UPF – Information Retrieval and Web Analytics

Final Project Part 1

Jhonatan Barcos Gambaro (u198728)

Daniel Alexander Yearwood Agames (u214976)

Part 1: Text Processing and Exploratory Data Analysis

1. Delivery information

Firstly, relevant information regarding the delivery of the project is recorded.

Link Github: https://github.com/laxhoni/IRWA_Final_Project

TAG: part-1-submission:

https://github.com/laxhoni/IRWA_Final_Project/releases/tag/part-1-submission

2. Introduction

The aim of this first part of the project is to prepare the document corpus, a dataset based on fashion products from an e-commerce site. To achieve our goal, we have divided the development of the project into two distinct parts:

[3] Text pre-processing and analysis and strategy for different fields: Creation of a natural language processing (NLP) pipeline for unstructured text fields, as well as defining the strategy for dealing with structured categorical and numerical fields, focusing on optimisation for text search and filtering, which will be implemented in future practices.

[4] Data exploration and analysis: Analysing the resulting dataset after processing and understanding its characteristics and the relationship between them.

3. Text Pre-Processing

3.1. Processing Pipeline

As a first step in processing the textual components of our corpus, we defined the text processing pipeline by creating the clean_text_simple function. This function performs the following steps to achieve our objective:

```python
def clean_text_simple(text):
    word_tokens = word_tokenize(text.lower())
    textos_limpios = ' '.join([word for word in word_tokens if word not in stop_words and word.isalnum()])
    textos_limpios = ' '.join([stemmer.stem(word) for word in word_tokenize(textos_limpios)])
    return textos_limpios
```

1. **Conversion to lowercase**: All words in our corpus are converted to lowercase to avoid duplicates, thus minimising the size of the corpus.
2. **Removal of stopwords with nltk**: Words that do not add value to our corpus are removed. The NLTK library is used for this purpose.
3. **Tokenization with nltk**: The corpus is converted into tokens for greater accuracy.
4. **Punctuation removal**: Punctuation marks are removed as they do not add semantic value and make searching difficult.
5. **Stemming with nltk's PorterStemmer**: Words are reduced to their suffixes to minimise the size of the corpus and maximise search speed.

3.2. Processing Pipeline [Bonus]

After analysing the cleanliness of the text, it has been found that there is considerable room for improvement with respect to the initial pipeline. Therefore, the following decisions have been made and a new pipeline has been defined using the clean_text function:

```python
# Define new stop words that depends on the domini of the data
stop_words_domini = {
    'made', 'india', 'proudly', 'use', 'year', 'round',
    'look', 'design', 'qualiti', 'day', 'make',
    'feel', 'perfect', 'great', 'wash', 'style',
}
stop_words = stop_words.union(stop_words_domini)

def clean_text(text):
    # NEW: Remove numbers
    text = re.sub(r'\d+', '', text)
    word_tokens = word_tokenize(text.lower())
    # NEW: Stop words updated
    textos_limpios = ' '.join([word for word in word_tokens if word not in stop_words and word.isalnum()])
    textos_limpios = ' '.join([stemmer.stem(word) for word in word_tokenize(textos_limpios)])
    return textos_limpios
```

1. Add frequent words inherent to the domain of our corpus that do not add value to the set of stopwords. If we check these words, defined in stop_words_domini, we can see that they are generic words that would be relevant for other corpora but are so common in ours that they add no value.
2. Remove numerical values from the corpus, as numbers such as ‘100’ were very frequent due to the definition of discounts in the corpus.

Once the processing pipeline has been defined, it is applied to the textual components 'title' and 'description'. Below we can see the difference between these variables before and after cleaning:

Title: Solid Women Multicolor Track Pants

Cleaned Title: solid women multicolor track pant

Description: Yorker trackpants made from 100% rich combed cotton giving it a rich look.Designed for Comfort,Skin friendly fabric,itch-free waistband & great for all year round use Proudly made in India

Cleaned Description: yorker trackpant rich comb cotton give rich comfort skin friendli fabric waistband

We observe how the application of our pipeline has drastically minimised the size of our corpus and correctly selected the keywords for each textual variable.

3.3. Handle of category, sub_category, brand, product_details, and seller during pre-processing.

Firstly, in order to make the best decision about how to treat these variables, we print the values in these columns as well as the number of unique values to check their cardinality.

```
Displaying first 3 products of the columns to analyze:
```

	category	sub_category	brand	product_details	seller
0	Clothing and Accessories	Bottomwear	York	[{'Style Code': '1005COMBO2'}, {'Closure': 'El...	Shyam Enterprises
1	Clothing and Accessories	Bottomwear	York	[{'Style Code': '1005BLUE'}, {'Closure': 'Draw...	Shyam Enterprises
2	Clothing and Accessories	Bottomwear	York	[{'Style Code': '1005COMBO4'}, {'Closure': 'El...	Shyam Enterprises

```
Displaying unique values in each column to analyze:

category          4
sub_category     24
brand           325
seller          535
dtype: int64
```

Once these variables have been analysed, we can proceed to make decisions:

Should they be merged into a single text field, indexed as separate fields in the inverted index or any other alternative?

