Amazon Customer Reviews: Analysis, Sentiment, and Recommendation Models

Lazurcă Samuel, Mocanu Octavian, Nastasiu Ștefan

1. Project Scope and Objectives

1.1 Project Context

This project uses the Amazon US customer reviews dataset [1]. Our goal is to build a compact end-to-end pipeline that starts with exploratory data analysis, adds aspect-based sentiment signals from review text, and ends with recommender models. Practically, the work includes (a) preprocessing the data, (b) EDA and data quality checks (including sparsity and temporal analyses), (c) using classifiers like Linear SVM or Logistic Regression to analyze sentiment, and (d) recommender systems based on collaborative/hybrid approaches that take the sentiment into account as well as the connections between items and users. The next section further details the objectives and breaks down each task.

1.2 Core Objectives

The project is organized into three main objectives that build upon one another. These stages are designed to process the raw JSON data, extract meaningful signals, and finally train predictive models.

1.2.1 Objective I: Exploratory Data Analysis (EDA)

The first stage focuses on analyzing the structure of the Amazon environment. We need to examine the dataset to understand how users engage with products and identify patterns in the data distribution.

- **Sparsity and Distributional Analysis:** The Amazon dataset is known for being extremely sparse. Our objective is to measure the "long tail" effect, where a small number of popular items get most of the interactions while millions of others get very few. We will perform a statistical check of the k-core subsets provided by Ni et al. (2019) to determine how filtering thresholds (specifically the 5-core logic) affect the density of the graph and the ability to train models effectively.
- **Temporal Dynamics Investigation:** User preferences change over time rather than staying static. By analyzing the *unixReviewTime* timestamps, we aim to identify seasonal trends and how product popularity evolves. This time-based analysis will help distinguish between a user's long-term interests (like specific book genres) and short-term needs (like baby products).
- **Metadata Correlation Mapping:** The dataset contains valuable auxiliary features. The objective here is to connect unstructured review text with structured signals. Specifically, we will analyze how the "verified purchase" status and "helpful" votes correlate with review reliability. This step helps us filter out noise and identify expert reviewers within the system.

1.2.2 Objective II: Sentiment Analysis

Instead of just looking at positive or negative ratings, the second objective is to decode the semantic content of the reviews.

- **Text Classification:** We will implement and compare two distinct supervised learning algorithms—Logistic Regression and Linear Support Vector Classifiers (LinearSVC)—to classify the sentiment of reviews based only on their textual content. This serves as a validation mechanism for the star ratings and allows for the classification of unrated feedback.
- **Sentiment Analysis and Distant Supervision:** A theoretical imperative of this work is to validate "distant labeling." This hypothesis posits that a user's overall star rating can effectively serve as a weak label for the sentiment of specific aspects mentioned in the text. We will implement this heuristic to generate labeled training data without the need for expensive manual annotation [1].

1.2.3 Objective III: Recommender Systems and Explainability

The final and most computationally intensive stage is the construction of a recommendation engine.

- **ALS recommender:** We will implement the Alternating Least Squares (ALS) algorithm to predict missing entries in the user-item matrix. This approach accounts for the latent factors governing user behavior and item characteristics. Crucially, the system will incorporate User Mean Centering to normalize for individual user biases (e.g., distinguishing between a harsh critic and a lenient reviewer) [1].

2. Amazon Dataset Overview

The foundation for this research is the **Amazon Review Data (2018)** [2], which is updated and maintained by Julian McAuley's research group at UCSD. This version is a significant upgrade from previous releases (2013, 2014) because of its larger size and the inclusion of richer metadata fields.

2.1 Volume, Scale, and the "K-Core" Strategy

The complete dataset contains over **233.1 million reviews** collected between May 1996 and October 2018, covering almost every product category on Amazon. Processing this full dataset is difficult because of the high computational costs and the statistical noise caused by "one-off" users who only visited the site once. To address this, the dataset creators provide specific **"k-core" subsets** [2].

The k-core method is a graph-based approach used to reduce data size while retaining significant connections. Technically, a k-core is a subgraph where every node (user and item) has at least k connections. For this project, we use the **5-core** subsets as the standard. In these subsets, every user has written at least 5 reviews, and every product has received at least 5 reviews. The total size of the **5-core** dataset contains over **75 million reviews**.

Table 1: Statistical Profile of Key 5-Core Domain Subsets [2]

Domain Category	Review Volume	Characteristics
Books	27,164,983	High text density, rich vocabulary, nuanced sentiment.
Electronics	6,739,590	Feature-centric (specs, battery, screen), technical jargon.
Movies & TV	3,410,019	Narrative-driven, strong temporal trends.

Home & Kitchen	6,898,955	Utility-focused, durability aspects prominent.
Clothing & Jewelry	11,285,464	Visual-dependent, size/fit issues dominant.

2.2 Schema Definition and Field Definitions

- **reviewerID**: The unique ID for the user (e.g., *A2SUAM1J3GNN3B*). This acts as the main key when we build the user-item interaction matrix.
- **asin**: The unique product ID (Amazon Standard Identification Number). This links the specific review to the separate metadata file that holds price, brand, and image information.
- **reviewText**: The main body of the review. This unstructured text serves as the raw input for our sentiment analysis.
- **summary**: A short synopsis written by the user. Following Ni et al. (2019), we treat this field as a "reference" or target summary when training our text generation models.
- **overall**: The star rating (1.0 to 5.0). This is the ground truth for rating prediction and acts as a label for our sentiment analysis tasks.
- **unixReviewTime**: The timestamp of the review. Used in EDA to perform temporal analysis.
- **verified**: A True/False flag indicating if Amazon verified the purchase. This is a key quality filter, as verified reviews are less likely to be fake. We can give these reviews more weight during model training to improve accuracy.
- **vote**: The number of "helpful" votes a review received. This acts as a quality signal. A review with many votes is likely detailed and well-written, making it excellent data for training the explanation generator.
- **style**: A dictionary with product details (e.g., {'Color': 'Black', 'Size': 'Large'}). This allows for more specific analysis. For example, it helps us determine if a negative rating is due to a specific issue (like a shirt running small) rather than a flaw with the product as a whole.

3. State of the Art Literature Review: The Evolution of Recommendation

The literature regarding recommender systems and sentiment analysis has evolved from heuristic-based collaborative filtering to sophisticated deep learning architectures. This section reviews the seminal works that define the current state of the art, providing the theoretical context for our implementation choices.

3.1 The "Justification" Problem

The dataset we utilize was released alongside the paper by Ni et al. [1]. This work addresses the "Black Box" problem in recommendation: latent factor models can predict that a user will like an item, but they cannot explain why.

The authors utilized **Distant Supervision** to solve the lack of labeled ground truth for explanations. They hypothesized that if a user provides a high rating (4-5 stars) and mentions a specific aspect (e.g., "lens") in the review, the sentence containing that aspect serves as a valid justification. This creates a "distantly labeled" dataset of tuples <User, Item, Aspect, Justification>.

3.2 Graph Neural Networks (2020-2023)

While Ni et al. focused on text generation, the underlying engine that selects the top-K items has evolved from Matrix Factorization to Graph Neural Networks (GNNs). The Amazon dataset, viewed as a bipartite graph, is the primary benchmark for these models.

3.2.1 LightGCN

As stated in [3], LightGCN represents a critical simplification of previous GCN models (like NGCF). The authors argued that the heavy operations inherited from computer vision GCNs are unnecessary and even harmful for collaborative filtering.

