Comprehensive Documentation:

Machine Learning Model Pipeline

1. Overview

This section introduces the project's objective: building an end-to-end pipeline for machine learning using GCP. The pipeline incorporates preprocessing, automated workflows with Cloud Composer, and model training and inferences with Vertex AI.

Objective:

- Transition from local data preprocessing to cloud-based orchestration.
- Develop and evaluate machine learning models using both AutoML and custom built ML models.
- Ensure reproducibility, scalability, and fairness in model development.

2. Data Loading

2.1. Implementing Airflow DAG using Cloud Composer

- The transition from a local processing environment to the cloud was facilitated using Google Cloud Composer. After preprocessing the data locally in Visual Studio Code, it was essential to ensure scalability and automation, prompting the move to Google Cloud Platform (GCP). The data was uploaded to Google Cloud Storage (GCS), which served as a centralized repository for storing raw and processed data. The use of Cloud Composer enabled the orchestration of complex workflows through Airflow's Directed Acyclic Graphs (DAGs). These DAGs automated the execution of preprocessing modules, ensuring seamless integration with the cloud infrastructure. Additionally, all dependencies and Python libraries required for preprocessing were mirrored in the Composer environment, maintaining consistency with the local setup. This transition allowed the preprocessing pipeline to handle larger datasets with greater efficiency while ensuring that every step was reproducible and scalable for future tasks.

3. Cloud Composer Workflow

Google Cloud Composer was employed to manage the data processing workflow by automating tasks and ensuring seamless integration of preprocessing modules developed locally. By leveraging Composer's orchestration capabilities, complex workflows were structured into easily manageable steps, ensuring reproducibility and scalability. This section outlines the setup of

Cloud Composer, the details of DAG 1 for data processing, and an overview of challenges encountered along with their resolutions.

3.1. Setting Up Cloud Composer

- The primary purpose of setting up Cloud Composer was to orchestrate complex workflows using Airflow's Directed Acyclic Graphs (DAGs). This setup provided a robust, managed environment for automating the data processing pipeline. To ensure compatibility, the Composer environment was configured with all necessary google cloud services, such as google-cloud-aiplatform, google-cloud-storage
- . Additionally, integration with Google Cloud Storage (GCS) allowed seamless data ingestion and output handling, ensuring that the environment was ready for executing DAGs designed to process and transform data efficiently.

3.2. DAG 1: Data Processing Workflow

The first DAG (DAG 1) implemented a fully automated data processing pipeline. It orchestrated the flow of tasks required to clean and preprocess raw data stored in Google Cloud Storage (GCS).

Pipeline Steps:

1. **Load Raw Data**: The workflow began by ingesting raw data files directly from GCS buckets into memory. This step ensured compatibility with the processing pipeline and validated the initial state of the data.
2. **Integrate Local Modules**: Custom Python modules previously developed locally were integrated into the DAG. These modules executed the preprocessing steps, including:
    - Feature engineering to derive additional attributes.
    - Cleaning operations to handle missing values, duplicates, and inconsistencies.
3. **Save Processed Data**: The cleaned and transformed datasets were written back to GCS in a structured format, ensuring they were ready for subsequent machine learning tasks.

Outputs:

The DAG produced fully cleaned and preprocessed datasets stored in GCS. These outputs maintained consistency and quality, making them suitable for downstream operations like training machine learning models in Vertex AI.

