

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 Arifat	21-45692-3

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 Summary

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")
```

The screenshot shows the RStudio interface. In the top-left pane, there is a code editor with the following R script:

```

1 install.packages("rvest")
2 install.packages("httr")
3 install.packages("xml2")
4 install.packages("dplyr")
5 install.packages("stringr")
6
7
8 library(rvest)
9 library(httr)
10 library(xml2)
11 library(dplyr)
12 library(stringr)
13
14
15
16 categories <- list(
17   "World" = "https://edition.cnn.com/world",
18   "Business" = "https://edition.cnn.com/business",
19   "Health" = "https://edition.cnn.com/health",
20   "Technology" = "https://edition.cnn.com/technology"
)

```

In the bottom-left pane, the Console tab shows the output of running the script. It includes a warning about Rtools, the download of the stringr package, and the unpacking of the downloaded binary packages.

The right side of the interface features the Global Environment pane, which lists objects like `text_data`, `url`, and various functions. Below it is the Files pane, which displays a file tree with files like `titanic.csv`, `Business.xlsx`, and `CNN_Business_News.xlsx`.

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)

```

The screenshot shows the RStudio interface. In the top-left pane, there is a code editor with the same R script as the previous screenshot.

In the bottom-left pane, the Console tab shows the output of running the script. It includes the installation of the `rvest`, `httr`, `xml2`, `dplyr`, and `stringr` packages, the attachment of the `dplyr` package, and the masking of objects from other packages.

The right side of the interface features the Global Environment pane and the Files pane, which displays a file tree with files like `titanic.csv`, `Business.xlsx`, and `CNN_Business_News.xlsx`.

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[grep("^\d{4}/\d{2}/\d{2}/", links)]

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

  return(unique(full_links))
}
```

The screenshot shows the RStudio interface. The code editor on the left contains the R script for the `get_article_links` function. The environment browser in the top right shows various global variables and their types and values. The file browser at the bottom right lists files in the current directory, including CSV and XLSX files related to the project.

```
28
29   Links <- page %>%
30     html_nodes("a") %>%
31     html_attr("href") %>%
32     unique()
33
34   article_links <- links[grep("^\d{4}/\d{2}/\d{2}/", links)]
35
36   full_links <- paste0("https://edition.cnn.com", article_links)
37
38   return(unique(full_links))
39 }

40
41
42
43
44+ extract_news_details <- function(url, category_name) {
45+   tryCatch({
46+     page <- read_html(url)
47+
48+
49+     links <- page %>%
50+       html_nodes("a") %>%
51+       html_attr("href") %>%
52+       unique()
53+
54+     article_links <- links[grep("^\d{4}/\d{2}/\d{2}/", links)]
55+
56+     full_links <- paste0("https://edition.cnn.com", article_links)
57+
58+     return(unique(full_links))
59+
60+   })
61 }
```

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")

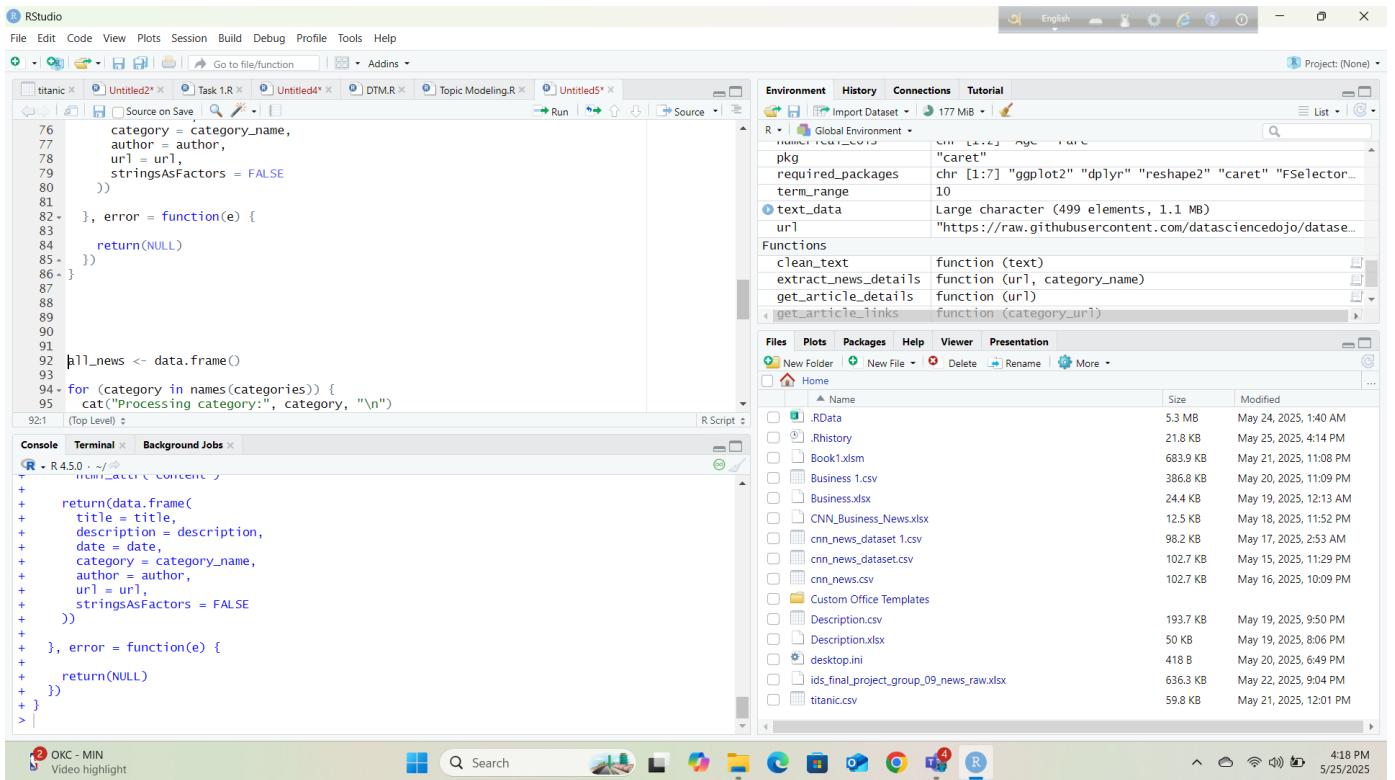
    description <- page %>%
      html_node('meta[name="description"]') %>%
      html_attr("content")

    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)
  }
}

