Computational Social Science Cookbook with R

Wei Wang, Mete Akcaoglu, Joshua Rosenberg, Shaun Kellogg

2025-02-16

Table of contents

Preface

Preface

Welcome to *Computational Social Science Cookbook with R*!

The book is organized around six sections. Within these six sections are specific chapters, which serve as cookbook "entries". While the section overviews (the first bullet point within each section) introduce the techniques or methodologies used in the section's chapters, the entries (the subsequent bullet points) are intended to address a specific, narrow problem, as well a to provide a sample for researchers in writing their research questions, methods, results (and discussions) sections based on the analyses.

Section 1 Getting Started

Abstract: In this section, we will introduce the steps to get started with using R and RStudio. The section will also cover the key concepts in working with R such as packages, R markdown. We will also cover important steps in preparing data, which we will be using frequently in the rest of the book. Since these topics have been covered in so much more detail in other books, or other resources online, we will end the section with a list of key resources on these topics that are easily accessible online.

- Section Overview
- Using R and RStudio
  - Foundational skills: Data, packages, functions, projects
  - Reproducibility and R Markdown documents
  - Base R: assignment, equality, dollar-sign indexing, xx
  - R files
  - Project setup within RStudio
- Core Data Wrangling Capabilities
  - Selecting and renaming variables
  - Creating new variables
  - Arranging variables
  - Joining data

Section 2 Capturing and Analyzing Text Data with Computational Methods

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 sources (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. Modern data collection techniques include web scraping, accessing social media APIs, or downloading large online documents. Given the increased size of text data, analysis now requires computational approaches (e.g., dictionary-based, frequency-based, machine learning) that go beyond manual coding. 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 (Broadening the Horizon)

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.

```r
# 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**.

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

# 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:

```
# 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()`:

```
# 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:

```
# 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.

```r
# 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 = TRU
```

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.

```r
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.

```r
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.

```r
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.

```r
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.

```r
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()`.

```r
# 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.

```r
# 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.

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

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

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

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

```r
# 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.

```r
# 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

2.3.1 Purpose + Case

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 (Saldana) 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 current allowances. 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.

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.

2.3.2 Sample Research Questions

- **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 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" "wor

# 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.

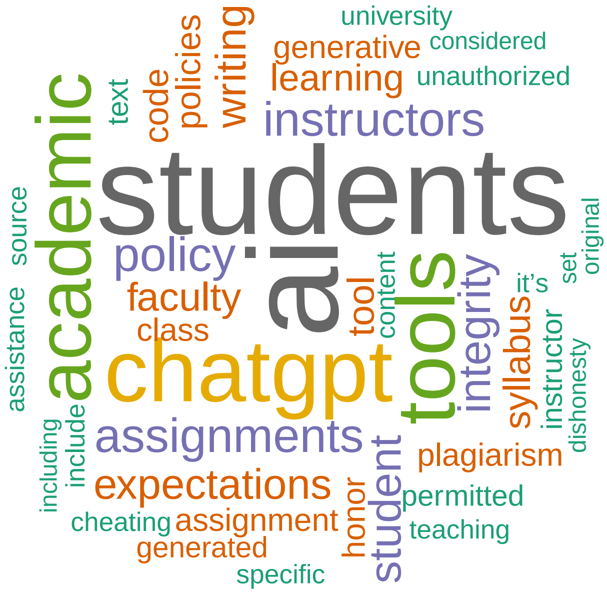
```
# Tokenize text, remove stop words, and count word frequencies
word_frequency <- university_policies %>%
  unnest_tokens(word, Stance) %>%  # Tokenize the 'Policy_Text' column
  anti_join(stop_words) %>%             # Remove common stop words
  count(word, sort = TRUE)              # Count and sort 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.