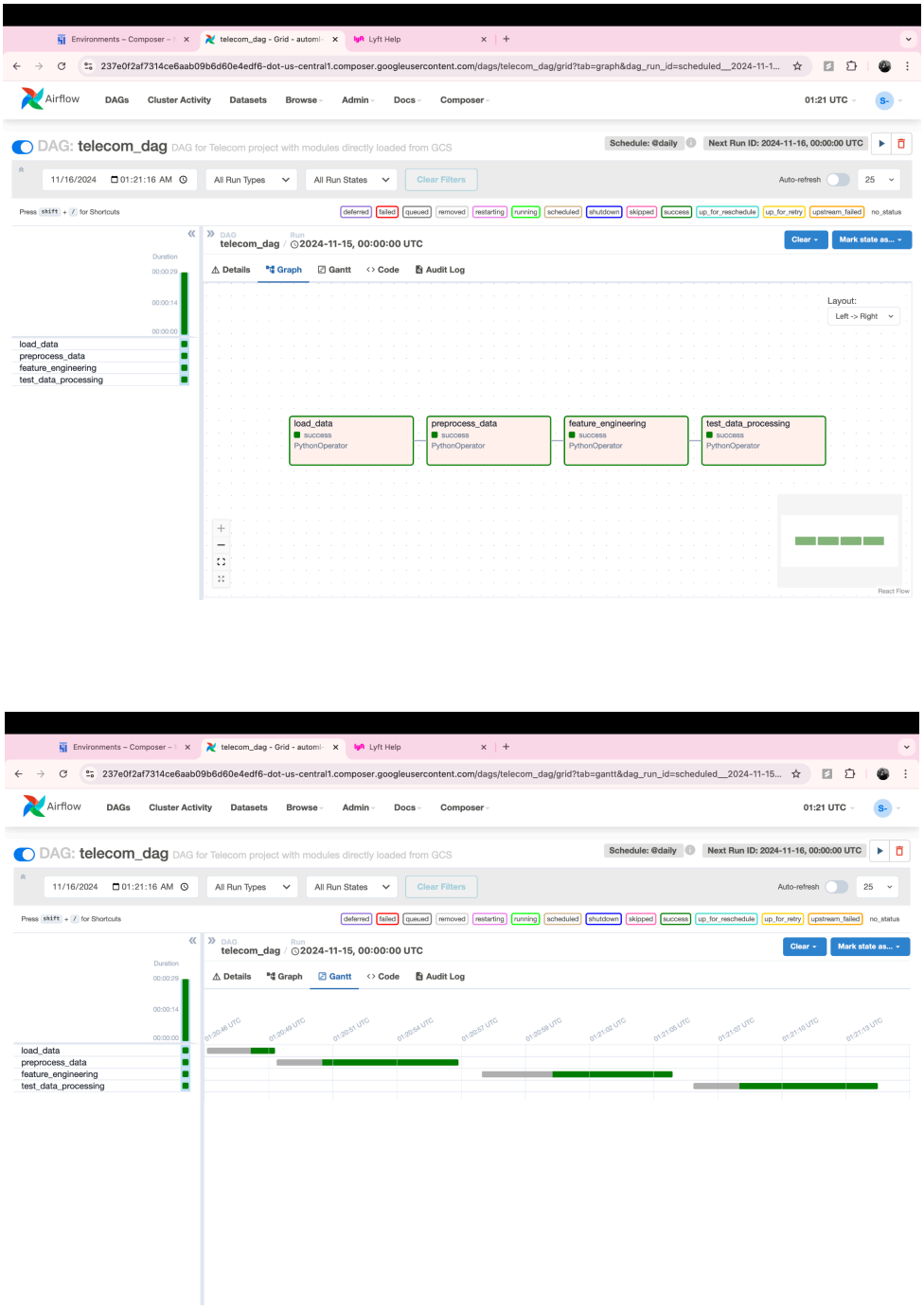


Fig: Data pipeline on cloud composer

4. Model Training on Vertex AI

The model training process on Vertex AI involved multiple approaches to leverage AutoML capabilities and custom model development, each offering distinct benefits and limitations. Vertex AI provided a streamlined environment for training, evaluating, and deploying machine learning models with minimal manual intervention while maintaining flexibility for custom workflows. Below is a detailed explanation of the approaches used.

4.1. Approach 1: AutoML via Colab Notebooks

To gain greater control over the AutoML process, a second approach involved writing Python scripts in Google Colab to interact with the Vertex AI APIs. These scripts automated tasks such as uploading data, initiating training jobs, and retrieving evaluation metrics. Long-running training tasks (typically lasting over two hours) were scheduled on GCP to optimize resource usage and manage workflow efficiently.

Benefits:

- This approach provided more control over data preprocessing, model configuration, and evaluation compared to the UI-based AutoML.
- Customizability allowed fine-tuning of AutoML parameters to better suit the specific problem.

Challenges:

- Scheduling long-running tasks on GCP required careful resource planning to avoid interruptions or inefficiencies.
- Triggering longrunning notebooks is not possible using cloud functions.
- The extended runtime for training necessitated optimized infrastructure and monitoring.

.

4.2. Approach 2: Custom Model Development

For greater flexibility, a custom model was developed using XGBoost. This approach involved manually training the model on the preprocessed dataset, optimizing hyperparameters through trial and error, and evaluating performance. The entire process required coding expertise to design and implement the workflow.

Workflow:

- The dataset was loaded from Google Cloud Storage (GCS), and preprocessing steps were applied programmatically to ensure data consistency and quality.
- XGBoost was used as the model framework, and hyperparameters were tuned manually to achieve the best possible results.
- Once trained, the model was Dockerized to create a portable, consistent runtime environment for execution.

- The Docker image was built and pushed to Google Container Registry (GCR) to facilitate easy access.
- In Google Colab, the Docker image was pulled from GCR and run to execute the custom-built machine learning code for XGBoost, ensuring scalability and reproducibility across different environments.

Drawbacks:

- The model's accuracy was limited to ~65%, which was suboptimal compared to AutoML outputs.
- There was no automated generation of performance metrics, requiring manual implementation of validation and visualization tasks.
- Dockerization and deployment added complexity to the workflow, increasing the time and resources needed for completion.

4.3. Approach 3: Airflow-Integrated AutoML

This approach combined the automation of AutoML with the workflow orchestration capabilities of Airflow. A new DAG was designed to handle the entire AutoML process seamlessly within Cloud Composer. DAG operators were programmed to check for existing models and initiate training only if necessary. This logic optimized resource usage by reusing pre-trained models whenever available, avoiding redundant computations.

Pipeline Steps:

1. **Model Check**: The DAG first checked whether a pre-trained model existed in the GCP Model Registry.
2. **Training Execution**: If no model was found, AutoML training was triggered using Vertex AI APIs using CreateAutoMLTabularTrainingJobOperator class
3. **Model Evaluation**: After training, the DAG integrated model evaluation metrics to select the best-performing model for deployment.

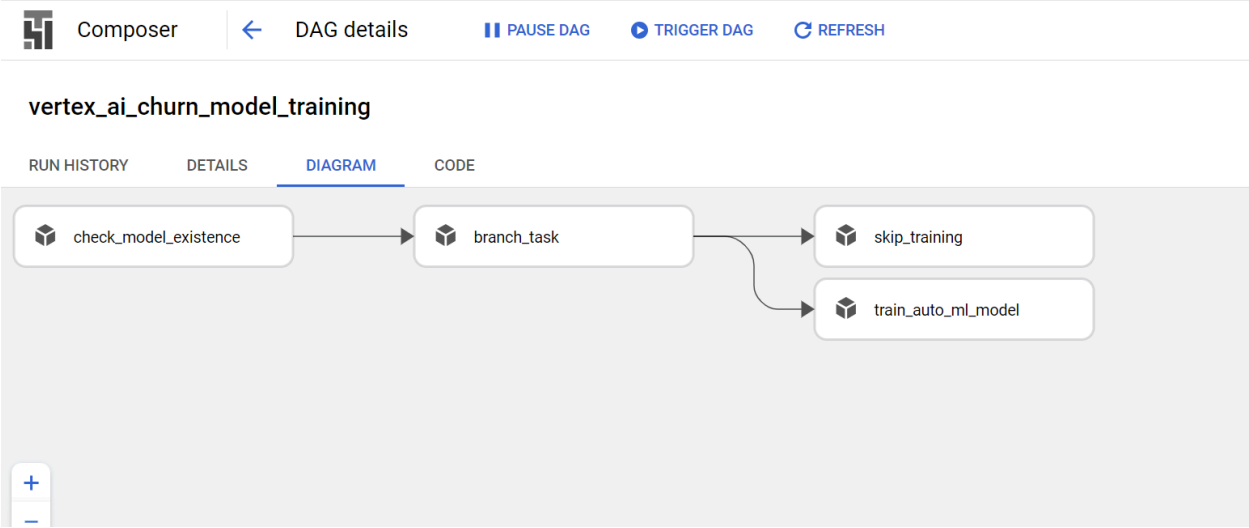
Advantages:

- This approach fully automated the training workflow, reducing manual intervention and improving efficiency.
- The integration of AutoML with Airflowcomposer ensured a seamless connection between preprocessing, training, and evaluation tasks.

4.4 Conditional Model Utilization Workflow

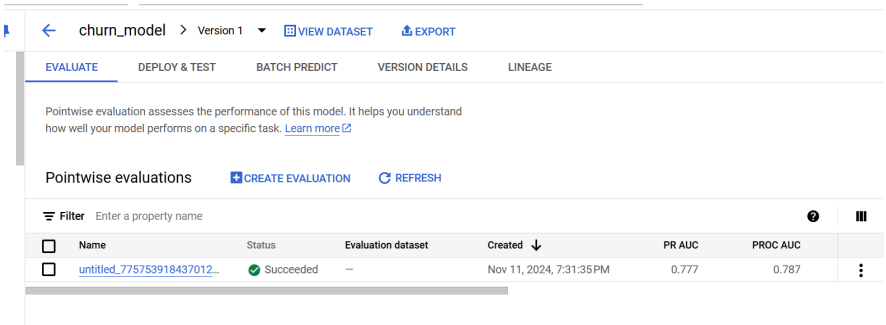
The workflow is designed to first check the Model Registry for an existing trained model before proceeding with any training steps. If a trained model is found in the registry, the pipeline skips the training process and directly uses the pre-existing model for predictions or further tasks. This

approach saves time and computational resources by leveraging previously validated models. However, if no trained model is available in the registry, the pipeline automatically initiates the training process. It begins by splitting the available data into training and testing subsets, ensuring an appropriate validation strategy. The model is then trained using the training data, evaluated on the test set, and registered in the Model Registry for future use, maintaining a seamless and efficient model lifecycle.



4.5 Notification and alerts

Once the model is trained and batch predictions are completed, an email notification is automatically sent to confirm the successful completion of the process.



EVALUATE | DEPLOY & TEST | BATCH PREDICT | VERSION DETAILS | LINEAGE

untitled_77···4370120910 | COMPARE | CREATE EVALUATION

Threshold target

Filter

All labels	0.777
0	0.833
1	0.461

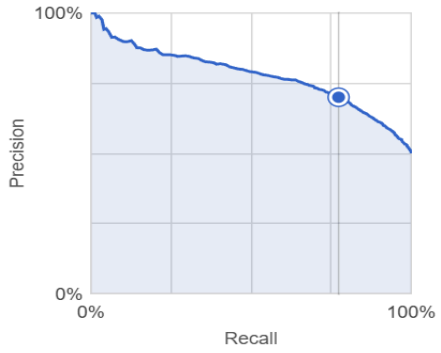
Evaluation details

Confidence threshold 0.5

All labels

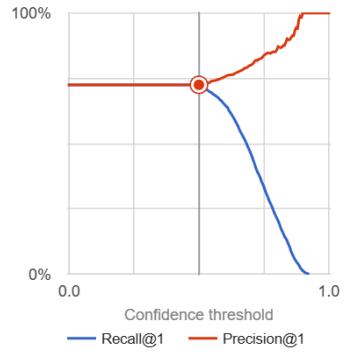
PR AUC	0.777
ROC AUC	0.787
Log loss	0.557
Micro-average F1	0.7250704
Macro-average F1	0.5060168
Micro-average precision	72.5%
Micro-average recall	72.5%
Total items	0
Training items	0
Validation items	0
Test items	0

Precision-recall curve



ROC curve

Precision-recall by threshold

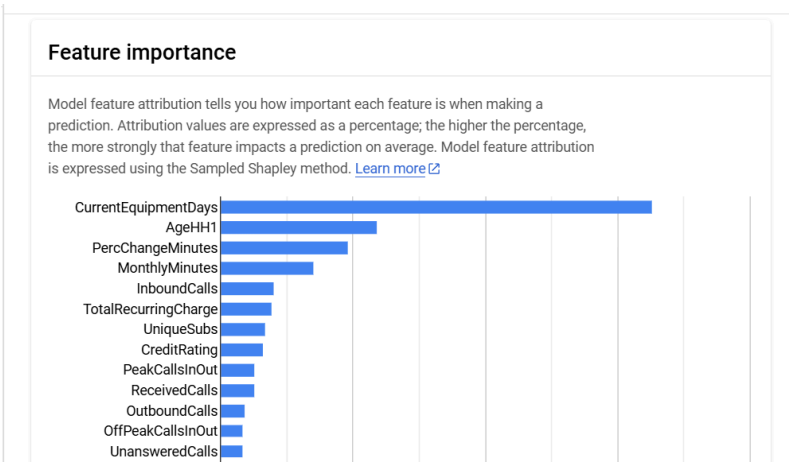


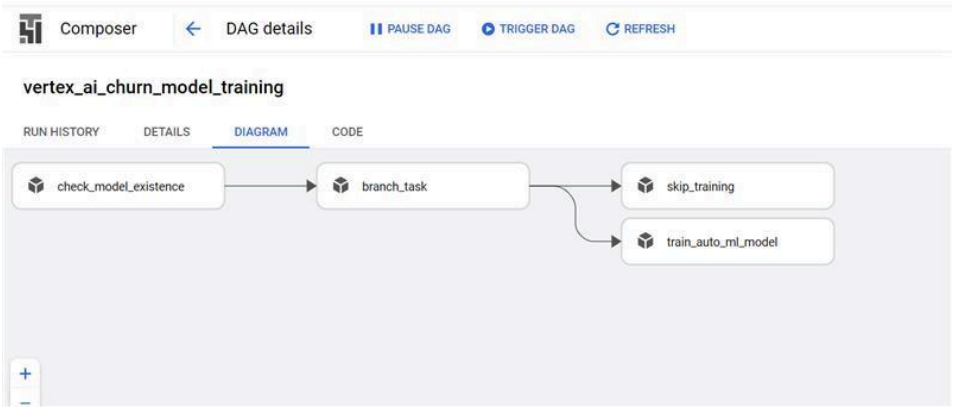
Confusion matrix

Item counts

A confusion matrix shows how the model classified each label in the evaluation dataset. The blue, bold cells indicate a correct prediction. A data item is moved to the dropped column if it does not meet the confidence threshold for any label.

