End-to-End Data Management Pipeline for Machine Learning

Group 99

Name	BITS ID
Ayush Chauhan	2023DC04080
Ashish Mishra	2023DC04229
Chenduluru Surya Vamsi	2023DC04369
Srisailapu Jayaram Manideep	2023DC04081

Video Demonstration:
https://drive.google.com/file/d/1cVgSPk__UHxalRDZA0lB2ATjTJh2Knv/view?usp=drive_link

Data Ingestion: -

- **Data Sources:** The script ingests customer churn data from a **CSV file** and an **API providing a ZIP file**.
- **Automation & Storage:** Data is fetched, processed, and stored in a **timestamped raw format** for tracking.
- **Error Handling:** Exceptions are handled gracefully, logging errors for failed ingestion attempts.
- **Logging & Monitoring:** All ingestion events (success/failure) are recorded in a log file for monitoring.

Raw Data Storage: -

- **Storage System:** The script stores raw ingested data in a **local data lake** for structured organization.
- **Efficient Organization:** Data is **partitioned by source and timestamp (YYYY/MM/DD)** for easy retrieval.
- **Validation & Error Handling:** Empty files are skipped, and errors during storage are handled gracefully.
- **Automation & Scalability:** The folder structure ensures **scalability** and maintains a **clean data hierarchy**.

Data Validation: -

- **Data Quality Checks:** The script validates **missing values, duplicates, data types, outliers, and anomalies** in customer churn data.
- **Automated Validation:** It uses **pandas** to systematically assess numeric and categorical columns for inconsistencies.
- **Comprehensive Reporting:** A **data quality report** is generated in CSV format, summarizing detected issues.
- **Actionable Insights:** The report provides a structured view of data quality, aiding in **error resolution** before further processing.

Data Preparation: -

I used **Pandas**, **NumPy**, **Scikit-learn**, and **Seaborn** for data preparation and cleaning:

- **Handling Missing Values:**

  - For numerical columns: Imputed missing values using mean() or median().
  - For categorical columns: Imputed missing values using mode() or used a placeholder "Unknown".
- **Standardizing and Normalizing Numerical Attributes:**
  - Applied MinMaxScaler from **Scikit-learn** to normalize numerical data.
  - Ensured values were scaled between 0 and 1 for consistent model performance.
- **Encoding Categorical Variables:**
  - Applied LabelEncoder for binary categorical variables like gender.
  - Used OneHotEncoder for multi-class categorical variables like country.
- **Exploratory Data Analysis (EDA):**
  - Used **Seaborn** and **Matplotlib** for visualization:
    - Histograms to identify data distribution.
    - Box plots to detect outliers.
    - Correlation heatmaps to understand feature relationships.

Data Transformation and Storage: -

I implemented feature engineering and transformation using **Pandas** and **NumPy**:

- **Created Aggregated Features:**
  - Calculated total customer spend using groupby() and sum().
  - Derived customer tenure using join_date and today's date.
- **Derived New Features:**
  - Created Churn_Risk_Score by combining customer engagement metrics.
  - Calculated Activity Frequency based on login and purchase behaviour.
- **Scaling and Normalizing Features:**
  - Applied StandardScaler and MinMaxScaler for consistent scaling.
- **Stored Transformed Data in a Database:**
  - Designed an SQL schema to store transformed data.
  - Connected to **SQLite** using sqlite3 and stored the transformed data.
  - Wrote sample SQL queries to retrieve and analyze the data.

Feature Store: -

- **Feature Store Setup:** A **SQLite database** is used to store engineered features, metadata, and versioning information.
- **Feature Transformation:** Categorical features are **one-hot encoded**, and the transformed data is stored in the **feature_data table**.
- **Metadata Management:** The **feature_metadata table** maintains descriptions, sources, and versions for each feature.
- **Dataset Versioning:** The **dataset_versions table** logs dataset updates with timestamps for reproducibility.
- **Scalability:** The structure ensures easy **retrieval, automation, and tracking** of feature versions for ML training.

