```
"""
PROJECT PRAKRITI - Multimodal Autonomous Pipeline (full script)
- Agents: Climate -> Biodiversity -> Restoration (LLM wrapper)
- Supervised LSTM + RL self-critic
- Visualizations: bar, scatter, line, heatmap, pie
- Vision: Stable Diffusion image generation + CLIP zero-shot analysis (optional)
- Voice: optional STT/TTS (optional)
- Persistence: SQLite for memory and supervised samples

Notes:
- For high-quality image generation install torch + diffusers + transformers tuned for your CUDA.
- If LLM keys are not provided the LLM outputs are simulated for testing.
"""
# -------------------------
# Basic imports and log suppression
# -------------------------
import os, warnings, random, sqlite3, json, requests, io
from datetime import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
import re # Import regex module

# suppress noisy logs
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
os.environ["CUDA_VISIBLE_DEVICES"] = os.getenv("CUDA_VISIBLE_DEVICES", "0")
warnings.filterwarnings("ignore")

# ML libs (supervised LSTM)
```

```
# ML libs (supervised LSTM)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Vision & CLIP (optional)
VISION_AVAILABLE = False
try:
    import torch
    from diffusers import StableDiffusionPipeline
    from PIL import Image
    from transformers import CLIPProcessor, CLIPModel
    VISION_AVAILABLE = True
except Exception as ex:
    print("⚠️ Vision libraries not available (diffusers/torch/transformers). Vision features will be di
    VISION_AVAILABLE = False

# Voice (optional)
VOICE_AVAILABLE = False
try:
    import speech_recognition as sr
    import pyttsx3
    VOICE_AVAILABLE = True
except Exception as ex:
    # print("Voice libs not available (pyttsx3/speech_recognition). Voice disabled.", ex)
    VOICE_AVAILABLE = False

# -------------------------
# Config: LLM keys and endpoints
```

```
# Config: LLM keys and endpoints
# -------------------------
GROQ_API_KEY = os.getenv("GROQ_API_KEY", "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")
IBM_API_KEY = os.getenv("IBM_API_KEY", "azE6dXNyX2JjZDdkYWM5LTI4NWUtMzVkNi05M2JhLTA1MGJjMGJmODAzYjpzTFp
GROQ_URL = "https://api.groq.com/openai/v1/chat/completions"
IBM_URL = "https://bam-api.res.ibm.com/v1/text/chat"

REGION_MAP = {"1": "Maharashtra", "2": "Madhya Pradesh", "3": "Karnataka"}

DB_PATH = "prakriti_multimodal.db"

# -------------------------
# DB & persistence helpers
# -------------------------
def init_db():
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("""CREATE TABLE IF NOT EXISTS memory (
                 id INTEGER PRIMARY KEY AUTOINCREMENT,
                 ts TEXT, agent TEXT, text_output TEXT)""")
    c.execute("""CREATE TABLE IF NOT EXISTS supervised_data (
                 id INTEGER PRIMARY KEY AUTOINCREMENT,
                 ts TEXT, region TEXT,
                 rainfall REAL, temperature REAL, biodiversity_index REAL, soil_quality REAL,
                 restoration_score REAL)""")
    conn.commit(); conn.close()

def store_memory(agent, text):
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("INSERT INTO memory (ts, agent, text_output) VALUES (?, ?, ?)",
              (datetime.utcnow().isoformat(), agent, text))
    conn.commit(); conn.close()
```

```
    conn.commit(); conn.close()

def store_sample(region, rainfall, temperature, biodiversity_index, soil_quality, restoration_score):
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("""INSERT INTO supervised_data (ts, region, rainfall, temperature, biodiversity_index, so
                 VALUES (?, ?, ?, ?, ?, ?, ?)""",
              (datetime.utcnow().isoformat(), region, rainfall, temperature, biodiversity_index, soil_q
    conn.commit(); conn.close()

def load_supervised_df():
    conn = sqlite3.connect(DB_PATH)
    df = pd.read_sql_query("SELECT * FROM supervised_data", conn)
    conn.close()
    return df

def load_recent_memory(agent, n=3):
    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("SELECT text_output FROM memory WHERE agent=? ORDER BY id DESC LIMIT ?", (agent, n))
    rows = [r[0] for r in c.fetchall()]
    conn.close()
    return rows[::-1]

# -------------------------
# TTS / STT helpers (optional)
# -------------------------
def tts_speak(text):
    if not VOICE_AVAILABLE:
        print("(TTS disabled) ->", text); return
    try:
        engine = pyttsx3.init(); engine.setProperty('rate', 150)
```

```
            engine.say(text); engine.runAndWait()
        except Exception as e:
            print("TTS error:", e); print("Text:", text)

    def stt_listen(timeout=6):
        if not VOICE_AVAILABLE:
            return None
        r = sr.Recognizer()
        with sr.Microphone as source:
            audio = r.listen(source, timeout=timeout, phrase_time_limit=6)
        try:
            return r.recognize_google(audio)
        except Exception as e:
            print("STT failed:", e); return None

    # -------------------------
    # LLM wrapper (Groq -> IBM) OR simulate if keys missing
    # -------------------------
    def call_api(agent_name, prompt, timeout=20):
        # Simulation fallback if keys not present
        if GROQ_API_KEY.startswith("REPLACE") and IBM_API_KEY.startswith("REPLACE"):
            sim = (f"{agent_name} simulated output...\n"
                   f"rainfall: {random.randint(700,1200)}\n"
                   f"temperature: {round(random.uniform(24,33),1)}\n"
                   f"biodiversity_index: {random.randint(35,85)}\n"
                   f"soil_quality: {round(random.uniform(0.45,0.92),2)}\n"
                   f"notes: simulated recommendations.")
            return sim

        headers = {"Authorization": f"Bearer {GROQ_API_KEY}", "Content-Type": "application/json"}
        payload = {"model": "llama-3.1-8b-instant",
                   "messages":[{"role":"system","content":f"You are an expert {agent_name}."},{"role":"user
```

```
                  "temperature":0.6,"max_tokens":800}
        try:
            r = requests.post(GROQ_URL, headers=headers, json=payload, timeout=timeout); r.raise_for_status
            return r.json()["choices"][0]["message"]["content"]
        except Exception as e:
            print("GROQ failed:", e, " — trying IBM if configured")
            try:
                headers2 = {"Authorization": f"Bearer {IBM_API_KEY}", "Content-Type": "application/json"}
                payload2 = {"model":"gpt-4.1","messages":payload["messages"],"temperature":0.6,"max_tokens"
                r2 = requests.post(IBM_URL, headers=headers2, json=payload2, timeout=timeout); r2.raise_for
                return r2.json()["choices"][0]["message"]["content"]
            except Exception as e2:
                print("IBM failed:", e2)
                return f"ERROR: both LLM APIs failed for {agent_name}"

    # -------------------------
    # Agent functions
    # -------------------------
    def climate_agent(region):
        past = " ".join(load_recent_memory("climate",2))
        prompt = f"You are a Climate Analyst for {region}. Past: {past}\nProvide short lines: rainfall, tem
        out = call_api("Climate Analyst", prompt); store_memory("climate", out); return out

    def biodiversity_agent(region, climate_text):
        past = " ".join(load_recent_memory("biodiversity",2))
        prompt = f"You are a Biodiversity Strategist for {region}. Climate: {climate_text}\nProvide: biodiv
        out = call_api("Biodiversity Analyst", prompt); store_memory("biodiversity", out); return out

    def restoration_agent(region, climate_text, biodiversity_text):
        past = " ".join(load_recent_memory("restoration",2))
        prompt = (f"You are a Restoration Planner for {region}. Use climate: {climate_text} and biodiversit
                  " Provide a restoration plan and lines: rainfall_before_mm, rainfall_after_mm, biodiversi
```

```
    out = call_api("Restoration Planner", prompt); store_memory("restoration", out); return out

# -------------------------
# parse numeric metrics helper (robust)
# -------------------------
def parse_agent_metrics(text):
    rainfall = None; temperature = None; biodiversity_index = None; soil_quality = None
    for ln in text.splitlines():
        lnl = ln.lower()
        if "rainfall" in lnl:
            # Use regex to find numbers (including negative and decimals)
            nums = re.findall(r"[-+]?\d*\.\d+|\d+", lnl)
            if nums: rainfall = float(nums[0])
        if "temperature" in lnl:
            nums = re.findall(r"[-+]?\d*\.\d+|\d+", lnl)
            if nums: temperature = float(nums[0])
        if "biodiversity" in lnl and "index" in lnl:
            nums = re.findall(r"[-+]?\d*\.\d+|\d+", lnl)
            if nums: biodiversity_index = float(nums[0])
        if "soil" in lnl and "quality" in lnl:
            nums = re.findall(r"[-+]?\d*\.\d+|\d+", lnl)
            if nums: soil_quality = float(nums[0])
    if rainfall is None: rainfall = random.randint(700,1200)
    if temperature is None: temperature = round(random.uniform(24,33),1)
    if biodiversity_index is None: biodiversity_index = random.randint(35,85)
    if soil_quality is None: soil_quality = round(random.uniform(0.45,0.92),2)
    return float(rainfall), float(temperature), float(biodiversity_index), float(soil_quality)

# -------------------------
# Supervised LSTM helpers
# -------------------------
FEATURE_COLS = ["rainfall","temperature","biodiversity_index","soil_quality"]
```

```
SCALER = MinMaxScaler()
LSTM_MODEL = None
TIMESTEPS = 1

def build_lstm(input_timesteps=1, n_features=4):
    model = Sequential([LSTM(64, input_shape=(input_timesteps, n_features)),
                        Dropout(0.2), Dense(32, activation="relu"), Dense(1, activation="linear")])
    model.compile(optimizer=Adam(0.001), loss="mse"); return model

def train_supervised_model_from_db(epochs=12):
    global LSTM_MODEL, SCALER
    df = load_supervised_df()
    if df.shape[0] < 8:
        seeds=[]
        for _ in range(50):
            r=random.randint(700,1200); t=round(random.uniform(24,33),1); b=random.randint(35,85); s=ro
