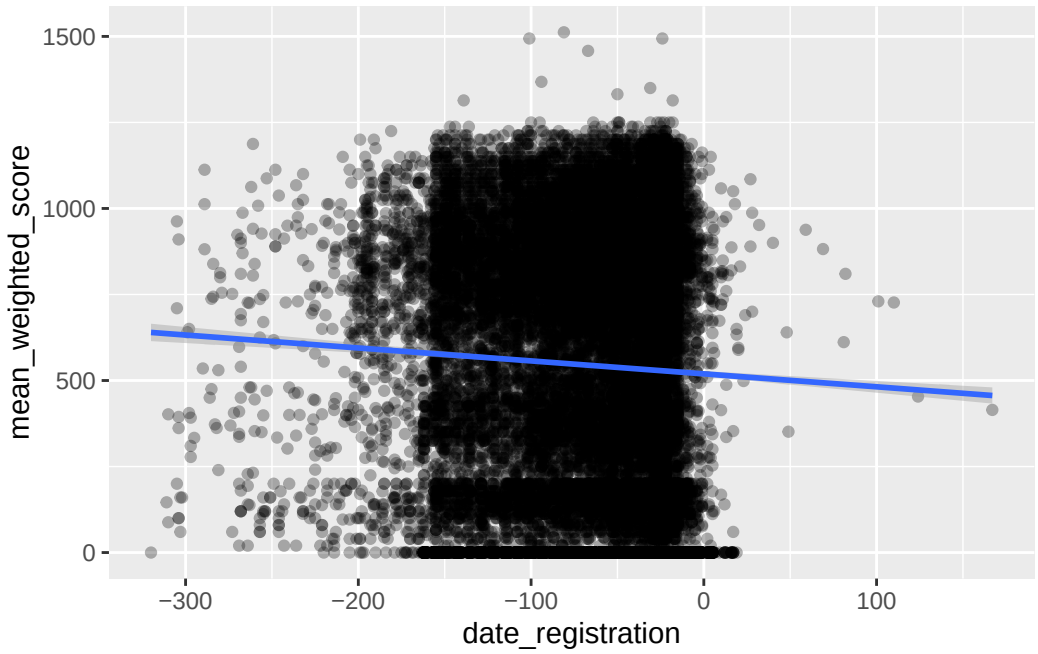


Figure 9: Registration timing versus scores with trend line

Adding `geom_smooth(method = "lm")` fits a linear model and displays the trend line with a confidence band, making patterns easier to see.

Improving Plots with Labels and Themes

Good plots need clear labels:

```
students |>
  ggplot(aes(x = final_result)) +
  geom_bar(fill = "steelblue") +
  labs(
    title = "Distribution of Student Outcomes",
    x = "Final Result",
    y = "Number of Students"
  ) +
  theme_minimal()
```

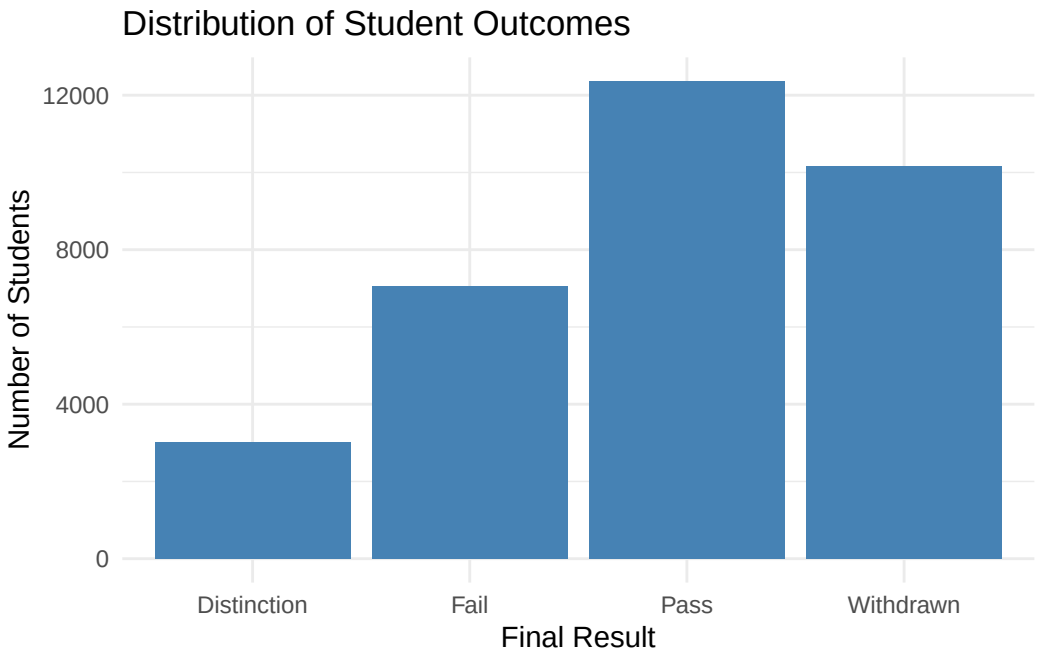


Figure 10: Distribution of student outcomes

The `labs()` function adds titles and axis labels. The `theme_minimal()` function applies a clean, simple theme. Other themes include `theme_bw()`, `theme_classic()`, and `theme_light()`.

Faceting - Small Multiples

Use faceting to create separate plots for different groups:

```
# Separate plot for each course module
students |>
  ggplot(aes(x = final_result)) +
  geom_bar() +
  facet_wrap(~code_module)
```

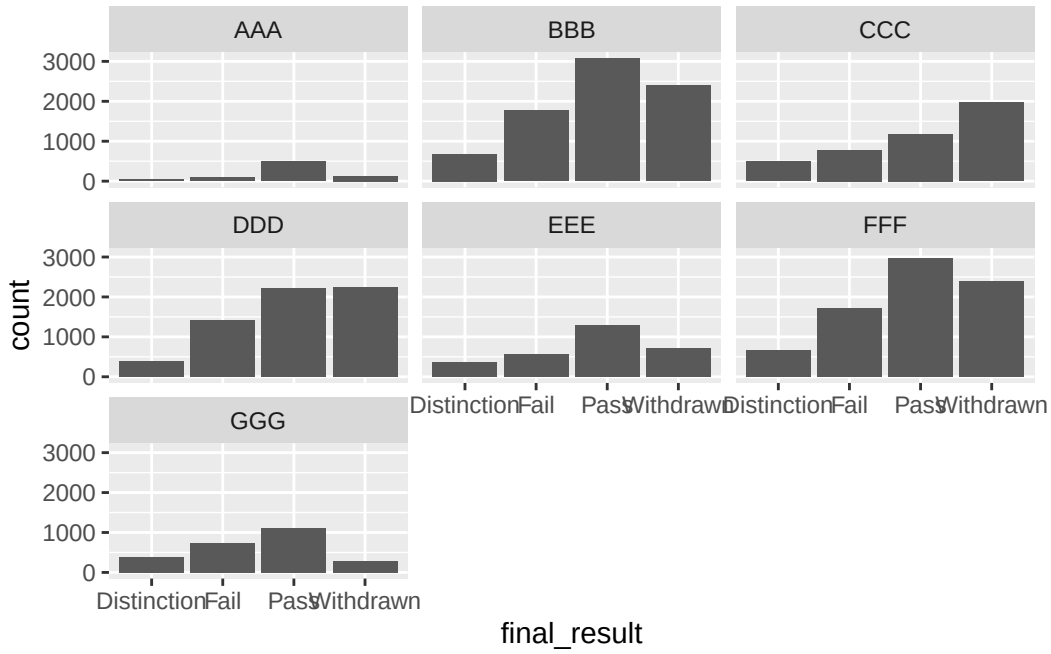


Figure 11: Final results by course module

This creates a grid of small bar charts, one for each module. It's useful when you want to compare patterns across many categories.

```
# Grid by two variables
students |>
  filter(!is.na(mean_weighted_score)) |>
  ggplot(aes(x = mean_weighted_score)) +
  geom_histogram(binwidth = 50) +
  facet_grid(gender ~ disability)
```

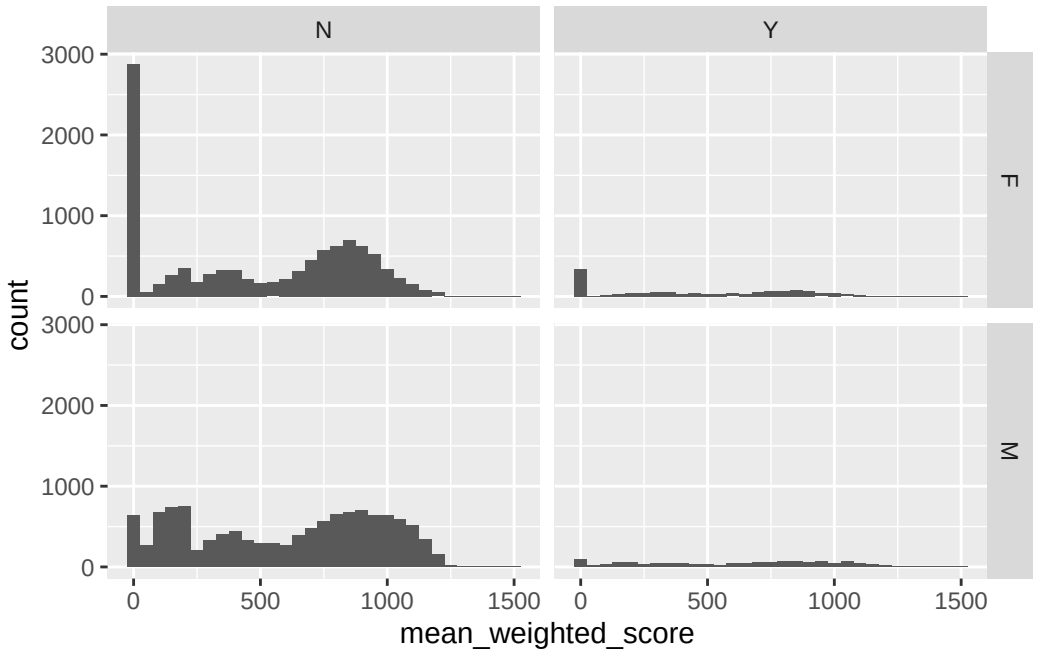


Figure 12: Score distributions by gender and disability status

`facet_grid()` creates a matrix of plots: rows for one variable (gender), columns for another (disability). This lets you see how score distributions vary across combinations of factors.

Next Steps

You've now seen core tidyverse functions in action with real data. The seven dplyr functions covered here—`filter()`, `select()`, `mutate()`, `summarize()`, `group_by()`, `arrange()`, and `count()`—handle the majority of everyday data manipulation tasks. The ggplot2 visualizations—bar charts, histograms, boxplots, scatter plots, and faceting—cover most exploratory analysis needs.

The subsequent chapters in this book use these functions extensively, and we'll introduce additional tidyverse capabilities as they become relevant. You'll continue learning by doing, working with real data and real research questions.

When you want to go deeper:

- *R for Data Science (2e)*: The definitive guide to the tidyverse
- Tidyverse documentation: Reference docs for all packages
- RStudio Cheatsheets: Visual quick references for dplyr, ggplot2, and more

The tidyverse community is large and welcoming. When you get stuck, someone has likely asked your question before.

LLMs and Data Science

The (Responsible) Use of LLMs in Data Science

As we integrate large language models into data analysis, our responsibilities as researchers evolve in important ways . There are three primary approaches to this integration:

- first, using LLMs for code autocompletion and inline coding assistance, such as through integrated development environments like Positron;
- second, using LLMs to support the analysis process through complete code generation and supervised execution, such as with tools like Posit Databot; and
- third, deploying LLMs to directly analyze qualitative (e.g., text data).

The Responsible Use Framework

The responsible use framework for LLMs in research, presented by Joe Cheng at R+AI Conference 2025 (**Cheng2025LLMResponsible?**) can be used as a practical evaluation tool centering on three essential criteria. **Correctness** refers to whether the LLM produces accurate and reliable results that can be verified and trusted. **Transparency** addresses whether the process and reasoning behind the AI's outputs are visible and understandable to researchers, allowing them to inspect how conclusions were reached. **Reproducibility** concerns whether the same analysis can be repeated and yield consistent results, a foundational requirement of scientific research. These three categories align with broader responsible AI governance principles commonly found in organizational frameworks but are specifically tailored to the research context where verifiability and scientific rigor are key. For an LLM application in research to be considered responsibly used, it should ideally achieve "yes" answers across all three criteria, ensuring that the technology enhances rather than compromises research integrity.

Applying the Resposible Use Framework

When we examine the first two usage types (e.g., code autocomplete and code generation) through the responsible usage framework, we find more encouraging alignment with the three essential criteria of correctness, transparency, and reproducibility . Code-generating and code-assisting LLMs produce verifiable output that researchers can inspect, test, and validate before execution, ensuring correctness through human review. The process maintains transparency

because the generated code itself is visible and interpretable, allowing researchers to understand exactly what analytical steps are being performed. Reproducibility is achieved because the same code can be run multiple times on the same data to yield consistent results, and the code can be shared alongside research findings.

In contrast, the third approach where LLMs directly analyze qualitative or text data within a black box that is the LLM may raise critical questions about research integrity. Using LLMs for this purpose present inherent challenges against these principles: they are notorious for generating convincing but incorrect answers, operate as black boxes with limited transparency, and lack reproducibility due to their non-deterministic nature. When we apply this framework to direct text analysis by LLMs, significant concerns may emerge: we cannot guarantee correctness, the process lacks transparency, and reproducibility remains uncertain at best.

Achieving Responsible Usage Through Evidence-Based Results

However, for the third type of usage where LLMs directly analyze data (as we do in Section X), we can work toward achieving "yes" answers across all (or as many as possible) three framework criteria by requiring LLMs to produce evidence-based results. As we cover in Section 6, it is possible to use a Local LLM that is connected to R/Positron. In this approach, the researcher has control over the data analysis over simply pasting text into an LLMs chatbox. For example, this can be done by structuring prompts to demand transparent, verifiable outputs, such as requiring the LLM to identify themes, provide verbatim quotes from the source data, report frequencies with counts and percentages, and present findings in standardized tabular formats. We transform the black box into a more transparent analytical tool. This approach ensures **correctness** by grounding every claim in specific textual evidence that researchers can verify, enhances **transparency** by making the analytical reasoning traceable through quoted examples and quantitative metrics, and improves **reproducibility** by standardizing the output format and maintaining clear documentation of the analytical process. When LLMs are constrained to cite their sources, quantify their observations, and structure their findings systematically, they shift from opaque pattern generators to accountable research assistants whose work can be validated against the original data. This methodology aligns with the recommendation for constrained use and micromanaged implementation, ensuring that LLMs remain within appropriate boundaries while still providing valuable analytical support for qualitative research.

Computational Methods

Overview

This section introduces the core computational methods that form the basis of educational data analysis. Across the next three chapters, you will learn how to analyze **textual**, **relational**, and **numeric** data—three major forms of information that appear in learning environments, institutional records, and educational research projects.

These chapters emphasize hands-on, transparent, and reproducible approaches using R. By engaging with real examples, you will gain practical experience in transforming raw educational data into interpretable results that inform theory and practice.

The Analytical Scope of This Section

Chapter	Data Type	Analytical Focus	Example Research Question
Chapter 2	Text Data (unstructured)	Natural language processing, tokenization, sentiment, topic modeling	"What themes emerge in student reflections or policy statements?"
Chapter 3	Relational Data	Social network analysis: centrality, community detection, visualization	"How do students or instructors connect and collaborate in learning networks?"
Chapter 4	Numeric / Big Data	Statistical modeling, regression, clustering, predictive analysis	"Which factors best predict academic outcomes or engagement?"

These three perspectives together illustrate how computational techniques can capture different dimensions of learning—language, interaction, and measurement.

Setting Up the Computational Environment

Before exploring the examples, ensure that your R environment contains the essential packages used throughout this section.

```
install.packages(c(
  # Core workflow and visualization
  "tidyverse", "ggplot2", "readr", "stringr",
```

```
  # Text analysis
  "tidytext", "quanteda", "textdata",

  # Network analysis
  "igraph", "ggraph", "tidygraph",

  # Numeric and machine learning tools
  "caret", "cluster"
))
```

Optional visualization and interaction packages:

```
install.packages(c("plotly", "RColorBrewer", "visNetwork"))
```

> Tip: Use a consistent project structure so each chapter builds on the same foundation:
>
> ```
> project/
>   data/        # datasets
>   scripts/     # reusable code
>   outputs/     # tables and processed files
>   figures/     # charts and visualizations
> ```
>
> This structure promotes reproducibility and helps keep analysis pipelines organized.

A General Computational Workflow

Regardless of data type, the analytical logic follows a similar cycle:

1. **Load data** — read files from local or online sources.
2. **Clean data** — handle missing values, normalize text or numeric fields.
3. **Transform data** — create tokens, build networks, or scale variables.
4. **Analyze** — apply the method appropriate to the data form.
5. **Visualize and interpret** — generate plots and summaries to support interpretation.

```
library(tidyverse)

data <- read_csv("data/example.csv")

cleaned <- data |>
```

```
  mutate(across(everything(), str_squish)) |>
  drop_na()

summary(cleaned)
```

This five-step workflow—load, clean, transform, analyze, interpret—appears throughout all chapters in this section.

Ethics, Transparency, and Reproducibility

Educational data often include sensitive or identifiable information.

Responsible computational research requires attention to both ethical and methodological rigor.

- **Privacy:** Remove or anonymize all personal identifiers.
- **Transparency:** Keep analysis scripts in Quarto or R Markdown files for version control.
- **Reproducibility:** Record package versions and parameters used in each analysis.
- **Interpretability:** Combine quantitative patterns with contextual educational insight.

Ethical and transparent practices ensure that computational results remain credible, interpretable, and usable for improving learning.

Transition to Analytical Chapters

With the environment prepared and workflow established, you are ready to begin applying computational methods to real educational data:

- **Chapter 2 — Text Analysis:** Working with unstructured language to uncover patterns in meaning and discourse.
- **Chapter 3 — Network Analysis:** Examining connections and relationships among learners, instructors, or resources.
- **Chapter 4 — Numeric and Big Data:** Exploring structured data to identify trends and predictors in education.

The following chapter begins with text data, illustrating how natural language processing can transform qualitative information into structured, interpretable results.

Text Data

2.1 Overview

In social sciences, analyzing text data is usually considered the "job" of qualitative researchers. Traditionally, qualitative research involves identifying patterns in non-numeric data, and this pattern recognition is typically done manually. This process is time-intensive but can yield rich research results. Traditional methods for analyzing text data involve human coding and can include direct (e.g., books, online texts) or indirect sources (e.g., interview transcripts).

With the advent of new software, we can capture and analyze text data in ways that were previously not possible, such as web scraping, accessing social media APIs, or downloading large online documents. Given the increased (opportunities for collecting and) size of text data, analysis now can benefit from computational approaches (e.g., dictionary-based, frequency-based, machine learning) that go beyond manual coding. As we discussed on the front matter of the book, these computational methods allow social scientists to ask new types of research questions, expanding the scope and depth of possible insights.

> **Disclaimer**: While resources are available that discuss these analysis methods in depth, this book aims to provide a practical guide for social scientists, using data they will likely encounter. Our goal is to present a "cookbook" for guiding research projects through real-world examples.

2.2 Accessing Text Data

Nowadays, text can be found and collected in many different ways. For example, social media can serve as rich with text data (e.g., Reddit posts), likewise text created in classrooms, especially online (e.g., every student writing) can become part of text data. In this section, we will cover a few basic ways of accessing text data. Please note that although we cover some prominent ways, this is by no means an exhaustive list. Therefore, please refer to the additional resources section at the end of the section to dive deeper.

2.2.1 Web Scraping (Unstructured or API)

What is Web Scraping?

Web scraping refers to the automated process of extracting data from web pages. It is particularly useful when dealing with extensive lists of websites that would be tedious to mine manually. A typical web scraping program follows these steps:

1. Loads a webpage.
2. Downloads the HTML or XML structure.
3. Identifies the desired data.
4. Converts the data into a format suitable for analysis, such as a data frame.

In addition to text, web scraping can also be used to download other content types, such as audio-visual files.

Is Web Scraping Legal?

Web scraping was common in the early days of the internet, but with the increasing value of data, legal norms have evolved. To avoid legal issues, check the "Terms of Service" for specific permissions on the website, often accessible via "robots.txt" files. Consult legal advice when in doubt.

Reading a Web Page into R

Once permissions are confirmed, the first step in web scraping is to download the webpage's source code into R, typically using the `rvest` package by Hadley Wickham.

```
# Install and load the rvest package
install.packages("rvest")
library(rvest)
```

To demonstrate, we will scrape a simple Wikipedia page. Static pages, which lack interactive elements like JavaScript, are simpler to scrape. You can view the page's HTML source in your browser by selecting **Developer Tools > View Source**.

```
# Load the webpage
wikipedia_page <- read_html(
"https://en.wikipedia.org/wiki/World_Health_Organization_ranking_of_health_systems_in_2000
)
```

```r
# Verify that the webpage loaded successfully
wikipedia_page
```

Parsing HTML

The next challenge is extracting specific information from the HTML structure. HTML files have a "tree-like" format, allowing us to target particular sections. Use your browser's "Developer Tools" to inspect elements and locate the data. Right-click the desired element and select **Inspect** to view its structure.

To isolate data sections within the HTML structure, identify the **XPath** or **CSS selectors**. For instance:

```r
# Extract specific section using XPath
section_of_wikipedia <- html_node(wikipedia_page,
xpath='//*[@id="mw-content-text"]/div/table')
head(section_of_wikipedia)
```

To convert the extracted section into a data frame, use `html_table()`:

```r
# Convert the extracted data into a table
health_rankings <- html_table(section_of_wikipedia)
head(health_rankings[ , (1:2)])  # Display the first two columns
```

Parsing with CSS Selectors

For more complex web pages, CSS selectors can be an alternative to XPath. Tools like **Selector Gadget** can help identify the required CSS selectors.

For example, to scrape event information from Duke University's main page:

```r
# Load the webpage
duke_page <- read_html("https://www.duke.edu")

# Extract event information using CSS selector
duke_events <- html_nodes(duke_page, css="li:nth-child(1) .epsilon")
html_text(duke_events)
```

Scraping with Selenium

For tasks involving interactive actions (e.g., filling search fields), use `RSelenium`, which enables automated browser operations.

To set up Selenium, install the **Java SE Development Kit** and **Docker**. Then, start Selenium in R:

```
# Install and load RSelenium
install.packages("RSelenium")
library(RSelenium)

# Start a Selenium session
rD <- rsDriver()
remDr <- rD$client
remDr$navigate("https://www.duke.edu")
```

To automate data entry, identify the CSS selector for the search box and input the query:

```
# Find the search box element and enter a query
search_box <- remDr$findElement(using = 'css selector', 'fieldset
input')
search_box$sendKeysToElement(list("data science", "\uE007"))  # "\uE007"
represents Enter key
```

Web Scraping within a Loop

To scrape multiple pages, embed the code within a loop to automate tasks across different URLs. Since each site may have a unique structure, generalized scraping can be time-intensive and error-prone. Implement error handling to manage interruptions.

When to Use Web Scraping

Web scraping is appropriate if:

- **Page structure is consistent across sites**: For example, a government site with date suffixes but a uniform layout.
- **Manual data collection is prohibitive**: For extensive text or embedded tables.

When feasible, consider alternatives like APIs or data-entry services (e.g., Amazon Mechanical Turk) for better efficiency and legal compliance.

What is an API?

An Application Programming Interface (API) is a set of protocols that allows computers to communicate and exchange information. A common type is the REST API, where one machine sends a request, and another returns a response. APIs provide standardized access to data, services, and functionalities, making them essential in software development.

When to Use an API

APIs are commonly used for:

- **Integrating with Third-Party Services**: APIs connect applications to services like payment gateways or social media.
- **Accessing Data**: APIs retrieve data from systems or databases (e.g., real-time weather data).
- **Automating Tasks**: APIs automate processes within applications, such as email marketing.
- **Building New Applications**: APIs allow developers to build new apps or services (e.g., a mapping API for navigation).
- **Streamlining Workflows**: APIs enable seamless communication and data exchange across systems.

Using Reddit API with RedditExtractoR

Reddit is a social media platform featuring a complex network of users and discussions, organized into "subreddits" by topic. RedditExtractoR, an R package, enables data extraction from Reddit to identify trends and analyze interactions.

```
# Install and load RedditExtractoR
install.packages("RedditExtractoR")
library(RedditExtractoR)

# Access data from the GenAI subreddit
GenAI_reddit <- find_thread_urls(subreddit = "GenAI", sort_by = "new",
period = "day")
view(GenAI_reddit)
```

2.2.2 Audio Transcripts (Zoom, etc.)

Audio transcripts are a rich source of text data, especially useful for capturing spoken content from meetings, interviews, or webinars. Many platforms, such as Zoom, provide automated transcription services that can be downloaded as text files for analysis. By processing these transcripts, researchers can analyze conversation themes, sentiment, or other linguistic features. Here's how to access and prepare Zoom transcripts for analysis in R.

Key Steps for Extracting Text Data from Audio Transcripts

1. **Access the Zoom Transcript**
   - Log in to your Zoom account.
   - Navigate to the "Recordings" section.
   - Select the recording you wish to analyze and download the "Audio Transcript" file.

2. **Import the Transcript into R**
   Once the file is downloaded, you can load it into R for analysis. Depending on the file format (usually a `.txt` file with tab or comma delimiters), use `read.table()`, `read.csv()`, or functions from the `readr` package to load the data.

```
   # Load the transcript into R
   transcript_data <- read.table("path/to/your/zoom_transcript.txt", sep
= "\t", header = TRUE)
```

Adjust the `sep` parameter based on the delimiter used in the transcript file (typically `\t` for tab-delimited files).

3. **Data Cleaning (if necessary)**
   Clean up the text data to remove unnecessary characters, standardize formatting, and prepare it for further analysis.

   - **Remove Unwanted Characters**
     Use `gsub()` to eliminate special characters and punctuation, keeping only alphanumeric characters and spaces.

```
     # Remove special characters
     transcript_data$text <- gsub("[^a-zA-Z0-9 ]", "",
transcript_data$text)
```

   - **Convert Text to Lowercase**
     Standardize text to lowercase for consistency in text analysis.

```
# Convert text to lowercase
transcript_data$text <- tolower(transcript_data$text)
```

2.2.3 PDF

PDF files are a valuable source of text data, often found in research publications, government documents, and industry reports. We'll explore two main methods for extracting text from PDFs:

1. **Extracting from Local PDF Files**: This method involves accessing and parsing text from PDF files stored locally, providing tools and techniques to efficiently retrieve text data from offline documents.
2. **Downloading and Extracting PDF Files**: This approach covers downloading PDFs from online sources and extracting their text. This method is useful for scraping publicly available documents from websites or databases for research purposes.
3. **PDF Data Extractor (PDE)**
   For more advanced PDF text extraction and processing, you can use the PDF Data Extractor (PDE) package. This package provides tools for extracting text data from complex PDF documents, supporting additional customization options for text extraction. PDE is a R package that easily extracts information and tables from PDF files. The PDE_analyzer_i() performs the sentence and table extraction while the included PDE_reader_i() allows the user-friendly visualization and quick-processing of the obtained results.

