Lead Conversion Classification

Complete MLOPs implementation in AWS Cloud

Subramanya Rithwik Jakka

Table of Contents

Part 1: Business & System Overview

---

1. Executive Summary & Vision

1.1. Problem Statement

In the highly competitive B2B sales environment, marketing and sales teams invest significant resources in acquiring leads. However, a large proportion of these leads fail to convert into paying customers, resulting in wasted efforts and suboptimal ROI.

Currently, there is a lack of intelligent mechanisms to prioritize high-potential leads based on historical data and behavioral attributes.

1.2. Business Objectives

The core business goal is to enable **data-driven sales optimization** by accurately identifying leads with the highest likelihood of conversion. By leveraging machine learning and modern MLOps practices, this system empowers sales representatives to focus their efforts on the most promising prospects.

This initiative directly supports the following strategic objectives:

- **Maximize Conversion Rate**: By predicting the probability of conversion for each lead, the system allows teams to allocate time and energy toward leads with higher success potential.
- **Maximize Lead Prioritization**: The model ranks leads based on predicted conversion likelihood, enabling targeted and prioritized outreach strategies.
- **Minimize Wasted Outreach**: Reduces time spent on low-quality or cold leads by eliminating guesswork through predictive analytics.
- **Maximize ROI**: Focused efforts on high-potential leads translate into improved sales efficiency, higher revenue, and better return on marketing and operational investments.

1.3. Solution Overview

To address the lead conversion prediction challenge, we have architected and implemented a **scalable, cloud-native machine learning pipeline** on **Amazon Web Services (AWS)**. This end-to-end system automates the entire ML lifecycle—from raw data ingestion to real-time inference—ensuring efficiency, reproducibility, and scalability.
The solution leverages a **modern AWS-native data and ML stack**:

- **AWS S3** for secure and scalable data storage
- **AWS Glue and Redshift** for data transformation and querying
- **Amazon SageMaker** for model development and training
- **MLflow** for experiment tracking and model lifecycle management

The resulting pipeline produces a **high-performing predictive model** that scores incoming leads and integrates seamlessly into downstream applications via a deployed **prediction API**. This empowers sales teams with **actionable intelligence** to prioritize high-value leads, optimize outreach, and ultimately increase conversion rates.

---

2. Success Criteria

The success of this project is measured against clearly defined business and technical metrics.

2.1. Business Success Criteria

1. Increase Conversion Rate by Prioritizing High-Potential Leads by at least 20%:

- Focus sales efforts on leads most likely to convert, as identified by predictive modeling.
- Elevate the overall conversion rate by at least 20% by systematically targeting high-value opportunities.

2. Improve Sales Team Efficiency by Reducing Time on Low-Value Leads

- Decrease the time and effort spent pursuing low-potential leads.
- Optimize resource allocation, enabling the sales team to concentrate on leads with greater conversion probability.

2.2. Machine Learning Success Criteria

1. Prediction Accuracy ≥ 90%

- The model must correctly classify at least 90% of all leads, indicating strong overall predictive performance.
- High accuracy ensures reliable identification and separation of high- and low-potential leads.

2. Precision for "Converted" Class ≥ 85%

- For leads predicted as "converted," at least 85% must actually convert.
- This minimizes false positives, ensuring that sales efforts are focused on genuinely high-potential leads.

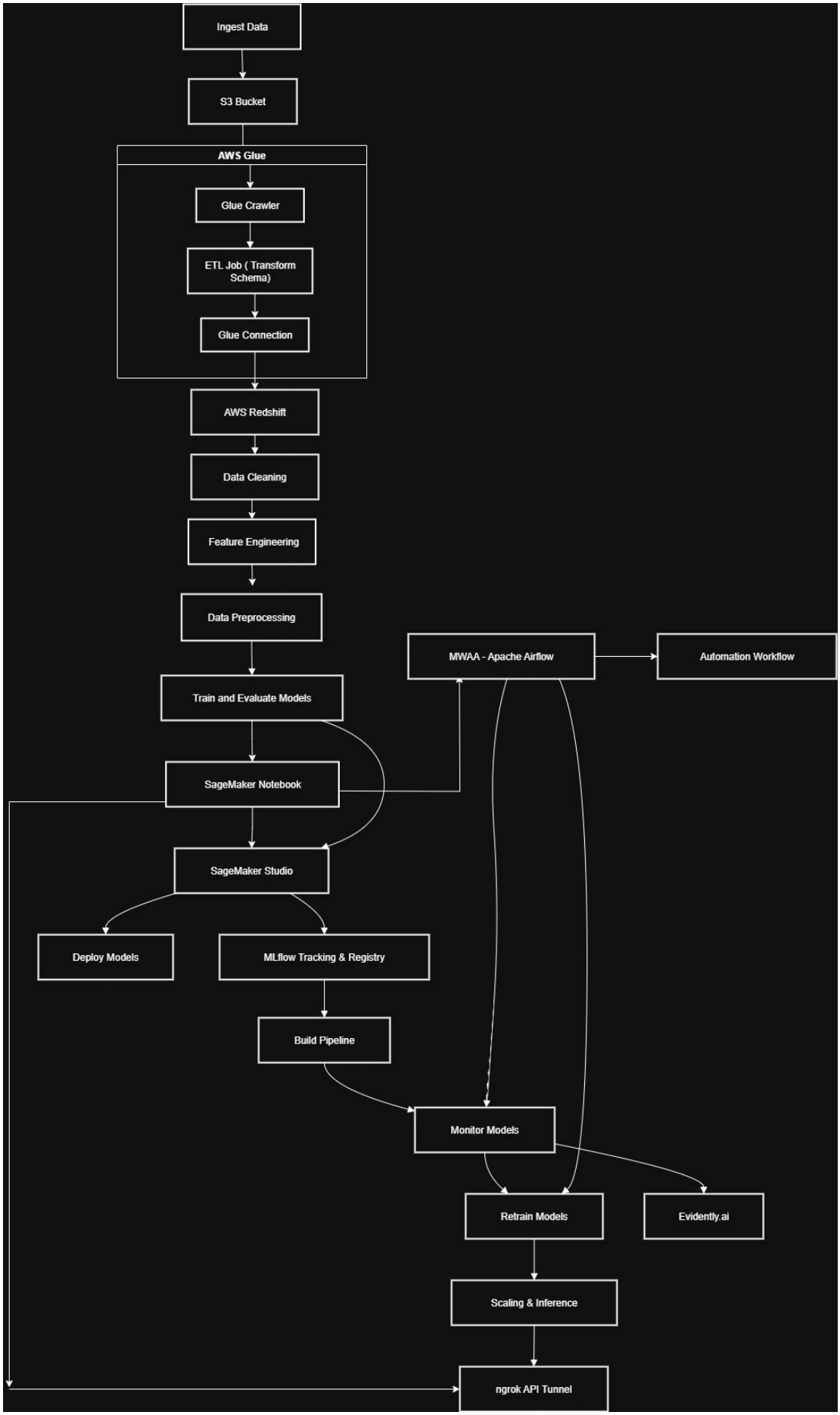
3. ROC AUC ≥ 0.90

- The model’s ROC AUC (Receiver Operating Characteristic Area Under Curve) should be 0.90 or higher.
- A high ROC AUC score demonstrates strong ability to distinguish between converted and non-converted leads, supporting confident prioritization and decision-making.

---

3. System Architecture: A Deep Dive

3.1. Architectural Blueprint Diagram : The system is designed as a modular, event-driven architecture that ensures scalability, security, and maintainability.


Ingest Data
S3 Bucket
AWS Glue
Glue Crawler
ETL Job (Transform Schema)
Glue Connection
AWS Redshift
Data Cleaning
Feature Engineering
Data Preprocessing
MWAA - Apache Airflow
Automation Workflow
Train and Evaluate Models
SageMaker Notebook
SageMaker Studio
Deploy Models
MLflow Tracking & Registry
Build Pipeline
Monitor Models
Retrain Models
Evidently.ai
Scaling & Inference
ngrok API Tunnel

This diagram illustrates a cloud-based, automated end-to-end machine learning (ML) pipeline utilizing AWS services, workflow orchestration, and tools for monitoring, retraining, and deployment.

1. Data Ingestion and Storage

- **Ingest Data:** Data is collected from one or more raw sources.
- **S3 Bucket:** The ingested data is stored securely in AWS S3, a scalable object storage service.

2. Data Cataloging and Transformation (AWS Glue)

- **AWS Glue:** A serverless data integration service.
    - **Glue Crawler:** Scans the data in S3, infers schema, and creates metadata tables.
    - **ETL Job (Transform Schema):** Performs Extract, Transform, Load (ETL) operations to clean and reformat the data.
    - **Glue Connection:** Facilitates integration and migration of transformed data.

3. Data Warehousing

- **AWS Redshift:** The transformed data is loaded into AWS Redshift, a scalable cloud data warehouse, for large-scale analytics and further processing.

4. Data Preparation

- **Data Cleaning:** Removal of noise, inconsistencies, or errors in the dataset.
- **Feature Engineering:** Creating new features or transforming variables to improve model effectiveness.
- **Data Preprocessing:** Further manipulation to ready data for modeling (e.g., normalization, encoding).

5. Model Development

- **Train and Evaluate Models:** ML models are built, trained, and evaluated.
- **SageMaker Notebook / Studio:** Amazon SageMaker is used for interactive development and model experimentation.

6. Model Deployment and Management

- **Deploy Models:** Trained models are deployed for inference (predictions).
- **MLflow Tracking & Registry:** Manages run metadata, model versions, and provenance.
- **Build Pipeline:** Automates deployment, testing, and integration steps.

7. Orchestration and Automation

- **MWAA - Apache Airflow:** Manages, schedules, and automates the pipeline's workflows.

- **Automation Workflow:** Enables repeatable, automated execution of steps from ingestion to deployment.

8. Monitoring and Feedback

- **Monitor Models:** Ongoing tracking of model performance and system health.
- **Evidently.ai:** Supports advanced model monitoring and drift detection.
- **Retrain Models:** Triggers retraining cycles if models deteriorate or new data arrives.
- **Scaling & Inference:** Supports model scaling and deploying to production for live inference.

9. API & External Integration

- **ngrok API Tunnel:** Creates secure tunnels to expose local web servers or APIs to the internet, enabling integrations, and remote access.

Key Features

- **Automated:** End-to-end orchestration using Apache Airflow.
- **Modular:** Clear separation of ingestion, transformation, modeling, deployment, and monitoring.
- **Scalable:** Built atop AWS's scalable infrastructure for storage, compute, and analytics.
- **Trackable:** MLflow and Evidently.ai ensure model versioning, traceability, and performance monitoring.
- **Continuous Feedback:** Automated retraining and monitoring loops enable adaptive and robust ML solutions.

3.2. Key Component Breakdown

- **Data Storage (Amazon S3):** Serves as the foundational storage layer. Different buckets/prefixes are used for `raw-data`, `processed-data`, and `model-artifacts`, ensuring a clean separation of concerns.
- **IAM (Identity and Access Management):** Provides the security backbone.
    - **AWS Glue Role:** Grants AWS Glue permissions to access S3 data and connect to the Redshift cluster.
    - **AWS SageMaker Role:** Allows SageMaker to read data from Redshift/S3 and write model artifacts back to S3.
- **AWS Glue (ETL Platform):** The serverless data integration engine.
    - **Connection to Redshift:** A pre-configured connector allowing Glue jobs to read from and write to Redshift.
    - **Crawler & Data Catalog:** Automatically scans and catalogs data schemas, making data discoverable and queryable.
    - **ETL Job (`s3ToRedshift`):** The core job that extracts raw S3 data, cleans it, and loads it into Redshift.
- **Networking (VPC):** Creates a private, isolated network for our resources.
    - **Security Group:** A virtual firewall controlling traffic to VPC endpoints.
    - **Endpoints:** Provide secure, private connections for S3, Redshift, Secrets Manager, KMS, and STS, preventing data exposure to the public internet.

- **Data Warehouse (Amazon Redshift):** The central repository for structured, analytics-ready data. It serves as the single source of truth for the machine learning model.
- **SageMaker (ML Platform):** The integrated development environment for all ML tasks.
    - **SageMaker Studio Lab:** Used for data preparation, interactive development, and model building.
    - **Artifact Management:** All trained models are saved to S3 for persistence and versioning.
- **Serving and Monitoring Stack:**
    - **MLflow:** Integrated for comprehensive experiment tracking, model logging, and lifecycle management via the Model Registry.
    - **Flask:** A lightweight web framework used to create a simple API to serve the model's prediction endpoint.
    - **Ngrok:** A utility to create a secure tunnel, exposing the local Flask API for testing and integration.
    - **MLflowUI:** The web dashboard for tracking, comparing, and visualizing ML experiments.

3.3. End-to-End Data and Model Flow

1. **Raw Data Ingestion:** Raw lead data (`.csv`) is uploaded to the designated Amazon S3 bucket.
2. **ETL with AWS Glue:** The Glue Crawler scans the S3 bucket and updates the Data Catalog. A scheduled Glue ETL job then extracts the data, transforms it, and loads it into Amazon Redshift.
3. **Redshift Data Access:** Redshift now holds clean, structured tables, ready for machine learning.
4. **Data Access for ML:** Amazon SageMaker securely accesses the Redshift data via VPC endpoints and its IAM role. Training, validation, and test sets are prepared.
5. **ML Development in SageMaker:** Models are built, trained, and evaluated in SageMaker Studio Lab. MLflow tracks all experiments and metrics. The final model artifacts are saved to S3.
6. **Model Logging & Serving:** The best model is registered in the MLflow Model Registry and promoted to "Production." It is then served as an API via a Flask application, exposed for testing with Ngrok.
7. **Feedback Loop & Automation:** Glue schedules ensure continuous data ingestion. The MLOps framework supports systematic retraining and versioning.

---

4. Data Dictionary

4.1. Feature Descriptions

The following table describes the features available in the dataset.

Feature Name	Explanation
id	Unique identifier for each lead.

Feature Name	Explanation
Lead Origin	How the lead originally entered the system.
Lead Source	The specific source or channel through which the lead was acquired.
Do Not Email	Indicates if the lead opted out of email communications (Yes/No).
Do Not Call	Indicates the lead's preference for receiving calls (Yes/No).
Converted	**Target Variable:** Whether the lead converted (1 = Yes, 0 = No).
TotalVisits	The total number of times the lead visited the website.
Total Time Spent on Website	Cumulative time in minutes the lead spent Browse the website.
Page Views Per Visit	The average number of pages viewed by the lead per session.
Last Activity	The last recorded interaction initiated by the lead.
Country	The country of the lead.
Specialization	The area of interest or specialty indicated by the lead.
How did you hear about X	How the lead heard about the organization.
What is your current occupation	The current profession of the lead.
What matters most...	The lead's primary motivation for choosing a course.
Search	Indicates if the lead used the website search feature (Yes/No).
Magazine	Indicates if the lead found out through a magazine (Yes/No).
Newspaper Article	Whether the lead read about the organization in a newspaper (Yes/No).
X Education Forums	Indicates interaction through an education forum (Yes/No).
Newspaper	Indicates engagement through other newspaper sources (Yes/No).
Digital Advertisement	Clicks or engagement via digital ads (Yes/No).
Through Recommendations	Indicates if the lead was referred by someone else (Yes/No).
Receive More Updates...	Whether the lead wants to receive course updates (Yes/No).
Tags	A categorized tag based on the lead's current status or behavior.
Lead Quality	A categorical assessment of the lead's quality by a human agent.
Update me on Supply...	Subscription to supply chain content updates (Yes/No).
Get updates on DM...	Subscription to digital marketing content updates (Yes/No).
City	The city of the lead.
Asymmetrique Activity Index	A behavioral activity score calculated by a third-party vendor.
Asymmetrique Profile Index	A profile matching score indicating alignment with the target persona.

Feature Name	Explanation
Asymmetrique Activity Score	A detailed activity engagement score from the vendor.
Asymmetrique Profile Score	A detailed profile fit score from the vendor.
I agree to pay... by cheque	Consent to pay by cheque (Yes/No).
A free copy of...	Whether the lead requested a free resource (e.g., eBook) (Yes/No).
Last Notable Activity	The most significant recent action taken by the lead.

Export to Sheets

Part 2: Foundational AWS Infrastructure Setup

---

5. Identity & Access Management (IAM)

5.1. The Principle of Least Privilege

Security is paramount. Our architecture adheres to the principle of least privilege, meaning every component is granted only the permissions essential for its designated function. This is achieved by creating highly specific IAM roles that services like Glue and SageMaker assume to perform their tasks, minimizing potential security risks.

5.2. Creating the Core Service Roles

We create two primary roles: one for Glue and one for SageMaker. The process below details the creation of the Glue role, which is the more complex of the two.

Console Walkthrough: Creating `glueaccessrole`

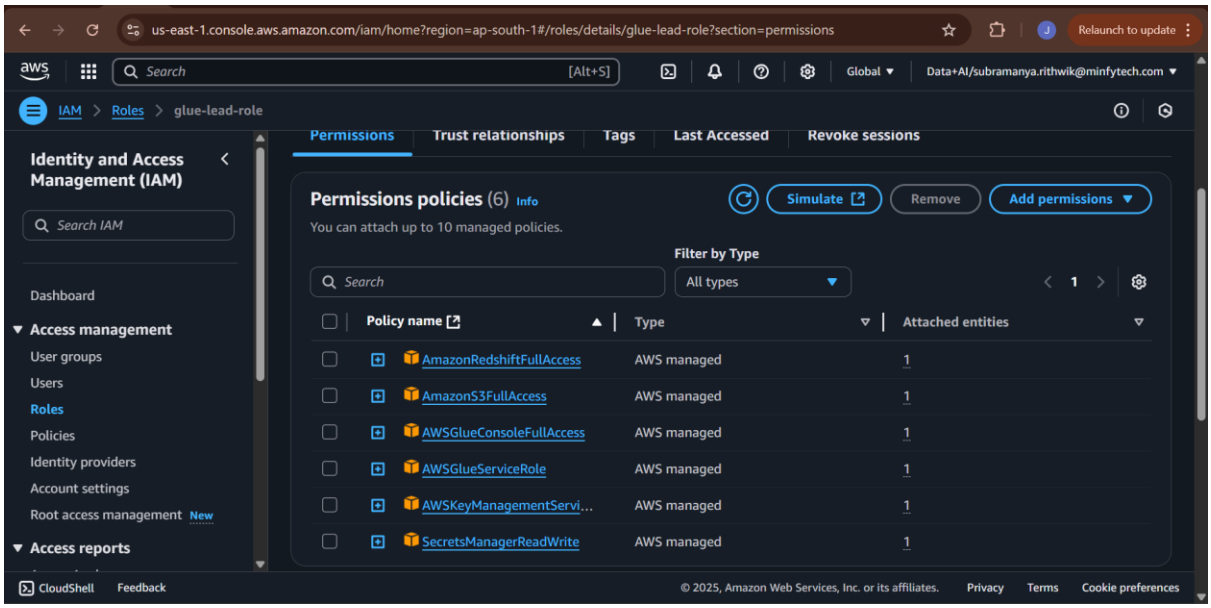
1. **Navigate to IAM:** In the AWS Management Console, go to the IAM service.
2. **Create Role:** Click on **Roles** in the left navigation pane, then click **Create role**.
3. **Select Trusted Entity:**
    - **Trusted entity type:** Select **AWS service**.
    - **Use case:** Choose **Glue** from the dropdown menu. This automatically creates a trust policy allowing the AWS Glue service (`glue.amazonaws.com`) to assume this role.
    - Click **Next**.
4. **Add Permissions:** On the "Add permissions" page, search for and attach the following AWS managed policies. For a production environment, these should be replaced with more fine-grained, custom policies.
    - `AmazonS3FullAccess`
    - `AmazonRedshiftFullAccess`
    - `AWSGlueConsoleFullAccess`
    - `AWSGlueServiceRole`
    - `SecretsManagerReadWrite`
    - `AWSKeyManagementServicePowerUser`
    - Click **Next**.
5. **Name and Review:**

- **Role name:** `glue-lead-role`
- Review the attached policies and the trust policy. The trust policy JSON should look like this:

  JSON

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Principal": {
                  "Service": "glue.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
          }
      ]
  }
  ```

- Click **Create role**.



A similar process is followed to create the SageMaker role, selecting **SageMaker** as the use case in Step 3.

---

6. Networking & Security (VPC)

6.1. VPC and Subnet Strategy

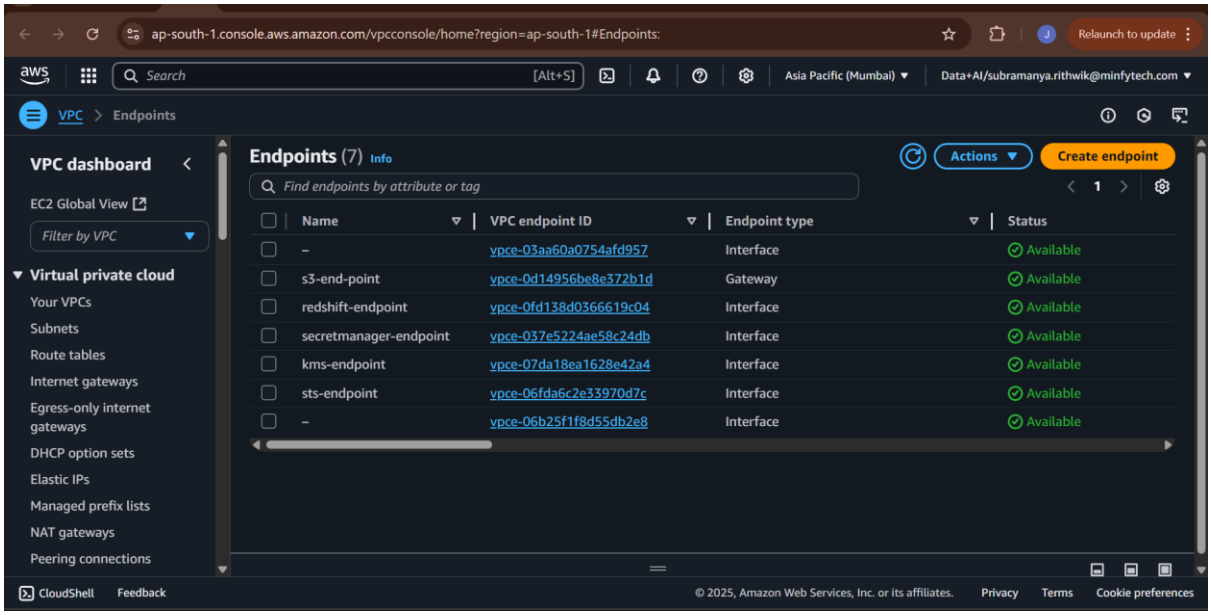
To ensure all inter-service communication is secure and isolated from the public internet, all resources are deployed within a Virtual Private Cloud (VPC). We utilize the default VPC for this guide, but a custom VPC is recommended for production. The VPC contains multiple subnets across different Availability Zones for high availability.

6.2. Creating VPC Endpoints for Secure Communication

VPC Endpoints allow services within the VPC to communicate with other AWS services as if they were on the same network, without traffic ever leaving the AWS backbone.

Console Walkthrough: Creating Endpoints

1. Navigate to the **VPC** service in the AWS Console.
2. Select **Endpoints** and click **Create endpoint**.
3. **Endpoint 1: S3 (Gateway)**
    - **Service category:** AWS services
    - **Service name:** Search for `s3` and select the service with `Type: Gateway`.
    - **VPC:** Select your target VPC.
    - **Route tables:** Select the route tables for the subnets where your resources reside. This adds a route to the table, directing S3 traffic through the gateway.
    - Click **Create endpoint**.
4. **Endpoint 2: Redshift (Interface)**
    - Click **Create endpoint** again.
    - **Service category:** AWS services
    - **Service name:** Search for `redshift` and select the service `com.amazonaws.<region>.redshift`.
    - **VPC:** Select your target VPC.
    - **Subnets:** Select at least two subnets in different Availability Zones.
    - **Security groups:** Select the security group created in the next section.
    - Click **Create endpoint**.
5. **Endpoints 3, 4, 5: Secrets Manager, KMS, and STS (Interface)**
    - Repeat the process for the following services, selecting the same VPC, subnets, and security group for each:
        - `com.amazonaws.<region>.secretsmanager`
        - `com.amazonaws.<region>.kms`
        - `com.amazonaws.<region>.sts`

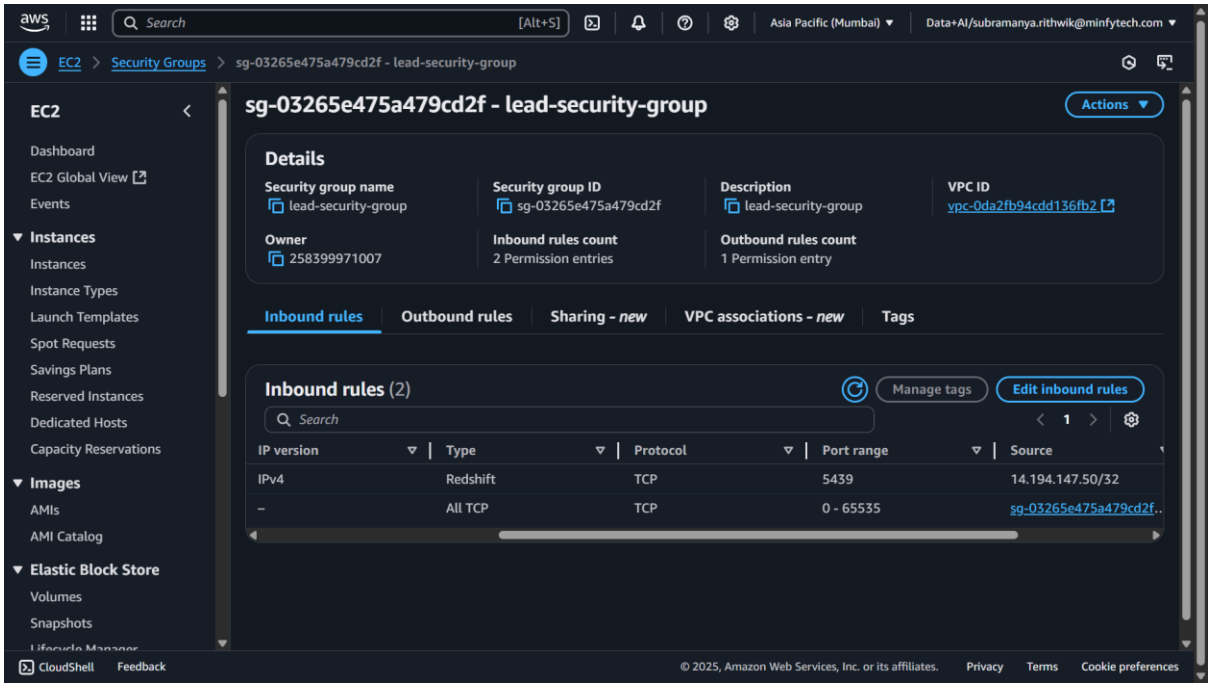


6.3. Configuring the Security Group Firewall

A security group acts as a stateful firewall for resources, controlling inbound and outbound traffic.

Console Walkthrough: Configuring the Security Group

1. Navigate to **VPC > Security Groups** and click **Create security group**.
2. **Name and VPC:** Name it `lead-security-group` and associate it with your VPC.
3. **Configure Inbound Rules:** Select the new security group and click the **Inbound rules** tab, then **Edit inbound rules**.
    - **Rule 1 (For Redshift Client Access):**
        - **Type:** `Redshift`
        - **Protocol:** `TCP`
        - **Port range:** `5439`
        - **Source:** `My IP` (to allow access from your local machine).
    - **Rule 2 (For Internal Communication):**
        - **Type:** `All TCP`
        - **Protocol:** `TCP`
        - **Port range:** `0 - 65535`
        - **Source:** `Custom`. Start typing the ID of the `lead-security-group` group itself (`sg-xxxxxxxx`). This self-referencing rule allows all resources within the security group to communicate with each other freely. This is critical for Glue, SageMaker, and Redshift to interact.
4. **Save Rules:** Click **Save rules**. The default outbound rule, which allows all traffic out, is sufficient for this project.



Part 3: Data Engineering & Warehousing

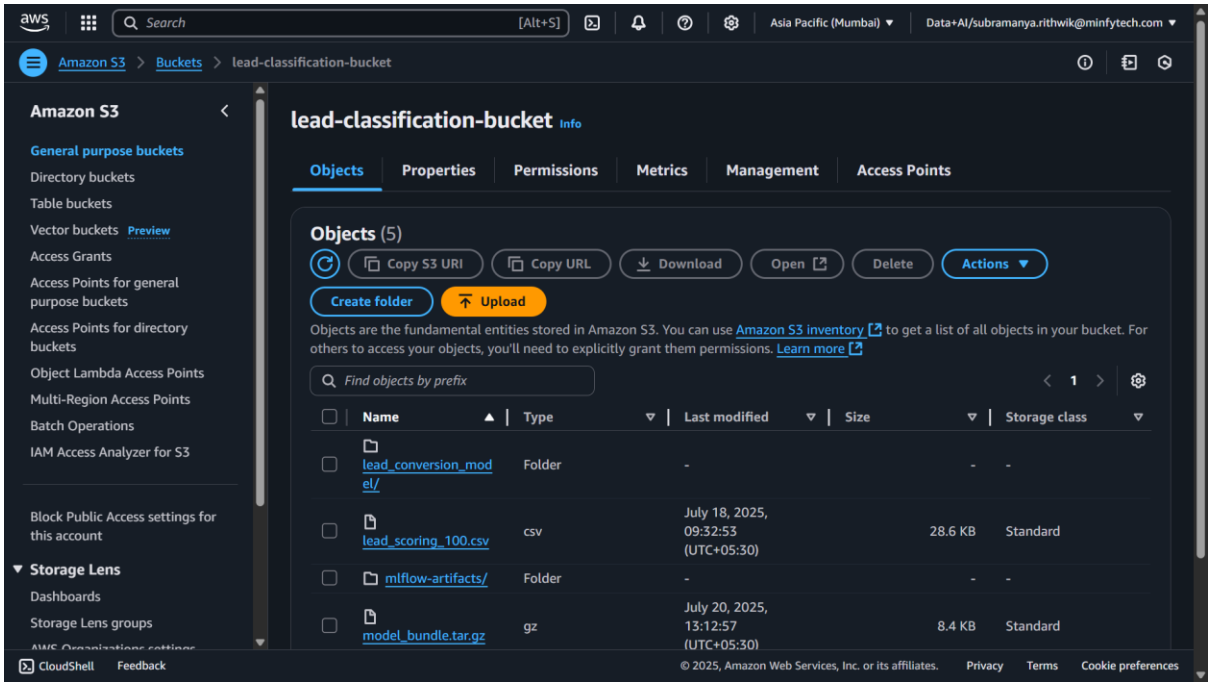
---

7. Data Lake & Warehouse Setup

7.1. S3 Bucket Configuration

Amazon S3 is the foundation of our data storage. We use a single bucket with a logical prefix structure to organize data by stage.

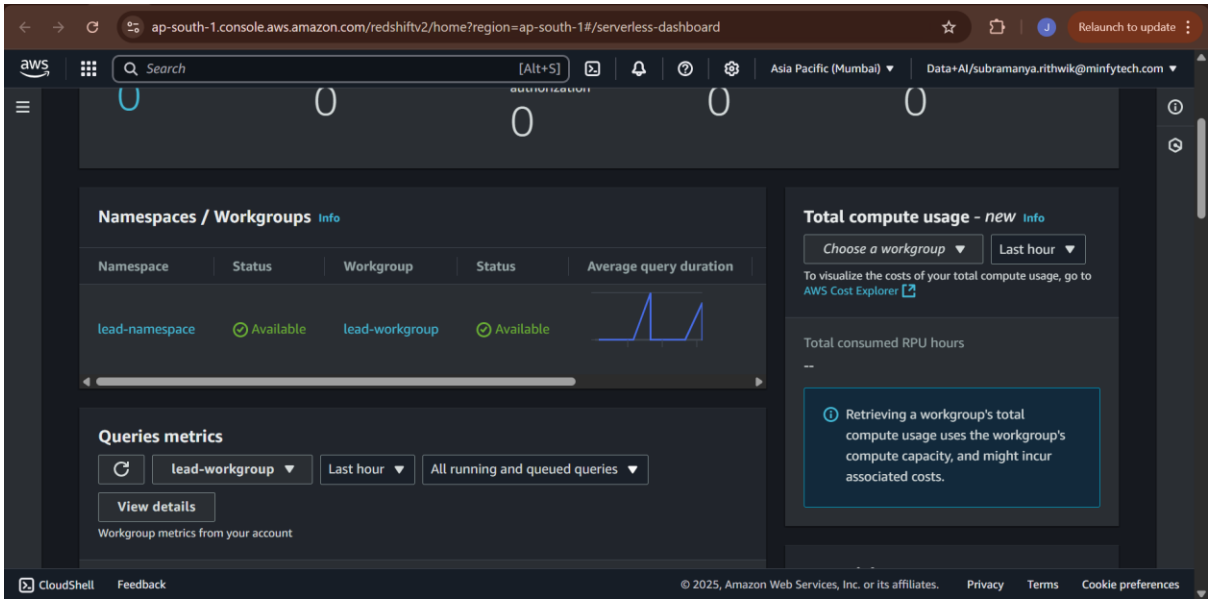
- **Bucket Name:**
- **Prefixes:**
  - `s3://leadconversiondata /raw-data/`: For incoming, unaltered lead data.
  - `s3://leadconversiondata/processed-data/`: For data cleaned and feature-engineered by Glue.
  - `s3://leadconversiondata/model-artifacts/`: For storing trained model binaries, SHAP plots, and other ML assets.



7.2. Amazon Redshift Serverless Provisioning

Redshift Serverless provides a simple, auto-scaling data warehouse.

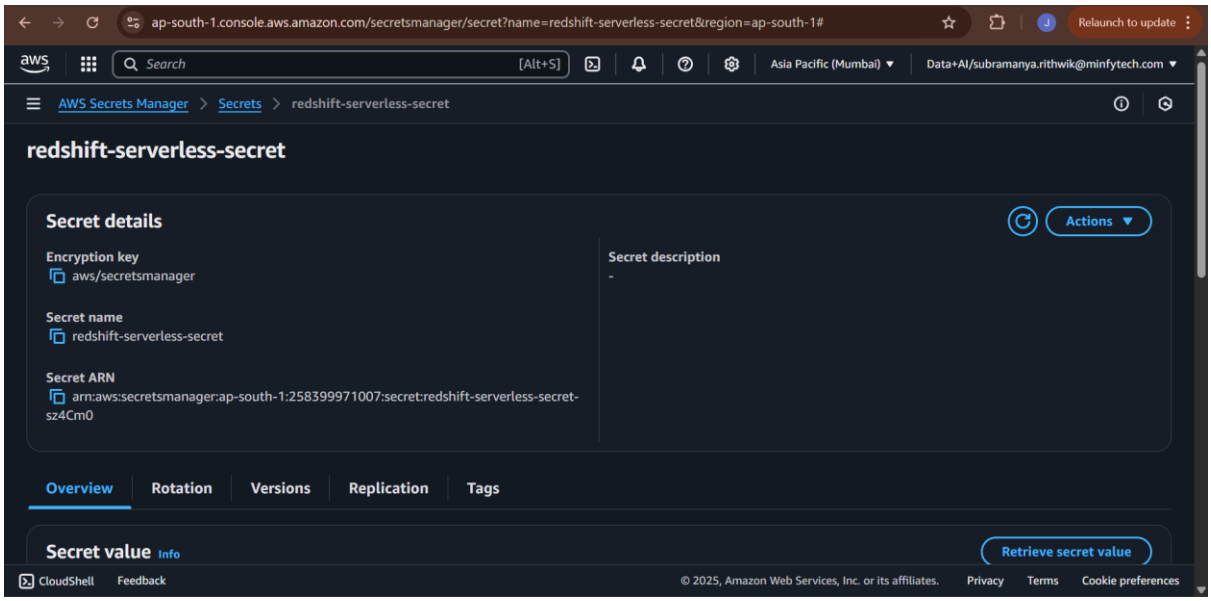
1. Navigate to the **Amazon Redshift** service and select **Try Redshift Serverless**.
2. Configure the workgroup and namespace, accepting the defaults.
3. In the **Network and security** section, associate the workgroup with your VPC and the `glue-lead-role` security group.
4. Review and create the serverless endpoint. This will take several minutes.



7.3. Securing Credentials with AWS Secrets Manager

We avoid hardcoding database credentials by storing them securely in Secrets Manager.

1. In the Redshift console, set a password for the default `awsuser`.
2. Navigate to **AWS Secrets Manager** and click **Store a new secret**.
3. **Secret type:** Select `Credentials for Redshift cluster`.
4. Enter the `awsuser` username and the password you set.
5. **Secret name:** Use a descriptive name like `prod/redshift/credentials`.
6. Complete the process, disabling automatic rotation for this guide. Note the ARN of the created secret.



---

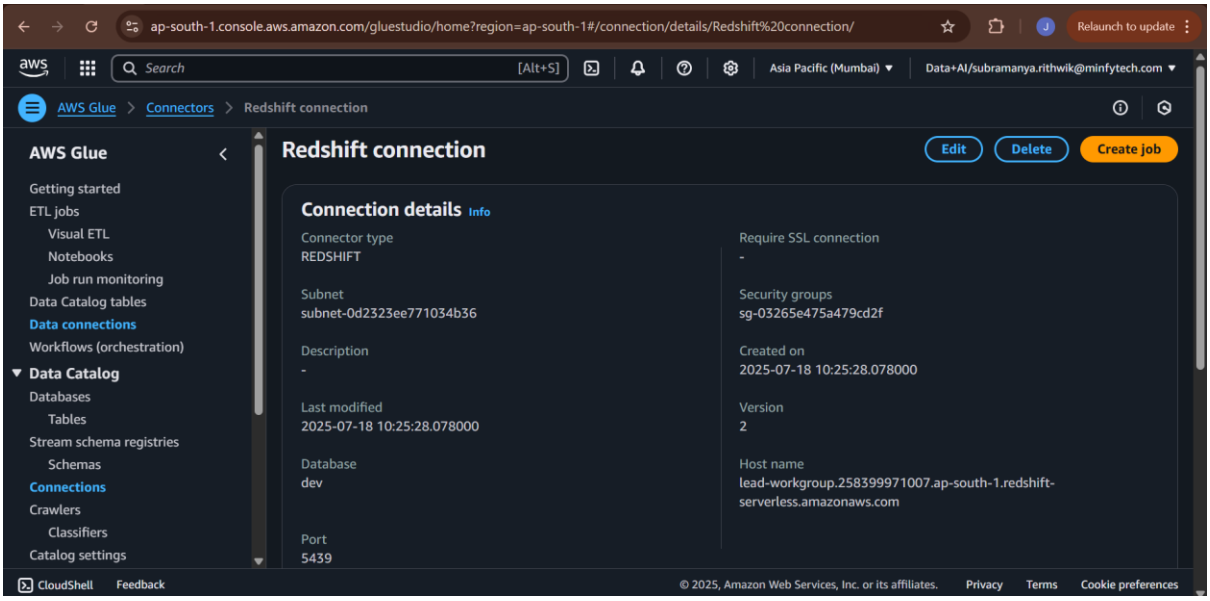
8. The Ingestion Pipeline: S3 to Redshift

8.1. ETL with AWS Glue: Overview

AWS Glue provides a serverless framework to extract data from S3, transform it, and load it into Redshift. This process relies on a Glue Connection to securely access Redshift and a Glue Crawler to understand the data's structure.

8.2. Creating the Glue Connection to Redshift

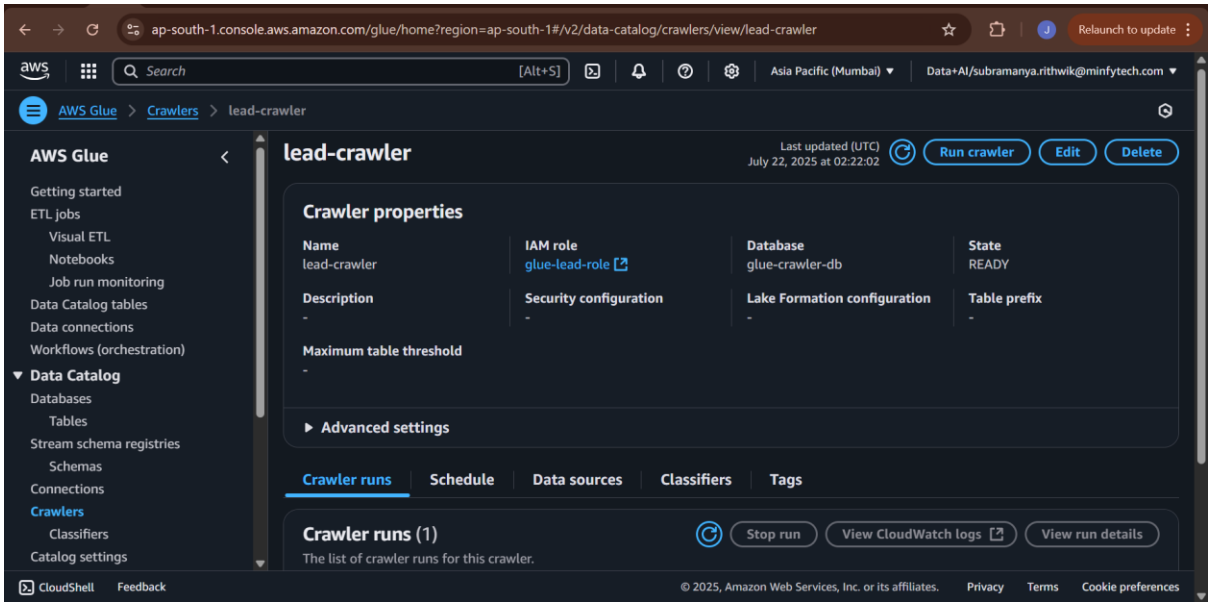
1. Navigate to **AWS Glue** and select **Connections**.
2. **Create connection:**
    - **Name:** `Redshift Connection`
    - **Type:** `Amazon Redshift`
    - Select `Connect to a Redshift Serverless workgroup` and choose your workgroup.
    - For **Authentication**, select the AWS Secret (`prod/redshift/credentials`) you created.
3. **Test Connection:** After creating, select the connection and click **Actions > Test connection**. Choose the `glue-lead-role`. A "Success" message confirms that your networking and permissions are correct.



8.3. The Glue Crawler: Automatic Schema Discovery

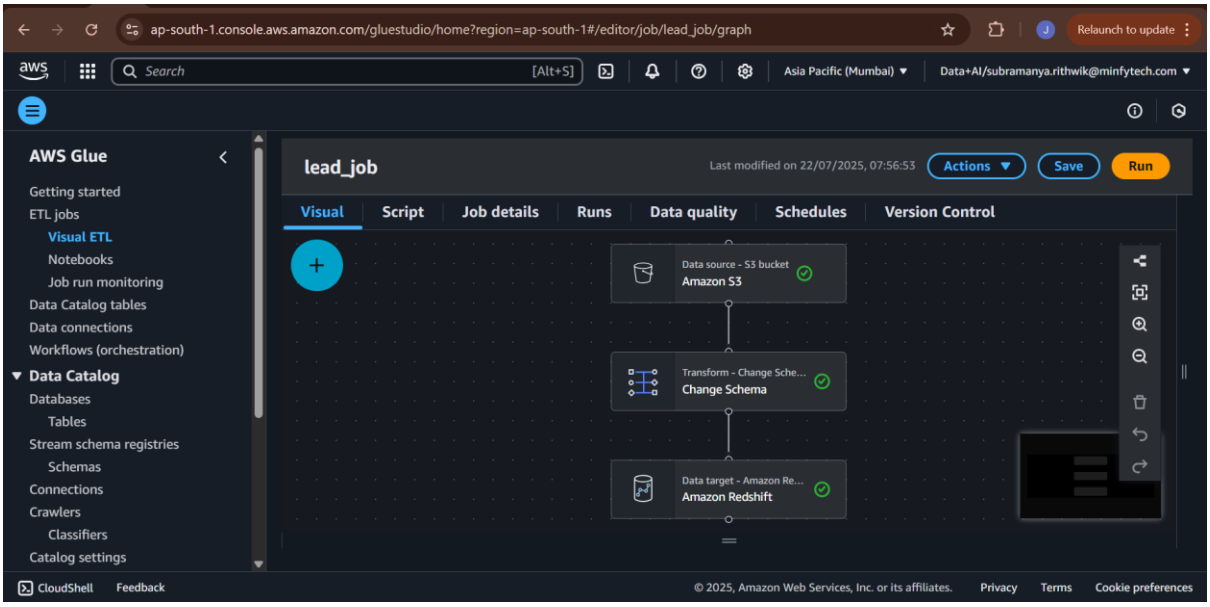
The crawler scans our raw data and creates a metadata table in the Glue Data Catalog.

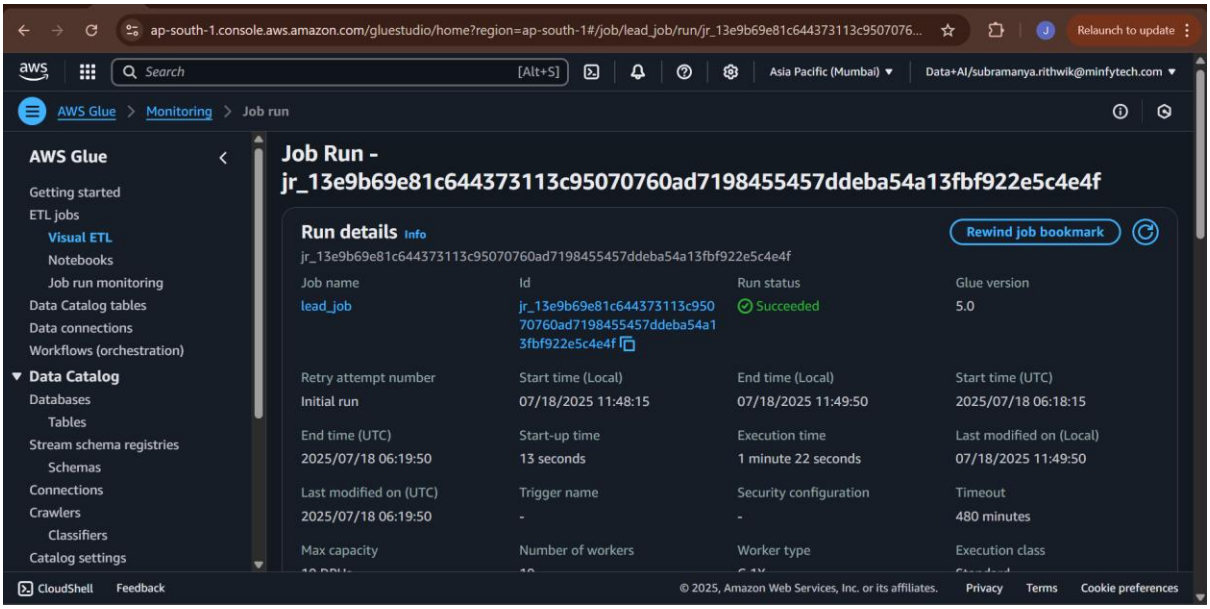
1. In Glue, select **Crawlers** and **Create crawler**.
2. **Name:** `lead-crawler`.
3. **Data source:** Point it to the S3 path `s3://leadconversiondata /raw-data/`.
4. **IAM Role:** Select `glue-lead-role`.
5. **Output:** Create a new database in the Glue Data Catalog named `glue-crawler-db`.
6. Run the crawler. Upon completion, a new table will be visible in the `glue-crawler-db` database.



8.4. Building the Visual ETL Job (`s3ToRedshift`)

1. In Glue, navigate to **Glue Studio** and create a new **Visual ETL** job.
2. **Source Node:** Select **AWS Glue Data Catalog** and choose the table created by the crawler.
3. **Transform Node:** Add any desired transformations. For a simple ingestion, you might use the **Change Schema** transform to ensure data types are correct.
4. **Target Node:** Select **Amazon Redshift**.
    - **Connection:** Choose `Redshift Connection`.
    - **Target options:** Specify the target database (`dev`) and table name (`public.lead_scoring`).
5. **Job Details:** Name the job `lead_job` and assign the `glue-lead-role`.
6. Save and **Run** the job. After it succeeds, you can query the `public. lead_scoring` table in Redshift.





---

9. Data Access from SageMaker

9.1. Using the Redshift Data API with `boto3`

To load data into our SageMaker notebook for ML development, we use the Redshift Data API. This avoids managing traditional JDBC/ODBC drivers and is ideal for serverless interaction. The following Python script automates this process.

9.2. Step-by-Step Code Walkthrough

Step 1: Configuration This block sets up all the necessary parameters to connect to Redshift.

Python

