**Code:**

import os

import time

from datetime import datetime, timezone, timedelta

import requests

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

pd.set\_option("display.max\_columns", None)

API\_KEY = "988175af8865af1df3b4dec67e3deb7a"

CITY = "Karachi,PK"

USE\_COORDS = False

LAT = 31.5204

LON = 74.3587

HEAT\_INDEX\_ALERT\_C = 40.0

PLAIN\_TEMP\_ALERT\_C = 40.0

RAIN\_3H\_ALERT\_MM = 50.0

RAIN\_24H\_ALERT\_MM = 100.0

# Logging

LOG\_PATH = "weather\_log.csv"

TZ\_OFFSET\_HOURS = 5

LOCAL\_TZ = timezone(timedelta(hours=TZ\_OFFSET\_HOURS))

PLOT\_LAST\_N = 200

def \_owm\_params():

if USE\_COORDS:

return {"lat": LAT, "lon": LON, "appid": API\_KEY, "units": "metric"}

else:

return {"q": CITY, "appid": API\_KEY, "units": "metric"}

def fetch\_current\_weather():

"""Get current weather (temp, humidity, rain last 1h, wind)."""

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

r = requests.get(url, params=\_owm\_params(), timeout=20)

r.raise\_for\_status()

data = r.json()

temp = data["main"]["temp"]

humidity = data["main"]["humidity"]

wind\_speed = data.get("wind", {}).get("speed", np.nan)

rain\_1h = data.get("rain", {}).get("1h", 0.0)

return {

"temp": float(temp),

"humidity": float(humidity),

"wind\_speed": float(wind\_speed) if wind\_speed is not None else np.nan,

"rain\_1h": float(rain\_1h),

"raw": data,

}

def fetch\_forecast\_3h():

"""Get 5-day / 3-hourly forecast and compute rainfall for next 3h and 24h."""

url = "https://api.openweathermap.org/data/2.5/forecast"

r = requests.get(url, params=\_owm\_params(), timeout=20)

r.raise\_for\_status()

data = r.json()

df = pd.json\_normalize(data["list"])

df["time\_utc"] = pd.to\_datetime(df["dt"], unit="s", utc=True)

df["time\_local"] = df["time\_utc"].dt.tz\_convert(LOCAL\_TZ)

df["rain\_3h"] = df.get("rain.3h", pd.Series([0]\*len(df))).fillna(0.0)

rain\_next\_3h = float(df.iloc[0]["rain\_3h"]) if len(df) > 0 else 0.0

rain\_next\_24h = float(df.iloc[:8]["rain\_3h"].sum()) if len(df) >= 1 else 0.0

return {

"rain\_next\_3h": rain\_next\_3h,

"rain\_next\_24h": rain\_next\_24h,

"df\_forecast": df,

"raw": data,

}

def c\_to\_f(c):

return (c \* 9/5) + 32.0

def f\_to\_c(f):

return (f - 32.0) \* 5/9

def heat\_index\_celsius(temp\_c, rh):

"""

Compute Heat Index in °C using the Rothfusz regression (NOAA).

Valid mainly for T > ~26.7°C and RH > 40%. We'll still compute for reference.

"""

T = c\_to\_f(temp\_c)

R = rh

HI\_f = (-42.379 + 2.04901523\*T + 10.14333127\*R

- 0.22475541\*T\*R - 6.83783e-3\*T\*T - 5.481717e-2\*R\*R

+ 1.22874e-3\*T\*T\*R + 8.5282e-4\*T\*R\*R - 1.99e-6\*T\*T\*R\*R)

return f\_to\_c(HI\_f)

def assess\_risks(temp\_c, rh, rain\_1h, rain\_next\_3h, rain\_next\_24h):

hi\_c = heat\_index\_celsius(temp\_c, rh)

heat\_alert = (hi\_c >= HEAT\_INDEX\_ALERT\_C) or (temp\_c >= PLAIN\_TEMP\_ALERT\_C)

flood\_alert = (rain\_next\_3h >= RAIN\_3H\_ALERT\_MM) or (rain\_next\_24h >= RAIN\_24H\_ALERT\_MM)

reasons = []

if heat\_alert:

if hi\_c >= HEAT\_INDEX\_ALERT\_C:

reasons.append(f"Heat Index {hi\_c:.1f}°C ≥ {HEAT\_INDEX\_ALERT\_C}°C")

if temp\_c >= PLAIN\_TEMP\_ALERT\_C:

reasons.append(f"Temp {temp\_c:.1f}°C ≥ {PLAIN\_TEMP\_ALERT\_C}°C")

if flood\_alert:

if rain\_next\_3h >= RAIN\_3H\_ALERT\_MM:

reasons.append(f"Rain next 3h {rain\_next\_3h:.1f} mm ≥ {RAIN\_3H\_ALERT\_MM} mm")

if rain\_next\_24h >= RAIN\_24H\_ALERT\_MM:

reasons.append(f"Rain next 24h {rain\_next\_24h:.1f} mm ≥ {RAIN\_24H\_ALERT\_MM} mm")

return {

"heat\_alert": bool(heat\_alert),

"flood\_alert": bool(flood\_alert),

"heat\_index\_c": hi\_c,

"reasons": reasons,

}

def now\_local():

return datetime.now(tz=LOCAL\_TZ)

def log\_row(row\_dict, path=LOG\_PATH):

"""Append a single row to CSV (create if missing)."""

df\_row = pd.DataFrame([row\_dict])

if not os.path.exists(path):

df\_row.to\_csv(path, index=False)

else:

df\_row.to\_csv(path, mode="a", index=False, header=False)

def load\_log(path=LOG\_PATH):

if not os.path.exists(path):

return pd.DataFrame()

df = pd.read\_csv(path, parse\_dates=["time\_local"])

return df

def plot\_history(path=LOG\_PATH, last\_n=PLOT\_LAST\_N):

df = load\_log(path)

if df.empty:

print("No log data yet.")

return

df\_tail = df.tail(last\_n)

# Plot Temperature

plt.figure(figsize=(10, 4))

plt.plot(df\_tail["time\_local"], df\_tail["temp\_c"], label="Temperature (°C)")

plt.ylabel("°C")

plt.xlabel("Time")

plt.title("Temperature history")

plt.legend()

plt.tight\_layout()

plt.show()

plt.figure(figsize=(10, 4))

plt.plot(df\_tail["time\_local"], df\_tail["rain\_1h\_mm"], label="Rain last 1h (mm)")

plt.plot(df\_tail["time\_local"], df\_tail["rain\_next\_3h\_mm"], label="Rain next 3h (mm)")

plt.plot(df\_tail["time\_local"], df\_tail["rain\_next\_24h\_mm"], label="Rain next 24h (mm)")

plt.ylabel("mm")

plt.xlabel("Time")

plt.title("Rainfall history")

plt.legend()

plt.tight\_layout()

plt.show()

if not API\_KEY or API\_KEY == "PASTE\_YOUR\_OPENWEATHERMAP\_KEY\_HERE":

raise ValueError("Please paste your OpenWeatherMap API key in the API\_KEY variable and run again.")

cur = fetch\_current\_weather()

fc = fetch\_forecast\_3h()

risk = assess\_risks(

temp\_c=cur["temp"],

rh=cur["humidity"],

rain\_1h=cur["rain\_1h"],

rain\_next\_3h=fc["rain\_next\_3h"],

rain\_next\_24h=fc["rain\_next\_24h"],

)

row = {

"time\_local": now\_local(),

"city": CITY if not USE\_COORDS else f"{LAT},{LON}",

"temp\_c": cur["temp"],

"humidity\_pct": cur["humidity"],

"wind\_speed\_ms": cur["wind\_speed"],

"rain\_1h\_mm": cur["rain\_1h"],

"rain\_next\_3h\_mm": fc["rain\_next\_3h"],

"rain\_next\_24h\_mm": fc["rain\_next\_24h"],

"heat\_index\_c": risk["heat\_index\_c"],

"heat\_alert": risk["heat\_alert"],

"flood\_alert": risk["flood\_alert"],

"reasons": "; ".join(risk["reasons"]),

}

log\_row(row, LOG\_PATH)

print(f"[{row['time\_local']}] — {row['city']}")

print(f"Temp: {row['temp\_c']:.1f}°C | RH: {row['humidity\_pct']:.0f}% | Wind: {row['wind\_speed\_ms']:.1f} m/s")

print(f"Rain: last 1h {row['rain\_1h\_mm']:.1f} mm | next 3h {row['rain\_next\_3h\_mm']:.1f} mm | next 24h {row['rain\_next\_24h\_mm']:.1f} mm")

print(f"Heat Index: {row['heat\_index\_c']:.1f}°C")

if row["heat\_alert"] or row["flood\_alert"]:

print("⚠️ ALERTS:")

if row["heat\_alert"]:

print(" • Heatwave risk")

if row["flood\_alert"]:

print(" • Flood risk")

if row["reasons"]:

print(" Reasons:")

for reason in risk["reasons"]:

print(f" - {reason}")

else:

print("✅ No alerts triggered at this moment.")

plot\_history(LOG\_PATH, PLOT\_LAST\_N)

INTERVAL\_MIN = 30

def run\_once\_and\_log():

cur = fetch\_current\_weather()

fc = fetch\_forecast\_3h()

risk = assess\_risks(

temp\_c=cur["temp"],

rh=cur["humidity"],

rain\_1h=cur["rain\_1h"],

rain\_next\_3h=fc["rain\_next\_3h"],

rain\_next\_24h=fc["rain\_next\_24h"],

)

row = {

"time\_local": now\_local(),

"city": CITY if not USE\_COORDS else f"{LAT},{LON}",

"temp\_c": cur["temp"],

"humidity\_pct": cur["humidity"],

"wind\_speed\_ms": cur["wind\_speed"],

"rain\_1h\_mm": cur["rain\_1h"],

"rain\_next\_3h\_mm": fc["rain\_next\_3h"],

"rain\_next\_24h\_mm": fc["rain\_next\_24h"],

"heat\_index\_c": risk["heat\_index\_c"],

"heat\_alert": risk["heat\_alert"],

"flood\_alert": risk["flood\_alert"],

"reasons": "; ".join(risk["reasons"]),

}

log\_row(row, LOG\_PATH)

print(f"[{row['time\_local']}] Logged. Alerts: Heat={row['heat\_alert']} Flood={row['flood\_alert']}")

try:

while True:

run\_once\_and\_log()

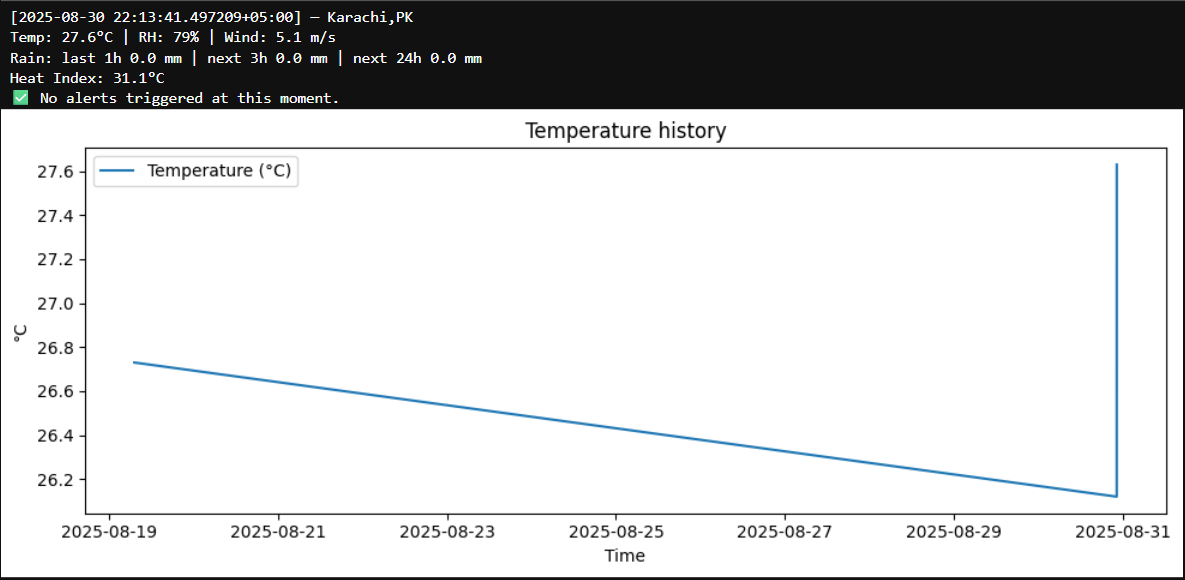
plot\_history(LOG\_PATH, PLOT\_LAST\_N)

time.sleep(INTERVAL\_MIN \* 60)

except KeyboardInterrupt:

print("Monitoring stopped by user.")

**Output:**

****

