**AMERICAN INTERNATIONALA close up of a sign

Description automatically generated**

**UNIVERSITY-BANGLADESH**

Choose an item.

**Project**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Assignment Title: | Data Science Final Project | | | |
| Assignment No: | **Click here to enter text.** | | Date of Submission: | 25 May 2025 |
| Course Title: | Introduction to Data Science | | | |
| Course Code: | **Click here to enter text.** | | Section: | E |
| Semester: | Spring | 2024-25 | Course Teacher: | Abdus Salam |

**Declaration and Statement of Authorship:**

1. I/we hold a copy of this Assignment/Case-Study, which can be produced if the original is lost/damaged.
2. This Assignment/Case-Study is my/our original work and no part of it has been copied from any other student’s work or from any other source except where due acknowledgement is made.
3. No part of this Assignment/Case-Study has been written for me/us by any other person except where such collaboration has been authorized by the concerned teacher and is clearly acknowledged in the assignment.
4. I/we have not previously submitted or currently submitting this work for any other course/unit.
5. This work may be reproduced, communicated, compared and archived for the purpose of detecting plagiarism.
6. I/we give permission for a copy of my/our marked work to be retained by the faculty for review and comparison, including review by external examiners.
7. I/we understand that Plagiarism is the presentation of the work, idea or creation of another person as though it is your own. It is a formofcheatingandisaveryseriousacademicoffencethatmayleadtoexpulsionfromtheUniversity. Plagiarized material can be drawn from, and presented in, written, graphic and visual form, including electronic data, and oral presentations. Plagiarism occurs when the origin of their arterial used is not appropriately cited.
8. I/we also understand that enabling plagiarism is the act of assisting or allowing another person to plagiarize or to copy my/our work.

*\* Student(s) must complete all details except the faculty use part.*

\*\* Please submit all assignments to your course teacher or the office of the concerned teacher.

|  |  |
| --- | --- |
| Group Name/No.: | 9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Name** | **ID** | **Program** | **Signature** |
| 1 | SABIHA IQBAL ETU | 22-47227-1 | BSc [CSE] |  |
| 2 | MUKSHIT SAFI OWASI | 22-47251-1 | BSc [CSE] |  |
| 3 | MAHMUDUS SAMI MAAHI | 22-46446-1 | BSc [CSE] |  |
| 4 | MD. JAHIRUL ISLAM NAHID | 22-47212-1 | BSc [CSE] |  |

|  |  |  |
| --- | --- | --- |
| ***Faculty use only*** | | |
| FACULTY COMMENTS | **Marks Obtained** |  |
|  |
|  |
|  | **Total Marks** |  |
|  |
|  |

**Introduction:**

In this project, we focused on extracting and processing news data from The Daily Star, one of Bangladesh's leading English-language news portals. We selected five diverse categories—**Business, Sports, Entertainment, Youth**, and **Life & Living**—and scraped **100 news articles from each,** resulting in a dataset of **500 articles**. For each article, we collected the **category, news link, title, description,** and **publication time.** The primary goal was to apply a comprehensive text preprocessing pipeline to the **description** field to prepare the data for further natural language processing tasks. The steps included **emoji handling** (replacing emojis with text), **text cleaning** (converting to lowercase, removing punctuation, numbers, and extra spaces), **tokenization**, **stopword removal, stemming and lemmatization,** and **basic spell correction.** The cleaned and processed text was stored in a new column called processed\_description. This workflow demonstrates how web scraping and text preprocessing can be combined to build a structured, clean dataset suitable for content analysis.

**Extract news from the News portal:**

**Required Library:**

A screenshot of a computer screen

AI-generated content may be incorrect.

The command is install.packages("rvest") installs the **rvest** package in R, which is used for web scraping, and library(rvest) loads the package so you can use its functions to extract data from web pages.

**Select Categories and their URL:**

A screenshot of a computer

AI-generated content may be incorrect.

This R code defines a named list called categories\_list that maps each news **category** to its corresponding **URL** on The Daily Star website. Each element of the list has a category name (e.g., Business, Sports, etc.) as the **key**, and the URL of that category's news page as the **value.** This list will be useful for looping through each category to scrape news articles from their respective pages.