```
import boto3
import pandas as pd
import time

# Sets AWS region, Redshift workgroup name, database name, secret ARN, and
SQL query.
# The secret_arn is created in AWS Secrets Manager to store Redshift
credentials securely.
region = 'ap-south-1'
workgroup_name = 'lead-workgroup'
database_name = 'dev'
secret_arn = 'arn:aws:secretsmanager:ap-south-
1:123456789012:secret:prod/redshift/credentials-xxxxxx' # Replace with your
ARN
sql = 'SELECT * FROM public.lead_scoring'
```

Step 2: Create Redshift Data API Client The `boto3` library is used to create a low-level client for the Redshift Data API service.

Python

```
# Uses boto3 to create a Redshift Data API client.
# This is how you interact with Redshift without needing a persistent JDBC
connection.
client = boto3.client('redshift-data', region_name=region)
```

Step 3: Run the SQL Query The `execute_statement` function sends the SQL query to Redshift asynchronously.

Python
```
# Executes the given SQL query on the Redshift Serverless workgroup.
# Returns a statement_id used to track the query's progress.
response = client.execute_statement(
    WorkgroupName=workgroup_name,
    Database=database_name,
    SecretArn=secret_arn,
    Sql=sql
)
statement_id = response['Id']
```

Step 4: Wait for Query Completion Because the execution is asynchronous, we must poll the API to check the query's status.

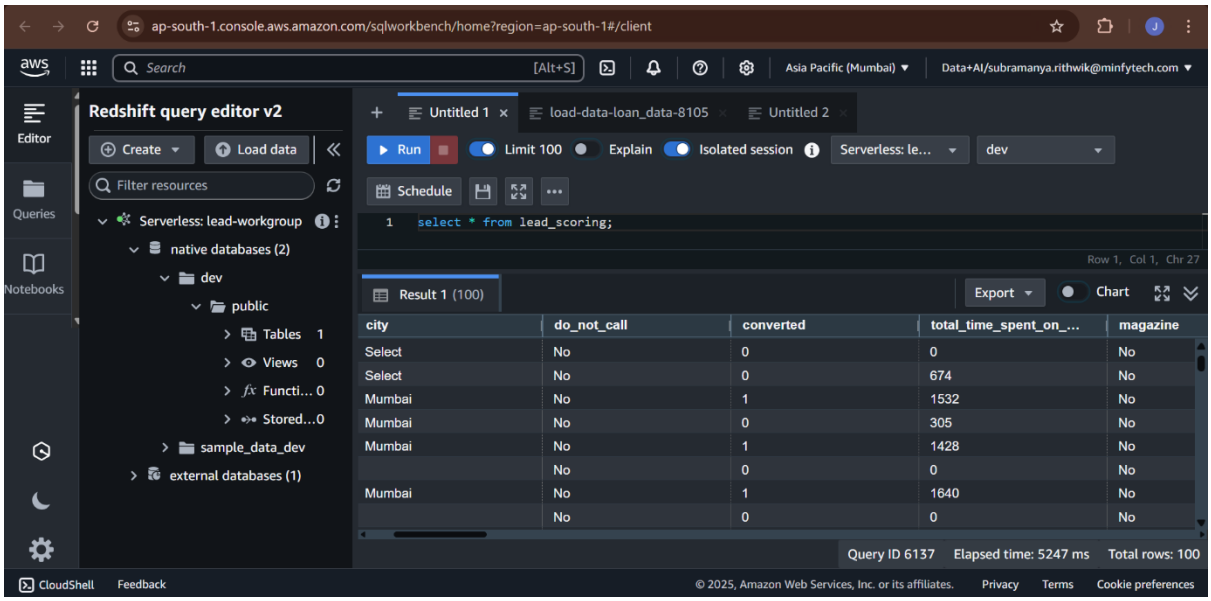
Python
```
# Repeatedly checks the query status using the statement_id.
# Waits in a loop (sleeping for 1 second) until the query either FINISHES
or FAILS.
desc = client.describe_statement(Id=statement_id)
while desc['Status'] not in ['FINISHED', 'FAILED', 'ABORTED']:
    time.sleep(1)
    desc = client.describe_statement(Id=statement_id)
```

Step 5 & 6: Retrieve and Parse Query Results Once finished, we fetch the results and parse them into a structure suitable for a DataFrame.

Python
```
# Retrieves the actual data after the query is finished.
result = client.get_statement_result(Id=statement_id)

# Extracts column names from metadata.
columns = [col['name'] for col in result['ColumnMetadata']]
rows = result['Records']

# Iterates over each record and extracts values.
data = []
for row in rows:
    data.append([list(col.values())[0] if col else None for col in row])
```

Step 7 & 8: Convert to Pandas DataFrame and Display The final step is to create a Pandas DataFrame, the standard tool for data manipulation in Python.

Python
```
# Converts the extracted data and columns into a standard Pandas DataFrame.
df = pd.DataFrame(data, columns=columns)