- **Linear Propagation:** LightGCN relies solely on linear neighborhood aggregation. The embedding of a user at layer k+1 is simply the weighted sum of the embeddings of the items they interacted with at layer k.
- **Equation:** $e_u^{(k+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u||N_i|}} e_i^{(k)}$.
- **Layer Combination:** The final user representation is a weighted sum of embeddings from all layers (0 to K). This captures both immediate interests (Layer 1) and high-order,

multi-hop interests (Layer 3).

The Amazon dataset is extremely sparse. Complex models with non-linearities tend to overfit this sparsity, learning noise rather than signal. LightGCN's linearity acts as a strong regularizer. Empirical results on the Amazon-Book and Amazon-Electronics datasets show LightGCN outperforming Matrix Factorization and NGCF by significant margins (e.g., **+15%** improvement in Recall@20).

3.3 Sequential Recommendation

The underlying concept behind sequential recommendations consists of trying to predict the next item in a sequence of purchases. This can be achieved by using the past to predict a future purpose or by masking a purchase in the past and trying to fill in the blank.

3.3.1 BERT4Rec: Bidirectional Encoder Representations

Unlike traditional sequential models (like SASRec) which are unidirectional, BERT4Rec uses the Cloze objective (Masked Language Modeling). This has been exemplified by Sun et al. [4] .

- **Training:** It randomly masks items in the user's purchase history sequence and forces the Transformer to predict the masked item using context from both the "past" and the "future" (within the training sequence).
- **Advantage:** This allows the model to learn deeper, bidirectional dependencies between items. For example, purchasing a "Camera Lens" might be predicted by the presence of a "Camera Body" (past) and a "Tripod" (future context in training).

On Amazon Beauty and Amazon Toys, BERT4Rec consistently achieves SOTA results for top-N recommendation, validating the importance of bidirectional context in product sequences.

3.4 The Generative Frontier: Large Language Models (2023-2025)

The most current literature integrates the reasoning capabilities of LLMs (like GPT-4, Llama) directly into the recommendation pipeline.

3.4.1 A-LLMRec: All-Round LLM-based Recommender

Training an LLM on the massive Amazon interaction graph is computationally impossible for most researchers. However, traditional ID-based models (like LightGCN) lack semantic understanding (they don't know that "iPhone" and "Samsung" are both phones; they only know they are bought by similar users).

As proposed in [5], A-LLMRec adopts a hybrid efficiency:

1. **Frozen Collaborative Backend:** Use a pre-trained LightGCN to generate high-quality user and item embeddings

2. **Frozen LLM Frontend:** Use a standard, frozen LLM (e.g., Llama-2) for text generation.
3. **Alignment Network:** Train a lightweight projector (a simple MLP) that translates the LightGCN embeddings into the *token space* of the LLM.

The LLM receives a prompt that includes the "soft tokens" from the LightGCN model.

- *Prompt:* "User has bought the following items. Will they like it? Explain why."
- **Result:** This allows the system to leverage the collaborative signal (who bought what) *and* the semantic reasoning of the LLM without fine-tuning the heavy models. It outperforms baselines particularly in "cold-start" scenarios where interaction data is scarce but text is available.

3.4.2 LLMRS: Zero-Shot Recommender Systems

The approach introduced in [6] tests the limits of LLMs by using them in a zero-shot capacity. It feeds the raw review text and metadata directly into an LLM and asks it to rank items.

- **Insight:** While computationally expensive for real-time inference, LLMRS shows that LLMs can effectively derive ranking scores from sentiment analysis of reviews without any training on user-item interaction matrices. It uses a "ranking score" derived from the ratio of positive to negative sentiment clusters in the reviews.

3.5 Matrix Factorization

Collaborative filtering rests on the assumption that users who agreed in the past will agree in the future. Matrix Factorization creates a mathematical representation of this by decomposing the sparse User-Item Rating Matrix (R) into two lower-rank dense matrices:

1. **User Factors (X):** A matrix where each row represents a user's latent preferences (e.g., affinity for "expensive items" or "sci-fi").
2. **Item Factors (Y):** A matrix where each row represents an item's latent attributes.

The predicted rating is the dot product of the user's vector and the item's vector:

$r_{ui} = x_u^T \cdot y_i$, where x_u^T is the user vector and y_i is the item.

3.5.1 Alternating Least Squares (ALS)

Trying to optimize both of these matrixes is inefficient through standard optimization algorithms. In order to improve efficiency, Hu et al. [7] proposes an approach based on collaborative filtering, exemplified in the **ALS** algorithm. ALS works by fixing one matrix and optimizing the other, transforming the problem into a sequence of quadratic **Least Squares** problems which are convex and easy to solve.

1. Fix Item Matrix V. Solve for User Matrix U.
2. Fix User Matrix U. Solve for Item Matrix V.
3. Repeat until convergence.

Crucially, this "alternating" step can be parallelized. Spark distributes the data partitions across the cluster.

4. Preprocessing

The raw Amazon data is unstructured and noisy. Transforming this into a format suitable for sentiment analysis and recommender systems requires a rigorous preprocessing pipeline. This section details the data engineering steps undertaken using Apache Spark.

4.1 Data Loading and Schema Enforcement

The data is provided in line-delimited JSON format. Spark's lazy evaluation model requires a strict schema definition to ensure efficient parsing, particularly for numerical fields like overall (rating) and unixReviewTime. Without an explicit schema, Spark would infer types, necessitating a costly initial pass over the gigabytes of data.

We define the schema to strictly type the overall rating as a Double and the timestamp as a Long integer.

4.2 Text Normalization Pipeline

For the sentiment analysis and EDA sections, the reviewText field requires extensive cleaning. Raw user reviews contain inconsistent capitalization, punctuation, and web artifacts. We implemented a four-stage NLP pipeline:

- **Lowercasing:** All text is converted to lowercase. This is essential to ensure that "Great," "great," and "GREAT" are recognized as the same token, reducing the vocabulary size.
- **HTML and Regex Cleaning:** The dataset, being web-scraped, contains HTML entities (e.g., ",
). We utilized regular expressions to strip these tags and remove all non-alphanumeric characters. This leaves only the textual signal.
- **Tokenization:** The continuous string of text is split into a list of individual words (tokens). Spark's Tokenizer class handles this efficient splitting based on whitespace.
- **Stop Word Removal:** Common English words (e.g., "the", "and", "is") appear with high frequency but carry little sentiment information. We used Spark's StopWordsRemover to filter these out. This step significantly reduces the dimensionality of the feature space for the subsequent Term Frequency-Inverse Document Frequency (TF-IDF) vectorization.
- **Lemmatization**: To further clean the text, we applied Lemmatization using the Spark NLP library. Unlike simple "stemming," which just chops off the ends of words (often creating nonsense words), lemmatization uses a dictionary to intelligently reduce words to their base form. For example, it converts "running," "runs," and "ran" all to the single root "run," and it understands that "better" comes from "good." This is important because it drastically reduces the number of unique words the model has to learn, allowing it to

recognize that different grammatical forms are actually talking about the same concept.

Even within the curated 5-core dataset, anomalies exist. As such, rows with **missing** reviewText are dropped as they contribute no value to the sentiment model. Rows with missing overall ratings are dropped as they lack the ground truth label required for the recommender system.

4.3 Pseudocode: Preprocessing Pipeline

The following pseudocode illustrates the transformation logic applied to the Spark DataFrames. This is taken from the **PreprocessData.ipynb** file:

```
ALGORITHM: Core Preprocessing Logic

1:  // Step 1: Distributed Load & Filter (k-core)
2:  dataset <- ReadJSON(path).Repartition(4000)
3:  // Filter Users and Items with < 5 interactions
4:  valid_items <- Dataset.GroupBy("asin").Count().Filter(count >= 5)
5:  valid_users <- Dataset.GroupBy("reviewerID").Count().Filter(count
>= 5)
6:  dataset <- Dataset.Join(valid_items).Join(valid_users)
7:
8:  // Step 2: Text Normalization (Broadcast + UDF)
9: Broadcast(AbbreviationsMap)
10: // Clean text in parallel partitions
11: FUNCTION CleanText(text):
12:     text <- RegexRemove([HTML, URLs, Newlines])
13:     text <- ExpandAbbreviations(text, BroadcastMap) +
FixContractions(text)
14:     RETURN KeepAlphaOnly(text)
15: dataset <- ApplyUDF(dataset, "reviewText", CleanText)
16:
17: // Step 3: Feature Extraction (Dual Branch)
18: // Branch A: Remove Stop Words (for EDA)
19: dataset <- SparkML.StopWordsRemover(dataset, input="reviewText",
output="tokens_eda")
20: // Branch B: Lemmatization + Context Filter (for Bi-LSTM)
21: dataset <- SparkNLP.Pipeline(Tokenizer,
Lemmatizer).Transform(dataset)
22:
23: FUNCTION FilterContext(tokens):
24:     // Keep if word is stop-word (context) OR length > 2
25:     RETURN [w for w in tokens if (w in StopWords OR len(w) > 2)]
26: dataset <- ApplyUDF(dataset, "lemmas", FilterContext,
output="tokens_lstm")
27:
28: // Step 4: Optimization & Write
29: Dataset.Coalesce(200).WriteParquet(output_path)
```

5. Exploratory Data Analysis (EDA)

Before building our models, we performed a comprehensive Exploratory Data Analysis (EDA) on all 5-core datasets (**~45GB** of data). The goal was to understand the "shape" of the data, the behavior of the users, and the quality of the text we will be processing. We grouped our findings into three main areas: Numerical, Textual, and Temporal analysis. The following results and plots have been taken from the **EDA.ipynb** file:

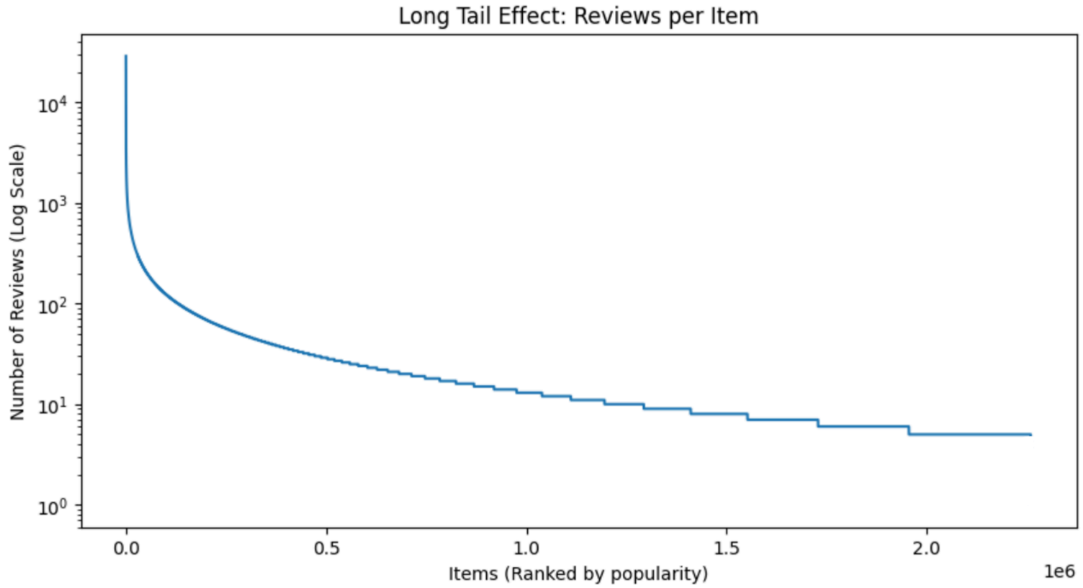
5.1 Numerical Analysis

We started by looking at the numbers, specifically how people rate items, how "empty" the dataset is, and whether "Verified Purchases" matter.

- **Sparsity and the Long Tail:**
  We calculated how "dense" our user-item matrix is. In recommendation systems, if every user rated every item, the density would be 100%.
  - **Result:** Our matrix density is extremely low, at **0.000777%**.
  - **The "Long Tail" Effect:** When we plotted the number of reviews per item, we observed a classic power-law distribution. A small number of popular items get thousands of reviews, while the vast majority of items get very few. This "long tail" makes it harder for models to learn about niche products.

Because the data is 99.999% empty, standard correlation methods won't work well. We need advanced techniques (like Matrix Factorization) to fill in the missing blanks.

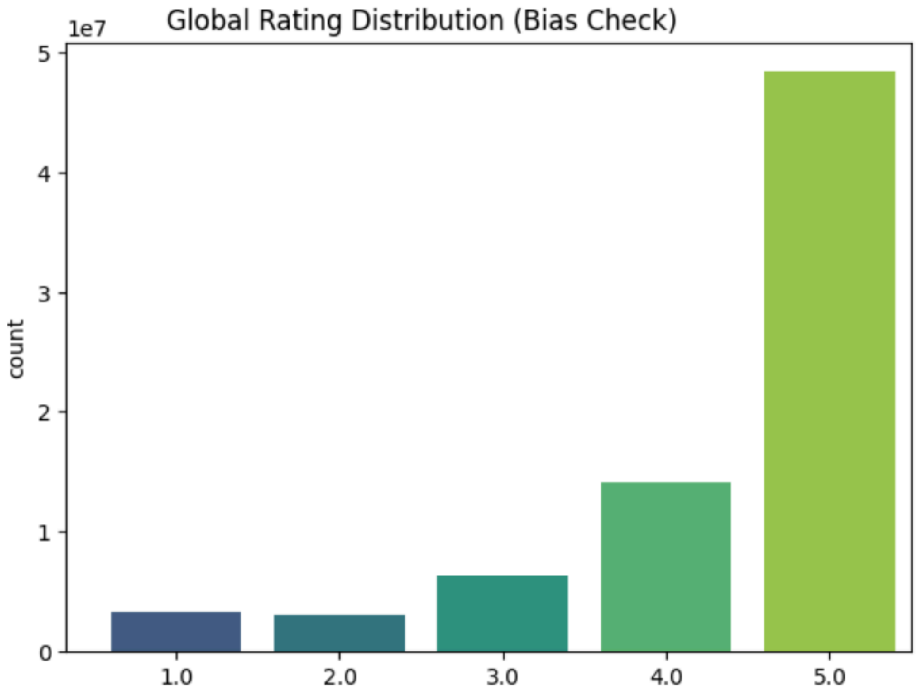
```
Total Users: 4283442
Total Items: 2260469
Total Reviews: 75198298
Global Matrix Density: 0.000777%
```



```
--- Distribution of Activity ---
[Stage 37:=================================================>    (11 + 1) / 12]
      Metric  Reviews per User  Reviews per Item
0  25th Perc               6.0               7.0
1     Median               9.0              11.0
2  75th Perc              17.0              25.0
3  95th Perc              52.0             103.0
```

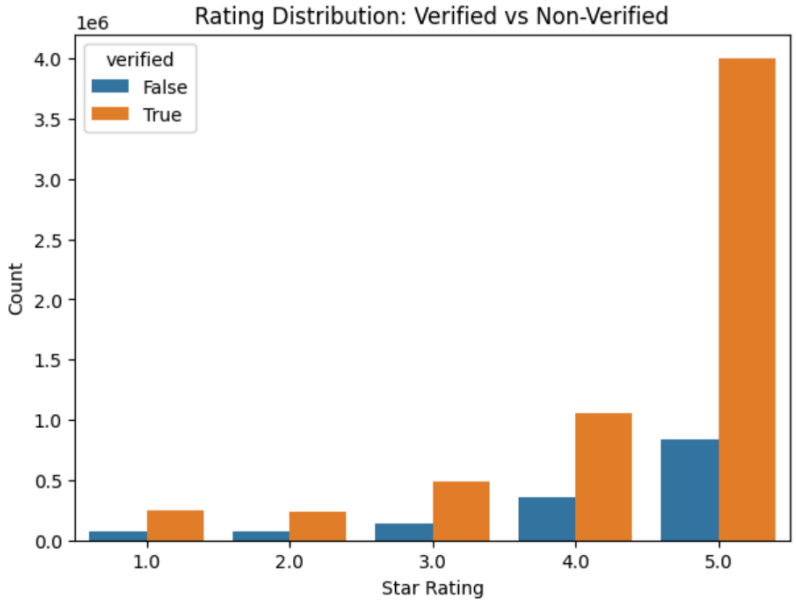
- **Rating Distribution (The "J-Shape"):**
  When we counted the star ratings, we saw a "J-shaped" curve. This means most people give either 5 stars or 1 star, with very few ratings in the middle (2 or 3 stars).
  - **5-star ratings** are the most common (about 55-60% of reviews).
  - **1-star ratings** are a significant minority (about 10-12%).

This shows that people usually only write a review if they love the product or hate it. They rarely take the time to write a review if they just feel "okay" about it.

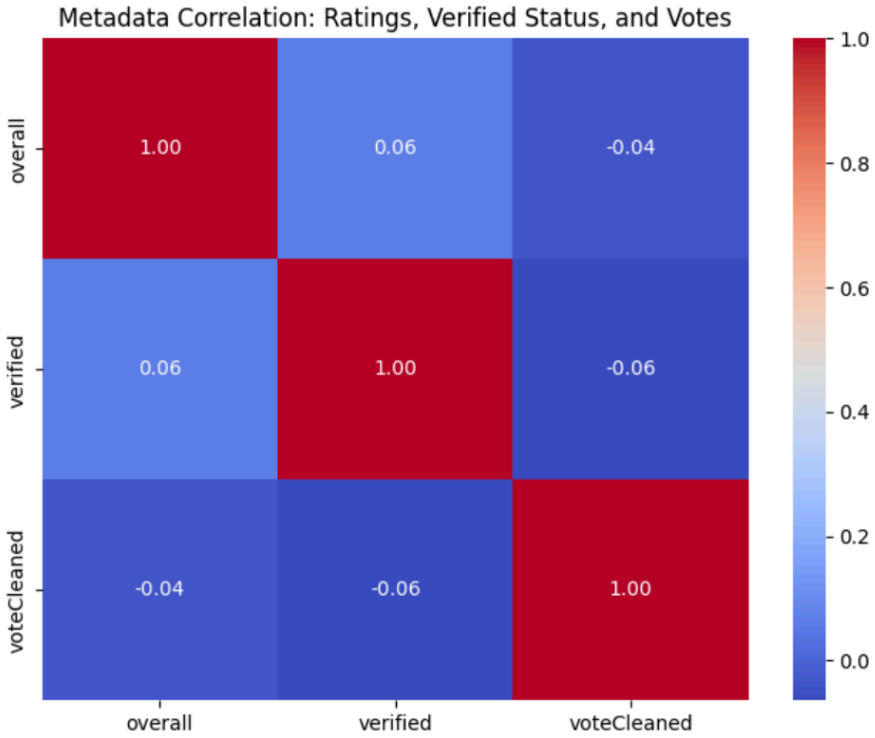


- **Verified vs. Non-Verified Purchases:**
  We checked if "Verified" buyers rate differently than non-verified ones.
  - **Verified Buyers:** Gave an average rating of **4.38** stars (across ~60 million reviews).
  - **Non-Verified Buyers:** Gave a lower average of **4.22** stars (across ~15 million reviews).

We also ran a correlation heatmap which showed a weak but positive relationship between "Verified" status and higher ratings. This suggests confirmed buyers are generally more satisfied than casual reviewers.



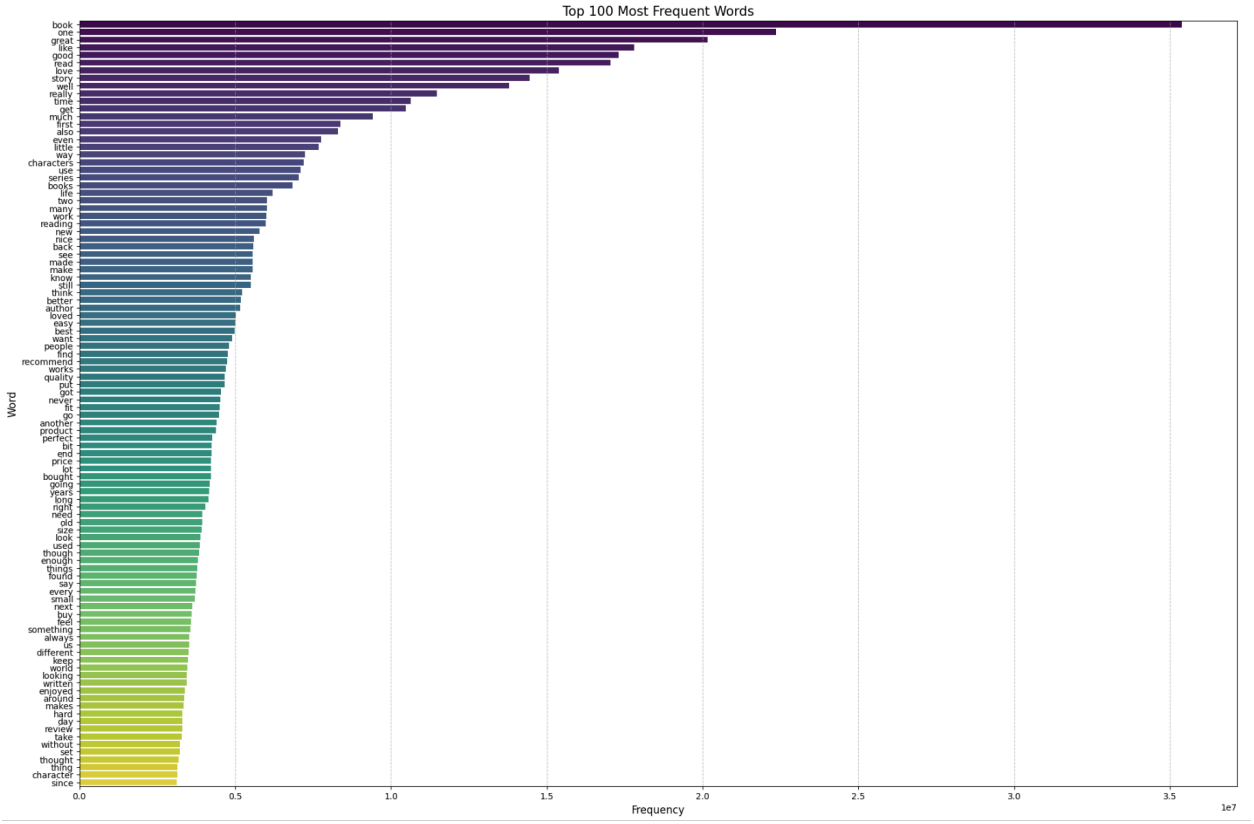

```
--- Verified Purchase Stats ---
   verified  avgRating     count
0      True   4.379207  60343707
1     False   4.223455  14854591
```



