AMERICAN INTERNATIONAL UNIVERSITY BANGLADESH

Faculty of Science And Technology

Project Report Cover Sheet

Project Title: Scrap data from a news portal, apply suitable data preparation steps, and find the most discussed topics from the collected news content.

Due Date: 25 May 2025 Semester: Spring 24-25

Course Title: Introduction To Data Science Section: A

Course Instructor: **DR. ABDUS SALAM** Group Number: 09

No.	Student Name	Student ID
1	Morshed Al-Jaber Bishal	21-45834-3
2	Sanviraj Aynul Siam	21-45834-3
3	NAHID HASAN	22-46543-1
4	Tanvir Arafat	21-45692-3

ids_final_project_group_09_news_raw

Introduction

The CNN News Portal Scraper is a data collection tool designed to systematically extract and compile news articles from CNN's official website. Utilizing the power of R and key web scraping packages (rvest, httr, xml2, dplyr, and stringr), this project focuses on gathering real-time headlines, publication dates, authorship, summaries, and URLs from five major news categories: World, Business, Health, Entertainment, and Sport. The objective is to build a structured dataset that reflects current global events and media coverage trends, which can be used for further analysis such as topic modeling, sentiment analysis, media bias tracking, or public opinion mining. This project successfully collects up to 100 valid news articles from each selected category on CNN's website. For each article, the scraper extracts the title, description, publication date, author, and direct URL. Articles are filtered and cleaned to ensure accuracy and consistency. The system gracefully handles missing data and page errors using robust error handling via tryCatch. All collected data is compiled into a comprehensive data frame and exported as a CSV file for further analysis. The final output represents a rich and diverse news dataset ideal for academic research, data science projects, and media trend visualization.

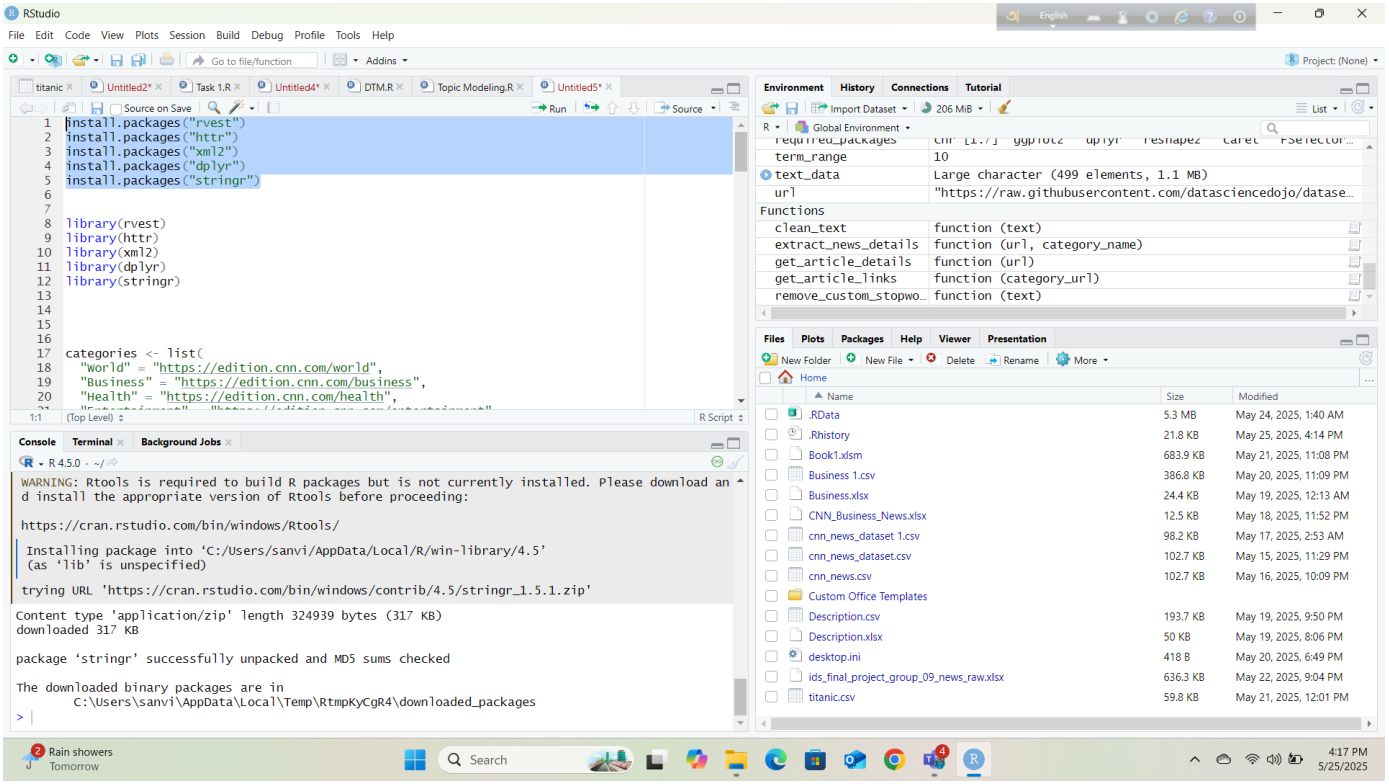
Dataset Summery

The cleaned CNN news dataset comprises a total of 500 articles across five major categories: World, Business, Health, Entertainment, and Sport, each contributing nearly 100 articles, ensuring a balanced representation. The dataset spans a date range from April 2024 to May 2025, capturing over a year's worth of news coverage. Each article record includes key fields such as title, description, publication date, category, author, and URL.

Text Preprocessing

This text preprocessing script is designed to clean and standardize the news article descriptions for further analysis. It begins by loading the raw dataset from an Excel file and then applies a series of text cleaning steps focused on the description column. The process includes converting all text to lowercase for uniformity, removing punctuation and numeric characters, eliminating extra whitespaces, and most importantly, filtering out common English stopwords using a custom stopword removal function. These transformations reduce noise in the data and prepare the textual content for downstream tasks like sentiment analysis, topic modeling, or machine learning. The cleaned dataset is then saved as a CSV file for future use. Overall, this pipeline ensures that the text is simplified, standardized, and analytics-ready.

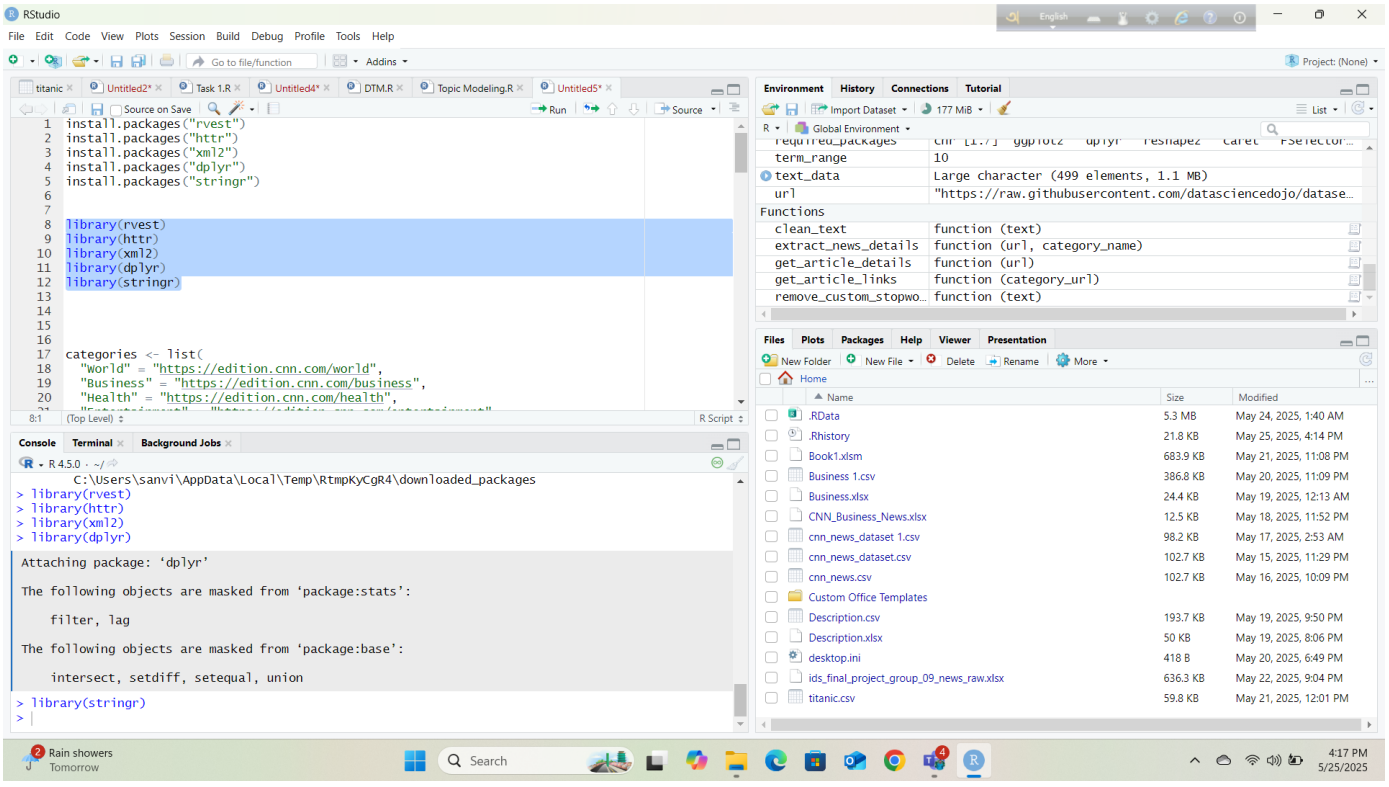
```
install.packages("rvest")
install.packages("httr")
install.packages("xml2")
install.packages("dplyr")
install.packages("stringr")
```



Purpose: These install the necessary R packages for:

1. `rvest` and `xml2`: Web scraping and HTML/XML parsing.
2. `httr`: HTTP requests (though not used directly here, can handle headers/requests).
3. `dplyr`: Data manipulation and wrangling.
4. `stringr`: String operations (like trimming, regex cleaning, etc.).

library(rvest)
library(httr)
library(xml2)
library(dplyr)
library(stringr)

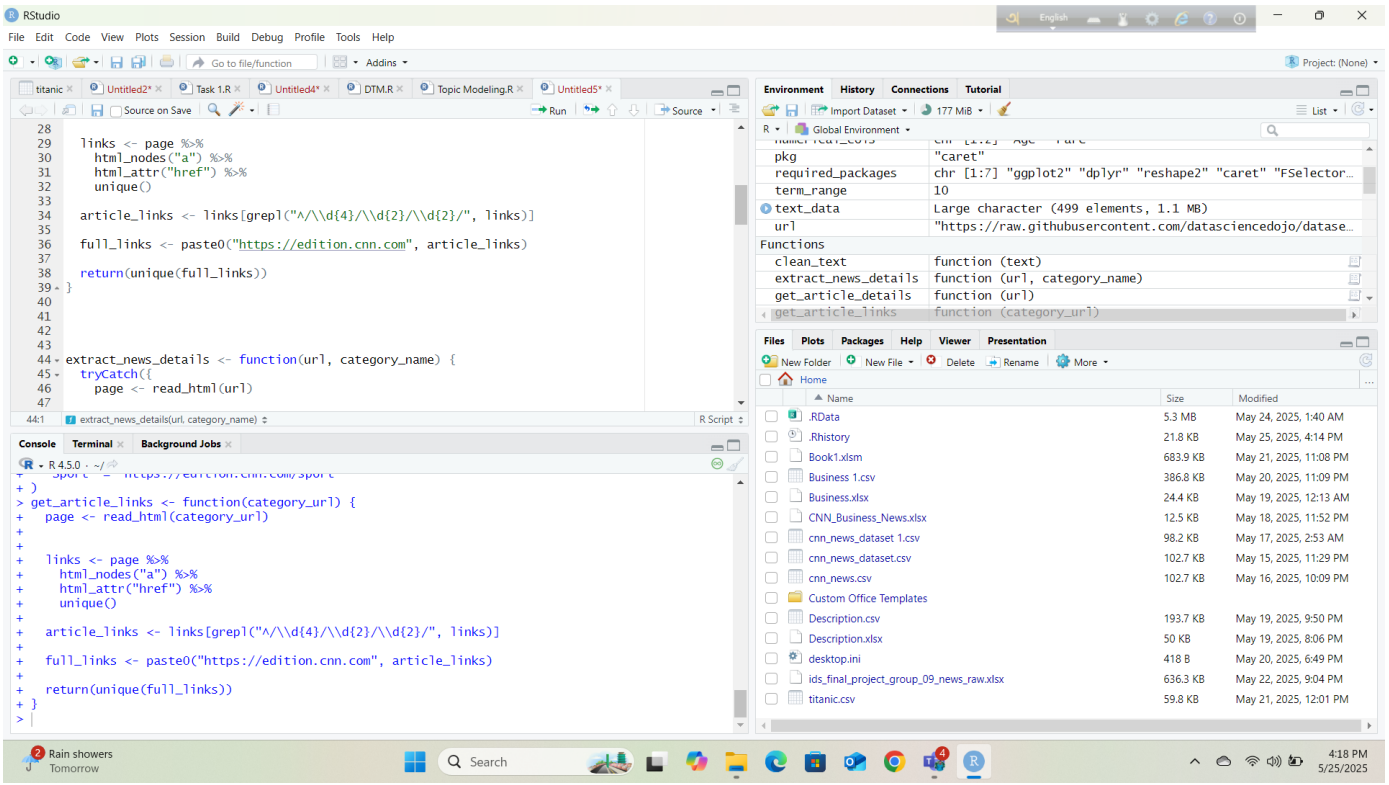


Purpose: This loads all those installed packages into my current R session. Without this, I can't use their functions.

```
categories <- list(
  "World" = "https://edition.cnn.com/world",
  "Business" = "https://edition.cnn.com/business",
  "Health" = "https://edition.cnn.com/health",
  "Entertainment" = "https://edition.cnn.com/entertainment",
  "Sport" = "https://edition.cnn.com/sport"
)
```

Purpose: This list defines the categories of news I want to scrape, each paired with its corresponding CNN URL

```
get_article_links <- function(category_url) {
  page <- read_html(category_url)

  links <- page %>%
    html_nodes("a") %>%
    html_attr("href") %>%
    unique()

  article_links <- links[grepl("^/\\d{4}/\\d{2}/\\d{2}/", links)]

  full_links <- paste0("https://edition.cnn.com", article_links)

  return(unique(full_links))
}
```

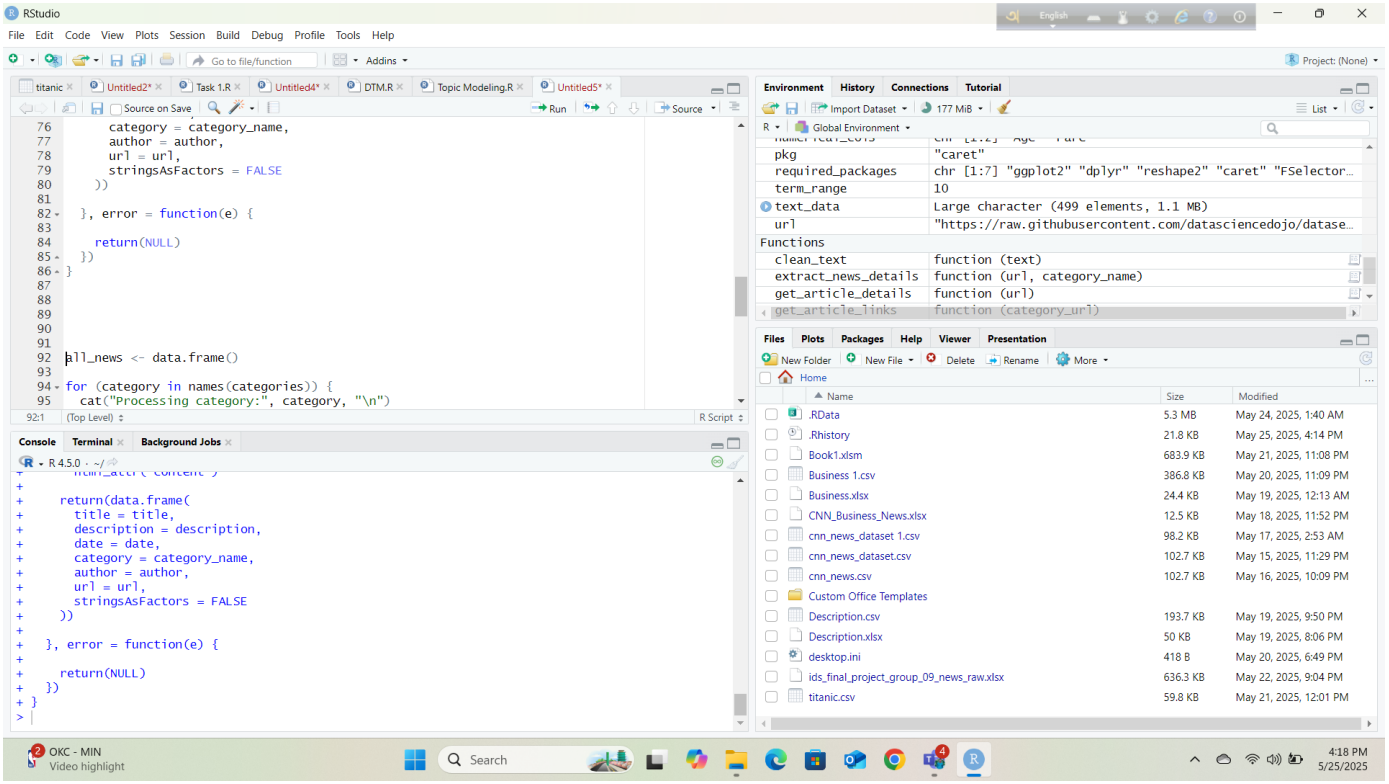


This function:

1. Loads a category page's HTML.
2. Extracts all hyperlink (`<a>`) tags and gets their `href` values.

3. Filters links that look like articles (e.g., `/2024/05/25/world/article-title.html`).
4. Adds the base URL (`https://edition.cnn.com`) to get the full link.

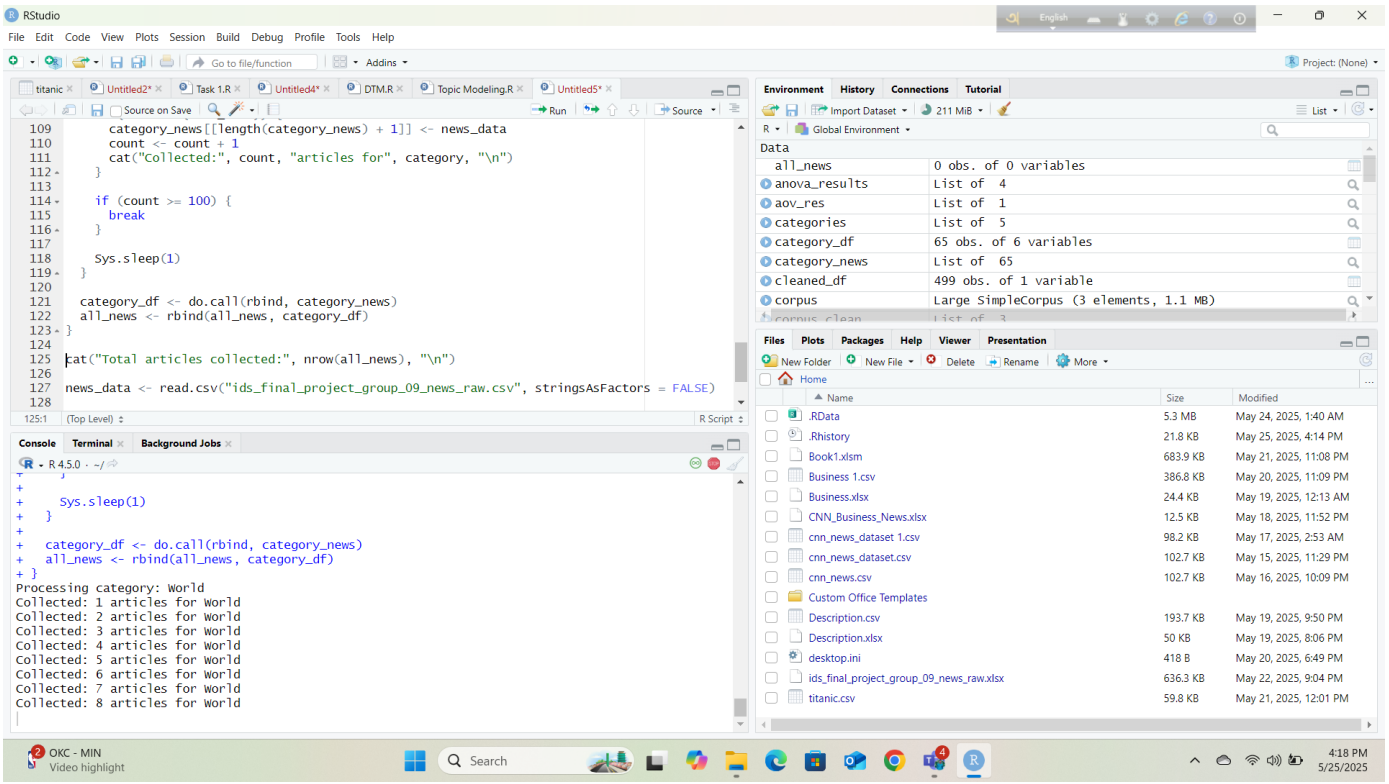
```
extract_news_details <- function(url, category_name) {
  tryCatch({
    page <- read_html(url)

    title <- page %>%
      html_node("h1") %>%
      html_text(trim = TRUE)

    date <- page %>%
      html_node('meta[itemprop="datePublished"]') %>%
      html_attr("content")

    description <- page %>%
      html_node('meta[name="description"]') %>%
      html_attr("content")

    if (is.na(description) || description == "") {
      description <- page %>%
        html_nodes("p") %>%
        html_text() %>%
        paste(collapse = " ") %>%
        str_squish()
    }

    author <- page %>%
      html_node('meta[name="author"]') %>%
      html_attr("content")

    return(data.frame(
      title = title,
      description = description,
      date = date,
      category = category_name,
      author = author,
      url = url,
      stringsAsFactors = FALSE
    ))

  }, error = function(e) {

    return(NULL)
  })
}
```



