NANDHINI D

Nandhinidhamodharan26@gmail.com

Build an AI- agent to Answer E- commerce Data Questions

Initial insights on the datasets given:

1. Product level eligibility

* This dataset consists of four features such as “eligibility\_timestamp\_utc”, “item\_id”, “eligibility” and “message”.
* Eligibility\_timestamp\_utc: This feature represents the whether that respective product is eligible for “ad” based on the pricing of the product price.
* Item\_id: This feature represents unique identity given for the respective product.
* Eligibility: It is binary feature which conveys whether the respective product is qualified for publishing advertisement in the platforms.(e.g., TV, radio, social media etc.,).
* Message: This is text feature where it holds the description about the products pricing.

1. Product level Ad sales and Metrics

* This dataset consists of features such as “date”, “item\_id”,”ad\_sales”,”impressions”, “ad\_spend”, “clicks” and “units\_sold”.
* Date: consists of date stamp of each product visits and sales on the respective date.
* Item\_id: This feature represents unique identity given for the respective product.
* Ad\_sales: This feature represents how the sales is affected or impacted the advertisements that has been established to the product.
* Impressions: This explains how many new people or how many have viewed the respective product in a given day.
* Ad\_spend: This feature is about tracking the advertisement spending’s on the product.
* Clicks: Total number of times that the respective person have clicked and viewed on the product.
* Units\_sold: This feature represents total number of units sold on a particular product based on the advertisements and metrics.

1. Product level total sales and metrics

* This dataset consists of features such as “date”, “item\_id”, “total\_sales” and “total\_units\_ordrered”.
* Date: This feature marks the date stamp regarding the items identity and total units ordered.
* Item\_id: This feature represents unique identity given for the respective product.
* Total\_sales: This feature represents the total amount of profit made by the product after the advertisement.
* Total\_units\_ordered: This represents the total units ordered by the customer.

The following is the AI agent build to respond to the prompt asked by the user and understand using LLM (Large Language Models) such as google gemini 2.5 using an API key.

I have used python programming language for building the AI agent. And I have enclosed the coding and SQL data base is integrated with the cleaned data, so that the AI can query the database and understand the question asked by the user and respond to it with better understanding.

Tools Used: pandas, flask, post, sqlite, google generative ai, matplotlib.pyplot

Programming languages used: python, html.

PYTHON CODE:

import pandas as pd

import sqlite3

import matplotlib.pyplot as plt

import seaborn as sns

import google.generativeai as genai

from datetime import datetime

import re

from flask import Flask, request, jsonify

import json

app = Flask(\_\_name\_\_)

DATABASE\_NAME = 'ecommerce\_data.db'

# --- Gemini Configuration ---

genai.configure(api\_key="AIzaSyCtnWAGeFFd04JLSfqkQle4ioJDevrGrRI") # Replace with your actual key

model = genai.GenerativeModel("gemini-1.5-pro")

# --- Fix time format for eligibility table ---

def fix\_time\_format(dt\_str):

if not isinstance(dt\_str, str):

return None

try:

if ' ' in dt\_str and '.' in dt\_str.split(' ')[1] and dt\_str.count(':') < 2:

date\_part, time\_part = dt\_str.split(' ')

time\_parts = time\_part.split('.')

time\_part\_fixed = ':'.join(f"{int(t):02d}" for t in time\_parts)

return f"{date\_part} {time\_part\_fixed}"

return dt\_str

except Exception as e:

print(f"Error fixing time format for '{dt\_str}': {e}")

return None

# --- Load and process CSV file ---

def load\_and\_process\_data(file\_path, table\_name, conn):

print(f"Loading {file\_path}...")

try:

df = pd.read\_csv(file\_path, encoding='iso-8859-1')

except Exception as e:

print(f"Error loading {file\_path}: {e}")

return

if 'eligibility\_datetime\_utc' in df.columns:

df['eligibility\_datetime\_utc'] = df['eligibility\_datetime\_utc'].apply(fix\_time\_format)

df['eligibility\_datetime\_utc'] = pd.to\_datetime(df['eligibility\_datetime\_utc'], errors='coerce')

df['date\_only'] = df['eligibility\_datetime\_utc'].dt.date

df['hour'] = df['eligibility\_datetime\_utc'].dt.hour

for col in df.columns:

if df[col].dtype == 'object':

try:

df[col] = pd.to\_numeric(df[col], errors='coerce')

except ValueError:

pass

try:

df.to\_sql(table\_name, conn, if\_exists='replace', index=False)

print(f"Successfully saved {table\_name}.")

except Exception as e:

print(f"Error saving {table\_name}: {e}")

# --- Create combined view ---

def create\_combined\_view(conn):

cursor = conn.cursor()

cursor.execute("DROP VIEW IF EXISTS combined\_metrics")

cursor.execute("""

CREATE VIEW combined\_metrics AS

SELECT

e.product\_id,

e.eligibility\_datetime\_utc,

e.date\_only,

e.hour,

a.ad\_sales,

a.roas,

a.cpc,

t.total\_sales,

t.units\_sold

FROM eligibility\_table e

LEFT JOIN ad\_sales\_metrics a ON e.product\_id = a.product\_id

LEFT JOIN total\_sales\_metrics t ON e.product\_id = t.product\_id;

""")

conn.commit()

# --- Get schema from SQLite ---

def get\_database\_schema(conn):

cursor = conn.cursor()

cursor.execute("SELECT name FROM sqlite\_master WHERE type IN ('table', 'view');")

tables = cursor.fetchall()

all\_schemas = {}

for table\_name\_tuple in tables:

table\_name = table\_name\_tuple[0]

cursor.execute(f"PRAGMA table\_info({table\_name});")

columns = cursor.fetchall()

all\_schemas[table\_name] = "\n".join([f" {col[1]} ({col[2]})" for col in columns])

schema\_string = ""

for table, schema in all\_schemas.items():

schema\_string += f"Table: {table}\n{schema}\n\n"

return schema\_string.strip()

# --- Extract SQL from Gemini response ---

def extract\_sql\_query(response\_text):

sql\_match = re.search(r"```sql\s\*(.\*?)\s\*```", response\_text, re.DOTALL)

if sql\_match:

return sql\_match.group(1).strip()

if response\_text.strip().upper().startswith(("SELECT", "PRAGMA", "WITH")):

return response\_text.strip()

return None

# --- Ask Gemini for SQL ---

def ask\_gemini\_for\_sql(question, schema\_description):

prompt = f"""You are an AI agent working with the following SQLite database schema.

{schema\_description}

Based on the schema, write an appropriate SQLite SQL query to answer this question.

Respond ONLY with the SQL query enclosed in a markdown code block (```sql...```).

Question: {question}

"""

try:

response = model.generate\_content(prompt)

return response.text

except Exception as e:

print(f"Error calling Gemini API: {e}")

return None

# --- Execute SQL and return result ---

def execute\_sql\_and\_get\_results(sql\_query, conn):

try:

df = pd.read\_sql(sql\_query, conn)

return df, None

except Exception as e:

return None, str(e)

# --- Main Agent Logic ---

def answer\_question(question, conn, db\_schema):

gemini\_response = ask\_gemini\_for\_sql(question, db\_schema)

if not gemini\_response:

return {"error": "Gemini response failed"}, 500

sql\_query = extract\_sql\_query(gemini\_response)

if not sql\_query:

return {"error": f"Invalid SQL response:\n{gemini\_response}"}, 400

print(f"\n🧠 Question: {question}")

print(f" SQL Generated:\n{sql\_query}\n")

result\_df, error = execute\_sql\_and\_get\_results(sql\_query, conn)

if error:

return {"error": error, "sql\_query": sql\_query}, 500

if result\_df is not None and not result\_df.empty:

return {

"question": question,

"sql\_query": sql\_query,

"results": result\_df.to\_dict(orient='records')

}, 200

else:

return {

"question": question,

"sql\_query": sql\_query,

"explanation": "The query returned no results.",

"results": []

}, 200

# --- API Endpoint ---

@app.route('/ask', methods=['POST'])

def ask\_api():

question = request.json.get('question')

if not question:

return jsonify({"error": "No question provided"}), 400

conn = sqlite3.connect(DATABASE\_NAME)

schema = get\_database\_schema(conn)

response, status = answer\_question(question, conn, schema)

conn.close()

return jsonify(response), status

# --- API Endpoint ---

# Avoid re-registering the route if running in an interactive environment

app.view\_functions.pop('ask\_api', None) # <-- ADD THIS LINE

@app.route('/ask', methods=['POST'])

def ask\_api():

question = request.json.get('question')

if not question:

return jsonify({"error": "No question provided"}), 400

conn = sqlite3.connect(DATABASE\_NAME)

schema = get\_database\_schema(conn)

response, status = answer\_question(question, conn, schema)

conn.close()

return jsonify(response), status

# --- Initialize DB ---

def initialize\_database():

conn = sqlite3.connect(DATABASE\_NAME)

load\_and\_process\_data("D:\\DAA\\CDC\\ANARIX\\Product-Level Eligibility Table (mapped).csv", 'eligibility\_table', conn)

load\_and\_process\_data("D:\\DAA\\CDC\\ANARIX\\Product-Level Ad Sales and Metrics (mapped).csv", 'ad\_sales\_metrics', conn)

load\_and\_process\_data("D:\\DAA\\CDC\\ANARIX\\Product-Level Ad Sales and Metrics (mapped).csv", 'total\_sales\_metrics', conn)

create\_combined\_view(conn)

conn.close()

print("Database initialized.")

# --- Main Entrypoint ---

if \_\_name\_\_ == '\_\_main\_\_':

initialize\_database()

print(" AI Agent running at http://127.0.0.1:5000/ask")

print("Example: curl -X POST -H \"Content-Type: application/json\" -d '{\"question\": \"What is my total sales?\"}' http://127.0.0.1:5000/ask")

app.run(debug=False, port=5000)

from flask import Flask, request, jsonify, render\_template

@app.route('/')

def home():

return render\_template("index.html")

HTML:

<!DOCTYPE html>

<html>

<head>

<title>AI Product Metrics Q&A</title>

</head>

<body>

<h1>Ask a Question about Product Metrics</h1>

<label for="question">Enter your question:</label><br>

<input type="text" id="question" name="question" size="80"><br><br>

<button onclick="submitQuestion()">Ask</button>

<h2>Answer:</h2>

<pre id="responseBox"></pre>

<script>

async function submitQuestion() {

const question = document.getElementById("question").value;

const response = await fetch("/ask", {

method: "POST",

headers: {

"Content-Type": "application/json"

},

body: JSON.stringify({ question: question })

});

const result = await response.json();

document.getElementById("responseBox").textContent = JSON.stringify(result, null, 2);

}

</script>

</body>

</html>

OBSERVATIONS:

* It is observed that the advertisement and pricing of the product has a serious effect on the sales of the product.
* The data had a combination of datatypes such as integer, date time, dichotomous and unique id of the products.
* When the product is reduced for the pricing of the amazon, from then the product is made available for customers and advertisement are established to the customer.
* The impressions are the interpretations that is made on the advertisements, which ensures that the product have been established to the customers.

CONCLUSIONS:

* The AI agent uses google gemini 2.5 LLM (Large Language Model) which uses the SQL database that is been integrated to it.
* This model tries to understand the prompt that is been asked by the users and tries to retrieve the response by the asked question using SQL queries.
* Still the model is very new. It needs more training of the prompts to understand the context of the questions that is being asked.

RESULTS:

I think this is a really young model which needs more training, but yet it tries to understand the questions that is been asked to it. The results of the questions are as followed.