Data Versioning: -

- **GitHub Repository:** All dataset versions are stored in [this repository].
- **Version Control:** Git and Git LFS are used to track raw and transformed datasets.

- **Commit History:** Each dataset update is committed with version tags and descriptions.
- **Reproducibility:** Ensures traceability and rollback for consistent data processing.

Model Building: -

1. **Data Loading & Preprocessing**
   a. Data is loaded from the **feature store (SQLite)**.
   b. **Customer IDs are removed** as they are not predictive features.
   c. Features (**X**) and target (**y = churn**) are separated.
   d. **Train-Test split** (80-20) with **stratification** to maintain class balance.
   e. **Standardization** is applied to numerical features.
2. **Experiment Tracking with MLflow**
   a. **MLflow experiment** is created for tracking.
   b. Train & Evaluate Models:
      i. **Logistic Regression** and **Random Forest** are trained.
      ii. **Metrics logged:** Accuracy, Precision, Recall, F1 Score, and ROC-AUC.
   c. **ROC Curve is generated and saved** for each model.
   d. **Trained models are logged** using mlflow.sklearn.log_model().
3. **Final Deliverables**
   a. **Trained models** (Logistic Regression & Random Forest).
   b. **Performance metrics** (saved in MLflow).
   c. **ROC curves for visualization**.
   d. **Model artifacts versioned in MLflow** for reproducibility.
4. **Summary of Model Performance**
   a. **Logistic Regression:** High accuracy (81.80%) but poor recall (22.36%), meaning it misses many churn cases.
   b. **Random Forest:** Better overall, with 85.95% accuracy and improved recall (43.98%), making it a stronger predictor.
   c. **Key Insight:** Random Forest outperforms Logistic Regression in detecting churn while maintaining good precision.
   d. **Feature Importance:** Random Forest's higher F1-score (56.03%) suggests it captures complex patterns better.
   e. **Business Impact:** Improving recall further is crucial to reducing customer churn and increasing retention.
   f. **Next Steps:** Optimize Random Forest with hyperparameter tuning, improve feature engineering, and explore boosting models like XGBoost.

Orchestrating the Data Pipeline: -

This DAG automates the **customer churn prediction pipeline** with Airflow.

Key Features

✅ **Dynamically Executes Scripts**

- Each folder (1_data_ingestion, 2_raw_data_storage, etc.) contains an index.py script.
- The DAG **dynamically executes** these scripts sequentially.