Purpose: The extract news details function in R is designed to scrape detailed information from a given CNN article URL and organize it into a structured data frame. It takes two inputs: the article URL and the name of the news category (such as "World" or "Health"). Inside the function, it first attempts to read the HTML content of the page using read_html(). It then extracts the article's title using the <h1> tag, the publication date from the meta tag with itemprop="datePublished", and a brief summary from the meta description tag. If the summary is missing or empty, it falls back to extracting all paragraph text (<p> tags), concatenates it, and removes extra whitespace to create a makeshift description. The function also captures the author's name from the meta tag with name="author". All of this information is returned as a single-row data frame. The entire process is wrapped in a tryCatch block to handle errors gracefully — if any part of the extraction fails (due to missing elements, network issues, or malformed HTML), the function simply returns NULL instead of crashing. This allows the scraping process to continue running smoothly across multiple articles.

```
all_news <- data.frame()

for (category in names(categories)) {
  cat("Processing category:", category, "\n")

  category_url <- categories[[category]]
  article_urls <- get_article_links(category_url)

  article_urls <- unique(article_urls)[1:150]

  category_news <- list()
  count <- 0

  for (link in article_urls) {
    news_data <- extract_news_details(link, category)
```

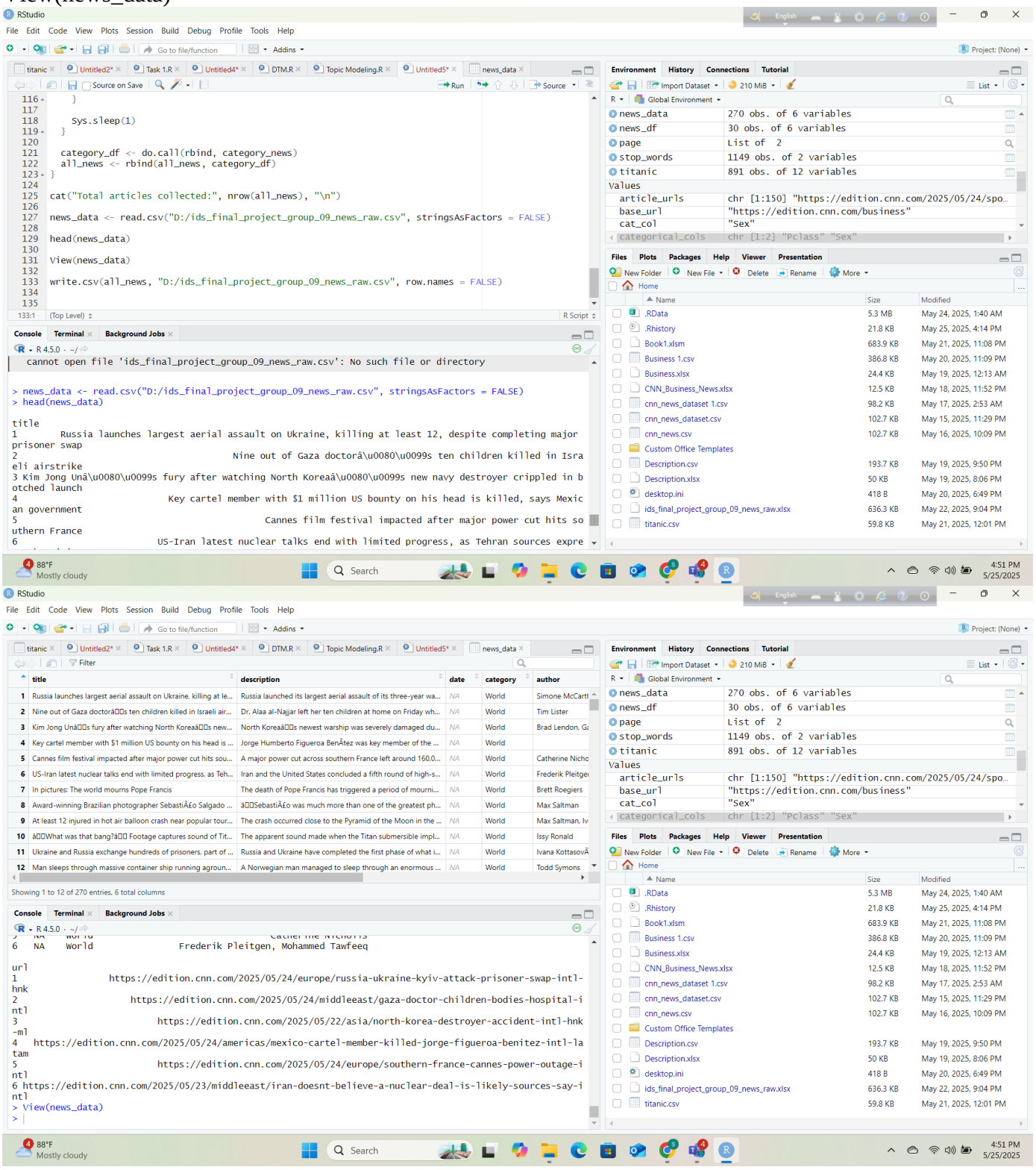
```
  if (!is.null(news_data)) {
    category_news[[length(category_news) + 1]] <- news_data
    count <- count + 1
    cat("Collected:", count, "articles for", category, "\n")
  }

  if (count >= 100) {
    break
  }

  Sys.sleep(1)
 }

 category_df <- do.call(rbind, category_news)
 all_news <- rbind(all_news, category_df)
}
```



Purpose: This block of R code is responsible for scraping news articles from multiple categories on CNN and compiling them into a single dataset. It begins by initializing an empty data frame called all_news to store the final results. Then, for each news category defined in the categories list (e.g., "World", "Business", etc.), the code prints a message indicating the current category being processed. It retrieves up to 150 unique article URLs from the category page using the get_article_links function. For each of these URLs, it calls the extract_news_details function to scrape the article's content. Valid articles (i.e., those not returning NULL) are stored in a list called category_news, and a counter tracks how many articles have been successfully collected. Once 100 valid articles are gathered (or the URL list is exhausted), the loop stops for that category. A one-second delay (Sys.sleep(1)) between each request is used to avoid overloading the server or getting rate-limited. After processing a category, the list of article data frames is combined into a single data frame using do.call(rbind, ...), and appended to all_news. This process repeats for every category, ultimately producing a comprehensive data frame containing news articles from all specified CNN categories.

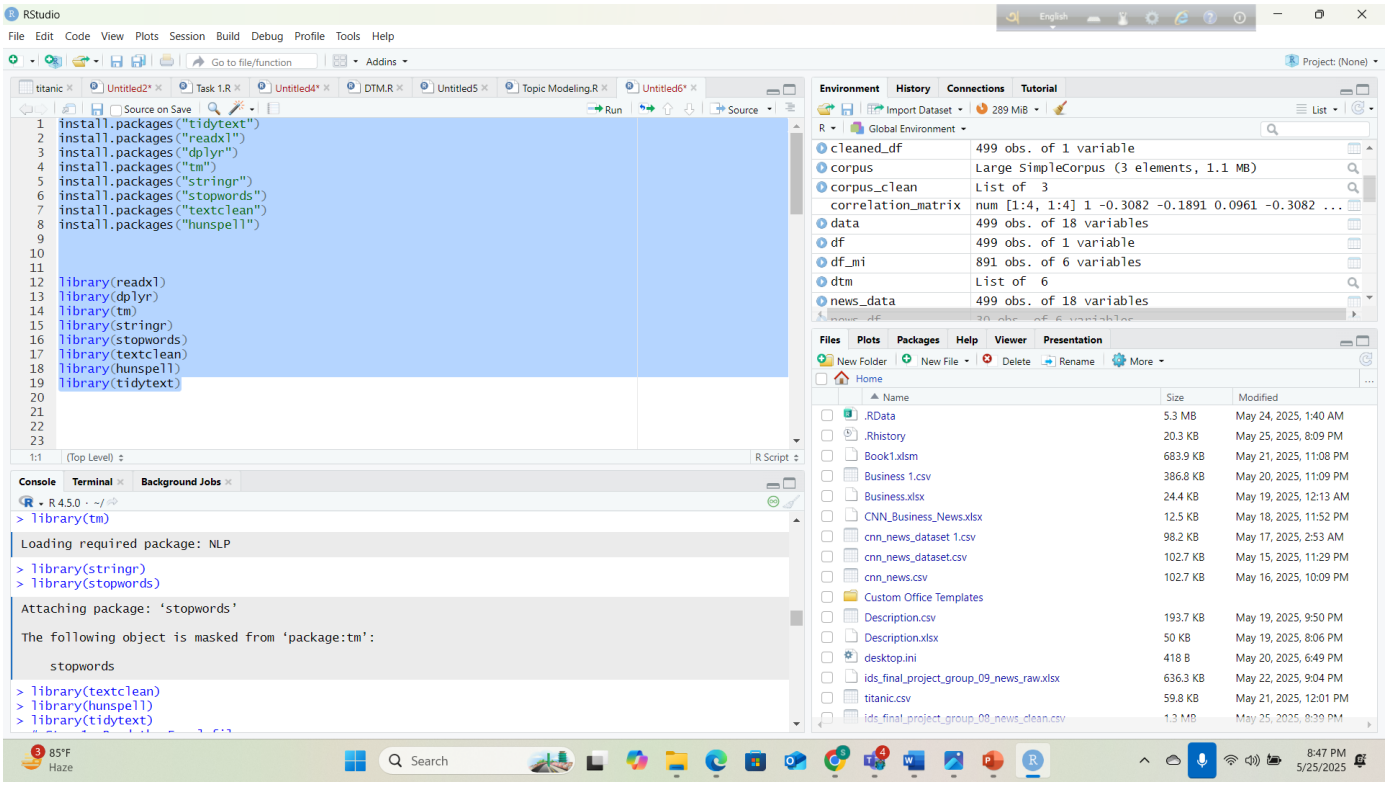
```
write.csv(all_news, "D:/ids_final_project_group_09_news_raw.csv", row.names = FALSE)
news_data <- read.csv("D:/ids_final_project_group_09_news_raw.csv", stringsAsFactors = FALSE)
head(news_data)
```

View(news_data)



