Final Project Part 1: Text Processing and Exploratory Data Analysis

The first step for building our RAG (Retrieval Augmented Generation) system is Text Processing and Exploratory Data Analysis (EDA).
We were provided with an e-commerce fashion dataset containing product information, which we need to prepare for later indexing and retrieval. The main objective is to clean, normalize, and structure the textual and categorical data in order to improve efficiency and effectiveness when using the future search engine.

The dataset contains a collection of fashion product records, where each product is represented as a JSON object with the following fields:

- title, description: which are the textual fields and those that contain the most information
- brand, category, sub_category, and seller: categorical fields
- discount, selling_price, actual_price, average_rating: numerical fields
- product_details: this is a list of key-value pairs containing categorical fields such as fabric, pattern, color…
- pid, url: are the identifiers which we preserve as is for reference and later evaluation

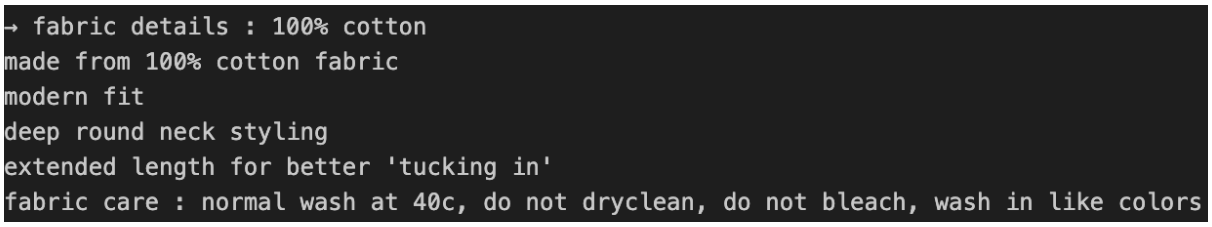
<u>Part 1: Data preparation</u>

First of all, we pre-processed the text fields: title and description. To efficiently do this task, we created two functions: **setup_preprocessing_tools()** and **preprocess_text()**.

The first one, **setup_preprocessing_tools()**, initializes a stemmer and defines the stop words. The stemmer is used to reduce each word to its root form (for example, *running → run* or *dresses → dress*), allowing words with the same semantic meaning to be treated as equivalent during search. In the stopwords list, we added and removed some that we considered good practice for our search engine.

In particular, we added *made* and *wear*, which appear 5642 and 8748 times in the description but have no added value. We also added *comfort*, *quality*, *look*, *perfect*, *style*, *good*, and *cool*. This decision was based on the fact that these tokens are typically used in marketing contexts; however, in the reality of a search engine, users rarely include such generic adjectives in their queries. We also validated this choice using the validation_labels.csv file provided. By examining the examples of queries and their relevance labels, we could observe which words contributed meaningfully to relevance and which did not.

Besides adding words to the stopwords list, we removed some that we considered important to keep: *no* and *not*. These two particular words are highly relevant in the searching process, since their presence or absence can mean the opposite (ex, “do not dryclean” vs. “do dryclean”).
In particular, we noticed that some product descriptions include washing or fabric care information, which can be relevant to users, and both words often appear in these contexts.



```
→ fabric details : 100% cotton
made from 100% cotton fabric
modern fit
deep round neck styling
extended length for better 'tucking in'
fabric care : normal wash at 40c, do not dryclean, do not bleach, wash in like colors
```

Fig 1. An example of a description where '*not'* appears and adds valuable information

The second function, **preprocess_text()**, performs cleaning and normalization of free text fields. We first lowercase the text (*title* + *description*), remove punctuation and digits, and tokenize it using NLTK's **word_tokenize()** function. Then, we remove the stopwords previously defined and apply stemming.
Additionally, we introduced a color normalization mapping to unify color variations such as *navy* → *blue* in order to improve recall for color-related search queries where different words often refer to similar tones.

To apply all these steps, we decided to combine both title and description into a single field (processed_text). Both fields contain descriptive information relevant to search, and their combination enriches the representation, ensuring that the search engine captures all descriptive terms appearing in either field.

