Report 1

SNumbers: u264332, u264443, u264202

Names: Levente Olivér Bódi, Riccardo Zamuner, Giada Izzo

Github repository: https://github.com/Levilevi01/Info_Retrieval/tree/main

Repository TAG: **IRWA-2025-part-1**

Introduction

In this project, we explored a dataset of fashion products from an e-commerce platform. The goal was to clean and process the data so it could be used effectively for information retrieval, and to better understand its contents through some exploratory analysis. We wrote and ran our code in a Jupyter notebook, where we tried out different preprocessing steps and looked at various statistics and patterns in the data.
In this report, we'll go over what we did in the notebook, explain some of the choices we made, and highlight a few interesting findings from the analysis.

Part 1

We would like to reflect on what we did in the code.

Section 1

For Section 1, we imported the necessary libraries first, alongside the nltk stopwords package. After, we wrote 2 functions, one for preprocessing textual data, and one for preprocessing numerical data. These are combined in the preprocess_document function, which preprocesses with the help of the functions described above. In the preprocess_text function we tokenize the text, convert it to lower case, eliminate stopwords, stem it, remove punctuation, just as described in the assignment.

Section 2

The dataset contains 2 columns, namely crawled_at, and images, which the assignment does not mention in the compulsory return fields, however, for now, we keep them. The fields are read into a pandas dataframe in Part 2.

Section 3

Pros of Keeping Separate Fields:

- Maintaining separate fields allows end users to precisely filter products. If a product's category, sub-category, or brand exactly matches the user's filter criteria, it is considered relevant; otherwise, it is excluded.

- Merging all product attributes into a unified representation ensures that every aspect of a product contributes to its overall relevance. For example, if a user performs a general query (rather than a strict filter) involving brands, products from the requested brands will rank higher, while other potentially interesting products may still appear as relevant results.

Cons of Keeping Separate Fields:

- Evaluating overall product relevance across multiple independent fields may require additional computation, especially when considering general or partial relevance.

- This approach removes the ability to apply explicit filters, as all attributes are treated uniformly within the same relevance model.

A hybrid approach could also work: certain categorical fields might be encoded using one-hot representations and stored in the inverted index structure, allowing a balance between filtering and retrieval depending on how one wants to build their search engine.

Moreover, after conducting exploratory data analysis (EDA), we observed that the dataset contains relatively few distinct categories, sub-categories, and brands. Consequently, merging these fields is unlikely to significantly affect product relevance, as many products share identical values, making their inverse document frequency (IDF) contributions relatively small.

Finally, since product details consist of a mixed array of entries, we concluded that merging all these values into a single representation is the most practical approach. Similar products often share overlapping entries or values, making unified indexing more effective for relevance estimation.

Section 4

- out_of_stock: boolean. A boolean type is appropriate, as the field can take only two states: in stock or out of stock. It is also useful as a filter to not show any non available products.

- selling_price: numerical. A numerical type allows for efficient range filtering and sorting operations.

- discount: numerical. It could be argued whether to store it as a decimal (i.e. 60% = 0.6) or as a whole number and then remember to divide by 100 when calculating actual price. We choose the second option for no particular reason.

- actual_price: numerical. Same as selling_price.

- average_rating: numerical. A numerical type supports filtering and ranking within retrieval systems.