# Prints the first 5 rows to verify the data was loaded correctly.
print("Data successfully loaded from Redshift:")
print(df.head())
```



Part 4: Machine Learning Development

---

10. Amazon SageMaker: The ML Development Environment

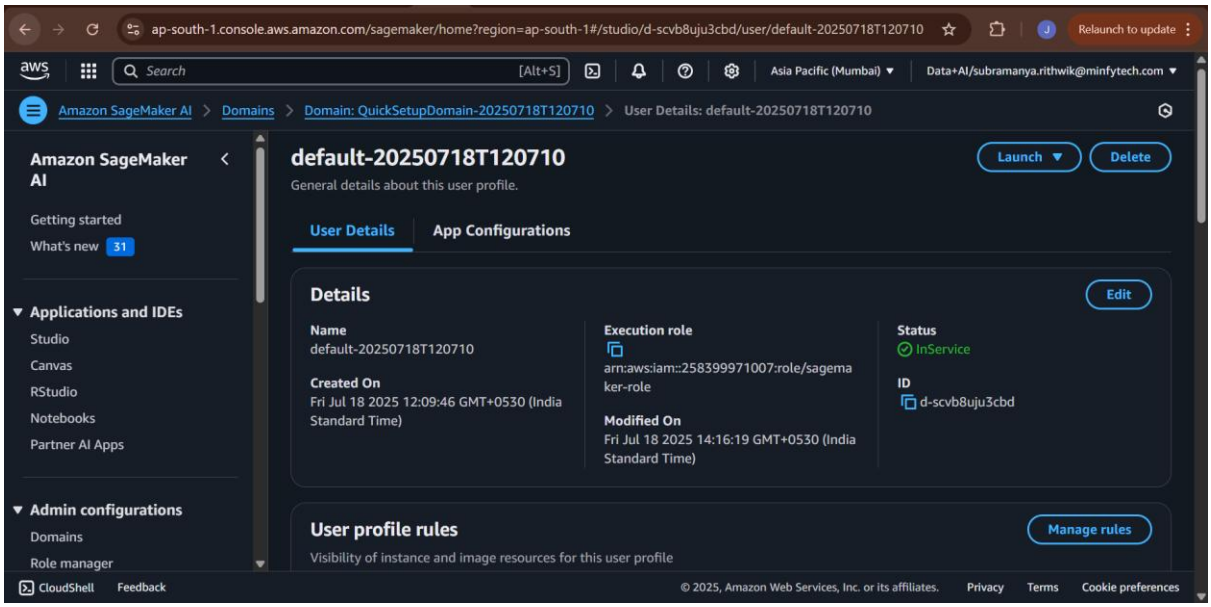
All machine learning development for this project is conducted within **Amazon SageMaker**, a fully managed service that provides every developer and data scientist with the ability to prepare, build, train, and deploy high-quality ML models quickly. We specifically use **SageMaker Studio**, an integrated development environment (IDE) for machine learning.

10.1. Creating a SageMaker Domain

A SageMaker Domain is the primary entry point for using SageMaker Studio. It consists of a list of authorized users, a variety of security controls, and an Amazon Elastic File System (EFS) volume that contains the data, notebooks, and other artifacts for the users in the Domain.

Console Walkthrough: Creating the Domain

1. **Navigate to SageMaker:** In the AWS Management Console, go to the **Amazon SageMaker** service.
2. **Create a Domain:** On the SageMaker dashboard, click on **Domain** in the left navigation pane, then click **Create Domain**.
3. **Setup:** Choose the **Standard setup** option.
4. **Authentication:** For this guide, select **AWS Identity and Access Management (IAM)** as the authentication method.
5. **Configure Network and Storage:**
    - **VPC:** Select the same Virtual Private Cloud (VPC) that your Redshift cluster and other resources are in. This is critical for secure communication.
    - **Subnet(s):** Choose the private subnets associated with your VPC. Selecting multiple subnets across different Availability Zones ensures high availability.
    - **Security Group:** Attach the `mlops-sg` security group that was configured in Part 2. This allows the SageMaker environment to communicate with other services like Redshift and the MLflow server through the private VPC endpoints.
6. **IAM Role:** Create a new IAM role or use an existing one that has the necessary permissions. This role should have, at a minimum:
    - `AmazonSageMakerFullAccess` policy.
    - Permissions to access the S3 bucket (`lead-classification-bucket`).
    - Permissions to interact with the Redshift Data API.
7. **Create User Profile:** Within the domain setup, add a default user profile. Give it a name like `ml-developer`.
8. **Review and Submit:** Review all configurations and click **Submit** to create the Domain. This process can take several minutes.

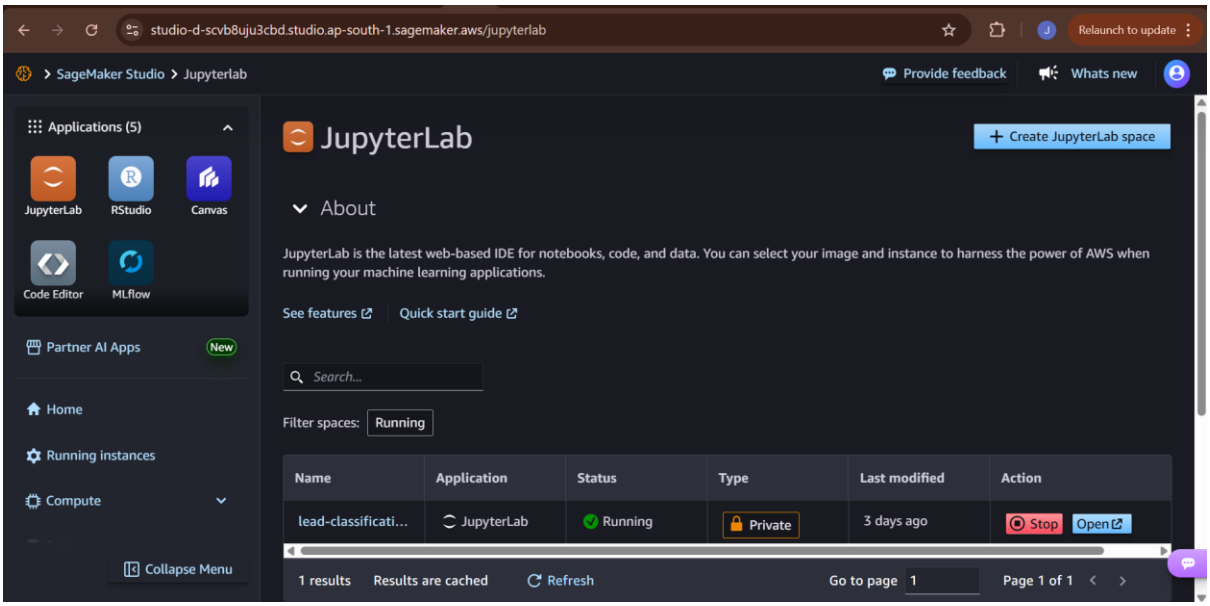


10.2. Launching SageMaker Studio

Once the Domain status is "Ready," you can launch the Studio IDE for your user.

1. From the SageMaker Domain page, find your user profile (e.g., `ml-developer`) in the user list.
2. On the right side of the user row, click **Launch** and select **Studio**.

3. A new browser tab will open, and the SageMaker Studio environment will begin to load. This can take a minute or two on the first launch as it provisions the necessary resources.

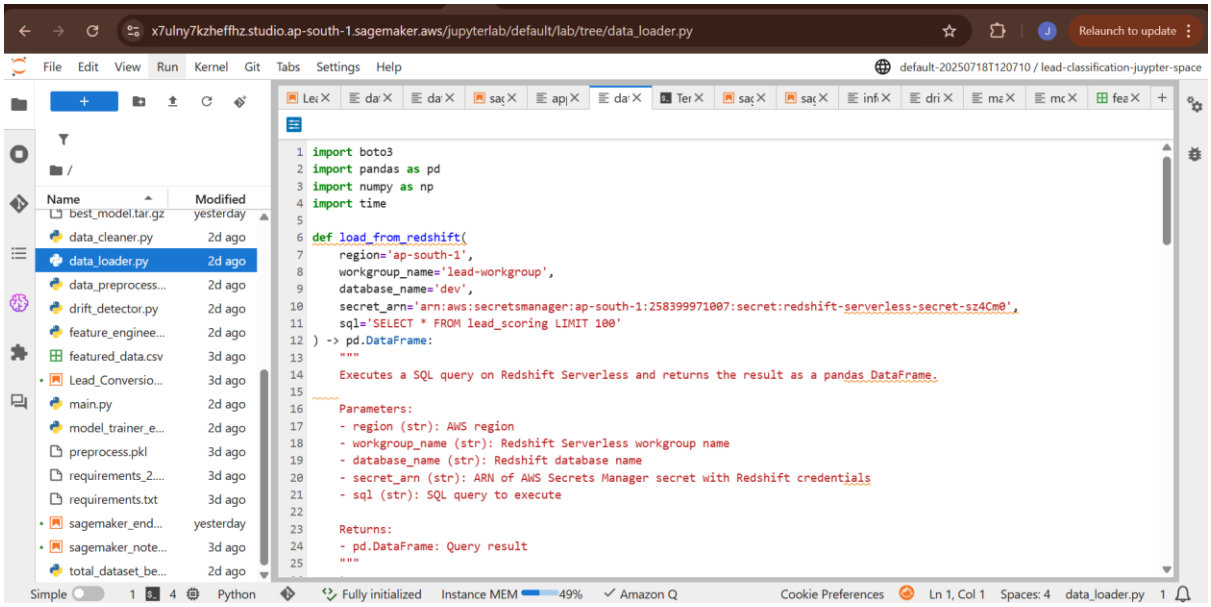


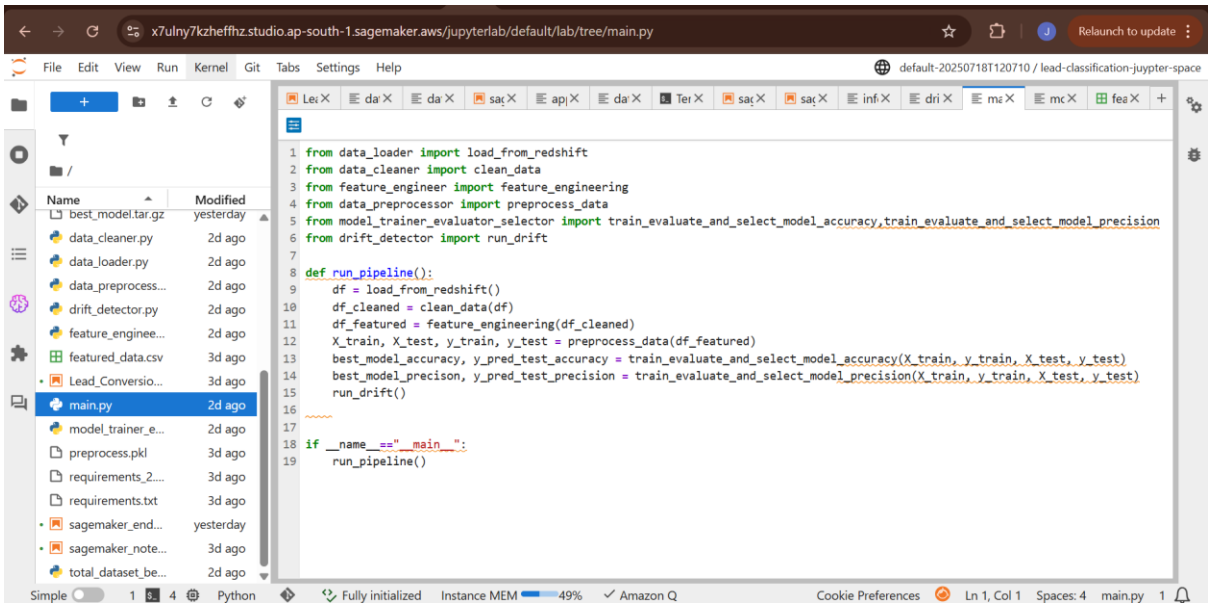
10.3. Opening the JupyterLab Notebook

SageMaker Studio provides a fully managed JupyterLab environment. This is where all coding, from data exploration to model training, will take place.

1. **Open Launcher:** Once Studio has loaded, you will be presented with a launcher screen. If not, you can open it from the **File > New Launcher** menu.
2. **Create Notebook:** Under the "Notebooks and compute resources" section, click on **Notebook**.
3. **Select Kernel:** A dialog will pop up asking you to select an image and kernel. Choose the **Data Science** image and the **Python 3** kernel. This provides a Python environment pre-loaded with common data science libraries like pandas, NumPy, and scikit-learn.
4. The new notebook file (e.g., `Untitled.ipynb`) will open, and you are now ready to begin the ML development process.
5. You can directly use your local .py files and made changes according to cloud.

With the SageMaker Studio environment running, the first step in our ML workflow is to load the data from Redshift and begin the exploratory data analysis.





---

11. Exploratory Data Analysis (EDA) and Cleaning

11.1 Initial Data Inspection

Before any modeling, the dataset loaded from Redshift is explored to understand its structure, completeness, and potential quality issues. The following steps are conducted:

- **Data Preview (df.head())**
  View the top few records to understand column structure and sample values.
- **Data Types & Nulls (df.info())**
  Check for missing values, data types (e.g., float64, object), and shape.

- **Descriptive Statistics (df.describe())**
  Analyze numeric columns for mean, standard deviation, min, max, and percentiles.
- **Missing Value Summary (df.isnull().sum())**
  Detect columns with missing entries that need imputation or cleaning.
- **Duplicate Records (df.duplicated().sum())**
  Identify and remove duplicate rows that can bias learning.
- **Column Name Review (df.columns)**
  Validate column naming conventions before standardization.

11.2 Data Cleaning and Standardization

Implemented in clean_data(df) function. This step ensures that the dataset is free of inconsistencies and missing values are handled systematically:

- **Dropped Columns:**
  - **Identifiers:** 'Prospect ID', 'Lead Number' – unique IDs with no predictive power.
  - **Constant Value Columns:** 'Magazine', 'Receive More Updates About Our Courses', etc.
- **Handled Nulls and Placeholder Values:**
  - Replaced "Select" and "" with "Missing" or appropriate defaults.
  - Filled 'Country' with 'India', 'What is your current occupation' with 'Unemployed', etc.
- **Consolidated and Mapped Labels:**
  - Applied custom mappings to group sparse or noisy categorical values:
    - Lead Source (e.g., 'bing' → 'Other')
    - Specialization (e.g., 'e-business' → 'Business')
    - Tags (e.g., 'ringing', 'busy' → 'Not Reachable')
    - Lead Quality (e.g., 'low in relevance' → 'Low')
    - Lead Profile, How did you hear... are similarly grouped.
- **Ordinal Encoding:**

    - Asymmetrique Profile Index, Asymmetrique Activity Index: '01.High' → 3, '02.Medium' → 2, '03.Low' → 1

- **Imputation:**
    - **For numerical fields like 'TotalVisits', 'Asymmetrique Profile Score':**
      Used median to fill missing values after coercion to numeric.
- **Text Normalization:**
    - Lowercased and stripped whitespace from all string cells using applymap().

---

12. Feature Engineering

Executed through the feature_engineering(df) function. This step creates derived features using domain insights:

- **Engagement Score**
  Weighted combination of:
    - Total Time Spent on Website (40%)
    - Page Views Per Visit (30%)
    - TotalVisits (30%)
- **Combined Asymmetrique Score**
    - Sum of Asymmetrique Activity Score and Asymmetrique Profile Score.
- **Is New Tag**
    - **Binary indicator:** 1 if 'Tags' contains 'student', else 0.
- **High Interaction**
    - Binary feature based on 'Last Activity' values such as 'SMS Sent', 'Email Opened'.
- **Potential Lead**
    - Derived from 'Lead Profile' — 1 if it contains 'potential'.

These features improve signal-to-noise ratio and help models capture behavioral patterns.

---

13. Preprocessing Pipeline

Implemented via preprocess_data(df). This stage ensures reproducibility and consistency in feature transformation.

13.1 Feature Grouping

- **Numerical: Website behavior & profile scores**
    - 'TotalVisits', 'Total Time Spent on Website', etc.
- **Ordinal:**
    - 'Asymmetrique Activity Index', 'Asymmetrique Profile Index', 'Lead Quality'
- **Categorical:** All remaining object-based columns

13.2 Data Splitting

- Performed an 80/20 stratified split into training and testing sets.
- Ensures balanced distribution of the target variable (Converted).

13.3 Outlier Handling

- Applied Winsorization (1% on both sides) on numerical features to cap extreme values and mitigate outliers' impact.

13.4 Pipeline Composition

- **Numerical Pipeline:**
    - SimpleImputer(strategy='mean')
    - PowerTransformer(method='yeo-johnson')
    - MinMaxScaler
- **Ordinal Pipeline:**
    - SimpleImputer(strategy='most_frequent')
    - OrdinalEncoder with predefined order for each feature
- **Categorical Pipeline:**
    - SimpleImputer(strategy='most_frequent')
    - OneHotEncoder(handle_unknown='ignore')

Combined via ColumnTransformer.

13.5 Saving Pipeline

- The final preprocessor is saved using joblib to preprocessor.pkl.

---

14. Model Training and Evaluation

Two separate pipelines target different business needs:

14.1 Training for Accuracy

Function: train_evaluate_and_select_model_accuracy()

- **Models Used:**
  - Logistic Regression, Random Forest, XGBoost, LightGBM
- **Cross-validation:**
  - 5-fold stratified CV using F1-score as the scoring metric
- **Evaluation Metrics:**
  - Accuracy, Precision, Recall, F1-score, ROC-AUC
- **Model Selection:**
  - Based on best test set accuracy
- **MLflow Integration:**
  - Logs hyperparameters, metrics, model artifact
  - Registers best model to SageMaker-hosted MLflow Registry
  - Promotes model to Staging and then Production

```
Best Model: xgboost
Test Accuracy: 0.9271
Test F1 Score: 0.9075

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.95      0.94       955
           1       0.92      0.89      0.91       637

    accuracy                           0.93      1592
   macro avg       0.93      0.92      0.92      1592
weighted avg       0.93      0.93      0.93      1592
```

14.2 Training for Precision (Class 1)

Function: train_evaluate_and_select_model_precision()

- Same models and structure as above.
- Optimizes using precision_score for class 1 to reduce false positives.
- Selected model is also logged and registered in MLflow.

```
Best Model: xgboost
Accuracy: 0.8763
Precision (class 1): 0.9564
F1 Score: 0.8239

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.98      0.90       955
           1       0.96      0.72      0.82       637

    accuracy                           0.88      1592
   macro avg       0.90      0.85      0.86      1592
ghted avg          0.89      0.88      0.87      1592
```

---

15. Model Explainability with SHAP

SHAP analysis is used to build trust and interpretability in the deployed models:

- **Explainer Type:**
  - **TreeExplainer:** For tree-based models (RF, XGB, LGBM)
  - **LinearExplainer:** For Logistic Regression
- **SHAP Summary Plot:**
  - Shows top features pushing predictions toward 1 (Converted) or 0 (Not Converted)
  - Logged as .png in MLflow
- **Fail-Safe Handling:**
  - If SHAP fails due to model incompatibility or data mismatch, logs warning

---

16. Data Drift Detection

Performed via run_drift() and log_evidently_report() using Evidently AI.

- **Reference vs. Current Split:**

    - Train vs. Test sets from initial stratified split
- **Reports Generated:**
    - DataDriftPreset and DataSummaryPreset
    - Saved as .html and .json
    - Logged to MLflow experiment: *Lead Conversion Prediction Evidently*
- **Metrics Logged:**
    - Drifted columns count and share
    - Row count and column count
    - Per-feature value drift
- **Automation:**
    - Automatically aligns common columns and avoids drift failures due to schema mismatches

---

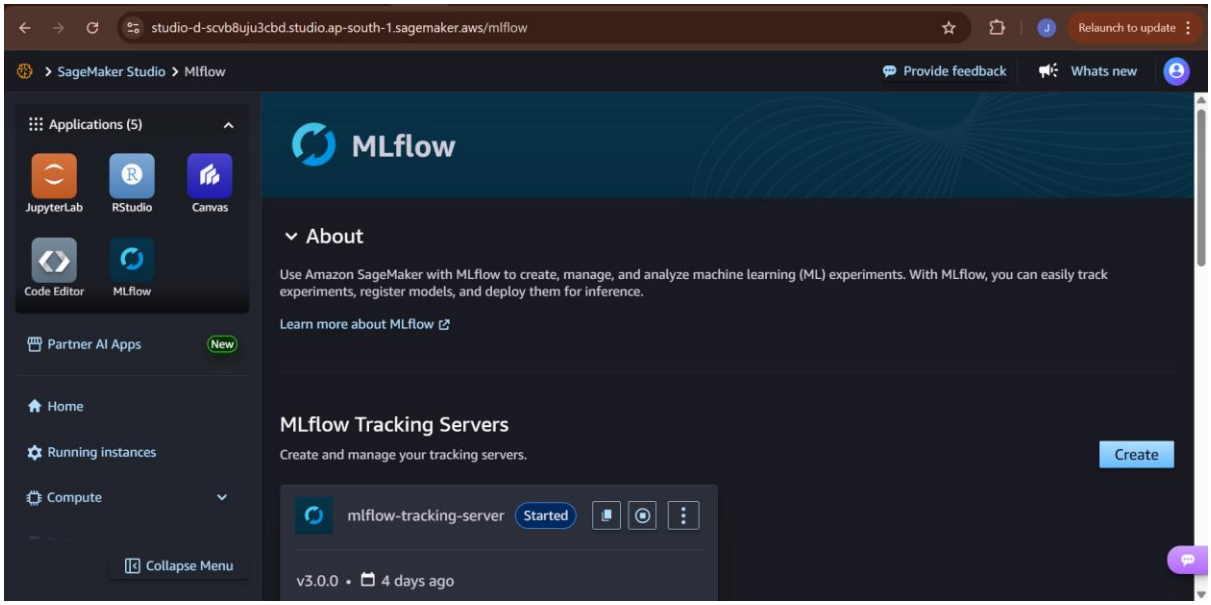
Part 5: MLOps, Deployment & Monitoring

---

16. Experiment Tracking with MLflow

16.1. Setting Up the MLflow Tracking Server

To centralize experiment tracking, we deploy an MLflow tracking server on a small EC2 instance within our VPC. This provides a persistent, shared location for all developers to log and view experiment results. The server is exposed via its private IP within the VPC, and the security group is configured to allow traffic on port 5000.



16.2. Integrating MLflow into the Training Pipeline

MLflow is seamlessly woven into our `train_pipeline` function.

- `mlflow.set_tracking_uri()`: Points the logging client to our EC2-hosted server.
- `mlflow.set_experiment()`: Organizes runs under a specific experiment name, "Lead Conversion Prediction".
- `with mlflow.start_run()`:: Creates a new run context for each model, automatically capturing metadata.
- `mlflow.log_param()` & `mlflow.log_metric()`: Logs hyperparameters and evaluation scores.
- `mlflow.sklearn.log_model()`: Logs the trained model object, its dependencies, and a `conda.yaml` file, making it fully reproducible.
- `mlflow.log_artifact()`: Logs supplementary files like the SHAP plot.

---

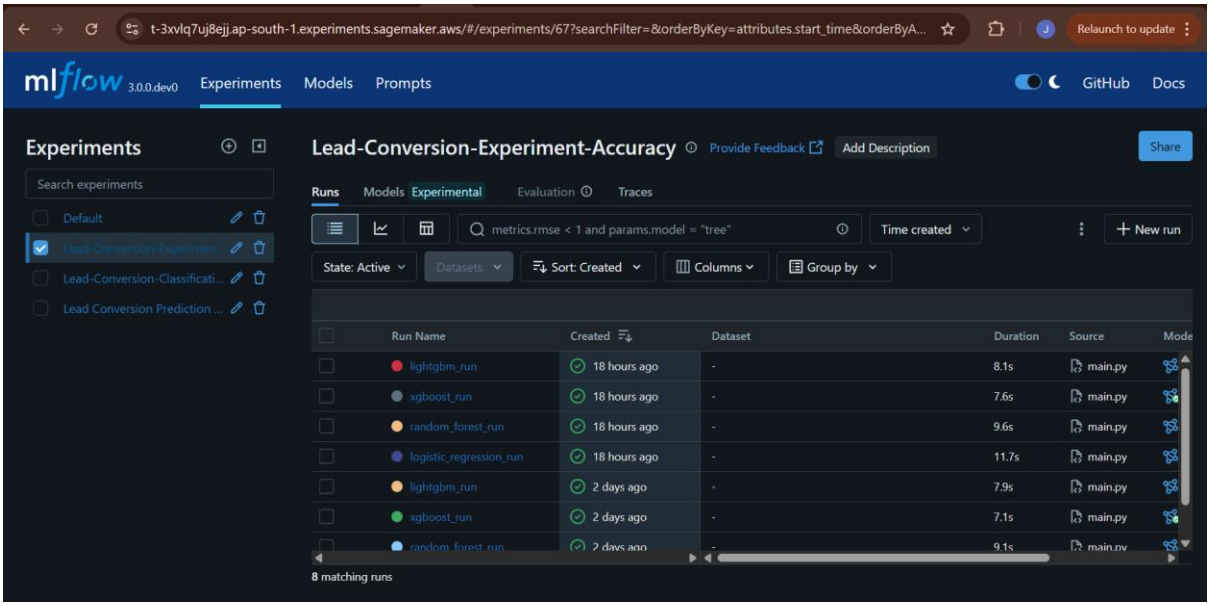
17. Model Registration and Lifecycle Management

17.1. Saving and Registering the Champion Model

After evaluating all models, our pipeline identifies the "champion" model (typically the one with the highest ROC-AUC or F1-score). This model is then prepared for production.

1. **Retrain:** The champion model is retrained on the *entire* dataset (training + validation) to learn from all available data.

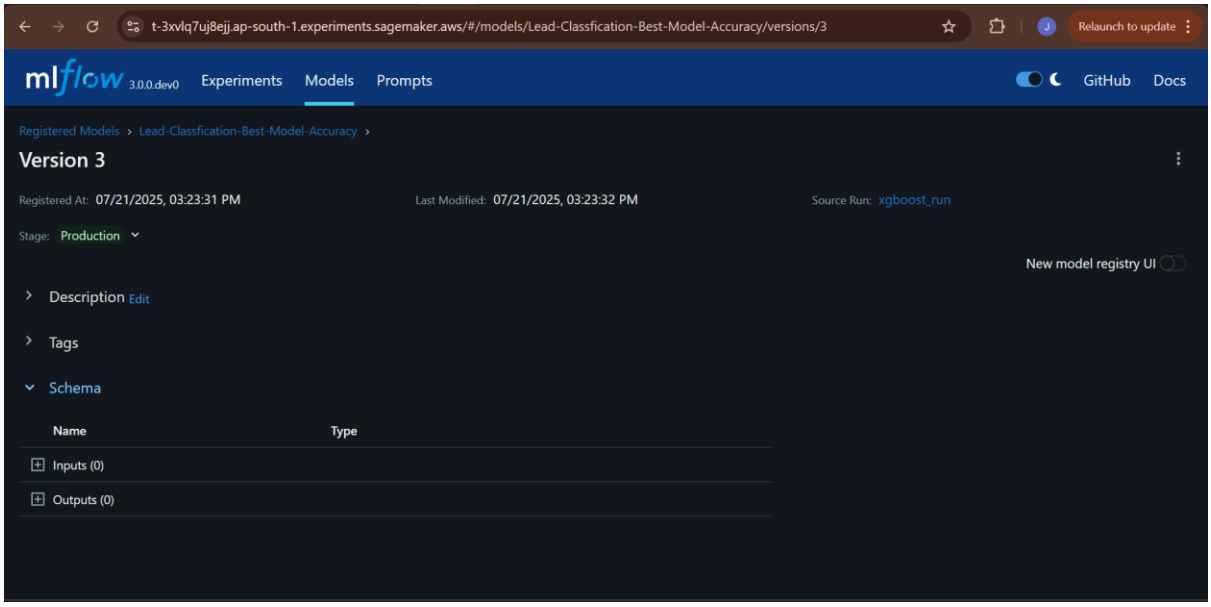
2. **Build Full Pipeline:** The retrained model is combined with the fitted `preprocessor` into a single, end-to-end `scikit-learn` pipeline. This ensures that raw data can be fed directly into the final artifact for prediction.
3. **Log and Register:** This full pipeline is logged to a final MLflow run and then **registered** in the MLflow Model Registry with a specific name (e.g., `Lead-Conversion-Classification-Precision`).



17.2. The MLflow Model Registry

The Model Registry is a central hub for managing the lifecycle of our models. It allows us to:

- **Version Models:** Every time we register a model with the same name, a new version is created.
- **Manage Stages:** We can assign stages to model versions, such as **Staging**, **Production**, or **Archived**.
- **Transition Models:** We can programmatically transition our newly registered model to the "Production" stage, signaling that it is the official version to be used for inference. This is a key step in controlled, automated deployments.



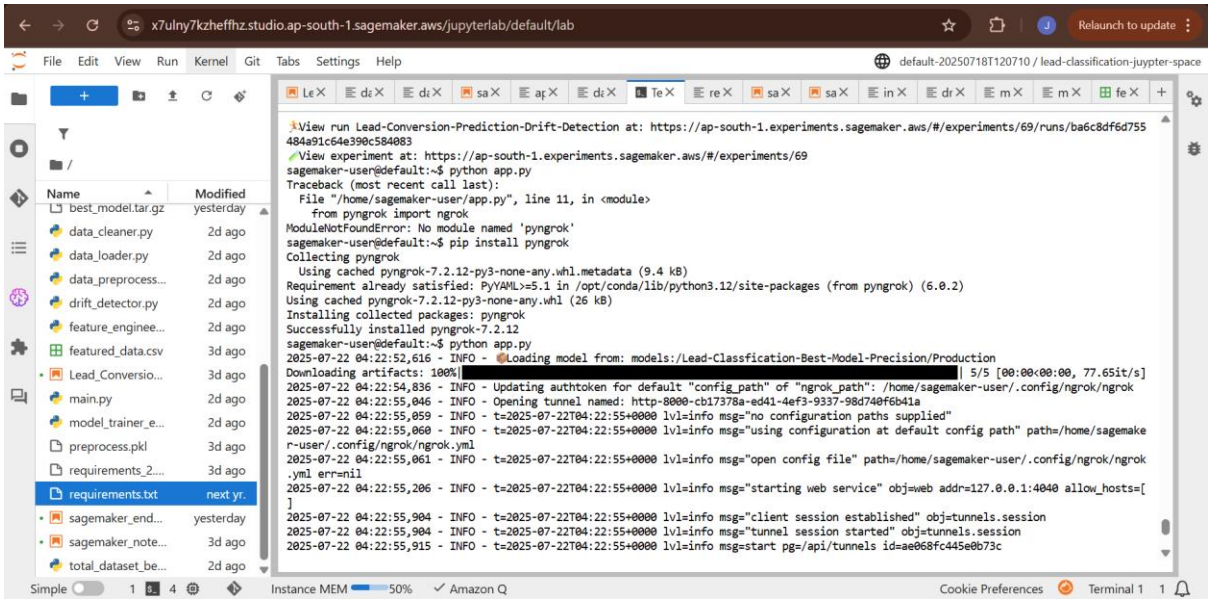
---

18. Serving and Inference

18.1. Loading the Production Model from the Registry

To ensure our application always uses the correct, approved model, we have a function that automatically discovers and loads the latest "Production" version from the MLflow Model Registry.

- `get_latest_production_model_name()`: This function connects to the MLflow client, searches the registry, and returns the URI of the model currently in the "Production" stage.
- `mlflow.sklearn.load_model()`: This function takes the model URI and loads the complete, deployment-ready pipeline into memory.



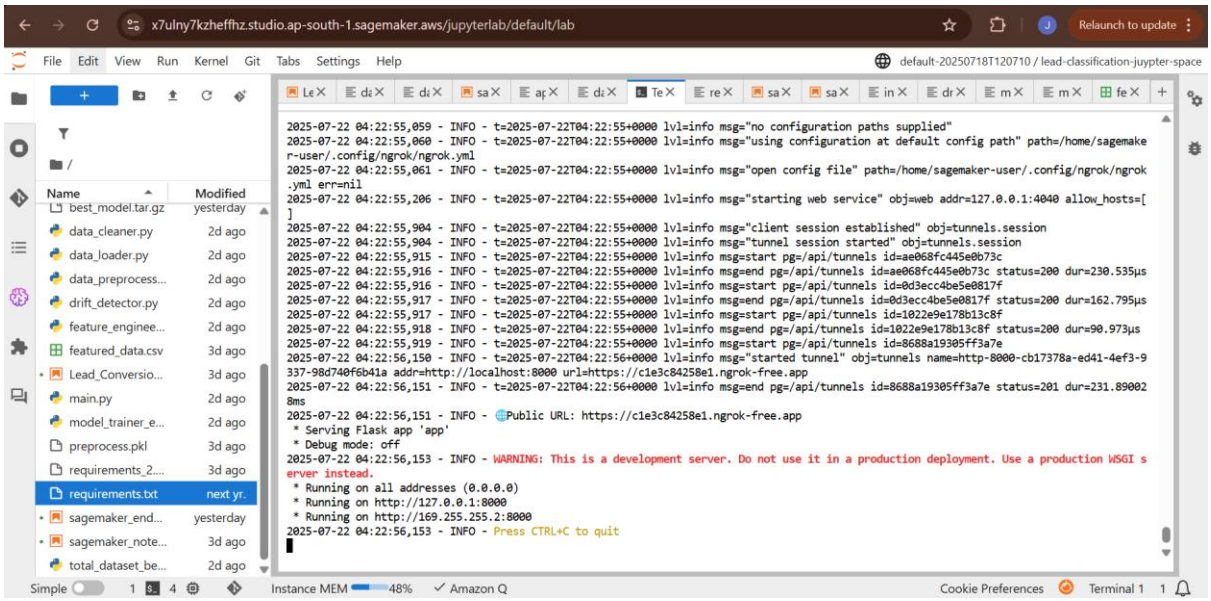
18.2. Real-Time Prediction with Flask and Ngrok

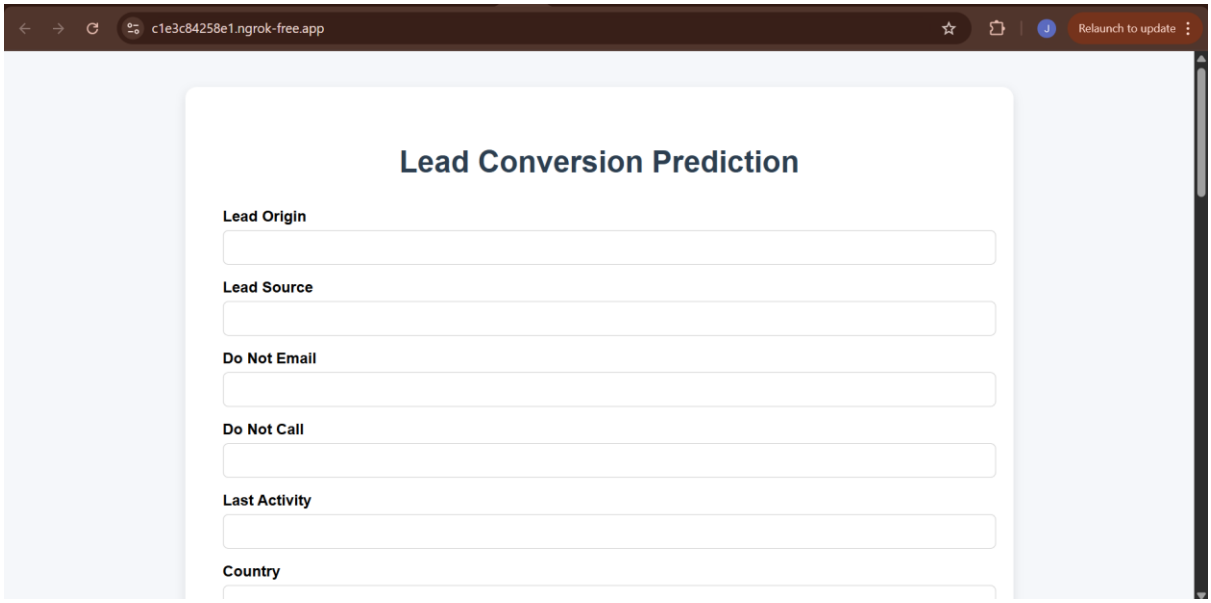
For testing and demonstration, we wrap our prediction logic in a lightweight Flask web application.

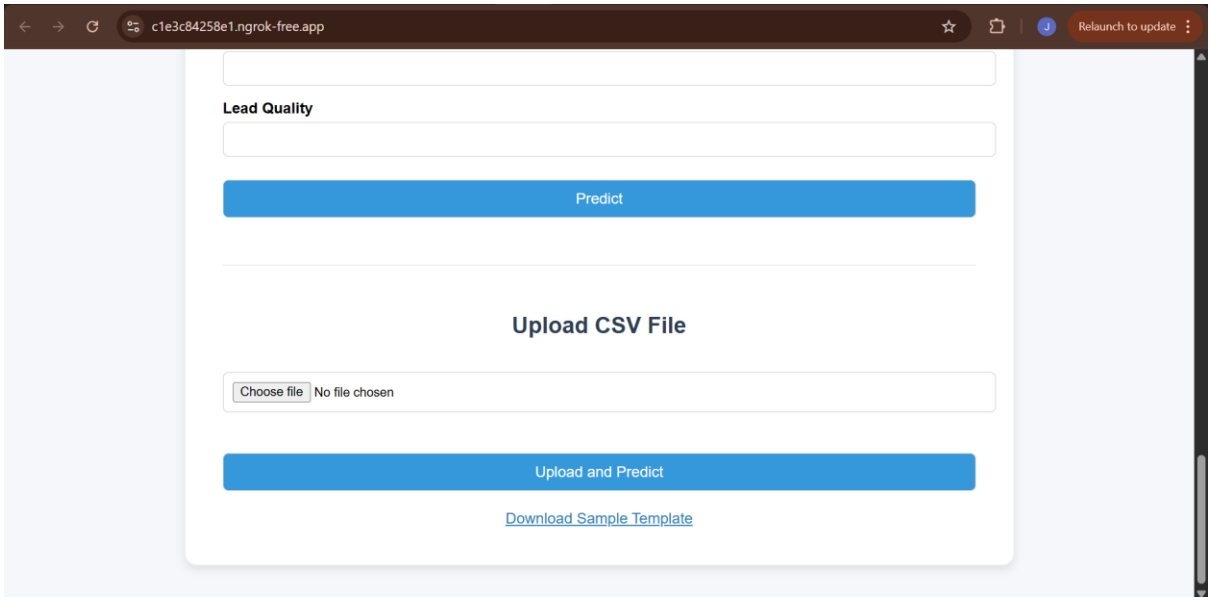
- **Flask App (`app.py`):**
    1. **Initialization:** Sets up the Flask app and connects to the MLflow tracking server.
    2. **Model Loading:** On startup, it calls the functions to load the latest production model from the registry.
    3. **`/predict` Route:** Defines an API endpoint that accepts POST requests with lead data (e.g., in CSV or JSON format).
    4. **Prediction Logic:** It takes the incoming data, passes it through the loaded pipeline's `.predict()` method, and returns the prediction (0 or 1) in a JSON response.

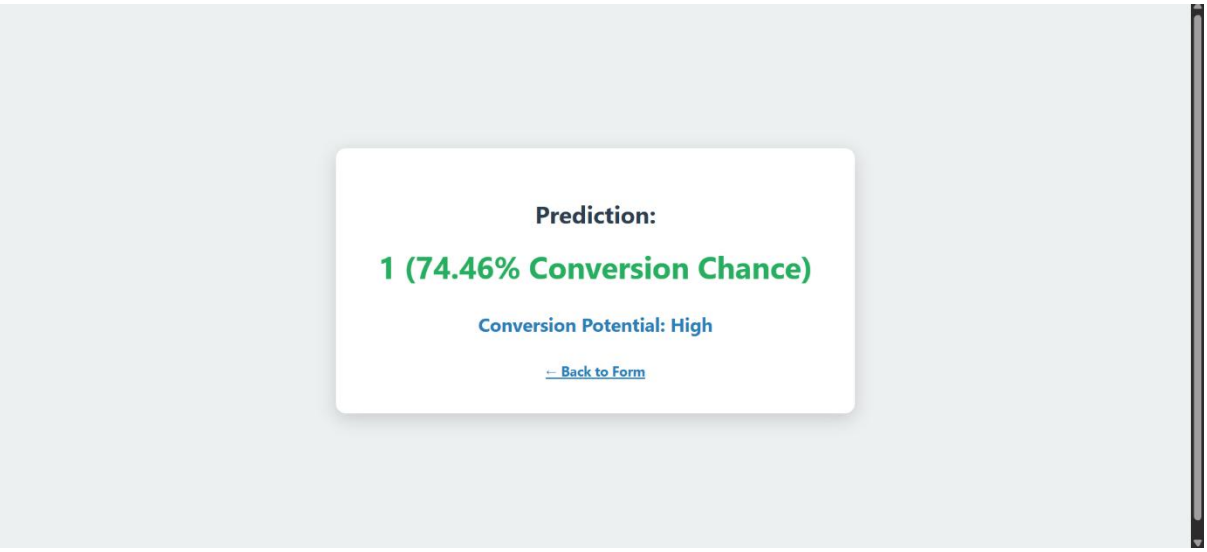
- **Ngrok Tunnel:**

    1. The Flask app is run locally (`python app.py`).
    2. The `ngrok http 8080` command is run in a separate terminal.
    3. Ngrok creates a secure public URL that tunnels requests directly to the local Flask application, making it accessible from the internet for end-to-end testing.




c1e3c84258e1.ngrok-free.app
Relaunch to update
Lead Conversion Prediction
Lead Origin
Lead Source
Do Not Email
Do Not Call
Last Activity
Country


c1e3c84258e1.ngrok-free.app
Relaunch to update
Lead Quality
Predict
Upload CSV File
Choose file
No file chosen
Upload and Predict
Download Sample Template



---

19. Post-Deployment: Drift Detection

19.1. Monitoring for Data Drift with Evidently AI

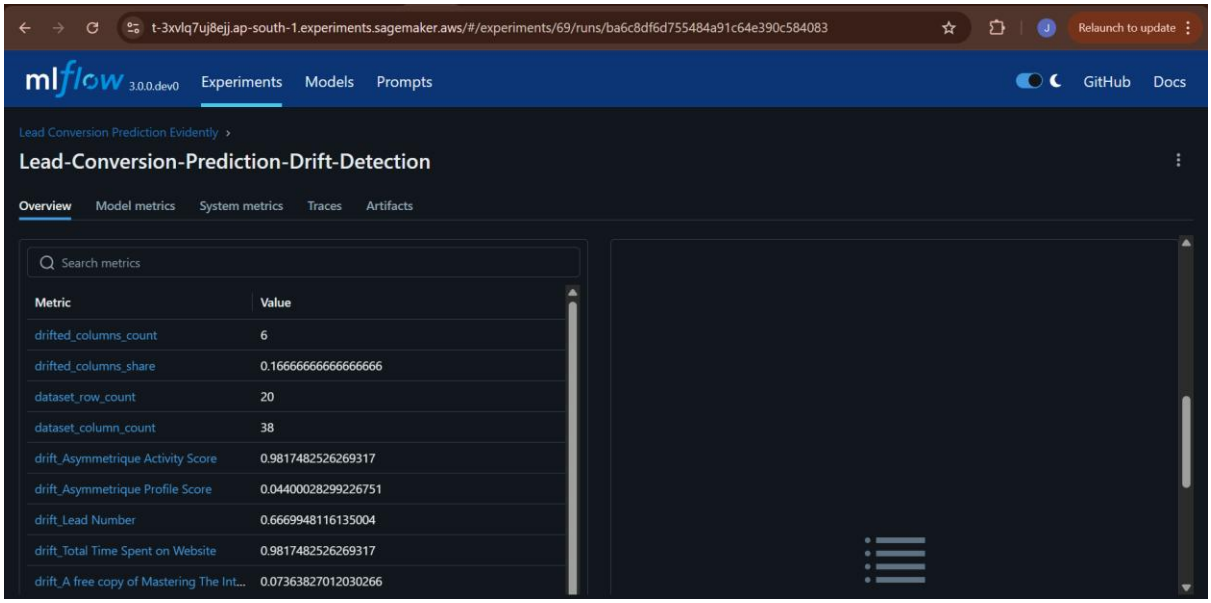
To ensure our model's performance doesn't degrade over time, we proactively monitor for **data drift**—a change in the statistical properties of the production data compared to the training data.

- **Tool:** We use the **Evidently AI** library, which specializes in generating comprehensive data drift and model performance reports.
- **Process:** We periodically run a script that takes our original training data as a *reference* dataset and a sample of recent production data as the *current* dataset.
- **Report Generation:** Evidently's `DataDriftPreset` generates a detailed report comparing the distributions of every feature between the two datasets.

19.2. Automating Drift Analysis and Logging

This process is automated and integrated with MLflow for tracking.

1. **Start MLflow Run:** A new run is initiated to store the drift analysis results.
2. **Generate Report:** The Evidently drift report is generated in two formats: a visual HTML file and a structured JSON object.
3. **Log HTML Report:** The interactive HTML report is logged as an artifact to MLflow, allowing for detailed visual inspection.
4. **Extract and Log Metrics:** The script parses the JSON report to extract key metrics, such as the `Share of Drifted Columns` and per-feature drift scores. These numerical values are logged as metrics in MLflow, making it easy to track drift trends over time and set up automated alerts.



---

20. Future State: Full Automation with MWAA

20.1. Orchestrating the MLOps Lifecycle with Airflow

To achieve full, hands-off automation, the entire MLOps workflow is orchestrated using **Amazon Managed Workflows for Apache Airflow (MWAA)**. MWAA is a managed service that makes it easy to run Apache Airflow on AWS. The entire automated process—from data drift detection to model retraining and deployment—is defined as a **Directed Acyclic Graph (DAG)**, which is a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies.

20.2. Setting up the MWAA Environment

Before deploying the DAG, you must set up the MWAA environment and its required resources.

Step 1: Create an S3 Bucket for MWAA Assets MWAA uses an S3 bucket to store your DAGs, custom plugins, and Python dependencies (`requirements.txt`).

1. Create a new S3 bucket with a unique name (e.g., `mwaa-lead-conversion-dags`).
2. Enable **versioning** on this bucket to keep a history of your DAGs and configuration files.

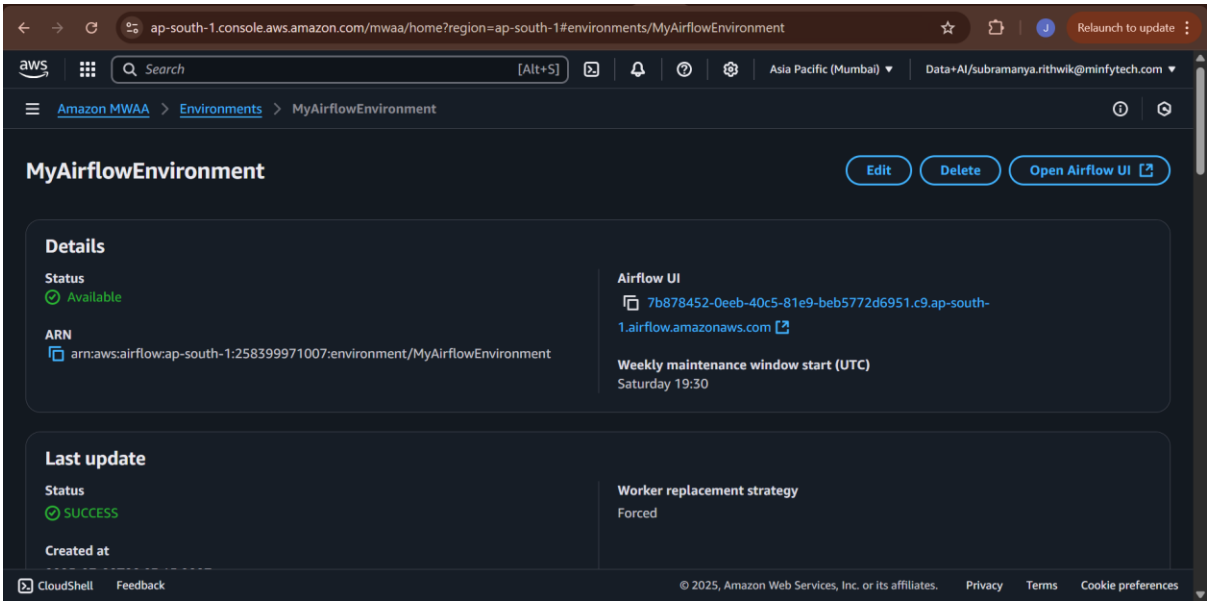
Step 2: Configure the VPC and Networking MWAA requires a specific network setup to run securely and access other resources.

1. **Use the same VPC:** The MWAA environment must be launched in the same VPC as your Redshift, SageMaker, and MLflow resources to communicate with them privately via the VPC endpoints.

2. **Subnets:** You must provide at least two **private subnets** in different Availability Zones for high availability.
3. **Internet Access:** The private subnets must have a route to a **NAT Gateway**. This is required for the Airflow workers to install Python dependencies from public repositories like PyPI.
4. **Security Group:** The MWAA environment should use the same `mlops-sg` security group created in Part 2. This ensures it has the necessary inbound and outbound rules to communicate with the MLflow server, Redshift, S3, and other services.

Step 3: Create the MWAA Environment

1. Navigate to the **Managed Workflows for Apache Airflow** service in the AWS Console.
2. Click **Create environment**.
3. **Details:**
    - **Name:** `MyAirflowEnvironment`
    - **Airflow version:** Choose a recent, stable version (e.g., 2.8.1).
    - **S3 bucket:** Browse and select the S3 bucket you created in Step 1.
    - **DAGs folder path:** `dags`
    - **Requirements file:** `requirements.txt`
4. **Networking:** Select your VPC, the private subnets, and the `mlops-sg` security group.
5. **Execution role:** Create a new IAM role that grants MWAA permissions to access its S3 bucket and CloudWatch Logs. Attach policies that also allow it to interact with SageMaker, Redshift, and any other services your DAG will orchestrate.
6. Click **Create environment**. Provisioning can take up to 30 minutes.



20.3. Airflow Workflow Lead Conversion Retraining

Structure the Project S3 Folder: On your S3 bucket, create the following folder structure:

airflow-lead23/

├── dags/

│ └── lead_retrain_dag.py ← Apache Airflow DAG

│

├── requirements/

│ └── requirements.txt ← Python packages required by MWAA and our code

│

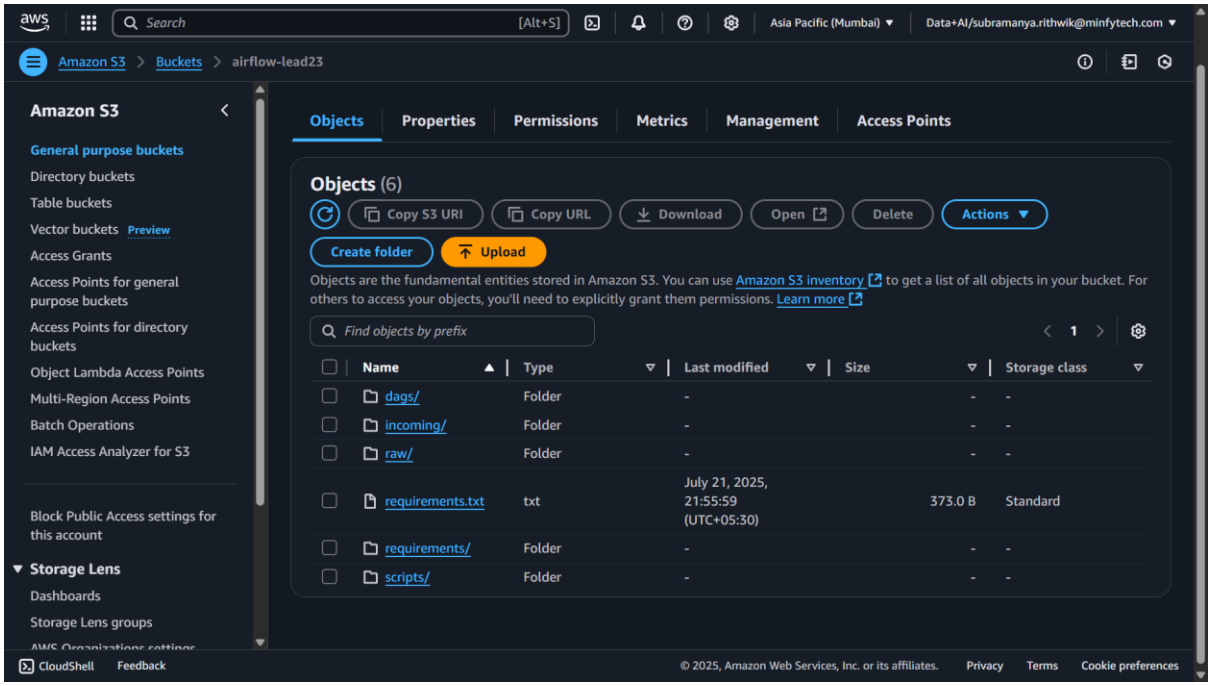
├── incoming/

│ └── new_data.csv ← New uploaded dataset to trigger drift detection

│

├── scripts/

│ └── main.py ← Contains entire ML pipeline (load → clean → train → register)



Automate an end-to-end retraining pipeline that:
1. Detects data drift between incoming lead data (from S3) and reference data (from Redshift).

2. Retrains the lead conversion model (main.py in S3) only if drift > 0.3.
3. Skips retraining if drift is within acceptable threshold (≤ 0.3).

Architecture Overview:

Component	Technology Used
Workflow Engine	Apache Airflow (MWAA)
Cloud Provider	AWS
Data Storage	S3 (for incoming datasets and main.py)
Data Warehouse	Amazon Redshift Serverless
Drift Detection	Evidently AI
Model Training	Python (main.py script via BashOperator)

Data Sources

New Data

• Source: S3://airflow-lead23/incoming/new_data.csv
• Loaded by: boto3.download_file

Reference Data

• Source: Amazon Redshift table: lead_scoring
• Loaded by: boto3.client('redshift-data')

DAG: lead_conversion_retrain

A DAG scheduled daily to check for drift and conditionally retrain the model.

Workflow Steps

Step 1: Install Python Dependencies

• Operator: BashOperator
• Command: pip install -r /usr/local/airflow/requirements/requirements.txt

Step 2: Check for Data Drift

• Operator: PythonOperator
• Task: check_drift
• Performs the following:
  - Downloads new data from S3
  - Pulls reference data from Redshift using SQL
  - Renames and cleans columns
  - Runs Evidently's DataDriftPreset
  - Calculates drift score and pushes it to XCom

Step 3: Branch on Drift Score

• Operator: BranchPythonOperator
• Decides to retrain or skip based on drift threshold (0.3)

Step 4A: Retrain Model (If Drift > 0.3)

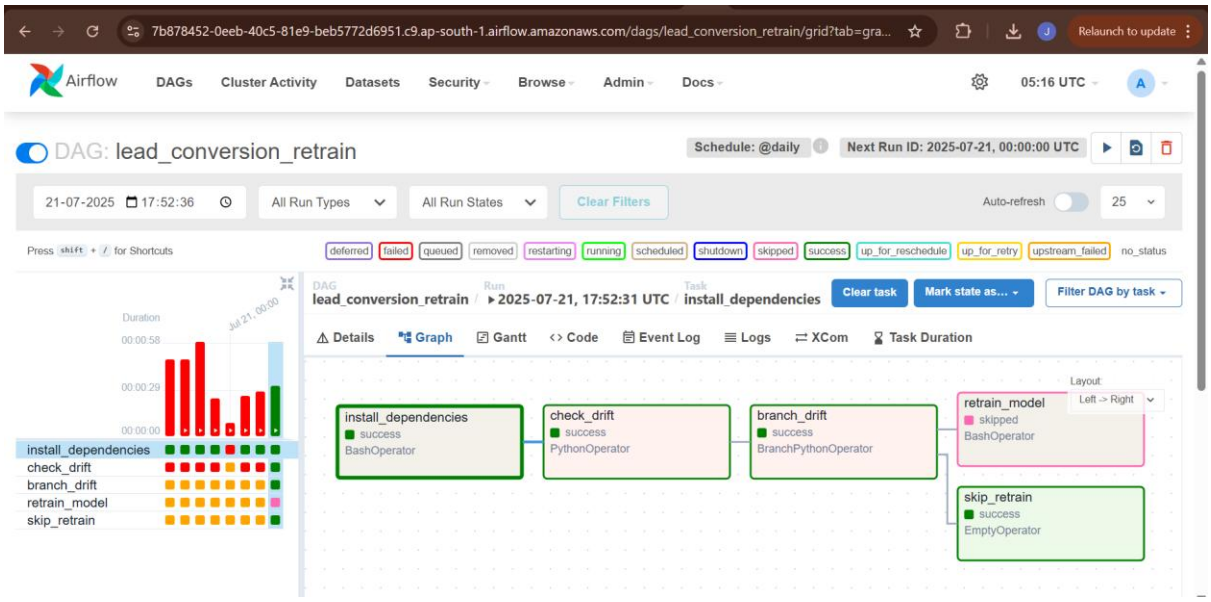
• Operator: BashOperator
• Command: aws s3 cp s3://airflow-lead23/scripts/main.py ./ && python main.py

Step 4B: Skip Retraining (If Drift ≤ 0.3)

• Operator: DummyOperator
• Ends DAG branch with no retraining

Python Dependencies

• evidently>=0.4.14
• boto3
• pandas
• numpy
• psycopg2
These are listed in /usr/local/airflow/requirements/requirements.txt



21. SageMaker Model Deployment

To deploy a trained lead conversion model as a SageMaker real-time inference endpoint using a custom script and required artifacts including model, preprocessor, and data transformation logic.

21.1 Folder Structure (Used in SageMaker)

All files are placed in the working directory: lead_model_deploy/.