We apply a hybrid approach to handle these fields. Each field is indexed separately to preserve its distinct meaning, and a combined `all_text` field is created (including title, description, brand, category, sub_category, seller, and product_details) for full-text retrieval and ranking.

Justify your choice, considering how their distinctiveness may affect retrieval effectiveness.

These fields provide different types of information.

For example, `title` and `description` describe the product in natural language and should be processed as normal text.

On the other hand, `brand`, `category`, and `seller` represent structured metadata that identify key characteristics of a product.

`product_details` adds specific attributes like color or fabric, which can also be flattened into text to support attribute-based queries.

If we merged all of them into a single text field, we would lose the semantic distinction between textual content and metadata, which could reduce precision.

Keeping them completely separate, however, could lower recall, since relevant terms might appear across multiple fields.

By combining both ideas—maintaining individual fields and also building an aggregated one—we balance precision and recall (through the combined text representation).

What are the pros and cons of each approach?

Approach	Description	Pros	Cons
Single merged field	All fields concatenated into one text	Simple implementation; high recall	Loses field meaning; cannot boost or filter by specific attributes
Separate fields only	Each field indexed independently	Enables accurate filtering and field weighting	Lower recall if query terms are distributed across fields
Hybrid (recommended)	Index fields separately and also create a merged `all_text` field	Balances recall and precision; supports both text search and structured filters	Slightly more complex indexing and larger index size

We'll use the hybrid approach chosen for improvement of the search in the next labs but we just build the base of our system.

Below, we define all the functions necessary to implement this hybrid system, which will allow us to define new features to improve the basic inverted index search implementation in the following parts of the project.

1. **preprocess_full_text**: to preprocess text for full-text field (All Text)

```python
def preprocess_full_text(text):
    if not isinstance(text, str):
        return []

    word_tokens = nltk.word_tokenize(text.lower())

    tokens_nets = [
        stemmer.stem(word)
        for word in word_tokens
        if word not in stop_words and word.isalnum()
    ]

    return tokens_nets
```

2. **preprocess_keyword**: to preprocess text for keyword fields (Fielded Index)

```python
def preprocess_keyword(text):
    if not isinstance(text, str):
        return None

    text = text.lower().strip()
    if not text:
        return None

    text = re.sub(r'\s+', '_', text)
    text_net = ''.join(char for char in text if char.isalnum() or char == '_')

    return text_net
```

3. **process_product_details**: to process product_details list of dicts into text for All Text and keyword fields for Fielded Index

```python
def process_product_details(details_list):
    text_for_all_text_list = []
    keyword_fields_dict = {}
    if not isinstance(details_list, list):
        return '', keyword_fields_dict
    for item_dict in details_list:
        if isinstance(item_dict, dict):
            for key, value in item_dict.items():
                if isinstance(value, str):
                    # 1. Recall fields per al Canal 1 (All Text)
                    text_for_all_text_list.append(key)
                    text_for_all_text_list.append(value)

                    # 2. Precision fields per al Canal 2 (Fielded Index)
                    kw_key_name = f"detail_{preprocess_keyword(key)}"
                    kw_value = preprocess_keyword(value)
                    if kw_key_name and kw_value:
                        keyword_fields_dict[kw_key_name] = kw_value

    return ' '.join(text_for_all_text_list), keyword_fields_dict
```

Once the functions have been defined, they are applied to the different columns of our dataset, resulting in new characteristics broken down by category, description, etc. The most relevant ones for improving our search system are shown below.

	pid	all_text	brand_kw	category_kw	detail_color	detail_fabric
0	TKPFCZ9EA7H5FYZH	[solid, women, multicolor, track, pant, yorker...	york	clothing_and_accessories	multicolor	cotton_blend
1	TKPFCZ9EJZV2UVRZ	[solid, men, blue, track, pant, yorker, trackp...	york	clothing_and_accessories	blue	cotton_blend
2	TKPFCZ9EHFCY5Z4Y	[solid, men, multicolor, track, pant, yorker, ...	york	clothing_and_accessories	multicolor	cotton_blend
3	TKPFCZ9ESZZ7YWEF	[solid, women, multicolor, track, pant, yorker...	york	clothing_and_accessories	multicolor	cotton_blend
4	TKPFCZ9EVXKBSUD7	[solid, women, brown, grey, track, pant, yorke...	york	clothing_and_accessories	brown_grey	cotton_blend

3.4. Handling of Numeric and Boolean Fields

How should the fields `out_of_stock`, `selling_price`, `discount`, `actual_price`, and `average_rating` be handled during pre-processing? Should they be indexed as textual terms?

Instead, they should be **converted and normalized** to numerical or boolean types for later use in ranking, sorting, or filtering:

- `out_of_stock` → boolean (`True`/`False`)

- `selling_price`, `actual_price` → float values

- `discount` → percentage value

- `average_rating` → numeric rating (float)

This ensures the search engine can use these features to refine results (for example, filtering available items or boosting products with higher ratings) without polluting the text index.

To achieve our goal, we define the clean_numeric function below, which will be applied to the columns “selling_price”, “actual_price”, “discount”, and “average_rating”. Furthermore, boolean is applied to “out of stock” and the valid range of “discount” is checked, between 0-100.

```python
# Helper function to clean numeric fields
def clean_numeric(value):
    if isinstance(value, str):
        value = re.sub(r'[^\d.,]', '', value).replace(',', '')
    try:
        return float(value)
    except:
        return np.nan

# Apply cleaning to numeric columns
for col in ['selling_price', 'actual_price', 'discount', 'average_rating']:
    products_cleaned[col] = products_cleaned[col].apply(clean_numeric)

# Ensure discount is in valid range
products_cleaned['discount'] = products_cleaned['discount'].clip(0, 100)
```

Finally, we visualise these new columns resulting from the cleaning of the numerical variables:

	pid	out_of_stock	selling_price	actual_price	discount	average_rating
0	TKPFCZ9EA7H5FYZH	False	921.0	2999.0	69.0	3.9
1	TKPFCZ9EJZV2UVRZ	False	499.0	1499.0	66.0	3.9
2	TKPFCZ9EHFCY5Z4Y	False	931.0	2999.0	68.0	3.9
3	TKPFCZ9ESZZ7YWEF	False	911.0	2999.0	69.0	3.9
4	TKPFCZ9EVXKBSUD7	False	943.0	2999.0	68.0	3.9

4. Exploratory Data Analysis

4.1. Dataset overview

As a first step, we explored the structure and completeness of the dataset to understand its composition and quality before applying any transformation.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28080 entries, 0 to 28079
Columns: 144 entries, _id to all_text
dtypes: bool(1), datetime64[ns](1), float64(4), object(138)
memory usage: 30.7+ MB

