**IE7374 Machine Learning Operations**

**Topic: Automated-BiLingual-Complaint-System**

**Group 06**

**Rahul Nenavath**

**Rajkumar Pandey**

**Sayam Khatri**

**Anirudh Hegde**

**Archit Singh**

**Venkata Mani Sivasai Shanmukha Rajendra**

Table of Contents

[**Automated-BiLingual-Complaint-System** 3](#_Toc181728894)

[**Project Setup** 3](#_Toc181728895)

[**Prerequisites** 3](#_Toc181728896)

[**Step-by-Step Guide** 3](#_Toc181728897)

[**Project Overview** 5](#_Toc181728898)

[**Data Acquisition** 5](#_Toc181728899)

[**Data Schema** 6](#_Toc181728900)

[**Data Preprocessing Pipeline** 7](#_Toc181728901)

[**Preprocessing Steps** 7](#_Toc181728902)

[**1. Data Loading** 7](#_Toc181728903)

[**2. Basic Filtering** 7](#_Toc181728904)

[**3. Language Detection** 7](#_Toc181728905)

[**4. Data Cleaning** 8](#_Toc181728906)

[**5. Abusive Content Filtering** 8](#_Toc181728907)

[**6. PII Data Anonymization** 8](#_Toc181728908)

[**7. Data Aggregation** 8](#_Toc181728909)

[**8. Abusive Content Filtering** 9](#_Toc181728910)

[**9. Success Email Trigger** 9](#_Toc181728911)

[**10. Slack Alerts Integration** 9](#_Toc181728912)

[**Output** 9](#_Toc181728913)

[**Pipeline Orchestration (Airflow DAGs)** 11](#_Toc181728914)

[**Tracking and Logging** 13](#_Toc181728915)

[**Key Features:** 13](#_Toc181728916)

[**Data Versioning with DVC** 14](#_Toc181728917)

[**Pipeline Flow Optimization** 15](#_Toc181728918)

[**Data Schema & Statistics Generation** 17](#_Toc181728919)

[*1. Data Quality Assurance* 17](#_Toc181728920)

[*2. Statistics Generation and Schema Analysis* 17](#_Toc181728921)

[**Anomaly Detection & Alerts** 18](#_Toc181728922)

[1. Anomaly Detection 18](#_Toc181728923)

[2. Alerts 18](#_Toc181728924)

[**Data Bias Detection and Mitigation** 19](#_Toc181728925)

[**1. Detecting Bias in Data:** 19](#_Toc181728926)

[**2. Data Slicing:** 19](#_Toc181728927)

[**3. Bias Mitigation:** 19](#_Toc181728928)

[**Preprocessing Module Unit Tests** 20](#_Toc181728929)

[**License Information** 20](#_Toc181728930)

**Automated-BiLingual-Complaint-System**

**Project Setup**

**Prerequisites**

* Ensure you have **Docker** installed.
* Ensure you have **Visual Studio Code** installed.
* Allocate sufficient resources to Docker (e.g., memory, CPU) to ensure the project runs smoothly.

**Step-by-Step Guide**

**Step 1: Clone the Repository**

Clone the repository to your local machine and navigate into the project directory:

***git clone https://github.com/rajpandeygithub/Automated-BiLingual-Complaint-System.git***

***cd Automated-BiLingual-Complaint-System***

**Step 2: Open the Project in Visual Studio Code**

Open the project folder (Automated-BiLingual-Complaint-System) in **Visual Studio Code**.

**Step 3: Initialize and Start Docker Containers**

**Note: All necessary libraries are listed in the requirements.txt file, making it easy to install dependencies automatically. This file is also used in the docker-compose.yaml setup to ensure all dependencies are installed when running the project.**

Run the following commands in your Visual Studio Code terminal to initialize and start the Docker containers for Airflow:

1. Initialize Airflow:

***docker compose up airflow-init***

1. Start all containers:

***docker compose up***

**Step 4: Access the Airflow Web Interface**

Once the containers are up and running, open your browser and go to:

[***http://localhost:8080/home***](http://localhost:8080/home)