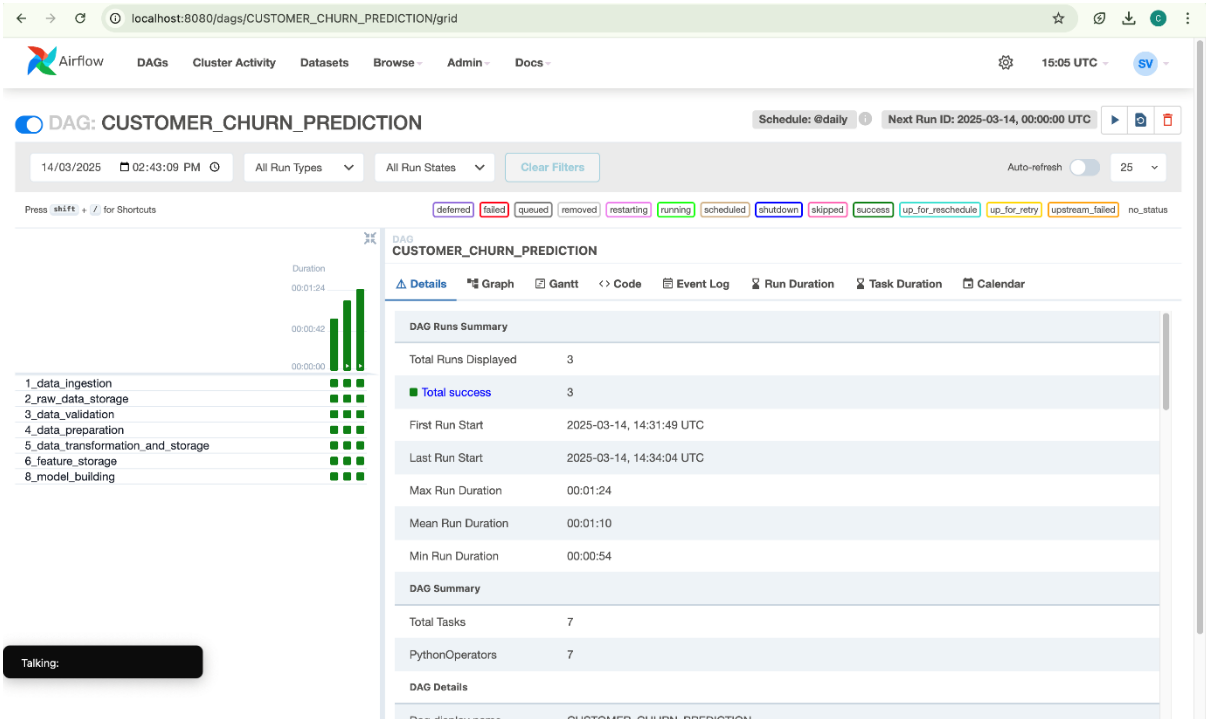
✅ DAG Configuration

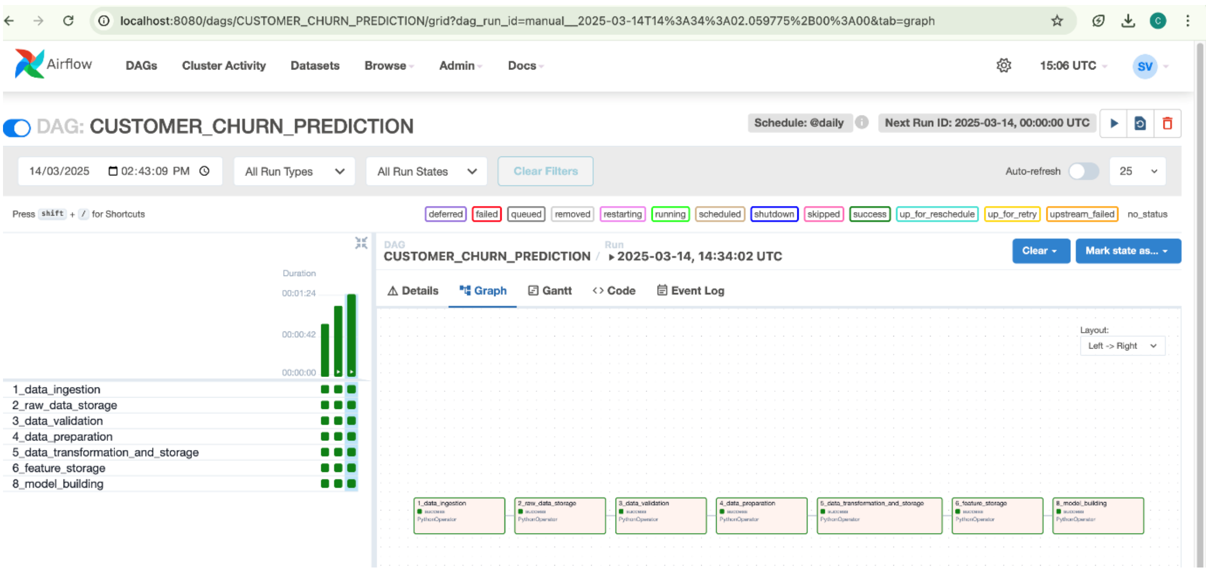
- **Owner:** airflow
- **Start Date:** March 14, 2024
- **Schedule Interval:** @daily
- **Catchup Disabled:** Ensures backfill does not happen.

✅ Task Dependencies

- Each step **executes in sequence**, ensuring proper workflow.

☐ Screenshots of successful pipeline runs in the orchestration tool





☐ **Logs or monitoring dashboard screenshots**