WILDFIRESAI - DATA PIPELINE (Base Notebook)

Cell 1 — Environment Sanity Check

This initial cell performs a minimal verification that the Python environment is correctly initialized and functional.
It prints a confirmation message and the active Python version to ensure compatibility with the pipeline and its dependencies.
Although trivial, this step serves as a quick diagnostic guard against misconfigured kernels or environments before more complex imports and data pipelines are executed.

In [1]:
```
print("Python is working")
import sys
print("Version:", sys.version)
```

```
Python is working
Version: 3.11.13 (main, Jun  5 2025, 08:21:08) [Clang 14.0.6 ]
```

Cell 2 — Imports and Version Audit

This cell imports the core Python libraries required for the pipeline, covering file system management (`os`, `pathlib`), date/time handling (`datetime`), scientific computation (`numpy`, `pandas`), visualization (`matplotlib`), HTTP requests (`requests`), and the OpenAI SDK (`openai`).

In addition, it prints the active version of each dependency.
This version audit is critical for reproducibility and debugging, ensuring that all components of the environment are consistent with the pipeline requirements and compatible with future extensions.

In [2]:
```
import os
from pathlib import Path
from datetime import date, datetime, timedelta, timezone

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import requests
import openai

print("Python:", sys.version.split()[0])
print("NumPy:", np.__version__)
print("Pandas:", pd.__version__)
print("Matplotlib:", plt.matplotlib.__version__)
print("Requests:", requests.__version__)
print("OpenAI SDK:", getattr(openai, "__version__", "unknown"))
```

```
Python: 3.11.13
NumPy: 2.3.3
Pandas: 2.3.2
Matplotlib: 3.10.6
Requests: 2.32.5
OpenAI SDK: 1.107.1
```

Cell 3 — Global Configuration

This cell establishes the global configuration of the pipeline. It defines:

- Project directories (raw / processed / reports)
- Default spatio-temporal parameters (Spain bbox, 7-day window)
- Preview controls
- Environment credentials (OpenAI, NASA FIRMS) loaded once Utilities `save_df` and `preview` are defined here so they are globally available.

In [16]:
```python
# --- Project paths (shared) ---
PROJECT_ROOT = Path.cwd()
RAW_DIR       = PROJECT_ROOT / "data" / "raw"
PROCESSED_DIR = PROJECT_ROOT / "data" / "processed"
REPORTS_DIR   = PROJECT_ROOT / "reports"
for p in (RAW_DIR, PROCESSED_DIR, REPORTS_DIR):
    p.mkdir(parents=True, exist_ok=True)

# --- Default spatial/temporal parameters ---
# Spain bounding box: (W, S, E, N)
SPAIN_BBOX = (-9.5, 35.0, 3.5, 43.9)

TODAY      = date.today()
DAYS_BACK  = 7                          # default window
DATE_FROM  = TODAY - timedelta(days=DAYS_BACK)
DATE_TO    = TODAY

MAX_ROWS_PREVIEW = 10                   # preview rows in displays

# --- Credentials from environment (loaded once) ---
OPENAI_API_KEY = os.environ.get("OPENAI_API_KEY")
FIRMS_TOKEN    = os.environ.get("FIRMS_TOKEN")      # optional (Bearer en
FIRMS_MAP_KEY  = os.environ.get("FIRMS_MAP_KEY")    # required for FIRMS

def _mask(v: str | None, n: int = 6) -> str:
    return (v[:n] + "…") if v else "—"

print("Config ready.")
print(f"Paths: raw={RAW_DIR.name}, processed={PROCESSED_DIR.name}, report
print(f"BBOX: {SPAIN_BBOX} | Window: {DATE_FROM} → {DATE_TO}")
print(
    "Tokens:",
    f"OPENAI={_mask(OPENAI_API_KEY)} |",
    f"FIRMS_TOKEN={_mask(FIRMS_TOKEN)} |",
    f"FIRMS_MAP_KEY={_mask(FIRMS_MAP_KEY)}",
)

# --- Utilities made globally available ---------------------------------
def save_df(df: pd.DataFrame, path: Path) -> None:
    """Save a DataFrame to CSV and print the path (parents created if nee
    path.parent.mkdir(parents=True, exist_ok=True)
    df.to_csv(path, index=False)
    print(f"Saved: {path}")

def preview(df: pd.DataFrame, n: int = MAX_ROWS_PREVIEW) -> None:
    """Display the first n rows in a consistent way across the notebook."
    display(df.head(n))
```

```
Config ready.
Paths: raw=raw, processed=processed, reports=reports
BBOX: (-9.5, 35.0, 3.5, 43.9) | Window: 2025-09-07 → 2025-09-14
Tokens: OPENAI=sk-pro… | FIRMS_TOKEN=eyJ0eX… | FIRMS_MAP_KEY=27f8d7…
```

Cell 4 — Verify API Key (OpenAI)

This cell validates the availability of the **OpenAI API key** in the environment.

- If the key is detected, the code prints a confirmation and masks the output (showing only the prefix for verification).
- If the key is missing, a clear warning is issued, instructing the user to update their `~/.zshrc` configuration.

This ensures that downstream components relying on the OpenAI API (e.g., model queries or natural language processing tasks) will execute reliably and securely.

In [17]:
```
api_key = os.environ.get("OPENAI_API_KEY")
if api_key:
    print("API key found")
    print("Starts with:", api_key[:12] + "********")
else:
    print("API key not found. Please check your ~/.zshrc file")
```

```
API key found
Starts with: sk-proj-NFyC********
```

Cell 5 — Connect to OpenAI (Optional)

This cell establishes an optional connection to the **OpenAI API**.

- The client is initialized with the API key retrieved from the environment (see Cell 4).
- If the key is valid, a confirmation message is displayed.

This step is not strictly required for the wildfire pipeline itself but is included for extending the workflow with **AI-based analysis, natural language interfaces, or advanced modeling** in later stages.

In [18]:
```
# Optional: only if we plan to use OpenAI right now.
from openai import OpenAI
client = OpenAI(api_key=os.environ.get("OPENAI_API_KEY"))
print("OpenAI client connected")
```

```
OpenAI client connected
```

Cell 6 — Test OpenAI Connection (Optional, Paid)

This cell performs a **sanity check** of the OpenAI client connection.

- It sends a lightweight test prompt to the `gpt-4o-mini` model.
- The model is asked to produce a short haiku on *AI and wildfire resilience*.
- This verifies that the authentication and request pipeline are working.

Note: Executing this cell will consume API credits.
It should be run only once to validate connectivity and left commented or skipped in regular workflows.

In [19]:
```
# Optional test (may consume credits).
resp = client.responses.create(
    model="gpt-4o-mini",
    input="Write a 3-line haiku about AI and wildfire resilience."
)
print("Model response:\n")
print(resp.output_text)
```

```
Model response:

Machines learn to grow,
Forecast flames, sow seeds of hope—
Nature's strength enhanced.
```

Cell 7 — Verify FIRMS Token (NASA FIRMS)

This cell verifies that the **NASA FIRMS Bearer token** is available in the environment.

- The token is required for accessing FIRMS API endpoints (satellite fire detection data).
- It is loaded from the shell configuration file (`~/.zshrc`) into the environment.
- For security, only the first characters of the token are displayed.

If the token is missing, the user must check the shell configuration and reload the session with:

```
source ~/.zshrc
```

In [20]:
```
import os

firms_token = os.environ.get("FIRMS_TOKEN")

if firms_token:
    print("FIRMS token found")
    print("Starts with:", firms_token[:12] + "********")
else:
    print("FIRMS token not found. Please check your ~/.zshrc file")
```

```
FIRMS token found
Starts with: eyJ0eXAiOiJK********
```

Cell 8 — Verify FIRMS MAP_KEY (NASA FIRMS)

This cell verifies that the **NASA FIRMS MAP_KEY** (legacy API key) is available in the environment.

- The MAP_KEY is required for the **CSV-style FIRMS API**, where the key is embedded directly in the request URL.
- It is stored in the shell configuration (`~/.zshrc`) and loaded into the environment for use in this notebook.
- For security, only the first characters of the key are displayed.

If the MAP_KEY is not found, the pipeline raises an explicit error instructing the user to update and reload their environment:

```
export FIRMS_MAP_KEY="your_key_here"
source ~/.zshrc
```

In [21]:
```
import os

FIRMS_MAP_KEY = os.environ.get("FIRMS_MAP_KEY")

if FIRMS_MAP_KEY:
    print("FIRMS MAP_KEY found")
    print("Starts with:", FIRMS_MAP_KEY[:6] + "********")
else:
    raise RuntimeError(
        "FIRMS_MAP_KEY not found.\n"
        "Please check your ~/.zshrc file and reload with: source ~/.zshrc
    )
```

```
FIRMS MAP_KEY found
Starts with: 27f8d7********
```

Cell 9 — Connect to FIRMS API (MAP_KEY mode)

This cell establishes a connection to the **NASA FIRMS API** using the legacy **MAP_KEY** authentication scheme.
Key operations performed here:

1. **Authentication**
    - The `FIRMS_MAP_KEY` is embedded directly in the API request URL.
    - Ensures access to FIRMS CSV endpoints even if Bearer token support is unavailable.
2. **Spatial & Temporal Filtering**
    - The request is bounded by the Spain region (defined in `SPAIN_BBOX`).
    - The temporal window is restricted to the **last 7 days**, consistent with FIRMS near-real-time (NRT) availability.
3. **Data Retrieval**

- Fire detections are retrieved from the **VIIRS SNPP NRT** product.
- The response is parsed into a Pandas DataFrame and validated.

4. **Persistence & Preview**

   - The raw CSV is saved to the `data/raw/` directory with a timestamped filename.
   - A quick preview (`head(5)`) is displayed to confirm structure and content.

This step ensures the pipeline begins with authoritative, high-resolution fire detection data, suitable for downstream cleaning, integration with weather datasets, and scientific validation.

In [22]:
```python
from io import StringIO  # only needed here

# Ensure MAP_KEY is available (from Cell 7)
if not FIRMS_MAP_KEY:
    raise RuntimeError("FIRMS_MAP_KEY not found. Did you export it in ~/.

# Spain bounding box (W, S, E, N)
w, s, e, n = SPAIN_BBOX

# FIRMS product and temporal window
product = "VIIRS_SNPP_NRT"
days = 7  # valid range: 1..10

out_csv = RAW_DIR / "firms_last7d_es_raw.csv"

# MAP_KEY style (token in path)
url = f"https://firms.modaps.eosdis.nasa.gov/api/area/csv/{FIRMS_MAP_KEY}
params = {"west": w, "south": s, "east": e, "north": n}

r = requests.get(url, params=params, timeout=60)
if r.status_code != 200 or "Invalid" in r.text[:200]:
    raise RuntimeError(f"FIRMS API request failed: HTTP {r.status_code} 

# Parse and save
df_raw = pd.read_csv(StringIO(r.text), dtype={"acq_time": "string"})
df_raw.to_csv(out_csv, index=False)

print("FIRMS API connection successful (MAP_KEY mode)")
print(f"Rows fetched: {len(df_raw)}")
print(f"Saved raw CSV → {out_csv}")

display(df_raw.head(5))
```

```
FIRMS API connection successful (MAP_KEY mode)
Rows fetched: 450072
Saved raw CSV → /Users/evareysanchez/WildfiresAI/data/raw/firms_last7d_es_
raw.csv
```

	latitude	longitude	bright_ti4	scan	track	acq_date	acq_time	satellite	instru
0	43.49886	24.40615	312.59	0.52	0.42	2025-09-08	1	N	
1	43.94419	25.56161	304.79	0.48	0.40	2025-09-08	1	N	
2	43.97284	25.52557	298.51	0.48	0.40	2025-09-08	1	N	
3	44.07837	22.10577	301.36	0.45	0.47	2025-09-08	1	N	
4	44.07931	22.10626	302.72	0.45	0.47	2025-09-08	1	N	

Cell 10 — Open-Meteo Helper (Hourly, ERA5 Archive)

This cell defines a reusable **helper function** to query the Open-Meteo Archive API for historical weather data based on the **ERA5 reanalysis dataset**. No API key is required.

Main features:

1. **Endpoint & Variables**
   - Uses the ERA5 archive endpoint (`https://archive-api.open-meteo.com/v1/archive`).
   - Retrieves hourly variables relevant to wildfire dynamics (temperature, humidity, wind, precipitation, and surface pressure).
2. **Function `fetch_open_meteo`**
   - Inputs: latitude, longitude, start date, end date, and selected variables.
   - Returns a **Pandas DataFrame** with hourly weather records in UTC.
   - Columns include `time_utc`, meteorological variables, and coordinates (`lat`, `lon`).
3. **Error Handling & Validation**
   - Raises explicit errors if the API response is invalid (e.g., missing hourly data).
   - Ensures numeric casting of meteorological fields for consistent analysis.
4. **Persistence**
   - Provides a lightweight `save_df` utility to persist datasets under the `data/raw/` folder.
   - Facilitates traceability and reproducibility of weather datasets.

This component is critical for aligning **FIRMS fire detections** with **meteorological context**, enabling joint analyses (e.g., fire spread vs. weather dynamics) and subsequent model training.

In [24]:
```python
from typing import Iterable

# Endpoint correcto para histórico/reanálisis (ERA5):
OPEN_METEO_BASE = "https://archive-api.open-meteo.com/v1/archive"

# Variables horarias por defecto (puedes añadir/quitar)
OPEN_METEO_HOURLY = [
    "temperature_2m",
    "relative_humidity_2m",
    "dew_point_2m",
    "windspeed_10m",
    "windgusts_10m",
    "winddirection_10m",
    "precipitation",
    "surface_pressure",
]

# Pequeño helper local por si save_df aún no existe
if "save_df" not in globals():
    import pandas as pd
    from pathlib import Path
    def save_df(df: pd.DataFrame, path: Path) -> None:
        path.parent.mkdir(parents=True, exist_ok=True)
        df.to_csv(path, index=False)
        print(f"Saved: {path}")

def fetch_open_meteo(
    lat: float,
    lon: float,
    date_from: date,
    date_to: date,
    hourly: Iterable[str] = OPEN_METEO_HOURLY,
    model: str | None = "era5",  # "era5" o "era5_land"; None para auto
) -> pd.DataFrame:
    """Fetch hourly weather (UTC) for a point and date window using ERA5
    params = {
        "latitude":  round(float(lat), 5),
        "longitude": round(float(lon), 5),
        "start_date": str(date_from),
        "end_date":   str(date_to),
        "hourly":     ",".join(hourly),
        "timezone":   "UTC",
    }
    if model:
        params["models"] = model

    r = requests.get(OPEN_METEO_BASE, params=params, timeout=60)
    if r.status_code != 200:
        raise RuntimeError(f"Open-Meteo error {r.status_code}: {r.text[:2

    js = r.json()
    if "hourly" not in js or "time" not in js["hourly"]:
        raise RuntimeError("Open-Meteo response missing hourly data.")

    df = pd.DataFrame(js["hourly"]).rename(columns={"time": "time_utc"})
    df["time_utc"] = pd.to_datetime(df["time_utc"], utc=True, errors="coe
    df["lat"] = round(float(lat), 5)
    df["lon"] = round(float(lon), 5)
    # Asegurar tipos numéricos
    for c in df.columns:
        if c not in ("time_utc", "lat", "lon"):
            df[c] = pd.to_numeric(df[c], errors="coerce")
    return df
```

Cell 11 — Open-Meteo Smoke Test (Spain Centroid)

This cell performs a **sanity check** of the Open-Meteo integration by querying historical weather data at the **centroid of Spain's bounding box**. The goal is to validate that the pipeline can fetch, persist, and visualize meteorological information.

Main steps:

1. **Centroid Calculation**
   - Latitude and longitude are derived from the midpoint of Spain's bounding box (`SPAIN_BBOX` , defined in Cell 3).
2. **Weather Fetching**
   - Calls `fetch_open_meteo` (Cell 10) to retrieve ERA5 reanalysis data between `DATE_FROM` and `DATE_TO` .
   - Produces an **hourly-resolution DataFrame** with variables such as temperature, humidity, wind, and precipitation.
3. **Persistence**
   - Saves raw hourly data under `data/raw/` .
   - Computes a **daily aggregated summary** (mean temperature, mean humidity, mean windspeed, total precipitation) and stores it under `data/processed/` .
4. **Visualization**
   - Generates a **time-series plot of temperature** at the centroid (°C vs UTC time).
   - Saves the figure under the `reports/` directory for reproducibility and reporting.

This smoke test confirms that **Open-Meteo is properly integrated**, and it sets the foundation for linking weather conditions with FIRMS fire detections in later steps.

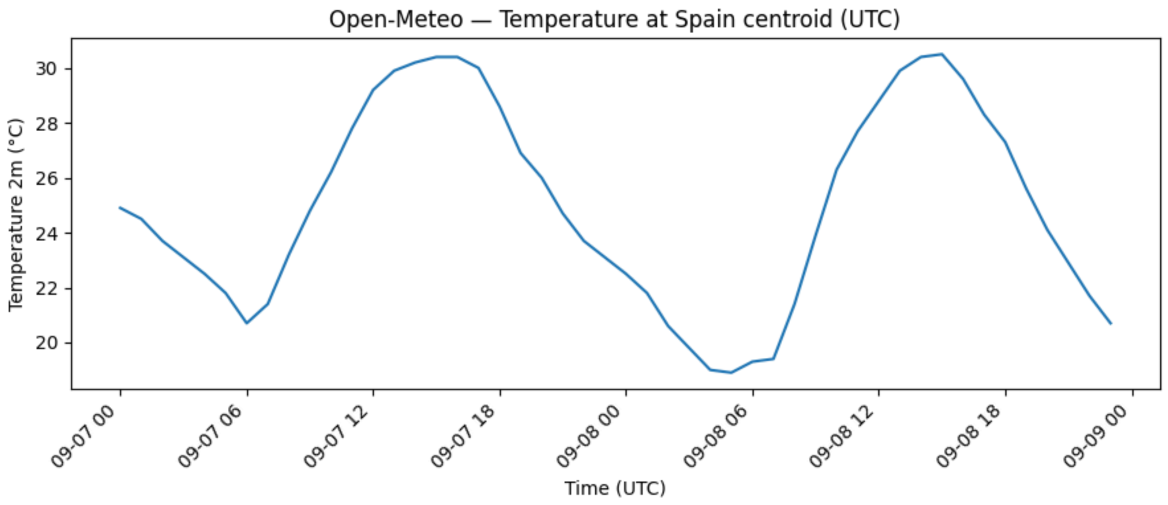
In [25]:
```
# Centroid of Spain bbox (W, S, E, N) from Global Configuration (Cell 3)
W, S, E, N = SPAIN_BBOX
lat_c = (S + N) / 2.0
lon_c = (W + E) / 2.0

# Fetch hourly weather (UTC)
df_weather_centroid = fetch_open_meteo(lat_c, lon_c, DATE_FROM, DATE_TO,

# Save raw weather CSV under data/raw/
raw_weather_path = RAW_DIR / f"openmeteo_es_centroid_{DATE_FROM}_{DATE_TO
save_df(df_weather_centroid, raw_weather_path)
```

```
# Display quick preview
print(f"Rows (hourly): {len(df_weather_centroid)}")
display(df_weather_centroid.head(10))

# Also save a daily summary (mean temperature, etc.) to processed/
daily_summary = (
    df_weather_centroid.assign(day=df_weather_centroid["time_utc"].dt.flo
    .groupby("day")
    .agg(
        temp_mean=("temperature_2m", "mean"),
        rh_mean=("relative_humidity_2m", "mean"),
        wind_mean=("windspeed_10m", "mean"),
        precip_sum=("precipitation", "sum"),
    )
    .reset_index()
)
daily_out = PROCESSED_DIR / f"openmeteo_es_centroid_daily_{DATE_FROM}_{DA
save_df(daily_summary, daily_out)

# Plot: Temperature vs time (and save figure)
plt.figure(figsize=(9,4))
plt.plot(df_weather_centroid["time_utc"], df_weather_centroid["temperatur
plt.title("Open-Meteo — Temperature at Spain centroid (UTC)")
plt.xlabel("Time (UTC)")
plt.ylabel("Temperature 2m (°C)")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Save figure to reports/
fig_path = REPORTS_DIR / f"openmeteo_temp_centroid_{DATE_FROM}_{DATE_TO}.
plt.figure(figsize=(9,4))
plt.plot(df_weather_centroid["time_utc"], df_weather_centroid["temperatur
plt.title("Open-Meteo — Temperature at Spain centroid (UTC)")
plt.xlabel("Time (UTC)")
plt.ylabel("Temperature 2m (°C)")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.savefig(fig_path, dpi=160)
plt.close()
print(f"Saved figure: {fig_path}")
```

```
Saved: /Users/evareysanchez/WildfiresAI/data/raw/openmeteo_es_centroid_202
5-09-07_2025-09-14.csv
Rows (hourly): 192
```

	time_utc	temperature_2m	relative_humidity_2m	dew_point_2m	windspee
0	2025-09-07 00:00:00+00:00	24.9	41.0	10.6	
1	2025-09-07 01:00:00+00:00	24.5	47.0	12.4	
2	2025-09-07 02:00:00+00:00	23.7	52.0	13.3	
3	2025-09-07 03:00:00+00:00	23.1	57.0	14.1	
4	2025-09-07 04:00:00+00:00	22.5	59.0	14.2	
5	2025-09-07 05:00:00+00:00	21.8	62.0	14.2	
6	2025-09-07 06:00:00+00:00	20.7	65.0	14.0	
7	2025-09-07 07:00:00+00:00	21.4	66.0	14.7	
8	2025-09-07 08:00:00+00:00	23.2	62.0	15.4	
9	2025-09-07 09:00:00+00:00	24.8	50.0	13.6	

```
Saved: /Users/evareysanchez/WildfiresAI/data/processed/openmeteo_es_centro
id_daily_2025-09-07_2025-09-14.csv
```



```
Saved figure: /Users/evareysanchez/WildfiresAI/reports/openmeteo_temp_cent
roid_2025-09-07_2025-09-14.png
```