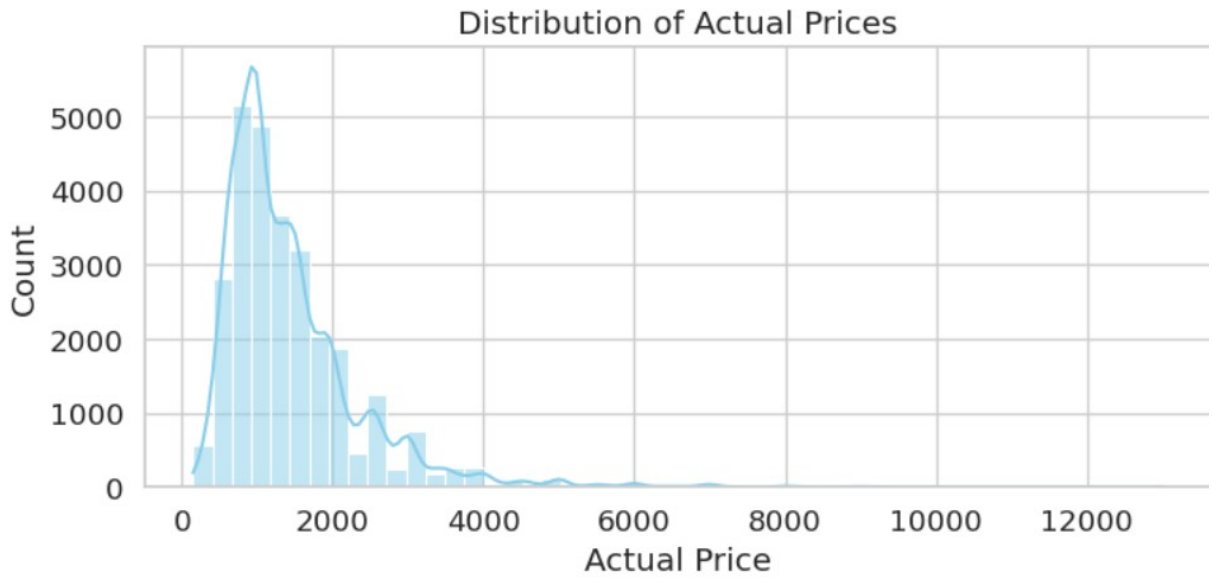
Part 2

After checking the plots and exploring the data, we found out that the dataset contains 855 fields which do not have a discount, 777 fields which do not have actual price, and 2 fields which do not have selling price. We impute 0 for the discount, because we hypothesize that if there is no data on discount, there is no actual discount existing. After that, we impute the selling prices to the missing actual price fields, since if discount = 0, then selling price = actual price. The 2 rows which do not have a selling price we drop, since price is crucial information.

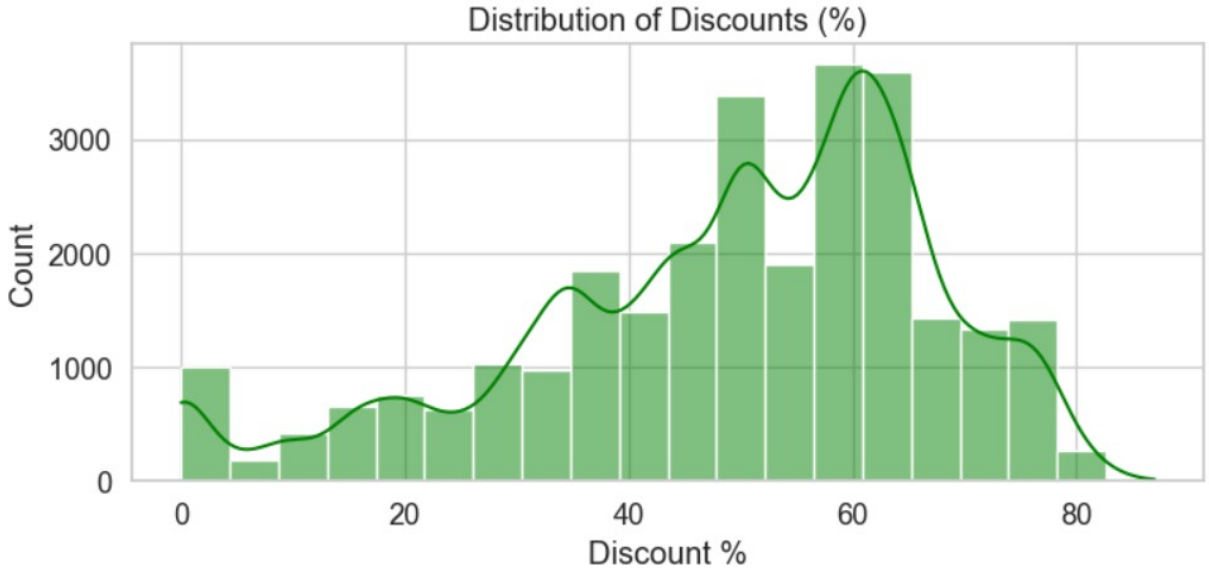
Another column which has a lot of missing records, namely 2261, is average rating. This we will leave for now, since we do not know yet what we need it for and we do not think it is sensible either to drop, or to impute it with zeroes or averages of other ratings.

We also found out that 11149 records are missing for description and 2009 missing for brand. The products without a brand field got ‘no brand’ imputed.

We also want to reflect on some distributions and plots of the data. The actual price of the data is approximately normally distributed with a mean of 1442.79.