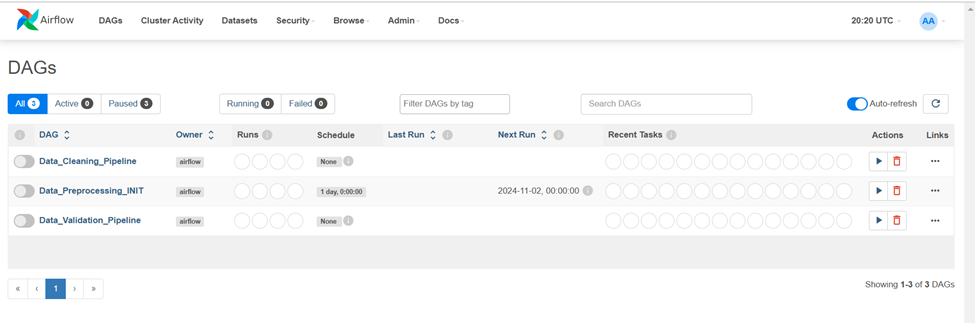
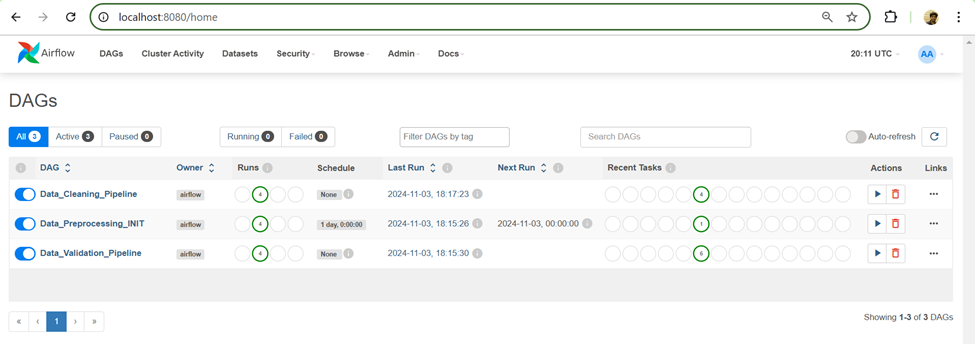
***Login with default username password:***

***Username: airflow2***

***Password: airflow2***

**Step 5: Enable and Run DAGs**

In the Airflow web interface:

1. Enable the toggles for all DAGs to activate them.
2. **First-time Setup**: If this is your first time running the DAGs, they should start automatically. [](https://private-user-images.githubusercontent.com/86112540/382602796-59721744-b1eb-435a-a7c8-365cac76f783.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..-EShJXPun1iLM2aJ9VtAnWUyn0qylVVUPZVGYAWZZIY)
3. **Subsequent Runs**: If the DAGs have been run before, trigger the Data\_Preprocessing\_INIT DAG manually. This will initiate a sequence:
   * Data\_Preprocessing\_INIT will trigger data\_validation\_trigger, which will, in turn, trigger the Data\_Validation\_Pipeline. [](https://private-user-images.githubusercontent.com/86112540/382602832-23bfa4cd-d7d8-4b91-9993-966873143c04.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..uwVvdNOleEVZr9ArgC75IGq7RSK1dPH9YOIEEtrZvr0)

**Step 6: Shut Down Docker Containers**

To stop and shut down the Docker containers, go to the **Visual Studio Code** terminal and run:

***docker compose down***

**Project Overview**

This project implements an automated system for handling customer complaints in the banking domain. Leveraging machine learning, the system analyzes complaints submitted in two languages (English & Hindi), and categorizes them using two distinct classifiers. The first classifier identifies the banking product related to the complaint (e.g., credit card, loan, account services), while the second routes the complaint to the appropriate department for efficient resolution. This solution streamlines complaint management, enhances response times, and significantly improves customer satisfaction by ensuring complaints are swiftly directed to the relevant teams.

**Key features include:**

* Simplified UI for complaint submission.
* Support for two languages (English, & Hindi).
* Automatic classification of products and departments using machine learning.
* Agent assignment based on language proficiency and availability.

**Data Acquisition**

We make use of the Consumer Complaint Database from Consumer Financial Protection Bureau (CFPB). This database compiles real-time U.S. consumer complaints regarding financial products and services. It includes data on consumer narratives (anonymized), banking products, and company responses, providing insight into financial sector issues.

For more information: [Link](https://www.consumerfinance.gov/data-research/consumer-complaints/)  
For API Access: [Link](https://cfpb.github.io/api/ccdb/api.html)

**Data Schema**

| **Attribute Name** | **Description** | **Data Type** |
| --- | --- | --- |
| complaint\_id | Unique identifier for each complaint | Integer |
| date\_received | Date when CFPB received the complaint | Datetime |
| date\_resolved | Date when the complaint was resolved by the bank | Integer |
| time\_resolved\_in\_days | Duration in days taken to resolve the complaint by the bank | Integer |
| complaint | Consumer's answer to "what happened" from the complaint. Consumers must opt-in to share their narrative. The narrative is not published unless the consumer consents, and consumers can opt-out at any time. The CFPB takes reasonable steps to scrub personal information from each complaint that could be used to identify the consumer. | String |
| complaint\_hindi | Text content of the complaint (in Hindi) | String |
| product | The type of product the consumer identified in the complaint | String |
| department | The department responsible for handling the complaint | String |
| sub\_product | The type of sub-product the consumer identified in the complaint | String |
| issue | The issue the consumer identified in the complain | String |
| sub\_issue | The sub-issue the consumer identified in the complaint | String |
| company | Company associated with the complaint | String |
| state | The state of the mailing address provided by the consumer | String |
| Zipcode | The mailing ZIP code provided by the consumer | String |
| company\_response\_consumer | This is how the company responded. For example, "Closed with explanation" | String |
| consumer\_consent\_provided | Identifies whether the consumer opted in to publish their complaint narrative. The narrative is not published unless the consumer consents and consumers can opt-out at any time | String |
| submitted\_via | How the complaint was submitted to the CFPB | String |
| date\_sent\_to\_company | The date when CFPB sent the complaint to the company | String |
| timely\_response | Whether the company gave a timely response | String |
| consumer\_disputed | Whether the consumer disputed the company’s response | String |

**Data Preprocessing Pipeline**

The preprocessing pipeline performs comprehensive data cleaning, filtering, and anonymization on the Consumer Complaint Database from Consumer Financial Protection Bureau (CFPB). The pipeline includes multiple stages to ensure data quality and privacy.

**Preprocessing Steps**

**1. Data Loading**

* Loads the raw dataset from Google Cloud Storage
* Dataset format: Parquet

**2. Basic Filtering**

* Removes records with insufficient word count
* Filters records based on date range (March 19, 2015 to July 28, 2024)
* Converts date fields to proper date format

**3. Language Detection**

* Uses fast\_langdetect (based on fast-text) for language identification
* Implements multi-threaded processing for improved performance
* Retains only English (EN) and Hindi (HI) complaints
* Removes records in other languages

**4. Data Cleaning**

* Converts complaint text to lowercase
* Removes special characters using regex
* Eliminates duplicate records based on:
  + Product
  + Sub-product
  + Complaint text
* Removes records with null values in critical fields:
  + Product
  + Sub-product
  + Department
  + Complaint text

**5. Abusive Content Filtering**

* Removes words from a set of abusive words
* Implements Bloom filter for abusive words set to efficient filter out abusive words
* Replaces abusive words with placeholder text
* Processes text while maintaining sentence structure

**6. PII Data Anonymization**

* Detects and masks personally identifiable information (PII)
* PII types handled:
  + Bank account numbers
  + Routing numbers
  + Credit card numbers (multiple formats)
  + Transaction dates
  + SSN/TIN numbers
  + EIN numbers
  + Passport numbers
  + Email addresses
  + Phone numbers
  + Home addresses
  + Demographic information (race, ethnicity, gender)
  + Transaction amounts

**7. Data Aggregation**

* Joins filtered datasets on complaint ID
* Selects and maintains relevant columns

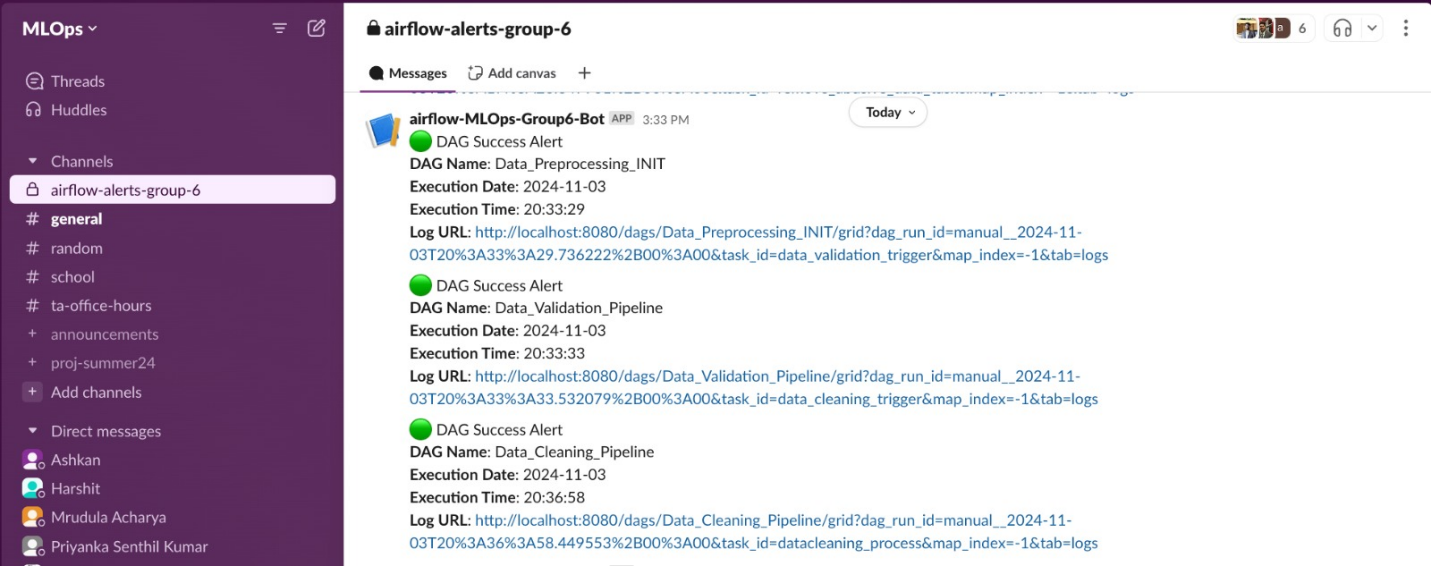
**8. Abusive Content Filtering**

* Removes words from a set of abusive words
* Implements Bloom filter for efficient filtering
* Replaces abusive words with placeholder text
* Processes text while maintaining sentence structure

**9. Success Email Trigger**

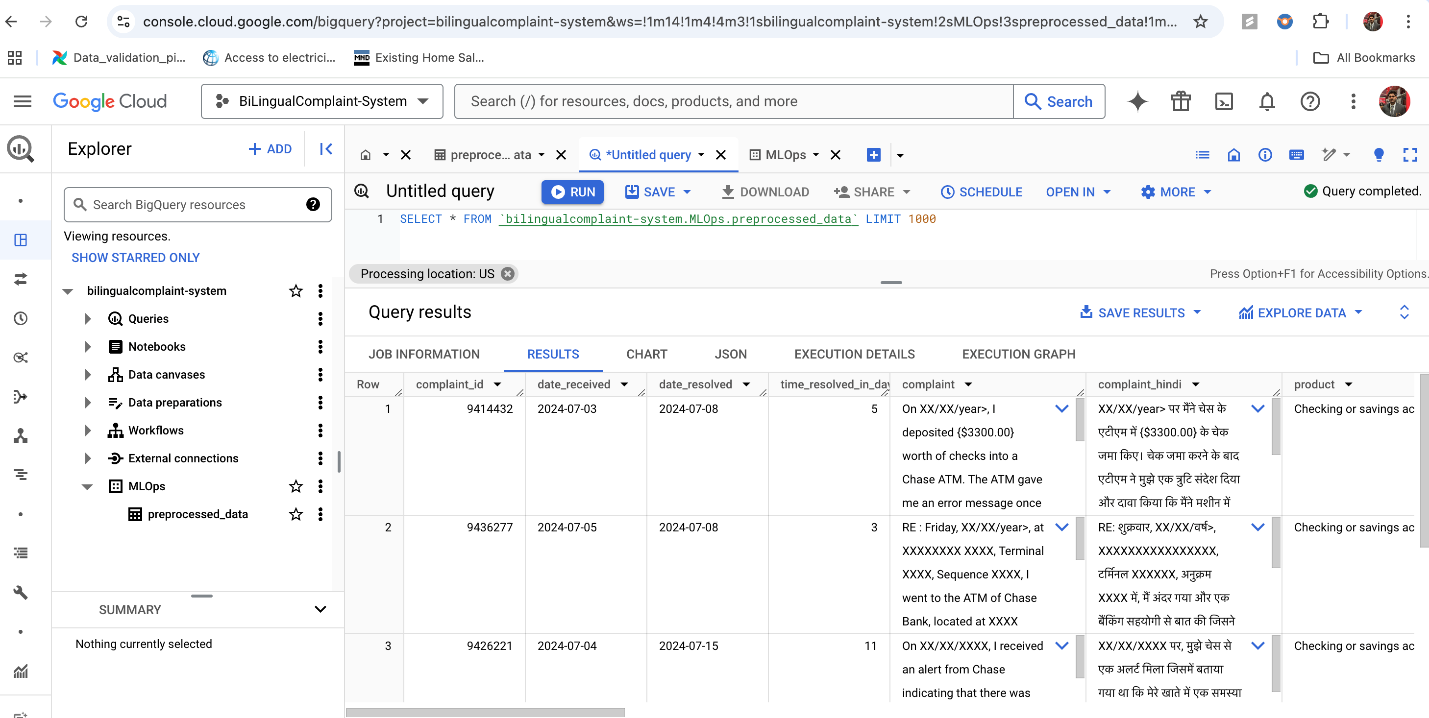
* Once all the pipelines have successfully run, a success email is triggered to notify stakeholders of the pipeline completion

**10. Slack Alerts Integration**

* On every DAG success and failure, we trigger a slack notification a private channel in the MLOps workspace. [](https://private-user-images.githubusercontent.com/86112540/382608589-324c90ac-e6fa-40c5-a61e-8a9e3c138bf6.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..3z7lsscnp8eUrc9irkL8gNlHGhuz_OTqZXf2J9pnajE)

**Output**

* Final preprocessed dataset saved in Parquet format
* Location: data/preprocessed\_dataset.parquet and bilingualcomplaint-system.MLOps.preprocessed\_data(BigQuery)
* Includes comprehensive logging of all preprocessing steps
* Send a pipeline success / failure email

[](https://private-user-images.githubusercontent.com/154296811/382862002-07eed6ff-5853-41f4-8bbb-24d25db45eaf.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJnaXRodWIuY29tIiwiYXVkIjoicmF3LmdpdGh1YnVzZXJjb250ZW50LmNvbSIsImtleSI6ImtleTUiLCJleHAiOjE3MzA3NTQ1OTAsIm5iZiI6MTczMDc1NDI5MCwicGF0aCI6Ii8xNTQyOTY4MTEvMzgyODYyMDAyLTA3ZWVkNmZmLTU4NTMtNDFmNC04YmJiLTI0ZDI1ZGI0NWVhZi5wbmc_WC1BbXotQWxnb3JpdGhtPUFXUzQtSE1BQy1TSEEyNTYmWC1BbXotQ3JlZGVudGlhbD1BS0lBVkNPRFlMU0E1M1BRSzRaQSUyRjIwMjQxMTA0JTJGdXMtZWFzdC0xJTJGczMlMkZhd3M0X3JlcXVlc3QmWC1BbXotRGF0ZT0yMDI0MTEwNFQyMTA0NTBaJlgtQW16LUV4cGlyZXM9MzAwJlgtQW16LVNpZ25hdHVyZT05NDM1ODEwNzRjOGU5NDZkZDYwNmY0NDJjZDZiNzNjMjI4NmE5MTI5MTFiMjQ3NzhjODMzYmI4MWMwZWE2YWEzJlgtQW16LVNpZ25lZEhlYWRlcnM9aG9zdCJ9.oWu89gEvMxkoREw-UMXrQmqujALlJ31sOwFGEQMfPAc)

**Pipeline Orchestration (Airflow DAGs)**

Summarizing entire Airflow Orchestration Graph Below:

A diagram of a data flow

Description automatically generated

A diagram of data cleaning

Description automatically generated

Summarizing entire Airflow Orchestration Graph Below Using Dags:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

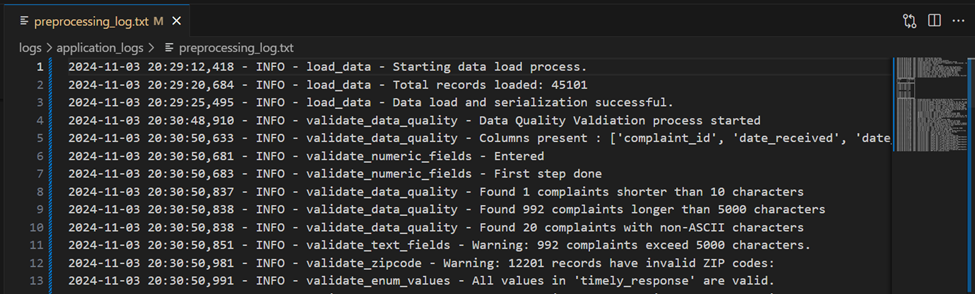
Description automatically generated

**Tracking and Logging**

Our pipeline includes detailed logging to track progress and identify issues about errors during data processing. We use Python's logging library to set up a custom logger that captures important information at each stage of the process.

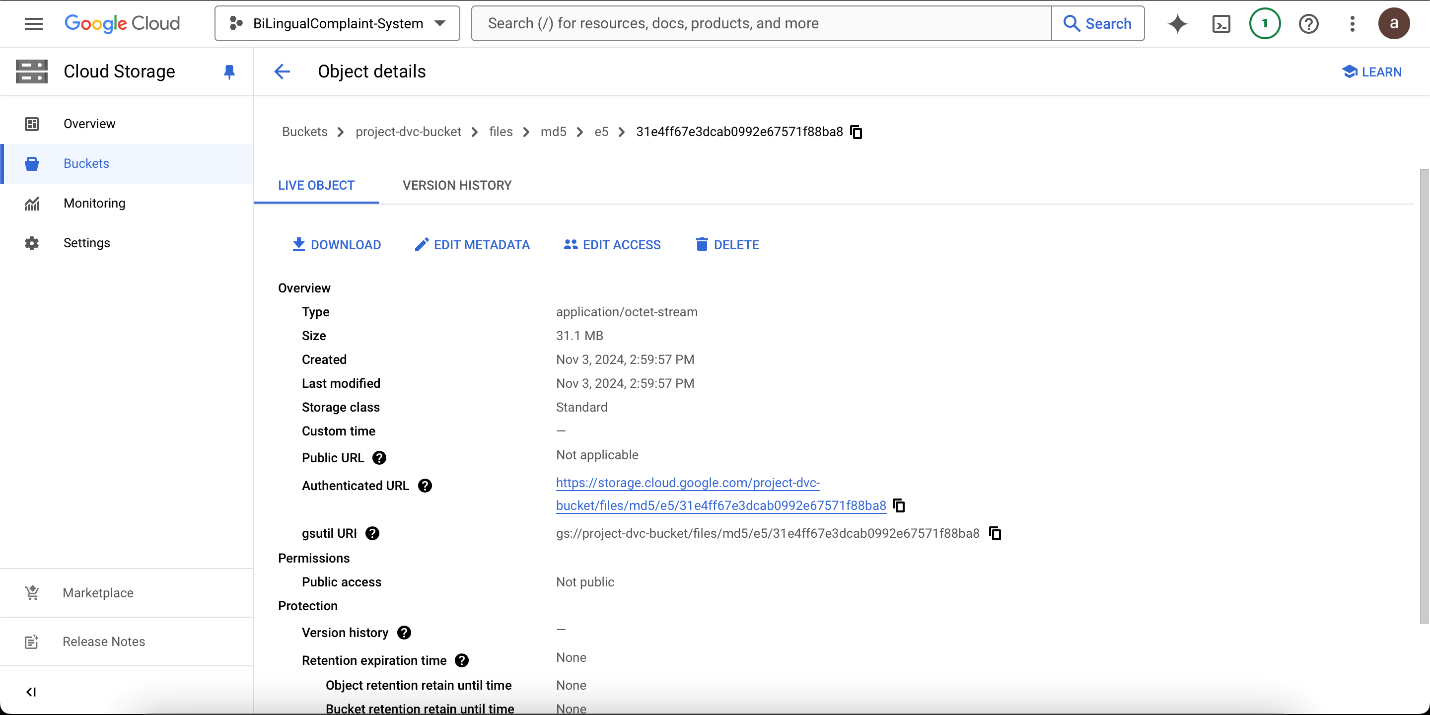
**Key Features:**

* **Progress Tracking**: Logs are generated at each stage of the pipeline, including all functions.
* **Error Monitoring**: Errors are logged with relevant details, making it easier to locate and fix issues quickly.
* **Custom Log Path**: All logs are saved in logs/application\_logs/preprocessing\_log.txt for easy access and troubleshooting.

[](https://private-user-images.githubusercontent.com/86112540/382613463-1de54f85-f5c7-4e31-ac97-1e120b5812eb.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..EjnAmlehx-GH2zkMGyjD0aW8EwwvkQSrkl2i_cjTRHk)

**Data Versioning with DVC**

* DVC (Data Version Control) to manage and version control our datasets throughout the preprocessing pipeline
* Raw data is loaded from Google Cloud Storage (GCS), processed and cleaned using our Airflow pipeline, and the preprocessed data is then stored back to
* Metadata versions are stored in GCS

[](https://private-user-images.githubusercontent.com/143234149/382628378-cd978868-aabe-488a-a295-93d838bc520c.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJnaXRodWIuY29tIiwiYXVkIjoicmF3LmdpdGh1YnVzZXJjb250ZW50LmNvbSIsImtleSI6ImtleTUiLCJleHAiOjE3MzA3NTQ1OTAsIm5iZiI6MTczMDc1NDI5MCwicGF0aCI6Ii8xNDMyMzQxNDkvMzgyNjI4Mzc4LWNkOTc4ODY4LWFhYmUtNDg4YS1hMjk1LTkzZDgzOGJjNTIwYy5wbmc_WC1BbXotQWxnb3JpdGhtPUFXUzQtSE1BQy1TSEEyNTYmWC1BbXotQ3JlZGVudGlhbD1BS0lBVkNPRFlMU0E1M1BRSzRaQSUyRjIwMjQxMTA0JTJGdXMtZWFzdC0xJTJGczMlMkZhd3M0X3JlcXVlc3QmWC1BbXotRGF0ZT0yMDI0MTEwNFQyMTA0NTBaJlgtQW16LUV4cGlyZXM9MzAwJlgtQW16LVNpZ25hdHVyZT1mNTVlNTI0ZjYwY2NlYTZmNmE1YjI1NTEwZDNiOTAwNGUwYTI0MGZlZmJiMTQwNmEzNWYzZDY4OTZhNmI2NjAyJlgtQW16LVNpZ25lZEhlYWRlcnM9aG9zdCJ9.gmRxrM0Yfxz8Bo4OPhkQQe8r1vaKc0PBj0U4_UDEYuw)

**Pipeline Flow Optimization**

* In Filter by Language task, Multi-threading was used to detect the language of the text in parallel. Before multi-threading the time to complete language detection was 3.43 seconds, and after multi-threading the execution time was reduced to 2.28 seconds.
* In Remove Abuse content task, we have a list of abusive words to remove. So to efficiently remove abusive words we made use of Bloom Filters. This reduced our execution time from 1.25 to 1.65 seconds.

Below is the Gantt chart illustrating the pipeline flow after optimization:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Data Schema & Statistics Generation**

Here's a high-level overview of what are the steps we do in the context of feeding quality data to the modelling stage.

### *1. Data Quality Assurance*

* **Text Validation:** Perform several checks on consumer complaint narratives to ensure data quality, as the quality of data directly impacts the performance of text classification models. The following issues are validated:
  + Empty Entries: Checked for any entries that lack text.
  + Extremely Short or Long Complaints: Complaints that are too short or excessively long are flagged for review.
  + Non-ASCII Characters: Identified any characters that fall outside the standard ASCII range.
* **Interdependent Constraint Validation:** Checked for logical consistency between related fields. This step helps maintain the overall quality of the dataset by ensuring that related data points align correctly.
* **Categorical Data Validation:** Verified that categorical values (some of which serve as output labels) contain valid entries. This ensures that the model is trained on appropriate and meaningful labels.

### *2. Statistics Generation and Schema Analysis*

* **Data Statistics Generation:** Generating comprehensive statistics from the DataFrame containing complaint data using TensorFlow Data Validation (TFDV). These statistics offer insights into the distribution and characteristics of the data, serving as a foundational reference for quality assessment.
* **Outlier Detection:** Identified unusually long complaints that could skew the classification model or indicate special cases needing separate handling.
* **Class Distribution Analysis:** Examined the distribution of complaints across different categories (e.g., products, issues). Understanding class imbalance is vital, as it may require techniques like oversampling or undersampling to ensure effective model training.
* **Temporal Analysis:** Analyzed yearly trends in complaint volumes and categories. This information can be leveraged to create time-based features and understand whether the classification model needs to account for temporal shifts in complaint patterns.
* **Duplicate Detection:** Identified repeated complaints to prevent data leakage and ensure that the classification model does not overfit to duplicated data.

**Anomaly Detection & Alerts**

Incorporated a comprehensive data quality assessment and anomaly detection process tailored for consumer complaint data ensuring that our data meets predefined quality standards.

### 1. Anomaly Detection

* **Schema Inference:** Inferred a schema from the generated statistics, defining the expected data structure, types, and constraints. This schema acts as a reference for validating incoming datasets against established business rules.
* **Anomaly Detection:** Checked for anomalies in the dataset by validating it against the inferred schema. It filters specific columns that we're interested in (e.g., complaint details, department, product) to focus on critical areas, logging any detected issues.

A white rectangular object with text

Description automatically generated with medium confidence

* **Anomaly Resolution:** To fix the detected anomalies, relaxed the minimum and maximum fraction constraints for values that must come from the domain for the features where anomalies were identified. This adjustment allowed us to accommodate legitimate data variations.
* **Schema Freezing:** Once the anomalies were fixed, inferred a schema that defined the expected data structure, types, and constraints. We then "froze" this schema and stored it in a file, creating a consistent reference point for validating incoming data when our application is released to the public. Freezing the schema is crucial as it ensures that any data processed in production adheres to the established rules and standards, helping us maintain high data quality and reliability.

### 2. Alerts

* **Alerts for Anomalies:** To ensure prompt communication, alerts were generated and sent via email whenever anomalies were detected in the data. This proactive approach allowed us to stay informed and address / fix potential anomalies swiftly.

A screenshot of a computer

Description automatically generated

**Data Bias Detection and Mitigation**

The project routes customer complaints to the correct product and department based solely on complaint narratives, with demographic or personally identifiable information (PII) redacted to ensure privacy and mitigate demographic bias. This approach aligns with responsible ML fairness principles, ensuring that the model is not biased on demographic factors like location, gender, religion, ethnicity etc.

**1. Detecting Bias in Data:**

* We evaluate model fairness across complaint-related categories, focusing on product, department, to detect performance imbalances. We’ll record each type of bias discovered, the slice affected, and the metrics indicating bias. The bias mitigation technique will be documented with explanations, parameters used, any trade-offs and reasoning.

**2. Data Slicing:**

* We will use Fairlearn's MetricFrame to define and evaluate slices of data once predictions are available. MetricFrame enables us to track performance across slices and detect bias by calculating metrics like accuracy etc for each slice.

**3. Bias Mitigation:**

If significant discrepancies appear across slices:

* Re-sampling Underrepresented Groups: If some slices have fewer instances, we’ll balance them by oversampling (duplicating underrepresented entries) or under-sampling (reducing overrepresented entries).
* Once the ML model is built, fairness constraints through Fairlearn can be applied to optimize for balanced performance across slices, ensuring no slice disproportionately affects routing accuracy.
* For certain slices, we may apply customized decision thresholds to improve performance for groups with historically lower accuracy.

## **Preprocessing Module Unit Tests**

* Unit tests have been developed to validate the functions in preprocessing.py, covering data loading, record filtering, language detection, abusive content removal, and data aggregation processes.
* These tests ensure correct data transformation and filtering logic is applied to meet preprocessing requirements.

## **License Information**

The Consumer Financial Protection Bureau (CFPB) data is open to the public under the OPEN Government Data Act, enabling transparency and broad accessibility. Users should follow privacy guidelines, particularly around any personally identifiable information (PII) within consumer data.

For further details, refer to the [CFPB Data Policy](https://www.consumerfinance.gov/data/).