University for the Creative Arts

BERLIN SCHOOL OF BUSINESS & INNOVATION

Essay / Assignment Title:

Detecting Fake Reviews and Sentiment Classification in Amazon's Industrial & Scientific Category: An End-to-End Big Data Pipeline Approach.

Programme title:

MSc Data Analytics

Name:

HERNAN ENRIQUE GUZMAN MORATTO

Year:

2025

CONTENTS

Statement of compliance with academic ethics and the avoidance of plagiarism

I honestly declare that this dissertation is entirely my own work and none of its part has been copied from printed or electronic sources, translated from foreign sources and reproduced from essays of other researchers or students. Wherever I have been based on ideas or other people texts I clearly declare it through the good use of references following academic ethics.

(In the case that is proved that part of the essay does not constitute an original work, but a copy of an already published essay or from another source, the student will be expelled permanently from the program).

Name and Surname (Capital letters):

.....HERNAN ENRIQUE GUZMAN MORATTO.....

Date: 2025/06/20

INTRODUCTION

In the digital commerce era, user-generated content plays a crucial role in consumer behavior and business models. The research is focused on developing a scalable Big Data pipeline for Amazon review processing and analysis within the Industrial & Scientific category.

The primary objectives are sentiment classification to determine the tone of reviews and fake review detection based on metadata and textual features. These efforts aim to enhance user trust, improve recommendation systems, and automate content moderation.

Following the CRISP-DM methodology, the approach employs a big data stack, including Google Cloud Platform, HDFS, Apache Hive, and PySpark. Data ingestion, cleaning, feature engineering, and model development were performed in a distributed environment.

Correlation analysis helped to identify relevant features to implement in the classification models Random Forest Classifier, Logistic Regression, and Decision Tree. The results indicate that combining structured metadata with textual data improves model performance, with the fake review detection model achieving over 94% accuracy. This project showcases modern big data tools and offers a replicable framework for e-commerce analytics.

All scripts used for the development of this project can be found in GitHub:

https://github.com/HerGuzmanMoratto/amazon-bigdata-pipeline

METHODOLOGY.

This project follows a structured methodology created to implement a scalable end-to-end Big-Data Pipeline, from the data ingestion, transformation, analysis, and machine learning application, and replicability of customer review data from the Industrial & Scientifics category of Amazon Reviews Dataset (Hou, 2024). The methodology to implement aligns with CRISP-DM process model for data mining covering six key phases: business understanding, data understanding, data preparation, modeling, evaluation, and deployment (Schröer, Kruse and Gómez, 2021).

1. Data acquisition and storage: the raw .jsonl review and meta datasets were retrieved from a public Amazon Reviews repository and uploaded to Google Cloud Storage (GCS), to be later used in Hadoop Distributed File System (HDFS) for distributed processing.

2. Data preparation and processing: PySpark was used to parse, clean, and integrate review data with meta data. Deep transformation was performed to the files to standardize the columns, handle nulls, clean nested columns with weird values, and drop unnecessary columns. The integrated dataset was saved in GCS in CVS format.

3. Exploratory Data Analysis (EDA): EDA was executed with both Hive and PySpark to compare their performance in terms of speed and complexity. This dual approach served as a basis for performance benchmarking and deeper insights into helpful_votes distribution, price categorization, ratings distribution, etc.

4. Feature engineering: features were enhanced for two supervised learning tasks: fake review and sentiment classification. A correlation matrix was computed to analyze multicollinearity and choose the most relevant features.

5. Model training and evaluation: using PySpark MLlib, classification models (Random Forest Classifier, Logistic Regression, and Decision Tree) were trained and evaluated on the binary classification problems mentioned. Models were later evaluated using Accuracy, F1-Score, AUC, weighted precision and recall.

6. Infrastructure and tools: the full stack includes Google Could Platform, Apache Spark, Apache Hive, Hadoop HDFS, and PySpark, which were chosen due to their scalability, flexibility, integration support and capabilities for machine learning workflows (Zaharia *et al.*, 2016).

By following this methodology, reproducibility and scalability for real-world applications to support business use cases are ensured (Al Gharbi *et al.*, 2020).

PHASE 1. PROBLEM DEFINITION AND DATASET SELECTION.

The first phase of this project focuses on building an end-to-end data pipeline to analyze customer reviews on platforms that rely on user-generated content. In this context, online platforms like Amazon, due to volume, velocity and variety (Agrawal *et al.*, 2011).

The idea is to cover two key aspects of machine learning tasks:

I. Sentiment Classification to automatically determine if a review expresses a positive or negative sentiment.

II. Fake Review Detection to label potential suspicious or fake reviews based on both metadata and text features.

These aspects align real-world business needs, like improving customers trustworthiness, filtering out spam or fake content, and better customer understanding on scale (Choo, Yu and Chi, 2017). Using classification systems for both sentiment and review detection, business can improve the quality of product feedback shown to end users, automate quality content review filtering, and improve recommendation systems delivering trustworthy content (Jindal and Liu, 2008).

1.1. DATASET OVERVIEW.

The Industrial & Scientific dataset contained reviews and meta data in JSON Lines (.jsonl) with approx. 5.1M records. Header's validation was performed with Python in Jupiter Notebook to ensure that the dataset met the minimum requirements before merging and cleaning it down to approximately 3.6M records for ML implementation.

The columns used to develop this project were:

- parent_asin: for files joining.
- Text: Text of the review
- Rating: measured from 1 to 5

- helpful_vote: Number of helpful votes
- verified_purchase: Whether the purchase was verified
- price: Product price
- Additional metadata from the matching meta file (for future use).

1.2. PROBLEM FORMULATION.

As explained in the previous subsection, the scope focuses on two supervised tasks:

- Sentiment Classification:
  - ~ Label = sentiment_label (0 for negative and 1 for positive)
  - ~ Based on ratings (1-2 for negative, 4-5 for positive, excluding 3 for being neutral)
- Fake Review detection.:
  - ~ Label = is_fake (0 for Authentic and 1 for fake)
  - ~ Heuristic Used: Reviews with verified_purchase = 0 and helpful_vote = 0 are labeled as potentially fake.
- Features: price, verified_purchase, and helpful vote from metadata along with text features (TF-IDF) from text column in the original review dataset.

These problems were selected because they reflect real-world and relevant scenarios in eCommerce analytics for demonstrating an end-to-end pipeline (Pang and Lee, 2006).

1.3. TECHNOLOGIES TO APPLY.

The project relies on modern big-data technologies to build an end-to-end data pipeline, from data ingestion to machine learning modeling. The following tools and platforms were applied based on their scalability, flexibility, and compatibility with structured and unstructured data:

- Google Cloud Platform (GCP): infrastructure management, data storage (Google Cloud Storage), and deploying a Dataproc cluster to run distributed Hadoop, Spark, and Hive workloads.

- Apache Hadoop (HDFS): distributed storage to upload and process the dataset.
- Apache Hive: for SQL-like querying for EDA and data validation.
- Apache Spark with PySpark: core processing tool for data engineering, feature correlation, and machine learning pipeline development.
- Spark Mllib: machine learning models implementation.
- Python and Jupiter notebook: local data exploration and schema validation before uploading into pipeline.
- GitHub: project documentation to store scripts used to develop this project.

PHASE 2. DATA PREPROCESSING WITH SPARK AND HIVE.

Phase 2 involved transforming raw Industrial & Scientific review and meta data into a clean and enriched dataset for analysis and machine learning tasks. A distributed data processing was applied using PySpark and Hive, focusing on:

- Wrangling and cleaning large JSONL files
- Joining the review dataset with the meta dataset in one file.
- Storing outputs in GCS and HDFS.
- Performing statistical summaries and query analysis using Hive.
- Comparing Hive and PySpark on performance and use case applicability.

2.1. DATA INGESTION AND PREPARATION

Two key datasets were used as it was mentioned in Phase 1:

- Industrial_and_Scientific.jsonl containing user review data.
- Industrial_and_Scientific_metadata.jsonl with product data such as prices and categories.

Both files were manually uploaded into a bucket in GCS and then copied into a local directory using SSH interface in GCS. From GCS to local directory, and finally into HDFS folder as shown in Figure 1, screenshot of the data ingestion script.

```
herguzman12@cluster-4f69-m:~$ gsutil cp gs://assignment_pipeline/Industrial_and_Scientific.jsonl .
Copying gs://assignment_pipeline/Industrial_and_Scientific.jsonl...
\ [1 files][  2.2 GiB/  2.2 GiB]   46.7 MiB/s
Operation completed over 1 objects/2.2 GiB.
herguzman12@cluster-4f69-m:~$ gsutil cp gs://assignment_pipeline/meta_Industrial_and_Scientific.jsonl .
Copying gs://assignment_pipeline/meta_Industrial_and_Scientific.jsonl...
| [1 files][  1.0 GiB/  1.0 GiB]   45.1 MiB/s
Operation completed over 1 objects/1.0 GiB.
herguzman12@cluster-4f69-m:~$ ls -lh Industrial_and_Scientific.jsonl
-rw-rw-r-- 1 herguzman12 herguzman12 2.2G Jun 16 18:19 Industrial_and_Scientific.jsonl
herguzman12@cluster-4f69-m:~$ ls -lh meta_Industrial_and_Scientific.jsonl
-rw-rw-r-- 1 herguzman12 herguzman12 1.1G Jun 16 18:20 meta_Industrial_and_Scientific.jsonl
herguzman12@cluster-4f69-m:~$ hdfs dfs -mkdir -p /data/reviews
herguzman12@cluster-4f69-m:~$ hdfs dfs -put Industrial_and_Scientific.jsonl /data/reviews/
herguzman12@cluster-4f69-m:~$ hdfs dfs -put meta_Industrial_and_Scientific.jsonl /data/reviews/
herguzman12@cluster-4f69-m:~$
```

Figure 1. Data Ingestion Process Through SSH. Author's Property.

2.2. SCHEMA INSPECTION AND CLEANING WITH PYSPARK

In this phase the schema was read and explored using PySpark. Figure 2 displays the schema for the review dataset and shows the problems faced with the meta dataset, as it required some hard cleaning. It contained wrong format, images, URL, and other types of content, so it was uploaded as text from HDFS and then preprocessed by replacing values from price column and converting them into float for proper analysis.

```
>>> df = spark.read.json("hdfs:///data/reviews/Industrial_and_Scientific.jsonl")
>>> raw_meta = spark.read.text("hdfs:///data/reviews/meta_Industrial_and_Scientific.jsonl")
>>> df.printSchema()
root
 |-- asin: string (nullable = true)
 |-- helpful_vote: long (nullable = true)
 |-- images: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- attachment_type: string (nullable = true)
 |    |    |-- large_image_url: string (nullable = true)
 |    |    |-- medium_image_url: string (nullable = true)
 |    |    |-- small_image_url: string (nullable = true)
 |-- parent_asin: string (nullable = true)
 |-- rating: double (nullable = true)
 |-- text: string (nullable = true)
 |-- timestamp: long (nullable = true)
 |-- title: string (nullable = true)
 |-- user_id: string (nullable = true)
 |-- verified_purchase: boolean (nullable = true)

>>> raw_meta.printSchema()
root
 |-- value: string (nullable = true)

>>> raw_meta = spark.read.json("hdfs:///data/reviews/meta_Industrial_and_Scientific.jsonl")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/spark/python/pyspark/sql/readwriter.py", line 284, in json
    return self._df(self._jreader.json(self._spark._sc._jvm.PythonUtils.toSeq(path)))
  File "/usr/lib/spark/python/lib/py4j-0.10.9.5-src.zip/py4j/java_gateway.py", line 1321, in __call__
  File "/usr/lib/spark/python/pyspark/sql/utils.py", line 196, in deco
    raise converted from None
pyspark.sql.utils.AnalysisException: Found duplicate column(s) in the data schema: `assembly required`, `
region of origin`, `display size`, `display technology`, `finish type black`, `finish type zinc plated`,
ial chrome`, `material galvanized steel`, `material polymer`, `material polypropylene`, `material polyure
```

Figure 2. Review Dataset Schema. Author's Property.