The next step in our data preparation is the pre-processing of categorial fields: *brand*, *category*, *sub_category*, and *seller*. For these fields, we created a function **clean_categorical()**, which converts these attributes to lowercase and replaces spaces with underscores. We chose this approach instead of applying text pre-processing (stemming, tokenization, etc.) since categorical values represent labels and such transformations could distort their meaning. This simple normalization ensures consistency – for example, if the *brand* is Calvin Klein, it becomes calvin_klein.

For the numerical fields, we created another function, **clean_numeric()**. This function converts string representations of numeric values into usable numeric types (float or int). It first removes any non-numeric characters (such as percentages in the *discount* field) and then safely casts the cleaned value to the desired numerical format defined in the implementation. This function is applied to *discount*, *selling_price*, *actual_price*, and *average_rating*, and ensures that they can be correctly used for quantitative analysis and later stages such as ranking or computing statistics. This will allow us to compare products numerically based on their prices, discounts, and ratings, which could not be done if they remained as text.

Lastly, we needed to deal with *product_details*: the list of key-value pairs containing descriptive attributes such as fabric, color, pattern, or closure type. To process this field, we extracted all the string values from the list and concatenated them into a single text string, excluding *Style Code*, which is a combination of numbers and letters (i.e. '1005COMBO2') that provides no useful information for search. This was then passed through the same **preprocess_text()** function. By doing this, we ensured that all descriptive information embedded within the product details was normalized and tokenized consistently.

List of English stopwords: https://gist.github.com/sebleier/554280

Part 2: Exploratory Data Analysis

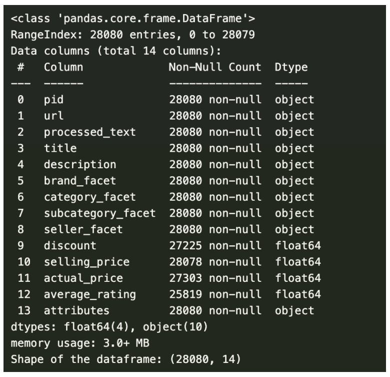
Setting Up and Understanding Data

To become familiar with the data we are working with, we need to perform Exploratory Data Analysis (EDA). To visualize data, correlations between features, etc., we installed and imported *pandas*, *matplotlib, pyplot,* and *wordcloud*.

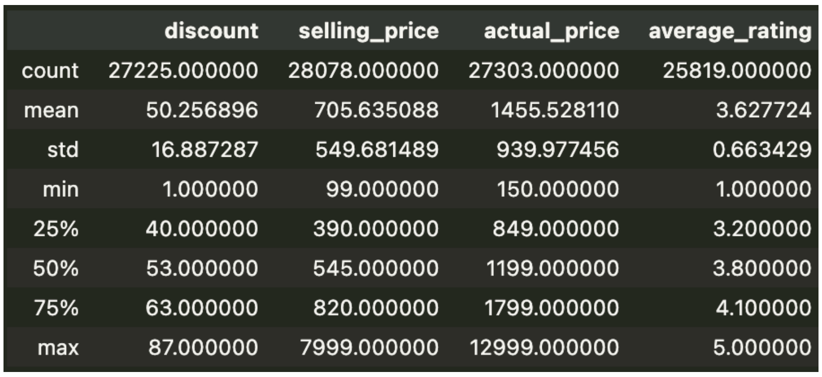
To apply these libraries, we built a dataframe of our previously processed corpus using the pandas Python library.

```
df = pd.DataFrame(processed_corpus)
```