**Function to Scrape News Articles from a Single Category**

A computer screen shot of a computer code

AI-generated content may be incorrect.

In this part of the project, we created a function named scrape\_category to automate the process of scraping news article titles and their corresponding links from a specific category page on The Daily Star website. The function accepts two inputs: the category URL and the category name. It then iteratively loads each paginated section of the category page, extracting the article titles and links using CSS selectors. To ensure only unique articles are stored, it filters out duplicates during each loop iteration. This process continues until 100 unique articles are collected for the given category. Additionally, a 1-second delay is added between requests to follow responsible web scraping practices. The output of this function is a clean and structured list of news titles and links for each selected category, laying the foundation for collecting full news content and metadata in the next steps.

**Extracting Descriptions and Publication Times for News Articles**

A screenshot of a computer code

AI-generated content may be incorrect.

A computer screen shot of a computer code

AI-generated content may be incorrect.

In this part, we trimmed the collected news titles and links to exactly 100, then used two functions to extract the description (by combining up to three meaningful paragraphs) and the publication time from each article page. These were applied to all links using sapply(), and the results—category, news link, title, description, and time—were stored in a structured data frame for further use. The get\_time function extracts the publication or update time from the page using the .color-iron selector and filters relevant time-related text using pattern matching (e.g., "Published on:" or "Last update on:").

**Scraping All Categories and Saving News Data**

A computer screen shot of a group

AI-generated content may be incorrect.

In this section, we automated the scraping process for all selected categories by using lapply() to loop through each category in the categories\_list. For each category, the scrape\_category function is called, and the results are combined into a single data frame using do.call(rbind, ...). This aggregated dataset, containing 500 news articles across five categories, is then saved to a CSV file named ids\_final\_project\_group\_9\_news\_raw.csv using write.csv(). A confirmation message is printed after successful completion.

**The output of extracting news from the news portal**

**A screenshot of a computer

AI-generated content may be incorrect.**

This is the output we get when extracting the information from the online news portal The Daily Star. We extract 100 news articles and save them (category, news link, title, time) in csv file.

**Applying the text processing steps :**

**Why Preprocessing Is Important -**

Raw text data is often messy and inconsistent. It contains contractions ("don't"), emojis, punctuation, HTML tags, numbers, spelling mistakes, and irrelevant words (stopwords) that can negatively affect the accuracy and performance of models or analyses. Preprocessing helps standardize the text, reduce noise, and highlight meaningful content.

The core objective of this work is to perform detailed **text preprocessing** on the news **description** field to prepare it for downstream tasks like content classification, sentiment analysis, or summarization. The following **text processing steps** were applied:

1. **Emoji Handling** – Replaced emojis with their textual descriptions.
2. **Text Cleaning** – Converted text to lowercase, removed punctuation, numbers, and extra whitespace.
3. **Tokenization** – Split text into individual tokens or words.
4. **Stopword Removal** – Eliminated common English stopwords that do not contribute much to meaning.
5. **Stemming and Lemmatization** – Reduced words to their root or base forms to normalize the vocabulary.
6. **Spell Checking** – Applied basic spell correction to improve textual quality, enforcing lowercase consistency.

The processed content was stored in a new column named processed\_description and the complete dataset was exported to a CSV file for further analysis or modeling. This pipeline showcases the integration of web scraping with natural language preprocessing, establishing a foundation for robust content mining and NLP-based news analytics.

**Required Libraries**

**A screenshot of a computer program

AI-generated content may be incorrect.**

These packages are used for text processing in R: **tm** handles text cleaning and management; textclean fixes messy or incorrect text; textstem reduces words to their base forms; tokenizers break text into words or sentences; and hunspell checks and corrects spelling errors. Together, they prepare text for analysis or modeling.