True label \ Predicted label	0	1	Dropped
0	**98%**	2%	0%
1	**90%**	10%	0%

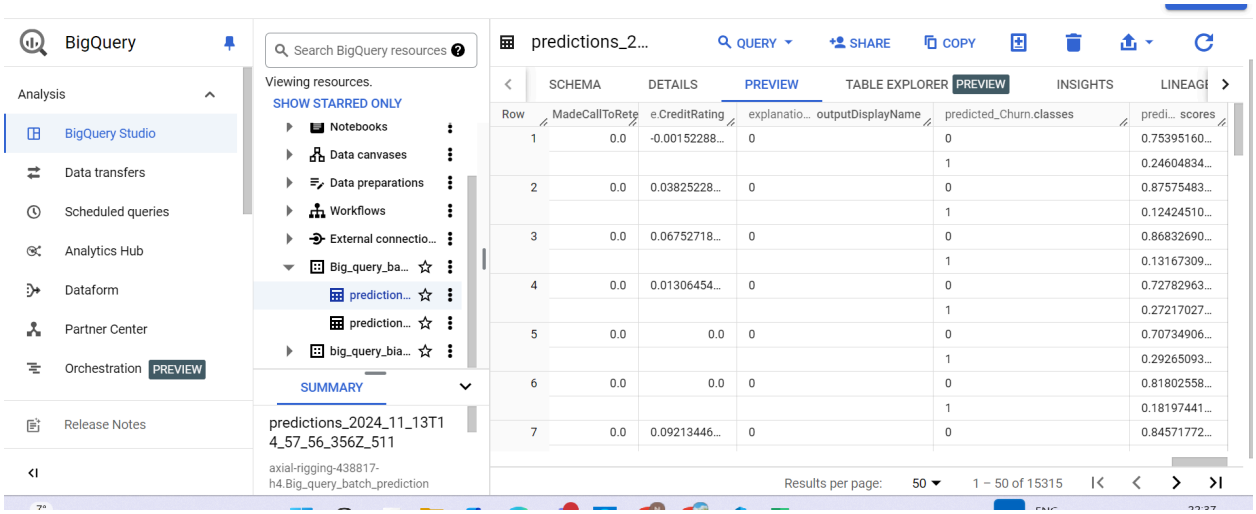




5. Model Validation and Bias Detection

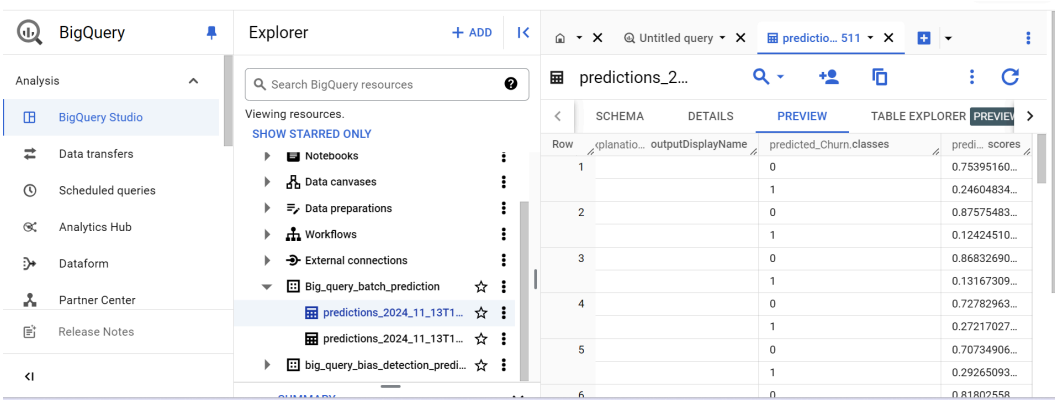
5.1. Validation

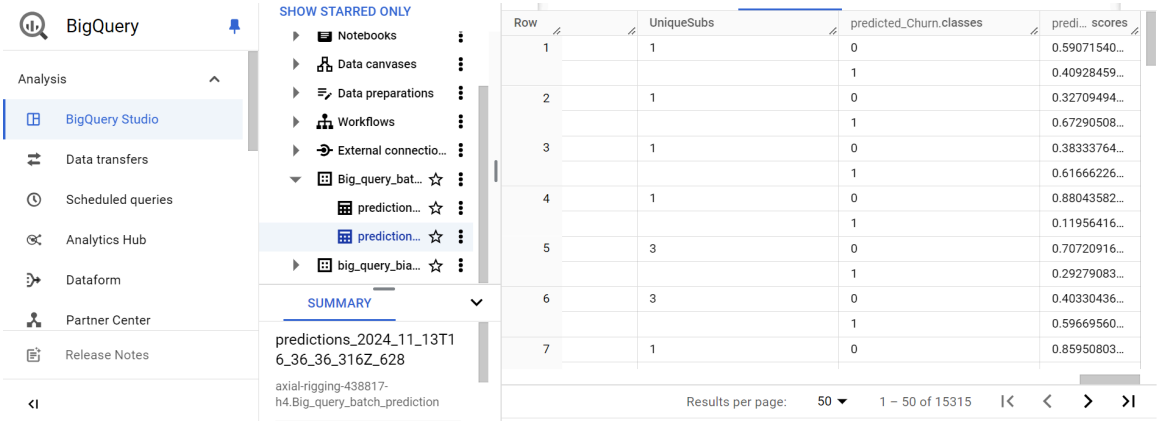
- For the validation of the dataset, AutoML handled the entire process seamlessly, ensuring a robust and efficient workflow. It performed dataset validation to check for inconsistencies, missing values, and other issues before proceeding with model training. Additionally, AutoML conducted hyperparameter tuning to optimize the model's performance by systematically adjusting parameters to achieve the best results. Once training was complete, AutoML evaluated multiple candidate models and automatically selected