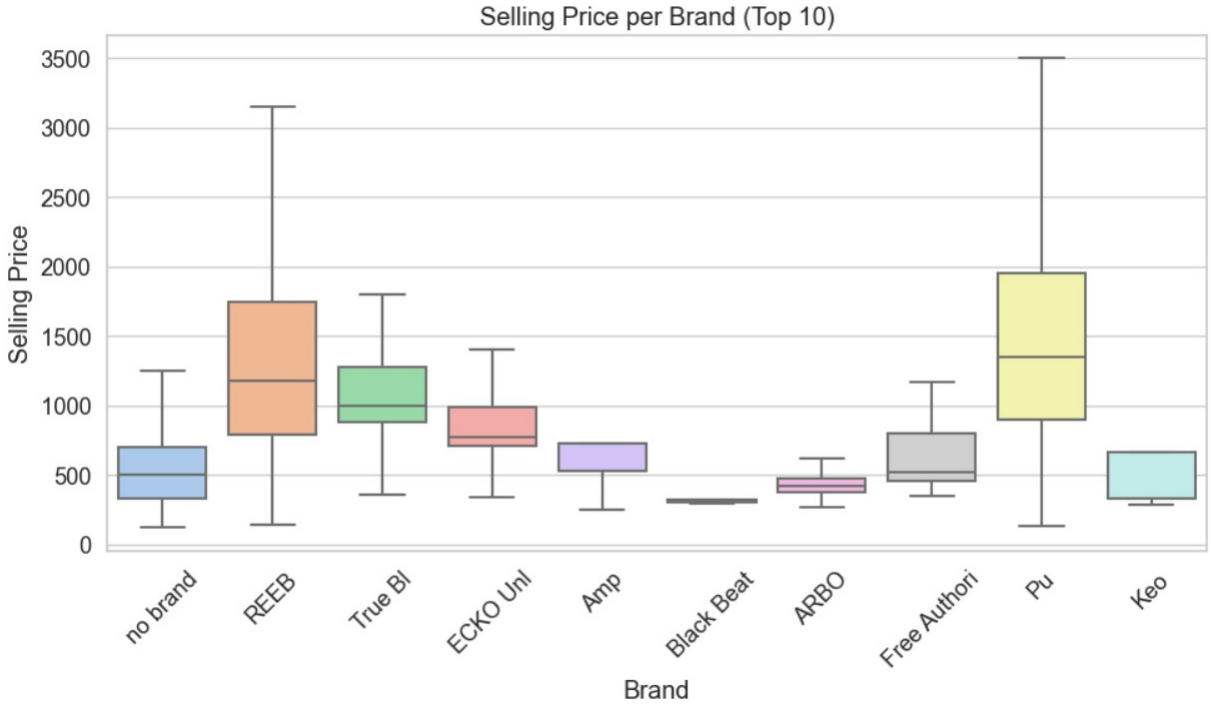


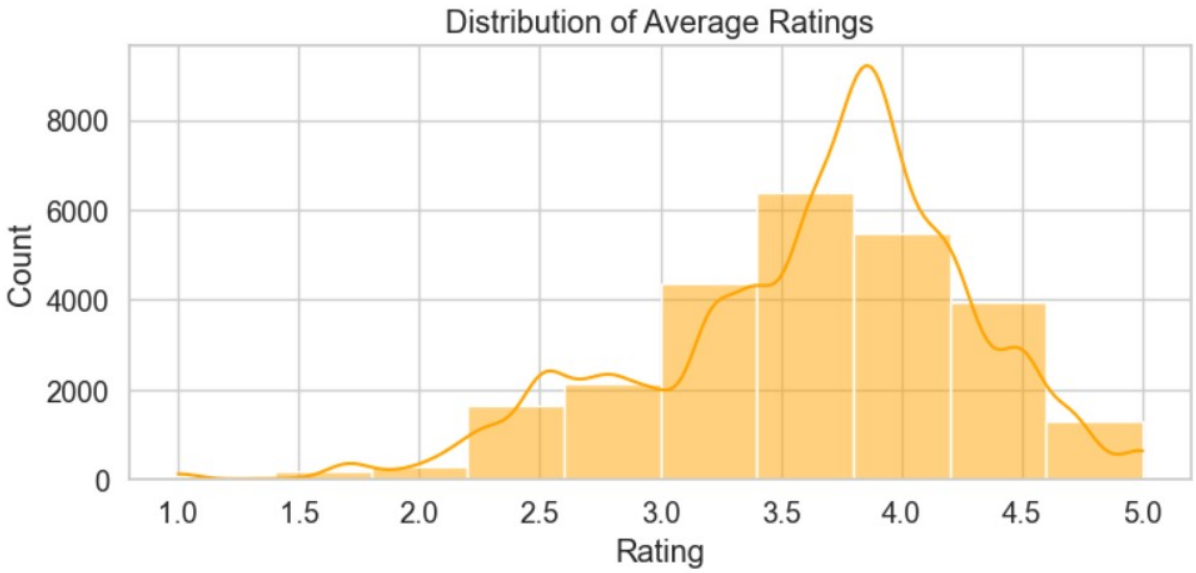
The discounts seem also more or less normally distributed, however, it has a little peak at 0, meaning no discounts, and it is a bit left skewed.



The selling price distribution is right skewed, with a mean of 705.64.

We also plotted the selling price of the top 10 brands in a boxplot, and the average rating distribution:





We also did an exploration word counting, average sentence length and vocabulary size. Besides that, we discovered the top products by specific categories, based on rating, price, and discount.

These last results can be seen in the notebook after running the code.