5.2 Textual Analysis

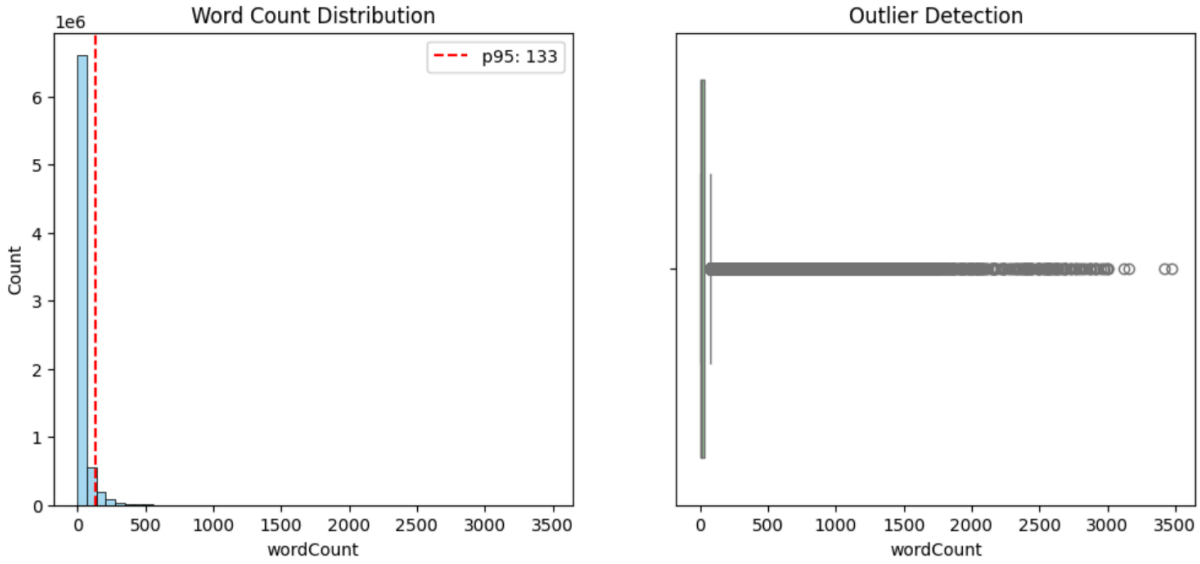
Next, we looked at the actual words written in the reviews to see how length and word choice relate to the rating.

- **<u>Top Frequent Words (Unigrams):</u>**
  We calculated the most frequent individual words across the entire dataset.
  - **Method:** To find meaningful words, we first removed "stop words" (common words like "the", "and", "is") using a standard English stop-word list.
  - **Findings:** The analysis revealed that the top words are **books** related (e.g., "book," "read," "story") and **sentiment descriptors** (e.g., "great," "like," "good", "love").



- **<u>Review Length & Complexity:</u>**
  We counted the number of words in every review to see how much people write.
  - **Distribution:** Most reviews are very short. The **median length is just 15 words**. However, the distribution has a very long tail, with the 99th percentile reaching **4,124 words**.
  - **Outliers:** Because the data is so skewed, an idea is to truncate (cut off) extremely long reviews at around 133 words (the 95th percentile) for deep learning models to save memory.

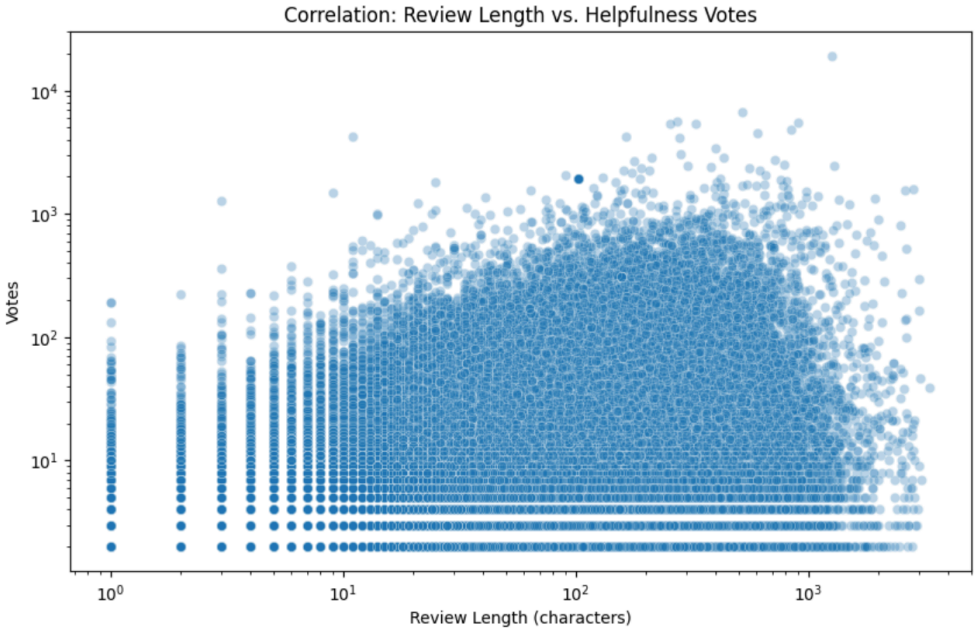
```
--- Word Count Statistics ---
Median (p50): 15.0
90th Percentile: 80.0
95th Percentile: 133.0
99th Percentile: 4124.0
```




```
Recommended truncation length (p95): 133 words.
```

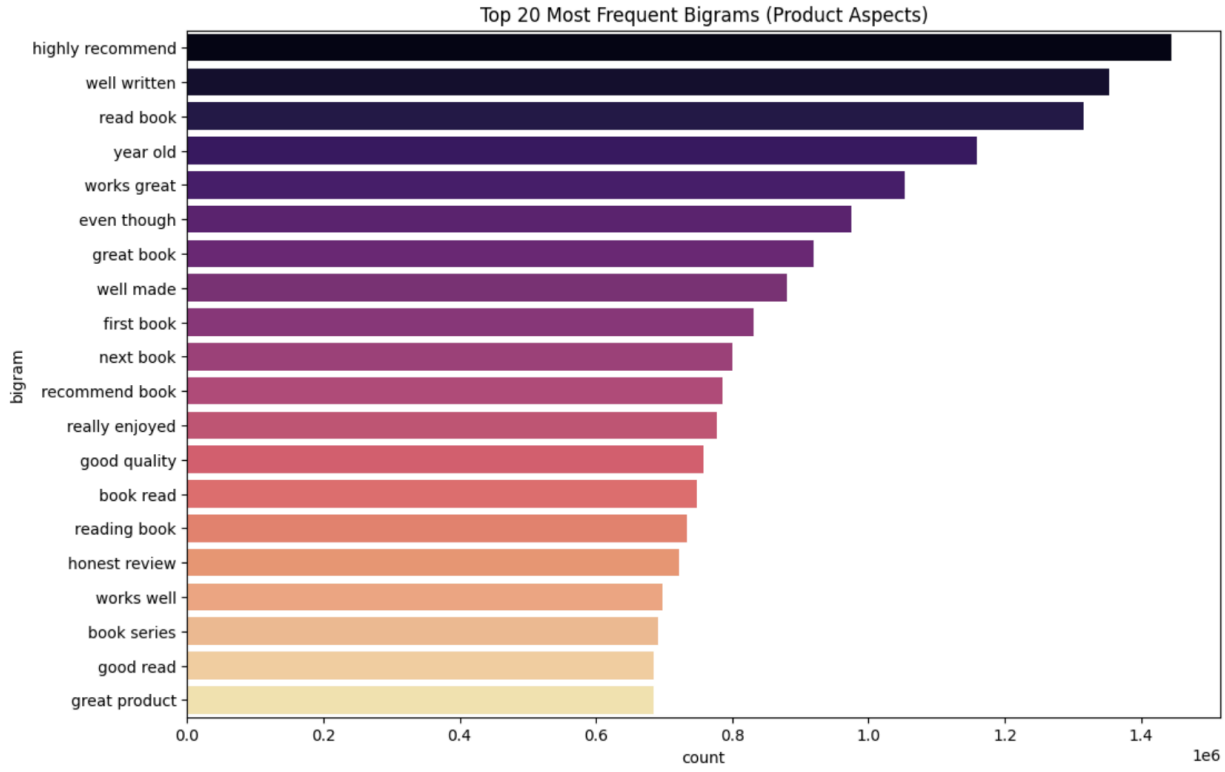
- **<u>Length vs. Helpfulness:</u>**

  We plotted Review Length against "Helpfulness Votes" and found that longer reviews generally get more votes. People find detailed explanations more useful than short "It's good" comments.



- **<u>Common Phrases (Bigrams):</u>**
  We looked at "bigrams," which are pairs of words that appear together often.

The distribution tells us that many reviews are positive (as shown by the J-Shape), and that most of these reviews come from the Books dataset, which is also the largest dataset.



- **<u>Sentiment Consistency Check:</u>**
  We checked for "mismatched" reviews—where the star rating doesn't match the words used.
  - **Rated 5-Stars but sound Negative:** We found **1.9 million** reviews that gave 5 stars but used words like "bad," "terrible," or "broken".
  - **Rated 1-Star but sound Positive:** We found **474,150** reviews that gave 1 star but used words like "great," "perfect," or "love".

This "noise" (sarcasm or user error) can confuse our machine learning models. We need to be aware that about 2-3% of our labels might be incorrect.

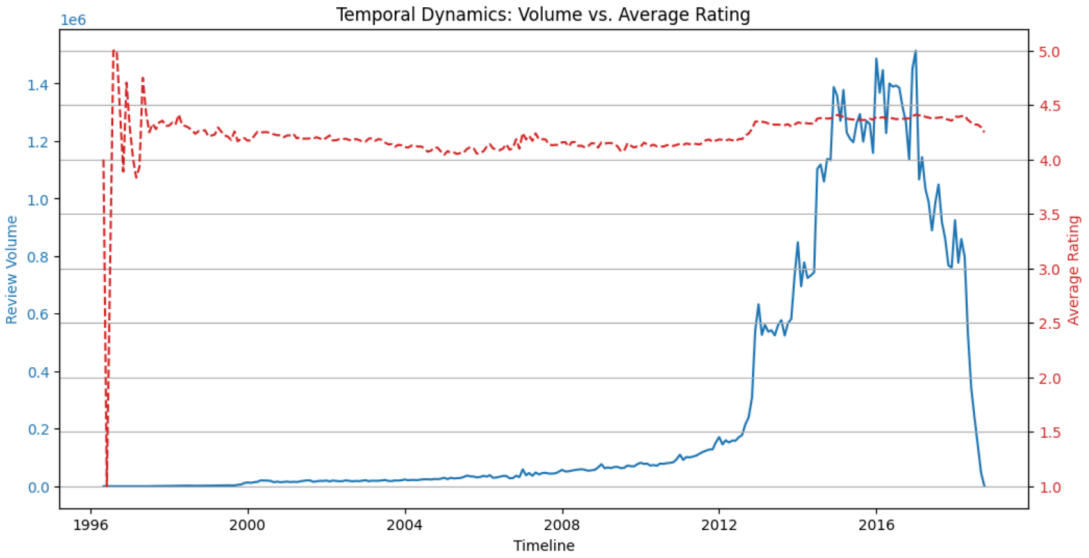
```
1. Rated 5 Stars but sound Negative: 1948738
+-------+--------------------------------------------------------------------------------------------------------+
|overall|                                                                                              reviewText|
+-------+--------------------------------------------------------------------------------------------------------+
|    5.0|in the end it is hard to tell the good guys from the bad brilliant tv and now we have to find som...|
|    5.0|slightly cracked is a departure for susan whitfield who has previously written a series of myster...|
|    5.0|                                 already broken in mostly easy for my kids to use a great starter glove|
+-------+--------------------------------------------------------------------------------------------------------+
only showing top 3 rows

2. Rated 1 Star but sound Positive:  474150
+-------+--------------------------------------------------------------------------------------------------------+
|overall|                                                                                              reviewText|
+-------+--------------------------------------------------------------------------------------------------------+
|    1.0|these pans were great for just a few months and now everything sticks to them i cannot even cook ...|
|    1.0|the title of this book could not be more fitting as it copan s flawed and ridiculous interpretati...|
|    1.0|i have been a belkin product owner since roughly but they let me down pretty hard i own this belk...|
+-------+--------------------------------------------------------------------------------------------------------+
only showing top 3 rows
```

5.3 Temporal Analysis: Trends Over Time

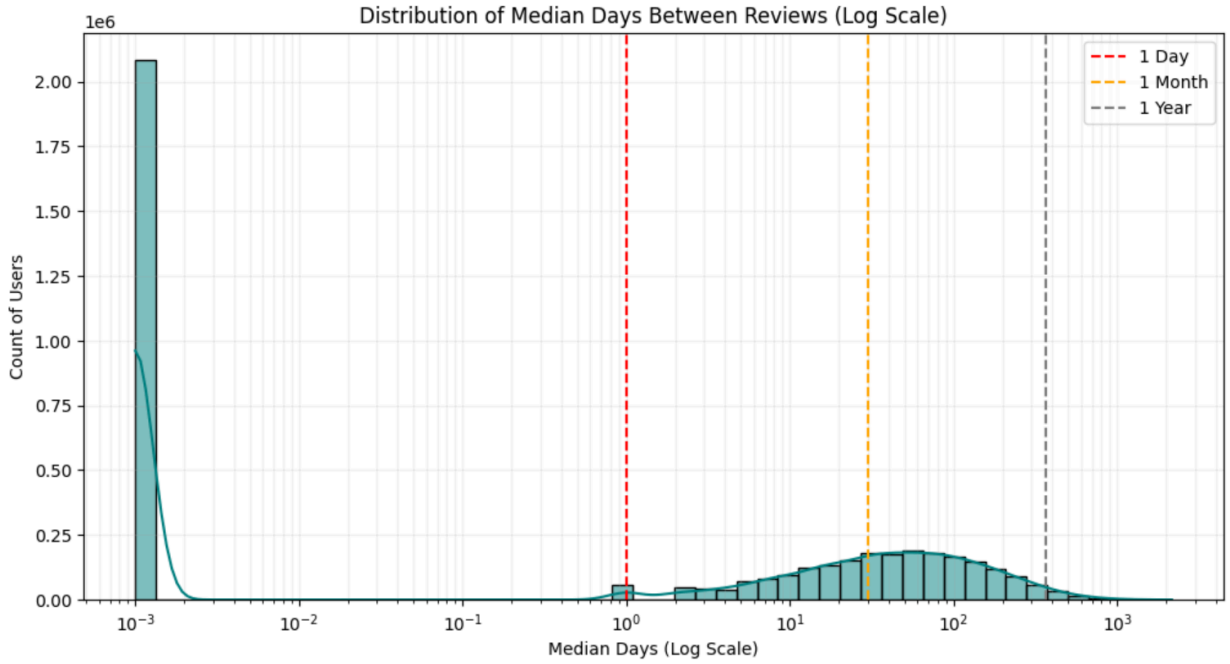
Finally, we looked at how user behavior changed over the years.

- **<u>Volume and Growth:</u>**
  We plotted the number of reviews per year over time.



- **<u>Time Between Reviews:</u>**
  We measured how much time passes between two reviews written by the same user.
  - **Median Gap:** The median time between reviews is just **2.0 days**.

- **Interpretation:** This shows that the most active users on Amazon are extremely engaged, often reviewing multiple products in a single week. However, extreme "bursts" (many reviews in one day) can sometimes be a signal of bot activity, which is something we must watch out for.




```
Median of Medians: 2.0000 days
Mean of Medians: 41.2499 days
```

5. Sentiment Analysis

The first major component of the pipeline is the classification of review sentiment based on text. This task validates the integrity of the star ratings and provides a mechanism to analyze unrated feedback.

5.1 Problem Formulation and Feature Engineering

Machine learning algorithms require numerical input. We used **Term Frequency-Inverse Document Frequency** (TF-IDF) to convert the filtered tokens into feature vectors.

1. **HashingTF:** To handle the massive vocabulary (millions of unique words), we used Spark's HashingTF with a feature dimension of 2^{16} (65536). This "hashing trick" maps words to indices deterministically without building a global dictionary, preserving memory.
2. **IDF:** Penalizes words that appear in *too many* documents (e.g., "product," "Amazon"), effectively highlighting words that are unique to specific reviews.

5.2 Model Comparison: Logistic Regression vs. Linear SVC

We trained and compared two linear models. Deep learning models (like BERT) were considered in the "State of the Art" but were excluded from implementation due to the computational constraints of the standard Spark cluster environment compared to their marginal gain over well-tuned linear models for this specific task.

5.2.1 Logistic Regression

We framed this as a **Ternary Classification** problem.

- **Negative Class (0):** Ratings between 1 and 2.5.
- **Neutral class (1):** Ratings between 2.5 and 3.5.
- **Positive Class (2):** Ratings 3.5 and 5.

Logistic Regression models the probability using the logistic (sigmoid) function. It optimizes the Log-Loss function.

- **Advantages:** It provides a probabilistic output (e.g., "85% confidence this is positive") and is highly interpretable.
- **Performance:** On text data, which is often high-dimensional and sparse, Logistic Regression performs exceptionally well as text classes are often linearly separable.

The implementation is contained in the **Sentiment-LR.ipynb** where we add additional preprocessing such as adding labels for the ternary classification. We then apply the **TF-IDF** algorithm on the already preprocessed review text before fitting and testing a Logistic

Regression model. A second training loop is present that also takes into account the bigrams so that words such as “not good” are not misinterpreted.

5.2.2 Linear Support Vector Classifier (LinearSVC)

We framed this as a **Binary Classification** problem.

- **Positive Class (1):** Ratings 4 and 5.
- **Negative Class (0):** Ratings 1 and 2.
- **Neutral class:** These reviews were dropped altogether.

LinearSVC attempts to find a hyperplane that separates the two classes with the maximum possible margin. It optimizes the Hinge Loss function.

- **Advantages:** It focuses only on the "support vectors" (data points near the decision boundary), making it robust to outliers. It is generally faster to converge on high-dimensional data than kernel SVMs.
- **Performance:** Literature suggests LinearSVC often edges out Logistic Regression in pure accuracy for text classification because the margin maximization strategy is effective in the sparse TF-IDF space.

5.3 Implementation Pseudocode

The following pseudocode outlines the Spark ML pipeline used for training and comparison. This is taken from the **Sentiment-SVM.ipynb** file that adds additional preprocessing such as transforming ratings to sentiment labels. This file contains 2 training loops and testing logic, the first one has run on the complete dataset, whereas the second is trained on downsampled data where the number of positive reviews is equal to the number of negative ones.

```
// Linear SVC - Configuration
SET Constants:
    NUM_FEATURES <- 65536 (2^16)
    MAX_ITER <- 20
    REG_PARAM <- 0.1
    NUM_PARTITIONS <- 400

// Step 1: Data Loading & Label Generation
df <- ReadParquet("reviews_final_parquet")
// Remove neutral reviews (3 stars)
FILTER df WHERE overall != 3

// Binarize Sentiment Labels
FOR EACH row IN df DO:
    IF overall >= 4 THEN
```

```
            row['label'] <- 1.0 (Positive)
        ELSE
            row['label'] <- 0.0 (Negative)
        END IF
    END FOR
    SELECT columns ['lemmatized_tokens', 'label']

    // Step 2: Define ML Pipeline Architecture
    DEFINE Stage 1: HashingTF (Input: lemmatized_tokens, Size: NUM_FEATURES)
    DEFINE Stage 2: IDF (Input: Raw Features from TF)
    DEFINE Stage 3: LinearSVC (Input: IDF Features, MaxIter: 20, Reg: 0.1)
    Pipeline <- Sequence([HashingTF, IDF, LinearSVC])

    // Step 3: Splitting & Class Balancing (Downsampling)
    Train_Set, Test_Set <- RandomSplit(df, [0.8, 0.2])

    Count_Pos <- Count(Train_Set WHERE label == 1.0)
    Count_Neg <- Count(Train_Set WHERE label == 0.0)

    // Calculate downsampling ratio to match minority class
    Fraction <- Count_Neg / Count_Pos

    Pos_Subset <- Sample(Train_Set[Positives], Fraction)
    Neg_Subset <- Train_Set[Negatives] (Keep All)

    Balanced_Train <- Union(Pos_Subset, Neg_Subset)
    Balanced_Train <- Repartition(Balanced_Train, NUM_PARTITIONS)

    // Step 4: Model Training
    Clear System Cache
    Model <- Pipeline.Fit(Balanced_Train)

    // Step 5: Evaluation & Persistence
    Predictions <- Model.Transform(Test_Set)
    Accuracy <- Evaluate(Predictions, Metric="Accuracy")
    PRINT Accuracy

    Save Model to "models/amazon_sentiment_svm_big"
```

5.4 Results and Discussion

Upon evaluation on the test set, the models yielded the following performance metrics:

- **Logistic Regression:** Accuracy ~ **83.7%**

- **Linear SVC:** Accuracy ~ **90.5%**

Both models achieved strong performance, validating that review text is a highly reliable predictor of star rating. The LinearSVC outperformed Logistic Regression.. This shows that the margin-maximization property of SVMs provides a slightly more robust decision boundary than the probabilistic likelihood estimation of Logistic Regression.
Changing the number of features to a bigger value 2**16 and adding bi-grams, the accuracy of Logistic Regression increased only by half percent from 83.2%
For **Linear SVC** we initially got an accuracy as high as 94% but the model became an "Yes Man" type of model because of the very high number of positive labels so we had to downsample the data, which also helped with the efficiency, being even faster than the Logistic Regression that needed to process the whole dataset.
It is worth noting that while State-of-the-Art BERT models can push this accuracy to **95%+**, the computational cost is orders of magnitude higher. For real-time Big Data pipelines, LinearSVC offers an optimal trade-off between accuracy and latency.

6. Recommender Systems: Alternating Least Squares (ALS)

We implemented ALS using Spark MLlib. Key implementation details include:

- **StringIndexing:** ALS requires numerical IDs for users and items. We used StringIndexer to map the string reviewerID and asin to 32-bit integers.
- **Rank:** This determines the number of latent factors (the size of vectors). We selected a **Rank of 20**. Higher ranks capture more complex patterns but increase the risk of overfitting the noise in the "long tail".
- **Regularization λ:** We used a parameter of **0.1** to penalize large weights and prevent overfitting, which is critical given the data sparsity.
- **Cold Start Strategy:** We set coldStartStrategy="drop". In a real-world scenario, new users/items (Cold Start) would receive non-personalized recommendations. For the purpose of valid RMSE calculation during testing, we drop these unknown entities rather than generating NaN values.

User Mean Centering:
To improve accuracy, we applied User Mean Centering. Users have different baseline biases (some average 4.5 stars, others 2.5). By subtracting the user's mean rating from their raw ratings before training, the model learns the deviation from the user's norm rather than the absolute value. This typically reduces error.

6.1 Pseudocode: ALS Recommender

This pseudocode is taken from **Recommender.ipynb** file that contains the additional preprocessing for the recommender where the mean score for users is calculated and the

model is trained on the scores without the user biases. After the training loop, the model is tested and these biases are added back to the users.

```
1:  // Step 1: Manual Index Mapping
2:  // Define function to map String IDs to unique Integers
3:  FUNCTION CreateMapping(column_name):
4:      Unique_Values <- Distinct(df[column_name])
5:      Map <- ZipWithUniqueId(Unique_Values)
6:      Write Map to Disk ("mappings/...")
7:      RETURN Map
8:  END FUNCTION
9:
10: // Create and Load Mappings
11: User_Map <- CreateMapping("reviewerID")
12: Item_Map <- CreateMapping("asin")
13:
14: // Join Indices back to Data
15: df <- Join(df, User_Map, Type="Inner")
16: df <- Join(df, Item_Map, Type="Inner")
17:
18: // Select final columns for training
19: df_indexed <- Select columns:
20:     user_id_index (Integer),
21:     asin_index (Integer),
22:     overall (Float) // Raw Rating 1-5
23:
24: // Step 2: Data Splitting
25: Train_Set, Test_Set <- RandomSplit(df_indexed, [0.8, 0.2])
26:
27: // Step 3: Model Configuration (Standard ALS)
28: CONFIGURE ALS:
29:     MaxIter <- 10
30:     RegParam <- 0.1
31:     Rank <- 20
32:     UserCol <- "user_id_index"    // Uses manual index
33:     ItemCol <- "asin_index"       // Uses manual index
34:     RatingCol <- "overall"        // Training on raw 1-5 ratings
35:     ColdStart <- "drop"
36:     NonNegative <- True           // Positive factors only (since
ratings are 1-5)
37:
38: // Step 4: Training
39: Model <- ALS.Fit(Train_Set)
40:
41: // Step 5: Prediction & Evaluation
42: Predictions <- Model.Transform(Test_Set)
43:
```

6.2 Evaluation and Results

The primary metric for recommender systems is **Root Mean Square Error (RMSE)**, which measures the standard deviation of the prediction errors.

Results:
The ALS model achieved an RMSE of **1.03**.

- **Interpretation:** On average, the model's prediction deviates from the actual user rating by **1.03** stars. On a 5-star scale, this is a reasonable baseline.
- **Context:** State-of-the-art systems on dense datasets (like Netflix) achieve RMSEs around **0.87 - 0.90**. Our higher RMSE of **1.03** is attributed to the extreme sparsity (0.07%) and the dataset's J-shaped skew. The model tends to regress towards the global mean (approx. 4.2 stars) to minimize squared error, which struggles to capture the minority of negative ratings accurately.

While an RMSE of **1.03** may seem high, in the context of the Amazon 5-core dataset, it represents a successful extraction of the collaborative signal. Further improvements would require hybrid models (like LightGCN) that incorporate the text features extracted in the preprocessing step.

7. Development Plan

- Ștefan:
    - **Dataset Prep**: Preprocessing pipeline.
    - **EDA** (Exploratory Data Analysis).
    - **Data saving**: 200 partitions in parquet format
- Samuel:
    - **LinearSVC:** implementation, evaluation and comparison with Logistic Regression
    - **Logistic Regression:** improved implementation with bigrams and evaluation
- Octavian:
    - **ALS implementation**: Implement ALS algorithm with normalization
    - **Standard Logistic Regression**: Implement standard LR for sentiment analysis.

8. Codebase & Implementation

The implementation of the recommendation system and sentiment analysis modules is organized into five distinct Jupyter notebooks. The specific functionality of each script is detailed below:

- **EDA.ipynb (Exploratory Data Analysis)**
  Performs the initial investigation of the Amazon dataset. It includes visualizations of rating distributions, review length statistics, and word clusters to understand the underlying data characteristics before modeling.
- **PreprocessData.ipynb**
  This script handles the data engineering pipeline. It cleans raw data, filters out users and items with low interaction counts (k-core filtering), and performs textual preprocessing (tokenization, stop-word removal, lemmatization). Finally, it saves the processed data in Parquet format.
- **Recommender.ipynb**
  Implements a Collaborative Filtering model using Alternating Least Squares (ALS) for implicit feedback. It trains the ALS model (using the implicit library) and it evaluates the performance by calculating the root mean square error.
- **Sentiment-LR.ipynb**
  Trains a Logistic Regression classifier to predict review sentiment. It uses TfidfVectorizer to convert review text into numerical vectors.
- **Sentiment-SVM.ipynb**
  Implements a Support Vector Machine (LinearSVC) for sentiment analysis. This script is used to train and compare the performance of a SVC classifier against the Logistic Regression baseline to determine the most effective approach for text classification.

Github link: https://github.com/SamuelLazurca/BDA_Project_2025-2026

9. Conclusion

This project successfully demonstrated the application of Big Data technologies to the domain of e-commerce analytics. By leveraging Apache Spark, we processed the massive Amazon 5-core dataset (~75 million reviews).

1. **Exploratory Analysis:** We confirmed that Amazon reviews exhibit a "long-tail" product distribution and a "J-shaped" rating distribution. We identified a "positivity bias" in review length and a strong temporal growth pattern driven by platform adoption.
2. **Sentiment Analysis:** We showed that machine learning models can accurately classify sentiment from text alone, with **LinearSVC** achieving **90.5%** accuracy, slightly outperforming Logistic Regression. This proves that text provides a robust proxy for numerical ratings.
3. **Recommendation:** We built a scalable **ALS** recommender system. Despite extreme sparsity (99.93%), the model achieved an RMSE of **1.03**. While effective as a baseline, the "State of the Art" review suggests that overcoming this performance plateau requires integrating the semantic signals (from LLMs) with the collaborative signals (from GNNs/ALS).

The transition from single-node processing to distributed Spark clusters was not merely a technical convenience but a fundamental requirement for analyzing data of this volume.

10. Acknowledgements

Data processing and analysis were carried out with the support of project SMIS 124759 – RaaS-IS (Research as a Service Iași), funded through the Operational Programme Competitiveness.

Works Cited

1. Ni, J., Li, J., & McAuley, J. (2019, November). Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP) (pp. 188-197).
2. Jianmo Ni, Jiacheng Li, and Julian McAuley, "Amazon Review Data (2018)," UCSD, 2018. [Online]. Available: https://nijianmo.github.io/amazon/index.html#subsets .
3. He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020, July). Lightgcn: Simplifying and powering graph convolution network for recommendation. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval (pp. 639-648).
4. Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., & Jiang, P. (2019, November). BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In Proceedings of the 28th ACM international conference on information and knowledge management (pp. 1441-1450).
5. Kim, S., Kang, H., Choi, S., Kim, D., Yang, M., & Park, C. (2024, August). Large language models meet collaborative filtering: An efficient all-round llm-based recommender system. In Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 1395-1406).
6. John, A., Aidoo, T., Behmanush, H., Gunduz, I. B., Shrestha, H., Rahman, M. R., & Maaß, W. (2024). LLMRS: Unlocking potentials of LLM-based recommender systems for software purchase. arXiv preprint arXiv:2401.06676.
7. Hu, Y., Koren, Y., & Volinsky, C. (2008, December). Collaborative filtering for implicit feedback datasets. In 2008 Eighth IEEE international conference on data mining (pp. 263-272). Ieee.