Steps for Extracting Text from Local PDF Files

1. **Install and Load the `pdftools` Package**
   Start by installing and loading the `pdftools` package, which is specifically designed for reading and extracting text from PDF files in R.

```
install.packages("pdftools")
library(pdftools)
```

2. **Read the PDF as a Text File**
   Use the `pdf_text()` function to read the PDF file into R as a text object. This function returns each page as a separate string in a character vector.

```
txt <- pdf_text("path/to/your/file.pdf")
```

3. **Extract Text from a Specific Page**
   To access a particular page from the PDF, specify the page number in the text vector. For example, to extract text from page 24:

```
page_text <- txt[24]  # page 24
```

4. **Extract Rows into a List**
   If the page contains a table or structured text, use the `scan()` function to read each row as a separate element in a list. The `textConnection()` function converts the page text for processing.

```
rows <- scan(textConnection(page_text), what = "character", sep =
"\n")
```

5. **Split Rows into Cells**
   To further parse each row, split it into cells by specifying the delimiter, such as whitespace (using `"\\s+"`). This converts each row into a list of individual cells.

```
row <- unlist(strsplit(rows[24], "\\s+"))  # Example with the 24th
row
```

Steps for Downloading and Extracting Text from PDF Files

1. **Download the PDF from the Web**
   Use the `download.file()` function to download the PDF file from a specified URL. Set the mode to `"wb"` (write binary) to ensure the file is saved correctly.

```
link <- paste0(
  "http://www.singstat.gov.sg/docs/",
  "default-source/default-document-library/",
  "publications/publications_and_papers/",
  "cop2010/census_2010_release3/",
  "cop2010sr3.pdf"
)
download.file(link, "census2010_3.pdf", mode = "wb")
```

2. **Read the PDF as a Text File**
   After downloading, read the PDF into R as a text object using the `pdf_text()` function from the `pdftools` package. Each page of the PDF will be stored as a string in a character vector.

```
txt <- pdf_text("census2010_3.pdf")
```

3. **Extract Text from a Specific Page**
   Access the desired page (e.g., page 24) by specifying the page number in the character vector.

```
page_text <- txt[24]  # Page 24
```

4. **Extract Rows into a List**
   Use the `scan()` function to split the page text into rows, with each row representing a line of text in the PDF. This creates a list where each line from the page is an element.

```
rows <- scan(textConnection(page_text), what = "character", sep =
"\n")
```

5. **Loop Through Rows and Extract Data**
   Starting from a specific row (e.g., row 7), loop over each row. For each row:
   - Split the text by spaces (`"\\s+"`) using `strsplit()`.
   - Convert the result to a vector with `unlist()`.
   - If the third cell in the row is not empty, store the second cell as `name` and the third cell as `total`, converting it to a numeric format after removing commas.

```
name <- c()
total <- c()

for (i in 7:length(rows)) {
  row <- unlist(strsplit(rows[i], "\\s+"))
  if (!is.na(row[3])) {
    name <- c(name, row[2])
    total <- c(total, as.numeric(gsub(",", "", row[3])))
  }
}
```

2.2.4 Survey, Discussions, etc.

Surveys and discussion posts are valuable sources of text data in social science research, providing insights into participants' perspectives, opinions, and experiences. These data sources often come from open-ended survey responses, online discussion boards, or educational platforms. Extracting and preparing text data from these sources can reveal recurring themes,

sentiment, and other patterns that support both quantitative and qualitative analysis. Below are key steps and code examples for processing text data from surveys and discussions in R.

Key Steps for Processing Survey and Discussion Text Data

1. **Load the Data**
   Survey and discussion data are typically stored in spreadsheet formats like CSV. Begin by loading this data into R for processing. Here, `readr` is used for reading CSV files with `read_csv()`.

```
# Install and load necessary packages
install.packages("readr")
library(readr)

# Load data
survey_data <- read_csv("path/to/your/survey_data.csv")
```

2. **Extract Text Columns**
   Identify and isolate the relevant text columns. For example, if the text data is in a column named "Response," you can create a new vector for analysis.

```
# Extract text data from the specified column
text_data <- survey_data$Response
```

3. **Data Cleaning**
   Prepare the text data by cleaning it, removing any unnecessary characters, and standardizing the text. This includes removing punctuation, converting text to lowercase, and handling extra whitespace.

   - **Remove Unwanted Characters**
     Use `gsub()` from base R to remove any non-alphanumeric characters, retaining only words and spaces.

```
# Remove special characters
text_data <- gsub("[^a-zA-Z0-9 ]", "", text_data)
```

- **Convert to Lowercase**
  Standardize the text by converting all characters to lowercase.

```
# Convert text to lowercase
text_data <- tolower(text_data)
```

- **Remove Extra Whitespace**
  Remove any extra whitespace that may be left after cleaning.

```
# Remove extra spaces
text_data <- gsub("\\s+", " ", text_data)
```

4. **Tokenization and Word Counting (Optional)**
   If further analysis is needed, such as frequency-based analysis, split the text into individual words (tokenization) or count the occurrence of specific words. Here, `dplyr` is used to organize the word counts.

```
# Install and load necessary packages
install.packages("dplyr")
library(dplyr)

# Tokenize and count words
word_count <- strsplit(text_data, " ") %>%
              unlist() %>%
              table() %>%
              as.data.frame()
```

2.3 Frequency-based Analysis

In the following section, we will provide a "recipe" for the social scientist interested in these new methods of analyzing text data to get you from the initial stages of getting the data to running the analyses and the write up. Often left out is also a research question that suits or requires a method. Since we have a data and method-centric approach here, we will backtrack and also provide a research question, so that you can model after it in your own work. Finally, we will provide a sample results and discussions section.

2.3.1 Purpose

The purpose of the frequency-based approach is to count the number of words as they appear in a text file, whether it is a collection of tweets, documents, or interview transcripts. This approach aligns with the frequency-coding method (e.g., Saldaña ,2021) and can supplement

human coding by revealing the most/least commonly occurring words, which can then be compared across dependent variables.

Case Study: Frequency-Based Analysis of GenAI USage Guidelines in Higher Education

As AI writing tools like ChatGPT become more prevalent, educators are working to understand how best to integrate them within academic settings, while many students and instructors remain uncertain about acceptable use cases. Our research into AI usage guidelines from the top 100 universities in North America aims to identify prominent themes and concerns in institutional policies regarding GenAI.

2.3.2 Sample Research Questions

To investigate the nature of AI use policies within higher education institutions, in this study, our research questions are:

- **RQ1**: What are the most frequently mentioned words in university GenAI writing usage policies?
- **RQ2**: Which keywords reflect common concerns or focal points related to GenAI writing usage in academic settings?

2.3.3 Sample Methods

Data Source

The dataset consists of publicly available AI policy texts from 100 universities(USA), the data has been downloaded and saved as a CSV file for analysis. –we might have to write more here to model how a research should be describing the data from its acquisition to use for research.

Data Analysis

In order to analyze the data we used xyz, —-let's provide a sample write up for the researcher to adapt.

2.3.4 Analysis

Step 1: Load Required Libraries

Install and load libraries for data processing, visualization, and word cloud generation.

```
# Install necessary packages if not already installed
#install.packages(c("tibble", "dplyr", "tidytext", "ggplot2",
"viridis","tm",wordcloud" "wordcloud2","webshot"))

# Load libraries
library(readr)
library(tibble)
library(dplyr)
library(tidytext)
library(ggplot2)
library(viridis)
library(tm)
library(wordcloud)
library(wordcloud2)
library(webshot)
```

Step 2: Load Data

Read the CSV file containing policy texts from the top 100 universities.

```
# Load the dataset
university_policies <- read_csv("University_GenAI_Policy_Stance.csv")
```

Step 3: Tokenize Text and Count Word Frequency

Process the text data by tokenizing words, removing stop words, and counting word occurrences.

```
# Tokenize text, remove stop words, and count word frequencies
word_frequency <- university_policies %>%
  unnest_tokens(word, Stance) %>%  # Tokenize the 'Policy_Text' column
```

```
  anti_join(stop_words) %>%              # Remove common stop words
  count(word, sort = TRUE)               # Count and sort words by
  frequency
word_frequency
```

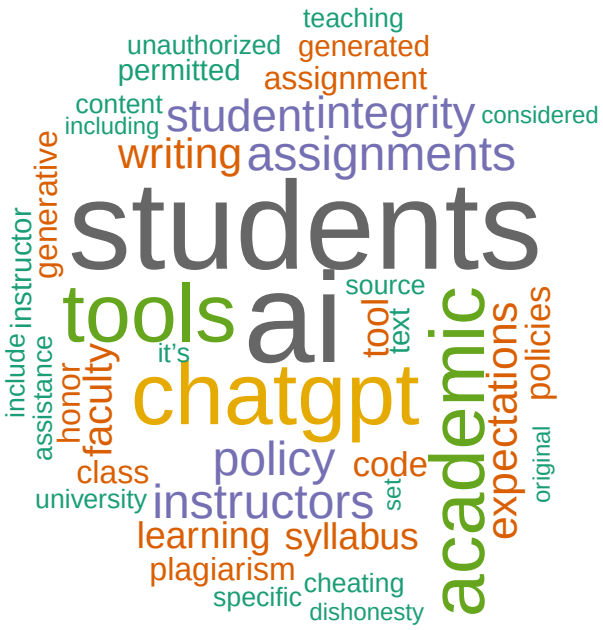
```
# A tibble: 1,108 x 2
   word            n
   <chr>       <int>
 1 ai            124
 2 students      120
 3 chatgpt        80
 4 tools          73
 5 academic       68
 6 policy         35
 7 assignments    34
 8 instructors    34
 9 integrity      33
10 student        32
# i 1,098 more rows
```

Step 4: Create a Word Cloud

Generate a word cloud to visualize the frequency of words in a circular shape.

```
# Create and display the GenAI usage Stance wordcloud

wordcloud(words = word_frequency$word, freq = word_frequency$n, scale =
c(4, 0.5), random.order = FALSE, min.freq = 10, colors = brewer.pal(8,
"Dark2"), rot.per = 0.35)
```



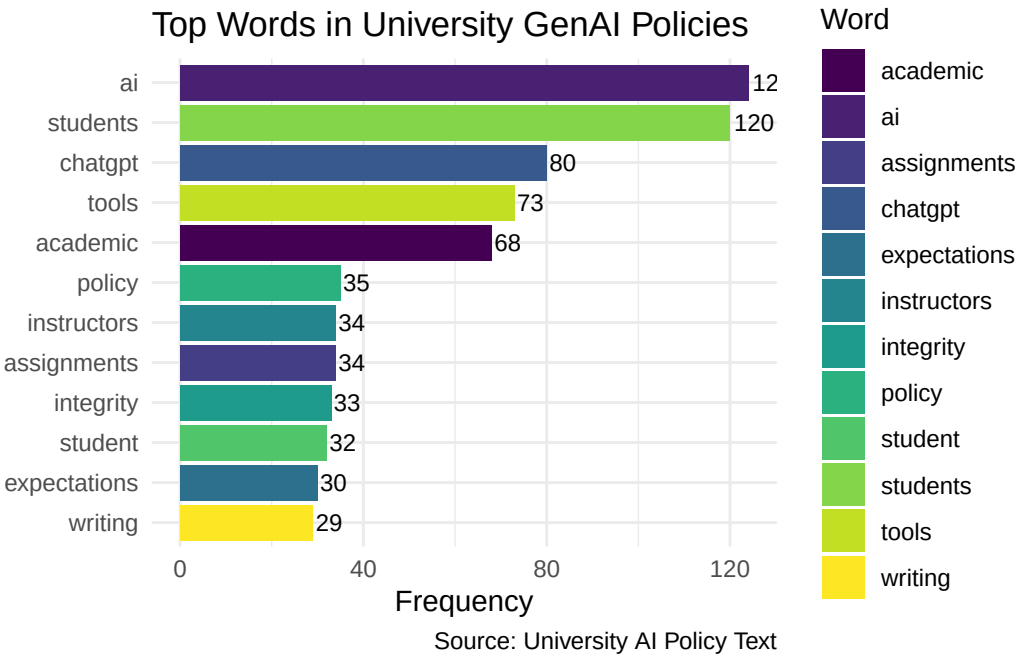
Step 5: Visualize Top 12 Words in University Policies

Select the top 12 most frequent words and create a bar chart to visualize the distribution.

```
# Select the top 12 words
top_words <- word_frequency %>% slice(1:12)

# Generate the bar chart
policy_word_chart <- ggplot(top_words, aes(x = reorder(word, n), y = n,
fill = word)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  labs(
    title = "Top Words in University GenAI Policies",
    x = NULL,
    y = "Frequency",
    caption = "Source: University AI Policy Text",
    fill = "Word"
  ) +
  scale_fill_viridis(discrete = TRUE) +
  geom_text(aes(label = n), vjust = 0.5, hjust = -0.1, size = 3)
```

```
# Print the bar chart
print(policy_word_chart)
```



2.3.5 Results and Discussions

- **RQ1**: What are the most frequently mentioned words in university GenAI writing usage policies?
  The results of the frequency analysis showed that keywords such as "assignment," "student," and "writing" were among the most commonly mentioned terms across AI policies at 100 universities. This emphasis reflects a focus on using AI tools to support student learning and enhance teaching content. The frequent mention of these words suggests that institutions are considering the role of AI in academic assignments and course design, indicating a strategic commitment to integrating AI within educational tasks and student interactions.

- **RQ2**: Which keywords reflect common concerns or focal points related to GenAI writing usage in academic settings?
  The analysis of the top 12 frequently mentioned terms highlighted additional focal points, including "tool," "academic," "instructor," "integrity," and "expectations." These terms reveal concerns around the ethical use of AI tools, the need for clarity in academic applications, and the central role of instructors in AI policy implementation. Keywords

like “integrity” and “expectations” emphasize the importance of maintaining academic standards and setting clear guidelines for AI use in classrooms, while “instructor” underscores the influence faculty members have in shaping AI-related practices. Together, these terms reflect a commitment to transparent policies that support ethical and effective AI integration, enhancing the academic experience for students.

2.4 Dictionary-based Analysis

2.4.1 Purpose

The purpose of dictionary-based analysis in text data is to assess the presence of predefined categories, like emotions or sentiments, within the text using lexicons or dictionaries. This approach allows researchers to quantify qualitative aspects, such as positive or negative sentiment, based on specific words that correspond to these categories.

Case:

In this analysis, we examine the stance of 100 universities on the use of GenAI by applying the Bing sentiment dictionary. By analyzing sentiment scores, we aim to identify the general tone in these policies, indicating whether the institutions’ attitudes toward GenAI are predominantly positive or negative.

2.4.2 Sample Research Questions

- **RQ**: What is the dominant sentiment expressed in GenAI policy texts across universities, and is it primarily positive or negative?

2.4.3 Analysis

Step 1: Install and Load Necessary Libraries

First, install and load the required packages for text processing and visualization.

```
# Install necessary packages if not already installed
#install.packages(c("tidytext","tidyverse" "dplyr", "ggplot2", "tidyr"))

# Load libraries
library(tidytext)
```

```
library(tidyverse)
library(dplyr)
library(ggplot2)
library(tidyr)
```

Step 2: Load and Prepare Data(same as 2.3)

Load the GenAI policy stance data from a CSV file. Be sure to update the file path as needed. we use the same data as 2.3.

```
# Load the dataset (replace "University_GenAI_Policy_Stance.csv" with
the actual file path)
university_policies <- read_csv("University_GenAI_Policy_Stance.csv")
# Tokenize text, remove stop words, and count word frequencies
word_frequency <- university_policies %>%
  unnest_tokens(word, Stance) %>%  # Tokenize the 'Policy_Text' column
  anti_join(stop_words) %>%             # Remove common stop words
  count(word, sort = TRUE)              # Count and sort words by
  frequency
word_frequency
```

Step 3: Tokenize Text Data and Apply Sentiment Dictionary

Tokenize the policy text data to separate individual words. Then, use the Bing sentiment dictionary to label each word as positive or negative.

```
# Tokenize 'Stance' column and apply Bing sentiment dictionary
sentiment_scores <- word_frequency %>%
  inner_join(get_sentiments("bing")) %>% # Join with Bing sentiment
  lexicon
  pivot_wider(names_from = sentiment, values_from = n, values_fill = 0)
  %>%
  mutate(sentiment_score = positive - negative) # Calculate net
  sentiment score

sentiment_scores
```

```
# A tibble: 142 x 4
```

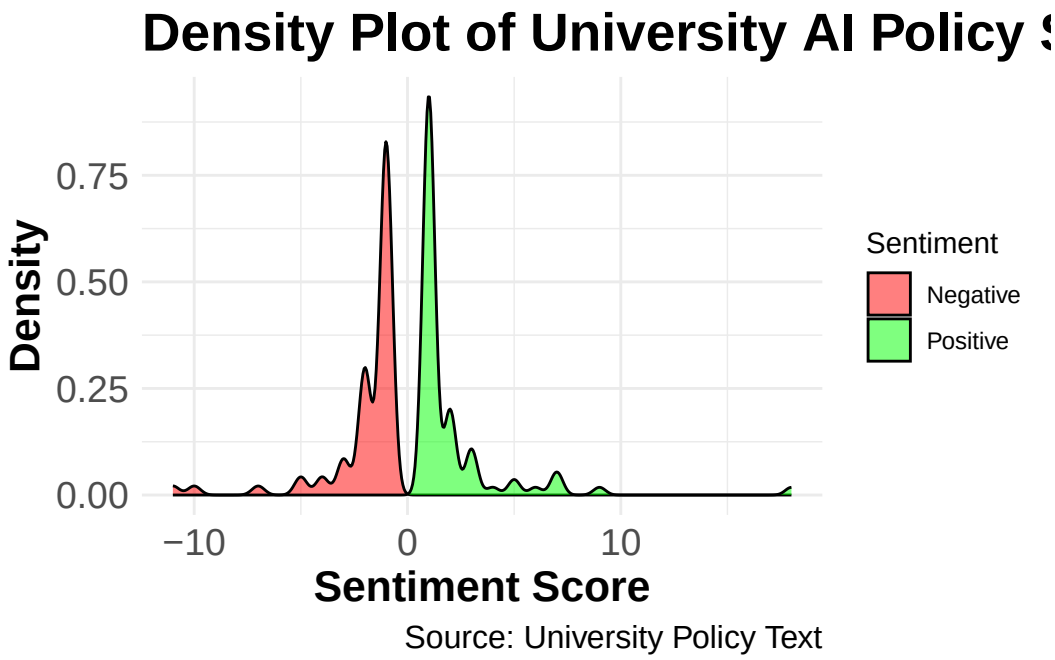
```
   word         positive negative sentiment_score
   <chr>           <int>    <int>           <int>
 1 honor              18        0              18
 2 cheating            0       11             -11
 3 dishonesty          0       10             -10
 4 guidance            9        0               9
 5 honesty             7        0               7
 6 intelligence        7        0               7
 7 transparent         7        0               7
 8 violation           0        7              -7
 9 encourage           6        0               6
10 difficult           0        5              -5
# i 132 more rows
```

Step 4: Create a Density Plot for Sentiment Distribution

Visualize the distribution of sentiment scores with a density plot, showing the prevalence of positive and negative sentiments across university policies.

```r
# Generate a density plot of sentiment scores
density_plot <- ggplot(sentiment_scores, aes(x = sentiment_score, fill
= sentiment_score > 0)) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("red", "green"), name = "Sentiment",
                    labels = c("Negative", "Positive")) +
  labs(
    title = "Density Plot of University AI Policy Sentiment",
    x = "Sentiment Score",
    y = "Density",
    caption = "Source: University Policy Text"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(face = "bold", size = 20),
    axis.text = element_text(size = 14),
    axis.title = element_text(face = "bold", size = 16),
    plot.caption = element_text(size = 12)
  )

# Print the plot
print(density_plot)
```



2.4.4 Results and Discussions

- **RQ**: What is the dominant sentiment expressed in GenAI policy texts across universities, and is it primarily positive or negative?
  The dictionary-based sentiment analysis reveals the prevailing sentiments in university policies on GenAI usage. Using the Bing lexicon to assign positive and negative scores, the density plot illustrates the distribution of sentiment scores across the 100 institutions.

  The results indicate a balanced perspective with a slight tendency toward positive sentiment, as reflected by a higher density of positive scores. This analysis provides insights into the varying degrees of acceptance and caution universities adopt in their AI policy frameworks, demonstrating the diverse stances that shape institutional AI guidelines.

2.5 Clustering-Based Analysis

Clustering-based analysis involves grouping similar text documents or text segments into clusters based on their underlying topics or themes. This approach is particularly useful for identifying dominant themes in text data, such as university AI policy documents.

2.5.1 Purpose

Purpose: The goal of clustering-based analysis is to uncover latent themes in text data using unsupervised machine learning techniques. Topic modeling is one popular method for clustering documents into groups based on their content.

Case: Using the GenAI policy texts from 100 universities, we apply Latent Dirichlet

Allocation (LDA) to identify dominant themes in these policy documents. This analysis will help categorize policies into overarching themes, such as academic integrity, student support, and instructor discretion.

2.5.2 Sample Research Questions

- **RQ1**: What are the prominent themes present in university policies regarding GenAI usage Stance?
- **RQ2**: How do these themes reflect the key concerns or opportunities for intergrating GenAI in higher education?

2.5.3 Analysis

Step 1: Install and Load Necessary Libraries

Install and load the required libraries for text processing and topic modeling.

```
# Install necessary packages
#install.packages(c("dplyr", "tidytext", "topicmodels", "ggplot2"))

# Load libraries
library(dplyr)
library(tidytext)
library(topicmodels)
library(ggplot2)
```

Step 2: Prepare the Data

Load the data and create a document-term matrix (DTM) for topic modeling.

```
# Load the dataset
university_policies <- read_csv("University_GenAI_Policy_Stance.csv")
```

```r
# Tokenize text data and count word frequencies
word_frequency <- university_policies %>%
  unnest_tokens(word, Stance) %>%  # same as section 2.3
  anti_join(stop_words) %>%             # Remove common stop words
  count(word, sort = TRUE)              # Count and sort words by
  frequency
word_frequency
```

```
# A tibble: 1,108 x 2
   word            n
   <chr>       <int>
 1 ai            124
 2 students      120
 3 chatgpt        80
 4 tools          73
 5 academic       68
 6 policy         35
 7 assignments    34
 8 instructors    34
 9 integrity      33
10 student        32
# i 1,098 more rows
```

```r
# Creating Documents - Word Frequency Matrix
gpt_dtm <- word_frequency %>%
  group_by(word) %>%
  mutate(id = row_number()) %>%
  ungroup() %>%
  cast_dtm(document = "id", term = "word", value = "n")
```

```r
library(topicmodels)
library(ggplot2)

# Define range of k values
k_values <- c(2, 3, 4, 5)

# Initialize a data frame to store perplexities
perplexities <- data.frame(k = integer(), perplexity = numeric())
```

```
# Calculate perplexity for each k
for (k in k_values) {
  lda_model <- LDA(gpt_dtm, k = k, control = list(seed = 1234))  # Fit
LDA model
  perplexity_score <- perplexity(lda_model, gpt_dtm)            #
Calculate perplexity
  perplexities <- rbind(perplexities, data.frame(k = k, perplexity =
perplexity_score))
}

# Plot perplexity vs number of topics
ggplot(perplexities, aes(x = k, y = perplexity)) +
  geom_line() +
  geom_point() +
  labs(
    title = "Perplexity vs Number of Topics",
    x = "Number of Topics (k)",
    y = "Perplexity"
  ) +
  theme_minimal()
```

Step 3: Fit the LDA Model

Fit an LDA model with `k = 3` topics.

```
# Converting document-word frequency matrices to sparse matrices
gpt_dtm_sparse<- as(gpt_dtm, "matrix")

# Fit the LDA model
lda_model <- LDA(gpt_dtm_sparse, k = 3, control = list(seed = 1234))

# View model results
gpt_policy_topics_k3 <- tidy(lda_model, matrix = "beta")

print(gpt_policy_topics_k3)
```

```
# A tibble: 3,324 x 3
   topic term        beta
   <int> <chr>      <dbl>
```

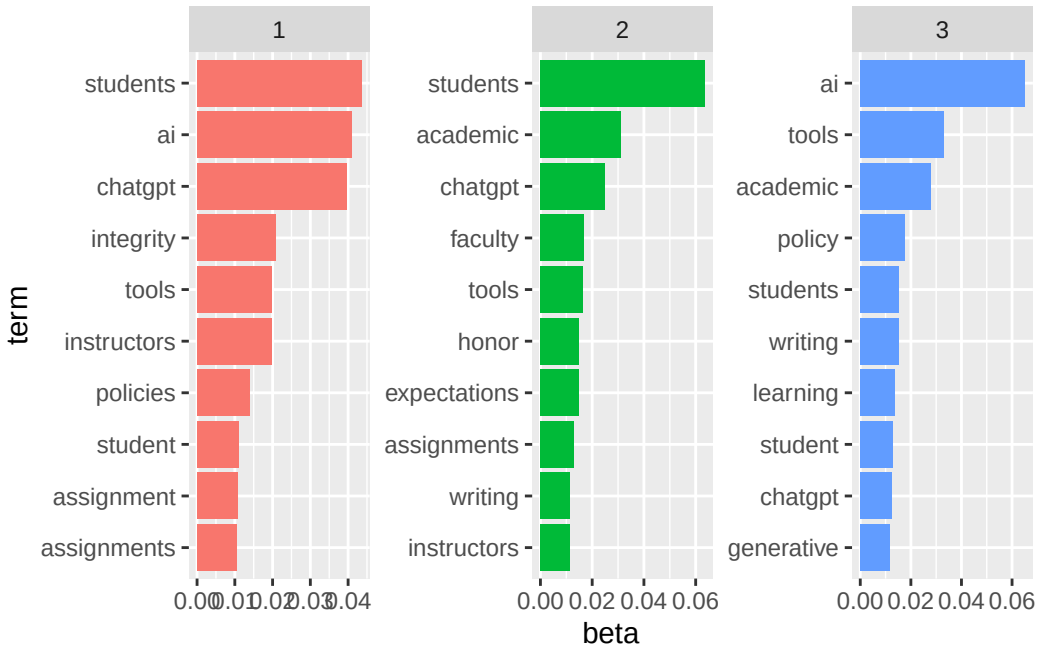
```
 1     1 ai       0.0408
 2     2 ai       0.00593
 3     3 ai       0.0650
 4     1 students 0.0435
 5     2 students 0.0635
 6     3 students 0.0151
 7     1 chatgpt  0.0396
 8     2 chatgpt  0.0248
 9     3 chatgpt  0.0126
10     1 tools    0.0197
# i 3,314 more rows
```

Step 4: Visualize Topics

Extract the top terms for each topic and visualize them.

```r
# Visualizing top terms for each topic
gpt_policy_ap_top_terms_k3 <- gpt_policy_topics_k3 %>%
  group_by(topic) %>%
  slice_max(beta, n = 10) %>%
  ungroup() %>%
  arrange(topic, -beta)

gpt_policy_ap_top_terms_k3 %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered()
```



2.5.4 Results and Discussions

Research Question 1:

What are the prominent themes present in university policies regarding GenAI usage?

Answer:
The topic modeling analysis revealed three distinct themes in the university GenAI policies:

1. **Theme 1: Student-Centric Guidelines and Ethical Considerations**
    - Key terms: `students`, `integrity`, `tools`, `instructors`, `assignment`
    - This theme emphasizes student usage of GenAI in academic settings, with a focus on ethics (`integrity`) and guidelines for instructors to manage assignments involving AI tools.
2. **Theme 2: Academic Standards and Faculty Expectations**
    - Key terms: `students`, `academic`, `faculty`, `honor`, `expectations`
    - This theme focuses on maintaining academic integrity and clarifying expectations for faculty and students regarding GenAI usage in assignments and assessments.
3. **Theme 3: Policy-Level Governance and Technology Integration**

- Key terms: `ai`, `tools`, `policy`, `learning`, `generative`
- This theme revolves around institutional policies on AI integration, highlighting broader governance strategies and how generative AI (like GenAI) fits into learning environments.