**Text Preprocessing**

**A screenshot of a computer program

AI-generated content may be incorrect.**

We defined four specific text preprocessing functions:

1. **expand\_contractions:** Converts shortened forms like “don’t” to “do not” using replace\_contraction().
2. **handle\_emojis:** Replaces emojis in the text with their descriptive words using replace\_emoji().
3. **clean\_text:** Converts text to lowercase, removes HTML tags, non-letter characters, extra spaces, and trims whitespace.
4. **tokenize\_text:** Splits the cleaned text into individual word tokens using tokenize\_words().

**Text Preprocessing**

A computer code with many text

AI-generated content may be incorrect.

Here’s what each function does specifically:  
5. **remove\_stopwords:** Filters out English stopwords from the token vector using stopwords("en").  
6. **stem\_and\_lemmatize:** First stems tokens with stem\_strings(), then lemmatizes them using lemmatize\_words() to get their base forms.  
7. **spell\_check:** Splits the text into words, checks each word’s spelling with hunspell\_check(), replaces misspelled words with the first suggested correction from hunspell\_suggest(), and returns the corrected lowercase text.

**A screenshot of a computer program

AI-generated content may be incorrect.**

This code defines a main function process\_text that applies a full text preprocessing pipeline to each element of a text vector. For each text entry, it:

1. Expands contractions (e.g., "don't" → "do not").
2. Replaces emojis with descriptive words.
3. Cleans the text by lowercasing, removing HTML tags, punctuation, numbers, and extra spaces.
4. Tokenizes the cleaned text into words.
5. Removes stopwords from the tokens.
6. Applies stemming and lemmatization to reduce words to their base forms.
7. Joins the processed tokens back into a single string.
8. Performs spell checking and correction.
9. Converts the final output to lowercase for consistency.

Then, it applies this function specifically to the description column of the news\_data dataframe, storing the cleaned text in a new column processed\_description.

Finally, it saves the updated dataframe with the processed text to a new CSV file at the specified location and prints a confirmation message indicating that preprocessing is complete.

**Final Output after doing Preprocessing on the description column**

**A screenshot of a computer

AI-generated content may be incorrect.**

Finally we get the clean text after doing the preprocessing and all other steps the description part is processed and saved in the csv file named ids\_final\_project\_group\_9\_news\_clean.

**Applying Top Modelling :**

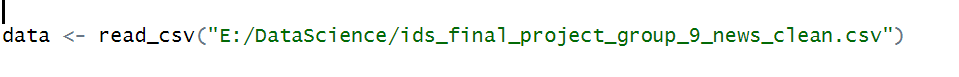
**Required Libraries**

**A white background with black dots

AI-generated content may be incorrect.**

These libraries help load data, clean and manage text (tm), build topic models (topicmodels), and handle data frames (dplyr, stringr).

**Loading the Preprocessed News Data**



In this step, we load the cleaned dataset into R using the read\_csv() function from the readr package. The file named ids\_final\_project\_group\_9\_news\_clean.csv contains the news articles that were collected and preprocessed in the earlier part of the project. This dataset includes multiple columns, but the one that is processed\_description, which holds the cleaned version of the news content. This step is essential because it brings our prepared data into the current R environment, making it ready for further analysis. Without loading this file, we would not be able to perform topic modeling on the news descriptions. This step forms the foundation for all the tasks that follow in the topic modeling part of the project.

**Creating a Text Corpus**

A black text on a white background

AI-generated content may be incorrect.

In this part, we create a text corpus using the VCorpus() function from the tm package. A corpus is a structured collection of text documents, and in this case, it is built from the processed\_description column of our dataset, which contains the cleaned news article descriptions. We use VectorSource() to tell R that the source of the text is a vector (a column) from our data frame. Creating a corpus is an important step in this project because it prepares the text for further processing and analysis. Once we create the corpus, we can apply various text cleaning functions and later convert it into a document-term matrix, which is needed for topic modeling.

**Cleaning the Text Again**

A black text on a white background

AI-generated content may be incorrect.

