IRWA Final Project - Part 1 Report

Text Processing & Exploratory Data Analysis

Course: Information Retrieval and Web Analytics (IRWA)
Authors: Lucas Andreu
Date: October 21, 2025
Institution: Universitat Pompeu Fabra (UPF)

Table of Contents

---

1. Executive Summary

This report documents the implementation decisions, assumptions, and methodology for **Part 1** of the IRWA Final Project, focusing on text preprocessing and exploratory data analysis of a fashion products dataset.

Key accomplishments: - Implemented a comprehensive text preprocessing pipeline using NLTK - Processed and enriched **28,080 fashion product records** - Generated statistical insights and 11 visualizations - Created a searchable corpus with normalized metadata

Dataset processed: Fashion products from e-commerce platforms
Total records: 28,080 products
Vocabulary size: 6,650 unique terms (after preprocessing)
Median price: 545 | **Median rating:** 3.8/5 | **Median discount:** 53%

---

2. Introduction

2.1 Project Context

Information retrieval systems require careful preprocessing of textual data to enable effective search and ranking. This project implements a complete pipeline for processing fashion e-commerce product data,

preparing it for subsequent ranking and retrieval tasks in Parts 2 and 3.

2.2 Objectives

1. **Text Preprocessing:** Clean and normalize product titles and descriptions
2. **Feature Extraction:** Parse and structure non-text fields (prices, ratings, categories)
3. **Exploratory Analysis:** Understand dataset characteristics and distributions
4. **Corpus Preparation:** Create a searchable index-ready dataset

---

3. Dataset Overview

3.1 Data Source

The dataset contains fashion product information scraped from e-commerce platforms, including: - Product titles and descriptions - Pricing information (actual price, selling price, discounts) - Ratings and reviews metadata - Categorical information (brand, category, sub-category) - Product details and specifications - Seller information

3.2 Data Structure

Input file: `fashion_products_dataset.json`
Format: JSON array of product objects
Key fields: - `pid`: Product identifier - `title`: Product name/title - `description`: Detailed product description - `brand`, `category`, `sub_category`: Categorical attributes - `actual_price`, `selling_price`, `discount`: Pricing information - `average_rating`: Customer rating - `product_details`: Structured product specifications - `url`: Product page link

---

4. Part 1: Text Preprocessing Pipeline

4.1 Design Decisions

4.1.1 Choice of NLTK **Decision:** Use NLTK (Natural Language Toolkit) for text processing.

Rationale: - **Maturity:** NLTK is a well-established library with extensive documentation - **Functionality:** Provides all required components (tokenization, stopwords, stemming) - **Educational value:** Industry-standard tool for NLP tasks - **Python integration:** Native Python library with no external dependencies beyond the library itself

Alternatives considered: - **spaCy:** More modern but heavier; overkill for basic preprocessing - **Custom regex:** Less reliable for tokenization edge cases - **TextBlob:** Simplified API but less control over individual steps

4.1.2 Preprocessing Steps Order **Decision:** Apply preprocessing in the following sequence:

```
Text normalization → Tokenization → Stopword removal → Stemming
```

Rationale: 1. **Normalization first:** Clean text before tokenization to avoid spurious tokens 2. **Tokenization:** Split into words after normalization ensures consistent splitting 3. **Stopword removal:** Remove common words before stemming (more efficient) 4. **Stemming last:** Apply to meaningful words only (after stopword removal)

4.2 Implementation Details

4.2.1 Text Normalization **Function:** `normalize_basic(text)`

Operations performed: 1. **Unicode normalization:** `unidecode()` to remove accents (é → e, ñ → n) 2. **Lowercase conversion:** Ensures case-insensitive matching 3. **URL removal:** Regex pattern `http[s]?://\S+` removes web links 4. **Special character handling:** - Dashes/underscores/slashes → spaces (brand-name → brand name) - Currency symbols removed (, $, €, £, ¥, %) 5. **Number removal:** All digits stripped (sizes, model numbers) 6. **Punctuation removal:** Only alphanumeric and spaces retained 7. **Whitespace normalization:** Multiple spaces collapsed to single space

Design choices: - **Remove numbers:** Product-specific codes (sizes, SKUs) don't help general search - **Keep spaces from special chars:** "red-shirt" → "red shirt" (two searchable terms) - **Aggressive cleaning:** Prioritize recall over preserving technical terms

4.2.2 Tokenization **Function:** `tokenize(text)`

Implementation: `nltk.tokenize.word_tokenize()`

Why word_tokenize over split()? - Handles contractions properly (don't → do, n't) - Splits punctuation correctly - Language-aware (uses Punkt sentence tokenizer internally)

Example:

```
Input:  "Women's cotton t-shirt, size M"
After normalize: "womens cotton t shirt size m"
After tokenize: ["womens", "cotton", "t", "shirt", "size", "m"]
```

4.2.3 Stopword Removal **Function:** `remove_stopwords(tokens)`

Implementation: Uses NLTK's English stopwords corpus

Stopwords removed: Common words like "the", "is", "at", "which", "on", etc.

Why remove stopwords? - **Reduce noise:** High-frequency words don't distinguish documents - **Improve TF-IDF:** Better discrimination in later ranking - **Reduce index size:** Fewer terms to store and process

Potential issue: Some domain-specific stopwords might be relevant (e.g., "long" in fashion) **Mitigation:** Standard stopword list is conservative; domain terms typically preserved

4.2.4 Stemming **Function:** `stem(tokens)`

Implementation: Porter Stemmer from NLTK

Why Porter Stemmer? - **Classic algorithm:** Well-understood behavior - **Fast:** Rule-based, no dictionary lookups - **Suitable for IR:** Aggressive stemming improves recall

Alternatives considered: - **Snowball Stemmer:** Slightly more aggressive, similar results - **Lemmatization (WordNet):** More accurate but slower; requires POS tagging - **No stemming:** Would reduce recall (e.g., "running shoes" vs "run shoes")

Trade-off: Stemming can over-conflate (e.g., "university" → "univers", "universe" → "univers") **Acceptable because:** In product search, recall is prioritized over precision

Examples:

```
running → run
shoes → shoe
dresses → dress
beautiful → beauti
```

4.3 Assumptions Made

4.3.1 Language Assumption **Assumption:** All text is in English.

Justification: - Dataset appears to be from English-language e-commerce sites - NLTK stopwords and stemmer are English-specific - No language detection performed

Risk: Non-English text would be poorly processed
Mitigation: Could add language detection (e.g., `langdetect`) if needed

4.3.2 Text Quality Assumption **Assumption:** Text is reasonably well-formed (no excessive typos or encoding errors).

Justification: - E-commerce platforms typically have quality control - Sellers are incentivized to write clear descriptions - `unidecode` handles most encoding issues

Observed issues: - Some HTML entities in descriptions (handled by normalization) - Occasional special characters (removed in cleaning)

4.3.3 Metadata Assumptions **Assumption:** Title and description are the primary text fields for search.

Justification: - Most product searches match against these fields - Other fields (brand, category) treated as faceted search filters - Product details contain structured specs, not free text

4.3.4 No Phrase Preservation **Assumption:** Individual tokens are sufficient; no phrase detection needed.

Justification: - Simplifies initial implementation - Bigrams/trigrams can be added in Part 2 if needed - Single-term search is common in e-commerce

Trade-off: Lose context for phrases like "running shoe" vs "shoe for running"

---

5. Part 2: Non-Text Fields Processing

5.1 Categorical Normalization

5.1.1 Design Decision: Normalize to Lowercase **Function:** `_norm_str(val)`

Operations: - Lowercase conversion - Accent removal via `unidecode` - Whitespace trimming - Empty string handling

Rationale: - **Consistency:** "Nike" and "NIKE" should be the same facet - **Exact matching:** Enables efficient filtering (no case-insensitive comparisons) - **Deduplication:** Reduces duplicate facet values

Fields normalized: - `brand_norm` - `category_norm` - `sub_category_norm` - `seller_norm`

5.1.2 Product Details Flattening **Function:** `flatten_product_details(pd)`

Challenge: Product details are stored as list of dicts with inconsistent structure.