```
lead_model_deploy/
├── inference.py              # Inference logic
├── data_cleaner.py           # Data preprocessing
├── feature_engineer.py       # Feature generation logic
```

```
├── best_model.pkl              # Trained model (pickle)
├── preprocessor.pkl            # Transformation pipeline (pickle)
├── requirements.txt            # Python dependency list
├── sagemaker_endpoint_trail.ipynb # Notebook to trigger deployment
├── model_bundle.tar.gz         # Archive uploaded to S3
```

21.2 Model Packaging and Upload

Compress the artifacts into a .tar.gz:

tar -czvf model_bundle.tar.gz inference.py data_cleaner.py feature_engineer.py best_model.pkl preprocessor.pkl

21.3 Upload the model bundle to S3:

aws s3 cp model_bundle.tar.gz s3://lead-classification-bucket/model_bundle.tar.gz

S3 Path: s3://lead-classification-bucket/model_bundle.tar.gz

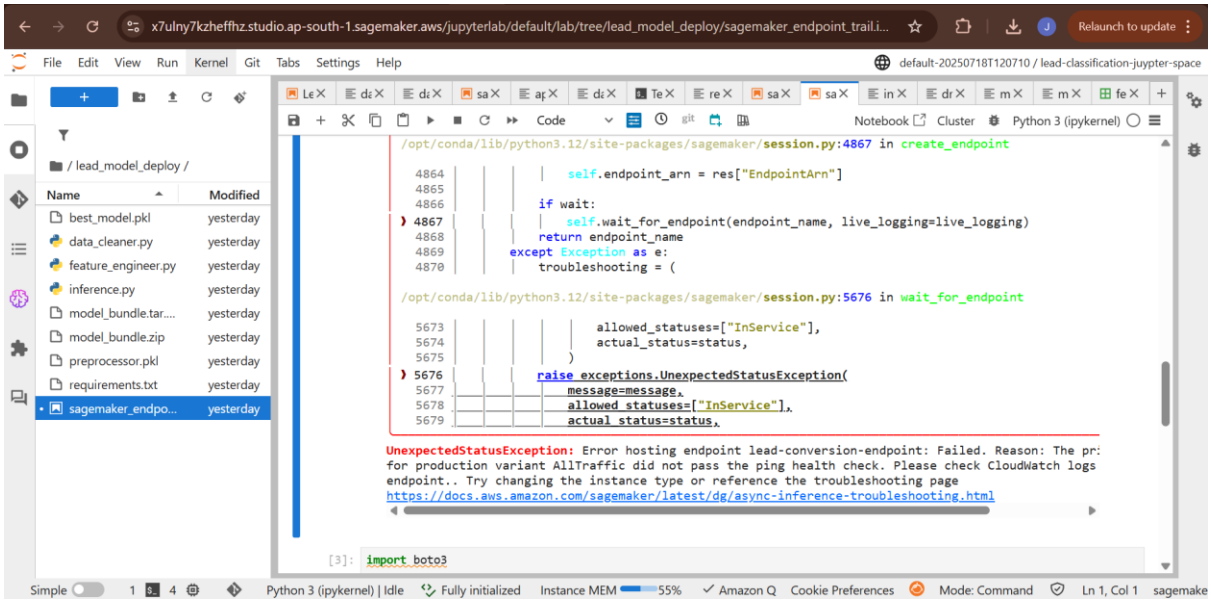
21.4 SageMaker Endpoint Creation via Notebook

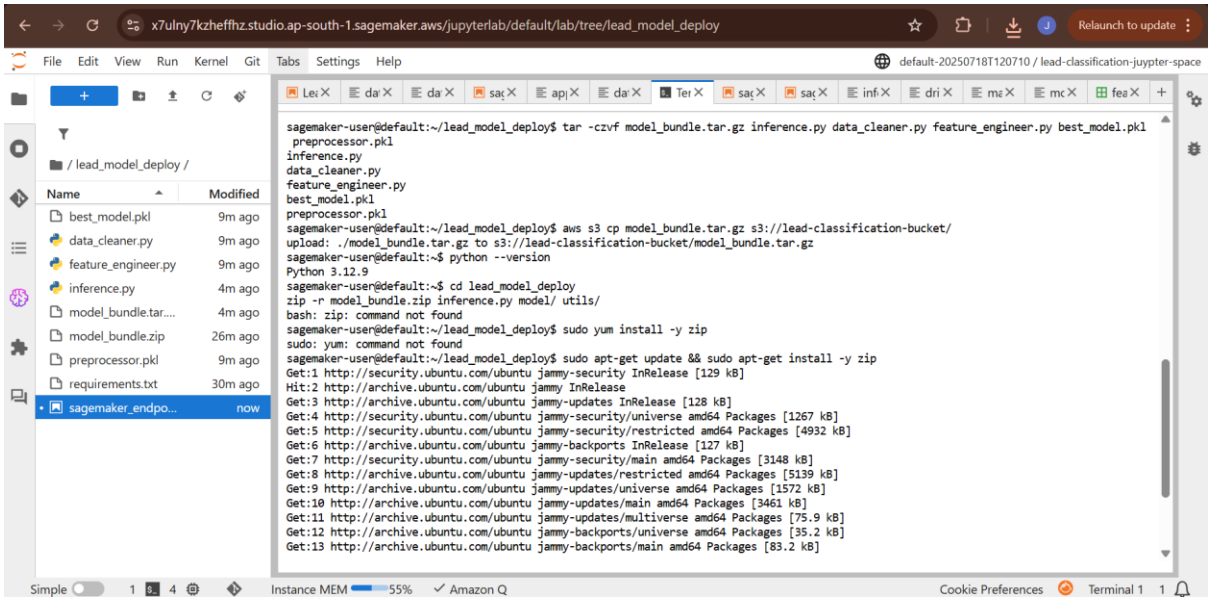
Using the sagemaker_endpoint_trail.ipynb, the model was deployed with the SKLearnModel class.

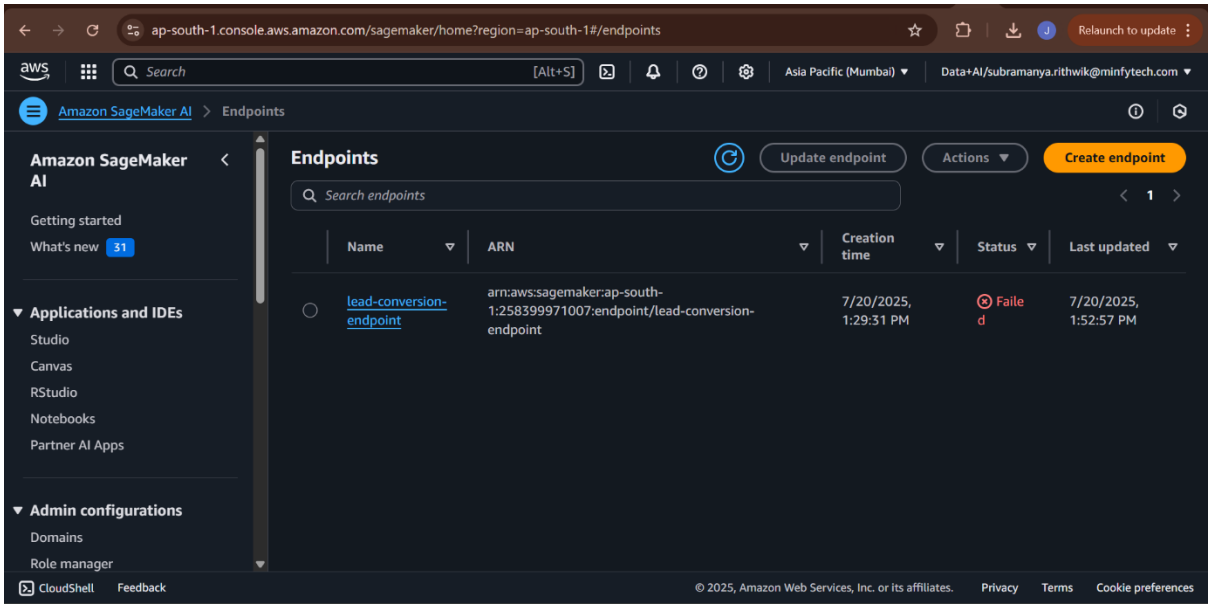
```
from sagemaker.sklearn.model import SKLearnModel
from sagemaker import get_execution_role, Session

role = get_execution_role()

model = SKLearnModel(
    model_data="s3://lead-classification-bucket/model_bundle.tar.gz",
    role=role,
    entry_point="inference.py",
    framework_version="1.2-1",
    sagemaker_session=Session()
)

predictor = model.deploy(
    instance_type="ml.m5.large",
    initial_instance_count=1,
    endpoint_name="lead-conversion-endpoint"
)
```

21.5. Deployment Result

Field	Value
Endpoint Name	lead-conversion-endpoint
Region	ap-south-1
Model Bundle	model_bundle.tar.gz
Status	Failed







21.6 Root Cause Analysis of Failure

The endpoint creation failed. The probable reasons is:

- Not passing health checks

Part 6: Appendix

---

22. Technology & Libraries Stack

- **Cloud Platform:** Amazon Web Services (AWS)
- **Data Storage:** Amazon S3, Amazon Redshift
- **Data Integration (ETL):** AWS Glue (Studio, Crawlers, Data Catalog)
- **Security & Networking:** AWS IAM, AWS VPC, AWS Secrets Manager, AWS KMS
- **ML Development:** Amazon SageMaker Studio Lab
- **MLOps & Tracking:** MLflow
- **Orchestration (Future):** Managed Workflows for Apache Airflow (MWAA)
- **Serving:** Flask, Ngrok
- **Core Python Libraries:**
    - **Data Handling:** `pandas`, `numpy`
    - **Visualization:** `matplotlib`, `seaborn`
    - **ML & Preprocessing:** `scikit-learn`
    - **Advanced Models:** `xgboost`, `lightgbm`
    - **Model Explainability:** `shap`
    - **Data Drift:** `evidently`

- **AWS Interaction:** `boto3`
- **Serialization:** `joblib`

23. Resources

- **Official Documentations:**
  - AWS Glue
  - Amazon SageMaker
  - MLflow
  - Evidently AI

24. GitHub Repository

- **GitHub link:** https://github.com/Rithwik-Minfy/Lead-Conversion-Classfication-Capstone