Predicting Sales Conversion Likelihood

Table of contents:

Problem Statement:

This document outlines the development, architecture, and deployment of a machine learning model designed to predict sales lead conversion likelihood. The project addresses the business problem of inefficient lead prioritization in a competitive B2B sales environment, where significant resources are expended on leads that ultimately do not convert.

The solution is a data-driven predictive model that calculates the probability of a lead converting into a paying customer .

P(conversion)=f(lead_attributes)). By analyzing historical lead data and behavioral attributes , the model classifies leads into high, medium, and low potential categories, enabling the sales team to focus their efforts on the most promising prospects and improve resource allocation.

The end-to-end MLOps architecture ensures the entire lifecycle is automated and monitored. Data is ingested via an ETL pipeline (AWS Glue, S3, Redshift), models are developed in SageMaker and tracked with MLflow, the best model is deployed as an endpoint for a web application, and the system is monitored for data drift using Apache Airflow and Evidently, with automated triggers for retraining.

---

1. Business Problem & Objective

- Problem Statement: In the B2B sales sector, marketing and sales teams invest heavily in lead acquisition. However, a significant portion of these leads fail to convert, leading to wasted effort and a suboptimal return on investment (ROI). The organization currently lacks an intelligent system to prioritize leads based on their potential.
- Objective: The primary goal is to build and deploy a predictive machine learning model that estimates the probability of a sales lead converting. This data-driven

solution will help sales representatives identify and focus on high-potential leads, thereby optimizing outreach strategies and improving resource allocation

Business Success Criteria:

To measure the success of this ML application from a business perspective, we can define the following specific, measurable, achievable, relevant, and time-bound (SMART) criteria:

- Increase in Sales Conversion Rate:
  - "Achieve a 15% increase in the overall sales conversion rate within the first six months of deploying the model." This is the most direct measure of the model's impact on the primary business goal.
- Improvement in Sales Team Productivity:
  - "Increase the number of qualified leads handled by each sales representative by 20% per quarter." This demonstrates that the sales team is spending more time on valuable interactions.
  - "Reduce the average time spent on non-converting leads by 30% within the first quarter." This shows the model is successfully filtering out low-quality leads.
- Enhancement of Marketing ROI:
  - "Improve the return on investment (ROI) from marketing campaigns by 10% in the next fiscal year." This will be achieved by converting more leads from the same marketing spend.
- Reduction in Sales Cycle Length:
  - "Shorten the average sales cycle length by 5% in the first six months." By prioritizing hot leads, the time from initial contact to closing a deal should decrease.

ML Success Criteria:

Accuracy: The model must be correct in its predictions for at least 85% of all leads.

Precision: To prevent wasting sales time, over 80% of leads the model flags as "likely to convert" must actually be good leads.

Recall: To avoid losing opportunities, the model must successfully identify over 75% of all leads that eventually convert.

F1-Score: The model must have an F1-Score greater than 0.80 to show it is both efficient and effective.

Data Description :

Numerical Features

Feature	Description	Statistical Properties
Lead Number	A unique identifier for each lead.	This is a discrete numerical identifier and not used for statistical analysis.
TotalVisits	The total number of times a lead has visited the website.	Mean: 3.45, Median: 3.0, Standard Deviation: 4.85, Range: 0 to 251
Total Time Spent on Website	The total amount of time in seconds that a lead has spent on the website.	Mean: 487.7, Median: 248.0, Standard Deviation: 548.0, Range: 0 to 2272
Page Views Per Visit	The average number of pages viewed by the lead per visit.	Mean: 2.36, Median: 2.0, Standard Deviation: 2.16, Range: 0 to 55

Asymmetrique Activity Score	A score assigned by a third-party service based on the lead's activity.	Mean: 14.3, Median: 14.0, Standard Deviation: 1.39, Range: 7 to 18
Asymmetrique Profile Score	A score assigned by a third-party service based on the lead's profile.	Mean: 16.3, Median: 16.0, Standard Deviation: 1.8, Range: 11 to 20

Categorical Features

Feature	Description & Format (Categorical)
Lead Origin	The source from which the lead was generated. (e.g., 'API', 'Landing Page Submission')
Lead Source	The specific source of the lead. (e.g., 'Google', 'Organic Search')
Do Not Email	Indicates if the lead has opted out of email communication. (Categorical: 'Yes', 'No')
Do Not Call	Indicates if the lead has opted out of phone communication. (Categorical: 'Yes', 'No')

Converted	The target variable, indicating if the lead converted. (Binary: 1 for Yes, 0 for No)
Last Activity	The last action taken by the lead. (e.g., 'Email Opened', 'Page Visited on Website')
Country	The country of the lead. (e.g., 'India', 'United States')
Specialization	The lead's professional specialization. (e.g., 'Business Administration', 'Marketing Management')
How did you hear about X Education	How the lead heard about the company. (e.g., 'Online Search', 'Word of Mouth')
What is your current occupation	The lead's current employment status. (e.g., 'Unemployed', 'Working Professional')
What matters most to you in choosing a course	The lead's primary motivation for choosing a course. (e.g., 'Better Career Prospects')

Search, Magazine, Newspaper Article, X Education Forums, Newspaper, Digital Advertisement, Through Recommendations	Binary indicators of whether the lead was acquired through these channels. (Categorical: 'Yes', 'No')
Receive More Updates About Our Courses	Indicates if the lead wants to receive more updates. (Categorical: 'Yes', 'No')
Tags	Tags assigned to the lead by the sales team. (e.g., 'Interested in other courses', 'Ringing')
Lead Quality	A qualitative assessment of the lead's quality. (e.g., 'Low in Relevance', 'Might be')
Lead Profile	A profile category assigned to the lead. (e.g., 'Potential Lead', 'Select')
City	The city of the lead. (e.g., 'Mumbai', 'Thane & Outskirts')
Asymmetrique Activity Index	A categorical index based on the lead's activity. (e.g., '01.High', '02.Medium', '03.Low')

Asymmetrique Profile Index	A categorical index based on the lead's profile. (e.g., '01.High', '02.Medium', '03.Low')
I agree to pay the amount through cheque	Indicates if the lead agreed to pay by cheque. (Categorical: 'Yes', 'No')
A free copy of Mastering The Interview	Indicates if the lead requested a free copy of the book. (Categorical: 'Yes', 'No')
Last Notable Activity	The last significant action taken by the lead. (e.g., 'Modified', 'Email Opened')

	Prospect ID	Lead Number	Lead Origin	Lead Source	Do Not Email	Do Not Call	Converted	TotalVisits	Total Time Spent on Website	Page Views Per Visit	...	Get updates on DM Content	Lead Profile	Ci
0	7927b2df-8bba-4d29-b9a2-b6e0beafe620	660737	API	Olark Chat	No	No	0	0.0	0	0.0	...	No	Select	Sele
1	2a272436-5132-4136-86fa-dcc88c88f482	660728	API	Organic Search	No	No	0	5.0	674	2.5	...	No	Select	Sele
2	8cc8c611-a219-4f35-ad23-fdfd2656bd8a	660727	Landing Page Submission	Direct Traffic	No	No	1	2.0	1532	2.0	...	No	Potential Lead	Mumb
3	0cc2df48-7cf4-4e39-9de9-19797f9b38cc	660719	Landing Page Submission	Direct Traffic	No	No	0	1.0	305	1.0	...	No	Select	Mumb
4	3256f628-e534-4826-9d63-4a8b88782852	660681	Landing Page Submission	Google	No	No	1	2.0	1428	1.0	...	No	Select	Mumb

The analysis covers univariate, bivariate, and multivariate perspectives to uncover key insights, understand data quality, and identify features that are most influential in predicting whether a lead will be converted.

Key Findings:

- **Target Variable Imbalance:** The dataset is moderately imbalanced, with **38.5%** of leads marked as "Converted" (1) and **61.5%** as "Not Converted" (0). This suggests that techniques to handle imbalance may be necessary for building a robust model.
- **Strongest Predictors:** The most significant indicators of a lead converting are the **Total Time Spent on Website** and the **Last Activity**. Leads who spend more time on the website and have recent, positive engagement activities (like "SMS Sent") are far more likely to convert.
- **Data Quality Issues:** Several columns contain a high percentage of missing values (e.g., `How did you hear about X Education`, `Lead Profile`, `Lead Quality`), which will require careful imputation or removal during preprocessing.

Below is a detailed breakdown of the analysis.

---

2. Univariate Analysis: Understanding Individual Features

This analysis examines the characteristics of single variables to understand their distribution and composition.

Target Variable (`Converted`)

- The target variable shows a clear imbalance. Out of 9,240 leads, **5,679 (61.5%)** did not convert, while **3,561 (38.5%)** did. This is a crucial observation for the modeling phase.

Numerical Features

- **Total Time Spent on Website:** This feature's distribution is skewed, with a large number of leads spending very little time on the website. However, there is a substantial group that spends a significant amount of time, suggesting this could be a key differentiator.

- **TotalVisits & Page Views Per Visit:** Both of these metrics are heavily right-skewed. The majority of leads have very few visits and page views, indicating that high engagement is an exception rather than the norm.

Categorical Features

- **Lead Origin:** Most leads originate from **Landing Page Submissions**, followed by **API** calls and the **Lead Add Form**.
- **Lead Source:** The primary sources of leads are **Google** and **Direct Traffic**. **Olark Chat** and **Organic Search** are also significant contributors.
- **Last Activity:** The most common last activity is **Email Opened**, followed closely by **SMS Sent**. This highlights the importance of communication channels in the sales funnel.
- **Occupation:** A large majority of leads are categorized as **Unemployed**, which is the most frequent occupation status in the dataset.

---

3. Bivariate Analysis: Relationships with the Target Variable

This section explores how individual features relate to the `Converted` target variable.

Numerical Features vs. Conversion

- **Total Time Spent on Website vs. Converted:** This is the most powerful numerical predictor. The boxplot clearly shows that leads who converted spent, on average, a significantly longer time on the website compared to those who did not.
- **TotalVisits vs. Converted:** There is a slight tendency for converted leads to have more visits, but the overlap is substantial, making it a less decisive feature than time spent.
- **Page Views Per Visit vs. Converted:** Similar to total visits, converted leads tend to have slightly more page views per visit, but the relationship is not as strong as with time spent.

Categorical Features vs. Conversion

- **Lead Origin vs. Converted:** Leads from the **Lead Add Form** and **Lead Import** have a much higher conversion rate than those from Landing Pages or APIs.

- **Lead Source vs. Converted:** Leads sourced from **References** and **Welingak Website** show a very high probability of conversion. In contrast, sources like Google have a more balanced conversion rate.
- **Last Activity vs. Converted:** Leads whose last activity was **SMS Sent** have an exceptionally high conversion rate. This suggests that receiving an SMS is a strong indicator of high intent. Conversely, activities like "Olark Chat Conversation" or "Email Opened" have lower conversion rates.
- **Occupation vs. Converted: Working Professionals** have the highest conversion rate among all occupation types, making this a very valuable predictive feature.

---

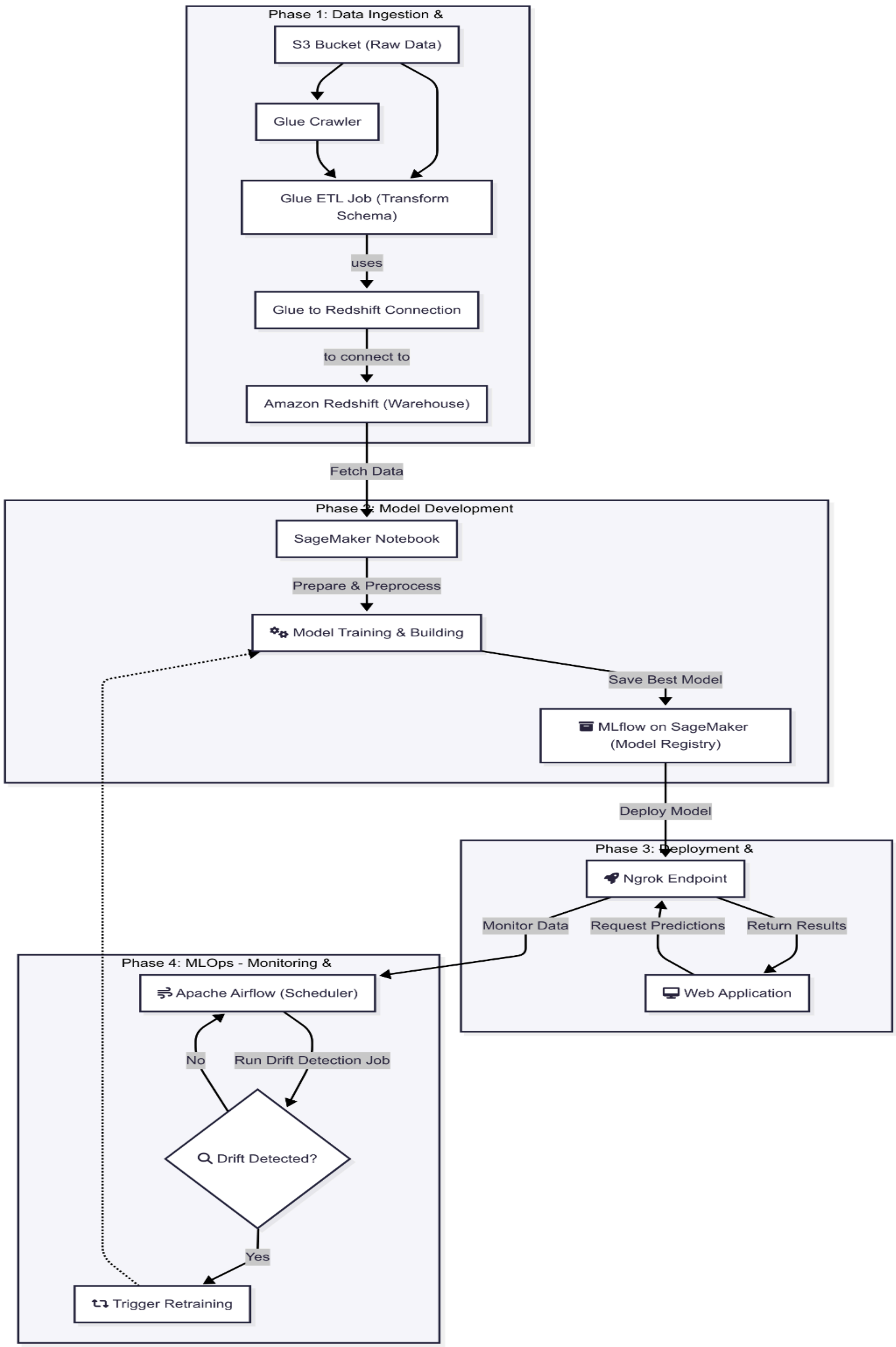
4. Multivariate Analysis: Inter-feature Relationships

This analysis examines the relationships between multiple features at once.

Correlation of Numerical Features

- The correlation heatmap reveals a strong positive correlation between `TotalVisits` and `Page Views Per Visit` (correlation coefficient of **0.52**). This is expected, as more visits naturally lead to more page views.

- `Total Time Spent on Website` has a moderate positive correlation with both `TotalVisits` and `Page Views Per Visit`.


Phase 1: Data Ingestion &
S3 Bucket (Raw Data)
Glue Crawler
Glue ETL Job (Transform Schema)
uses
Glue to Redshift Connection
to connect to
Amazon Redshift (Warehouse)
Fetch Data
Phase 2: Model Development
SageMaker Notebook
Prepare & Preprocess
Model Training & Building
Save Best Model
MLflow on SageMaker (Model Registry)
Deploy Model
Phase 3: Deployment &
Ngrok Endpoint
Monitor Data
Request Predictions
Return Results
Web Application
Phase 4: MLOps - Monitoring &
Apache Airflow (Scheduler)
No
Run Drift Detection Job
Drift Detected?
Yes
Trigger Retraining

Data Architecture Diagram for Lead Conversion:

Phase 1: Data Ingestion & Transformation

This initial phase is responsible for collecting raw data and transforming it into a clean, structured format suitable for machine learning. This is a classic ETL (Extract, Transform, Load) pipeline.

Detailed Breakdown of Phase 1 Components

1. S3 Bucket (Raw Data)

- **What it is**: Amazon S3 (Simple Storage Service) is a highly durable and scalable object storage service. Think of it as a vast, secure cloud-based hard drive where you can store any type of file, from CSVs and images to log files.
- **Role in this Pipeline**: The S3 bucket acts as the **Data Lake** or the initial landing zone. This is where the raw `Lead Scoring.csv` file, or any other raw lead data from various sources (e.g., web forms, CRM exports), is first collected. It holds the data in its original, unaltered format.
- **Why it's used**:
    - **Durability and Availability**: S3 is designed for 99.999999999% (11 nines) of durability, meaning the raw data is extremely safe from being lost.
    - **Scalability**: It can store a virtually unlimited amount of data, making the pipeline future-proof as the volume of leads grows.
    - **Decoupling**: Storing raw data here separates the data collection process from the data processing steps. This means the transformation jobs can be run or re-run without affecting the original source data.

2. Glue Crawler

- **What it is**: An AWS Glue Crawler is an automated program that connects to your data stores (like S3), scans the data, and infers its schema.
- **Role in this Pipeline**: The crawler points to the S3 bucket containing the raw lead data. It examines the files to automatically identify the data structure, such as column names (`Lead Origin`, `TotalVisits`, etc.), data types (string,

integer, float), and table partitions. It then populates this information as a metadata table in the **AWS Glue Data Catalog**.

