**CITIZEN AI**

**Project Documentation**

1. **Introduction :**

* Project title : CITIZEN AI
* Team leader : PRIYADHARSHINI B
* Team member : PAVITHRA.R
* Team member : NANDHINI.A
* Team member : MARTHA.G

1. **About:**

This document explains the Python code for the Citizen AI Platform, an application built using Gradio and the IBM Granite language model. The platform provides various services, including city analysis, citizen services, an AI assistant chat, and a feedback form, all accessible through a secure login system.

## **Dependencies and Setup:**

The code begins by installing the necessary libraries using pip:

* transformers: For using the IBM Granite language model.
* torch: The PyTorch library for deep learning operations.
* gradio: The library used to build the web-based user interface.

!pip install transformers torch gradio -q

The script then imports the required modules from these libraries: gradio as gr, torch, AutoTokenizer, and AutoModelForCausalLM.

## **4. Model Loading:**

The core of the application is the ibm-granite/granite-3.2-2b-instruct model. The code loads this pre-trained model and its corresponding tokenizer from the Hugging Face Transformers library. It checks for a GPU (torch.cuda.is\_available()) to optimize performance by moving the model to the GPU if available.

model\_name = "ibm-granite/granite-3.2-2b-instruct"

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

model = AutoModelForCausalLM.from\_pretrained(

model\_name,

torch\_dtype=torch.float16 if torch.cuda.is\_available() else torch.float32,

device\_map="auto" if torch.cuda.is\_available() else None

)

## **5. AI Response Generation:**

The generate\_response function is a reusable helper function for generating text using the loaded model. It takes a prompt and a max\_length as input. It tokenizes the prompt, generates a response from the model, and decodes the output back into a human-readable string.

def generate\_response(prompt, max\_length=1024):

# Tokenize the input prompt

inputs = tokenizer(prompt, return\_tensors="pt", truncation=True, max\_length=512)

# Generate a response from the model

with torch.no\_grad():

outputs = model.generate(...)

# Decode the response and return it

response = tokenizer.decode(outputs[0], skip\_special\_tokens=True)

return response.replace(prompt, "").strip()

## **6. Application Functions:**

The code defines several functions that represent the core features of the application, each of which uses the generate\_response function with a specific prompt:

* **city\_analysis(city\_name):** Formulates a prompt to request a detailed analysis of a city's safety, including crime rates and traffic safety, and returns the AI's response.
* **citizen\_interaction(query):** Creates a prompt designed for a "government assistant" persona to provide information on public services and civic issues.
* **assistant\_chat(message, history):** Manages a conversational chat flow, taking user messages and chat history to generate a new AI response.
* **collect\_feedback(name, rating, comments):** This is a simple function that simulates collecting user feedback and returns a confirmation message.

## **7. Login System:**

A basic, in-memory login system is implemented using a dictionary VALID\_USERS. The login function checks the provided username and password against this dictionary. If the credentials are valid, it updates the Gradio UI to hide the login form and show the main application content.

VALID\_USERS = {

"citizen": "ai123",

"admin": "admin123",

"guest": "guest123"

}

def login(username, password):

if username in VALID\_USERS and VALID\_USERS[username] == password:

return gr.update(visible=False), gr.update(visible=True), f"✅ Welcome {username}!"

else:

return gr.update(visible=True), gr.update(visible=False), "❌ Invalid Username or Password"

## **8. Gradio UI Layout:**

The user interface is built using gr.Blocks(), which allows for a more complex and dynamic layout.

* **Login Group:** A gr.Group is used to contain the username and password textboxes, and the login button. This group is initially visible.
* **Main Group:** Another gr.Group contains all the application tabs. This group is initially hidden (visible=False) and is only made visible after a successful login.
* **Tabs:** The gr.Tabs() component organizes the different features into separate, easy-to-navigate tabs:
  + **City Analysis:** Contains a textbox for a city name and a button to get an analysis.
  + **Citizen Services:** Has a textbox for a citizen query and a button to get a government-style response.
  + **AI Assistant:** A standard chatbot interface with a text input and a clear button.
  + **Feedback:** A form with fields for name, rating, and comments.