Example input:

```
[
  {"Fabric": "Cotton"},
  {"Sleeve Length": "Full Sleeve"},
  {"Pattern": "Solid"}
]
```

Solution: 1. Iterate over list, extract all key-value pairs 2. Normalize keys and values to lowercase 3. Create both: - **Dict representation:** For structured access (`{fabric: cotton, ...}`) - **Text representation:** For search (`"fabric: cotton; sleeve length: full sleeve"`)

Output fields: - `product_details_map`: Dict for filtering - `product_details_text`: String for text search

Rationale: - **Dual format:** Supports both faceted filtering and free-text search - **Searchable specs:** Users can search "cotton full sleeve"

5.2 Numeric Parsing

5.2.1 Price Parsing Function: `_to_float_price(val)`

Challenges: - Prices stored as strings: `" 2,999"`, `"$19.99"` - Various currency symbols - Comma separators

Solution: 1. Remove commas 2. Extract numeric pattern with regex: `(\d+(\.\d+)?)` 3. Convert to float

Assumptions: - First number found is the price - Decimal separator is period (.) - All prices in same currency (comparison valid)

Fields parsed: - `actual_price_num`: Original price - `selling_price_num`: Discounted price

5.2.2 Discount Parsing Function: `_to_int_discount_percent(val)`

Input formats: `"69% off"`, `"15%"`, `"discount: 25"`

Solution: Regex `(\d+)\s*%?` extracts numeric percentage

Assumptions: - Percentage is the discount (not markup) - Values range 0-100

Field: `discount_pct`

5.2.3 Rating Parsing Function: `_to_float_rating(val)`

Simple conversion: `float(str(val).strip())`

Assumptions: - Ratings are 0-5 scale (standard) - Stored as numeric strings

Field: `average_rating_num`

5.2.4 Boolean Parsing Function: `_to_bool(val)`

Handles: `true/false`, `yes/no`, `1/0`, Boolean types

Field: `out_of_stock_bool`

5.3 Feature Engineering

5.3.1 Price Buckets Function: `bucket_price(x)`

Buckets: - `low`: $<$ 1000 - `mid`: 1000 - 2999 - `high`: 3000

Rationale: - **UX:** Simplified price filtering ("Show me affordable options") - **Boosting:** Different budget segments can be weighted - **Analysis:** Price tier distributions

Assumption: Bucket thresholds are reasonable for the market (India pricing observed)

5.3.2 Discount Buckets Function: `bucket_discount(p)`

Buckets: - `0`: No discount - `1-20`: Small discount - `21-40`: Medium discount - `41+`: High discount

Rationale: - **Promotional filtering:** "Show high-discount items" - **Ranking feature:** Discount tier as signal

5.3.3 Rating Buckets Function: `bucket_rating(r)`

Buckets: - `0-2`: Poor - `2-3.5`: Below average - `3.5-4.5`: Good - `4.5-5`: Excellent

Rationale: - **Quality filtering:** "Only show well-rated products" - **Common UX pattern:** Star ratings in ranges

5.3.4 Metadata Text Field

Function: `enrich_record()` creates `metadata_clean`

Combines: - Brand - Category - Sub-category - Seller - Product details text

Rationale: - **Recall boost:** User searches "Nike running shoes" matches brand + category - **Single field:** Simplifies indexing (one field for all metadata) - **Preprocessed:** Stemmed and cleaned like title/description

Example:

```
Original: brand="Nike", category="Footwear", sub_category="Running Shoes"
metadata_clean: "nike footwear run shoe"
```