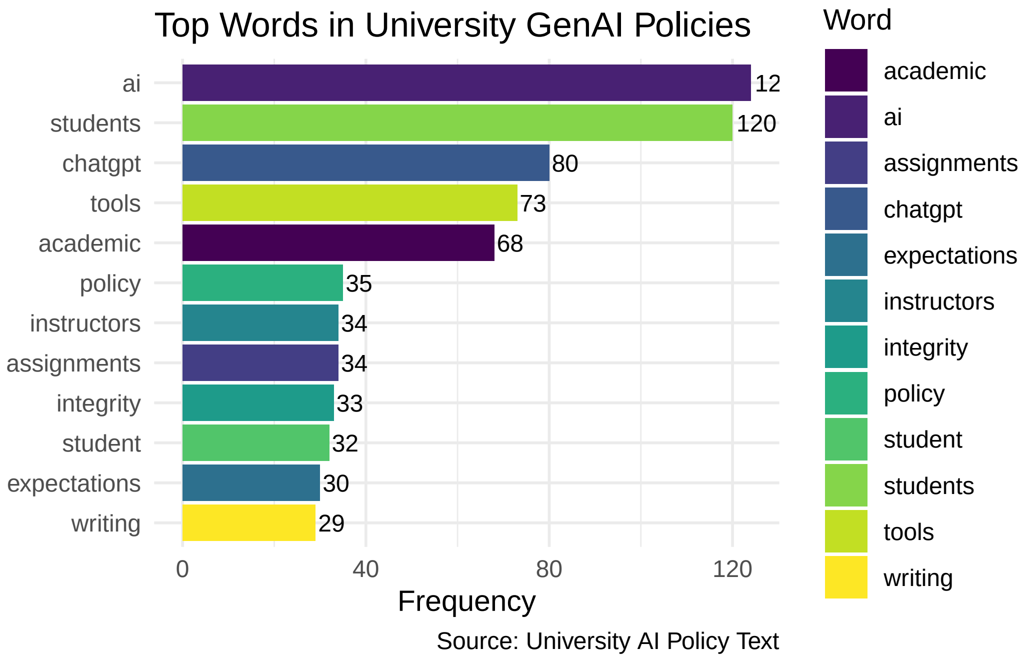
```
# Create and display the GenAI usage Stance wordcloud

wordcloud(words = word_frequency$word, freq = word_frequency$n, scale = c(4, 0.5), random.or
```



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)
```



2.3.4 Results and Discussions

- **RQ1**: What are the most frequently mentioned words in university GenAI writing usage policies?
  The frequency analysis shows that keywords such as "assignment," "student," and "writing" are 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 highlights 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 + Case

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
```

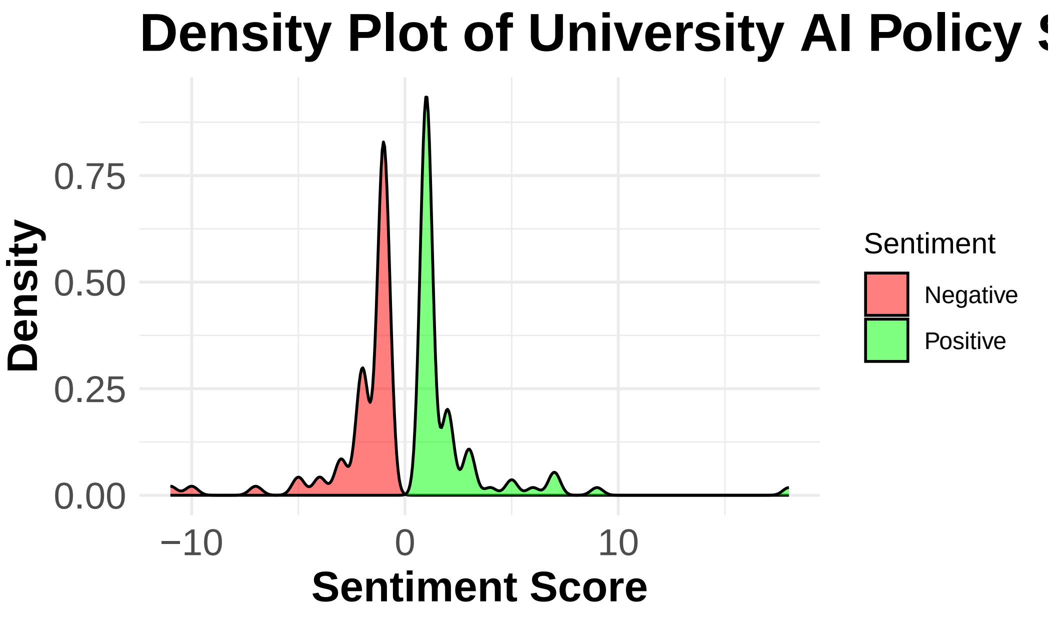
```
# A tibble: 142 x 4
   word         positive negative sentiment_score
   <chr>           <int>    <int>           <int>
 1 honor              18        0              18
 2 cheating            0       11             -11
 3 dishonesty          0       10             -10
 4 guidance            9        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.

```
# 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)
```



Source: University Policy Text

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 + CASE

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.

```r
# 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")

# 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
```

```
# 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")
```

```
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.

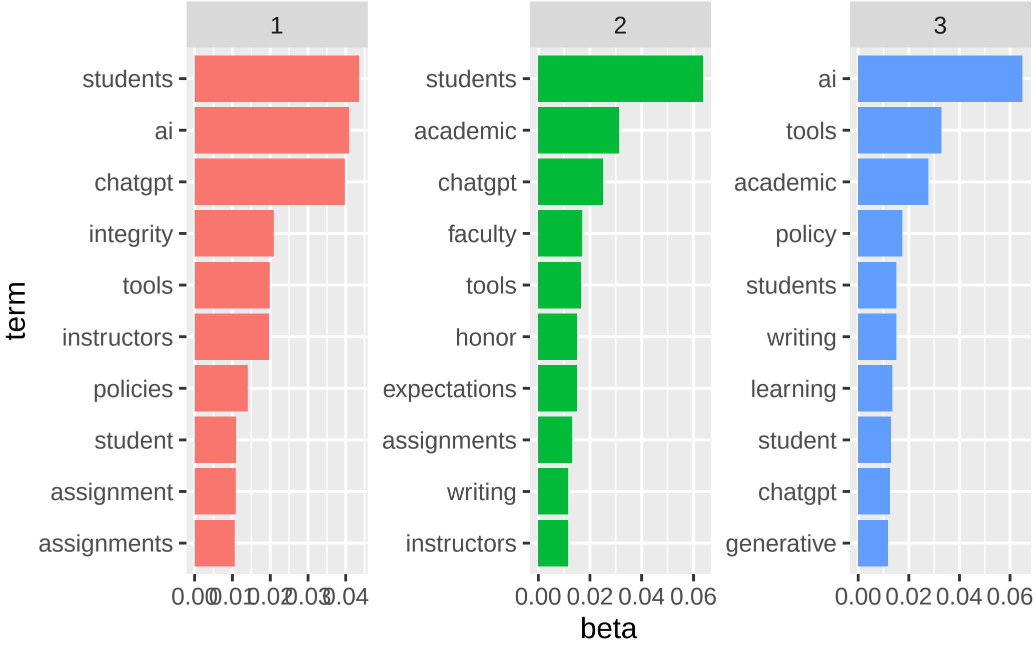
```
# Converting document-word frequency matrices to sparse matrices
gpt_dtm_sparse<- as(gpt_dtm, "matrix")

# 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.

```
# 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.

Section 3 Multimodal Data (Images, Video, Audio)

Abstract: Conventionally, for us (the social scientists) we start the research process by generating research questions based on our previous knowledge and theories in the field and that is still the way to go about it. But, it is also possible that our observations of the world can guide our research questions. In this section, we will discuss how social scientists can look beyond conventional data types, and learn about capturing and analyzing multimodal data.

Section 4 Social Network Analyses (Relational Data)

Abstract: 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.

Section 5 Secondary Analysis of Big Data (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)).

5.1 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.

5.2 Accessing Big data (Broadening the Horizon)

5.2.1 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()`.

```
# 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.

```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.

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

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

5.2.2 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.

5.3 Logistic Regression ML

5.3.1 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.

5.3.2 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?

5.3.3 Analysis

Loading Required Packages

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

```r
# 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.

```r
# 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

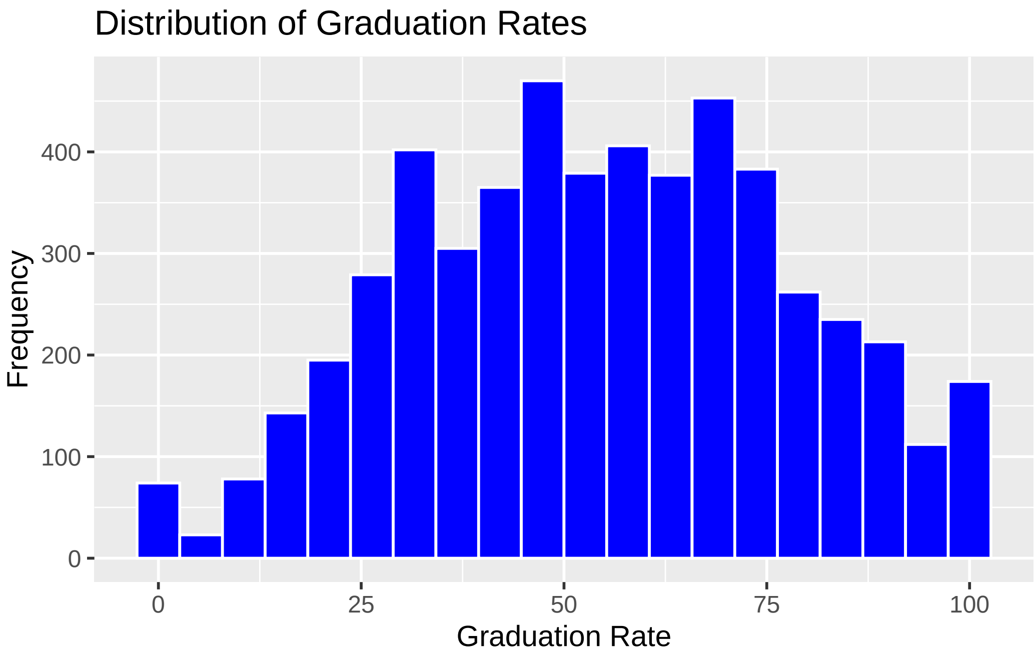
Select relevant variables, filter the dataset, and create the dependent variable `good_grad_rate`.

```
# 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_fina
    avg_salary = drvhr2022_average_salary_equated_to_9_months_of_full_time_instructional_sta
  ) %>%
  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.

```
# 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_ai

# 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) %>%
  bind_cols(ipeds) # Combine predictions with original data

# Calculate and display accuracy
```

```r
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
```

5.3.4 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, $p < 2e\text{-}16$
    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.

5.4 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).

5.4.1 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).

5.4.2 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?

5.4.3 Analysis

Loading Required Packages

Below we load the necessary packages. (Note: All code chunks are set to `eval=FALSE` for instructional purposes.)

```
# 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_submitte
```

```
code_module_dates <- assessments %>%
    group_by(code_module, code_presentation) %>%
    summarize(quantile_cutoff_date = quantile(date_submitted, probs = 0.25, na.rm = TRUE), .g

# 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 = N
        warning = function(w) { tibble(term = NA, estimate = NA, std.error = NA, statistic =
    )
}

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_s

# Finally, join with students_and_assessments to get the outcome variable
students_assessments_and_interactions <- left_join(students_and_assessments, interactions_fea

# 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	549167.31	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	
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.

```r
# 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 0.00288 Preprocessor1_Model1
2 kap         binary     0.267     4 0.00580 Preprocessor1_Model1
3 npv         binary     0.591     4 0.00569 Preprocessor1_Model1
4 ppv         binary     0.704     4 0.00166 Preprocessor1_Model1
5 sensitivity binary     0.814     4 0.00427 Preprocessor1_Model1
6 specificity binary     0.440     4 0.00355 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.

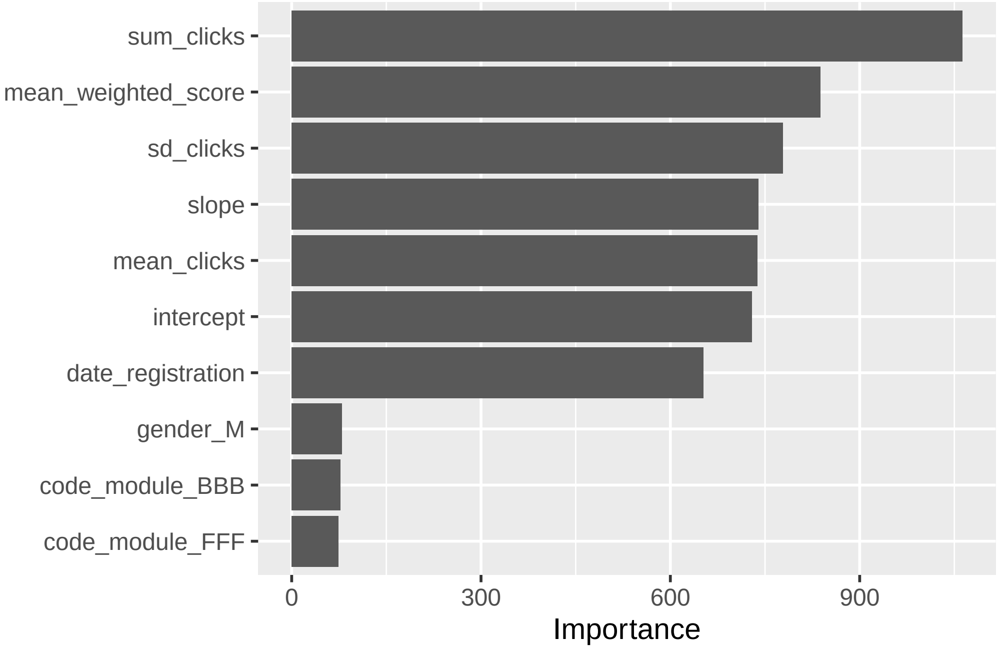
```r
# Split data into training and testing sets (e.g., 33% for testing)
set.seed(20230712)
train_test_split <- initial_split(students_assessments_and_interactions, prop = 0.67, strata
data_train <- training(train_test_split)
data_test <- testing(train_test_split)

# 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
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
```

5.4.4 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.

Section 6 Communication Collaboration Practices

Abstract

A key part of social science research (and, any analysis involving computational tools) is communicating about the output or findings. In this section, we describe how to communicate with colleagues or the wider world through the use of a variety of tools, especially R Markdown and git/GitHub. We also discuss how to collaborate on projects that involve computational methods and "good" (flexible yet principled) practices for doing so based on our experience and prior scholarship.

Conclusion

References