```

```

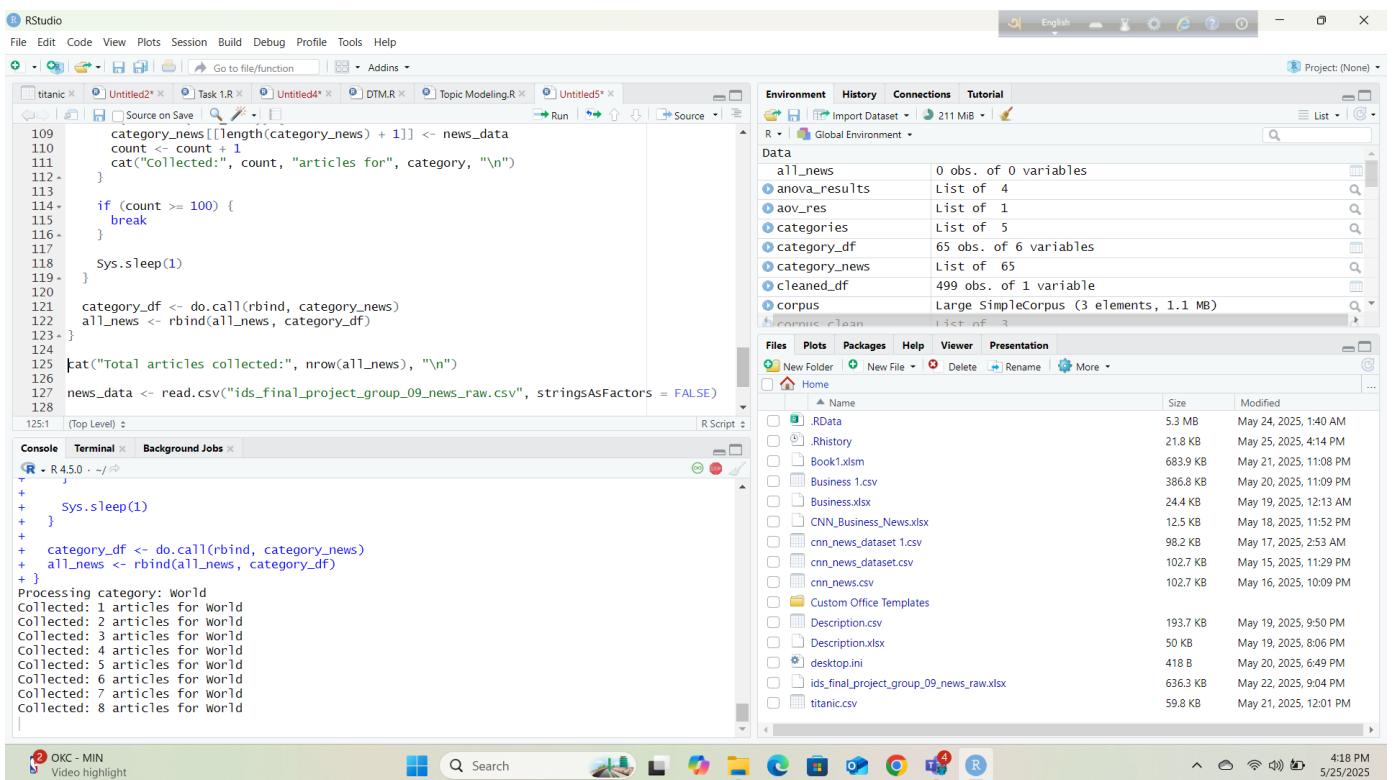
if (!is.null(news_data)) {
  category_news[[length(category_news) + 1]] <- news_data
  count <- count + 1
  cat("Collected:", count, "articles for", category, "\n")
}

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)

The screenshot displays two RStudio sessions side-by-side, both titled "titanic".

Session 1 (Left):

- Code:**

```
116 +
117
118 Sys.sleep(1)
119 +
120
121 category_df <- do.call(rbind, category_news)
122 all_news <- rbind(all_news, category_df)
123 +
124
125 cat("total articles collected:", nrow(all_news), "\n")
126
127 news_data <- read.csv("D:/ids_final_project_group_09_news_raw.csv", stringsAsFactors = FALSE)
128
129 head(news_data)
130
131 View(news_data)
132
133 write.csv(all_news, "D:/ids_final_project_group_09_news_raw.csv", row.names = FALSE)
134
135
```
- Console:**

```
R - R 4.5.0 - ~/r
cannot open file 'ids_final_project_group_09_news_raw.csv': No such file or directory

> news_data <- read.csv("D:/ids_final_project_group_09_news_raw.csv", stringsAsFactors = FALSE)
> head(news_data)

title
1 Russia launches largest aerial assault on Ukraine, killing at least 12, despite completing major
prisoner swap
2 Nine out of Gaza doctora\u00f1os ten children killed in Isra
eli airstrike
3 Kim Jong Un\u00e1\u00f3s fury after watching North Korea\u00e1\u00f3s new navy destroyer crippled in b
otted launch
4 Key cartel member with $1 million US bounty on his head is killed, says Mexic
an government
5 Cannes film festival impacted after major power cut hits so
uthern France
6 US-Iran latest nuclear talks end with limited progress, as Tehran sources expre
```

Session 2 (Right):

- Environment:**

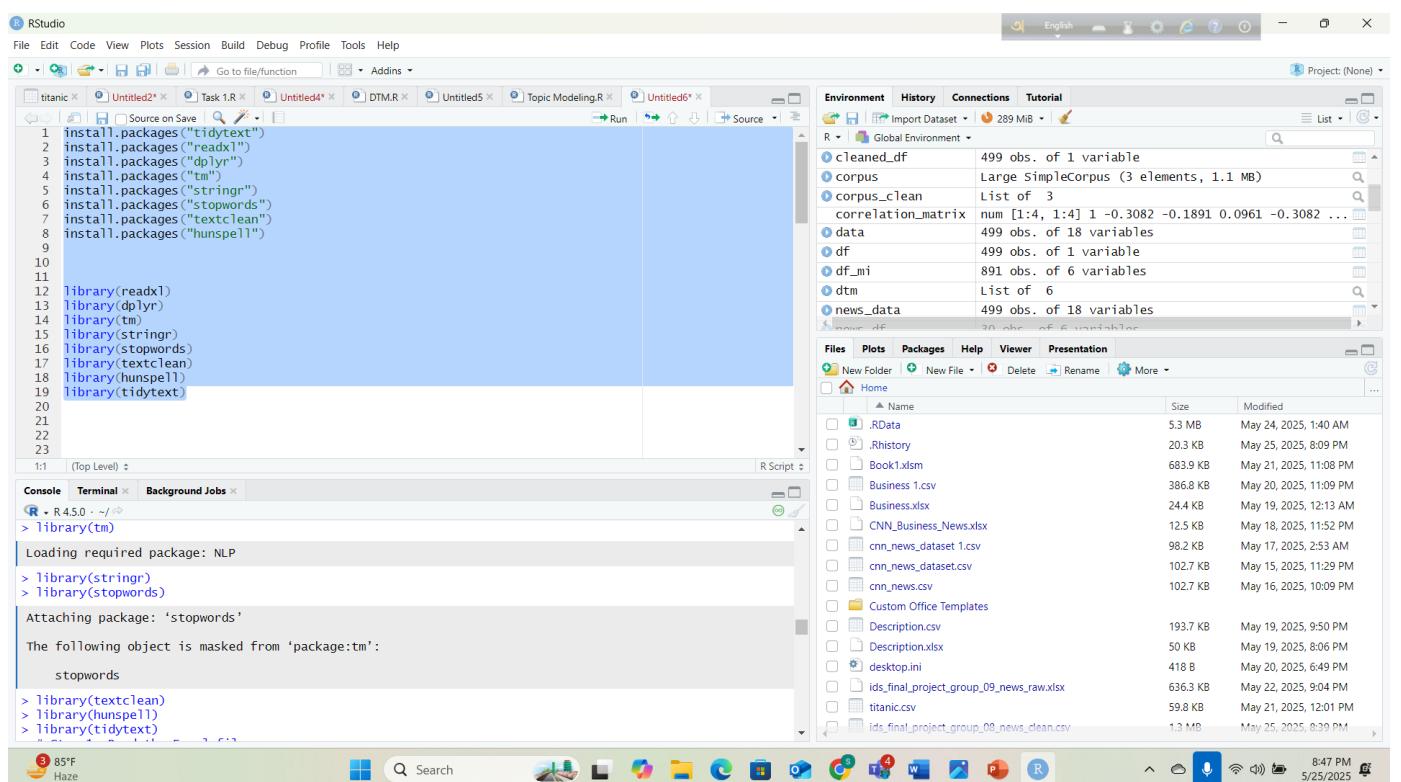
Name	Type	Size	Modified
.RData	Binary	5.3 MB	May 24, 2025, 1:40 AM
.Rhistory	Text	21.8 KB	May 25, 2025, 4:14 PM
Book1.xlsx	Excel	683.9 KB	May 21, 2025, 11:08 PM
Business1.csv	CSV	386.8 KB	May 20, 2025, 11:09 PM
Business.xlsx	Excel	24.4 KB	May 19, 2025, 12:13 AM
CNN_Business_News.xlsx	Excel	12.5 KB	May 18, 2025, 11:52 PM
cnn_news_dataset 1.csv	CSV	98.2 KB	May 17, 2025, 2:53 AM
cnn_news_dataset.csv	CSV	102.7 KB	May 15, 2025, 11:29 PM
cnn_news.csv	CSV	102.7 KB	May 16, 2025, 10:09 PM
Custom Office Templates	Office	193.7 KB	May 19, 2025, 9:50 PM
Description.csv	CSV	50 KB	May 19, 2025, 8:06 PM
Description.xlsx	Excel	418 B	May 20, 2025, 6:49 PM
desktop.ini	File	636.3 KB	May 22, 2025, 9:04 PM
ids_final_project_group_09_news.xlsx	Excel	59.8 KB	May 21, 2025, 12:01 PM
titanic.csv	CSV	59.8 KB	May 21, 2025, 12:01 PM
- Files:** A sidebar showing various files and folders, including .RData, .Rhistory, Book1.xlsx, Business1.csv, Business.xlsx, CNN_Business_News.xlsx, cnn_news_dataset 1.csv, cnn_news_dataset.csv, cnn_news.csv, Custom Office Templates, Description.csv, Description.xlsx, desktop.ini, ids_final_project_group_09_news.xlsx, and titanic.csv.

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.

ids_final_project_group_09_news_clean

```
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 = " "))
}
```

The screenshot shows the RStudio interface. The Source editor on the left contains R code for reading an Excel file, cleaning text, and removing stopwords. The Console tab below it shows the execution of this code, including the creation of a dataset and the removal of stopwords. The Environment browser on the right lists global variables like 'text_data' and functions like 'remove_custom_stopwords'. The Files browser at the bottom shows a project directory with various files and folders.