## **9. Linking UI and Functions:**

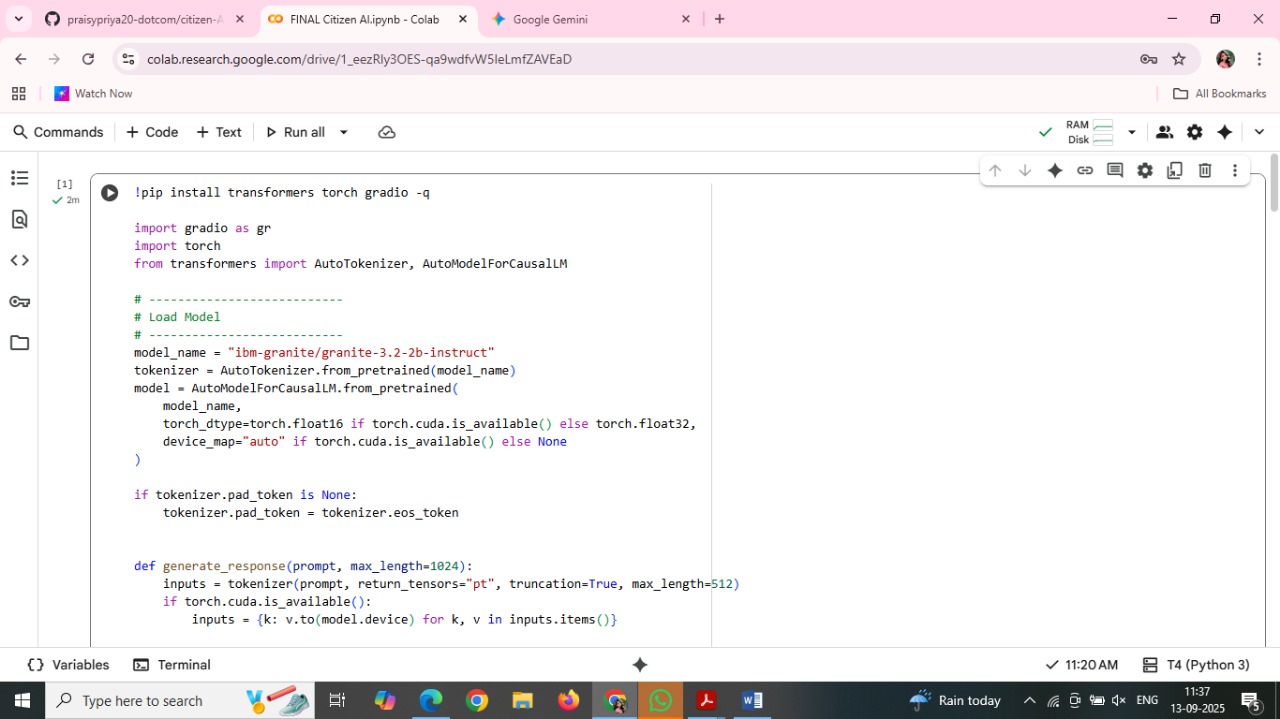
The click method on the Gradio components is used to link UI events (like a button click) to the Python functions.

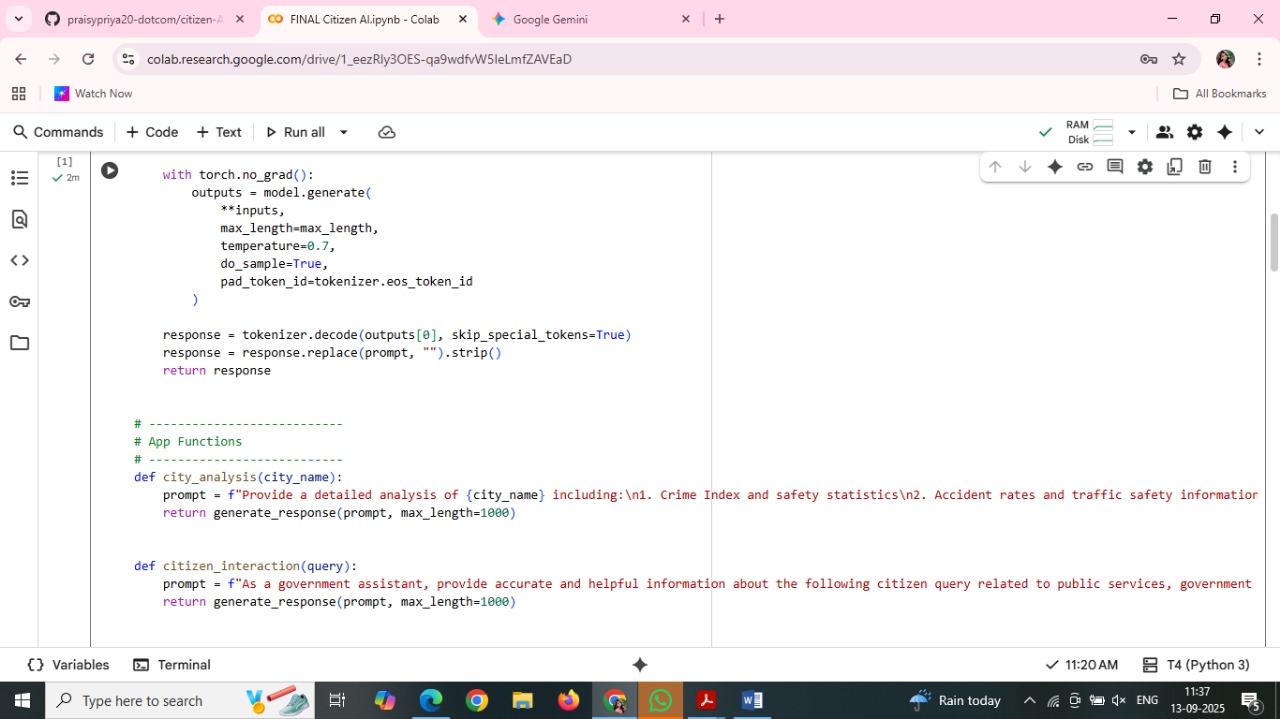
* login\_btn.click(...): When the login button is clicked, the login function is called. The outputs of this function are used to update the visibility of the login and main groups, and display a message.
* analyze\_btn.click(...): Calls the city\_analysis function and displays the output in the city\_output textbox.
* query\_btn.click(...): Calls the citizen\_interaction function.
* msg.submit(...): Handles the chat functionality, calling assistant\_chat to manage the conversation flow.
* submit\_btn.click(...): Calls collect\_feedback to process the feedback form data.

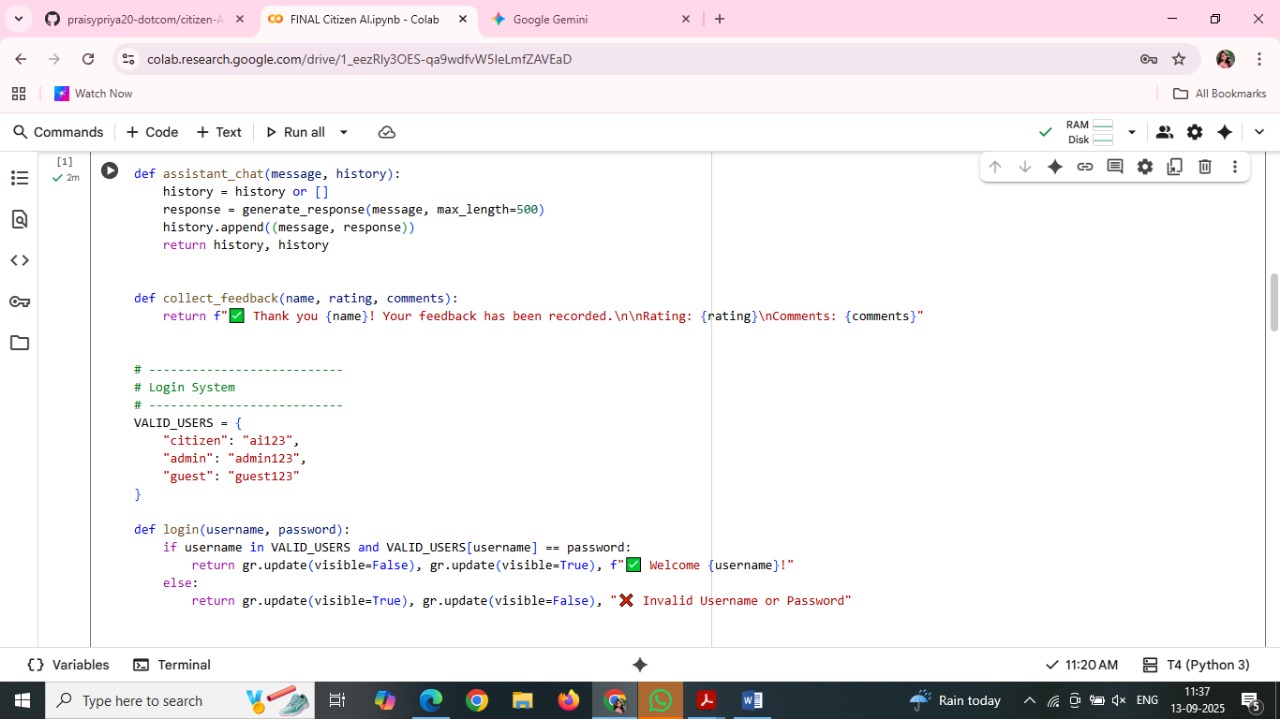
## **10. Launching the Application:**

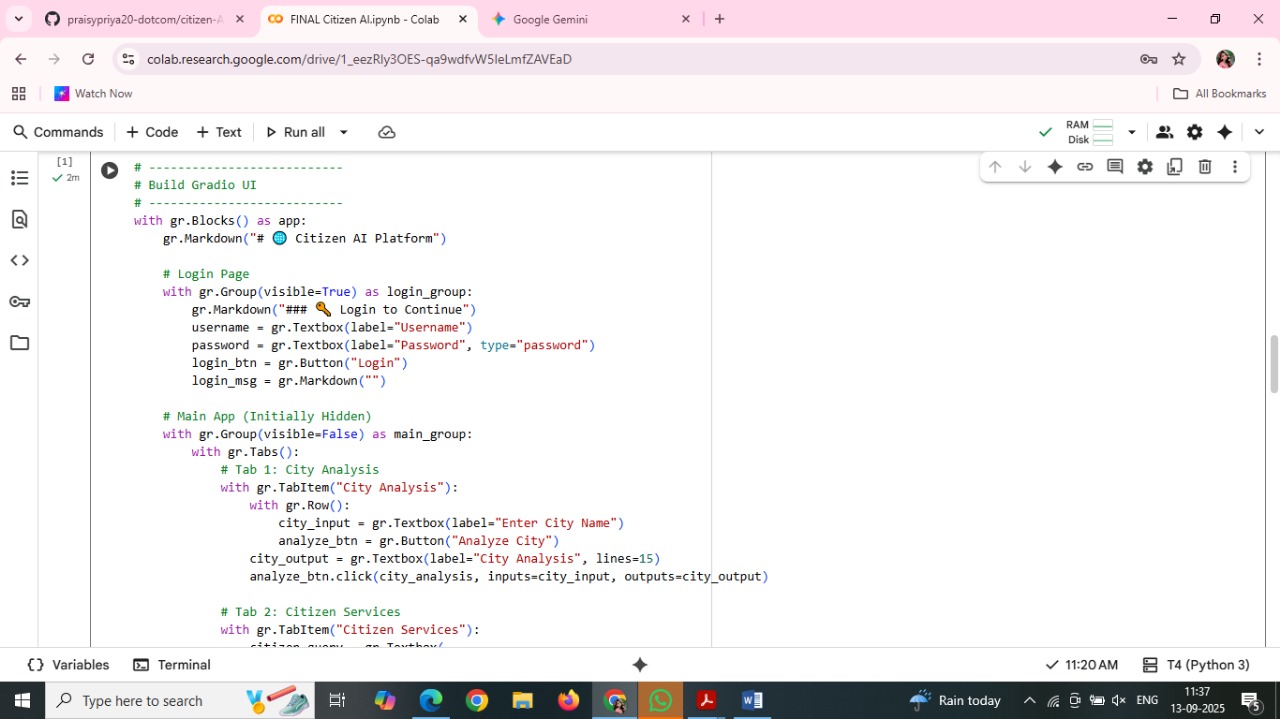
Finally, app.launch(share=True) starts the Gradio web server, making the application accessible in a web browser. The share=True argument creates a public, temporary URL that can be shared with others.

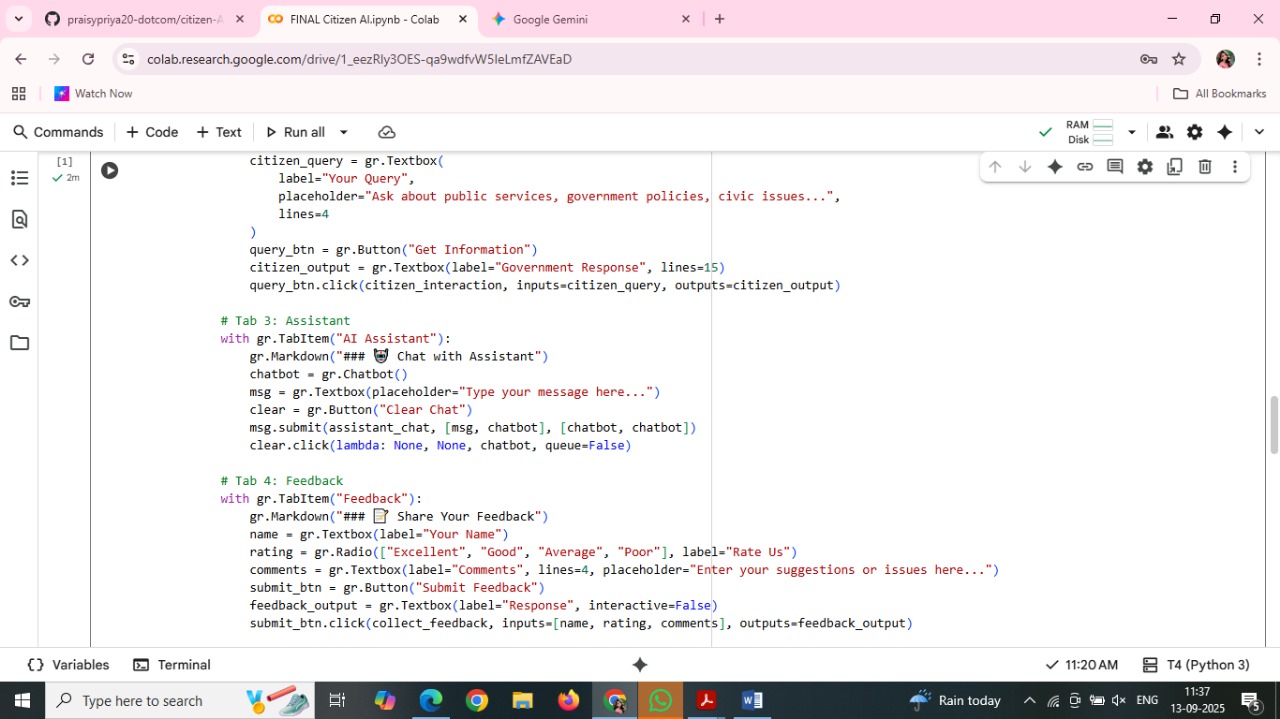
**11. Screenshots:**

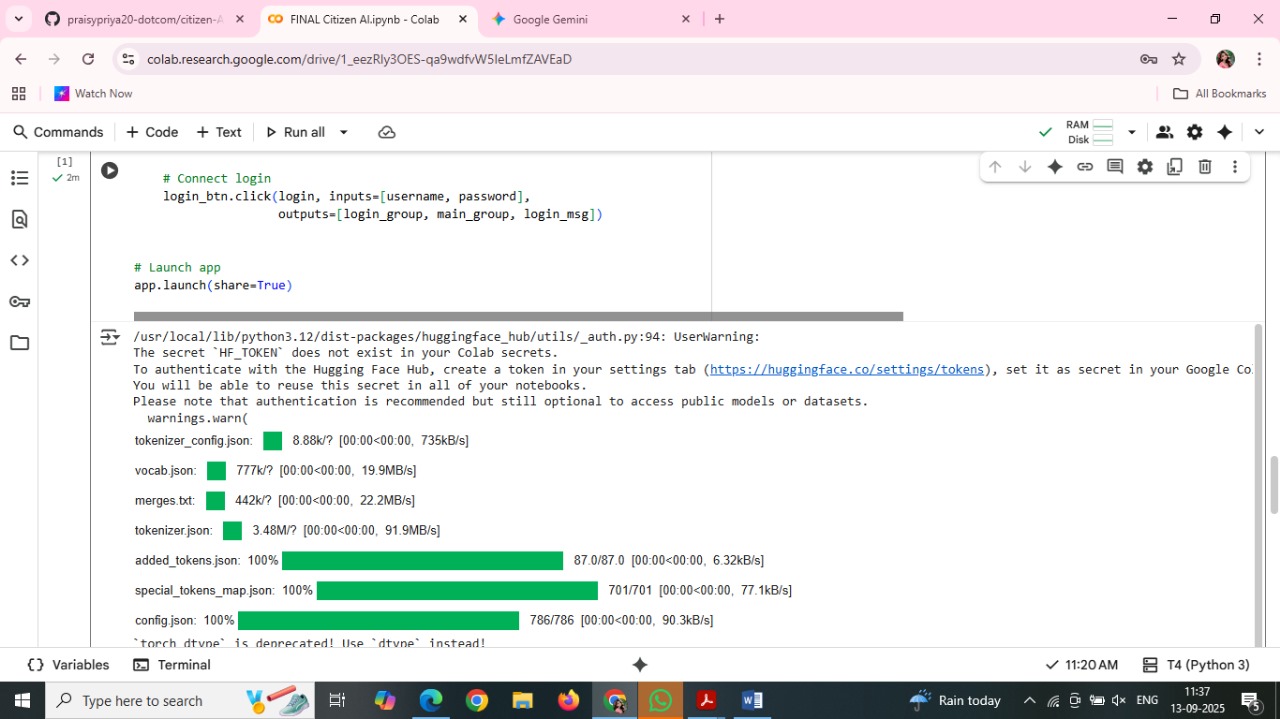
****

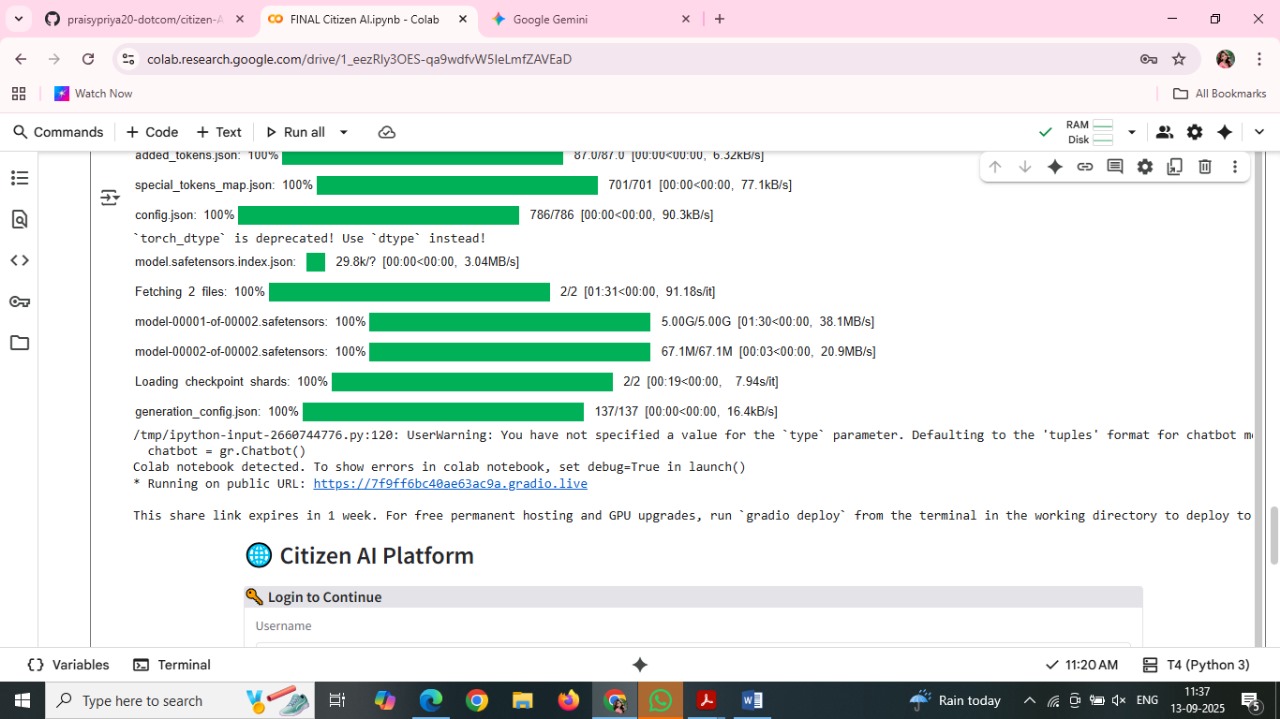


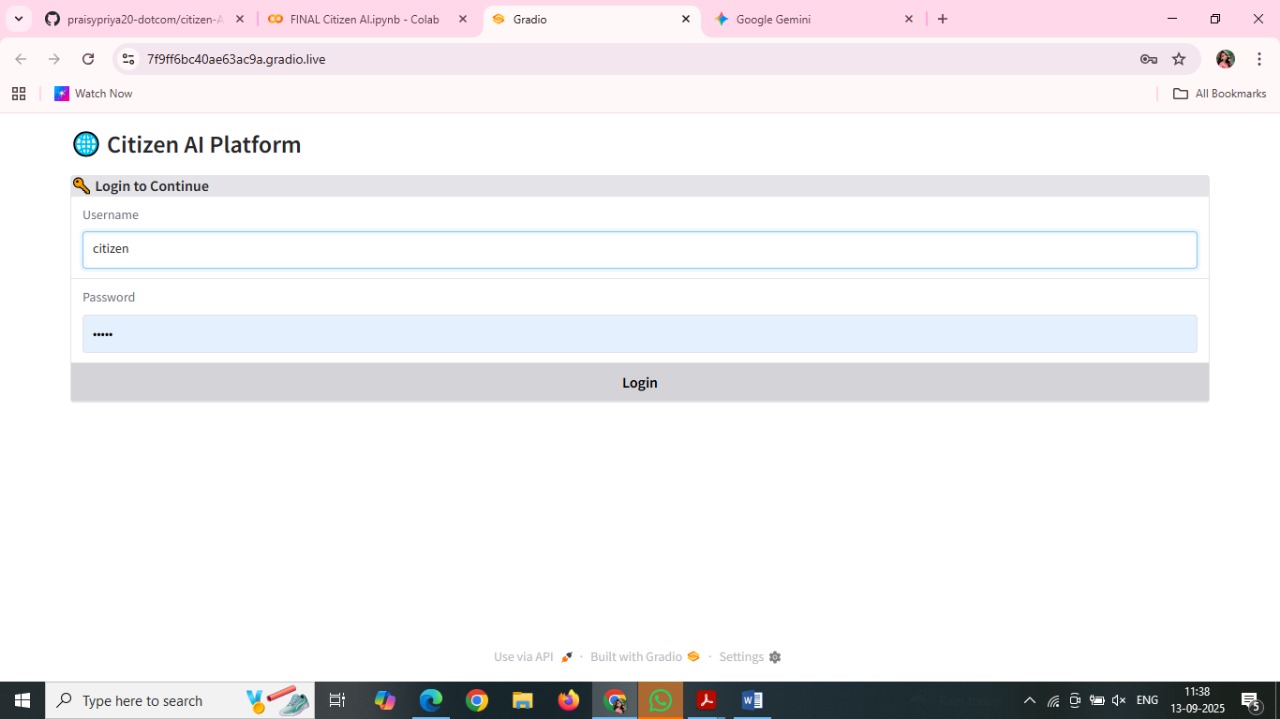


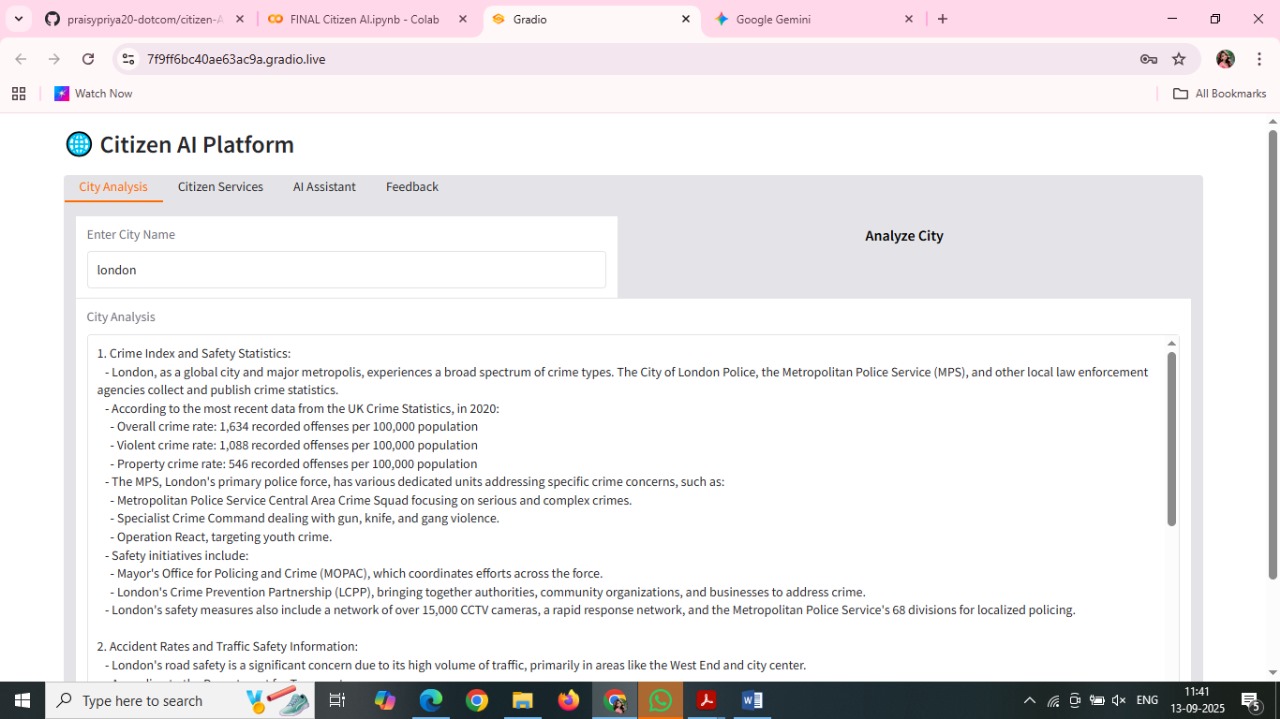


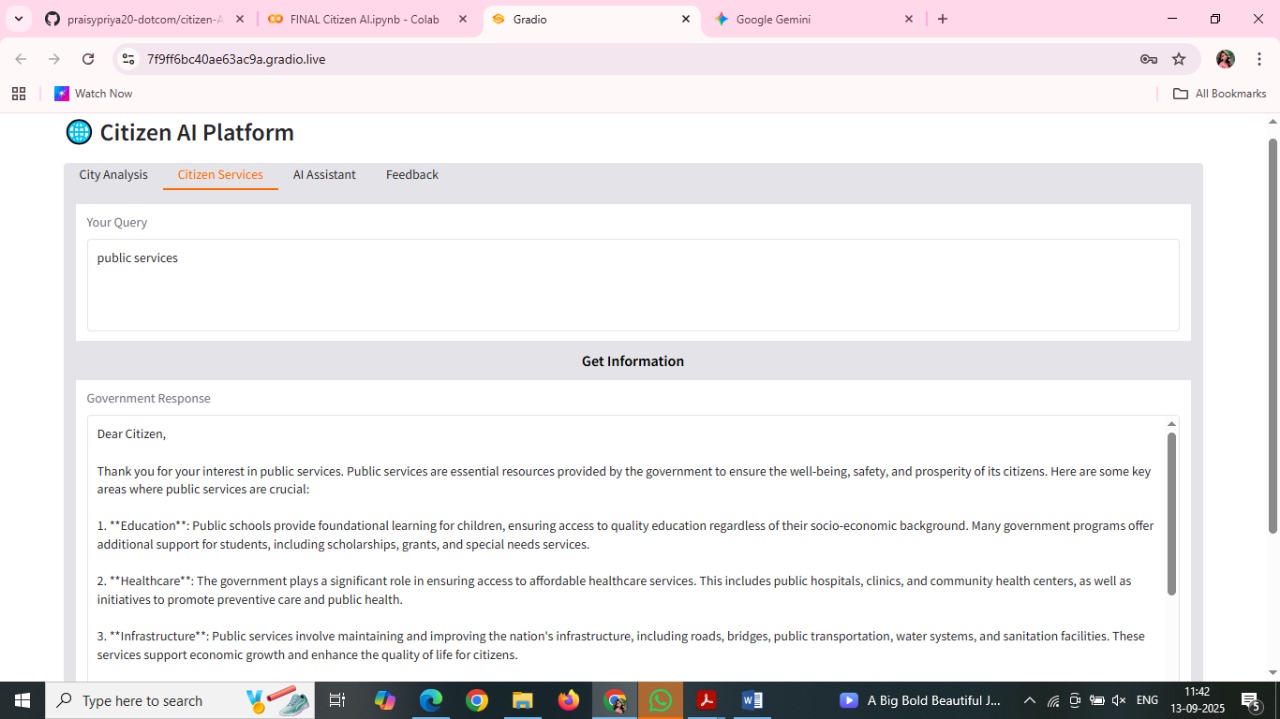


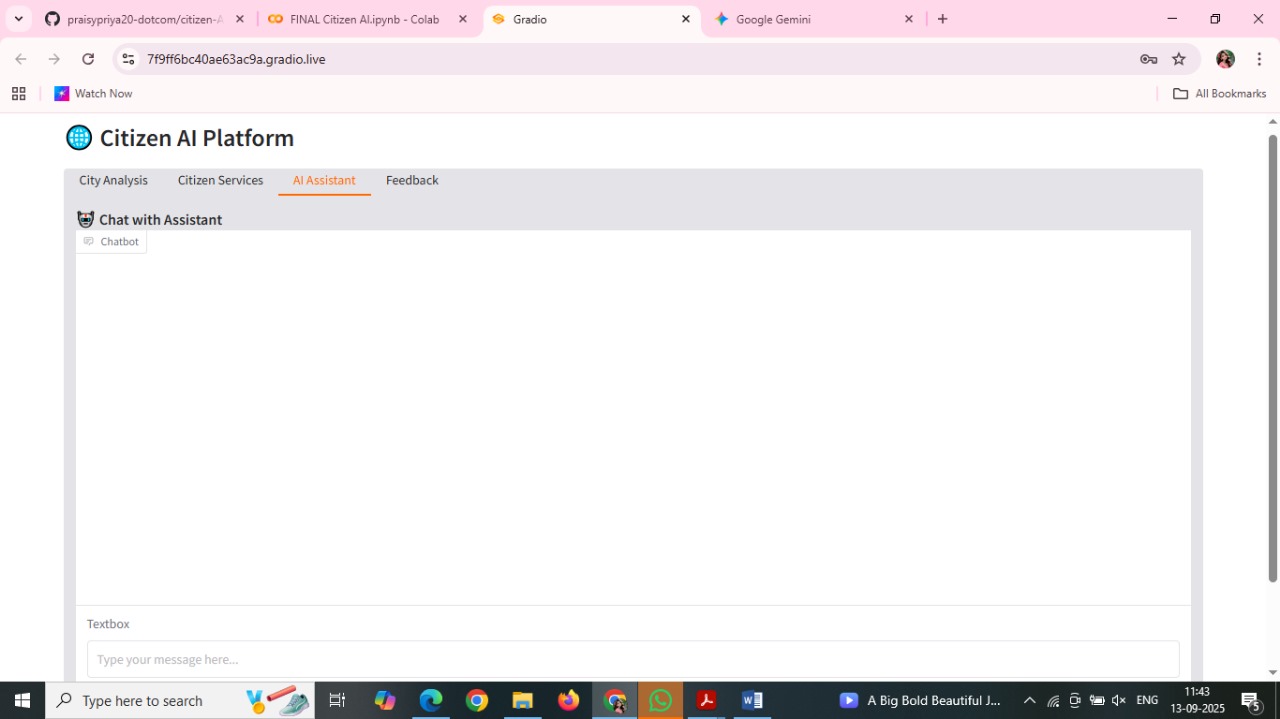