Figure 3 shows the dataset schema after proper handling. The price column had string formatting like "$29.99", so it was converted to String and cast to float. After doing so, the schema was now readable and ready to join the two datasets. Additional columns were dropped, such as "title" and "images".

```
>>> raw_meta = spark.read.text("hdfs:///data/reviews/meta_Industrial_and_Scientific.jsonl")
>>> from pyspark.sql.functions import from_json, col, regexp_replace
>>> from pyspark.sql.types import *
>>> schema_meta = StructType([
...     StructField("main_category", StringType(), True),
...     StructField("title", StringType(), True),
...     StructField("average_rating", DoubleType(), True),
...     StructField("rating_number", IntegerType(), True),
...     StructField("features", StringType(), True),
...     StructField("description", StringType(), True),
...     StructField("price",  StringType(), True),
...     StructField("images", StringType(), True),
...     StructField("videos", StringType(), True),
...     StructField("store", StringType(), True),
...     StructField("categories", StringType(), True),
...     StructField("details", StringType(), True),
...     StructField("parent_asin", StringType(), True),
...     StructField("bought_together", StringType(), True)
... ])
>>> metadata = raw_meta.select(from_json(col("value"), schema_meta).alias("data")).select("data.*")
>>> df2 = metadata.withColumn("price", regexp_replace("price", "[$,]", "").cast("float"))
>>> df2.printSchema()
root
 |-- main_category: string (nullable = true)
 |-- title: string (nullable = true)
 |-- average_rating: double (nullable = true)
 |-- rating_number: integer (nullable = true)
 |-- features: string (nullable = true)
 |-- description: string (nullable = true)
 |-- price: float (nullable = true)
 |-- images: string (nullable = true)
 |-- videos: string (nullable = true)
 |-- store: string (nullable = true)
 |-- categories: string (nullable = true)
 |-- details: string (nullable = true)
 |-- parent_asin: string (nullable = true)
 |-- bought_together: string (nullable = true)
```

Figure 3. Meta Data Preprocessing for Data Integration. Author's Property.

2.3. DATA INTEGRATION AND NEW DATASET STORAGE.

Cleaned datasets were joined using left join on "parent_asin" to retain all review records. Although "asin" was ideal, its absence in meta dataset required fallback on "parent_asin", which may introduce inaccuracies, but the decision of proceeding was taken as this project is for educational purposes.

This approach served as a basis to combine the review data with corresponding metadata such as main_category, average_rating, features, and price. Figure 4 displays the cleaned dataset schema after this process. The cleaned and merged dataset had 5.1M records and it was saved as csv file into the GCS to perform querying with HIVE, PysPark, and EDA for ML modeling and further analysis.

```
>>> ds_clean = df.join(df2_clean, on="parent_asin", how="left")
>>> ds_clean.printSchema()
root
 |-- parent_asin: string (nullable = true)
 |-- asin: string (nullable = true)
 |-- helpful_vote: long (nullable = true)
 |-- rating: double (nullable = true)
 |-- text: string (nullable = true)
 |-- timestamp: long (nullable = true)
 |-- title: string (nullable = true)
 |-- user_id: string (nullable = true)
 |-- verified_purchase: boolean (nullable = true)
 |-- main_category: string (nullable = true)
 |-- average_rating: double (nullable = true)
 |-- rating_number: integer (nullable = true)
 |-- features: string (nullable = true)
 |-- description: string (nullable = true)
 |-- price: float (nullable = true)
 |-- videos: string (nullable = true)
 |-- store: string (nullable = true)
 |-- categories: string (nullable = true)
 |-- details: string (nullable = true)
 |-- bought_together: string (nullable = true)

>>> ds_clean.coalesce(1).write.option("header", "true").option("delimiter", ",").mode("overwrite").csv("gs://assi
gnment_pipeline/Industrial_and_Scientific_clean")
>>> ds_clean.count()
5183005
```

Figure 4. Data Integration and Storage. Author's Property.

2.4. TABLE SETUP AND INITIAL EDA QUERYING WITH HIVE.

The processed data was registered as an external HIVE table for SQL Analysis. Queries to inspect the dataset from the basic such as count all, count verified purchases, check average rating, to advanced like checking the helpful vote distribution and checking a final preview of summary statistics. The helpful_vote field showed an extremely right-skewed distribution, with a max of 5611 and a minimum of 10 after querying for insights. Therefore, logarithmic binning in Hive was applied to capture the long-tail behavior more accurately, showing it is highly imbalanced. Figure 5 displays the helpful_vote distribution histogram, reflecting a typical long-tail distribution where attention concentrates on a few highly visible items.

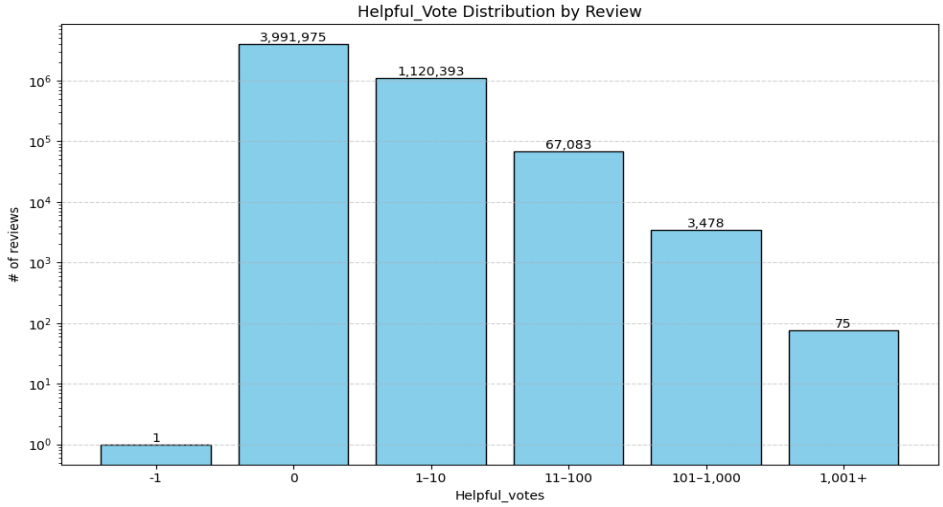

Figure 5. Helpful_Vote Distribution. Author's Property.

Figure 6 shows the snapshot of table creation and the result of the initial querying, and Figure 7 shows the results of the summary statistics obtained from querying. A set of 10 queries were ran to get a detailed overview of the dataset, filtering out values and grouping by category and price buckets, these queries will be crucial for dataset enhancement.

```
Hive Session ID = aed91809-0810-4b2c-9225-ec1fe05f97f2
hive> CREATE EXTERNAL TABLE IF NOT EXISTS industrial_reviews (
    >   parent_asin STRING,
    >   asin STRING,
    >   helpful_vote BIGINT,
    >   rating DOUBLE,
    >   text STRING,
    >   `timestamp` BIGINT,
    >   title STRING,
    >   user_id STRING,
    >   verified_purchase BOOLEAN,
    >   main_category STRING,
    >   average_rating DOUBLE,
    >   rating_number INT,
    >   features STRING,
    >   description STRING,
    >   price DOUBLE,
    >   videos STRING,
    >   store STRING,
    >   categories STRING,
    >   details STRING,
    >   bought_together STRING
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE
    > LOCATION 'gs://assignment_pipeline/Industrial_and_Scientific_clean/'
    > TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 2.702 seconds
hive> SELECT COUNT(*) AS total_reviews FROM industrial_reviews;
Query ID = herguzman12_20250616204316_92b1a493-a9d9-45db-88f8-06f9aa246617
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1750005256541_0007)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 58.06 s
--------------------------------------------------------------------------------
OK
5183005
```

Figure 6. Table Creation and Initial Check Using Hive. Author's Property.

```
    >   main_category,
    >   COUNT(*) AS total_valid_reviews,
    >   ROUND(AVG(rating), 2) AS avg_rating,
    >   MAX(helpful_vote) AS max_votes,
    >   ROUND(AVG(price), 2) AS avg_price
    > FROM industrial_reviews
    > WHERE main_category IS NOT NULL
    >   AND main_category != ''
    >   AND main_category NOT LIKE 'true'
    >   AND rating IS NOT NULL
    >   AND helpful_vote IS NOT NULL
    >   AND price IS NOT NULL
    >   AND price BETWEEN 1 AND 10000 -- realistic range
    > GROUP BY main_category
    > HAVING COUNT(*) > 500
    > ORDER BY total_valid_reviews DESC;
Query ID = herguzman12_20250616210617_7cc17959-5d3f-4265-8f20-7ae6f7c00997
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1750005256541_0008)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ...... container     SUCCEEDED     31         31        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 109.86 s
--------------------------------------------------------------------------------
OK
Industrial & Scientific 47916   4.35    394     88.1
Tools & Home Improvement        13102   4.32    216     77.35
Amazon Home     11437   4.32    160     81.93
Health & Personal Care  4074    4.35    63      23.5
Automotive      2871    4.41    92      168.01
Office Products 1724    3.99    145     53.64
All Beauty      992     3.88    44      19.11
Five Stars      902     4.98    26      1194.54
All Electronics 588     4.18    6       99.38
Sports & Outdoors       559     4.55    37      28.82
AMAZON FASHION  548     4.18    26      28.94
Time taken: 123.297 seconds, Fetched: 11 row(s)
```

Figure 7. Statistics Overview Query with Hive. Author's Property.

2.5. PYSPARK.

Identical analyses were performed with PySpark for speed comparison and simplicity. PySpark's outperformed Hive in memory processing and proved more suitability for ML enhancement. As a result of this, the dataset and feature enhancing steps, which take place in phase 3, were completed by using PySpark. Figure 8 illustrates the results of the same initial queries, but instead of creating a table, it checks the schema and performs the initial count.

```
>>> start = time.time()
>>> ds_clean.printSchema()
root
 |-- parent_asin: string (nullable = true)
 |-- asin: string (nullable = true)
 |-- helpful_vote: integer (nullable = true)
 |-- rating: double (nullable = true)
 |-- text: string (nullable = true)
 |-- timestamp: long (nullable = true)
 |-- title: string (nullable = true)
 |-- user_id: string (nullable = true)
 |-- verified_purchase: boolean (nullable = true)
 |-- main_category: string (nullable = true)
 |-- average_rating: double (nullable = true)
 |-- rating_number: integer (nullable = true)
 |-- features: string (nullable = true)
 |-- description: string (nullable = true)
 |-- price: double (nullable = true)
 |-- videos: string (nullable = true)
 |-- store: string (nullable = true)
 |-- categories: string (nullable = true)
 |-- details: string (nullable = true)
 |-- bought_together: string (nullable = true)

>>> end = time.time()
>>> print(f"Time taken: {round(end - start, 2)} seconds")
Time taken: 0.01 seconds
>>> start = time.time()
>>> ds_clean.count()
5183005
>>> end = time.time()
>>> print(f"Time taken: {round(end - start, 2)} seconds")
Time taken: 53.61 seconds
```

Figure 8. Schema Overview and Initial Count Using PySpark. Author's Property.

2.6. PYSPARK VS HIVE COMPARISON.

Both tools were compared in terms of performance, flexibility, use case usability, and integration complexity, speed and expressiveness.

PySpark offers better performance and flexibility due to its processing memory and APIs with Python, which is ideal for machine learning purposes and complex transformations. Hive can be used when simplicity and SQL ad-hoc like queries, but not when handling data in which the schema is not known beforehand or the data is nasty, containing things such as images, links, audios, etc. This comparison is shown in Table 1, in which it is shown that for handling unstructured complex and big datasets, PySpark outperforms Hive. A fully cleaned, merged, and feature-enhanced datasets with data available in GCS, funding solid basis for PySpark-based EDA and ML tasks.

Table 1. PySpark VS Hive Comparison. Author's property.

Metric	PySpark	Hive
Overall Performance	Faster in memory computation	Slower due to MapReduce job execution
Flexibility	Rich with Python APIs for feature engineering & modeling	Static SQL-based transformations
Scalability	Scales efficiently across large clusters with Spark engine	Scales via Hadoop, but less efficient unstructured data.
expressiveness	More flexible as it's Python alike	SQL-like, more verbose for cleaning
Use Case Suitability	ML workflows, streaming, complex joins	Ad-hoc summary stats, batch SQL queries
Integration Complexity	Requires SparkSession, column expressions	Simple SQL syntax, quick to write
Debugging & Logging	Easier logging via PySpark logs, easier local testing	Errors can be buried in MapReduce logs

PHASE 3. DATASET ENHANCEMENT FOR MACHINE LEARNING.

After the successful completion of phase 2 data ingestion, preprocessing and overview, this part of the project focuses on getting a deeper EDA and performing initial feature diagnostics to confirm which columns of the dataset are meaningful for feature-engineering to train and test the model in the next phase. Phase 3 served as the basis for transforming cleaned CSV raw data into a structured machine learning dataset.

3.1. EDA USING PYSPARK.

The cleaned and joined dataset obtained from phase two was initially checked and it contained over 5 million records. Hive and PySpark initial analysis pointed out some outliers which were considered in this part of the project while designing the final dataset. Furthermore, null values, unformatted cells and empty values were filtered out, narrowing the dataset down to approximately 3.6M records.

Moreover, the aggregations performed to inspect product category frequency, price buckets, and review verification status led to the design of labels “sentimental_label” and “is_fake”, key features for the ML modeling as they will be the target labels. Figure 9 shows the snapshot of this part of the code in SSH environment.

```
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import col, when
>>> from pyspark.ml.feature import VectorAssembler
>>> from pyspark.ml.stat import Correlation
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> import pandas as pd
>>> import numpy as np
>>> cleaned = ds_clean.filter(
...     (col("main_category").isNotNull()) &
...     (col("main_category") != "") &
...     (~col("main_category").like("true")) &
...     (col("rating").isNotNull()) &
...     (col("helpful_vote").isNotNull()) &
...     (col("price").isNotNull()) &
...     (col("price").between(1, 10000))
... )
>>>
>>> ds_labeled = cleaned \
...     .withColumn("helpful_vote", col("helpful_vote").cast("int")) \
...     .withColumn("price", col("price").cast("double")) \
...     .withColumn("rating", col("rating").cast("double")) \
...     .withColumn("verified_purchase", col("verified_purchase").cast("int")) \
...     .withColumn("sentiment_label", when(col("rating") >= 4, 1).when(col("rating") <= 2, 0)) \
...     .filter(col("rating").isNotNull() & col("sentiment_label").isNotNull()) \
...     .withColumn("is_fake", when((col("verified_purchase") == 0) & (col("helpful_vote") == 0), 1).otherwise(0))
>>> ds_labeled.count()
3620007
>>>
```

Figure 9. Dataset Overview After Second Preprocessing. Author's Property.

3.2. CORRELATION ANALYSIS FOR FEATURE SELECTION.

A correlation analysis was done to evaluate multicollinearity and possible predictive relationships between dataset feature using PySpark's .corr() function on numeric columns "price", "helpful_vote", "verified_purchase", "sentiment_label", "is_fake", "rating", "average_rating", and "rating_number".

These columns were converted into vectors using VectorAssembler fuction and then analyzed using Pearson correlation matrix. The investigation demonstrated that "sentiment_label" is highly positive correlated with rating, which confirms that the label was created properly. "is_fake" has a strong negative correlation with "verified_purchase", expected as most fakes are unverified. These results are shown in Figure 10, displaying the SSH script and output of this part. The correlation matrix was stored in GCS as txt format.

```
>>> numeric_cols = ["price", "helpful_vote", "verified_purchase", "sentiment_label", "is_fake", "rating", "average_rating", "rating_number"]
>>> ds_numeric = ds_labeled.select([col(c).cast("double") for c in numeric_cols])
>>> from pyspark.sql.functions import col, when, sum as spark_sum
>>> null_counts = ds_labeled.select([
...     spark_sum(when(col(c).isNull(), 1).otherwise(0)).alias(f"{c}_nulls")
...     for c in numeric_cols
... ])
>>> null_counts.show()
+-----------+-----------------+---------------------+-------------------+------------+-----------+-------------------+-------------------+
|price_nulls|helpful_vote_nulls|verified_purchase_nulls|sentiment_label_nulls|is_fake_nulls|rating_nulls|average_rating_nulls|rating_number_nulls|
+-----------+-----------------+---------------------+-------------------+------------+-----------+-------------------+-------------------+
|          0|                0|                    0|                  0|           0|          0|                  0|                  0|
+-----------+-----------------+---------------------+-------------------+------------+-----------+-------------------+-------------------+


>>> vec_assembler = VectorAssembler(inputCols=numeric_cols, outputCol="features_vector")
>>> ds_vector = vec_assembler.transform(ds_labeled).select("features_vector")
>>> corr_matrix = Correlation.corr(ds_vector, "features_vector", method="pearson").head()[0]
>>> corr_array = corr_matrix.toArray()
>>> feature_names = numeric_cols
>>> corr_df = pd.DataFrame(corr_array, index=feature_names, columns=feature_names)
>>> print(corr_df)
                    price  helpful_vote  verified_purchase  ...    rating  average_rating  rating_number
price            1.000000      0.024932          -0.050321  ... -0.010902       -0.084184      -0.038756
helpful_vote     0.024932      1.000000          -0.001991  ... -0.026870       -0.011779       0.002589
verified_purchase -0.050321    -0.001991           1.000000  ...  0.010035        0.029570       0.034520
sentiment_label  -0.008715     -0.024699           0.003818  ...  0.964516        0.245807      -0.054227
is_fake           0.019638     -0.017248          -0.876106  ...  0.011763       -0.010517      -0.027876
rating           -0.010902     -0.026870           0.010035  ...  1.000000        0.258504      -0.052696
average_rating   -0.084184     -0.011779           0.029570  ...  0.258504        1.000000       0.041619
rating_number    -0.038756      0.002589           0.034520  ... -0.052696        0.041619       1.000000

[8 rows x 8 columns]
```

Figure 10. SSH Script and Correlation Matrix. Author's property.