Cell 12 — Open-Meteo at First Fire Location (Optional)

This cell enriches the FIRMS detections by fetching weather conditions at the **geographic location of the first fire point** available. It is optional, but critical for validating the integration of fire and meteorological data.

Main steps:

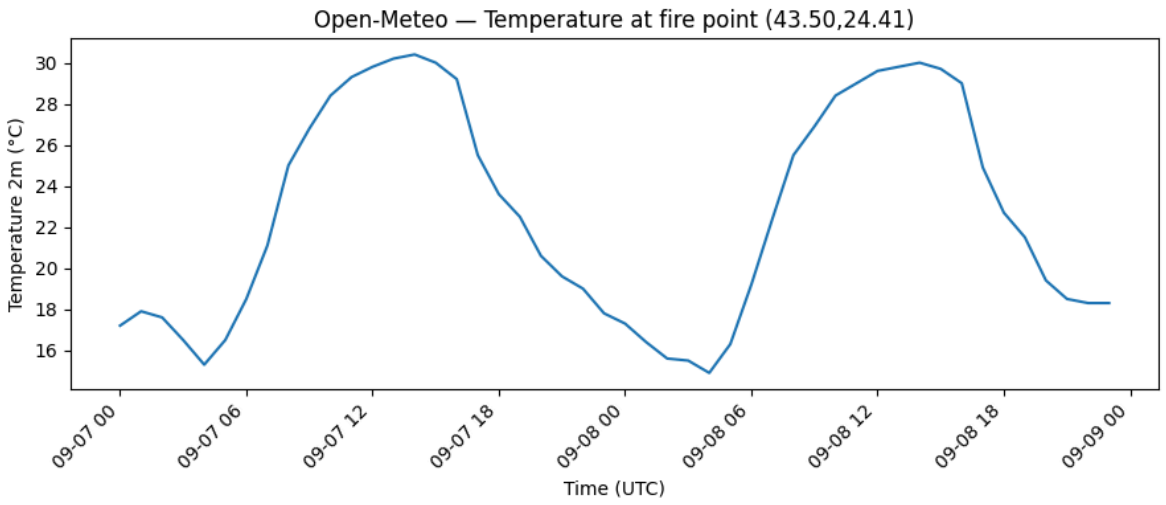
1. **Candidate Fire Location**
   - Attempts to extract latitude/longitude from `df_raw` (raw FIRMS) or `df_fires` (processed FIRMS).
   - Accepts flexible column names (`latitude/longitude` , `lat/lon` , `x/y`).
2. **Weather Retrieval**
   - If a valid fire location exists, queries Open-Meteo's ERA5 archive for hourly conditions.
   - Produces a DataFrame (`df_weather_point`) with temperature, humidity, windspeed, precipitation, and more.
3. **Persistence**
   - Saves raw hourly weather data under `data/raw/` .
   - Aggregates to **daily summaries** (mean temperature, mean humidity, mean windspeed, precipitation sum) and saves under `data/processed/` .
4. **Visualization**
   - Generates and displays a **time-series chart** of temperature at the fire location.
   - Saves the chart under `reports/` for later reporting and analysis.

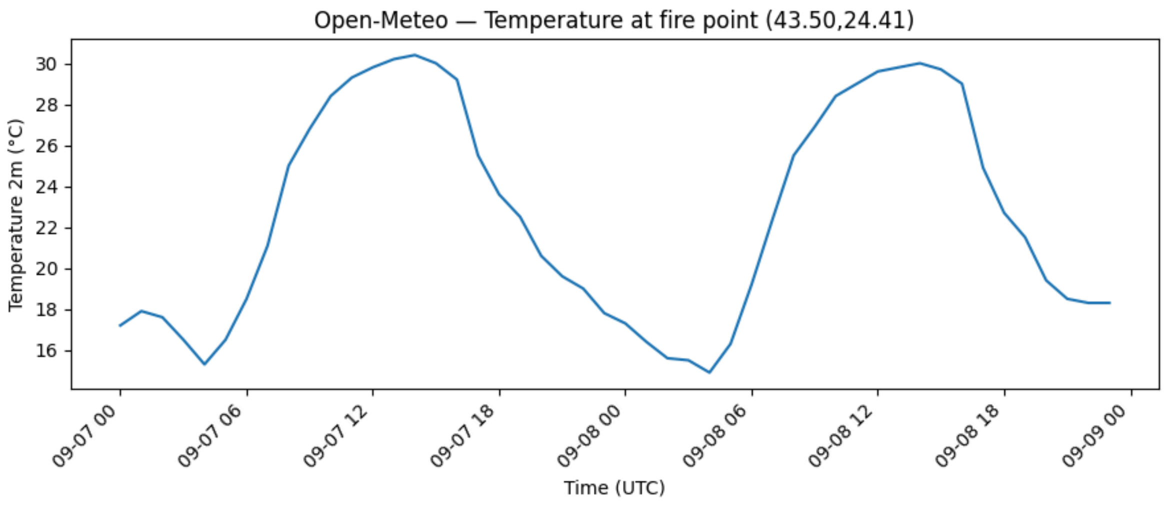
This optional step serves as a **prototype for future joins**: linking individual fire detections with their local weather context. It provides a first look at how meteorology and active fire events intersect in the pipeline.

In [26]:
```python
candidate_lat, candidate_lon = None, None

if "df_raw" in globals() and isinstance(df_raw, pd.DataFrame) and not df_
    cand = df_raw.rename(columns=str.lower)
    for a, b in (("latitude","longitude"), ("lat","lon"), ("y","x")):
        if a in cand.columns and b in cand.columns:
            candidate_lat = float(cand.iloc[0][a])
            candidate_lon = float(cand.iloc[0][b])
            break

elif "df_fires" in globals() and isinstance(df_fires, pd.DataFrame) and n
    candidate_lat = float(df_fires.iloc[0]["latitude"])
    candidate_lon = float(df_fires.iloc[0]["longitude"])

if candidate_lat is not None and candidate_lon is not None:
    df_weather_point = fetch_open_meteo(candidate_lat, candidate_lon, DAT

    # Save raw weather CSV
    raw_pt_path = RAW_DIR / f"openmeteo_at_fire_{DATE_FROM}_{DATE_TO}.csv
    save_df(df_weather_point, raw_pt_path)

    # Daily summary → processed
    daily_summary_fire = (
        df_weather_point.assign(day=df_weather_point["time_utc"].dt.floor
        .groupby("day")
        .agg(
            temp_mean=("temperature_2m", "mean"),
            rh_mean=("relative_humidity_2m", "mean"),
            wind_mean=("windspeed_10m", "mean"),
            precip_sum=("precipitation", "sum"),
        )
        .reset_index()
    )
    daily_pt_out = PROCESSED_DIR / f"openmeteo_at_fire_daily_{DATE_FROM}_
    save_df(daily_summary_fire, daily_pt_out)

    print(f"Weather fetched at fire point lat={candidate_lat:.4f}, lon={c
    preview(df_weather_point)

    # Plot: Temperature vs time
    plt.figure(figsize=(9,4))
    plt.plot(df_weather_point["time_utc"], df_weather_point["temperature_
    plt.title(f"Open-Meteo – Temperature at fire point ({candidate_lat:.2
    plt.xlabel("Time (UTC)")
    plt.ylabel("Temperature 2m (°C)")
    plt.xticks(rotation=45, ha="right")
    plt.tight_layout()
    plt.show()

    # Save figure
    fig_path = REPORTS_DIR / f"openmeteo_firepoint_temp_{DATE_FROM}_{DATE
    plt.figure(figsize=(9,4))
    plt.plot(df_weather_point["time_utc"], df_weather_point["temperature_
    plt.title(f"Open-Meteo – Temperature at fire point ({candidate_lat:.2
    plt.xlabel("Time (UTC)")
    plt.ylabel("Temperature 2m (°C)")
    plt.xticks(rotation=45, ha="right")
    plt.tight_layout()
    plt.show()
```

```
Saved: /Users/evareysanchez/WildfiresAI/data/raw/openmeteo_at_fire_2025-09
-07_2025-09-14.csv
Saved: /Users/evareysanchez/WildfiresAI/data/processed/openmeteo_at_fire_d
aily_2025-09-07_2025-09-14.csv
Weather fetched at fire point lat=43.4989, lon=24.4062
```

	time_utc	temperature_2m	relative_humidity_2m	dew_point_2m	windspee
0	2025-09-07 00:00:00+00:00	17.2	63.0	10.2	
1	2025-09-07 01:00:00+00:00	17.9	58.0	9.5	
2	2025-09-07 02:00:00+00:00	17.6	59.0	9.6	
3	2025-09-07 03:00:00+00:00	16.5	64.0	9.6	
4	2025-09-07 04:00:00+00:00	15.3	69.0	9.8	
5	2025-09-07 05:00:00+00:00	16.5	65.0	10.0	
6	2025-09-07 06:00:00+00:00	18.5	58.0	10.2	
7	2025-09-07 07:00:00+00:00	21.1	52.0	10.8	
8	2025-09-07 08:00:00+00:00	25.0	40.0	10.4	
9	2025-09-07 09:00:00+00:00	26.8	34.0	9.7	





