EMPLOYEE ATTRITION ANALYSIS

PRESENTED BY:

GROUP NO.4

- KARTHAVYA BHAT-24030242027
- NAKSHATRA ZAWARE- 24030242036
- SHANIYA JOSEPH- 24030242058

PREDICT ATTRITION

Step 1: Exploratory Data Analysis (EDA) & Preprocessing

- Our initial analysis revealed that the dataset was highly imbalanced, with significantly fewer employees who had left ('Yes') compared to those who had not.
- To address this, we used the SMOTE (Synthetic Minority Over-sampling Technique) to balance the training data by generating synthetic samples of the minority class.

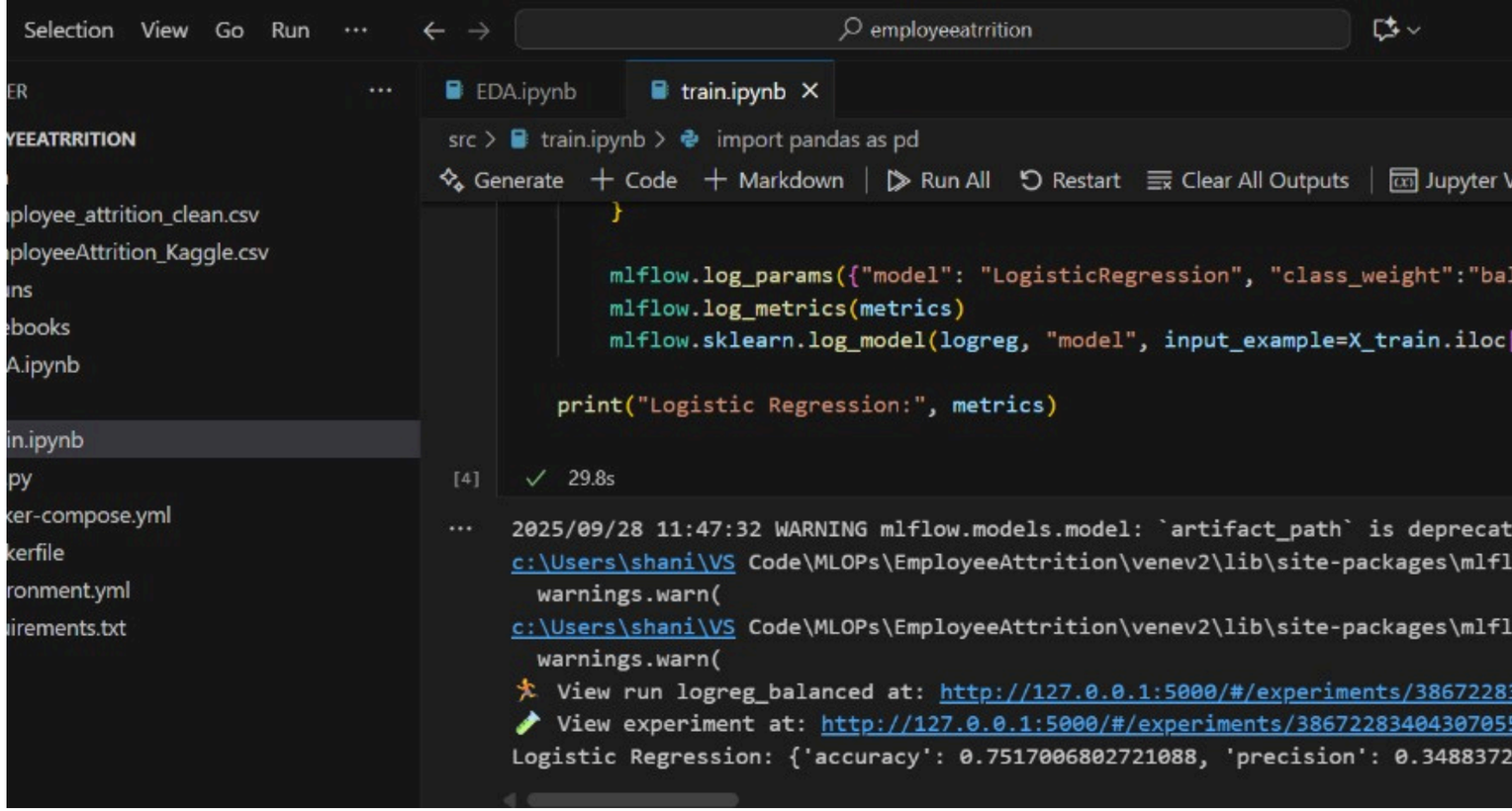
Step 2: Model Experimentation

- We trained and evaluated three different classification models to find the best fit for our problem:

i. XGBoost
ii. Random Forest
iii. Logistic Regression

Step 3: Final Model Selection

- After comparing their performance, we selected Logistic Regression as our final model..
- Justification: For this use case, correctly identifying employees who might leave is the top priority. Logistic Regression gave us the best Recall score, which means it was the most effective at finding the highest percentage of actual leavers.