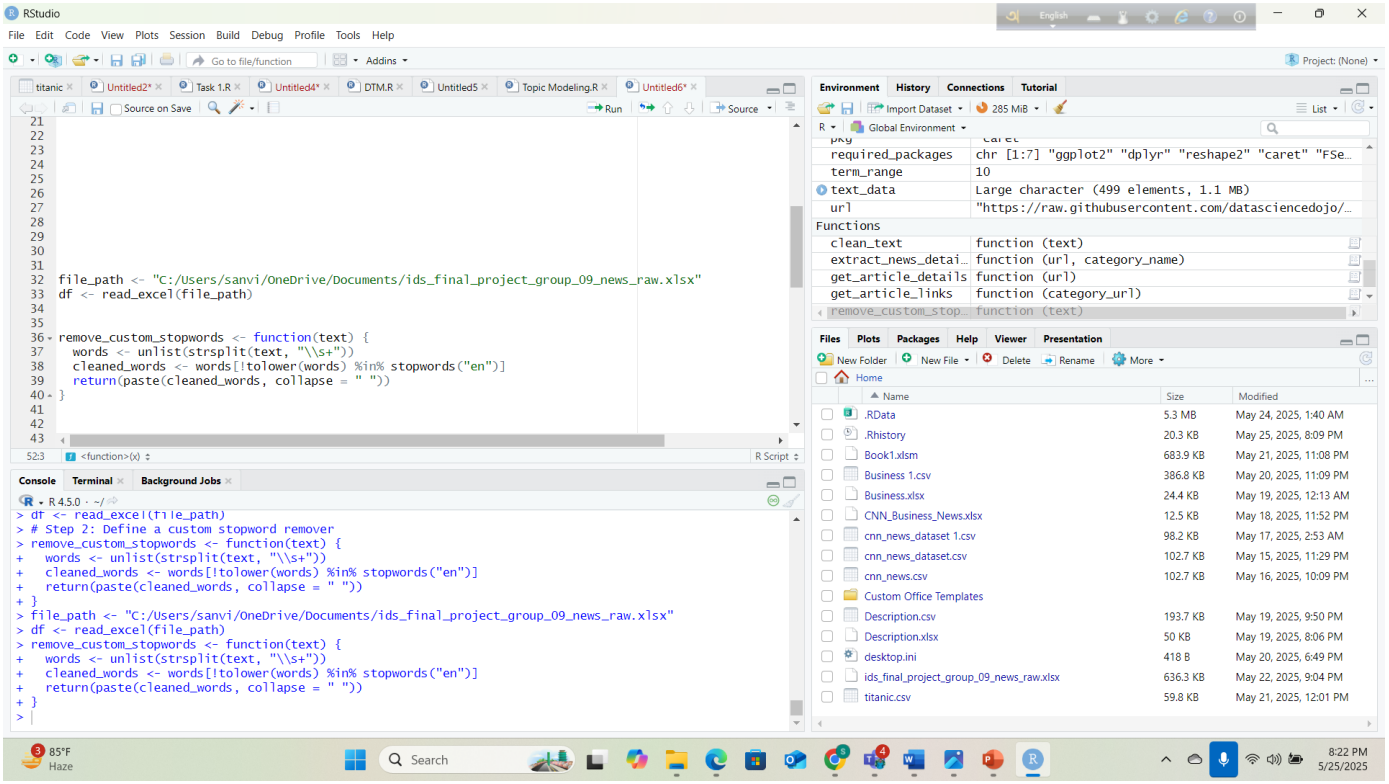
Purpose: Dumps all our clean, structured, juicy CNN news data into a CSV file. That's our raw dataset, ready for analysis, modeling, or doomscrolling.

```
install.packages("tidytext")
install.packages("readxl")
install.packages("dplyr")
install.packages("tm")
install.packages("stringr")
install.packages("stopwords")
install.packages("textclean")
install.packages("hunspell")

library(readxl)
library(dplyr)
library(tm)
library(stringr)
library(stopwords)
library(textclean)
library(hunspell)
library(tidytext)
```



1. **Installing and Loading Required Packages**
   Purpose: This section ensures that all necessary libraries are installed and available in the environment. It includes tools for data manipulation (dplyr), reading Excel files (readxl), text mining and string operations (tm, stringr), working with stopwords (stopwords), emoji and text cleaning (textclean), spell-checking (hunspell), and text analysis (tidytext). These packages provide the foundational functions needed for reading, processing, and transforming raw textual news data.

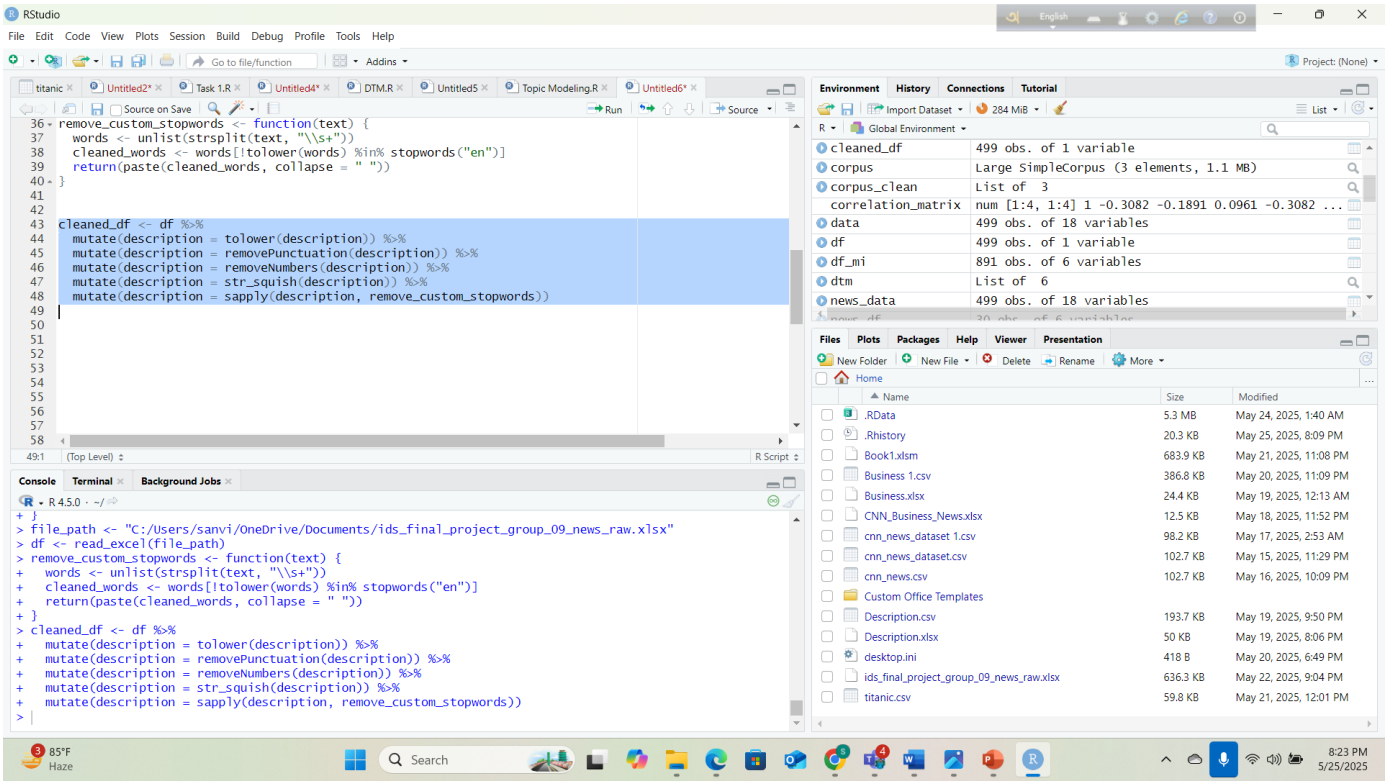
```
file_path <- "D:/ids_final_project_group_09_news_raw.xlsx"
df <- read_excel(file_path)
remove_custom_stopwords <- function(text) {
  words <- unlist(strsplit(text, "\\s+"))
  cleaned_words <- words[!tolower(words) %in% stopwords("en")]
  return(paste(cleaned_words, collapse = " "))
}
```



2. Load the Raw Dataset And Define a Custom Stopword Remover

Purpose: Reads the raw Excel file containing CNN news data into a dataframe (`df`) from the specified file path.Defines a function that removes common English stopwords (like "and", "the", "is") from a given string. It splits the text into individual words, filters out the stopwords, and then rejoins the cleaned words.

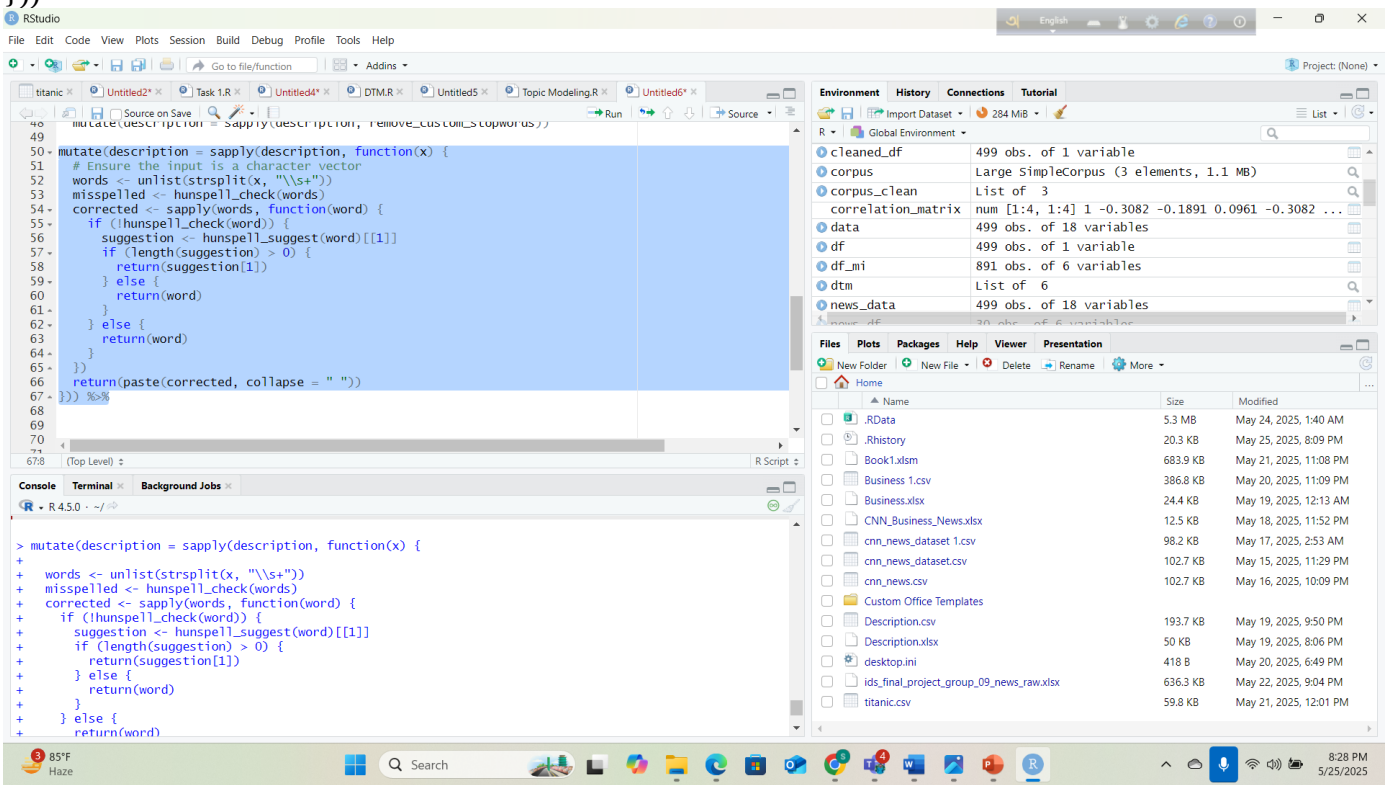
```
cleaned_df <- df %>%
  mutate(description = tolower(description)) %>%
  mutate(description = removePunctuation(description)) %>%
  mutate(description = removeNumbers(description)) %>%
  mutate(description = str_squish(description)) %>%
  mutate(description = sapply(description, remove_custom_stopwords))
```