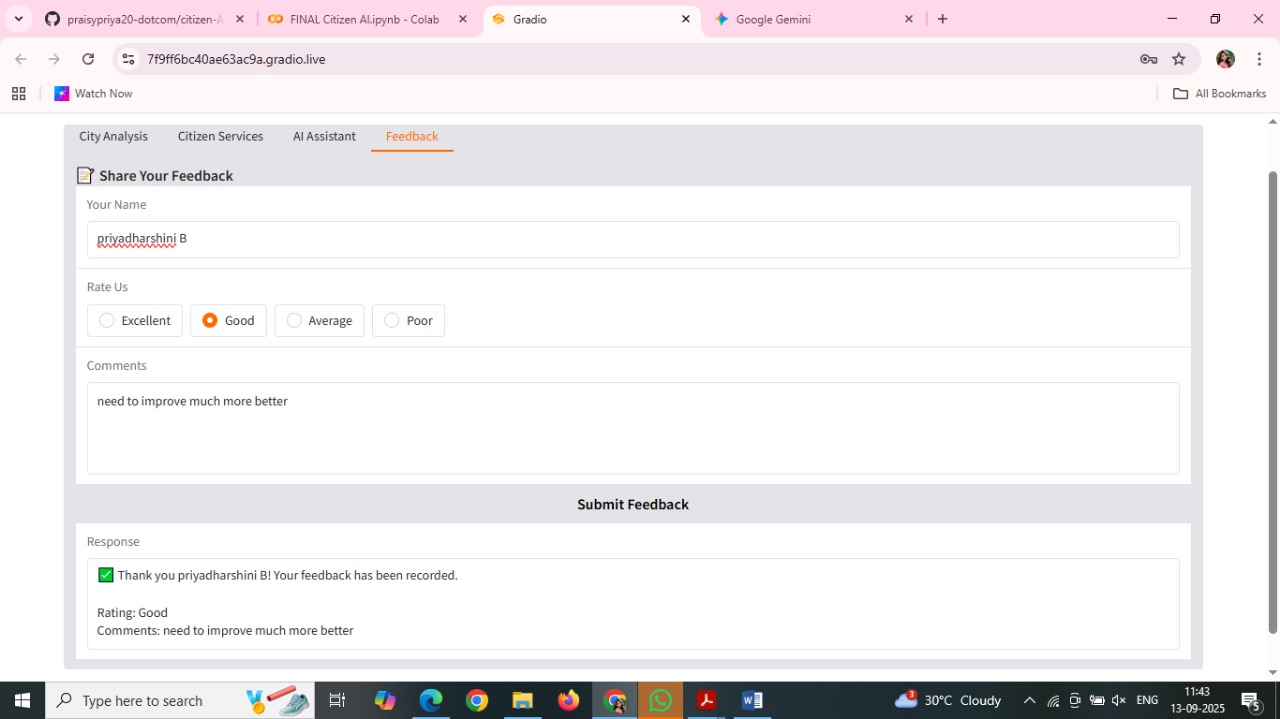












### 12. Future Enhancements:

* **User Sessions and History Tracking:** The project plans to add the ability to track user sessions and interaction history. This will allow for a more personalized experience.
* **Security:** For secure deployments, the project can integrate token-based authentication (JWT or API keys), OAuth2 with IBM Cloud credentials, and role-based access for different users (e.g., admin, citizen, researcher).