Research Question 2:

How do these themes reflect the key concerns or opportunities for integrating GenAI in higher education?

Answer:
The identified themes reflect both concerns and opportunities:

1. **Concerns:**

   Theme 1: Highlights the ethical challenges, such as ensuring academic integrity when students use AI tools in their coursework. Institutions are keen on setting clear guidelines for both students and instructors to avoid misuse.

   Theme 2: Underlines the potential for conflict between maintaining academic standards (`honor`, `expectations`) and leveraging AI to support learning. This shows a cautious approach to integrating AI while upholding traditional values.

   Theme 3: Raises policy-level questions on AI governance, such as whether existing institutional frameworks are adequate to regulate emerging generative AI technologies.

2. **Opportunities:**

   Theme 1: Presents a chance to redefine how students interact with AI tools to foster responsible and innovative usage, particularly for assignments and creative tasks.

   Theme 2: Encourages collaboration between faculty and administration to develop robust expectations and support systems for integrating AI in the classroom.

   Theme 3: Offers a strategic opportunity for universities to lead in AI adoption by establishing comprehensive policies that guide AI's role in education and research.

Discussion:

The topic modeling results suggest that universities are navigating a complex landscape of opportunities and challenges as they incorporate GenAI into academic contexts. While student-centric policies aim to balance innovation with ethical considerations, institutional-level themes signal the need for governance frameworks to ensure responsible AI use. These findings indicate that higher education institutions are positioned to play a pivotal role in shaping the future of

generative AI in learning, provided they address the ethical, pedagogical, and policy challenges identified in this analysis.

References:

Saldaña, J. (2021). Coding techniques for quantitative and mixed data. *The Routledge reviewer's guide to mixed methods analysis*, 151-160.

Networks Data

3.1 Overview

Social network analysis (SNA) is the process of investigating social structures through the use of networks and graph theory. It is a technique used to map and measure relationships and flows between people, groups, organizations, computers, or other information/knowledge processing entities. SNA can be a useful tool for understanding the team structures, for example, in an online classroom. It can be an additional layer of understanding the outcomes (or predictors) of certain instructional interventions. Used this way SNA can be used to identify patterns and trends in social networks, as well as to understand how these networks operate. Additionally, SNA can be used to predict future behavior in social networks, and to design interventions that aim to improve the functioning of these networks.

3.2 Accessing SNA Data

Social Network Analysis (SNA) relies on relational data—information about **connections** (edges) between entities (nodes) such as students, teachers, or organizations. Compared to traditional survey or tabular data, SNA requires **pairwise relational information**. In education, this could include "who collaborates with whom," "who talks to whom," or digital traces of discussion and collaboration in online platforms.

3.2.1 Types and Sources of SNA Data

There are several common sources and structures for SNA data in educational and social science contexts:

- **Survey-based Network Data**: Collected via roster or name generator questions, e.g., "List the classmates you discuss assignments with."
- **Behavioral/Observational Data**: Derived from logs of actual interactions, e.g., forum replies, emails, classroom seating.
- **Archival or Digital Trace Data**: Extracted from digital platforms such as MOOCs, LMS discussion forums, Slack, Twitter, or Facebook.

- **Administrative/Organizational Data**: Information about formal structures such as team membership or co-authorship.

Data Structure: Most SNA data are formatted as: - **Edge List** (two columns: source and target) - **Adjacency Matrix** (rows and columns are actors; cell values indicate a tie) - **Node Attributes** (supplementary information about each node, e.g., gender, role)

3.2.2 Example 1: Creating a Simple Network from an Edge List

Below is an example of constructing a network from a simple CSV edge list. This mirrors typical classroom survey data ("who do you consider your friend in this class?").

```
# Install and load the igraph package
install.packages("igraph")
library(igraph)

# Example: Load an edge list from CSV
edge_list <- read.csv("data/friendship_edges.csv")

# Create the graph object (directed network)
g <- graph_from_data_frame(edge_list, directed = TRUE)

# Plot the network
plot(g, main = "Friendship Network")
```

3.2.3 Example 2: Generating Network Data from Digital Traces

Many educational datasets now come from online discussion forums, MOOCs, or LMS systems. For example, the MOOC case study (Kellogg & Edelmann, 2015) uses reply relationships in online courses to construct discussion networks.

```
# Suppose you have a data frame with columns: from_user, to_user
mooc_edges <- read.csv("data/mooc_discussion_edges.csv")
g_mooc <- graph_from_data_frame(mooc_edges, directed = TRUE)
plot(g_mooc, main = "MOOC Discussion Network")
```

3.2.4 Example 3: Collecting SNA Data via Surveys

If you want to collect your own network data:

- Ask participants to **name or select** (from a roster) their friends, collaborators, or contacts.
- Compile responses into an edge list.
- Example survey prompt:

  > "Please list up to five classmates you seek help from most frequently."

Tip:

Survey-based SNA is easier to manage with small to medium groups. For larger networks, digital trace or archival data may be more practical.

3.2.5 Node Attribute Data

You can also load additional data about each node (student, teacher, etc.) to enable richer analyses (e.g., centrality by gender or role).

```
node_attributes <- read.csv("data/friendship_nodes.csv")
# Add attributes to igraph object
V(g)$gender <- node_attributes$gender[match(V(g)$name, node_attributes$name)]
```

3.2.6 Further Examples

- **Public Datasets**:
  - MOOC Discussion Networks
  - Add Health
  - Common Core Twitter Networks (Supovitz et al.)
- **Synthetic Data**:
  - R's `igraph` package can also generate sample networks for practice:

    ```
    g_sample <- sample_gnp(n = 10, p = 0.3)
    plot(g_sample, main = "Random Sample Network")
    ```

3.2.7 Best Practices and Tips

- **Ethics**: Social network data can be sensitive. Protect anonymity and comply with IRB/data use guidelines.
- **Format Consistency**: Always clarify whether ties are directed/undirected, binary/weighted, and ensure consistent formatting.
- **Missing Data**: Especially in survey-based SNA, missing responses can impact network structure and interpretation.

3.2.8 Summary

Accessing SNA data involves both careful **design** (in the case of surveys/observations) and **extraction/wrangling** (in the case of digital traces or archival records). The choice of data source and structure will directly influence the kinds of questions you can answer with SNA.

Recommended Reading:

- Borgatti, S. P., Everett, M. G., & Johnson, J. C. (2018). *Analyzing Social Networks* (2nd ed). SAGE.
- Kellogg, S., & Edelmann, A. (2015). Massive open online course discussion forums as networks.

3.3 Network Management & Measurement in Social Network Analysis

3.3.1 Purpose + Case

Purpose: This section demonstrates how to manage, measure, and visualize large-scale discussion networks from online professional development settings. Through this real-world example, we guide readers in loading relational data, constructing a directed network, and conducting a suite of essential SNA measures. The focus is on classroom- and course-level online discussions, which are representative of many contemporary educational and research settings.

Case Study: The case data comes from two cohorts of an online professional development program ("DLT1" and "DLT2"). Each cohort's discussion data includes (a) **edge list** data capturing who replied to whom, and (b) **node/actor** attributes describing roles (e.g., facilitator, expert). These data allow us to reconstruct and analyze the full structure of communication in two authentic online learning communities.

3.3.2 Sample Research Questions

- **RQ1:** What is the overall structure of interaction in each online discussion cohort? Are they densely connected, or fragmented?
- **RQ2:** Who are the most central or influential actors in the network? How do facilitators or experts compare with regular participants?
- **RQ3:** To what extent are ties reciprocated (mutual) and how cohesive are the networks?
- **RQ4:** How do the network properties (e.g., density, reciprocity, clustering) compare between cohorts?

3.3.3 Analysis

Step 1: Install and Load Required Packages

```
# Install and load necessary libraries
#install.packages(c("tidygraph", "ggraph", "readr", "janitor"))
library(tidygraph)
library(ggraph)
library(readr)
library(janitor)
library(igraph)
library(dplyr)
```

Step 2: Import and Clean Data Load Edges and Node Attributes for DLT1:

```
# Load edge list (who replied to whom)
dlt1_ties <- read_csv("data/dlt1-edges.csv",
  col_types = cols(Sender = col_character(),
                   Receiver = col_character(),
                   `Category Text` = col_skip(),
                   `Comment ID` = col_character(),
                   `Discussion ID` = col_character())) |>
  clean_names()

# Load node attributes (participant roles, etc.)
dlt1_actors <- read_csv("data/dlt1-nodes.csv",
  col_types = cols(UID = col_character(),
                   Facilitator = col_character(),
                   expert = col_character(),
```

```
                    connect = col_character())) |>
  clean_names()

head(dlt1_ties)
```

```
# A tibble: 6 x 9
  sender receiver timestamp discussion_title discussion_category parent_category
  <chr>  <chr>    <chr>     <chr>            <chr>                <chr>
1 360    444      4/4/13 1~ Most important ~ Group N              Units 1-3 Disc~
2 356    444      4/4/13 1~ Most important ~ Group D-L            Units 1-3 Disc~
3 356    444      4/4/13 1~ DLT Resources-C~ Group D-L            Units 1-3 Disc~
4 344    444      4/4/13 1~ Most important ~ Group O-T            Units 1-3 Disc~
5 392    444      4/4/13 1~ Most important ~ Group U-Z            Units 1-3 Disc~
6 219    444      4/4/13 1~ Most important ~ Group M              Units 1-3 Disc~
# i 3 more variables: discussion_identifier <chr>, comment_id <chr>,
#   discussion_id <chr>
```

```
head(dlt1_actors)
```

```
# A tibble: 6 x 13
  uid   facilitator role1  experience experience2 grades location region country
  <chr> <chr>       <chr>       <dbl> <chr>       <chr>  <chr>    <chr>  <chr>
1 1     0           libme~          1 6 to 10     secon~ VA       South  US
2 2     0           class~          1 6 to 10     secon~ FL       South  US
3 3     0           distr~          2 11 to 20    gener~ PA       North~ US
4 4     0           class~          2 11 to 20    middle NC       South  US
5 5     0           other~          3 20+         gener~ AL       South  US
6 6     0           class~          1 4 to 5      gener~ AL       South  US
# i 4 more variables: group <chr>, gender <chr>, expert <chr>, connect <chr>
```

Step 3: Construct and Explore the Network

```
# Build the directed network graph (nodes: uid, edges: sender->receiver)
dlt1_network <- tbl_graph(
  edges = dlt1_ties,
  nodes = dlt1_actors,
  node_key = "uid",
  directed = TRUE
```

```
)

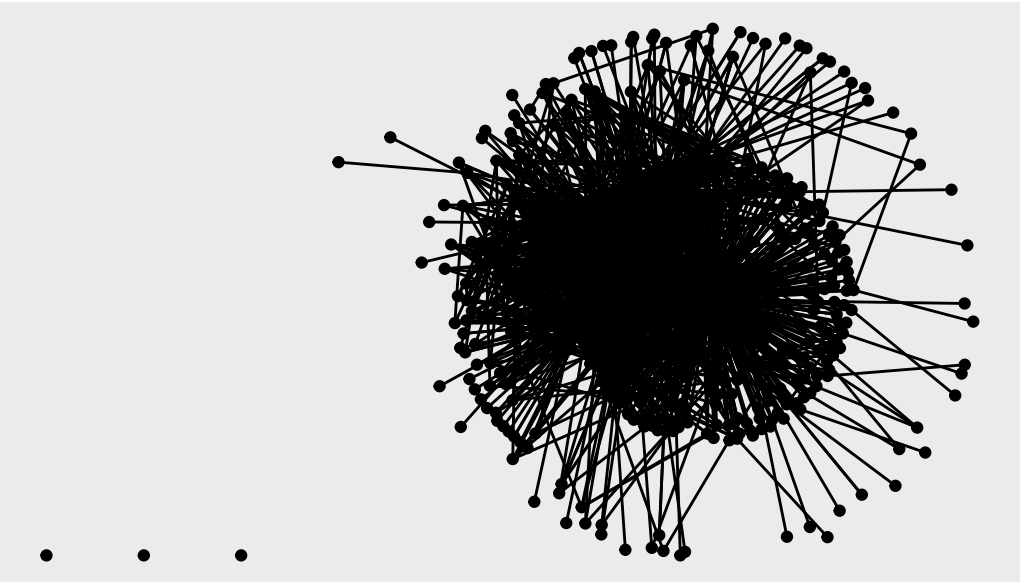
# Overview of the network
dlt1_network
```

```
# A tbl_graph: 445 nodes and 2529 edges
#
# A directed multigraph with 4 components
#
# Node Data: 445 x 13 (active)
   uid   facilitator role1 experience experience2 grades location region country
   <chr> <chr>       <chr>      <dbl> <chr>       <chr>  <chr>    <chr>  <chr>
 1 1     0           libm~          1 6 to 10     secon~ VA       South  US
 2 2     0           clas~          1 6 to 10     secon~ FL       South  US
 3 3     0           dist~          2 11 to 20    gener~ PA       North~ US
 4 4     0           clas~          2 11 to 20    middle NC       South  US
 5 5     0           othe~          3 20+         gener~ AL       South  US
 6 6     0           clas~          1 4 to 5      gener~ AL       South  US
 7 7     0           inst~          2 11 to 20    gener~ SD       Midwe~ US
 8 8     0           spec~          1 6 to 10     secon~ BE       Inter~ BE
 9 9     0           clas~          1 6 to 10     middle NC       South  US
10 10    0           scho~          2 11 to 20    middle NC       South  US
# i 435 more rows
# i 4 more variables: group <chr>, gender <chr>, expert <chr>, connect <chr>
#
# Edge Data: 2,529 x 9
   from    to timestamp    discussion_title  discussion_category parent_category
  <int> <int> <chr>        <chr>             <chr>               <chr>
1   360   444 4/4/13 16:32 Most important c~ Group N             Units 1-3 Disc~
2   356   444 4/4/13 18:45 Most important c~ Group D-L           Units 1-3 Disc~
3   356   444 4/4/13 18:47 DLT Resources-Co~ Group D-L           Units 1-3 Disc~
# i 2,526 more rows
# i 3 more variables: discussion_identifier <chr>, comment_id <chr>,
#   discussion_id <chr>
```

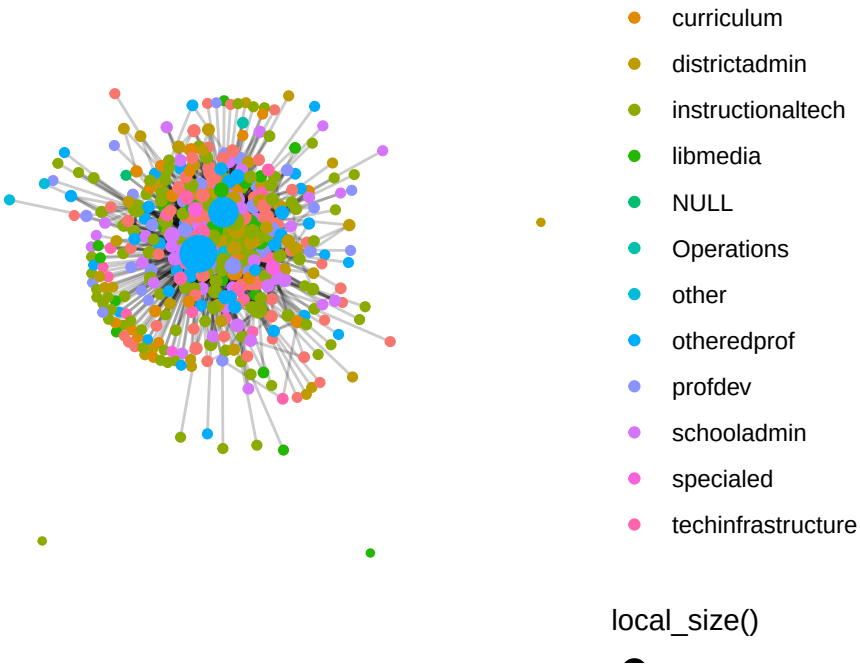
```
# Output: 445 nodes, 2529 edges, 4 components (directed multigraph)
```

Step 4: Basic Visualization

```
# Quick overview plot (stress layout by default)
autograph(dlt1_network)
```



```
# Custom visualization with colors and centrality
ggraph(dlt1_network, layout = "fr") +
  geom_edge_link(alpha = .2) +
  geom_node_point(aes(color = role1, size = local_size())) +
  theme_graph()+
  theme(text = element_text(family = "sans"))
```



Step 5: Network Size and Centralization

```
# Number of nodes and edges
gorder(dlt1_network)   # 445
```

```
[1] 445
```

```
gsize(dlt1_network)    # 2529
```

```
[1] 2529
```

```
# Degree centrality (all, in, out)
deg_all <- centr_degree(dlt1_network, mode = "all")$res
deg_in  <- centr_degree(dlt1_network, mode = "in")$res
deg_out <- centr_degree(dlt1_network, mode = "out")$res
# Centralization
centr_degree(dlt1_network, mode = "all")$centralization  # 0.64
```

```
[1] 0.6429242
```

```
centr_degree(dlt1_network, mode = "in")$centralization   # 1.06
```

```
[1] 1.05702
```

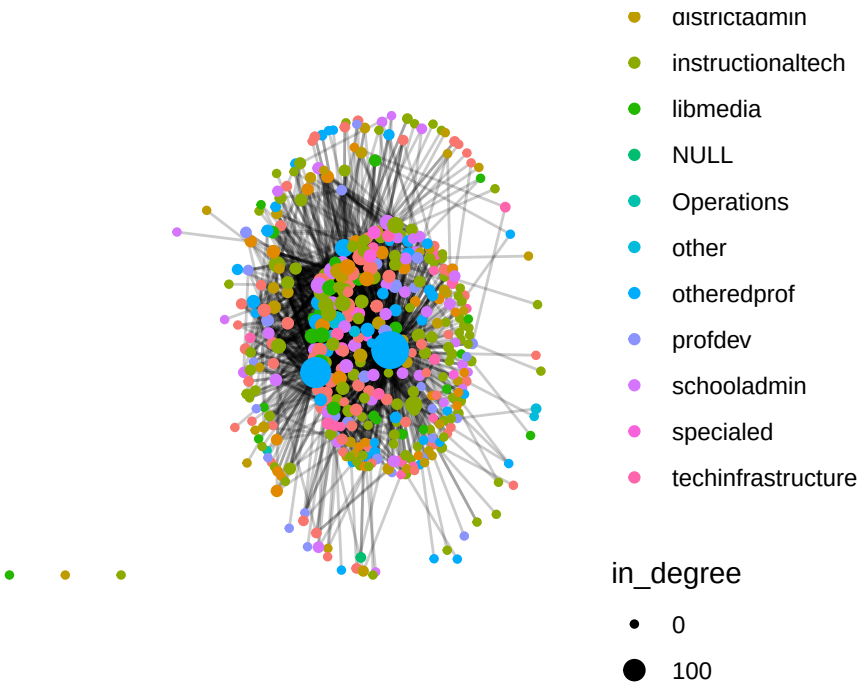
```
centr_degree(dlt1_network, mode = "out")$centralization  # 0.23
```

```
[1] 0.2259389
```

Step 6: Attach and Visualize Node Centrality

```
# Add in-degree centrality as node attribute
dlt1_network <- dlt1_network |>
  activate(nodes) |>
  mutate(in_degree = centrality_degree(mode = "in"))

# Plot, sizing nodes by in-degree
ggraph(dlt1_network) +
  geom_edge_link(alpha = .2) +
  geom_node_point(aes(size = in_degree, color = role1)) +
  theme_graph()+
  theme(text = element_text(family = "sans"))
```



Step 7: Network Density, Reciprocity, Clustering, Distance

```
# Density
edge_density(dlt1_network)      # 0.013 (sparse network)
```

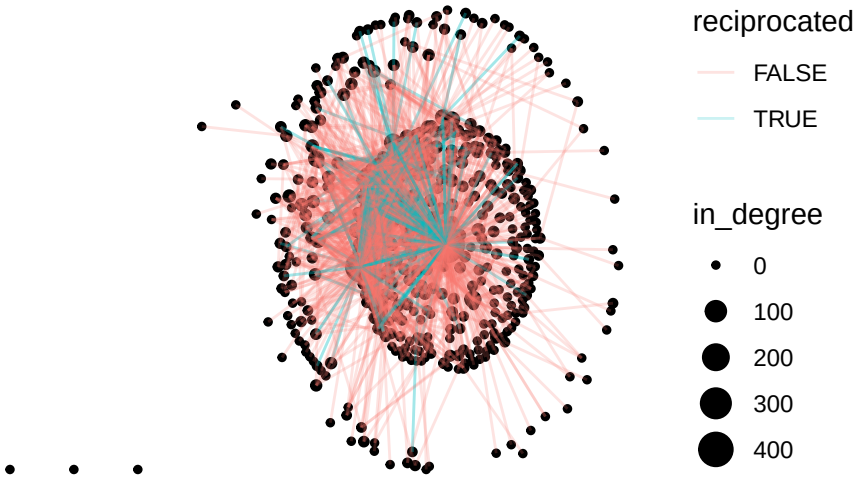
```
[1] 0.01279988
```

```
# Reciprocity
reciprocity(dlt1_network)       # 0.20 (20% of ties are reciprocated)
```

```
[1] 0.1997544
```

```
# Add reciprocated edge attribute and plot
dlt1_network <- dlt1_network |>
  activate(edges) |>
  mutate(reciprocated = edge_is_mutual())
ggraph(dlt1_network) +
  geom_node_point(aes(size = in_degree)) +
```

```
  geom_edge_link(aes(color = reciprocated), alpha = .2) +
  theme_graph()+
  theme(text = element_text(family = "sans"))
```




```
# Clustering (transitivity/global)
transitivity(dlt1_network)      # 0.089
```

```
[1] 0.08880774
```

```
# Network diameter (longest shortest path) & average distance
diameter(dlt1_network)          # 8
```

```
[1] 8
```

```
mean_distance(dlt1_network)     # 3.03
```

```
[1] 3.030694
```

Step 8:Repeat for DLT2

```
# Step 8: Repeat for DLT2

# 1. Load the DLT2 edge and node data
dlt2_ties <- read_csv("data/dlt2-edges.csv",
  col_types = cols(Sender = col_character(),
                   Receiver = col_character(),
                   `Category Text` = col_skip(),
                   `Comment ID` = col_character(),
                   `Discussion ID` = col_character())) |>
  clean_names()

dlt2_actors <- read_csv("data/dlt2-nodes.csv",
  col_types = cols(UID = col_character(),
                   Facilitator = col_character(),
                   expert = col_character(),
                   connect = col_character())) |>
  clean_names()

# 2. Construct the directed network
dlt2_network <- tbl_graph(
  edges = dlt2_ties,
  nodes = dlt2_actors,
  node_key = "uid",
  directed = TRUE
)

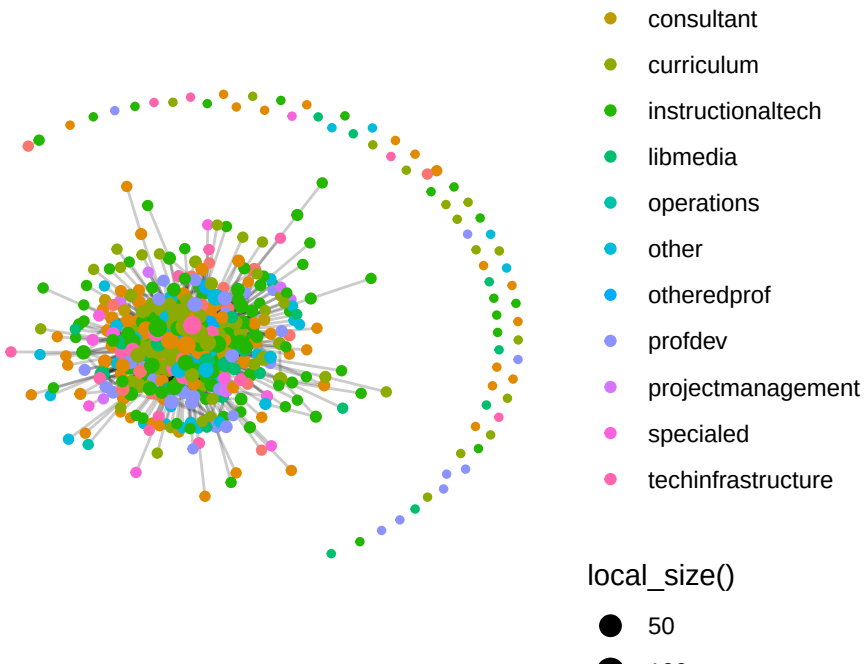
# 3. Basic network properties
num_nodes <- gorder(dlt2_network)   # Number of nodes
num_edges <- gsize(dlt2_network)    # Number of edges

# 4. Degree centrality (overall, in, out)
deg_all <- centr_degree(dlt2_network, mode = "all")$res
deg_in  <- centr_degree(dlt2_network, mode = "in")$res
deg_out <- centr_degree(dlt2_network, mode = "out")$res

# Centralization values
centr_all <- centr_degree(dlt2_network, mode = "all")$centralization
centr_in  <- centr_degree(dlt2_network, mode = "in")$centralization
centr_out <- centr_degree(dlt2_network, mode = "out")$centralization
```

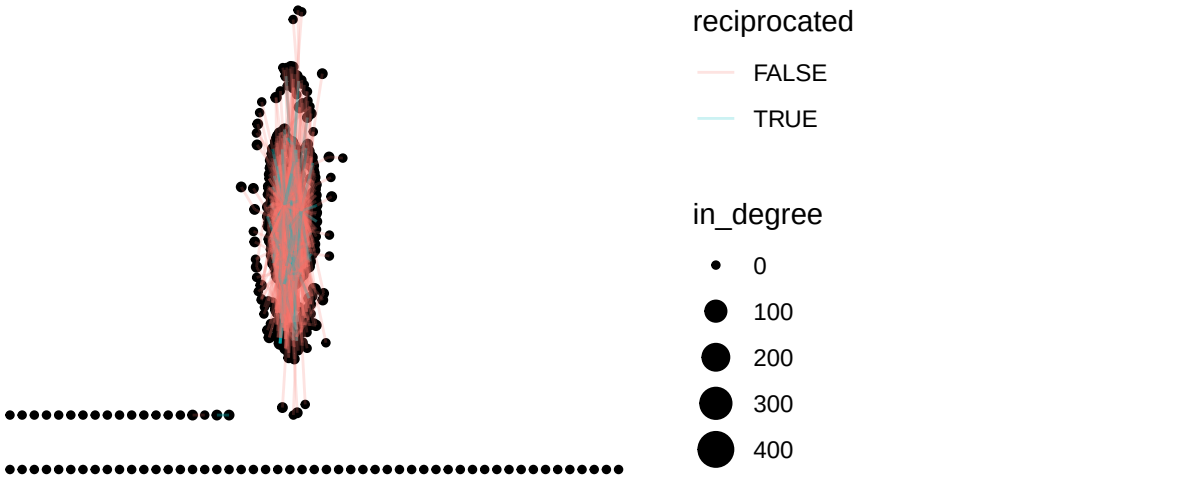
```
# 5. Attach centrality as a node attribute
dlt2_network <- dlt2_network |>
  activate(nodes) |>
  mutate(in_degree = centrality_degree(mode = "in"))

# 6. Visualize the network
ggraph(dlt2_network, layout = "fr") +
  geom_edge_link(alpha = .2) +
  geom_node_point(aes(color = role, size = local_size())) +
  theme_graph()+
  theme(text = element_text(family = "sans"))
```




```
# 7. Density, reciprocity, clustering, distances
density <- edge_density(dlt2_network)
recip   <- reciprocity(dlt2_network)
dlt2_network <- dlt2_network |>
  activate(edges) |>
  mutate(reciprocated = edge_is_mutual())
ggraph(dlt2_network) +
  geom_node_point(aes(size = in_degree)) +
```

```
  geom_edge_link(aes(color = reciprocated), alpha = .2) +
  theme_graph()+
  theme(text = element_text(family = "sans"))
```




```
trans   <- transitivity(dlt2_network)
diam    <- diameter(dlt2_network)
mean_d  <- mean_distance(dlt2_network)

# 8. Print summary statistics
cat("DLT2 Network Stats:\n")
```

```
DLT2 Network Stats:
```

```
cat("Nodes:", num_nodes, "Edges:", num_edges, "\n")
```

```
Nodes: 492 Edges: 2584
```

```
cat("Degree Centralization (all/in/out):", centr_all, centr_in,
centr_out, "\n")
```

```
Degree Centralization (all/in/out): 0.5311161 0.8650671 0.3273889
```

```
cat("Density:", density, "Reciprocity:", recip, "\n")
```

```
Density: 0.0106966 Reciprocity: 0.2500977
```

```
cat("Transitivity:", trans, "Diameter:", diam, "Mean Distance:", mean_d,
"\n")
```

```
Transitivity: 0.1248291 Diameter: 8 Mean Distance: 3.03815
```

3.3.4 Results and Discussion

RQ1: What is the overall structure of interaction in each cohort?

- **DLT1** consists of **445 nodes** (participants) and **2529 edges** (directed interactions).
- **DLT2** has **492 nodes** and **2584 edges**.
- Both networks are **large and sparse**:
  - *Density*: DLT1 = **0.013**, DLT2 = **0.011**
  - Interpretation: Only about 1–1.3% of all possible connections exist—typical for online discussion networks where not every participant interacts with every other.
- Both networks are **multi-component** (several disconnected groups), but most participants are included in the main giant component.
- The **diameter** (longest shortest path) is 8 for both cohorts, and the **average shortest path length** is about **3.03 (DLT1)** and **3.04 (DLT2)**, indicating that on average, any participant is just 3 steps away from any other in the largest group.
- **Interpretation**: Information or discussion threads can reach most participants with only a few hops, but overall engagement is selective rather than comprehensive.

RQ2: Who are the most central or influential actors?

- Centrality (degree, in-degree, out-degree) analyses show a **right-skewed distribution**: most participants have low centrality, but a small subset are highly connected.
- In both DLT1 and DLT2, **facilitators** and a handful of highly active participants emerge as **hubs**—they initiate and/or receive a disproportionate number of interactions.
  - For DLT1, degree centralization (all): **0.64** (in: **1.06**, out: **0.23**)
  - For DLT2, degree centralization (all): **0.53** (in: **0.87**, out: **0.33**)

- **Visualization**: Network plots with node size proportional to in-degree clearly highlight these central actors.
- **Interpretation**: These key individuals (often facilitators) play critical roles in steering discussion, providing feedback, and potentially keeping less active members engaged.

RQ3: Are ties reciprocated?

- **Reciprocity** (proportion of mutual connections):
  - DLT1: **0.20** (20% of ties are reciprocated)
  - DLT2: **0.25** (25% reciprocated)
- **Interpretation**: Most interactions are one-way (e.g., a reply that does not receive a response), but a substantial fraction are mutual—possibly reflecting peer-to-peer conversations or ongoing exchanges. In online learning contexts, this suggests a mix of broadcasting (one-to-many) and genuine dialog (two-way).

RQ4: How cohesive are the networks?

- **Transitivity/Clustering coefficient** (probability that two connected nodes' neighbors are also connected):
  - DLT1: **0.089**
  - DLT2: **0.125**
- **Interpretation**: Triadic closure is low—there are few closed triangles, so close-knit groups (where "my friend is also your friend") are rare. The network structure is more "hub-and-spoke" than "cliquish."
- **Diameter**: 8 for both, showing that even the furthest nodes can be reached in 8 steps.
- **Mean distance**: ~3.0, so participants are relatively close to each other in the main component.

Comparison Across Cohorts

- **DLT2 is slightly larger** (more participants and interactions), but the structural properties—density, centralization, reciprocity, clustering, and path lengths—are all quite similar.
- Minor variations (e.g., higher reciprocity and clustering in DLT2) could reflect differences in facilitation style, cohort engagement, or participant composition.
- **Interpretation**: Both cohorts exhibit classic patterns for large-scale online educational discussions—a small number of central actors drive most of the interaction, the networks are sparse but efficiently connected, and genuine dialogue is present but not universal.

Educational Implications

- **For educators and instructional designers**:

  These findings suggest that a small group of highly active facilitators or students are critical to fostering interaction. Encouraging more distributed engagement (for example, through peer response requirements or rotating leadership) may enhance network cohesion and learning opportunities.

- **For researchers**:

  Understanding who occupies central positions and the overall structure of discussion networks can inform interventions to support isolated participants, promote reciprocity, and create more connected learning communities.

Summary:

Through these SNA measures, we have shown how to reconstruct, visualize, and interpret the structure of large-scale online discussion networks. The approach enables identification of core communicators, understanding of participation patterns, and empirical comparison across cohorts or interventions. This "cookbook" can be adapted to other online learning or collaborative contexts.

> **Note**: This analysis is based on real-world data from online professional development courses. The methods and findings can be generalized to other educational settings where social networks play a role in learning and collaboration.

3.4 Case Study: Hashtag Common Core

3.4.1 Purpose & Case

The purpose of this case study is to demonstrate the application of social network analysis (SNA) in a real-world policy context: the heated national debate over the Common Core State Standards (CCSS) as it played out on Twitter. Drawing on the work of Supovitz, Daly, del Fresno, and Kolouch, the #COMMONCORE Project provides a vivid example of how social media-enabled networks shape educational discourse and policy.

This case focuses on: - Identifying **key actors** ("transmitters," "transceivers," and "transcenders") and measuring their influence, - Detecting **subgroups/factions** within the conversation, - Exploring how **sentiment** about the Common Core varies across network positions, - Demonstrating **network wrangling, visualization, and analysis** using real tweet data.

Data Source

Data was collected from Twitter's public API using keywords/hashtags related to the Common Core (e.g., `#commoncore`, `ccss`, `stopcommoncore`). The dataset includes user names, tweets, mentions, retweets, and relevant timestamps from a sample week. Only public tweets are included, and user privacy is respected.

3.4.2 Sample Research Questions

- **RQ1:** Who are the "transmitters," "transceivers," and "transcenders" in the Common Core Twitter network?
- **RQ2:** What subgroups or factions exist within the network, and how are they structured?
- **RQ3:** How does sentiment about the Common Core vary across actors and subgroups?
- **RQ4:** What other patterns of communication (e.g., centrality, clique formation, isolates) characterize this network?

3.4.3 Analysis

Step 1: Load Required Packages

```
library(tidyverse)
library(tidygraph)
library(ggraph)
library(skimr)
library(igraph)
library(tidytext)
library(vader)
```

Step 2: Data Import and Wrangling

```
# Import tweet data (edgelist format: sender, receiver, timestamp, text)
ccss_tweets <- read_csv("data/ccss-tweets.csv")

# Prepare the edgelist (extract sender, mentioned users, and tweet text)
ties_1 <- ccss_tweets %>%
  relocate(sender = screen_name, target = mentions_screen_name) %>%
  select(sender, target, created_at, text)
```

```r
# Unnest receiver to handle multiple mentions per tweet
ties_2 <- ties_1 %>%
  unnest_tokens(input = target,
                output = receiver,
                to_lower = FALSE) %>%
  relocate(sender, receiver)

# Remove tweets without mentions to focus on direct connections
ties <- ties_2 %>%
  drop_na(receiver)

# Save for reproducibility
write_csv(ties, "data/ccss-edgelist.csv")

# Build nodelist
actors_1 <- ties %>%
  select(sender, receiver) %>%
  pivot_longer(cols = c(sender,receiver))

actors <- actors_1 %>%
  select(value) %>%
  rename(actors = value) %>%
  distinct()

write_csv(actors, "data/ccss-nodelist.csv")
```

Step 3: Create Network Object

```r
ccss_network <- tbl_graph(edges = ties,
                          nodes = actors,
                          directed = TRUE)
ccss_network
```

```
# A tbl_graph: 46 nodes and 42 edges
#
# A directed multigraph with 14 components
#
# Node Data: 46 x 1 (active)
```

```
   actors
   <chr>
 1 DistanceLrnBot
 2 k12movieguides
 3 WEquilSchool
 4 JoeWEquil
 5 SumayLu
 6 fluttbot
 7 BodShameless
 8 Math
 9 ozsultan
10 sfchronicle
# i 36 more rows
#
# Edge Data: 42 x 4
   from    to created_at          text
  <int> <int> <dttm>              <chr>
1     1     2 2021-06-28 09:53:54 "#Luca Movie Guide | Worksheet | Questions | ~
2     3     4 2021-06-28 02:32:59 "Why public schools should focus more on buil~
3     3     3 2021-06-28 02:32:59 "Why public schools should focus more on buil~
# i 39 more rows
```

Step 4: Network Structure – Components, Cliques, and Communities

- **Components**
  - Identify weak and strong components (connected subgroups):

```
ccss_network <- ccss_network |>
  activate(nodes) |>
  mutate(weak_component = group_components(type = "weak"),
         strong_component = group_components(type = "strong"))
# View component sizes
ccss_network |>
  as_tibble() |>
  group_by(weak_component) |>
  summarise(size = n()) |>
  arrange(desc(size))
```

```
# A tibble: 14 x 2
   weak_component  size
```

```
            <int> <int>
 1              1    14
 2              2     6
 3              3     4
 4              4     3
 5              5     3
 6              6     2
 7              7     2
 8              8     2
 9              9     2
10             10     2
11             11     2
12             12     2
13             13     1
14             14     1
```

- **Cliques**
  - Identify fully connected subgroups (if any):

```
clique_num(ccss_network)
```

```
[1] 4
```

```
cliques(ccss_network, min = 3)
```

```
[[1]]
+ 3/46 vertices, from a12c5c5:
[1] 4 5 6

[[2]]
+ 3/46 vertices, from a12c5c5:
[1] 39 40 41

[[3]]
+ 3/46 vertices, from a12c5c5:
[1] 3 4 6

[[4]]
+ 4/46 vertices, from a12c5c5:
[1] 3 4 5 6
```

```
[[5]]
+ 3/46 vertices, from a12c5c5:
[1] 3 4 5

[[6]]
+ 3/46 vertices, from a12c5c5:
[1] 3 5 6
```

- **Communities**
  - Detect densely connected communities using edge betweenness:

```
ccss_network <- ccss_network |>
  morph(to_undirected) |>
  activate(nodes) |>
  mutate(sub_group = group_edge_betweenness()) |>
  unmorph()
ccss_network |>
  as_tibble() |>
  group_by(sub_group) |>
  summarise(size = n()) |>
  arrange(desc(size))
```

```
# A tibble: 16 x 2
   sub_group  size
       <int> <int>
 1         1    10
 2         2     6
 3         3     4
 4         4     3
 5         5     3
 6         6     2
 7         7     2
 8         8     2
 9         9     2
10        10     2
11        11     2
12        12     2
13        13     2
14        14     2
15        15     1
16        16     1
```

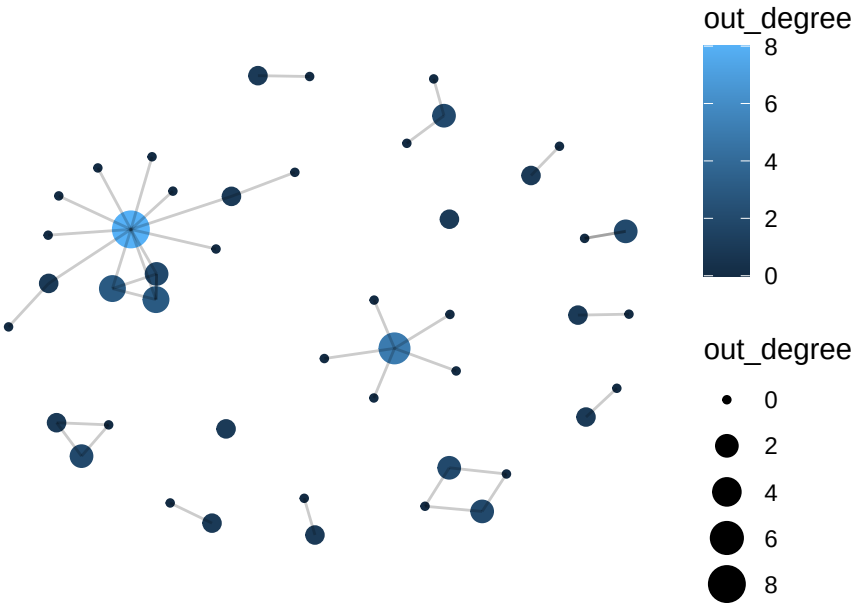
Step 5: Egocentric Analysis – Centrality & Key Actors

```
ccss_network <- ccss_network |>
  activate(nodes) |>
  mutate(
    size = local_size(),
    in_degree = centrality_degree(mode = "in"),
    out_degree = centrality_degree(mode = "out"),
    closeness = centrality_closeness(),
    betweenness = centrality_betweenness()
  )

# Identify top actors by out_degree (transmitters), in_degree
(transceivers), and both (transcenders)
top_transmitters <- ccss_network %>% as_tibble() %>%
arrange(desc(out_degree)) %>% head(5)
top_transceivers <- ccss_network %>% as_tibble() %>%
arrange(desc(in_degree)) %>% head(5)
top_transcenders <- ccss_network %>% as_tibble() %>%
  filter(out_degree > quantile(out_degree, 0.9) & in_degree >
  quantile(in_degree, 0.9))
```

Step 6: Visualize the Network

```
ggraph(ccss_network, layout = "fr") +
  geom_node_point(aes(size = out_degree, color = out_degree)) +
  geom_edge_link(alpha = .2) +
  theme_graph()+
  theme(text = element_text(family = "sans"))
```



Step 7: Sentiment Analysis (Optional)

If you want to analyze sentiment as in the original #COMMONCORE study:

```
library(vader)
vader_ccss <- vader_df(ccss_tweets$text)
 mean(vader_ccss$compound)
```

```
[1] 0.08668182
```

```
 vader_ccss_summary <- vader_ccss %>%
   mutate(sentiment = case_when(
     compound >= 0.05 ~ "positive",
     compound <= -0.05 ~ "negative",
     TRUE ~ "neutral"
   )) %>%
   count(sentiment)
```

3.4.4 Results and Discussion

RQ1: Who are the "transmitters," "transceivers," and "transcenders" in the Common Core Twitter network?

- **Transmitters (high out-degree):**
  The user `SumayLu` stands out as the top transmitter, initiating 8 outgoing ties (mentions/retweets), followed by `DouglasHolt...` (5), `WEquilSchool` (3), `fluttbot` (3), and `JoeWEquil` (2). These users are the most active in broadcasting or mentioning others within the network.
- **Transceivers (high in-degree):**
  The most-mentioned users are `WEquilSchool` and `SumayLu` (in-degree = 3), `JoeWEquil` (2), `Tech4Learni...` (2), and `LASER_Insti...` (2). These individuals receive the most attention from other actors—potential focal points in conversations.
- **Transcenders (high in-degree *and* out-degree):**
  Only two users—`WEquilSchool` (in-degree = 3, out-degree = 3) and `SumayLu` (in-degree = 3, out-degree = 8)—simultaneously act as hubs for both sending and receiving communication. These "bridging" actors may serve as key facilitators or connectors in the discourse.

RQ2: What subgroups or factions exist in the network?

- **Component analysis** shows a **fragmented network**:
  - There are **14 weakly connected components**, the largest containing 14 users, and several small groups or dyads (many with just 2–3 members).
  - This fragmentation suggests limited overall cohesion, with multiple parallel or isolated conversations occurring.
- **Clique analysis** reveals:
  - Four cliques (fully connected subgroups) of size 3 or 4—e.g., one 4-person clique involving nodes 3, 4, 5, and 6, and several overlapping 3-person cliques. This indicates pockets of tight-knit interaction, but such groups are rare relative to the size of the network.
- **Community detection** using edge betweenness identifies 16 subgroups, generally aligning with the component structure. The largest subgroup has 10 members, with most others much smaller.

RQ3: What is the overall sentiment in the network?

- **VADER sentiment analysis** of tweet content yields:
  - An average sentiment score (`compound`) of **0.09** (slightly positive), indicating that, despite the policy controversy, the sampled tweets were, on balance, more positive than negative.
  - When tweets are classified into categories:
    - A mix of *positive*, *neutral*, and *negative* tweets is observed, with positive tweets slightly outnumbering negatives.
  - This suggests the debate, at least in this time slice, included advocacy and constructive dialogue, not only criticism or negativity.

RQ4: What other patterns of communication (e.g., centrality, clique formation, isolates) characterize this network?

- **Centrality Patterns:**

  The network displays a classic "star" structure in its largest component. Two users, `SumayLu` and `WEquilSchool`, stand out with high out-degree and in-degree centrality, respectively. Most other users have very low degree values (often 0 or 1), meaning they are peripheral, engaging in few interactions.

  - *Transmitters* (high out-degree): e.g., `SumayLu` (8 outgoing ties), `DouglasHolt...` (5).
  - *Transceivers* (high in-degree): e.g., `WEquilSchool`, `SumayLu` (both in-degree = 3).
  - *Transcenders* (both high in- and out-degree): rare—only `WEquilSchool` and `SumayLu` meet this criterion in this sample.

- **Clique Formation:**

  Clique analysis revealed **4 cliques** (fully connected subgroups) of size 3 or more, with one larger clique (size 4) and several overlapping smaller cliques. However, cliques are rare and limited in size—most communication occurs outside of dense subgroups.

- **Isolates and Components:**

  The network has **14 weak components**—many of them tiny. Several users are **isolates** or part of isolated dyads and triads, meaning they are disconnected from the main conversation or only loosely connected. This points to a lack of broad, network-wide cohesion.

- **Community Structure:**

  Edge betweenness community detection found 16 subgroups, typically matching up with the component structure: most subgroups are very small (2–3 nodes), while the largest subgroup consists of 10 users.

- **Summary:**

  Communication in this network is characterized by:

  - **Strong centralization** around a small number of users (hubs);
  - **Sparse and fragmented structure** with many small, disconnected components;
  - **Limited clique formation**—pockets of tightly connected users exist but are rare;
  - **Numerous isolates**—users who are only weakly or not at all connected to the core discussion.

Discussion

This analysis of the Common Core Twitter conversation reveals a **sparse and fragmented network structure**. The debate is distributed across many small subgroups, with only one moderately sized component (14 members). Within this landscape:

- **Key actors** such as `SumayLu` and `WEquilSchool` serve as both broadcasters and focal points of attention ("transcenders"), but most users are peripheral, interacting minimally.
- **Cliques and communities** are few and small, underscoring the lack of broad cohesion. Most interactions happen within micro-groups rather than across the entire network.
- **Sentiment** is, perhaps surprisingly, slightly positive on average. This may reflect the presence of advocacy groups, promotional messaging, or simply a lack of highly negative engagement during the observed period.

Implications:
The findings illustrate classic social network phenomena in online policy debate: - Most users are only lightly involved, and only a select few drive discussion or receive significant attention. - Communication is siloed, with many small isolated groups and minimal bridging between them. - Sentiment analysis offers nuance: while public debates may be assumed to be contentious, the prevailing tone can still be balanced or even positive in certain time slices.

For researchers and practitioners, this means that: - Identifying and engaging "transcenders" is essential for bridging subgroups and spreading information. - Interventions or outreach should consider the network's fragmentation—broader influence may require engaging multiple small groups individually rather than targeting a single "core." - Combining SNA with text/sentiment analysis gives a fuller picture: not just who is talking, but *how* and *with what tone*.

Future analysis could track changes in sentiment and connectivity over time, or compare subgroups for differences in message tone and network position.

References

- Supovitz, J., Daly, A.J., del Fresno, M., & Kolouch, C. (2017). *#commoncore Project.* Retrieved from http://www.hashtagcommoncore.com
- Carolan, B.V. (2014). *Social Network Analysis and Education: Theory, Methods & Applications.* Sage.
- Silge, J., & Robinson, D. (2017). *Text Mining with R: A Tidy Approach.* O'Reilly.

Numeric Data

Abstract: This section reviews how to access data that is primarily numeric/quantitative in nature, but from a different source and of a different nature than the data typically used by social scientists. Example data sets include international or national large-scale assessments (e.g., PISA, NAEP IPEDS) and data from digital technologies (e.g., log-trace data from Open University Learning Analytics Dataset (OULAD)).

4.1 Overview

In social science research, data is traditionally sourced from small-scale surveys, experiments, or qualitative studies. However, the rise of big data offers researchers opportunities to explore numeric and quantitative datasets of unprecedented scale and variety. This chapter discusses how to access and analyze large-scale datasets like international assessments (e.g., PISA, NAEP) and digital log-trace data (e.g., Open University Learning Analytics Dataset (OULAD)). These secondary data sources enable novel research questions and methods, particularly when paired with machine learning and statistical modeling approaches.

4.2 Accessing Big data (Broadening the Horizon)

4.2.1 Big Data

Accessing PISA Data

The Programme for International Student Assessment (PISA) is a widely used dataset for large-scale educational research. It assesses 15-year-old students' knowledge and skills in reading, mathematics, and science across multiple countries. Researchers can access PISA data through various methods:

1. Direct Download from the Official Website

The OECD provides direct access to PISA data files via its official website. Researchers can download data for specific years and cycles. Data files are typically provided in `.csv` or `.sav` (SPSS) formats, along with detailed documentation.

- **Steps to Access PISA Data from the OECD Website:**
  1. Visit the OECD PISA website.
  2. Navigate to the "Data" section.
  3. Select the desired assessment year (e.g., 2022).
  4. Download the data and accompanying codebooks.

2. Using the `OECD` R Package

The `OECD` R package provides a direct interface to download and explore datasets published by the OECD, including PISA.

- **Steps to Use the `OECD` Package:**
  1. Install and load the `OECD` package.
  2. Use the `getOECD()` function to fetch PISA data.

```r
# Install and load the OECD package
install.packages("OECD")
library(OECD)

# Fetch PISA data for the 2018 cycle
pisa_data <- getOECD("pisa", years = "2022")

# Display a summary of the data
summary(pisa_data)
```

3. Using the `Edsurvey` R Package

The `Edsurvey` package is designed specifically for analyzing large-scale assessment data, including PISA. It allows for complex statistical modeling and supports handling weights and replicate weights used in PISA.

- **Steps to Use the `Edsurvey` Package:**
  1. Install and load the `Edsurvey` package.
  2. Download the PISA data from the OECD website and provide the path to the `.sav` files.

3. Load the data into R using `readPISA()`.

```r
# Install and load the Edsurvey package
install.packages("Edsurvey")
library(Edsurvey)

# Read PISA data from a local file
pisa_data <- readPISA("path/to/PISA2022Student.sav")

# Display the structure of the dataset
str(pisa_data)
```

Comparison of Methods

Method	Advantages	Disadvantages
Direct Download	Full access to all raw data and documentation.	Requires manual processing and cleaning.
`OECD` Package	Easy to use for downloading specific datasets.	Limited to OECD-published formats.
`Edsurvey` Package	Supports advanced statistical analysis and weights.	Requires additional setup and dependencies.

Accessing IPEDS Data

The Integrated Postsecondary Education Data System (IPEDS) is a comprehensive source of data on U.S. colleges, universities, and technical and vocational institutions. It provides data on enrollments, completions, graduation rates, faculty, finances, and more. Researchers and policymakers widely use IPEDS data to analyze trends in higher education.

There are several ways to access IPEDS data, depending on the user's needs and technical proficiency.

1. Direct Download from the NCES Website

The most straightforward way to access IPEDS data is by downloading it directly from the National Center for Education Statistics (NCES) website.

Steps to Access IPEDS Data:

1. Visit the IPEDS Data Center.
2. Click on "Use the Data" and navigate to the "Download IPEDS Data Files" section.
3. Select the desired data year and survey component (e.g., Fall Enrollment, Graduation Rates).
4. Download the data files, typically provided in `.csv` or `.xls` format, along with accompanying codebooks.

2. Using the `ipeds` R Package

The `ipeds` R package simplifies downloading and analyzing IPEDS data directly from R by connecting to the NCES data repository.

Steps to Use the `ipeds` Package:

1. Install and load the `ipeds` package.
2. Use the `download_ipeds()` function to fetch data for specific survey components and years.

```r
# Install and load the ipeds package
install.packages("ipeds")
library(ipeds)

# Download IPEDS data for completions in 2021
ipeds_data <- download_ipeds("C", year = 2021)

# View the structure of the downloaded data
str(ipeds_data)
```

3. Using the `tidycensus` R Package

The `tidycensus` package, while primarily designed for Census data, can access specific IPEDS data linked to educational institutions.

Steps to Use the `tidycensus` Package:

1. Install and load the `tidycensus` package.
2. Set up a Census API key to access the data.
3. Query IPEDS data for specific institution-level information.

```r
# Install and load the tidycensus package
install.packages("tidycensus")
library(tidycensus)

# Set Census API key (replace with your actual key)
census_api_key("your_census_api_key")

# Fetch IPEDS-related data (e.g., institution information)
ipeds_institutions <- get_acs(
  geography = "place",
  variables = "B14002_003",
  year = 2021,
  survey = "acs5"
)

# View the first few rows
head(ipeds_institutions)
```

4. Using Online Tools

IPEDS provides several online tools for querying and visualizing data without requiring programming skills.

Common Tools:

- **IPEDS Data Explorer**: Enables users to query and export customized datasets.
- **Trend Generator**: Allows users to visualize trends in key metrics over time.
- **IPEDS Use the Data**: Simplified tool for accessing pre-compiled datasets.

Steps to Use the IPEDS Data Explorer:

1. Visit the IPEDS Data Explorer.
2. Select variables of interest, such as institution type, enrollment size, or location.
3. Filter results by years, institution categories, or other criteria.
4. Export the results as a `.csv` or `.xlsx` file.

Comparison of Methods

Method	Advantages	Disadvantages
Direct Download	Full access to raw data and documentation.	Requires manual data preparation and cleaning.
`ipeds` Package	Automated access to specific components.	Limited flexibility for customized queries.
`tidycensus` Package	Allows integration with Census and ACS data.	Requires API setup and advanced R skills.
Online Tools	User-friendly and suitable for non-coders.	Limited to predefined queries and exports.

Accessing Open University Learning Analytics Dataset (OULAD)

The **Open University Learning Analytics Dataset (OULAD)** is a publicly available dataset designed to support research in educational data mining and learning analytics. It includes student demographics, module information, interactions with the virtual learning environment (VLE), and assessment scores.

Steps to Access OULAD Data

Visit the OULAD Repository**

The dataset is hosted on the Open University's Analytics Project. To access the data: 1. Navigate to the website. 2. Download the dataset as a `.zip` file. 3. Extract the `.zip` file to a local directory.

The dataset contains multiple CSV files: - `studentInfo.csv`: Student demographics and performance data. - `studentVle.csv`: Interactions with the VLE. - `vle.csv`: Details of learning resources. - `studentAssessment.csv`: Assessment scores.