```
21
22
23
24
25
26
27
28
29
30
31
32 file_path <- "C:/Users/sanvi/OneDrive/Documents/ids_final_project_group_09_news_raw.xlsx"
33 df <- read_excel(file_path)
34
35
36 remove_custom_stopwords <- function(text) {
37   words <- unlist(strsplit(text, "\\s+"))
38   cleaned_words <- words[!tolower(words) %in% stopwords("en")]
39   return(paste(cleaned_words, collapse = " "))
40 }
41
42
43 <function(x) <

```

```
R 4.5.0 - ./R
> dt <- read_excel(file_path)
> # Step 2: Define a custom stopword remover
> remove_custom_stopwords <- function(text) {
+   words <- unlist(strsplit(text, "\\s+"))
+   cleaned_words <- words[!tolower(words) %in% stopwords("en")]
+   return(paste(cleaned_words, collapse = " "))
+ }
> file_path <- "C:/Users/sanvi/OneDrive/Documents/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 = " "))
+ }
>
```

Name	Size	Modified
.RData	5.3 MB	May 24, 2025, 1:40 AM
.Rhistory	20.3 KB	May 25, 2025, 8:09 PM
Book1.xlsx	683.9 KB	May 21, 2025, 11:08 PM
Business 1.csv	386.8 KB	May 20, 2025, 11:09 PM
Business.xlsx	24.4 KB	May 19, 2025, 12:13 AM
CNN_Business_News.xlsx	12.5 KB	May 18, 2025, 11:52 PM
cnn_news_dataset 1.csv	98.2 KB	May 17, 2025, 2:53 AM
cnn_news_dataset.csv	102.7 KB	May 15, 2025, 11:29 PM
cnn_news.csv	102.7 KB	May 16, 2025, 10:09 PM
Custom Office Templates		
Description.csv	193.7 KB	May 19, 2025, 9:50 PM
Description.xlsx	50 KB	May 19, 2025, 8:06 PM
desktop.ini	418 B	May 20, 2025, 6:49 PM
ids_final_project_group_09_news_raw.xlsx	636.3 KB	May 22, 2025, 9:04 PM
titanic.csv	59.8 KB	May 21, 2025, 12:01 PM

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))
```

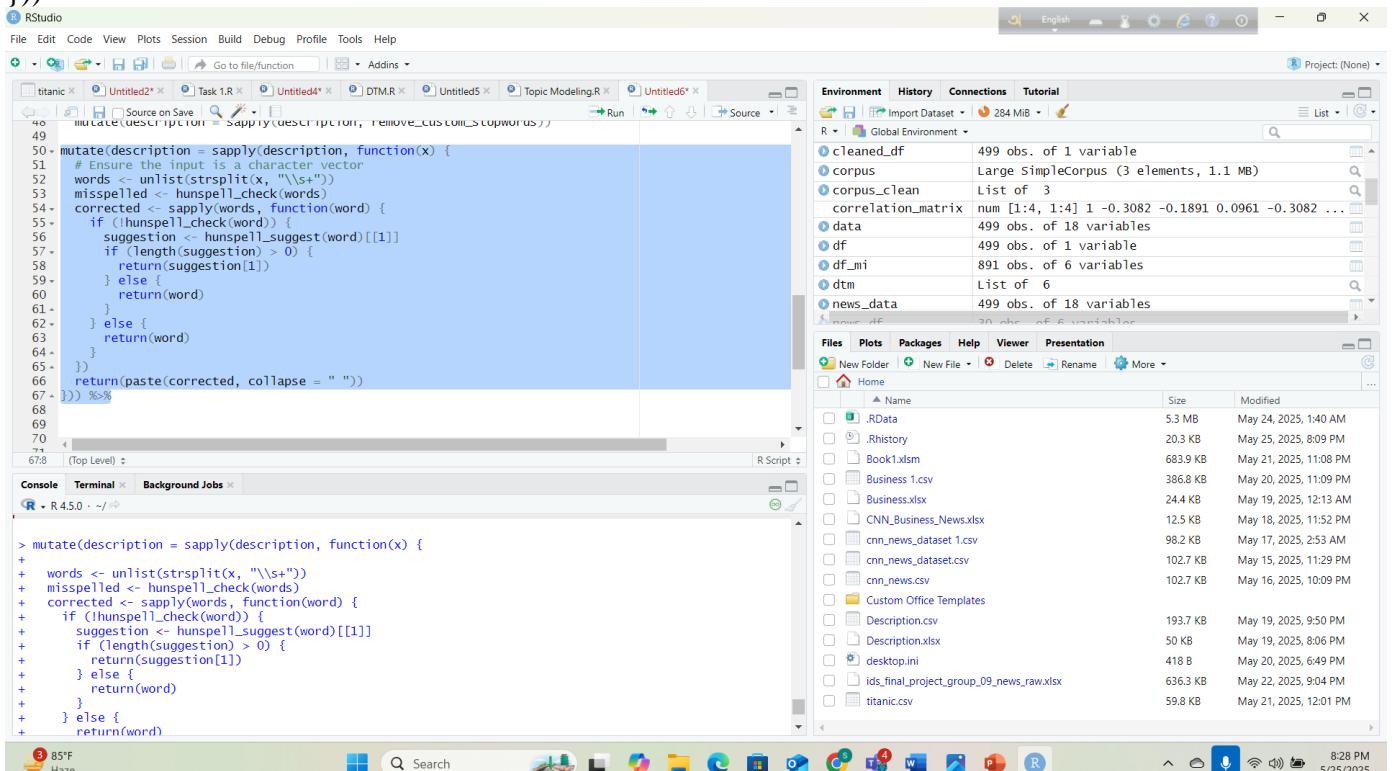
The screenshot shows the RStudio interface with the following details:

- File Menu:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Editor Pane:** Shows R code for cleaning a dataset. The code includes functions for removing custom stopwords, punctuation, numbers, and squishing words. It also includes code to read an Excel file and apply the cleaning process to the data.
- Environment Pane:** Displays the global environment with various objects and their characteristics.
- Console Pane:** Shows the R command history and output.
- Bottom Bar:** Includes icons for search, file operations, and system status.

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)) %>%
```

The screenshot shows the RStudio interface with the following details:

- Top Bar:** Shows "RStudio" and various menu options: File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Left Panel:** A code editor window containing R code. The code includes a function definition for `correct_sentence` and a script block for cleaning the dataset. The script ends with `cat("Cleaned paragraph-level data saved as 'ids_final_project_group_08_news_clean.csv'\n")`.
- Right Panel:**
 - Environment Tab:** Shows the global environment with objects like `cleaned_df`, `corpus`, `corpus_clean`, `correlation_matrix`, `data`, `df`, `df_mi`, `dtm`, and `news_data`.
 - Files Tab:** Shows a file tree with files like .RData, .Rhistory, Book1.xlsx, Business 1.csv, Business.xlsx, CNN_Business_News.xlsx, cnn_news_dataset 1.csv, cnn_news_dataset.csv, cnn_news.csv, Custom Office Templates, Description.csv, Description.xlsx, desktop.ini, ids_final_project_group_09_news_raw.xlsx, and titanic.csv.
- Bottom Bar:** Shows system status icons (weather, battery, signal) and the date/time (8:28 PM, 5/25/2025).

Stemming and Emoji Removal

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

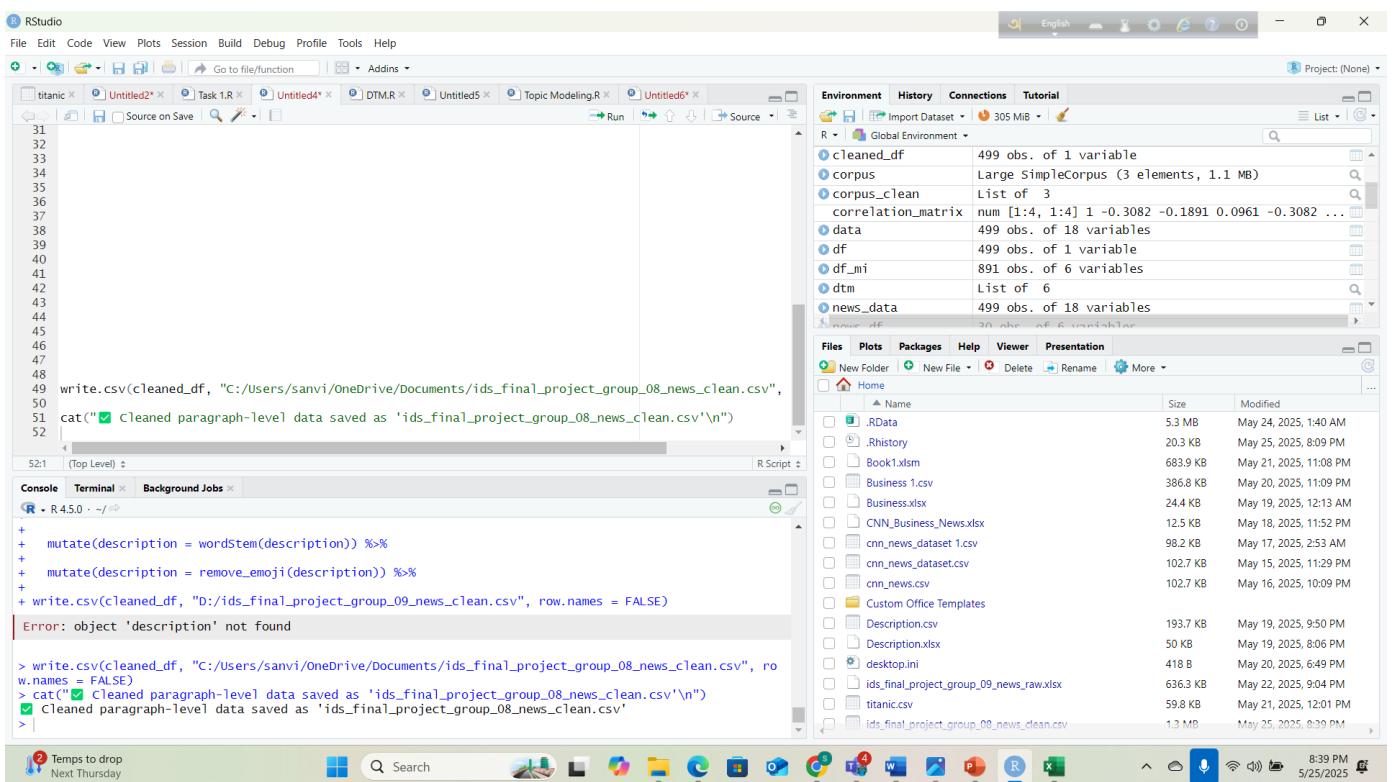
remove_emoji(description) strips out any emojis or symbols that might've survived earlier cleaning steps.

