**WOODPECKER’S HACKATHON**

DataBot

AI Chatbot for Enhanced Communication and Analytics

TeamName : TechGaints

TeamMembers : Matanki S.K.

Shivani G.

Krishika R.

**Table of Contents**

1. Introduction

2. Overview

3. Existing Solutions

4. Need for the Project

5. Proposed Solution

6. Software Used

7. Flow of the Solution

8. Features

9. Code Explanation

10. Benefits

11. Results

12. Analysis

1. **Introduction**

* DataBot is an advanced web application designed to simplify database interactions by leveraging natural language processing and automated SQL query generation.
* The project integrates with MySQL and PostgreSQL databases to allow users to perform complex data queries and visualizations through a user-friendly interface.

**2. Overview**

* DataBot combines natural language processing with robust data querying and visualization capabilities.
* Users can connect to their databases, enter questions in natural language, and receive SQL queries along with graphical representations of the data.
* The application aims to bridge the gap between non-technical users and complex database systems.

**3. Existing Solutions**

* Current solutions for database querying often require manual SQL writing or the use of advanced query builders which can be challenging for non-technical users.
* Existing tools generally lack integration with AI models for natural language processing and may not provide automated visualization recommendations.

**4. Need for the Project**

* There is a growing need for intuitive tools that simplify data analysis for users without deep technical knowledge. DataBot addresses this need by:

- Allowing users to interact with databases using natural language.

- Automatically generating SQL queries based on user input.

- Providing visualizations to help users understand data insights quickly.

**5. Proposed Solution**

* DataBot provides a solution that integrates natural language processing with data querying and visualization. The application:

- Connects to MySQL and PostgreSQL databases.

- Translates natural language queries into SQL using a Generative AI model.

- Executes queries and displays results in a graphical format.

**6. Software Used**

* Streamlit: For building the interactive web application.
* Google Generative AI: For translating natural language into SQL queries.
* mysql.connector and psycopg2: For connecting to MySQL and PostgreSQL databases, respectively.
* Pandas:For data manipulation and analysis.
* Plotly Express: For creating visualizations.
* SQLAlchemy: For database schema retrieval.

**7. Flow of the Solution**

1. Database Connection:

- User selects the database type and provides connection details (host, port, user, password, database).

- The application connects to the selected database using appropriate connection functions.

2. Schema Retrieval:

- The application retrieves and displays the schema information of the database tables.

3. Query Processing:

- User inputs a natural language question.

- The Generative AI model translates the question into SQL.

- The application executes the SQL query on the connected database.

4. Visualization:

- The query results are analyzed.

- Appropriate visualizations are generated and displayed based on the data.

**8. Features**

* Natural Language Processing: Converts user questions into SQL queries.
* Database Connectivity: Supports MySQL and PostgreSQL databases.
* Dynamic Visualization: Generates charts based on query results.
* Historical Chat: Maintains a history of user queries and responses.
* User Interface: Intuitive web-based interface for ease of use.

**9. Code Explanation**

The following code components are central to the DataBot project:

* Imports and Configuration

import streamlit as st

import google.generativeai as genai

import mysql.connector

import psycopg2

import pandas as pd

import plotly.express as px

from langchain.sql\_database import SQLDatabase

import urllib.parse

from sqlalchemy import create\_engine

* Imports necessary libraries and sets up the Google Generative AI API key.

GOOGLE\_API\_KEY = 'your-google-api-key'

genai.configure(api\_key=GOOGLE\_API\_KEY)

* Defines classes for handling messages from the AI and user.

class AIMessage:

def \_\_init\_\_(self, content):

self.content = content

class HumanMessage:

def \_\_init\_\_(self, content):

self.content = content

* Database Connection Functions

def init\_database\_mysql(user, password, host, port, database):

password = urllib.parse.quote(password)

db\_uri = f"mysql+mysqlconnector://{user}:{password}@{host}:{port}/{database}"

return SQLDatabase.from\_uri(db\_uri)

def init\_database\_postgresql(user, password, host, port, database):

password = urllib.parse.quote(password)

db\_uri = f"postgresql+psycopg2://{user}:{password}@{host}:{port}/{database}"

return SQLDatabase.from\_uri(db\_uri)

* Functions to initialize database connections.
* **Schema Retrieval Functions**

def get\_mysql\_table\_schema(db\_uri):

engine = create\_engine(db\_uri)

query\_tables = "SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_SCHEMA = DATABASE();"

table\_names = pd.read\_sql(query\_tables, engine)

# Retrieve schema information

return schema\_info

def get\_postgresql\_table\_schema(db\_uri):

engine = create\_engine(db\_uri)

query\_tables = "SELECT table\_name FROM information\_schema.tables WHERE table\_schema = 'public';"

table\_names = pd.read\_sql(query\_tables, engine)

# Retrieve schema information

return schema\_info

* Functions to retrieve table schema from MySQL and PostgreSQL databases.
* Streamlit App Configuration

st.set\_page\_config(page\_title="DataBot", page\_icon=":speech\_balloon:")

with st.sidebar:

# UI for database connection

if st.button("Connect"):

# Connect to selected database

* Configures the Streamlit app interface and connects to the database based on user input.
* Query Processing and Visualization

def get\_gemini\_response(question, prompt):

model = genai.GenerativeModel('gemini-pro')

response = model.generate\_content([prompt[0], question])

return response.text

def read\_sql\_query(sql, config):

conn = mysql.connector.connect(\*\*config)

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

conn.close()

return df

def generate\_chart(df, chart\_type):

if chart\_type == 'bar':

fig = px.bar(df, x=df.columns[0], y=df.columns[1])

# Generate other chart types

st.plotly\_chart(fig, use\_container\_width=True)

* Handles query generation, data retrieval, and chart creation.

**10. Benefits**

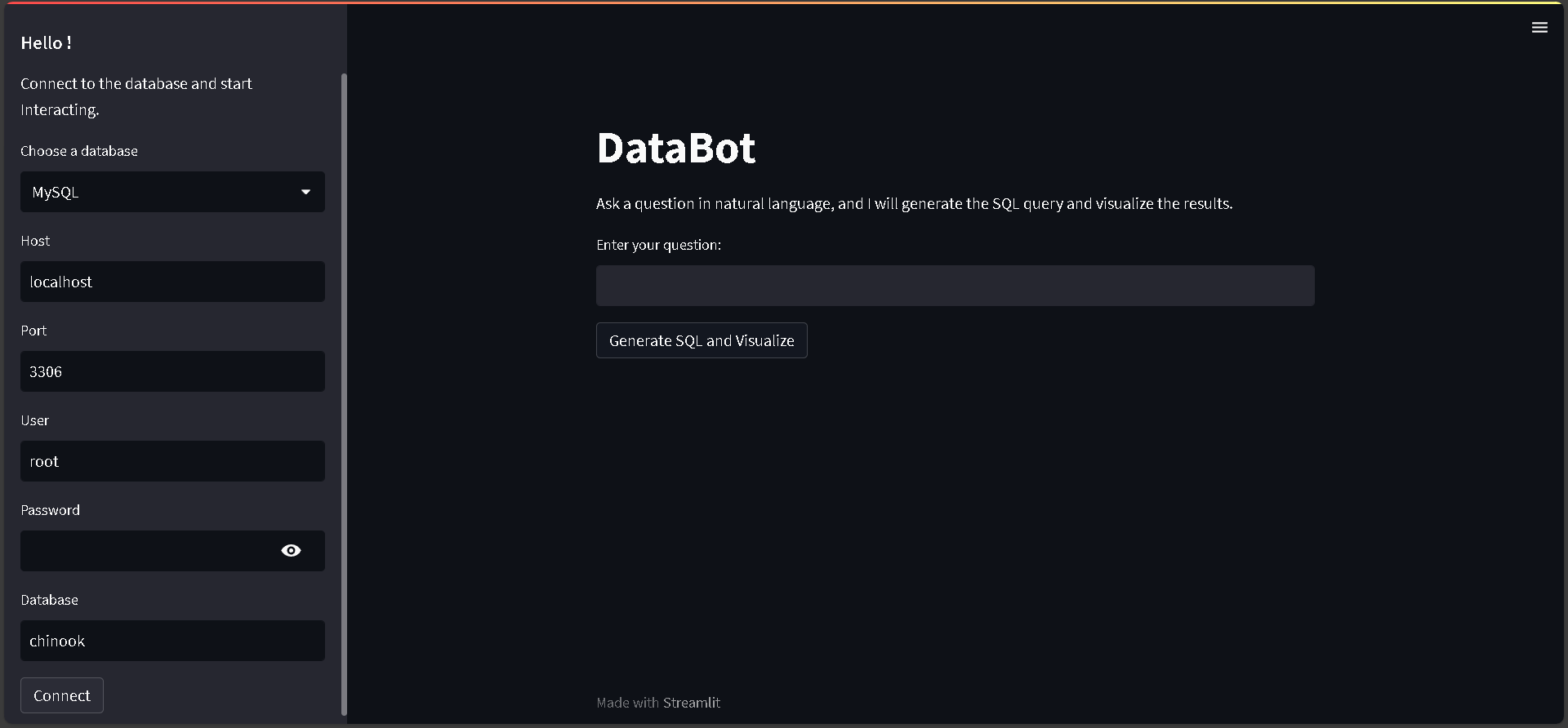
1.Ease of Use: Allows non-technical users to interact with databases using natural language.

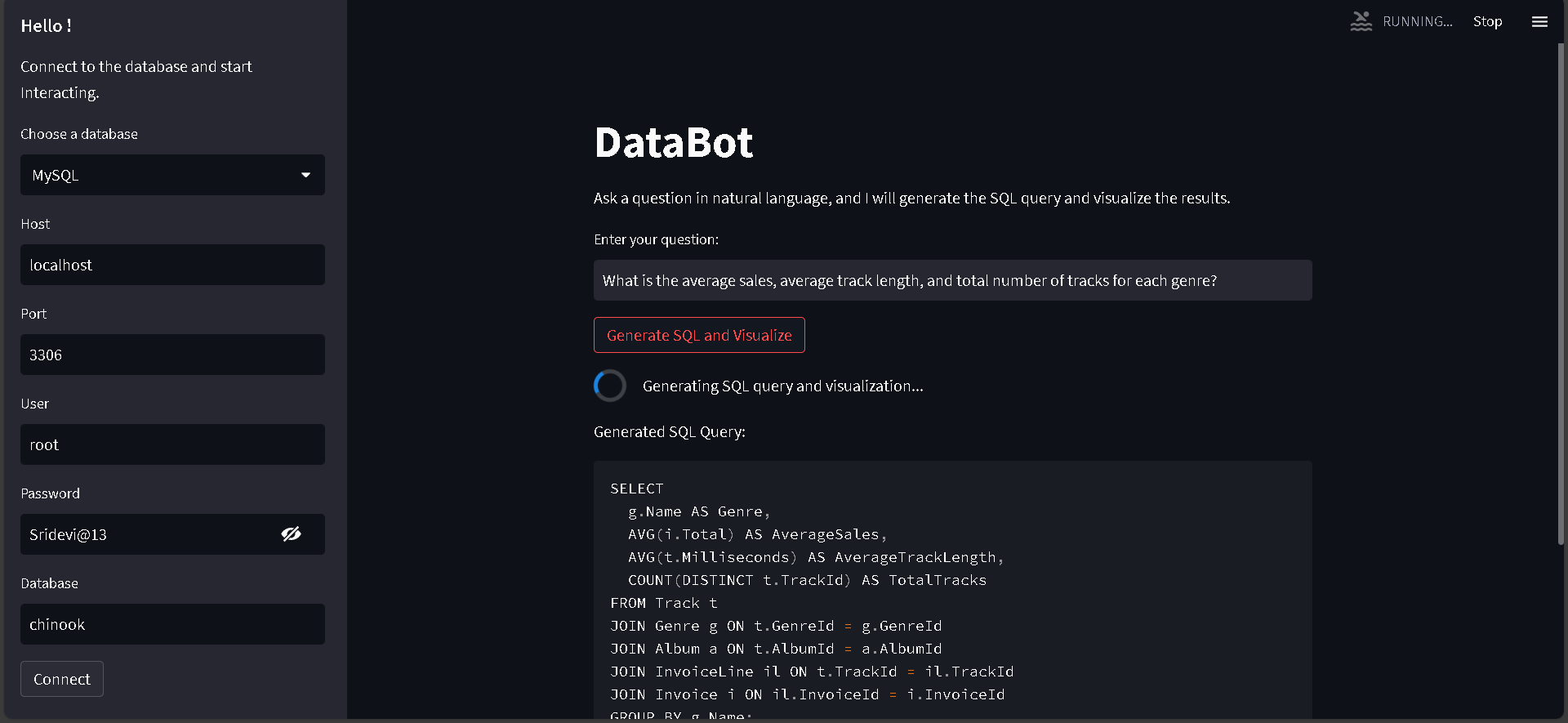
2. Automated Query Generation: Automatically generates SQL queries from user inputs.

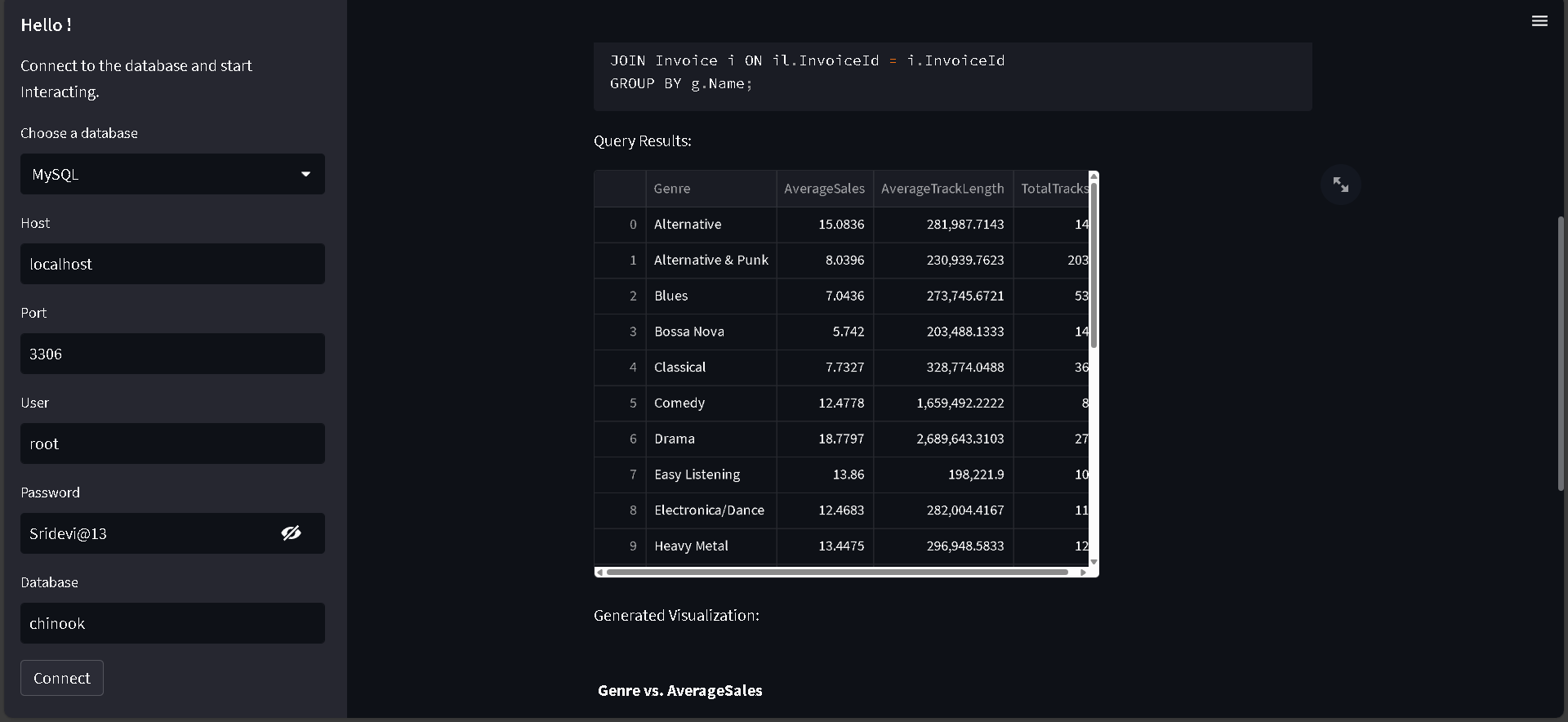
3. Integrated Visualization: Provides immediate visual representation of data for better insights.

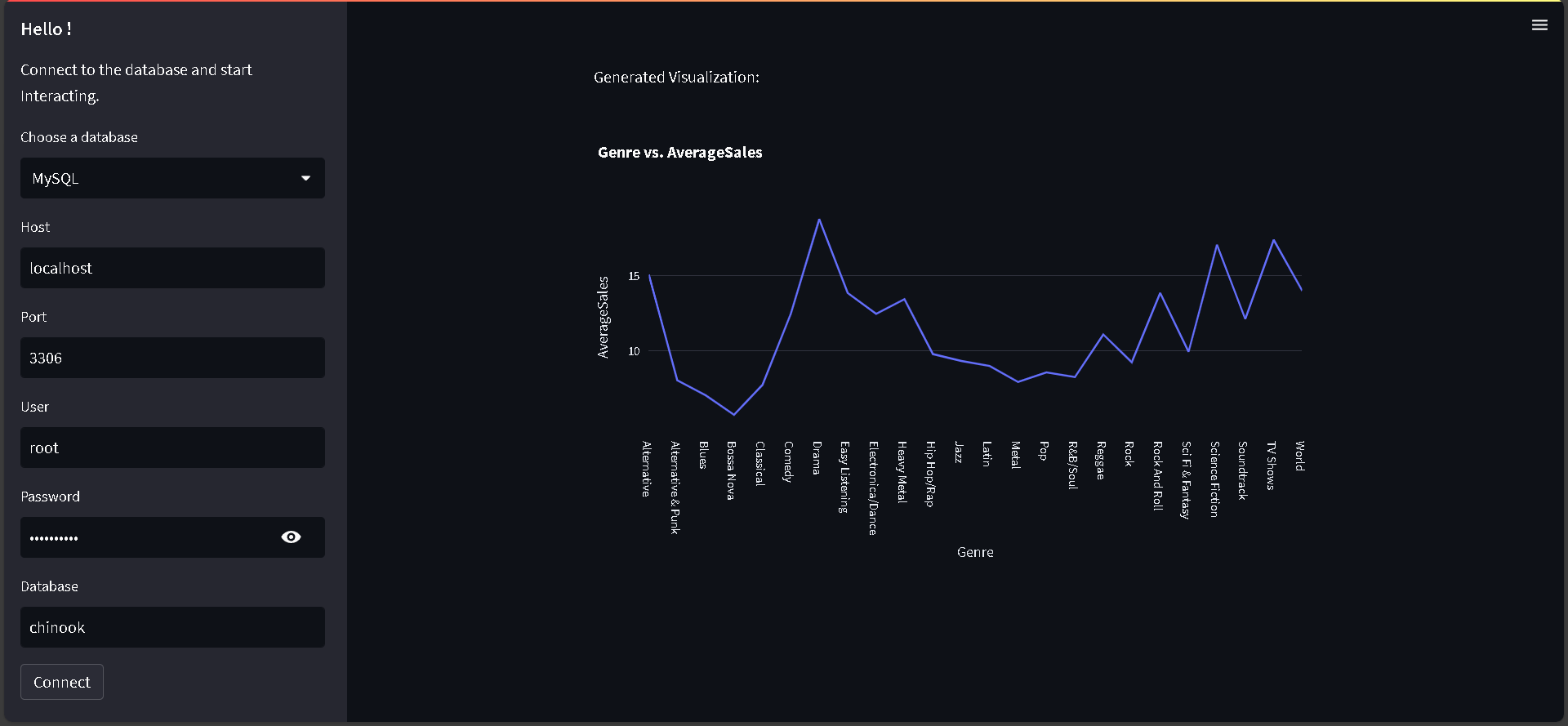
4. Historical Context: Maintains a history of interactions for reference and continuity.

**11. Results**

****

****

****

****

**12. Analysis**

* Efficiency: The integration of natural language processing and automated visualizations significantly enhances efficiency in data analysis.
* Accuracy: AI-driven SQL query generation ensures accurate results, reducing manual errors.
* User Satisfaction: The intuitive interface and automation improve user satisfaction by simplifying complex tasks.
* Scalability:The solution can be extended to support additional databases or functionalities, making it a scalable tool for various data analysis needs.