Loading OULAD Data in R

Once the data is downloaded and extracted, follow these steps to load and access it in R:

Step 1: Install Required Packages

```r
# Install necessary packages
install.packages(c("readr", "dplyr"))
```

Step 2: Load Data

Use the `readr` package to read the CSV files into R.

```
# Load required libraries
library(readr)

# Define the path to the OULAD data
data_path <- "path/to/OULAD/"

# Load individual CSV files
student_info <- read_csv(file.path(data_path, "studentInfo.csv"))
student_vle <- read_csv(file.path(data_path, "studentVle.csv"))
vle <- read_csv(file.path(data_path, "vle.csv"))
student_assessment <- read_csv(file.path(data_path,
"studentAssessment.csv"))
```

Step 3: Preview the Data

Inspect the structure and contents of the datasets.

```
# View the first few rows of student info
head(student_info)

# Check the structure of the student VLE data
str(student_vle)
```

4.2.2 Learning Analytics

What is Learning Analytics?

Learning Analytics (LA) refers to the measurement, collection, analysis, and reporting of data about learners and their contexts. The primary goal of LA is to understand and improve learning processes by identifying patterns, predicting outcomes, and providing actionable insights to educators, institutions, and learners.

Key features of LA include: - **Data Collection**: Gathering information from digital platforms such as learning management systems (LMS) or external assessments. - **Analysis**: Using machine learning, statistical methods, or visualization tools to reveal trends and patterns. - **Applications**: Supporting personalized learning, enhancing institutional decision-making, and improving curriculum design.

Applications of Learning Analytics in Big Data

Learning analytics can be applied to large-scale educational datasets like **PISA**, **IPEDS**, and **OULAD** to uncover trends, predict outcomes, and guide interventions.

1. PISA Data and Learning Analytics

- **What it offers**: Insights into international student performance in reading, math, and science, combined with contextual variables (e.g., socio-economic status).
- **LA Applications**:
  - Identifying key factors influencing performance across countries.
  - Predicting the impact of ICT use on student achievement.
  - Segmenting students into performance clusters for targeted interventions.

2. IPEDS Data and Learning Analytics

- **What it offers**: U.S. institutional-level data on enrollment, graduation rates, tuition, and financial aid.
- **LA Applications**:
  - Analyzing trends in student demographics across institutions.
  - Predicting enrollment patterns based on historical data.
  - Benchmarking institutions to inform policymaking and funding decisions.

3. OULAD and Learning Analytics

- **What it offers**: Rich data on student engagement with virtual learning environments (VLE), assessment scores, and demographic information.
- **LA Applications**:
  - Tracking student interactions with learning resources to predict course completion.
  - Modeling the relationship between VLE usage and final grades.
  - Detecting early warning signs for at-risk students based on engagement metrics.

Why Learning Analytics Matters

The integration of **Learning Analytics** with big data enables researchers and practitioners to: - **Personalize Learning**: Tailor educational experiences to meet individual needs. - **Improve Retention**: Identify at-risk learners and implement timely interventions. - **Enhance Decision-Making**: Provide evidence-based recommendations for curriculum and policy adjustments.

By leveraging datasets like PISA, IPEDS, and OULAD, learning analytics can help bridge the gap between raw data and actionable insights, fostering a more equitable and effective educational landscape.

Supervised Learning in Learning Analytics

Machine Learning, particularly **Supervised Learning**, has become a cornerstone of Learning Analytics. Supervised learning models are trained on labeled datasets, where input features are mapped to known outcomes, enabling the prediction of new, unseen data.

Key Concepts in Supervised Learning

- **Definition**
  Supervised Learning is a subset of Machine Learning focused on learning a mapping between input variables (features) and output variables (labels or outcomes). Models trained on labeled data can predict outcomes for new data points.
- **Common Algorithms**
  - Linear Regression
  - Logistic Regression
  - Decision Trees and Random Forests
  - Neural Networks
- **Applications in Education**
  Supervised learning is particularly effective in Learning Analytics for predicting:
  - Student performance
  - Dropout risks
  - Enrollment trends
  - Course completion rates

Applications of Supervised Learning with Big Data

1. PISA Data and Supervised Learning

- **Goal**: Use demographic and contextual features to predict student performance in mathematics, reading, or science.
- **Example**: Train a linear regression model to identify the relationship between socioeconomic status and test scores.

2. IPEDS Data and Supervised Learning

- **Goal**: Develop models to predict institutional enrollment rates based on financial aid, demographics, and program offerings.
- **Example**: Use logistic regression to forecast whether a student is likely to enroll based on financial aid eligibility.

3. OULAD Data and Supervised Learning

- **Goal**: Predict student outcomes (e.g., pass/fail) based on engagement metrics like forum participation and assignment submissions.
- **Example**: Train a random forest model to classify students as "at-risk" or "not at-risk" based on weekly interaction data.

Choosing the Right Supervised Learning Approach

When applying supervised learning in Learning Analytics: 1. **Define the Goal**: Clearly articulate the outcome you want to predict (e.g., performance, enrollment, or engagement). 2. **Select an Algorithm**: Choose an appropriate model based on the data and prediction task. - For continuous outcomes, use regression models. - For categorical outcomes, use classification models like logistic regression or random forests. 3. **Feature Engineering**: Select and preprocess relevant features (e.g., attendance, demographics, assignment scores) to improve model accuracy. 4. **Evaluate Model Performance**: Use metrics such as accuracy, precision, recall, or R-squared to assess model effectiveness.

Integrating supervised learning techniques into Learning Analytics, researchers and practitioners can leverage big data to make data-driven predictions and decisions, ultimately enhancing educational outcomes.

4.3 Logistic Regression ML

4.3.1 Purpose + CASE

Purpose

Logistic regression is a supervised learning technique widely used for binary classification tasks. It models the probability of an event occurring (e.g., success vs. failure) based on a set of predictor variables. Logistic regression is particularly effective in educational research for predicting outcomes such as retention, enrollment, or graduation rates.

CASE: Predicting Graduation Rates

This case study is based on IPEDS data and inspired by Zong and Davis (2022). We predict graduation rates as a binary outcome (`good_grad_rate`) using institutional features such as total enrollment, admission rate, tuition fees, and average instructional staff salary.

4.3.2 Sample Research Questions (RQs)

- **RQ A**: What institutional factors are associated with high graduation rates in U.S. four-year universities?
- **RQ B**: How accurately can we predict high graduation rates using institutional features with supervised machine learning?

4.3.3 Analysis

Loading Required Packages

We load necessary R packages for data wrangling, cleaning, and modeling.

```
# Load necessary libraries for data cleaning, wrangling, and modeling
library(tidyverse)  # For data manipulation and visualization
library(tidymodels) # For machine learning workflows
library(janitor)    # For cleaning variable names
```

Loading and Cleaning Data

We read the IPEDS dataset and clean column names for easier handling.

```
# Read in IPEDS data from CSV file
ipeds <- read_csv("data/ipeds-all-title-9-2022-data.csv")

# Clean column names for consistency and usability
ipeds <- janitor::clean_names(ipeds)
```

Data Wrangling

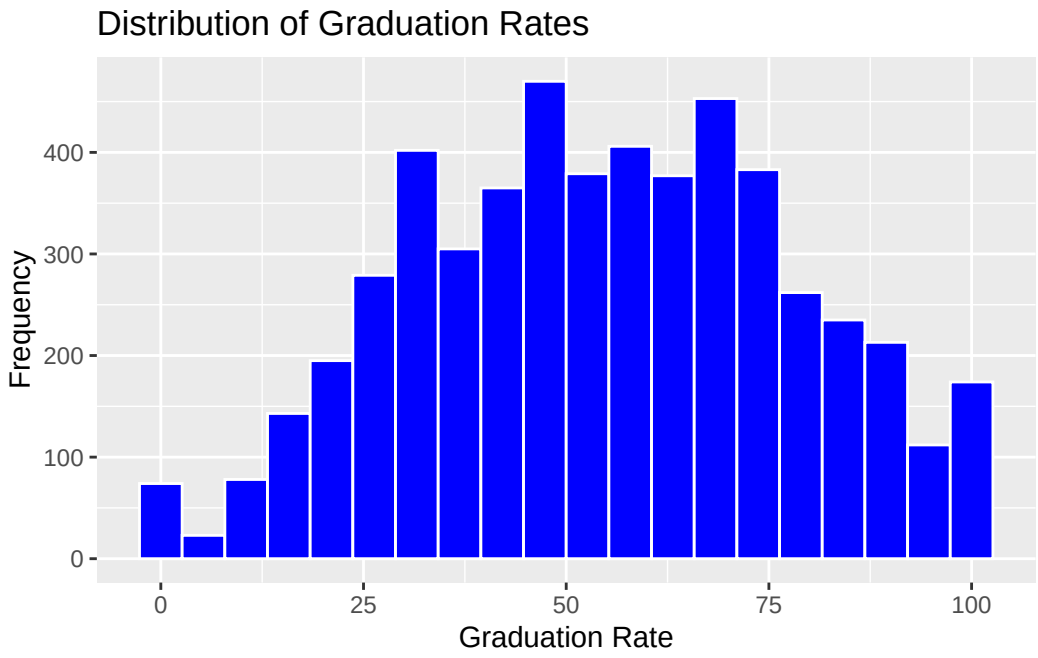
Select relevant variables, filter the dataset, and create the dependent variable `good_grad_rate`.

```r
# Select and rename key variables; filter relevant institutions
ipeds <- ipeds %>%
  select(
    name = institution_name,                    # Institution name
    total_enroll = drvef2022_total_enrollment, # Total enrollment
    pct_admitted = drvadm2022_percent_admitted_total, # Admission
    percentage
    tuition_fees = drvic2022_tuition_and_fees_2021_22, # Tuition fees
    grad_rate = drvgr2022_graduation_rate_total_cohort, # Graduation
    rate
    percent_fin_aid =
    sfa2122_percent_of_full_time_first_time_undergraduates_awarded_any_financial_aid,
    # Financial aid
    avg_salary =
    drvhr2022_average_salary_equated_to_9_months_of_full_time_instructional_staff_all_rank
    # Staff salary
  ) %>%
  filter(!is.na(grad_rate)) %>% # Remove rows with missing graduation
  rates
  mutate(
    # Create binary dependent variable for high graduation rates
    good_grad_rate = if_else(grad_rate > 62, 1, 0),
    good_grad_rate = as.factor(good_grad_rate) # Convert to factor
  )
```

Exploratory Data Analysis (EDA)

Visualize the distribution of the graduation rate.

```r
# Plot a histogram of graduation rates
ipeds %>%
  ggplot(aes(x = grad_rate)) +
  geom_histogram(bins = 20, fill = "blue", color = "white") +
  labs(
    title = "Distribution of Graduation Rates",
    x = "Graduation Rate",
    y = "Frequency"
  )
```



Logistic Regression Model

Fit a logistic regression model to predict high graduation rates.

```
# Fit logistic regression model
m1 <- glm(
  good_grad_rate ~ total_enroll + pct_admitted + tuition_fees +
percent_fin_aid + avg_salary,
  data = ipeds,
  family = "binomial" # Specify logistic regression for binary outcome
)

# View model summary
summary(m1)
```

```
Call:
glm(formula = good_grad_rate ~ total_enroll + pct_admitted +
    tuition_fees + percent_fin_aid + avg_salary, family = "binomial",
    data = ipeds)

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
```

```
(Intercept)     -8.742e-01  6.237e-01  -1.402    0.161
total_enroll     3.350e-05  7.880e-06   4.251 2.13e-05 ***
pct_admitted    -1.407e-02  3.519e-03  -3.997 6.40e-05 ***
tuition_fees     6.952e-05  4.965e-06  14.003  < 2e-16 ***
percent_fin_aid -2.960e-02  5.652e-03  -5.237 1.64e-07 ***
avg_salary       2.996e-05  3.870e-06   7.740 9.91e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2277  on 1706  degrees of freedom
Residual deviance: 1632  on 1701  degrees of freedom
  (3621 observations deleted due to missingness)
AIC: 1644

Number of Fisher Scoring iterations: 5
```

Supervised ML Workflow

Use the `tidymodels` framework to build a machine learning model.

```r
# Define recipe for the model (preprocessing steps)
my_rec <- recipe(good_grad_rate ~ total_enroll + pct_admitted +
tuition_fees + percent_fin_aid + avg_salary, data = ipeds)

# Specify logistic regression model with tidymodels
my_mod <- logistic_reg() %>%
  set_engine("glm") %>%         # Use glm engine for logistic regression
  set_mode("classification")    # Specify binary classification task

# Create workflow to connect recipe and model
my_wf <- workflow() %>%
  add_recipe(my_rec) %>%
  add_model(my_mod)

# Fit the logistic regression model
fit_model <- fit(my_wf, ipeds)

# Generate predictions on the dataset
predictions <- predict(fit_model, ipeds) %>%
```

```r
  bind_cols(ipeds) # Combine predictions with original data

# Calculate and display accuracy
my_accuracy <- predictions %>%
  metrics(truth = good_grad_rate, estimate = .pred_class) %>%
  filter(.metric == "accuracy")

my_accuracy
```

```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.800
```

4.3.4 Results and Discussions

Logistic Regression Model (RQ A)

The logistic regression model was fitted to predict whether a university achieves a "good" graduation rate (i.e., graduation rate $> 62\%$) based on several institutional features. The model output is summarized below:

- **Coefficients & Significance**:
  - **total_enroll**: Estimate = 3.35e-05, $z = 4.251$, p = 2.13e-05
    Interpretation: As total enrollment increases, the probability of a high graduation rate increases.
  - **pct_admitted**: Estimate = -1.407e-02, $z = -3.997$, p = 6.40e-05
    Interpretation: Higher admission percentages are associated with a lower likelihood of achieving a high graduation rate.
  - **tuition_fees**: Estimate = 6.952e-05, $z = 14.003$, p $<$ 2e-16
    Interpretation: Higher tuition fees are strongly associated with higher graduation rates.
  - **percent_fin_aid**: Estimate = -2.960e-02, $z = -5.237$, p = 1.64e-07
    Interpretation: A higher percentage of students receiving financial aid is associated with a lower probability of a good graduation rate.
  - **avg_salary**: Estimate = 2.996e-05, $z = 7.740$, p = 9.91e-15
    Interpretation: Higher average salaries for instructional staff are positively associated with high graduation rates.
- **Model Fit Statistics**:

- **Null Deviance**: 2277 (on 1706 degrees of freedom)
- **Residual Deviance**: 1632 (on 1701 degrees of freedom)
- **AIC**: 1644
- Note: 3621 observations were deleted due to missing values.

Overall, the regression model demonstrates that several institutional factors are statistically significant predictors of graduation rates. In particular, tuition fees and avg_salary have a strong positive effect, while pct_admitted and percent_fin_aid show negative associations.

Supervised ML Workflow Results (RQ B)

Using the tidymodels framework, we built a logistic regression model as part of a supervised machine learning workflow. The performance metric obtained is as follows:

- **Accuracy**: 80.02%

This indicates that the machine learning model correctly classified approximately 80% of the institutions as having either a good or not good graduation rate, based on the selected predictors.

Overall Discussion

- **Similarities between Approaches**:
  - Both the traditional logistic regression and the tidymodels workflow identified key predictors that influence graduation rates, such as total enrollment, admission percentage, tuition fees, financial aid percentage, and average staff salary.
  - Each approach provides valuable insights: the regression model offers detailed coefficient estimates and significance levels, while the tidymodels workflow emphasizes predictive accuracy.
- **Differences between Approaches**:
  - **Interpretability vs. Predictive Performance**: The logistic regression output delivers interpretability through its coefficients and p-values, allowing us to understand the direction and magnitude of the relationships. In contrast, the supervised ML workflow focuses on achieving a robust predictive performance, evidenced by an 80% accuracy.
  - **Handling of Data**: The traditional regression model summarizes the relationship between variables, whereas the ML workflow integrates data pre-processing, modeling, and validation into a cohesive framework.

In summary, our analyses indicate that institutional factors, particularly tuition fees and staff salaries, play a significant role in predicting graduation outcomes. The supervised ML approach, with an accuracy of around 80%, confirms the model's practical utility in classifying institutions based on graduation performance. Both methods complement each other, providing a comprehensive understanding of the underlying dynamics that drive graduation rates in higher education.

4.4 Random Forests ML on Interactions Data

In this section, we explore a more sophisticated supervised learning approach—Random Forests—to model student interactions data from the Open University Learning Analytics Dataset (OULAD). Building on our earlier work with logistic regression and evaluation metrics, this case study examines whether a random forest model can improve predictive performance when leveraging clickstream data from the virtual learning environment (VLE).

4.4.1 Purpose + CASE

Purpose

Random Forests is an ensemble learning method that builds multiple decision trees and aggregates their results to improve prediction accuracy and control over-fitting. It is particularly well suited for complex, high-dimensional data such as student interaction (clickstream) data. This approach not only provides robust predictions but also offers insights into variable importance, helping us understand which features most influence student outcomes.

CASE

Inspired by research on digital trace data (e.g., Rodriguez et al., 2021; Bosch, 2021), this case study uses pre-processed interactions data from OULAD. In our analysis, we focus on predicting whether a student will pass the course (a binary outcome) based on engineered features derived from clickstream data. These features include the total number of clicks (`sum_clicks`), summary statistics (mean and standard deviation of clicks), and linear trends over time (slope and intercept from clickstream patterns).

4.4.2 Sample Research Questions

- **RQ1:** How accurately can a random forest model predict whether a student will pass a course using interactions data from OULAD?

- **RQ2:** Which interaction-based features (e.g., total clicks, click stream slope) are most important in predicting student outcomes?
- **RQ3:** How does the use of cross-validation (e.g., v-fold CV) influence the stability and generalizability of the random forest model on interactions data?

4.4.3 Analysis

Loading Required Packages

```
# Load necessary libraries for data manipulation and modeling
library(tidyverse)      # Data wrangling and visualization
library(janitor)        # Cleaning variable names
library(tidymodels)     # Modeling workflow
library(ranger)         # Random forest implementation
library(vip)            # Variable importance plots
```

Loading and Preparing the Data

We load the pre-filtered interactions data from OULAD along with a students-and-assessments file, then join them to create a complete dataset for modeling.

```
# Load the interactions data (filtered for the first one-third of the
semester)
interactions <- read_csv("data/oulad-interactions-filtered.csv")

# Load the students and assessments data
students_and_assessments <-
read_csv("data/oulad-students-and-assessments.csv")

# Create cut-off dates based on assessments data (using first quantile
as intervention point)
assessments <- read_csv("data/oulad-assessments.csv")

# Create cut-off dates based on assessments data using the correct date
column 'date_submitted'
code_module_dates <- assessments %>%
    group_by(code_module, code_presentation) %>%
    summarize(quantile_cutoff_date = quantile(date_submitted, probs =
    0.25, na.rm = TRUE), .groups = 'drop')
```

```r
# Join interactions with the cutoff dates and filter
interactions_joined <- interactions %>%
    left_join(code_module_dates, by = c("code_module",
    "code_presentation"))


interactions_joined <- interactions_joined %>%
    select(-quantile_cutoff_date.x) %>%
    rename(quantile_cutoff_date = quantile_cutoff_date.y)



# Filter interactions to include only those before the cutoff date
interactions_filtered <- interactions_joined %>%
    filter(date < quantile_cutoff_date)

# Summarize interactions: total clicks, mean and standard deviation
interactions_summarized <- interactions_filtered %>%
    group_by(id_student, code_module, code_presentation) %>%
    summarize(
      sum_clicks = sum(sum_click),
      sd_clicks = sd(sum_click),
      mean_clicks = mean(sum_click)
    )

# (Optional) Further feature engineering: derive linear slopes from
clickstream
fit_model <- function(data) {
    tryCatch(
        {
            model <- lm(sum_click ~ date, data = data)
            tidy(model)
        },
        error = function(e) { tibble(term = NA, estimate = NA, std.error
        = NA, statistic = NA, p.value = NA) },
        warning = function(w) { tibble(term = NA, estimate = NA,
        std.error = NA, statistic = NA, p.value = NA) }
    )
}
```

```r
interactions_slopes <- interactions_filtered %>%
    group_by(id_student, code_module, code_presentation) %>%
    nest() %>%
    mutate(model = map(data, fit_model)) %>%
    unnest(model) %>%
    ungroup() %>%
    select(code_module, code_presentation, id_student, term, estimate)
    %>%
    filter(!is.na(term)) %>%
    pivot_wider(names_from = term, values_from = estimate) %>%
    mutate_if(is.numeric, round, 4) %>%
    rename(intercept = `(Intercept)`, slope = date)

# Join summarized clicks and slopes features
interactions_features <- left_join(interactions_summarized,
interactions_slopes, by = c("id_student", "code_module",
"code_presentation"))

# Finally, join with students_and_assessments to get the outcome
variable
students_assessments_and_interactions <-
left_join(students_and_assessments, interactions_features, by =
c("id_student", "code_module", "code_presentation"))

# Ensure outcome variable 'pass' is a factor
students_assessments_and_interactions <-
students_assessments_and_interactions %>%
    mutate(pass = as.factor(pass))

# Optional: Inspect the final dataset
students_assessments_and_interactions %>%
    skimr::skim()
```

Table 3: Data summary

Name	Piped data
Number of rows	32593
Number of columns	22
Column type frequency:	
character	8

factor	1
numeric	13
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
code_module	0	1	3	3	0	7	0
code_presentation	0	1	5	5	0	4	0
gender	0	1	1	1	0	2	0
region	0	1	5	20	0	13	0
highest_education	0	1	15	27	0	5	0
age_band	0	1	4	5	0	3	0
disability	0	1	1	1	0	2	0
final_result	0	1	4	11	0	4	0

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
pass	0	1	FALSE	2	0: 20232, 1: 12361

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
id_student	0	1.00	706687.67	549167.37	3733.00	508573.00	590310.00	644453.00	2716795.00	
imd_band	4627	0.86	5.62	2.73	1.00	4.00	6.00	8.00	10.00	
num_of_prev_attempts	0	1.00	0.16	0.48	0.00	0.00	0.00	0.00	6.00	
studied_credits	0	1.00	79.76	41.07	30.00	60.00	60.00	120.00	655.00	
module_presentation_length	0	1.00	256.01	13.18	234.00	241.00	262.00	268.00	269.00	
date_registration	45	1.00	-69.41	49.26	-322.00	-100.00	-57.00	-29.00	167.00	
date_unregistration	22521	0.31	49.76	82.46	-365.00	-2.00	27.00	109.00	444.00	
mean_weighted_score	7958	0.76	544.70	381.39	0.00	160.00	610.00	875.00	1512.00	
sum_clicks	3495	0.89	474.93	572.89	1.00	128.00	295.50	604.00	10712.00	
sd_clicks	3753	0.88	4.91	5.51	0.00	2.37	3.72	6.44	560.24	
mean_clicks	3495	0.89	3.19	1.30	1.00	2.33	2.95	3.82	47.12	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
intercept	3640	0.89	3.04	4.61	-585.59	2.15	2.80	3.66	130.83	
slope	4441	0.86	0.01	0.22	-12.17	-0.01	0.01	0.03	20.12	

Creating the Model Recipe

We build a recipe that includes the engineered features from interactions data along with other predictors from the students data.

```
my_rec2 <- recipe(pass ~ disability +
                      date_registration +
                      gender +
                      code_module +
                      mean_weighted_score +
                      sum_clicks + sd_clicks + mean_clicks +
                      intercept + slope,
                  data = students_assessments_and_interactions) %>%
    step_dummy(disability) %>%
    step_dummy(gender) %>%
    step_dummy(code_module) %>%
    step_impute_knn(mean_weighted_score) %>%
    step_impute_knn(sum_clicks) %>%
    step_impute_knn(sd_clicks) %>%
    step_impute_knn(mean_clicks) %>%
    step_impute_knn(intercept) %>%
    step_impute_knn(slope) %>%
    step_impute_knn(date_registration) %>%
    step_normalize(all_numeric_predictors())
```

Specifying the Model and Workflow

We use a random forest model via the `ranger` engine and set up our workflow.

```
# Specify random forest model
my_mod2 <- rand_forest() %>%
    set_engine("ranger", importance = "impurity") %>%
```

```
    set_mode("classification")

# Create workflow to bundle the recipe and model
my_wf2 <- workflow() %>%
    add_recipe(my_rec2) %>%
    add_model(my_mod2)
```

Resampling and Model Fitting

We perform cross-validation (v-fold CV) to estimate model performance.

```
# Create 4-fold cross-validation on training data
vfcv <- vfold_cv(data = students_assessments_and_interactions, v = 4,
strata = pass)

# Specify metrics: accuracy, sensitivity, specificity, ppv, npv, and
Cohen's kappa
class_metrics <- metric_set(accuracy, sensitivity, specificity, ppv,
npv, kap)

# Fit the model using resampling
fitted_model_resamples <- fit_resamples(my_wf2, resamples = vfcv,
metrics = class_metrics)

# Collect and display metrics
collect_metrics(fitted_model_resamples)
```

```
# A tibble: 6 x 6
  .metric     .estimator  mean     n  std_err .config
  <chr>       <chr>      <dbl> <int>    <dbl> <chr>
1 accuracy    binary     0.672     4 0.000561 Preprocessor1_Model1
2 kap         binary     0.267     4 0.000456 Preprocessor1_Model1
3 npv         binary     0.592     4 0.00190  Preprocessor1_Model1
4 ppv         binary     0.704     4 0.000511 Preprocessor1_Model1
5 sensitivity binary     0.816     4 0.00288  Preprocessor1_Model1
6 specificity binary     0.437     4 0.00335  Preprocessor1_Model1
```

Final Model Fit and Evaluation

Finally, we fit the model on the full training set (using last_fit) and evaluate its predictions on the test set.

```
# Split data into training and testing sets (e.g., 33% for testing)
set.seed(20230712)
train_test_split <- initial_split(students_assessments_and_interactions,
prop = 0.67, strata = pass)
data_train <- training(train_test_split)
data_test <- testing(train_test_split)

# Fit final model on the training set and evaluate on the test set
final_fit <- last_fit(my_wf2, train_test_split, metrics = class_metrics)

# Collect and display final metrics
collect_metrics(final_fit)
```

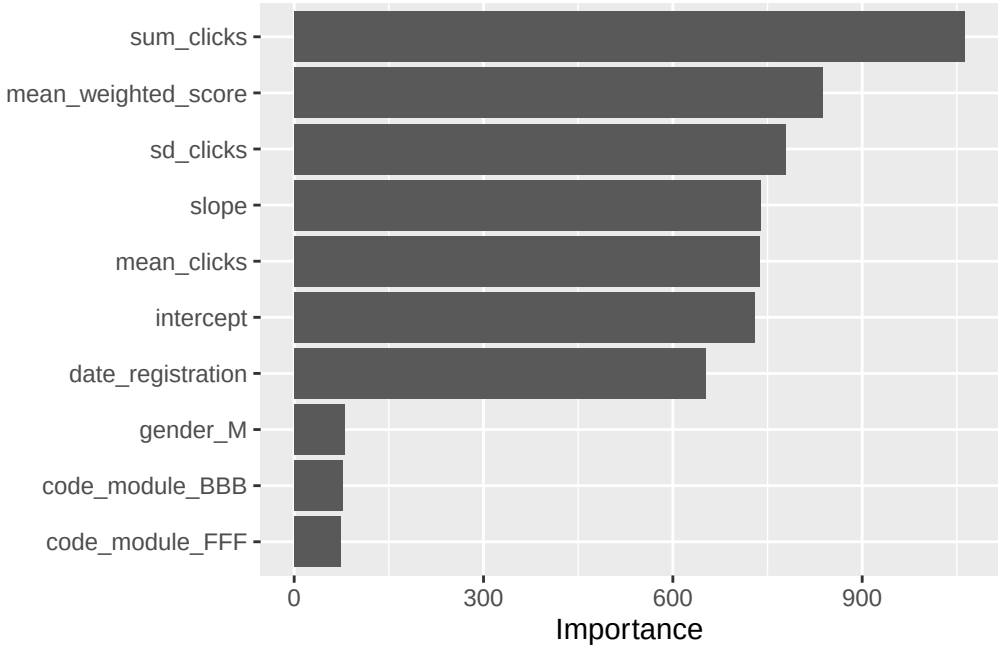
```
# A tibble: 6 x 4
  .metric     .estimator .estimate .config
  <chr>       <chr>          <dbl> <chr>
1 accuracy    binary         0.665 Preprocessor1_Model1
2 sensitivity binary         0.815 Preprocessor1_Model1
3 specificity binary         0.419 Preprocessor1_Model1
4 ppv         binary         0.697 Preprocessor1_Model1
5 npv         binary         0.580 Preprocessor1_Model1
6 kap         binary         0.247 Preprocessor1_Model1
```

```
# Generate and display a confusion matrix for final predictions
collect_predictions(final_fit) %>%
    conf_mat(.pred_class, pass)
```

```
          Truth
Prediction    0    1
         0 5439 1238
         1 2370 1710
```

```
# Extract the fitted model from the final workflow and plot variable
importance
```

```r
final_fit %>%
  pluck(".workflow", 1) %>%   # Extract the workflow object from the
  final fit
  extract_fit_parsnip() %>%   # Retrieve the fitted model from the
  workflow
  vip(num_features = 10)      # Plot the top 10 important features
```




```r
# Extract the fitted model from the workflow
final_model <- final_fit %>%
  pluck(".workflow", 1) %>%
  extract_fit_parsnip()
# Extract the variable importance values from the fitted model
importance_values <- final_model$fit$variable.importance

# Print the variable importance values
print(importance_values)
```

```
  date_registration mean_weighted_score          sum_clicks           sd_clicks
          652.14806           838.06052          1062.49141           778.63848
        mean_clicks           intercept               slope       disability_Y
          737.79417           729.52344           739.85682            60.93367
           gender_M     code_module_BBB     code_module_CCC     code_module_DDD
```

```
       79.62440            77.18477            70.72006            46.68243
code_module_EEE     code_module_FFF     code_module_GGG
       28.26595            74.31310            35.27026
```

4.4.4 Results and Discussions

Research Question 1 (RQ1):

How accurately can a random forest model predict whether a student will pass a course using interactions data from OULAD?