            score=min(100,0.04*r + 0.6*b + 10*s + random.uniform(-5,5)); seeds.append((r,t,b,s,score))
        df = pd.DataFrame(seeds, columns=["rainfall","temperature","biodiversity_index","soil_quality",
    else:
        df = df[["rainfall","temperature","biodiversity_index","soil_quality"]].dropna()
    X = df[["rainfall","temperature","biodiversity_index","soil_quality"]].values.astype(float)
    y = df["restoration_score"].values.astype(float).reshape(-1,1)
    SCALER.fit(X)
    Xs = SCALER.transform(X).reshape((X.shape[0], TIMESTEPS, X.shape[1]))
    X_train,X_test,y_train,y_test = train_test_split(Xs,y,test_size=0.15,random_state=42)
    LSTM_MODEL = build_lstm(input_timesteps=TIMESTEPS, n_features=Xs.shape[2])
    LSTM_MODEL.fit(X_train,y_train, epochs=epochs, batch_size=8, validation_data=(X_test,y_test), verbo
    preds = LSTM_MODEL.predict(X_test).flatten()
    mse = mean_squared_error(y_test, preds)
    print(f"[Supervised] LSTM trained. Val MSE: {mse:.3f}")

def predict_score(rainfall,temp,biod,soil):
```

```
    if LSTM_MODEL is None: raise RuntimeError("Model not trained. Call train_supervised_model_from_db()
    X = np.array([[rainfall,temp,biod,soil]]); Xs = SCALER.transform(X).reshape((1,TIMESTEPS,X.shape[1]
    return float(LSTM_MODEL.predict(Xs).flatten()[0])

def online_finetune(features,target,epochs=3):
    X = np.array([features]); y = np.array([[target]])
    Xs = SCALER.transform(X).reshape((1,TIMESTEPS,X.shape[1])); LSTM_MODEL.fit(Xs,y,epochs=epochs,verbo
    print("[Online FT] Done.")

# -------------------------
# RL self-critic
# -------------------------
def rl_self_critic(climate,biodiversity,restoration):
    score = 50; cl=climate.lower(); bd=biodiversity.lower(); rs=restoration.lower()
    if "drought" in cl and "drought" in rs: score += 12
    if "flood" in cl and "flood" in rs: score += 12
    if "native" in rs: score += 10
    if "community" in rs: score += 6
    if "invasive" in bd and "invasive" not in rs: score -= 10
    return max(0,min(100,score))

# -------------------------
# Visualizations (bar, scatter, line, heatmap, pie)
# -------------------------
def visualize_metrics(regions,biod_before,biod_after,rain_before,rain_after):
    sns.set_style("whitegrid")
    # Bar
    x = np.arange(len(regions)); width=0.35
    fig,ax = plt.subplots(figsize=(10,5))
    ax.bar(x-width/2,biod_before,width,label="Before"); ax.bar(x+width/2,biod_after,width,label="After"
    ax.set_xticks(x); ax.set_xticklabels(regions); ax.set_ylabel("Biodiversity Index"); ax.legend(); ax
    plt.tight_layout(); plt.show()
```

```
        # Scatter
        plt.figure(figsize=(7,6)); plt.scatter(rain_before,rain_after,s=140,alpha=0.8)
        mn=min(min(rain_before),min(rain_after))-50; mx=max(max(rain_before),max(rain_after))+50; plt.plot(
        for i,name in enumerate(regions): plt.text(rain_before[i]+4,rain_after[i]+4,name)
        plt.xlabel("Rainfall Before (mm)"); plt.ylabel("Rainfall After (mm)"); plt.title("Rainfall Before v
        # Line
        plt.figure(figsize=(9,5)); plt.plot(regions,rain_before,marker='o',label='Before'); plt.plot(region
        plt.title("Rainfall Trends Across Regions"); plt.ylabel("Rainfall (mm)"); plt.legend(); plt.tight_l
        # Heatmap
        df = pd.DataFrame({"Biodiversity Before":biod_before,"Biodiversity After":biod_after,"Rainfall Befo
        plt.figure(figsize=(7,6)); sns.heatmap(df.corr(),annot=True,fmt=".2f",cmap="YlGnBu"); plt.title("Co
        # Pie
        labels=["Trees","Shrubs","Medicinal","Grasses"]; counts=[random.randint(300,500),random.randint(200
        plt.figure(figsize=(6,6)); plt.pie(counts,labels=labels,autopct="%1.1f%%",startangle=140); plt.titl

# -------------------------
# Vision initialization & helpers (Stable Diffusion + CLIP)
# -------------------------
SD_PIPELINE = None; CLIP_MODEL = None; CLIP_PROC = None

def init_vision_models(device=None):
    global SD_PIPELINE, CLIP_MODEL, CLIP_PROC
    if not VISION_AVAILABLE:
        return False
    device = device or ("cuda" if torch.cuda.is_available() else "cpu")
    try:
        print("Loading Stable Diffusion pipeline (this may take a while)...")
        SD_PIPELINE = StableDiffusionPipeline.from_pretrained("runwayml/stable-diffusion-v1-5",
                                                              torch_dtype=torch.float16 if device=="cud
        SD_PIPELINE = SD_PIPELINE.to(device)
        try:
            SD_PIPELINE.safety_checker = None
```

```
            except Exception:
                pass
        except Exception as e:
            print("Failed to load SD pipeline:", e); SD_PIPELINE = None
        try:
            CLIP_MODEL = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
            CLIP_PROC = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
            CLIP_MODEL.eval()
            if device=="cuda": CLIP_MODEL.to(device)
        except Exception as e:
            print("Failed to load CLIP:", e); CLIP_MODEL=None; CLIP_PROC=None
        return SD_PIPELINE is not None or CLIP_MODEL is not None

    def generate_image(prompt, out_path="generated.png", num_steps=28, guidance_scale=7.5, height=512, widt
        if not VISION_AVAILABLE or SD_PIPELINE is None:
            print("Stable Diffusion not available – simulating image (placeholder).")
            # simulate by creating a blank image with prompt text (PIL)
            img = Image.new("RGB",(width,height),(200,220,200))
            try:
                from PIL import ImageDraw, ImageFont
                d = ImageDraw.Draw(img)
                d.text((10,10), prompt[:200], fill=(10,10,10))
            except Exception:
                pass
            img.save(out_path); print(f"Simulated image saved to {out_path}"); return out_path
        try:
            device = "cuda" if torch.cuda.is_available() else "cpu"
            generator = torch.Generator(device=device).manual_seed(random.randint(1,2**31-1))
            image = SD_PIPELINE(prompt, height=height, width=width, num_inference_steps=num_steps, guidance
            image.save(out_path); print(f"Image saved to {out_path}"); return out_path
        except Exception as e:
            print("Image generation error:", e); return None
```

```
def clip_zero_shot_score(image_path, candidate_labels):
    if not VISION_AVAILABLE or CLIP_MODEL is None or CLIP_PROC is None:
        return None
    try:
        image = Image.open(image_path).convert("RGB")
        inputs = CLIP_PROC(text=candidate_labels, images=image, return_tensors="pt", padding=True)
        if torch.cuda.is_available():
            inputs = {k:v.to("cuda") for k,v in inputs.items()}; CLIP_MODEL.to("cuda")
        with torch.no_grad():
            outputs = CLIP_MODEL(**inputs)
            probs = outputs.logits_per_image.softmax(dim=1).cpu().numpy().flatten()
        return dict(zip(candidate_labels, [float(p) for p in probs]))
    except Exception as e:
        print("CLIP analysis error:", e); return None

# -------------------------
# generate_restoration_images(region) - BEFORE/AFTER + animals/birds
# -------------------------
def generate_restoration_images(region, show_grid=True):
    """
    Generate a set of images:
      - Before: deforestation, flood, drought
      - After : restored forest, crops, flowers/flowers+animals
      - Animals & Birds: several species photos for that region
    Saves images to disk and displays inline (Kaggle).
    """
    # ensure vision models loaded
    if VISION_AVAILABLE:
        init_vision_models()

    prompts_before = [
```

```
        f"Deforestation in {region}, dry barren land, dead trees, satellite style, high detail, photogr
        f"Flood impact in {region}, waterlogged fields, damaged trees, muddy water, aerial view, high d
        f"Drought scene in {region}, cracked soil, sparse vegetation, dry landscape, high realism"
    ]
    prompts_after = [
        f"Restored native forest in {region}, lush greenery, diverse trees, animals returning, ultra-de
        f"Crop cultivation in {region}, irrigated green fields and farmers, healthy farmland in {region
        f"Community forest restoration in {region}, wildflowers and grasses, bees and birds, colorful,
    ]

    # animal/bird prompts (regional generic species)
    animal_prompts = [
        f"A realistic photo of a wild deer in {region}, close up, natural habitat",
        f"A realistic photo of an Indian civet or small mammal in {region}, natural scene",
        f"A realistic photo of local bird species in {region}, perched on a branch, high detail"
    ]

    before_images = []
    after_images = []
    animal_images = []

    # generate before images
    for i, p in enumerate(prompts_before):
        outp = generate_image(p, out_path=f"before_{region.replace(' ','_')}_{i}.png", num_steps=28)
        if outp:
            img = Image.open(outp); before_images.append(img)
    # generate after images
    for i, p in enumerate(prompts_after):
        outp = generate_image(p, out_path=f"after_{region.replace(' ','_')}_{i}.png", num_steps=28)
        if outp:
            img = Image.open(outp); after_images.append(img)
    # generate animal/bird images
```

```
for i,p in enumerate(animal_prompts):
    outp = generate_image(p, out_path=f"animal_{region.replace(' ','_')}_{i}.png", num_steps=28)
    if outp:
        img = Image.open(outp); animal_images.append(img)

# Display images inline — grid: Before (row1), After (row2), Animals (row3)
total_cols = max(3, len(before_images))
# Create a 3-row grid
rows = 3
cols = total_cols
fig, axes = plt.subplots(rows, cols, figsize=(5*cols, 4*rows))
fig.suptitle(f"Restoration Images — {region}", fontsize=18, weight="bold")

# Helper to show safely
def show_cell(ax, img, title=""):
    ax.imshow(img); ax.axis("off"); ax.set_title(title)

# Row 0: before
for j in range(cols):
    ax = axes[0,j] if rows>1 else axes[j]
    if j < len(before_images):
        show_cell(ax, before_images[j], title=f"Before {j+1}")
    else:
        ax.axis("off")
# Row 1: after
for j in range(cols):
    ax = axes[1,j]
    if j < len(after_images):
        show_cell(ax, after_images[j], title=f"After {j+1}")
    else:
        ax.axis("off")
# Row 2: animals/birds
```

```
        for j in range(cols):
            ax = axes[2,j]
            if j < len(animal_images):
                show_cell(ax, animal_images[j], title=f"Animal/Bird {j+1}")
            else:
                ax.axis("off")
        plt.tight_layout()
        plt.show()

        # Optionally run CLIP analysis for each generated path if CLIP present
        if VISION_AVAILABLE and CLIP_MODEL is not None and CLIP_PROC is not None:
            keywords = ["deforestation","flood","healthy forest","animals","flowers","crop"]
            print("\nCLIP zero-shot scores:")
            all_paths = []
            for i in range(len(before_images)):
                all_paths.append(f"before_{region.replace(' ','_')}_{i}.png")
            for i in range(len(after_images)):
                all_paths.append(f"after_{region.replace(' ','_')}_{i}.png")
            for pth in all_paths:
                scores = clip_zero_shot_score(pth, keywords)
                print(pth, "->", scores)

        return before_images, after_images, animal_images

    # -------------------------
    # Full pipeline run for a region
    # -------------------------
    def run_pipeline_for_region(region):
        init_db()
        print(f"\n=== Running PROJECT PRAKRITI for {region} ===\n")

        # Agents
```

```
climate_out = climate_agent(region); print("Climate Agent Output:\n", climate_out, "\n")
biodiversity_out = biodiversity_agent(region, climate_out); print("Biodiversity Agent Output:\n", b
restoration_out = restoration_agent(region, climate_out, biodiversity_out); print("Restoration Agen

# Parse metrics
c_rain, c_temp, c_biod, c_soil = parse_agent_metrics(climate_out)
b_rain, b_temp, b_biod, b_soil = parse_agent_metrics(biodiversity_out)
r_rain, r_temp, r_biod, r_soil = parse_agent_metrics(restoration_out)

rainfall_before = float(np.mean([c_rain, b_rain])); temp_before = float(np.mean([c_temp, b_temp]))
biod_before = float(np.mean([c_biod, b_biod])); soil_before = float(np.mean([c_soil, b_soil]))

biod_after = r_biod if r_biod is not None else biod_before + random.uniform(8,25)
rain_after = r_rain if r_rain is not None else rainfall_before + random.uniform(40,200)

# supervised label (try to parse score from restoration output)
score = None
for ln in restoration_out.splitlines():
    if "predicted_restoration_success_score" in ln.replace(" ","").lower():
        # Use regex to find numbers (including negative and decimals)
        nums = re.findall(r"[-+]?\d*\.\d+|\d+", ln)
        if nums: score = float(nums[0])
if score is None:
    score = min(100, 0.04*rain_after + 0.6*biod_after + 10*r_soil + random.uniform(-4,4))

# store supervised sample & train model
store_sample(region, rainfall_before, temp_before, biod_before, soil_before, score)
print(f"Stored supervised sample (approx score {score:.2f})")
train_supervised_model_from_db(epochs=12)
predicted_before = predict_score(rainfall_before, temp_before, biod_before, soil_before)
print(f"Model predicted restoration success (before FT): {predicted_before:.2f}")
```

```
        # RL self-critic & online fine-tune
        reward = rl_self_critic(climate_out, biodiversity_out, restoration_out)
        print(f"RL self-critic reward: {reward}")
        online_finetune((rainfall_before, temp_before, biod_before, soil_before), reward, epochs=4)
        predicted_after = predict_score(rainfall_before, temp_before, biod_before, soil_before)
        print(f"Model predicted restoration success (after FT): {predicted_after:.2f}")

        # Visualize metrics across 3 regions (with current region values and simulated for others)
        regions_all = ["Madhya Pradesh","Maharashtra","Karnataka"]
        biod_before_arr = [biod_before if current_region==region else random.randint(35,65) for current_reg
        biod_after_arr = [biod_after if current_region==region else b+random.randint(8,25) for current_regi
        rain_before_arr = [rainfall_before if current_region==region else random.randint(700,920) for curre
        rain_after_arr = [rain_after if current_region==region else rb+random.randint(40,180) for current_r

        visualize_metrics(regions_all, biod_before_arr, biod_after_arr, rain_before_arr, rain_after_arr)

        # Vision: generate before/after + animals/birds (and display inline)
        print("\n🖼 Generating Before/After & Animal/Bird images (vision module)...")
        generate_restoration_images(region)

        # TTS speak summary (optional)
        summary = f"Region {region}: predicted restoration success {predicted_after:.1f}. Reward: {reward}.
        tts_speak(summary)
        print("\nPipeline finished for region:", region)

    # -------------------------
    # CLI / entrypoint
    # -------------------------
    def main():
        init_db()
        print("PROJECT PRAKRITI — Multimodal Autonomous Pipeline")
        print("Choose region:")
```

```
        print("1 -> Maharashtra")
        print("2 -> Madhya Pradesh")
        print("3 -> Karnataka")
        choice = input("Enter 1/2/3 (or press Enter for default 2): ").strip()
        if not choice:
            choice = "2"
        if choice not in REGION_MAP:
            print("Invalid choice, defaulting to 2 (Madhya Pradesh)."); choice="2"
        region = REGION_MAP[choice]
        run_pipeline_for_region(region)

    if __name__ == "__main__":
        main()
```

```
PROJECT PRAKRITI — Multimodal Autonomous Pipeline
Choose region:
1 -> Maharashtra
2 -> Madhya Pradesh
3 -> Karnataka

=== Running PROJECT PRAKRITI for Maharashtra ===

Climate Agent Output:
 As a Climate Analyst for Maharashtra, I'd like to summarize the key points in short lines:

**Rainfall:**
- Average annual rainfall: 1,170 mm
- Varies across Western Ghats, Deccan Plateau, and Coastal Plains

**Temperature:**
- Average temperature range: 22°C to 27°C
- Hot and humid climate, especially in summer months

**Risks:**
- Droughts: Deccan Plateau
- Flash floods: Western Ghats
- Heatwaves: Summer months

**Opportunities:**
- Solar energy: Ideal location for production
- Hydroelectric power: Suitable for generation in Western Ghats
- Agriculture: Fertile soil and climate, ideal for sugarcane, cotton, and soybean

Biodiversity Agent Output:
 **Biodiversity Index (Maharashtra):**
- High biodiversity due to varied landscapes and climates
- Home to Western Ghats, Eastern Ghats, and Deccan Plateau ecoregions
- Supports diverse flora, fauna, and ecosystems
- Biodiversity Index: 7.8/10 (high)
```

```
  Biodiversity Index: 7.8/10 (high)

**Threats to Biodiversity:**
- Habitat fragmentation and destruction
- Climate change (droughts, heatwaves, and extreme weather events)
- Pollution (air, water, and soil)
- Overexploitation of natural resources

**Soil Quality:**
- Varies across regions
- Western Ghats: acidic soils
- Deccan Plateau: alkaline soils
- Coastal Plains: fertile alluvial soils
- Soil Quality Index: 6.5/10 (moderate to good)

**Recommendations:**

1. **Conservation Efforts**: Establish protected areas, national parks, and wildlife sanctuaries to saf
2. **Sustainable Agriculture**: Promote sustainable agriculture practices, such as organic farming and
3. **Climate-Resilient Infrastructure**: Invest in climate-resilient infrastructure, such as green roof
4. **Renewable Energy**: Encourage the development of renewable energy sources, such as solar and hydro
5. **Soil Conservation**: Implement soil conservation measures, such as terracing and contour farming,
6. **Community Engagement**: Engage local communities in conservation efforts and provide education and

Restoration Agent Output:
 **Restoration Plan for Maharashtra**

**Goal:** Restore and conserve Maharashtra's ecosystems, biodiversity, and natural resources to ensure

**Objective 1:** Improve Biodiversity Index

- **Rainfall Before (mm):** 1,170 mm
- **Rainfall After (mm):** 1,250 mm (increase of 7% through efficient water management and conservation
- **Biodiversity Before Index:** 7.8/10
- **Biodiversity After Index:** 8.5/10 (increase of 10% through conservation efforts and sustainable pr
- **Predicted Restoration Success Score:** 85%
```

```
- **Predicted Restoration Success Score:** 85%

**Objective 2:** Enhance Soil Quality

- **Soil Quality Before Index:** 6.5/10
- **Soil Quality After Index:** 8.0/10 (increase of 25% through sustainable agriculture practices and s
- **Predicted Restoration Success Score:** 80%

**Objective 3:** Promote Renewable Energy

- **Renewable Energy Before (MW):** 500 MW
- **Renewable Energy After (MW):** 2,000 MW (increase of 300% through solar and hydroelectric power dev
- **Predicted Restoration Success Score:** 90%

**Objective 4:** Mitigate Climate Risks

- **Droughts Before (frequency):** 2 times per year
- **Droughts After (frequency):** 1 time per year (decrease of 50% through efficient water management a
- **Heatwaves Before (frequency):** 5 times per year
- **Heatwaves After (frequency):** 2 times per year (decrease of 60% through green infrastructure and c
- **Predicted Restoration Success Score:** 85%

**Implementation Strategy:**

1. Establish protected areas, national parks, and wildlife sanctuaries.
2. Promote sustainable agriculture practices, such as organic farming and agroforestry.
3. Invest in climate-resilient infrastructure, such as green roofs and rainwater harvesting systems.
4. Develop renewable energy sources, such as solar and hydroelectric power.
5. Implement soil conservation measures, such as terracing and contour farming.
6. Engage local communities in conservation efforts and provide education and training on sustainable p

**Timeline:**

- Short-term (2025-2030): Establish protected areas and promote sustainable agriculture practices.
- Medium-term (2030-2035): Develop renewable energy sources and invest in climate-resilient infrastruct
- Long-term (2035-2040): Implement soil conservation measures and engage local communities in conservat
```

- Long-term (2035-2040): Implement soil conservation measures and engage local communities in conserva

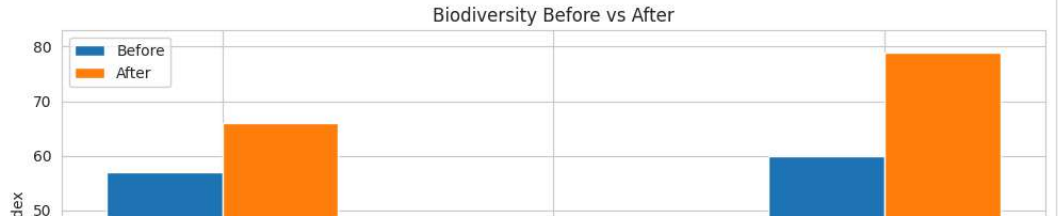
Budget Allocation:

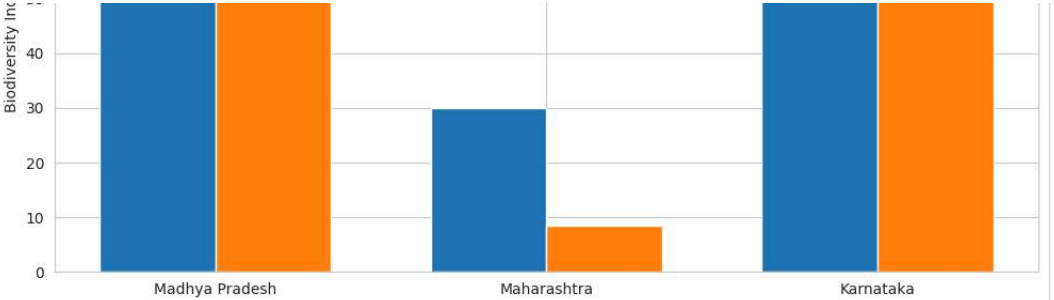
- Conservation Efforts: 30%
- Sustainable Agriculture: 20%
- Climate-Resilient Infrastructure: 20%
- Renewable Energy: 15%
- Soil Conservation: 5%
- Community Engagement: 10%