Text Cleaning Pipeline

Purpose: This block of R code performs a **multi-step text cleaning pipeline** on the description column of the dataset df, storing the output in a new dataframe called cleaned_df. First, it converts all text in the description field to lowercase using tolower(), ensuring uniformity and preventing case-sensitive mismatches. Then, it removes all punctuation marks with removePunctuation() and strips out any numeric digits using removeNumbers(), helping to eliminate non-informative characters. After that, str_squish() is applied to trim and condense any extra whitespace between words. Finally, it applies a custom-defined function remove_custom_stopwords via sapply(), which filters out common English stopwords (like "the", "is", "and") that usually carry little semantic weight in text analysis. Together, these steps result in a much cleaner, standardized, and analysis-ready textual dataset.

```
mutate(description = sapply(description, function(x) {
  words <- unlist(strsplit(x, "\\s+"))
  corrected <- sapply(words, function(word) {
    if (!hunspell_check(word)) {
      suggestion <- hunspell_suggest(word)[[1]]
      if (length(suggestion) > 0) {
        return(suggestion[1])
      } else {
        return(word)
      }
    } else {
      return(word)
    }
  })
  return(paste(corrected, collapse = " "))
}))
```

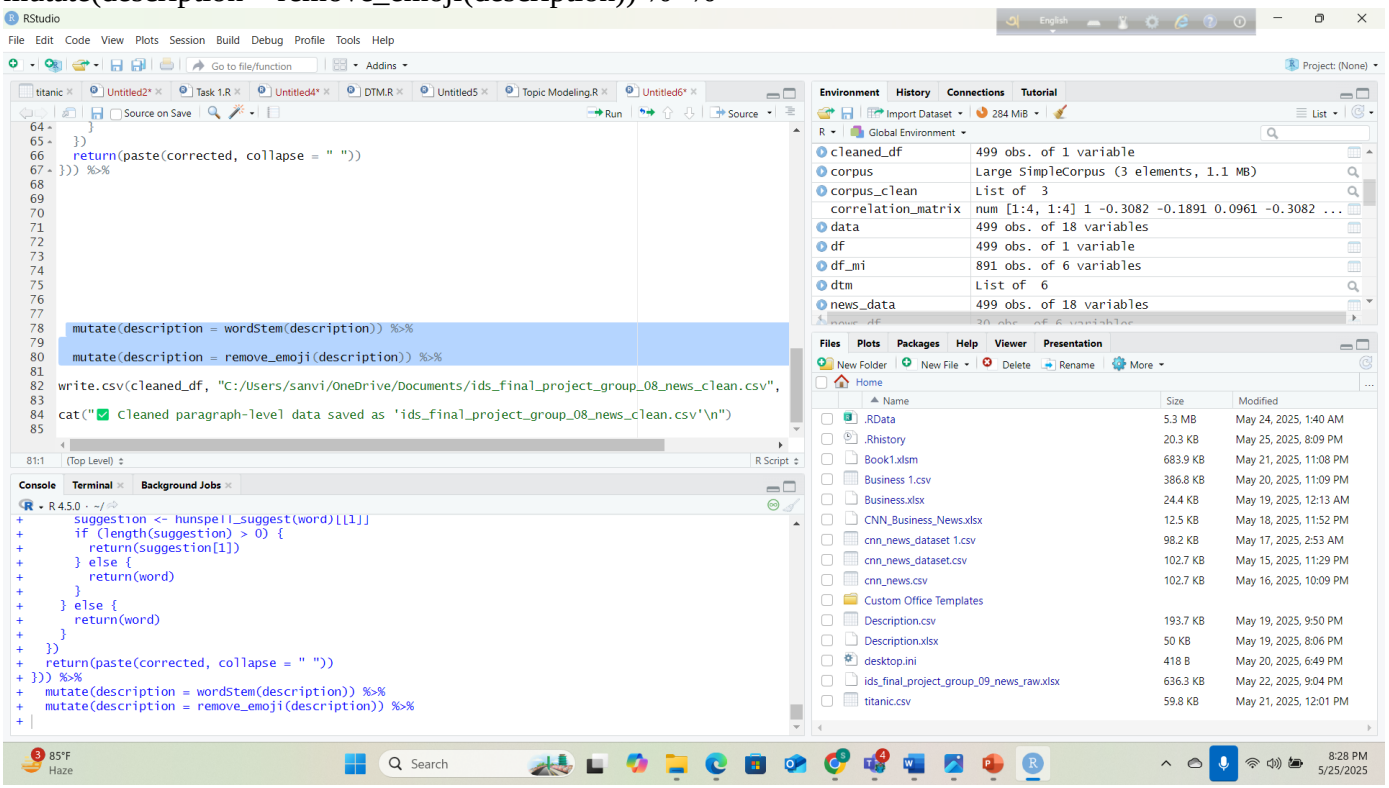


Spell Correction

Purpose: This block of code applies **spell correction** to each entry in the description column using the hunspell package. It works by splitting each description into individual words using strsplit(), then checking each word for spelling errors with hunspell_check(). If a word is found to be misspelled, hunspell_suggest() generates a list of possible corrections, and the first suggestion is used as a replacement. If no suggestions are available, or if the word is already correct, the original word is kept. After processing

all the words, they are recombined into a corrected sentence using paste(). This automated correction process helps improve the accuracy and quality of the text data, making it more reliable for downstream tasks like text mining, classification, or sentiment analysis.

mutate(description = wordStem(description)) %>%
mutate(description = remove_emoji(description)) %>%



Stemming and Emoji Removal

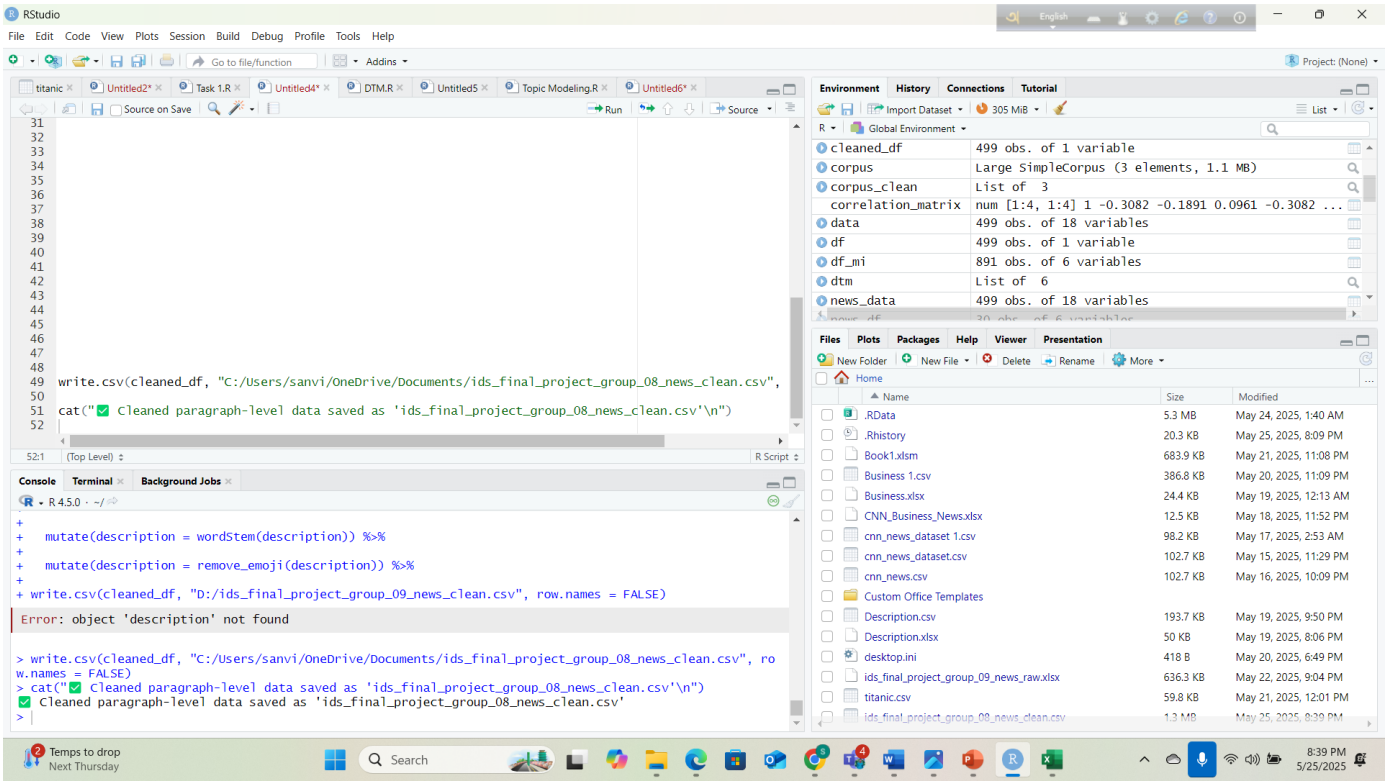
Purpose: wordStem(description) reduces words to their root form to generalize word forms.

remove_emoji(description) strips out any emojis or symbols that might've survived earlier cleaning steps.

write.csv(cleaned_df, "D:/ids_final_project_group_09_news_clean.csv", row.names = FALSE)

cat("☑ Cleaned

paragraph-level data saved as 'ids_final_project_group_08news_clean.csv'\n")



Purpose: Saves the fully cleaned dataset to a new CSV file in the specified directory and prints a confirmation message to the console.

Install Required Packages-

```
install.packages("readr")
install.packages("tm")
install.packages("SnowballC")
install.packages("topicmodels")
install.packages("tidytext")
install.packages("dplyr")
install.packages("ggplot2")
install.packages("reshape2")
install.packages("tidyr")
```

Purpose: We use these packages for run the code perfectly

Load Libraries-

```
library(readr)
library(tm)
library(SnowballC)
library(topicmodels)
library(tidytext)
library(dplyr)
library(ggplot2)
library(reshape2)
library(tidyr)
```

Purpose: After installing the packages we must have load those packages

Load CSV File-

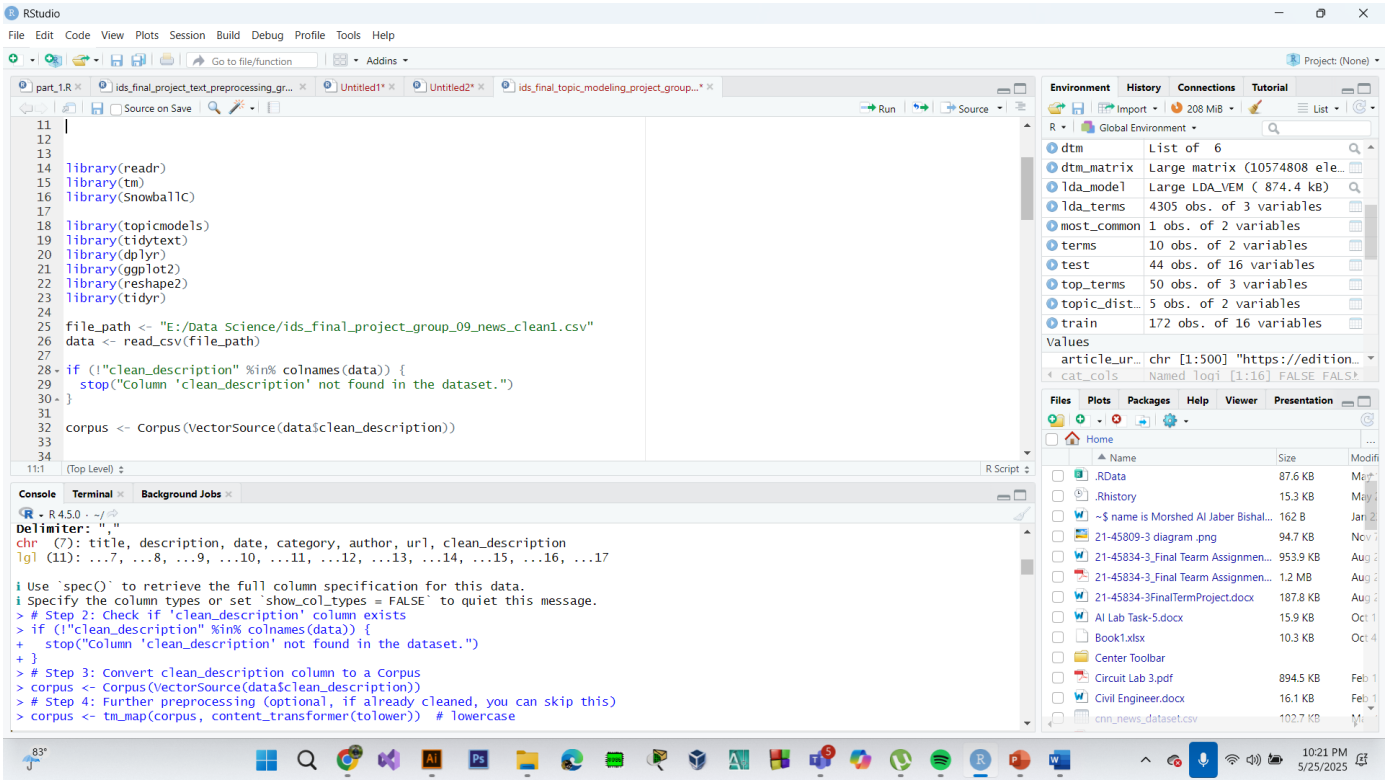
file_path <- "E:/Data Science/ids_final_project_group_09_news_clean1.csv"
data <- read_csv(file_path)

Purpose: I have to notice the path carefully when load the file

Create Text Corpus-

corpus <- Corpus(VectorSource(data$clean_description))

Purpose: Converts your clean_description text into a format (Corpus) that R can work with for text mining.

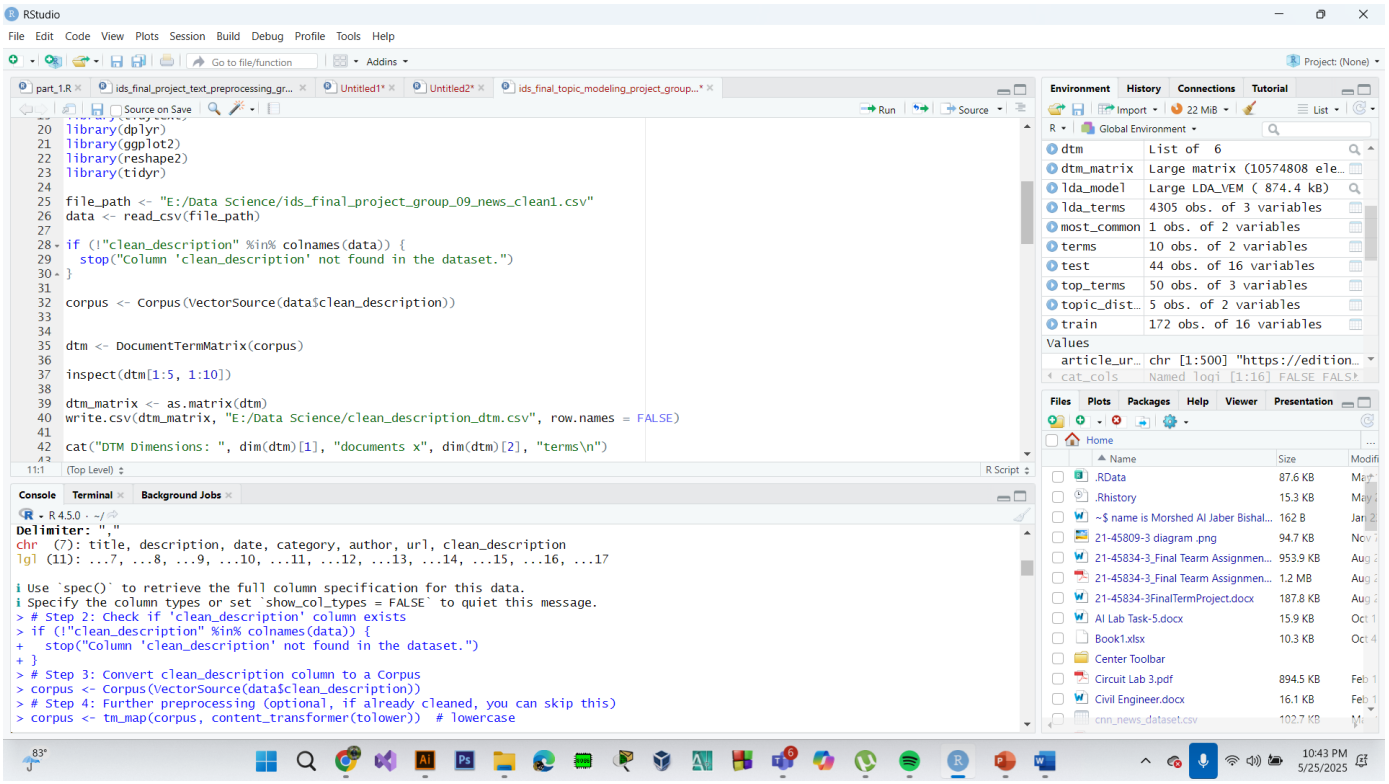


Purpose: Converts your clean_description text into a format (Corpus) that R can work with for text mining.

Creating a Corpus and Document-Term Matrix (DTM)

corpus <- Corpus(VectorSource(data$clean_description))
dtm <- DocumentTermMatrix(corpus)

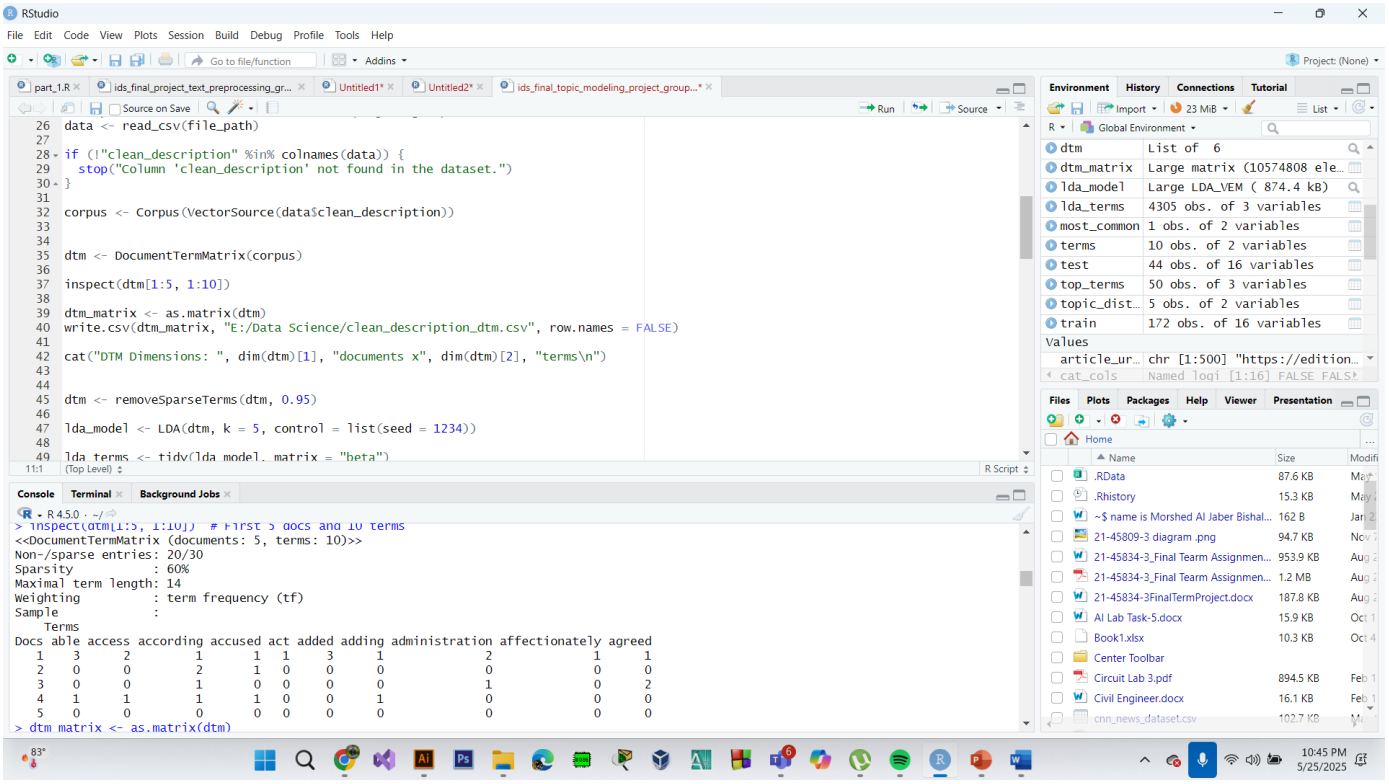
inspect(dtm[1:5, 1:10])
dtm_matrix <- as.matrix(dtm)
write.csv(dtm_matrix, "E:/Data Science/clean_description_dtm.csv", row.names = FALSE)
cat("DTM Dimensions: ", dim(dtm)[1], "documents x", dim(dtm)[2], "terms\n")



Purpose: This block transforms the clean_description text column into a text corpus, then builds a Document-Term Matrix (DTM). Each row in the DTM represents a document, and each column represents a term. The script previews the first 5 documents and 10 terms, saves the DTM as a CSV, and prints its dimensions.