average_rating    2261
discount           855
actual_price       777
selling_price        2
brand                0
out_of_stock         0
seller               0
sub_category         0
category             0
dtype: int64
```

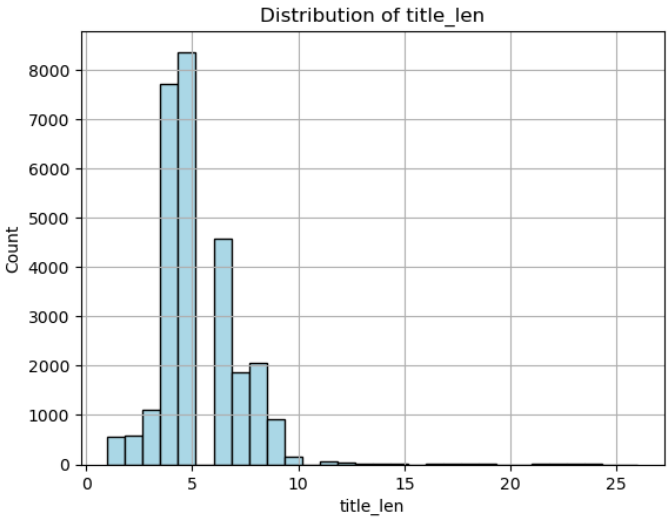
We can observe that the dataset contains 28,080 documents and 144 columns, combining textual, categorical, and numerical data.

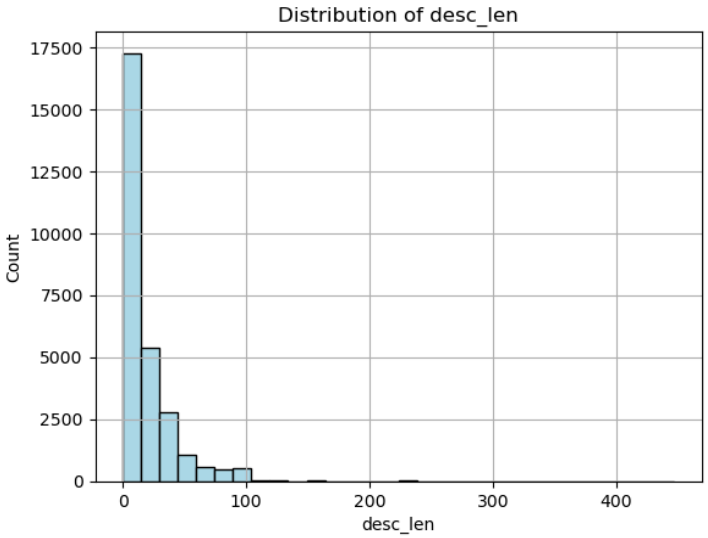
The main fields are complete, and only a few numerical attributes such as average_rating, discount, and actual_price present missing values.

This confirms that the dataset is consistent and suitable for analysis.

4.2. Textual Analysis

Once the cleaning pipeline was applied, we analysed the main textual fields (title and description) to understand their composition and richness.





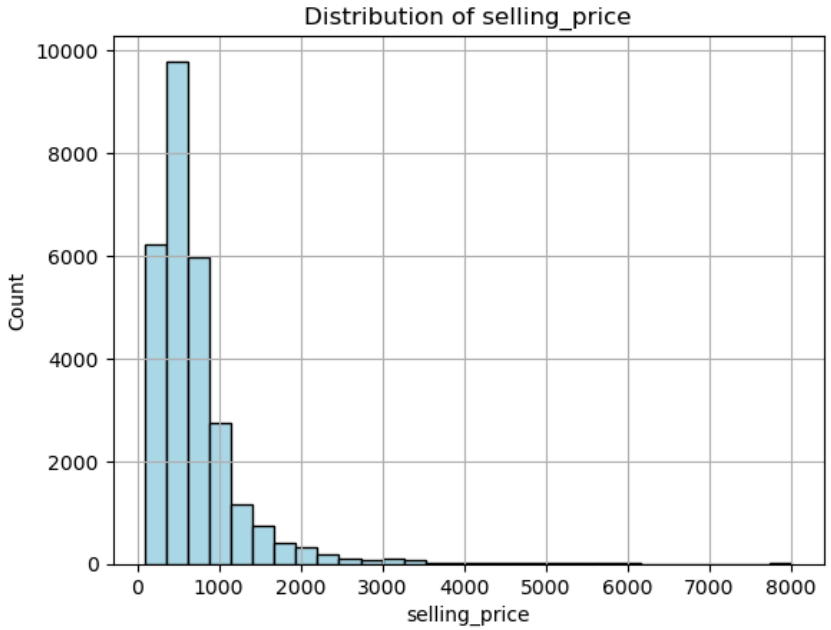
Most titles range between four and six tokens, while descriptions usually stay below one hundred tokens, with some longer cases exceeding three hundred. This indicates that titles provide concise information about the product, while descriptions add richer contextual details.

After preprocessing, the vocabulary contains 4,404 unique terms. The most frequent words are directly related to product type, materials and style, such as cotton, shirt, fit, fabric or casual. This confirms the domain-specific nature of the dataset.

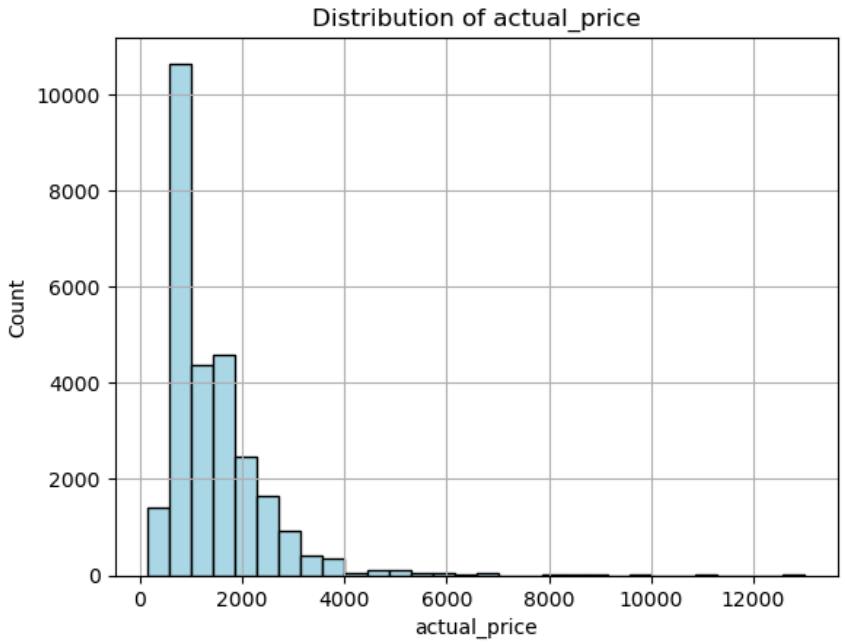
4.3. Price, Discount, and Rating Analysis

We analysed prices, discounts and ratings to understand scale and variability before using them for filtering and ranking.

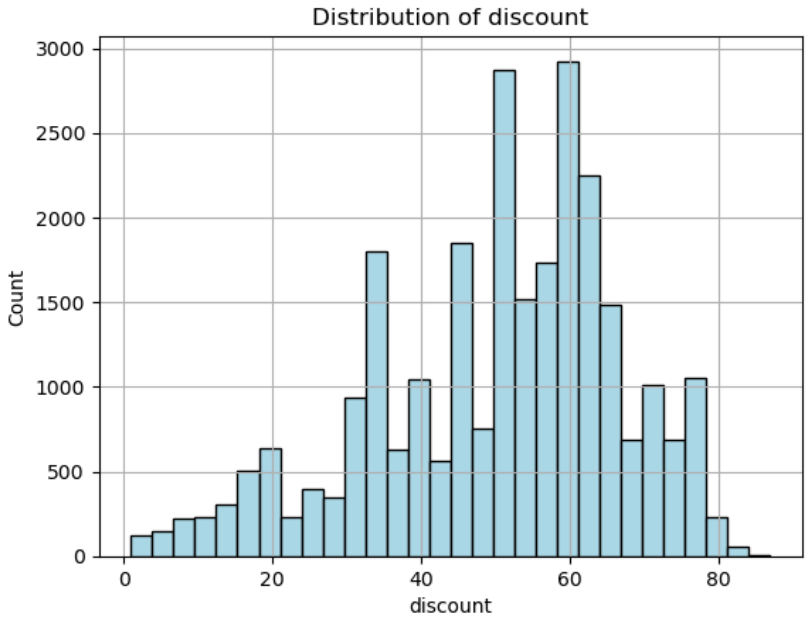
	selling_price	actual_price	discount	average_rating