Cell 13 — Build Clean FIRMS DataFrame (df_fires)

Purpose. Convert the raw FIRMS CSV (Cell 9) into a clean, analysis-ready table:

normalize schema and types

construct a UTC timestamp (acq_datetime)

filter by Spain bbox and DATE_FROM → DATE_TO (UTC)

drop duplicates and sort

persist the result to data/processed/ and expose it as df_fires

In [35]:

```
# --- Guards -----------------------------------------------------------------
import pandas as pd
import numpy as np

assert "SPAIN_BBOX" in globals(), "Missing SPAIN_BBOX (run Global Configu
assert "DATE_FROM" in globals() and "DATE_TO" in globals(), "Missing DATE
assert "PROCESSED_DIR" in globals(), "Missing PROCESSED_DIR (run Global C

# Prefer df_raw from Cell 9; otherwise try to read the saved raw CSV
if "df_raw" not in globals() or not isinstance(df_raw, pd.DataFrame) or d
    from pathlib import Path
    raw_fallback = (RAW_DIR / "firms_last7d_es_raw.csv")
    if raw_fallback.exists():
        df_raw = pd.read_csv(raw_fallback, dtype={"acq_time": "string"})
        print(f"Loaded raw fallback: {raw_fallback}")
    else:
        raise RuntimeError("No FIRMS raw dataframe found. Run Cell 9 firs

# --- Helpers ----------------------------------------------------------------
def _parse_acq_datetime(acq_date, acq_time) -> pd.Series:
    """Combine acq_date (YYYY-MM-DD) and acq_time (HHMM) into a UTC times
    t = pd.Series(acq_time, dtype="string").str.zfill(4)
    dt = pd.Series(acq_date, dtype="string") + " " + t.str[:2] + ":" + t.
    return pd.to_datetime(dt, utc=True, errors="coerce")

def _clean_firms(df_in: pd.DataFrame) -> pd.DataFrame:
    df = df_in.copy()
    df.columns = [c.lower() for c in df.columns]

    # flexible renames (handles common variants)
    rename_map = {
        "lat": "latitude", "long": "longitude", "lon": "longitude",
        "brightness": "brightness", "bright_ti4": "bright_ti4", "bright_t
        "acq_date": "acq_date", "acq_time": "acq_time",
        "confidence": "confidence", "daynight": "daynight",
        "satellite": "satellite", "instrument": "instrument",
        "scan": "scan", "track": "track", "version": "version", "frp": "f
    }
    df.rename(columns={k: v for k, v in rename_map.items() if k in df.col

    # required columns
    req = ["latitude", "longitude", "acq_date", "acq_time"]
    missing = [c for c in req if c not in df.columns]
    if missing:
        raise ValueError(f"Raw FIRMS is missing required columns: {missin

    # timestamp
    df["acq_datetime"] = _parse_acq_datetime(df["acq_date"], df["acq_time

    # numeric coercion
    for c in ["latitude","longitude","frp","bright_ti4","bright_ti5","bri
        if c in df.columns:
            df[c] = pd.to_numeric(df[c], errors="coerce")

    # confidence normalization (text → numeric if needed)
    if "confidence" in df.columns:
        if df["confidence"].dtype == "object" or str(df["confidence"].dty
            map_text = {"low": 20, "nominal": 50, "high": 85}
            df["confidence_text"] = df["confidence"].str.lower().map(lamb
            df["confidence_num"]  = df["confidence"].str.lower().map(map_
        else:
            df["confidence_num"] = pd.to_numeric(df["confidence"], errors
    else:
        df["confidence_num"] = np.nan

    return df

def _clip_bbox(df: pd.DataFrame, bbox) -> pd.DataFrame:
    w, s, e, n = bbox
    m = df["longitude"].between(w, e) & df["latitude"].between(s, n)
    return df.loc[m].copy()

def _clip_timerange(df: pd.DataFrame, start_utc: pd.Timestamp, end_utc_ex
    m = (df["acq_datetime"] >= start_utc) & (df["acq_datetime"] < end_utc
    return df.loc[m].copy()

# --- Clean + Filter -----------------------------------------------------
df_tmp = _clean_firms(df_raw)

# time window as UTC [start, end)
start_utc = pd.Timestamp(DATE_FROM).tz_localize("UTC")
end_utc_exclusive = pd.Timestamp(DATE_TO + pd.Timedelta(days=1)).tz_local

df_tmp = _clip_bbox(df_tmp, SPAIN_BBOX)
df_tmp = _clip_timerange(df_tmp, start_utc, end_utc_exclusive)

# de-duplicate & sort
df_fires = (
    df_tmp.drop_duplicates(subset=["acq_datetime", "latitude", "longitude
          .sort_values("acq_datetime")
          .reset_index(drop=True)
)

# --- Persist & Report ---------------------------------------------------
out_path = PROCESSED_DIR / f"firms_es_{DATE_FROM}_{DATE_TO}.csv"
out_path.parent.mkdir(parents=True, exist_ok=True)
df_fires.to_csv(out_path, index=False)

print(" FIRMS cleaned dataframe ready → df_fires")
print(f"Rows: {len(df_fires)}  |  Saved: {out_path}")
if len(df_fires):
    print(f"Time range (UTC): {df_fires['acq_datetime'].min()} → {df_fire
    print(f"BBox: {SPAIN_BBOX}")
```

```
# Preview
display(df_fires.head(min(10, len(df_fires))))
```

```
 FIRMS cleaned dataframe ready → df_fires
Rows: 475  |  Saved: /Users/evareysanchez/WildfiresAI/data/processed/firms
_es_2025-09-07_2025-09-14.csv
Time range (UTC): 2025-09-08 01:43:00+00:00 → 2025-09-14 14:33:00+00:00
BBox: (-9.5, 35.0, 3.5, 43.9)
```

	latitude	longitude	bright_ti4	scan	track	acq_date	acq_time	satellite	instru
0	35.14813	-6.10657	304.86	0.50	0.49	2025-09-08	143	N	
1	40.72105	-3.59960	295.89	0.43	0.46	2025-09-08	143	N	
2	38.80831	-5.00350	300.96	0.48	0.48	2025-09-08	143	N	
3	38.80440	-4.95728	300.73	0.47	0.48	2025-09-08	143	N	
4	36.96589	-1.90277	307.78	0.48	0.40	2025-09-08	143	N	
5	36.13130	1.23995	313.23	0.39	0.36	2025-09-08	143	N	
6	36.12734	1.24351	326.47	0.39	0.36	2025-09-08	143	N	
7	35.82257	-0.32437	315.38	0.42	0.37	2025-09-08	143	N	
8	35.81514	-0.26941	313.27	0.42	0.37	2025-09-08	143	N	
9	36.12593	1.24090	323.19	0.39	0.36	2025-09-08	143	N	

Cell 14 — Data Inventory (with last modified + CSV)

This cell performs a **systematic inventory of the project's data folders** to ensure full traceability of the pipeline. It inspects the contents of `data/raw/` and `data/processed/`, reports file sizes and modification timestamps, and consolidates the results into a structured DataFrame.

Main steps:

1. **Directory Resolution**
   - Supports multiple naming conventions (`RAW_DIR` / `DATA_RAW`, `PROCESSED_DIR` / `DATA_PROC`).
   - Ensures project directories are initialized before proceeding.
2. **Inventory Generation**
   - Iterates through all `.csv` files in the raw and processed folders.

- Extracts **file name, size (KB), and last modified time**.
- Appends results to a unified list for reporting.

3. **Persistence**

- Saves the inventory itself as a CSV file under `data/processed/` → `data_inventory.csv`.
- This creates a **meta-trace** of all generated artifacts, useful for reproducibility and audits.

4. **Visualization**

- Prints a human-readable summary to the console.
- Displays the first 20 rows of the inventory DataFrame inside the notebook.

This step ensures that the pipeline maintains a **transparent record of all inputs and outputs** generated so far, a requirement for scientific reproducibility and future integration with automated QA systems.

In [36]:
```python
import pandas as pd
from pathlib import Path

# 1) Ensure raw FIRMS dataframe is available; if not, reload from the sav
raw_path = RAW_DIR / "firms_last7d_es_raw.csv"
if "df_raw" not in globals() or df_raw is None or df_raw.empty:
    if raw_path.exists():
        df_raw = pd.read_csv(raw_path, dtype={"acq_time": "string"})
        print(f"Loaded raw FIRMS from {raw_path}")
    else:
        raise RuntimeError("df_raw is missing and raw CSV not found. Run

# 2) Normalize column names to lowercase
df = df_raw.rename(columns=str.lower).copy()

# 3) Basic schema validation: ensure required fields are present
required = {"latitude", "longitude", "acq_date", "acq_time"}
missing = [c for c in required if c not in df.columns]
if missing:
    raise RuntimeError(f"Raw FIRMS missing columns: {missing}")

# 4) Build UTC acquisition datetime from acq_date + acq_time
t = pd.Series(df["acq_time"], dtype="string").str.zfill(4)
dt = pd.Series(df["acq_date"], dtype="string") + " " + t.str[:2] + ":" +
df["acq_datetime"] = pd.to_datetime(dt, utc=True, errors="coerce")

# 5) Convert key numeric columns to proper dtypes
for c in ("latitude", "longitude", "frp", "bright_ti4", "bright_ti5", "sc
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors="coerce")

# 6) Apply spatial (Spain bounding box) and temporal filters
W, S, E, N = SPAIN_BBOX
m_geo = df["longitude"].between(W, E) & df["latitude"].between(S, N)
m_time = (df["acq_datetime"] >= pd.to_datetime(DATE_FROM).tz_localize("UT
         (df["acq_datetime"] <= pd.to_datetime(DATE_TO).tz_localize("UTC"

df_fires = df.loc[m_geo & m_time].copy()
```

```
# 7) Drop duplicates and sort chronologically
df_fires = (
    df_fires.drop_duplicates(subset=["acq_datetime", "latitude", "longitu
            .sort_values("acq_datetime")
            .reset_index(drop=True)
)

print(f" df_fires built. Rows: {len(df_fires)}")

# 8) Save cleaned dataset to processed folder
out_file = PROCESSED_DIR / f"fires_firms_es_{DATE_FROM}_{DATE_TO}.csv"
df_fires.to_csv(out_file, index=False)
print("Saved:", out_file)

# 9) Quick preview of the cleaned dataset
display(df_fires.head(10))
```

```
 df_fires built. Rows: 364
Saved: /Users/evareysanchez/WildfiresAI/data/processed/fires_firms_es_2025
-09-07_2025-09-14.csv
```

	latitude	longitude	bright_ti4	scan	track	acq_date	acq_time	satellite	instru
0	35.14813	-6.10657	304.86	0.50	0.49	2025-09-08	143	N	
1	40.72105	-3.59960	295.89	0.43	0.46	2025-09-08	143	N	
2	38.80831	-5.00350	300.96	0.48	0.48	2025-09-08	143	N	
3	38.80440	-4.95728	300.73	0.47	0.48	2025-09-08	143	N	
4	36.96589	-1.90277	307.78	0.48	0.40	2025-09-08	143	N	
5	36.13130	1.23995	313.23	0.39	0.36	2025-09-08	143	N	
6	36.12734	1.24351	326.47	0.39	0.36	2025-09-08	143	N	
7	35.82257	-0.32437	315.38	0.42	0.37	2025-09-08	143	N	
8	35.81514	-0.26941	313.27	0.42	0.37	2025-09-08	143	N	
9	36.12593	1.24090	323.19	0.39	0.36	2025-09-08	143	N	