```
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")

ids_final_topic_modeling_project_group_09



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("tidyverse")
```

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(tidyverse)
```

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.

The screenshot shows the RStudio interface with the following details:

- Code Editor:** Displays the R script for loading data and creating a corpus. It includes imports for various packages like readr, tm, SnowballC, topicmodels, tidytext, dplyr, ggplot2, reshape2, and tidyverse. It checks for the 'clean_description' column and creates a corpus object from it.
- Environment View:** Shows the global environment with objects like dtm, dtm_matrix, lda_model, lda_terms, most_common, terms, test, top_terms, topic_dist, and train.
- File Explorer:** Shows the project structure with files like RData, .Rhistory, and various Word and PDF documents.
- Console:** Displays the R session output, including the command to run the script and the resulting output.
- Taskbar:** Shows the Windows taskbar with various application icons.
- System Tray:** Shows the date and time (10:21 PM 5/25/2025).

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")
```

The screenshot shows the RStudio interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help, and Project (None). The main area has tabs for 'part_1.R', 'ids_final_project_text_preprocessing_gr...', 'Untitled1*', 'Untitled2*', and 'ids_final_topic_modeling_project_group...'. The code editor contains R script code for data cleaning, creating a Document-Term Matrix (DTM), and saving it as a CSV. The Environment browser shows objects like dtm, dtm_matrix, lda_model, lda_terms, terms, test, top_terms, topic_dist, and train. The Files browser shows various files in the current directory, including RData, Rhistory, and several Word and PDF documents. The bottom console shows the execution of the R code.

```

20 library(dplyr)
21 library(cpp11)
22 library(reshape2)
23 library(tidyR)
24
25 file_path <- "E:/Data Science/ids_final_project_group_09_news_clean1.csv"
26 data <- read_csv(file_path)
27
28 if (!"clean_description" %in% colnames(data)) {
29   stop("Column 'clean_description' not found in the dataset.")
30 }
31
32 corpus <- Corpus(VectorSource(data$clean_description))
33
34
35 dtm <- DocumentTermMatrix(corpus)
36
37 inspect(dtm[1:5, 1:10])
38
39 dtm_matrix <- as.matrix(dtm)
40 write.csv(dtm_matrix, "E:/Data Science/clean_description_dtm.csv", row.names = FALSE)
41
42 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)
}
```

The screenshot shows the RStudio interface. In the top-left, there's a code editor with an R script containing LDA topic modeling code. The code includes reading a CSV file, creating a corpus, building a DocumentTermMatrix, fitting an LDA model, and extracting the beta matrix. The R console below shows the execution of these commands and some statistics about the matrix. On the right, there's a file browser pane showing various files like RData, history, and word documents.

```

26 data <- read_csv(file_path)
27
28 if (!"clean_description" %in% colnames(data)) {
29   stop("Column 'clean_description' not found in the dataset.")
30 }
31
32 corpus <- Corpus(VectorSource(data$clean_description))
33
34
35 dtm <- DocumentTermMatrix(corpus)
36
37 inspect(dtm[1:5, 1:10])
38
39 dtm_matrix <- as.matrix(dtm)
40 write.csv(dtm_matrix, "E:/Data Science/clean_description_dtm.csv", row.names = FALSE)
41
42 cat("DTM Dimensions: ", dim(dtm)[1], "documents x", dim(dtm)[2], "terms\n")
43
44
45 dtm <- removeSparseTerms(dtm, 0.95)
46
47 lda_model <- LDA(dtm, k = 5, control = list(seed = 1234))
48
49 lda_terms <- tidy(lda_model, matrix = "beta")

```

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)

```

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Source Console Terminal Background Jobs
R 4.5.0 - ~/r
===== Topic 1 =====
# A tibble: 10 × 2
  term      beta
  <chr>    <dbl>
1 said     0.0349
2 trump    0.0290
3 "        0.0202
4 tariffs  0.0195
5 president 0.0155
6 trade    0.0116
7 prices   0.0113
8 china    0.0111
9 last     0.00915
10 trump's 0.00888

===== Topic 2 =====
# A tibble: 10 × 2
  term      beta
  <chr>    <dbl>
1 "        0.0395
2 said     0.0272
3 's       0.0232
4 like     0.0163
5 "        0.0141
6 just     0.0115
7 one      0.00944
8 're      0.00904
9 can      0.00850
10 time    0.00846

===== Topic 3 =====
# A tibble: 10 × 2
  term      beta
  <chr>    <dbl>
1 said     0.0398
2 health   0.0293
3 "        0.0274
4 kennedy  0.0163
5 cancer   0.0122
6 's       0.0114
7 "        0.0111
8 "        0.0110
9 "        0.0109
10 public   0.00938

===== Topic 4 =====
# A tibble: 10 × 2
  term      beta
  <chr>    <dbl>
1 said     0.0316
2 "        0.0222
3 new     0.0118
4 cnn     0.0104
5 one     0.00987
6 first   0.00978
7 "        0.00948
8 according 0.00920
9 also    0.00905
10 two    0.00897