---

6. Part 3: Exploratory Data Analysis

6.1 Statistical Analysis

6.1.1 Vocabulary Size

Metric: Unique tokens across title, description, and metadata fields

Purpose: - Understand corpus diversity - Estimate index size - Assess term sparsity

Calculation:

```
all_text = " ".join(df[col].tolist() for col in text_sources)
vocab_size = len(set(all_text.split()))
```

Insight: Larger vocabulary → more diverse products, but also more sparse term frequencies

6.1.2 Document Length Distributions

Metrics: - Title word count (after preprocessing) - Description word count (after preprocessing)

Visualizations: Histograms showing distribution

Purpose: - Identify outliers (very short/long texts) - Understand typical document length for ranking algorithms - Detect data quality issues

Insights from our data: - **Titles:** Median of 6 words after preprocessing (product names are concise) - **Descriptions:** Median of 9 words after preprocessing - **Distribution:** See visualizations in `data/eda_outputs/`: - `hist_title_word_count.png` - Shows title length distribution - `hist_desc_word_count.png` - Shows description length distribution

6.1.3 Missing Data Analysis

Metric: Count of null/empty values per field

Purpose: - Identify data quality issues - Decide on handling strategies (imputation vs. exclusion)

Key findings from our dataset: - **2,261 products (8.05%)** lack ratings - likely new items without customer reviews - **Pricing almost complete:** Only 2 products (0.007%) missing selling price - **Brand information:** 2,009 products (7.15%) missing brand data - **Seller information:** 1,643 products (5.85%) missing seller data - **Discount data:** 855 products (3.04%) without discount information - **Perfect categorical coverage:** All products have category and sub-category

See full missing value analysis in `data/eda_outputs/summary.txt`

6.2 Visualizations

6.2.1 Word Clouds

Generated for: 1. Titles (`wc_title.png`) 2. Descriptions (`wc_description.png`) 3. Metadata (`wc_metadata.png`)

Purpose: - Visual understanding of most frequent terms - Identify domain vocabulary - Spot unexpected terms (data issues)

Insights from generated word clouds: - Fashion-specific terms dominate (shirt, dress, cotton, sleeve, fabric) - Brand names prominent in metadata cloud - Product attributes highly visible (size, color, style, pattern) - Generic terms still appear despite stopword removal (design, style)

Word clouds available in `data/eda_outputs/`: - `wc_title.png` - Most frequent terms in product titles - `wc_description.png` - Most frequent terms in descriptions - `wc_metadata.png` - Most frequent brand, category, and specification terms

6.2.2 Distribution Histograms Plots generated (see `data/eda_outputs/`):

1. **Price distribution** (`hist_prices.png`)
   - **Median price:** 545
   - Most products in 300- 1500 range
   - Long tail of expensive items (up to 20,000+)
   - Right-skewed distribution (typical for e-commerce)
   - **Insight:** Majority are affordable/mid-range fashion items
2. **Rating distribution** (`hist_ratings.png`)
   - **Median rating:** 3.8/5 stars
   - Distribution centered around 3.5-4.5 stars
   - Relatively few 1-2 star products (possible survivorship bias)
   - 8.05% of products have no ratings yet
   - **Insight:** Most products are well-received by customers
3. **Discount distribution** (`hist_discounts.png`)
   - **Median discount:** 53%
   - Most products have 40-70% discount
   - Peak around 50-60% (aggressive promotional pricing)
   - Only 3.04% of products lack discount data
   - **Insight:** Highly promotional market; discounts are the norm