count	28078.000000	27303.000000	27225.000000	25819.000000
mean	705.635088	1455.528110	50.256896	3.627724
std	549.681489	939.977456	16.887287	0.663429
min	99.000000	150.000000	1.000000	1.000000
25%	390.000000	849.000000	40.000000	3.200000
50%	545.000000	1199.000000	53.000000	3.800000
75%	820.000000	1799.000000	63.000000	4.100000
max	7999.000000	12999.000000	87.000000	5.000000



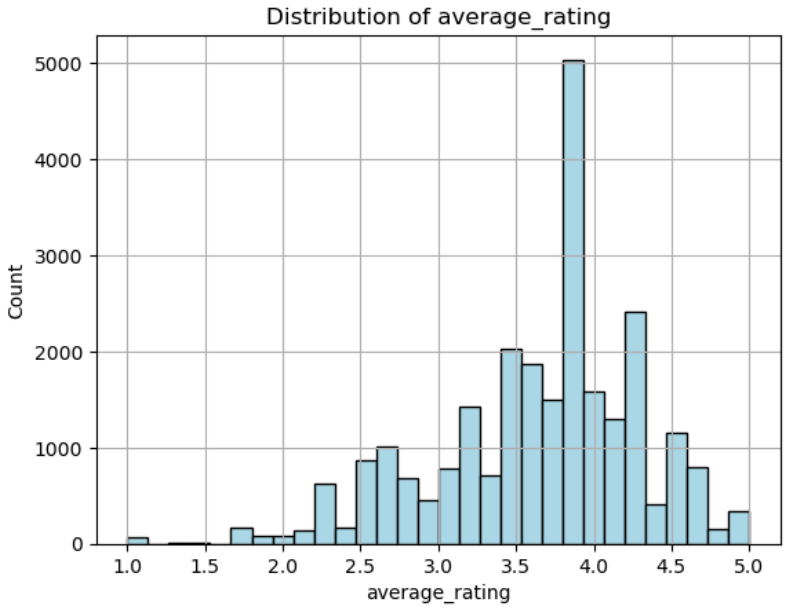
Selling prices are right-skewed, with most products between 390–820 (Q1–Q3) and a median of 545. A small tail reaches premium items up to 7,999.



Actual prices follow a similar pattern (median 1,199, Q1–Q3 849–1,799), confirming consistent discounting across the catalogue.



Discounts centre around 50–60% (median 53%; Q1–Q3 40–63%). Provides a useful signal for ranking variants of the same product.



Ratings are generally positive (mean 3.63, median 3.8) with most values between 3.2 and 4.1. This supports using rating as a mild booster rather than a hard filter.

Top 5 most expensive products:

	title	selling_price	brand
2067	solid women track suit	7999.0	REEB
11010	checker singl breast parti women full sleev bl...	7998.0	True Bl
25815	full sleev solid women casual jacket	7799.0	Pu
26089	full sleev print men sweatshirt	7799.0	Pu
6895	skinni men blue jean	7794.0	G

When looking at the most expensive products, we can see that they belong mainly to sportswear and outerwear categories. Items such as track suits and jackets from brands like Ree and Pu reach the highest prices, which explains the long right tail in the price distribution.

Top 5 biggest discounts:

	title	discount	brand
906	print women neck white	87.0	Jack Roy
902	print women neck grey	86.0	Jack Roy
903	print women neck white	86.0	Jack Roy
18249	print men neck multicolor pack	85.0	yellowvib
9813	solid balclava cap	84.0	Gracew

The biggest discounts correspond to specific promotions, with some items showing reductions above 80%. These extreme cases come mostly from brands such as Jack Roy and Yellowvib, and represent typical clearance or sales-season offers.

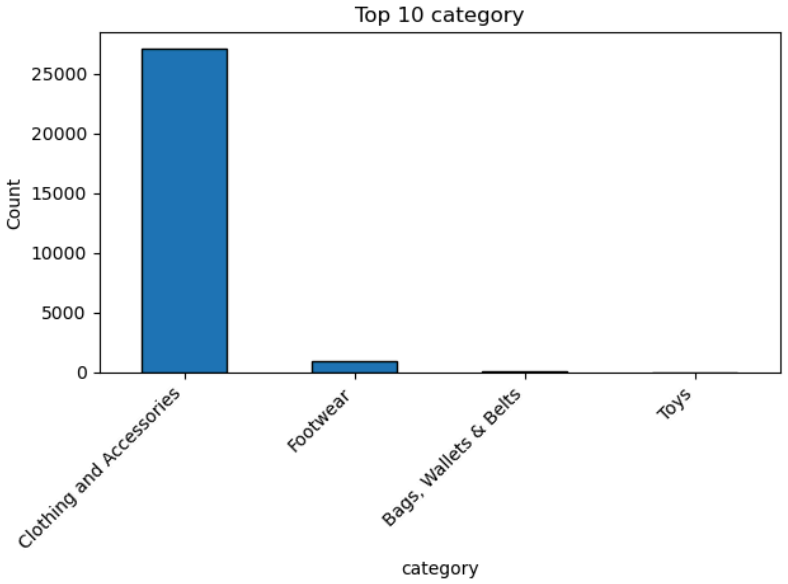
Top 5 highest rated products:

	title	average_rating	brand
27767	solid women neck blue	5.0	Oka
12332	print women hood neck black	5.0	ATTIITU
12279	print women hood neck grey	5.0	ATTIITU
23852	graphic print men neck blue	5.0	Free Authori
12235	solid women neck white black	5.0	ATTIITU

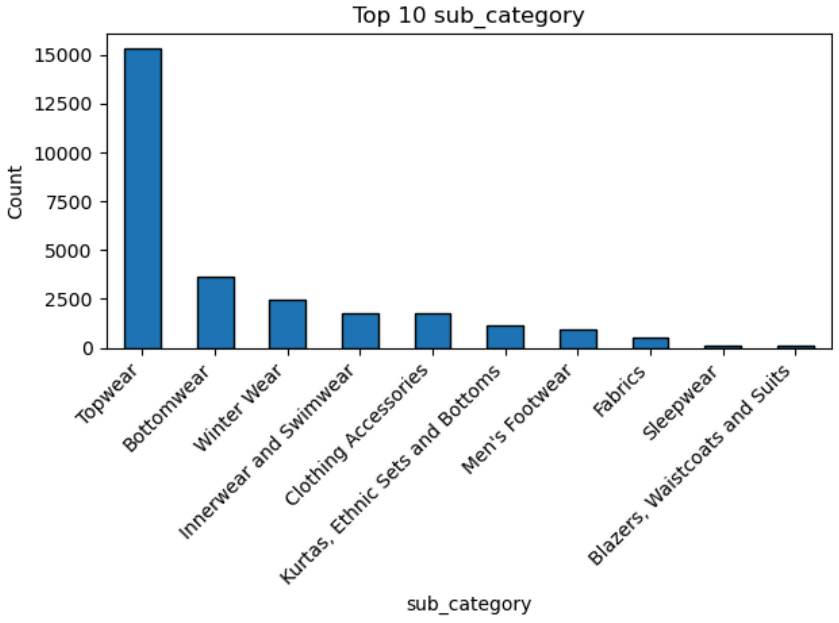
Finally, the highest rated products all reach 5.0 stars, concentrated in a few brands like ATTIITU and Oka. These cases are limited in number but show consistent satisfaction among buyers, especially for hooded sweatshirts and casual tops.

4.4. Categorical Fields

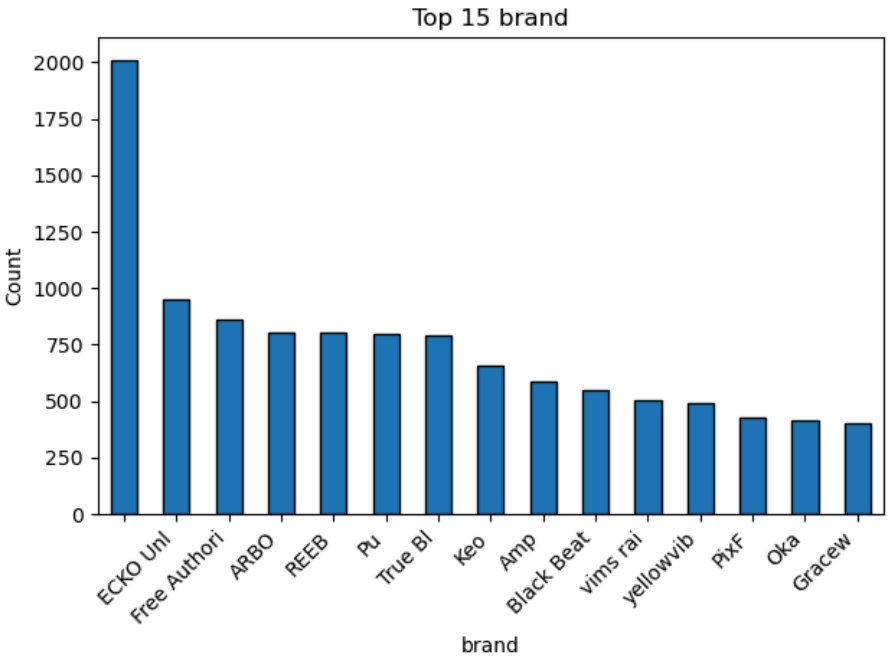
We analysed the distribution of categories, sub_categories, brands and sellers to understand corpus composition and potential biases



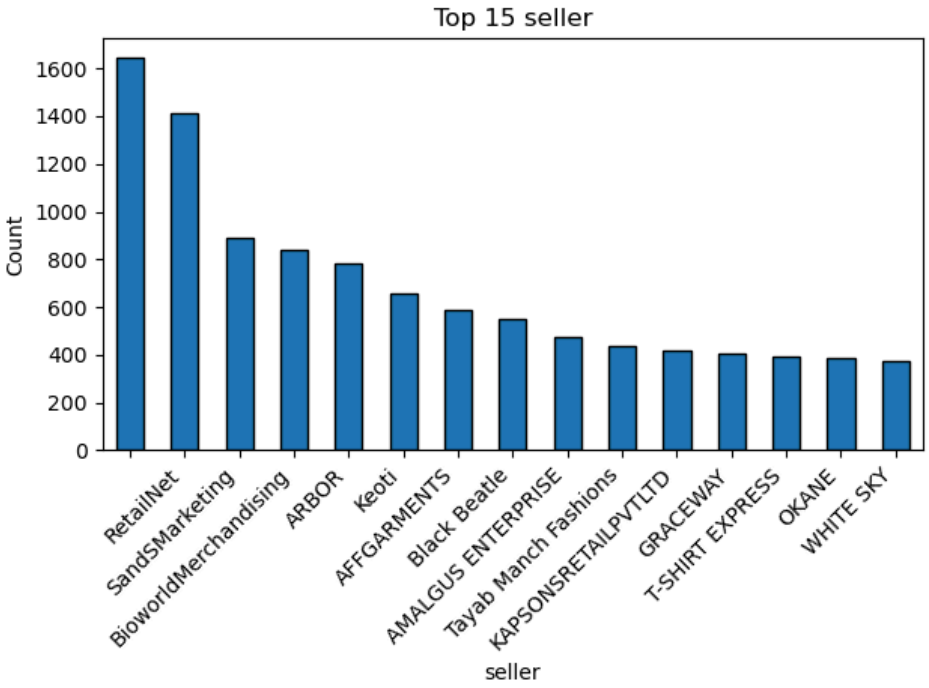
The dataset is strongly concentrated in Clothing and Accessories, with very small shares for Footwear and the rest of categories. This confirms that the corpus mainly targets apparel.



Within categories, Topwear clearly dominates, followed by Bottomwear, Winter Wear, and Innerwear and Swimwear. This skew helps explain the vocabulary observed in the textual analysis.



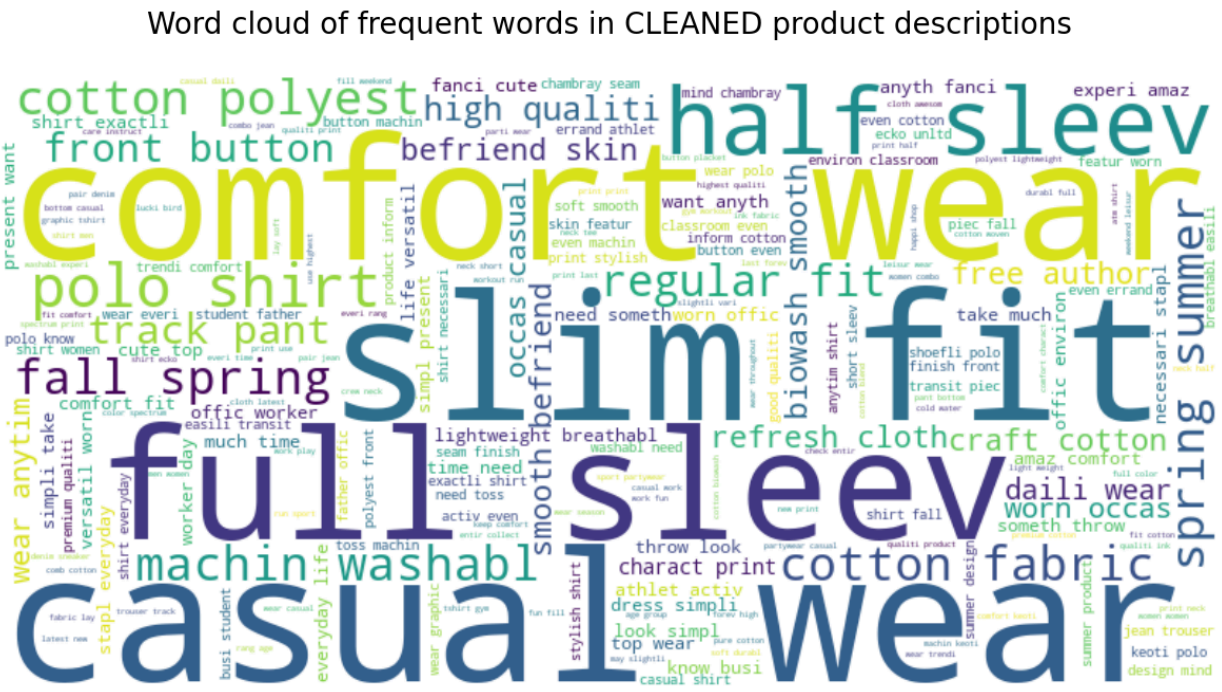
The brand distribution shows a long tail. A small set of brands concentrates most products, led by ECKO Unl, Free Authori, ARBO, REEB, Pu and True Bl.