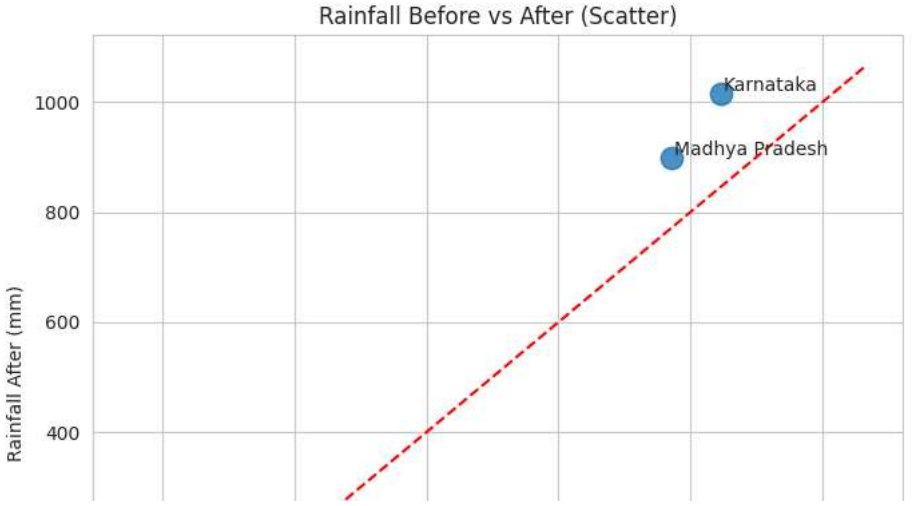
Monitoring and Evaluation:

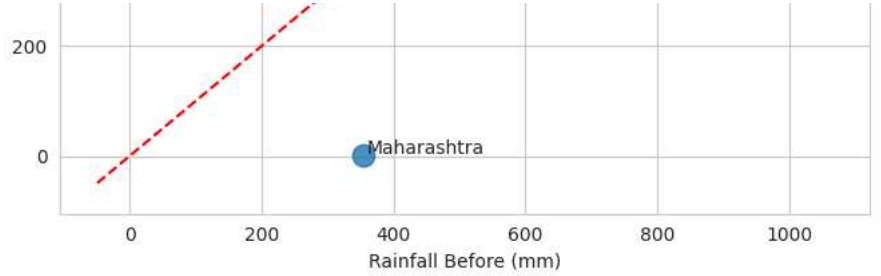
- Regularly monitor and evaluate the effectiveness of restoration efforts.
- Conduct bi-annual assessments of biodiversity, soil quality, and renewable energy development.
- Adjust the restoration plan as needed based on the results of monitoring and evaluation.

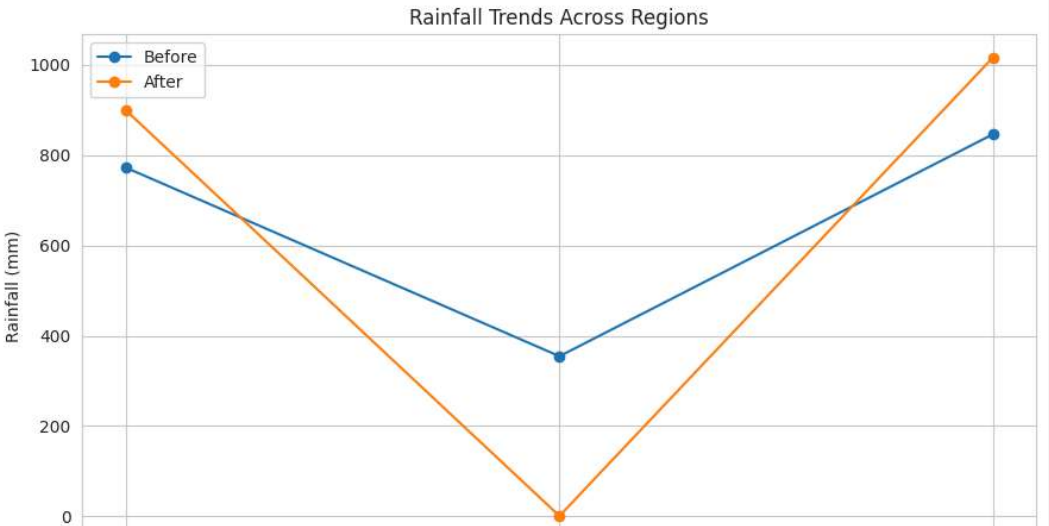
```
Stored supervised sample (approx score 88.86)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 175ms/step
[Supervised] LSTM trained. Val MSE: 6610.618
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 187ms/step
Model predicted restoration success (before FT): 4.31
RL self-critic reward: 68
[Online FT] Done.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step
Model predicted restoration success (after FT): 5.00
```