- **Why it's used**:
  - **Automation**: It eliminates the need for a data engineer to manually define the schema for every dataset. This is especially useful when the source data format might change over time (e.g., new columns are added).
  - **Schema Management**: The Glue Data Catalog becomes a central repository for metadata, making the data easily discoverable and usable by other AWS services like the Glue ETL job.

3. Glue ETL Job (Transform Schema)

- **What it is**: AWS Glue ETL is a serverless data integration service. "ETL" stands for Extract, Transform, and Load. "Serverless" means you don't need to provision or manage any underlying servers; AWS handles the infrastructure automatically.
- **Role in this Pipeline**: This is the core data processing engine. It performs three key actions:
  - **Extract**: It uses the schema from the Glue Data Catalog to read the raw data from the S3 bucket.
  - **Transform**: It applies a series of data cleaning and transformation rules. This is where the logic from the project's data cleaning phase would be implemented (e.g., handling missing values, correcting data types, standardizing categorical values, creating new features). The goal is to convert the raw data into a pristine, analysis-ready format.
  - **Load**: After transformation, it prepares the data to be loaded into the data warehouse.
- **Why it's used**:
  - **Power and Scalability**: It runs on a managed Apache Spark environment, which is designed to process large-scale datasets efficiently and in parallel.
  - **Cost-Effectiveness**: You only pay for the resources used while the ETL job is running.

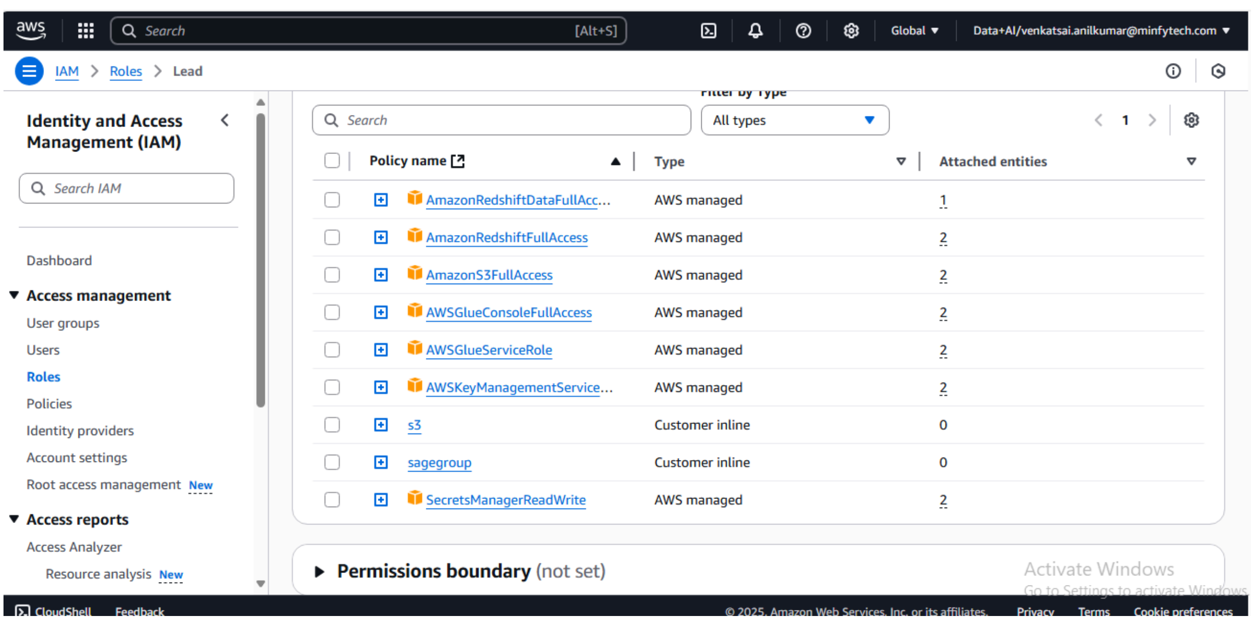
4. Glue to Redshift Connection

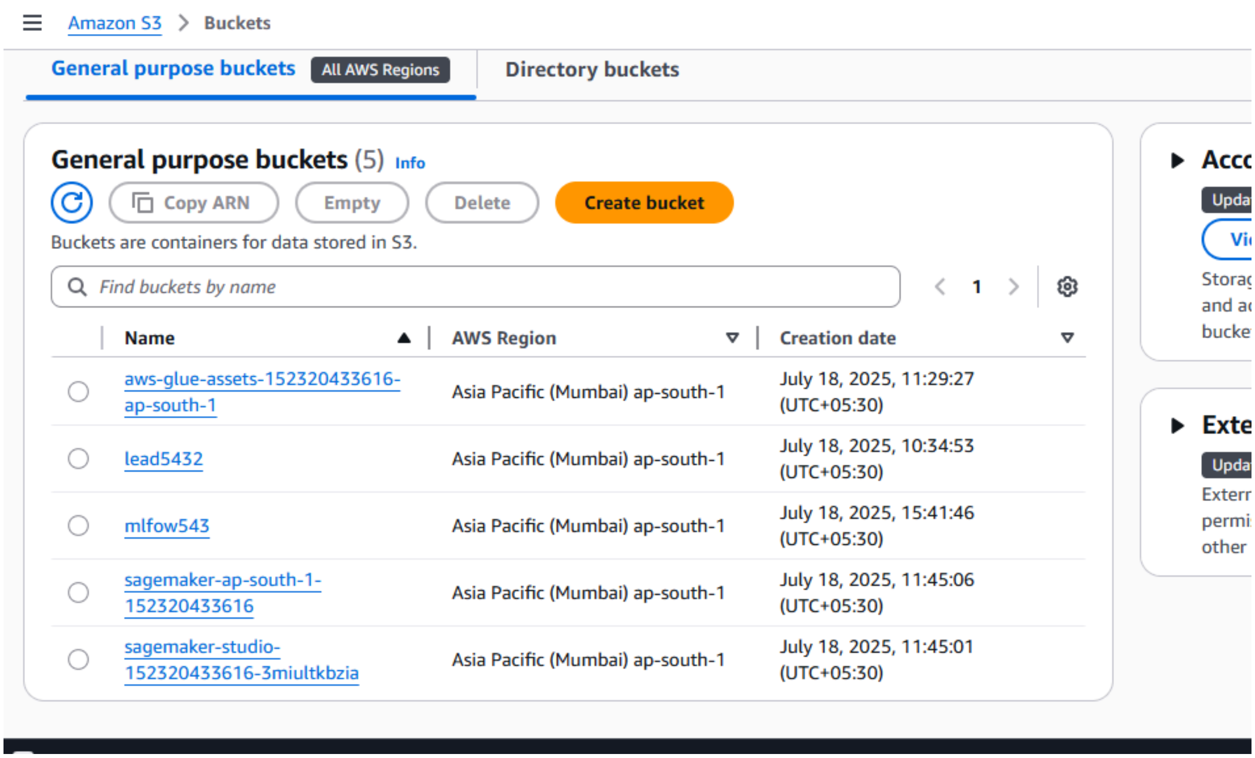
- **What it is**: This represents the data loading mechanism that moves the transformed data from the Glue ETL job into the Amazon Redshift data warehouse.
- **Role in this Pipeline**: It acts as the conduit between the processing environment (Glue) and the structured storage layer (Redshift). It handles the technical details of the data transfer, ensuring the data is loaded correctly into the appropriate Redshift tables.

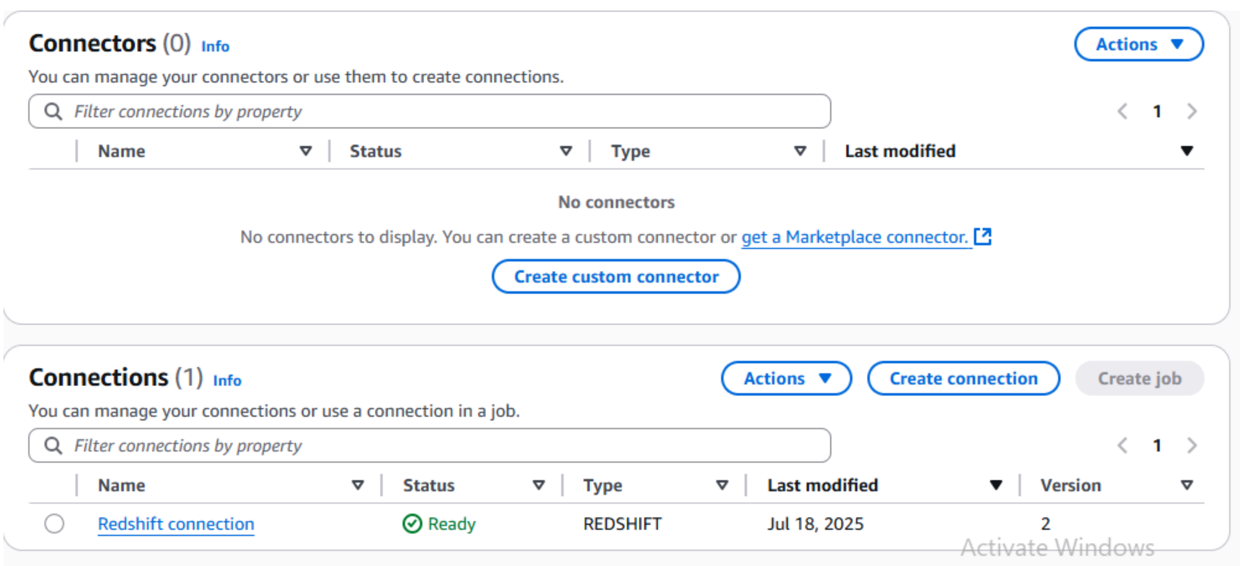
- **Why it's used**:
    - **Integration**: AWS services are designed to work together seamlessly. This connection is optimized for high-throughput data transfer, making the loading process fast and reliable.

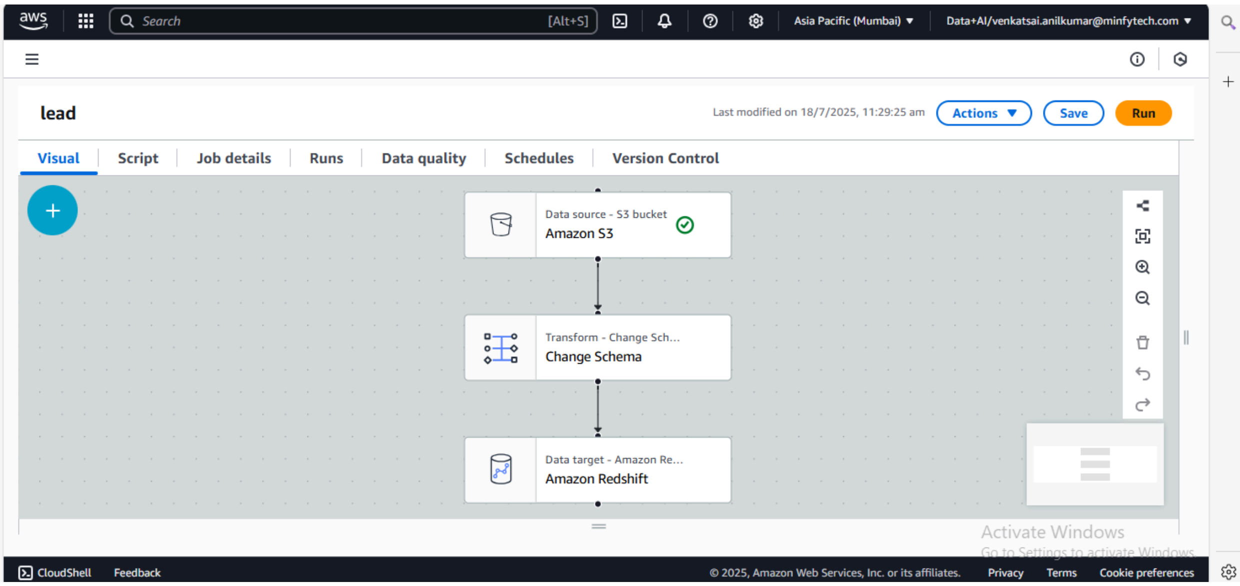
5. Amazon Redshift (Warehouse)

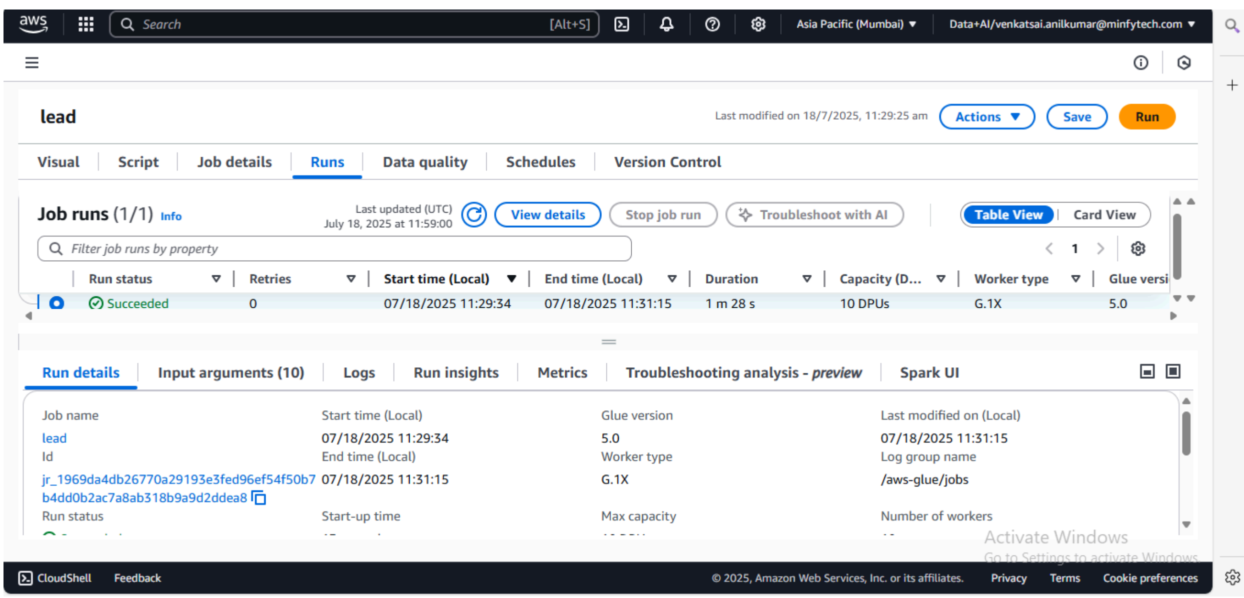
- **What it is**: Amazon Redshift is a fully managed, petabyte-scale cloud data warehouse service. It is designed for high-performance analysis and business intelligence.
- **Role in this Pipeline**: Redshift serves as the **single source of truth** for the clean, structured data. Unlike the data lake in S3 which holds raw data, the data warehouse holds analysis-ready data. All subsequent phases, especially Model Development, will query this Redshift database to get the data they need.
- **Why it's used**:
    - **Query Performance**: Redshift uses columnar storage and massively parallel processing (MPP), which makes it extremely fast for running the complex analytical queries required to extract feature sets for model training.
    - **Structured Data**: It provides a familiar SQL interface, making it easy for data scientists and analysts to access and manipulate the data.
    - **Concurrency**: It can efficiently handle multiple simultaneous requests from different users or applications.

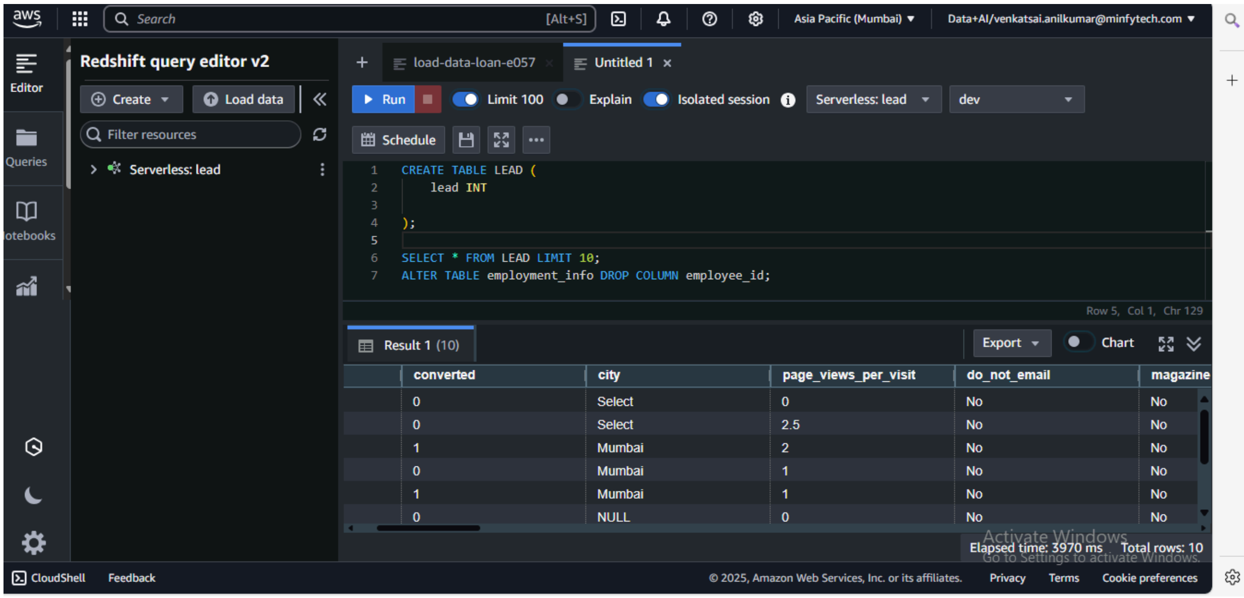



Amazon S3 > Buckets
General purpose buckets
All AWS Regions
Directory buckets
General purpose buckets (5) Info
Copy ARN
Empty
Delete
Create bucket
Buckets are containers for data stored in S3.
Find buckets by name
1
Name
AWS Region
Creation date
aws-glue-assets-152320433616-ap-south-1
Asia Pacific (Mumbai) ap-south-1
July 18, 2025, 11:29:27 (UTC+05:30)
lead5432
Asia Pacific (Mumbai) ap-south-1
July 18, 2025, 10:34:53 (UTC+05:30)
mlfow543
Asia Pacific (Mumbai) ap-south-1
July 18, 2025, 15:41:46 (UTC+05:30)
sagemaker-ap-south-1-152320433616
Asia Pacific (Mumbai) ap-south-1
July 18, 2025, 11:45:06 (UTC+05:30)
sagemaker-studio-152320433616-3miultkbzia
Asia Pacific (Mumbai) ap-south-1
July 18, 2025, 11:45:01 (UTC+05:30)


Connectors (0) Info
Actions
You can manage your connectors or use them to create connections.
Filter connections by property
1
Name
Status
Type
Last modified
No connectors
No connectors to display. You can create a custom connector or get a Marketplace connector.
Create custom connector
Connections (1) Info
Actions
Create connection
Create job
You can manage your connections or use a connection in a job.
Filter connections by property
1
Name
Status
Type
Last modified
Version
Redshift connection
Ready
REDSHIFT
Jul 18, 2025
2
Activate Windows


Search [Alt+S]
Asia Pacific (Mumbai)
Data+AI/venkatsai.anilkumar@minfytech.com
lead
Last modified on 18/7/2025, 11:29:25 am
Actions
Save
Run
Visual
Script
Job details
Runs
Data quality
Schedules
Version Control
Data source - S3 bucket
Amazon S3
Transform - Change Sch...
Change Schema
Data target - Amazon Re...
Amazon Redshift
Activate Windows
Go to Settings to activate Windows.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
Asia Pacific (Mumbai)
Data+AI/venkatsai.anilkumar@minfytech.com
lead
Last modified on 18/7/2025, 11:29:25 am
Actions
Save
Run
Visual
Script
Job details
Runs
Data quality
Schedules
Version Control
Job runs (1/1) Info
Last updated (UTC) July 18, 2025 at 11:59:00
View details
Stop job run
Troubleshoot with AI
Table View
Card View
Filter job runs by property
Run status | Retries | Start time (Local) | End time (Local) | Duration | Capacity (D... | Worker type | Glue versi
Succeeded | 0 | 07/18/2025 11:29:34 | 07/18/2025 11:31:15 | 1 m 28 s | 10 DPUs | G.1X | 5.0
Run details
Input arguments (10)
Logs
Run insights
Metrics
Troubleshooting analysis - preview
Spark UI
Job name
lead
Id
jr_1969da4db26770a29193e3fed96ef54f50b7b4dd0b2ac7a8ab318b9a9d2ddea8
Run status
Start time (Local)
07/18/2025 11:29:34
End time (Local)
07/18/2025 11:31:15
Start-up time
Glue version
5.0
Worker type
G.1X
Max capacity
Last modified on (Local)
07/18/2025 11:31:15
Log group name
/aws-glue/jobs
Number of workers
Activate Windows
Go to Settings to activate Windows.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



Phase 2: Model Development

This phase covers the work of the data scientist: building, training, and selecting the best machine learning model.

Detailed Breakdown of Phase 2 Components

1. Fetch Data

- **What it is**: This is the initial step of the model development phase, where the data scientist executes queries to retrieve the necessary data from the **Amazon Redshift** data warehouse.
- **Purpose in this Pipeline**: It ensures that all experiments and model training activities are performed on the official, clean, and centrally-managed dataset created in Phase 1. This prevents inconsistencies that can arise from using different or outdated local data files.
- **Key Benefits**:
  - **Consistency**: Guarantees that everyone on the team is working from the same "single source of truth."
  - **Performance**: Redshift is optimized for fast data retrieval, allowing for efficient access to large datasets required for training.

2. SageMaker Notebook

- **What it is**: An Amazon SageMaker Notebook is a fully managed, web-based interactive development environment (IDE) that runs Jupyter Notebook.
- **Purpose in this Pipeline**: This is the data scientist's primary workspace. It's where all the code for data exploration, preprocessing, model training, and evaluation is written and executed. The notebook environment comes pre-packaged with common machine learning libraries and integrates seamlessly with other AWS services.
- **Key Benefits**:
    - **Managed Environment**: AWS handles the server setup, maintenance, and security, allowing the data scientist to focus on building the model.
    - **Scalability**: The underlying compute power of the notebook can be easily scaled up or down to handle datasets of different sizes.
    - **Collaboration**: Notebooks can be easily shared among team members, promoting collaboration and peer review.

3. Prepare & Preprocess

- **What it is**: This step involves the final data transformations required to make the data suitable for machine learning algorithms. While some cleaning was done in Phase 1, this stage focuses on model-specific preparations.
- **Purpose in this Pipeline**: The code within the SageMaker notebook will perform tasks such as:
    - **Feature Engineering**: Creating new predictive features from existing ones.
    - **Train-Test Split**: Dividing the data into a training set (to teach the model) and a testing set (to evaluate its performance on unseen data).
    - **Scaling Numerical Features**: Standardizing the range of numerical columns (e.g., scaling `TotalVisits` to a 0-1 range) so that no single feature dominates the model's learning process.
    - **Encoding Categorical Features**: Converting text-based columns (like `Lead Origin`) into a numerical format that the model can understand.
- **Key Benefits**: This critical step directly impacts model performance. Proper preprocessing ensures that the model receives clean, well-formatted, and meaningful data, leading to more accurate predictions.

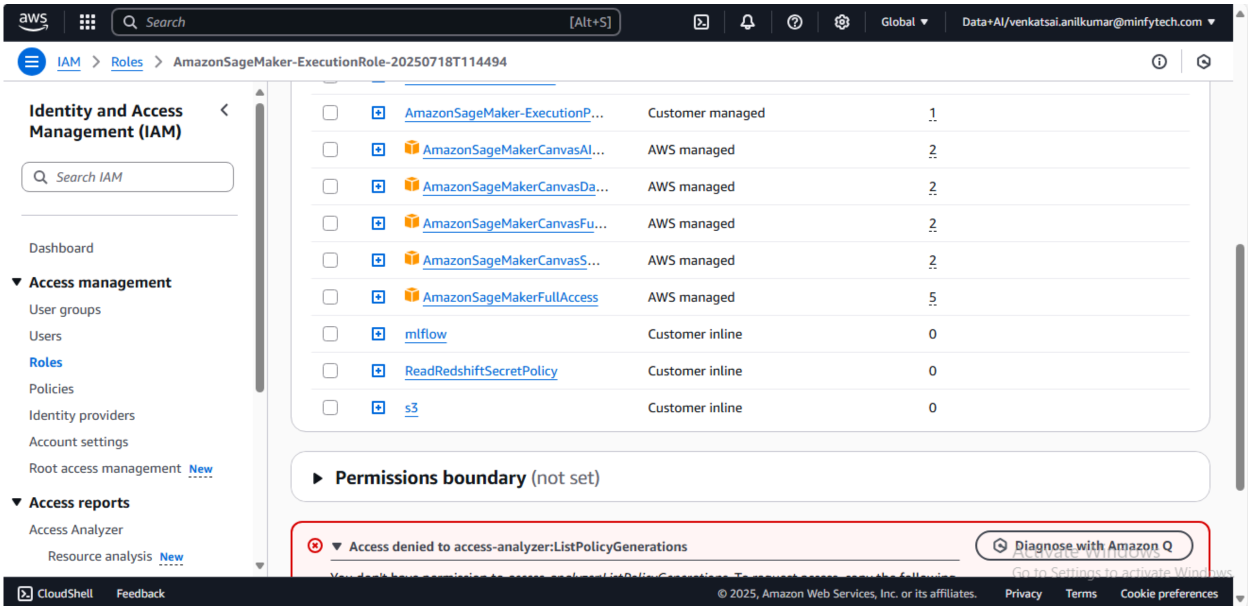
4. Model Training & Building

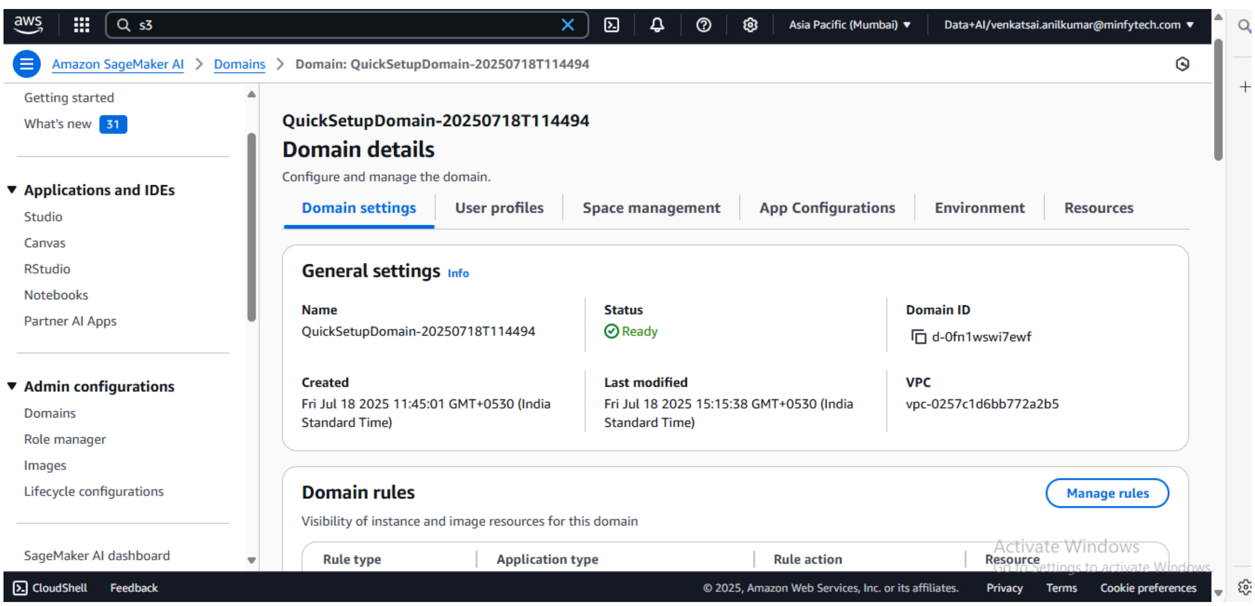
- **What it is**: This is the process where the machine learning algorithm learns patterns from the training data.
- **Purpose in this Pipeline**: One or more algorithms (e.g., Logistic Regression, XGBoost) are fed the preprocessed training data. The algorithm adjusts its

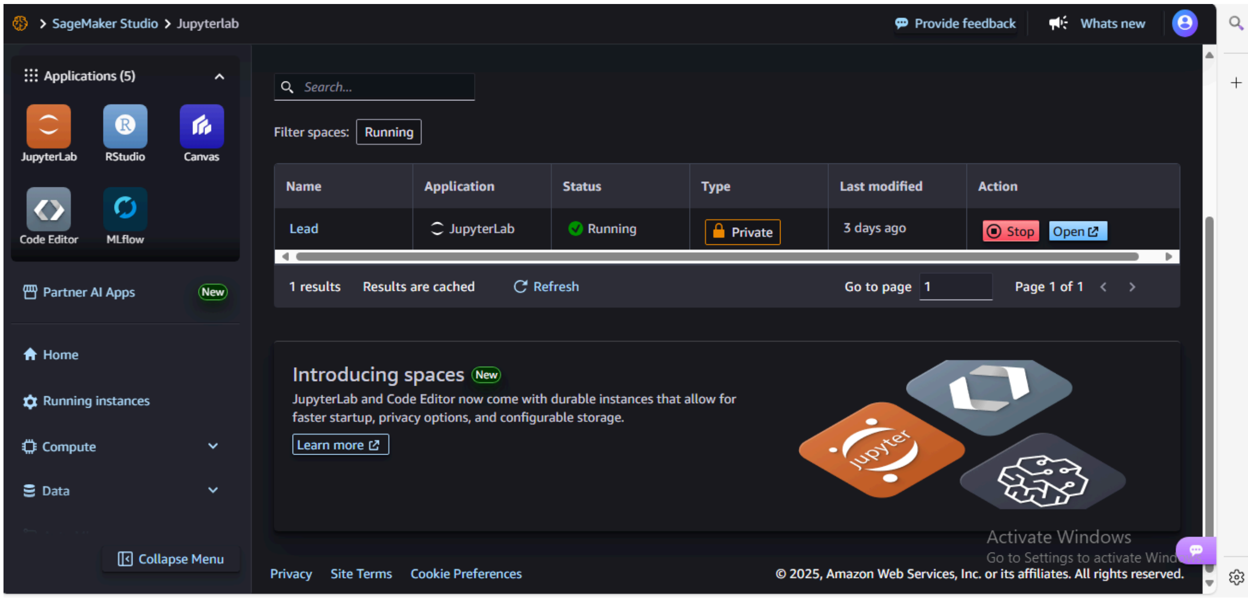
internal parameters to find the relationships between the input features (like `Total Time Spent on Website`) and the target variable (`Converted`). This step often includes **hyperparameter tuning**, where different settings for the algorithm are tested to find the combination that yields the "Best Model" based on performance metrics like accuracy or F1-score.
- **Key Benefits**: This is the core value-generating step where the raw data is transformed into a predictive tool.

5. MLflow on SageMaker (Model Registry)

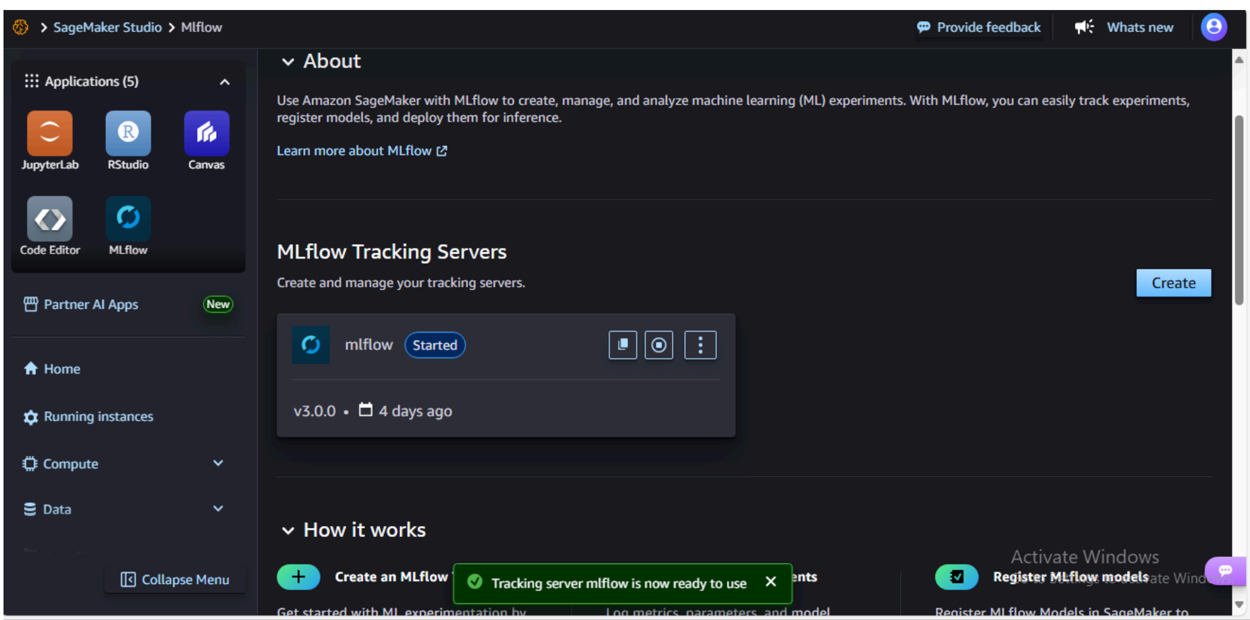
- **What it is**: MLflow is an open-source platform for managing the end-to-end machine learning lifecycle. The **Model Registry** is a centralized component for versioning, managing, and tracking models.
- **Purpose in this Pipeline**: When the "Best Model" is identified, it is not just saved as a file. It is formally **registered** in MLflow. This process captures:
    - **The Model Artifact**: The actual trained model file.
    - **Version Control**: The model is assigned a version number (e.g., `v1`, `v2`), allowing you to track changes over time.
    - **Metadata**: Key information is stored alongside the model, including the performance metrics, the specific hyperparameters used, and a link to the code that generated it.
- **Key Benefits**:
    - **Reproducibility and Governance**: Creates a clear, auditable trail of how each model was built and how it performed.
    - **Lifecycle Management**: Allows you to manage the model's stage (e.g., `Staging`, `Production`, `Archived`), which is essential for a controlled deployment process.
    - **CI/CD for ML**: The registry enables automation. Deployment scripts can programmatically query the registry to fetch the latest model approved for production, streamlining the path from experiment to deployment.
-


IAM
Roles
AmazonSageMaker-ExecutionRole-20250718T114494
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Access Analyzer
Resource analysis New
AmazonSageMaker-ExecutionP... Customer managed 1
AmazonSageMakerCanvasAI... AWS managed 2
AmazonSageMakerCanvasDa... AWS managed 2
AmazonSageMakerCanvasFu... AWS managed 2
AmazonSageMakerCanvasS... AWS managed 2
AmazonSageMakerFullAccess AWS managed 5
mlflow Customer inline 0
ReadRedshiftSecretPolicy Customer inline 0
s3 Customer inline 0
Permissions boundary (not set)
Access denied to access-analyzer:ListPolicyGenerations
Diagnose with Amazon Q
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


s3
Asia Pacific (Mumbai)
Data+AI/venkatsai.anilkumar@minfytech.com
Amazon SageMaker AI
Domains
Domain: QuickSetupDomain-20250718T114494
Getting started
What's new 31
Applications and IDEs
Studio
Canvas
RStudio
Notebooks
Partner AI Apps
Admin configurations
Domains
Role manager
Images
Lifecycle configurations
SageMaker AI dashboard
QuickSetupDomain-20250718T114494
Domain details
Configure and manage the domain.
Domain settings
User profiles
Space management
App Configurations
Environment
Resources
General settings Info
Name
QuickSetupDomain-20250718T114494
Status
Ready
Domain ID
d-0fn1wswi7ewf
Created
Fri Jul 18 2025 11:45:01 GMT+0530 (India Standard Time)
Last modified
Fri Jul 18 2025 15:15:38 GMT+0530 (India Standard Time)
VPC
vpc-0257c1d6bb772a2b5
Domain rules
Manage rules
Visibility of instance and image resources for this domain
Rule type
Application type
Rule action
Resource
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



SageMaker Studio > Jupyterlab
Provide feedback
Whats new
Applications (5)
JupyterLab
RStudio
Canvas
Code Editor
MLflow
Partner AI Apps
New
Home
Running instances
Compute
Data
Collapse Menu
Search...
Filter spaces: Running
Name
Application
Status
Type
Last modified
Action
Lead
JupyterLab
Running
Private
3 days ago
Stop
Open
1 results
Results are cached
Refresh
Go to page 1
Page 1 of 1
Introducing spaces
New
JupyterLab and Code Editor now come with durable instances that allow for faster startup, privacy options, and configurable storage.
Learn more
Privacy
Site Terms
Cookie Preferences
© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.



