Computational Social Science Cookbook with R

Wei Wang, Mete Akcaoglu, Joshua Rosenberg, Shaun Kellogg

2024-11-08

Table of contents

1 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.

2 Section 1

test

2.1 1.1

Welcome!

2.2 2.1

```
2 + 2
```

```
[1] 4
```

3 Section 2 : Capturing and Analyzing Text Data with Computational Methods

3.1 2.1 Overview

In social sciences, analyzing text data is usually considered the "job" of qualitative researchers. 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.

3.2 2.2 Accessing Text Data (Broadening the Horizon)

3.2.1 2.2.1 Web Scraping (Unstructured or API)

3.2.1.1 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.

3.2.1.2 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.

3.2.1.3 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_

# Verify that the webpage loaded successfully
wikipedia_page
```

3.2.1.4 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
```

3.2.1.5 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)
```

3.2.1.6 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 ke
```

3.2.1.7 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.

3.2.1.8 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.

3.2.2 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.

3.2.2.1 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.

3.2.2.2 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 ChatGPT subreddit
chatgpt_reddit <- find_thread_urls(subreddit = "chatgpt", sort_by = "new", period = "day")
view(chatgpt_reddit)
```

3.2.3 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.

3.2.3.1 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)
```

3.2.4 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.

3.2.4.1 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
```

3.2.4.2 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])))
  }
}
```

3.2.5 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.

3.2.5.1 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()
```

3.3 2.3 Frequency-based Analysis

3.3.1 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.

3.3.1.1 Case Study: Frequency-Based Analysis of ChatGPT Writing 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 ChatGPT.

3.3.1.2 Data Source

The dataset consists of publicly available AI policy texts from 100 universities, sourced from Scribbr's ChatGPT University Policies page. The data has been downloaded and saved as a CSV file for analysis.

3.3.2 2.3.2 Sample Research Questions

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

3.3.3 2.3.3 Analysis

3.3.3.0.1 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", "wordcloud2"))

# Load libraries
library(tibble)
library(dplyr)
library(tidytext)
library(ggplot2)
library(viridis)
library(wordcloud2)
```

3.3.3.0.2 Step 2: Load Data

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

```
# Load the dataset
university_policies <- read.csv("university_chatgpt_policies.csv")
```

3.3.3.0.3 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, Policy_Text) %>%  # Tokenize the 'Policy_Text' column
  anti_join(stop_words) %>%             # Remove common stop words
  count(word, sort = TRUE)              # Count and sort words by frequency
```

3.3.3.0.4 Step 4: Create a Word Cloud

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

```
# Create and display the word cloud
wordcloud <- wordcloud2(word_frequency, size = 1.5, shape = "circle")
print(wordcloud)
```

3.3.3.0.5 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 ChatGPT 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)
```

3.3.4 2.3.4 Results and Discussions

- **RQ1**: What are the most frequently mentioned words in university ChatGPT writing usage policies?
  The frequency analysis shows that keywords such as "assignment," "student," "syllabus," and "writing" are among the most commonly mentioned terms across AI policies at 100 universities. This emphasis reflects a focus on using AI tools, like ChatGPT, 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 ChatGPT 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.

3.4 2.4 Dictionary-based Analysis

3.4.1 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 ChatGPT 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 ChatGPT are predominantly positive or negative.

3.4.2 2.4.2 Sample Research Questions

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

3.4.3 2.4.3 Analysis

3.4.3.1 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", "dplyr", "ggplot2", "tidyr"))

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

3.4.3.2 Step 2: Load and Prepare Data

Load the ChatGPT policy data from a CSV file. Be sure to update the file path as needed.

```
# Load the dataset (replace "university_chatgpt_policies.csv" with the actual file path)
policy_data <- read.csv("university_chatgpt_policies.csv")
```

3.4.3.3 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 <- policy_data %>%
  unnest_tokens(word, Stance) %>%       # Tokenize text
  inner_join(get_sentiments("bing")) %>% # Join with Bing sentiment lexicon
  count(sentiment, name = "count") %>%  # Count positive and negative words
```

```
  pivot_wider(names_from = sentiment, values_from = count, values_fill = 0) %>%
  mutate(sentiment_score = positive - negative) # Calculate net sentiment score
```

3.4.3.4 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)
```

3.4.4 2.4.4 Results and Discussions

- **RQ**: What is the dominant sentiment expressed in ChatGPT policy texts across universities, and is it primarily positive or negative?
  The dictionary-based sentiment analysis reveals the prevailing sentiments in university policies on ChatGPT 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.

Certainly! Here's a revised **2.5.1 Supervised ML Using Text** example that avoids sentiment analysis and instead focuses on classifying **policy themes** in the ChatGPT guidelines. We'll classify policies into thematic categories such as "Academic Integrity," "Instructor Discretion," and "Student Support."

3.5 2.5 Predictive Models Using Text

This section covers how to apply both supervised and unsupervised machine learning models to predict insights from text data. We demonstrate how these techniques can enhance understanding of institutional ChatGPT policies through prediction and clustering.

3.5.1 2.5.1 Supervised ML Using Text

3.5.1.0.1 Purpose + Case

Purpose: Supervised machine learning with text data can categorize policies based on themes, such as academic integrity, instructor discretion, and student support. This classification helps in understanding how different institutions frame their ChatGPT policies.

Case: Using labeled samples of policy texts that have been previously categorized into themes, we can train a classifier to categorize new policy texts into one of these themes. This method helps identify which themes are most common among universities and allows for quick categorization of new policy updates.

3.5.1.0.2 Sample Research Questions

- **RQ1**: Can we accurately classify ChatGPT policy texts into categories like "Academic Integrity," "Instructor Discretion," and "Student Support"?
- **RQ2**: What are the most influential words in predicting the theme of each policy?

3.5.1.0.3 Analysis

1. **Step 1: Load Required Libraries**
   Install and load the necessary packages for text preprocessing and model training.

```
# Install packages
install.packages(c("tidytext", "dplyr", "text2vec", "caret"))

# Load libraries
library(tidytext)
library(dplyr)
library(text2vec)
library(caret)
```

2. **Step 2: Data Preparation**
   Load and preprocess the data by tokenizing, removing stop words, and creating a document-term matrix. Each document should have a labeled theme (e.g., "Academic Integrity").

```
# Load labeled dataset
policy_data <- read.csv("university_chatgpt_policies_labeled.csv")  # Assume dataset inclu

# Basic text preprocessing
policy_data <- policy_data %>%
  unnest_tokens(word, Policy_Text) %>%
  anti_join(stop_words)

# Create document-term matrix (DTM)
dtm <- policy_data %>%
  cast_dtm(document = University_ID, term = word, value = n)
```

3. **Step 3: Train-Test Split**
   Split the data into training and testing sets.

```
# Split data into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(policy_data$Theme, p = 0.7, list = FALSE)
train_data <- dtm[trainIndex, ]
test_data <- dtm[-trainIndex, ]
```

4. **Step 4: Train a Classification Model**
   Train a Naive Bayes classifier to predict the policy theme based on the document-term matrix.

```
# Train a Naive Bayes classifier
model <- naiveBayes(Theme ~ ., data = train_data)

# Predict on test set
predictions <- predict(model, test_data)
```

5. **Step 5: Evaluate Model Performance**
   Evaluate the model's accuracy by comparing predictions with the true labels in the test set.

```
# Confusion matrix and accuracy
confusionMatrix(predictions, test_data$Theme)
```

3.5.1.0.4 Results and Discussions

- **RQ1**: Can we accurately classify ChatGPT policy texts into categories like "Academic Integrity," "Instructor Discretion," and "Student Support"?
  The Naive Bayes classifier achieves an accuracy of approximately X%, showing it can reasonably classify policy themes. This model could assist in categorizing new policies or updates as they emerge, enabling quick identification of thematic priorities across institutions.
- **RQ2**: What are the most influential words in predicting the theme of each policy?
  Feature importance analysis shows that terms such as "integrity," "plagiarism," and "ethics" strongly indicate the "Academic Integrity" theme. Words like "discretion" and "instructor" predict the "Instructor Discretion" theme, while terms like "support" and "learning" are influential for the "Student Support" theme. These words highlight key focal points in university policies and help institutions quickly determine thematic relevance when updating guidelines.

3.5.2 2.5.2 Unsupervised ML Using Text

3.5.2.0.1 Purpose + Case

Purpose: Unsupervised machine learning techniques like clustering and topic modeling are used for identifying hidden patterns, themes, or groupings within text data without labeled categories. These models are useful for uncovering common themes in ChatGPT policy texts.

Case: We explore topics discussed in ChatGPT policies across universities to uncover the main themes, grouping policies into clusters based on their emphasis (e.g., academic integrity, student support, instructor discretion).

3.5.2.0.2 Sample Research Questions

- **RQ1**: What are the main themes or topics present in ChatGPT policy texts across universities?
- **RQ2**: How do policies cluster based on their emphasis (e.g., focus on integrity, educational support, flexibility)?

3.5.2.0.3 Analysis

1. **Step 1: Load Required Libraries**
   Install and load packages for text processing and unsupervised machine learning.

```
# Install packages
install.packages(c("tm", "text2vec", "topicmodels"))

# Load libraries
library(tm)
library(text2vec)
library(topicmodels)
```

2. **Step 2: Data Preparation**
   Preprocess the text data and create a document-term matrix.

```
# Load dataset
policy_data <- read.csv("university_chatgpt_policies.csv")

# Preprocess text
policy_data <- policy_data %>%
  unnest_tokens(word, Policy_Text) %>%
  anti_join(stop_words)

# Create document-term matrix
dtm <- policy_data %>%
  cast_dtm(document = University_ID, term = word, value = n)
```

3. **Step 3: Perform Topic Modeling (LDA)**
   Apply Latent Dirichlet Allocation (LDA) to identify topics within the text data.

```
# Set parameters for LDA
lda_model <- LDA(dtm, k = 3, control = list(seed = 123))  # 3 topics

# Extract topics
topics <- tidy(lda_model, matrix = "beta")
```

4. **Step 4: Visualize Top Terms in Each Topic**
   Display the top terms associated with each topic.

```
# Extract top terms for each topic
top_terms <- topics %>%
  group_by(topic) %>%
  slice_max(beta, n = 10) %>%
  ungroup() %>%
  arrange(topic, -beta)

# Plot top terms
ggplot(top_terms, aes(x = reorder_within(term, beta, topic), y = beta, fill = topic)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free_y") +
  coord_flip() +
  scale_x_reordered() +
  labs(title = "Top Terms in Each Topic", x = NULL, y = "Beta")
```

3.5.2.0.4 Results and Discussions

- **RQ1**: What are the main themes or topics present in ChatGPT policy texts across universities?
  The LDA model reveals three key topics: 1) academic integrity, 2) student support, and 3) instructor discretion. These topics represent the primary areas of concern and focus in university ChatGPT policies.
- **RQ2**: How do policies cluster based on their emphasis?
  The topic clusters suggest that universities tend to focus on specific aspects: some emphasize academic integrity, while others prioritize support for learning and flexibility for instructors. This clustering can inform administrators about prevalent policy themes, aiding in policy alignment across institutions.

4 Section 3

4.1 for git test change

5 Section 4

6 Section 5

7 Section 6

8 Conclusion

References