Here, we perform additional text cleaning on the corpus to ensure it is fully ready for topic modeling. Although the text was already cleaned earlier, it is necessary to apply these transformations again after creating the corpus, because tm\_map() functions work specifically with corpus objects. First, we convert all text to lowercase using content\_transformer(tolower) so that the same words in different cases are treated equally. Then, we remove all punctuation and numbers. We also remove a set of custom stopwords like common words such as “will,” “can,” “say,” and others that do not carry much meaning and can negatively affect model performance. Lastly, we use stripWhitespace to remove any extra spaces left behind. These preprocessing steps clean and normalize the text, making it consistent and reducing noise, which helps the topic modeling algorithm identify more meaningful patterns in the data.

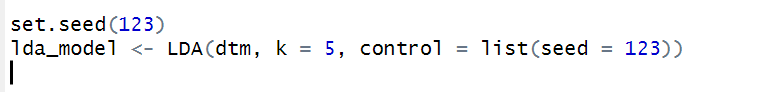
**Document-Term Matrix (DTM)**

A screenshot of a computer code

AI-generated content may be incorrect.

In this part, we convert the cleaned corpus into a Document-Term Matrix (DTM) using the DocumentTermMatrix() function. A DTM is a structured table where each row represents a document, and each column represents a unique word. The values in the matrix show how often each word appears in each document. This format is essential for topic modeling, as it allows algorithms like LDA to analyze word usage patterns across documents. After creating the DTM, we remove sparse terms using removeSparseTerms(dtm, 0.99). This means we keep only the words that appear in at least 1% of the documents, which helps remove rare or irrelevant terms. Then we check the row totals using apply(dtm, 1, sum) and remove any rows where the total is zero, which indicates documents that have no remaining words after filtering. This step ensures our DTM is clean, compact, and ready for topic modeling.

**The LDA Topic Model**



This part of the code sets a starting point for randomness using set.seed(123) so that the results are the same every time we run it. Then, we use LDA model to the document-term matrix (dtm) using the LDA() function. The parameter k = 5 specifies that the model should identify 5 distinct topics within the corpus. The control = list(seed = 123) argument further ensures that the internal random processes within the LDA algorithm are consistent across runs. Essentially, this part trains the topic model to uncover the hidden thematic structure of the text data by grouping words that frequently co-occur into five topics.

**Extracting Top Words for Each Topic**

A screen shot of a computer screen

AI-generated content may be incorrect.

This section extracts and displays the most representative words for each topic generated by the LDA model. The function terms(lda\_model, 10) is used to retrieve the top 10 terms (words) that have the highest probability of belonging to each topic in the model. These terms are considered the most relevant keywords that characterize the content of each topic. By examining these top terms, we can gain insight into the themes or subjects that the model has discovered within the text data. After extracting these terms, the code prints a message "Top terms in each topic:" to inform the actual lists of the top words for each of the five topics. This output is crucial for interpreting and labeling the topics in a meaningful way, as it provides a summary of the key vocabulary associated with each topic.

**Interpret Topics**

A screenshot of a computer

AI-generated content may be incorrect.

A computer code with text

AI-generated content may be incorrect.

The interpret\_topic function is designed to assign a meaningful label to each topic identified by the LDA model based on its top words. It takes a list of important words for a topic and converts them all to lowercase to ensure consistent matching. The function then compares these words against predefined keyword lists for five different news categories: Economic & Business News, Sports News, Entertainment News, Lifestyle & Living, and Youth & Education. For each category, it counts how many of the topic’s words appear in that category’s keyword list. Finally, it selects the category with the highest number of matching words as the best description for that topic and returns its name. This helps interpret the topics in human-readable terms by linking the model’s output to familiar subject areas.

A computer code with text

AI-generated content may be incorrect.

Here the loop iterates over the five topics generated by the LDA model, processing each topic sequentially. For every topic index i from 1 to 5, it first extracts the corresponding column of top terms from the top\_terms matrix, which represents the ten most significant words associated with that topic. These terms are then passed to the interpret\_topic function, which analyzes the words and determines the most appropriate category label for the topic based on predefined keyword lists. After obtaining the interpreted category, the loop prints a formatted message that includes the topic number, the list of top words for that topic, and the category label as an interpretation of the topic’s theme. Overall, this loop automates the process of summarizing the key terms for each topic and providing a human-readable label that helps in understanding and communicating the results of the topic modeling.