Extracting and Displaying Top Terms for Each Topic

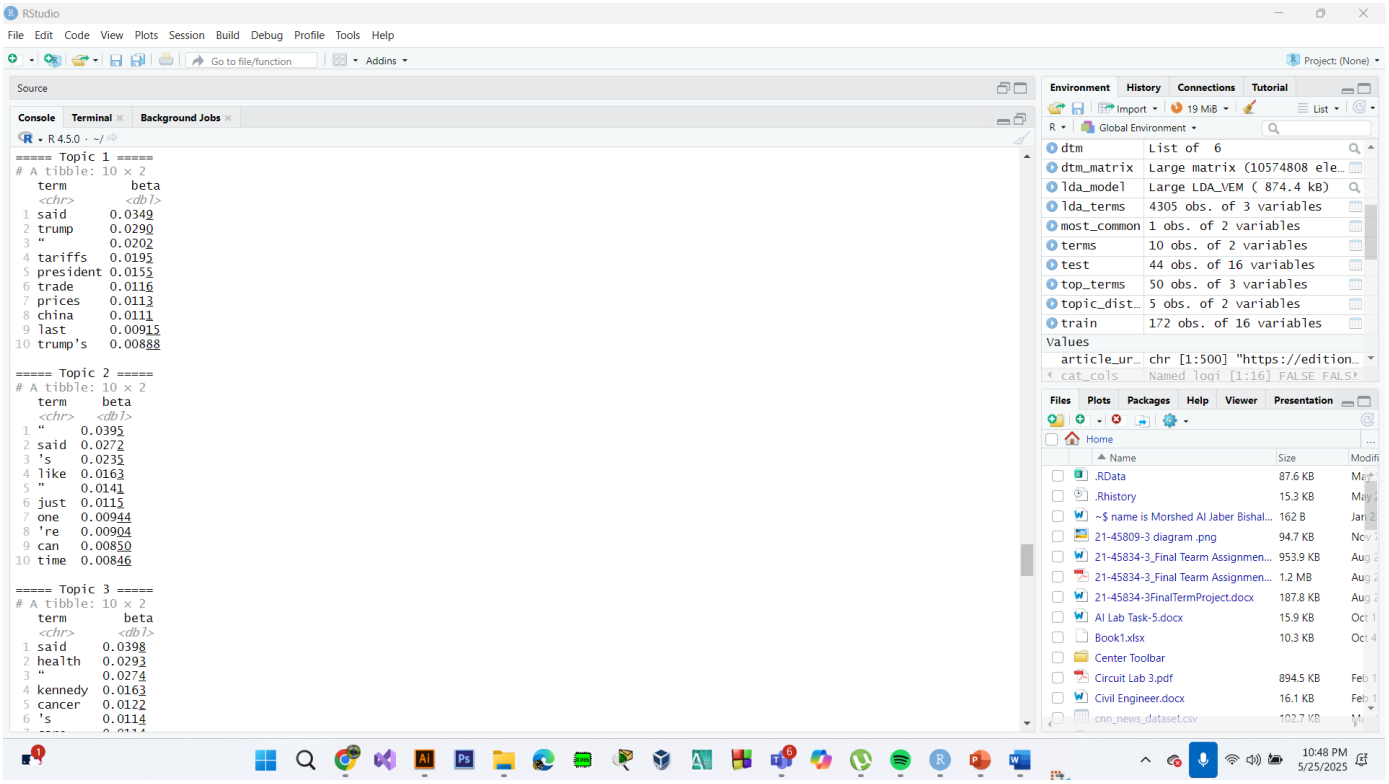
```
lda_terms <- tidy(lda_model, matrix = "beta")

top_terms <- lda_terms %>%
  filter(topic %in% 1:5) %>%
  group_by(topic) %>%
  top_n(10, beta) %>%
  arrange(topic, -beta) %>%
  ungroup()

for (i in 1:5) {
  cat(paste0("\n===== Topic ", i, " =====\n"))
  terms <- top_terms %>%
    filter(topic == i) %>%
    select(term, beta)
  print(terms)
}
```

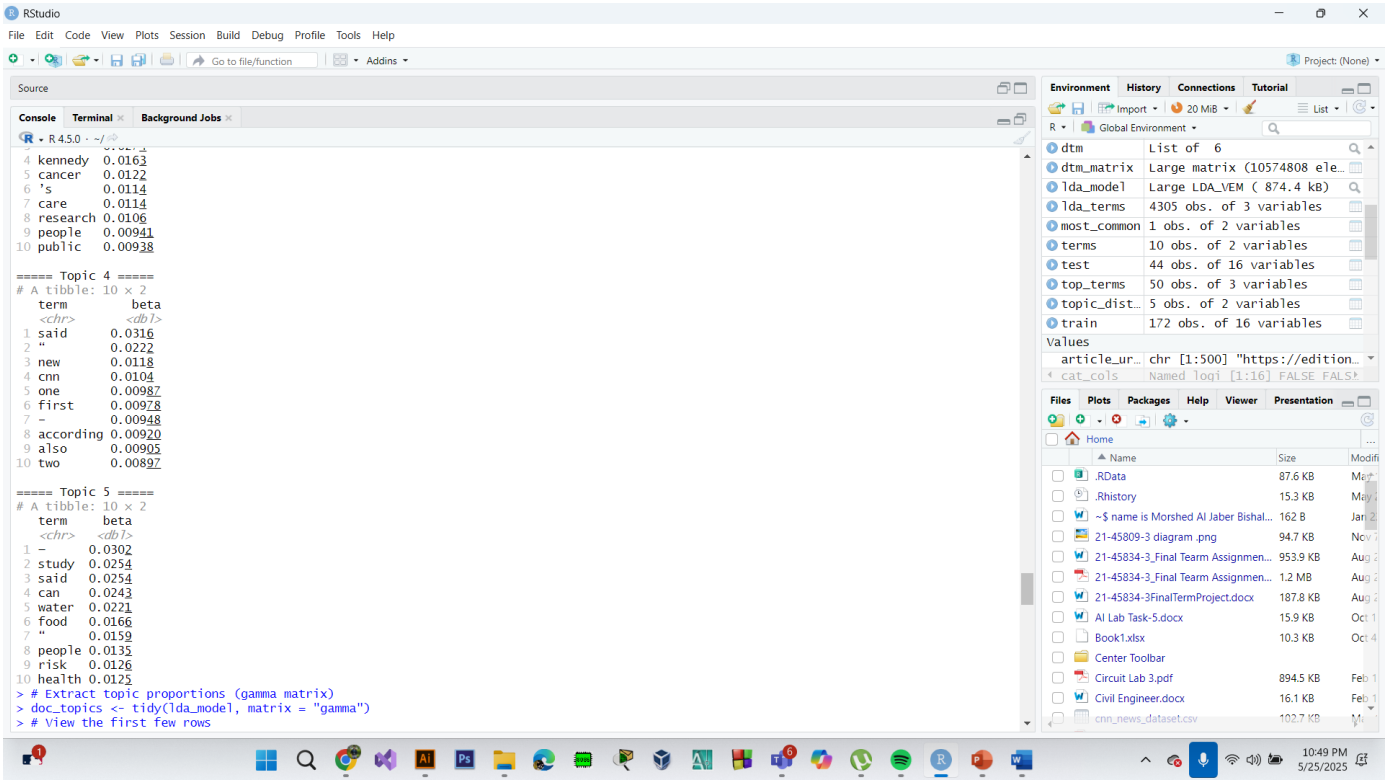


Purpose: This block extracts the `beta` matrix, which shows the probability of each word belonging to each topic. It then filters the top 10 words per topic and prints them topic by topic. These words help us interpret what each topic is about.

Viewing Topic Proportions for the First 5 Documents-

```
doc_topics %>%
  filter(document %in% as.character(1:5)) %>%
  pivot_wider(
    names_from = topic,
    values_from = gamma,
    names_prefix = "Topic_"
  ) %>%
  arrange(as.numeric(document)) -> doc_topic_wide

print(doc_topic_wide)
```

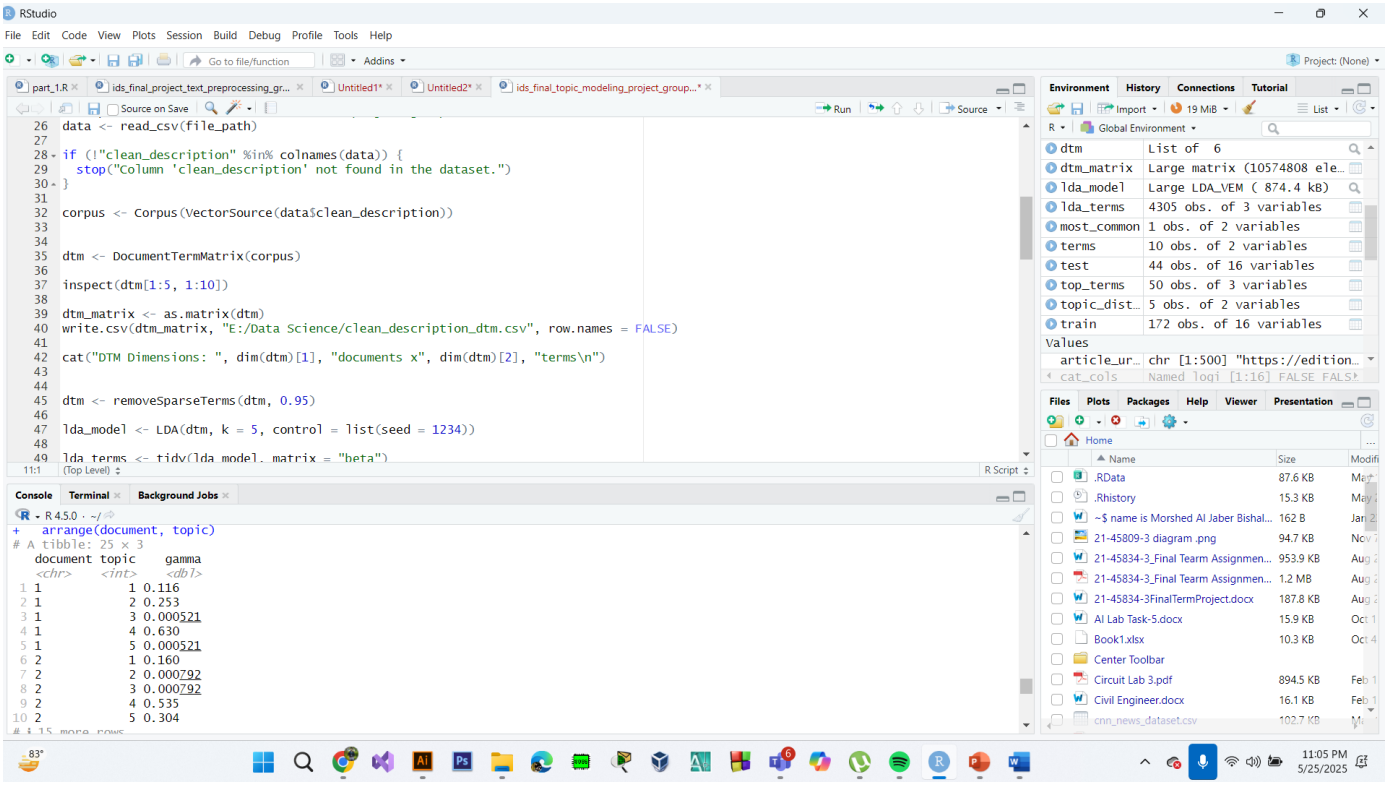




Purpose: This part (though placed slightly before the actual creation of doc_topics, which should be corrected) displays the topic proportions (gamma values) for the first 5 documents. It reshapes the data so that each document is a row and each topic is a column with its respective proportion.

Finding and Counting Dominant Topics-

```
dominant_topic <- doc_topics %>%
  group_by(document) %>%
  slice_max(gamma) %>%
  ungroup()

topic_distribution <- dominant_topic %>%
  count(topic, sort = TRUE)

cat("\n\n========== Topic Distribution Across Documents ==========\n")
print(topic_distribution)
```

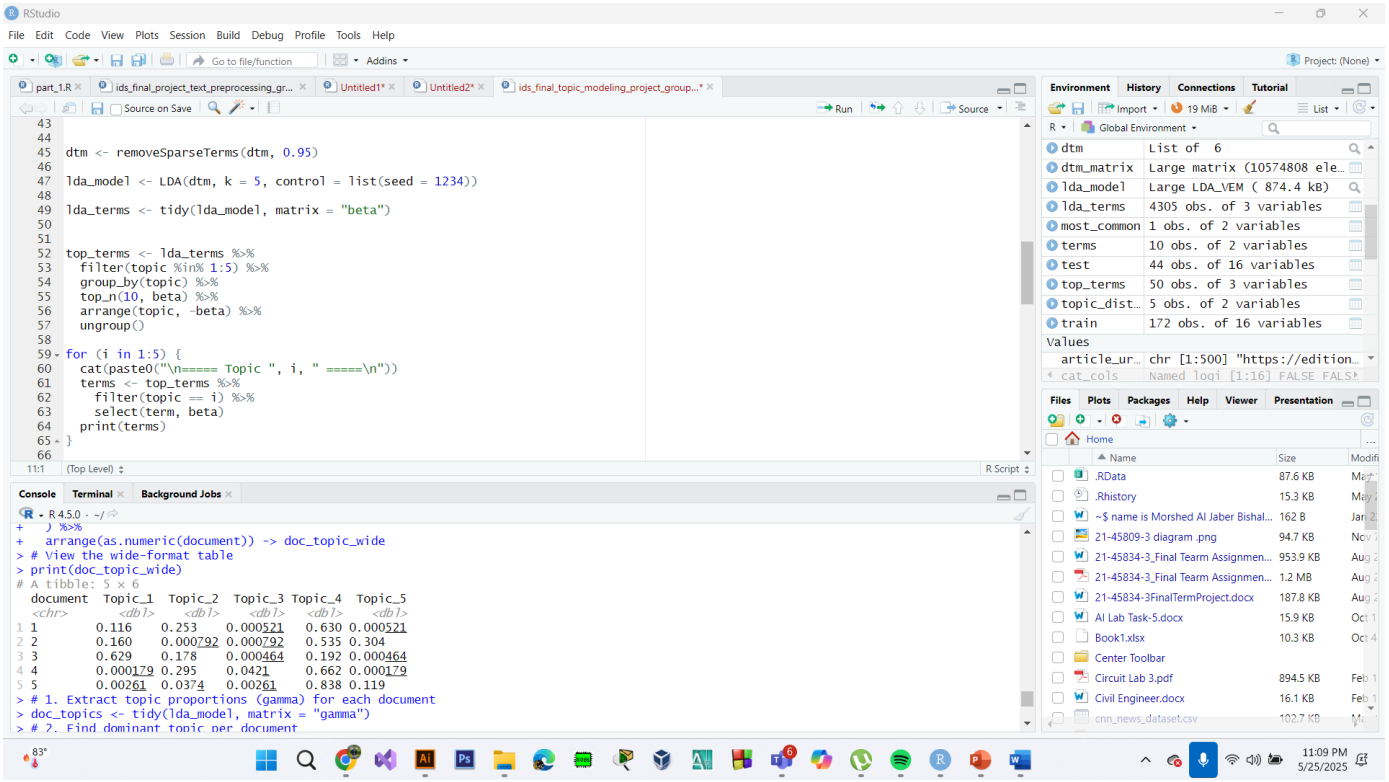


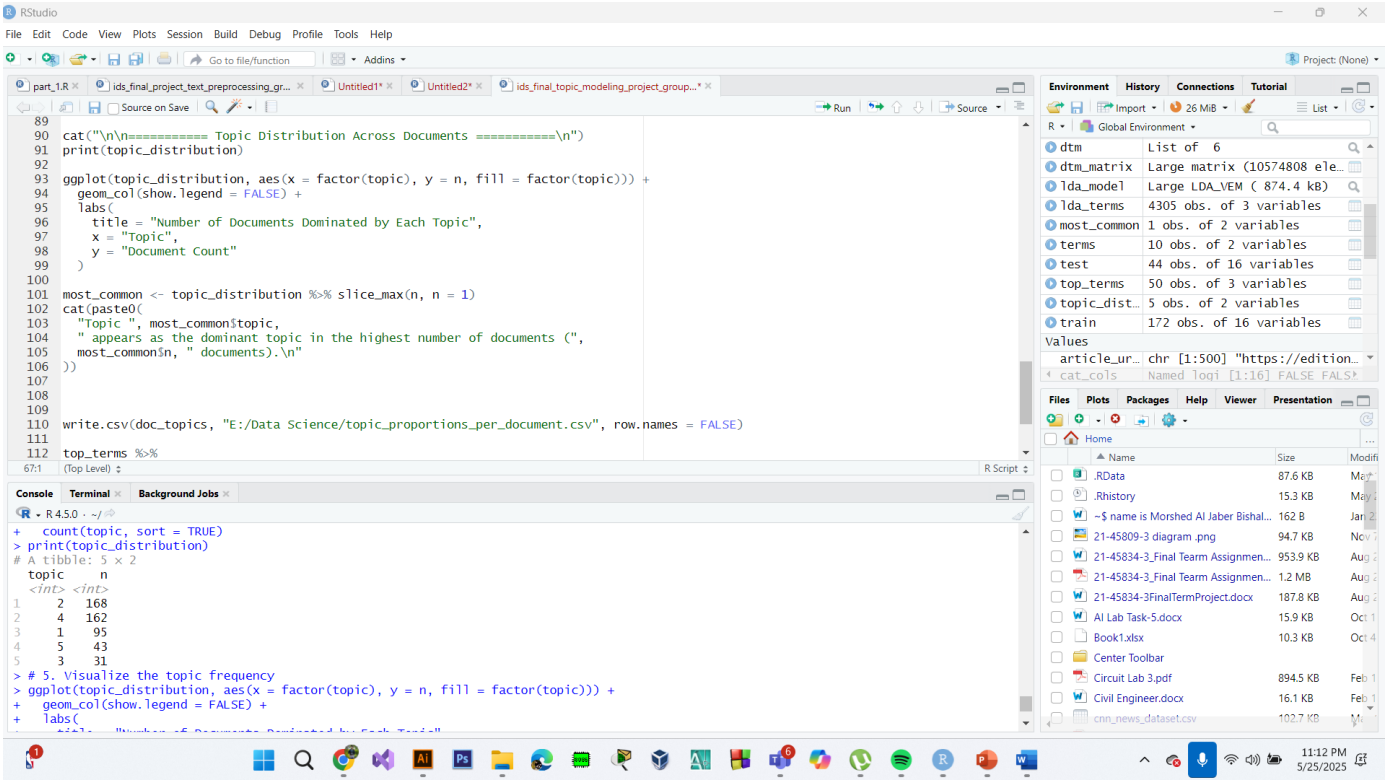
Purpose: This block identifies the most dominant topic for each document by selecting the highest gamma value. Then it counts how many documents each topic dominates and prints the result — showing which topic appears most frequently.

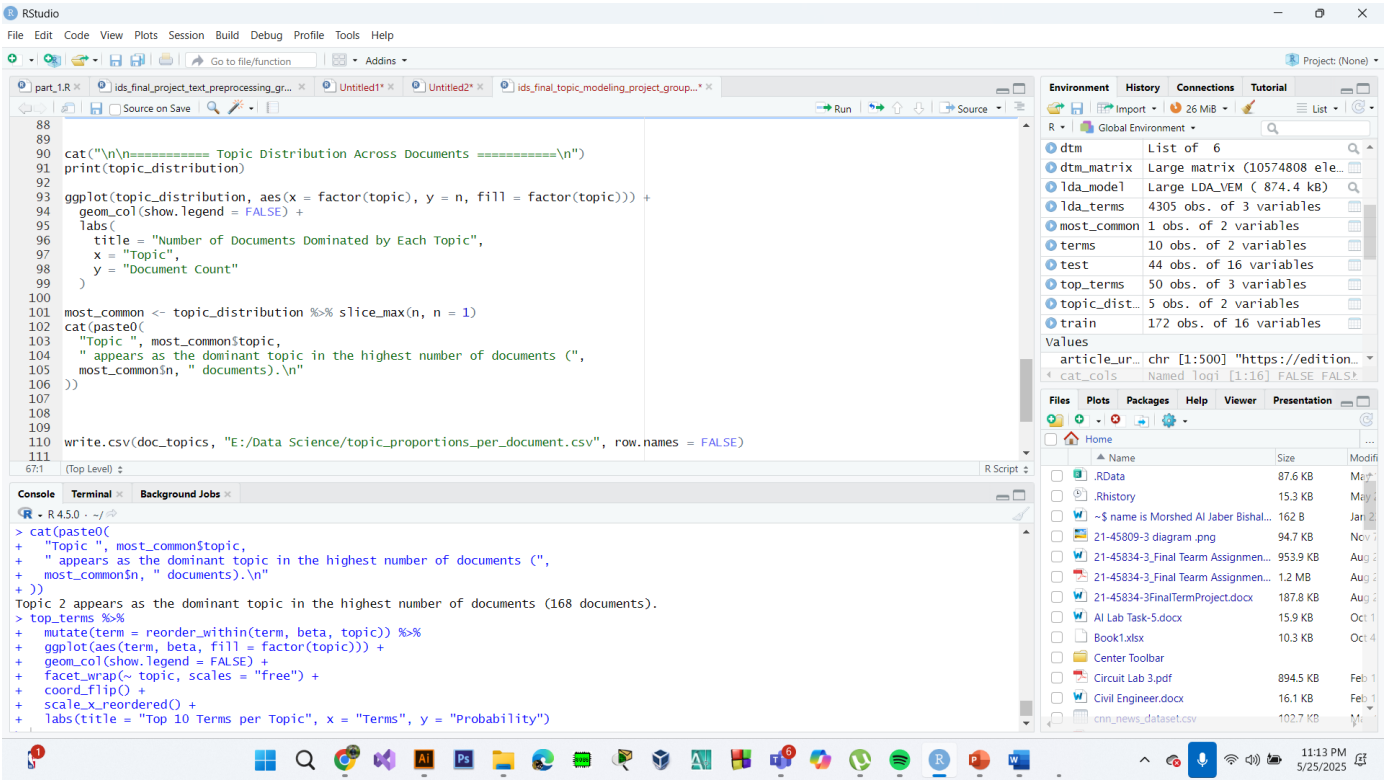
visualizing the Topic Distribution

```
doc_topics %>%
  filter(document %in% as.character(1:5)) %>%  # documents 1 to 5
  pivot_wider(
    names_from = topic,
    values_from = gamma,
    names_prefix = "Topic_"
  ) %>%
  arrange(as.numeric(document)) -> doc_topic_wide

print(doc_topic_wide)
```

```r
doc_topics <- tidy(lda_model, matrix = "gamma")

dominant_topic <- doc_topics %>%
  group_by(document) %>%
  slice_max(gamma) %>%
  ungroup()

topic_distribution <- dominant_topic %>%
  count(topic, sort = TRUE)
```







Discussion:

In topic 1-there have terms said, tariffs, president, trade, prices, china, fact, trumps.
This topic seems centered on economic trade policy, political statements, and international relations, especially involving U.S. and China. Terms like tariffs, president, trumps, and china point toward global trade discussions, possibly in a geopolitical context. In topic 2 there have terms said, like, just, people, know, time, think, really, going, want
This topic reflects general public discourse or opinion, maybe social commentary or interviews. Words like think, know, really, people, and said indicate reported speech or dialogue-heavy articles. In topic 3 there have terms said, health, cancer, kennedy, research, people, public.
This topic strongly points to health-related reporting, especially concerning cancer research or public health policy. The term kennedy might relate to a specific political figure or public initiative. In topic 4 there have terms said, men, even, cnn, accused, according, also, two.
This looks like a topic about criminal justice, reporting on legal proceedings, accusations, or news reports. The combination of accused, cnn, and according suggests legal or investigative reporting. In topic 5 there have terms study, can, said, school, people, root, public, health

This topic overlaps health and education, possibly discussing public health studies, school initiatives, or educational awareness programs. The word study being top-ranked confirms a research/academic tone. Topic 2 was found in 167 documents, which indicated that it was the most frequently occurred topic in the documents and was thus the highest to be ranked a specific topic. But the leading words in this topic — “said,” “like,” “just,” common punctuation like “’s” and “” and others — suggest it largely captured general speech, quotes and other conversational language and not a specific topic, say health or politics. So, though it appears most often, Topic 2 is probably