PROJECT TITLE

PROJECT DOCUMENTATION

1.Introduction

- Project title : **Sustainable Smart City Assistant**
- Team Leader : **KILLIVALAVAN R**
- Team member: ANTONY M
- Team member : BALAJI D
- Team member : PUSHPARAJ J

2.project overview

- **Purpose** :

The purpose of a Sustainable Smart City Assistant is to empower cities and their residents to thrive in a more eco-conscious and connected urban environment. By leveraging AI and real-time data, the assistant helps optimize essential resources like energy, water, and waste, while also guiding sustainable behaviors among citizens through personalized tips and services. For city officials, it serves as a decision making partner—offering clear insights, forecasting tools, and summarizations of complex policies to support strategic planning. Ultimately, this assistant bridges technology, governance, and community engagement to foster greener cities that are more efficient, inclusive, and resilient.

1. Introduction

Brief overview of environmental problems in the modern world.

Importance of technology and AI in sustainability.

Objective of this project → Provide eco-friendly tips and summarize policy documents to assist individuals, researchers, and policymakers.

2. Problem Statement

Challenges:

People lack practical eco-friendly guidance.

Government/NGO policy documents are lengthy and complex.

Need:

An intelligent assistant that generates actionable tips.

Automatic summarization of PDF policy files for easy understanding.

3. Objectives of the Project

1. To design a user-friendly assistant for sustainable living.

2. To use AI language models (IBM Granite) for generating eco-tips.

3. To integrate policy summarization from PDF or text.

4. To provide a web-based solution using Gradio.

4. Literature Review

Artificial Intelligence in Sustainability:

AI supports waste management, smart grids, renewable energy forecasting.

NLP in Policy Analysis:

Natural Language Processing simplifies complex legal/policy texts.

Existing Solutions:

Chatbots exist, but lack domain-specific eco advice.

Most summarizers don’t focus on sustainability policies.

5. Methodology

5.1 Tools & Technologies

Language: Python

Framework: Gradio

Library: HuggingFace Inference API (Granite Model), PyPDF2 Hosting: Local machine / HuggingFace Spaces

5.2 Workflow

1. Input: User provides keywords or uploads PDF.

2. Preprocessing: Extract text (if PDF uploaded).

3. AI Model: IBM Granite generates tips/summaries.

4. Output: Display results in user-friendly UI.

6. System Architecture

+------------------+ +------------------+ +-------------------+ | User Input | ---> | AI Model (Granite)| ---> | AI Output |

| (Keywords/PDF) | | Text Generation | | (Tips/Summary) | +------------

-----+ +------------------+ +-------------------+

7. IMPLEMENTATION

7.1 Extracting Text from PDF

Used PyPDF2 to read PDF documents.

Extracted text is cleaned and passed to Granite model.

7.2 Eco Tips Generator

Input: Problem keywords.

AI Prompt: "Give eco-friendly tips for: <keywords>"

Output: Practical, short, actionable eco tips.

7.3 Policy Summarization

Input: PDF or manual policy text.

AI Prompt: "Summarize the following policy document: ..."

Output: Summary of key provisions, rules, and implications.

7.4 Gradio Interface

Built two tabs:

1. Eco Tips Generator

2. Policy Summarization

8. Sample Outputs

Example 1: Eco Tips

Input: plastic pollution

Output:

Use reusable cloth bags instead of plastics.

Reduce bottled water usage by carrying refillable bottles.

Promote biodegradable packaging materials.

Conduct awareness drives in schools and communities.

Example 2: Policy Summarization

Input PDF: Renewable Energy Policy 2023

Output:

Encourages installation of rooftop solar.

Provides subsidies for EV charging stations.

Mandates waste-to-energy plants in urban areas.

Introduces stricter penalties for polluting industries.

9. Results & Discussion

The assistant successfully provides domain-relevant eco tips.

Summaries are short, clear, and actionable.

Helps students, policymakers, NGOs, and citizens.

10. Advantages

Simple, lightweight, and mobile-friendly.

Works with real-world PDF policies.

Can be expanded into multilingual support.

Uses cloud-hosted Granite LLM, no heavy GPU required.

11. Limitations

Depends on HuggingFace API availability.

Summarization may miss minor details.

Accuracy depends on the AI model.

12. Future Enhancements

Add speech-to-text input (voice queries).

Provide data visualization for policy insights.

Support regional languages (Tamil, Hindi, etc.).

Enable policy comparison tool between multiple documents.

Deploy on HuggingFace Spaces for global access.

13. Conclusion

The project demonstrates how AI + NLP can support sustainability.

The Eco Assistant encourages individuals to adopt eco-friendly habits.

Policy Analyzer makes government regulations easier to understand.

This work has social, educational, and environmental benefits.

PROGRAM CODE:

```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load model and tokenizer
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

def disease_prediction(symptoms):
    prompt = f"Based on the following symptoms, provide possible medical conditions and general medication suggestions. Always emphasize the importance of consulting a doctor for proper diagnosis.\n\nSymptoms: {symptoms}\n\nPossible conditions and recommendations:\n\n**IMPORTANT: This is for informational purposes only. Please consult a healthcare professional for proper diagnosis and treatment.**\n\nAnalysis:"
    return generate_response(prompt, max_length=1200)

def treatment_plan(condition, age, gender, medical_history):
    prompt = f"Generate personalized treatment suggestions for the following patient information. Include home remedies and general medication guidelines.\n\nMedical Condition: {condition}\nAge: {age}\nGender: {gender}\nMedical History: {medical_history}\n\nPersonalized treatment plan including home remedies and medication guidelines:\n\n**IMPORTANT: This is for informational purposes only. Please consult a healthcare professional for proper treatment.**\n\nTreatment Plan:"
    return generate_response(prompt, max_length=1200)

# Create Gradio interface
with gr.Blocks() as app:
```

```
    gr.Markdown("# Medical AI Assistant")
    gr.Markdown("**Disclaimer: This is for informational purposes only. Always consult healthcare professionals for medical advice.**")

    with gr.Tabs():
        with gr.TabItem("Disease Prediction"):
            with gr.Row():
                with gr.Column():
                    symptoms_input = gr.Textbox(
                        label="Enter Symptoms",
                        placeholder="e.g., fever, headache, cough, fatigue...",
                        lines=4
                    )
                    predict_btn = gr.Button("Analyze Symptoms")

                with gr.Column():
                    prediction_output = gr.Textbox(label="Possible Conditions & Recommendations", lines=20)

            predict_btn.click(disease_prediction, inputs=symptoms_input, outputs=prediction_output)

        with gr.TabItem("Treatment Plans"):
            with gr.Row():
                with gr.Column():
                    condition_input = gr.Textbox(
                        label="Medical Condition",
                        placeholder="e.g., diabetes, hypertension, migraine...",
                        lines=2
                    )
                    age_input = gr.Number(label="Age", value=30)
                    gender_input = gr.Dropdown(
                        choices=["Male", "Female", "Other"],
                        label="Gender",
                        value="Male"
                    )
```

```
            history_input = gr.Textbox(
                label="Medical History",
                placeholder="Previous conditions, allergies, medications or None",
                lines=3
            )
            plan_btn = gr.Button("Generate Treatment Plan")

        with gr.Column():
            plan_output = gr.Textbox(label="Personalized Treatment Plan", lines=20)

    plan_btn.click(treatment_plan, inputs=[condition_input, age_input, gender_input, history_input], outputs=plan_output)

app.launch(share=True)
```

OUTPUT: