MESINKIRA

ML/AI ENGINEER

ASSESSMENT

2025

GitHub Repository: Vishean's MesinKira Assessment

Dataset: Amazon Fine Food Reviews - Kaggle

13/02/2025

CONTENTS

Tasks	Status	Notes
S3 Bucket – Raw Data	Successfully stored.	https://mkar.s3.ap-southeast-1.amazonaws.com/raw_reviews/
IAM – Roles and User	Allocated Permissions.	• S3 Read and write. • AWS Glue Full access and S3 interaction. • AWS Lambda – Load model, invoke API Gateway and access S3.
AWS Glue – PySpark – Preprocess	Successful. Enabled Job bookmark (only preprocess new data).	s3://aws-glue-assets-257394483869-ap-southeast-1/scripts/
S3 Bucket – Processed Data	Successfully stored.	https://mkar.s3.ap-southeast-1.amazonaws.com/processed_data/
AWS Glue – PySpark – Model Training	Successful	s3://aws-glue-assets-257394483869-ap-southeast-1/scripts/
S3 Bucket – Models	Successfully stored.	https://mkar.s3.ap-southeast-1.amazonaws.com/models/
AWS Glue Workflow	Successful Job completion.	Set Trigger – Preprocess then training.
AWS Lambda Function	Load model, accept API request, return prediction	Attached Inline Policy. Encountered error during test run.
API Gateway	HTTP API	Invoke URL tested – Unsuccessful.

PART A

AWS Machine Learning Pipeline.

Introduction

This report details the implementation of an end-to-end AWS Glue pipeline for sentiment analysis. The objective was to automate data preprocessing, model training, and model deployment using AWS Glue workflows, triggers, and S3 storage in preparation for deployment using API Gateway and AWS Lambda.

The project successfully:

- Processed raw review data from Amazon S3.
- Trained a sentiment analysis model using Logistic Regression with TF-IDF vectorization.
- Saved the trained model and vectorizer to AWS S3 for inference.
- Automated the entire workflow using AWS Glue Workflows and Triggers.

Workflow Overview

The AWS Glue pipeline consists of:

1. AWS Glue Preprocessing Job – Cleans and processes raw review data.
2. AWS Glue Model Training Job – Trains a machine learning model on processed data.
3. AWS Glue Workflow – Orchestrates the execution of Glue jobs.
4. AWS Glue Triggers – Ensures sequential execution from preprocessing to model training.

S3 Bucket Structure

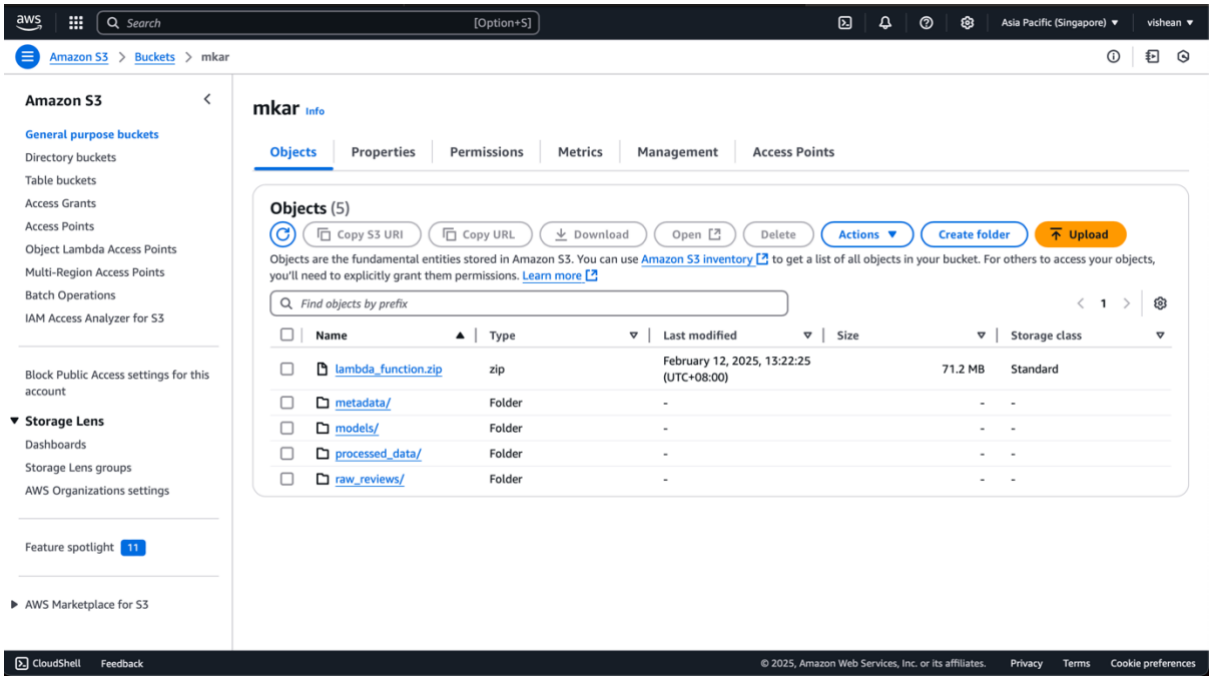

Figure 1. S3 Bucket contents.

Implementation Details

AWS Glue Preprocessing Job

- Reads raw reviews from s3://mkar/raw_reviews/.
- Cleans data by filtering necessary columns (Text, Score).
- Ensures "Score" is always an integer and removes invalid data.
- Saves cleaned data in s3://mkar/processed_data/.
- Uses a checkpoint file (processed_files.txt) to prevent reprocessing old files.

AWS Glue Model Training Job

- Loads processed data from s3://mkar/processed_data/.
- Splits data into training and testing sets.
- Applies TF-IDF vectorization to transform text data.
- Trains a Logistic Regression model.
- Saves the trained model and vectorizer to s3://mkar/models/.

Workflow Execution Flow

1. Preprocessing Job (raw_preprocess)
    - This job is responsible for cleaning and preparing the dataset for model training.
    - It serves as the initial step in the workflow.
2. Trigger (train_model_trigger)
    - This is the only trigger in the workflow, positioned between preprocessing and training.

    - It activates the training job only upon successful completion of the preprocessing job.
3. Training Job (train_model)
    - This job trains the sentiment analysis model using the preprocessed data.
    - Once triggered, it runs independently to produce the final trained model.

Trigger & Dependency Management

- The workflow relies on a single trigger (train_model_trigger), ensuring sequential execution of the two jobs.
- The preprocessing job does not require an explicit trigger; it initiates as part of the workflow execution.
- The transition from preprocessing to training is managed via the trigger, ensuring data dependencies are met before training begins.

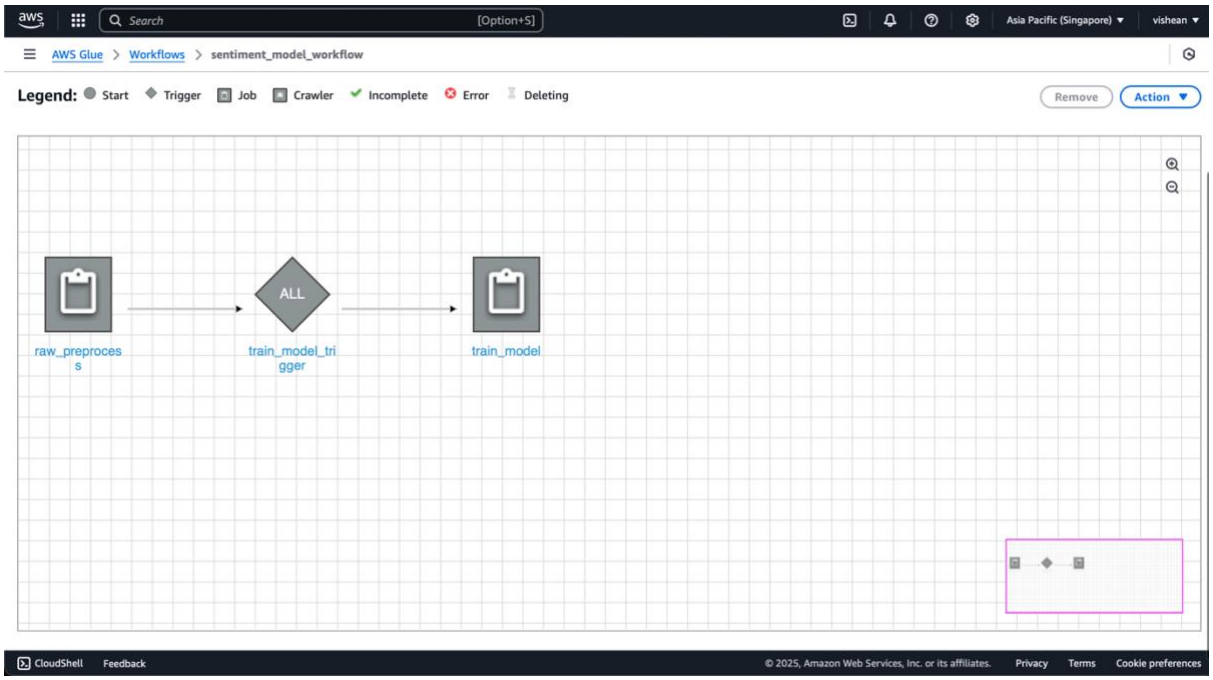

Figure 2. Gue Workflow containing 1 trigger.

API Gateway and Lambda Function

Step 1: Automating Data Preprocessing & Model Training

AWS Glue was used to preprocess data and train a sentiment analysis model. The trained model (sentiment_model.pkl) and vectorizer (vectorizer.pkl) were stored in an S3 bucket (mkar/models/).

Step 2: Deploying a Lambda Function

AWS Lambda function (sentiment_analysis_api) was created to:

- Load the trained model from S3.
- Accept an API request with text input.
- Return a sentiment prediction as a response.

Permissions and IAM Roles Involved:

- IAM Role: sentiment_analysis_api-role-5a6u756l was assigned to the Lambda function.
- IAM Policy Attached:
    - AmazonS3ReadOnlyAccess (Allows Lambda to read from S3).
    - Custom inline policy to allow explicit access to mkar/models/:
    - {
    - "Version": "2012-10-17",
    - "Statement": [
    - {
    - "Effect": "Allow",
    - "Action": ["s3:GetObject", "s3:ListBucket"],
    - "Resource": [
    - "arn:aws:s3:::mkar",
    - "arn:aws:s3:::mkar/models/*"
    -]
    - }
    -]
    }

Step 3: Setting Up API Gateway

API Gateway was set up with an HTTP API (instead of REST API) to serve predictions. The API endpoint was exposed at:

https://wb1dpjh1hd.execute-api.ap-southeast-1.amazonaws.com/v1/predict

The API Gateway was integrated with the Lambda function to process POST requests.

Building React App

Outline of Work Done to Create and Deploy the React App

1. Setting Up the React Project

- Installed Node.js to enable running JavaScript outside the browser.
- Initialized a React project using create-react-app and configured necessary dependencies.
- Installed Webpack and Babel manually due to a mixed setup.
- Ensured all required packages, including React, ReactDOM, Webpack, Babel, and necessary loaders, were installed.

2. Configuring and Running the Development Server

- Created the index.js file in the src folder, ensuring it correctly renders the SentimentAnalysis component.
- Modified the React rendering logic to use React 18+ syntax with createRoot() instead of the deprecated ReactDOM.render().
- Ensured the public/index.html file contained a to mount the React app.
- Ran the development server using npm start and resolved multiple issues related to package dependencies.

3. Debugging and Fixing Critical Errors

- Identified missing files, such as SentimentAnalysis.jsx, and ensured proper imports.
- Fixed Webpack errors by correctly configuring webpack.config.js and adding Babel presets.
- Resolved React errors related to deprecated packages, JSX parsing issues, and incorrect imports.
- Addressed build-related issues, ensuring that ESLint, Babel, and Webpack dependencies were properly installed and configured.

4. Optimizing and Building for Production

- Created a production build of the React app using npm run build.
- Ensured the built files were structured correctly in the build/ directory.
- Fixed issues where the app failed to load by serving the build/ folder through a local server rather than opening index.html directly.

5. Deploying the React App

- Tested the production build locally using serve -s build and Python’s HTTP server.
- Configured the "homepage" field in package.json to ensure correct relative paths when deployed.

PART B

Task Subjective Assessment.

Question 1a.

SageMaker is not strictly necessary in this setup since AWS Lambda handles inference by fetching the trained model from S3. However, SageMaker can be integrated for better scalability, version control, and optimized ML model serving. If we replace Lambda-based inference with SageMaker Hosting Services. The workflow changes by deploying the trained model as a SageMaker Endpoint**,** which API Gateway would call directly instead of Lambda. This approach allows for continuous hosting, eliminating cold starts and improving latency. Compared to Lambda, SageMaker is more scalable, supports versioning and A/B testing, and is better suited for high-traffic applications, but it incurs higher costs due to always-on instances. In contrast, Lambda is a cost-effective, serverless solution ideal for low-traffic applications, but it may suffer from cold starts due to model loading. Choosing between the two depends on the application's needs.

Question 2a.

To process a large dataset stored in Amazon S3 using AWS Glue and PySpark, we leverage AWS Glue's serverless ETL capabilities to efficiently handle large-scale data transformations. First, an AWS Glue Crawler scans the dataset in S3, infers its schema, and registers it in the AWS Glue Data Catalog for structured access. Next, an AWS Glue ETL Job is created using PySpark, which reads the dataset from S3. Various transformations can be applied, such as filtering, aggregations, handling missing values, and data type conversions. The processed data is then written back to S3 for further use. To automate the workflow, an AWS Glue Workflow can be set up using nodes to link multiple jobs. For example, the first Glue Job can handle data preprocessing, transforming and cleaning the raw dataset. Once this job completes, a trigger automatically starts the second Glue Job, which loads the processed data and trains the machine learning model. This workflow ensures an efficient, automated data pipeline, enabling seamless preprocessing and model training without manual intervention.

Question 3a.

AWS Amplify simplifies front-end development by providing a fully managed platform to build, deploy, and scale web and mobile applications. It offers an easy-to-use framework for integrating backend services such as authentication, APIs, storage, and hosting without requiring deep AWS expertise. Developers can quickly connect their applications to cloud-powered backends, automate deployments, and manage infrastructure with minimal configuration. Amplify supports popular front-end frameworks like React, Angular, and Vue, enabling seamless API integration and real-time data synchronization. By streamlining cloud

setup and deployment, AWS Amplify accelerates development workflows, allowing developers to focus on building user-friendly applications.