3.3. FINAL DATASET STORAGE.

As the final step of phase, the dataset with the final columns and their correct format were saved in the GCS as csv file with the columns: "asin", "price", "helpful_vote", "verified_purchase", "rating", "sentiment_label", "is_fake", "average_rating", "rating_number", "main_category", and "text". Figure 11 displays a snapshot of the final part of the code used to save the featured engineered dataset into the GCS.

```
>>> final_cols = [
...     "asin",                 # Product ID (for grouping/analysis)
...     "price",                # Numeric feature
...     "helpful_vote",         # Numeric feature
...     "verified_purchase",    # Categorical/binary feature
...     "rating",               # user rating
...     "sentiment_label",      # Main classification label (positive vs negative)
...     "is_fake",              # 2nd label (fake review detection)
...     "average_rating",       # Metadata feature
...     "rating_number",        # number of ratings
...     "main_category",        # Useful for segmentation
...     "text"                  # Raw text
... ]
>>> final_ds = ds_labeled.select(final_cols)
>>> final_ds.coalesce(1).write \
...     .option("header", True) \
...     .mode("overwrite") \
...     .csv("gs://assignment_pipeline/feature_engineered_ds/")
```

Figure 11. Dataset Enhanced Final Columns. Author's property.

This section of the study reinforced the choice to proceed with analysis and modeling using PySpark due to its better performance in these tasks, complete integration with ML pipeline and its capabilities to work structured and unstructured data. After completing EDA and correlation analysis, the dataset is ready for modeling purposes.

PHASE 4. ADVANCED ANALYTICS AND MACHINE LEARNING.

This phase of the project covers the implementation of machine learning algorithms to analyze the business aspects of the dataset based on the labeled columns selected in phase 3.

4.1. FAKE REVIEW DETECTION WITH PYSPARK.

The first part of the analysis focuses on developing a binary classification model to detect fake reviews in the selected dataset using PySpark MLib Pipeline. The aim was to classify reviews into is_fake = 1 when it was suspicious or fake and is_fake = 0 when it was a genuine, using features from both datasets to get the text and numerical data.

4.1.1. Data preparation.

From the enhanced dataset created in phase 3, the numerical features: price, helpful_vote, verified_purchase were chosen and column text for text analysis. The target label is_fake. Out of the total dataset, a random 10% sample out of 3.6M records were selected to optimize training time and memory resources while keeping the data characteristics.

4.1.2. Feature engineering.

A mixed feature pipeline was crafted by combining structured metadata and text features:

- Text pipeline: Tokenizer to split the dataset into tokens. StopWordRemover to remove common English words such as “the”, “a”, “and”, etc. And HashingTF - IDF (TF-IDF) vectorizing was applied to convert them into numerical features.
- Metadata pipeline: Vector assembler was used to combine price, helpful_vote, and verified_purchase into a meta_features vector.
- Final assembler: combines meta_features and text_features into a final vector to input into the model.

4.1.3. Model preparation and training.

The Random Forest Classifier algorithm with 50 trees was chosen due to its ability to deal with robust, mixed and unbalanced datasets. The full pipeline included the preprocessing stages (tokenizer, remover, TF-IDT, assemblers and the classifier, to make it reproducible. Figure 12 shows the snapshot of the PySpark script for the steps performed up to here.

4.1.4. Model evaluation and storage.

The model was evaluated on a 20% test split using binary and multiclass evaluators, with satisfactory results as shown in Figure 12. The metrics indicate a high-performance model, especially in terms of discriminative power (AUC ROC), having an accuracy and F1- Score over 94%. The model and evaluation results were stored in GCS for reproducibility and deployment.

```
>>> from pyspark.ml.feature import VectorAssembler
>>> from pyspark.ml.classification import RandomForestClassifier
>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator, BinaryClassificationEvaluator
>>> from pyspark.ml import Pipeline
>>> from pyspark.sql import SparkSession
>>> from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, IDF
>>> spark = SparkSession.builder.appName("FakeReviewAnalysis").getOrCreate()
25/06/18 13:33:17 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
>>> ds_ml = spark.read.option("header", True).option("inferSchema", True).csv("gs://assignment_pipeline/outputs/feature_engineered_ds/")
>>> selected_cols2 = ["price", "helpful_vote", "verified_purchase", "text", "is_fake"]
>>> ds_f = ds_ml.select(*selected_cols2).na.drop()
>>> ds_f_sampled = ds_f.sample(withReplacement=False, fraction=0.1, seed=42)
>>> tokenizer = Tokenizer(inputCol="text", outputCol="tokens")
>>> remover = StopWordsRemover(inputCol="tokens", outputCol="filtered")
>>> hashingTF = HashingTF(inputCol="filtered", outputCol="raw_features", numFeatures=1000)
>>> idf = IDF(inputCol="raw_features", outputCol="text_features")
>>> meta_assembler = VectorAssembler(
...     inputCols=["price", "helpful_vote", "verified_purchase"],
...     outputCol="meta_features"
... )
>>> final_assembler = VectorAssembler(
...     inputCols=["meta_features", "text_features"],
...     outputCol="features"
... )
>>> rf = RandomForestClassifier(labelCol="is_fake", featuresCol="features", numTrees=50)
>>> pipeline = Pipeline(stages=[tokenizer, remover, hashingTF, idf, meta_assembler, final_assembler, rf])
>>> train_data, test_data = ds_f_sampled.randomSplit([0.8, 0.2], seed=42)
>>> model = pipeline.fit(train_data)
>>> predictions = model.transform(test_data)
>>> binary_eval = BinaryClassificationEvaluator(labelCol="is_fake")
>>> multi_eval = MulticlassClassificationEvaluator(labelCol="is_fake", predictionCol="prediction")
>>> print("Accuracy:", multi_eval.setMetricName("accuracy").evaluate(predictions))
Accuracy: 0.9628506968954588
>>> print("F1 Score:", multi_eval.setMetricName("f1").evaluate(predictions))
F1 Score: 0.9446275928967887
>>> print("AUC:", binary_eval.setMetricName("areaUnderROC").evaluate(predictions))
AUC: 0.9940501158840712
>>> print("PR AUC:", binary_eval.setMetricName("areaUnderPR").evaluate(predictions))
PR AUC: 0.7454572107981
```

Figure 12. Random Forest Classifier Implementation. Author's Property.

4.2. SENTIMENT CLASSIFICATION ANALYSIS WITH PYSPARK

The second part of the analysis aimed to identify if a customer review expressed a positive or negative sentiment. This binary classification approach helps in modeling real-world eCommerce applications like automated tagging of reviews, customer satisfaction analysis, and recommendations based on trustworthy.

4.2.1. Data preparation.

Opposite to the approach used in the fake review section, there was no need to reduce the dataset to train and test the model. The numerical columns price, helpful_vote, and verified purchase were chosen from the enhanced model to develop the analysis in addition to the label column called sentiment_label was created during data engineering phase, which will be the target column.

4.2.2. Feature engineering.

Only structured metadata was used for this model, therefore the columns mentioned in data preparation were used to feed Vector Assembler. These were chosen as they can be associated with the tone of a review — e.g., verified purchases are likely more positive, while unhelpful votes could suggest dissatisfaction.

4.2.3. Model preparation and training.

Three ML models were tested—Logistic Regression, Decision Tree, and Random Forest—to determine the most effective classifier using structured metadata. Each model was implemented using the features assembled in the previous step, and train with 80% of the data and tested on the remaining 20%. Later, predictions were generated based on testing data. Figure 13 shows the script in SSH for the whole process.

4.2.4. Model evaluation and storage.

The models were evaluated using Accuracy, F1 Score, Weighted Precision, Weighted Recall, Area Under ROC Curve (AUC), Area Under Precision-Recall Curve (PR AUC).

The results showed that all models provided similar scores on measures, as is illustrated in Figure 14. Random Forest performance was slightly better on PR AUC and AUC, indicating that it can deal with datasets with imbalanced classes. However, all models showed strong performance in terms of Accuracy, F1 Score, weighted precision and recall, which indicates good generalization and a well-structured schema for training and testing data. The results were stored in a local file and uploaded to GCS to ensure reproducibility.

These models enable data-driven decision-making by filtering out noise, increasing customers' trustworthiness in feedback, and placing users in front of credible, sentiment-aware content, which ultimately builds better overall customer experience and drives strategic product insight. Accurately detecting fake reviews helps strengthen content authenticity and user trust, critical for platforms like Amazon. Automating sentiment analysis allows businesses to monitor customer satisfaction on a scale, identify problematic products, and refine recommendation systems.

```
>>> from pyspark.ml.feature import VectorAssembler
>>> from pyspark.ml.classification import LogisticRegression, DecisionTreeClassifier, RandomForestClassifier
>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator, BinaryClassificationEvaluator
>>> from pyspark.ml import Pipeline
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder.appName("ClasificationAnalysis").getOrCreate()
25/06/18 15:54:05 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
>>> ds_ml = spark.read.option("header", True).option("inferSchema", True).csv("gs://assignment_pipeline/outputs/feature_engineered_ds/")
>>> selected_features = ["price", "helpful_vote", "verified_purchase"]
>>> label_col = "sentiment_label"
>>> assembler = VectorAssembler(
...     inputCols=selected_features,
...     outputCol="features"
... )
>>> data_ml = ds_ml.select(*(selected_features + [label_col]))
>>> data_ml = assembler.transform(data_ml).select("features", label_col)
>>> train_data, test_data = data_ml.randomSplit([0.8, 0.2], seed=42)
>>> lr = LogisticRegression(labelCol=label_col, featuresCol="features")
>>> dt = DecisionTreeClassifier(labelCol=label_col, featuresCol="features")
>>> rf = RandomForestClassifier(labelCol=label_col, featuresCol="features", numTrees=50)
>>> lr_model = lr.fit(train_data)
>>> dt_model = dt.fit(train_data)
>>> rf_model = rf.fit(train_data)
>>> lr_preds = lr_model.transform(test_data)
>>> dt_preds = dt_model.transform(test_data)
>>> rf_preds = rf_model.transform(test_data)
>>> def evaluate_binary_model(predictions, label_col):
...     multi_eval = MulticlassClassificationEvaluator(labelCol=label_col, predictionCol="prediction")
...     binary_eval = BinaryClassificationEvaluator(labelCol=label_col, rawPredictionCol="rawPrediction")
...     results = {}
...     results["Accuracy"] = multi_eval.setMetricName("accuracy").evaluate(predictions)
...     results["F1 Score"] = multi_eval.setMetricName("f1").evaluate(predictions)
...     results["Weighted Precision"] = multi_eval.setMetricName("weightedPrecision").evaluate(predictions)
...     results["Weighted Recall"] = multi_eval.setMetricName("weightedRecall").evaluate(predictions)
...     results["AUC"] = binary_eval.setMetricName("areaUnderROC").evaluate(predictions)
...     results["PR AUC"] = binary_eval.setMetricName("areaUnderPR").evaluate(predictions)
...     return results
```

Figure 13. Model Implementation for Sentiment Classification. Author's Property.

```
>>> import subprocess
>>> # Create a dictionary to hold all evaluation results
>>> all_results = {
...     "Logistic Regression": evaluate_binary_model(lr_preds, label_col),
...     "Decision Tree": evaluate_binary_model(dt_preds, label_col),
...     "Random Forest": evaluate_binary_model(rf_preds, label_col)
... }
>>>
>>> # Step 10.Print results
>>> for model_name, metrics in all_results.items():
...     print(f"\n=== {model_name} ===")
...     for metric, value in metrics.items():
...         print(f"{metric}: {value:.4f}")
...

=== Logistic Regression ===
Accuracy: 0.8396
F1 Score: 0.7666
Weighted Precision: 0.7427
Weighted Recall: 0.8396
AUC: 0.5589
PR AUC: 0.8583

=== Decision Tree ===
Accuracy: 0.8397
F1 Score: 0.7665
Weighted Precision: 0.7051
Weighted Recall: 0.8397
AUC: 0.5000
PR AUC: 0.8397

=== Random Forest ===
Accuracy: 0.8397
F1 Score: 0.7665
Weighted Precision: 0.7051
Weighted Recall: 0.8397
AUC: 0.5893
PR AUC: 0.8678
```

Figure 14. Sentiment Classification Model Evaluation Results. Author's Property.