Cell 15 — Fetch Open-Meteo for Fire Points (robust, cached)

This cell enriches **FIRMS fire detections with localized weather conditions** by systematically querying the Open-Meteo ERA5 archive for each unique fire point. Compared to a naive batch approach, this implementation is designed for **resilience and reproducibility**.

Main steps:

1. **Preconditions**
    - Verifies that `df_fires` exists and is non-empty.
    - Ensures the pipeline only queries weather data when fire detections are available.
2. **Unique Fire Locations**
    - Extracts latitude/longitude pairs from FIRMS detections.
    - Rounds coordinates to 0.01° (~1.1 km) to avoid redundant requests while preserving spatial fidelity.
    - Drops duplicates, keeping only distinct fire points.
3. **Sampling Strategy & Limits**
    - Caps the number of processed points (`MAX_POINTS = 15` by default).
    - Prevents **API throttling** and keeps runtime manageable.
    - Prints the total vs. processed points for transparency.
4. **Resilient Weather Retrieval**
    - Implements **per-point caching** (`data/raw/openmeteo_cache/`), so repeated runs automatically reuse existing CSVs.
    - Uses `requests.Session()` with **retry logic (exponential backoff)** to handle transient network errors.
    - Applies a **delay between calls** to respect API rate limits.
    - Normalizes outputs (lat/lon rounding, hourly alignment) for later joins with fire data.
5. **Aggregation & Persistence**
    - Concatenates all weather frames into a single DataFrame.
    - Saves combined data under `data/raw/` with a date-stamped filename.
    - Provides console feedback and previews the first rows for verification.

In [37]:
```
from time import sleep
from pathlib import Path
from io import StringIO
import json
import pandas as pd
import requests

# --- Preconditions
if "df_fires" not in globals() or df_fires is None or df_fires.empty:
    raise RuntimeError("df_fires is empty. Build it first (Cells 9→12).")

# --- Unique fire points (rounded to reduce duplicates)
pts = (
    df_fires[["latitude", "longitude"]]
    .dropna()
    .round(2)                     # 0.01° ~ 1.1 km
    .drop_duplicates()
    .reset_index(drop=True)
)
```

```
num_pts = len(pts)

# --- Safety cap and rate limiting
MAX_POINTS = 15                  # reduce a 15 para evitar timeouts
DELAY_SEC  = 0.8                 # pausa entre peticiones (ajustable)
RETRIES    = 3                   # reintentos por punto
TIMEOUT    = 60                  # timeout por petición (s)

if num_pts > MAX_POINTS:
    print(f"Found {num_pts} unique points; sampling first {MAX_POINTS} to
    pts = pts.iloc[:MAX_POINTS].copy()
    num_pts = MAX_POINTS

# --- Cache dir per-point (resumable)
CACHE_DIR = RAW_DIR / "openmeteo_cache"
CACHE_DIR.mkdir(parents=True, exist_ok=True)

session = requests.Session()

def fetch_point_with_retries(lat_i: float, lon_i: float) -> pd.DataFrame:
    """Call fetch_open_meteo with retries/backoff and return a normalized
    backoff = 1.5
    err_last = None
    for attempt in range(1, RETRIES + 1):
        try:
            df_w = fetch_open_meteo(lat_i, lon_i, DATE_FROM, DATE_TO, mod
            # Normalizar columnas y claves de join
            df_w = df_w.rename(columns={"lat": "latitude", "lon": "longit
            df_w["lat_round"] = df_w["latitude"].round(1)
            df_w["lon_round"] = df_w["longitude"].round(1)
            df_w["time_hour"] = pd.to_datetime(df_w["time_utc"]).dt.floor
            return df_w
        except Exception as e:
            err_last = e
            if attempt < RETRIES:
                sleep(backoff)
                backoff *= 2
            else:
                raise err_last

weather_frames = []

for i, row in pts.iterrows():
    lat_i = float(row["latitude"])
    lon_i = float(row["longitude"])

    # archivo de caché por punto (usa coordenadas redondeadas para nombre
    key = f"{lat_i:.2f}_{lon_i:.2f}_{DATE_FROM}_{DATE_TO}.csv".replace("
    cache_file = CACHE_DIR / key

    print(f"[{i+1}/{num_pts}] Weather lat={lat_i:.2f}, lon={lon_i:.2f} …"
    if cache_file.exists():
        # usar caché
        df_w = pd.read_csv(cache_file, parse_dates=["time_utc", "time_hou
        print("(cached)")
    else:
        # llamar con reintentos y guardar caché
        df_w = fetch_point_with_retries(lat_i, lon_i)
        df_w.to_csv(cache_file, index=False)
        print("ok")
        sleep(DELAY_SEC)  # rate limiting

    weather_frames.append(df_w)

# --- Combine & save batch
if not weather_frames:
    raise RuntimeError("No weather frames were fetched. Check earlier ste

weather_points = pd.concat(weather_frames, ignore_index=True)
raw_weather_points_path = RAW_DIR / f"openmeteo_firepoints_{DATE_FROM}_{D
save_df(weather_points, raw_weather_points_path)

print(f" Weather fetched for {num_pts} fire point(s). Rows total: {len(we
display(weather_points.head(10))
```

```
Found 166 unique points; sampling first 15 to limit API calls.
[1/15] Weather lat=35.15, lon=-6.11 … (cached)
[2/15] Weather lat=40.72, lon=-3.60 … (cached)
[3/15] Weather lat=38.81, lon=-5.00 … (cached)
[4/15] Weather lat=38.80, lon=-4.96 … (cached)
[5/15] Weather lat=36.97, lon=-1.90 … (cached)
[6/15] Weather lat=36.13, lon=1.24 … (cached)
[7/15] Weather lat=35.82, lon=-0.32 … (cached)
[8/15] Weather lat=35.82, lon=-0.27 … (cached)
[9/15] Weather lat=35.81, lon=-0.26 … (cached)
[10/15] Weather lat=35.80, lon=-0.24 … (cached)
[11/15] Weather lat=35.78, lon=0.54 … (cached)
[12/15] Weather lat=35.53, lon=-0.28 … (cached)
[13/15] Weather lat=35.46, lon=-0.40 … (cached)
[14/15] Weather lat=35.34, lon=1.36 … (cached)
[15/15] Weather lat=37.10, lon=-7.96 … (cached)
Saved: /Users/evareysanchez/WildfiresAI/data/raw/openmeteo_firepoints_2025
-09-07_2025-09-14.csv
 Weather fetched for 15 fire point(s). Rows total: 2880
```

	time_utc	temperature_2m	relative_humidity_2m	dew_point_2m	windspee
0	2025-09-07 00:00:00+00:00	23.8	87.0	21.4	
1	2025-09-07 01:00:00+00:00	23.3	90.0	21.5	
2	2025-09-07 02:00:00+00:00	23.1	90.0	21.4	
3	2025-09-07 03:00:00+00:00	22.6	93.0	21.4	
4	2025-09-07 04:00:00+00:00	22.1	96.0	21.4	
5	2025-09-07 05:00:00+00:00	22.6	94.0	21.5	
6	2025-09-07 06:00:00+00:00	22.7	90.0	21.1	
7	2025-09-07 07:00:00+00:00	23.5	87.0	21.2	
8	2025-09-07 08:00:00+00:00	24.5	86.0	21.9	
9	2025-09-07 09:00:00+00:00	25.8	78.0	21.7	

Cell 16 — Weather Join Validator (debug only)

This intermediate cell provides a **sanity check** before performing the full spatiotemporal join between FIRMS and Open-Meteo (Cell 16).

Purpose:

- Ensure that both datasets (`fires` and `weather_points`) contain the necessary harmonized keys for merging.
- Validate that the temporal (`datetime_hour` / `time_hour`) and spatial (`lat_round` , `lon_round`) alignment columns are present.
- Print row counts, preview available keys, and highlight potential gaps in coverage.

Why important:
Skipping this step can result in silent join failures (empty merges or missing columns), which would propagate errors downstream.
By explicitly checking schemas and alignment, this validator cell reduces debugging time and improves pipeline robustness.

Note:

- This is a **diagnostic-only step**: it does not persist outputs.
- Safe to skip in production once the pipeline has stabilized.

In [42]:
```
# --- Guards: ensure required dataframes exist ---------------------------
if "df_fires" not in globals() and "df_raw" not in globals():
    raise RuntimeError("No FIRMS dataframe found (df_fires or df_raw miss
if "weather_points" not in globals() or weather_points is None or weather
    raise RuntimeError("weather_points is empty. Re-run Cell 15 (batched

# Choose the fire dataframe: prefer df_fires if available
fires_src = None
if "df_fires" in globals() and isinstance(df_fires, pd.DataFrame) and not
    fires_src = df_fires.copy()
elif "df_raw" in globals() and isinstance(df_raw, pd.DataFrame) and not d
    fires_src = df_raw.copy()
else:
    raise RuntimeError("Both df_fires and df_raw are empty — cannot conti

# Normalize schemas for comparison
fires = fires_src.rename(columns=str.lower).copy()
wx    = weather_points.rename(columns=str.lower).copy()

# --- Check temporal alignment columns -----------------------------------
print("=== Temporal alignment check ===")
print("FIRMS datetime columns:", [c for c in fires.columns if "date" in c
print("Weather datetime columns:", [c for c in wx.columns if "time" in c]

# --- Check spatial alignment columns ------------------------------------
print("\n=== Spatial alignment check ===")
print("FIRMS lat/lon columns:", [c for c in fires.columns if "lat" in c o
print("Weather lat/lon columns:", [c for c in wx.columns if "lat" in c or

# --- Row counts ---------------------------------------------------------
print("\n=== Row counts ===")
print("FIRMS rows:", len(fires))
print("Weather rows:", len(wx))

# --- Quick preview of keys ----------------------------------------------
print("\n=== Preview of join keys ===")
if "acq_datetime" in fires.columns:
    fires["datetime_hour"] = pd.to_datetime(fires["acq_datetime"], utc=Tr
elif {"acq_date", "acq_time"}.issubset(fires.columns):
    t  = pd.Series(fires["acq_time"], dtype="string").str.zfill(4)
    dt = pd.Series(fires["acq_date"], dtype="string") + " " + t.str[:2] +
    fires["datetime_hour"] = pd.to_datetime(dt, utc=True, errors="coerce"

fires["lat_round"] = pd.to_numeric(fires.get("latitude", pd.Series()), er
fires["lon_round"] = pd.to_numeric(fires.get("longitude", pd.Series()), e

if "time_hour" not in wx.columns and "time_utc" in wx.columns:
    wx["time_hour"] = pd.to_datetime(wx["time_utc"]).dt.floor("h")
if "lat_round" not in wx.columns and "latitude" in wx.columns:
    wx["lat_round"] = pd.to_numeric(wx["latitude"], errors="coerce").roun
if "lon_round" not in wx.columns and "longitude" in wx.columns:
    wx["lon_round"] = pd.to_numeric(wx["longitude"], errors="coerce").rou

display(fires[["datetime_hour","lat_round","lon_round"]].head(5))
display(wx[["time_hour","lat_round","lon_round"]].head(5))

print("\n Validation complete — proceed to Cell 17 for the actual join.")
```

```
=== Temporal alignment check ===
FIRMS datetime columns: ['acq_date', 'acq_time', 'acq_datetime']
Weather datetime columns: ['time_utc', 'time_hour']

=== Spatial alignment check ===
FIRMS lat/lon columns: ['latitude', 'longitude']
Weather lat/lon columns: ['relative_humidity_2m', 'latitude', 'longitude',
'lat_round', 'lon_round']

=== Row counts ===
FIRMS rows: 364
Weather rows: 2880

=== Preview of join keys ===
```

	datetime_hour	lat_round	lon_round
0	2025-09-08 01:00:00+00:00	35.1	-6.1
1	2025-09-08 01:00:00+00:00	40.7	-3.6
2	2025-09-08 01:00:00+00:00	38.8	-5.0
3	2025-09-08 01:00:00+00:00	38.8	-5.0
4	2025-09-08 01:00:00+00:00	37.0	-1.9

	time_hour	lat_round	lon_round
0	2025-09-07 00:00:00+00:00	35.2	-6.1
1	2025-09-07 01:00:00+00:00	35.2	-6.1
2	2025-09-07 02:00:00+00:00	35.2	-6.1
3	2025-09-07 03:00:00+00:00	35.2	-6.1
4	2025-09-07 04:00:00+00:00	35.2	-6.1

```
 Validation complete — proceed to Cell 17 for the actual join.
```

Cell 17 — Join FIRMS and Open-Meteo Data

This cell performs the first **spatiotemporal integration** between satellite fire detections (NASA FIRMS) and local weather conditions (Open-Meteo).

Operations

1. **Schema harmonization**
   - Convert FIRMS timestamps to UTC and normalize to **hourly** resolution (`datetime_hour`).
   - Apply **soft spatial rounding** (0.1°) on lat/lon in both datasets to align with the meteorological grid.
2. **Join**
   - Merge on the composite keys: **(`datetime_hour`, `lat_round`, `lon_round`)**.
   - Produce a joined table with fire attributes and co-located hourly weather.

3. **Persistence**

    - Save the integrated dataset to `data/processed/` with a date-scoped filename.

4. **Diagnostics & Visualization**

    - Generate **Daily Fire Detections** figure from FIRMS.
    - Compute and plot **Daily Mean Temperature** at joined fire locations (if coverage is sufficient).
    - Store figures under `reports/` for traceability.

Notes

- This step assumes weather retrieval for fire points has been completed (see **Cell 14**).
- Join quality depends on temporal alignment (hour rounding) and spatial tolerance (0.1°). These parameters can be tuned in subsequent iterations to maximize coverage without over-matching.

In [43]:
```
# 0) Guards & inputs
REPORTS_DIR_EFF = globals().get("REPORTS_DIR")
DATA_PROC_EFF   = globals().get("PROCESSED_DIR", globals().get("DATA_PROC
assert REPORTS_DIR_EFF is not None and DATA_PROC_EFF is not None, "Run th
assert "DATE_FROM" in globals() and "DATE_TO" in globals(), "Pipeline par

if "df_fires" in globals() and isinstance(df_fires, pd.DataFrame) and not
    fires_src = df_fires.copy()
elif "df_raw" in globals() and isinstance(df_raw, pd.DataFrame) and not d
    fires_src = df_raw.copy()
else:
    raise RuntimeError("No FIRMS dataframe found (df_fires/df_raw). Run C

if "weather_points" not in globals() or weather_points is None or weather
    raise RuntimeError("weather_points is empty. Re-run Cell 14 (batched

fires = fires_src.rename(columns=str.lower).copy()
wx    = weather_points.rename(columns=str.lower).copy()

# 1) Harmonize schema ---------------------------------------------------
# FIRMS timestamp → UTC and hour
if "acq_datetime" in fires.columns:
    fires["acq_datetime"] = pd.to_datetime(fires["acq_datetime"], utc=Tru
elif {"acq_date", "acq_time"}.issubset(fires.columns):
    t  = pd.Series(fires["acq_time"], dtype="string").str.zfill(4)
    dt = pd.Series(fires["acq_date"], dtype="string") + " " + t.str[:2] +
    fires["acq_datetime"] = pd.to_datetime(dt, utc=True, errors="coerce")
else:
    raise RuntimeError("FIRMS dataframe lacks acq_datetime or (acq_date,

fires["datetime_hour"] = fires["acq_datetime"].dt.floor("h")

# FIRMS rounded coords
if "latitude" not in fires.columns or "longitude" not in fires.columns:
    raise RuntimeError("FIRMS dataframe lacks latitude/longitude columns.
fires["lat_round"] = pd.to_numeric(fires["latitude"],  errors="coerce").r
fires["lon_round"] = pd.to_numeric(fires["longitude"], errors="coerce").r
```

```
# WEATHER: ensure datetime_hour + rounded coords exist
if "datetime_hour" not in wx.columns:
    if "time_hour" in wx.columns:
        wx["datetime_hour"] = pd.to_datetime(wx["time_hour"], utc=True, e
    elif "time_utc" in wx.columns:
        wx["datetime_hour"] = pd.to_datetime(wx["time_utc"], utc=True, er
    else:
        raise RuntimeError("weather_points lacks time_hour/time_utc to co

if "lat_round" not in wx.columns or "lon_round" not in wx.columns:
    wx["lat_round"] = pd.to_numeric(wx["latitude"],  errors="coerce").rou
    wx["lon_round"] = pd.to_numeric(wx["longitude"], errors="coerce").rou

# Sanity: keys must exist and be datetime/float
wx["datetime_hour"]    = pd.to_datetime(wx["datetime_hour"], utc=True, err
fires["datetime_hour"] = pd.to_datetime(fires["datetime_hour"], utc=True,

# 2) Join (hour + ~0.1° grid) ---------------------------------------------
join_keys = ["datetime_hour", "lat_round", "lon_round"]
df_join = pd.merge(
    fires, wx,
    how="left",
    left_on=join_keys,
    right_on=join_keys,
    suffixes=("", "_wx")
)

print("Joined rows:", len(df_join))
display(df_join.head(10))

# 3) Save joined CSV ------------------------------------------------------
joined_out = DATA_PROC_EFF / f"fires_weather_es_{DATE_FROM}_{DATE_TO}.csv
joined_out.parent.mkdir(parents=True, exist_ok=True)
df_join.to_csv(joined_out, index=False)
print(f"Saved joined CSV → {joined_out}")

# 4) Figure: daily FIRMS detections ---------------------------------------
daily_det = (
    df_join.assign(day=df_join["acq_datetime"].dt.floor("D"))
           .groupby("day")["acq_datetime"].count()
           .reset_index(name="detections")
)

plt.figure(figsize=(8, 4.2))
plt.plot(daily_det["day"], daily_det["detections"])
plt.title("Daily Fire Detections (FIRMS)")
plt.xlabel("Date (UTC)")
plt.ylabel("Detections")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
fig1 = REPORTS_DIR_EFF / f"joined_daily_detections_{DATE_FROM}_{DATE_TO}.
plt.savefig(fig1, dpi=160)
plt.show()
print("Saved:", fig1)

# 5) Figure: daily mean temperature (if present) --------------------------
temp_candidates = [c for c in ["temperature_2m", "temperature_2m_wx"] if
temp_col = temp_candidates[0] if temp_candidates else None
```

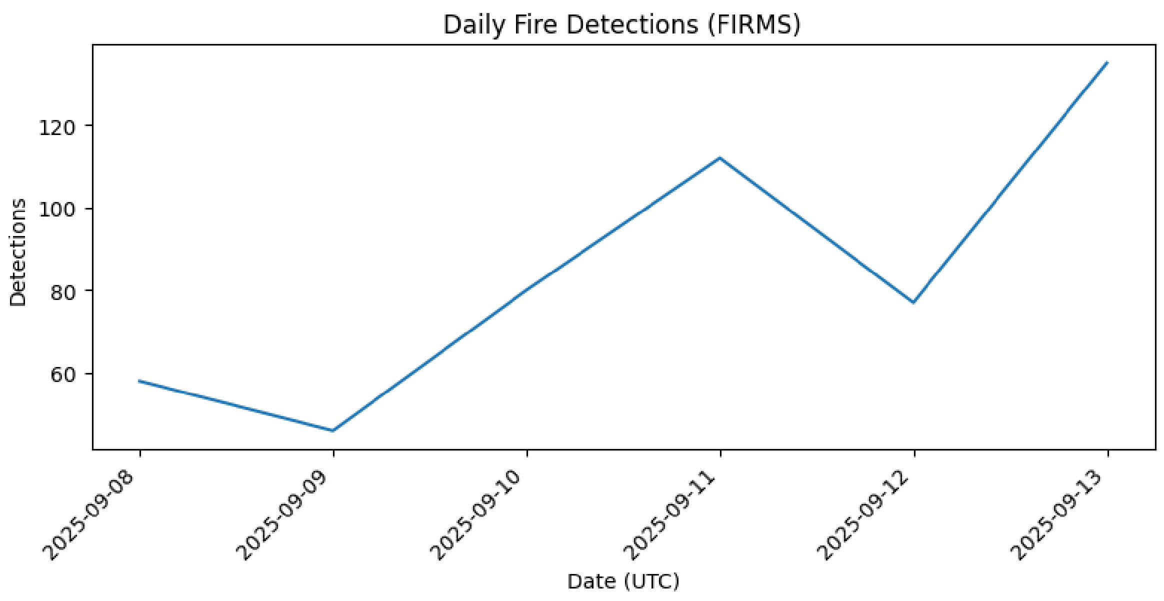
```
if temp_col and pd.to_numeric(df_join[temp_col], errors="coerce").notna()
    daily_temp = (
        df_join.assign(day=df_join["datetime_hour"].dt.floor("D"))
               .groupby("day")[temp_col].mean()
               .reset_index(name="temp_mean")
    )
    if len(daily_temp):
        plt.figure(figsize=(8, 4.2))
        plt.plot(daily_temp["day"], daily_temp["temp_mean"])
        plt.title("Daily Mean Temperature (Joined Weather)")
        plt.xlabel("Date (UTC)")
        plt.ylabel("Temp 2m (°C)")
        plt.xticks(rotation=45, ha="right")
        plt.tight_layout()
        fig2 = REPORTS_DIR_EFF / f"joined_daily_temp_{DATE_FROM}_{DATE_TO
        plt.savefig(fig2, dpi=160)
        plt.show()
        print("Saved:", fig2)
    else:
        print("No daily temperature to plot (empty after grouping).")
else:
    print("No temperature_2m available in joined dataframe — check join c
```

```
Joined rows: 508
```

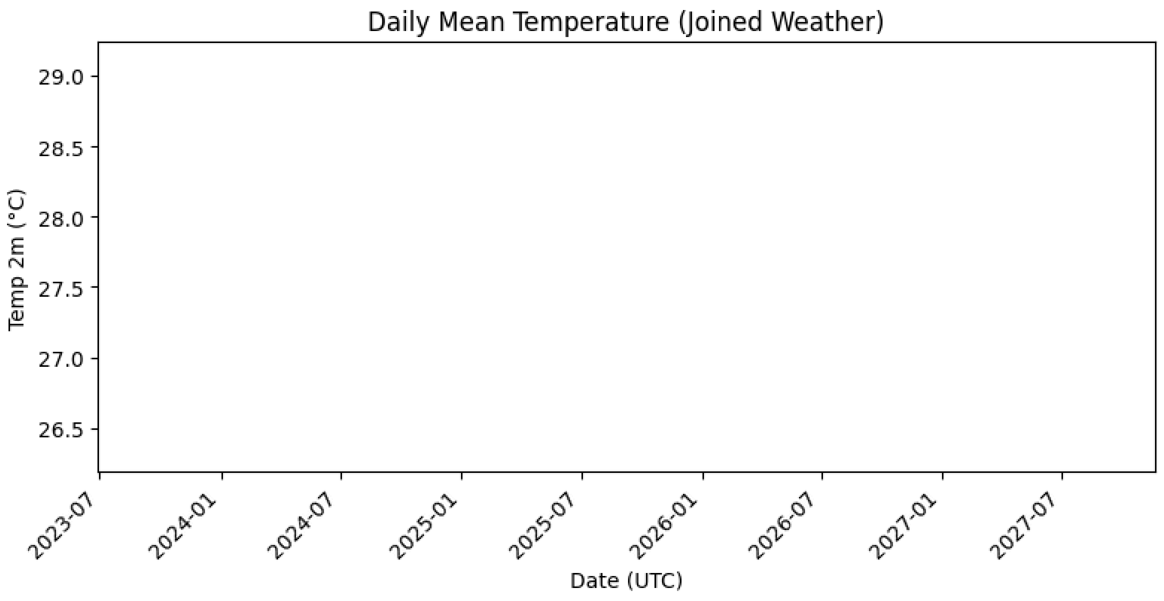
	latitude	longitude	bright_ti4	scan	track	acq_date	acq_time	satellite	instru
0	35.14813	-6.10657	304.86	0.50	0.49	2025-09-08	143	N	
1	40.72105	-3.59960	295.89	0.43	0.46	2025-09-08	143	N	
2	38.80831	-5.00350	300.96	0.48	0.48	2025-09-08	143	N	
3	38.80831	-5.00350	300.96	0.48	0.48	2025-09-08	143	N	
4	38.80440	-4.95728	300.73	0.47	0.48	2025-09-08	143	N	
5	38.80440	-4.95728	300.73	0.47	0.48	2025-09-08	143	N	
6	36.96589	-1.90277	307.78	0.48	0.40	2025-09-08	143	N	
7	36.13130	1.23995	313.23	0.39	0.36	2025-09-08	143	N	
8	36.12734	1.24351	326.47	0.39	0.36	2025-09-08	143	N	
9	35.82257	-0.32437	315.38	0.42	0.37	2025-09-08	143	N	

10 rows × 30 columns

```
Saved joined CSV → /Users/evareysanchez/WildfiresAI/data/processed/fires_w
eather_es_2025-09-07_2025-09-14.csv
```



```
Saved: /Users/evareysanchez/WildfiresAI/reports/joined_daily_detections_20
25-09-07_2025-09-14.png
```



```
Saved: /Users/evareysanchez/WildfiresAI/reports/joined_daily_temp_2025-09-
07_2025-09-14.png
```

In [ ]:

In [ ]:

Cell — Common Utilities

In [27]:

```python
import pandas as pd

def save_df(df: pd.DataFrame, path: Path) -> None:
    """Save a DataFrame to CSV and print the path."""
    path.parent.mkdir(parents=True, exist_ok=True)
    df.to_csv(path, index=False)
    print(f"Saved: {path}")

def preview(df: pd.DataFrame, n: int = MAX_ROWS_PREVIEW) -> None:
    """Quick preview of a DataFrame."""
    display(df.head(n))
```

Cell — FIRMS Cleaning (helpers + clean_firms)

In [48]:
```
def _parse_acq_datetime(acq_date, acq_time):
    """Combine acq_date (YYYY-MM-DD) and acq_time (HHMM) into a UTC datet
    t = pd.Series(acq_time, dtype="string").str.zfill(4)
    dt = pd.Series(acq_date, dtype="string") + " " + t.str[:2] + ":" + t.
    return pd.to_datetime(dt, utc=True, errors="coerce")

def _clip_bbox(df, bbox):
    """Filter rows inside a bounding box (W,S,E,N)."""
    w, s, e, n = bbox
    m = df["longitude"].between(w, e) & df["latitude"].between(s, n)
    return df.loc[m].copy()

def _clip_timerange(df, start_utc, end_utc):
    """Filter rows within a time window [start_utc, end_utc]."""
    return df.loc[(df["acq_datetime"] >= start_utc) & (df["acq_datetime"]

def clean_firms(df_raw: pd.DataFrame) -> pd.DataFrame:
    """Clean and standardize a raw FIRMS dataframe."""
    df = df_raw.copy()
    df.rename(columns={c: c.lower() for c in df.columns}, inplace=True)

    rename_map = {
        "lat":"latitude","lon":"longitude","long":"longitude",
        "acq_date":"acq_date","acq_time":"acq_time","frp":"frp",
        "confidence":"confidence","daynight":"daynight",
        "bright_ti4":"bright_ti4","bright_ti5":"bright_ti5",
        "brightness":"brightness","satellite":"satellite","instrument":"i
        "scan":"scan","track":"track","version":"version"
    }
    df.rename(columns={k:v for k,v in rename_map.items() if k in df.colum

    # Required
    required = ["latitude","longitude","acq_date","acq_time"]
    miss = [c for c in required if c not in df.columns]
    if miss:
        raise ValueError(f"Missing required FIRMS columns: {miss}")

    # Datetime
    df["acq_datetime"] = _parse_acq_datetime(df["acq_date"], df["acq_time

    # Numerics
    for c in ["latitude","longitude","frp","bright_ti4","bright_ti5","bri
        if c in df.columns:
            df[c] = pd.to_numeric(df[c], errors="coerce")

    # Confidence normalization
    if "confidence" in df.columns:
        if df["confidence"].dtype == "object" or str(df["confidence"].dty
            map_text = {"low":20, "nominal":50, "high":85}
            df["confidence_text"] = df["confidence"].str.lower().map(lamb
            df["confidence_num"]  = df["confidence"].str.lower().map(map_
        else:
            df["confidence_num"]  = pd.to_numeric(df["confidence"], error
    else:
        df["confidence_num"] = np.nan

    # Dedup + sort
    df = (df
          .drop_duplicates(subset=["acq_datetime","latitude","longitude"]
          .sort_values("acq_datetime")
          .reset_index(drop=True))
    return df
```

Cell — Build Processed Dataset (clean + filter + save)

