Project Documentation

1. Introduction

- **Project Title**: Citizen AI – Intelligent Citizen Engagement Platform
- **Team Members**:
  - Bavithra.R
  - Praseetha.M
  - Yogalakshmi.J
  - Rathi.P
  - Aartheeswari.L

2. Project Overview

Purpose

The purpose of *Citizen AI* is to create an **intelligent citizen engagement platform** that empowers communities, governments, and service providers to collaborate more effectively. By combining AI-driven insights with real-time data, the platform helps optimize public services, simplify policy communication, and enhance citizen participation.

For **citizens**, it offers a conversational assistant that provides clear updates, sustainability guidance, and quick access to government services. For **officials and administrators**, it acts as a decision-making partner with forecasting tools, feedback analytics, and anomaly detection. Ultimately, Citizen AI strengthens the bond between governance and community, driving **transparency, inclusivity, and smarter decision-making**.

Key Features

1. **Conversational Interface** – Natural language queries for policies, services, and updates.
2. **Policy Summarization** – Transforms lengthy documents into simple, actionable summaries.
3. **Resource Forecasting** – Predicts demand for public services using AI models.
4. **Citizen Tip Generator** – Offers personalized advice on sustainability and civic participation.
5. **Feedback Loop** – Gathers and analyzes citizen input for better decision-making.

6. **KPI Forecasting** – Tracks and projects progress in governance initiatives.
7. **Anomaly Detection** – Identifies unusual trends in service data for early intervention.
8. **Multimodal Input** – Accepts text, PDFs, and CSVs for policy/service analysis.
9. **Streamlit/Gradio UI** – Intuitive dashboards for both citizens and officials.

---

3. Architecture

- **Frontend (Streamlit):**
  Provides dashboards, chat interface, service feedback forms, and visualizations.
- **Backend (FastAPI):**
  Exposes APIs for chat, summarization, forecasting, and feedback management.
- **LLM Integration (IBM Watsonx Granite):**
  Powers summarization, conversational responses, and citizen engagement tips.
- **Vector Database (Pinecone):**
  Stores embeddings of policies and public documents for semantic search.
- **ML Modules:**
  - **Forecasting:** Predict service demand and citizen trends.
  - **Anomaly Detection:** Detect unusual behaviors or system usage.

4. Setup Instructions

Prerequisites

- Python 3.9+
- pip & virtual environment tools
- API keys: IBM Watsonx & Pinecone
- Internet connection

Installation Process

1. Clone repository.
2. Install dependencies: pip install -r requirements.txt.
3. Add .env with credentials.
4. Run backend server (uvicorn app.main:app --reload).

5. Launch frontend (streamlit run smart_dashboard.py).
6. Upload citizen feedback data or policies and interact with the platform.

5. Folder Structure:

```
city-analysis-citizen-ai/
│
├── notebooks/
│   └── city_services_ai.ipynb      # Google Colab notebook (main project file)
│
├── src/                        # Source code
│   ├── __init__.py
│   ├── app.py                  # Main Gradio app
│   ├── model_loader.py         # Handles model/tokenizer loading
│   ├── inference.py            # generate_response, city_analysis, citizen_interaction
│   └── ui.py                   # Gradio UI components
│
├── docs/                       # Documentation
│   ├── project_documentation.md     # Full documentation (markdown)
│   ├── project_documentation.pdf    # Exported version for submission
│   └── folder_structure.png         # Diagram of folder structure
│
├── requirements.txt            # Python dependencies
├── README.md                   # Project overview & usage
└── LICENSE                     # License info (optional)
```

6. Running the Application

- Start FastAPI backend.
- Run Streamlit frontend.
- Navigate via sidebar to access chat, policy summaries, forecasts, and reports.
- Citizens can provide feedback, officials can analyze results in real time.

7. API Documentation

- **POST /chat/ask** → Ask a question, get AI-powered responses.
- **POST /upload-doc** → Upload policies/service docs for analysis.
- **GET /search-docs** → Search policies/documents using natural language.
- **GET /get-citizen-tips** → Generate personalized civic/sustainability tips.
- **POST /submit-feedback** → Capture citizen feedback for analytics.

(All APIs are documented in **Swagger UI** for testing.)

8. Authentication

(Current demo is open access.)
Planned enhancements:

- Token-based authentication (JWT/API keys)
- OAuth2 with IBM Cloud credentials
- Role-based access (citizen, official, admin)
- User history & session tracking

---

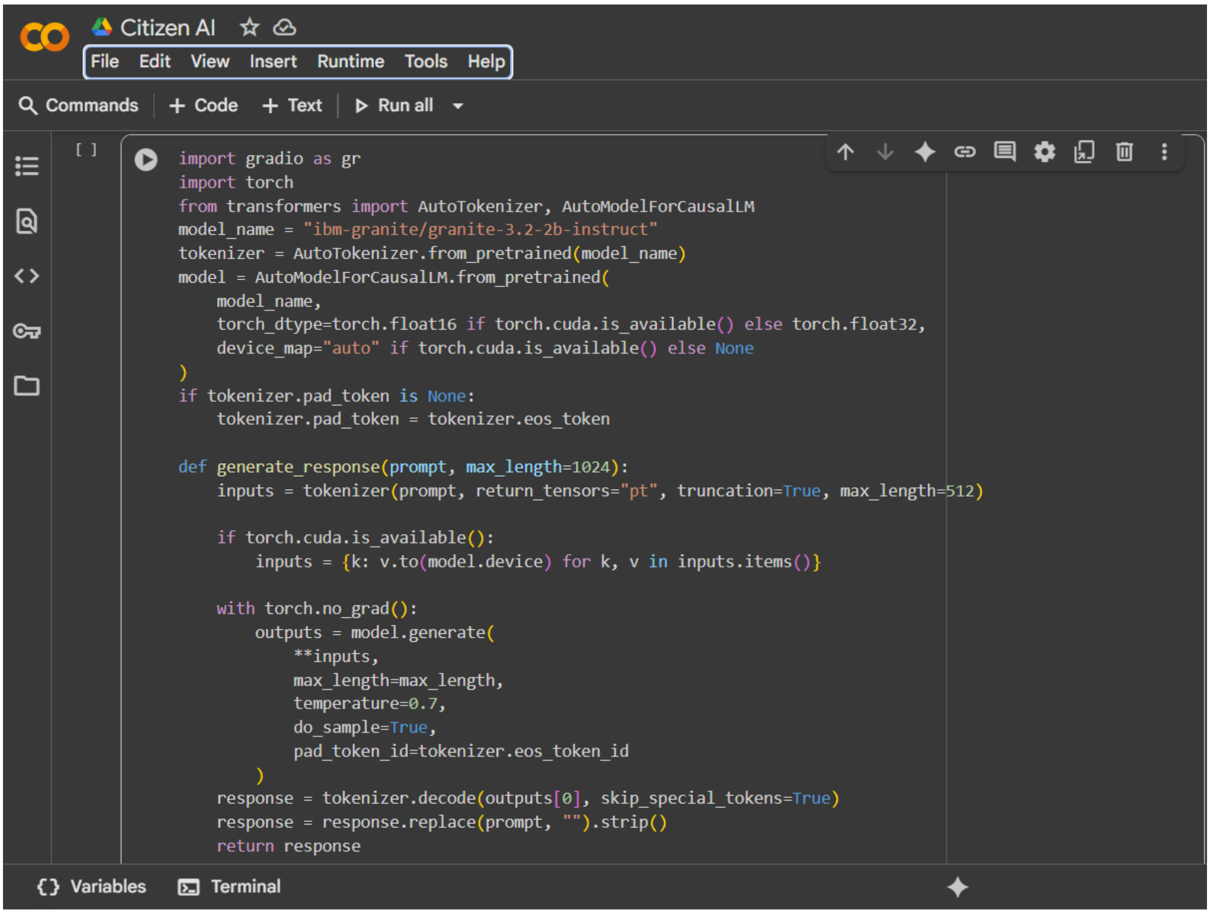
9. User Interface

- Sidebar-based navigation
- Service KPI dashboards
- Tabs for chat, summaries, forecasting
- Feedback forms and reports
- PDF export option
- Accessible, minimalist design for non-technical users

