Information Retrieval and Web Analytics, 2025

Universitat
Pompeu Fabra
Barcelona

Project Part 1: Text Processing and Exploratory Data Analysis

Team: G_004

Alex De La Haya Gutiérrez, 268169

Marc Guiu Armengol, 268920

Nil Tomàs Plans, 268384

Github URL: https://github.com/niiltomas/irwa-search-engine-G_004.git

Tag: IRWA-2025-part-1

Index of Contents

Part 1

1.1.

Effective preprocessing of text data is a fundamental step to ensure accurate and meaningful analysis.

First we begin by tokenizing the text using the *word_tokenize*() function, which converts each string into a list of individual words (tokens), allowing for precise handling of each term.

Next, all tokens were converted to lowercase to standardize the text and avoid duplication caused by variations of capital letters.

Following this, stopwords and punctuation were removed. Using *NLTK* list of english stopwords along with Python's *string.punctuation*, we eliminated words and
symbols with little semantic value, reducing noise and focusing on meaningful content.
Finally, we applied the function *PorterStemmer*() to reduce words to their root form (e.g. "running" to "run") in order to simplify the vocabulary and improve consistency.

Finally, the cleaned tokens were joined back into strings and stored in new fields, *title_proc* and *description_proc*, in a copy of the original dataset. This preserves the original data while providing a processed version ready for analysis and modeling.

1.2.

For this step, we are required to return specific fields for each document:
{*pid, title, description, brand, category, sub_category, product_details, seller, out_of_stock, selling_price, discount, actual_price, average_rating, url*}. We do so by creating a new dataset keeping only necessary and ordered columns.

1.3.

We will handle categorical fields: *category, sub_category, brand, product_details, and seller* by keeping them as SEPARATE FIELDS instead of merging them into a single text field.
Why? Each one carries different semantic information (example: brand-> Nike, seller -> Flipkart). Merging them could reduce retrieval precision (searching Nike would match both brands and sellers). And we can also apply different weights per field in the inverted index.

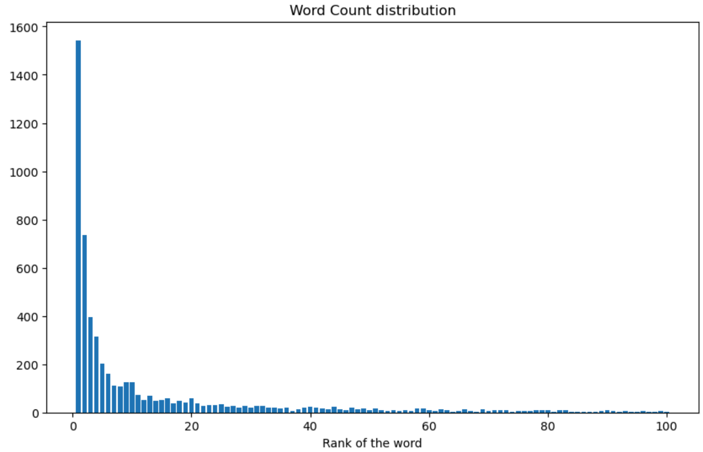
1.4.

We will keep *out_of_stock* as a boolean variable. Categories (*selling_price, discount, actual_price, and average_rating*) will be turned to floats numerical categories. It makes more sense to do so because we can search or filter on our index more efficiently. Example: Nike product that costs more than 50$ if our index is sorted it is much faster.

Part 2

2.1. Word Counting Distribution

In order to determine the word count distribution, we've designed a code that counts the word occurrences for all documents (**on the title and description fields after being preprocessed**), computes how many words share each frequency, which displays the following Word Count distribution chart.



We can clearly observe that the distribution follows *Zipf's Law,* which states that the product of a word's rank and its frequency is approximately constant. That means,
rank × *frequency* ≈ *constant*, or *rank* × *P(word occurrence)* ≈ *0.1* for english words.
In other words, a few words occur very frequently, while most words occur rarely.