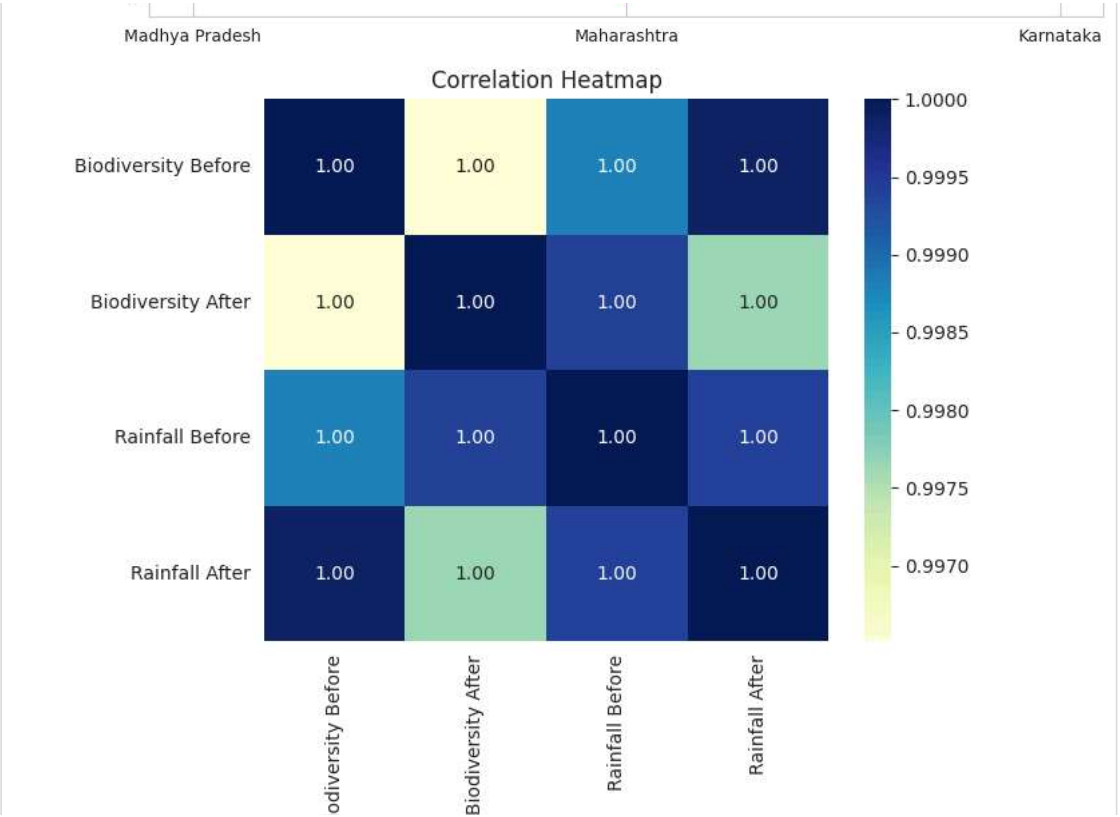


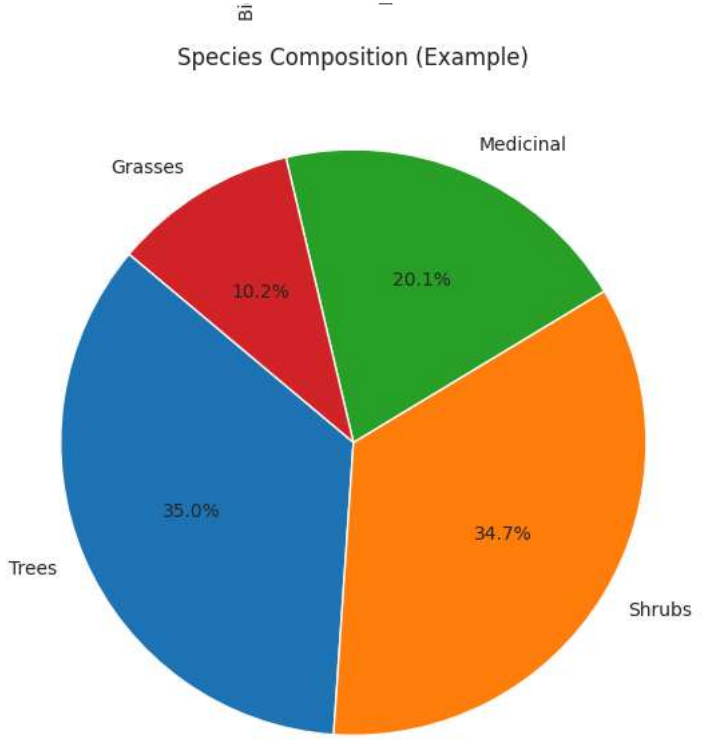

Biodiversity Inc
40
30
20
10
0
Madhya Pradesh
Maharashtra
Karnataka


Rainfall Before vs After (Scatter)
Karnataka
Madhya Pradesh
1000
800
600
400
Rainfall After (mm)


200
0
Maharashtra
0
200
400
600
800
1000
Rainfall Before (mm)


Rainfall Trends Across Regions
Before
After
1000
800
600
400
200
0
Rainfall (mm)