Step 4: Systematic Tracking with MLflow

- As shown in the code, the training process for our chosen model was integrated with MLflow.
- We logged crucial information for the run:
- Parameters: The model's configuration (class_weight='balanced').
- Metrics: The model's performance (Accuracy, Precision, Recall).
- Model Artifact: The final, saved model file ready for deployment.

Step 5: Results & Evaluation

- The terminal output shows the final metrics for our selected Logistic Regression model. The links direct us to the MLflow UI, ensuring this final experiment is fully documented and reproducible.

LOG EXPERIMENTS IN MLFLOW

Step 1: Initialize the Experiment Run

- Before training, we start a new MLflow run.
- This acts as a "digital lab notebook" for a single experiment, creating a unique ID to store everything related to this specific training session.

Step 2: Log Model Parameters

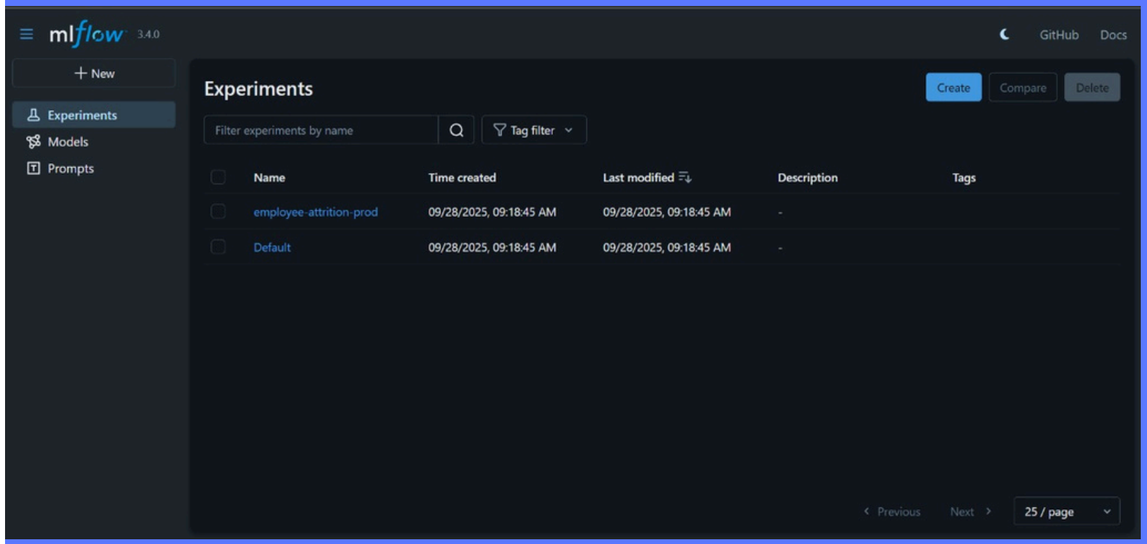
- We use the mlflow.log_params() function to save the model's configuration.
- As shown in the code, this includes key settings like the model type (LogisticRegression) and hyperparameters like class_weight. This ensures we can perfectly recreate the experimental setup later.

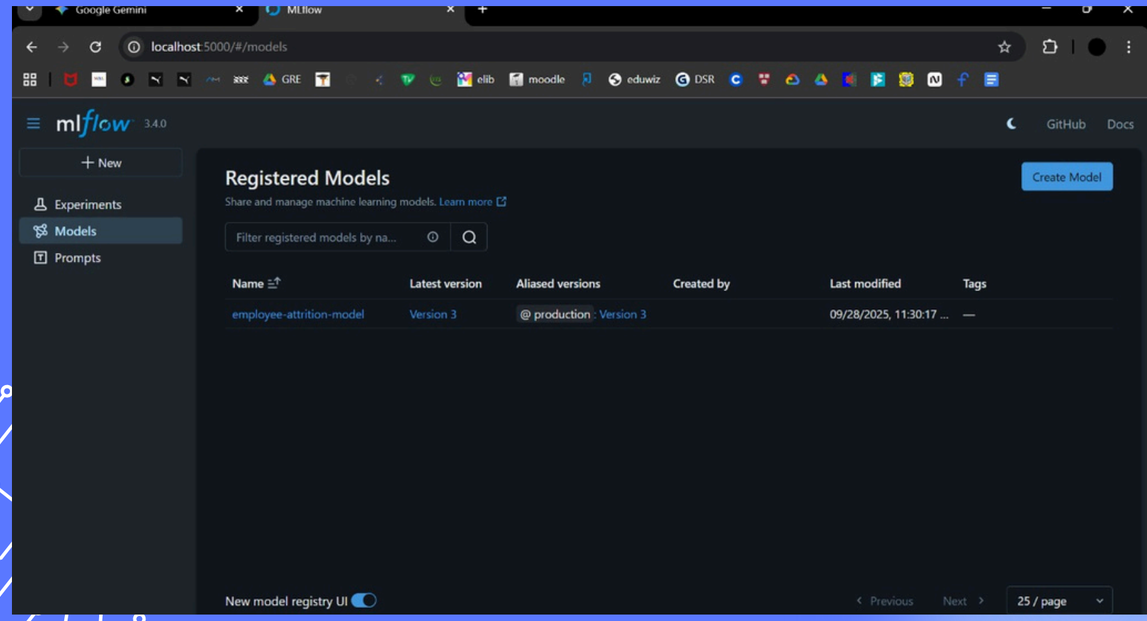
Step 3: Log Performance Metrics

- After the model makes predictions, we use mlflow.log_metrics() to record its performance.
- This captures the results, such as accuracy, precision, and recall, allowing us to quantitatively compare different models.

Step 4: Log the Model as an Artifact

- Finally, mlflow.sklearn.log_model() packages and saves the trained model file itself.
- This "artifact" is the most important output, as it's the file we will later deploy to make real predictions. The terminal output provides a direct link to this logged run for easy access.





Step 1: Define the Core Environment

- We created an environment.yml file to serve as the single source of truth for our project's software requirements.
- We first defined a unique name for our Conda environment (employee_attrition_env) and locked the python version to 3.10 for consistency.

Step 2: Specify All Dependencies

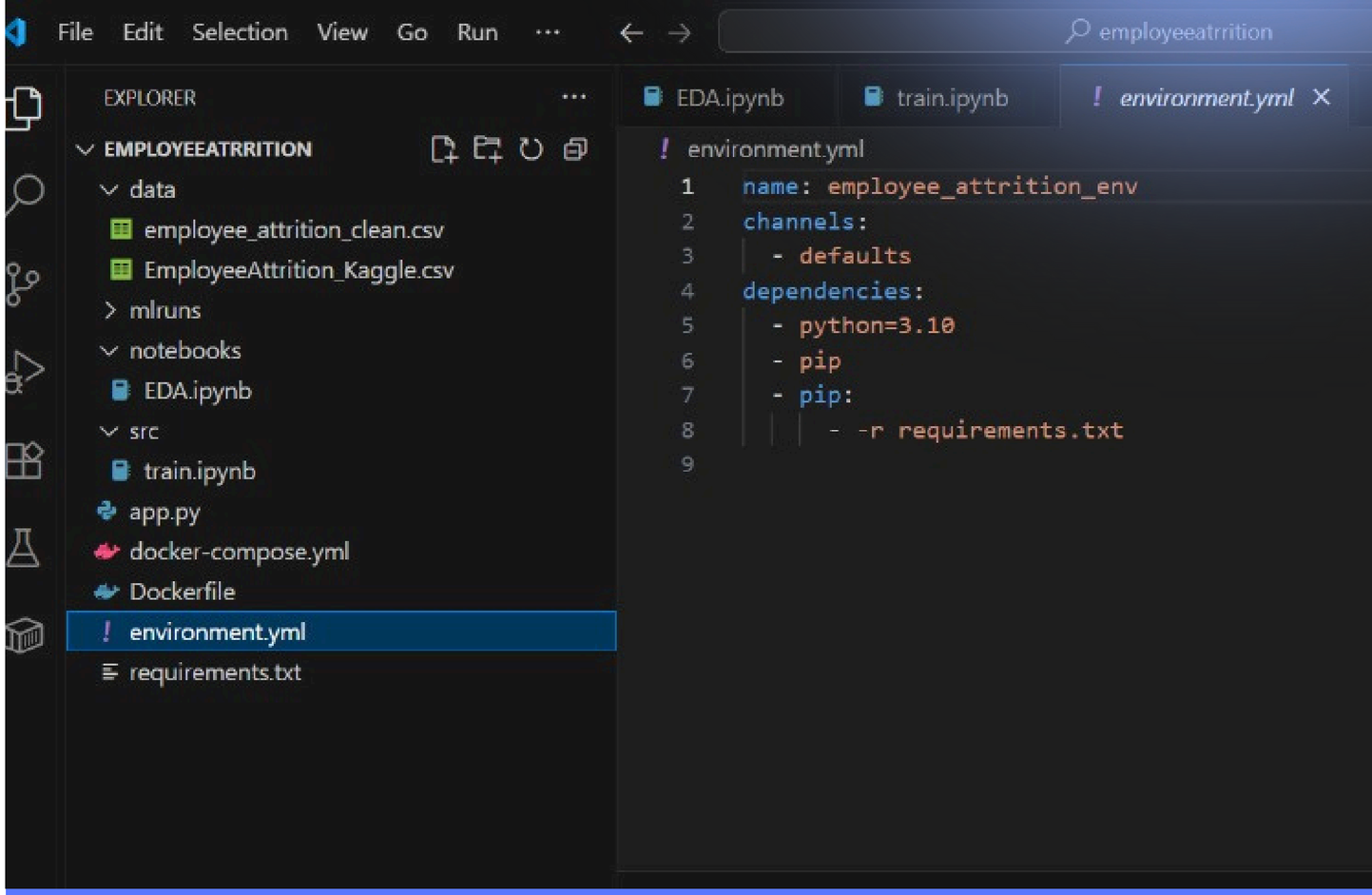
- All required libraries for the project were listed under the dependencies section.
- For better organization, we used a pip command to install all Python packages from a separate requirements.txt file.

Step 3: Create the Local Environment

- This file allows any team member to perfectly replicate the development environment with a single command: conda env create -f environment.yml.
- This creates an isolated workspace, ensuring that the code works consistently for everyone involved in the project.

Step 4: Automate Builds for Deployment

- The most critical role of this file is in automation. Our Dockerfile uses this exact file to build the container for production.
- This guarantees that the environment in the deployed container is an identical match to our development environment, ensuring a reliable and reproducible deployment.



PACKAGE WITH ENVIRONMENT .YML

DOCKERIZE

Step 1: Prepare Your Project Directory

- Before you begin, ensure your project folder has the following structure. The Dockerfile and your model artifact folder (which you can download from MLflow) must be in the same directory.

Step 2: Create the Dockerfile

- Create a file named Dockerfile (with no extension) and add the following code. This file contains the instructions for building your image.

Step 3: Build the Docker Image

- Open your terminal in the employee-attrition-project directory and run the docker build command. This command reads the Dockerfile and creates your container image.
- The -t flag tags your image with a name (employeeattrition-api) and version (latest).
- The . at the end specifies that the current directory is the build context.
- After it completes, you can see your new image by running docker images.

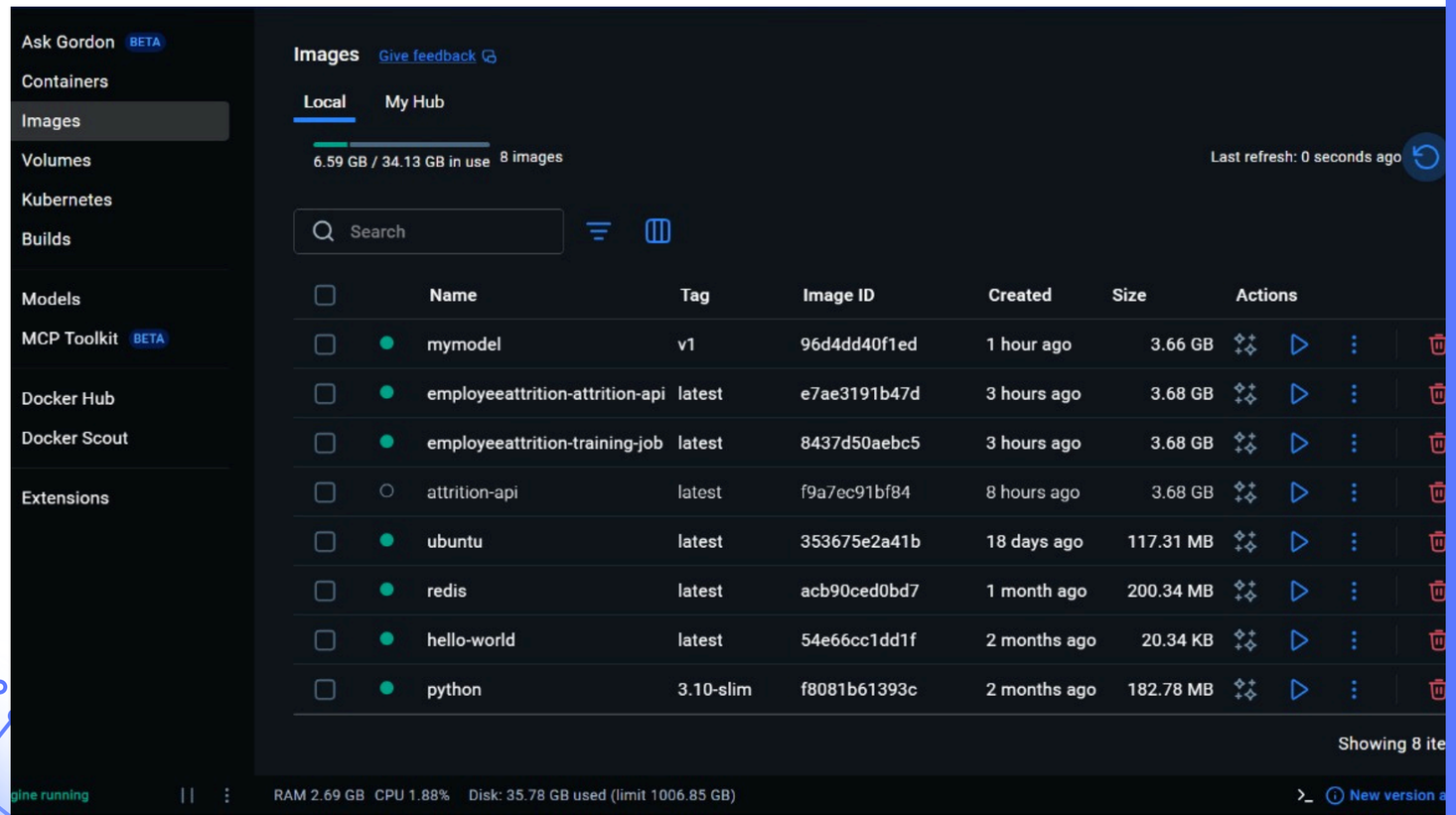
Step 4: Run the Docker Container

- Now, run the image you just built. This command starts a container from your image.
- The -p 8000:8000 flag maps your computer's port 8000 to the container's port 8080.
- The -d flag runs the container in the background (detached mode).

You can see your running container with the docker ps command.

Step 5: Test the API Endpoint

- Finally, send a request to the running container to verify that the model is serving predictions correctly. You can use the curl command for this.



Step 1: Build a User-Friendly Interface

- To make our model usable, we developed a clean and simple web user interface.
- This front-end application was built using FastAPI, which communicates directly with our deployed machine learning model.

Step 2: Input Employee Information

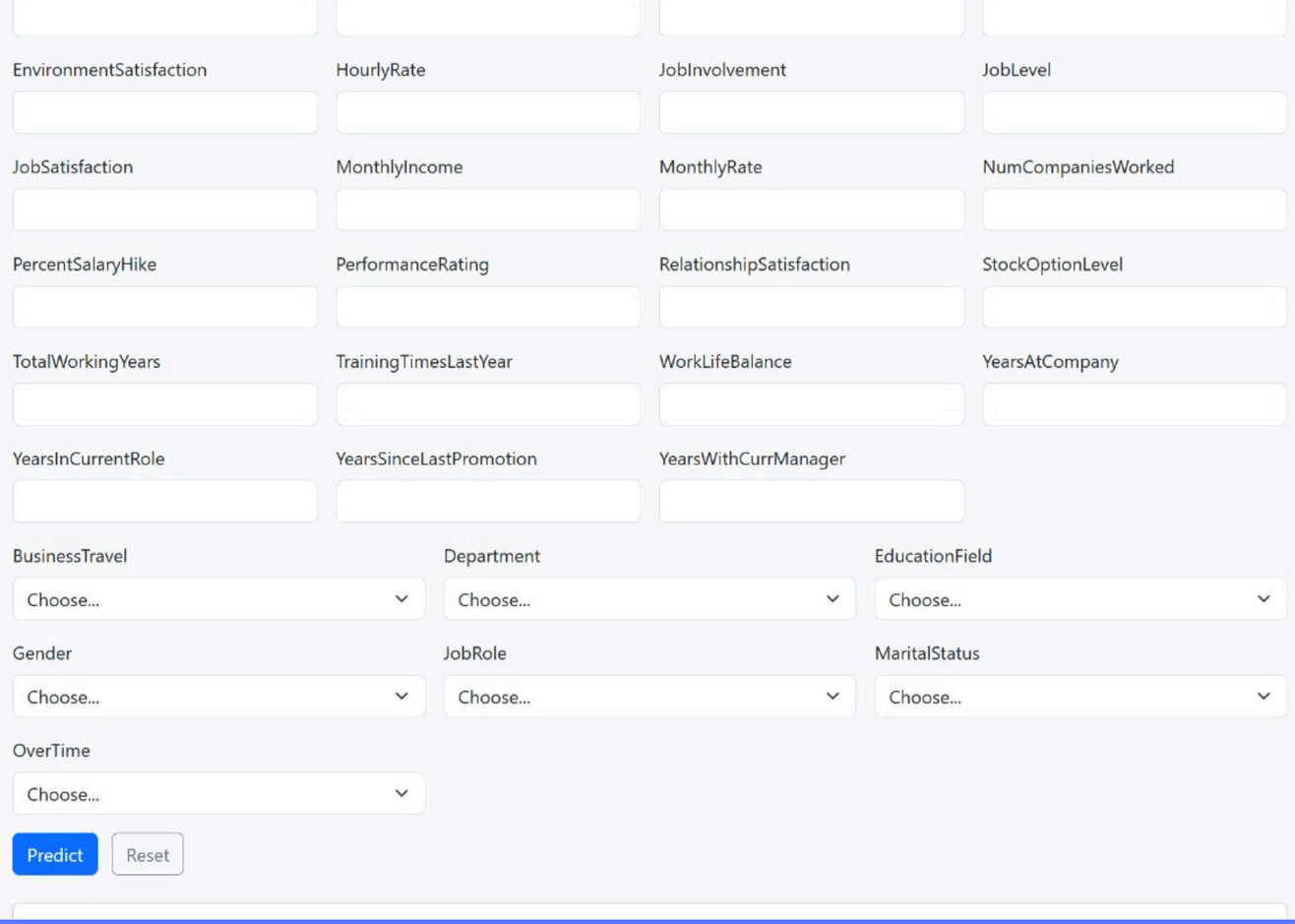
- The UI provides an intuitive form where a user, such as an HR manager, can enter the details of an employee.
- It includes fields for all the necessary features our model uses, including numerical data like MonthlyIncome and categorical data like JobRole.

Step 3: Request a Real-Time Prediction

- Once the form is filled, the user clicks the "Predict" button.
- This action sends the employee's data to our model's API endpoint, which processes the information in real-time.

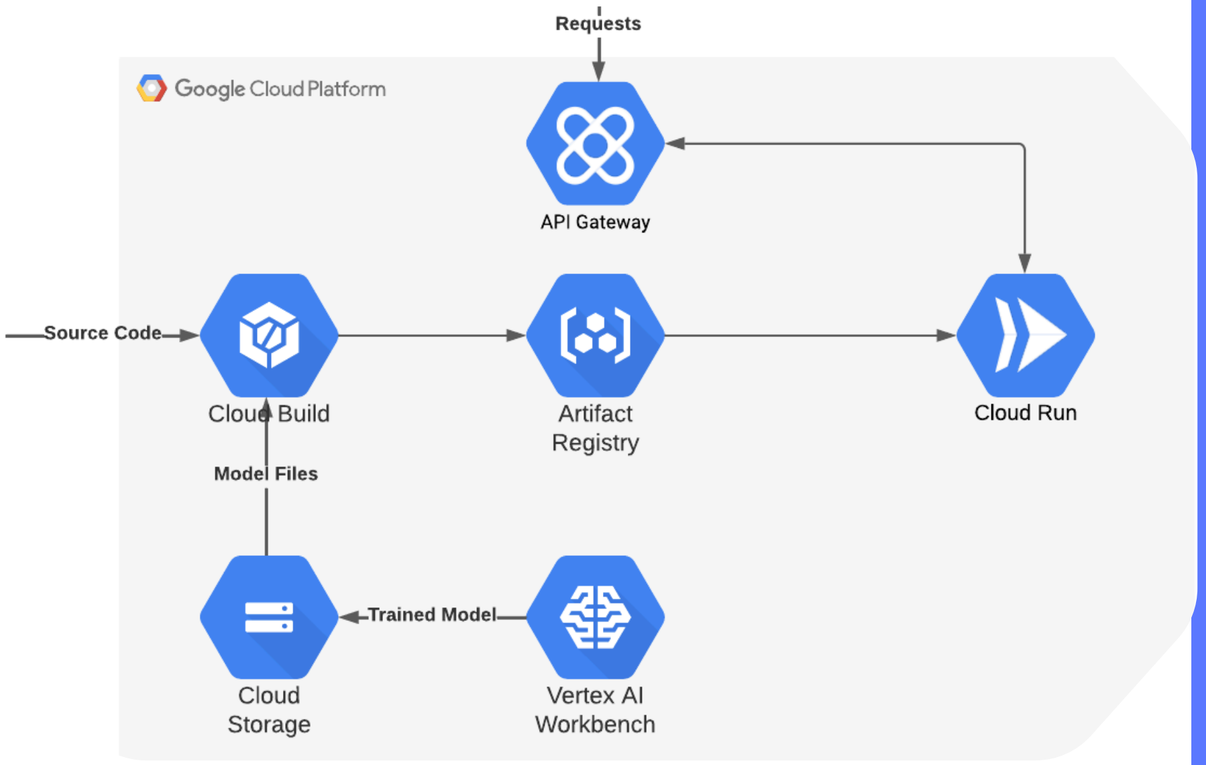
Step 4: Receive and Display the Outcome

- The model returns a prediction—whether the employee is likely to leave or not.
- This outcome is then displayed instantly in the "Result" box at the bottom of the page, turning our MLOps pipeline into a practical, interactive decision-support tool.



UI OF EMPLOYEE ATTRITION PREDICTION

DEPLOY TO GCP AI PLATFORM



Deploying the Model with Google Cloud Vertex AI:

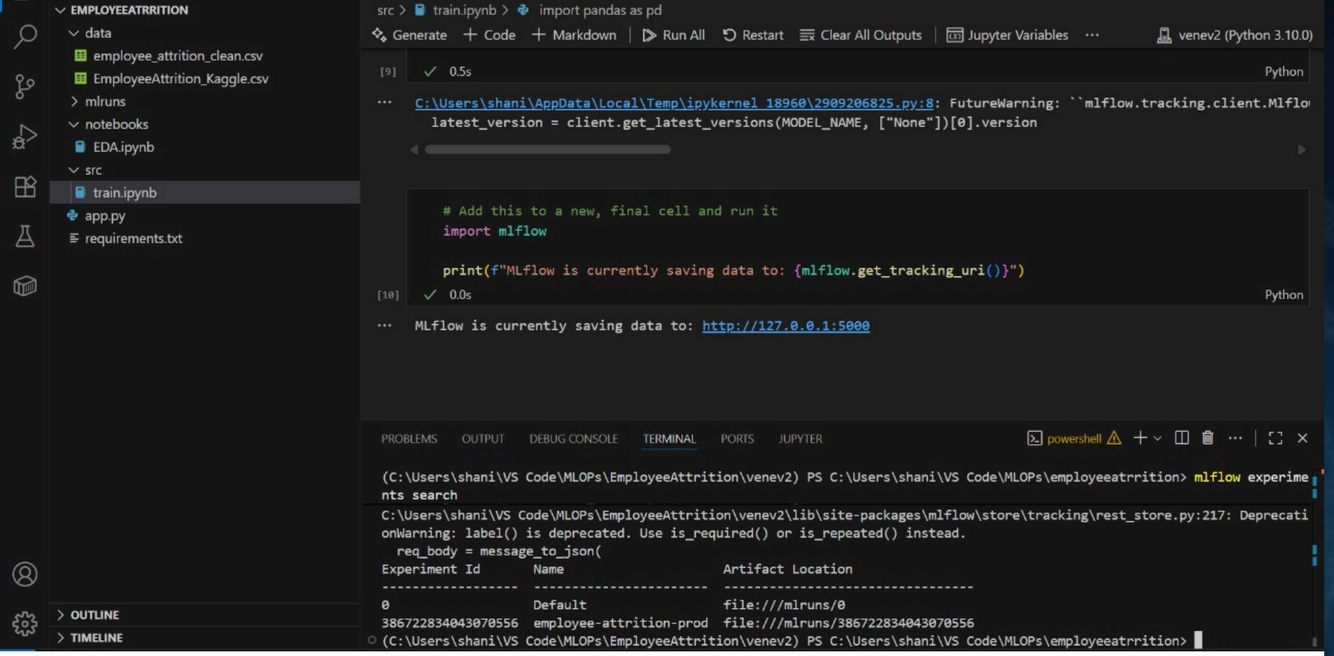
Our goal is to take the containerized model and launch it as a live, scalable API.

1. **Push Container to Google Artifact Registry -**
   - First, we build the Docker image and upload it to Google's private container storage.
2. **Deploy on Vertex AI -**
   - In the Google Cloud Console, we use Vertex AI to host our container.
   - Actions:

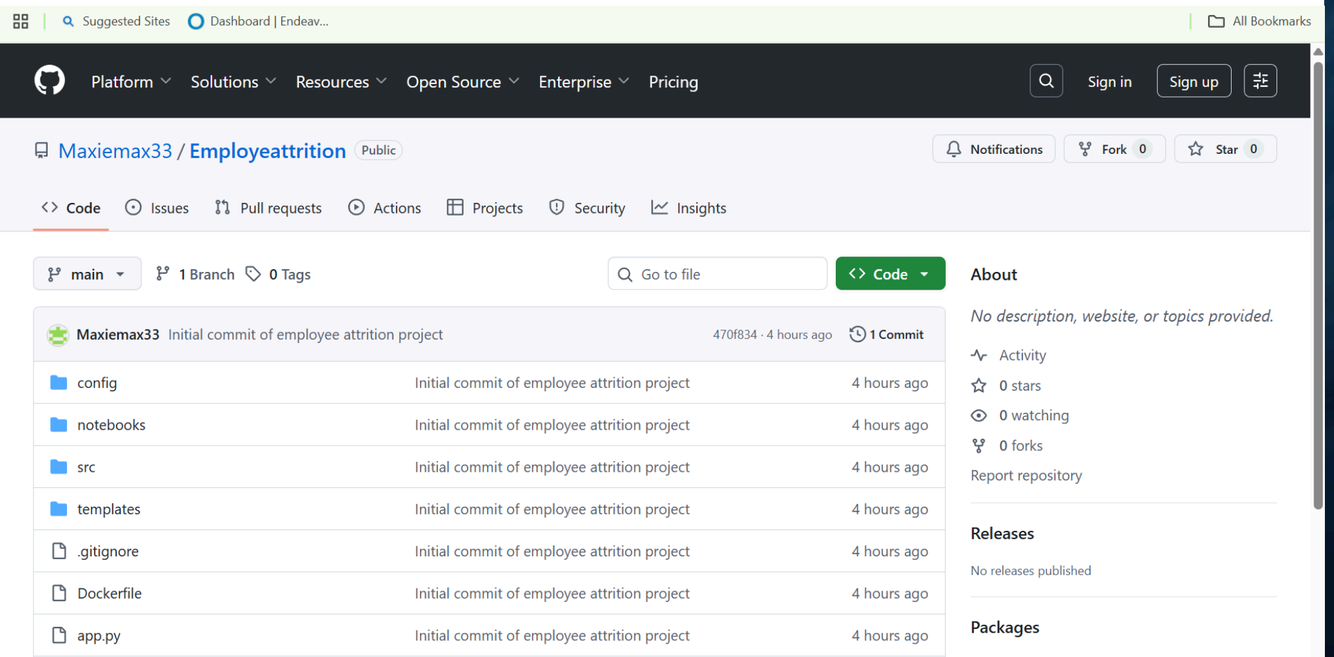
i. Import Model: Point Vertex AI to the container image in Artifact Registry.

ii. Create Endpoint: Deploy the model to a public URL, which handles servers and scaling automatically.

3. **Test the Live API Endpoint**
   - We send new data to the live URL and get a prediction back.


EMPLOYEEATTRITION
data
employee_attrition_clean.csv
EmployeeAttrition_Kaggle.csv
mlruns
notebooks
EDA.ipynb
src
train.ipynb
app.py
requirements.txt
src > train.ipynb > import pandas as pd
Generate + Code + Markdown | Run All | Restart | Clear All Outputs | Jupyter Variables | venev2 (Python 3.10.0)
[9] 0.5s Python
C:\Users\shani\AppData\Local\Temp\ipykernel_18960\2909206825.py:8: FutureWarning: ``mlflow.tracking.client.Mlflo
latest_version = client.get_latest_versions(MODEL_NAME, ["None"])[0].version
# Add this to a new, final cell and run it
import mlflow
print(f"MLflow is currently saving data to: {mlflow.get_tracking_uri()}")
[10] 0.0s Python
MLflow is currently saving data to: http://127.0.0.1:5000
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
powershell
(C:\Users\shani\VS Code\MLOPs\EmployeeAttrition\venev2) PS C:\Users\shani\VS Code\MLOPs\employeeatrrition> mlflow experiments search
C:\Users\shani\VS Code\MLOPs\EmployeeAttrition\venev2\lib\site-packages\mlflow\store\tracking\rest_store.py:217: DeprecationWarning: label() is deprecated. Use is_required() or is_repeated() instead.
req_body = message_to_json(
Experiment Id Name Artifact Location
0 Default file:///mlruns/0
386722834043070556 employee-attrition-prod file:///mlruns/386722834043070556
(C:\Users\shani\VS Code\MLOPs\EmployeeAttrition\venev2) PS C:\Users\shani\VS Code\MLOPs\employeeatrrition>
OUTLINE
TIMELINE

Github link


Suggested Sites
Dashboard | Endeav...
All Bookmarks
Platform Solutions Resources Open Source Enterprise Pricing
Sign in
Sign up
Maxiemax33 / Employeattrition
Public
Notifications
Fork 0
Star 0
Code Issues Pull requests Actions Projects Security Insights
main
1 Branch
0 Tags
Go to file
Code
Maxiemax33 Initial commit of employee attrition project
470f834 · 4 hours ago
1 Commit
config Initial commit of employee attrition project 4 hours ago
notebooks Initial commit of employee attrition project 4 hours ago
src Initial commit of employee attrition project 4 hours ago
templates Initial commit of employee attrition project 4 hours ago
.gitignore Initial commit of employee attrition project 4 hours ago
Dockerfile Initial commit of employee attrition project 4 hours ago
app.py Initial commit of employee attrition project 4 hours ago
About
No description, website, or topics provided.
Activity
0 stars
0 watching
0 forks
Report repository
Releases
No releases published
Packages


THANK
YOU