2.2. Average sentence length

We have also calculated the average sentence length which consists of **18.84 words**.

2.3. Vocabulary size

After the implementation we can observe that there are **6143 different words** in the dataset used

2.4. Ranking of products based on rating, price, discount

We proceeded to rank the top 10 products based on rating, price, and discount metrics separately.

For the ratings, since some products had multiple ratings, we decided to use the average of those ratings to ensure consistency.

For the top 10 cheapest products, in cases where a product had multiple selling prices, we chose the lowest price to maintain uniformity. (This choice was somewhat arbitrary.) We could have used the average of all prices, but for the purposes of this task, we considered this approach more appropriate.

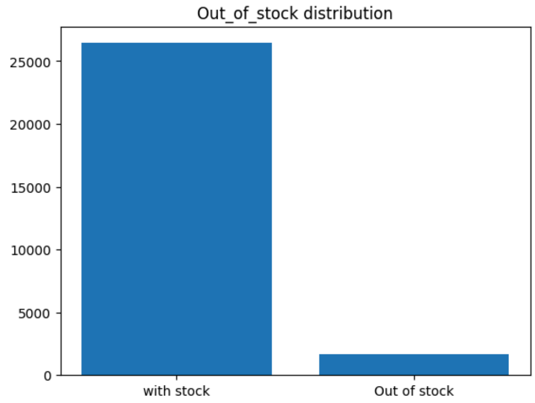
For the top 10 highest discounts, we applied the same method as for the cheapest products. Instead of taking the minimum price, we selected the highest discount for products with multiple entries.

2.5. Top sellers and brands

To identify the 10 top-selling products and the 10 most popular brands, we created the function *computing_top*(), which counts how many times each value in the specified field appears in the dataset.

2.6.Out_of_stock distribution

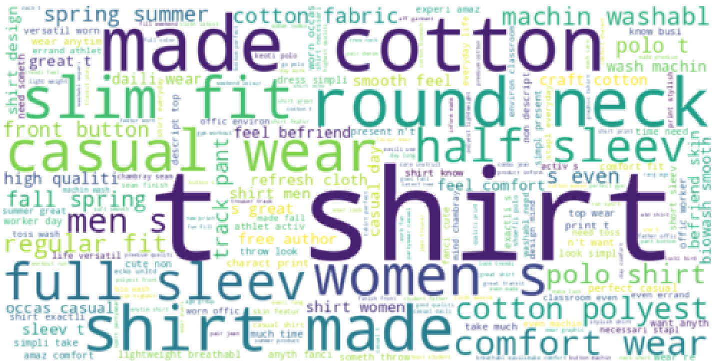
After implementing the respective algorithm, we've observed that the number of products in-stock is much higher than the number of products out of stock. This is what we expected, a business must have supply to sell to its customers.



2.7. Word clouds for the most frequent words

We have plotted the most frequent words using word clouds. The bigger the name the more frequent. We can observe that the most common words are t-shirt, made, cotton and round neck, among others.

To achieve the task, we've used the field of description after being pre-processed in order to achieve a higher relevance and avoid that stopping words appeared.



2.8. Entity recognition

We categorized words in the *description_proc* field into entity types using the library *spaCy*, for natural language processing. SpaCy's Named Entity Recognition (NER) automatically identifies entities like dates (DATE), organizations (ORG), persons(PERSONS), among others.

We applied the *en_core_web_sm* model to the first 200 processed descriptions in the pre-processed dataset, extracted only the entity types, and counted their occurrences. This approach allows us to quickly and consistently extract structured information from product descriptions for further analysis.

We chose the *en_core_web_sm* model because it is lightweight, fast, and sufficient for basic entity recognition tasks, as required in the exercise.

For example, the entities found among the top of higher appearances are: CARDINAL, PERSON, and DATE.