🌍 AIDER: AI-Powered Early Warning and Disaster Response Assistant[¶](https://www.kaggle.com/code/bellomalik/aider-bello-malik#%F0%9F%8C%8D-AIDER:-AI-Powered-Early-Warning-and-Disaster-Response-Assistant)

🔥 Project Motivation

As climate change accelerates, disasters like floods, wildfires, storms, and droughts are striking harder and faster. Governments and responders need better tools for situational awareness and rapid decision-making. Unfortunately, traditional early warning systems are often slow, localized, or inaccessible. As well, while solutions like AIDER have been developed at high-level in developed countries, Nigeria and some third-world African countries still grapples with a handy, explanatory solution. Hence, the need for a lightweight and mundane consumer-grade AI like AIDER.

AIDER (AI-Driven Early Response) is an intelligent assistant designed to help communities, responders, and authorities:

* Detect early warning signs using real-time weather, satellite, and news data.
* Analyze and summarize risks using advanced AI.
* Provide structured, actionable emergency briefs.
* Simulate emergency responses based on location and severity.

🤖 Why GenAI?

Generative AI can understand and act on complex, multimodal data — combining weather feeds, live satellite imagery, and global news events to give context-aware emergency insights. GenAI’s ability to retrieve, interpret, and simulate real-world conditions makes it ideal for disaster preparedness and response.

🎯 Project Objectives

This project aims to:

1. Ingest and process weather, satellite, and news data for any global city.
2. Use Google’s Gemini Pro to:

* Analyze this data intelligently,
* Generate emergency summaries and risk assessments,
* Recommend response actions at two levels (Authorities & Response Teams).

1. Integrate:

* 🔁 Retrieval-Augmented Generation (RAG)
* 🧠 Multimodal image + text understanding
* ✍️ Few-shot prompt engineering
* 🧾 Structured outputs (e.g., bullet points, urgency scoring)

1. Make AIDER dynamic and extensible — able to plug into real alerting systems or disaster dashboards.

🧠 GenAI Capabilities Used:

* 🧠 Multimodal Understanding: Uses Gemini Pro to analyze satellite imagery alongside text-based weather + news data.
* 📚 Retrieval-Augmented Generation (RAG): Automatically pulls in real-time disaster-related news from Serper.dev.
* 🎯 Few-shot Prompting: Provides detailed task instructions to Gemini to emulate a professional emergency analyst.
* 🧾 Structured Output Generation: Ensures consistent, professional summaries and recommendations (bullet points, urgency score, etc.).
* 🧩 Function Simulation: Designed to support future integrations like sending alerts, mapping disaster zones, or triggering workflows.

🌦️ Phase 1: Real-Time Disaster Data Collection

To build a responsive AI disaster assistant, we collect **structured weather data** and **unstructured real-world news headlines**. This ensures AIDER understands both local conditions and breaking disaster reports. We will then use OpenAI GPT to summarise risk factors in human readable text.

🌍 1. Weather Data (OpenWeatherMap)

We fetch:

* Temperature
* Humidity
* Wind speed
* Weather description (e.g. storm, rain, heatwave)
* GPS coordinates

This structured data can signal early warnings (e.g. high winds = cyclone risk).

🧠 Why Use OpenWeatherMap?

* Provides current + forecasted weather
* Includes extreme weather flags (storms, heatwaves, etc.)
* Supports geolocation or city names
* Public and free (with generous limits)

🗞️ 2. News Data (Google News via Serper.dev)

We search global or local news for disaster-related terms like:

* "flood"
* "earthquake"
* "wildfire"
* ### 3: GenAI-generated summaries of risk factors (Gemeini API) Using the info, AIDER summarizes current events, grounds its reasoning, and generates situational briefs.