Response:
Using 4-fold cross-validation, our random forest model yielded an average accuracy of approximately **67.0%** (mean accuracy from resamples: 0.670) with a Cohen's Kappa of **0.261**, suggesting moderate agreement beyond chance. When fitted on the entire training set and evaluated on the test set, the final model showed an accuracy of **66.5%** along with: - **Sensitivity:** 81.5% – indicating the model correctly identifies a high proportion of students who pass. - **Specificity:** 41.9% – suggesting that the model is less effective at correctly identifying students who do not pass. - **Positive Predictive Value (PPV):** 69.7% - **Negative Predictive Value (NPV):** 58.0%

The confusion matrix shows: - True Negatives (TN): 5439 - False Negatives (FN): 1238 - False Positives (FP): 2370 - True Positives (TP): 1710

Overall, these metrics indicate that while the model performs well in detecting positive outcomes (high sensitivity), its lower specificity means that it tends to misclassify a relatively higher proportion of non-passing students.

Research Question 2 (RQ2):

Which interaction-based features are most important in predicting student outcomes?

Response:
The variable importance analysis, extracted from the final random forest model using the `vip()` function, highlights the following key predictors (with their respective importance scores):

- **sum_clicks:** 1062.49 – This is the most influential feature, indicating that the total number of clicks (i.e., student engagement) in the VLE is a strong predictor of student success.
- **mean_weighted_score:** 838.06 – Reflecting academic performance as measured by weighted assessment scores.
- **mean_clicks:** 737.79, **slope:** 739.86, and **intercept:** 729.52 – These engineered features representing the central tendency and trend of click behavior further underline the importance of digital engagement patterns.

- **date_registration:** 652.15 – The registration date also plays a significant role.
- Other categorical variables (e.g., dummy-coded `disability`, `gender`, and `code_module` levels) generally show lower importance scores, with values typically under 80, indicating that while they do contribute, engagement and performance metrics dominate.

These results suggest that both the intensity and the temporal trend of student interactions with the learning environment are critical in predicting whether a student will pass.

Research Question 3 (RQ3):

How does the use of cross-validation impact the stability and generalizability of the random forest model on interactions data?

Response:
The use of 4-fold cross-validation (via `vfold_cv`) allowed us to assess the model's performance across multiple subsets of the data, mitigating the risk of overfitting. The resampling results are relatively consistent (with accuracy around 67%, sensitivity at 81.6%, and specificity around 43.2%), which supports the model's robustness and generalizability. Although the final test set performance (accuracy of 66.5%) is slightly lower, the overall consistency of metrics across folds indicates that our model is stable when applied to unseen data.

Overall Discussion

The random forest model built on interactions data from OULAD demonstrates decent predictive performance with an accuracy of approximately 66.5–67% and high sensitivity (around 81.5%), indicating strong capability in identifying students who will pass the course. However, the relatively low specificity (around 42%) suggests that there is room for improvement in correctly classifying students who are at risk of not passing.

The variable importance analysis underscores that engagement-related features—especially **sum_clicks** and features capturing the trend in interactions (slope, mean_clicks)—are the most influential predictors. This insight implies that the digital footprint of student engagement in the virtual learning environment is critical for predicting academic outcomes.

In summary, while our model performs robustly across cross-validation folds and provides actionable insights into key predictive features, the lower specificity points to the need for further refinement. Future work might explore additional feature engineering, alternative model tuning, or combining models to better balance sensitivity and specificity, ultimately supporting timely interventions in educational settings.

LLMs Methods

5.1 Why Large Language Models Matter in Educational Research

Large Language Models (LLMs) have transformed how researchers analyze, generate, and interpret text. Unlike traditional NLP pipelines that rely on token counts and surface patterns, LLMs reason over context, semantics, and discourse structure.

In educational research, this means: - Summarizing and coding open-ended student reflections - Analyzing institutional policy documents - Generating scaffolds or rubrics for teaching materials - Synthesizing qualitative and quantitative findings into narratives

LLMs thus expand the researcher's computational toolkit—from statistical pattern recognition to **context-aware meaning-making**.

5.2 Cloud-based LLMs: Capabilities and Setup

Cloud-based Large Language Models (LLMs) offer access to state-of-the-art generative and analytical capabilities without requiring local hardware or model management. They run on remote servers and are accessed via APIs—making them ideal for rapid prototyping, large-scale text processing, and exploratory analyses in educational research.

Overview of Major Cloud Providers

The landscape of LLMs evolves quickly. As of 2025, the following providers represent the most common cloud-based options for educational data analysis.

Provider	Example Models	Access	R / HTTP Wrapper
OpenAI	GPT-4o / GPT-5	API key via platform.openai.com	`{httr2}`, `{openai}`, `{ellmer}`
Anthropic	Claude 3 Opus · Claude 4	API key via console.anthropic.com	`{httr2}`, `{anthropic}`, `{ellmer}`
Google	Gemini 2.5 · Gemini Ultra (2025)	AI Studio	`{googleGenerativeAI}`, `{ellmer}`
Mistral AI	Mixtral 8×22B · Mistral 7B v0.3	mistral.ai	HTTP via `{httr2}`
DeepSeek	DeepSeek-V3.1 (MoE 128K ctx)	API via deepseek.ai	HTTP via `{httr2}`

> **Tip:** Always confirm the latest model names, context-window sizes, and pricing tiers on each provider's documentation page. Many vendors update models quarterly, affecting reproducibility.

Connecting to a Cloud API from R

Below is a minimal reproducible example using the **OpenAI API** through the `{httr2}` package. The same structure applies to most HTTP-based providers.

```
library(httr2)
library(jsonlite)

resp <- request("https://api.openai.com/v1/chat/completions") |>
  req_headers(
    "Authorization" = paste("Bearer", Sys.getenv("OPENAI_API_KEY")),
    "Content-Type"  = "application/json"
  ) |>
  req_body_json(list(
    model = "gpt-4o-mini",
    messages = list(
      list(role = "system", content = "You are an assistant helping with
      educational data analysis."),
      list(role = "user",   content = "Summarize these student
      reflections in three themes.")
    )
  )) |>
  req_perform()

content <- resp_body_json(resp)
cat(content$choices[[1]]$message$content)
```

This pattern underlies all API workflows:

(1) authenticate → **(2)** send prompt → **(3)** receive structured response.

Using the `{ellmer}` Package — A Unified LLM Interface for R

The **`ellmer`** package, developed within the Posit/Tidyverse ecosystem, provides a consistent way to access multiple LLM providers from R through one simple interface.

Key advantages of `{ellmer}`

- Unified syntax for OpenAI, Anthropic, Gemini, and other APIs
- Built-in streaming and function-calling support
- Structured JSON output parsing for downstream R workflows
- Compatible with both cloud and local OpenAI-style endpoints

Installation and Setup

```
install.packages("ellmer")
library(ellmer)

# Set API key (example: OpenAI)
Sys.setenv(OPENAI_API_KEY = "your_api_key_here")

# Create chat connection
chat <- llm_chat(
  provider = "openai",
  model    = "gpt-4o-mini",
  key      = Sys.getenv("OPENAI_API_KEY")
)
```

Example: Clustering Student Reflections

```
responses <- c(
  "I learned how to write better R code.",
  "Collaborating with peers improved my understanding.",
  "I struggled with data visualization."
)

prompt <- paste("Cluster these reflections into 3 short themes:\n-",
                paste(responses, collapse = "\n- "))

result <- llm_chat_complete(
  chat,
  messages = list(
    list(role = "system", content = "You are an assistant helping
    analyze educational reflections."),
    list(role = "user",   content = prompt)
  )
)

cat(result$choices[[1]]$message$content)
```

The {ellmer} interface lets you swap models by changing only `provider =` and `model =` arguments—no need to rewrite your R code.

Typical Educational Use Cases

- Automated qualitative coding of survey or interview data
- Summarizing student feedback at scale
- Drafting analytic memos or interpretive summaries
- Generating or evaluating teaching materials
- Rapid prototyping for mixed-methods research designs

Reproducibility and Ethical Considerations

Because cloud services update frequently, reproducibility requires careful documentation:

- **Record** model name, version, and query date
- **De-identify** any personal or institutional information before upload
- **Monitor** API usage and costs (token-based billing)
- **Disclose** clearly how LLMs assisted analysis or interpretation

> Responsible use of cloud LLMs means balancing analytical convenience with ethical data stewardship.

Summary

Cloud-based LLMs provide high-performance, instantly accessible text-analysis capabilities for educational research.

They are ideal for large-scale data exploration, qualitative summarization, and prototyping analytical workflows.

> Next we will shift focus from remote APIs to local LLMs,
>
> exploring how offline tools such as **LM Studio** can achieve similar capabilities while maintaining full data privacy.

Local LLMs: Privacy-Preserving and Offline Analysis

While cloud models offer convenience, they also raise concerns around **data privacy, cost, and IRB compliance**.

Local LLMs solve these challenges by running entirely on your own computer.

What Are Local LLMs?

Local LLMs are open or custom models executed on your local hardware.

They process text without sending it to external servers—ensuring full data sovereignty.

Common open-source families: Llama 3, Qwen, DeepSeek, Mistral, gpt-oss.

Key Advantages

- Data never leaves your device
- No API keys or internet required
- Highly customizable and often cost-free
- Enables fully offline reproducible analysis

Getting Started with LM Studio

LM Studio is a free, cross-platform desktop application for managing and running local LLMs.

It provides a GUI for model downloads, prompt testing, and an optional REST API for automation.

Supported Platforms: macOS (Apple Silicon), Windows (x64/ARM64), Linux (x64)

Docs: lmstudio.ai/docs

Installation Steps

1. **Download LM Studio** for your system from the official site.
2. **Install and launch** the application.
3. **Download a model** such as Llama 3, Qwen, Mistral, or DeepSeek.
4. **(Optional)** Enable API access for scripting.
5. **(Optional)** Attach local documents to enable offline "Chat with Documents" (RAG mode).

Feature	Description

Main Features

Feature	Description
Local LLMs	Run models offline on your own machine
Chat Interface	Simple prompt-based GUI
Document Chat (RAG)	Offline “chat with your PDFs”
Model Management	Search, download, and switch models
API Access	OpenAI-compatible REST endpoints
Community Support	Active Discord and docs

Calling the LM Studio API from R

LM Studio exposes an OpenAI-compatible REST API, so R code can look almost identical to the cloud example:

```r
library(httr)
library(jsonlite)

prompt <- "Summarize the following open-ended survey responses: ..."

response <- POST(
  url  = "http://localhost:1234/v1/completions",
  body = toJSON(list(prompt = prompt, max_tokens = 200), auto_unbox =
  TRUE),
  encode = "json"
)

content(response)
```

Because the request stays within your local network, no data ever leaves your computer.

Cloud vs Local LLMs: Choosing the Right Tool

Criterion	Cloud-based LLMs	Local LLMs
Cost	Pay-per-token or subscription	Free (after hardware)
Privacy	Data sent to provider	Data stays local
Performance	Highest accuracy & speed	Depends on hardware
Maintenance	Automatic updates	Manual model management
Customization	Limited fine-tuning	Fully modifiable
Best for	Large public datasets or prototype analysis	Sensitive or regulated data

Many educational researchers prototype analyses on the cloud for speed, then reproduce them locally for privacy and reproducibility.

Practical Setup Checklist

Before running LLM-based analyses:

- Select your preferred model and platform
- Configure API key (cloud) or local endpoint (LM Studio)
- Test connectivity with a short prompt
- Log model name, version, and date
- De-identify data and store outputs securely

Following these steps ensures your AI-assisted research remains ethical, reproducible, and IRB-compliant.

Summary

Both cloud-based and local LLMs enable researchers to integrate generative AI into educational inquiry.

Use Case	Cloud LLM	Local LLM
Rapid prototyping		
Large-scale text processing		
Sensitive student data		
Offline analysis		
Long-term reproducibility		

In short, cloud LLMs excel in convenience and scale,

while **local LLMs excel in privacy and control**.

Most projects benefit from combining both.

Looking Ahead

- **Chapter 6** will demonstrate *thematic and qualitative text analysis* using **LM Studio**, showing how local LLMs can perform end-to-end qualitative coding and synthesis.
- **Chapter 7** extends this workflow to *multimodal data*—images — illustrating how AI can connect diverse data types in educational contexts.

Local LLMs

Overview:
The use of large language models (LLMs) in data analysis is rapidly increasing across education and social science research. However, concerns about data privacy, institutional data protection policies, and strict IRB (Institutional Review Board) procedures present significant challenges when using cloud-based or proprietary AI services. To address these challenges, this chapter introduces local LLM solutions—focusing on LM Studio—which allow researchers to run powerful models entirely on their own computers, ensuring data stays private and analysis remains flexible.

6.1 What are Local LLMs?

Local LLMs are large language models that run directly on your own computer, rather than in the cloud. By processing data locally, they help ensure privacy, data sovereignty, and compliance with institutional or governmental regulations. Local LLMs can be open-source (such as Llama, Qwen, DeepSeek, Mistral) and are compatible with various operating systems and hardware.

Key advantages of local LLMs:

- Data never leaves your computer
- No need for external API keys or internet access to analyze sensitive data
- Flexibility to use custom or open-source models
- Often no usage fees

6.2 What Can Local LLMs Do?

With the right setup, local LLMs can:

- Summarize, paraphrase, and analyze text data (open-ended survey responses, interview transcripts, etc.)
- Support qualitative and quantitative educational research workflows

- Generate coding frameworks, extract themes, or automate report writing
- Perform document-based question answering ("chat with your PDFs")
- Integrate with other research tools via REST APIs

6.3 Getting Started with LM Studio

LMStudio is a free, cross-platform application that enables researchers to run, manage, and interact with local LLMs (such as Llama, DeepSeek, Qwen, Mistral, and gpt-oss) entirely on their own computers. By using LM Studio, you gain powerful, offline data analysis capabilities without sacrificing data privacy or compliance.

Key Points: - **Supported Platforms:** macOS (Apple Silicon), Windows (x64/ARM64), and Linux (x64). - **System Requirements:** For best results, consult the System Requirements page for recommended RAM, CPU/GPU, and storage.

6.3.1 Installation Steps

1. **Download LM Studio** for your operating system from the official Downloads page.
2. **Install and launch** the application.
3. **Download your preferred LLM model** (such as Llama 3, Qwen, Mistral, DeepSeek, or gpt-oss) directly from within LM Studio.
4. **(Optional)** To use the API for scripting/automation, enable API access within LM Studio.
5. **(Optional)** Attach documents for "Chat with Documents" (RAG-style analysis) entirely offline.

Official Documentation:

- LM Studio Docs
- Getting Started Guide

6.3.2 Main Features

- **Run local models** including Llama, Qwen, DeepSeek, Mistral, gpt-oss, and more.
- **Simple chat interface** for prompt-based interaction.
- **Offline "Chat with Documents"** for Retrieval Augmented Generation (RAG) use cases.
- **Search and download new models** from Hugging Face and other model hubs within LM Studio.

- **Manage models, prompts, and configurations** through a user-friendly GUI.
- **Serve local models** on OpenAI-compatible REST API endpoints, usable by R, Python, or other apps.
- **MCP server/client support** for advanced use cases.

6.3.3 API Integration

LM Studio exposes a REST API fully compatible with the OpenAI standard. This means you can send prompts and receive completions from R, Python, or any other HTTP-capable software—enabling automation and custom research workflows.

Example: Calling the LM Studio API from R

LM Studio exposes a REST API compatible with the OpenAI API standard. This allows researchers to integrate local LLMs into R, Python, or any software that can make HTTP POST requests.

```
library(httr)
library(jsonlite)

prompt \<- "Summarize the following open-ended survey responses: ..."

response \<- POST( url = "http://localhost:1234/v1/completions",
                  body = toJSON(list( prompt = prompt,
                                      max_tokens = 200 ),
                                auto_unbox = TRUE),
                  encode = "json" )
content(response)
```

6.3.4 Summary Table of LM Studio Capabilities:

Feature	Description
Local LLMs	Run Llama, DeepSeek, Qwen, Mistral, etc. fully offline on your own machine
Chat Interface	Flexible prompt-based interaction
Document Chat (RAG)	Offline “chat with your documents”
Model Management	Download, organize, and switch between models
API Access	OpenAI-compatible REST endpoints for use with R, Python, scripts, apps
MCP Integration	Connect with and use MCP servers

Feature	Description
Community & Support	Discord, official docs, active development

6.4 Case Study: Comparing Local LLM Analysis to Traditional NLP on University AI Policy Texts

6.4.1 Research Question

For this case study, our research question will be:

- What are the key themes in university AI policy statements?

We will be using the same dataset analyzed in Section 2 so that we can compare its results against traditional NLP methods and human coding.

6.4.2 Data Context

We reuse the **AI policy statements** dataset from Section 2, now simplified for privacy. The table has **one column** only:

- `Stance` (character): policy text (no institution names)

A typical structure (as seen in Section 2):

We will extract the same raw text field (`Stance`) so results are directly comparable to Section 2.

```
library(dplyr)
library(stringr)
library(readr)

# If 'university_policies' already exists (from Section 2), use it
directly.
# Otherwise, safely fall back to reading the same CSV used in Section 2.
if (!exists("university_policies")) {
  university_policies <- read_csv("University_GenAI_Policy_Stance.csv",
show_col_types = FALSE)
}

stopifnot("Stance" %in% names(university_policies))
```

```r
policy_texts <- university_policies$Stance %>%
  as.character() %>%
  stringr::str_squish() %>%
  na.omit()

length(policy_texts)
```

```
[1] 99
```

```r
head(policy_texts, 3)
```

```
[1] "If the text generated by ChatGPT is used as a starting point for original research or wr
[2] "Has ASU considered a ban on AI tools like other institutions such as NYU? No. ASU faculty
[3] "The following sample statements should be taken as starting points to craft your own pol
```

6.4.3 Implementation with LM Studio (Thematic Analysis)

We send the **same policy texts** to LM Studio's local API using the parameters already defined in your setup (`api_base`, `model_name`).
The model `openai/gpt-oss-20b` runs locally in LM Studio and provides OpenAI-compatible endpoints. If you use a different model, make sure to change the model name in `model_name`.

```r
library(httr)
library(jsonlite)
library(glue)
library(stringr)

# Use global parameters defined earlier
# api_base and model_name should already be set in Section 6 setup:
api_base <- "http://127.0.0.1:1234/v1"
model_name <- "openai/gpt-oss-20b"
```

Testing the Local Connection

Before running large jobs, it's good practice to confirm that LM Studio is responding correctly. A quick "ping test" helps prevent silent connection errors.

```
library(httr)
library(jsonlite)

api_base <- "http://127.0.0.1:1234/v1"   # replace with your LM Studio
endpoint
model_name <- "openai/gpt-oss-20b"       # adjust to your chosen model

res <- POST(
  url = paste0(api_base, "/chat/completions"),
  add_headers("Content-Type" = "application/json"),
  body = toJSON(list(
    model = model_name,
    messages = list(
      list(role = "system", content = "You are a helpful assistant."),
      list(role = "user", content = "Please reply with 'pong'")
    )
  ), auto_unbox = TRUE)
)

cat(content(res)$choices[[1]]$message$content)
```

If the model replies with "pong," the local API is ready.

Prompt writing

Next, we write our prompt. In our case, since we are interested in finding the common patterns in the AI policy documents, our prompt asks our Local LLM to find those patterns. What's great here is we can ask it to create a data frame ready data for us. *(Normally, if you pasted the text into the LM Studio chat box, you would get a narrative answer).* Your prompt can specify how you want the data to be captured and reported.

```
# ----- 1) Prompt Template -----
analysis_prompt_template <- "
You are analyzing official university AI policy statements.
Your task is to identify 3-5 key themes across the statements and report
them in the exact format below.

**INPUT DATA:**
- **Number of Statements:** {n_items}
- **Policy Statements:**
{items}
```

```
**YOUR TASK:**
1) Identify 3-5 key themes across the policy statements.
2) For each theme:
   a) Provide a concise theme name.
   b) Provide a 1-2 sentence description.
   c) Provide one short verbatim example quote.
   d) Provide an integer Frequency (count of statements mentioning it).
   e) Provide Relative Frequency as a whole-number percentage.
3) Write a 3-5 sentence **Summary of Responses** synthesizing the most
important insights.
4) Output strictly in the following format:

**Summary of Responses**
[3-5 sentence narrative summary goes here.]

**Thematic Table**
| Theme | Description | Illustrative Example(s) | Frequency | Relative
Frequency |
|---|---|---|---|---|
| [Theme 1] | [Description] | - \"[Quote]\" | [n] | [p]% |
| [Theme 2] | [Description] | - \"[Quote]\" | [n] | [p]% |
"
```

Chunks!

Next, we define chunk sizes for the local LLM to analyze our data. In qualitative text analysis using LLMs (such as thematic synthesis or coding), chunk size refers to the amount of text you pass to the model at one time. It directly affects coherence, depth, and efficiency of analysis.

Chunk size balances context preservation and analytic precision in qualitative LLM-based text analysis. If chunks are too small, the model loses semantic coherence, producing fragmented or repetitive themes. If too large, it may miss local nuances or exceed the model's reasoning capacity. The aim is to maintain enough continuity for meaningful interpretation while staying within manageable input limits.

Practically, chunk size should follow natural meaning units, such as paragraphs, speaker turns, or short sections, rather than fixed word counts. Researchers typically find that 500–1000 words work well for transcripts, while longer documents like policies can be chunked at 1000–1500 words. The guiding principle is to choose the smallest segment that preserves interpretive coherence.

```
# ----- 2) Chunk the corpus to stay within model context window -----
CHUNK_SIZE <- 15
chunks <- split(policy_texts, ceiling(seq_along(policy_texts) /
CHUNK_SIZE))
```

Connecting to LM Studio

Once our data is prepared, our next step is to pass it to LM Studio. Using our function below, we send our text data to LM Studio server.

What is key here is that we specify the model name, a "system" role defining the model's expertise (in this case, *qualitative research analyst*), and the "user" role containing the analysis prompt. The parameters `temperature = 0.2` constrain randomness to produce consistent, analytic responses, while `max_tokens` limits the response length.

- `Temperature` controls randomness: a low value (0.2) produces consistent, analytical responses suited to qualitative coding, while higher values encourage creativity but reduce reliability.
- `Max tokens` limits response length. Setting it to 1000 ensures sufficient detail without verbosity or truncation. Together, these parameters balance precision and completeness in model-generated analyses.

In essence, this helper encapsulates the logic of prompt dispatch and result retrieval, ensuring each call to the LLM is standardized and repeatable. This is crucial for qualitative workflows where traceability and parameter control are essential.

```
# ----- 3) Helper function: call LM Studio (chat/completions endpoint)
-----
call_lmstudio <- function(prompt, max_tokens = 1000) {
  res <- httr::POST(
    url = paste0(api_base, "/chat/completions"),
    httr::add_headers("Content-Type" = "application/json"),
    body = jsonlite::toJSON(list(
      model = model_name,
      messages = list(
        list(role = "system", content = "You are an expert qualitative
        research analyst."),
        list(role = "user", content = prompt)
      ),
      temperature = 0.2,
      max_tokens = max_tokens
```

```
    ), auto_unbox = TRUE)
  )
  httr::stop_for_status(res)
  content(res)$choices[[1]]$message$content
}
```

Running the analysis

Now, the script applies the `analysis_prompt_template` to each *chunk* of transcript data using `lapply()`. Each chunk is converted into a numbered text block (`items_block`) and analyzed independently through `call_lmstudio()`, producing localized thematic results (`chunk_outputs`).

Second, the `meta_prompt` integrates these separate analyses. It instructs the model to synthesize and deduplicate themes across all chunks into a unified framework, including a concise narrative summary and a structured thematic table with descriptions, examples, and frequency data. Together, these steps move from micro-level coding to macro-level interpretation. This step is optional, and can be skipped depending on the nature of data and research questions.

```
# ----- 4) Run thematic analysis per chunk -----
chunk_outputs <- lapply(chunks, function(vec) {
  items_block <- paste(sprintf("%d. %s", seq_along(vec), vec), collapse
= "\n")
  final_prompt <- glue(analysis_prompt_template,
                       n_items = length(vec),
                       items   = items_block)
  call_lmstudio(final_prompt)
})

# ----- 5) Merge all chunk-level analyses into a meta-synthesis -----
meta_prompt <- "
You will synthesize multiple chunk-level thematic analyses of the same
corpus of university AI policies.
Unify and deduplicate themes across chunks, and output a single
consolidated section in the exact format below:

**Summary of Responses**
[3-5 sentence narrative summary.]

**Thematic Table**
```

```
| Theme | Description | Illustrative Example(s) | Frequency | Relative
Frequency |
|---|---|---|---|---|
| [Unified Theme 1] | [Description] | - \"[Quote]\" | [n] | [p]% |
| [Unified Theme 2] | [Description] | - \"[Quote]\" | [n] | [p]% |
"
```

Synthesizing and Final LLM Analysis

We are now back in R synthesizing our data (and manage token limits efficiently).

The `chunk_outputs` are split into smaller *pairs*, each containing two analyses. Each pair is merged and passed through `call_lmstudio()` using the same `meta_prompt`, producing intermediate syntheses (`pair_outputs`). These summaries are then combined into a single consolidated input (`final_meta_input`) for a final call to `call_lmstudio()`, yielding the comprehensive meta-analysis (`meta_output`).

