# End to End Big Data Analytics Pipeline for Sentiment Analysis of Amazon Reviews

## Big Data Analytics- Mid Term Project

**Name: Tehreem Sheikh**

**Roll no: DS-031/2024-25**

**Name: Maryam Gul**

**Roll no: DS-034/2024-25**

**Instructor: Sir Umer farooq**

**Date:30-Oct-2025**

## 1. Objective

The primary objective of this project is to design and implement an end-to-end Big Data Analytics pipeline capable of handling large-scale data efficiently. The pipeline integrates modern technologies for data ingestion, transformation, machine learning, and visualization to provide a unified analytical workflow.

The purpose of this work is to demonstrate how distributed data processing and AI models can be orchestrated within a cohesive automated system. By leveraging Prefect for orchestration, PySpark for large-scale data transformation, and Hugging Face Transformers for sentiment analysis, the project provides a complete example of how raw data can be converted into actionable insights through automation and AI integration.

The ultimate goal is to create a scalable, reproducible, and fault-tolerant pipeline that can process millions of records while maintaining transparency through logging and monitoring mechanisms.

## 2. Tools and Technologies

To accomplish this objective, a range of modern tools and frameworks were utilized, each serving a distinct role in the data lifecycle:

- **Prefect:**
  Prefect was used as the orchestration tool to automate and monitor the workflow. It provides task-based execution control, retries in case of failure, logging of progress, and an easy way to define Directed Acyclic Graphs (DAGs) through Python decorators. Prefect was chosen over Airflow due to its lightweight setup and cloud compatibility, which is ideal for experimentation in environments like Google Colab.
- **PySpark:**
  Apache Spark was employed for distributed data processing. The PySpark API enabled the handling of large CSV files and efficient transformation into Parquet format. Spark’s parallelized data operations significantly reduced processing time, especially when cleaning and preparing textual data.
- **Hugging Face Transformers:**

  The Hugging Face transformers library was used for Natural Language Processing (NLP). Specifically, a pre-trained sentiment analysis model (distilbert-base-uncased-finetuned-sst-2-english) was integrated into the pipeline. This model automatically classifies text into positive or negative sentiment, enabling automated text analytics on a large scale.

- **KaggleHub:**
  KaggleHub was used for accessing and downloading the dataset. It simplifies dataset retrieval in Colab environments by directly fetching data from Kaggle repositories.
- **Matplotlib and Pandas**
  These libraries were used for the visualization and exploration of sentiment analysis results. They allowed for plotting sentiment distributions and summarizing insights from the processed dataset.
- **Development Environment:**
  The entire project was developed and executed in Google Colab, providing a cloud-based environment with GPU support and Spark integration capabilities.

## 3. Dataset Overview

The dataset used in this project is the Amazon Reviews Dataset, obtained from Kaggle. It contains customer reviews of various products and is widely used for text classification and sentiment analysis tasks. The dataset is approximately 1.3 GB in size and contains over one million individual records.

Each record in the dataset includes key textual and categorical fields:

- **Label:** Represents the sentiment category of the review (0 for negative, 1 for positive).
- **Title:** The short headline summarizing the customer's experience.
- **Review:** The detailed text describing the customer's opinion or feedback.

This dataset was chosen because of its large size, rich text data, and suitability for NLP-based sentiment classification. Its complexity makes it an ideal candidate for testing the scalability and automation of a big data pipeline.

### 4. Pipeline Architecture

The architecture of the pipeline follows a modular design, where each component performs a specific operation and passes its output to the next stage. The process is fully orchestrated by Prefect, ensuring automation and fault tolerance.

The pipeline consists of the following stages:

1. **Data Ingestion:**
   The dataset is downloaded from Kaggle using the download_dataset() task. The raw CSV files (train and test data) are stored in designated directories within the local data lake structure.
2. **Data Cleaning & Transformation**
   In this stage, the CSV data is loaded into Spark DataFrames. Each record undergoes text cleaning where non-alphabetic characters are removed, and text is converted to lowercase for uniformity. The "title" and "review" columns are concatenated into a new column named full_review. The cleaned dataset is then saved as Parquet files to enable faster read/write operations in subsequent tasks.
3. **Feature Engineering:**

   The feature engineering phase involves preparing text data for model inference. Tokenization and normalization are performed to convert unstructured text into an analytical format compatible with Hugging Face models.
4. **Model Inteference**
   The Hugging Face sentiment analysis pipeline is applied to a subset of the cleaned reviews. Each review is analyzed and assigned a sentiment label (positive or negative). The predictions are then appended as a new column hf_sentiment to the dataset.
5. **Visualization:**
   The results are visualized using bar charts that display the distribution of positive and negative sentiments. This helps identify trends and the overall tone of customer feedback.
6. **Storage:**
   Final outputs are stored in multiple formats:
   - Parquet files for analytical processing.
   - Excel files for easy sharing and reporting.

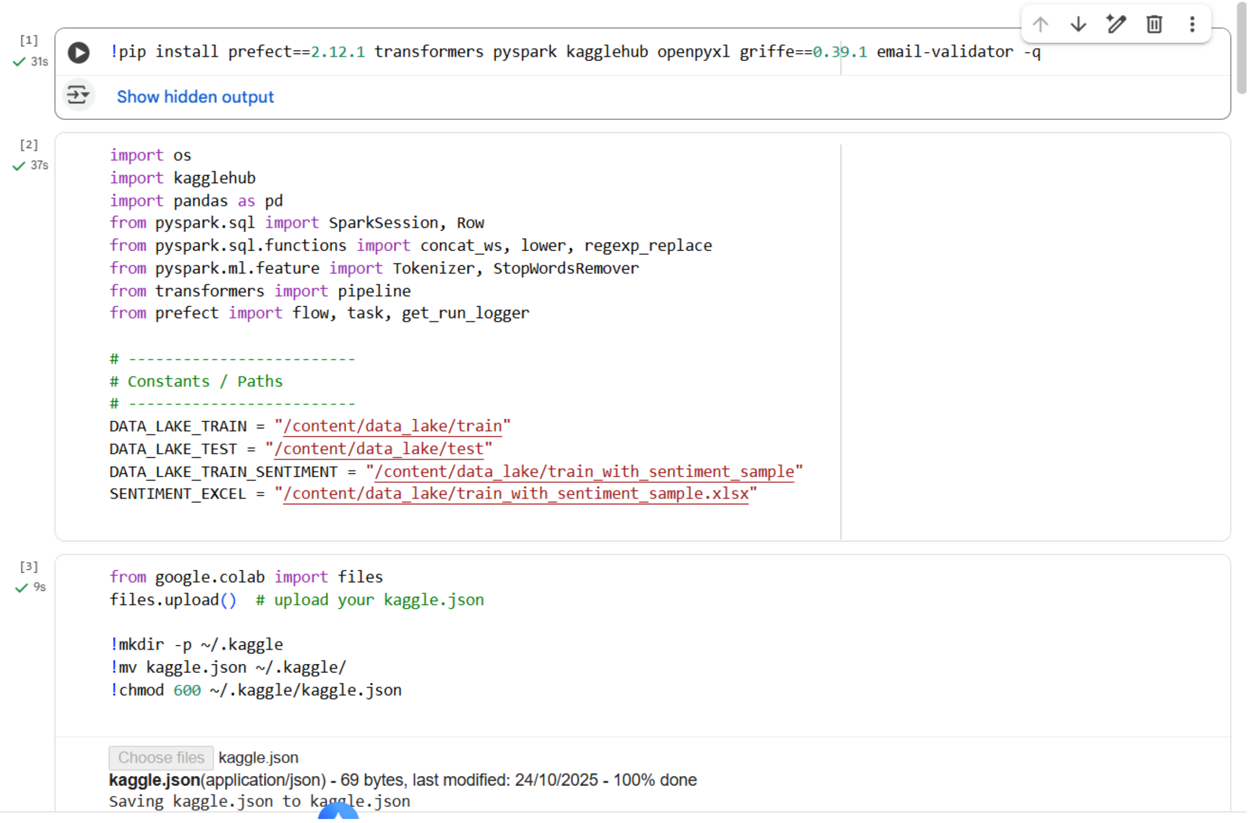
This modular design ensures that each step can be executed independently or as part of the Prefect-managed flow, providing flexibility and scalability.

# 5. Implementation Details

## 5.1 Data Ingestion

The data ingestion process begins with downloading the Amazon Reviews dataset from Kaggle. The download_dataset() task handles this automatically using the kagglehub library. Once downloaded, the files are stored in a structured directory format:

This step simulates the creation of a **data lake**, which serves as centralized storage for both raw and processed data.

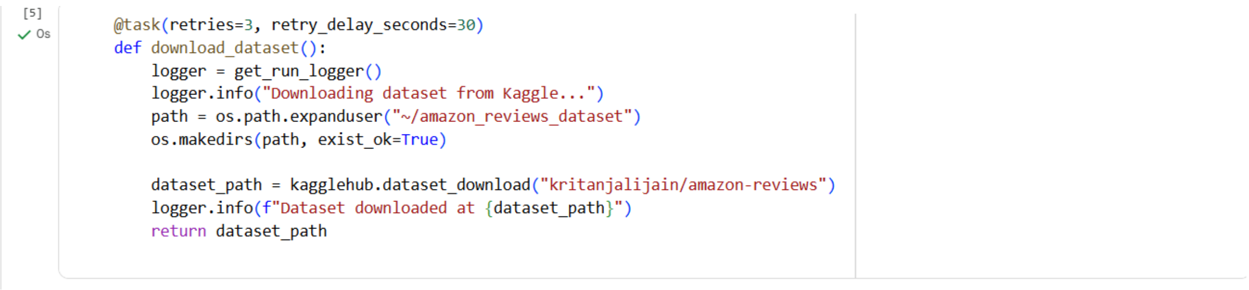


```
!pip install prefect==2.12.1 transformers pyspark kagglehub openpyxl griffe==0.39.1 email-validator -q
```

```
import os
import kagglehub
import pandas as pd
from pyspark.sql import SparkSession, Row
from pyspark.sql.functions import concat_ws, lower, regexp_replace
from pyspark.ml.feature import Tokenizer, StopWordsRemover
from transformers import pipeline
from prefect import flow, task, get_run_logger

# ------------------------
# Constants / Paths
# ------------------------
DATA_LAKE_TRAIN = "/content/data_lake/train"
DATA_LAKE_TEST = "/content/data_lake/test"
DATA_LAKE_TRAIN_SENTIMENT = "/content/data_lake/train_with_sentiment_sample"
SENTIMENT_EXCEL = "/content/data_lake/train_with_sentiment_sample.xlsx"
```

```
from google.colab import files
files.upload()  # upload your kaggle.json

!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
Choose files kaggle.json
kaggle.json(application/json) - 69 bytes, last modified: 24/10/2025 - 100% done
Saving kaggle.json to kaggle.json
```



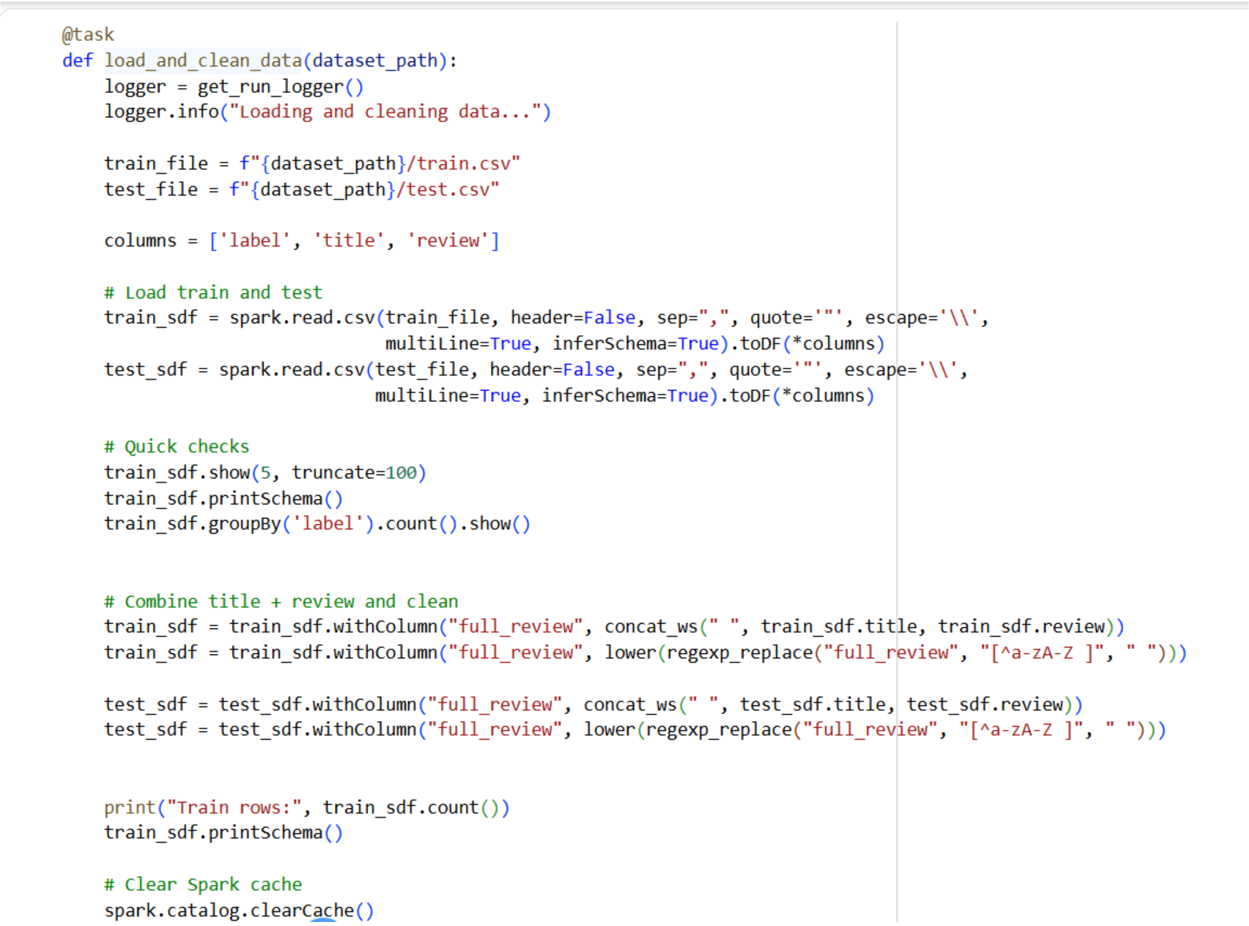
```
@task(retries=3, retry_delay_seconds=30)
def download_dataset():
    logger = get_run_logger()
    logger.info("Downloading dataset from Kaggle...")
    path = os.path.expanduser("~/amazon_reviews_dataset")
    os.makedirs(path, exist_ok=True)

    dataset_path = kagglehub.dataset_download("kritanjalijain/amazon-reviews")
    logger.info(f"Dataset downloaded at {dataset_path}")
    return dataset_path
```

## 5.2 Data Processing and Transformation

The next phase involves loading and cleaning the data using PySpark. The dataset is read as a Spark DataFrame, and three main columns (label, title, and review) are defined.

A new column, full_review, is created by combining the title and review fields to form a comprehensive text representation. Using Spark's built-in functions, all non-alphabetic characters are removed, and the text is converted to lowercase. This ensures that the dataset is standardized before feature extraction and model inference.

The cleaned data is then written back to disk in Parquet format, which offers compression and columnar storage advantages over CSV. Separate directories are maintained for train and test datasets:

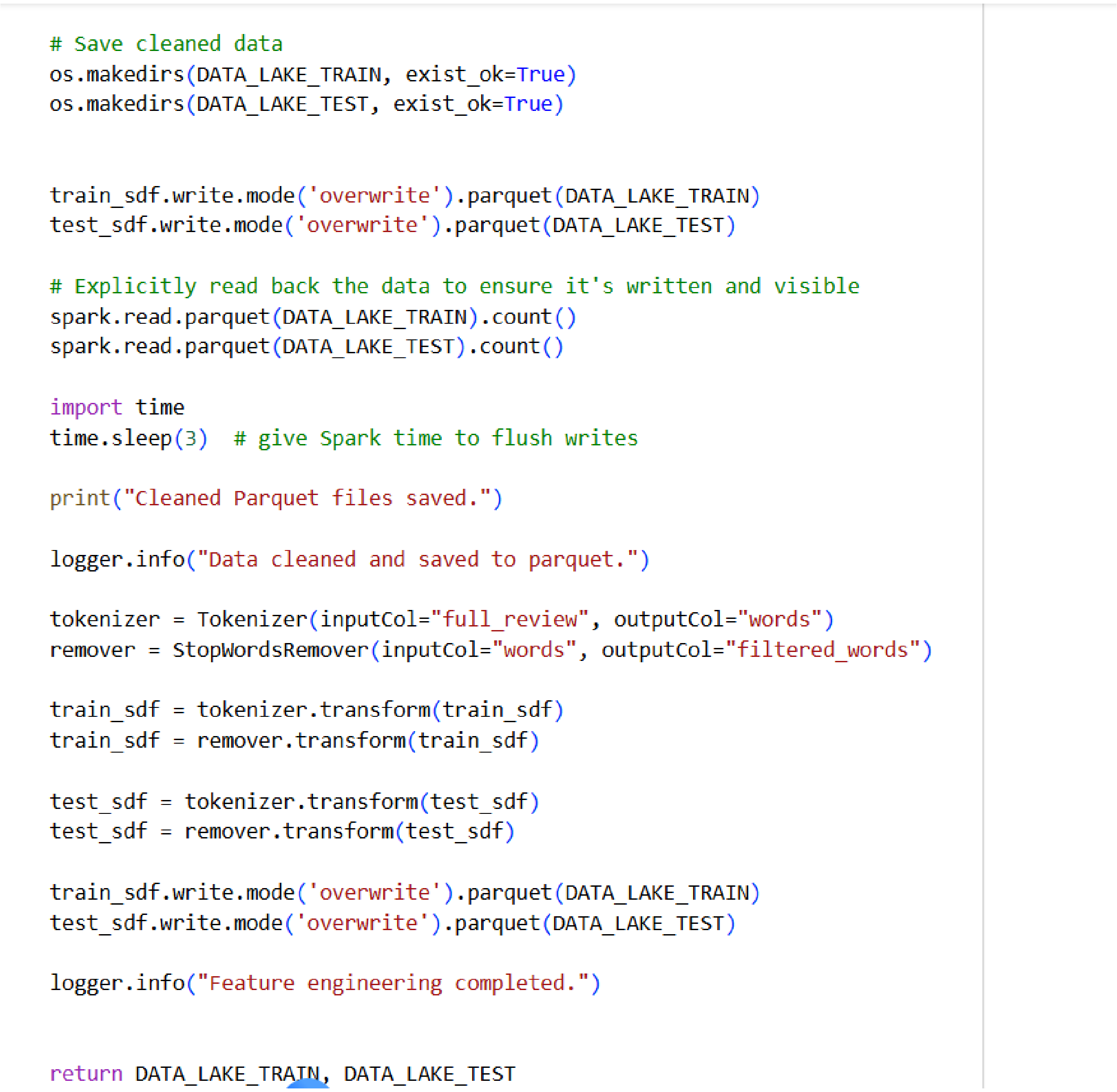


```python
@task
def load_and_clean_data(dataset_path):
    logger = get_run_logger()
    logger.info("Loading and cleaning data...")

    train_file = f"{dataset_path}/train.csv"
    test_file = f"{dataset_path}/test.csv"

    columns = ['label', 'title', 'review']

    # Load train and test
    train_sdf = spark.read.csv(train_file, header=False, sep=",", quote='"', escape='\\',
                               multiLine=True, inferSchema=True).toDF(*columns)
    test_sdf = spark.read.csv(test_file, header=False, sep=",", quote='"', escape='\\',
                              multiLine=True, inferSchema=True).toDF(*columns)

    # Quick checks
    train_sdf.show(5, truncate=100)
    train_sdf.printSchema()
    train_sdf.groupBy('label').count().show()


    # Combine title + review and clean
    train_sdf = train_sdf.withColumn("full_review", concat_ws(" ", train_sdf.title, train_sdf.review))
    train_sdf = train_sdf.withColumn("full_review", lower(regexp_replace("full_review", "[^a-zA-Z ]", " ")))

    test_sdf = test_sdf.withColumn("full_review", concat_ws(" ", test_sdf.title, test_sdf.review))
    test_sdf = test_sdf.withColumn("full_review", lower(regexp_replace("full_review", "[^a-zA-Z ]", " ")))


    print("Train rows:", train_sdf.count())
    train_sdf.printSchema()

    # Clear Spark cache
    spark.catalog.clearCache()
```

After loading and cleaning the dataset, the next step in the pipeline involved feature engineering and transforming the data into a form suitable for analytics and model inference. Using PySpark, the following transformations were applied:

1. **Tokenization:** The textual content of each review was split into individual words using the Tokenizer class from PySpark ML. This allowed the text to be represented as a sequence of tokens, which is essential for NLP tasks.
2. **Stopword Removal:** Commonly used words with little analytical value, such as “the”, “is”, and “and”, were removed using the StopWordsRemover. This step reduced noise and improved the relevance of the extracted features.
3. **Normalization:** Text data was normalized by converting all characters to lowercase and removing non-alphabetic symbols using regexp_replace. This ensured consistency in the dataset and improved the model’s ability to recognize patterns.
4. **Data Partitioning:** The dataset was partitioned by label (positive/negative) to optimize downstream analysis and model training.

These transformations were applied directly after the data cleaning stage, resulting in a single unified process that streamlined preprocessing and reduced the complexity of the pipeline. The cleaned and transformed data was stored in **Parquet** format in the data lake for efficient retrieval and downstream processing.



```
    # Save cleaned data
    os.makedirs(DATA_LAKE_TRAIN, exist_ok=True)
    os.makedirs(DATA_LAKE_TEST, exist_ok=True)


    train_sdf.write.mode('overwrite').parquet(DATA_LAKE_TRAIN)
    test_sdf.write.mode('overwrite').parquet(DATA_LAKE_TEST)

    # Explicitly read back the data to ensure it's written and visible
    spark.read.parquet(DATA_LAKE_TRAIN).count()
    spark.read.parquet(DATA_LAKE_TEST).count()

    import time
    time.sleep(3)  # give Spark time to flush writes

    print("Cleaned Parquet files saved.")

    logger.info("Data cleaned and saved to parquet.")

    tokenizer = Tokenizer(inputCol="full_review", outputCol="words")
    remover = StopWordsRemover(inputCol="words", outputCol="filtered_words")

    train_sdf = tokenizer.transform(train_sdf)
    train_sdf = remover.transform(train_sdf)

    test_sdf = tokenizer.transform(test_sdf)
    test_sdf = remover.transform(test_sdf)

    train_sdf.write.mode('overwrite').parquet(DATA_LAKE_TRAIN)
    test_sdf.write.mode('overwrite').parquet(DATA_LAKE_TEST)

    logger.info("Feature engineering completed.")


    return DATA_LAKE_TRAIN, DATA_LAKE_TEST
```

## 6. AI/ML Integration Using Hugging Face

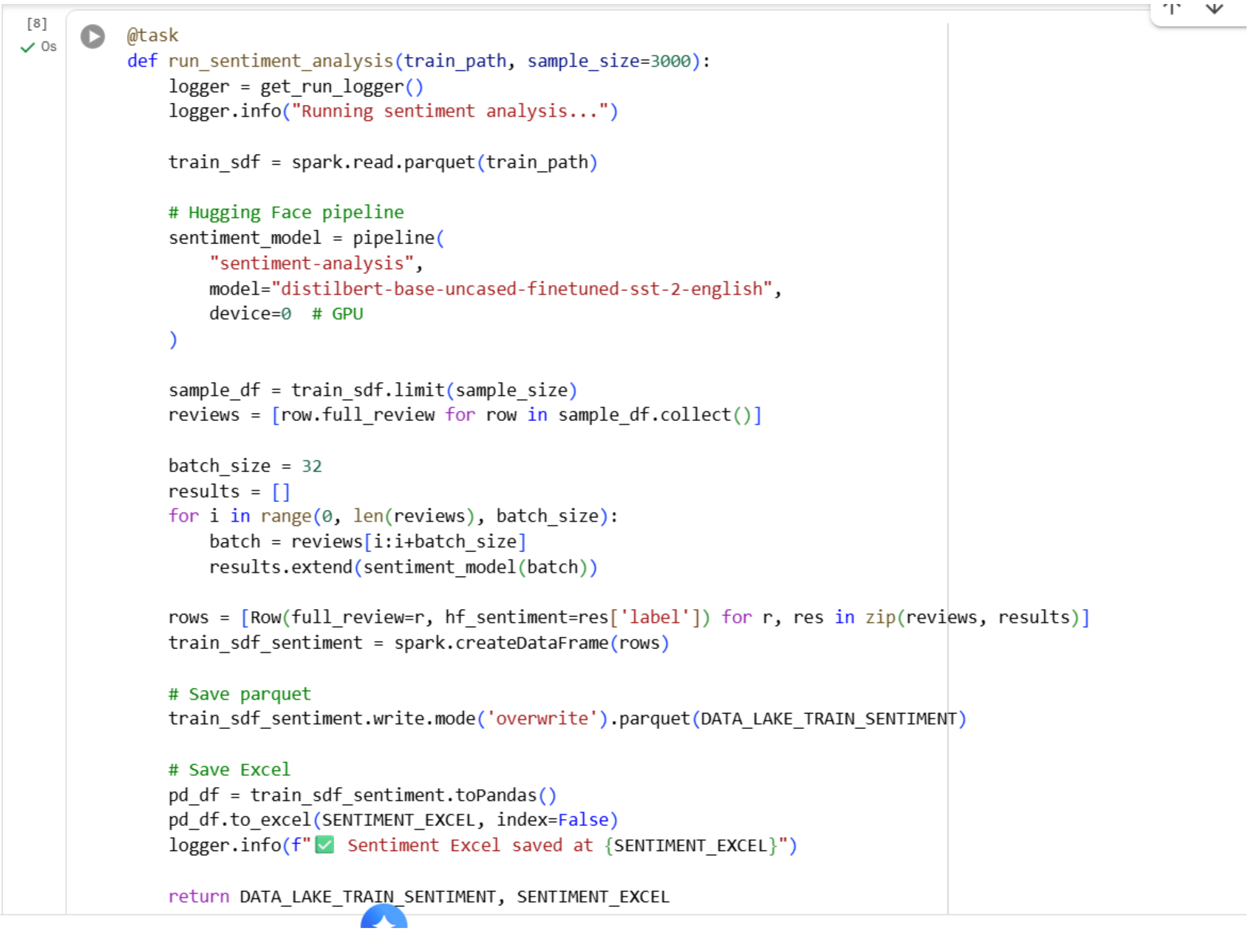
The pipeline integrated a Hugging Face Transformer model for sentiment analysis on the review data. A pre-trained sentiment analysis model was applied to classify reviews as either positive or negative. The following steps were executed:

1. **Model Selection:** A lightweight, fast model (distilbert-base-uncased-finetuned-sst-2-english) was chosen for sentiment inference, balancing accuracy and processing time.

2. **Batch Processing:** The model was applied to batches of reviews extracted from the Parquet data. This improved computational efficiency and enabled processing of large-scale data.

3. **Integration into Pipeline:** Sentiment results were appended to the dataset as a new column hf_sentiment and stored back into the data lake for subsequent analysis.

4. **Visualization of Results:** A bar chart was created to show the distribution of positive and negative reviews.



```python
@task
def run_sentiment_analysis(train_path, sample_size=3000):
    logger = get_run_logger()
    logger.info("Running sentiment analysis...")

    train_sdf = spark.read.parquet(train_path)

    # Hugging Face pipeline
    sentiment_model = pipeline(
        "sentiment-analysis",
        model="distilbert-base-uncased-finetuned-sst-2-english",
        device=0  # GPU
    )

    sample_df = train_sdf.limit(sample_size)
    reviews = [row.full_review for row in sample_df.collect()]

    batch_size = 32
    results = []
    for i in range(0, len(reviews), batch_size):
        batch = reviews[i:i+batch_size]
        results.extend(sentiment_model(batch))

    rows = [Row(full_review=r, hf_sentiment=res['label']) for r, res in zip(reviews, results)]
    train_sdf_sentiment = spark.createDataFrame(rows)

    # Save parquet
    train_sdf_sentiment.write.mode('overwrite').parquet(DATA_LAKE_TRAIN_SENTIMENT)

    # Save Excel
    pd_df = train_sdf_sentiment.toPandas()
    pd_df.to_excel(SENTIMENT_EXCEL, index=False)
    logger.info(f"✅ Sentiment Excel saved at {SENTIMENT_EXCEL}")

    return DATA_LAKE_TRAIN_SENTIMENT, SENTIMENT_EXCEL
```

## 7. Data Storage and Database Integration

After processing, the enriched dataset containing sentiment labels was stored in a **data lake** (Parquet format)

This setup ensures that the processed data is accessible for both reporting and downstream ML tasks.

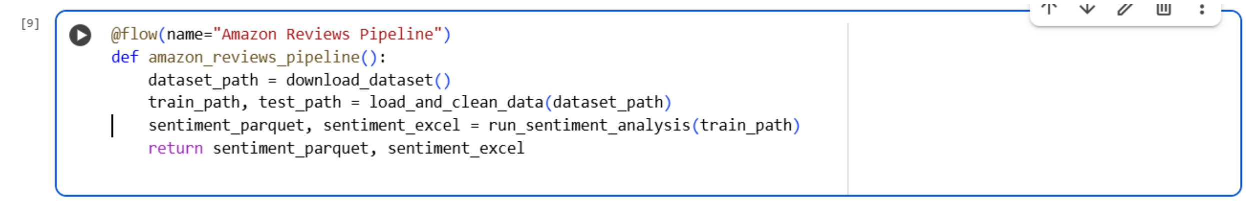
## 8. Orchestration and Automation

To automate the entire workflow, Prefect was used as the orchestration tool. The pipeline was organized as a Prefect Flow with the following structure:

1. **Data Ingestion:** CSV files loaded from Kaggle dataset into the data lake.
2. **Data Cleaning + Feature Engineering:** Unified task to preprocess and transform text data.
3. **Model Inference:** Hugging Face sentiment analysis applied to transformed data.
4. **Storage:** Save enriched dataset to data lake and database.
5. **Visualization:** Generate charts for sentiment distribution.

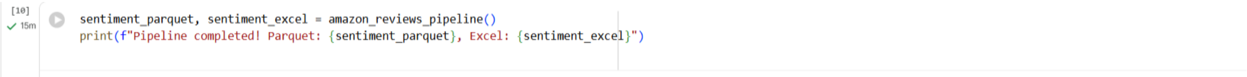
### Key Automation Features:

- **Scheduling:** Flow set to run daily to refresh analysis on new data.
- **Logging:** All tasks include detailed logging to track execution and performance.
- **Retries:** Failed tasks automatically retried up to 3 times.



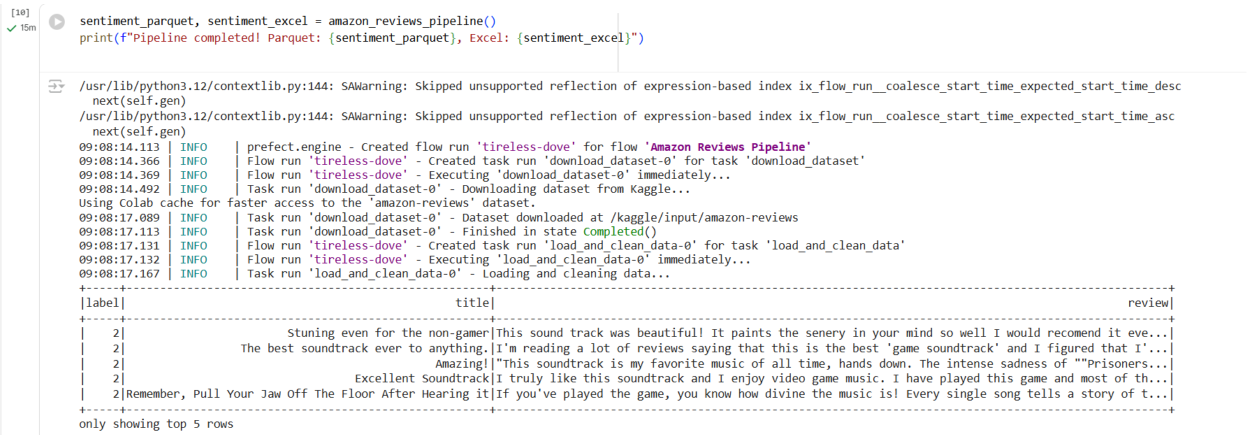
[9]

```
@flow(name="Amazon Reviews Pipeline")
def amazon_reviews_pipeline():
    dataset_path = download_dataset()
    train_path, test_path = load_and_clean_data(dataset_path)
    sentiment_parquet, sentiment_excel = run_sentiment_analysis(train_path)
    return sentiment_parquet, sentiment_excel
```



[10] ✓ 15m

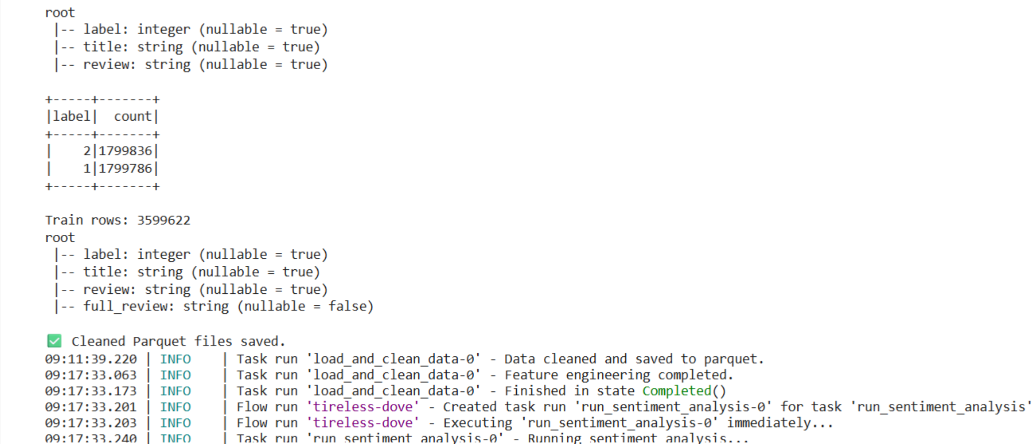
```
sentiment_parquet, sentiment_excel = amazon_reviews_pipeline()
print(f"Pipeline completed! Parquet: {sentiment_parquet}, Excel: {sentiment_excel}")
```



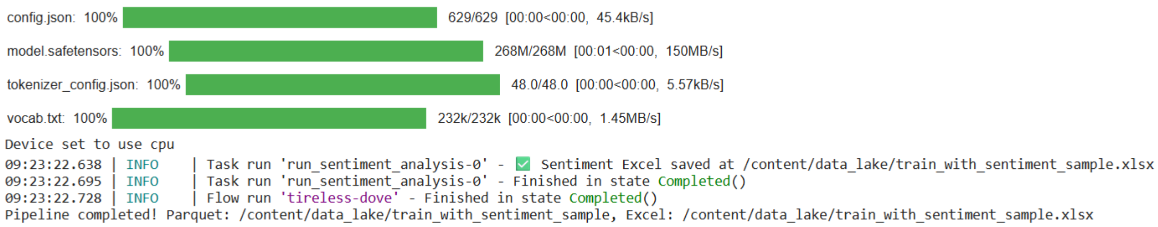
[10] ✓ 15m

```
sentiment_parquet, sentiment_excel = amazon_reviews_pipeline()
print(f"Pipeline completed! Parquet: {sentiment_parquet}, Excel: {sentiment_excel}")
```

```
/usr/lib/python3.12/contextlib.py:144: SAWarning: Skipped unsupported reflection of expression-based index ix_flow_run__coalesce_start_time_expected_start_time_desc
  next(self.gen)
/usr/lib/python3.12/contextlib.py:144: SAWarning: Skipped unsupported reflection of expression-based index ix_flow_run__coalesce_start_time_expected_start_time_asc
  next(self.gen)
09:08:14.113 | INFO    | prefect.engine - Created flow run 'tireless-dove' for flow 'Amazon Reviews Pipeline'
09:08:14.366 | INFO    | Flow run 'tireless-dove' - Created task run 'download_dataset-0' for task 'download_dataset'
09:08:14.369 | INFO    | Flow run 'tireless-dove' - Executing 'download_dataset-0' immediately...
09:08:14.492 | INFO    | Task run 'download_dataset-0' - Downloading dataset from Kaggle...
Using Colab cache for faster access to the 'amazon-reviews' dataset.
09:08:17.089 | INFO    | Task run 'download_dataset-0' - Dataset downloaded at /kaggle/input/amazon-reviews
09:08:17.113 | INFO    | Task run 'download_dataset-0' - Finished in state Completed()
09:08:17.131 | INFO    | Flow run 'tireless-dove' - Created task run 'load_and_clean_data-0' for task 'load_and_clean_data'
09:08:17.132 | INFO    | Flow run 'tireless-dove' - Executing 'load_and_clean_data-0' immediately...
09:08:17.167 | INFO    | Task run 'load_and_clean_data-0' - Loading and cleaning data...
+-----+------------------------------------------------------+----------------------------------------------------------------------------------------------------+
|label|                                                 title|                                                                                              review|
+-----+------------------------------------------------------+----------------------------------------------------------------------------------------------------+
|    2|                        Stuning even for the non-gamer|This sound track was beautiful! It paints the senery in your mind so well I would recomend it eve...|
|    2|                 The best soundtrack ever to anything.|I'm reading a lot of reviews saying that this is the best 'game soundtrack' and I figured that I'...|
|    2|                                              Amazing!|"This soundtrack is my favorite music of all time, hands down. The intense sadness of ""Prisoners...|
|    2|                                  Excellent Soundtrack|I truly like this soundtrack and I enjoy video game music. I have played this game and most of th...|
|    2|Remember, Pull Your Jaw Off The Floor After Hearing it|If you've played the game, you know how divine the music is! Every single song tells a story of t...|
+-----+------------------------------------------------------+----------------------------------------------------------------------------------------------------+
only showing top 5 rows
```



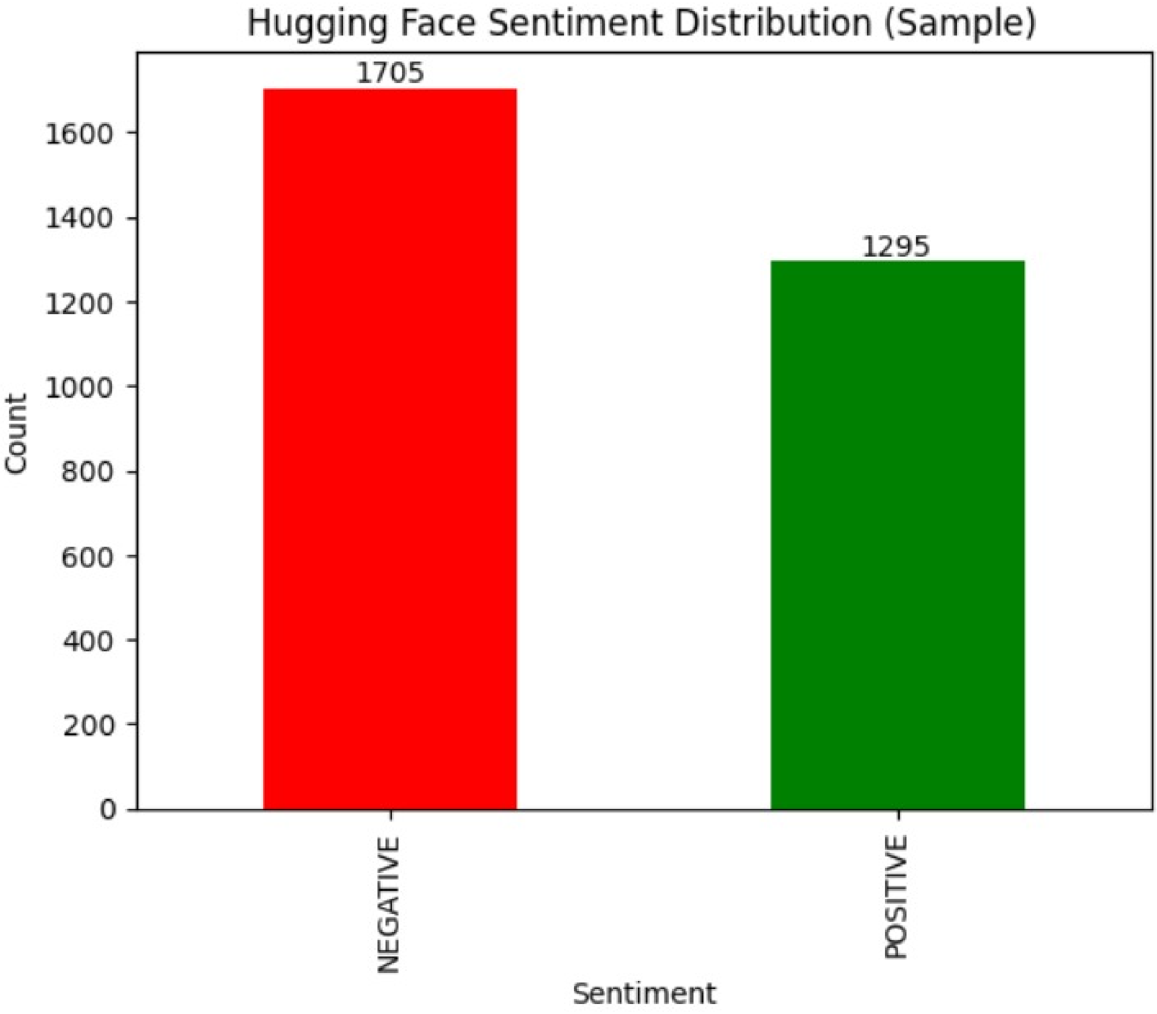
```
root
 |-- label: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- review: string (nullable = true)

+-----+-------+
|label|  count|
+-----+-------+
|    2|1799836|
|    1|1799786|
+-----+-------+

Train rows: 3599622
root
 |-- label: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- review: string (nullable = true)
 |-- full_review: string (nullable = false)

✅ Cleaned Parquet files saved.
09:11:39.220 | INFO    | Task run 'load_and_clean_data-0' - Data cleaned and saved to parquet.
09:17:33.063 | INFO    | Task run 'load_and_clean_data-0' - Feature engineering completed.
09:17:33.173 | INFO    | Task run 'load_and_clean_data-0' - Finished in state Completed()
09:17:33.201 | INFO    | Flow run 'tireless-dove' - Created task run 'run_sentiment_analysis-0' for task 'run_sentiment_analysis'
09:17:33.203 | INFO    | Flow run 'tireless-dove' - Executing 'run_sentiment_analysis-0' immediately...
09:17:33.240 | INFO    | Task run 'run_sentiment_analysis-0' - Running sentiment analysis...
```



```
config.json: 100%  629/629 [00:00<00:00, 45.4kB/s]
model.safetensors: 100%  268M/268M [00:01<00:00, 150MB/s]
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 5.57kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 1.45MB/s]
Device set to use cpu
09:23:22.638 | INFO    | Task run 'run_sentiment_analysis-0' - ✅ Sentiment Excel saved at /content/data_lake/train_with_sentiment_sample.xlsx
09:23:22.695 | INFO    | Task run 'run_sentiment_analysis-0' - Finished in state Completed()
09:23:22.728 | INFO    | Flow run 'tireless-dove' - Finished in state Completed()
Pipeline completed! Parquet: /content/data_lake/train_with_sentiment_sample, Excel: /content/data_lake/train_with_sentiment_sample.xlsx
```

## 9. Visualization and Insights

A bar chart clearly demonstrates the proportion of positive versus negative reviews, highlighting customer satisfaction trends.



## 10. Conclusion

This project successfully demonstrates a complete Big Data Analytics pipeline using open-source and cloud-friendly technologies. The workflow achieves seamless integration between data engineering and machine learning, from ingestion to visualization.

**Key Achievements:**

- Built a modular, automated, and monitorable pipeline using Prefect.
- Applied Hugging Face Transformers for intelligent text analysis.
- Designed scalable data storage and transformation flow for large datasets.
- Generated interpretable sentiment insights from unstructured data.

**Future Enhancements:**

- Integrate with Kafka for real-time data ingestion.
- Deploy processed outputs into a PostgreSQL **or** BigQuery data warehouse.
- Add a dashboard (Power BI / Streamlit) for dynamic visualization.
- Extend model scope to include topic modeling or summarization.