Seller frequency follows a similar pattern. A few large vendors (e.g., Retailnet, SandsMarketing, BioworldMerchandising) account for a significant portion of the inventory, while many others have limited presence.

4.5. Word Cloud

To visualise the overall vocabulary of the corpus, we generated a word cloud from the cleaned product descriptions.



5. Statement of use of generative Artificial Intelligence

5.1. For which aspects of the practice we have used AI

Definition and understanding of the objectives of our project: First of all, we have used generative AI to guarantee a correct and complete understanding of the objectives of the practice at a technical and mathematical level. Likewise, we have been able to establish the statement through practical examples to be able to develop the practice with total guarantees at the level of understanding.

Resolving doubts about the problems that arose during the development of the practice: As we were solving the proposed statement, different problems arose at code level as execution errors, which we easily solved through VSCode debugging with the help of generative AI to detect where in our code was the error in order to solve it.

Complex code writing guided by the provided notebooks and prompt engineering techniques to adapt our base code with new modalities in order to simplify our initial implementation and improve the results based on the different evaluation methods used.

Code review and validation of our model: Once our code was finalized, we performed an exhaustive analysis of it to verify that it was correct.

5.2. To what extent has it facilitated the realization of the practice

In general, access to these new tools in a responsible and coherent way has been a great advantage when carrying out the practice, especially in terms of time, since it has facilitated different aspects such as:

Quick understanding of complex concepts, for which, in the past, it would take us much more time to find the indicated information on the internet, interpret it and adapt it to our needs.

Resolution of errors, for which, as in the previous point, in the past we could take a long time to detect and solve correctly.

Obtaining explanations about python library methods that we do not know in depth in a much faster and more efficient way than consulting the library guide.

Verification of the content of the memory to verify that we did not leave anything relevant for the total understanding of our work developed in this practice.

5.3. How did you verify and adapt the information obtained

As mentioned in the previous section, we consider the use of these new tools to be a great advantage over the past, as long as they are used responsibly and conscientiously. Therefore, we have used a cross-validation method to ensure that the information provided by the generative AI was correct and optimal. For this purpose, the following methodology was followed:

Checking on the validity of the answers through the Internet and resources provided in the referenced notebooks.

Contrasting the theoretical explanations with the Internet and referenced notebooks, especially in the mathematical part since the generative AIs are not perfected in this area.

Use of AI as a support tool and not as a substitute for critical thinking to solve the different problems posed.