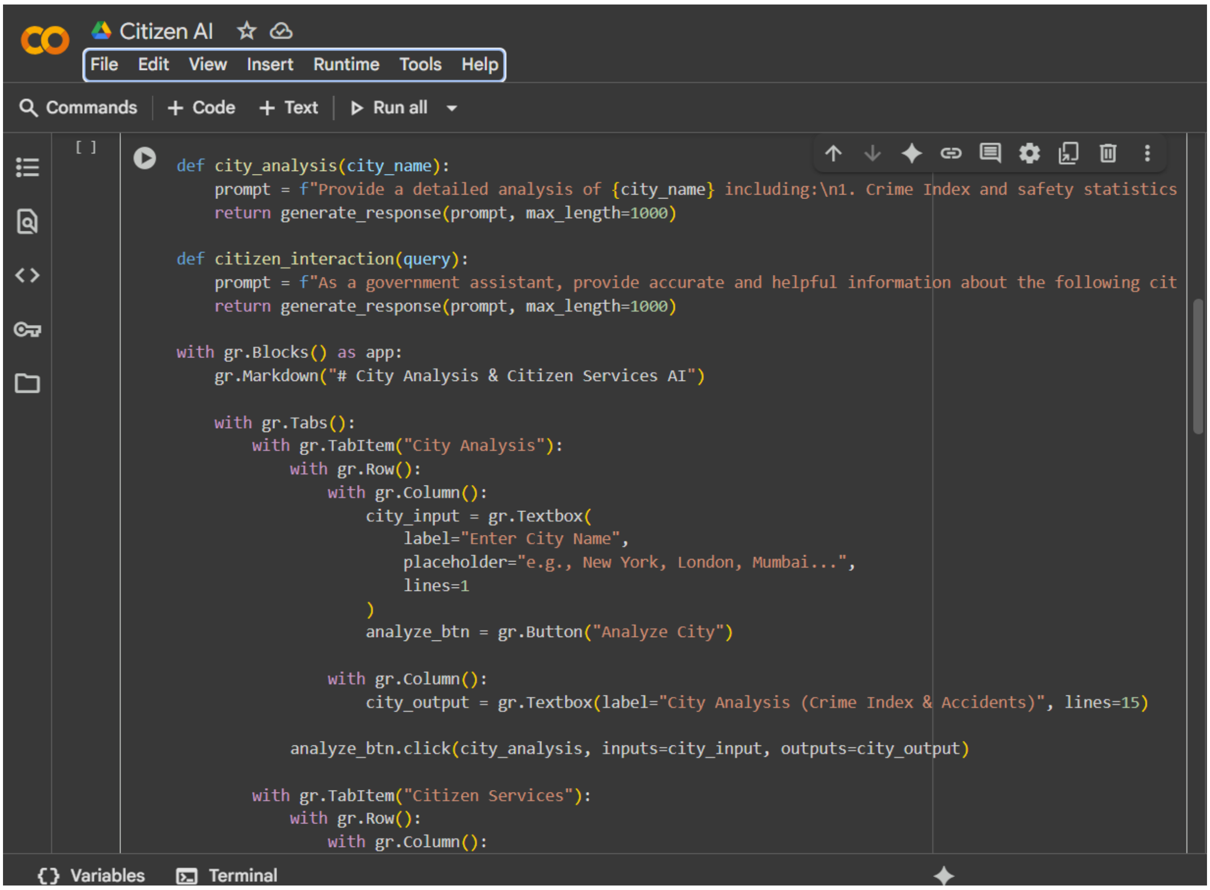
10. Testing

- **Unit Tests:** For summarization and utilities
- **API Tests:** Swagger UI & Postman
- **Manual Tests:** Citizen interactions, uploads, and reports
- **Edge Cases:** Large files, malformed inputs, missing credentials

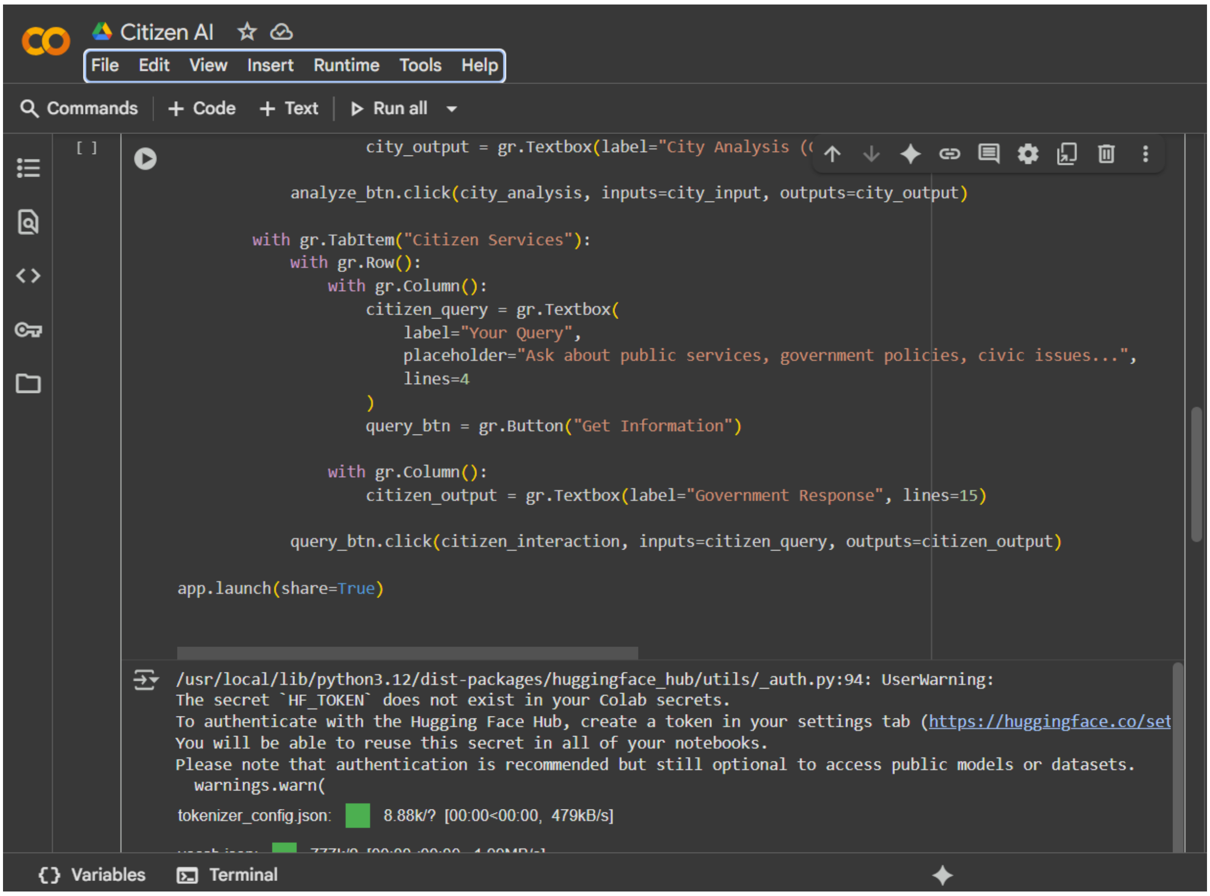
11. Screenshots:



```python
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
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
)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()
    return response
```

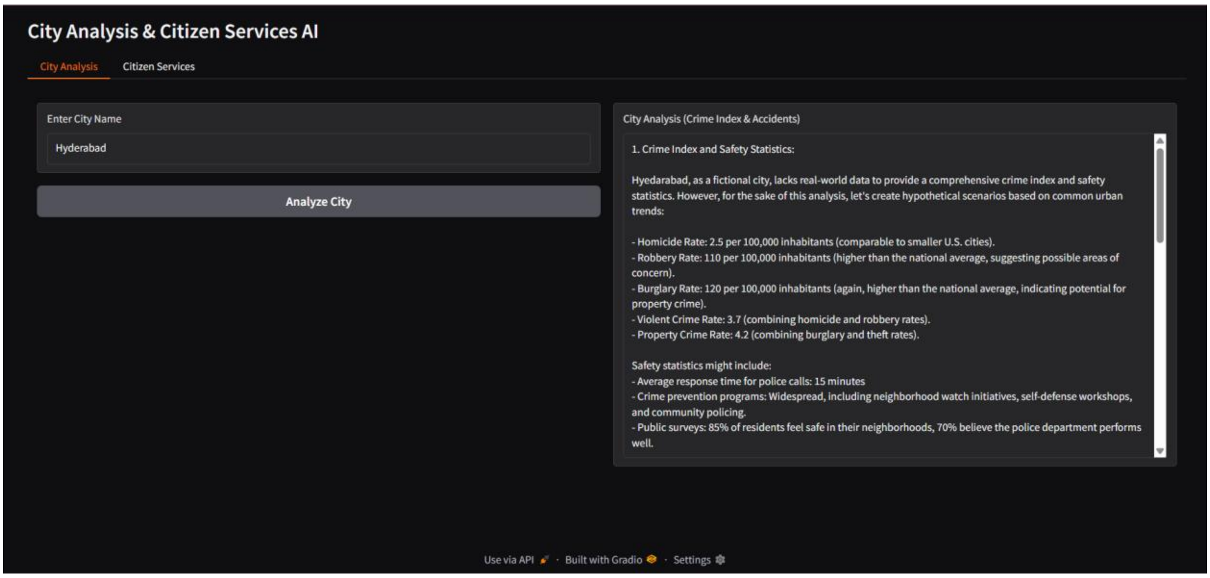


```python
def city_analysis(city_name):
    prompt = f"Provide a detailed analysis of {city_name} including:\n1. Crime Index and safety statistics
    return generate_response(prompt, max_length=1000)

def citizen_interaction(query):
    prompt = f"As a government assistant, provide accurate and helpful information about the following cit
    return generate_response(prompt, max_length=1000)

with gr.Blocks() as app:
    gr.Markdown("# City Analysis & Citizen Services AI")

    with gr.Tabs():
        with gr.TabItem("City Analysis"):
            with gr.Row():
                with gr.Column():
                    city_input = gr.Textbox(
                        label="Enter City Name",
                        placeholder="e.g., New York, London, Mumbai...",
                        lines=1
                    )
                    analyze_btn = gr.Button("Analyze City")

                with gr.Column():
                    city_output = gr.Textbox(label="City Analysis (Crime Index & Accidents)", lines=15)

            analyze_btn.click(city_analysis, inputs=city_input, outputs=city_output)

        with gr.TabItem("Citizen Services"):
            with gr.Row():
                with gr.Column():
```



```
                    city_output = gr.Textbox(label="City Analysis ((

            analyze_btn.click(city_analysis, inputs=city_input, outputs=city_output)

        with gr.TabItem("Citizen Services"):
            with gr.Row():
                with gr.Column():
                    citizen_query = gr.Textbox(
                        label="Your Query",
                        placeholder="Ask about public services, government policies, civic issues...",
                        lines=4
                    )
                    query_btn = gr.Button("Get Information")

                with gr.Column():
                    citizen_output = gr.Textbox(label="Government Response", lines=15)

            query_btn.click(citizen_interaction, inputs=citizen_query, outputs=citizen_output)

app.launch(share=True)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/set
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 🟩 8.88k/? [00:00<00:00, 479kB/s]
```


City Analysis & Citizen Services AI

Enter City Name: Hyderabad

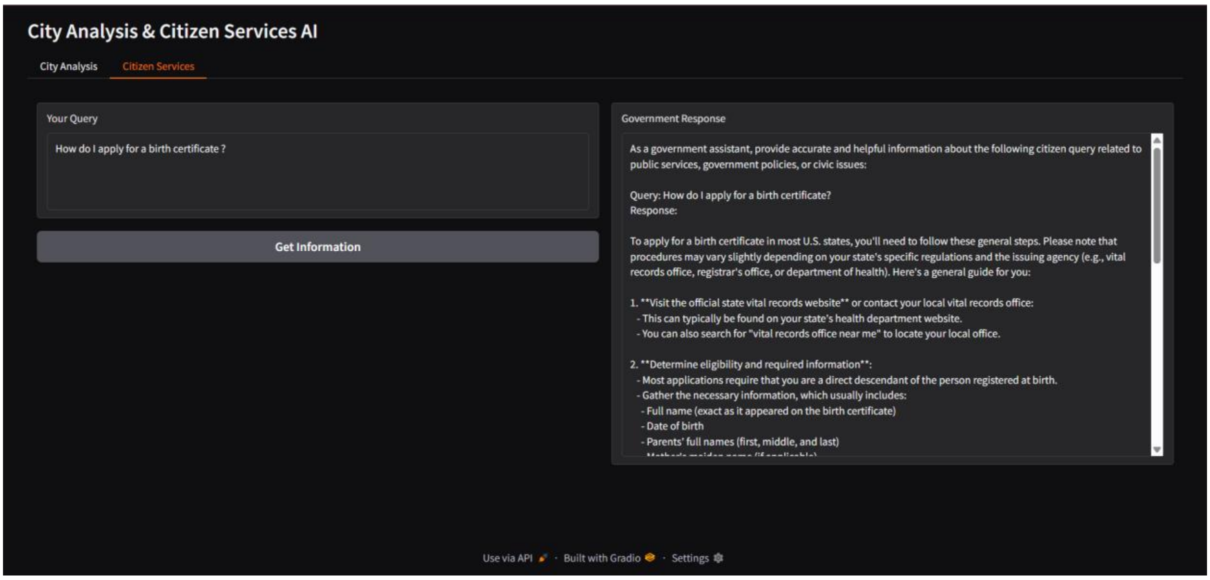
City Analysis (Crime Index & Accidents)

1. Crime Index and Safety Statistics:

Hyedarabad, as a fictional city, lacks real-world data to provide a comprehensive crime index and safety statistics. However, for the sake of this analysis, let's create hypothetical scenarios based on common urban trends:

- Homicide Rate: 2.5 per 100,000 inhabitants (comparable to smaller U.S. cities).
- Robbery Rate: 110 per 100,000 inhabitants (higher than the national average, suggesting possible areas of concern).
- Burglary Rate: 120 per 100,000 inhabitants (again, higher than the national average, indicating potential for property crime).
- Violent Crime Rate: 3.7 (combining homicide and robbery rates).
- Property Crime Rate: 4.2 (combining burglary and theft rates).

Safety statistics might include:
- Average response time for police calls: 15 minutes
- Crime prevention programs: Widespread, including neighborhood watch initiatives, self-defense workshops, and community policing.
- Public surveys: 85% of residents feel safe in their neighborhoods, 70% believe the police department performs well.


City Analysis & Citizen Services AI
City Analysis
Citizen Services
Your Query
Get Information
Government Response
Use via API · Built with Gradio · Settings

How do I apply for a birth certificate ?

```
As a government assistant, provide accurate and helpful information about the following citizen query related to
public services, government policies, or civic issues:

Query: How do I apply for a birth certificate?
Response:

To apply for a birth certificate in most U.S. states, you'll need to follow these general steps. Please note that
procedures may vary slightly depending on your state's specific regulations and the issuing agency (e.g., vital
records office, registrar's office, or department of health). Here's a general guide for you:

1. **Visit the official state vital records website** or contact your local vital records office:
  - This can typically be found on your state's health department website.
  - You can also search for "vital records office near me" to locate your local office.

2. **Determine eligibility and required information**:
  - Most applications require that you are a direct descendant of the person registered at birth.
  - Gather the necessary information, which usually includes:
   - Full name (exact as it appeared on the birth certificate)
   - Date of birth
   - Parents' full names (first, middle, and last)
   - Mother's maiden name (if applicable)
```

12. Known Issues:

- Limited handling of multi-language inputs
- Forecasting accuracy depends on available datasets
- IoT/real-time integration is not yet enabled

13. Future Enhancements:

- **IoT Integration:** Connect live city sensor data (traffic, waste, utilities).
- **Mobile App Version:** Expand accessibility on smartphones.
- **Multi-language Support:** Engage diverse citizen groups.
- **AI-driven Sentiment Analysis:** Understand citizen emotions from feedback.
- **Gamification & Rewards:** Encourage active citizen participation.