the best-performing model based on metrics such as precision, recall, F1 score, and ROC AUC. This comprehensive process eliminated the need for manual intervention while ensuring that the most accurate and reliable model was chosen for deployment.



5.2 Inferences after model validation

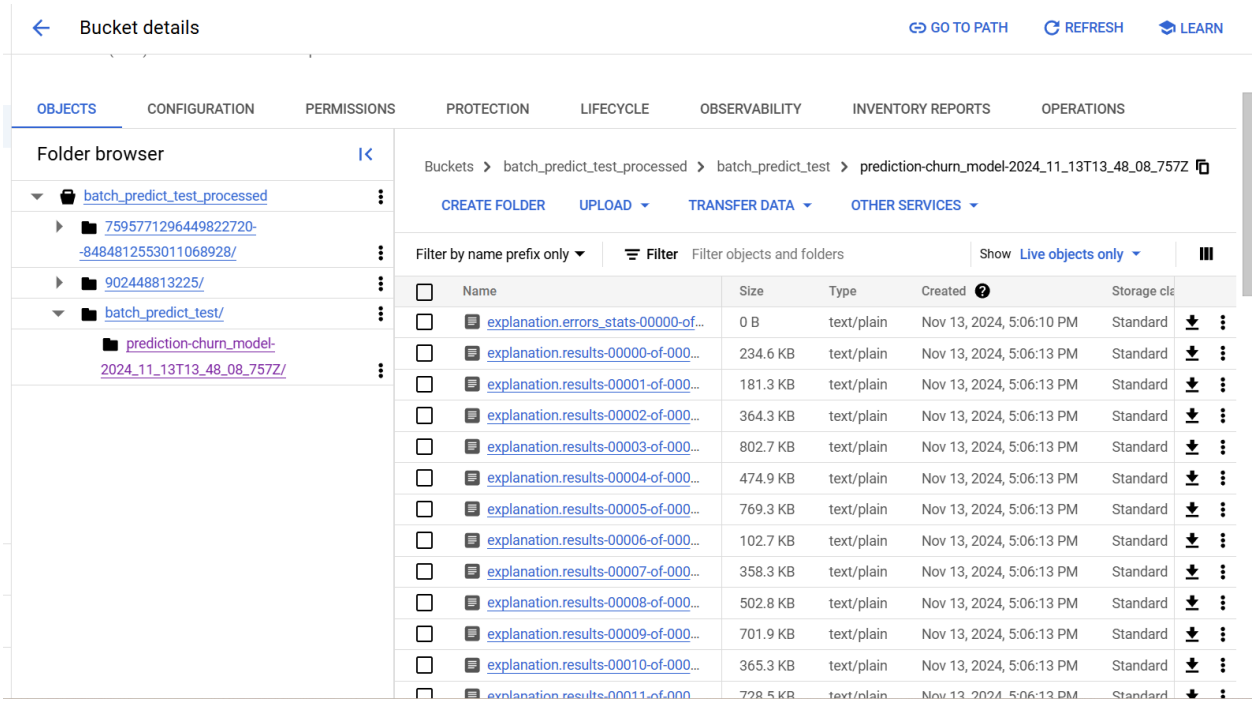
After the model was validated, a test set was used to generate model inferences in batches. These batch inferences were then efficiently processed and stored in BigQuery, Google Cloud's data warehouse, for further analysis and integration with downstream systems. By leveraging BigQuery, the predictions could be queried and analyzed at scale, ensuring that the results were both accessible and actionable for subsequent workflows.