6.2.3 Categorical Bar Charts Plots generated (see `data/eda_outputs/`):

1. **Top brands** (`bar_top_brands.png`)
   - Identifies the 15 most represented brands in the catalog
   - Shows brand distribution (some brands have significantly more products)
   - Useful for understanding catalog composition
   - **Application:** Potential brand-based boosting or diversification strategies
2. **Top sellers** (`bar_top_sellers.png`)
   - Shows the 15 most active sellers
   - Reveals concentration (do few sellers dominate?)
   - **Application:** Quality signal (established sellers might indicate reliability)
3. **Out-of-stock distribution** (`bar_out_of_stock.png`)
   - **5.85% out of stock** vs 94.15% available
   - Inventory availability is generally good
   - **Application:** Decide whether to hide out-of-stock items or just penalize in ranking

6.2.4 Product Rankings & Tables Generated CSV tables (see `data/eda_outputs/`):

1. **Top-rated products** (`top_rated_products.csv`)
   - Top 50 products by average rating (sorted by rating, then price)
   - Columns: pid, title, brand, category, sub_category, average_rating_num, selling_price_num, discount_pct, url
   - **Use case:** Feature high-quality products; understand what makes products well-rated
2. **Cheapest products** (`cheapest_products.csv`)
   - Top 50 cheapest products by selling price
   - Helps understand budget segment of catalog

- **Use case:** Budget-conscious search filters; "Under 500" categories

3. **Top discounted products** (`top_discounted_products.csv`)
   - Top 50 products by discount percentage
   - Identifies best deals in the catalog
   - **Use case:** "Best deals" section; promotional campaigns

6.3 Key Insights

6.3.1 Corpus Characteristics From our KPI analysis (`data/eda_outputs/kpi_summary.txt`):

- **Size:** 28,080 fashion products (substantial corpus for indexing)
- **Vocabulary:** 6,650 unique preprocessed terms across title, description, and metadata
- **Text length:**
  - Median title: 6 words (after preprocessing)
  - Median description: 9 words (after preprocessing)
- **Pricing:**
  - Median selling price: 545
  - Only 2 products missing price information (99.99% complete)
- **Ratings:**
  - Median rating: 3.8/5 stars
  - 2,261 products (8.05%) without ratings (likely new items)
- **Discounts:**
  - Median discount: 53% (highly promotional dataset)
  - 855 products (3.04%) with no discount information
- **Stock availability:**
  - 5.85% out of stock (good availability overall)
- **Metadata completeness:**
  - 7.15% missing brand information
  - 5.85% missing seller information
  - 100% have category and sub-category data
- **Diversity:** Multiple categories (clothing, footwear, accessories)
- **Text quality:** Generally good, some variation in description detail

6.3.2 Search Implications From our EDA findings:

1. **Short text fields:** Median title (6 words) and description (9 words) are concise
   - **Implication:** Multi-field search (title + description + metadata) essential to maximize recall
   - **Action:** Weight title higher but include all text fields in search
2. **High discount prevalence:** Median 53% discount across catalog
   - **Implication:** "Best deal" ranking (high discount + good rating) is valuable
   - **Action:** Consider discount as a ranking signal; filter by discount buckets
3. **Rating quality:** Median 3.8/5 stars, but 8% have no ratings
   - **Implication:** Rating-based filtering (e.g., 4+ stars) would exclude 2,261 products
   - **Action:** Use rating as boost signal rather than hard filter; handle missing ratings gracefully
4. **Pricing diversity:** Median 545, but wide range (100 to 20,000+)
   - **Implication:** Price buckets (low/mid/high) useful for filtering
   - **Action:** Enable price range filters; consider budget-aware ranking
5. **Stock availability:** 94.15% in stock
   - **Implication:** Out-of-stock items are minority
   - **Action:** Filter by default or apply ranking penalty rather than exclude
6. **Brand/seller data:** 7-8% missing brand/seller info
   - **Implication:** Cannot rely on brand as universal filter
   - **Action:** Treat as optional facet; handle missing values in metadata search

6.3.3 Ranking Considerations **Features for Part 2 ranking:** - **Text relevance:** BM25/TF-IDF scores from preprocessed text - **Rating signal:** `average_rating_num` as quality indicator - **Price signal:** `selling_price_num` for budget-conscious users - **Discount signal:** `discount_pct` for deal-seekers - **Availability:** `out_of_stock_bool` as filter (or ranking penalty)

6.4 Visual Analysis Results

This section presents the key visual insights from our exploratory data analysis. All visualizations are available in `data/eda_outputs/` directory.