SageMaker Studio > Mlflow
Provide feedback
Whats new
Applications (5)
JupyterLab
RStudio
Canvas
Code Editor
MLflow
Partner AI Apps
New
Home
Running instances
Compute
Data
Collapse Menu
About
Use Amazon SageMaker with MLflow to create, manage, and analyze machine learning (ML) experiments. With MLflow, you can easily track experiments, register models, and deploy them for inference.
Learn more about MLflow
MLflow Tracking Servers
Create and manage your tracking servers.
Create
mlflow
Started
v3.0.0
4 days ago
How it works
Tracking server mlflow is now ready to use
Register MLflow models

```python
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])

# Use the verified, cleaned feature lists here
preprocessor = ColumnTransformer(transformers=[
    ('num', numerical_transformer, numerical_features),
    ('cat', categorical_transformer, categorical_features)
], remainder='passthrough')

# Save the preprocessor pipeline
joblib.dump(preprocessor, 'preprocess.pkl')
print(f"Preprocessing pipeline saved at: preprocess.pkl")

# Create the full pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])


# --- Step 7: Now, this command should execute successfully ---
X_transformed = pipeline.fit_transform(X)

print("\nPreprocessing successful!")
print("Shape of transformed data:", X_transformed.shape)
```

```
Preprocessing pipeline saved at: preprocess.pkl

Preprocessing successful!
Shape of transformed data: (10, 74)
```

Sales.ipynb | app.py | Terminal 1

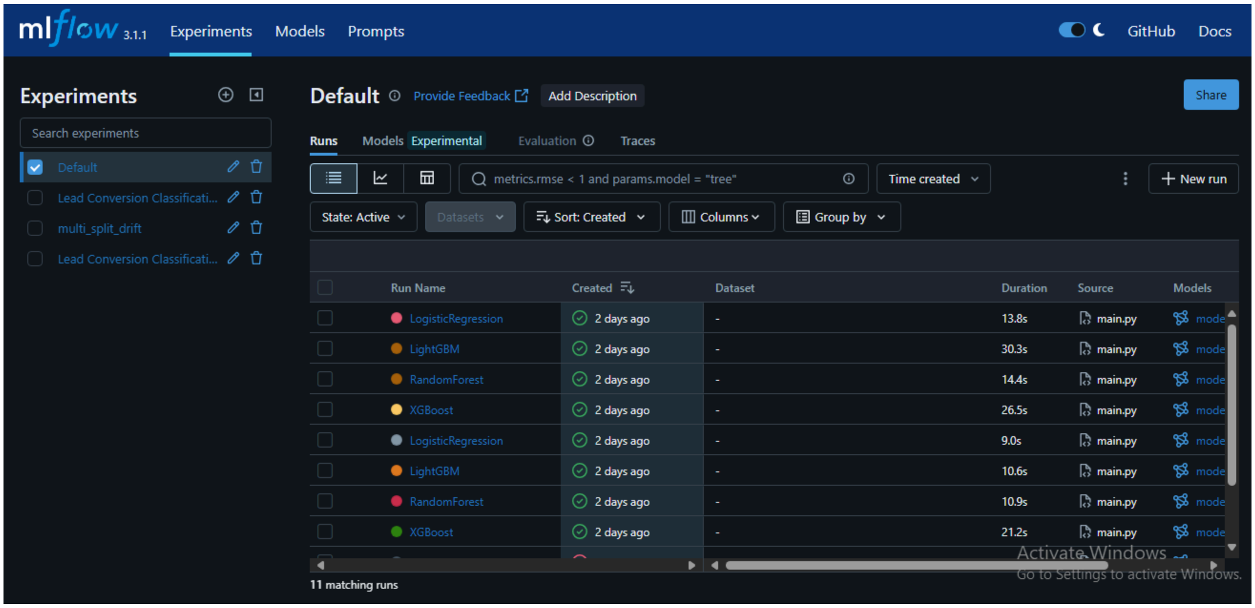
Code | Notebook | Cluster | Python 3 (ipykernel)

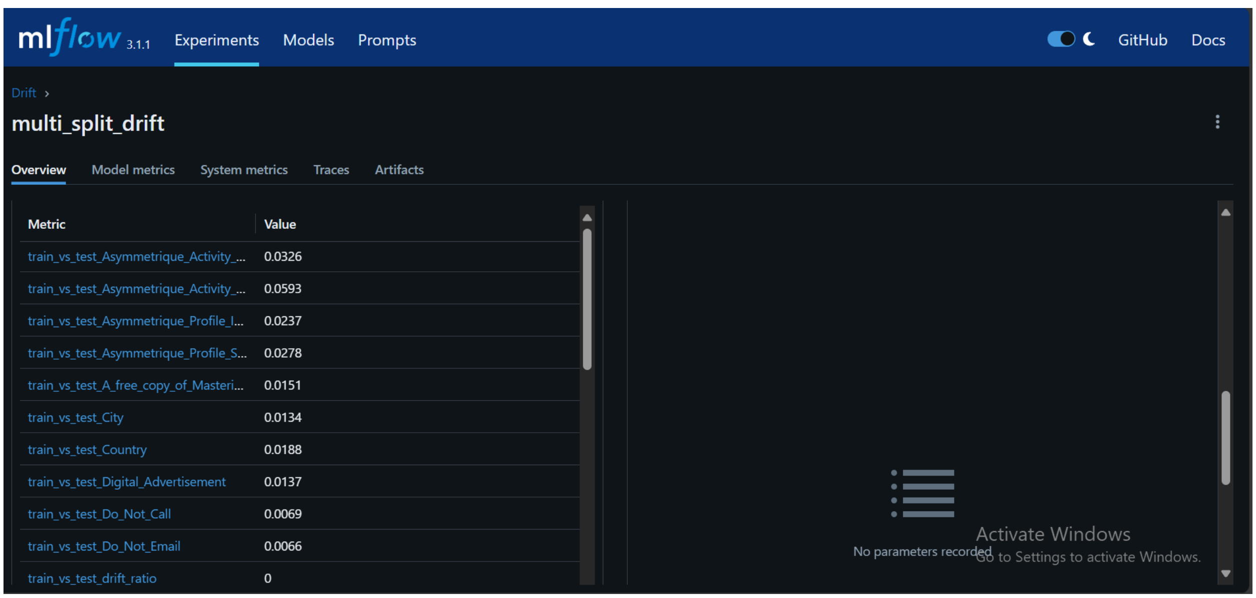
```
d10
View experiment at: https://ap-south-1.experiments.sagemaker.aws/#/experiments/1

--- Training LogisticRegression ---
Fitting 3 folds for each of 3 candidates, totalling 9 fits
2025/07/18 20:36:09 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_e
xample` parameter when logging the model to auto infer the model signature.
SHAP plot saved & logged: shap_outputs/LogisticRegression_shap_summary.png
View run LogisticRegression at: https://ap-south-1.experiments.sagemaker.aws/#/experiments/1/runs/f967b21222ee4a2fbeb
4fcd7bc26454c
View experiment at: https://ap-south-1.experiments.sagemaker.aws/#/experiments/1

--- Model Performance Summary ---
                model  f1_score   roc_auc  accuracy  precision  recall
0             XGBoost  0.875000  0.989583      0.90      0.875   0.875
1        RandomForest  0.933333  1.000000      0.95      1.000   0.875
2            LightGBM  0.769231  0.937500      0.85      1.000   0.625
3  LogisticRegression  0.933333  1.000000      0.95      1.000   0.875

--- Best model identified: RandomForest (F1 Score: 0.9333) ---

Registering the best model to MLflow Model Registry...
Registering model 'RandomForest' from run_id: 59dd4c376b1945b1b2eb206ea86df13d
Registered model 'RandomForest' already exists. Creating a new version of this model...
2025/07/18 20:36:11 INFO mlflow.store.model_registry.abstract_store: Waiting up to 300 seconds for model version to fini
sh creation. Model name: RandomForest, version 2
Created version '2' of model 'RandomForest'.
Successfully registered model 'RandomForest', Version: 2
Promoting model version 2 to 'Staging'...
Model 'RandomForest' version 2 successfully moved to 'Staging'.
```

Activate Windows
Go to Settings to activate Windows.

ython 3 (ipykernel) | Idle | Initialized (additional servers needed) | Instance MEM 72% | Amazon Q | Cookie Preferences | Mode: Command | Ln 1, Cc

Search | ENG





Phase 3: Deployment

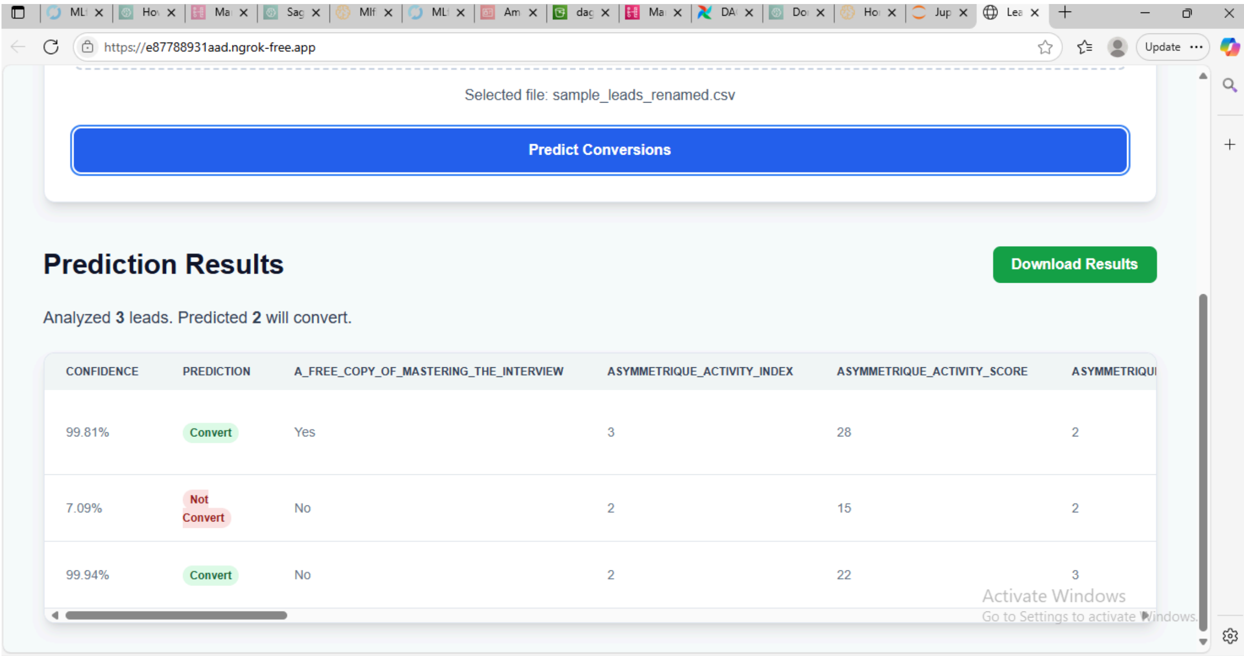
This phase takes the trained model and makes it available for use by other applications.

- **Deploy Model**: The best model from the Model Registry is deployed as a live service, creating an endpoint that can receive data and return predictions.
- **Ngrok Endpoint**: In this diagram, Ngrok is used to create a public URL for the model endpoint. This is useful for development and testing, allowing external

applications to easily access the model. In a production environment, this would typically be replaced by a more robust solution like Amazon API Gateway.

- **Web Application**: A front-end application (e.g., a website or dashboard) provides a user interface. This application allows users to:
    1. Input new data for which a prediction is needed.
    2. Send a **Request Predictions** call to the model's endpoint.
    3. Receive the model's prediction as **Return Results** and display it to the user.

```
sagemaker-user@default:~$ python app.py
2025-07-20 22:10:20,049 - INFO - ✅MLflow tracking URI set to: arn:aws:sagemaker:ap-south-1:152320433616:mlflow-tracking-server/mlflow
2025-07-20 22:10:20,670 - INFO - ✅Found latest production model: LogisticRegression (Version 5)
2025-07-20 22:10:20,670 - INFO - 📦Loading model from URI: models:/LogisticRegression/Production
Downloading artifacts: 100%|██████████████████████████████████████████████| 5/5 [00:00<00:00, 84.65it/s]
2025-07-20 22:10:21,691 - INFO - ✅MLflow model loaded successfully.
2025-07-20 22:10:21,717 - INFO - ✅Preprocessor 'preprocessor.pkl' loaded successfully.
2025-07-20 22:10:21,718 - INFO - ✅Model columns 'model_columns.json' loaded successfully.
2025-07-20 22:10:21,719 - INFO - Updating authtoken for default "config_path" of "ngrok_path": /home/sagemaker-user/.config/ngrok/ngrok
2025-07-20 22:10:21,732 - INFO - Opening tunnel named: http-8080-b86a3e2d-b0fd-46b3-a5b4-57ecd46fad9f
2025-07-20 22:10:21,743 - INFO - t=2025-07-20T22:10:21+0000 lvl=info msg="no configuration paths supplied"
2025-07-20 22:10:21,743 - INFO - t=2025-07-20T22:10:21+0000 lvl=info msg="using configuration at default config path" path=/home/sagemak
er-user/.config/ngrok/ngrok.yml
2025-07-20 22:10:21,744 - INFO - t=2025-07-20T22:10:21+0000 lvl=info msg="open config file" path=/home/sagemaker-user/.config/ngrok/ngro
k.yml err=nil
2025-07-20 22:10:21,770 - INFO - t=2025-07-20T22:10:21+0000 lvl=info msg="starting web service" obj=web addr=127.0.0.1:4040 allow_hosts=
[]
2025-07-20 22:10:22,389 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg="client session established" obj=tunnels.session
2025-07-20 22:10:22,390 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg="tunnel session started" obj=tunnels.session
2025-07-20 22:10:22,409 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg=start pg=/api/tunnels id=b9a8dedcef0f2d5c
2025-07-20 22:10:22,409 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg=end pg=/api/tunnels id=b9a8dedcef0f2d5c status=200 dur=567.761µ
s
2025-07-20 22:10:22,410 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg=start pg=/api/tunnels id=b417807847d91dc8
2025-07-20 22:10:22,410 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg=end pg=/api/tunnels id=b417807847d91dc8 status=200 dur=117.879µ
s
2025-07-20 22:10:22,411 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg=start pg=/api/tunnels id=81820f3d643a7218
2025-07-20 22:10:22,411 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg=end pg=/api/tunnels id=81820f3d643a7218 status=200 dur=85.274µs
2025-07-20 22:10:22,412 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg=start pg=/api/tunnels id=f6a07ef4260b6ab8
2025-07-20 22:10:22,624 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg="started tunnel" obj=tunnels name=http-8080-b86a3e2d-b0fd-46b3-
a5b4-57ecd46fad9f addr=http://localhost:8080 url=https://22355ba88354.ngrok-free.app
2025-07-20 22:10:22,625 - INFO - 🌐Public URL: https://22355ba88354.ngrok-free.app
2025-07-20 22:10:22,625 - INFO - t=2025-07-20T22:10:22+0000 lvl=info msg=end pg=/api/tunnels id=f6a07ef4260b6ab8 status=201 dur=212.1812
01ms
```



Apache Airflow (Scheduler)

- **What it is**: Apache Airflow is a powerful open-source platform used to programmatically author, schedule, and monitor complex workflows. Workflows in Airflow are defined as **DAGs (Directed Acyclic Graphs)**, which are essentially scripts that lay out tasks and their dependencies.
- **Purpose in this Pipeline**: Airflow acts as the **orchestrator** or the "brain" of the MLOps cycle. It is responsible for running the routine maintenance tasks automatically without human intervention. For this pipeline, its primary job is to kick off the `Run Drift Detection Job` on a regular schedule (e.g., daily, weekly).
- **Key Benefits**:
  - **Automation**: Eliminates the need for a person to manually run monitoring scripts.
  - **Reliability**: Airflow has built-in mechanisms for retrying failed tasks, sending alerts, and providing detailed logs, making the entire process robust.
  - **Complex Scheduling**: It can manage complex dependencies. For example, it can be configured to only start the drift job *after* the daily data ingestion pipeline (Phase 1) has successfully completed.

2. Monitor Data

- **What it is**: This is not a specific tool but a crucial process. It involves capturing and storing the input data that is being sent to the live model endpoint (from Phase 3) for predictions.
- **Purpose in this Pipeline**: This captured data represents the "real-world" data that the model is currently facing. It is collected over a specific time window (e.g., the last 24 hours) and serves as the dataset that will be analyzed for drift. Without this monitoring and logging, there would be no data to analyze.
- **How it's done**: Typically, the prediction endpoint is configured to log every request it receives to a storage location like an S3 bucket or a database.

3. Run Drift Detection Job

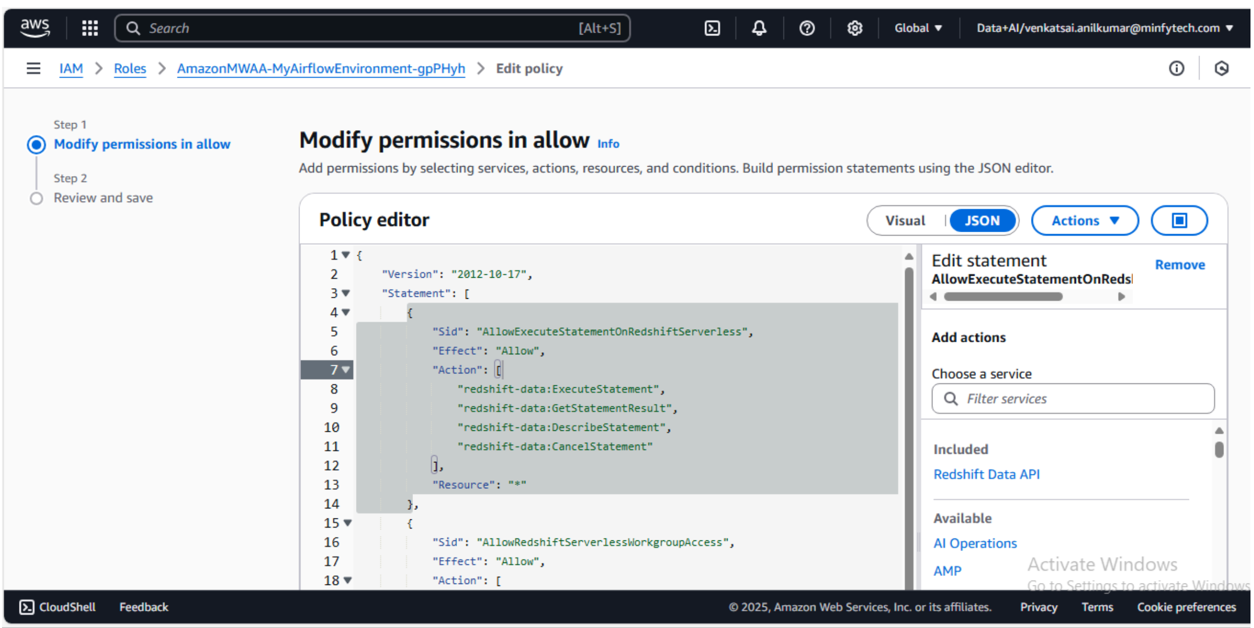
- **What it is**: This is an automated script or task that is executed by the Airflow scheduler.
- **Purpose in this Pipeline**: This job is responsible for detecting **data drift**. Data drift occurs when the statistical properties of the live production data change significantly from the data the model was originally trained on. This job compares two datasets:
    - **The Reference Dataset**: The original training data (the baseline).
    - **The Current Dataset**: The recently captured "Monitored Data."
- **The Process**: The job uses a data drift tool (like **Evidently**, which was mentioned in your project files) to perform statistical comparisons on the features. It checks for things like:
    - **Changes in mean, median, or standard deviation** for numerical features.
    - **Changes in the frequency distribution** of categories for categorical features.
    - **The emergence of new, unseen categories**. The output of this job is a report or a score that quantifies how much the data has "drifted."

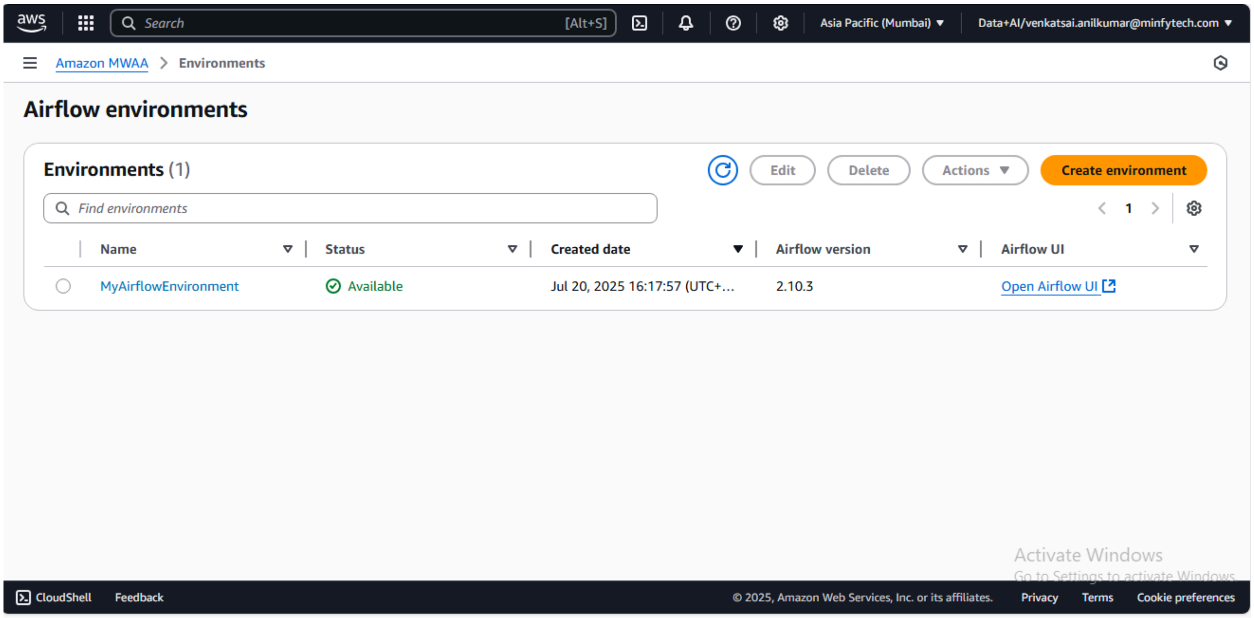
4. Drift Detected? (Decision Point)

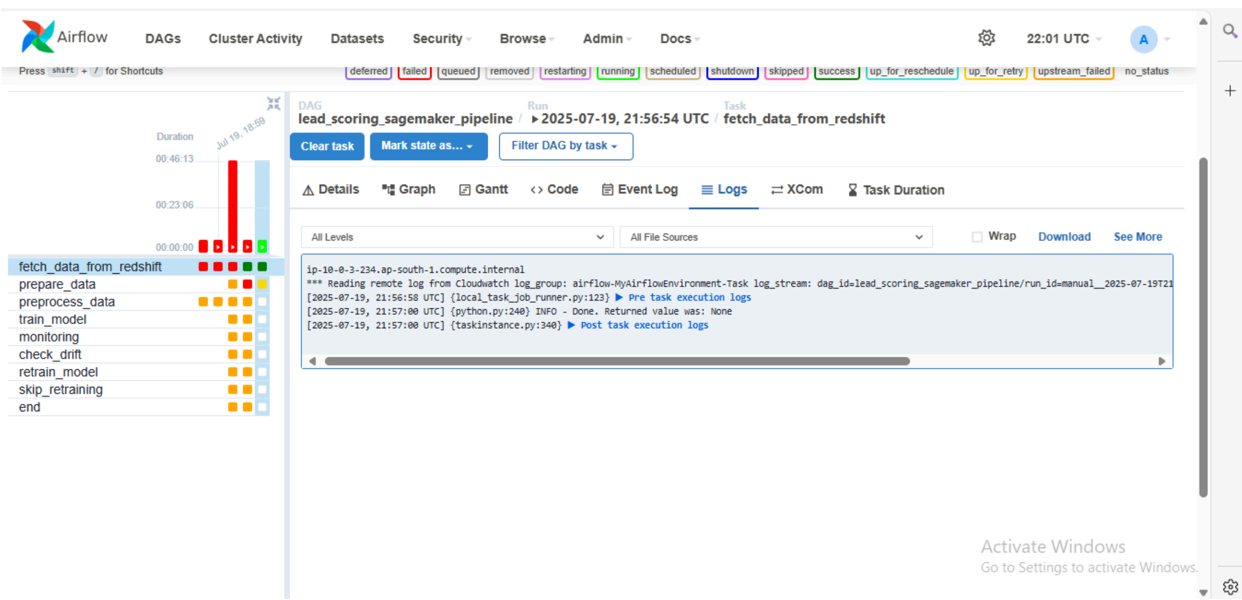
- **What it is**: This is a conditional step in the Airflow workflow that acts on the results of the drift detection job.
- **Purpose in this Pipeline**: It uses a predefined threshold to make a decision. For example, a rule might be "If more than 10% of the features have drifted, or if the overall drift score is above 0.2, then proceed."
    - **If No**: The live data is still similar enough to the training data. The model is considered healthy, and the workflow ends until the next scheduled run.
    - **If Yes**: This is a critical alert. It signifies that the real-world conditions have changed, and the model's predictions may no longer be accurate because it is seeing data it wasn't trained to handle.

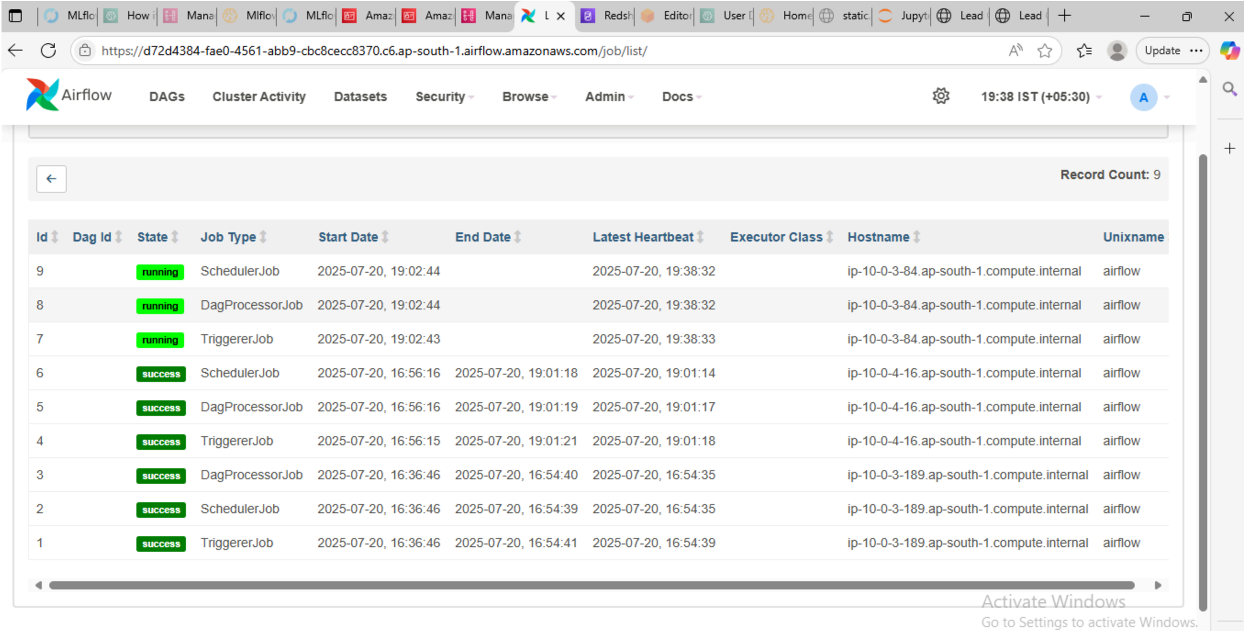
5. Trigger Retraining

- **What it is**: This is the automated action taken when significant drift is confirmed.
- **Purpose in this Pipeline**: This step creates a closed-loop system. The "Yes" outcome from the decision block automatically kicks off a new model training pipeline. As the dotted line shows, this action loops back to **Phase 2 (Model Development)**.
- **The Retraining Cycle**:
    1. The trigger instructs the system to start a new training run.
    2. This new run will fetch the **latest clean data** from the Redshift warehouse, which now includes the recent data that caused the drift.
    3. The model is retrained on this fresh, more representative data.
    4. The newly trained model is evaluated, and if it performs better than the current production model, it is registered in the MLflow Model Registry.
    5. This can then trigger a subsequent deployment pipeline to replace the old, underperforming model with the new one.




aws
Search
[Alt+S]
Asia Pacific (Mumbai)
Data+AI/venkatsai.anilkumar@minfytech.com
Amazon MWAA
Environments
Airflow environments
Environments (1)
Edit
Delete
Actions
Create environment
Find environments
Name
Status
Created date
Airflow version
Airflow UI
MyAirflowEnvironment
Available
Jul 20, 2025 16:17:57 (UTC+...
2.10.3
Open Airflow UI
Activate Windows
Go to Settings to activate Windows.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
22:01 UTC
Press shift + / for Shortcuts
deferred failed queued removed restarting running scheduled shutdown skipped success up_for_reschedule up_for_retry upstream_failed no_status
DAG
lead_scoring_sagemaker_pipeline
Run
2025-07-19, 21:56:54 UTC
Task
fetch_data_from_redshift
Clear task
Mark state as...
Filter DAG by task
Duration
00:46:13
00:23:06
00:00:00
Details
Graph
Gantt
Code
Event Log
Logs
XCom
Task Duration
All Levels
All File Sources
Wrap
Download
See More
fetch_data_from_redshift
prepare_data
preprocess_data
train_model
monitoring
check_drift
retrain_model
skip_retraining
end
ip-10-0-3-234.ap-south-1.compute.internal
*** Reading remote log from Cloudwatch log_group: airflow-MyAirflowEnvironment-Task log_stream: dag_id=lead_scoring_sagemaker_pipeline/run_id=manual__2025-07-19T21
[2025-07-19, 21:56:58 UTC] {local_task_job_runner.py:123} ▶ Pre task execution logs
[2025-07-19, 21:57:00 UTC] {python.py:240} INFO - Done. Returned value was: None
[2025-07-19, 21:57:00 UTC] {taskinstance.py:340} ▶ Post task execution logs
Activate Windows
Go to Settings to activate Windows.



Id	Dag Id	State	Job Type	Start Date	End Date	Latest Heartbeat	Executor Class	Hostname	Unixname
9		running	SchedulerJob	2025-07-20, 19:02:44		2025-07-20, 19:38:32		ip-10-0-3-84.ap-south-1.compute.internal	airflow
8		running	DagProcessorJob	2025-07-20, 19:02:44		2025-07-20, 19:38:32		ip-10-0-3-84.ap-south-1.compute.internal	airflow
7		running	TriggererJob	2025-07-20, 19:02:43		2025-07-20, 19:38:33		ip-10-0-3-84.ap-south-1.compute.internal	airflow
6		success	SchedulerJob	2025-07-20, 16:56:16	2025-07-20, 19:01:18	2025-07-20, 19:01:14		ip-10-0-4-16.ap-south-1.compute.internal	airflow
5		success	DagProcessorJob	2025-07-20, 16:56:16	2025-07-20, 19:01:19	2025-07-20, 19:01:17		ip-10-0-4-16.ap-south-1.compute.internal	airflow
4		success	TriggererJob	2025-07-20, 16:56:15	2025-07-20, 19:01:21	2025-07-20, 19:01:18		ip-10-0-4-16.ap-south-1.compute.internal	airflow
3		success	DagProcessorJob	2025-07-20, 16:36:46	2025-07-20, 16:54:40	2025-07-20, 16:54:35		ip-10-0-3-189.ap-south-1.compute.internal	airflow
2		success	SchedulerJob	2025-07-20, 16:36:46	2025-07-20, 16:54:39	2025-07-20, 16:54:35		ip-10-0-3-189.ap-south-1.compute.internal	airflow
1		success	TriggererJob	2025-07-20, 16:36:46	2025-07-20, 16:54:41	2025-07-20, 16:54:39		ip-10-0-3-189.ap-south-1.compute.internal	airflow

Error:

```
         ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/airflow/.local/lib/python3.11/site-packages/airflow/models/baseoperator.py", line 417, in wrapper
    return func(self, *args, **kwargs)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/airflow/.local/lib/python3.11/site-packages/airflow/operators/python.py", line 238, in execute
    return_value = self.execute_callable()
                 ^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/airflow/.local/lib/python3.11/site-packages/airflow/operators/python.py", line 256, in execute_callable
    return runner.run(*self.op_args, **self.op_kwargs)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/airflow/.local/lib/python3.11/site-packages/airflow/utils/operator_helpers.py", line 252, in run
    return self.func(*args, **kwargs)
         ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/local/airflow/dags/dag 1.py", line 104, in run_prepare
    run_sagemaker_job('data_preparation.py', 'prepare-job', f'raw/lead_scoring_{ds}.csv', **kwargs)
  File "/usr/local/airflow/dags/dag 1.py", line 79, in run_sagemaker_job
    import sagemaker
ModuleNotFoundError: No module named 'sagemaker'
[2025-07-20, 21:06:44 UTC] {taskinstance.py:1225} INFO - Marking task as FAILED. dag_id=lead_scoring_sagemaker_pipeline, task_id=prepare_data, ru
[2025-07-20, 21:06:44 UTC] {taskinstance.py:340} ▶ Post task execution logs
```

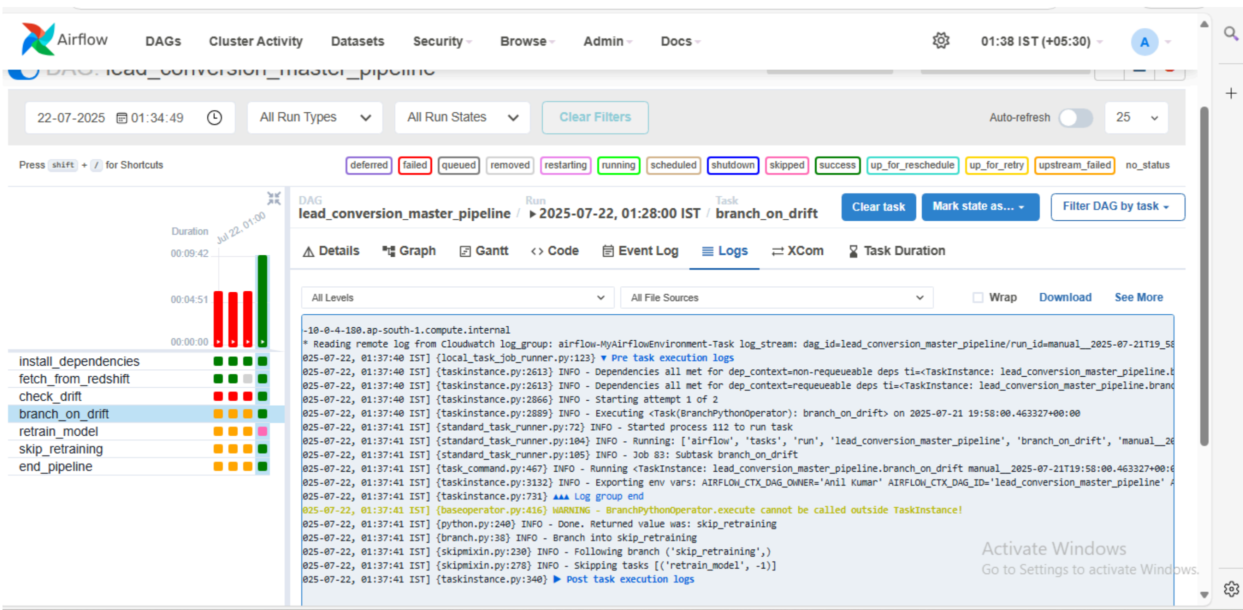
Here, I am trying to do an Airflow task that is trying to import the `sagemaker` Python package (used to interact with Amazon SageMaker), but it is **not installed in the environment** where the task is running.

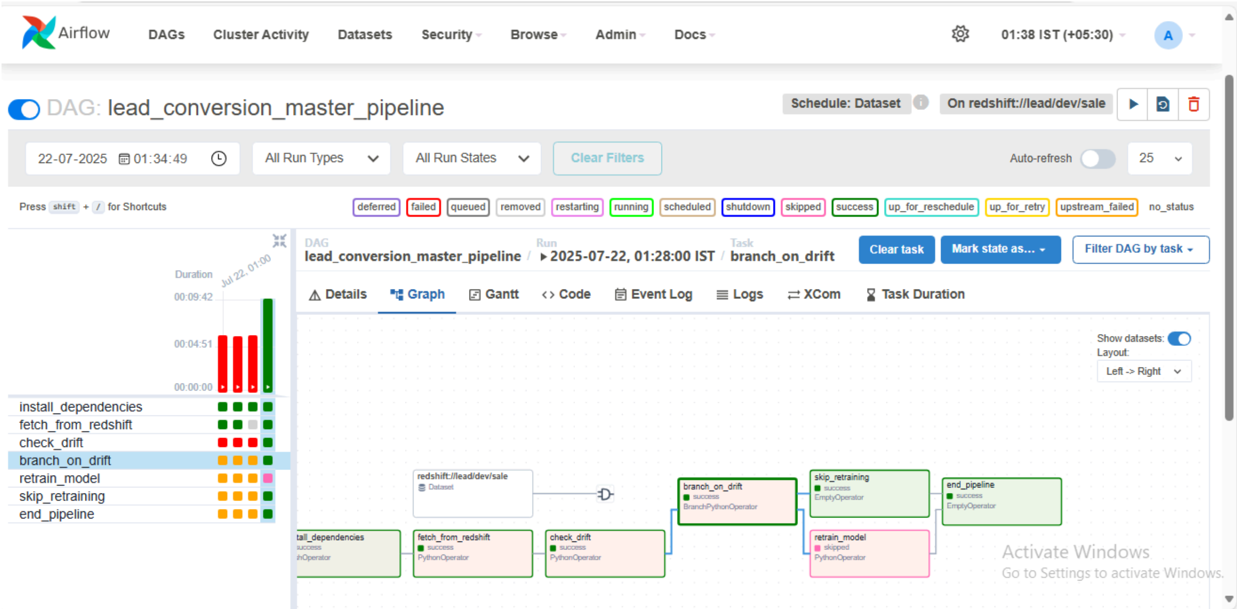
Here I am trying to automate the process of ML flow by sagemaker jobs. When the data drift found it will directly trigger the retraining pipeline. By we can able to get rid of from the data drift.

But because of not installed Sagemaker module, I am not able to access Sagemaker.

After these ,I have try do in a different then it worked for me.here in s3 bucket i have created a script folder here i kept the full pipeline python code in then .i have created a new folder kept the new data in to check the drift .if drift were found ,It will run the full_pipeline python script.for both data combined.

Here are the results:





About the Imports:

Category	Library / Module	Purpose / Functionality
Data Handling & Computing	`pandas`	Used for advanced manipulation of tabular data through its powerful DataFrame structures.
	`numpy`	Provides support for efficient numerical and mathematical operations on large, multi-dimensional arrays and matrices.
Data Visualization	`matplotlib.pyplot`	A foundational library for creating a wide range of static, animated, and interactive visualizations.
	`seaborn`	Built on top of matplotlib, this library is used for creating more attractive and informative statistical graphics.

Machine Learning (Scikit-learn)	`Pipeline`, `ColumnTransformer`	Tools to build and manage sequences of data transformation and modeling steps.
	`MinMaxScaler`, `OneHotEncoder`	Preprocessing utilities to scale numerical data and convert categorical data into a machine-readable format.
	`SimpleImputer`	Used for handling missing data points within the dataset.
	`train_test_split`, `GridSearchCV`	Functions to split data into training and testing sets and to perform exhaustive searches for optimal model hyperparameters.
	`LogisticRegression`, `RandomForest`	Implementations of standard classification algorithms used for prediction.
	`sklearn.metrics`	A module containing functions to evaluate the performance of classification models (e.g., accuracy, precision, recall).
Advanced Modeling & Preprocessing	`feature_engine.outliers.Winsorizer`	A specific tool used to manage and cap outliers in the data.
	`imblearn.SMOTE`	An over-sampling technique used to correct class imbalance in the dataset.
	`xgboost.XGBClassifier`	An optimized and powerful gradient boosting algorithm for classification tasks.

MLOps & Experiment Tracking	`mlflow`	A platform to manage the end-to-end machine learning lifecycle, including tracking experiments, logging metrics, and registering models.
Model Explainability	`shap`	A library used to explain the output of machine learning models, helping to understand feature importance and how predictions are made.
Data Drift & Validation	`evidently`	A tool designed to detect and visualize data drift, helping to monitor and validate data stability over time.
Utilities & Miscellaneous	`joblib`	Used for efficient saving and loading of Python objects, particularly useful for ML models and pipelines.
	`os`	Provides a way of using operating system-dependent functionality, such as managing file paths and directories.
	`re`	The regular expressions module, used for advanced string searching and manipulation, often for filtering column names.
	`warnings`	Used to control how warning messages are handled and displayed during code execution.

Data Ingestion:

Step 1: Configuration

```
region = 'ap-south-1'

workgroup_name = 'default-workgroup'

database_name = 'dev'

secret_arn = '<your-secret-arn>'

sql = 'SELECT * FROM sale'
```

- Sets AWS region, Redshift **workgroup name**, database name, secret ARN for authentication, and SQL query.
- secret_arn is created using **AWS Secrets Manager** and stores Redshift credentials securely.

Step 2: Create Redshift Data API Client

```
client = boto3.client('redshift-data', region_name=region)
```

- Uses boto3 to create a Redshift Data API client.
- This is how you interact with Redshift **without needing a JDBC connection**.

Step 3: Run the SQL Query

```
response = client.execute_statement(

    WorkgroupName=workgroup_name,

    Database=database_name,

    SecretArn=secret_arn,

    Sql=sql

    )
```

- Executes the given SQL query on the Redshift Serverless **workgroup**.
- Returns a statement_id used to track the query.

Step 4: Wait for Query Completion

```
statement_id = response['Id']

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)
```

- Repeatedly checks the query status.
- Waits (sleep) until the query either **FINISHES** or **FAILS**.

Step 5: Retrieve Query Results

```
result = client.get_statement_result(Id=statement_id)
```

- Retrieves the actual data after the query is finished.
- Returns both **column names** and **row data**.

Step 6: Parse the Results

```
columns = [col['name'] for col in result['ColumnMetadata']]
rows = result['Records']
data = []
for row in rows:
    data.append([list(col.values())[0] if col else None for col in row])
```

- Extracts **column names** from metadata.
- Iterates over each record and extracts values (since each value is a dictionary like {'stringValue': 'abc'}).

Step 7: Convert to Pandas DataFrame

```
df = pd.DataFrame(data, columns=columns)
```

- Converts the extracted data and columns into a standard Pandas DataFrame for analysis or ML input.

Step 8: Display the Result

```
print(df.head())
```

- Prints the first 5 rows to check the output.

Phase 1: Initial Data Inspection

Objective: To get a high-level understanding of the dataset's structure, content, and quality. This is the first step in any exploratory data analysis (EDA) process.

Code Command	What It Does	Key Insights Gained
`df.head()`	Displays the first five rows of the DataFrame.	Provides a quick preview of the data, column names, and sample values.
`df.info()`	Shows a concise summary, including column names, data types, and non-null counts.	Helps identify incorrect data types and the extent of missing information across columns.
`df.describe()`	Generates descriptive statistics for all numerical columns.	Reveals the central tendency (mean), dispersion (std), and range (min/max) of numerical data.
`df.isnull().sum()`	Counts the number of missing (null or NaN) values in each column.	Pinpoints which features have missing data and how much, guiding the data cleaning strategy.
`df.duplicated().sum()`	Counts the total number of complete duplicate rows in the dataset.	Checks for data redundancy, which could bias model training if not handled.
`df.columns`	Returns a list of all column names.	Useful for a quick reference of all available features.

Phase 2: Data Cleaning and Standardization

Objective: To tidy the dataset by standardizing formats and removing irrelevant information, making it reliable for modeling.

Code Command	What It Does	Outcome & Importance
`df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_")`	Standardizes all column names by removing leading/trailing whitespace, converting to lowercase, and replacing spaces with underscores.	This ensures column names are consistent and easy to access in code, preventing errors (e.g., `df.lead_number` vs. `df['Lead Number']`).
`df.drop(['prospect_id', 'lead_number'], axis=1, inplace=True)`	Removes the `prospect_id` and `lead_number` columns from the DataFrame.	These columns are unique identifiers that offer no predictive value for a general model, so they are dropped to reduce noise.
`df.replace(["Select", "", None], np.nan, inplace=True)`	Finds all instances of "Select", empty strings, and `None` across the DataFrame and	This unifies all forms of missing or placeholder data into a standard `NaN` format, which is essential for

		replaces them with `np.nan`.	accurate missing value imputation.

Feature Engineering & Data Preparation

This table details the key data transformation and preparation steps performed after initial cleaning. These actions are designed to create a clean, robust feature set and prepare the data for model training.

Step	Columns Affected	Description	Rationale / Benefit
Custom Mapping & Consolidation	`Country`, `City`, `Lead Source`, `Lead Profile`, `What is your current occupation`, `Last Notable Activity`, `Tags`, `Specialization`, `Lead Quality`	A unified function was used to group granular categorical values into broader, more meaningful categories.	This reduces the number of unique categories (cardinality), handles inconsistencies, and creates stronger, more generalized features for the model.
Geographic Tiering	`Country`, `City`	Countries were mapped into tiers (e.g., "Tier 1," "Tier 2"), and cities were categorized as "Metro" or "Non-Metro."	Converts raw location data into a more useful feature representing market type or economic region.
Source & Profile Grouping	`Lead Source`, `Lead Profile`, `What is your current occupation`	Specific sources like "Google" and "bing" were grouped into "Search Engine." Occupations were consolidated into standard categories like "Student" and "Professional."	Standardizes lead origins and professional backgrounds, making the features more robust and easier for the model to interpret.

Behavioral Categorization	`Last Notable Activity`, `Tags`	Activities and tags with high variability were mapped to a smaller set of standardized statuses, such as "Engaged via Email" or "Trying to Contact."	Simplifies complex behavioral data into clear signals of lead engagement and current status.
Feature Separation	`converted` (and all other columns)	The `converted` column was isolated as the target variable (`y`), while all other columns were designated as the feature set (`X`).	This is a standard and necessary step to prepare data for supervised machine learning, separating the independent variables from the dependent variable.
Class Imbalance Check	`converted` (Target Variable)	The distribution of the target variable was visualized using `value_counts()` and `seaborn.countplot()`.	This analysis is crucial to identify if there is an imbalance between the classes (converted vs. not converted), which would require special handling (like SMOTE) during model training to prevent bias

Separate Features and Target

X = df.drop(columns=["converted"]):

All columns except the target (converted) become features.

y = df["converted"]:

The target label is isolated to y.

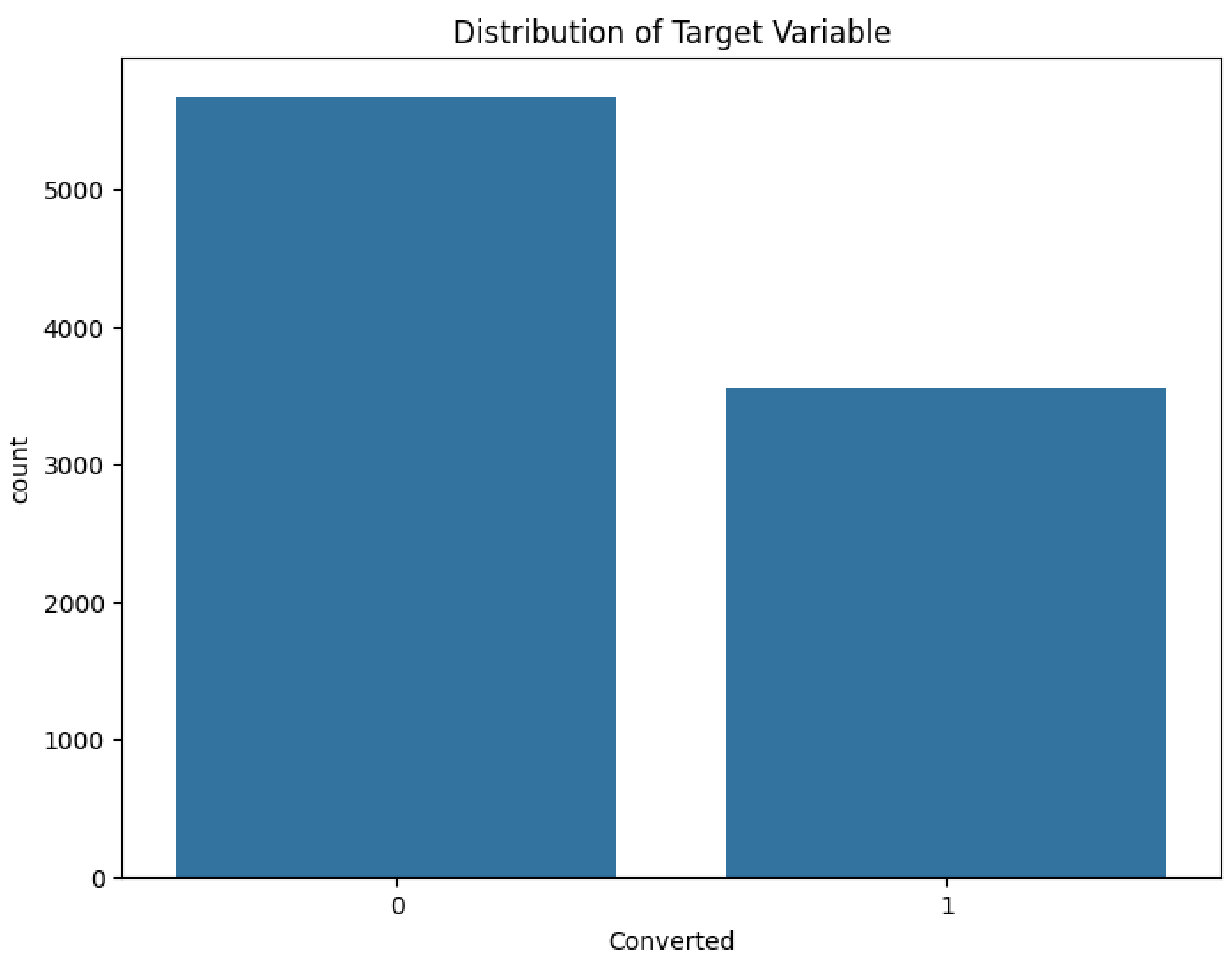
Class Imbalance Visualization

Before further processing, you check how balanced your target variable is:

Code:

```
print("\n Class distribution:")

print(y.value_counts())

sns.countplot(x=y)

plt.title("Class Balance: Converted (0 vs 1)")

plt.show()
```

Prints and plots the number of positive/negative samples (converted = 1 or 0).



The `Preprocessor` class is a crucial component of the machine learning pipeline. Its primary purpose is to take the raw, messy dataset and transform it into a clean, numerical format that machine learning algorithms can understand and learn from.

It achieves this by:

- **Automating Feature Identification**: It automatically detects which columns are numerical and which are categorical.
- **Handling Missing Data**: It systematically fills in missing values using appropriate strategies for different data types.
- **Scaling and Encoding**: It scales numerical features to a common range and converts categorical text data into a meaningful numerical representation.
- **Encapsulation**: It bundles all these steps into a single, reusable object that can be fitted once on the training data and then applied consistently to any new data (like the test set or future live data).

This approach ensures that the same transformations are applied every time, which is critical for preventing data leakage and ensuring the model's reliability.

---

1. Class Structure and Methods

1.1. Initialization (`__init__`)

Code:

Python

```
def __init__(self):
    self.preprocessor = None
    self.numerical_features = []
    self.categorical_features = []
```

-
- **Explanation**: When a `Preprocessor` object is created, it initializes three key attributes:
    - `self.preprocessor`: This will hold the main `ColumnTransformer` object once it's built and fitted. It starts as `None`.

- `self.numerical_features`: An empty list that will later store the names of all numerical columns.
- `self.categorical_features`: An empty list that will later store the names of all categorical columns.

1.2. Fitting the Pipeline (`fit`)

Code:

Python

```python
def fit(self, X, y):
    # ...
```

-
- **Explanation**: The `fit` method is the most critical part of the setup. It learns the necessary transformations from the **training data only**. It takes the feature set `X` and the target variable `y` as input.

  Steps Inside `fit`:
  1. **Identify Feature Types**: It automatically identifies which columns in the training data `X` are numerical (like `TotalVisits`) and which are categorical (like `Lead Source`).
  2. **Define Numerical Transformer**: A `Pipeline` is created for numerical features:
     - `SimpleImputer(strategy='median')`: This step handles missing numerical data by filling `NaN` values with the median of their respective columns. The median is chosen over the mean because it is more robust to outliers.
     - `MinMaxScaler()`: This step scales all numerical features to a range between 0 and 1. This is important because it prevents features with larger scales (like `Total Time Spent on Website`) from dominating the model's learning process.
  3. **Define Categorical Transformer**: A separate `Pipeline` is created for categorical features:
     - `SimpleImputer(strategy='constant', fill_value='Unknown')`: This handles missing categorical data by filling `NaN` values with the string "Unknown".
     - `TargetEncoder(...)`: This is a powerful encoding technique. Instead of just creating binary columns (like One-Hot Encoding), it

encodes each category with the average value of the target variable (`y`). For example, if the "Google" category in `Lead Source` has an average conversion rate of 45%, "Google" will be replaced by a value close to 0.45. This directly embeds the relationship between the category and the conversion outcome into the feature itself, often leading to better model performance.

    - `min_samples_leaf` and `smoothing` are parameters to prevent overfitting and stabilize the encoding for rare categories.

4. **Build and Fit the Preprocessor**: The numerical and categorical transformers are combined into a single `ColumnTransformer`. This object is then fitted on the training data (`X`, `y`). During this "fitting" process, it learns and stores the medians for scaling, the target means for encoding, and all other parameters needed for transformation.

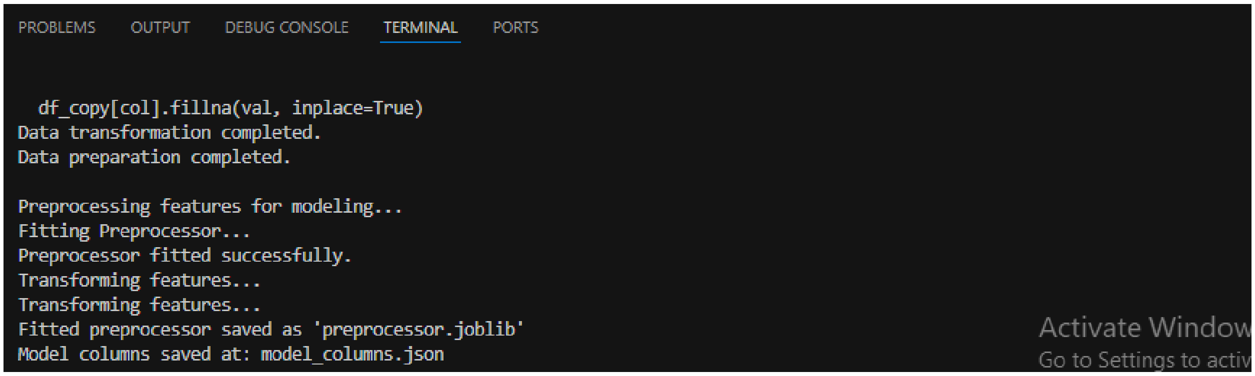
1.3. Transforming Data (`transform`)

Code:

```python
def transform(self, X):
    # ...
```

-
- **Explanation**: The `transform` method applies the transformations that were **learned during the `fit` step** to new, unseen data (`X`).
- **How It Works**:
  1. It first checks if the preprocessor has been fitted. If not, it raises an error.
  2. It then uses the fitted `self.preprocessor` to transform the input DataFrame `X`. It will:
     - Fill missing numerical values with the **medians learned from the training set**.
     - Scale numerical values using the **min/max ranges learned from the training set**.
     - Encode categorical values using the **target means learned from the training set**.
  3. Finally, it converts the transformed data back into a pandas DataFrame, preserving the original index and using the appropriate feature names for better interpretability downstream.

By separating `fit` and `transform`, this class ensures that information from the test set does not "leak" into the training process, which is a fundamental principle of building robust and reliable machine learning models.



The `train_model` function is designed to automate the process of training and evaluating multiple classification algorithms. It systematically identifies the best hyperparameters for each model and logs the entire experiment, including performance metrics and the trained model itself, to MLflow for reproducibility and comparison.

1.. Function Breakdown

- **Model and Parameter Definition**:
    - The function begins by defining a dictionary named `models`. This dictionary contains four different classification algorithms that will be compared: `XGBoost`, `RandomForest`, `LightGBM`, and `LogisticRegression`.
    - This selection provides a good mix of powerful tree-based ensemble methods (XGBoost, RandomForest, LightGBM) and a robust linear model (LogisticRegression), allowing for a comprehensive evaluation of different modeling approaches.
    - For each model, a corresponding set of hyperparameters (`params`) is defined. These are the parameters that will be tuned to find the optimal configuration for each algorithm.
- **Hyperparameter Tuning with `GridSearchCV`**:
    - The function iterates through each model in the `models` dictionary.
    - For each model, it uses `GridSearchCV` to perform an exhaustive search over the specified hyperparameter grid.

- **Cross-Validation Strategy**: `StratifiedKFold(n_splits=3)` is used as the cross-validation strategy. This is a critical choice for this dataset because the target variable (`converted`) is imbalanced. Stratified K-Fold ensures that each fold of the cross-validation has the same proportion of converted and non-converted leads as the original dataset, preventing biased performance estimates.
- **Scoring Metric**: The search is optimized based on the `f1` score. The F1 score is an excellent metric for imbalanced classification problems as it represents the harmonic mean of precision and recall, providing a better measure of the model's accuracy than simple accuracy alone.

- **Handling Class Imbalance**:
  - For the `RandomForest`, `LightGBM`, and `LogisticRegression` models, the `class_weight='balanced'` parameter is used. This instructs the algorithms to automatically adjust the weights of each class to be inversely proportional to their frequency. This means the model will place more importance on correctly classifying the minority class (converted leads), which is essential for this business problem.
- **Performance Evaluation**:
  - After finding the best model through grid search, it is evaluated on the unseen test set (`X_test`, `y_test`).
  - A comprehensive set of classification metrics is calculated:
    - **Accuracy**: The overall percentage of correct predictions.
    - **Precision**: The proportion of predicted conversions that were actually correct. This is important to ensure the sales team's time is not wasted on false positives.
    - **Recall**: The proportion of actual conversions that the model correctly identified. This is crucial to ensure that the model does not miss out on potential customers.
    - **F1 Score**: The balance between precision and recall.
    - **ROC AUC**: A measure of the model's ability to distinguish between the two classes across all possible thresholds.
    - **Log Loss**: A metric that evaluates the performance of a classification model where the prediction input is a probability value between 0 and 1.
- **MLflow Integration**:
  - For each model trained, a new run is started in MLflow using `mlflow.start_run()`.
  - Within each run, the following are logged:

        - The best hyperparameters found by `GridSearchCV` (`mlflow.log_params`).
        - All the calculated performance metrics (`mlflow.log_metrics`).
        - The final, trained model object itself (`mlflow.sklearn.log_model`).
    - This creates a detailed and organized record of every experiment, making it easy to compare model performance and reproduce results in the future.

---

2. `save_and_register_best_model` Function: Final Model Selection and Registration

2.1. Purpose

This function takes the results from the `train_model` function, identifies the single best-performing model, saves it as a local artifact, and registers it in the MLflow Model Registry. This formalizes the model's status as a "production candidate," making it ready for deployment.

2.2. Function Breakdown

- **Identifying the Best Model**:
    - The function first sorts all the model results based on their `f1_score` in descending order.
    - It selects the model with the highest F1 score as the "best model." This ensures that the model chosen for production is the one that performs best on the key business metric for this imbalanced dataset.
- **Saving the Model Locally**:
    - The best model object is serialized and saved to a local file named `best_model.pkl` using `joblib.dump`. This creates a persistent artifact of the trained model that can be easily loaded for inference in a deployment environment.
- **MLflow Model Registration**:
    - This is a critical step in the MLOps lifecycle. The function uses the `MlflowClient` to interact with the MLflow server.
    - A new MLflow run is initiated, specifically for logging and registering the final production candidate model.

    - `mlflow.sklearn.log_model` is called with the `registered_model_name` parameter. This action performs two crucial tasks:
        1. It logs the model as an artifact in the MLflow run.
        2. It creates a new version of the model in the MLflow Model Registry under the specified name (e.g., "XGBoost").
    - The final metrics and hyperparameters for this best model are also logged to this run, creating a clean and definitive record for the production model.
- **Model Staging with Aliases**:
    - To manage the model's lifecycle, MLflow uses aliases (the modern approach, replacing stages).
    - The function uses `client.set_registered_model_alias` to assign the alias `"production"` to the newly registered model version.
    - This step is fundamental for CI/CD (Continuous Integration/Continuous Deployment) pipelines. It programmatically designates this specific model version as the one approved for deployment. Automated deployment scripts can then query the MLflow Model Registry and pull the model version that has the "production" alias, ensuring a smooth and reliable transition from experiment to production.

The `main` function in this script serves as the master controller for the entire machine learning workflow. It is designed to be executed as a single, cohesive process that takes raw data and produces a trained, evaluated, and production-ready model.

The key principles guiding this pipeline are:

- **Modularity**: The pipeline is broken down into logical steps (data loading, cleaning, preprocessing, modeling, monitoring), with each step handled by a dedicated function or class imported from other project files.
- **Reproducibility**: By using `mlflow`, every part of the experiment—from hyperparameters to performance metrics—is logged, ensuring that results can be reliably reproduced.
- **Preventing Data Leakage**: The script carefully splits the data into training and testing sets *before* any cleaning or preprocessing. All transformations are learned from the training data only and then applied to the test data, which is a critical best practice to ensure the model's evaluation is unbiased.

---

2. Pipeline Execution Steps

The `main` function executes the following steps in sequence:

Step 1: Load Data

- **Code**: `df = load_data('Lead Scoring.csv')`
- **Explanation**: This is the first step, where the raw dataset is loaded into a pandas DataFrame from the specified CSV file using the `load_data` function.

Step 2: Data Splitting

- **Code**: `X_train, X_test, y_train, y_test = train_test_split(X, y, ...)`
- **Explanation**: To prevent data leakage, the dataset is immediately split into a training set (80%) and a testing set (20%).
  - `stratify=y` is a crucial parameter used here. Because the dataset has an imbalanced class distribution (more non-converted leads than converted ones), stratification ensures that both the training and testing sets have the same proportion of each class as the original dataset. This guarantees that the model is trained and evaluated on representative data.

Step 3: Data Cleaning

- **Code**: `cleaner = DataCleaner()`, `cleaner.fit(train_df)`, `cleaner.transform(...)`
- **Explanation**: This step uses the `DataCleaner` class to perform initial data tidying.
  - **Fit on Training Data Only**: The `cleaner` is fitted **only on the training data** (`train_df`). This is where it learns the rules for cleaning, such as which columns to drop or how to handle specific placeholder values.
  - **Transform Both Sets**: The fitted `cleaner` is then used to transform both the training and testing sets. This ensures that the exact same cleaning rules are applied consistently to both datasets, maintaining data integrity.

Step 4: Feature Preprocessing

- **Code**: `preprocessor = Preprocessor()`, `preprocessor.fit(...)`, `X_train_processed = preprocessor.transform(...)`
- **Explanation**: This step uses the `Preprocessor` class to convert the cleaned data into a format suitable for machine learning.
    - **Fit on Cleaned Training Data Only**: The `preprocessor` is fitted **only on the cleaned training data**. During this step, it learns the parameters for scaling (e.g., the min/max values for `MinMaxScaler`) and encoding (e.g., the target means for `TargetEncoder`) from the training data alone.
    - **Transform Both Sets**: The fitted `preprocessor` then applies these learned transformations to both the training and testing sets.
    - **Saving Artifacts**:
        - `preprocessor.joblib`: The entire fitted `Preprocessor` object is saved. This is essential because it allows this exact pipeline to be loaded and used later for making predictions on new, live data.
        - `model_columns.json`: The list of feature names that the model expects is saved. This is a best practice for ensuring that the input data for future predictions has the correct structure.

Step 5: Model Training and Evaluation

- **Code**: `results, best_models = train_model(...)`
- **Explanation**: This step calls the `train_model` function, which systematically trains and evaluates multiple machine learning models (e.g., XGBoost, RandomForest) using hyperparameter tuning and cross-validation. It returns a summary of the performance of all models and a dictionary of the best-trained model objects.

Step 6: Save and Register the Best Model

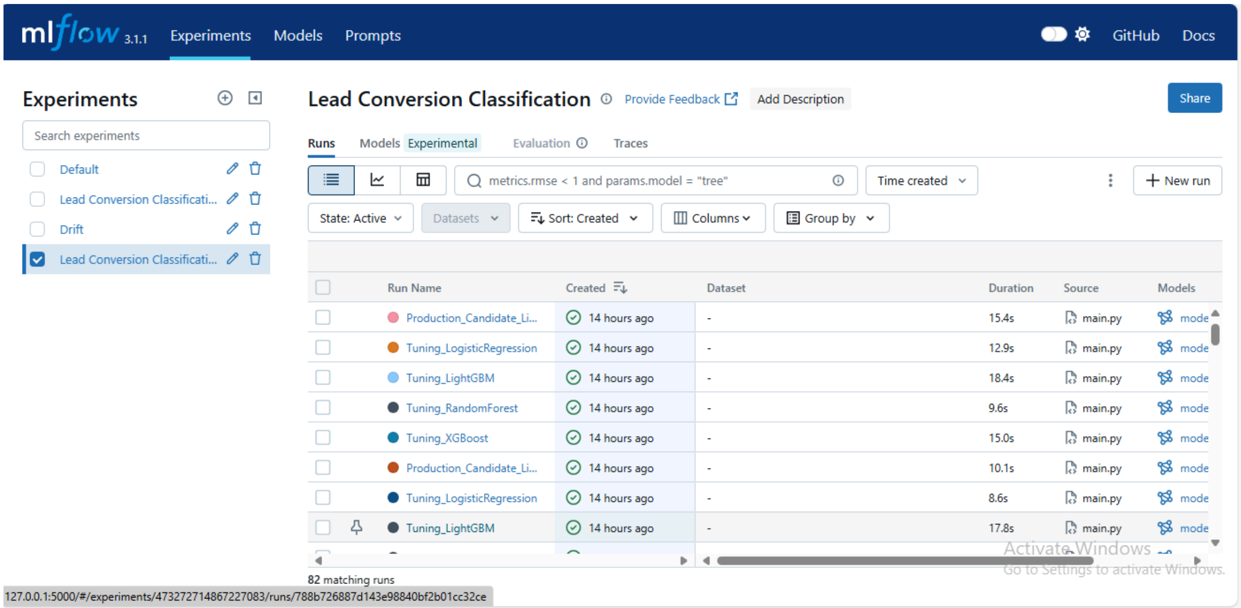
- **Code**: `save_and_register_best_model(results, best_models)`
- **Explanation**: This step takes the results from the previous step, identifies the single best model based on its F1 score, and prepares it for production.
    - It saves the final model object locally as `best_model.pkl`.
    - It registers this model in the **MLflow Model Registry**, creating a new version and assigning it the alias **"production"**. This formally designates the model as the one to be used in the deployment environment.

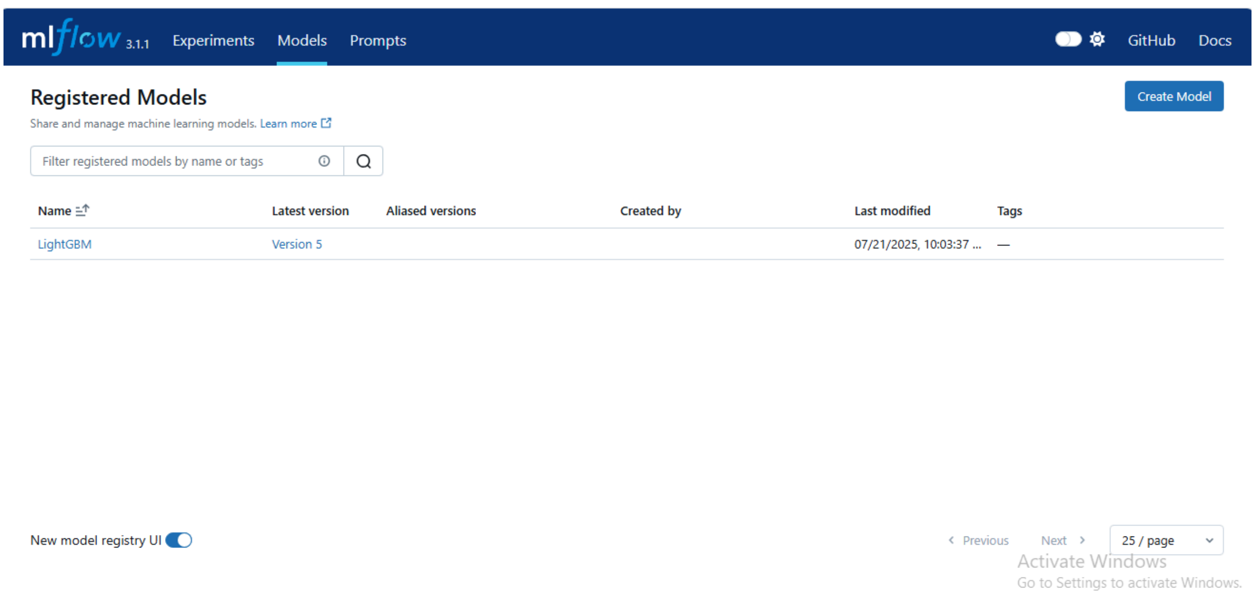
Step 7: Data Drift Monitoring

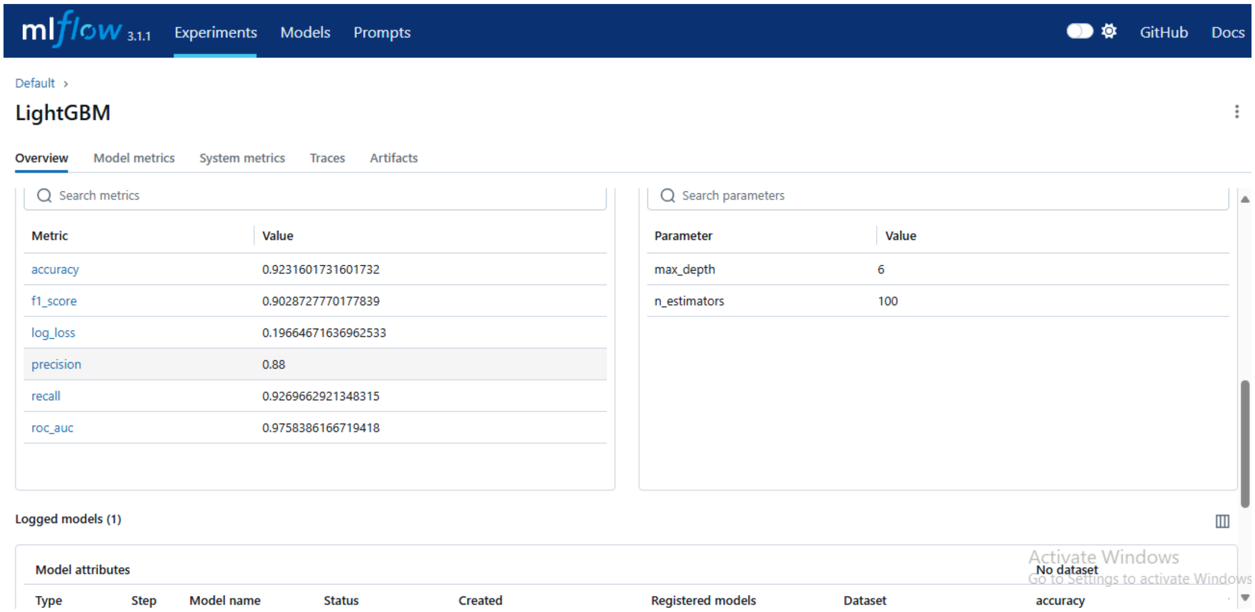
- **Code**: `generate_evidently_report(...)`
- **Explanation**: As a final step in the pipeline, this function uses the **Evidently** library to generate a data drift report.
    - It compares the cleaned training data (`train_df_cleaned`) with the cleaned testing data (`current_data_for_report`).
    - This report provides a detailed analysis of whether the statistical distribution of the features in the test set has shifted compared to the training set. This is a crucial practice for MLOps, as significant data drift can degrade a model's performance over time, signaling a need for retraining.

```
--- All Model Performance Summary ---

  Model: XGBoost
    - Best Params: {'max_depth': 3, 'n_estimators': 200}
    - Accuracy: 0.9199
    - Precision: 0.8961
    - Recall: 0.8961
    - F1 Score: 0.8961
    - Roc Auc: 0.9737
    - Log Loss: 0.2028

  Model: RandomForest
    - Best Params: {'max_depth': None, 'n_estimators': 200}
    - Accuracy: 0.9210
    - Precision: 0.8942
    - Recall: 0.9017
    - F1 Score: 0.8979
    - Roc Auc: 0.9713
    - Log Loss: 0.2404

  Model: LightGBM
    - Best Params: {'max_depth': 6, 'n_estimators': 100}
    - Accuracy: 0.9232
    - Precision: 0.8800
    - Recall: 0.9270
    - F1 Score: 0.9029
    - Roc Auc: 0.9758
    - Log Loss: 0.1966
```

cmd


mlflow 3.1.1
Experiments
Models
Prompts
GitHub
Docs
Experiments
Search experiments
Default
Lead Conversion Classificati...
Drift
Lead Conversion Classificati...
Lead Conversion Classification
Provide Feedback
Add Description
Share
Runs
Models Experimental
Evaluation
Traces
metrics.rmse < 1 and params.model = "tree"
Time created
New run
State: Active
Datasets
Sort: Created
Columns
Group by
Run Name
Created
Dataset
Duration
Source
Models
Production_Candidate_Li... 14 hours ago - 15.4s main.py mode
Tuning_LogisticRegression 14 hours ago - 12.9s main.py mode
Tuning_LightGBM 14 hours ago - 18.4s main.py mode
Tuning_RandomForest 14 hours ago - 9.6s main.py mode
Tuning_XGBoost 14 hours ago - 15.0s main.py mode
Production_Candidate_Li... 14 hours ago - 10.1s main.py mode
Tuning_LogisticRegression 14 hours ago - 8.6s main.py mode
Tuning_LightGBM 14 hours ago - 17.8s main.py mode
82 matching runs
127.0.0.1:5000/#/experiments/473272714867227083/runs/788b726887d143e98840bf2b01cc32ce
Activate Windows
Go to Settings to activate Windows.


mlflow 3.1.1
Experiments
Models
Prompts
GitHub
Docs
Registered Models
Share and manage machine learning models. Learn more
Create Model
Filter registered models by name or tags
Name
Latest version
Aliased versions
Created by
Last modified
Tags
LightGBM Version 5 07/21/2025, 10:03:37 ... —
New model registry UI
Previous
Next
25 / page
Activate Windows
Go to Settings to activate Windows.



This document provides a comprehensive explanation of the Flask web application script. This script is responsible for deploying the trained sales conversion model as a web service, allowing users to get predictions by uploading a CSV file.

---

1. Overview and Purpose

The `app.py` script creates a lightweight web server using **Flask**, a popular Python web framework. Its primary purpose is to serve as the bridge between the trained machine learning model and the end-user.

The application is designed to:

- **Load the ML Model at Startup**: It pre-loads the preprocessing pipeline and the trained model into memory when the application starts, ensuring fast response times for prediction requests.
- **Provide a User Interface**: It serves a simple HTML page (`index.html`) that acts as the front-end for users to interact with.
- **Handle File Uploads**: It provides an API endpoint (`/predict`) that accepts CSV file uploads.

- **Process and Predict**: It takes the uploaded data, processes it using the exact same steps as the training pipeline, feeds it to the model, and generates predictions.
- **Return Results**: It sends the predictions and confidence scores back to the user in a clean, easy-to-read JSON format.

---

2. Application Setup and Initialization

This section covers the initial setup, including loading all necessary machine learning artifacts.

Code:

```python
app = Flask(__name__)
logging.basicConfig(...)

try:
    preprocessor = joblib.load("preprocessor.joblib")
    model = joblib.load("saved_models/best_model.pkl")
    with open('model_columns.json', 'r') as f:
        model_columns = json.load(f)
except Exception as e:
    # ... error logging
```

-
- **Explanation**:
    1. **Flask App Creation**: A new Flask application instance is created.
    2. **Logging Configuration**: `logging` is configured to provide detailed, timestamped messages. This is crucial for debugging issues in a production environment.
    3. **Loading Artifacts at Startup**: The script immediately tries to load three essential files:
        - `preprocessor.joblib`: The **fitted preprocessing pipeline**. This is critical because it contains all the learned parameters (medians for imputation, scaling ranges, etc.) from the training data.
        - `best_model.pkl`: The final, trained machine learning model object.
        - `model_columns.json`: A list of the exact feature columns (and their order) that the model was trained on.

4. **Error Handling**: A `try...except` block wraps the file loading. If any of these essential files are missing, the application will log a fatal error and will not be able to serve predictions, preventing crashes during a user request.

---

3. API Endpoints

The application exposes two main routes (URL endpoints).

3.1. Home Route (`/`)

Code:

Python

```
@app.route("/")
def home():
    return render_template("index.html")
```

*
* **Explanation**: This is the main landing page of the application. When a user navigates to the root URL, this function renders and returns the `index.html` file, which contains the user interface for uploading a file.

3.2. Predict Route (`/predict`)

Code:

Python

```
@app.route("/predict", methods=["POST"])
def predict():
    # ...
```

*
* **Explanation**: This is the core endpoint that handles the machine learning logic. It only accepts `POST` requests, which is the standard method for sending data (like a file) to a server.

  Workflow Inside `predict`:

  1. **Server Health Check**: It first checks if the `preprocessor`, `model`, and `model_columns` were loaded correctly at startup. If not, it returns a `500 Internal Server Error`.
  2. **File Validation**: It validates the incoming request to ensure a CSV file was actually uploaded. If not, it returns a `400 Bad Request` error.

3. **Data Reading and Cleaning**: The uploaded CSV is read into a pandas DataFrame. The `clean_col_names` utility function is called to standardize the column names (e.g., "Lead Source" becomes "lead_source"), ensuring they match the format used during training.
4. **Data Validation and Alignment**: This is a critical step for robustness.
   - It compares the columns in the uploaded CSV with the `model_columns` list.
   - If the uploaded file is **missing** any columns the model expects, it adds them and fills them with `NaN` (missing values). This prevents the pipeline from breaking if the input data is incomplete.
   - It then reorders the columns of the uploaded data (`df_aligned`) to **exactly match the order** of `model_columns`. This is essential, as scikit-learn pipelines are sensitive to column order.
5. **Preprocessing and Prediction**:
   - The aligned data is passed to the `preprocessor.transform()` method, which applies all the learned scaling and encoding rules.
   - The resulting processed data is fed into `model.predict()` to get the final classification (Convert/Not Convert) and `model.predict_proba()` to get the confidence score (probability of conversion).
6. **Formatting the Response**:
   - The predictions and confidence scores are added as new columns to the original uploaded DataFrame, making the output easy for the user to understand.
   - A key correction is made: `df_uploaded.replace({np.nan: None})`. `NaN` values are not valid in JSON and can cause errors in web browsers. This line replaces them with `None` (which becomes `null` in JSON), ensuring a valid response.
   - The final DataFrame is converted to a JSON object and sent back to the user with a `200 OK` status.

---

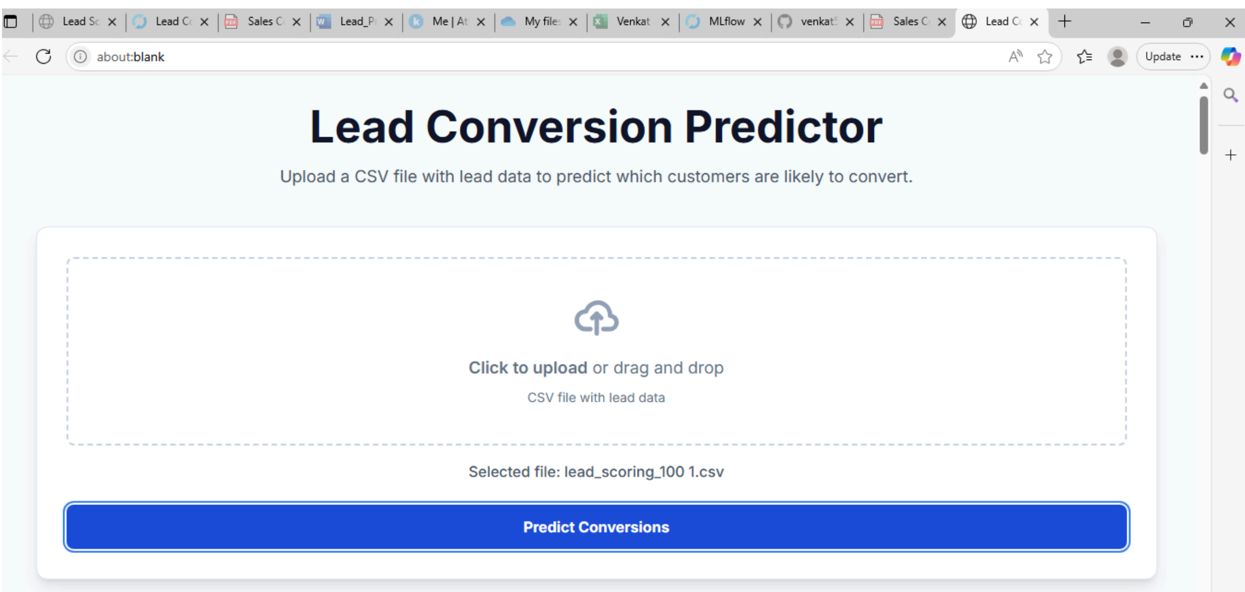
4. Running the Application

Code:

Python

```
if __name__ == "__main__":
    port = int(os.environ.get("PORT", 8080))
    app.run(host="0.0.0.0", port=port, debug=True)
```

-
- **Explanation**:
    - `if __name__ == "__main__":` ensures that the web server only runs when the script is executed directly.
    - `host="0.0.0.0"` makes the application accessible on the local network, not just on the local machine.
    - `port=8080` sets the port the application will run on. Using an environment variable `PORT` makes it flexible for deployment on various cloud platforms.
    - `debug=True` runs the application in debug mode, which provides detailed error messages and automatically restarts the server when code changes are made. **This should be set to `False` in a production environment.**



Analyzed **100** leads. Predicted **49** will convert.

PROSPECT ID	LEAD SOURCE	TOTAL TIME SPENT ON WEBSITE	PREDICTION	CONFIDENCE
7927b2df-8bba-4d29-b9a2-b6e0beafe620	Olark Chat	0	Not Convert	1.24%
2a272436-5132-4136-86fa-dcc88c88f482	Organic Search	674	Not Convert	11.33%
8cc8c611-a219-4f35-ad23-fdfd2656bd8a	Direct Traffic	1532	Convert	99.06%
0cc2df48-7cf4-4e39-9de9-19797f9b38cc	Direct Traffic	305	Not Convert	1.94%
3256f628-e534-4826-9d63-4a8b88782852	Google	1428	Convert	78.91%
2058ef08-2858-443e-a01f-a9237db2f5ce	Olark Chat	0	Not Convert	1.25%
9fae7df4-169d-489b-afe4-0f3d752542ed	Google	1640	Convert	99.09%
20ef72a2-fb3b-45e0-924e-551c5fa59095	Olark Chat	0	Not Convert	21.59%
cfa0128c-a0da-4656-9d47-0aa4e67bf690	Direct Traffic	71	Convert	87.44%
af465dfc-7204-4130-9e05-33231863c4b5	Google	58	Convert	83.95%