5.3. Sensitivity Analysis

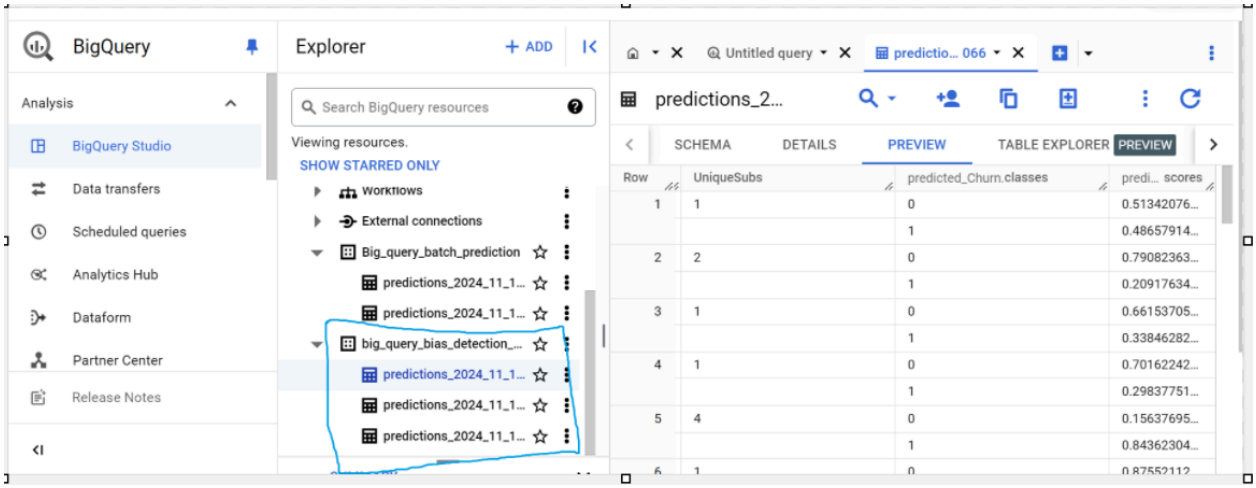
Sensitivity analysis was performed using Vertex AI AutoML to evaluate the robustness of the model and understand the impact of various and hyperparameters. Vertex AI AutoML's automated hyperparameter tuning was leveraged to optimize the model without manual intervention, ensuring that the best configuration was achieved for accurate predictions. The built-in explanation tools provided by Vertex AI were utilized to analyze feature importance, identifying how individual features influenced the model's predictions. SHAP (SHapley Additive exPlanations) values were generated to quantify the contribution of each feature to the predictions, offering insights into their relative significance. Additionally, Vertex AI's sensitivity analysis capabilities were employed to examine the relationship between features and the model's outcomes. The results of this analysis, including detailed explanations and visualizations, were saved in a designated Google Cloud Storage (GCS) bucket for further review and presentation. Key features critical to optimizing the model's accuracy and robustness were identified through this comprehensive analysis.



5.4. Bias Detection

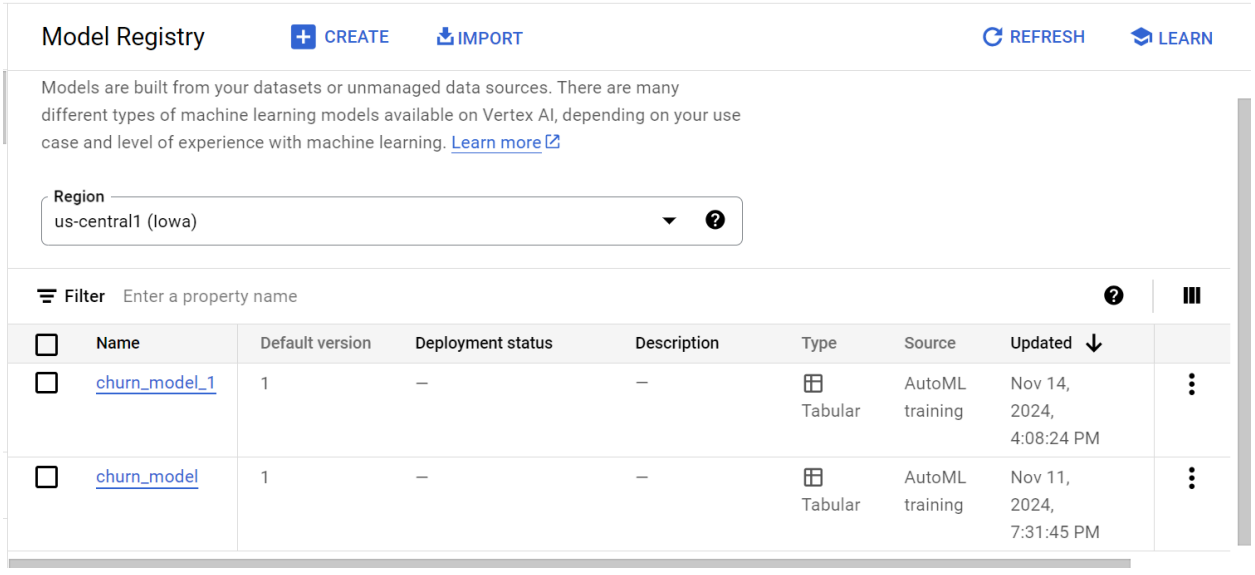
For bias detection, we focused on the `CurrentEquipmentDays` column by dividing it into three slices to evaluate the model's behavior across different groups. These slices

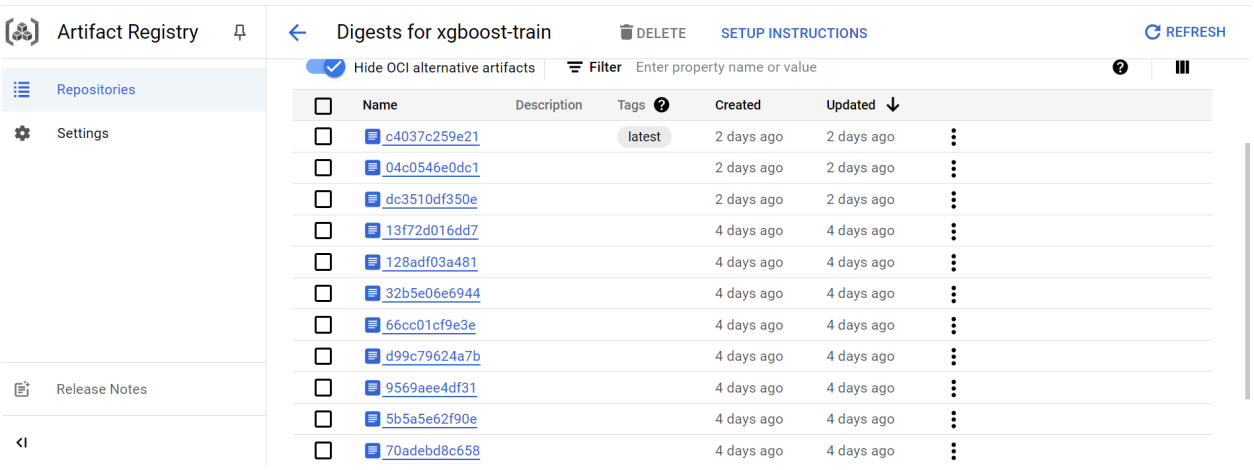
represented distinct ranges or categories within the column, allowing us to assess whether the model's predictions varied significantly between them. The results of these bias detection analyses, including predictions and inferences for each slice, were stored in a **BigQuery** table. This enabled efficient querying and deeper exploration of potential disparities, ensuring that the model's outputs were fair and consistent across the defined groups.



6. Experiment Tracking and Hyperparameter Tuning

- Vertex AI efficiently managed experiment tracking and streamlined the entire machine learning workflow. It maintained comprehensive logs of hyperparameter configurations, model performance metrics, and results visualizations, such as confusion matrices and feature importance graphs, ensuring that every experiment was traceable and reproducible. Additionally, Vertex AI facilitated robust model versioning by automatically saving and organizing trained models in a centralized model registry. This registry provided easy access to past versions, enabling comparisons and ensuring reproducibility across iterations. Furthermore, Vertex AI handled artifact management by maintaining an artifacts registry, which included datasets, trained models, and other resources essential for experiment tracking and deployment. This cohesive ecosystem ensured a seamless and well-documented development process.





Conclusion

This project successfully demonstrates the development of a fully cloud-native machine learning pipeline, eliminating the need for any local dependencies. By leveraging Google Cloud Platform (GCP) services such as Cloud Composer, Vertex AI, BigQuery, and Google Cloud Storage, all components of the pipeline—from data preprocessing to model training and deployment—were executed seamlessly on the cloud. The transition from local workflows to GCP ensured scalability, efficiency, and reproducibility, enabling the handling of large datasets and complex processes without relying on local environments.

The use of Cloud Composer to orchestrate Airflow workflows streamlined the data preprocessing and model training tasks, integrating seamlessly with GCP's managed services. Vertex AI facilitated robust model development, including AutoML and custom models, while

maintaining centralized experiment tracking, model versioning, and artifact management. The pipeline was designed to operate entirely within the cloud ecosystem, ensuring that all data, models, and results were processed, stored, and accessed through GCP services.

This cloud-centric approach highlights the scalability and flexibility of modern cloud-based workflows, showcasing the ability to manage end-to-end machine learning processes without any reliance on local systems. By fully utilizing GCP's ecosystem, this project establishes a robust, reproducible, and future-proof framework for deploying advanced machine learning models in a cloud-native environment.