===== Topic 5 =====
# A tibble: 10 × 2
  term      beta
  <chr>    <dbl>
1 "        0.0302
2 study   0.0254
3 said     0.0254
4 can     0.0243
5 water   0.0221
6 food    0.0166
7 "        0.0159
8 people  0.0135
9 risk    0.0126
10 health  0.0125
> # Extract topic proportions (gamma matrix)
> doc_topics <- tidy(lda_model, matrix = "gamma")
> # View the first few rows

```

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Source Console Terminal Background Jobs
R 4.5.0 - ~/r
===== Topic 1 =====
# A tibble: 10 × 2
  term      beta
  <chr>    <dbl>
1 said     0.0349
2 trump    0.0290
3 "        0.0202
4 tariffs  0.0195
5 president 0.0155
6 trade    0.0116
7 prices   0.0113
8 china    0.0111
9 last     0.00915
10 trump's 0.00888

===== Topic 2 =====
# A tibble: 10 × 2
  term      beta
  <chr>    <dbl>
1 "        0.0395
2 said     0.0272
3 's       0.0232
4 like     0.0163
5 "        0.0141
6 just     0.0115
7 one      0.00944
8 're      0.00904
9 can      0.00850
10 time    0.00846

===== Topic 3 =====
# A tibble: 10 × 2
  term      beta
  <chr>    <dbl>
1 said     0.0398
2 health   0.0293
3 "        0.0274
4 kennedy  0.0163
5 cancer   0.0122
6 's       0.0114
7 "        0.0111
8 "        0.0110
9 "        0.0109
10 public   0.00938

===== Topic 4 =====
# A tibble: 10 × 2
  term      beta
  <chr>    <dbl>
1 said     0.0316
2 "        0.0222
3 new     0.0118
4 cnn     0.0104
5 one     0.00987
6 first   0.00978
7 "        0.00948
8 according 0.00920
9 also    0.00905
10 two    0.00897

===== Topic 5 =====
# A tibble: 10 × 2
  term      beta
  <chr>    <dbl>
1 "        0.0302
2 study   0.0254
3 said     0.0254
4 can     0.0243
5 water   0.0221
6 food    0.0166
7 "        0.0159
8 people  0.0135
9 risk    0.0126
10 health  0.0125
> # Extract topic proportions (gamma matrix)
> doc_topics <- tidy(lda_model, matrix = "gamma")
> # View the first few rows

```

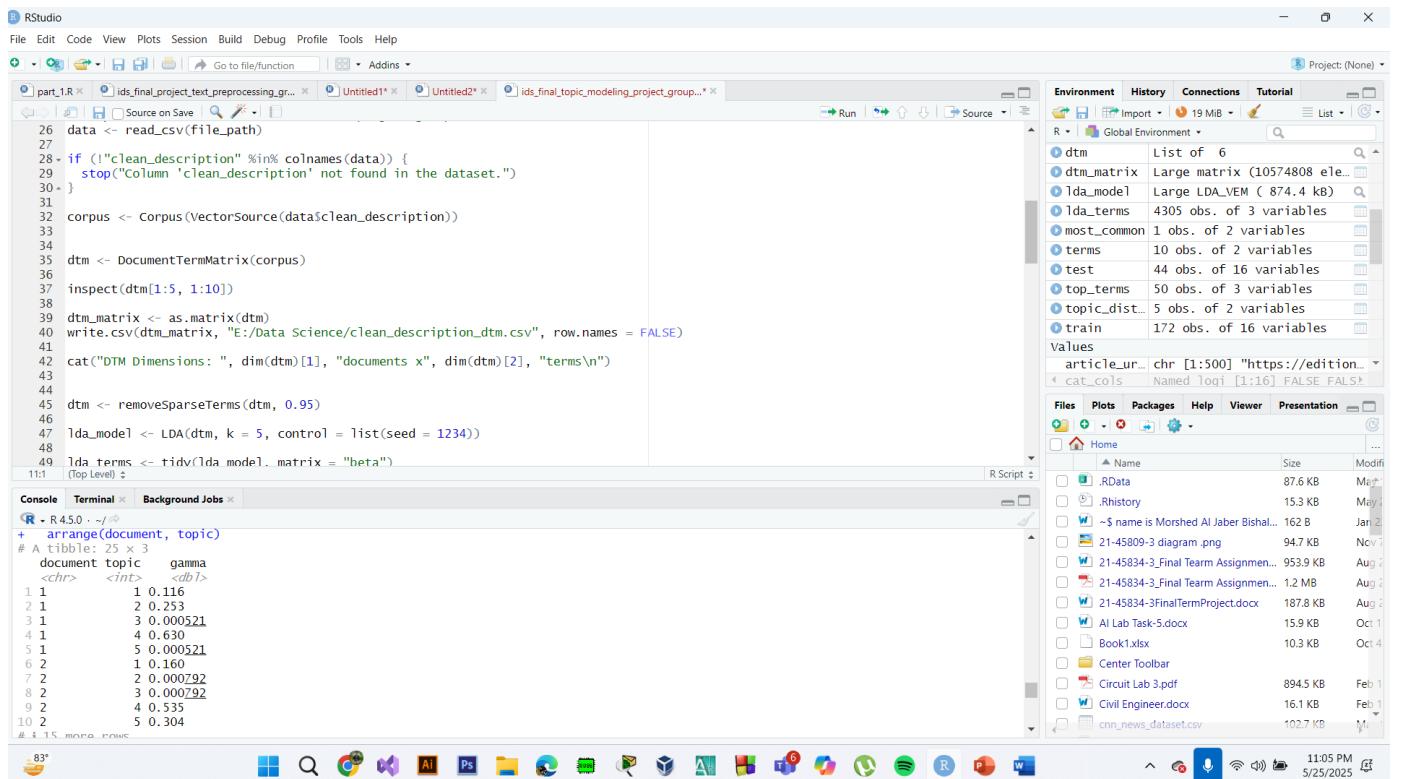
Purpose: This part (though placed slightly before the actual creation of doc_topics, which should be corrected) displays the topic proportions (gamma values) for the first 5 documents. It reshapes the data so that each document is a row and each topic is a column with its respective proportion.

