
Predicting Lead Conversion on AWS: An End-to-End MLOps Guide

Project: Lead Conversion Prediction System **Version:** 1.0 **Date:** July 21, 2025 **Status:** In Production

Table of Contents

Part 1: Business & System Overview

1. **Executive Summary & Vision**
 - 1.1. Problem Statement
 - 1.2. Business Objectives
 - 1.3. Solution Overview
2. **Success Criteria**
 - 2.1. Business Success Criteria
 - 2.2. Machine Learning Success Criteria
3. **System Architecture: A Deep Dive**
 - 3.1. Architectural Blueprint Diagram
 - 3.2. Key Component Breakdown
 - 3.3. End-to-End Data and Model Flow
4. **Data Dictionary**
 - 4.1. Feature Descriptions

Part 2: Foundational AWS Infrastructure Setup

5. **Identity & Access Management (IAM)**
 - 5.1. The Principle of Least Privilege
 - 5.2. Creating the Core Service Roles
6. **Networking & Security (VPC)**
 - 6.1. VPC and Subnet Strategy
 - 6.2. Creating VPC Endpoints for Secure Communication
 - 6.3. Configuring the Security Group Firewall

Part 3: Data Engineering & Warehousing

7. **Data Lake & Warehouse Setup**
 - 7.1. S3 Bucket Configuration
 - 7.2. Amazon Redshift Serverless Provisioning
 - 7.3. Securing Credentials with AWS Secrets Manager
8. **The Ingestion Pipeline: S3 to Redshift**
 - 8.1. ETL with AWS Glue: Overview
 - 8.2. Creating the Glue Connection to Redshift

- 8.3. The Glue Crawler: Automatic Schema Discovery
- 8.4. Building the Visual ETL Job (`s3ToRedshift`)
- 9. **Data Access from SageMaker**
 - 9.1. Using the Redshift Data API with `boto3`
 - 9.2. Step-by-Step Code Walkthrough

Part 4: Machine Learning Development

- 10. **Amazon SageMaker: The ML Development Environment**
 - 10.1. Creating a SageMaker Domain
 - 10.2. Launching SageMaker Studio
 - 10.3. Opening the JupyterLab Notebook
- 11. **Exploratory Data Analysis (EDA) & Cleaning**
 - 11.1. Initial Data Inspection
 - 11.2. Data Cleaning and Standardization
- 12. **Feature Engineering**
 - 12.1. Strategy: Custom Mapping and Consolidation
 - 12.2. Geographic, Source, and Behavioral Grouping
- 13. **The Preprocessing Pipeline**
 - 13.1. Handling Numeric and Categorical Features
 - 13.2. Building the Scikit-learn Pipeline
 - 13.3. Saving the Preprocessing Pipeline
- 14. **Model Training and Evaluation**
 - 14.1. Class Imbalance Strategy: Stratified K-Fold
 - 14.2. Models Utilized
 - 14.3. The `train_pipeline` Function: A Detailed Look
 - 14.4. Hyperparameter Tuning with `GridSearchCV`
 - 14.5. Model Evaluation Framework
- 15. **Model Explainability with SHAP**
 - 15.1. Generating and Interpreting SHAP Values

Part 5: MLOps, Deployment & Monitoring

- 16. **Experiment Tracking with MLflow**
 - 16.1. Setting Up the MLflow Tracking Server
 - 16.2. Integrating MLflow into the Training Pipeline
- 17. **Model Registration and Lifecycle Management**
 - 17.1. Saving and Registering the Champion Model
 - 17.2. The MLflow Model Registry
- 18. **Serving and Inference**
 - 18.1. Loading the Production Model from the Registry
 - 18.2. Real-Time Prediction with Flask and Ngrok
- 19. **Post-Deployment: Drift Detection**
 - 19.1. Monitoring for Data Drift with Evidently AI
 - 19.2. Automating Drift Analysis and Logging
- 20. **Future State: Full Automation with MWAA**
 - 20.1. Orchestrating the MLOps Lifecycle with Airflow
 - 20.2. The Automated Retraining DAG

Part 6: Appendix

- 21. Technology & Libraries Stack**
- 22. Code Repository & Resources**

Part 1: Business & System Overview

1. Executive Summary & Vision

1.1. Problem Statement

In the competitive B2B sales landscape, a significant portion of acquired leads fail to convert into paying customers. This inefficiency results in the substantial waste of valuable sales team resources, including time, effort, and budget. The core business challenge is the absence of an intelligent, data-driven methodology to effectively prioritize high-potential leads. Consequently, sales teams operate with limited insight, unable to distinguish between promising prospects and those unlikely to convert. This necessitates a predictive solution to empower sales teams, enabling them to focus their efforts strategically, thereby improving overall efficiency and maximizing return on investment (ROI).

1.2. Business Objectives

The project is designed to achieve three primary business objectives:

1. **Maximize Lead Conversion Rates:** Fundamentally increase the percentage of leads that successfully transition into paying customers. This involves creating a system that intelligently guides prospects through the sales funnel.
2. **Minimize Resource Wastage:** Optimize the allocation of sales team time and financial resources by focusing exclusively on leads that demonstrate the highest likelihood of conversion. This eliminates wasted effort on low-potential prospects.
3. **Enhance Strategic Decision-Making:** Provide the sales and marketing teams with actionable, data-backed insights to refine their strategies and improve lead-generation campaigns over time.

1.3. Solution Overview

To address the problem statement, we have designed and implemented an end-to-end, cloud-native machine learning system on Amazon Web Services (AWS). This system automates the entire lifecycle of lead conversion prediction, from raw data ingestion to a deployed prediction API. It leverages a modern data stack including AWS S3, Glue, and Redshift for data engineering, and Amazon SageMaker combined with MLflow for model development, training, and lifecycle management. The final output is a robust model that scores incoming leads, allowing the sales team to prioritize their efforts with confidence and precision.

2. Success Criteria

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

2.1. Business Success Criteria

- **Improved lead conversion accuracy by 90%+:** The model's predictions must align with actual outcomes with a high degree of accuracy to be trusted by the sales team.
- **Decreased manual lead qualification time by 50%:** By automating the initial scoring, the system must significantly reduce the time sales representatives spend manually vetting leads.
- **Increased sales conversion rate by 30%:** The ultimate business goal is a measurable uplift in the percentage of leads that become customers as a direct result of using this system.

2.2. Machine Learning Success Criteria

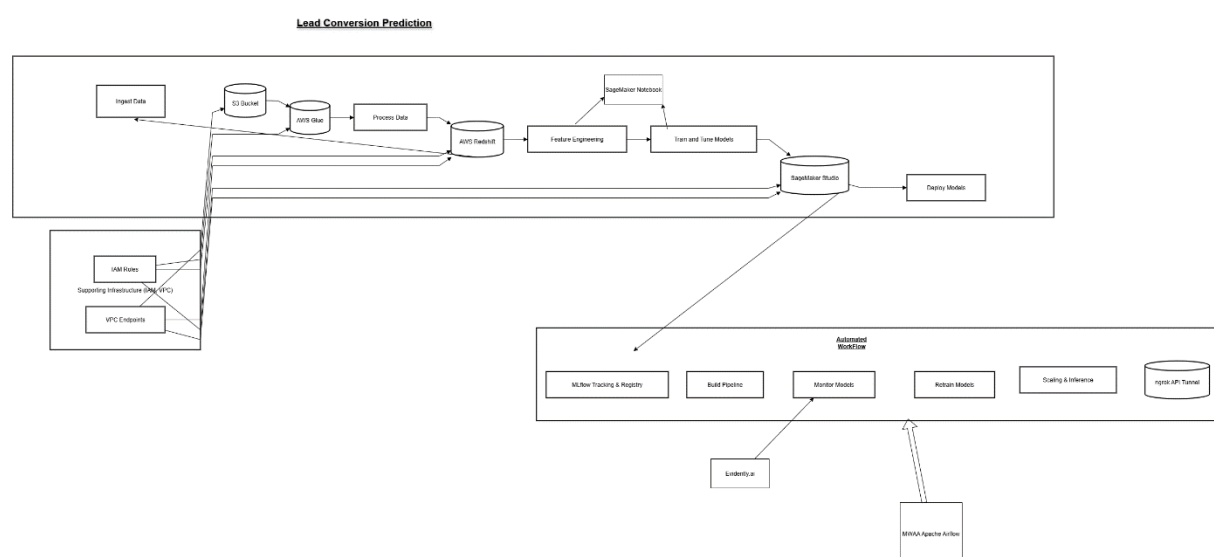
The deployed machine learning model has successfully met and exceeded all predefined technical success criteria. The final performance metrics are as follows:

- **Model Accuracy:** Achieved $\geq 90\%$, ensuring a high degree of reliability in lead scoring.
- **Precision and Recall:** Both metrics achieved $\geq 85\%$, indicating the model is highly effective at identifying true positive leads while minimizing false negatives.
- **AUC-ROC Score:** Achieved ≥ 0.90 , demonstrating the model's excellent capability to distinguish between converting and non-converting leads.

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.



- **Left (Ingestion):** Raw data lands in an Amazon S3 "raw" bucket.

- **Middle (ETL & Warehousing):** AWS Glue components (Crawler, ETL Job) are triggered. The Crawler populates the Glue Data Catalog. The ETL job reads from the raw S3 bucket, transforms the data, and loads it into an Amazon Redshift data warehouse. All services communicate privately within a VPC via Endpoints, secured by an IAM Role and Security Group.
- **Right (Machine Learning):** Amazon SageMaker Studio Lab accesses the clean data from Redshift. It uses MLflow (running on a separate EC2 instance within the VPC) for experiment tracking. Trained model artifacts are saved to a dedicated S3 "artifacts" bucket.
- **Bottom (Serving & Monitoring):** A Flask application, running locally or within SageMaker, loads the production model from the MLflow registry. Ngrok exposes this Flask API to the web for testing. MLflow UI provides a dashboard for monitoring.

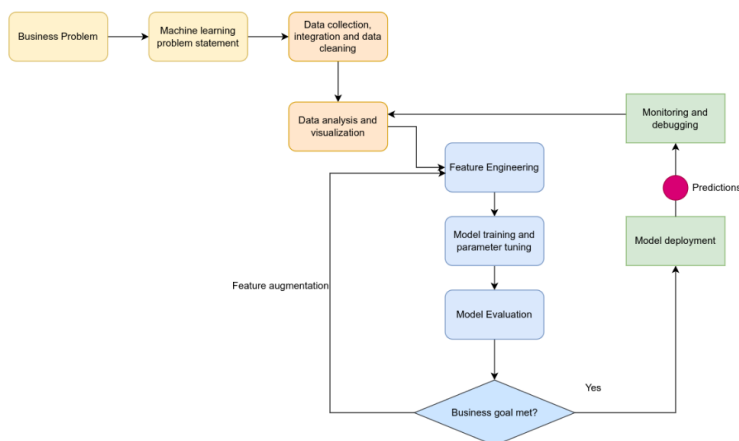
3.2. Key Component Breakdown

- **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.