6.4.1 Text Analysis Visualizations

Word Clouds: - `wc_title.png` - Reveals product naming patterns - `wc_description.png` - Shows descriptive language focus - `wc_metadata.png` - Highlights brand and category distribution

Key observation: Fashion vocabulary is domain-specific with terms like "cotton", "sleeve", "dress", "shirt" dominating. This validates our decision to use standard English stopwords rather than custom domain stopwords.

Length Distributions: - `hist_title_word_count.png` - Most titles 4-8 words (median: 6) - `hist_desc_word_count.png` - Descriptions 5-15 words (median: 9)

Key observation: Both fields are concise, making every word important for search. This justifies aggressive preprocessing (stemming, stopword removal) since there's limited redundancy.

6.4.2 Numeric Features Visualizations

Price Analysis (`hist_prices.png`): - Clear concentration in budget-to-mid range (300- 1500) - Long tail distribution extends to luxury items (20,000+) - **Implication:** Price bucketing into low/mid/high effectively captures user intent

Rating Analysis (`hist_ratings.png`): - Most products cluster around 3.5-4.5 stars - Relatively few extreme ratings (1-star or 5-star) - **Implication:** Rating can be used as quality signal but won't dramatically differentiate products

Discount Analysis (`hist_discounts.png`): - Heavy skew toward high discounts (40-70%) - Median 53% indicates aggressive promotional pricing - **Implication:** Discount is a strong ranking signal; users expect deals

6.4.3 Categorical Features Visualizations

Brand Distribution (`bar_top_brands.png`): - Top brands have significantly more products than others - Shows market concentration - **Implication:** Brand boosting for popular brands may improve relevance

Seller Distribution (`bar_top_sellers.png`): - Similar concentration pattern - Established sellers have larger catalogs - **Implication:** Seller reputation could be quality signal

Stock Availability (`bar_out_of_stock.png`): - 94.15% in stock vs 5.85% out of stock - Good inventory health overall - **Implication:** Out-of-stock filtering won't drastically reduce result set

6.4.4 Product Rankings

Generated CSV tables provide curated views of the dataset:

1. **`top_rated_products.csv`** - Identifies quality products (rating 4.5)
2. **`cheapest_products.csv`** - Reveals budget segment characteristics
3. **`top_discounted_products.csv`** - Shows promotional products (discount 70%)

These tables are useful for: - **Manual inspection:** Verify data quality - **Test queries:** Create relevance judgments for evaluation (Part 3) - **UI features:** "Top Rated" or "Best Deals" sections

---

7. Technical Challenges & Solutions

7.1 NLTK Data Dependencies

Challenge: NLTK requires downloading data files (stopwords, punkt tokenizer).

Solution: Implemented `_ensure_nltk()` function:

```
def _ensure_nltk():
    try:
        stopwords.words("english")
    except LookupError:
        nltk.download("stopwords")
    try:
        word_tokenize("test")
    except LookupError:
        nltk.download("punkt")
        nltk.download("punkt_tab")
```

Benefit: Automatic download if missing; notebook runs without manual setup.

7.2 Path Management

Challenge: Running notebook from different directories breaks file paths.

Solution: Dynamic path resolution:

```
NOTEBOOK_DIR = Path(__file__).resolve().parent if "__file__" in globals() else Path().resolve()
REPO_ROOT = NOTEBOOK_DIR.parents[1] if NOTEBOOK_DIR.name == "part_1" else NOTEBOOK_DIR
DATA_DIR = REPO_ROOT / "data"
```

Benefit: Works from both `part_1/` directory and repo root.

7.3 Missing/Malformed Data

Challenge: Not all products have complete data (ratings, prices, details).

Solution: - **Null handling:** Convert to `None` if parsing fails - **Optional fields:** Functions return `Optional[type]` - **Validation:** Check field existence before accessing

Example:

```
def _to_float_price(val: Any) -> Optional[float]:
    if val is None:
        return None
    # ... parsing logic ...
    except ValueError:
        return None
```

7.4 Encoding Issues

Challenge: Special characters, accents, currency symbols in raw data.

Solution: - `unidecode()` for accent removal - Regex-based cleaning for currency symbols - UTF-8 encoding for JSON I/O

7.5 Performance

Challenge: Processing 1000+ records with multiple text fields.

Solution: - List comprehension for batch processing: `[process_record(r) for r in data]` - No unnecessary iterations - Preprocessing done once, output saved to file

Performance: Processing completes in seconds (acceptable for this dataset size).

---

8. Conclusions & Future Work

8.1 Summary of Achievements

Robust preprocessing pipeline: NLTK-based, handles edge cases
Structured data extraction: Numeric fields parsed and validated
Feature engineering: Buckets and metadata fields for ranking
Comprehensive EDA: Statistical insights and visualizations
Reproducible workflow: Automated NLTK setup, path handling

8.2 Limitations

1. **English-only:** No multilingual support
2. **No phrase detection:** Single tokens only (no bigrams)
3. **Generic stopwords:** Domain-specific stopwords not customized
4. **Simple stemming:** No lemmatization (semantic accuracy sacrificed for speed)
5. **Static buckets:** Price/discount ranges hardcoded (not data-driven)

8.3 Future Enhancements

For Part 2 (Ranking): - **N-grams:** Add bigrams for phrase matching ("running shoes") - **Field weighting:** Title matches weighted higher than description - **Synonyms:** "sneakers" = "shoes" via synonym expansion - **Spell correction:** Handle typos in queries

For Part 3 (Evaluation): - **Query log analysis:** Real user queries to inform preprocessing decisions - **A/B testing:** Compare stemmed vs. lemmatized results - **Multilingual:** Add language detection and language-specific processing

Advanced Features: - **Named Entity Recognition:** Extract brands, materials from descriptions (spaCy) - **Sentiment analysis:** Infer product quality from review snippets - **Image features:** If product images available, use visual similarity

8.4 Lessons Learned

1. **Data quality matters:** Time spent on cleaning pays off in retrieval quality
2. **EDA is essential:** Understanding data distributions informs algorithmic choices
3. **Assumptions are risky:** Document all assumptions for transparency
4. **Reproducibility:** Automated setup (NLTK download) improves usability
5. **Trade-offs are necessary:** Stemming vs. lemmatization, recall vs. precision

---

9. References

Libraries & Tools

- **NLTK:** Bird, Steven, Edward Loper and Ewan Klein (2009). *Natural Language Processing with Python.* O'Reilly Media Inc.

  - Stopwords: `nltk.corpus.stopwords`
  - Tokenization: `nltk.tokenize.word_tokenize`
  - Stemming: `nltk.stem.PorterStemmer`

- **Porter Stemmer:** Porter, M.F. (1980). *An algorithm for suffix stripping.* Program, 14(3), 130-137.
- **Unidecode:** Python library for Unicode transliteration https://pypi.org/project/Unidecode/
- **Pandas:** McKinney, W. (2010). *Data Structures for Statistical Computing in Python.* Proceedings of the 9th Python in Science Conference.
- **Matplotlib:** Hunter, J.D. (2007). *Matplotlib: A 2D graphics environment.* Computing in Science & Engineering, 9(3), 90-95.
- **WordCloud:** Mueller, A. (2023). *Word Cloud Generator in Python* https://github.com/amueller/word_cloud

Information Retrieval Concepts

- **BM25:** Robertson, S., & Zaragoza, H. (2009). *The Probabilistic Relevance Framework: BM25 and Beyond.* Foundations and Trends in Information Retrieval, 3(4), 333-389.
- **TF-IDF:** Salton, G., & Buckley, C. (1988). *Term-weighting approaches in automatic text retrieval.* Information Processing & Management, 24(5), 513-523.
- **Text Preprocessing Best Practices:** Manning, C.D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval.* Cambridge University Press.

Dataset

- Fashion products dataset compiled from public e-commerce platforms
- Dataset includes product information, pricing, ratings, and metadata
- No personally identifiable information (PII) included

Appendix A: Output Files

All generated files are available in the repository under the `data/` directory.

A.1 Generated Datasets

File	Description	Records	Fields Added
`fashion_products_dataset.json`	Original dataset	28,080	N/A (input)
`fashion_products_dataset_preprocessed.json`	Step 1 output: Original + preprocessed text fields	28,080	`title_tokens`, `title_clean`, `description_tokens`, `description_clean`
`fashion_products_dataset_enriched.json`	Step 2 output: All fields + parsed numeric + buckets	28,080	16 additional fields including normalized categories, parsed numbers, buckets, and metadata

Total fields in final dataset: 37 fields

A.2 EDA Outputs (in `data/eda_outputs/`)

Visualizations (PNG format):

Filename	Type	Description
`hist_title_word_count.png`	Histogram	Title length distribution (median: 6 words)
`hist_desc_word_count.png`	Histogram	Description length distribution (median: 9 words)
`wc_title.png`	Word Cloud	Most frequent terms in product titles
`wc_description.png`	Word Cloud	Most frequent terms in descriptions
`wc_metadata.png`	Word Cloud	Most frequent brand, category, and specification terms
`hist_ratings.png`	Histogram	Rating distribution (median: 3.8/5)
`hist_prices.png`	Histogram	Price distribution (median: 545)
`hist_discounts.png`	Histogram	Discount distribution (median: 53%)
`bar_out_of_stock.png`	Bar Chart	Stock availability (5.85% out of stock)
`bar_top_brands.png`	Bar Chart	Top 15 brands by product count
`bar_top_sellers.png`	Bar Chart	Top 15 sellers by product count

Total visualizations generated: 11 images

Data Tables (CSV format):

Filename	Rows	Description
`top_rated_products.csv`	50	Highest-rated products (sorted by rating, then price)
`cheapest_products.csv`	50	Cheapest products by selling price
`top_discounted_products.csv`	50	Highest discount percentages

Summary Files (TXT format):

Filename	Content
`summary.txt`	Missing values analysis, column list, vocabulary size
`kpi_summary.txt`	Key performance indicators: n_products, vocab_size, medians, ratios

A.3 Key Statistics Summary

Corpus Statistics: - **Total products:** 28,080 - **Vocabulary size:** 6,650 unique terms (after preprocessing) - **Median title length:** 6 words - **Median description length:** 9 words

Pricing Statistics: - **Median selling price:** 545 - **Price completeness:** 99.99% (only 2 missing) - **Median discount:** 53% - **Discount completeness:** 96.96%

Quality Metrics: - **Median rating:** 3.8/5 stars - **Rating completeness:** 91.95% (2,261 without ratings) - **Out of stock ratio:** 5.85%

Metadata Completeness: - **Brand data:** 92.85% complete - **Seller data:** 94.15% complete - **Category/sub-category:** 100% complete

Appendix B: Code Statistics

Total lines of code: ~690 lines (including comments)

Function breakdown: - Text preprocessing: 8 functions - Non-text processing: 11 functions - EDA & visualization: 5 functions - Utility functions: 3 functions

External dependencies: - `nltk` (v3.9.1) - `pandas` (v2.3.2) - `matplotlib` (v3.10.7) - `wordcloud` (v1.9.4) - `unidecode` (v1.3.8)

End of Report