PHASE 5. REFLECTION AND RECOMMENDATIONS.

This final phase reflects on the key challenges encountered throughout the development of the big end-to-end data pipeline and offers practical recommendations for future improvements.

5.1. CHALLENGES FACED.

Several technical challenges were encountered while developing this project. Main issues included:

- Data quality and format issues: the meta dataset contained inconsistencies like nested columns, string-format numbers; and irrelevant columns like images and links. These required manual inspection and cleaning. Inconsistent schema and special characters, such as price column cleaned with regexp_replace() function and later cast.
- Join key inconsistencies: the original plan of joining on asin proved was not possible due to its absent presence in the meta dataset. Parent_asin was consequently resorted to as an interim measure, which, while working, may have caused some inaccuracies.
- Hive table initialization errors: the first attempt to review the datasets with Hive was not possible as the table had misleading header and field delimiters, which derived in failing. This was resolved by doing an initial clean with PySpark.
- Cluster memory constraints: the Spark executor failed during fake review modeling training. To resolve this, the data was sampled to 10% and TF-IDF features limited to 1,000. Only after these steps, the model be trained without errors.

5.2. RECOMMENDATIONS AND IMPROVEMENTS.

- Cluster resource monitoring and auto-scaling: implementing monitoring tools like StackDriver for GCP (which was the tool used in this project) and enabling auto-scaling would help manage memory issues better.
- Schema validation scripts: creating automated scripts to check schema consistency before data ingestion woold prevent errors in joins or Hive initialization.

- Feature selection automation: automated feature importance validation tools can help further adjust the input features for model training, reduce overfitting, or enhance accuracy.

5.3. ENVIRONMENT COMPARISON: CLOUD VS DOCKER VS LOCAL IDE.

Google Cloud Platform (GCP), part of Google Cloud Services (GCS), was selected to develop this project considering accessibility, scalability, easy set-up, and web-like interface. Although it required careful resource management and job design, it was the most suitable due to the size of the data and the need for distributed computation. Table 2 displays a summary comparison of the pros and cons of each environment.

Table 2. Environments Comparison. Author's property.

Environment	Pros	Cons
Cloud (GCP)	Scalable, easy data sharing, high-performance clusters, GCS support	Expensive, may crash on long jobs, requires quota configuration
Docker	Portable, consistent setup, good for CI/CD pipelines	Complex to configure for big data, limited to host machine
Local IDE	Easy debugging and prototyping, full control	Not scalable, crashes with large data, not suitable for parallel jobs

CONCLUSIONS.

- This project was able to demonstrate the capabilities of a distributed big data pipeline (from data ingestion to advanced analytics), showing that tools like Spark and Hive can deal with large-scale, unstructured datasets and convert them into clean, enhanced data suitable for machine learning.

- While Hive and Spark offer capabilities to perform the same tasks, during the project development it was shown that Hive queries for EDA took longer than with PySpark, demonstrating than in terms of speed with the basic cluster configuration, PySpark outperforms Hive.

- Combining structured features with text features (using TF-IDF) significantly improved model performance for fake review detection, with an accuracy over 94% and a high F1 Score, proving the value of feature engineering in real-world applications.

- Google did provide the scalability needed to handle big datasets, but it also brought resource management issues. Sample size optimization and pipeline complexity were crucial to avoiding crashes during model training without compromising the model's performance.

- This research may serve as a basis for further exploration, tackling the challenges faced by the limitations of handling big data unstructured datasets, expanding the scope to a fake review implementation on a full-scale dataset, and deploying the final validated models.

BIBLIOGRAPHY.

I. Agrawal, D. *et al.* (2011) 'Challenges and Opportunities with Big Data'.

II. Al Gharbi, S. *et al.* (2020) 'Using Data-Mining CRISP-DM Methodology to Predict Drilling Troubles in Real-Time', in. *SPE Asia Pacific Oil & Gas Conference and Exhibition*, p. D013S103R013. Available at: https://doi.org/10.2118/202326-MS.

III. Choo, E., Yu, T. and Chi, M. (2017) 'Detecting opinion spammer groups and spam targets through community discovery and sentiment analysis', *Journal of Computer Security*, 25(3), pp. 283–318. Available at: https://doi.org/10.3233/JCS-16941.

IV. Hou, Y. (2024) *Amazon Reviews'23*, *Amazon Reviews'23*. Available at: https://amazon-reviews-2023.github.io/# (Accessed: 19 June 2025).

V. Jindal, N. and Liu, B. (2008) 'Opinion spam and analysis', in. *Proceedings of the 2008 international conference on web search and data mining*, pp. 219–230.

VI. Pang, B. and Lee, L. (2006) 'Opinion Mining and Sentiment Analysis', *Foundations and Trends® in Information Retrieval*, 2(1–2), pp. 1–135. Available at: https://doi.org/10.1561/1500000001.

VII. Schröer, C., Kruse, F. and Gómez, J.M. (2021) 'A Systematic Literature Review on Applying CRISP-DM Process Model', *Procedia Computer Science*, 181, pp. 526–534. Available at: https://doi.org/10.1016/j.procs.2021.01.199.

VIII. Zaharia, M. *et al.* (2016) 'Apache Spark: a unified engine for big data processing', *Communications of the ACM*, 59(11), pp. 56–65. Available at: https://doi.org/10.1145/2934664.

APPENDIX.

All scripts used to develop this project are stored in GitHub:

https://github.com/HerGuzmanMoratto/amazon-bigdata-pipeline