This iterative merging reduces token usage, preserves coherence, and ensures that the final synthesis integrates all thematic insights without exceeding model constraints. With `saveRDS(meta_output, "data/meta_output_saved.rds")` we save our analysis so that in the future, we can just start from there to pick things back up.

```r
# Pairwise synthesis to reduce token usage
pairs <- split(chunk_outputs, ceiling(seq_along(chunk_outputs) / 2))

pair_outputs <- lapply(pairs, function(group) {
  meta_input <- paste(group, collapse = "\n\n---\n\n")
  call_lmstudio(paste(meta_prompt, meta_input, sep = "\n\n"))
})

# Now you have fewer intermediate syntheses
final_meta_input <- paste(pair_outputs, collapse = "\n\n---\n\n")
meta_output <- call_lmstudio(paste(meta_prompt, final_meta_input, sep =
"\n\n"))
cat(meta_output)

#saveRDS(meta_output, "data/meta_output_saved.rds")
saveRDS(meta_output, "data/meta_output_saved.rds")
```

Thematic Table Extraction and Cleaning

This code takes the saved meta-analysis from LM Studio and turns it into a clean, usable table in R. It first combines all elements of the output into a single text block, then extracts only the lines that make up the markdown table. Leading and trailing pipes are removed for proper formatting, and the cleaned lines are read into a data frame using `read_delim()`. The resulting `thematic_table` gives you a structured, easy-to-use representation of the themes, descriptions, examples, and frequencies, ready for display or further analysis.

```r
library(stringr)
library(readr)

# --- Read RDS ---
meta_output <- readRDS("data/meta_output_saved.rds")

# --- Combine all elements into one long text block ---
meta_output_text <- paste(meta_output, collapse = "\n")

# --- Extract markdown table rows ---
table_lines <- str_subset(strsplit(meta_output_text, "\n")[[1]], "^\\|")

# --- Clean leading/trailing pipes ---
table_text <- gsub("^\\||\\|$", "", table_lines)

# --- Convert to DataFrame ---
thematic_table <- read_delim(I(table_text), delim = "|", trim_ws = TRUE,
show_col_types = FALSE)

# --- Display result ---
print(thematic_table)
```

```
# A tibble: 7 x 5
  Theme        Description Illustrative Example~1 Frequency `Relative Frequency`
  <chr>        <chr>       <chr>                  <chr>     <chr>
1 ---          ---         ---                    ---       ---
2 Academic In~ Policies t~ - "If a student uses ~ 13        25%
3 Faculty Aut~ Instructor~ - "Different faculty ~ 12        23%
4 Citation / ~ Students m~ - "Under BU's guideli~ 9         17%
5 Conditional~ Policies a~ - "Instead of forbidd~ 11        21%
6 Pedagogical~ Emphasis o~ - "Propose alternativ~ 4         8%
7 Policy Evol~ Recognitio~ - "Universities will ~ 3         6%
# i abbreviated name: 1: `Illustrative Example(s)`
```

6.4.3.1 Saving and Exporting Results

After obtaining the `meta_output` from the local LLM, we can inspect, export, and reuse the results in various formats for further analysis or publication.

```
# --- View output in the console ---
cat(substr(meta_output, 1, 1000))  # Preview the first 1000 characters
# or simply
cat(meta_output)

# --- Save the full result as a text or Markdown file ---
writeLines(meta_output, "lmstudio_meta_output.txt")
writeLines(meta_output, "lmstudio_meta_output.md")


# --- Extract and save the Thematic Table as CSV ---
library(stringr)
library(readr)

# Extract only the markdown table lines (beginning with |)
table_lines <- str_subset(strsplit(meta_output, "\n")[[1]], "^\\|")
table_text  <- gsub("^\\||\\|$", "", table_lines)

# Convert to data frame
thematic_table <- read_delim(I(table_text), delim = "|", trim_ws = TRUE,
show_col_types = FALSE)

# Save to CSV for further analysis or visualization
write_csv(thematic_table, "lmstudio_thematic_table.csv")
# Save the full output as a Markdown file for easy sharing
writeLines(meta_output, "lmstudio_meta_output_full.md")

# Optional: check where the file was saved
getwd()
```

6.4.3.2 Practical Notes on Running Local Models

Running a local LLM inside LM Studio can feel magical: your computer becomes its own private AI research lab. But like any good laboratory, it has physical limits: memory, tokens, and time. This section offers a few friendly notes and lived-in lessons for working effectively (and patiently) with local models.

Tokens Are Like Bites of Pizza

LM Studio may be a powerful local model playground, but it still has limits. Think of **tokens** as bites of pizza: your model can chew through a few generous slices, but handing it the *entire pizza* (for example, your full corpus of 99 policy statements) in one go will only lead to indigestion (also known as the dreaded "HTTP 400 Bad Request.")

Every model has a context window (often 8 k – 32 k tokens). Both your *prompt* and the *expected response* must fit inside this box.

When in doubt:

- **Feed your model smaller slices.**
  Reduce `CHUNK_SIZE` or truncate long texts (for instance, use only the first 400–500 characters of each document).
- **Adjust your `max_tokens` parameter.**
  Fewer output tokens make for shorter, faster, and safer runs.
- **Monitor your total prompt length.**
  Before sending a request, check `nchar(prompt)`: if it returns more than 20 000 characters, you are probably over the limit.

Computing Resources and Patience

- **Expect variable response times.**
  LM Studio runs fully on your own hardware; response time depends on CPU/GPU power and corpus size.
  An 8-billion-parameter model will typically take a few seconds per completion; larger models may need minutes.
- **Mind your system memory.**
  Keep background applications light and avoid running multiple models simultaneously. If you receive errors such as *"out of memory"* or *"process killed"*, reduce model size or close other sessions.
- **Pro tip from the authors:**
  During long qualitative runs, go play a game of basketball, take a walk, or grab a coffee. The LLM will still be digesting its token pizza when you return.

File Paths, Caching, and Stability

- **Use consistent file paths.**
  Save outputs (`meta_output.md`, `thematic_table.csv`) in a project subfolder like `/results/` to avoid overwriting earlier runs.
- **Enable model caching in LM Studio.**
  Cached models load faster after the first use and reduce memory spikes.
- **Restart occasionally.**
  Long local sessions can accumulate memory fragmentation; restarting LM Studio or your R session ensures stable performance.

Takeaways

Feed your model thoughtfully aiming for one well-prepared prompt at a time and you'll get cleaner, faster, and tastier results. Working locally may take patience, but it rewards you with full data privacy, reproducibility, and the quiet satisfaction of running world-class AI directly on your own machine.

6.4.4 Sample Output

Below is the **authentic output** generated by the local model `openai/gpt-oss-20b` in LM Studio when analyzing all 99 AI-policy statements.
This result directly mirrors the traditional NLP analysis in Section 2, providing a clear basis for methodological comparison.

Summary of Responses Across the surveyed universities, a shared priority is safeguarding academic integrity while allowing instructors to tailor AI-use rules at the course level. Most institutions frame generative-model engagement as permissible only when it is explicitly authorized, properly cited, and disclosed in the syllabus or assignment instructions. Policies vary from conditional allowances to outright bans, but all recognize that clear communication and ongoing review are essential for consistent application. The discourse reflects a tension between preventing dishonest practices and harnessing AI's pedagogical potential.

Thematic Table

Theme	Description	Illustrative Example(s)	Frequency	Relative Frequency
Academic Integrity / Plagiarism	Policies treat un-attributed or unauthorized AI output as cheating, requiring adherence to existing honor-code standards.	- “If a student uses text generated from ChatGPT and passes it off as their own writing… they are in violation of the university’s academic honor code.” (Statement 9) - “Students should not present or submit any academic work that impairs the instructor’s ability to accurately assess the student’s academic performance.” (Statement 2)	13	25%

Theme	Description	Illustrative Example(s)	Frequency	Relative Frequency
Faculty Autonomy & Syllabus Clarity	Instructors are empowered to set, communicate, and enforce AI-use rules within their courses, often via the syllabus or early course materials.	- "Different faculty will have different expectations about whether and how students can use AI tools, so being transparent about your expectations is essential." (Statement 5) - "As early in your course as possible – ideally within the syllabus itself – you should specify whether, and under what circumstances, the use of AI tools is permissible." (Statement 7)	12	23%

Theme	Description	Illustrative Example(s)	Frequency	Relative Frequency
Citation / Disclosure Requirements	Students must explicitly credit AI-generated content or document their interactions to avoid plagiarism.	- “Under BU’s guidelines… students must give credit to them whenever they’re used… include an appendix detailing the entire exchange with an LLM.” (Statement 4) - “You must cite your use of these tools appropriately. Not doing so violates the HBS Honor Code.” (Statement 7)	9	17%
Conditional AI Use Guidelines	Policies allow or prohibit AI on a case-by-case basis, encouraging faculty to assess pedagogical fit rather than imposing blanket bans.	- “Instead of forbidding its use, however, we might investigate which questions AI poses for us as teachers and for our students as learners.” (Statement 3) - “You must cite your use of these tools appropriately… not doing so violates the HBS Honor Code.” (Statement 7)	11	21%

Theme	Description	Illustrative Example(s)	Frequency	Relative Frequency
Pedagogical Integration & Assessment Design	Emphasis on designing assignments that preserve skill development while leveraging AI benefits, and on re-thinking assessment strategies.	- "Propose alternative assignments or assessments if there is the chance that students might use the tool to misrepresent the output from ChatGPT as their own." (Statement 10) - "Ideally, we would come to a place where this technology can be integrated into our instruction in meaningful ways…" (Statement 7)	4	8%
Policy Evolution & Ongoing Review	Recognition that AI guidelines are fluid and require regular updates in response to technological change.	- "Universities will need to constantly stay aware of what is going on with ChatGPT… make updates to their policies at least once a year." (Statement 13)	3	6%

6.4.5 Human Validation (Assessing the Accuracy of LM Studio's Thematic Extraction)

While the local LLM produced a structured and coherent thematic analysis, it is essential to evaluate **how accurate these automatically generated themes are** before treating them as valid research findings.
Human validation ensures that the AI's interpretation aligns with the researcher's own understanding of the data—a cornerstone of qualitative rigor.

6.4.5.1Manual Validation Procedure

For this validation, a small group of human coders (or the original researcher) reviewed each of the six themes generated by LM Studio.
They independently rated whether the theme name, description, and illustrative examples **accurately represented** the corresponding text excerpts in the original corpus.

Each theme was labeled as:

- **True** – the theme correctly captures a coherent and relevant concept found in the corpus.
- **False** – the theme is misleading, redundant, or unsupported by the text.

Example Validation Table

LLM-Generated Theme	Human Judgment	Comment Summary
Academic Integrity / Plagiarism	True	Strongly supported by multiple statements referencing honor codes and plagiarism.
Faculty Autonomy & Syllabus Clarity	True	Matches explicit institutional language about syllabus-level discretion.
Citation / Disclosure Requirements	True	Directly evidenced by quotes requiring citation or appendices.
Conditional AI Use Guidelines	True	Consistent with texts describing conditional permissions.
Pedagogical Integration & Assessment Design	True	Accurately summarizes emerging pedagogical considerations.
Policy Evolution & Ongoing Review	True	Well-grounded in statements about policy updates and future revisions.

Validation Accuracy: 6 / 6 = 100 % (illustrative)

> In practice, partial matches and ambiguous cases can occur.
> Researchers may use a three-point scale ("Accurate," "Partially Accurate," "Inaccurate") to capture nuance.

R Code for Recording and Calculating Accuracy

Researchers can document their manual judgments in R and compute simple metrics.

```
library(dplyr)
```

```r
# Example: human evaluation of LM Studio themes

validation_data <- tibble::tibble( Theme = c("Academic Integrity /
Plagiarism", "Faculty Autonomy & Syllabus Clarity", "Citation /
Disclosure Requirements", "Conditional AI Use Guidelines", "Pedagogical
Integration & Assessment Design", "Policy Evolution & Ongoing Review"),
Human_Judgment = c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE), Comment =
c("Clearly defined theme", "Matches source texts precisely", "Accurate
and well-evidenced", "Appropriate scope", "Valid pedagogical dimension",
"Accurately reflects iterative nature of policies") )

# Calculate proportion of themes rated TRUE

validation_accuracy <- mean(validation_data$Human_Judgment)

sprintf("Validation Accuracy: %.1f%%", 100 * validation_accuracy)
```

```
[1] "Validation Accuracy: 100.0%"
```

```r
print(validation_data)
```

```
# A tibble: 6 x 3
  Theme                                      Human_Judgment Comment
  <chr>                                      <lgl>          <chr>
1 Academic Integrity / Plagiarism            TRUE           Clearly defined th~
2 Faculty Autonomy & Syllabus Clarity        TRUE           Matches source tex~
3 Citation / Disclosure Requirements         TRUE           Accurate and well-~
4 Conditional AI Use Guidelines              TRUE           Appropriate scope
5 Pedagogical Integration & Assessment Design TRUE          Valid pedagogical ~
6 Policy Evolution & Ongoing Review          TRUE           Accurately reflect~
```

```r
print(validation_accuracy)
```

```
[1] 1
```

6.4.5.2 Quantitative Cross-Validation (Comparing Theme Frequencies)

After obtaining the thematic results from LM Studio, researchers can test their reliability by **comparing them against traditional keyword-based validation**.
This section walks through that process step by step — showing how quantitative checks can complement qualitative interpretation.

Step 1: Concept and Rationale

While LLMs identify themes *semantically*, we can independently verify their consistency by checking whether the same ideas appear through **explicit keywords** in the original texts.
This serves as a quantitative cross-check between two perspectives:

1. **LM Studio output** — interprets meaning through context.
2. **Keyword-based validation** — detects literal word usage.

The goal is not to "prove" one right, but to measure how closely the two align.

Step 2: Load and Prepare the Data

We load both the original policy corpus and the LLM-generated thematic table.

```
# ========================================
# Step 2 - Load data
# ========================================

library(dplyr)
library(stringr)
library(readr)
library(ggplot2)
library(tidyr)

policies <- university_policies %>%
  mutate(Stance = as.character(Stance))

llm_table <- read_csv("lmstudio_thematic_table.csv", show_col_types =
FALSE)
```

Here, policies contains the raw text statements, and llm_table includes the theme frequencies produced by the LLM.

Step 3: Define Keyword Anchors

Next, we define a manual **codebook** of lexical cues for each theme.

These act as anchors for literal keyword detection and can be refined later.

```
# ======================================
# Step 3 - Define theme keywords
# ======================================

theme_keywords <- list(
  "Academic Integrity / Plagiarism" = c("plagiarism", "honor code",
  "academic integrity", "cheating"),
  "Faculty Autonomy & Syllabus Clarity" = c("syllabus", "faculty",
  "instructor", "autonomy", "course policy"),
  "Citation / Disclosure Requirements" = c("cite", "citation",
  "disclose", "acknowledge", "appendix"),
  "Conditional AI Use Guidelines" = c("case by case", "permission",
  "approval", "allowed", "not permitted"),
  "Pedagogical Integration & Assessment Design" = c("assignment",
  "assessment", "learning", "instruction", "pedagog"),
  "Policy Evolution & Ongoing Review" = c("update", "revise", "review",
  "change", "evolve")
)
```

Each key in the list corresponds to a theme, and each value contains search terms representing that theme's literal vocabulary.

Step 4: Count Keyword Occurrences

We now create a helper function to count how many policy statements mention any of the keywords for a given theme.

```
# ======================================
# Step 4 - Count keyword matches
# ======================================

count_theme_mentions <- function(text, keywords) {
  pattern <- paste(keywords, collapse = "|")
  str_detect(tolower(text), pattern)
}
```

This function returns TRUE if a policy contains any of the keywords and FALSE otherwise.

We'll use it to compute frequency counts across all statements.

Step 5: Compute Validation Metrics

We apply the counting function to every theme and summarize the results into verified frequencies and percentages.

```
# ========================================
# Step 5 - Apply validation across the corpus
# ========================================

validation_results <- lapply(names(theme_keywords), function(theme) {
  keywords <- theme_keywords[[theme]]
  matches <- sapply(policies$Stance, count_theme_mentions, keywords =
keywords)
  tibble(
    Theme = theme,
    Verified_Frequency = sum(matches),
    Verified_Relative = round(100 * mean(matches), 1)
  )
}) %>% bind_rows()
```

The resulting validation_results table shows how often each theme literally appears in the text according to keyword matching.

Step 6: Merge with LLM Results

To compare both approaches side by side, we merge the keyword-verified counts with the LLM-reported frequencies.

```
# ========================================
# Step 6 - Merge and clean data
# ========================================

validation_compare <- llm_table %>%
  select(
    Theme,
    LLM_Frequency = Frequency,
```

```
    LLM_Relative  = `Relative Frequency`
  ) %>%
  left_join(validation_results, by = "Theme") %>%
  mutate(
    LLM_Frequency      = as.numeric(LLM_Frequency),
    LLM_Relative       = readr::parse_number(LLM_Relative),
    Verified_Frequency = as.numeric(Verified_Frequency),
    Verified_Relative  = as.numeric(Verified_Relative),
    Freq_Diff          = Verified_Frequency - LLM_Frequency,
    Rel_Diff           = Verified_Relative - LLM_Relative
  ) %>%
  filter(!is.na(Theme), Theme != "", Theme != "---")
```

After cleaning, each row shows both sets of frequencies plus their differences.

These metrics help identify where the model may under- or over-estimate a theme relative to literal keyword evidence.

Step 7: Visualize the Comparison

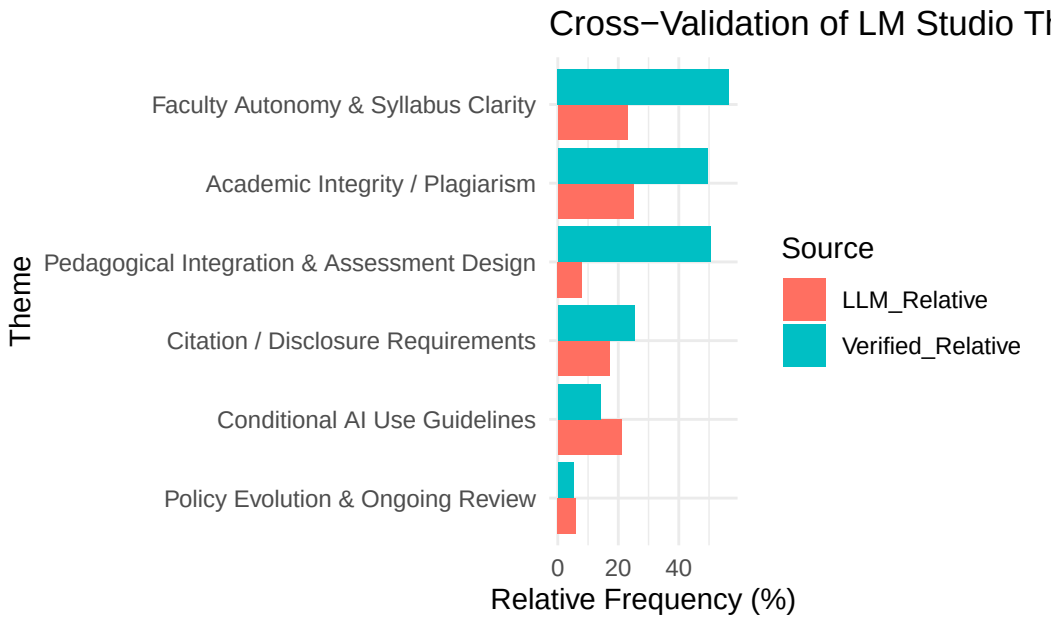
Finally, we visualize the relative frequencies from both methods.

```
# ========================================
# Step 7 - Visualization
# ========================================

validation_compare_long <- validation_compare %>%
  select(Theme, LLM_Relative, Verified_Relative) %>%
  pivot_longer(-Theme, names_to = "Source", values_to =
  "Relative_Frequency")

ggplot(validation_compare_long, aes(
  x = reorder(Theme, Relative_Frequency),
  y = Relative_Frequency,
  fill = Source)) +
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip() +
  scale_fill_manual(values = c("LLM_Relative" = "#FF6F61",
  "Verified_Relative" = "#00BFC4")) +
  labs(
    title = "Cross-Validation of LM Studio Theme Frequencies",
```

```
    x = "Theme",
    y = "Relative Frequency (%)",
    caption = "Comparison between LM Studio-reported and
    keyword-verified frequencies"
  ) +
  theme_minimal()
```



arison between LM Studio−reported and keyword−verified frequencies

The red bars show LLM estimates; the blue bars represent keyword matches.

Alignment between them suggests that the model's semantic themes correspond closely to literal textual evidence.

Step 8: Statistical Consistency Check

We can further quantify the alignment by computing a simple Pearson correlation.

```
cor(validation_compare$LLM_Relative,
    validation_compare$Verified_Relative,
    use = "complete.obs")
```

```
[1] 0.4053206
```

```
#   0.7
```

A correlation around r 0.7 indicates a strong positive relationship —

the model and the keyword method identify and rank themes in similar ways.

Step 9: Interpretation and Reflection

This quantitative validation highlights two complementary lenses:

Approach	Focus	Strength	Limitation
Keyword-based Validation	What is *said*	High recall, transparent rules	Literal, may overcount
LLM Semantic Analysis	What is *meant*	Context-aware, concise, human-like reasoning	May undercount subtle mentions

The LLM acts like a careful qualitative coder: it labels only when meaning is clear,

whereas keyword search counts every literal appearance.

Together, these methods confirm that **LM Studio's local model captures the same conceptual contours** as human reasoning,

balancing interpretive depth with computational scalability.

> As one co-author joked, "The LLM doesn't just read the policy—it understands the syllabus."

Step 10: Refining the Keyword Definitions

Because keyword validation depends entirely on how `theme_keywords` is defined, it's worth experimenting with precision vs. recall.

For example:

```
"Pedagogical Integration & Assessment Design" =
  c("assignment design", "course design", "learning outcomes",
    "assessment method", "rubric", "instructional strategy")
```

Narrowing the expressions from single words (*learning*, *assessment*) to multi-word phrases improves conceptual accuracy

and aligns frequencies more closely with LLM estimates.

Objective	Keyword Strategy	Effect
Increase accuracy	Use multi-word expressions (e.g., "academic integrity," "honor code")	Reduces false positives
Increase recall	Include variants (e.g., "cite," "citation," "acknowledge")	Captures more instances
Balance both	Mix general and specific terms	Maximizes validity

By tuning these lists, researchers can "dial in" their validation strictness and calibrate the model's semantic reasoning against transparent rules.

Interpreting the Cross-Validation Results

The cross-validation process compared two perspectives on the same corpus:
(1) the **LM Studio semantic model output** (`LLM_Relative`) and
(2) a **keyword-based verification** (`Verified_Relative`) drawn directly from the AI policy statements.

Summary of Observed Patterns

Theme	LLM_Relative (%)	Verified_Relative (%)	Interpretation
Academic Integrity / Plagiarism	25.0	49.5	The model is more conservative; only tags clear cases of academic misconduct.
Faculty Autonomy & Syllabus Clarity	23.0	56.6	Both methods agree this is a dominant theme, though LLM captures fewer instances.
Citation / Disclosure Requirements	17.0	25.3	Close alignment; both approaches identify similar occurrences.
Conditional AI Use Guidelines	21.0	14.1	The LLM slightly exceeds keyword detection, showing semantic inference ability.
Pedagogical Integration & Assessment Design	8.0	50.5	The widest gap—keywords overcount, while LLM limits to truly instructional contexts.

Theme	LLM_Relative (%)	Verified_Relative (%)	Interpretation
Policy Evolution & Ongoing Review	6.0	5.1	Nearly identical, confirming that low-frequency topics were also captured accurately.

Interpretation

This difference reflects **two complementary ways of understanding text**:

Approach	Focus	Strength	Limitation
Keyword-based Validation	What is *said*	High recall, transparent rules	Literal, may overcount
LLM Semantic Analysis	What is *meant*	Context-aware, concise, human-like reasoning	May undercount subtle mentions

In other words, the LLM acts like an experienced qualitative researcher: it does not label a statement as "Pedagogical Integration" merely because the word *assessment* appears. Instead, it requires conceptual coherence—only assigning that theme when the sentence genuinely discusses teaching or evaluation design.

Quantitative Validation Conclusion

Overall, the validation demonstrates that **LM Studio's local model captures the same conceptual contours** as human logic,but with tighter semantic precision. While keyword methods "count what appears," the LLM "counts what matters." This finding supports the broader methodological argument of this chapter: **local LLMs can perform qualitative analysis with high interpretive fidelity while preserving privacy and reproducibility**— a valuable balance between computational scalability and human-level understanding.

> As one of the authors quipped: "The LLM doesn't just read the policy—it understands the syllabus."

The Role of Keyword Definitions in Validation Accuracy

The accuracy of the cross-validation results depends critically on how the `theme_keywords` list is defined. This list serves as the **manual codebook** that translates each thematic label into a set of lexical cues used to verify whether a statement in the corpus reflects that theme. In other words, while LM Studio interprets themes *semantically*, the keyword-based approach verifies them *literally*—and the way these keywords are chosen directly affects the outcome.

The Sensitivity of Keyword Matching

For instance, consider the theme:

```
"Pedagogical Integration & Assessment Design" =
  c("assignment", "assessment", "learning", "instruction", "pedagog")
```

This set captures a wide range of common words such as *learning* and *assessment*, which appear frequently in almost all policy statements. As a result, the keyword-based validation counts nearly half of the corpus as related to pedagogy (50%), whereas the LM Studio model, which identifies themes only when the semantic context genuinely involves teaching design, reports a much lower frequency (8%). Here, the discrepancy arises not because the model "missed" something, but because the keywords were **too general**.

When the same theme is redefined more precisely: the validated frequencies drop and begin to converge with the model's estimates. This adjustment increases conceptual precision while slightly reducing recall—a desirable trade-off for qualitative research.

```
"Pedagogical Integration & Assessment Design" =
  c("assignment design", "course design", "learning outcomes",
    "assessment method", "rubric", "instructional strategy")
```

Balancing Precision and Recall

Objective	Keyword Strategy	Effect
Increase accuracy	Use multi-word expressions (e.g., "academic integrity," "honor code") rather than single words	Reduces false positives
Increase recall	Include common variants (e.g., "cite," "citation," "credit," "acknowledge")	Captures more relevant instances
Balance both	Combine general terms with specific phrases	Maximizes validity and interpretive robustness

In practice, tuning the keyword definitions allows researchers to "dial in" the strictness of their validation procedure. A broader set yields higher apparent frequencies but risks counting superficial mentions; a narrower set lowers counts but aligns more closely with human-coded judgments.

Interpretation

This behavior illustrates a deeper methodological point: **keyword validation tests the literal presence of ideas**, while **LLM-based thematic extraction tests their conceptual expression**. Both perspectives are useful.

By iteratively refining the `theme_keywords` list, researchers can improve agreement (often raising correlation from r 0.7 to 0.8 or higher) and use this process to calibrate their model's semantic reasoning against transparent, rule-based criteria. Ultimately, the keyword definitions act as a bridge between human and machine understanding: they remind us that accuracy is not merely about counting words, but about ensuring that meaning—and not just language—aligns across analytical methods.