Firstly, we need to look at the structure of the dataframe. To take a look at the structure of the dataframe, we used ***df.info()****,* ***print(df.shape),*** *and* ***df.describe()***.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28080 entries, 0 to 28079
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   pid                28080 non-null  object
 1   url                28080 non-null  object
 2   processed_text     28080 non-null  object
 3   title              28080 non-null  object
 4   description        28080 non-null  object
 5   brand_facet        28080 non-null  object
 6   category_facet     28080 non-null  object
 7   subcategory_facet  28080 non-null  object
 8   seller_facet       28080 non-null  object
 9   discount           27225 non-null  float64
 10  selling_price      28078 non-null  float64
 11  actual_price       27303 non-null  float64
 12  average_rating     25819 non-null  float64
 13  attributes         28080 non-null  object
dtypes: float64(4), object(10)
memory usage: 3.0+ MB
Shape of the dataframe: (28080, 14)
```



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

Fig 2. Output of *df.info()*, *print(df.shape)* and *df.describe()*

We can see how there are fields that describe the products, and the characteristics of each one of them.

- Firstly, we can say about shape and/or size:
    - It has **28,080 rows**, where each row represents a product listing (meaning, one product sold by a seller).
    - It has **14 columns**, representing different attributes, or features, of each listing.
    - The data occupies **around 3 MB**.

- We can also see the different **data types** and data **completeness** of the features:
    - There are **4 columns** that are numeric (float64): *discount, selling_price, actual_price* and *average_rating*
        - Precisely, the numeric data is the one that has some null cells. Present in:
            - ***discount*****:** There are 855 missing entries. Which probably means they do not have any discount applied.
            - ***selling_price*****:** Almost complete, missing only 2 entries. This might be an error in scraping.
            - ***actual_price*****:** This field has around 750 entries missing. This may be associated with an item without a discount, since the number of missing values is similar to the one in *discount*. Maybe the website typically only shows the selling price when

        no discount is applied (since then, actual_price is the same as selling_price).
    - ***average_rating:*** A lot of missing entries (around 2000). This might be simply because no one has rated those products.
  - And **10 columns** that are objects: *pid, url, processed_text, title, description, brand_facet, category_facet, subcategory_facet, seller_facet,* and *attributes*.

- Lastly, we can see the statistics associated with numerical fields.
  - **discount**
    - **Mean = 50.26%** which suggests that discounts are **commonly used** and not only in low percentages.
  - **selling_price**
    - **Mean = 705.63** and **median = 545.00,** but with a range = (99–7999), which indicates most products are low or mid price.
  - **actual_price**
    - **Mean = 1455.53** and **range = (150–12999),** which is higher than the original prices. This confirms that most listings have a discount.
  - **average_rating**
    - **Mean = 3.63,** and by the range, we know it is out of 5. Moreover, ratings cluster between 3 and 4 stars, showing that overall, the customer experience is positive.

Overview of Most Important Products, Brands, and Sellers

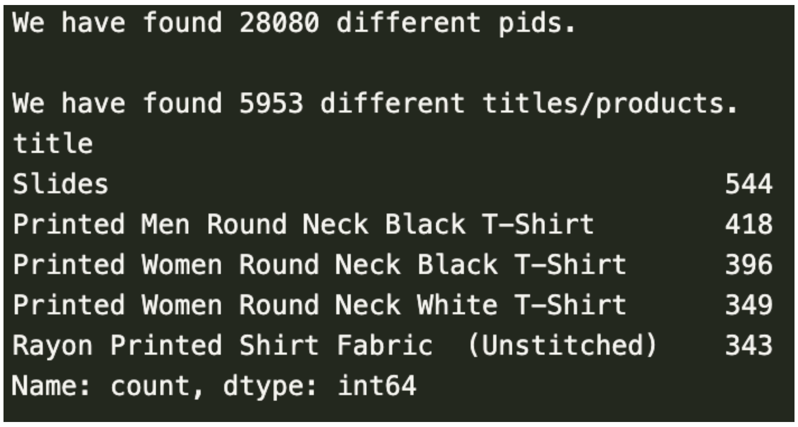
To verify the dataset structure and start exploring popularity among sellers, brands, and products, we analyzed repetitions across pids, titles, brands, and sellers.

```python
# Look for pid repetitions
print('We have found', df['pid'].nunique(), 'different pids. \n')

# Look for title repetitions
print('We have found', df['title'].nunique(), 'different titles/products.')
title_counts = df['title'].value_counts()
print(title_counts.head(), '\n')
```

Fig 3. Code looking for pid and title repetitions

The objective of the code above is to check whether each product listing is unique and to start to see the variety among products. The **pid quick check** helps verify this, although it does **not guarantee uniqueness** completely.



```
We have found 28080 different pids.

We have found 5953 different titles/products.
title
Slides                                      544
Printed Men Round Neck Black T-Shirt        418
Printed Women Round Neck Black T-Shirt      396
Printed Women Round Neck White T-Shirt      349
Rayon Printed Shirt Fabric  (Unstitched)    343
Name: count, dtype: int64
```

Fig 4. Output of fig. 4

The dataset contains 5,953 unique product titles among 28,080 listings, suggesting that many items are repeated under the same name. This likely reflects the same product being sold by different sellers or with different discounts, etc.

We did something similar for brands and sellers, and we also printed a histogram to see the difference between the top brands and sellers. It is important to highlight that from the 28,080 listings, we found 322 different brands and 535 different sellers.

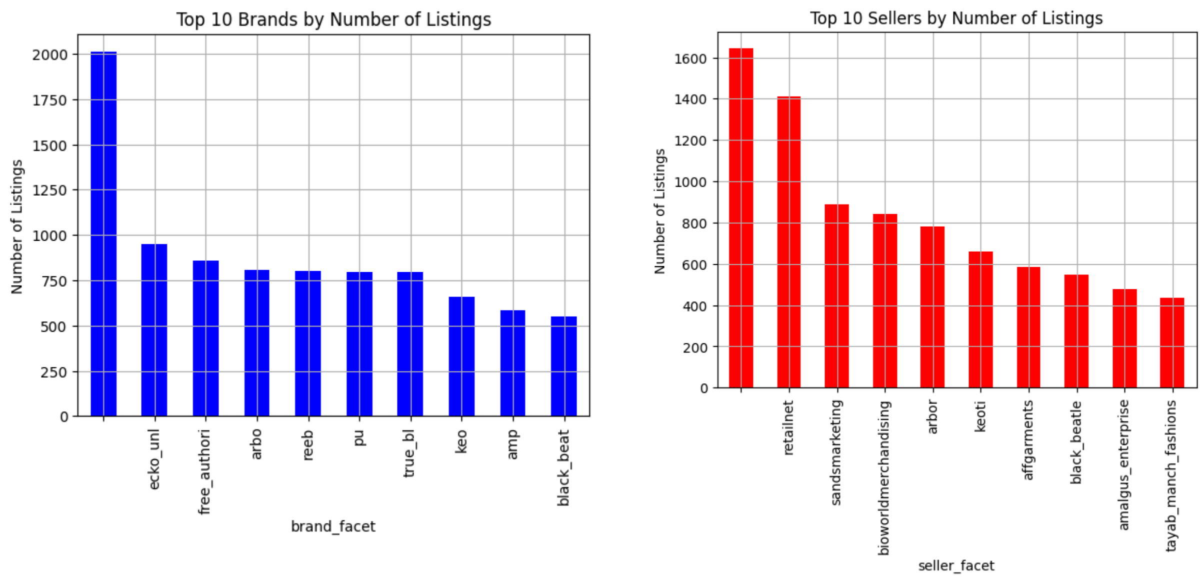


Fig 5. Output of fig. 4

We included some additional statistics as well: the top 10 best-rated products (on average), the top 10 cheapest products, and the top 10 with the largest discount percentage.