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)
```

```
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)
```

This screenshot shows the RStudio interface with the following details:

- Code Editor:** The main pane displays R code for extracting topic proportions and dominant topics from an LDA model.
- Environment View:** Shows the global environment with objects like `dtm`, `dtm_matrix`, `lda_model`, etc.
- File Explorer:** Shows files in the project directory, including various documents and RData files.
- Console:** Displays the output of the R code, including the resulting topic distribution table and the dominant topic per document.
- Taskbar:** Shows standard Windows taskbar icons.
- System Tray:** Shows the date and time (11:09 PM, 5/25/2025).

```
43
44
45 dtm <- removeSparseTerms(dtm, 0.95)
46
47 lda_model <- LDA(dtm, k = 5, control = list(seed = 1234))
48
49 lda_terms <- tidy(lda_model, matrix = "beta")
50
51
52 top_terms <- lda_terms %>%
  filter(topic %% 1:5) %>%
  group_by(topic) %>%
  top_n(10, beta) %>%
  arrange(topic, -beta) %>%
  ungroup()
53
54 for (i in 1:5) {
55   cat(paste0("\n===== Topic ", i, " =====\n"))
56   terms <- top_terms %>%
57     filter(topic == i) %>%
58     select(term, beta)
59   print(terms)
60 }
61
62
63
64
65
66
```

```
R - R 4.5.0 - ~/R
+ J %>%
+   arrange(as.numeric(document)) -> doc_topic_wide
> # View the wide-format table
> print(doc_topic_wide)
# A tibble: 5 × 6
  document Topic_1 Topic_2 Topic_3 Topic_4 Topic_5
  <dbl>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
1      1     0.116    0.253   0.000521   0.630   0.000521
2      2     0.160    0.000792  0.000792   0.535   0.304
3      3     0.629    0.178   0.000464   0.192   0.000464
4      4     0.000179  0.295    0.0421    0.662   0.000179
5      5     0.00261   0.0374   0.00261   0.838   0.119
```

```
> # 1. Extract topic proportions (gamma) for each document
> doc_topics <- tidy(lda_model, matrix = "gamma")
> # 2. Find dominant topic per document
```

This screenshot shows the RStudio interface with the following details:

- Code Editor:** The main pane displays R code for creating a plot of topic distribution across documents.
- Environment View:** Shows the global environment with objects like `dtm`, `dtm_matrix`, `lda_model`, etc.
- File Explorer:** Shows files in the project directory, including various documents and RData files.
- Console:** Displays the output of the R code, including the resulting topic distribution table and the dominant topic per document.
- Taskbar:** Shows standard Windows taskbar icons.
- System Tray:** Shows the date and time (11:12 PM, 5/25/2025).

```
89
90 cat("\n===== Topic Distribution Across Documents =====\n")
91 print(topic_distribution)
92
93 ggplot(topic_distribution, aes(x = factor(topic), y = n, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  labs(
    title = "Number of Documents Dominated by Each Topic",
    x = "Topic",
    y = "Document Count"
  )
94
95 most_common <- topic_distribution %>% slice_max(n, n = 1)
96 cat(paste0(
  "Topic ", most_common$topic,
  " appears as the dominant topic in the highest number of documents (",
  most_common$n, " documents).\n"
))
97
98
99
100
101
102
103
104
105
106
107
108
109
110 write.csv(doc_topics, "E:/Data Science/topic_proportions_per_document.csv", row.names = FALSE)
111
112 top_terms %>%
  count(topic, sort = TRUE)
> print(topic_distribution)
# A tibble: 5 × 2
  topic   n
  <int> <int>
1     2    168
2     4    162
3     1     95
4     5     43
5     3     31
```

```
> # 5. Visualize the topic frequency
> ggplot(topic_distribution, aes(x = factor(topic), y = n, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  labs(
    title = "Number of Documents Dominated by Each Topic",
    x = "Topic",
    y = "Document Count"
  )
```

The screenshot shows the RStudio interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The main workspace shows R code for generating a plot of topic distribution across documents and writing it to a CSV file. The code uses ggplot2 and dplyr packages. The right pane displays the Global Environment, showing objects like dtm, dtm_matrix, lda_model, lda_terms, most_common, terms, test, top_terms, topic_dist..., and train. Below the environment is a file browser showing various files and folders, including a folder named 'Home' containing RData, RHistory, and several Word documents related to a project.

```

88
89 cat("\n\n===== Topic Distribution Across Documents =====\n")
90 print(topic_distribution)
91
92 ggplot(topic_distribution, aes(x = factor(topic), y = n, fill = factor(topic))) +
93   geom_col(show.legend = FALSE) +
94   labs(
95     title = "Number of Documents Dominated by Each Topic",
96     x = "Topic",
97     y = "Document Count"
98   )
99
100 most_common <- topic_distribution %>% slice_max(n, n = 1)
101 cat(paste0(
102   "Topic ", most_common$topic,
103   " appears as the dominant topic in the highest number of documents (",
104   most_common$n, " documents).\n"
105 ))
106
107
108
109
110 write.csv(doc_topics, "E:/Data Science/topic_proportions_per_document.csv", row.names = FALSE)
111

```

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