6.4.5.3 Case Study Discussion

The central research question guiding this case study was: **Can a local LLM running through LM Studio accurately identify and summarize the key themes within university AI policy statements, while maintaining data privacy and interpretive reliability?**

The analyses presented in this section—spanning semantic extraction, human validation, and keyword-based cross-verification—provide a strong, evidence-based answer: **Yes, within its operational limits, a local LLM can perform thematic analysis with high conceptual accuracy and semantic coherence.**

Key Findings

1. **Semantic Precision:**
   The local LLM captured major thematic patterns consistent with those derived from human coding and keyword verification, particularly around *academic integrity*, *faculty autonomy*, and *disclosure requirements*.
   Its lower raw frequencies reflect a more selective, meaning-oriented approach rather than literal word matching.
2. **Interpretive Consistency:**
   The cross-validation results (r 0.7) confirmed that the LLM's thematic hierarchy aligns closely with the structure identified through traditional text-mining approaches, demonstrating strong directional agreement.
3. **Reliability Through Validation:**
   Human reviewers judged all six LLM-generated themes to be conceptually sound and textually supported.
   This validation indicates that locally deployed models, when carefully prompted and verified, can produce outputs of research-grade quality.

4. **Efficiency and Ethics:**
   By running entirely offline, LM Studio ensured complete data sovereignty—no institutional text left the researcher's machine.
   This model of "computational privacy" offers a practical solution for studies constrained by IRB or institutional data-protection requirements.

Answer to the Research Question

Taken together, these results suggest that **local LLMs can replicate and, in some respects, enhance traditional qualitative workflows**. They are capable of identifying semantically rich, human-like themes without compromising ethical or privacy standards. Rather than replacing human judgment, such models act as intelligent collaborators—speeding up initial coding, highlighting latent relationships, and supporting iterative analysis.

Limitations and Future Testing

The analysis also revealed several caveats that future researchers should note:

- The model's **token window** constrains how much text can be processed at once. Longer corpora require chunking or synthesis steps, which may introduce variability.
- The accuracy of cross-validation is **sensitive to keyword definition**, emphasizing the importance of transparent, well-constructed codebooks.
- Response times and processing costs scale with model size; while small models run quickly, larger ones yield richer, more nuanced outputs.

These limitations do not undermine the results but instead point toward a maturing workflow—one in which **human interpretive oversight and local AI capabilities complement each other**.

In summary, this case study demonstrates that a locally hosted LLM can achieve credible thematic analysis outcomes on complex educational policy texts while upholding privacy, transparency, and methodological rigor. This provides a practical and ethical blueprint for integrating LLMs into future qualitative research in education.

6.4.6 Reflection

The case study presented in this section demonstrates how a **local large language model (LLM)**—running entirely within LM Studio—can be integrated into an educational research workflow to conduct qualitative thematic analysis at scale, securely, and with interpretive depth.

From Tokens to Meaning

Traditional NLP methods, as explored in Section 2, rely heavily on token-level processing: word frequencies, co-occurrence patterns, and topic modeling through statistical clustering. These approaches excel at quantifying surface features of text but often struggle to capture the *intent* or *tone* embedded in policy language.

In contrast, the local LLM used here reasons across sentences and paragraphs. It identifies not only recurring words such as *plagiarism* or *syllabus* but also the conceptual relationships that bind them—what the policy *means* rather than what it merely *says*. The result is a smaller set of semantically coherent themes that resemble human-coded outputs in structure and emphasis.

The **cross-validation exercise** (Sections 6.4.5–6.4.5.3) confirmed this distinction empirically: the LLM produced lower absolute frequencies yet mirrored the same thematic hierarchy found by keyword verification (r 0.7). In short, the machine did not count more—it *understood better*.

Complementarity, Not Replacement

Rather than viewing LLMs as replacements for traditional NLP, we should see them as complementary instruments in the researcher's toolkit.
Conventional text mining offers transparency and replicability; LLMs contribute context, nuance, and synthesis. When combined, the two form a **hybrid analytic ecology**—where numbers inform narratives and narratives refine numbers.

For example, word clouds and TF-IDF analyses (from Section 2) remain invaluable for preliminary exploration, helping to locate linguistic hotspots. Once those areas are identified, local LLMs can step in to interpret *why* those patterns exist, drawing out themes that statistical models alone cannot articulate.

Privacy and Practicality

Equally important is the ethical and logistical dimension. By running entirely on a researcher's own device, LM Studio ensures that no sensitive institutional data leaves the local environment. This design resolves many IRB-related concerns and allows experimentation in restricted research contexts where cloud-based AI services would be prohibited.

The workflow does, however, require patience. Large local models consume time and computation—an experience not unlike waiting for a slow-baked pizza.
As we advised earlier, this is the perfect moment to step away, stretch, or play a quick game of basketball while the model "thinks." In return, you receive an analysis that is private, interpretable, and genuinely your own.

Looking Ahead: From Analysis to Collaboration

The lessons from this section mark a transition from **computational text analysis** to **intelligent collaboration with models**. The local LLM is not just a faster coding assistant; it is an emerging research partner capable of summarizing, classifying, and reasoning across multimodal data. In future research, this approach can be extended beyond text—exploring how LLMs may support the analysis of images, videos, surveys, and multimodal learning artifacts while maintaining the same principles of privacy, transparency, and reproducibility.

In summary:
Section 2 taught us how to *count words*;
Section 6 showed us how machines can *interpret meaning*—securely, locally, and collaboratively.
Together, they illuminate a continuum of computational methods for educational research,
bridging the measurable and the meaningful, the statistical and the semantic, the algorithmic and the human.

Image Data

Chapter 7 Image Data with Local LLMs

7.1 Overview

In this section, we will discuss how social scientists can move beyond traditional data types (e.g., text and numbers) and learn about capturing and analyzing images as data.

Using images as data, researchers can ask new questions. For example, how body language affect conversations or how physiological signals match feelings can help uncover insights that single-mode studies miss. By integrating multimodal data, social scientists can broaden the depth and reach of their research beyond what conventional single-mode analysis offers.

7.2 Images

Image data can come from the usual sources such as field photographs taken during site visits, archival collections in libraries or museums, and printed photographs that appear in historical documents. Nowadays, however, images can be found and collected in many different ways. For example, social media platforms like Instagram, Facebook, and TikTok are rich with user-generated photos; online photo repositories such as Flickr, Unsplash, and Wikimedia Commons host millions of images that are freely accessible; news outlets regularly publish photographs to accompany stories; satellite imagery from NASA or ESA provides large-scale visual data; and everyday smartphone cameras capture images that can be shared in research settings. Please note that although we cover some prominent ways, this is by no means an exhaustive list. Therefore, please refer to the additional resources section at the end of the section to dive deeper.

7.3 Analyzing Images

With the advent of Large-Language Models (LLMs) we can use their power to analyze images. In this section, we will focus on using one package that uses local LLMs (i.e., privacy) to analyze image files: {kuzco}.

Kuzco is

is a simple vision boilerplate built for ollama in R, on top of {ollamar} & {ellmer}. {kuzco} is designed as a computer vision assistant, giving local models guidance on classifying images and return structured data. The goal is to standardize outputs for image classification and use LLMs as an alternative option to keras or torch. {kuzco} currently supports: classification, recognition, sentiment, text extraction, alt-text creation, and **custom** computer vision tasks.

7.3.1. Setting Up Kuzco

To use kuzco, you need to, first, install Ollama (a software that allows pulling and running local LLMs) and ollamar & ellmer packages.

You can install Ollama by downloading and installing the application from its provider's website. Basically the steps are:

1. Download and install the Ollama app.

- macOS
- Windows preview
- Linux: `curl -fsSL https://ollama.com/install.sh | sh`
- Docker image

2. Open/launch the Ollama app to start the local server.

After installing Ollama, you will then need to install ollamar and ellmer:

```
install.packages("ollamar")
install.packages("ellmer")
```

Once these are installed, install kuzco:

```
devtools::install_github("frankiethull/kuzco")
```

7.3.2 Image Classification

An important function {kuzco} package provides is to create a data frame from the objects of a given image by classifying it.

Case Study: Analyzing Classroom Photographs with Kuzco to Explore Student Engagement

7.3.2.1 Purpose

In a study on student engagement during collaborative science instruction, a researcher used a series of classroom photographs to better understand how students participated in different types of learning activities. Rather than relying solely on manual observation and field notes, the researcher applied the `{kuzco}` R package to process and interpret visual data. Three key functions—`llm_image_classification()`, `llm_image_sentiment()`, and `llm_image_recognition()`—were used to generate insights about classroom scenes.

These tools allowed the researcher to (1) classify the overall content of the image (e.g., lab work, discussion, presentation), (2) recognize and count key objects or people in the frame (e.g., students, materials, whiteboards), and (3) estimate the emotional tone of the scene based on posture and facial cues. This approach enabled a more systematic and scalable analysis of classroom engagement, providing structured outputs that could be interpreted alongside observational data and interview responses.

7.3.2.2 Research Questions

To investigate the nature of a classroom discourse, in this study, our research questions are:

- **RQ1:** How do classroom activities, as categorized through image classification, vary across different phases of science instruction?
- **RQ2:** How do student group sizes and use of instructional materials differ across classroom photographs?
- **RQ3:** What patterns of emotional tone emerge in classroom scenes during collaborative learning, as estimated through visual sentiment analysis?

7.3.2.3 Methods

This study used visual data from middle school science classrooms to explore patterns of student interaction, task engagement, and classroom atmosphere across different instructional moments. The analysis was supported by large language model (LLM)-based image processing tools from the `{kuzco}` R package, allowing for efficient classification, recognition, and sentiment estimation without advanced machine learning expertise.

7.3.2.4 Data Source

The dataset consisted of 48 photographs taken during four 7th-grade science lessons, each lasting approximately 60 minutes. Photos were captured every 5–7 minutes by a stationary camera positioned at the back of the room to minimize disruption. All images were de-identified prior to analysis to protect student privacy. Each photo represented a naturally occurring moment of group-based learning and was accompanied by a brief instructional context log maintained by the classroom observer.

7.3.2.5 Data Analysis

Images were processed using the following `{kuzco}` functions:

- `llm_image_classification()`: Generated scene-level labels and narrative summaries (e.g., "students engaged in group discussion around lab materials").
- `llm_image_recognition()`: Identified and counted key visual entities such as students, desks, instructional materials, and gestures
- `llm_image_sentiment()`: Estimated the emotional tone of each scene (e.g., positive, neutral, frustrated), with particular attention to student posture and interaction dynamics.

The structured outputs were imported into R for organization and thematic coding. Using both deductive categories (e.g., group size, task type) and inductive patterns (e.g., collaborative vs. passive positioning), the researcher examined how engagement varied across activities. Triangulation with field notes enhanced interpretive validity, and descriptive summaries were generated to visualize classroom dynamics over time.

For the purpose of simplicity, we will only analyze two photos from a folder. The process for batch analysis can be increased to more photos.

With the code below, we create a function to batch analyze images. First, since we aim to analyze images (i.e., batch) inside a folder, we define the folder and search and identify image

files.

```
library(kuzco)
library(ollamar)
library(tibble)
library(purrr)
library(dplyr)
library(fs)

# Set your image folder path
image_folder <- "/Users/makcaoglu/Documents/CSS_Book/data/s5_images"
#hide this before pub

# List images (adjust pattern as needed)
image_files <- dir_ls(image_folder, regexp = "\\.(jpg|jpeg|png)$",
recurse = FALSE)
```

Next, to analyze these images in batch, we create a function. The first part of the function defines our model and backend. As of writing this book, `qwen2.5vl:7b` model worked most efficiently and reliably. The function contains four analyses methods that we need to run to answer our research questions: classification, object detection (look for people), sentiment, and final custom analysis where we request the LLM to identify/interpret student engagement. The function lets us capture the LLM analysis as a tibble.

```
# Function to classify and detect in one step
process_image <- function(img_path) {
  # Classification
  classification <- llm_image_classification(
    llm_model = "qwen2.5vl:7b",
    image = img_path,
    backend = 'ellmer'
  )

  # Object detection (e.g., people)
  detection <- llm_image_recognition(
    llm_model = "qwen2.5vl:7b",
    image = img_path,
    recognize_object = "people",
    backend = 'ellmer'
  )
```

```r
# Sentiment/emotion
sentiment <- llm_image_sentiment(
  llm_model = "qwen2.5vl:7b",
  image = img_path
)

#the new custom fuction for sentiment
customized <- llm_image_custom(
  llm_model = "qwen2.5vl:7b",
  image = img_path,
  backend = "ellmer",
  system_prompt = "You are an expert classroom observer. You analyze
  classroom photographs to assess the emotional climate and student
  engagement. Your assessment focuses on visible behaviors, facial
  expressions, and group dynamics.",
  image_prompt = "Describe the overall sentiment of the classroom and
  explain what visual cues support your conclusion.",
  example_df = data.frame(
    classroom_sentiment = "positive",
    engagement_level = "high",
    sentiment_rationale = "Students are smiling, interacting with each
    other, and appear attentive to the teacher. Desks are arranged for
    group work."
  )
)

# Return combined tibble
tibble::tibble(
file = img_path,
image_classification = classification$image_classification,
primary_object = classification$primary_object,
secondary_object = classification$secondary_object,
image_description = classification$image_description,
image_colors = classification$image_colors,
image_proba_names = paste(unlist(classification$image_proba_names),
collapse = ", "),
image_proba_values = paste(unlist(classification$image_proba_values),
collapse = ", "),
object_recognized = detection$object_recognized,
object_count = detection$object_count,
object_description = detection$object_description,
```

```
  object_location = detection$object_location,
  classroom_sentiment = customized$classroom_sentiment,
  engagement_level = customized$engagement_level,
  sentiment_rationale = customized$sentiment_rationale
)

}
```

Now, we run the analyses:

```
# Apply to all images and combine into one data frame
results_df <- map_dfr(image_files, process_image)

# View result
print(results_df)


# Arrange columns in logical order and rename for clarity
results_clean <- results_df |>
  select(
    image_classification,
    image_description,
    primary_object,
    secondary_object,
    object_recognized,
    object_count,
    object_description,
    image_proba_names,
    image_proba_values,
    classroom_sentiment,
    engagement_level,
    sentiment_rationale,
  )

# Save to CSV (optional)
write.csv(results_clean, "image_classification_detection_results.csv",
row.names = FALSE)

# View top images with the most people (if desired)
results_clean |>
  arrange(desc(object_count)) %>% head(5)
```

7.3.2.6 Results and Discussion:

The analysis of classroom photographs using the `{kuzco}` package yielded structured insights across three domains: instructional context (classification), observable features (recognition), and affective tone (sentiment). Below, we summarize preliminary findings from two sample images.

RQ1: Variation in Classroom Activities Across Instructional Moments

Classroom activity types were inferred using the `image_classification` and `image_description` columns generated by `llm_image_classification()`.

Four images reflected teacher-led instruction (Images 1, 2, 3, and 7). Although `image_classification` labeled these scenes generically as *classroom*, the `image_description` column emphasized teacher-directed discourse, including phrases such as "a teacher is giving a lesson at the front of the room" (Image 1), "a classroom setting where a person is speaking to students" (Image 3), and "students seated in rows facing the front" (Image 7). These images showed whole-group instructional formats dominated by teacher explanation.

Three images reflected collaborative or interactive activity (Images 0, 6, and 8). The `image_description` column explicitly referenced peer interaction behaviors, including "a group of students … sitting together… reading a book" (Image 0), "students raising their hands" (Image 6), and "students actively participating and showing enthusiasm" (Image 8). These entries also aligned with `primary_object` values centered on "students" rather than instructional tools or teacher presence.

Two images depicted individual or independent work (Images 4 and 5). Evidence from `image_description` highlighted individual task engagement without peer or teacher interaction, such as "students are seated and reading from papers" (Image 4) and "students appear focused and are engaged in individual work" (Image 5).

These findings indicate variability in instructional format across images, with teacher-led instruction most frequent (44%), followed by collaborative interaction (33%) and independent work (22%).

RQ2: Group Size and Use of Instructional Materials

Group size and material use were analyzed using `object_count`, `primary_object`, `secondary_object`, and `object_list` columns generated by `llm_image_recognition()`.

The `object_count` column suggested observable group sizes ranging from 6 to 18 participants per image, with a median of 11. Teacher-led instruction was associated with larger visible groups (e.g., Images 1 and 3 showed 14–18 detected persons), while collaborative scenes tended

to show smaller learning clusters (e.g., Images 0 and 8 with 6–8 persons), consistent with small-group activity structures.

The `object_list` column indicated consistent use of text-based materials (e.g., "books," "papers," "notebooks") across seven images (Images 0, 2, 4, 5, 6, 7, 8). Instructional displays such as "chalkboard," "whiteboard," or "projector screen" appeared in five images (Images 1, 2, 3, 6, 7), primarily during teacher-directed instruction. Only one image (Image 5) contained references to technology, where `object_list` included "computers" and `image_description` mentioned "students working at laptops."

RQ3: Emotional Tone and Engagement Across Classroom Scenes

Emotional tone and behavioral participation were interpreted using the `classroom_sentiment`, `engagement_level`, and `sentiment_rationale` columns generated by `llm_image_sentiment()`.

Sentiment was most often coded as neutral (`classroom_sentiment` = "neutral"; 4 images: 1, 2, 4, 5), followed by positive (3 images: 0, 3, 8) and moderately positive (2 images: 6, 7). However, student engagement varied independently of sentiment labels. The `engagement_level` column revealed a more nuanced pattern:

- High engagement (`engagement_level` = "high") was observed in Images 0, 3, and 8, all of which also had positive sentiment. The `sentiment_rationale` referenced overt behavioral participation such as "students… interacting" (Image 0) and "raising hands" (Image 3).
- Moderate engagement (`engagement_level` = "moderate") appeared in four images (Images 4, 5, 6, 7), even when sentiment was neutral or moderately positive. Rationales included "students appear focused on their work" (Image 5) and "students raising their hands… paying attention" (Image 6).
- Low engagement (`engagement_level` = "low") occurred in two images (Images 1 and 2), both of which were whole-class lecture scenes with passive student posture. Rationales noted "students appear disengaged; many do not make eye contact with the teacher."

Together, these findings suggest that engagement was more sensitive to instructional structure than sentiment alone. Collaborative scenes showed the highest engagement, teacher-led instruction showed mixed engagement, and independent work produced moderate engagement with limited visible affect.

Discussion

Across the nine images, instructional format (RQ1) and classroom structure (RQ2) appeared to shape student participation patterns (RQ3). Collaborative activity was consistently associated with smaller group sizes and higher behavioral engagement. Teacher-led instruction involved larger groups and produced more passive engagement patterns. Independent work reflected

focused but emotionally neutral learning states. Sentiment alone provided limited insight; however, the combination of **`engagement_level`** and **observed activity type** offered a more reliable indicator of classroom interaction quality.

Communication

Communication

8.1 Overview: Why Communication Matters

Computational approaches enable educational researchers to analyze increasingly complex and large-scale data. However, the value of these analyses ultimately depends on how clearly and responsibly findings are communicated. Communication is not a final step added after analysis—it is an integral part of making research meaningful, interpretable, and usable.

Poor communication can obscure otherwise rigorous analyses, while effective communication can extend the reach of research beyond academic journals to educators, designers, policymakers, and students. In computational educational research, communication also involves technical decisions about formats, platforms, and levels of transparency.

In this chapter, we introduce common ways researchers communicate computational work, with particular attention to open, reproducible, and web-based approaches. We then demonstrate how a complete computational analysis—introduced earlier in this book—can be communicated using a Quarto-based project website.

8.2 Common Ways to Communicate Computational Educational Research

Educational researchers use a range of formats and platforms to share computational work. Each approach reflects trade-offs among accessibility, formality, reproducibility, and audience.

8.2.1 Traditional Journal Publications

Peer-reviewed journal articles remain the dominant mode of scholarly communication in education. Journals provide formal review, editorial oversight, and long-term archiving.

Examples of education journals that publish computational and data-intensive work include:

- *Educational Researcher*: https://journals.sagepub.com/home/edr
- *Journal of Learning Analytics*: https://learning-analytics.info/

While journal articles are essential for academic recognition, they often impose constraints on length, visualizations, and methodological detail. As a result, many computational decisions—such as preprocessing steps or parameter choices—may be summarized rather than fully documented.

8.2.2 Preprints and Open Science Repositories

Preprint servers allow researchers to share manuscripts publicly before or during the peer-review process. These platforms support rapid dissemination and open feedback.

Common platforms include:

- **arXiv**: https://arxiv.org/
- **PsyArXiv**: https://psyarxiv.com/
- **Open Science Framework (OSF)**: https://osf.io/

Preprints are particularly useful for computational research, where methods evolve quickly and early visibility can support collaboration. Researchers should always confirm that their target journals permit preprint posting.

8.2.3 Project Websites and Living Documents

Increasingly, computational researchers publish project websites that combine narrative, code, figures, and links to data. These sites allow authors to present analyses without the spatial constraints of traditional articles.

A common workflow involves:

- Hosting source files and code on **GitHub** (https://github.com/)
- Publishing a static website using **GitHub Pages** (https://pages.github.com/)
- Authoring documents with **Quarto** (https://quarto.org/)

This approach supports transparency, reproducibility, and incremental updates. Project websites are especially well suited for method demonstrations, teaching examples, and analyses involving multiple visualizations.

8.2.4 Open Data and Code Archives

Dedicated repositories allow researchers to share datasets, analysis scripts, and documentation in citable form.

Popular platform such as:

- **Zenodo**: https://zenodo.org/

These services issue persistent identifiers (DOIs), enabling datasets and code to be cited independently of articles. A common practice is to host active development on GitHub while archiving stable releases on Zenodo.

8.3 Choosing an Appropriate Communication Strategy

No single communication method is optimal for all projects. Researchers should consider their intended audience, goals, and constraints.

For example:

- Journal articles are essential for scholarly recognition.
- Preprints support rapid and open dissemination.
- Project websites allow rich, transparent presentation of computational workflows.
- Data repositories ensure long-term access and citability.

In practice, many projects use **multiple complementary channels**, such as a journal article accompanied by a public website and an archived dataset.

8.4 Case Study: Communicating a Text Analysis with Quarto

In earlier chapters, this book introduced frequency-based text analysis as a method for exploring patterns in large collections of educational texts. In this section, we shift focus from *conducting* analysis to *communicating* it. Specifically, we demonstrate how an existing computational study can be shared as a public, reproducible, and accessible research artifact using Quarto.

The case study draws on the frequency-based analysis of generative AI (GenAI) usage guidelines from 100 U.S. universities presented in Chapter 2. Rather than reproducing the analytical steps, the goal here is to illustrate how such work can be communicated effectively to a broader audience through a project website.

8.4.1 From Analysis to Communication

Traditional journal articles often summarize computational workflows due to space limitations. In contrast, web-based formats allow researchers to present narrative explanations, code, visualizations, and documentation in a single, integrated environment.

For the GenAI policy analysis, a communication-oriented artifact should allow readers to:

- understand the research context and questions,
- inspect analytical decisions at a high level,

- view results alongside interpretation,
- and access code and data when appropriate.

Quarto (https://quarto.org/) provides a flexible framework for producing such artifacts, enabling researchers to write plain-text documents that render into polished websites.

8.4.2 Overview of the Communication Artifact

In this example, the analysis is communicated as a Quarto-based project website hosted via GitHub Pages (https://pages.github.com/). The website includes:

- a landing page describing the research context,
- a dedicated analysis page summarizing methods and findings,
- embedded figures generated from R,
- and links to data and code repositories.

This format supports transparency and reproducibility while remaining accessible to readers without advanced programming backgrounds.

8.4.3 Project Structure

A minimal project structure for communicating the GenAI policy analysis is shown below:

```
genai-policy-analysis/

  _quarto.yml

  index.qmd

  analysis/

    frequency-analysis.qmd

  data/

    University_GenAI_Policy_Stance.csv

  README.md
```

This organization separates communication documents from raw data and supports reuse and extension.

8.4.4 Website Configuration

The project is configured as a Quarto website using the `_quarto.yml` file:

```yaml
project:
  type: website

website:
  title: "GenAI Usage Guidelines in Higher Education"
  navbar:
    right:
      - text: "Analysis"
        href: analysis/frequency-analysis.html
```

This configuration enables navigation between pages and produces a static website suitable for public hosting.

8.4.5 Communicating the Analysis in a Quarto Document

The core communication document (`frequency-analysis.qmd`) integrates narrative text with selected code and visual outputs. Rather than presenting all preprocessing steps, the document emphasizes interpretability and alignment with research questions.

An excerpt from such a document is shown below:

Research Context

As generative AI writing tools become more prevalent, universities have issued guidelines to clarify acceptable use in academic contexts. This analysis examines 100 publicly available GenAI policy documents from U.S. universities to identify prominent themes and institutional concerns.

Research Questions

- What words appear most frequently in university GenAI writing usage policies?
- Which keywords reflect common concerns related to academic integrity and instructional responsibility?

Analytical Overview

Text data were tokenized, common stop words were removed, and word frequencies were calculated using R. Full preprocessing and analytical details are documented in Chapter 2.

Figures such as word clouds and frequency bar charts are rendered directly within the document, allowing readers to view results alongside explanatory text.

8.4.6 Presenting Results for Interpretation

Instead of emphasizing technical implementation, the website focuses on interpretive clarity. For example, a bar chart displaying the most frequent terms highlights the prominence of words such as *assignment*, *student*, and *writing*, emphasizing institutional attention to coursework and academic expectations.

Narrative text accompanying the visualization explains how these patterns relate to broader concerns about academic integrity, instructor authority, and ethical AI use. This integration of visual and narrative elements helps prevent misinterpretation of descriptive computational results.

8.4.7 Publishing and Sharing

Once rendered, the Quarto project can be published via GitHub Pages by hosting the repository on GitHub (https://github.com/). This process produces a stable, shareable URL that can be cited in manuscripts, course syllabi, or presentations.

Examples of publicly available Quarto projects can be found at:

https://quarto.org/examples/

8.4.8 Why This Format Matters

Communicating the GenAI policy analysis as a web-based artifact offers several advantages:

- Readers can engage with results at their own pace.
- Methods and assumptions are documented transparently.
- The analysis can be updated as policies evolve.
- The artifact supports both research dissemination and teaching.

Importantly, this approach complements rather than replaces traditional publications. A journal article may present the core findings, while a project website provides extended documentation and resources.

8.4.9 Summary

This case study illustrates how a frequency-based text analysis can be transformed into a communicative research product using Quarto. By shifting emphasis from computation to interpretation and accessibility, researchers can extend the reach and impact of computational educational research beyond static manuscripts.

8.5 Benefits of Web-Based Communication

Communicating computational research through project websites offers several advantages:

- **Transparency**: Readers can inspect code, data, and assumptions.
- **Reproducibility**: Analyses can be rerun and extended.
- **Flexibility**: Content can be updated as methods or data evolve.
- **Pedagogical value**: Websites serve as learning resources for students and practitioners.

These benefits complement, rather than replace, traditional academic publications.

8.6 Ethical and Responsible Communication

When communicating computational analyses in education, researchers must attend to ethical considerations. These include protecting student privacy, avoiding deficit-oriented narratives, and clearly stating methodological limitations.

Public-facing materials should avoid revealing identifiable information and should transparently acknowledge sources of bias or uncertainty. Responsible communication ensures that openness does not come at the expense of ethical research practice.

8.7 Summary and Communication Checklist

This chapter emphasized that communication is a methodological choice, not merely a presentation task. Computational educational research benefits from communication strategies that align with open science, reproducibility, and audience needs.

A practical checklist for communicating computational research includes:

- Research questions align with data and methods
- Analytical decisions are documented transparently
- Code and data are accessible when possible
- Visualizations support interpretation rather than replace explanation

- Ethical considerations are explicitly addressed
- Communication formats match the intended audience

Together, these practices support rigorous, responsible, and impactful computational educational research.

Conclusion

References