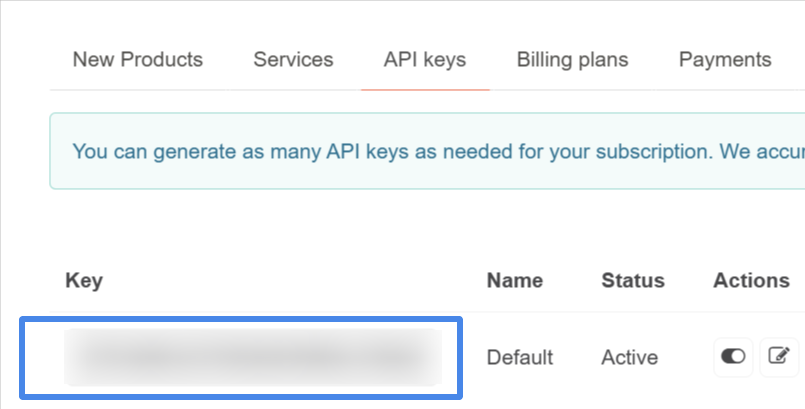
Steps in Phase 1

Step 1 : Register for the four APIs:

* OpenWeather: To get live weather
* Serper.dev: For live news headlines
* NASA API : For Weather Satelitte Imagery
* OpenAI API Key: For summarization

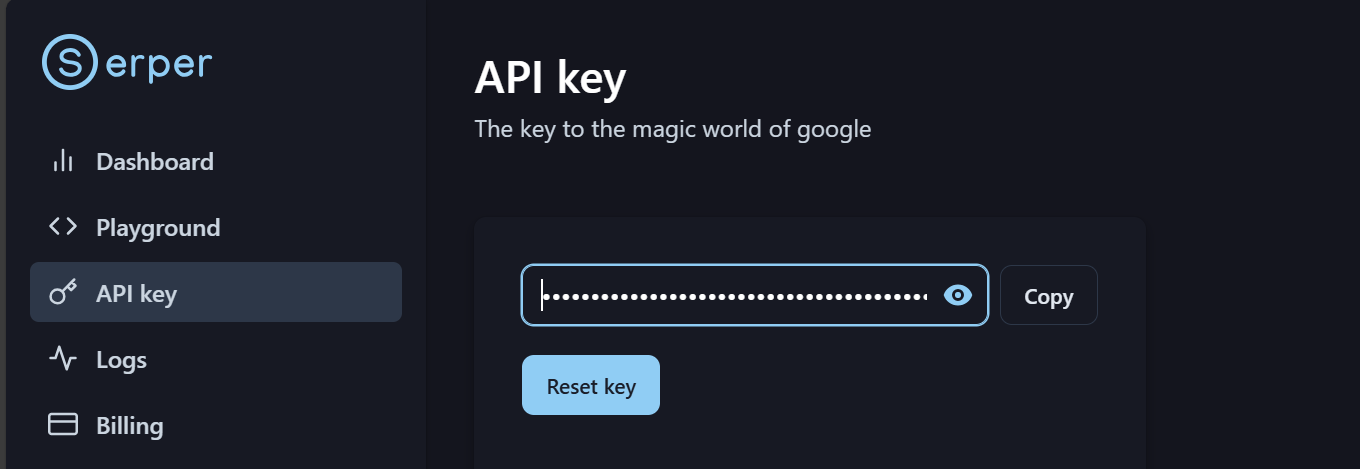
🔑 Register for Openweather API:

* Go to 👉 <https://home.openweathermap.org/users/sign_up>
* Sign up and log in
* Go to API keys, copy your default key (you can regenerate if needed).



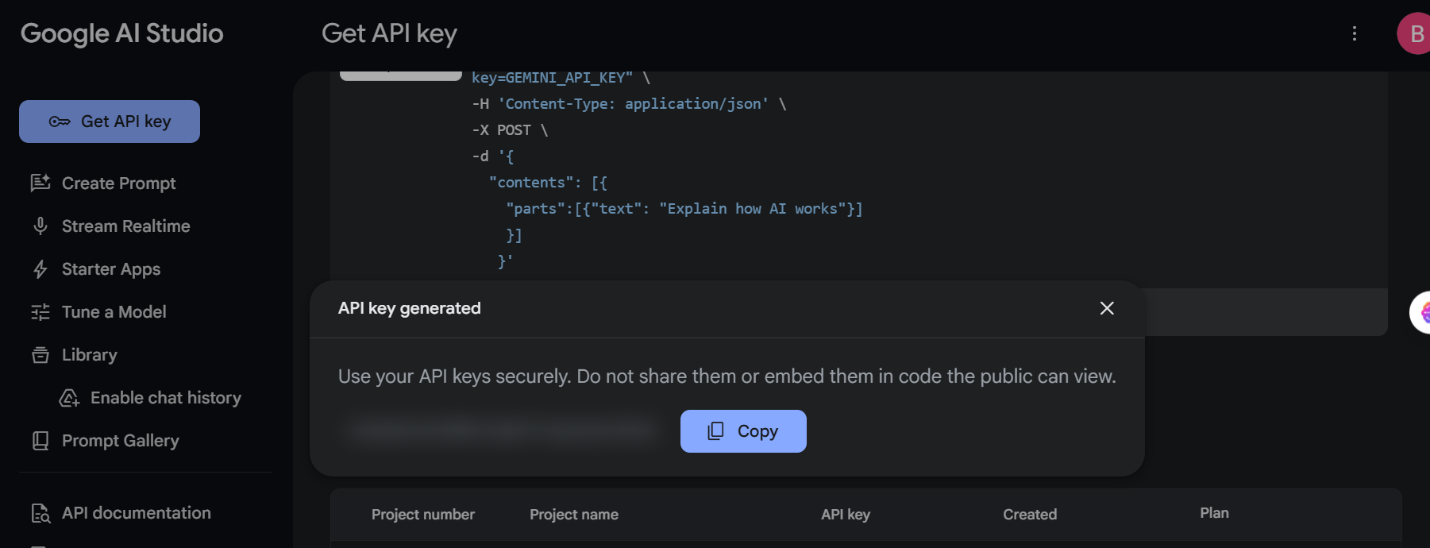
🔑 Register for Serperdev API:

* Go to 👉 [https://serper.dev](https://serper.dev/)
* Sign up
* Copy your API key from dashboard



# 🔑 Register for Gemini API Key:¶

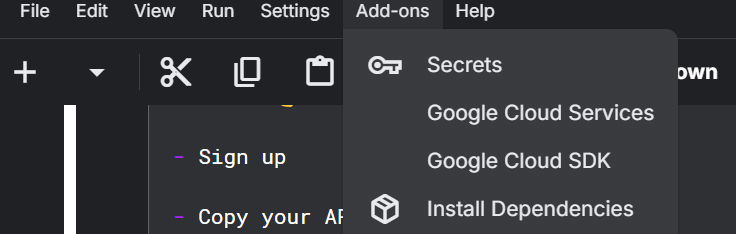
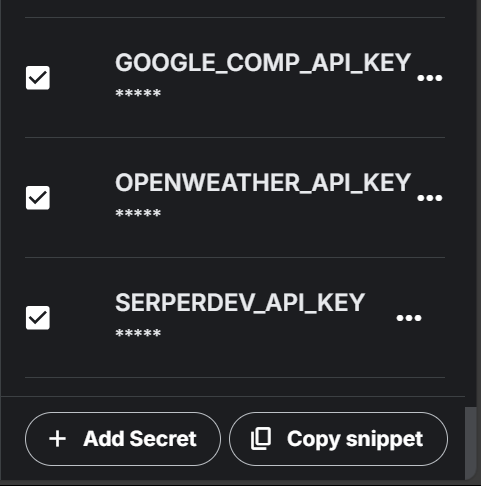
* + Follow the instructions on this site on how to register: <https://ai.google.dev/gemini-api/docs/api-key>
  + You should end up copying your API Key like this:



# 🔑 Register for NASA API Key:¶

* + Go to this website and register for API: <https://api.nasa.gov/>
  + Your API will be sent direclty to your mail (check spam in case)

# 🔑 Add all APIs as Kaggle Secret as follows:

* + Click on the menu bar "Add-on"
  + Click on Kaggle secrets and add as follows:
  + 
  + Make sure all are toggle on as shown above

Phase 2: Implementation

🔧 STEP 1: Install and Import Required Libraries

In [1]:

*# Kaggle already includes most required libraries. Install if needed.*

!pip install --quiet --upgrade google-generativeai

*# ✅ Import packages*

import requests

from datetime import datetime

import base64

from io import BytesIO

from PIL import Image

import google.generativeai as genai

from kaggle\_secrets import UserSecretsClient

from json import JSONDecodeError

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 175.4/175.4 kB 5.6 MB/s eta 0:00:00

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 37.0 MB/s eta 0:00:00

✅ Test: If no error is raised, proceed to the next step.