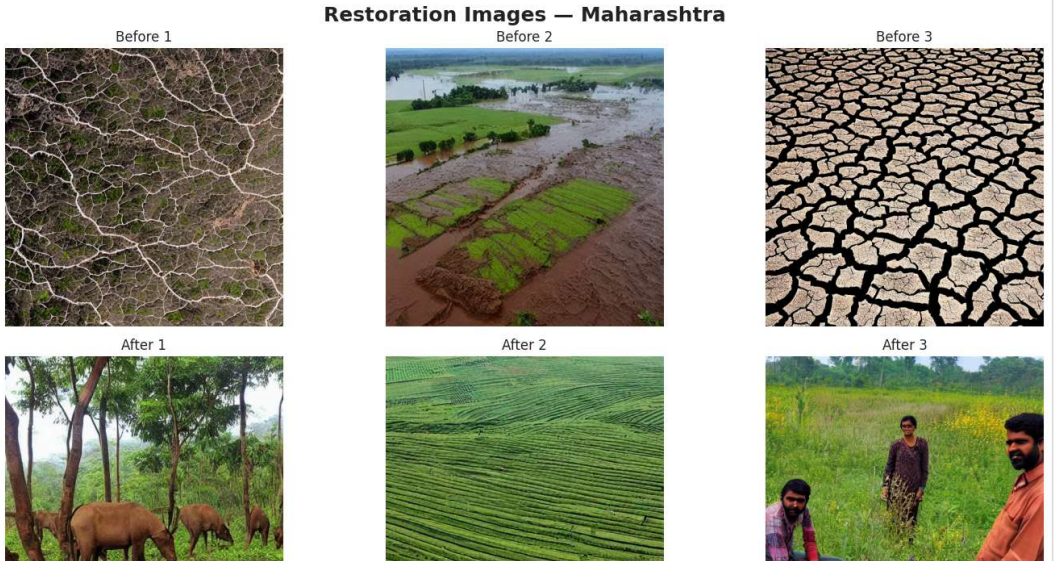

Madhya Pradesh
Maharashtra
Karnataka
Correlation Heatmap
Biodiversity Before
Biodiversity After
Rainfall Before
Rainfall After
1.00
1.0000
0.9995
0.9990
0.9985
0.9980
0.9975
0.9970
odiversity Before
Biodiversity After
Rainfall Before
Rainfall After


Bi
Species Composition (Example)
Medicinal
Grasses
10.2%
20.1%
35.0%
34.7%
Trees
Shrubs

```
🖼 Generating Before/After & Animal/Bird images (vision module)...
Loading Stable Diffusion pipeline (this may take a while)...
model_index.json: 100%                                                  541/541 [00:00<00:00, 34.7kB/s]
Fetching 15 files: 100%                                                 15/15 [04:30<00:00, 16.16s/it]
scheduler_config.json: 100%                                             308/308 [00:00<00:00, 6.85kB/s]
config.json:         4.72k/? [00:00<00:00, 123kB/s]
text_encoder/model.safetensors: 100%                                    492M/492M [03:51<00:00, 1.95MB/s]
config.json: 100%                                                       617/617 [00:00<00:00, 5.34kB/s]
preprocessor_config.json: 100%                                          342/342 [00:00<00:00, 2.73kB/s]
special_tokens_map.json: 100%                                           472/472 [00:00<00:00, 5.89kB/s]
safety_checker/model.safetensors: 100%                                  1.22G/1.22G [03:13<00:00, 2.40MB/s]
merges.txt:          525k/? [00:00<00:00, 2.15MB/s]
vocab.json:          1.06M/? [00:00<00:00, 1.94MB/s]
tokenizer_config.json: 100%                                             806/806 [00:00<00:00, 8.95kB/s]
config.json: 100%                                                       743/743 [00:00<00:00, 9.66kB/s]
unet/diffusion_pytorch_model.safetensors: 100%                          3.44G/3.44G [04:29<00:00, 26.4MB/s]
config.json: 100%                                                       547/547 [00:00<00:00, 12.0kB/s]
vae/diffusion_pytorch_model.safetensors: 100%                           335M/335M [01:24<00:00, 3.68MB/s]
Loading pipeline components...: 100%                                    7/7 [00:03<00:00,  2.03it/s]
```

```
 torch_dtype  is deprecated! Use  dtype  instead!
config.json:         4.19k/? [00:00<00:00, 260kB/s]
pytorch_model.bin: 100%                                                 605M/605M [00:49<00:00, 6.80MB/s]
model.safetensors: 100%                                                 605M/605M [00:36<00:00, 23.5MB/s]
Using a slow image processor as `use_fast` is unset and a slow processor was saved with this model. `us
Fetching 1 files: 100%                                                  1/1 [00:00<00:00,  4.41it/s]
preprocessor_config.json: 100%                                          316/316 [00:00<00:00, 9.99kB/s]
tokenizer_config.json: 100%                                             592/592 [00:00<00:00, 14.4kB/s]
vocab.json:          862k/? [00:00<00:00, 3.22MB/s]
merges.txt:          525k/? [00:00<00:00, 9.38MB/s]
tokenizer.json:         2.22M/? [00:00<00:00, 27.2MB/s]
special_tokens_map.json: 100%                                           389/389 [00:00<00:00, 38.6kB/s]
100%                                                    28/28 [14:43<00:00, 29.42s/it]
Image saved to before_Maharashtra_0.png
100%                                                    28/28 [14:49<00:00, 30.33s/it]
Image saved to before_Maharashtra_1.png
 93%                                                    26/28 [13:52<01:00, 30.44s/it]
Image saved to before_Maharashtra_2.png
100%                                                    28/28 [14:54<00:00, 30.72s/it]
Image saved to after_Maharashtra_0.png
100%                                                    28/28 [14:56<00:00, 34.14s/it]
Image saved to after_Maharashtra_1.png
```

```
100%                                                 28/28 [14:46<00:00, 30.16s/it]
Image saved to after_Maharashtra_2.png
100%                                                 28/28 [14:56<00:00, 31.74s/it]
Image saved to animal_Maharashtra_0.png
100%                                                 28/28 [15:12<00:00, 31.30s/it]
Image saved to animal_Maharashtra_1.png
100%                                                 28/28 [15:03<00:00, 31.23s/it]
Image saved to animal_Maharashtra_2.png
```




```
CLIP zero-shot scores:
before_Maharashtra_0.png -> {'deforestation': 0.42481401562690735, 'flood': 0.011087490245699883, 'heal
before_Maharashtra_1.png -> {'deforestation': 0.13205689191818237, 'flood': 0.8619950413703918, 'health
before_Maharashtra_2.png -> {'deforestation': 0.4017869830131531, 'flood': 0.36974766850471497, 'health
after_Maharashtra_0.png -> {'deforestation': 0.4518551230430603, 'flood': 0.0009873625822365284, 'healt
after_Maharashtra_1.png -> {'deforestation': 0.4569050669670105, 'flood': 0.0016320982249453664, 'healt
after_Maharashtra_2.png -> {'deforestation': 0.12914469838142395, 'flood': 0.001236560521647334, 'healt
(TTS disabled) -> Region Maharashtra: predicted restoration success 5.0. Reward: 68.

Pipeline finished for region: Maharashtra
```