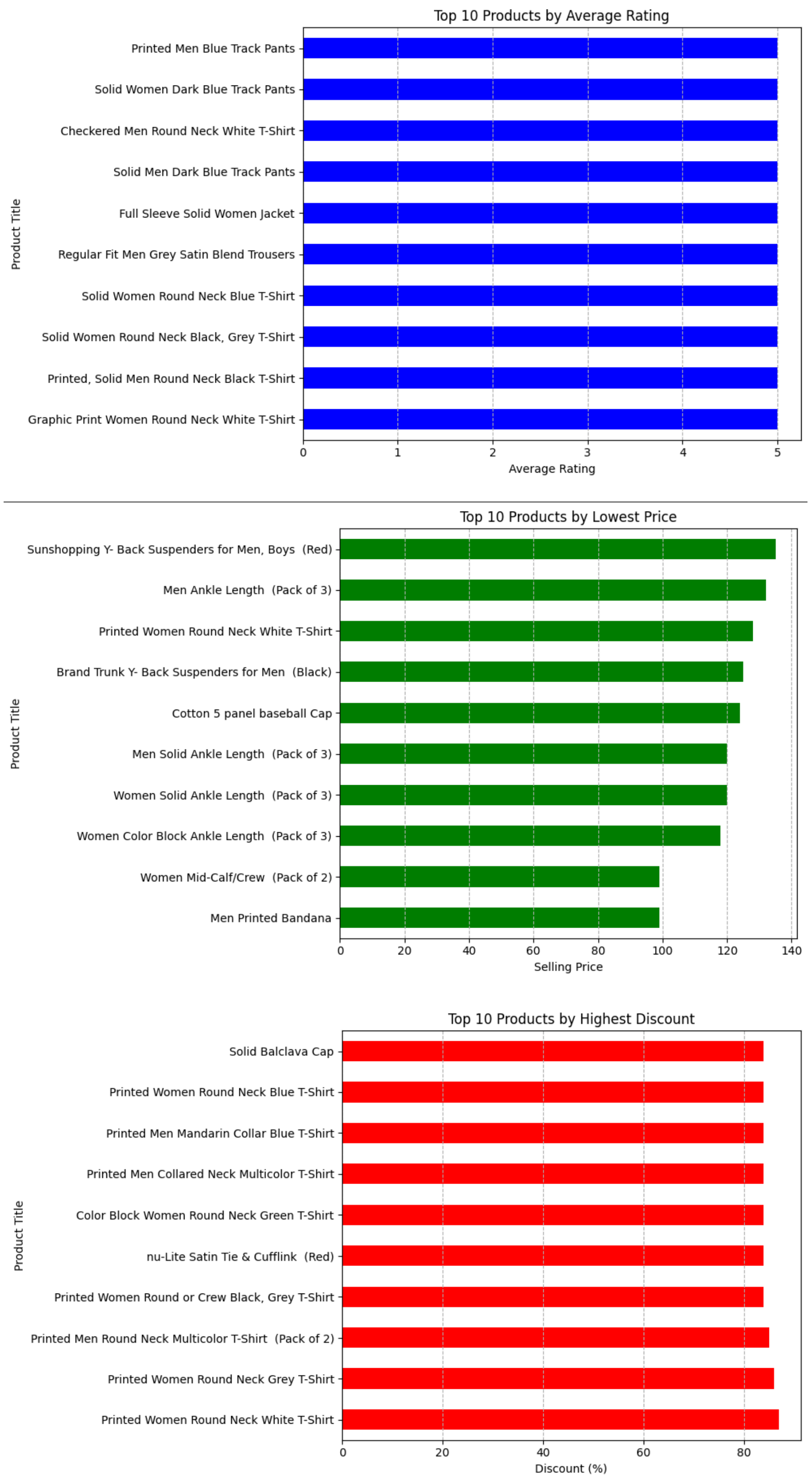


Fig 6. Product statistics

Analysis of Numerical Columns

In this section, we wanted to see the distribution of the numerical columns.

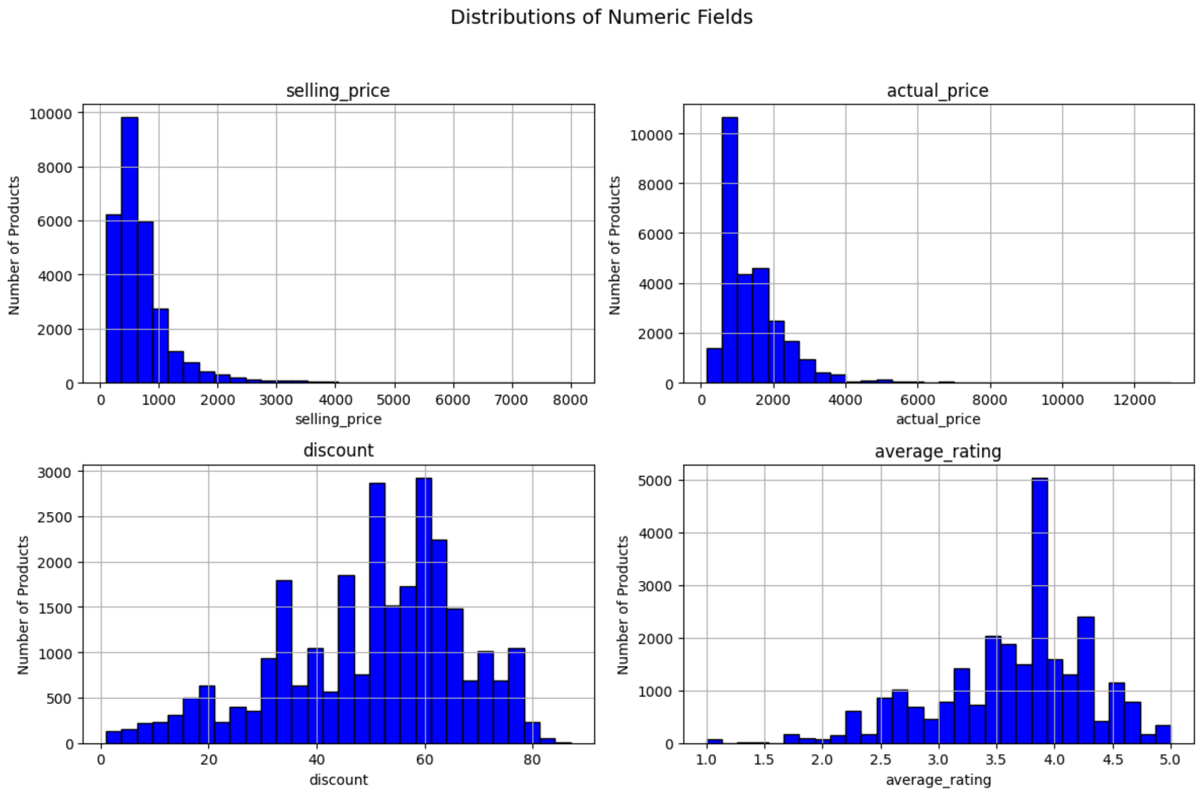


Fig 7. Distributions of Numeric Fields

Both *selling_price* and *actual_price* are strongly right-skewed: most products are at lower prices. The *discount* variable has peaks around 40–70 %, meaning sellers usually use standard promotional tiers such as 40 %, 50 %, or 60 % off rather than random percentages in the middle. Finally, the *average_rating* distribution is moderately left-skewed, concentrated between 3.5 and 4.5 stars, showing that customers generally rate products positively. Low ratings are rare.

When displaying this data some questions about discount distribution and where discounts were usually applied arose. So we displayed the following scatter graph:

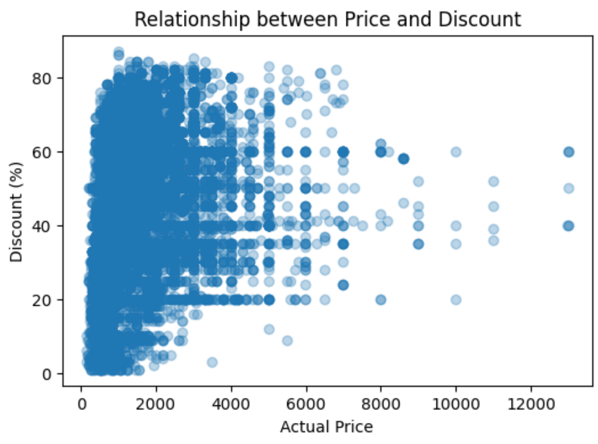


Fig 8. Relationship between Price and Discount

The scatter plot between *actual_price* and *discount* shows us that discount percentages are mostly independent of the product price. There’s no clear upward or downward trend, meaning neither expensive nor cheaper items are necessarily more discounted than the other.

However, the plot has a higher density of points in the lower to mid price products. This coincides with the fact that they have more presence in the dataframe and explain why most discounts appear in that price segment.At higher prices, the number of listings with discount drops, but the discount percentages remain within similar bounds.

Text Statistics

We analyzed the **word count** distribution to evaluate the length and variability of product *processed_text*. This provides insights into how much textual information each listing contains and helps identify whether they are any outliers or missing values that could affect analysis in the future.

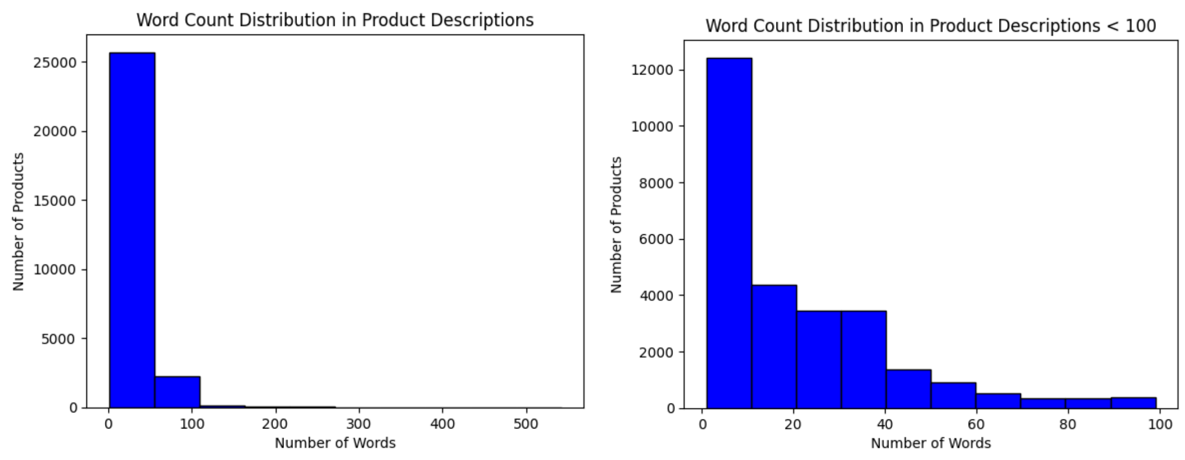


Fig 9. Relationship between Price and Discount

The word count distribution shows that the majority of product descriptions are brief, with only a few very long descriptions. The maximum value had 542 words but the average words per product was 23.22. Overall, the descriptions are short, which is a style typical of online product catalogs.

The dataset contains 4,633 unique words, showing richness and variety in the vocabulary for such short descriptions. To better understand which words appear most frequently within this vocabulary, a word cloud was generated to visually highlight the most common terms used across product descriptions.

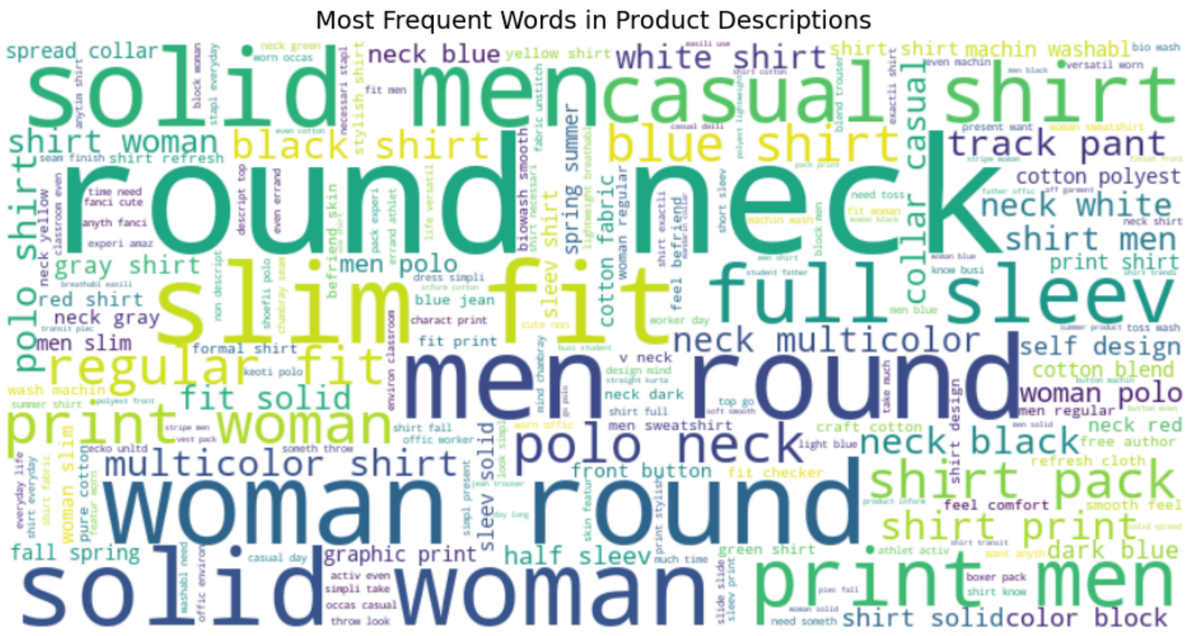


Fig 10. Word Cloud on Most Frequent Words in Product Descriptions

The word cloud shows that the most frequent words are “round”, “neck”, “solid”, “men”, “woman” and “shirt”. Which is under the expectation of fashion products’ descriptions. Other recurrent words related to style. Although not much colors appear (which we expected to encounter) some types of description about color or prints do appear as “color block” or “graphic print”.