🔐 STEP 2: Securely Load API Keys

In [2]:

*# ✅ Fetch secrets from Kaggle's secret manager*

user\_secrets = UserSecretsClient()

gemini\_api\_key = user\_secrets.get\_secret("GOOGLE\_COMP\_API\_KEY")

openweather\_api\_key = user\_secrets.get\_secret("OPENWEATHER\_API\_KEY")

serper\_api\_key = user\_secrets.get\_secret("SERPERDEV\_API\_KEY")

nasa\_api\_key = user\_secrets.get\_secret("NASA\_API\_KEY")

secret\_value\_0 = user\_secrets.get\_secret("Bello\_Malik\_Kaggle\_API") *# This is optional*

🤖 STEP 3: Initialize Gemini Model

In [3]:

*# Configure gemini api*

genai.configure(api\_key=gemini\_api\_key)

In [4]:

*# List and display available names to know*

for model **in** genai.list\_models():

print(model.name)

models/chat-bison-001

models/text-bison-001

models/embedding-gecko-001

models/gemini-1.0-pro-vision-latest

models/gemini-pro-vision

models/gemini-1.5-pro-latest

models/gemini-1.5-pro-001

models/gemini-1.5-pro-002

models/gemini-1.5-pro

models/gemini-1.5-flash-latest

models/gemini-1.5-flash-001

models/gemini-1.5-flash-001-tuning

models/gemini-1.5-flash

models/gemini-1.5-flash-002

models/gemini-1.5-flash-8b

models/gemini-1.5-flash-8b-001

models/gemini-1.5-flash-8b-latest

models/gemini-1.5-flash-8b-exp-0827

models/gemini-1.5-flash-8b-exp-0924

models/gemini-2.5-pro-exp-03-25

models/gemini-2.5-pro-preview-03-25

models/gemini-2.0-flash-exp

models/gemini-2.0-flash

models/gemini-2.0-flash-001

models/gemini-2.0-flash-exp-image-generation

models/gemini-2.0-flash-lite-001

models/gemini-2.0-flash-lite

models/gemini-2.0-flash-lite-preview-02-05

models/gemini-2.0-flash-lite-preview

models/gemini-2.0-pro-exp

models/gemini-2.0-pro-exp-02-05

models/gemini-exp-1206

models/gemini-2.0-flash-thinking-exp-01-21

models/gemini-2.0-flash-thinking-exp

models/gemini-2.0-flash-thinking-exp-1219

models/learnlm-1.5-pro-experimental

models/gemma-3-1b-it

models/gemma-3-4b-it

models/gemma-3-12b-it

models/gemma-3-27b-it

models/embedding-001

models/text-embedding-004

models/gemini-embedding-exp-03-07

models/gemini-embedding-exp

models/aqa

models/imagen-3.0-generate-002

models/gemini-2.0-flash-live-001

In [5]:

import time

import re

import google.generativeai as genai

from google.api\_core.exceptions import ResourceExhausted

import grpc

*# 🧠 Load model (v1.5 Pro)*

model = genai.GenerativeModel("models/gemini-2.5-pro-exp-03-25")

def extract\_retry\_delay(error\_str, default\_delay=10):

*"""*

*Extract retry delay from gRPC error string. Returns delay in seconds.*

*"""*

match = re.search(r'retry\_delay\s\*{\s\*seconds:\s\*(\d+)', error\_str)

if match:

return int(match.group(1))

return default\_delay

def get\_gemini\_analysis(prompt,

temperature=0.4,

top\_k=40,

top\_p=0.9,

max\_tokens=800,

retries=5,

verbose=True):

*"""*

*Generates response from Gemini with auto retry on quota exhaustion.*

*"""*

for attempt **in** range(retries):

try:

response = model.generate\_content(

prompt,

generation\_config=genai.types.GenerationConfig(

temperature=temperature,

top\_k=top\_k,

top\_p=top\_p,

max\_output\_tokens=max\_tokens

)

)

if verbose:

print("✅ Response generated successfully.")

return response.text

except (ResourceExhausted, grpc.\_channel.\_InactiveRpcError) as e:

error\_msg = str(e)

delay = extract\_retry\_delay(error\_msg)

print(f"⚠️ Quota or gRPC limit hit. Retrying in **{**delay**}** seconds... (Attempt **{**attempt + 1**}**/**{**retries**}**)")

time.sleep(delay)

except **Exception** as e:

print("❌ Unexpected error:", str(e))

break

return "⚠️ Failed to get a response after retries"

Optional: ✅ Test: Try a test completion.

In [6]:

prompt = "Give a one-line weather alert for potential flooding in Jakarta."

response = get\_gemini\_analysis(prompt)

print("**\n**💬 Gemini Response:**\n**", response)

✅ Response generated successfully.

💬 Gemini Response:

\*\*ALERT: Heavy rain forecast for Jakarta increases risk of localized flooding; monitor conditions and avoid low-lying areas.\*\*

🌦️ STEP 4: Fetch Real-Time Weather Data

In [7]:

def get\_weather(city, api\_key=openweather\_api\_key):

url = (

"https://api.openweathermap.org/data/2.5/weather"

f"?q=**{**city**}**&appid=**{**api\_key**}**&units=metric"

)

resp = requests.get(url)

if resp.status\_code != 200:

print(f"❌ Weather fetch failed (**{**resp.status\_code**}**): **{**resp.text**}**")

return None

data = resp.json()

return {

"temperature": f"**{**data['main']['temp']**}** °C",

"condition": data['weather'][0]['description'],

"humidity": f"**{**data['main']['humidity']**}**%",

"wind": f"**{**data['wind']['speed']**}** m/s",

"cloudiness": f"**{**data['clouds']['all']**}**%"

}

✅ Test:

In [8]:

weather = get\_weather("Lagos")

print(weather)

{'temperature': '29.74 °C', 'condition': 'overcast clouds', 'humidity': '67%', 'wind': '2.02 m/s', 'cloudiness': '97%'}

📰 STEP 5: Fetch Disaster News via Serper

The number of news is limited to 5 here as proof of concept but we can get way more

In [9]:

def get\_disaster\_news(query, api\_key=serper\_api\_key, num\_results=5):

url = "https://google.serper.dev/news"

headers = {"X-API-KEY": api\_key}

resp = requests.post(url, headers=headers, json={"q": query})

if resp.status\_code != 200:

print(f"❌ News fetch failed (**{**resp.status\_code**}**): **{**resp.text**}**")

return []

data = resp.json()

return [

{

"title": item.get("title"),

"link": item.get("link"),

"snippet": item.get("snippet"),

"source": item.get("source")

}

for item **in** data.get("news", [])[:num\_results]

]

✅ Test:

In [10]:

news = get\_disaster\_news("disaster in Lagos")

for n **in** news:

print(n["title"])

(PDF) Pre-Disaster Preparedness/Prevention and Mitigation Strategies for Floods: A Use Case of Lagos, Nigeria

Passengers injured as US-bound flight declares emergency, returns to Lagos

NCAA Investigates United Airlines’ Emergency Return to Lagos after 6 Seriously Injured

Lagos flood: Lekki, Oshodi, Iyano-Oworo hit as early morning continuous rain cause damage

Panic as US-bound United flight makes emergency return to Lagos, passengers injured

🛰️ STEP 6: Fetch & Encode Satellite Imagery from NASA API

Fetches a satellite image from NASA's Earth Imagery API for a given latitude and longitude on the current date. If an image is returned, it is converted to base64 encoding to allow easy embedding or transmission to the Gemini API. Encoding ensures the binary image data can be handled safely in text-based formats like JSON or HTML.

In [11]:

def get\_satellite\_image\_base64(lat, lon, api\_key=nasa\_api\_key):

date = datetime.today().strftime('%Y-%m-**%d**')

url = "https://api.nasa.gov/planetary/earth/imagery"

params = {

"lat": lat,

"lon": lon,

"date": date,

"dim": 0.1,

"cloud\_score": True,

"api\_key": api\_key

}

resp = requests.get(url, params=params)

ctype = resp.headers.get("Content-Type", "")

if "application/json" **in** ctype:

try:

meta = resp.json()

except JSONDecodeError:

print("❌ Invalid JSON from NASA.")

return {"base64": None, "cloud\_score": None}

img\_url = meta.get("url")

cloud\_score = meta.get("cloud\_score")

if **not** img\_url:

print("❌ No image URL in NASA response.")

return {"base64": None, "cloud\_score": cloud\_score}

img\_bytes = requests.get(img\_url).content

elif resp.status\_code == 200 **and** ("image/png" **in** ctype **or** resp.content.startswith(b"**\x89**PNG")):

img\_bytes = resp.content

cloud\_score = None

else:

print(f"❌ NASA image fetch failed: **{**resp.status\_code**}**")

return {"base64": None, "cloud\_score": None}

b64 = base64.b64encode(img\_bytes).decode()

return {"base64": b64, "cloud\_score": cloud\_score}

✅ Test:

In [12]:

sat = get\_satellite\_image\_base64(6.5244, 3.3792)

print("Base64 Present:", bool(sat["base64"]))

Base64 Present: True

✨ STEP 7: Satellite Image Analysis via Gemini

This step will help confirm if the API got the satellite image and can analyse it correclty to understand the climate condition of selected area

In [13]:

def analyze\_satellite\_image(lat, lon, base64\_img):

try:

img\_prompt = [

{"text": f"Analyze this satellite image at lat=**{**lat**}**, lon=**{**lon**}** for any environmental anomalies."},

{"inline\_data": {"mime\_type": "image/png", "data": base64\_img}}

]

response = model.generate\_content(img\_prompt)

return response.text

except **Exception** as e:

return f"⚠️ Gemini image analysis failed: **{**e**}**"

✅ Test:

In [14]:

*# if sat["base64"]:*

*# satellite\_analysis = analyze\_satellite\_image(6.5244, 3.3792, sat["base64"])*

*# print(satellite\_analysis)*

*# else:*

*# print("Satellite image not available.")*

STEP 8: Build Prompt for Gemini Disaster Report

In [15]:

def generate\_disaster\_summary(weather, news\_list, satellite\_analysis, location="Global"):

today = datetime.now().strftime("**%d**/%m/%Y")

context = f"""

You are AIDER – an AI-powered early warning and disaster response assistant for global resilience planning.

LOCATION: **{**location**}**

DATE: **{**today**}**

🌦️ WEATHER:

- Temperature: **{**weather['temperature']**}**

- Condition: **{**weather['condition']**}**

- Humidity: **{**weather['humidity']**}**

- Wind: **{**weather['wind']**}**

- Cloudiness: **{**weather['cloudiness']**}**

📰 HEADLINES:

"""

for i, article **in** enumerate(news\_list, 1):

context += f"**\n{**i**}**. **{**article['title']**}** (**{**article['source']**}**) - **{**article['snippet']**}**"

context += f"""

🛰️ SATELLITE ANALYSIS:

**{**satellite\_analysis**}**

🧠 TASKS:

1. 📍 Situation Summary

2. 🚨 Predicted Disaster Risks

3. 📌 Areas Most at Risk

4. 🛠️ Recommended Actions (Authority & Response levels)

5. ⏱️ Urgency Score (1–5)

"""

return context

📡 STEP 9: Generate Gemini-Based Emergency Brief

In [16]:

def get\_gemini\_analysis(prompt):

response = model.generate\_content(prompt)

return response.text

✅ Test:

You can uncomment this code cell to see how AIDER works. But note that rate limit applies.

In [17]:

*#prompt = generate\_disaster\_summary(weather, news, satellite\_analysis, location="Lagos")*

*#final\_report = get\_gemini\_analysis(prompt)*

*#print(final\_report)*

STEP 10: Full Pipeline – AIDER in Action

In [18]:

def run\_aider(city="Lagos", lat=6.5244, lon=3.3792):

print(f"**\n**🔎 Fetching data for **{**city**}**...")

weather = get\_weather(city)

news = get\_disaster\_news(f"disaster in **{**city**}**")

sat = get\_satellite\_image\_base64(lat, lon)

satellite\_analysis = "Satellite image unavailable or could not be analyzed."

if sat["base64"]:

satellite\_analysis = analyze\_satellite\_image(lat, lon, sat["base64"])

prompt = generate\_disaster\_summary(weather, news, satellite\_analysis, location=city)

report = get\_gemini\_analysis(prompt)

print("**\n**🧠 AIDER Report:")

print(report)

✅ Test:

In [19]:

run\_aider("Lagos", lat=6.5244, lon=3.3792)

🔎 Fetching data for Lagos...

🧠 AIDER Report:

Okay, processing the provided data for Lagos.

---

\*\*AIDER Early Warning & Disaster Response Report\*\*

\*\*LOCATION:\*\* Lagos, Nigeria

\*\*DATE:\*\* 12/04/2025

\*\*TIME:\*\* (Implied Current)

\*\*1. 📍 SITUATION SUMMARY:\*\*

Lagos is currently experiencing significant disruption due to \*\*active flooding\*\* in multiple areas, specifically \*\*Lekki, Oshodi, and Iyana-Oworo\*\*, triggered by continuous heavy rainfall. The Lagos State Emergency Management Agency (LASEMA) has issued warnings advising against non-essential travel. Weather conditions remain conducive to further rainfall (overcast, high humidity, 97% cloudiness). Concurrently, a \*\*separate aviation emergency\*\* occurred involving a United Airlines flight (UA613) bound for Washington D.C., which returned to Murtala Muhammed International Airport (MMIA) Lagos due to a technical issue, resulting in passenger injuries. The Nigerian Civil Aviation Authority (NCAA) is investigating this incident. Satellite analysis confirms heavy cloud cover, limiting ground visibility, but shows no \*visible\* large-scale environmental anomalies like oil spills or major sediment plumes in the observable water bodies. Background research confirms Lagos' known vulnerability to flooding, highlighting the need for effective preparedness.

\*\*2. 🚨 PREDICTED DISASTER RISKS:\*\*

\* \*\*Primary Risk:\*\* \*\*Worsening Flash Floods & Urban Flooding:\*\* Continued or renewed heavy rainfall on saturated ground could expand flooding to other low-lying and poorly drained areas, leading to further property damage, displacement, and disruption of essential services.

\* \*\*Secondary Risk:\*\* \*\*Transportation Disruption:\*\* Severe traffic congestion and road closures due to flooded streets. Potential delays or cancellations at MMIA due to weather and the ongoing investigation into the UA613 incident.

\* \*\*Tertiary Risk:\*\* \*\*Health Hazards:\*\* Increased risk of waterborne diseases (e.g., cholera, typhoid) due to contamination of water sources by floodwaters. Risk of injuries from debris or navigating flooded areas.

\* \*\*Infrastructure Strain:\*\* Potential damage to roads, buildings, and drainage systems. Risk of localized power outages.

\*\*3. 📌 AREAS MOST AT RISK:\*\*

\* \*\*Confirmed Flooding:\*\* Lekki, Oshodi, Iyana-Oworo.

\* \*\*High Flood Risk:\*\* Other known low-lying, coastal, and poorly drained areas (e.g., parts of Victoria Island, Ikoyi, Apapa, mainland communities near canals/lagoons).

\* \*\*Airport Zone:\*\* Murtala Muhammed International Airport (MMIA) environs due to the aviation incident and potential operational impacts from weather.

\* \*\*Major Transport Arteries:\*\* Highways and major roads connecting affected areas are at high risk of impassability.

\*\*4. 🛠️ RECOMMENDED ACTIONS:\*\*

\* \*\*Authorities (LASEMA, NEMA, Lagos State Government, NCAA, FAAN):\*\*

\* \*\*Immediate:\*\* Issue widespread, clear public alerts via multiple channels (SMS, radio, TV, social media) detailing flooded areas, road closures, safety precautions, and travel advisories. Reinforce LASEMA's warning.

\* \*\*Coordination:\*\* Activate/Maintain Emergency Operations Centre (EOC) for coordinated response across agencies.

\* \*\*Response Deployment:\*\* Deploy Rapid Response Squads, LASEMA Response Units, and medical teams to affected areas (Lekki, Oshodi, Iyana-Oworo) for assessment, rescue, and first aid.

\* \*\*Infrastructure Monitoring:\*\* Monitor critical infrastructure (drainage outlets, power grid, water supply). Assess damage and prioritize repairs. Initiate emergency drain clearance where feasible and safe.

\* \*\*Aviation Incident:\*\* Continue/expedite NCAA investigation into UA613. Ensure passenger welfare and manage potential airport disruptions (FAAN/Airport Authority).

\* \*\*Shelter:\*\* Prepare designated public buildings as temporary shelters for displaced persons if necessary.

\* \*\*Forecasting:\*\* Continuously monitor meteorological forecasts for rainfall predictions.

\* \*\*Response Teams (Field Level):\*\*

\* \*\*SAR:\*\* Conduct search and rescue operations in flooded zones, prioritizing vulnerable populations.

\* \*\*Medical:\*\* Provide first aid and transport injured individuals (from floods and the flight incident) to medical facilities. Set up temporary medical points if needed.

\* \*\*Safety:\*\* Cordon off dangerously flooded areas, collapsed structures, or downed power lines.

\* \*\*Traffic Management:\*\* Assist police/LASTMA in managing traffic diversions around flooded routes.

\* \*\*Needs Assessment:\*\* Conduct rapid assessments in affected communities to identify immediate needs (food, water, shelter, medical).

\* \*\*Public:\*\*

\* \*\*Heed Warnings:\*\* Strictly follow instructions from official emergency agencies (LASEMA, NEMA).

\* \*\*Avoid Travel:\*\* Postpone non-essential travel, particularly through known flood-prone or currently affected areas.

\* \*\*Flood Safety:\*\* \*\*DO NOT\*\* walk, swim, or drive through floodwaters. Turn Around, Don't Drown®.

\* \*\*Stay Informed:\*\* Monitor official news channels and emergency alerts.

\* \*\*Prepare:\*\* If in a flood-prone area, secure valuables, move items to higher ground, and prepare a basic emergency kit. Know evacuation routes.

\* \*\*Report:\*\* Report emergencies or dangerous situations to LASEMA (using emergency numbers like 112 or 767).

\* \*\*Health:\*\* Avoid contact with floodwater. Use safe water sources. Practice good hygiene.

\*\*5. ⏱️ URGENCY SCORE: 4 / 5\*\*

\*\*Justification:\*\* Active, reported flooding in multiple densely populated areas presents an immediate threat to life and property. The ongoing adverse weather conditions suggest the situation could worsen rapidly. The concurrent aviation emergency adds complexity and strains response resources. While not yet reported as a city-wide catastrophe, the combination of active flooding and potential for escalation warrants a high urgency level requiring immediate and coordinated action.

---

\*\*AIDER // Monitoring Lagos Situation Actively.\*\*

* This code cell might take some time because of the error handling for delays and rate limit check we included

Phase 3

Building a User Friendly UI for AIDER - Optional

📘 AIDER Interface with ipywidgets in Kaggle

This demonstrates how to build a simple interactive interface using ipywidgets to input location details and run our run\_aider function

📦 Step 1: Import Required Libraries

In [20]:

import ipywidgets as widgets

from IPython.display import display, clear\_output

* ipywidgets: Used to create interactive widgets (text inputs, buttons, etc.)
* IPython.display: Used to display widgets and clear outputs dynamically

🏙️ Step 2: Create Input Widgets

This wil help create user-friendly inputs for:

* City name
* Latitude
* Longitude

In [21]:

*# City input*

city\_input = widgets.Text(

value="Lagos",

description="City:",

placeholder="Enter city name"

)

*# Latitude input*

lat\_input = widgets.FloatText(

value=6.5244,

description="Latitude:"

)

*# Longitude input*

lon\_input = widgets.FloatText(

value=3.3792,

description="Longitude:"

)

* 💡 You can pre-fill the values or leave them empty for dynamic use.

🟢 Step 3: Create Run Button and Output Display

In [22]:

*# Run button*

run\_button = widgets.Button(

description="Run AIDER",

button\_style="success" *# You can also use: 'info', 'warning', 'danger', ''*

)

*# Output area for displaying result*

output\_area = widgets.Output()

⚙️ Step 4: Define What Happens When the Button Is Clicked

* This function reads the user input and calls a backend function run\_aider with the given parameters.

In [23]:

def on\_run\_button\_clicked(b):

with output\_area:

clear\_output() *# Clear any previous output*

*# Read values from the input fields*

city = city\_input.value

lat = lat\_input.value

lon = lon\_input.value

print(f"Running AIDER for: **{**city**}** (lat: **{**lat**}**, lon: **{**lon**}**)**\n**")

*# Call the backend AIDER function*

result = run\_aider(city, lat, lon)

*# Display the output if available*

if result **is** None:

print(" ")

else:

weather, news, sat, satellite\_analysis, report = result

print("AIDER AI Report:**\n**")

print(report)

* ✅ Note: run\_aider() should be defined earlier in your notebook. It must return a tuple of 5 elements, or this script will raise an unpacking error. But here I found a way to catch that error. More later

🔗 Step 5: Connect the Button with the Callback Function

In [24]:

run\_button.on\_click(on\_run\_button\_clicked)

* This line links the click event of the button to the function above.

🧱 Step 6: Arrange and Display the Interface

In [25]:

*# Organize layout vertically and display it*

ui = widgets.VBox([city\_input, lat\_input, lon\_input, run\_button, output\_area])

display(ui)

City:

Latitude:

Longitude:

Run AIDER

* This creates a vertically stacked layout of the widgets and displays the interactive interface.