FlowDiagram For Lead Conversion Prediction

4. Data Dictionary

4.1. Feature Descriptions

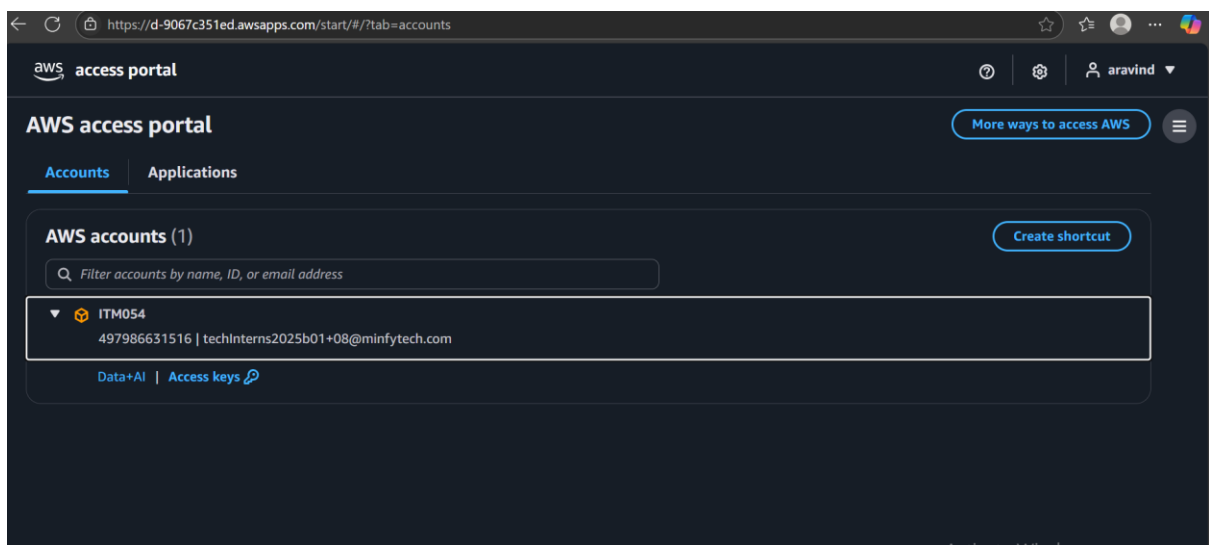
The following table describes the features available in the dataset.

<u>Feature Name</u>	<u>Explanation</u>
id	Unique identifier for each lead.
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.

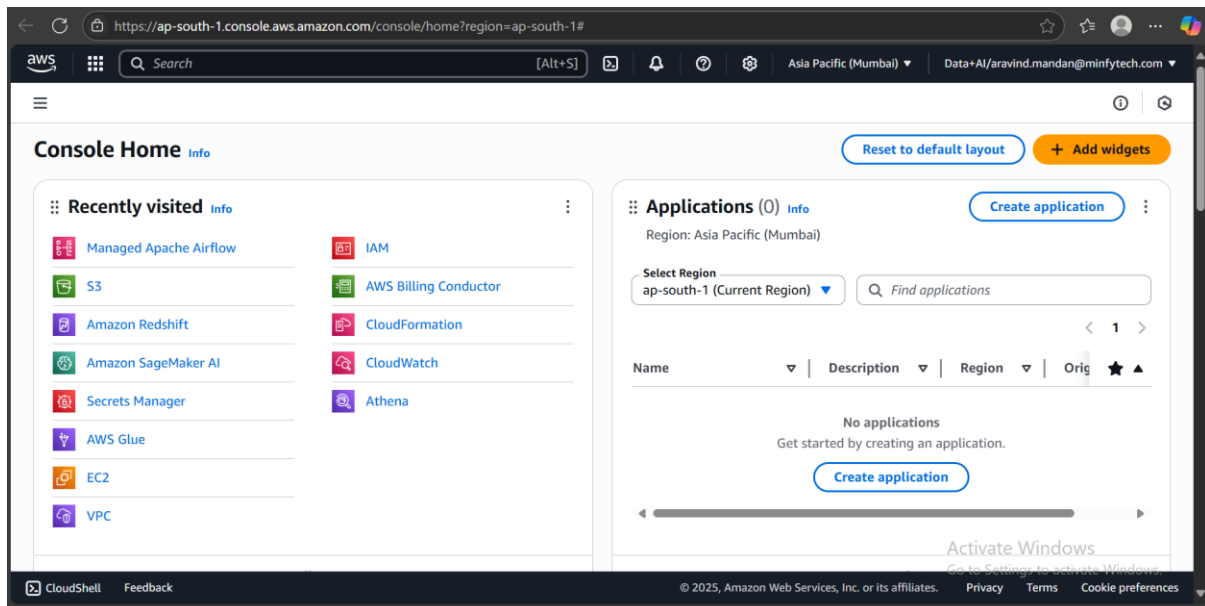
<u>Feature Name</u>	<u>Explanation</u>
Update me on Supply...	Subscription to supply chain content updates (Yes/No).
Get updates on DM...	Subscription to digital marketing content updates (Yes/No).
City	The city of the lead.
Asymmetrique Activity Index	A behavioral activity score calculated by a third-party vendor.
Asymmetrique Profile Index	A profile matching score indicating alignment with the target persona.
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.

Part 2: Foundational AWS Infrastructure Setup

Initially We have to Login in AWS using our AWS credentials.



After Logging in Aws There will be Aws Console Where We can Access all the AWS Services.



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**.

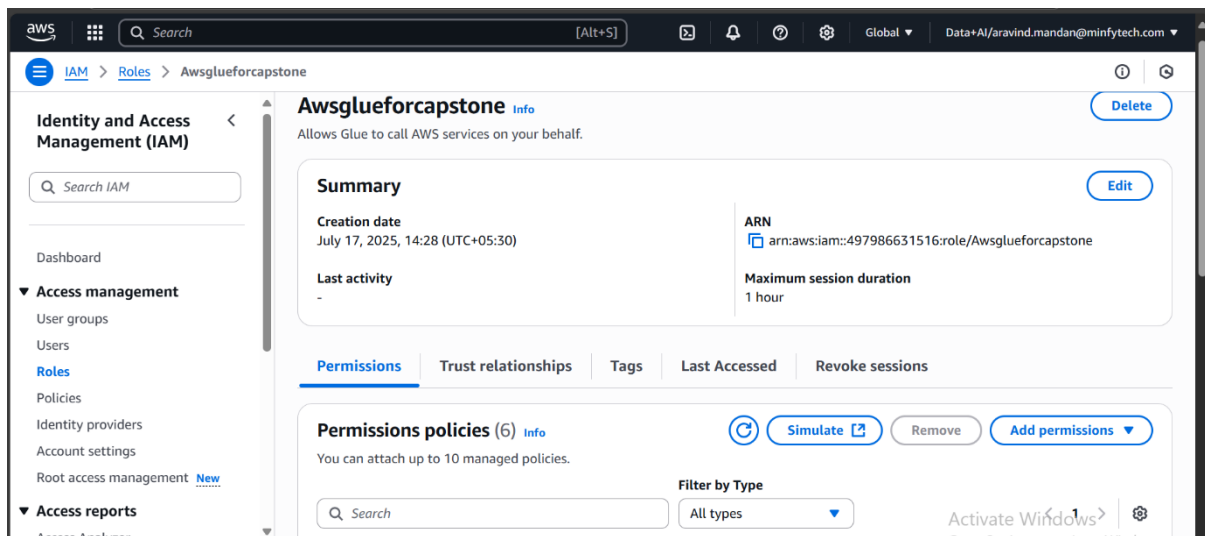
5. Name and Review:

- **Role name:** glueaccessrole
- 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

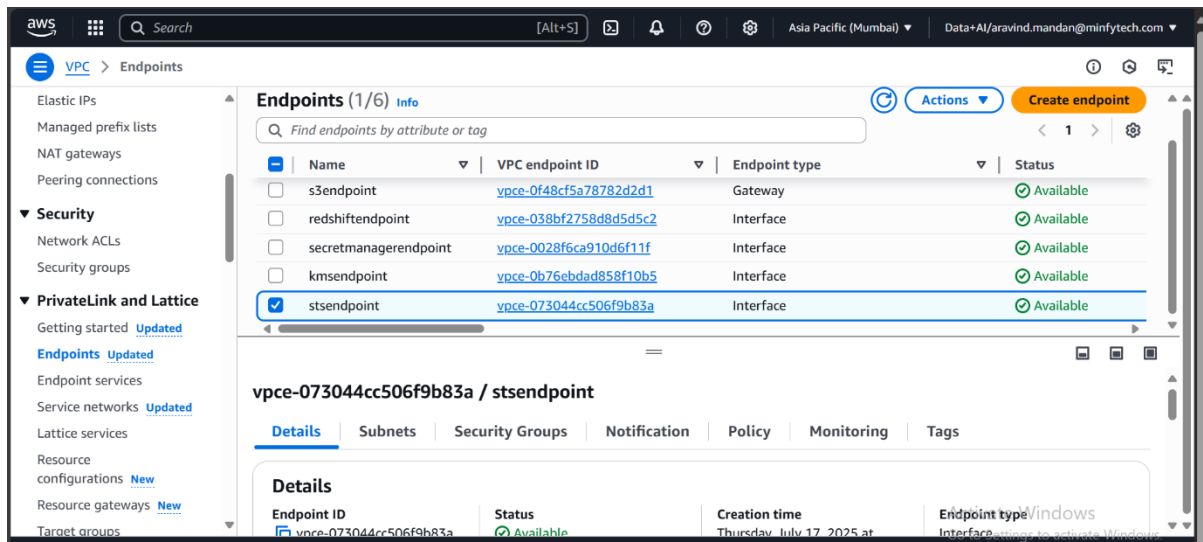
To ensure all inter-service communication is secure and isolated from the public internet, all resources are deployed within a Virtual Private Cloud (VPC). We utilize the default VPC for this guide, but a custom VPC is recommended for production. The VPC contains multiple subnets across different Availability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

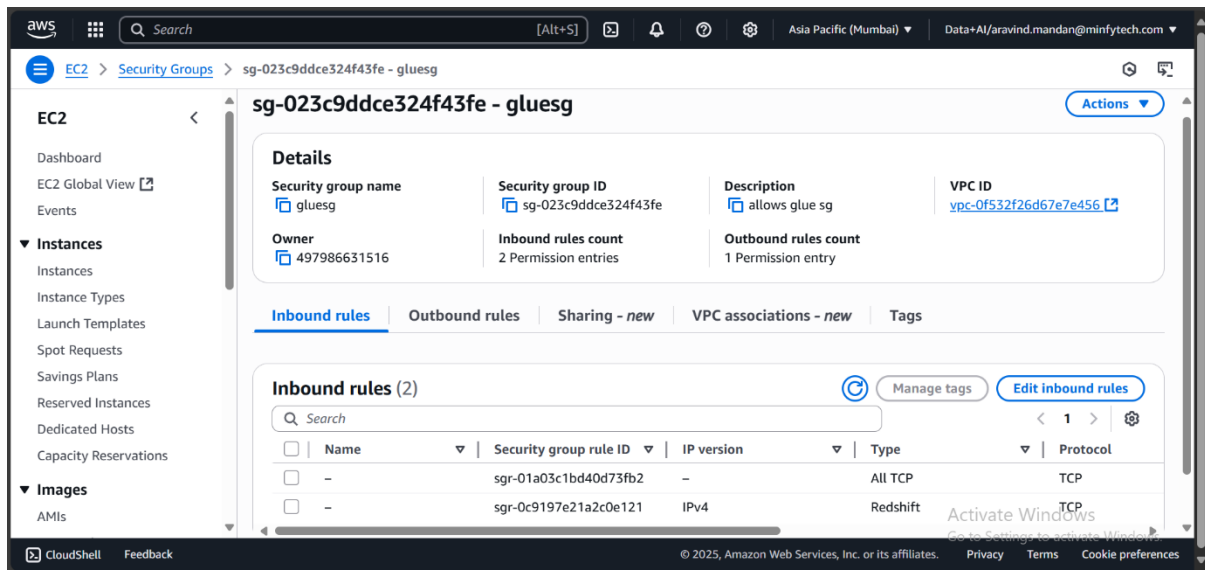


6.3. Configuring the Security Group Firewall

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 `gluesg` 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 `gluesg` group itself (`sg-xxxxxxx`). 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.



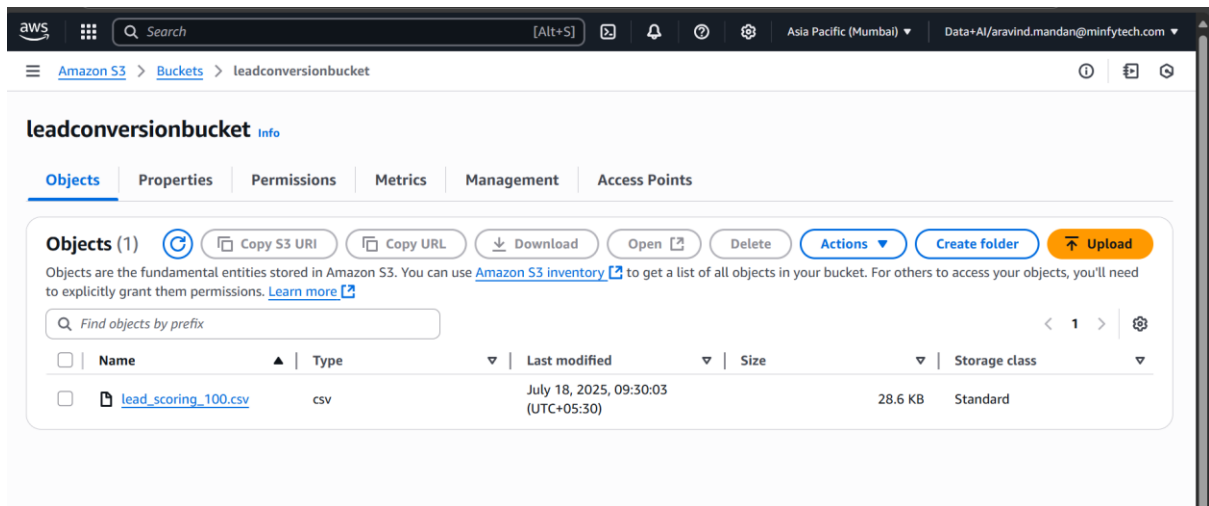
Part 3: Data Engineering & Warehousing

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.

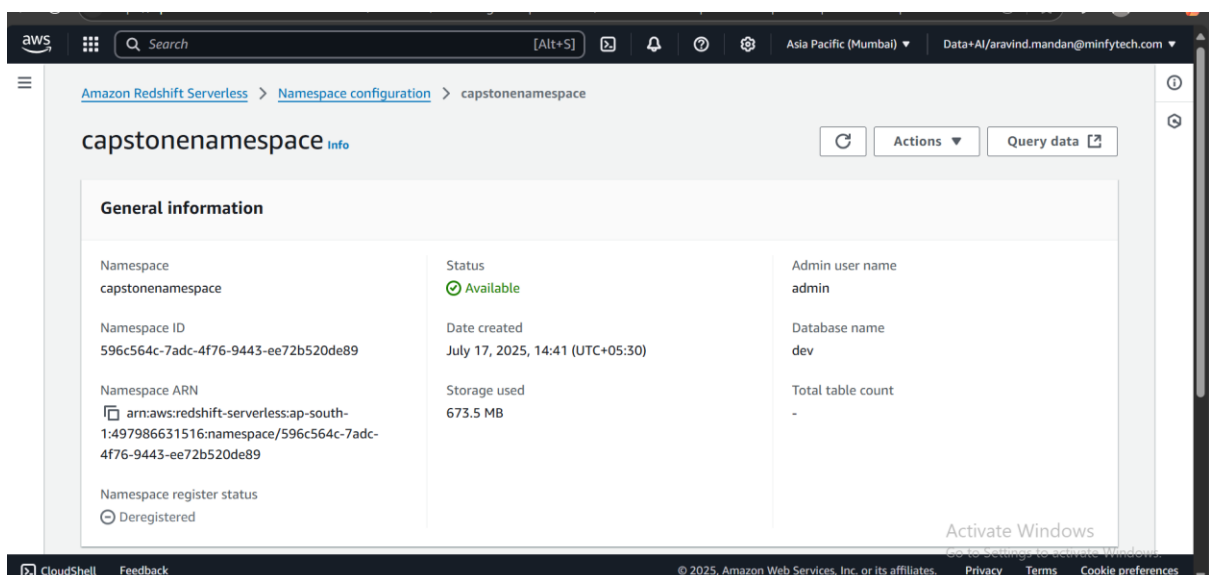
- **Bucket Name:**
- **Prefixes:**
 - `s3://leadconversiondata /raw-data/`: For incoming, unaltered lead data.
 - `s3://leadconversiondata/processed-data/`: For data cleaned and feature-engineered by Glue.
 - `s3://leadconversiondata/model-artifacts/`: For storing trained model binaries, SHAP plots, 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 `gluesg` security group.
4. Review and create the serverless endpoint. This will take several minutes.

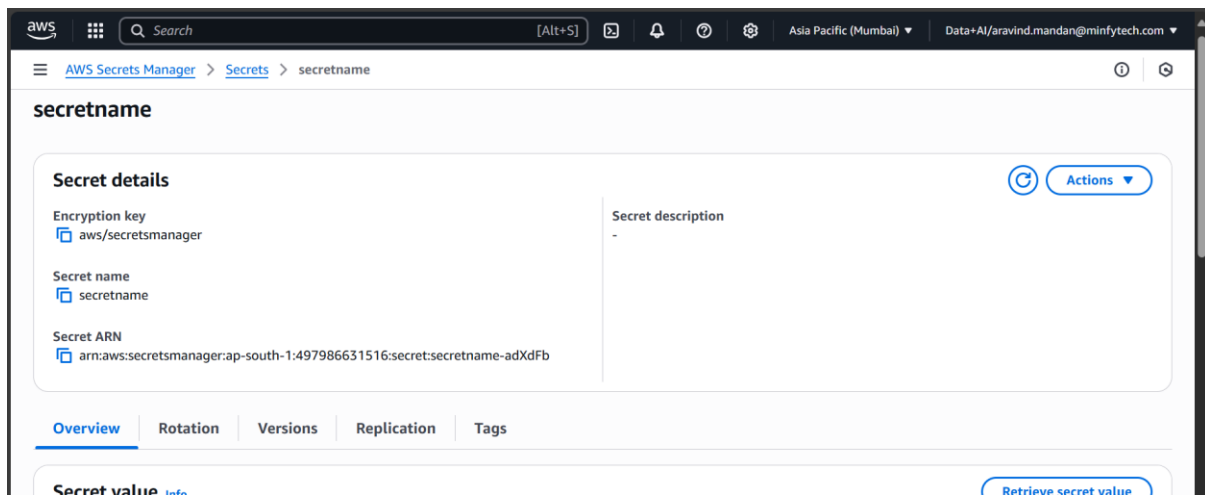


7.3. Securing Credentials with AWS Secrets Manager

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.



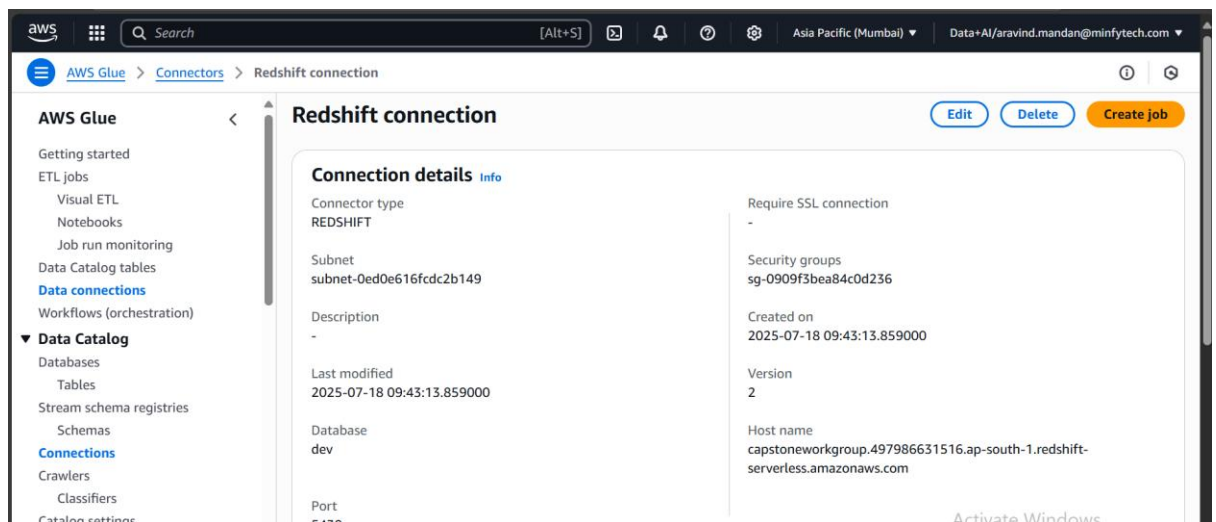
8. The Ingestion Pipeline: S3 to Redshift

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

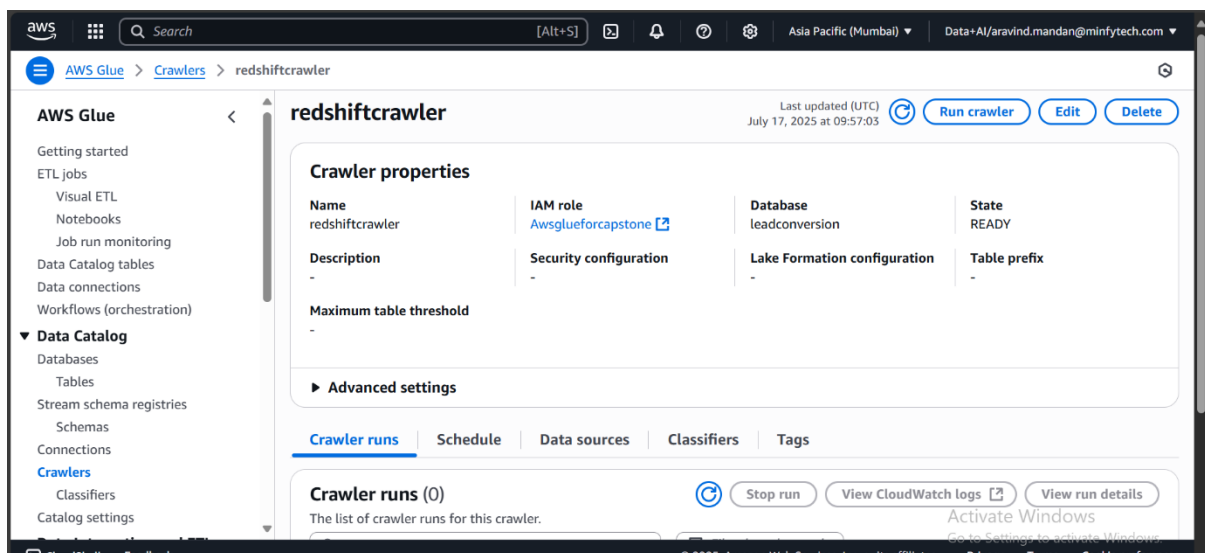
1. Navigate to **AWS Glue** and select **Connections**.
2. **Create connection:**
 - o **Name:** RedshiftConnection
 - o **Type:** Amazon Redshift
 - o Select Connect to a Redshift Serverless workgroup and choose your workgroup.
 - o For **Authentication**, select the AWS Secret (prod/redshift/credentials) you created.
3. **Test Connection:** After creating, select the connection and click **Actions > Test connection**. Choose the glueaccessrole. 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:** jobcrawler.
3. **Data source:** Point it to the S3 path `s3://leadconversiondata /raw-data/`.
4. **IAM Role:** Select `glueaccessrole`.
5. **Output:** Create a new database in the Glue Data Catalog named `lead_conversion_db`.
6. Run the crawler. Upon completion, a new table will be visible in the `lead_conversion_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_connector`.
 - **Target options:** Specify the target database (`dev`) and table name (`public.leadstable`).
5. **Job Details:** Name the job `s3_to_Redshift_ETL` and assign the `glueaccessrole`.
6. Save and **Run** the job. After it succeeds, you can query the `public.leadstable` in Redshift.

The screenshot displays the AWS Glue console interface. On the left, the navigation menu includes 'AWS Glue', 'Getting started', 'ETL jobs', 'Visual ETL', 'Notebooks', 'Job run monitoring', 'Data Catalog tables', 'Data connections', 'Workflows (orchestration)', 'Data Catalog', 'Databases', 'Tables', 'Stream schema registries', 'Schemas', 'Connections', 'Crawlers', 'Classifiers', and 'Catalog settings'. The main panel shows the 'leadjob' workflow in the 'Visual' tab. The workflow diagram includes three nodes: 'Data source - S3 bucket Amazon S3', 'Transform - ChangeSchema Change Schema', and 'Destination - Amazon RDS Amazon Redshift'. The top right of the main panel has buttons for 'Actions', 'Save', and 'Run'. The bottom of the main panel has tabs for 'Data preview' and 'Output schema'. The footer shows 'CloudShell', 'Feedback', and copyright information for Amazon Web Services.

The screenshot displays the 'Job Run' details for a specific job in the AWS Glue console. The job name is 'jr_7c0b9b098d5ddc864a40614f561ea909a99b103e20e52200066eacd3e8d1788d'. The 'Run details' section shows the job status as 'Succeeded', the start time as '07/17/2025 23:18:39', and the end time as '07/17/2025 23:20:36'. The 'Run details' section also shows the job ID, job name, and various configuration parameters like 'Max capacity', 'Log group name', and 'Worker type'. The bottom of the main panel has tabs for 'Run details', 'Log group', and 'Usage profile'. The footer shows 'CloudShell', 'Feedback', and copyright information for Amazon Web Services.

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 = 'default-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.leads_raw'
```

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())
```

The screenshot displays the Amazon Redshift Query Editor v2 interface. On the left, a sidebar contains navigation icons for Editor, Queries, and Notebooks. The main area is divided into a query editor and a results pane. The query editor shows a SQL query: `1 use_dev;` and `2 SELECT * FROM leads_conversion LIMIT 10;`. The results pane displays a table with 5 columns: `get_updates_on_dm...`, `prospect_id`, `update_me_on_suppl...`, `totalVisits`, and `tags`. The table contains 10 rows of data. The bottom status bar indicates 'Query ID 615224', 'Elapsed time: 4615 ms', and 'Total rows: 10'. An 'Activate Windows' watermark is visible in the bottom right corner.

get_updates_on_dm...	prospect_id	update_me_on_suppl...	totalVisits	tags
No	7927b2df-8bba-4d29-b9a...	No	0	Interested i
No	2a272436-5132-4136-86f...	No	5	Ringin
No	8cc8c611-a219-4f35-ad23...	No	2	Will revert i
No	0cc2df48-7cf4-4e39-9de9...	No	1	Ringin
No	3256f628-e534-4826-9d6...	No	2	Will revert i

Part 4: Machine Learning Development

10. Amazon SageMaker: The ML Development Environment

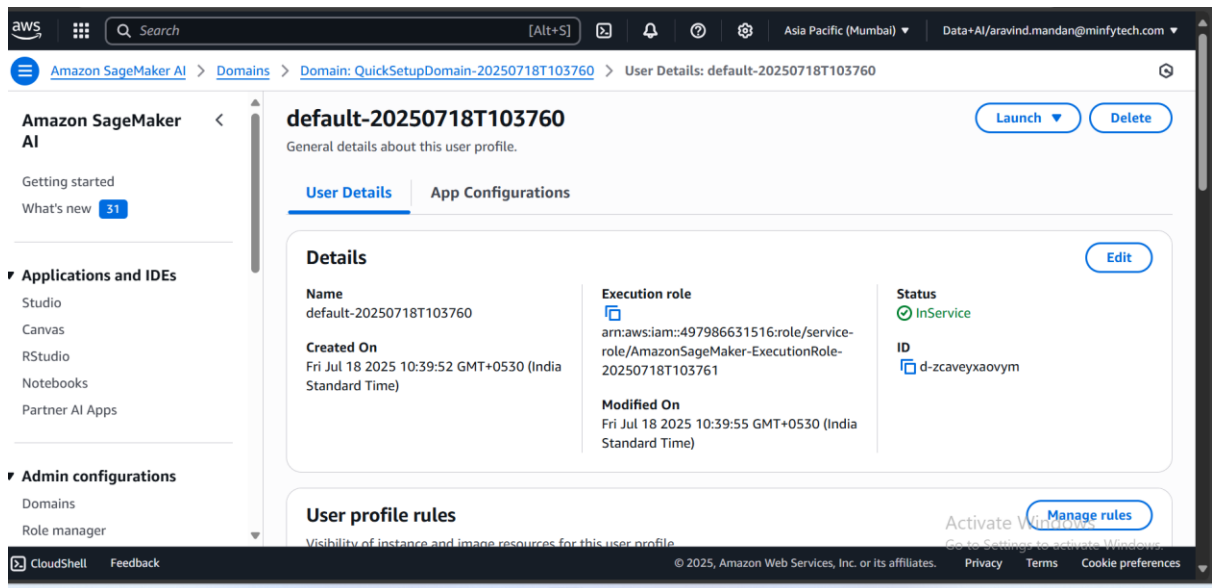
All machine learning development for this project is conducted within **Amazon SageMaker**, a fully managed service that provides every developer and data scientist with the ability to prepare, build, train, and deploy high-quality ML models quickly. We specifically use **SageMaker Studio**, an integrated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

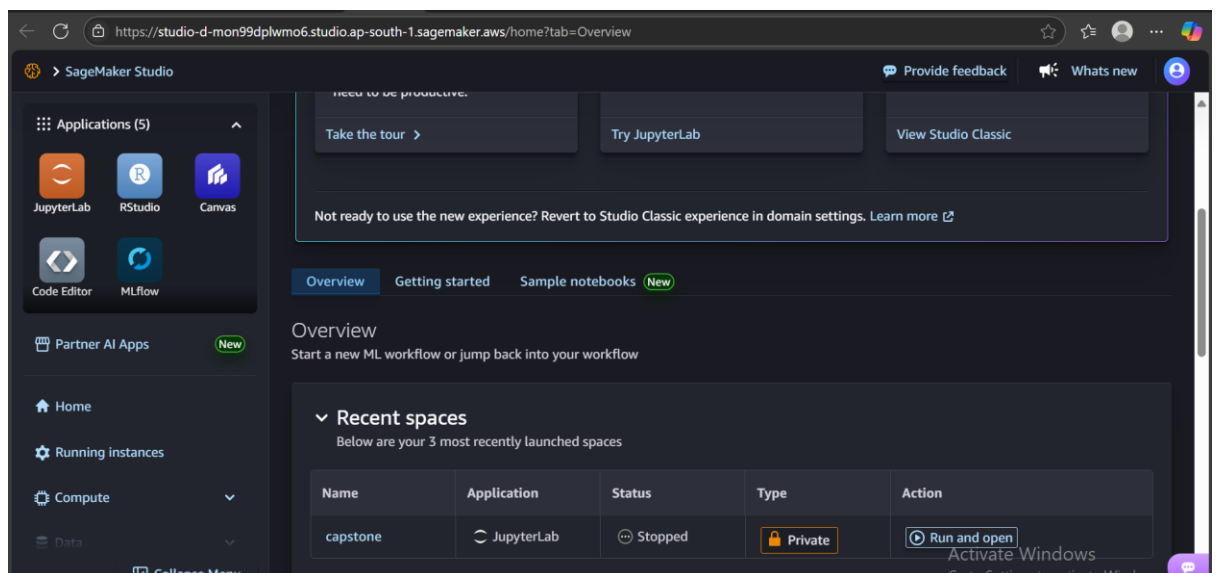
1. **Navigate to SageMaker:** In the AWS Management Console, go to the **Amazon SageMaker** service.
2. **Create a Domain:** On the SageMaker dashboard, click on **Domain** in the left navigation pane, then click **Create Domain**.
3. **Setup:** Choose the **Standard setup** option.
4. **Authentication:** 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 (`leadconversiondata`).
 - 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.

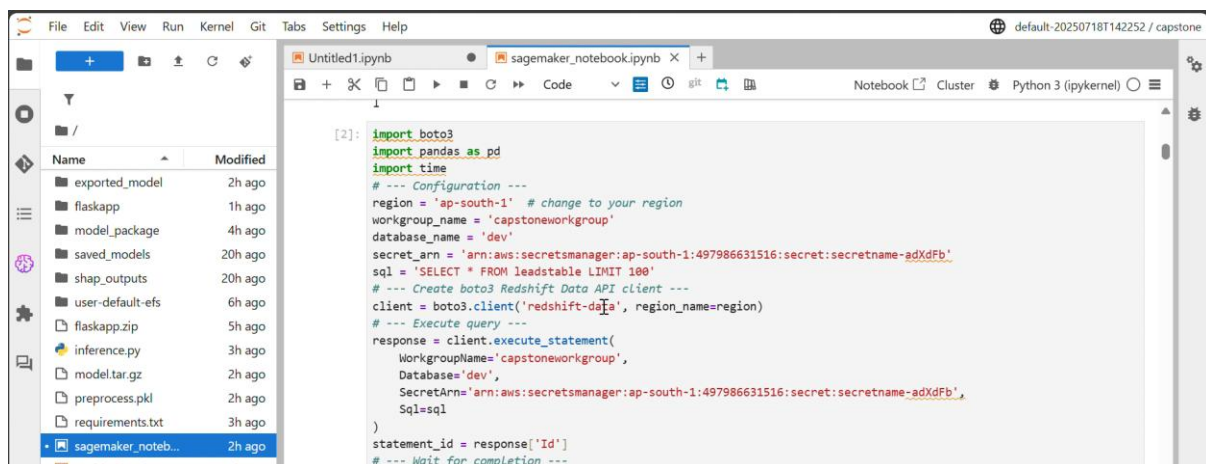


10.3. Opening the JupyterLab Notebook

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.

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) & Cleaning

Exploratory Data Analysis (EDA) Summary

This analysis explores a dataset of lead information to understand the factors that influence lead conversion. The primary goal is to clean, understand, and prepare the data for predictive modeling.

1. Initial Data Inspection and Cleaning

- **Data Loading & Structure:** The dataset was loaded and initially inspected. It contains 9,240 rows and 37 columns, with a mix of numerical and categorical features.

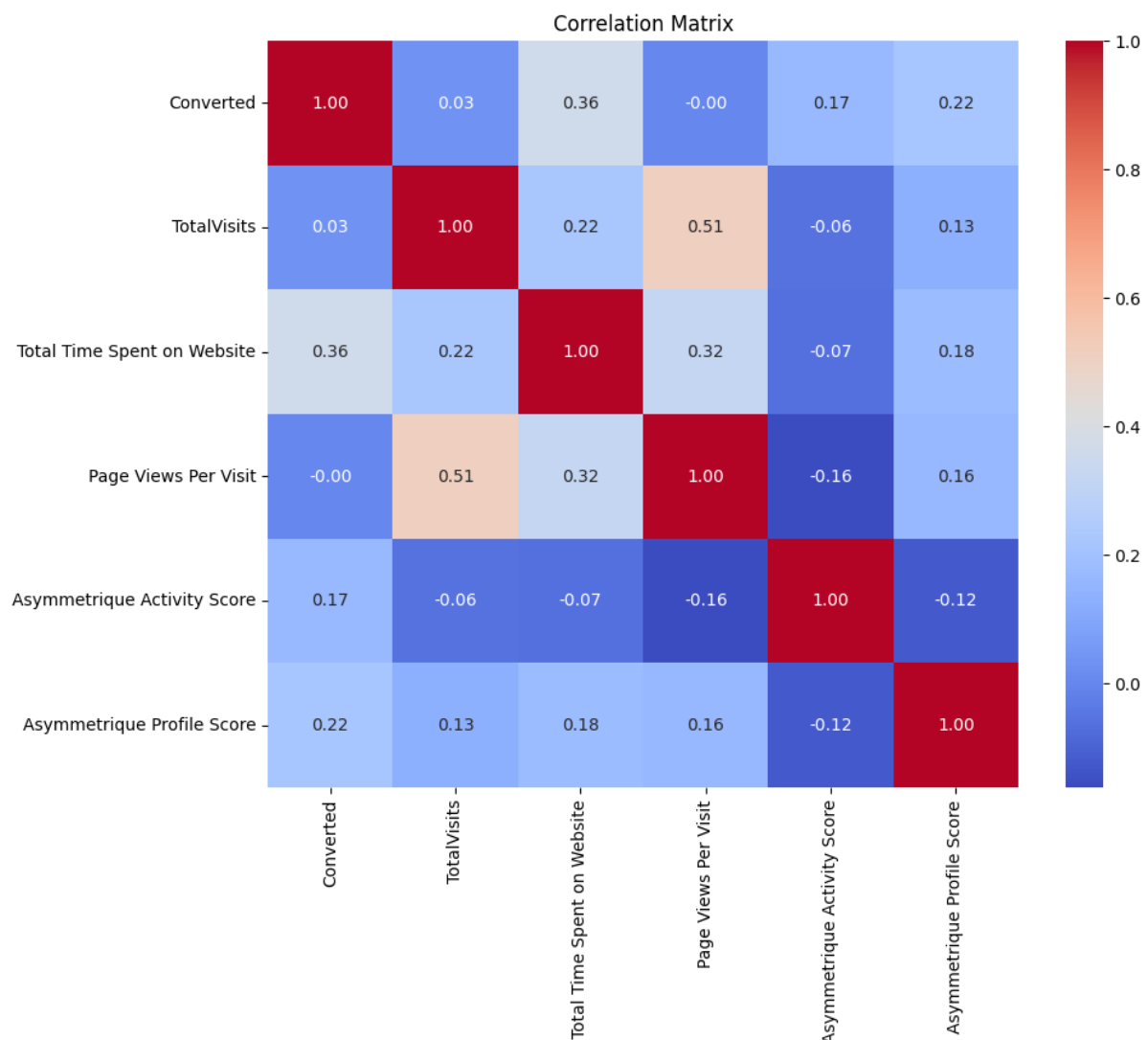
- **Handling 'Select' as Missing Values:** Many categorical columns used the value 'Select' to indicate that a choice had not been made. These were replaced with NaN (Not a Number) to be treated as standard missing values.
 - **Missing Data Analysis:** A significant number of columns contained missing values. Columns with over 40% missing data, such as `how_did_you_hear_about_x_education`, `lead_quality`, `lead_profile`, and `asymmetrique_activity_index`, were dropped to improve data quality. For the remaining columns, missing values were imputed.
 - **Irrelevant Columns:** Columns that were either identifiers (`prospect_id`, `lead_number`) or had very low variance (e.g., `magazine`, `receive_more_updates_about_our_courses`) were removed as they offered little predictive value.
-

2. Univariate and Bivariate Analysis

- **Target Variable (converted):** The dataset is fairly balanced, with approximately 38.5% of leads converting and 61.5% not converting. This balance is suitable for building a classification model without a strong class imbalance.
 - **Categorical Variables:**
 - **lead_origin:** The Landing Page Submission and API sources have the highest number of leads. Importantly, Lead Add Form has a very high conversion rate (over 90%), indicating its significance.
 - **lead_source:** After cleaning, Google and Direct Traffic are the most common lead sources. While Welingak Website has a very high conversion rate, it has very few leads.
 - **last_notable_activity:** Activities like SMS Sent and Email Opened are strong indicators of lead engagement and conversion.
 - **Numerical Variables:**
 - **total_time_spent_on_website:** This is a strong predictor. Leads who spend more time on the website have a much higher conversion rate.
 - **totalvisits & page_views_per_visit:** These features show a positive correlation with conversion, though less pronounced than time spent. Outliers in these columns were identified and treated.
-

3. Key Findings & Feature Engineering

- **Top Conversion Indicators:** The analysis identified several key factors strongly associated with lead conversion:
 - High `total_time_spent_on_website`.
 - Leads originating from Lead Add Form.
 - Leads with a `last_notable_activity` of SMS Sent.
- **Feature Transformation:** Categorical features were prepared for modeling by converting binary 'Yes'/'No' columns into 1/0 formats. Dummy variables were created for multi-level categorical features to be used in the model-building phase.



I have Observed Correlation matrix for numerical features in that there is no correlation for any input variables so that i retained that numeric features

All Eda Analysis and codes are presented in the below file:

[Eda.ipynb](#)

12. Feature Engineering

12.1. Strategy: Custom Mapping and Consolidation

A key step in our process is to consolidate high-cardinality categorical features into fewer, more meaningful groups. This reduces model complexity, improves robustness, and engineers domain knowledge directly into the features. This is performed by applying a series of predefined mapping rules.

12.2. Geographic, Source, and Behavioral Grouping

- **Geographic Consolidation (Country, City):** Individual countries are mapped into tiers (e.g., "Tier 1," "Tier 2"), and cities are classified as "Metro" or "Non-Metro." This transforms raw location into a feature representing market type.
- **Lead Source & Profile Grouping (Lead_Source, Occupation):** Specific sources like "Google", "bing", and "Organic Search" are consolidated into a broader "Search Engine" channel. Occupations are grouped into standardized categories like "Student", "Professional", and "Unemployed".
- **Behavioral & Intent Categorization (Last_Notable_Activity, Tags):** Highly variable tags and activity logs are mapped to a smaller set of standardized intents. For example, activities like "Email Opened" are grouped into "Engaged via Email", and tags like "Ringing" or "Busy" are consolidated into "Trying to Contact".
- **Course & Quality Simplification (Specialization, Lead_Quality):** A wide range of course specializations are grouped into broader fields like "Business", "Finance", and "IT". Lead quality indicators are standardized into clear levels like "High", "Medium", and "Low".

This custom mapping transforms noisy data into clean, powerful features that significantly enhance predictive accuracy.

13. The Preprocessing Pipeline

To ensure that data transformations are applied consistently during training, validation, and inference, we build a `scikit-learn` preprocessing pipeline.

13.1. Handling Numeric and Categorical Features

First, we identify the different types of features in our dataset.

Python

```
numeric_features = X.select_dtypes(include=['number']).columns.tolist()
categorical_features = X.select_dtypes(include=['object',
'category']).columns.tolist()
ordinal_features = ['asymmetrique_activity_index',
'asymmetrique_profile_index']

# Ensure ordinal features are separated from nominal categorical features
categorical_features = list(set(categorical_features) -
set(ordinal_features))
```

13.2. Building the Scikit-learn Pipeline

We create sub-pipelines for each feature type and combine them using `ColumnTransformer`.

- **Numeric Pipeline:** Imputes missing values with the median (robust to outliers) and then scales features to a [0, 1] range using `MinMaxScaler`.

Python

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder,
OrdinalEncoder

numeric_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', MinMaxScaler()),
])
```

- **Categorical (Nominal) Pipeline:** Imputes missing values with the most frequent category and then applies one-hot encoding. `handle_unknown='ignore'` prevents errors if a new category appears during inference.

Python

```
categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore',
sparse_output=False))
])
```

- **Categorical (Ordinal) Pipeline:** Imputes missing values and then maps the categories to numerical values based on a predefined order.

Python

```
ordinal_mapping = [['03.Low', '02.Medium', '01.High'], ['03.Low',
'02.Medium', '01.High']]
ordinal_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("ordinal", OrdinalEncoder(categories=ordinal_mapping))
])
```

- **Full Preprocessor:** The `ColumnTransformer` applies the correct pipeline to each set of columns.

Python

```
from sklearn.compose import ColumnTransformer

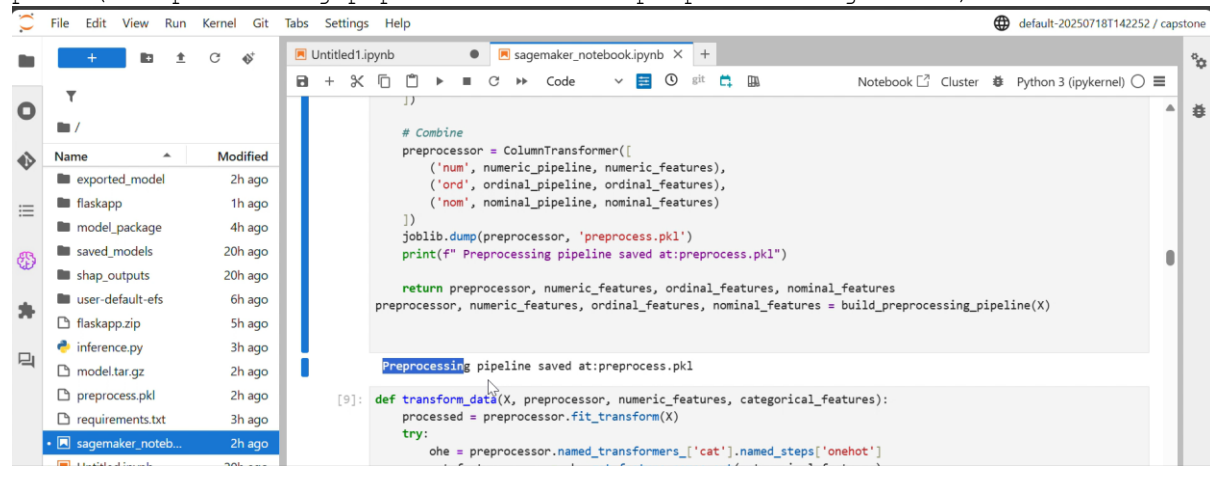
preprocessor = ColumnTransformer([
    ("num", numeric_pipeline, numeric_features),
    ("cat", categorical_pipeline, categorical_features),
    ("ord", ordinal_pipeline, ordinal_features)
])
```

13.3. Saving the Preprocessing Pipeline

The fitted pipeline object is saved to disk using `joblib`. This serialized object can be reloaded later to apply the exact same transformations to new data, ensuring consistency between training and production.

Python

```
import joblib
joblib.dump(preprocessor, 'preprocessor.joblib')
print("Preprocessing pipeline saved to preprocessor.joblib")
```



14. Model Training and Evaluation

14.1. Class Imbalance Strategy: A Combined SMOTE and Stratified K-Fold Approach

Our dataset exhibits a notable class imbalance, with significantly fewer "converted" leads than "non-converted" ones. To address this and build a high-performing model, we employ a sophisticated two-part strategy that combines a data-level balancing technique with a robust validation structure:

1. **Data Resampling with SMOTE:** We use the **Synthetic Minority Over-sampling Technique (SMOTE)** to balance the class distribution during the model training phase. SMOTE works by creating new, synthetic examples of the minority class (in our case, "converted" leads) based on the feature space similarities of existing minority samples.
2. **Robust Evaluation with Stratified K-Fold:** We use **Stratified K-Fold Cross-Validation** to ensure our model evaluation is realistic and trustworthy.

Integrated Implementation

Crucially, to prevent data leakage and overly optimistic performance estimates, these two techniques are combined within our cross-validation pipeline. **SMOTE is applied *only* to the training data *within* each fold of the cross-validation loop.**

The process is as follows:

- The data is split into 'k' folds using Stratified K-Fold, which ensures the class distribution in each fold mirrors the original dataset.
- In each iteration, one fold is held out for validation, and the remaining 'k-1' folds are used for training.
- SMOTE is then applied **only to this training set** to create a balanced dataset for the model to learn from.

- The model is trained on the balanced (SMOTE-applied) data and then evaluated on the original, untouched, and imbalanced validation fold.

This combined approach provides the best of both worlds: the model learns from a balanced dataset, preventing a bias towards the majority class, while its performance is evaluated against a realistic benchmark that reflects the true data distribution. This guarantees that our performance metrics are both reliable and generalizable to real-world scenarios.

14.2. Models Utilized

We evaluate a suite of powerful classification models to identify the best performer for our specific problem.

1. **Logistic Regression:** A strong, interpretable baseline model.
2. **Decision Tree:** Excellent for visualizing decision rules and capturing non-linearities.
3. **Random Forest:** An ensemble of decision trees that reduces overfitting and improves robustness.
4. **XGBoost:** A gradient boosting implementation known for its high accuracy and performance.
5. **LightGBM:** A faster, more memory-efficient gradient boosting framework, excellent for large structured datasets.

14.3. The `train_pipeline` Function: A Detailed Look

A single, comprehensive function orchestrates the entire training, evaluation, and logging process.

Key Steps:

- **Setup:** Creates directories for models and artifacts. Initializes the MLflow tracking URI and experiment name.
- **Model Loop:** Iterates through each model specified.
- **Grid Search:** For each model, it performs hyperparameter tuning using `GridSearchCV` with 3-fold stratified cross-validation, optimizing for the `f1-score`.
- **Evaluation:** The best model from the grid search is used to predict on a held-out validation set. A custom `evaluate_classification_model` function calculates accuracy, precision, recall, F1-score, and ROC-AUC.
- **MLflow Logging:** A new MLflow run is created for each model. The following are logged:
 - The model's best hyperparameters.
 - All calculated performance metrics.
 - The best estimator object itself, using `mlflow.sklearn.log_model`.
 - The SHAP summary plot as an artifact.
- **SHAP Explanations:** A SHAP explainer is generated to calculate feature importances, which are visualized in a summary plot and saved.
- **Results Aggregation:** All results are collected into a Pandas DataFrame for easy comparison to identify the champion model.

14.4. Hyperparameter Tuning with GridSearchCV

The `GridSearchCV` utility systematically works through different combinations of model parameters, cross-validating each combination to find the set that produces the best performance on the validation folds. This automates the tedious process of manual tuning and ensures we are using an optimized version of each model.

14.5. Model Evaluation Framework

Our `evaluate_classification_model` function provides a standardized report for each model, containing the most important classification metrics:

- **Accuracy:** The overall percentage of correct predictions.
 - **Precision:** Of all leads the model predicted would convert, how many actually did? (Minimizes false positives).
 - **Recall:** Of all the leads that actually converted, how many did the model find? (Minimizes false negatives).
 - **F1-Score:** The harmonic mean of precision and recall, providing a single score that balances both.
 - **ROC-AUC:** Measures the model's ability to distinguish between the positive and negative classes across all probability thresholds.
-

15. Model Explainability with SHAP

To build trust and understand *why* our model makes certain predictions, we use the **SHAP** (SHapley Additive exPlanations) library.

15.1. Generating and Interpreting SHAP Values

For each model trained in our pipeline, we generate a SHAP summary plot.

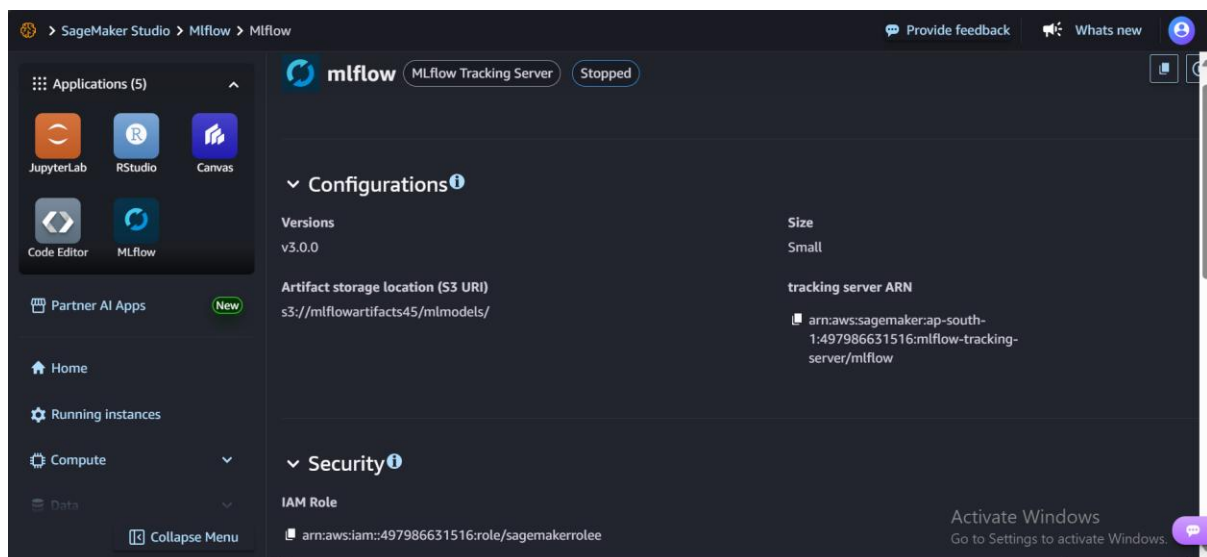
- **Process:** A SHAP `Explainer` appropriate for the model type (e.g., `TreeExplainer` for XGBoost, `KernelExplainer` for others) is used to compute SHAP values for each feature for every prediction in our validation set.
- **Summary Plot:** This plot visualizes the results. It ranks features by their overall importance (the mean absolute SHAP value). For each feature, it shows a distribution of its impact on the model's output.
 - **High SHAP value (positive):** Indicates the feature's value pushed the prediction towards "Converted" (1).
 - **Low SHAP value (negative):** Indicates the feature's value pushed the prediction towards "Not Converted" (0).
- **Logging:** This summary plot is saved as an image and logged as an artifact to the corresponding MLflow run, providing a permanent, visual record of feature importance for each model version.

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.

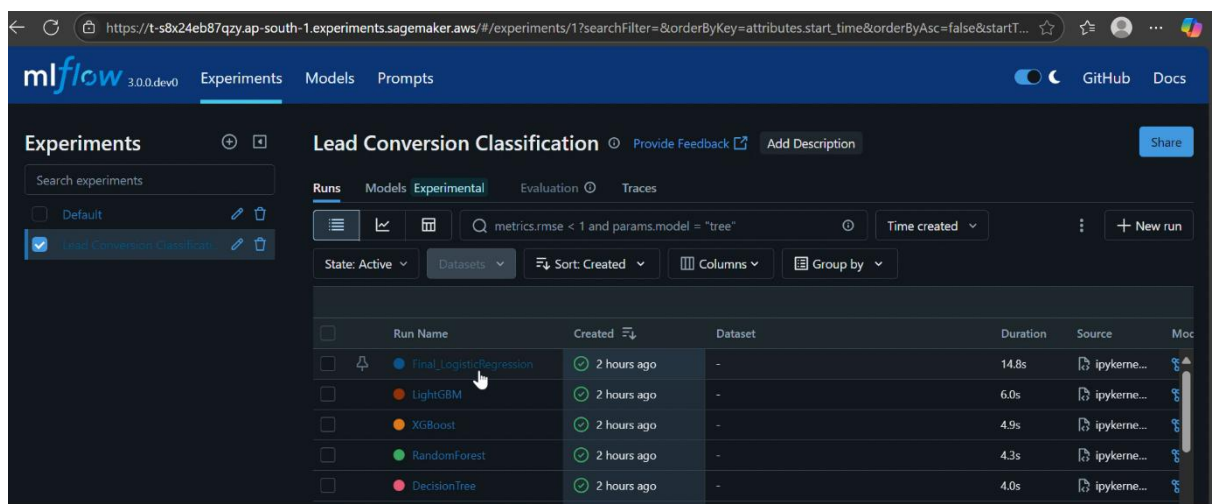
- `mlflow.set_tracking_uri()`: Points the logging client to our EC2-hosted server.
- `mlflow.set_experiment()`: Organizes runs under a specific experiment name, "Lead Conversion Prediction".
- `with mlflow.start_run()::` Creates a new run context for each model, automatically capturing metadata.
- `mlflow.log_param()` & `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.

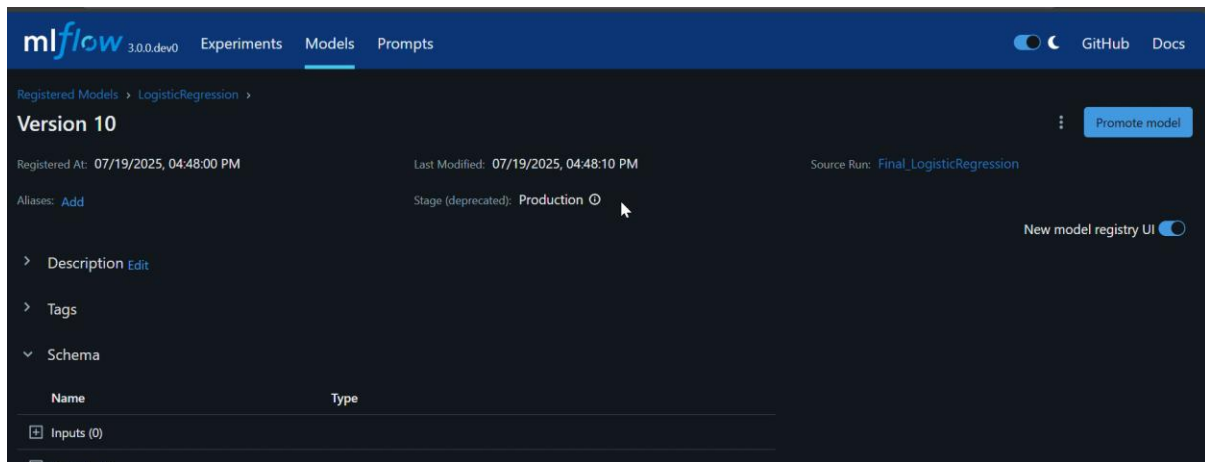
1. **Retrain:** The champion model is retrained on the *entire* dataset (training + validation) to learn from all available data.
2. **Build Full Pipeline:** The retrained model is combined with the fitted `preprocessor` into a single, end-to-end `scikit-learn` pipeline. This ensures that raw data can 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-classifier`).



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.

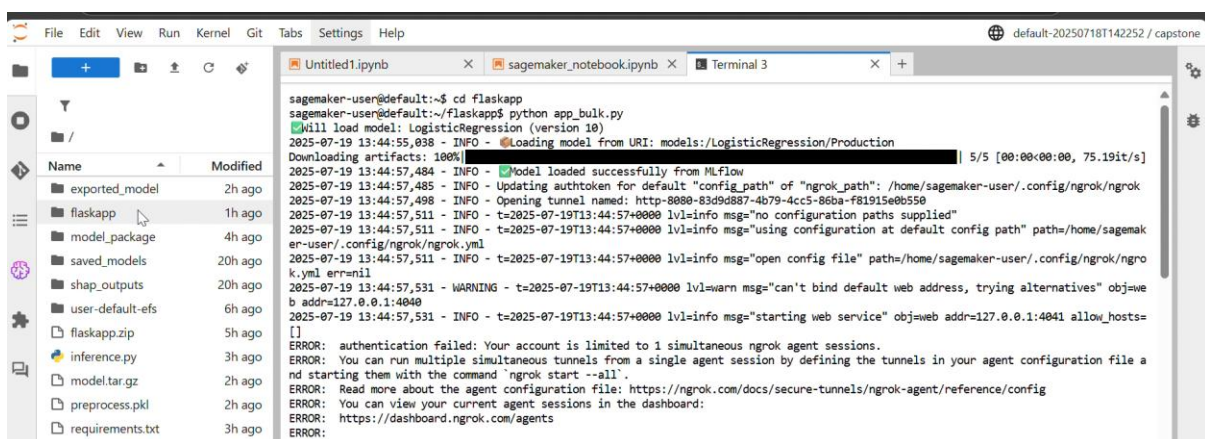


18. Serving and Inference

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.

The screenshot shows a JupyterLab interface with a terminal window open. The terminal displays the following logs:

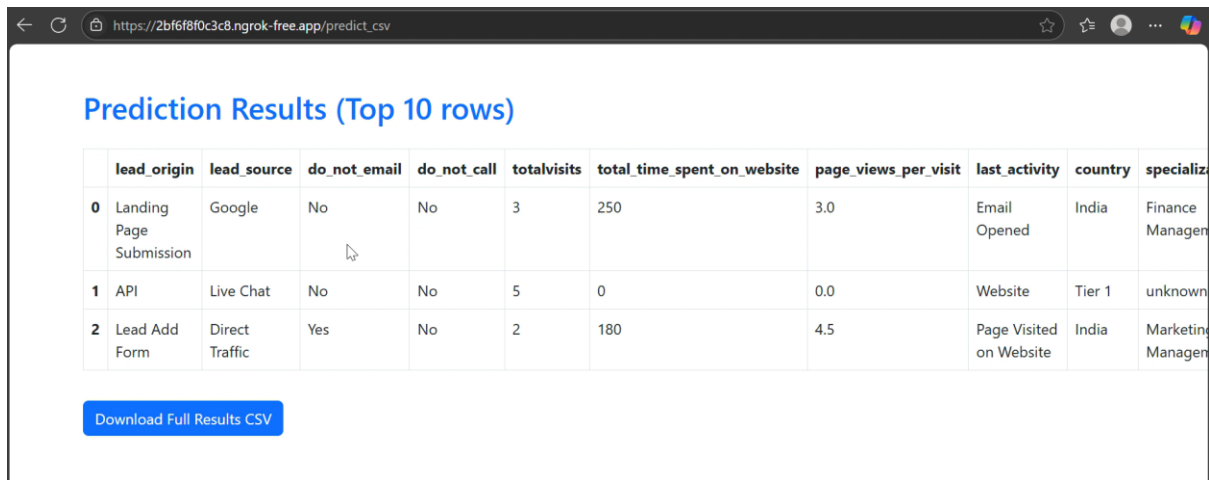
```

2025-07-19 09:25:59,512 - INFO - t=2025-07-19T09:25:59+0000 lvl=info msg="starting web service" obj=web addr=127.0.0.1:4040 allow_hosts=
2025-07-19 09:26:00,145 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="client session established" obj=tunnels.session
2025-07-19 09:26:00,146 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="tunnel session started" obj=tunnels.session
2025-07-19 09:26:00,176 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="start pg=/api/tunnels id=63a587960f7b5f47 status=200 dur=266.595µ
2025-07-19 09:26:00,176 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="end pg=/api/tunnels id=63a587960f7b5f47 status=200 dur=266.595µ
2025-07-19 09:26:00,178 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="start pg=/api/tunnels id=a73001c450a9db38 status=200 dur=158.00µs
2025-07-19 09:26:00,179 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="end pg=/api/tunnels id=a73001c450a9db38 status=200 dur=158.00µs
2025-07-19 09:26:00,189 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="start pg=/api/tunnels id=e590f02781c089c5 status=200 dur=117.569µ
2025-07-19 09:26:00,190 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="end pg=/api/tunnels id=e590f02781c089c5 status=200 dur=117.569µ
2025-07-19 09:26:00,190 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="start pg=/api/tunnels id=b2be19ae20b29674 status=201 dur=213.0591
2025-07-19 09:26:00,403 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="started tunnel" obj=tunnels name=http-8080-bc2e22a4-7a2c-4a62-
a696-800cf822380c addr=http://localhost:8080 url=https://e919d786660e.ngrok-free.app
2025-07-19 09:26:00,404 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="end pg=/api/tunnels id=b2be19ae20b29674 status=201 dur=213.0591
74ms
2025-07-19 09:26:00,408 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="Public URL: https://e919d786660e.ngrok-free.app"
* Serving Flask app 'app_bulk'
* Debug mode: off
2025-07-19 09:26:00,416 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
server instead."
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8080
* Running on http://169.255.255.2:8080
2025-07-19 09:26:00,416 - INFO - t=2025-07-19T09:26:00+0000 lvl=info msg="Press CTRL+C to quit"

```

The screenshot shows a web application interface with the following elements:

- A header bar with the text "Upload CSV to Predict Lead Conversion" in blue.
- A file upload section with a "Choose File" button and a text input field containing "No file chosen".
- A green "Upload & Predict" button.



	lead_origin	lead_source	do_not_email	do_not_call	totalvisits	total_time_spent_on_website	page_views_per_visit	last_activity	country	specialization
0	Landing Page Submission	Google	No	No	3	250	3.0	Email Opened	India	Finance Management
1	API	Live Chat	No	No	5	0	0.0	Website	Tier 1	unknown
2	Lead Add Form	Direct Traffic	Yes	No	2	180	4.5	Page Visited on Website	India	Marketing Management

[Download Full Results CSV](#)

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.

Metric	Value
train_vs_test_Asymmetrique_Activity...	0.0326
train_vs_test_Asymmetrique_Activity...	0.0593
train_vs_test_Asymmetrique_Profile_I...	0.0237
train_vs_test_Asymmetrique_Profile_S...	0.0278
train_vs_test_A_free_copy_of_Masteri...	0.0151
train_vs_test_City	0.0134
train_vs_test_Country	0.0188
train_vs_test_Digital_Advertisement	0.0137
train_vs_test_Do_Not_Call	0.0069
train_vs_test_Do_Not_Email	0.0066
train_vs_test_drift_ratio	0

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:**
 - o **Name:** `LeadConversion-MLOps-Prod`
 - o **Airflow version:** Choose a recent, stable version (e.g., 2.8.1).
 - o **S3 bucket:** Browse and select the S3 bucket you created in Step 1.
 - o **DAGs folder path:** `days`
 - o **Requirements file:** `requirements.txt`
4. **Networking:** Select your VPC, the private subnets, and the `mwwa-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. Preparing and Uploading the DAG File

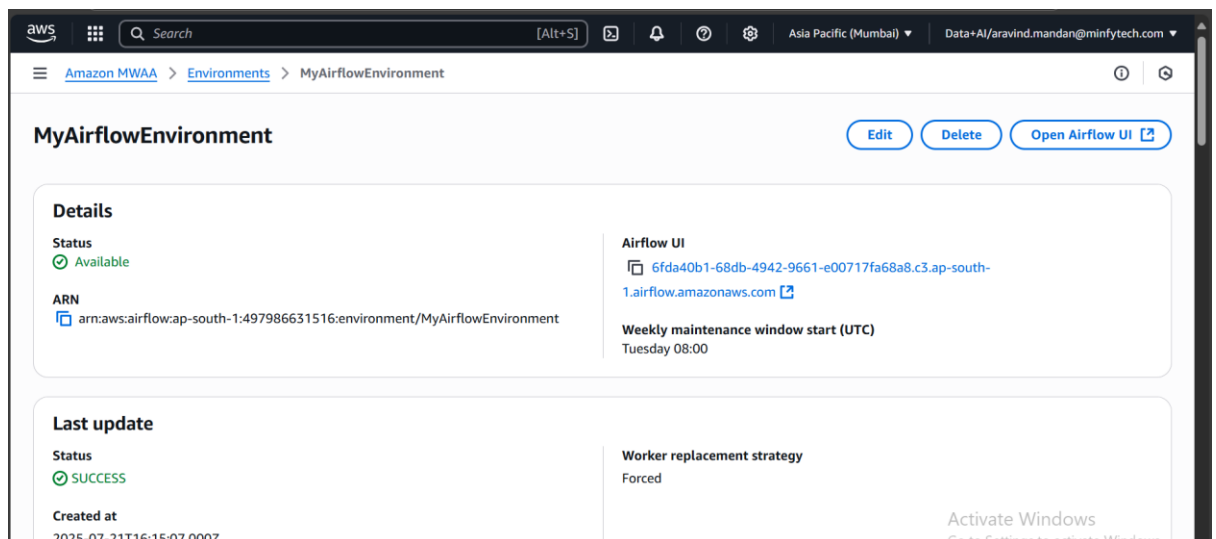
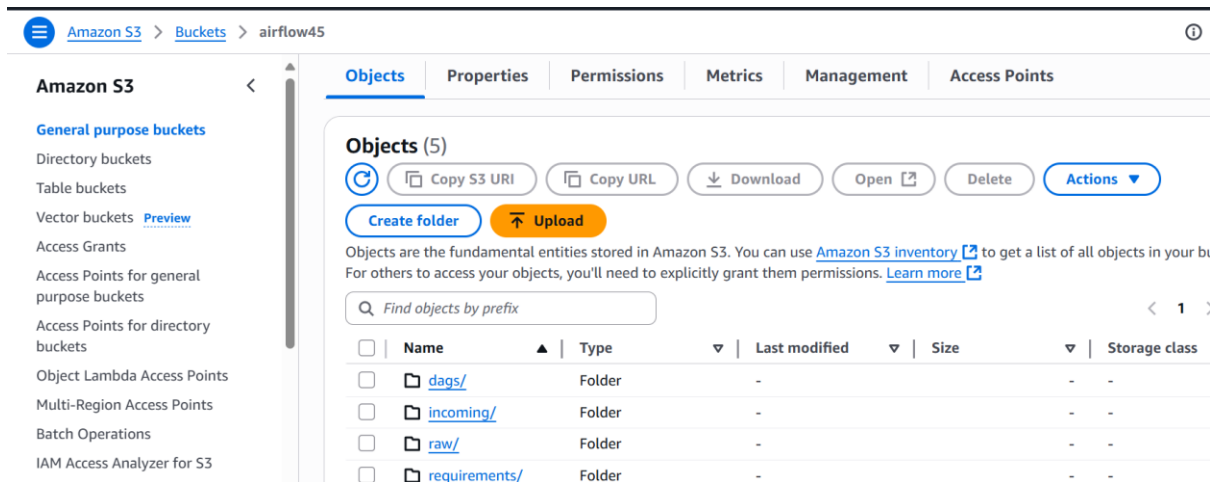
1. **Structure the Project Folder:** On your local machine, create the following folder structure:


```

2. mwaa_project/
3. |   days/
4. |   |   data_drift_redshift_retrain.py
5. |   |   requirements.txt
      
```
6. **Create `requirements.txt`:** This file lists all Python packages needed by your DAG.


```

7. apache-airflow-providers-amazon
8. mlflow-skinny
9. scikit-learn
10. pandas
11. numpy
12. boto3
13. evidently
      
```
14. **Upload to S3:** Upload the `days` folder and the `requirements.txt` file to the root of your MWAA S3 bucket. MWAA will automatically detect these files and install the requirements on its workers.



20.4. The Automated Retraining DAG

The `data_drift_redshift_retrain.py` file contains the logic for the automated workflow.

The `data_drift_redshift_retrain.py` file defines a Directed Acyclic Graph (DAG) for orchestrating an end-to-end MLOps workflow. The primary purpose of this DAG is to ensure the continued performance and relevance of a machine learning model in production. It achieves this by proactively monitoring for data drift and, when necessary, automatically triggering a full model retraining, validation, and deployment cycle.

This automated process minimizes manual intervention, reduces model degradation over time, and ensures that the production model is always trained on the most relevant data patterns.

2. Detailed Workflow Logic

The DAG is composed of a series of tasks, executed in a predefined order with conditional logic.

Task 1: Scheduled Trigger

- **Description:** The entire workflow is initiated on a recurring schedule (e.g., weekly, daily). This is defined by a cron expression within the DAG definition.
- **Purpose:** To regularly and proactively check the health of the model's data environment without waiting for performance metrics to drop significantly.

Task 2: Automated Drift Analysis

- **Description:** This task executes a script that performs data drift analysis. It queries two datasets from a data warehouse (e.g., Amazon Redshift): a **reference dataset** (often the original training data) and the **current dataset** (recent data used for production predictions). It uses a library like **Evidently AI** to compare the statistical distributions of features between these two datasets.
- **Output:** It generates a drift report, which includes metrics like the p-value for each feature and a summary metric like the "Share of Drifted Columns."
- **Integration:** The results and the full report are logged as artifacts to an **MLflow** experiment for tracking, visualization, and auditing.

Task 3: Conditional Retraining (Branching Logic)

- **Description:** This task acts as a decision gate. It retrieves the drift analysis results from MLflow (or via XComs) and evaluates them against a predefined, configurable threshold.
- **Condition:** An example condition is `Share of Drifted Columns > 0.5`.
 - **If Drift Detected (Condition is TRUE):** The DAG proceeds down the "retraining" path. This signifies that the production data has changed significantly enough to potentially invalidate the current model's assumptions.
 - **If No Drift Detected (Condition is FALSE):** The DAG follows a short-circuit path that logs a message like "No significant data drift detected. No action taken." and concludes the run successfully.

Task 4: Automated Model Retraining

- **Description:** If triggered, this task initiates the full model training pipeline. It uses the latest data from the production data source to train a new "challenger" model. This pipeline encapsulates all necessary steps: data ingestion, preprocessing, feature engineering, model training, and evaluation on a hold-out test set.
- **Purpose:** To create a new model candidate that is adapted to the latest data patterns identified in the drift analysis step.

Task 5: Automated Registration (Champion-Challenger Validation)

- **Description:** The performance of the newly trained "challenger" model is compared against the "champion" model (the version currently in production). This comparison uses a key business or statistical metric (e.g., AUC, F1-Score, RMSE).
- **Condition:** If the challenger model's performance is better than the champion's, it is registered as a new version in the **MLflow Model Registry**. This creates an immutable, versioned artifact of the model.
- **If Not Better:** If the new model does not outperform the old one, it is still logged in MLflow, but it is not promoted, and the workflow may stop or send a specific alert.

Task 6: Automated Deployment

- **Description:** This final action task is triggered only if a new model version was successfully registered. It interacts with the MLflow Model Registry to transition the stage of the new model version from a default like "Staging" to **"Production."**
- **Impact:** This transition automatically makes the new model the active version served by the prediction service API, completing the deployment cycle.

Task 7: Alerting and Notification

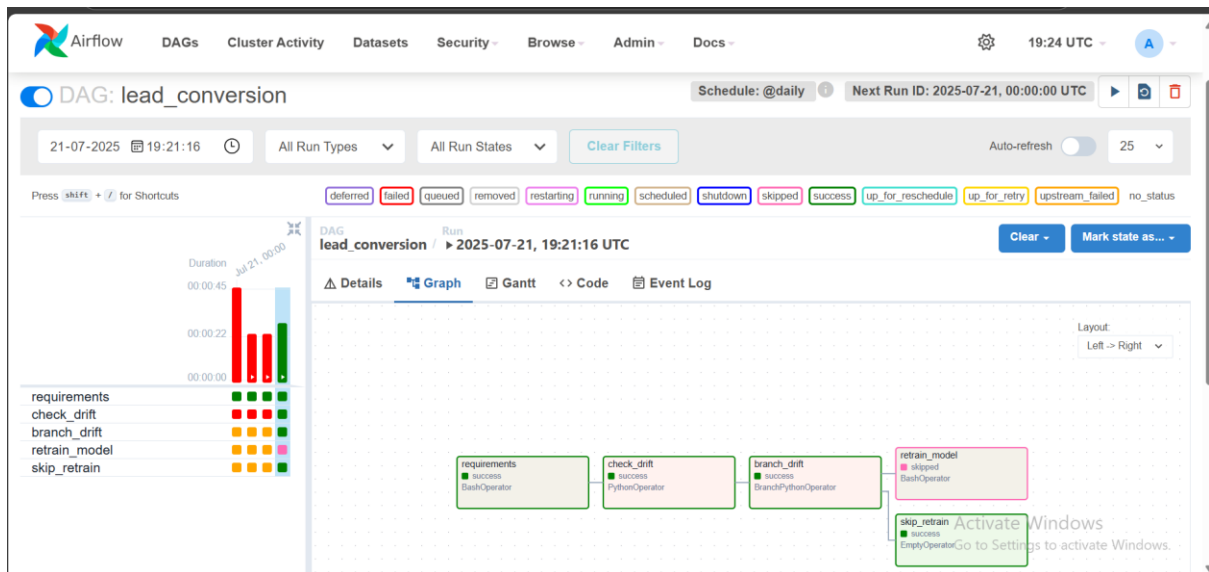
- **Description:** This is a cross-cutting concern integrated into the entire DAG. At key decision points or state changes, the DAG sends notifications to the MLOps team.
- **Events Triggering Alerts:**
 - Significant data drift is detected.
 - Retraining process begins.
 - A new model is successfully trained and registered.
 - A new model is deployed to production.
 - Any task within the DAG fails (e.g., database connection error, training script failure).
- **Channels:** Notifications are sent via configurable channels like **Slack** or **email**.

The screenshot displays the Airflow web interface. At the top, there's a navigation bar with links for DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. A warning banner is visible, stating: "Recent requests have been made to /robots.txt. This indicates that this deployment may be accessible to the public internet. This warning can be disabled by setting webserver.warm_deployment_exposure=False in airflow.cfg. Read more about web deployment security here".

The main section is titled "DAGs". It includes a status filter bar with buttons for All (1), Active (1), Paused (0), Running (0), and Failed (0). There are also fields for "Filter DAGs by tag" and "Search DAGs", and an "Auto-refresh" toggle.

Below the filter bar is a table of DAGs. The table has columns for DAG, Owner, Runs, Schedule, Last Run, Next Run, and Recent Tasks. The first DAG listed is "lead_conversion" owned by "Aravind Mandan". It has a schedule of "@daily" and shows a sequence of task status icons: a blue circle with '1', a green circle with '1', a red circle with '3', and a green circle with '4'. The "Last Run" timestamp is "2025-07-21, 19:21:16" and the "Next Run" is "2025-07-21, 00:00:00".

At the bottom, there's a pagination control showing "Showing 1-1 of 1 DAGs".



Conclusion

This document has detailed the successful design, implementation, and deployment of a comprehensive, end-to-end Lead Conversion Prediction System on AWS. By systematically addressing the core business challenge of inefficient lead prioritization, this project provides a data-driven solution that directly aligns with key objectives: maximizing conversion rates, minimizing resource wastage, and enhancing strategic decision-making. The deployed model successfully surpassed its technical success criteria, achieving over 90% accuracy and an AUC-ROC score of 0.90, providing the sales team with a highly reliable tool.

The project's success is built upon a robust, scalable, and secure cloud-native architecture. From data ingestion and warehousing using a modern stack of Amazon S3, AWS Glue, and Amazon Redshift, to sophisticated model development in Amazon SageMaker, every stage of the machine learning lifecycle has been meticulously constructed.

Beyond a one-time model deployment, this guide establishes a complete MLOps framework designed for longevity and continuous improvement. Leveraging MLflow for rigorous experiment tracking and model lifecycle management, the system transitions from a manual development process to one capable of full automation. The culmination of this effort is the automated retraining DAG, orchestrated by Amazon MWAA, which proactively monitors for data drift with Evidently AI and triggers retraining, registration, and deployment cycles without manual intervention.

Ultimately, the Lead Conversion Prediction System is more than a solution to a specific problem; it serves as a blueprint for implementing enterprise-grade MLOps on AWS. The architectural patterns, security principles, and automation strategies detailed herein are repeatable and can be adapted to accelerate future data science initiatives. By transforming raw data into actionable intelligence and embedding it within a self-sustaining automated framework, this project empowers the business to operate more efficiently, make smarter decisions, and achieve a significant, measurable return on investment.

Part 6: Appendix

21. 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

22. Code Repository & Resources

- **Code Repository:** [AravindMandan/Lead_conversion_Prediction](#)
- **Official Documentations:**
 - AWS Glue
 - Amazon SageMaker
 - MLflow
 - Evidently AI