In [29]:
```
from datetime import datetime, timezone, timedelta

# Source: prefer df_raw from Cell 9; otherwise read from file
raw_path = RAW_DIR / "firms_last7d_es_raw.csv"
if "df_raw" in globals() and isinstance(df_raw, pd.DataFrame) and not df_
    src = df_raw
    print("Using in-memory df_raw from Cell 9.")
elif raw_path.exists():
    print(f"Reading raw CSV: {raw_path}")
    src = pd.read_csv(raw_path, dtype={"acq_time":"string"})
else:
    raise FileNotFoundError(
        "No raw FIRMS data found. Run Cell 9 to download or place a CSV a
    )

# Clean
fires = clean_firms(src)

# Time window (UTC) derived from config dates
start_utc = pd.to_datetime(DATE_FROM).tz_localize("UTC")
end_utc   = pd.to_datetime(DATE_TO) + pd.Timedelta(days=1) - pd.Timedelta
end_utc   = end_utc.tz_localize("UTC", nonexistent="shift_forward", ambig

# Filter bbox + time
fires = _clip_bbox(fires, SPAIN_BBOX)
fires = _clip_timerange(fires, start_utc, end_utc)

print(f"Rows after clean+filters: {len(fires)}")

# Save processed
out_file = PROCESSED_DIR / f"fires_firms_es_{DATE_FROM}_{DATE_TO}.csv"
save_df(fires, out_file)

preview(fires)
df_fires = fires  # make it available for QA
```

```
Using in-memory df_raw from Cell 9.
Rows after clean+filters: 475
Saved: /Users/evareysanchez/WildfiresAI/data/processed/fires_firms_es_2025
-09-07_2025-09-14.csv
```

	latitude	longitude	bright_ti4	scan	track	acq_date	acq_time	satellite	i
2560	36.13130	1.23995	313.23	0.39	0.36	2025-09-08	143	N	
2564	36.12734	1.24351	326.47	0.39	0.36	2025-09-08	143	N	
2566	36.96589	-1.90277	307.78	0.48	0.40	2025-09-08	143	N	
2574	38.80440	-4.95728	300.73	0.47	0.48	2025-09-08	143	N	
2575	38.80831	-5.00350	300.96	0.48	0.48	2025-09-08	143	N	
2576	40.72105	-3.59960	295.89	0.43	0.46	2025-09-08	143	N	
2577	36.12593	1.24090	323.19	0.39	0.36	2025-09-08	143	N	
2579	35.82257	-0.32437	315.38	0.42	0.37	2025-09-08	143	N	
2580	35.46099	-0.39761	307.03	0.42	0.37	2025-09-08	143	N	
2581	35.80984	-0.25633	325.10	0.41	0.37	2025-09-08	143	N	

Cell — QA & Figures

In [30]:
```
from datetime import datetime, timezone

def _now_stamp():
    return datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%SZ")

def validate_firms_schema(df):
    issues = {"errors": [], "warnings": []}

    # Required columns
    required = ["acq_datetime", "latitude", "longitude"]
    missing = [c for c in required if c not in df.columns]
    if missing:
        issues["errors"].append(f"Missing required columns: {missing}")
        return issues

    # Types
    if not str(df["acq_datetime"].dtype).startswith("datetime64"):
        issues["errors"].append("acq_datetime is not datetime64[ns, UTC].
    if df["latitude"].dtype.kind not in "fi":
        issues["errors"].append("latitude is not numeric.")
    if df["longitude"].dtype.kind not in "fi":
        issues["errors"].append("longitude is not numeric.")

    # Nulls
    for c in ["acq_datetime", "latitude", "longitude"]:
        null_rate = df[c].isna().mean()
        if null_rate > 0:
            issues["errors"].append(f"Nulls in critical column {c}: {null

    # Geographic ranges
    if ((df["latitude"] < -90) | (df["latitude"] > 90)).any():
        issues["errors"].append("Latitude out of bounds [-90, 90].")
    if ((df["longitude"] < -180) | (df["longitude"] > 180)).any():
        issues["errors"].append("Longitude out of bounds [-180, 180].")

    # Temporal coverage
    try:
        min_t, max_t = df["acq_datetime"].min(), df["acq_datetime"].max()
        if pd.isna(min_t) or pd.isna(max_t):
            issues["warnings"].append("Temporal coverage unknown (NaT fou
        else:
            days_span = (max_t - min_t).days
            if days_span < 0:
                issues["errors"].append("acq_datetime not ordered or inva
            if days_span > 90:
                issues["warnings"].append(f"Large temporal span: {days_sp
    except Exception as e:
        issues["warnings"].append(f"Temporal coverage check failed: {e}")

    if "confidence_num" not in df.columns:
        issues["warnings"].append("confidence_num missing (ok for some pr

    return issues

def summarize_firms(df):
    out = {}
    out["n_rows"] = int(len(df))
    out["time_min"] = df["acq_datetime"].min()
    out["time_max"] = df["acq_datetime"].max()

    dup_keys = ["acq_datetime", "latitude", "longitude"]
    out["n_potential_duplicates"] = int(df.duplicated(subset=dup_keys, ke

    if "frp" in df.columns:
        frp = pd.to_numeric(df["frp"], errors="coerce")
        out["frp_count"] = int(frp.notna().sum())
        if frp.notna().any():
            out["frp_mean"] = float(frp.mean())
            out["frp_median"] = float(frp.median())
            out["frp_p90"] = float(frp.quantile(0.90))
            out["frp_p99"] = float(frp.quantile(0.99))
            out["frp_max"] = float(frp.max())
        else:
            out["frp_mean"] = out["frp_median"] = out["frp_p90"] = out["f
    else:
        out["frp_count"] = 0

    if "confidence_num" in df.columns:
        c = pd.to_numeric(df["confidence_num"], errors="coerce")
        out["conf_count"] = int(c.notna().sum())
        if c.notna().any():
            out["conf_mean"] = float(c.mean())
            out["conf_median"] = float(c.median())
            out["conf_min"] = float(c.min())
            out["conf_max"] = float(c.max())
        else:
            out["conf_mean"] = out["conf_median"] = out["conf_min"] = out
    if "daynight" in df.columns:
        out["daynight_counts"] = df["daynight"].value_counts(dropna=False

    if "satellite" in df.columns:
        out["satellite_counts"] = df["satellite"].value_counts(dropna=Fal
    if "instrument" in df.columns:
        out["instrument_counts"] = df["instrument"].value_counts(dropna=F

    daily = (
        df.assign(day=df["acq_datetime"].dt.floor("D"))
          .groupby("day")
          .agg(detections=("acq_datetime", "count"),
               frp_sum=("frp", "sum"))
          .reset_index()
    )
    out["daily"] = daily
    return out

def plot_and_save_figures(df, reports_dir: Path, tag: str):
    reports_dir.mkdir(parents=True, exist_ok=True)

    daily = df.assign(day=df["acq_datetime"].dt.floor("D")).groupby("day"
    fig1 = plt.figure(figsize=(8, 4.5))
    plt.plot(daily["day"], daily["detections"])
    plt.title("Daily FIRMS Detections")
    plt.xlabel("Date (UTC)")
    plt.ylabel("Count")
    plt.xticks(rotation=45, ha="right")
    plt.tight_layout()
    f1 = reports_dir / f"firms_daily_detections_{tag}.png"
    fig1.savefig(f1, dpi=160)
    plt.close(fig1)

    if "frp" in df.columns and pd.to_numeric(df["frp"], errors="coerce").
        frp = pd.to_numeric(df["frp"], errors="coerce").dropna()
        fig2 = plt.figure(figsize=(7.5, 4.5))
        plt.hist(frp, bins=60)
        plt.xscale("log")
        plt.title("FRP Distribution (log scale)")
        plt.xlabel("FRP (MW, log scale)")
        plt.ylabel("Frequency")
        plt.tight_layout()
        f2 = reports_dir / f"firms_frp_hist_{tag}.png"
        fig2.savefig(f2, dpi=160)
        plt.close(fig2)
    else:
        f2 = None

    subs = df.sample(min(5000, len(df)), random_state=42) if len(df) > 0
    fig3 = plt.figure(figsize=(6.5, 6))
    plt.scatter(subs["longitude"], subs["latitude"], s=4, alpha=0.6)
    plt.title("Geographic Distribution (subsample)")
    plt.xlabel("Longitude")
    plt.ylabel("Latitude")
    plt.tight_layout()
    f3 = reports_dir / f"firms_geo_scatter_{tag}.png"
    fig3.savefig(f3, dpi=160)
    plt.close(fig3)
    return {"daily": str(f1), "frp_hist": (str(f2) if f2 else None), "geo

# --------- Run QA ---------
if "df_fires" not in globals() or df_fires is None or len(df_fires) == 0:
    raise RuntimeError("No FIRMS data in df_fires. Run Cell 12 first.")

issues = validate_firms_schema(df_fires)
if issues["errors"]:
    print("❌ Schema/quality errors:")
    for e in issues["errors"]:
        print(" -", e)
    raise RuntimeError("Stop: critical QA errors found.")
else:
    print(" Schema checks passed.")

if issues["warnings"]:
    print("⚠️ Warnings:")
    for w in issues["warnings"]:
        print(" -", w)

summary = summarize_firms(df_fires)
print("\n=== FIRMS Summary ===")
print(f"Rows: {summary.get('n_rows')}")
print(f"Time range (UTC): {summary.get('time_min')} → {summary.get('time_
print(f"Potential duplicates (post-clean): {summary.get('n_potential_dupl

if "frp_count" in summary:
    print(f"FRP count: {summary['frp_count']}")

if "conf_count" in summary:
    print(f"Confidence count: {summary['conf_count']}")

# Save daily table and figures
tag = _now_stamp()
daily_out = PROCESSED_DIR / f"firms_daily_summary_{tag}.csv"
summary["daily"].to_csv(daily_out, index=False)
print(f"\nSaved daily summary: {daily_out}")

fig_paths = plot_and_save_figures(df_fires, REPORTS_DIR, tag)
print("Saved figures:")
for k, v in fig_paths.items():
    print(f" - {k}: {v}")

# Preview daily summary
summary["daily"].head(5)
```

```
✅ Schema checks passed.

=== FIRMS Summary ===
Rows: 475
Time range (UTC): 2025-09-08 01:43:00+00:00 → 2025-09-14 14:33:00+00:00
Potential duplicates (post-clean): 0
FRP count: 475
Confidence count: 0

Saved daily summary: /Users/evareysanchez/WildfiresAI/data/processed/firms
_daily_summary_20250914T183152Z.csv
Saved figures:
 - daily: /Users/evareysanchez/WildfiresAI/reports/firms_daily_detections_
20250914T183152Z.png
 - frp_hist: /Users/evareysanchez/WildfiresAI/reports/firms_frp_hist_20250
914T183152Z.png
 - geo_scatter: /Users/evareysanchez/WildfiresAI/reports/firms_geo_scatter
_20250914T183152Z.png
```

Out[30]:

	day	detections	frp_sum
0	2025-09-08 00:00:00+00:00	40	644.59
1	2025-09-09 00:00:00+00:00	36	139.64
2	2025-09-10 00:00:00+00:00	44	81.15
3	2025-09-11 00:00:00+00:00	72	322.06
4	2025-09-12 00:00:00+00:00	63	499.73

In [ ]: