Section 1: Environment Setup & Data Organization

0.1 Environment & Library versions (Colab)

```
# Colab environemnt set up
!pip -q install pandas pyarrow fastparquet geopandas shapely rasterio
scikit-learn matplotlib seaborn

import os, sys, json, textwrap, zipfile, warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import geopandas as gpd
import rasterio

pd.set_option("display.max_columns", 120)
pd.set_option("display.width", 180)

print("Python version:", sys.version)
print("Pandas:", pd.__version__)
print("Numpy:", np.__version__)
print("Scikit-learn:", sklearn.__version__)
print("GeoPandas:", gpd.__version__)
print("Rasterio:", rasterio.__version__)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 11.4 MB/s eta
0:00:00
ain, Oct 10 2025, 08:52:57) [GCC 11.4.0]
Pandas: 2.2.2
Numpy: 2.0.2
Scikit-learn: 1.6.1
GeoPandas: 1.1.1
Rasterio: 1.4.3
```

0.2 Mount Google Drive and set up project directories

```
from google.colab import drive
drive.mount('/content/drive')

PROJECT_ROOT = "/content/drive/MyDrive/CSE475 - Final Project"

RANDOM_STATE = 42
```

```
# Standard subfolders
RAW_DIR     = os.path.join(PROJECT_ROOT, "Datasets")        # raw
uploads
INTERIM_DIR = os.path.join(PROJECT_ROOT, "interim")         #
cleaned/intermediate
REPORTS_DIR = os.path.join(PROJECT_ROOT, "reports_figures") #
charts/tables

for p in [INTERIM_DIR, REPORTS_DIR]:
    os.makedirs(p, exist_ok=True)

print("\nProject root:", PROJECT_ROOT)
print("\nRAW_DIR:", RAW_DIR)
print("\nINTERIM_DIR:", INTERIM_DIR)
print("\nREPORTS_DIR:", REPORTS_DIR)
```

```
Mounted at /content/drive

Project root: /content/drive/MyDrive/CSE475 - Final Project

RAW_DIR: /content/drive/MyDrive/CSE475 - Final Project/Datasets

INTERIM_DIR: /content/drive/MyDrive/CSE475 - Final Project/interim

REPORTS_DIR: /content/drive/MyDrive/CSE475 - Final
Project/reports_figures
```

1.1 Inspect available raw datasets

```
from pathlib import Path
from collections import Counter

def walk_dir(d, exts=None):
    d = Path(d)
    for p in sorted(d.rglob("*")):
        if p.is_file():
            if exts is None or p.suffix.lower() in exts:
                size_mb = p.stat().st_size / (1024*1024)
                print(f"{p.relative_to(d)}  [{size_mb:,.2f} MB]")

# When no files are found
files = list(Path(RAW_DIR).rglob("*"))

ext_counts = Counter([p.suffix.lower() for p in files if p.is_file()])

if not files:
    print("No files found in RAW_DIR!")

# When files are found
print(f"Found {len(files)} files total.")
```

```
print("\n== RAW ==")
walk_dir(RAW_DIR, exts={".csv", ".zip", ".tif", ".geojson",
".parquet", ".xlsx"})
print("\nFile type counts:", ext_counts)
```

```
Found 6 files total.

== RAW ==
647_IET_measurements_50e72be656c7407bd07e316061beeb1d.csv  [0.10 MB]
All_LC_Sensor_Climate.csv  [34.10 MB]
GlobalLandTemperaturesByCity.csv  [508.15 MB]
HLS.L30.T12SVB.2024209T175727.v2.0.Fmask.tif  [1.50 MB]
LULC_temp_prediction.csv  [83.00 MB]
Where_Will_Tree_Planting_Improve_Urban_Heat_Health%3F.csv  [113.03 MB]

File type counts: Counter({'.csv': 5, '.tif': 1})
```

1.2 Load & validate core datasets (Kaggle GLT, CAP LTER IET)

```
# Confirming data integrity of main CSV files
from pathlib import Path
import pandas as pd

# --- Kaggle: Global Land Temperatures by City ---
kaggle_path = Path(RAW_DIR) / "GlobalLandTemperaturesByCity.csv"
if not kaggle_path.exists():
    folder_alt = Path(RAW_DIR) / "GlobalLandTemperaturesByCity"
    candidates = list(folder_alt.rglob("*.csv")) if
folder_alt.exists() else []
    if candidates:
        kaggle_path = candidates[0]

print(f" Kaggle dataset path: {kaggle_path}")
temps_city = pd.read_csv(kaggle_path)
print(f" Loaded Kaggle temperature data: {temps_city.shape[0]:,} rows
× {temps_city.shape[1]} columns")
display(temps_city.head(3))

# --- CAP LTER: IET Measurements ---
iet_candidates = sorted(Path(RAW_DIR).rglob("*IET*measurements*.csv"))
assert len(iet_candidates) >= 1, " IET measurements CSV not found in
RAW_DIR"
iet_path = iet_candidates[0]

print(f"\n CAP LTER dataset path: {iet_path}")
iet = pd.read_csv(iet_path)
print(f" Loaded CAP LTER IET data: {iet.shape[0]:,} rows ×
{iet.shape[1]} columns")
display(iet.head(3))
```

```
 Kaggle dataset path: /content/drive/MyDrive/CSE475 - Final
Project/Datasets/GlobalLandTemperaturesByCity.csv
 Loaded Kaggle temperature data: 8,599,212 rows × 7 columns
```


```
{"summary":"{\n  \"name\": \"display(iet\",\n  \"rows\": 3,\n
\"fields\": [\n    {\n      \"column\": \"dt\",\n      \"properties\":
{\n        \"dtype\": \"object\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"1743-11-01\",\n          \"1743-12-01\",\n
\"1744-01-01\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"AverageTemperature\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 6.068,\n
\"max\": 6.068,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          6.068\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"AverageTemperatureUncertainty\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
null,\n        \"min\": 1.737,\n        \"max\": 1.737,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          1.737\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"City\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
1,\n        \"samples\": [\n          \"\\u00c5rhus\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Country\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
1,\n        \"samples\": [\n          \"Denmark\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Latitude\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
1,\n        \"samples\": [\n          \"57.05N\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Longitude\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"10.33E\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
 CAP LTER dataset path: /content/drive/MyDrive/CSE475 - Final
Project/Datasets/647_IET_measurements_50e72be656c7407bd07e316061beeb1d
.csv
 Loaded CAP LTER IET data: 3,360 rows × 3 columns
```


```
{"summary":"{\n  \"name\": \"display(iet\",\n  \"rows\": 3,\n
\"fields\": [\n    {\n      \"column\": \"Subject ID\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"1A\",\n
\"1P\",\n          \"1E\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
```

```
\"column\": \"period\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"Sat, 8pm-12am\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"temperature\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0931014639478256,\n
\"min\": 25.14446667,\n        \"max\": 27.21875,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
25.14446667\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

print("\n=== Loading Additional Raw Datasets ===\n")

from pathlib import Path
import pandas as pd
import rasterio

# 1) All_LC_Sensor_Climate.csv
lc_climate_candidates =
list(Path(RAW_DIR).rglob("All_LC_Sensor_Climate*.csv"))
if lc_climate_candidates:
    lc_climate_path = lc_climate_candidates[0]
    print(" LC Sensor Climate path:", lc_climate_path)
    lc_climate = pd.read_csv(lc_climate_path)
    print(f" Loaded LC climate data: {lc_climate.shape[0]:,} rows ×
{lc_climate.shape[1]} columns")
    display(lc_climate.head(3))
else:
    print(" LC Sensor Climate dataset not found.")

# 2) LULC_temp_prediction.csv
lulc_candidates =
list(Path(RAW_DIR).rglob("LULC_temp_prediction*.csv"))
if lulc_candidates:
    lulc_path = lulc_candidates[0]
    print("\n LULC prediction path:", lulc_path)
    lulc = pd.read_csv(lulc_path)
    print(f" Loaded LULC prediction data: {lulc.shape[0]:,} rows ×
{lulc.shape[1]} columns")
    display(lulc.head(3))
else:
    print(" LULC prediction dataset not found.")

# 3) Tree-planting / Heat-Health dataset
tree_candidates =
list(Path(RAW_DIR).rglob("Where_Will_Tree_Planting*"))
if tree_candidates:
    tree_path = tree_candidates[0]
    print("\n Tree planting path:", tree_path)
    tree_df = pd.read_csv(tree_path)
```

```
    print(f" Loaded tree planting dataset: {tree_df.shape[0]:,} rows ×
{tree_df.shape[1]} columns")
    display(tree_df.head(3))
else:
    print(" Tree planting dataset not found.")

# 4) HLS Fmask raster (cloud/shade mask)
hls_candidates = list(Path(RAW_DIR).rglob("*.tif"))
if hls_candidates:
    hls_path = hls_candidates[0]
    print("\n HLS TIF path:", hls_path)
    with rasterio.open(hls_path) as src:
        print(" Raster CRS:", src.crs)
        print(" Raster size:", src.width, "x", src.height)
        print(" Raster bands:", src.count)
else:
    print(" No .tif files found for HLS.")


=== Loading Additional Raw Datasets ===

 LC Sensor Climate path: /content/drive/MyDrive/CSE475 - Final
Project/Datasets/All_LC_Sensor_Climate.csv
 Loaded LC climate data: 259,244 rows × 16 columns

{"summary":"{\n  \"name\": \"    print(\\\" No \",\n  \"rows\": 3,\n
\"fields\": [\n    {\n      \"column\": \"Date\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"6/11/2018\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Hour\",\n      \"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"13:00\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ibuttonID\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"M04\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Lat\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.050824990244302086,\n        \"min\": 36.13335,\n
\"max\": 36.23288,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          36.16523\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Lon\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.09606884319764339,\n
\"min\": -115.3001,\n        \"max\": -115.10812,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          -
115.10812\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"Impervious Surface\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.24336366451234012,\n        \"min\":
0.0,\n        \"max\": 0.486352358,\n        \"num_unique_values\":
3,\n        \"samples\": [\n          0.259718776\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Turf/Grass\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.393478163787549,\n        \"min\": 0.031430935,\n        \"max\":
0.816528925,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0.377171216\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Tree Canopy\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.14939514284770888,\n
\"min\": 0.183471074,\n        \"max\": 0.482216708,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.328370554\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Buildings\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.020056833118400626,\n        \"min\":
0.0,\n        \"max\": 0.034739454,\n        \"num_unique_values\":
2,\n        \"samples\": [\n          0.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Regional Air Temperature\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0,\n        \"min\": 36.66666667,\n        \"max\": 36.66666667,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
36.66666667\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Regional Relative Humidity\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\": 6.0,\n
\"max\": 6.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Regional Wind Speed\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 0.0,\n
\"max\": 0.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Mean Landsat Land Surface Temperature\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2.817490405776214,\n
\"min\": 42.93605986,\n        \"max\": 48.56359706,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
48.56359706\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Sensor Recorded Air Temperature\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.2583057392117918,\n
\"min\": 35.5,\n        \"max\": 38.0,\n        \"num_unique_values\":
3,\n        \"samples\": [\n          38.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"City\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          \"Las Vegas\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Diel\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          \"Daytime\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
 LULC prediction path: /content/drive/MyDrive/CSE475 - Final
Project/Datasets/LULC_temp_prediction.csv
 Loaded LULC prediction data: 693,932 rows × 14 columns
```


```
{"summary":"{\n  \"name\": \"    print(\\\" No \",\n  \"rows\": 3,\n
\"fields\": [\n    {\n      \"column\": \"City\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"Baltimore\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Latitude\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0001573922912118159,\n        \"min\":
39.37263,\n        \"max\": 39.372907,\n        \"num_unique_values\":
3,\n        \"samples\": [\n          39.372907\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Longitude\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.011864759851486013,\n        \"min\": -76.55342,\n        \"max\": -
76.53253,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
-76.53253\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Impervious\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.17440625360443168,\n        \"min\":
0.028056,\n        \"max\": 0.371389,\n        \"num_unique_values\":
3,\n        \"samples\": [\n          0.028056\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Grass\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\":
0.10444362051524896,\n        \"min\": 0.203889,\n        \"max\":
0.393611,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
0.203889\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Tree\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.2657837656485688,\n        \"min\": 0.227778,\n
\"max\": 0.701389,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.701389\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Building\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
```

```
0.07307442736233609,\n        \"min\": 0.066667,\n        \"max\":
0.204444,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
0.066667\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"LST\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2.130306504551242,\n        \"min\": 28.1021,\n
\"max\": 31.9305,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          28.1021\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Tair_pred25_75\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.9641304694917242,\n        \"min\": 27.721166,\n        \"max\":
29.631663,\n        \"num_unique_values\": 3,\n        \"samples\": [\
n          27.721166\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Tair_pred95\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.6339512703121064,\n        \"min\":
32.160609,\n        \"max\": 33.329234,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
32.160609\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Tair_pred25_75_nt\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.2721682779464459,\n        \"min\":
21.889037,\n        \"max\": 22.371962,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
22.371962\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Tair_pred95_nt\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.5316248162062545,\n        \"min\":
24.748836,\n        \"max\": 25.723821,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
24.869003\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Tair_diff_day\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.3884560013454976,\n        \"min\":
3.697571,\n        \"max\": 4.439443,\n        \"num_unique_values\":
3,\n        \"samples\": [\n          4.439443\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Tair_diff_nt\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.44160241304141445,\n        \"min\": 2.497041,\n        \"max\":
3.375803,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
2.497041\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
 Tree planting path: /content/drive/MyDrive/CSE475 - Final
Project/Datasets/Where_Will_Tree_Planting_Improve_Urban_Heat_Health
%3F.csv
 Loaded tree planting dataset: 72,246 rows × 194 columns
```

{"type":"dataframe"}

```
 HLS TIF path: /content/drive/MyDrive/CSE475 - Final
Project/Datasets/HLS.L30.T12SVB.2024209T175727.v2.0.Fmask.tif
 Raster CRS: EPSG:32612
 Raster size: 3660 x 3660
 Raster bands: 1
```

1.3 Inspect schema and normalize column names

```
# What columns do I have, and what do they look like?

# Are the column names consistent and easy to work with?

# Are dates properly formatted (as datetimes, not strings)?

def peek(df, name, n=3):
    print(f"\n=== {name} ===")
    print("Shape:", df.shape)
    display(df.sample(min(n, len(df))).head(n))
    print(df.dtypes.head(20))

peek(temps_city, "GlobalLandTemperaturesByCity")
peek(iet, "CAP-LTER IET measurements")

def clean_cols(df):
    return df.rename(columns={c: c.strip().lower().replace(" ", "_")
for c in df.columns})

# ---- Standardize column names for all datasets (where available)
----

datasets = {
    "temps_city": temps_city,
    "iet": iet,
    "lc_climate": lc_climate if 'lc_climate' in globals() else None,
    "lulc": lulc if 'lulc' in globals() else None,
    "tree_df": tree_df if 'tree_df' in globals() else None
}

for name, df in datasets.items():
    if df is None:
        print(f"\n{name}: Not loaded, skipping.")
        continue
    print(f"\nCleaning column names for {name} ...")
    df = clean_cols(df)
    datasets[name] = df  # update reference

temps_city = datasets["temps_city"]
```

```
iet        = datasets["iet"]

if "lc_climate" in datasets: lc_climate = datasets["lc_climate"]
if "lulc" in datasets: lulc = datasets["lulc"]
if "tree_df" in datasets: tree_df = datasets["tree_df"]

possible_date_cols = ["dt", "date", "time", "timestamp", "day"]

for name, df in datasets.items():
    if df is None:
        continue
    print(f"\nParsing date columns in {name} ...")
    for col in df.columns:
        if any(key in col for key in possible_date_cols):
            try:
                df[col] = pd.to_datetime(df[col], errors="ignore")
            except:
                pass

print("\n=== Schema inspection for ALL datasets ===")
for name, df in datasets.items():
    if df is not None:
        peek(df, name)
```

```
=== GlobalLandTemperaturesByCity ===
Shape: (8599212, 7)
```


```
{"repr_error":"0","type":"dataframe"}
```


```
dt                                object
AverageTemperature               float64
AverageTemperatureUncertainty    float64
City                              object
Country                           object
Latitude                          object
Longitude                         object
dtype: object

=== CAP-LTER IET measurements ===
Shape: (3360, 3)
```


```
{"summary":"{\n  \"name\": \"        peek(df, name)\",\n  \"rows\":
3,\n  \"fields\": [\n    {\n      \"column\": \"Subject ID\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"3J\",\n
\"2T\",\n          \"3F\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"period\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Sat, 4am-8am\",\n          \"Mon, 8am-12pm\",\n
```

\"Thu, 8pm-12am\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"temperature\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 2.566582601690832,\n   \"min\":
25.2811875,\n   \"max\": 30.364625,\n
\"num_unique_values\": 3,\n   \"samples\": [\n
25.2811875,\n   30.364625,\n   27.205625\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
Subject ID      object
period          object
temperature    float64
dtype: object

Cleaning column names for temps_city ...

Cleaning column names for iet ...

Cleaning column names for lc_climate ...

Cleaning column names for lulc ...

Cleaning column names for tree_df ...

Parsing date columns in temps_city ...

Parsing date columns in iet ...

Parsing date columns in lc_climate ...

Parsing date columns in lulc ...

Parsing date columns in tree_df ...

=== Schema inspection for ALL datasets ===

=== temps_city ===
Shape: (8599212, 7)
```

{"summary":"{\n \"name\": \"   peek(df, name)\",\n \"rows\":
3,\n \"fields\": [\n   {\n   \"column\": \"dt\",\n
\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":
\"1838-06-01 00:00:00\",\n   \"max\": \"1921-04-01 00:00:00\",\n
\"num_unique_values\": 3,\n   \"samples\": [\n   \"1921-
04-01 00:00:00\",\n   \"1838-06-01 00:00:00\",\n
\"1849-06-01 00:00:00\"\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"averagetemperature\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 7.7324838398365445,\n
\"min\": 12.396,\n   \"max\": 27.679,\n

\"num_unique_values\": 3,\n   \"samples\": [\n   27.679,\n
12.396,\n   17.989\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"averagetemperatureuncertainty\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
0.48157277054805886,\n   \"min\": 0.531,\n   \"max\":
1.494,\n   \"num_unique_values\": 3,\n   \"samples\": [\n
0.531,\n   1.027,\n   1.494\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"city\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   \"Onitsha\",\n   \"Ulan Ude\",\n
\"Erzincan\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"country\",\n   \"properties\": {\n   \"dtype\": \"string\",\
n   \"num_unique_values\": 3,\n   \"samples\": [\n
\"Nigeria\",\n   \"Russia\",\n   \"Turkey\"\n   ],\
n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"latitude\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n   \"samples\": [\n
\"5.63N\",\n   \"52.24N\",\n   \"39.38N\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"longitude\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n   \"samples\": [\n
\"6.46E\",\n   \"107.74E\",\n   \"39.54E\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
dt                                datetime64[ns]
averagetemperature                       float64
averagetemperatureuncertainty            float64
city                                      object
country                                   object
latitude                                  object
longitude                                 object
dtype: object

=== iet ===
Shape: (3360, 3)
```

{"summary":"{\n \"name\": \"   peek(df, name)\",\n \"rows\":
3,\n \"fields\": [\n   {\n   \"column\": \"subject_id\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n   \"samples\": [\n   \"2N\",\n
\"4D\",\n   \"2R\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"period\",\n   \"properties\": {\n   \"dtype\":
\"string\",\n   \"num_unique_values\": 2,\n   \"samples\":

[\n   \"Sun, 12am-4am\",\n   \"Mon, 12pm-4pm\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"temperature\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 2.87075590432896,\n   \"min\":
24.542875,\n   \"max\": 30.0209375,\n
\"num_unique_values\": 3,\n   \"samples\": [\n
28.7708125,\n   30.0209375\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
subject_id      object
period          object
temperature    float64
dtype: object
```

```
=== lc_climate ===
Shape: (259244, 16)
```

{"repr_error":"0","type":"dataframe"}

```
date                                          object
hour                                          object
ibuttonid                                     object
lat                                          float64
lon                                          float64
impervious_surface                           float64
turf/grass                                   float64
tree_canopy                                  float64
buildings                                    float64
regional_air_temperature                     float64
regional_relative_humidity                   float64
regional_wind_speed                          float64
mean_landsat_land_surface_temperature        float64
sensor_recorded_air_temperature              float64
city                                          object
diel                                          object
dtype: object
```

```
=== lulc ===
Shape: (693932, 14)
```

{"summary":"{\n \"name\": \"   peek(df, name)\",\n \"rows\":
3,\n \"fields\": [\n   {\n   \"column\": \"city\",\n
\"properties\": {\n   \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n   \"samples\": [\n
\"Portland\",\n   \"Phoenix\",\n   \"Tucson\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"latitude\",\n   \"properties\": {\n   \"dtype\":

```
\"number\",\n        \"std\": 4.137205982200588,\n        \"min\":
25.721159,\n        \"max\": 33.390451,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
25.721159,\n          33.390451,\n          32.245766\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"longitude\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
17.964782194114097,\n        \"min\": -111.836904,\n        \"max\": -
80.283348,\n        \"num_unique_values\": 3,\n        \"samples\": [\
n          -80.283348,\n          -111.836904,\n          -110.942331\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"impervious\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.10586812553958476,\n        \"min\":
0.0,\n        \"max\": 0.209338,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.077148438,\n          0.0,\n
0.209338\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"grass\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.10914838844921466,\n        \"min\": 0.0,\n        \"max\":
0.200195313,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0.200195313,\n          0.024722,\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tree\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.32636034969953875,\n        \"min\": 0.007222,\n        \"max\":
0.607421875,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0.607421875,\n          0.007222,\n          0.529478\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"building\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.14337194304080864,\n        \"min\": 0.115234375,\n        \"max\":
0.370556,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
0.115234375,\n          0.370556,\n          0.12987\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lst\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 8.222782098075829,\n
\"min\": 37.395303,\n        \"max\": 53.5526,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
37.395303,\n          53.5526,\n          48.1289\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tair_pred25_75\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
6.741805388089479,\n        \"min\": 29.371737,\n        \"max\":
41.141625,\n        \"num_unique_values\": 3,\n        \"samples\": [\
n          29.371737,\n          40.953884,\n          41.141625\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tair_pred95\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
7.001590953105037,\n        \"min\": 32.943798,\n        \"max\":
45.416985,\n        \"num_unique_values\": 3,\n        \"samples\": [\
n          32.943798,\n          45.416985,\n          44.692336\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tair_pred25_75_nt\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.5281715014299047,\n        \"min\": 25.453947,\n        \"max\":
30.40631,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
25.453947,\n          30.40631,\n          27.046653\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tair_pred95_nt\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4.3138771298515595,\n        \"min\": 25.730429,\n        \"max\":
34.266821,\n        \"num_unique_values\": 3,\n        \"samples\": [\
n          25.730429,\n          34.266821,\n          31.083114\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tair_diff_day\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"1970-01-01 00:00:00.000000003\",\n        \"max\": \"1970-01-01
00:00:00.000000004\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"1970-01-01 00:00:00.000000004\",\n
\"1970-01-01 00:00:00.000000003\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tair_diff_nt\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.1218570622523876,\n        \"min\": 0.276482,\n        \"max\":
4.036461,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
0.276482,\n          3.860511\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


```
city                          object
latitude                     float64
longitude                    float64
impervious                   float64
grass                        float64
tree                         float64
building                     float64
lst                          float64
tair_pred25_75               float64
tair_pred95                  float64
tair_pred25_75_nt            float64
tair_pred95_nt               float64
tair_diff_day         datetime64[ns]
tair_diff_nt                 float64
dtype: object

=== tree_df ===
Shape: (72246, 194)
```

```
{"type":"dataframe"}
```

```
objectid                          int64
geoid                             int64
name                             object
state                            object
county                           object
aland                             int64
awater                            int64
b01001_001e                       int64
b01001_calc_pctge65e            float64
b01001_calc_pctge65e_box_cox    float64
b03002_calc_pctnhwhitee         float64
pct_pop_minority                float64
b01001_003e                       int64
b01001_027e                       int64
total_young_people                int64
pct_young_people                float64
b17020_calc_pctpove             float64
b25002_calc_pctvace             float64
b15002_calc_pctlthse            float64
b08201_calc_pctnovehe           float64
dtype: object
```

Section 2 – Data Loading & Cleaning

2.1 Clean CAP-LTER IET Dataset

```
# ============================================================
# 2.1 Clean CAP-LTER IET Dataset (period parsing + TOD bucketing)
# ============================================================

import re
import pandas as pd
import numpy as np

df = iet.copy()

# 1) Clean temperature column
df["temperature"] = pd.to_numeric(df["temperature"], errors="coerce")

# 2) Parse "Sat, 8pm-12am" style periods
day_map = {"Mon":0,"Tue":1,"Wed":2,"Thu":3,"Fri":4,"Sat":5,"Sun":6}

def to_24h(hh, ampm):
    hh = int(hh)
    if ampm.lower() == "am":
        return 0 if hh == 12 else hh
```

```
    return 12 if hh == 12 else hh + 12

def parse_period(s):
    m = re.match(r"\s*([A-Za-z]{3})\s*,\s*(\d{1,2})\s*([ap]m)\s*-\s*(\
d{1,2})\s*([ap]m)\s*$",
                 str(s))
    if not m:
        return pd.Series([np.nan, np.nan, np.nan, np.nan],

index=["dow","start_hour","end_hour","duration_hr"])
    dow, h1, ampm1, h2, ampm2 = m.groups()
    s_h = to_24h(h1, ampm1)
    e_h = to_24h(h2, ampm2)
    duration = (e_h - s_h) % 24
    duration = 24 if duration == 0 else duration
    return pd.Series([day_map.get(dow, np.nan), s_h, e_h, duration],

index=["dow","start_hour","end_hour","duration_hr"])

parsed = df["period"].apply(parse_period)
df = pd.concat([df.drop(columns=["period"]), parsed], axis=1)

# 3) Time-of-day bucketing
def tod_bucket(h):
    if pd.isna(h): return np.nan
    h = int(h)
    if 5 <= h < 12:   return "morning"
    if 12 <= h < 17:  return "afternoon"
    if 17 <= h < 21:  return "evening"
    return "night"

df["tod"] = df["start_hour"].apply(tod_bucket)

# 4) Combine cleaned IET columns into iet_small
iet_small = pd.DataFrame()
required_cols = ["temperature", "dow", "start_hour", "duration_hr",
"tod"]

for col in required_cols:
    if col in df.columns:
        iet_small[col] = df[col]
    else:
        print(f"Warning: missing column: {col}")

# Remove rows missing key fields
iet_small = iet_small.dropna(subset=["temperature", "dow",
"start_hour", "tod"])

peek(iet_small, "IET cleaned subset")
print("IET shape:", iet_small.shape)
```

```
=== IET cleaned subset ===
Shape: (3219, 5)
```


```
{"summary":"{\n  \"name\": \"print(\\\"IET shape:\\\", iet_small\",\n
\"rows\": 3,\n  \"fields\": [\n    {\n      \"column\":
\"temperature\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.715994702121288,\n        \"min\":
23.2603125,\n        \"max\": 26.4546875,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
26.4546875,\n          23.7708125,\n          23.2603125\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"dow\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 1,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"start_hour\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8,\n        \"min\": 4,\n
\"max\": 20,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          20\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"duration_hr\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 4,\n
\"max\": 4,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"tod\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"evening\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
temperature    float64
dow              int64
start_hour       int64
duration_hr      int64
tod             object
dtype: object
IET shape: (3219, 5)
```

2.2 Clean Kaggle Arizona Temperature Data

```
# ============================================================
# 2.2 Clean Kaggle Phoenix Metro Temperature Dataset
# ============================================================

# --- Parse latitude/longitude strings ---
def parse_lat_lon(coord_str):
    if pd.isna(coord_str): return np.nan
    s = str(coord_str).strip()
```

```
    if s.endswith("N"): return float(s[:-1])
    if s.endswith("S"): return -float(s[:-1])
    if s.endswith("E"): return float(s[:-1])
    if s.endswith("W"): return -float(s[:-1])
    try:
        return float(s)
    except:
        return np.nan

temps_city["latitude"]  = temps_city["latitude"].apply(parse_lat_lon)
temps_city["longitude"] = temps_city["longitude"].apply(parse_lat_lon)

# --- Filter for Arizona cities ---
az_city_names = ["Phoenix", "Tempe", "Mesa", "Scottsdale", "Glendale"]
city_mask = temps_city["city"].isin(az_city_names)

latlon_mask = (
    (temps_city["country"] == "United States") &
    (temps_city["latitude"].between(31, 37)) &
    (temps_city["longitude"].between(-115, -109))
)

temps_az = temps_city[city_mask & latlon_mask].copy()

# Rename coordinate columns
temps_az = temps_az.rename(columns={"latitude":"lat",
"longitude":"lon"})

# Rename temperature column if needed
if "averagetemperature" in temps_az.columns:
    temps_az = temps_az.rename(columns={"averagetemperature":
"avg_temp_c"})

# Keep relevant columns
keep_cols = ["dt", "city", "country", "lat", "lon", "avg_temp_c"]
keep_cols = [c for c in keep_cols if c in temps_az.columns]
temps_az = temps_az[keep_cols].dropna(subset=["avg_temp_c", "lat",
"lon"])

peek(temps_az, "Kaggle AZ cleaned subset")
print("Kaggle AZ shape:", temps_az.shape)


=== Kaggle AZ cleaned subset ===
Shape: (10695, 6)

{"summary":"{\n  \"name\": \"print(\\\"Kaggle AZ shape:\\\",
temps_az\",\n  \"rows\": 3,\n  \"fields\": [\n    {\n      \"column\":
\"dt\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": \"1886-06-01 00:00:00\",\n        \"max\": \"1990-07-01
00:00:00\",\n        \"num_unique_values\": 3,\n        \"samples\":
```

```
[\n          \"1898-02-01 00:00:00\",\n          \"1990-07-01
00:00:00\",\n          \"1886-06-01 00:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"city\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"Scottsdale\",\n          \"Glendale\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"country\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n          \"United
States\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"lat\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0,\n        \"min\": 32.95,\n        \"max\": 32.95,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          32.95\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"lon\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 0.0,\n
\"min\": -112.02,\n        \"max\": -112.02,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          -112.02\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"avg_temp_c\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9.50008296455001,\n        \"min\": 14.198,\n        \"max\": 31.83,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          14.198\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


```
dt             datetime64[ns]
city                   object
country                object
lat                   float64
lon                   float64
avg_temp_c            float64
dtype: object
Kaggle AZ shape: (10695, 6)
```

2.3 Clean LC Sensor Climate Dataset

```
# ============================================================
# 2.3 Clean Land-Cover Sensor Climate Dataset (All_LC_Sensor_Climate)
# ============================================================

if "lc_climate" not in globals() or lc_climate is None:
    print("lc_climate not loaded, skipping Section 2.3.")
else:
    print("Raw LC climate shape:", lc_climate.shape)
    peek(lc_climate, "LC climate (raw)")

    # Ensure columns are standardized
```

```
    lc_climate = clean_cols(lc_climate)

    # 1) Try to parse any obvious date/time column
    lc_date_candidates = [c for c in lc_climate.columns if any(x in c
for x in ["date", "time", "timestamp", "datetime"])]
    for c in lc_date_candidates:
        try:
            lc_climate[c] = pd.to_datetime(lc_climate[c],
errors="coerce")
            print(f"Parsed datetime column in lc_climate: {c}")
        except Exception as e:
            print(f"Could not parse datetime column {c}: {e}")

    # 2) Convert likely numeric climate variables
    numeric_like = [c for c in lc_climate.columns if any(x in c for x
in ["temp", "tair", "rh", "humidity", "ws", "wind", "pressure"])]
    for c in numeric_like:
        lc_climate[c] = pd.to_numeric(lc_climate[c], errors="coerce")

    # 3) Drop rows with no climate values at all
    if numeric_like:
        lc_climate_clean = lc_climate.dropna(subset=numeric_like,
how="all").copy()
    else:
        lc_climate_clean = lc_climate.copy()

    # 4) Optional: rename lat/lon
    rename_geo = {}
    if "latitude" in lc_climate_clean.columns and "lat" not in
lc_climate_clean.columns:
        rename_geo["latitude"] = "lat"
    if "longitude" in lc_climate_clean.columns and "lon" not in
lc_climate_clean.columns:
        rename_geo["longitude"] = "lon"
    if rename_geo:
        lc_climate_clean = lc_climate_clean.rename(columns=rename_geo)

    peek(lc_climate_clean, "LC climate (clean)")
    print("LC climate (clean) shape:", lc_climate_clean.shape)
```

```
Raw LC climate shape: (259244, 16)

=== LC climate (raw) ===
Shape: (259244, 16)
```

{"repr_error":"0","type":"dataframe"}

```
date                                        object
hour                                        object
ibuttonid                                   object
lat                                        float64
```

```
lon                                          float64
impervious_surface                           float64
turf/grass                                   float64
tree_canopy                                  float64
buildings                                    float64
regional_air_temperature                     float64
regional_relative_humidity                   float64
regional_wind_speed                          float64
mean_landsat_land_surface_temperature        float64
sensor_recorded_air_temperature              float64
city                                          object
diel                                          object
dtype: object
Parsed datetime column in lc_climate: date

=== LC climate (clean) ===
Shape: (259244, 16)
```

{"repr_error":"0","type":"dataframe"}

```
date                                    datetime64[ns]
hour                                            object
ibuttonid                                       object
lat                                            float64
lon                                            float64
impervious_surface                             float64
turf/grass                                     float64
tree_canopy                                    float64
buildings                                      float64
regional_air_temperature                       float64
regional_relative_humidity                     float64
regional_wind_speed                            float64
mean_landsat_land_surface_temperature          float64
sensor_recorded_air_temperature                float64
city                                            object
diel                                            object
dtype: object
LC climate (clean) shape: (259244, 16)
```

2.4 Clean LULC Temprature Prediction Dataset

```python
# ============================================================
# 2.4 Clean LULC Temperature Prediction Dataset (LULC_temp_prediction)
# ============================================================

if "lulc" not in globals() or lulc is None:
    print("lulc not loaded, skipping Section 2.4.")
else:
    print("Raw LULC shape:", lulc.shape)
    peek(lulc, "LULC (raw)")
```

```
    # Ensure standardized column names
    lulc = clean_cols(lulc)

    # 1) Parse any datetime columns
    lulc_date_candidates = [c for c in lulc.columns if any(x in c for
x in ["date", "time", "timestamp", "datetime"])]
    for c in lulc_date_candidates:
        try:
            lulc[c] = pd.to_datetime(lulc[c], errors="coerce")
            print(f"Parsed datetime column in lulc: {c}")
        except Exception as e:
            print(f"Could not parse datetime column {c}: {e}")

    # 2) Normalize coordinate columns if present
    rename_geo = {}
    if "latitude" in lulc.columns and "lat" not in lulc.columns:
        rename_geo["latitude"] = "lat"
    if "longitude" in lulc.columns and "lon" not in lulc.columns:
        rename_geo["longitude"] = "lon"
    if rename_geo:
        lulc = lulc.rename(columns=rename_geo)

    # 3) Convert likely temperature / prediction columns to numeric
    lulc_temp_candidates = [c for c in lulc.columns if any(x in c for
x in ["temp", "temperature", "pred", "prediction"])]
    for c in lulc_temp_candidates:
        lulc[c] = pd.to_numeric(lulc[c], errors="coerce")

    # 4) Drop rows missing all target-like columns, if any
    if lulc_temp_candidates:
        lulc_clean = lulc.dropna(subset=lulc_temp_candidates,
how="all").copy()
    else:
        lulc_clean = lulc.copy()

    peek(lulc_clean, "LULC (clean)")
    print("LULC (clean) shape:", lulc_clean.shape)
```

```
Raw LULC shape: (693932, 14)

=== LULC (raw) ===
Shape: (693932, 14)
```


```
{"summary":"{\n  \"name\": \"    print(\\\"LULC (clean) shape:\\\",
lulc_clean\",\n  \"rows\": 3,\n  \"fields\": [\n    {\n
\"column\": \"city\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Phoenix\",\n          \"Tucson\",\n          \"Las
Vegas\"\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"latitude\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.9979936530320446,\n        \"min\":
32.290062,\n        \"max\": 36.185505,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
33.46637,\n          32.290062,\n          36.185505\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"longitude\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.2389904010473862,\n        \"min\": -115.189333,\n        \"max\": -
110.839071,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          -112.09461,\n          -110.839071,\n          -
115.189333\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"impervious\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.1632449145210962,\n        \"min\":
0.259328,\n        \"max\": 0.575937,\n        \"num_unique_values\":
3,\n        \"samples\": [\n          0.486667,\n          0.259328,\n
0.575937\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"grass\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.00320291101968194,\n        \"min\": 0.0,\n        \"max\":
0.005917,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
0.000833,\n          0.0,\n          0.005917\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tree\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.11937208175420805,\n
\"min\": 0.005833,\n        \"max\": 0.244576,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.005833,\n          0.12585,\n          0.244576\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"building\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.22095251161806995,\n        \"min\": 0.050299,\n        \"max\":
0.479444,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
0.479444,\n          0.050299,\n          0.17357\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lst\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.3468467673195171,\n
\"min\": 48.8027,\n        \"max\": 49.4865,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          49.4865,\
n          49.2457,\n          48.8027\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tair_pred25_75\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.5867248755140708,\n        \"min\": 39.294942,\n        \"max\":
40.431758,\n        \"num_unique_values\": 3,\n        \"samples\": [\
n          39.294942,\n          40.115293,\n          40.431758\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
```

```
}\n    },\n    {\n      \"column\": \"tair_pred95\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.2584006696011716,\n        \"min\": 43.722806,\n        \"max\":
44.232565,\n        \"num_unique_values\": 3,\n        \"samples\": [\
n          43.722806,\n          43.904051,\n          44.232565\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tair_pred25_75_nt\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.605003903286947,\n        \"min\": 26.784739,\n        \"max\":
29.845436,\n        \"num_unique_values\": 3,\n        \"samples\": [\
n          29.845436,\n          26.784739,\n          29.153069\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tair_pred95_nt\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.1663025179387774,\n        \"min\": 31.111614,\n        \"max\":
33.261959,\n        \"num_unique_values\": 3,\n        \"samples\": [\
n          33.261959,\n          31.111614,\n          32.9696\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tair_diff_day\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"1970-01-01 00:00:00.000000003\",\n        \"max\": \"1970-01-01
00:00:00.000000004\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"1970-01-01 00:00:00.000000003\",\n
\"1970-01-01 00:00:00.000000004\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tair_diff_nt\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.45628904623860234,\n        \"min\": 3.416523,\n        \"max\":
4.326875,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
3.416523,\n          4.326875\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


```
city                          object
latitude                     float64
longitude                    float64
impervious                   float64
grass                        float64
tree                         float64
building                     float64
lst                          float64
tair_pred25_75               float64
tair_pred95                  float64
tair_pred25_75_nt            float64
tair_pred95_nt               float64
tair_diff_day         datetime64[ns]
tair_diff_nt                 float64
dtype: object
```

```
=== LULC (clean) ===
Shape: (693932, 14)

{"summary":"{\n  \"name\": \"    print(\\\"LULC (clean) shape:\\\",
lulc_clean\",\n  \"rows\": 3,\n  \"fields\": [\n    {\n
\"column\": \"city\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Phoenix\",\n          \"Portland\",\n          \"Las
Vegas\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"lat\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 6.2896645414038055,\n        \"min\": 33.505626,\n
\"max\": 45.484245,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          33.505626,\n          45.484245,\n
36.168653\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"lon\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 5.405840836333138,\n        \"min\": -122.536538,\n
\"max\": -111.995627,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -111.995627,\n          -122.536538,\n
-115.183718\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"impervious\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.06612315624902773,\n        \"min\":
0.244167,\n        \"max\": 0.371111,\n        \"num_unique_values\":
3,\n        \"samples\": [\n          0.244167,\n          0.371111,\n
0.339744\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"grass\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.08899305321390728,\n        \"min\": 0.0,\n        \"max\":
0.156944,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
0.156944,\n          0.0,\n          0.005769\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tree\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.2482639172594358,\n
\"min\": 0.0,\n        \"max\": 0.495513,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.220278,\n          0.0,\n          0.495513\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"building\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.03854836880249712,\n        \"min\": 0.158056,\n        \"max\":
0.225278,\n        \"num_unique_values\": 3,\n        \"samples\": [\n
0.225278,\n          0.158056,\n          0.158974\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lst\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4.169017210726445,\n
\"min\": 39.2165,\n        \"max\": 46.9875,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          46.9875,\
n          39.2165,\n          45.7194\n        ],\n
```

\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"tair_pred25_75\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
6.764260529998556,\n   \"min\": 27.650288,\n   \"max\":
39.983113,\n   \"num_unique_values\": 3,\n   \"samples\": [\
n   38.63248,\n   27.650288,\n   39.983113\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"tair_pred95\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
3.9727030069564737,\n   \"min\": 36.306914,\n   \"max\":
43.558971,\n   \"num_unique_values\": 3,\n   \"samples\": [\
n   43.558971,\n   36.306914,\n   42.744145\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"tair_pred25_75_nt\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
6.281794476333363,\n   \"min\": 17.848307,\n   \"max\":
29.285438,\n   \"num_unique_values\": 3,\n   \"samples\": [\
n   29.285438,\n   17.848307,\n   28.069855\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"tair_pred95_nt\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
6.1551334250620435,\n   \"min\": 21.705624,\n   \"max\":
33.07629,\n   \"num_unique_values\": 3,\n   \"samples\": [\n
33.07629,\n   21.705624,\n   31.476048\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"tair_diff_day\",\n
\"properties\": {\n   \"dtype\": \"date\",\n   \"min\":
\"1970-01-01 00:00:00.000000002\",\n   \"max\": \"1970-01-01
00:00:00.000000008\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   \"1970-01-01 00:00:00.000000004\",\n
\"1970-01-01 00:00:00.000000008\",\n   \"1970-01-01
00:00:00.000000002\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"tair_diff_nt\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.24354773860648626,\n   \"min\":
3.406193,\n   \"max\": 3.857317,\n   \"num_unique_values\":
3,\n   \"samples\": [\n   3.790852,\n   3.857317,\n
3.406193\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

city	object
lat	float64
lon	float64
impervious	float64
grass	float64
tree	float64
building	float64
lst	float64
tair_pred25_75	float64
tair_pred95	float64

```
tair_pred25_75_nt           float64
tair_pred95_nt              float64
tair_diff_day        datetime64[ns]
tair_diff_nt                float64
dtype: object
LULC (clean) shape: (693932, 14)
```

2.5 Clean Tree-Planting / Urban Heat-Health Dataset

```
# ============================================================
# 2.5 Clean Tree-Planting / Urban Heat-Health Dataset
# ============================================================

if "tree_df" not in globals() or tree_df is None:
    print("tree_df not loaded, skipping Section 2.5.")
else:
    print("Raw tree-planting dataset shape:", tree_df.shape)
    peek(tree_df, "Tree planting / Heat-Health (raw)")

    # Ensure standardized column names
    tree_df = clean_cols(tree_df)

    # 1) Parse any date-like columns
    tree_date_candidates = [c for c in tree_df.columns if any(x in c
for x in ["date", "time", "timestamp", "year"])]
    for c in tree_date_candidates:
        try:
            tree_df[c] = pd.to_datetime(tree_df[c], errors="coerce")
            print(f"Parsed datetime column in tree_df: {c}")
        except Exception as e:
            print(f"Could not parse datetime column {c}: {e}")

    # 2) Normalize coordinates if present
    rename_geo = {}
    if "latitude" in tree_df.columns and "lat" not in tree_df.columns:
        rename_geo["latitude"] = "lat"
    if "longitude" in tree_df.columns and "lon" not in
tree_df.columns:
        rename_geo["longitude"] = "lon"
    if rename_geo:
        tree_df = tree_df.rename(columns=rename_geo)

    # 3) Convert obvious numeric indicators to numeric
    numeric_like = [c for c in tree_df.columns if any(x in c for x in
[
        "temp", "temperature", "heat", "index", "score",
"vulnerability", "exposure"
    ])]
    for c in numeric_like:
        tree_df[c] = pd.to_numeric(tree_df[c], errors="coerce")
```

```
    # 4) Drop rows missing all numeric metrics (if any were found)
    if numeric_like:
        tree_df_clean = tree_df.dropna(subset=numeric_like,
how="all").copy()
    else:
        tree_df_clean = tree_df.copy()

    peek(tree_df_clean, "Tree planting / Heat-Health (clean)")
    print("Tree dataset (clean) shape:", tree_df_clean.shape)
```

```
Raw tree-planting dataset shape: (72246, 194)

=== Tree planting / Heat-Health (raw) ===
Shape: (72246, 194)
```

{"type":"dataframe"}

```
objectid                          int64
geoid                             int64
name                             object
state                            object
county                           object
aland                             int64
awater                            int64
b01001_001e                       int64
b01001_calc_pctge65e            float64
b01001_calc_pctge65e_box_cox    float64
b03002_calc_pctnhwhitee         float64
pct_pop_minority                float64
b01001_003e                       int64
b01001_027e                       int64
total_young_people                int64
pct_young_people                float64
b17020_calc_pctpove             float64
b25002_calc_pctvace             float64
b15002_calc_pctlthse            float64
b08201_calc_pctnovehe           float64
dtype: object
Parsed datetime column in tree_df: count_fs_risk_100_year00
Parsed datetime column in tree_df: count_fs_risk_100_year30
Parsed datetime column in tree_df: pct_fs_risk_100_year00
Parsed datetime column in tree_df: pct_fs_risk_100_year00_box_cox
Parsed datetime column in tree_df: pct_fs_risk_100_year30
Parsed datetime column in tree_df: pct_fs_risk_100_year30_box_cox

=== Tree planting / Heat-Health (clean) ===
Shape: (72246, 194)
```

{"type":"dataframe"}

```
objectid                          int64
geoid                             int64
name                             object
state                            object
county                           object
aland                             int64
awater                            int64
b01001_001e                       int64
b01001_calc_pctge65e            float64
b01001_calc_pctge65e_box_cox    float64
b03002_calc_pctnhwhitee         float64
pct_pop_minority                float64
b01001_003e                       int64
b01001_027e                       int64
total_young_people                int64
pct_young_people                float64
b17020_calc_pctpove             float64
b25002_calc_pctvace             float64
b15002_calc_pctlthse            float64
b08201_calc_pctnovehe           float64
dtype: object
Tree dataset (clean) shape: (72246, 194)
```

2.6 Inspect HLS Fmask Raster (Shade/Cloud Mask)

```
# ============================================================
# 2.6 Inspect HLS Fmask Raster (cloud/shadow mask)
# ============================================================

from pathlib import Path
import rasterio

hls_candidates = list(Path(RAW_DIR).rglob("HLS.L30*.tif"))
if not hls_candidates:
    hls_candidates = list(Path(RAW_DIR).rglob("*.tif"))

if not hls_candidates:
    print("No HLS .tif rasters found in RAW_DIR, skipping Section
2.6.")
else:
    hls_path = hls_candidates[0]
    print("HLS TIF path:", hls_path)

    with rasterio.open(hls_path) as src:
        hls_meta = src.meta.copy()
        print("Raster CRS:", src.crs)
        print("Raster size:", src.width, "x", src.height)
        print("Raster bands:", src.count)
        print("Transform:", src.transform)
```

```
        # Approx pixel size in map units (meters for UTM)
        px_width = src.transform[0]
        px_height = -src.transform[4]
        print("Pixel size:", px_width, "x", px_height)

    # keep hls_path and hls_meta for later spatial sampling
```

```
HLS TIF path: /content/drive/MyDrive/CSE475 - Final
Project/Datasets/HLS.L30.T12SVB.2024209T175727.v2.0.Fmask.tif
Raster CRS: EPSG:32612
Raster size: 3660 x 3660
Raster bands: 1
Transform: | 30.00, 0.00, 399960.00|
| 0.00,-30.00, 3700020.00|
| 0.00, 0.00, 1.00|
Pixel size: 30.0 x 30.0
```

2.7 Dataset Summary

```
# ================================================================
# 2.7 Dataset Summary
# ================================================================

print("========== DATASET SUMMARY ==========\n")

def summarize_df(name, df, date_cols=None, lat_col=None, lon_col=None,
temp_cols=None):
    print(f"--- {name} ---")
    print("Shape:", df.shape)

    # Date ranges
    if date_cols:
        for c in date_cols:
            if c in df.columns and
pd.api.types.is_datetime64_any_dtype(df[c]):
                print(f"  {c}: {df[c].min()}  →  {df[c].max()}")

    # Lat/lon ranges
    if lat_col in df.columns:
        print(f"  {lat_col} range:", df[lat_col].min(), "→",
df[lat_col].max())
    if lon_col in df.columns:
        print(f"  {lon_col} range:", df[lon_col].min(), "→",
df[lon_col].max())

    # Temperature or numeric climate variable summary
    if temp_cols:
        for t in temp_cols:
            if t in df.columns:
                print(f"  {t} stats:")
```

```
                display(df[t].describe())

    print("\n")


# --- Kaggle AZ temperatures ---
if "temps_az" in globals():
    summarize_df(
        "Kaggle AZ Temperature Subset",
        temps_az,
        date_cols=["dt"],
        lat_col="lat",
        lon_col="lon",
        temp_cols=["avg_temp_c"]
    )

# --- IET microclimate dataset ---
if "iet_small" in globals():
    summarize_df(
        "IET Microclimate Measurements",
        iet_small,
        date_cols=["date"] if "date" in iet_small.columns else None,
        lat_col="lat" if "lat" in iet_small.columns else None,
        lon_col="lon" if "lon" in iet_small.columns else None,
        temp_cols=["temperature"]
    )

# --- LC Sensor Climate ---
if "lc_climate_clean" in globals():
    summarize_df(
        "LC Sensor Climate Dataset",
        lc_climate_clean,
        date_cols=[c for c in lc_climate_clean.columns if "date" in c
or "time" in c],
        lat_col="lat" if "lat" in lc_climate_clean.columns else None,
        lon_col="lon" if "lon" in lc_climate_clean.columns else None
    )

# --- LULC Prediction ---
if "lulc_clean" in globals():
    summarize_df(
        "LULC Temperature Prediction Dataset",
        lulc_clean,
        date_cols=[c for c in lulc_clean.columns if "date" in c or
"time" in c],
        lat_col="lat" if "lat" in lulc_clean.columns else None,
        lon_col="lon" if "lon" in lulc_clean.columns else None
    )

# --- Tree planting / heat-health ---
```

```
if "tree_df_clean" in globals():
    summarize_df(
        "Tree Planting / Heat-Health Dataset",
        tree_df_clean,
        date_cols=[c for c in tree_df_clean.columns if "date" in c or
"time" in c or "year" in c],
        lat_col="lat" if "lat" in tree_df_clean.columns else None,
        lon_col="lon" if "lon" in tree_df_clean.columns else None
    )

print("========== END OF SUMMARY ==========")
```

```
========== DATASET SUMMARY ==========

--- Kaggle AZ Temperature Subset ---
Shape: (10695, 6)
  dt: 1835-01-01 00:00:00  →  2013-09-01 00:00:00
  lat range: 32.95 → 32.95
  lon range: -112.02 → -112.02
  avg_temp_c stats:

count    10695.000000
mean        21.048769
std          7.945695
min          5.768000
25%         13.651000
50%         20.811000
75%         28.943000
max         34.379000
Name: avg_temp_c, dtype: float64



--- IET Microclimate Measurements ---
Shape: (3219, 5)
  temperature stats:

count    3219.000000
mean       27.605769
std         2.698557
min        15.656250
25%        25.781250
50%        27.302125
75%        29.312500
max        38.135438
Name: temperature, dtype: float64



--- LC Sensor Climate Dataset ---
Shape: (259244, 16)
```

```
  date: 2017-06-15 00:00:00  →  2018-09-17 00:00:00
  lat range: 25.703772 → 45.59590205
  lon range: -122.7498816 → -76.531892


--- LULC Temperature Prediction Dataset ---
Shape: (693932, 14)
  lat range: 25.706351 → 45.595624
  lon range: -122.750192 → -76.53029


--- Tree Planting / Heat-Health Dataset ---
Shape: (72246, 194)
  count_fs_risk_100_year00: 1970-01-01 00:00:00  →  1970-01-01
00:00:00.000024719
  count_fs_risk_100_year30: 1970-01-01 00:00:00  →  1970-01-01
00:00:00.000025974
  pct_fs_risk_100_year00: 1970-01-01 00:00:00  →  1970-01-01
00:00:00.000000100
  pct_fs_risk_100_year00_box_cox: 1969-12-31 23:59:59.999999997  →
1970-01-01 00:00:00.000000009
  pct_fs_risk_100_year30: 1970-01-01 00:00:00  →  1970-01-01
00:00:00.000000100
  pct_fs_risk_100_year30_box_cox: 1969-12-31 23:59:59.999999997  →
1970-01-01 00:00:00.000000009


========== END OF SUMMARY ==========
```

2.8 Save Cleaned Intermediate Datasets

```python
# Save clean interim subsets

# Kaggle AZ temperatures
temps_az_path = os.path.join(INTERIM_DIR, "kaggle_temps_az.parquet")
temps_az.to_parquet(temps_az_path, index=False)

# IET cleaned subset
iet_small_path = os.path.join(INTERIM_DIR, "iet_small.parquet")
iet_small.to_parquet(iet_small_path, index=False)

print("Saved:", temps_az_path)
print("Saved:", iet_small_path)

# Optional: save additional cleaned datasets if they exist
if "lc_climate_clean" in globals():
    lc_path = os.path.join(INTERIM_DIR, "lc_climate_clean.parquet")
    lc_climate_clean.to_parquet(lc_path, index=False)
    print("Saved:", lc_path)
```

```
if "lulc_clean" in globals():
    lulc_path = os.path.join(INTERIM_DIR, "lulc_clean.parquet")
    lulc_clean.to_parquet(lulc_path, index=False)
    print("Saved:", lulc_path)

if "tree_df_clean" in globals():
    tree_path = os.path.join(INTERIM_DIR,
"tree_heathealth_clean.parquet")
    tree_df_clean.to_parquet(tree_path, index=False)
    print("Saved:", tree_path)
```

```
Saved: /content/drive/MyDrive/CSE475 - Final
Project/interim/kaggle_temps_az.parquet
Saved: /content/drive/MyDrive/CSE475 - Final
Project/interim/iet_small.parquet
Saved: /content/drive/MyDrive/CSE475 - Final
Project/interim/lc_climate_clean.parquet
Saved: /content/drive/MyDrive/CSE475 - Final
Project/interim/lulc_clean.parquet
Saved: /content/drive/MyDrive/CSE475 - Final
Project/interim/tree_heathealth_clean.parquet
```

Section 3 – Exploratory Analysis & Baseline Modeling

3.0 Exploratory Data Analysis (EDA)

3.0.1 Summary Statistics for Phoenix Temprature Dataset

```
print("Phoenix / AZ Temperature Dataset")
print("Shape:", temps_az.shape)

display(temps_az.describe(include='all'))
```

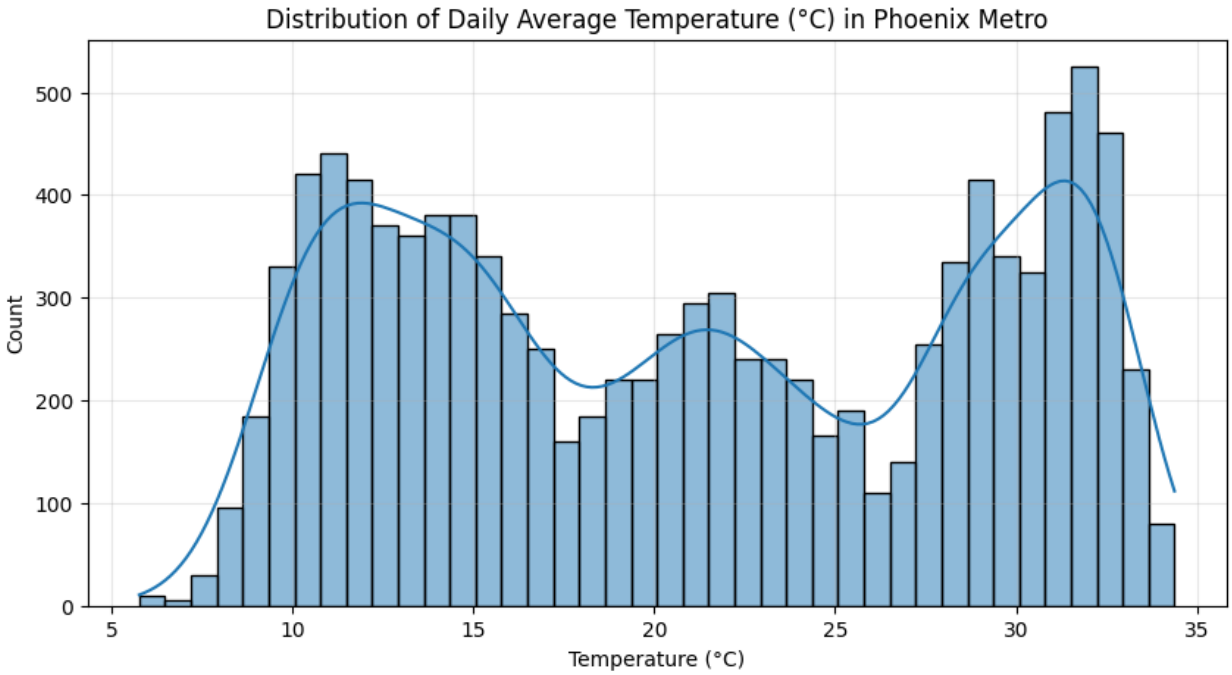
```
Phoenix / AZ Temperature Dataset
Shape: (10695, 6)
```


```
{"summary":"{\n  \"name\": \"display(temps_az\",\n  \"rows\": 11,\n
\"fields\": [\n    {\n      \"column\": \"dt\",\n      \"properties\":
{\n        \"dtype\": \"date\",\n        \"min\": \"1835-01-01
00:00:00\",\n        \"max\": \"2013-09-01 00:00:00\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n
\"10695\",\n          \"1924-07-19 16:19:31.388499200\",\n
\"1969-03-01 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"city\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
```

```
[\n          5,\n          \"2139\",\n          \"10695\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"country\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"10695\",\n          1,\n          \"United States\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lat\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3771.285780241269,\n
\"min\": 6.5444045597701134e-12,\n        \"max\": 10695.0,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
32.949999999999996,\n          6.5444045597701134e-12,\n
10695.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"lon\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3815.402065418531,\n        \"min\": -112.02,\n
\"max\": 10695.0,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          10695.0,\n          -112.02,\n
1.354357773172838e-11\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"avg_temp_c\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3774.5715596421915,\n        \"min\":
5.768000000000002,\n        \"max\": 10695.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
21.048769050958395,\n          28.943,\n          10695.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```

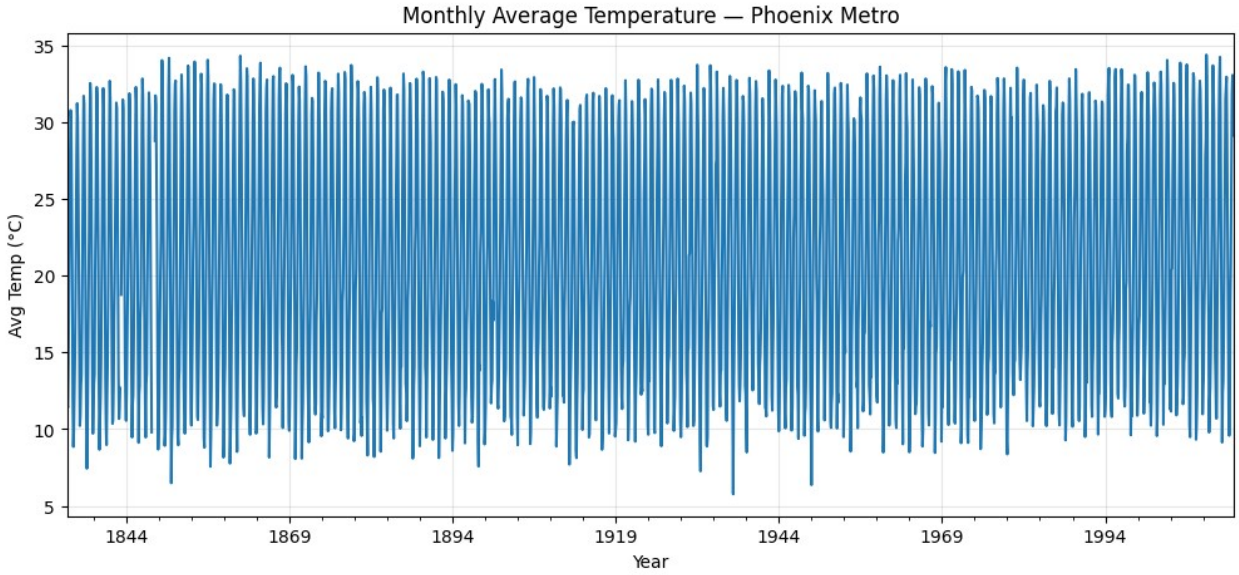

3.0.2 Distribution of Temperature (Histogram + KDE)

```
plt.figure(figsize=(10,5))
sns.histplot(temps_az["avg_temp_c"], bins=40, kde=True)
plt.title("Distribution of Daily Average Temperature (°C) in Phoenix
Metro")
plt.xlabel("Temperature (°C)")
plt.ylabel("Count")
plt.grid(alpha=0.3)
plt.show()
```



3.0.3 Monthly Mean Temperature Trends

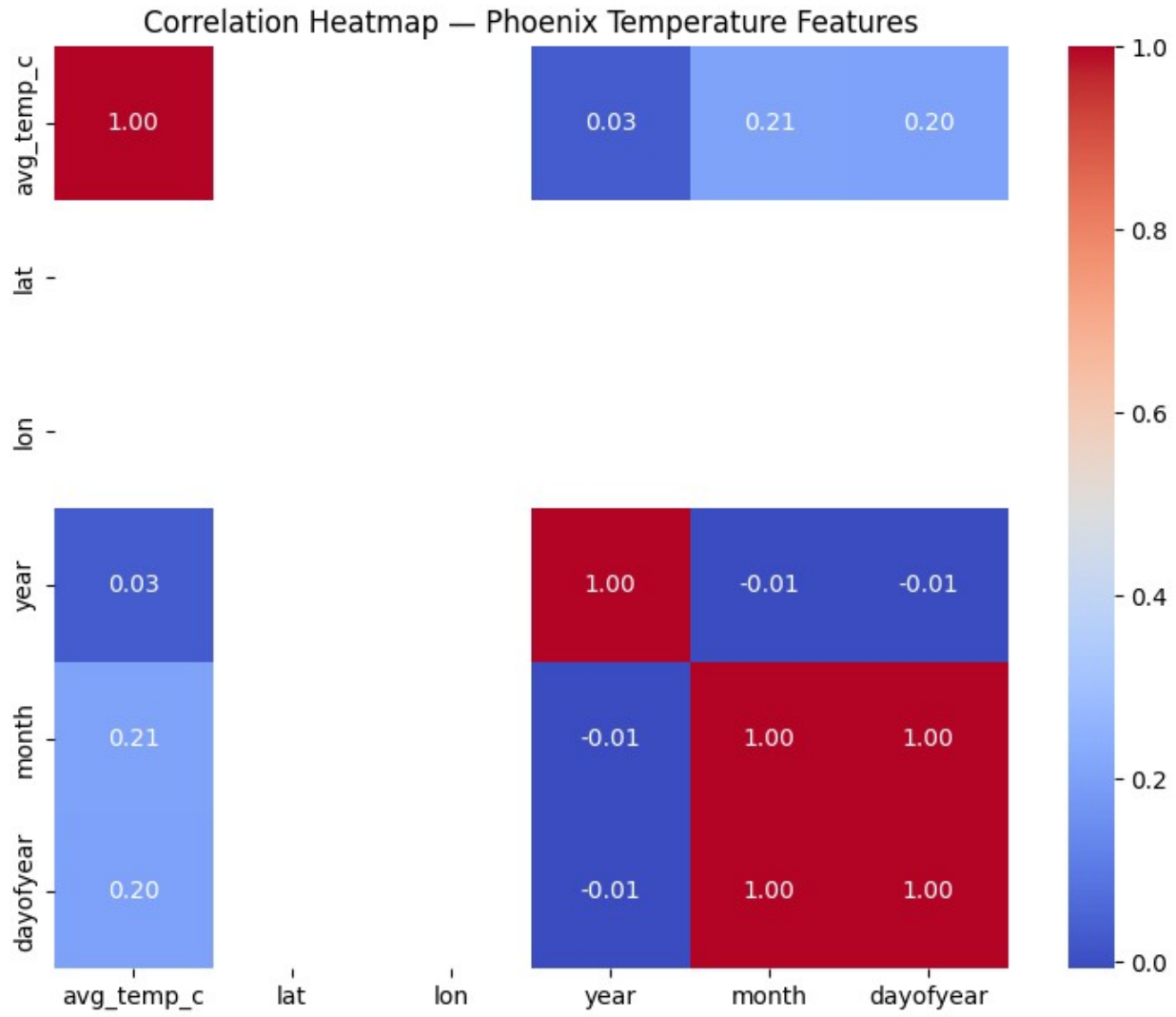
```
if "dt" in temps_az.columns:
    ts = temps_az.set_index("dt")["avg_temp_c"].resample("M").mean()

    plt.figure(figsize=(12,5))
    ts.plot()
    plt.title("Monthly Average Temperature — Phoenix Metro")
    plt.ylabel("Avg Temp (°C)")
    plt.xlabel("Year")
    plt.grid(alpha=0.3)
    plt.show()

    print("Hottest month:", ts.idxmax(), float(ts.max()))
    print("Coldest month:", ts.idxmin(), float(ts.min()))
else:
    print("No datetime column available for monthly trend plot.")
```




```
Hottest month: 2009-07-31 00:00:00 34.379
Coldest month: 1937-01-31 00:00:00 5.768000000000002
```

3.0.4 Correlation Heatmap

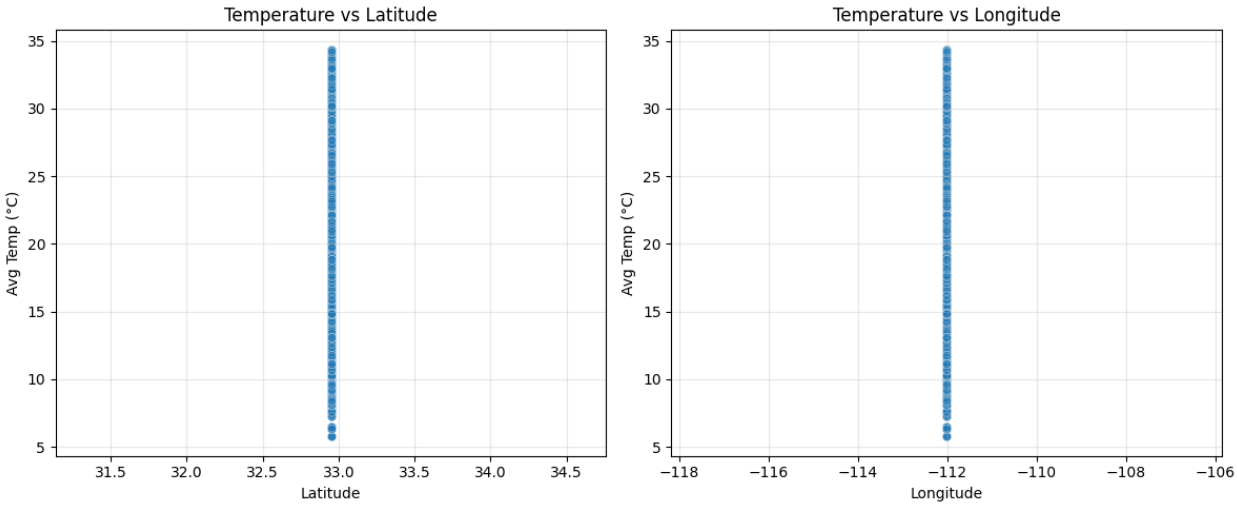
```
corr_df = temps_az.copy()

# Extract year/month/dayofyear first
if "dt" in corr_df.columns:
    corr_df["year"] = corr_df["dt"].dt.year
    corr_df["month"] = corr_df["dt"].dt.month
    corr_df["dayofyear"] = corr_df["dt"].dt.dayofyear

num_cols = ["avg_temp_c", "lat", "lon", "year", "month", "dayofyear"]
num_cols = [c for c in num_cols if c in corr_df.columns]

plt.figure(figsize=(9,7))
sns.heatmap(corr_df[num_cols].corr(), annot=True, cmap="coolwarm",
fmt=".2f")
plt.title("Correlation Heatmap — Phoenix Temperature Features")
plt.show()
```



3.0.5 Temperature vs. Latitude / Longitude

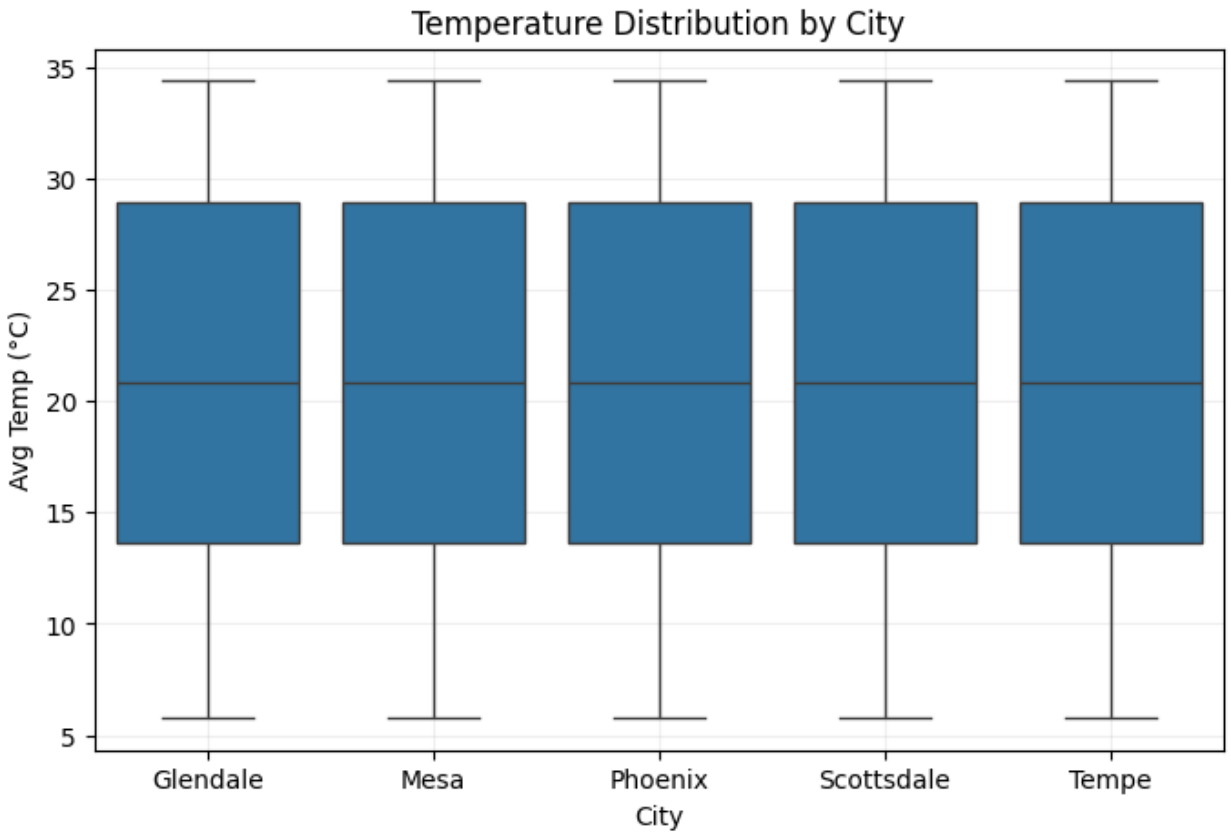
```
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
sns.scatterplot(data=temps_az, x="lat", y="avg_temp_c", alpha=0.4)
plt.title("Temperature vs Latitude")
plt.xlabel("Latitude")
plt.ylabel("Avg Temp (°C)")
plt.grid(alpha=0.3)

plt.subplot(1,2,2)
sns.scatterplot(data=temps_az, x="lon", y="avg_temp_c", alpha=0.4)
plt.title("Temperature vs Longitude")
plt.xlabel("Longitude")
plt.ylabel("Avg Temp (°C)")
plt.grid(alpha=0.3)
```

```
plt.tight_layout()
plt.show()
```



3.0.6 Temperature by City (comparison across Phoenix, Tempe, Mesa, etc.)

```
plt.figure(figsize=(8,5))
sns.boxplot(data=temps_az, x="city", y="avg_temp_c")
plt.title("Temperature Distribution by City")
plt.xlabel("City")
plt.ylabel("Avg Temp (°C)")
plt.grid(alpha=0.2)
plt.show()
```



3.1 Baseline Logistic Regression (Calendar + City + Coordinates)

```
# ============================================================
# 3.1 Baseline Logistic Regression
# Calendar + City + Coordinates (no shade features yet)
# ============================================================

from sklearn.model_selection import KFold
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

# ------------------------------------------------------------
# Prepare dataset
# ------------------------------------------------------------
base = temps_az.dropna(subset=["avg_temp_c"]).copy()

# Calendar features
if "dt" in base.columns:
    base["year"]      = base["dt"].dt.year
    base["month"]     = base["dt"].dt.month
    base["dayofyear"] = base["dt"].dt.dayofyear
```

```python
# Model input columns
num_cols = ["year", "month", "dayofyear", "lat", "lon"]
num_cols = [c for c in num_cols if c in base.columns]

cat_cols = ["city", "country"]
cat_cols = [c for c in cat_cols if c in base.columns]

X_cols = num_cols + cat_cols

print("Numeric features:", num_cols)
print("Categorical features:", cat_cols)
print("Total X columns:", len(X_cols))

# ------------------------------------------------------------
# Preprocessing Pipeline
# ------------------------------------------------------------
pre = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols),
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols)
    ],
    remainder="drop",
)

clf = Pipeline(steps=[
    ("pre", pre),
    ("lr", LogisticRegression(
        max_iter=1500,
        solver="lbfgs",
        random_state=RANDOM_STATE
    ))
])

# ------------------------------------------------------------
# Cross-validation
# ------------------------------------------------------------
# FIX: KFold handles continuous split generation better here
cv = KFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)

accs, f1s, rocs = [], [], []
fold_info = []

print("\nRunning 5-fold CV for Baseline Logistic Regression...")

# KFold splits based on index, y is optional but we pass it for
consistency
for k, (tr, te) in enumerate(cv.split(base[X_cols], base["avg_temp_c"]
)):
```

```
    # Compute the "hot" threshold on TRAIN ONLY to avoid leakage
    thr = base.iloc[tr]["avg_temp_c"].quantile(0.70)

    # Create binary target: 1 if >= 70th percentile, else 0
    y_train = (base.iloc[tr]["avg_temp_c"] >= thr).astype(int)
    y_test  = (base.iloc[te]["avg_temp_c"] >= thr).astype(int)

    X_train = base.iloc[tr][X_cols]
    X_test  = base.iloc[te][X_cols]

    # Fit
    clf.fit(X_train, y_train)

    # Predictions
    y_pred = clf.predict(X_test)
    y_prob = clf.predict_proba(X_test)[:, 1]

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    f1  = f1_score(y_test, y_pred)

    try:
        roc = roc_auc_score(y_test, y_prob)
    except ValueError:
        roc = np.nan   # Edge case: a fold with only one class

    accs.append(acc)
    f1s.append(f1)
    rocs.append(roc)

    fold_info.append({
        "fold": k + 1,
        "threshold_C": float(thr),
        "acc": acc,
        "f1": f1,
        "roc_auc": roc,
        "train_pos_rate": float(y_train.mean()),
        "test_pos_rate": float(y_test.mean()),
        "n_train": len(tr),
        "n_test": len(te),
    })

# ------------------------------------------------------------
# Summary of results
# ------------------------------------------------------------
print("\n=== Baseline Logistic Regression Results (5-fold CV) ===")
print(f"Accuracy:  {np.nanmean(accs):.3f} ± {np.nanstd(accs):.3f}")
print(f"F1 Score:  {np.nanmean(f1s):.3f} ± {np.nanstd(f1s):.3f}")
print(f"ROC-AUC:   {np.nanmean(rocs):.3f} ± {np.nanstd(rocs):.3f}")
```

```
diag = pd.DataFrame(fold_info)
display(diag)
```

```
Numeric features: ['year', 'month', 'dayofyear', 'lat', 'lon']
Categorical features: ['city', 'country']
Total X columns: 7

Running 5-fold CV for Baseline Logistic Regression...

=== Baseline Logistic Regression Results (5-fold CV) ===
Accuracy:  0.700 ± 0.005
F1 Score:  0.000 ± 0.000
ROC-AUC:   0.607 ± 0.008
```


```
{"summary":"{\n  \"name\": \"diag\",\n  \"rows\": 5,\n  \"fields\": [\
n    {\n      \"column\": \"fold\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 5,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          2,\n          5,\n          3\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"threshold_C\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.0192691463225533,\n        \"min\": 27.896,\n        \"max\":
27.941,\n        \"num_unique_values\": 4,\n        \"samples\": [\n
27.9175,\n          27.941,\n          27.896\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"acc\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.005078438752314275,\n
\"min\": 0.6951846657316503,\n        \"max\": 0.7068723702664796,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.7017297802711547,\n          0.7068723702664796,\n
0.6951846657316503\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"f1\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0,\n        \"min\": 0.0,\n        \"max\": 0.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          0.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"roc_auc\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.008961831886464624,\n        \"min\": 0.5971693325796988,\n
\"max\": 0.6175845759179092,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.5971693325796988\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"train_pos_rate\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8.264455133082766e-05,\n        \"min\": 0.30002337540906965,\n
\"max\": 0.30025712949976624,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.30014025245441794\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"test_pos_rate\",\n
```

```
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.005078438752314247,\n        \"min\": 0.29312762973352036,\n
\"max\": 0.30481533426834967,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.29827021972884527\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"n_train\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 8556,\n        \"max\": 8556,\n        \"num_unique_values\":
1,\n        \"samples\": [\n          8556\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"n_test\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 0,\n
\"min\": 2139,\n        \"max\": 2139,\n        \"num_unique_values\":
1,\n        \"samples\": [\n          2139\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"diag"}
```


3.2 Baseline Regression Models (Predict °C Directly)

```
# ============================================================
# 3.2 Baseline Regression Models
# Predict avg_temp_c using calendar + lat/lon + city
# ============================================================

from sklearn.model_selection import KFold, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.metrics import make_scorer, mean_absolute_error,
mean_squared_error, r2_score

# Prepare dataset
reg = temps_az.dropna(subset=["avg_temp_c"]).copy()

# Calendar features
if "dt" in reg.columns:
    reg["year"]      = reg["dt"].dt.year
    reg["month"]     = reg["dt"].dt.month
    reg["dayofyear"] = reg["dt"].dt.dayofyear

num_cols = ["year", "month", "dayofyear", "lat", "lon"]
num_cols = [c for c in num_cols if c in reg.columns]

cat_cols = ["city", "country"]
cat_cols = [c for c in cat_cols if c in reg.columns]

X_cols = num_cols + cat_cols
```

```
X = reg[X_cols]
y = reg["avg_temp_c"]

print("Numeric columns:", num_cols)
print("Categorical columns:", cat_cols)

# Preprocessing
pre = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols),
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols),
    ],
    remainder="drop"
)

# Models
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(
        n_estimators=200, min_samples_leaf=2,
random_state=RANDOM_STATE, n_jobs=-1),
    "Gradient Boosting": GradientBoostingRegressor(
        n_estimators=300, learning_rate=0.05,
random_state=RANDOM_STATE)
}

# Scorers
def rmse(y_true, y_pred): return mean_squared_error(y_true, y_pred,
squared=False)

rmse_scorer = make_scorer(rmse, greater_is_better=False)
mae_scorer  = make_scorer(mean_absolute_error,
greater_is_better=False)
r2_scorer   = make_scorer(r2_score)

kf = KFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)

results = []

print("\nRunning Baseline Regression Models...\n")

for name, model in models.items():
    pipe = Pipeline(steps=[("pre", pre), ("model", model)])

    rmse_scores = -cross_val_score(pipe, X, y, cv=kf,
scoring=rmse_scorer)
    mae_scores  = -cross_val_score(pipe, X, y, cv=kf,
scoring=mae_scorer)
    r2_scores   =  cross_val_score(pipe, X, y, cv=kf,
scoring=r2_scorer)
```

```
    results.append({
        "model": name,
        "rmse_mean": rmse_scores.mean(),
        "rmse_std":  rmse_scores.std(),
        "mae_mean": mae_scores.mean(),
        "mae_std":  mae_scores.std(),
        "r2_mean":  r2_scores.mean(),
        "r2_std":   r2_scores.std(),
    })

results_df = pd.DataFrame(results)
display(results_df)
```

```
Numeric columns: ['year', 'month', 'dayofyear', 'lat', 'lon']
Categorical columns: ['city', 'country']

Running Baseline Regression Models...
```


```
{"summary":"{\n  \"name\": \"results_df\",\n  \"rows\": 3,\n
\"fields\": [\n    {\n      \"column\": \"model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"Linear
Regression\",\n          \"Random Forest\",\n          \"Gradient
Boosting\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rmse_mean\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": null,\n
\"max\": null,\n        \"num_unique_values\": 0,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"rmse_std\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": null,\n
\"max\": null,\n        \"num_unique_values\": 0,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"mae_mean\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3.131633921455076,\n        \"min\":
0.3045175123913659,\n        \"max\": 5.997630414087969,\n
\"num_unique_values\": 3,\n        \"samples\": [],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"mae_std\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.03528081740183101,\n        \"min\": 0.011277427213056948,\n
\"max\": 0.07383122428310226,\n        \"num_unique_values\": 3,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"r2_mean\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.4562259529390372,\n        \"min\":
```

```
0.1977689989610925,\n        \"max\": 0.9963640804954075,\n
\"num_unique_values\": 3,\n        \"samples\": [],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"r2_std\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.004832209579061509,\n        \"min\": 0.0004724972574693202,\n
\"max\": 0.008858927193357697,\n        \"num_unique_values\": 3,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"results_df"}
```


Section 3.3 Feature Importance and Explainability

```
# ============================================================
# 3.3 Feature Importance and Explainability
# ============================================================

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
import matplotlib.pyplot as plt
import seaborn as sns

# ------------------------------------------------------------
# Prepare data
# ------------------------------------------------------------
reg = temps_az.dropna(subset=["avg_temp_c"]).copy()

if "dt" in reg.columns:
    reg["year"]      = reg["dt"].dt.year
    reg["month"]     = reg["dt"].dt.month
    reg["dayofyear"] = reg["dt"].dt.dayofyear

num_cols = ["year", "month", "dayofyear", "lat", "lon"]
num_cols = [c for c in num_cols if c in reg.columns]

cat_cols = ["city", "country"]
cat_cols = [c for c in cat_cols if c in reg.columns]

X_cols = num_cols + cat_cols
X = reg[X_cols]
y = reg["avg_temp_c"]

# Preprocessing
pre = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols),
```

```
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols)
    ],
    remainder="drop"
)

# Helper: extract feature names after preprocessing
def get_feature_names(preprocessor, num_cols, cat_cols, X):
    ohe = preprocessor.named_transformers_["cat"]
    cat_new_cols = list(ohe.get_feature_names_out(cat_cols))
    return num_cols + cat_new_cols


# ============================================================
# 3.3.1 Logistic Regression Coefficients
# ============================================================

log_reg = Pipeline([
    ("pre", pre),
    ("model", LogisticRegression(max_iter=1500,
random_state=RANDOM_STATE))
])

log_reg.fit(X, (y >= y.quantile(0.70)).astype(int))

# Extract feature names
feature_names = get_feature_names(log_reg.named_steps["pre"],
num_cols, cat_cols, X)
coeffs = log_reg.named_steps["model"].coef_.flatten()

lr_importance = pd.DataFrame({
    "feature": feature_names,
    "importance": coeffs
}).sort_values("importance", ascending=False)

print("=== Top Logistic Regression Coefficients ===")
display(lr_importance.head(15))


# ============================================================
# 3.3.2 Random Forest Feature Importance
# ============================================================

rf = Pipeline([
    ("pre", pre),
    ("model", RandomForestRegressor(
        n_estimators=300,
        min_samples_leaf=2,
        random_state=RANDOM_STATE,
        n_jobs=-1
    ))
```

```
])

rf.fit(X, y)

rf_importance = pd.DataFrame({
    "feature": feature_names,
    "importance": rf.named_steps["model"].feature_importances_
}).sort_values("importance", ascending=False)

print("=== Top Random Forest Features ===")
display(rf_importance.head(15))


# ============================================================
# 3.3.3 Gradient Boosting Feature Importance
# ============================================================

gb = Pipeline([
    ("pre", pre),
    ("model", GradientBoostingRegressor(
        n_estimators=300,
        learning_rate=0.05,
        random_state=RANDOM_STATE
    ))
])

gb.fit(X, y)

gb_importance = pd.DataFrame({
    "feature": feature_names,
    "importance": gb.named_steps["model"].feature_importances_
}).sort_values("importance", ascending=False)

print("=== Top Gradient Boosting Features ===")
display(gb_importance.head(15))


# ============================================================
# 3.3.4 Combined Importance Summary
# ============================================================

combined = (rf_importance
            .merge(gb_importance, on="feature", how="outer",
suffixes=("_rf","_gb"))
            .merge(lr_importance, on="feature", how="outer"))

combined = combined.rename(columns={"importance": "importance_lr"})
combined = combined.fillna(0).sort_values("importance_rf",
ascending=False)

print("=== Combined Feature Importance Table ===")
```

```
display(combined.head(20))


# ============================================================
# 3.3.5 Importance Plot (Random Forest)
# ============================================================

plt.figure(figsize=(10,6))
sns.barplot(data=rf_importance.head(15), x="importance", y="feature")
plt.title("Top 15 Feature Importances (Random Forest)")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# ============================================================
# 3.3.6 Importance Plot (Gradient Boosting)
# ============================================================

plt.figure(figsize=(10,6))
sns.barplot(data=gb_importance.head(15), x="importance", y="feature")
plt.title("Top 15 Feature Importances (Gradient Boosting)")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

```
=== Top Logistic Regression Coefficients ===
```


```
{"summary":"{\n  \"name\": \"plt\",\n  \"rows\": 11,\n  \"fields\": [\n    {\n      \"column\": \"feature\",\n      \"properties\": {\n\"dtype\": \"string\",\n        \"num_unique_values\": 11,\n\"samples\": [\n          \"city_Scottsdale\",\n          \"month\",\n\"country_United States\"\n        ],\n        \"semantic_type\":\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n\"column\": \"importance\",\n      \"properties\": {\n\"dtype\": \"number\",\n        \"std\": 1.3412715753192037,\n\"min\": -2.7965493191727893,\n        \"max\": 3.1790459397428528,\n\"num_unique_values\": 10,\n        \"samples\": [\n          -0.22432059824465167,\n          0.052034869123361596,\n          -0.04486411964893895\n        ],\n        \"semantic_type\": \"\",\n\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
=== Top Random Forest Features ===
```


```
{"summary":"{\n  \"name\": \"plt\",\n  \"rows\": 11,\n  \"fields\": [\n    {\n      \"column\": \"feature\",\n      \"properties\": {\n\"dtype\": \"string\",\n        \"num_unique_values\": 11,\n\"samples\": [\n          \"city_Glendale\",\n
```

```
\"dayofyear\",\n          \"lat\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"importance\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.19651698296455475,\n        \"min\": 0.0,\n        \"max\":
0.5081304975868915,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          3.890285176253761e-05,\n
0.4672309778563133,\n          4.175304713573798e-05\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

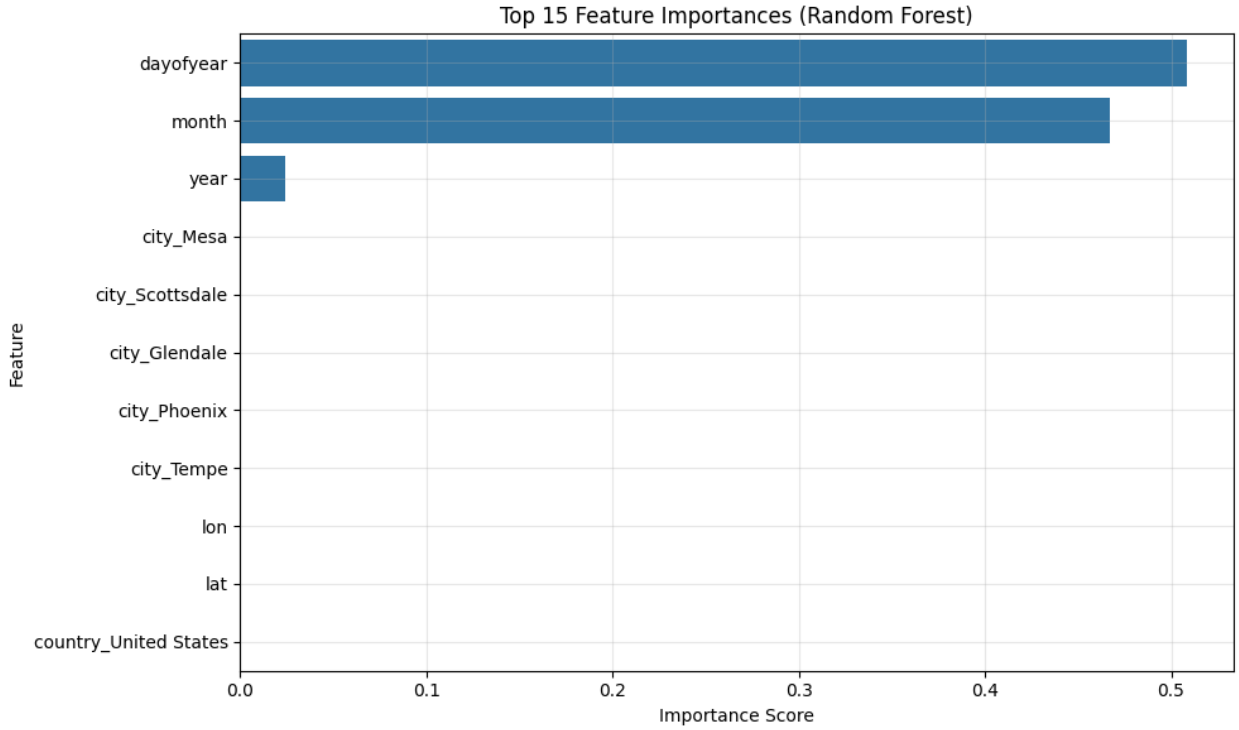
=== Top Gradient Boosting Features ===

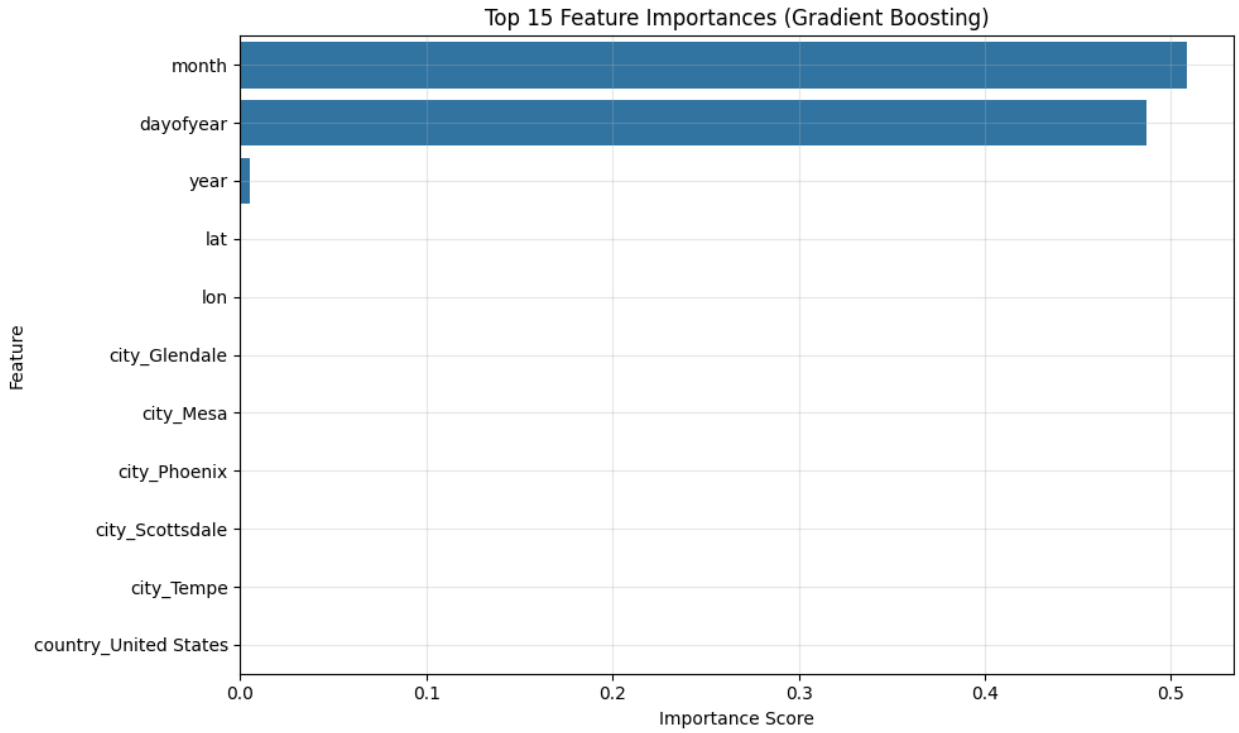

```
{"summary":"{\n  \"name\": \"plt\",\n  \"rows\": 11,\n  \"fields\": [\
n    {\n      \"column\": \"feature\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 11,\n
\"samples\": [\n          \"city_Glendale\",\n          \"month\",\n
\"city_Tempe\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"importance\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.20105191076502651,\n        \"min\":
0.0,\n        \"max\": 0.5082558961653219,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
0.48659763356257396,\n          0.0,\n          0.5082558961653219\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```

=== Combined Feature Importance Table ===


```
{"summary":"{\n  \"name\": \"plt\",\n  \"rows\": 11,\n  \"fields\": [\
n    {\n      \"column\": \"feature\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 11,\n
\"samples\": [\n          \"city_Glendale\",\n
\"dayofyear\",\n          \"lon\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"importance_rf\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.19651698296455475,\n        \"min\": 0.0,\n        \"max\":
0.5081304975868915,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          3.890285176253761e-05,\n
0.4672309778563133,\n          4.175304713573798e-05\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"importance_gb\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.20105191076502651,\n        \"min\": 0.0,\n        \"max\":
0.5082558961653219,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          0.5082558961653219,\n          0.0,\n
0.48659763356257396\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"importance_lr\",\n      \"properties\": {\n        \"dtype\":
```

```
\"number\",\n        \"std\": 1.3412715753192033,\n        \"min\": -
2.7965493191727893,\n        \"max\": 3.1790459397428528,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          0.0,\n
3.1790459397428528,\n          -0.044864119648939604\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```



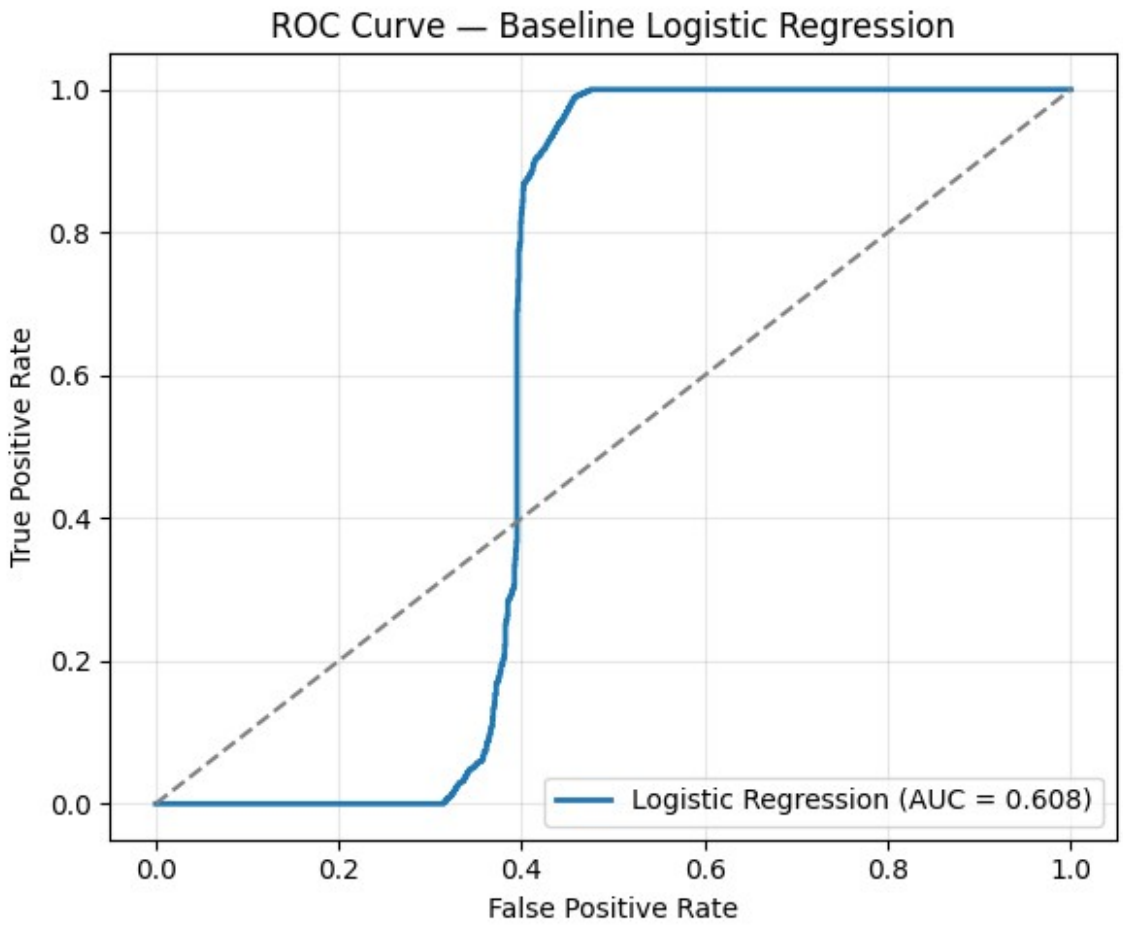
3.4 Diagnostics and Error Analysis

```
# ============================================================
# 3.4 Diagnostics and Error Analysis
# ============================================================

from sklearn.metrics import roc_curve, auc, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# ------------------------------------------------------------
# Helpers
# ------------------------------------------------------------
def plot_confusion(cm, labels):
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=labels, yticklabels=labels)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.show()


# ============================================================
# 3.4.1 ROC Curve for Logistic Regression (from Section 3.1)
# ============================================================
```

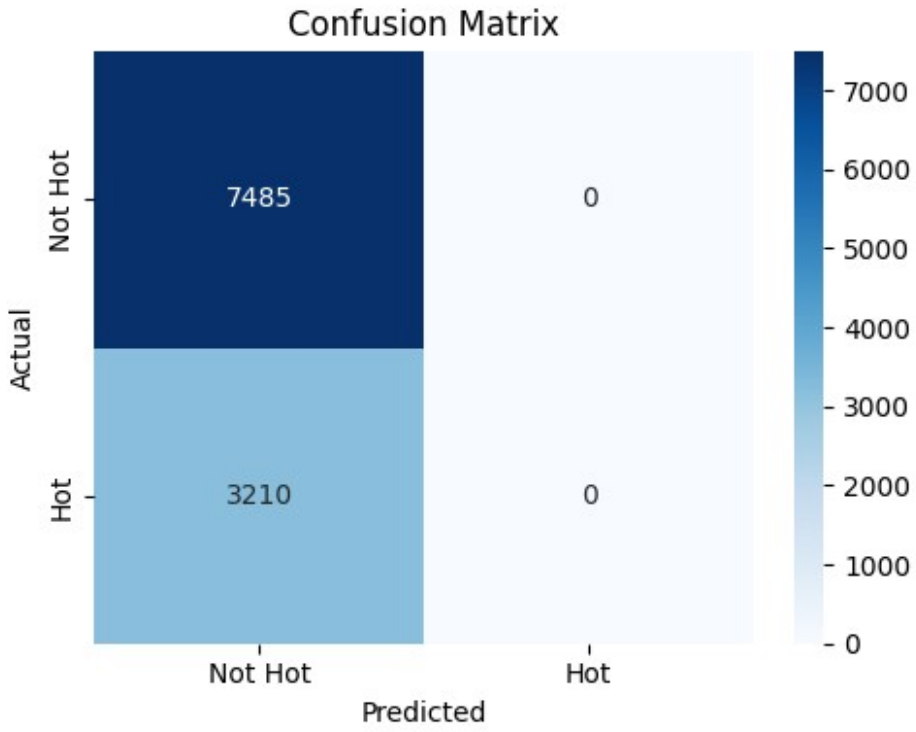
```python
# Refit the baseline model on full dataset for inspection
base = temps_az.dropna(subset=["avg_temp_c"]).copy()

if "dt" in base.columns:
    base["year"]      = base["dt"].dt.year
    base["month"]     = base["dt"].dt.month
    base["dayofyear"] = base["dt"].dt.dayofyear

num_cols = ["year", "month", "dayofyear", "lat", "lon"]
num_cols = [c for c in num_cols if c in base.columns]

cat_cols = ["city", "country"]
cat_cols = [c for c in cat_cols if c in base.columns]

X_cols = num_cols + cat_cols
X = base[X_cols]

# Logistic Regression threshold: use overall 70th percentile
thr = base["avg_temp_c"].quantile(0.70)
y = (base["avg_temp_c"] >= thr).astype(int)

# Fit
clf.fit(X, y)

# Predict probabilities
y_prob_full = clf.predict_proba(X)[:, 1]

# ROC
fpr, tpr, _ = roc_curve(y, y_prob_full)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"Logistic Regression (AUC = {roc_auc:.3f})",
linewidth=2)
plt.plot([0, 1], [0, 1], "--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve – Baseline Logistic Regression")
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

print("AUC:", roc_auc)


# ============================================================
# 3.4.2 Confusion Matrix (Logistic Regression)
# ============================================================
```

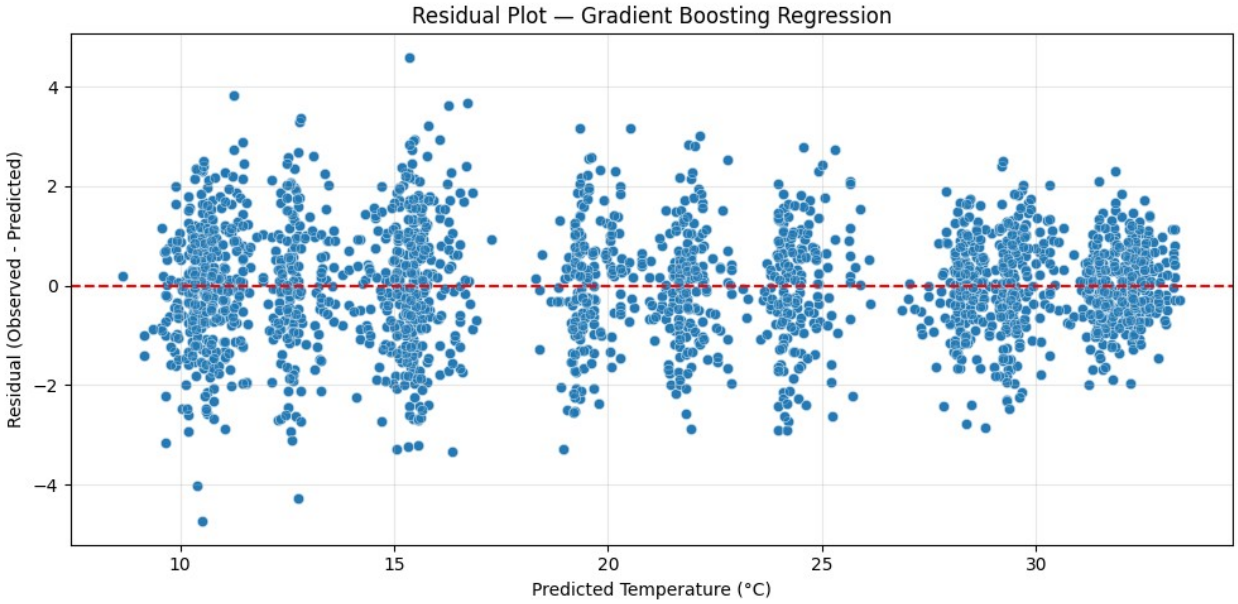
```
y_pred_full = clf.predict(X)
cm = confusion_matrix(y, y_pred_full)

plot_confusion(cm, labels=["Not Hot", "Hot"])


# ============================================================
# 3.4.3 Regression Diagnostics
# Fit Gradient Boosting Regression (best baseline regressor from 3.2)
# ============================================================

gb = Pipeline(steps=[
    ("pre", pre),
    ("model", GradientBoostingRegressor(
        n_estimators=300, learning_rate=0.05,
random_state=RANDOM_STATE
    ))
])

gb.fit(X, base["avg_temp_c"])
preds = gb.predict(X)

# ---------------- Residual Plot ----------------
residuals = base["avg_temp_c"] - preds

plt.figure(figsize=(10,5))
sns.scatterplot(x=preds, y=residuals, alpha=0.5)
plt.axhline(0, color="red", linestyle="--")
plt.xlabel("Predicted Temperature (°C)")
plt.ylabel("Residual (Observed - Predicted)")
plt.title("Residual Plot — Gradient Boosting Regression")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()


# ---------------- Predicted vs Actual ----------------
plt.figure(figsize=(7,7))
sns.scatterplot(x=base["avg_temp_c"], y=preds, alpha=0.4)
plt.xlabel("Actual Temperature (°C)")
plt.ylabel("Predicted Temperature (°C)")
plt.title("Predicted vs Actual Temperature — Gradient Boosting")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()


# ============================================================
# 3.4.4 Error Distribution Histogram
# ============================================================
```

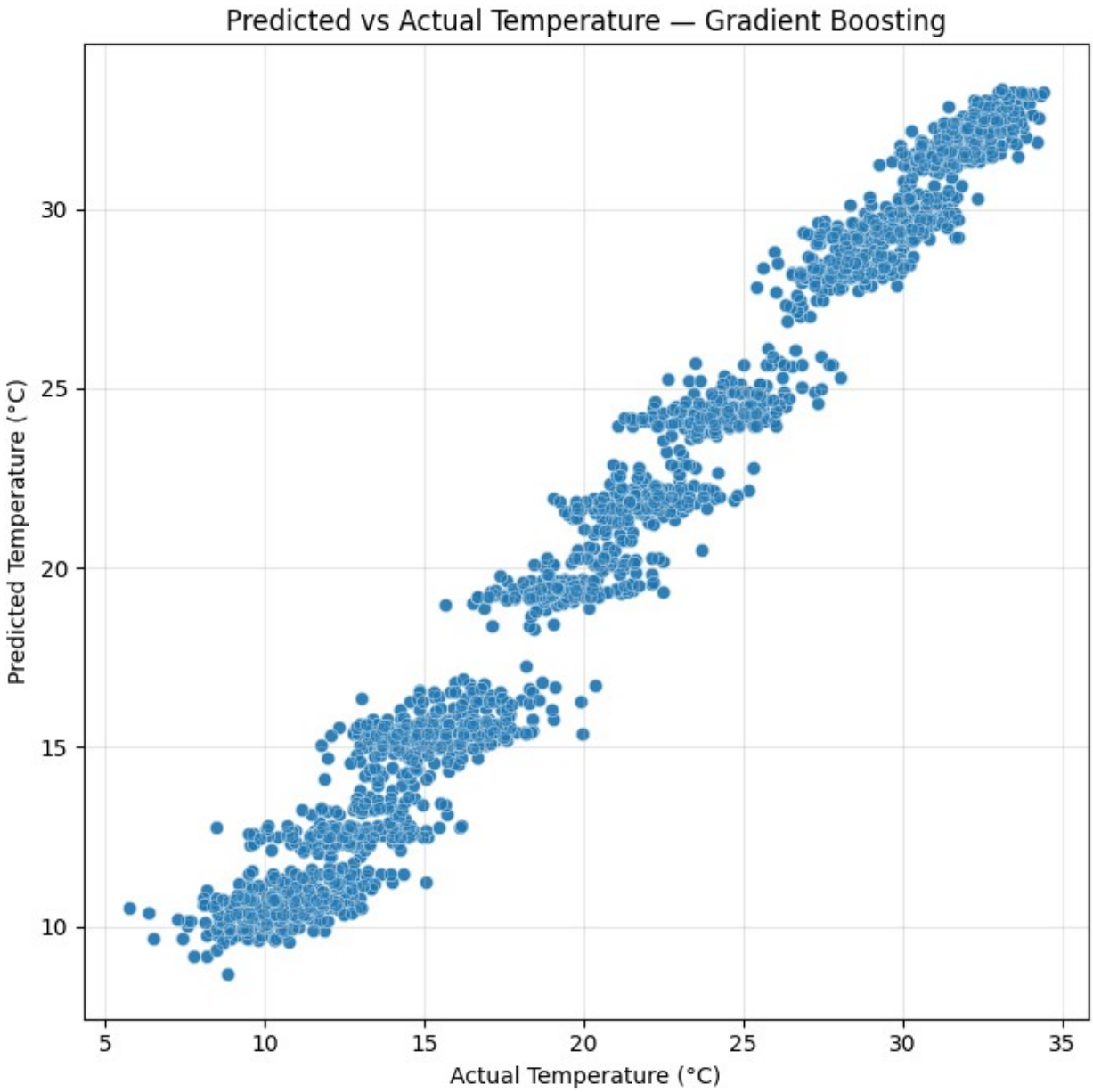
```
plt.figure(figsize=(8,5))
sns.histplot(residuals, bins=40, kde=True)
plt.title("Distribution of Residuals – Gradient Boosting")
plt.xlabel("Residual (Observed - Predicted)")
plt.ylabel("Count")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

print("Residual Mean:", residuals.mean())
print("Residual Std:", residuals.std())
```

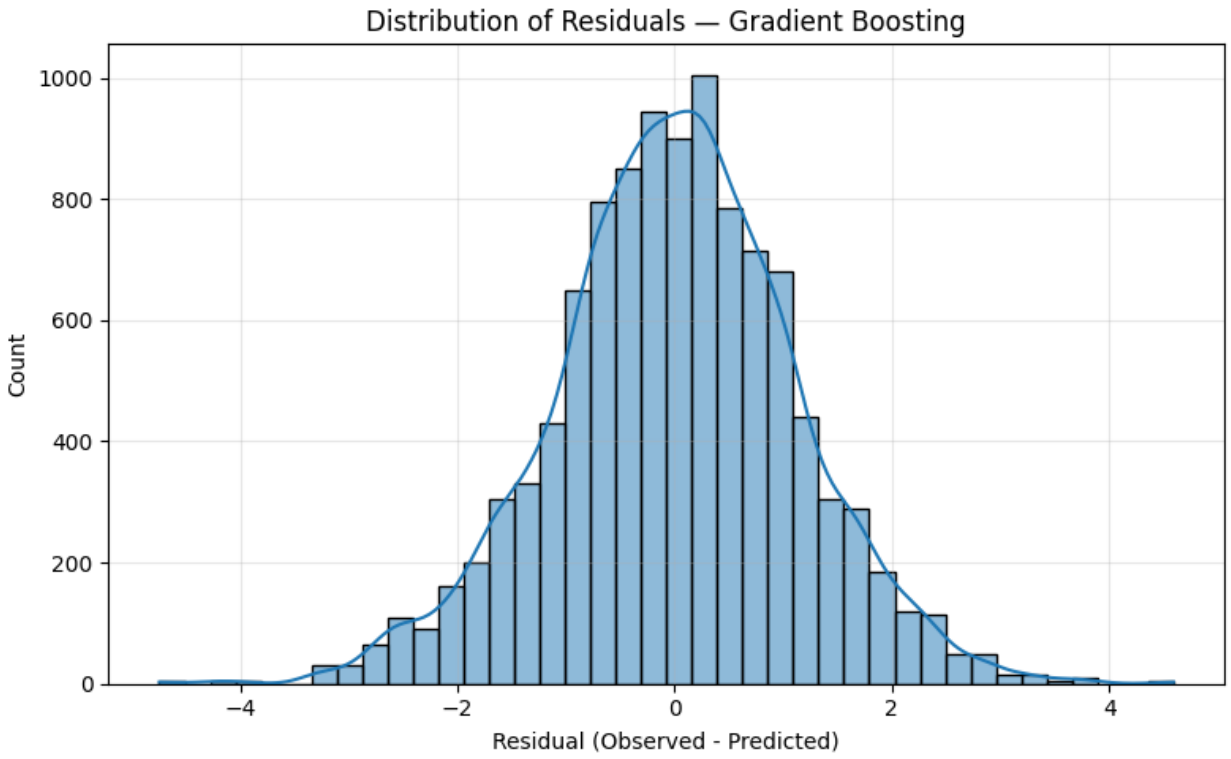



```
AUC: 0.6081238281339418
```


Confusion Matrix
Not Hot
Hot
Actual
7485
0
3210
0
Not Hot
Hot
Predicted
7000
6000
5000
4000
3000
2000
1000
0


Residual Plot — Gradient Boosting Regression
Residual (Observed - Predicted)
4
2
0
−2
−4
10
15
20
25
30
Predicted Temperature (°C)


Predicted vs Actual Temperature — Gradient Boosting
Predicted Temperature (°C)
Actual Temperature (°C)
10
15
20
25
30
5
10
15
20
25
30
35




```
Residual Mean: 1.0090105469244059e-16
Residual Std: 1.1216008191358817
```

Section 4 -Feature Engineering & Master Table

In this section, I:

- Define the analysis unit
- Add IET microclimate information (dynamic features)
- Attach static land-cover and NDVI / terrain indicators (LC sensors)
- Attach static LULC-based temperature prediction features
- Attach tree / heat-health priority metrics
- Define regression and classification targets

4.0 Load Cleaned Intermediate Datasets

```
iet_path   = os.path.join(INTERIM_DIR, "iet_small.parquet")
temps_path = os.path.join(INTERIM_DIR, "kaggle_temps_az.parquet")
lc_path    = os.path.join(INTERIM_DIR, "lc_climate_clean.parquet")
lulc_path  = os.path.join(INTERIM_DIR, "lulc_clean.parquet")
tree_path  = os.path.join(INTERIM_DIR,
"tree_heathealth_clean.parquet")

iet_small         = pd.read_parquet(iet_path)
```

```
temps_az         = pd.read_parquet(temps_path)
lc_climate_clean = pd.read_parquet(lc_path)
lulc_clean       = pd.read_parquet(lulc_path)
tree_df_clean    = pd.read_parquet(tree_path)

print("Loaded datasets:")
for name in ["iet_small", "temps_az", "lc_climate_clean",
"lulc_clean", "tree_df_clean"]:
    print(name)
```

```
Loaded datasets:
iet_small
temps_az
lc_climate_clean
lulc_clean
tree_df_clean
```

4.1 Build City-Day Base Table (from Kaggle temps)

```
city_base = temps_az.copy()

if not np.issubdtype(city_base["dt"].dtype, np.datetime64):
    city_base["dt"] = pd.to_datetime(city_base["dt"])

city_base["year"] = city_base["dt"].dt.year
city_base["month"] = city_base["dt"].dt.month
city_base["day"] = city_base["dt"].dt.day
city_base["dayofyear"] = city_base["dt"].dt.dayofyear
city_base["dow"] = city_base["dt"].dt.dayofweek

group_cols = ["city", "country", "dt", "year", "month", "day",
"dayofyear", "dow"]

cityday_base = (
    city_base
    .groupby(group_cols, as_index=False)
    .agg({"avg_temp_c": "mean", "lat": "mean", "lon": "mean"})
)

peek(cityday_base, "City-day base")
```

```
=== City-day base ===
Shape: (10695, 11)
```


```
{"summary":"{\n  \"name\": \"peek(cityday_base, \\\"City-day
base\\\")\",\n  \"rows\": 3,\n  \"fields\": [\n    {\n
\"column\": \"city\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Tempe\",\n          \"Mesa\",\n
\"Scottsdale\"\n        ],\n        \"semantic_type\": \"\",\n
```

\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"country\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 1,\n   \"samples\":
[\n   \"United States\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"dt\",\n   \"properties\": {\n   \"dtype\":
\"date\",\n   \"min\": \"1908-07-01 00:00:00\",\n   \"max\":
\"1967-05-01 00:00:00\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   \"1945-05-01 00:00:00\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"year\",\n   \"properties\": {\n
\"dtype\": \"int32\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   1945\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"month\",\n   \"properties\": {\n   \"dtype\":
\"int32\",\n   \"num_unique_values\": 2,\n   \"samples\": [\
n   7\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"day\",\n   \"properties\": {\n   \"dtype\": \"int32\",\n
\"num_unique_values\": 1,\n   \"samples\": [\n   1\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"dayofyear\",\n
\"properties\": {\n   \"dtype\": \"int32\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n   183\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"dow\",\n   \"properties\": {\
n   \"dtype\": \"int32\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   1\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"avg_temp_c\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 4.5445274048390685,\n
\"min\": 23.607000000000006,\n   \"max\": 31.701,\n
\"num_unique_values\": 3,\n   \"samples\": [\n   24.073\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"lat\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 0.0,\n
\"min\": 32.95,\n   \"max\": 32.95,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   32.95\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"lon\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 0.0,\n
\"min\": -112.02,\n   \"max\": -112.02,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   -112.02\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

```
city                  object
country               object
dt            datetime64[ns]
year                   int32
```

```
month                    int32
day                      int32
dayofyear                int32
dow                      int32
avg_temp_c             float64
lat                    float64
lon                    float64
dtype: object
```

4.2 Add IET Microclimate Features (weekday × time-of-day)

```
if "iet_small" not in globals():
    print("IET dataset not loaded; skip.")
    cityday_iet = None
else:
    df_i = iet_small.copy()

    cityday_iet = (
        df_i.groupby(["dow", "tod"])["temperature"]
            .mean()
            .unstack("tod")
            .rename(columns=lambda c: f"iet_temp_{c}")
            .reset_index()
    )

    cityday_env = cityday_base.merge(cityday_iet, on="dow",
how="left")
    peek(cityday_iet, "IET feature table")


=== IET feature table ===
Shape: (7, 5)

{"summary":"{\n  \"name\": \"    peek(cityday_iet, \\\"IET feature
table\\\")\",\n  \"rows\": 3,\n  \"fields\": [\n    {\n
\"column\": \"dow\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 4,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          1,\n          3,\n          4\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"iet_temp_afternoon\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.6352910583558552,\n        \"min\": 27.963516973248407,\n
\"max\": 29.136724693267972,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          27.963516973248407,\n
28.972580204968153,\n          29.136724693267972\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"iet_temp_evening\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.1864975271296172,\n        \"min\": 27.232586996233767,\n
```

```
\"max\": 27.57050395294872,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          27.232586996233767,\n
27.57050395294872,\n          27.538304681410256\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"iet_temp_morning\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.3187558271686246,\n        \"min\": 27.584561708860758,\n
\"max\": 28.202955238266664,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          27.75956490384615,\n
27.584561708860758,\n          28.202955238266664\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"iet_temp_night\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.22294725352404238,\n        \"min\": 26.268521097025314,\n
\"max\": 26.679861931410258,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          26.623240070189873,\n
26.268521097025314,\n          26.679861931410258\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

tod
dow                    int64
iet_temp_afternoon   float64
iet_temp_evening     float64
iet_temp_morning     float64
iet_temp_night       float64
dtype: object
```

4.3 Build LC + NDVI + Landcover Static Features

```
cityday_env = cityday_base.merge(cityday_iet, on="dow", how="left")
cityday_env["_key"] = 1

def summarize_static(df, prefix):
    num_cols = df.select_dtypes(include=['float',
'int']).columns.tolist()

    if len(num_cols) == 0:
        print(f"[Warning] No numeric columns found in {prefix}")
        return pd.DataFrame({_key: [1]})

    summary = df[num_cols].mean().to_frame().T

    summary = summary.add_prefix(f"{prefix}_")

    summary["_key"] = 1
    return summary

lc_static   = summarize_static(lc_climate_clean, "lc")
lulc_static = summarize_static(lulc_clean,        "lulc")
```

```
#merge
cityday_env = (
    cityday_env
    .merge(lc_static,   on="_key", how="left")
    .merge(lulc_static, on="_key", how="left")
    .drop(columns=["_key"])
)

print("cityday_env:", cityday_env.shape)
peek(cityday_env, "Added LC + NDVI + Landcover")
```

```
cityday_env: (10695, 38)

=== Added LC + NDVI + Landcover ===
Shape: (10695, 38)
```

```
{"type":"dataframe"}
```

```
city                              object
country                           object
dt                        datetime64[ns]
year                               int32
month                              int32
day                                int32
dayofyear                          int32
dow                                int32
avg_temp_c                       float64
lat                              float64
lon                              float64
iet_temp_afternoon               float64
iet_temp_evening                 float64
iet_temp_morning                 float64
iet_temp_night                   float64
lc_lat                           float64
lc_lon                           float64
lc_impervious_surface            float64
lc_turf/grass                    float64
lc_tree_canopy                   float64
dtype: object
```

4.4 Build Tree Health Static Features

```
cityday_tree = cityday_env.copy()
cityday_tree["_key"] = 1

tree_static = summarize_static(tree_df_clean, "tree")

cityday_tree = (
    cityday_tree
```

```
    .merge(tree_static, on="_key", how="left")
    .drop(columns=["_key"])
)

print("cityday_tree:", cityday_tree.shape)
peek(cityday_tree, "Added Trees")
```

```
cityday_tree: (10695, 223)

=== Added Trees ===
Shape: (10695, 223)

{"type":"dataframe"}

city                              object
country                           object
dt                        datetime64[ns]
year                               int32
month                              int32
day                                int32
dayofyear                          int32
dow                                int32
avg_temp_c                       float64
lat                              float64
lon                              float64
iet_temp_afternoon               float64
iet_temp_evening                 float64
iet_temp_morning                 float64
iet_temp_night                   float64
lc_lat                           float64
lc_lon                           float64
lc_impervious_surface            float64
lc_turf/grass                    float64
lc_tree_canopy                   float64
dtype: object
```

4.5 Build Regression / Classification Targets

```
cityday_targets = cityday_base.copy()

# regression target
cityday_targets["target_next_temp"] = (
    cityday_targets.groupby("city")["avg_temp_c"].shift(-1)
)

# classification target
thr = cityday_targets["avg_temp_c"].quantile(0.70)
cityday_targets["is_extreme"] = (cityday_targets["avg_temp_c"] >=
thr).astype(int)
```

```
# keep only target columns
cityday_targets = cityday_targets[["city", "dt", "target_next_temp",
"is_extreme"]]

peek(cityday_targets, "Targets")
```

```
=== Targets ===
Shape: (10695, 4)
```


```
{"summary":"{\n  \"name\": \"peek(cityday_targets,
\\\"Targets\\\")\",\n  \"rows\": 3,\n  \"fields\": [\n    {\n
\"column\": \"city\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"Scottsdale\",\n          \"Tempe\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"dt\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"1924-07-01 00:00:00\",\n
\"max\": \"2004-02-01 00:00:00\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"1924-07-01 00:00:00\",\n          \"1979-
09-01 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"target_next_temp\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 6.149137256558844,\n        \"min\":
20.363,\n        \"max\": 32.06,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          32.06,\n          22.922\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"is_extreme\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
city                          object
dt                    datetime64[ns]
target_next_temp             float64
is_extreme                     int64
dtype: object
```

4.6 Final merge

```
master = cityday_tree.copy()

# merge target columns
master = master.merge(
    cityday_targets,
    on=["city", "dt"],
    how="left"
)
```

```
peek(master, "MASTER TABLE (with targets)")
print("Final master shape:", master.shape)

# save
master_path = os.path.join(INTERIM_DIR, "cityday_master.parquet")
master.to_parquet(master_path, index=False)
print("Saved to:", master_path)
```

```
=== MASTER TABLE (with targets) ===
Shape: (10695, 225)

{"type":"dataframe"}

city                                object
country                             object
dt                          datetime64[ns]
year                                 int32
month                                int32
day                                  int32
dayofyear                            int32
dow                                  int32
avg_temp_c                         float64
lat                                float64
lon                                float64
iet_temp_afternoon                 float64
iet_temp_evening                   float64
iet_temp_morning                   float64
iet_temp_night                     float64
lc_lat                             float64
lc_lon                             float64
lc_impervious_surface              float64
lc_turf/grass                      float64
lc_tree_canopy                     float64
dtype: object
Final master shape: (10695, 225)
Saved to: /content/drive/MyDrive/CSE475 - Final
Project/interim/cityday_master.parquet
```

```
print("\n================ MASTER COLUMNS ================\n")
for c in master.columns:
    print(c)
print("\n================================================\n")
print("Total columns:", len(master.columns))
```

```
================ MASTER COLUMNS ================

city
country
```

```
dt
year
month
day
dayofyear
dow
avg_temp_c
lat
lon
iet_temp_afternoon
iet_temp_evening
iet_temp_morning
iet_temp_night
lc_lat
lc_lon
lc_impervious_surface
lc_turf/grass
lc_tree_canopy
lc_buildings
lc_regional_air_temperature
lc_regional_relative_humidity
lc_regional_wind_speed
lc_mean_landsat_land_surface_temperature
lc_sensor_recorded_air_temperature
lulc_lat
lulc_lon
lulc_impervious
lulc_grass
lulc_tree
lulc_building
lulc_lst
lulc_tair_pred25_75
lulc_tair_pred95
lulc_tair_pred25_75_nt
lulc_tair_pred95_nt
lulc_tair_diff_nt
tree_objectid
tree_geoid
tree_aland
tree_awater
tree_b01001_001e
tree_b01001_calc_pctge65e
tree_b01001_calc_pctge65e_box_cox
tree_b03002_calc_pctnhwhitee
tree_pct_pop_minority
tree_b01001_003e
tree_b01001_027e
tree_total_young_people
tree_pct_young_people
```

```
tree_b17020_calc_pctpove
tree_b25002_calc_pctvace
tree_b15002_calc_pctlthse
tree_b08201_calc_pctnovehe
tree_b08201_calc_pctnovehe_box_cox
tree_b28002_calc_pctnointe
tree_b18101_calc_pctde
tree_pct_hh_lives_alone
tree_b16004_calc_pctge18leae
tree_b01001_calc_pctdepende
tree_pct_hu_built_prior_1970
tree_b25002_001e
tree_b25002_calc_pcttotalowne
tree_b25002_calc_pcttotalrente
tree_mean_annual_est_pm2_5_µg_m3
tree_casthma_crudeprev
tree_area_sqkm
tree_pop_density_ppl_sqkm
tree_pop_density_ppl_sqkm_box_cox
tree_high_summer_mean_lst_f
tree_pct_treecanopy
tree_pct_treecanopy_box_cox
tree_pct_lackingcanopy
tree_pct_imperviousurfaces
tree_wf_housingdensity_mean
tree_wf_housingdensity_mean_box_cox
tree_wf_exp_type_mean
tree_wf_risktohome_mean
tree_wf_risktohome_mean_box_cox
tree_wf_hazardpotential_mean
tree_wf_hazardpotential_mean_box_cox
tree_cnt_rd_inter
tree_cnt_rd_inter_per_sqkm
tree_cnt_rd_inter_per_sqkm_box_cox
tree_count_property
tree_pct_tract_blw_sl_2050
tree_avg_vul_2050
tree_max_cc
tree_pct_riparian
tree_pct_riparian_square_root
tree_pct_area_protect
tree_pct_area_protect_box_cox
tree_pct_area_restore
tree_pct_area_restore_box_cox
tree_pct_tract_undev
tree_single_tract
tree_coastal_tract
tree_mangrove_length
tree_spartina_length
```

```
tree_man_index
tree_man_index_nat_rank
tree_man_index_nat_pctl
tree_man_index_nat_quan
tree_man_index_st_rank
tree_man_index_st_pctl
tree_man_index_cnty_rank
tree_man_index_cnty_pctl
tree_spart_index
tree_spart_index_nat_rank
tree_spart_index_nat_pctl
tree_spart_index_nat_quan
tree_spart_index_st_rank
tree_spart_index_st_pctl
tree_spart_index_cnty_rank
tree_spart_index_cnty_pctl
tree_flood_buddy_index
tree_flood_buddy_index_nat_rank
tree_flood_buddy_index_nat_pctl
tree_flood_buddy_index_nat_quan
tree_flood_buddy_index_st_rank
tree_flood_buddy_index_st_pctl
tree_flood_buddy_index_cnty_rank
tree_flood_buddy_index_cnty_pctl
tree_vul_pop_index
tree_vul_pop_index_nat_rank
tree_vul_pop_index_nat_pctl
tree_vul_pop_index_nat_quan
tree_vul_pop_index_st_rank
tree_vul_pop_index_st_pctl
tree_vul_pop_index_cnty_rank
tree_vul_pop_index_cnty_pctl
tree_trees_index
tree_trees_index_nat_rank
tree_trees_index_nat_pctl
tree_trees_index_nat_quan
tree_trees_index_st_rank
tree_trees_index_st_pctl
tree_trees_index_cnty_rank
tree_trees_index_cnty_pctl
tree_cooling_center_index
tree_cooling_center_index_nat_rank
tree_cooling_center_index_nat_pctl
tree_cooling_center_index_nat_quan
tree_cooling_center_index_st_rank
tree_cooling_center_index_st_pctl
tree_cooling_center_index_cnty_rank
tree_cooling_center_index_cnty_pctl
tree_heat_buddy_index
```

```
tree_heat_buddy_index_nat_rank
tree_heat_buddy_index_nat_pctl
tree_heat_buddy_index_nat_quan
tree_heat_buddy_index_st_rank
tree_heat_buddy_index_st_pctl
tree_heat_buddy_index_cnty_rank
tree_heat_buddy_index_cnty_pctl
tree_home_hardening_index
tree_home_hardening_index_nat_rank
tree_home_hardening_index_nat_pctl
tree_home_hardening_index_nat_quan
tree_home_hardening_index_st_rank
tree_home_hardening_index_st_pctl
tree_home_hardening_index_cnty_rank
tree_home_hardening_index_cnty_pctl
tree_air_filtration_index
tree_air_filtration_index_nat_rank
tree_air_filtration_index_nat_pctl
tree_air_filtration_index_nat_quan
tree_air_filtration_index_st_rank
tree_air_filtration_index_st_pctl
tree_air_filtration_index_cnty_rank
tree_air_filtration_index_cnty_pctl
tree_improved_egress_index
tree_improved_egress_index_nat_rank
tree_improved_egress_index_nat_pctl
tree_improved_egress_index_nat_quan
tree_improved_egress_index_st_rank
tree_improved_egress_index_st_pctl
tree_improved_egress_index_cnty_rank
tree_improved_egress_index_cnty_pctl
tree_in_flood_aware_index
tree_in_flood_aware_index_nat_rank
tree_in_flood_aware_index_nat_pctl
tree_in_flood_aware_index_nat_quan
tree_in_flood_aware_index_st_rank
tree_in_flood_aware_index_st_pctl
tree_in_flood_aware_index_cnty_rank
tree_in_flood_aware_index_cnty_pctl
tree_in_flood_egress_index
tree_in_flood_egress_index_nat_rank
tree_in_flood_egress_index_nat_pctl
tree_in_flood_egress_index_nat_quan
tree_in_flood_egress_index_st_rank
tree_in_flood_egress_index_st_pctl
tree_in_flood_egress_index_cnty_rank
tree_in_flood_egress_index_cnty_pctl
tree_pres_open_space_index
tree_pres_open_space_index_nat_rank
```

```
tree_pres_open_space_index_nat_pctl
tree_pres_open_space_index_nat_quan
tree_pres_open_space_index_st_rank
tree_pres_open_space_index_st_pctl
tree_pres_open_space_index_cnty_rank
tree_pres_open_space_index_cnty_pctl
tree_reduce_imp_surf_index
tree_reduce_imp_surf_index_nat_rank
tree_reduce_imp_surf_index_nat_pctl
tree_reduce_imp_surf_index_nat_quan
tree_reduce_imp_surf_index_st_rank
tree_reduce_imp_surf_index_st_pctl
tree_reduce_imp_surf_index_cnty_rank
tree_reduce_imp_surf_index_cnty_pctl
tree_restore_builtup_index
tree_restore_builtup_index_nat_rank
tree_restore_builtup_index_nat_pctl
tree_restore_builtup_index_nat_quan
tree_restore_builtup_index_st_rank
tree_restore_builtup_index_st_pctl
tree_restore_builtup_index_cnty_rank
tree_restore_builtup_index_cnty_pctl
tree_intervention_score
tree_shape__area
tree_shape__length
target_next_temp
is_extreme

================================================

Total columns: 225
```

Section 5 – City-Day Modeling on Master Table (Person C)

In this section we train and evaluate machine learning models using the final **city-day master feature table** built in Section 4:

- Regression target: `target_next_temp` (next-day average temperature by city-day)
- Classification target: `is_extreme` (top 30% hottest city-days)
- Models:
  - Logistic Regression (classification baseline)
  - Random Forest & Gradient Boosting (regression)
  - Optional: Random Forest & Gradient Boosting (classification)
- Outputs:
  - Metrics tables (R^2, RMSE, MAE, Accuracy, F1, ROC-AUC)

- Confusion matrix for classification
- Feature-importance bar charts (RF & GBM)
- Cross-validation scores and simple hyperparameter tuning

5.0 Load master table and define features / targets

```
import os
import numpy as np
import pandas as pd

# Reload master if it does not exist in memory
if "master" not in globals():
    master_path = os.path.join(INTERIM_DIR, "cityday_master.parquet")
    print("Reloading master from:", master_path)
    master = pd.read_parquet(master_path)

print("Master shape:", master.shape)

# Keep only rows where targets exist
model_df = master.dropna(subset=["target_next_temp",
"is_extreme"]).copy()
print("After dropping rows with missing targets:", model_df.shape)

# Targets
target_reg = "target_next_temp"
target_clf = "is_extreme"

# Columns that should NOT be used as ML features
exclude_cols = ["city", "dt", target_reg, target_clf]

# Candidate feature columns
candidate_cols = [c for c in model_df.columns if c not in
exclude_cols]

# Keep only numeric features (avoid strings such as 'United States')
numeric_feature_cols = [
    c for c in candidate_cols
    if np.issubdtype(model_df[c].dtype, np.number)
]

# Report which columns were excluded because they were not numeric
dropped = set(candidate_cols) - set(numeric_feature_cols)
print("Dropped non-numeric columns:", dropped)

# Final feature list
feature_cols = numeric_feature_cols

# Feature and target matrices
X = model_df[feature_cols]
y_reg = model_df[target_reg]
```

```
y_clf = model_df[target_clf]

print("Number of numeric features:", len(feature_cols))
print("Sample features:", feature_cols[:10])
```

```
Master shape: (10695, 225)
After dropping rows with missing targets: (10690, 225)
Dropped non-numeric columns: {'country'}
Number of numeric features: 220
Sample features: ['year', 'month', 'day', 'dayofyear', 'dow',
'avg_temp_c', 'lat', 'lon', 'iet_temp_afternoon', 'iet_temp_evening']
```

5.1 Train / Test split

```
from sklearn.model_selection import train_test_split

Xtr_reg, Xte_reg, ytr_reg, yte_reg = train_test_split(
    X, y_reg, test_size=0.2, random_state=RANDOM_STATE
)

Xtr_clf, Xte_clf, ytr_clf, yte_clf = train_test_split(
    X, y_clf, test_size=0.2, random_state=RANDOM_STATE, stratify=y_clf
)

print("Regression train/test:", Xtr_reg.shape, Xte_reg.shape)
print("Classification train/test:", Xtr_clf.shape, Xte_clf.shape)
```

```
Regression train/test: (8552, 220) (2138, 220)
Classification train/test: (8552, 220) (2138, 220)
```

5.2 Helper functions for metrics / tables

```
import numpy as np
from sklearn.metrics import (
    accuracy_score, f1_score, roc_auc_score,
    mean_squared_error, mean_absolute_error, r2_score
)
import pandas as pd

def regression_metrics(y_true, y_pred):
    """
    Return RMSE, MAE, and R^2 in a dict.
    We do NOT use the 'squared' argument to avoid conflicts with
    any custom mean_squared_error functions defined earlier.
    """
    mse = mean_squared_error(y_true, y_pred)   # may be sklearn or
custom
    rmse = np.sqrt(mse)
    return {
        "RMSE": rmse,
```

```
        "MAE":  mean_absolute_error(y_true, y_pred),
        "R2":   r2_score(y_true, y_pred),
    }

def classification_metrics(y_true, y_pred, y_proba=None):
    """Return Accuracy, F1, and ROC-AUC (if probabilities
available)."""
    out = {
        "Accuracy": accuracy_score(y_true, y_pred),
        "F1":       f1_score(y_true, y_pred),
    }
    if y_proba is not None:
        out["ROC_AUC"] = roc_auc_score(y_true, y_proba)
    return out

def show_results_table(results_dict, title):
    """results_dict: {model_name: {metric: value}}"""
    print(title)
    df_res = pd.DataFrame(results_dict).T
    display(df_res.round(4))
```

5.3 Logistic Regression baseline (classification)

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

log_clf = Pipeline([
    ("scaler", StandardScaler()),
    ("model", LogisticRegression(max_iter=1000))
])

log_clf.fit(Xtr_clf, ytr_clf)
y_pred_log = log_clf.predict(Xte_clf)
y_proba_log = log_clf.predict_proba(Xte_clf)[:, 1]

log_results = classification_metrics(yte_clf, y_pred_log, y_proba_log)
show_results_table({"Logistic Regression": log_results},
                   "Classification metrics — Logistic Regression
baseline")
```

```
Classification metrics — Logistic Regression baseline
```

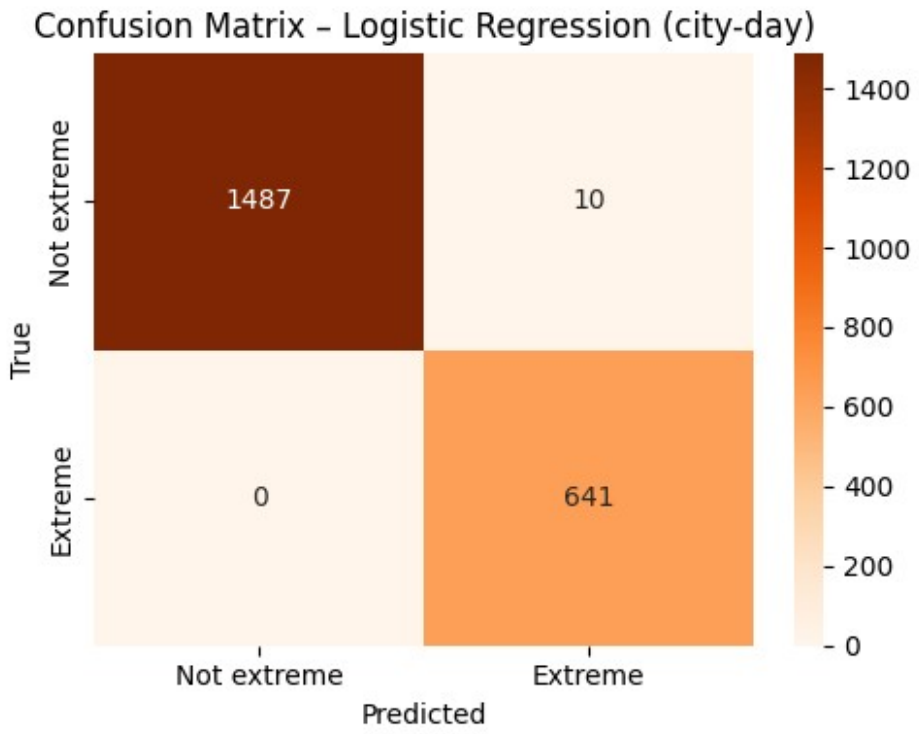

```
{"summary":"{\n  \"name\": \"                    \\\"Classification
metrics \\u2013 Logistic Regression baseline\\\")\",\n  \"rows\": 1,\n
\"fields\": [\n    {\n      \"column\": \"Accuracy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
null,\n        \"min\": 0.9953,\n        \"max\": 0.9953,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          0.9953\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
```

```
}\n    },\n    {\n      \"column\": \"F1\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
0.9923,\n        \"max\": 0.9923,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          0.9923\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ROC_AUC\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": null,\n
\"min\": 1.0,\n        \"max\": 1.0,\n        \"num_unique_values\":
1,\n        \"samples\": [\n          1.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


5.3.1 Confusion matrix for Logistic Regression

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(yte_clf, y_pred_log)

plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Oranges",
            xticklabels=["Not extreme", "Extreme"],
            yticklabels=["Not extreme", "Extreme"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix — Logistic Regression (city-day)")
plt.tight_layout()

fig_path = os.path.join(REPORTS_DIR, "cityday_conf_matrix_logreg.png")
plt.savefig(fig_path, dpi=300)
print("Saved confusion matrix to:", fig_path)
plt.show()
```

```
Saved confusion matrix to: /content/drive/MyDrive/CSE475 - Final
Project/reports_figures/cityday_conf_matrix_logreg.png
```



5.4 Random Forest & Gradient Boosting – Regression

```
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor

rf_reg = RandomForestRegressor(
    n_estimators=400,
    max_depth=None,
    random_state=RANDOM_STATE,
    n_jobs=-1
)
rf_reg.fit(Xtr_reg, ytr_reg)
y_pred_rf = rf_reg.predict(Xte_reg)

gb_reg = GradientBoostingRegressor(
    n_estimators=400,
    learning_rate=0.05,
    max_depth=3,
    random_state=RANDOM_STATE
)
gb_reg.fit(Xtr_reg, ytr_reg)
y_pred_gb = gb_reg.predict(Xte_reg)

reg_results = {
    "Random Forest (reg)": regression_metrics(yte_reg, y_pred_rf),
    "Gradient Boosting (reg)": regression_metrics(yte_reg, y_pred_gb),
}
```

```
show_results_table(reg_results, "Regression metrics — city-day master table")
```

```
Regression metrics — city-day master table
```


```
{"summary":"{\n  \"name\": \"show_results_table(reg_results,
\\\"Regression metrics \\u2013 city-day master table\\\")\",\n
\"rows\": 2,\n  \"fields\": [\n    {\n      \"column\": \"RMSE\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.6228903635472298,\n        \"min\": 0.2274,\n        \"max\":
1.1083,\n        \"num_unique_values\": 2,\n        \"samples\": [\n
1.1083,\n          0.2274\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"MAE\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.5501997864412527,\n        \"min\":
0.0827,\n        \"max\": 0.8608,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.8608,\n          0.0827\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"R2\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.013010764773832454,\n
\"min\": 0.9808,\n        \"max\": 0.9992,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0.9808,\n
0.9992\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


5.5 Random Forest & Gradient Boosting – Classification

```
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier

rf_clf = RandomForestClassifier(
    n_estimators=400,
    max_depth=None,
    random_state=RANDOM_STATE,
    n_jobs=-1
)
rf_clf.fit(Xtr_clf, ytr_clf)
y_pred_rf_clf = rf_clf.predict(Xte_clf)
y_proba_rf_clf = rf_clf.predict_proba(Xte_clf)[:, 1]

gb_clf = GradientBoostingClassifier(
    n_estimators=400,
    learning_rate=0.05,
    max_depth=3,
    random_state=RANDOM_STATE
)
gb_clf.fit(Xtr_clf, ytr_clf)
y_pred_gb_clf = gb_clf.predict(Xte_clf)
y_proba_gb_clf = gb_clf.predict_proba(Xte_clf)[:, 1]
```

```
clf_results = {
    "Logistic Regression":  log_results,
    "Random Forest (clf)":  classification_metrics(yte_clf,
y_pred_rf_clf, y_proba_rf_clf),
    "Gradient Boosting (clf)": classification_metrics(yte_clf,
y_pred_gb_clf, y_proba_gb_clf),
}
show_results_table(clf_results, "Classification metrics — model
comparison (city-day)")
```

```
Classification metrics — model comparison (city-day)
```

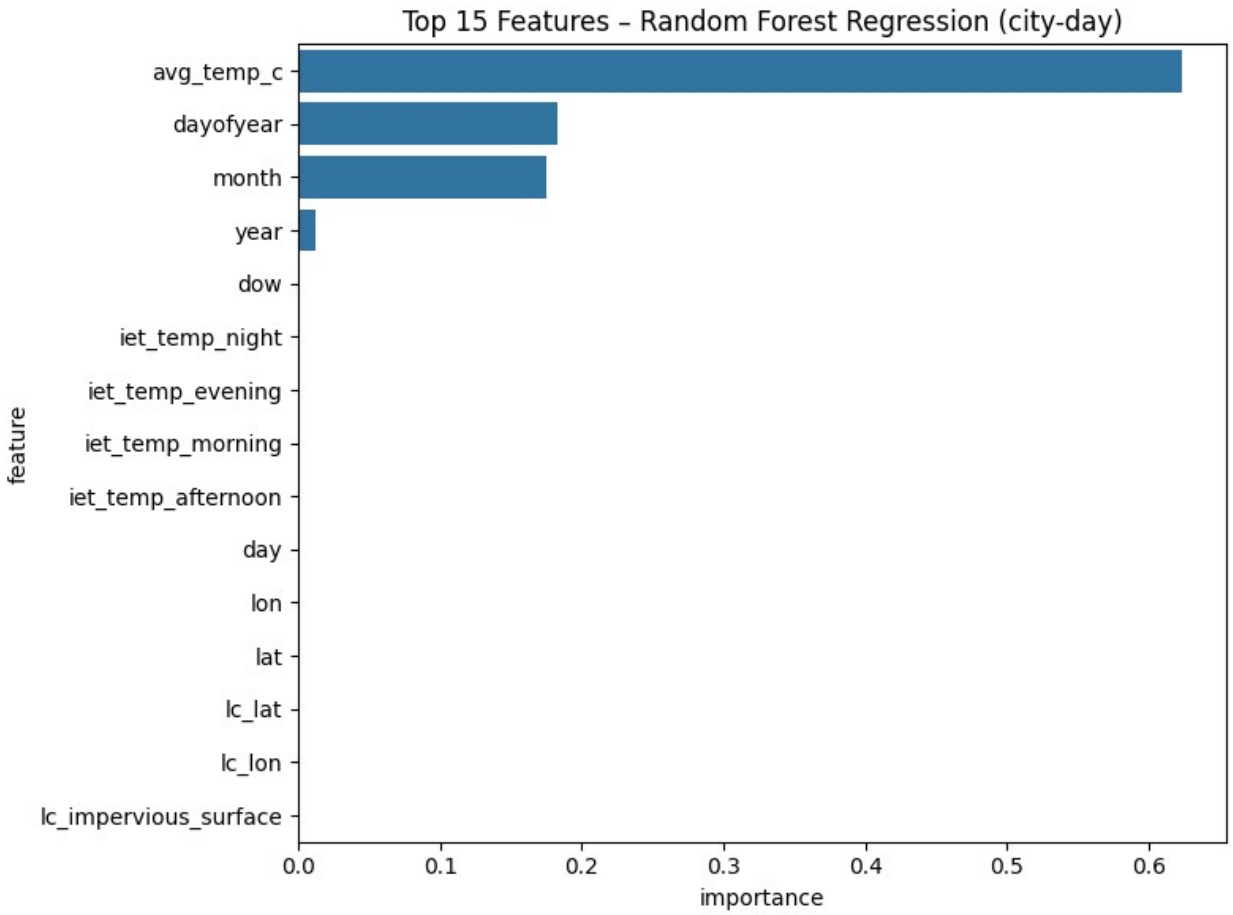

```
{"summary":"{\n  \"name\": \"show_results_table(clf_results,
\\\"Classification metrics \\u2013 model comparison (city-
day)\\\")\",\n  \"rows\": 3,\n  \"fields\": [\n    {\n
\"column\": \"Accuracy\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.002713546265191263,\n        \"min\":
0.9953,\n        \"max\": 1.0,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1.0,\n          0.9953\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"F1\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.004445597072760142,\n
\"min\": 0.9923,\n        \"max\": 1.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1.0,\n
0.9923\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ROC_AUC\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.0,\n        \"min\": 1.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          1.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


5.6 Feature importance – Random Forest & Gradient Boosting

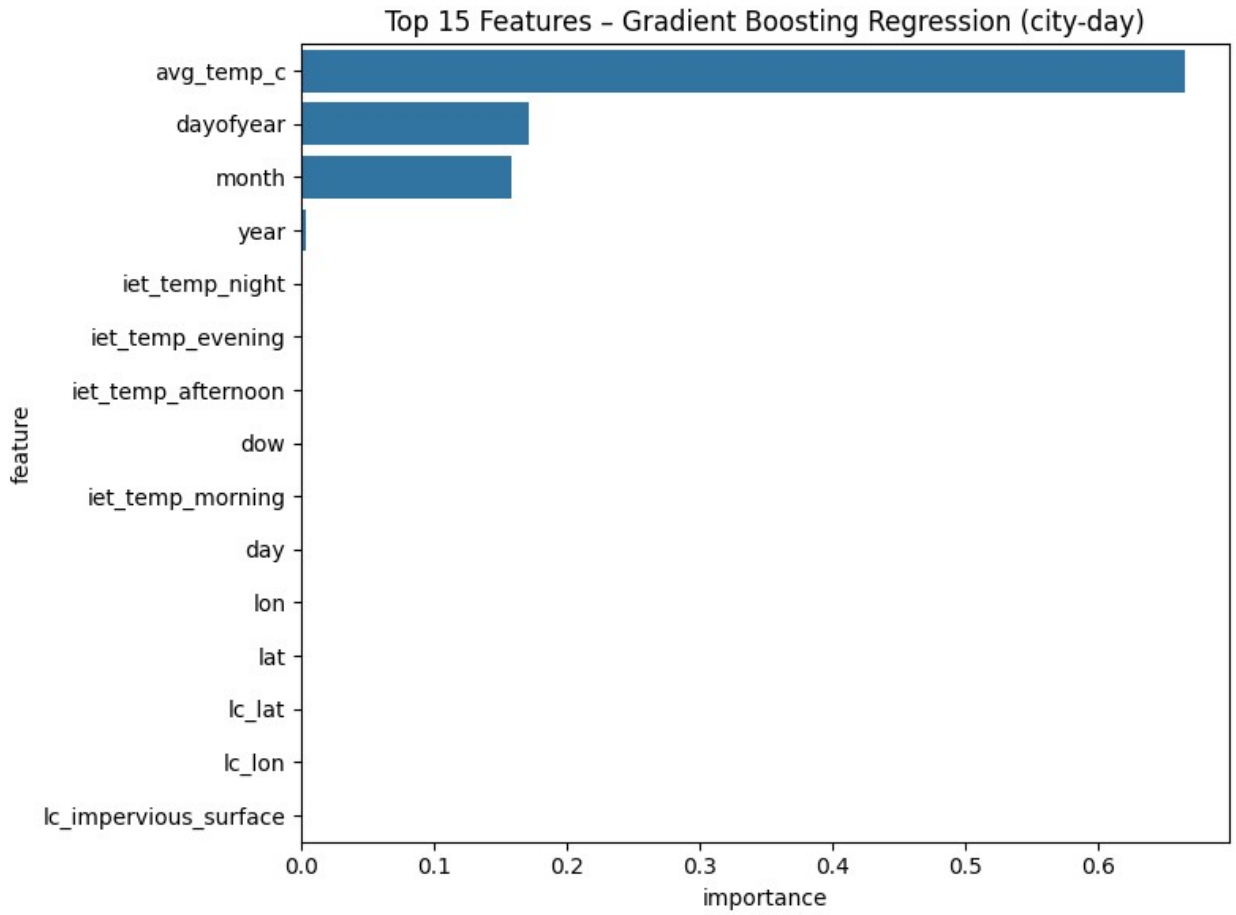
```
feature_names = feature_cols

# Random Forest importance
rf_importances = (
    pd.DataFrame({
        "feature": feature_names,
        "importance": rf_reg.feature_importances_
    })
    .sort_values("importance", ascending=False)
)

plt.figure(figsize=(8,6))
sns.barplot(data=rf_importances.head(15), x="importance", y="feature")
plt.title("Top 15 Features — Random Forest Regression (city-day)")
plt.tight_layout()
rf_path = os.path.join(REPORTS_DIR,
```

```
"cityday_feature_importance_rf_reg.png")
plt.savefig(rf_path, dpi=300)
print("Saved RF feature importance to:", rf_path)
plt.show()

# Gradient Boosting importance
gb_importances = (
    pd.DataFrame({
        "feature": feature_names,
        "importance": gb_reg.feature_importances_
    })
    .sort_values("importance", ascending=False)
)

plt.figure(figsize=(8,6))
sns.barplot(data=gb_importances.head(15), x="importance", y="feature")
plt.title("Top 15 Features — Gradient Boosting Regression (city-day)")
plt.tight_layout()
gb_path = os.path.join(REPORTS_DIR,
"cityday_feature_importance_gb_reg.png")
plt.savefig(gb_path, dpi=300)
print("Saved GBM feature importance to:", gb_path)
plt.show()
```

```
Saved RF feature importance to: /content/drive/MyDrive/CSE475 - Final
Project/reports_figures/cityday_feature_importance_rf_reg.png
```




```
Saved GBM feature importance to: /content/drive/MyDrive/CSE475 - Final
Project/reports_figures/cityday_feature_importance_gb_reg.png
```



5.7 Cross-validation for key models

```
from sklearn.model_selection import cross_val_score

# Logistic Regression – ROC-AUC
cv_log = cross_val_score(
    log_clf, X, y_clf,
    cv=5, scoring="roc_auc", n_jobs=-1
)
print("Logistic Regression CV ROC-AUC: ",
      cv_log.mean(), "+/-", cv_log.std())

# Random Forest Regression – R²
cv_rf_reg = cross_val_score(
    rf_reg, X, y_reg,
    cv=5, scoring="r2", n_jobs=-1
)
print("Random Forest Regression CV R²:",
      cv_rf_reg.mean(), "+/-", cv_rf_reg.std())
```

```
Logistic Regression CV ROC-AUC:  0.9999739468536657 +/- 0.0
Random Forest Regression CV R²: 0.9999867325828454 +/- 0.0
```

5.8 Simple hyperparameter tuning for RF regression

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    "n_estimators": [200, 400],
    "max_depth": [None, 10, 20],
}

rf_reg_base = RandomForestRegressor(random_state=RANDOM_STATE,
n_jobs=-1)

grid = GridSearchCV(
    rf_reg_base,
    param_grid=param_grid,
    cv=3,
    scoring="r2",
    n_jobs=-1
)

grid.fit(X, y_reg)

print("Best parameters:", grid.best_params_)
print("Best CV R²:", grid.best_score_)
```

```
Best parameters: {'max_depth': None, 'n_estimators': 400}
Best CV R²: 0.9999254978351767
```

5.9 Time-Series Cross-Validation (Regression)

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import r2_score
import numpy as np

# Use only the date columns that actually exist in master
date_cols = [c for c in ["year", "month", "day"] if c in
master.columns]

# Build column list for TS-CV and remove duplicates while preserving
order
ts_cols_base = date_cols + feature_cols + [target_reg]
ts_cols = list(dict.fromkeys(ts_cols_base))  # removes duplicates like
'year'

# Subset the master table
df_ts = master[ts_cols].copy()

# Drop any rows that contain NaNs in features or target
df_ts = df_ts.dropna(subset=feature_cols + [target_reg])
print("TS-CV data shape after dropping NaNs:", df_ts.shape)
```

```
# Sort chronologically to respect the time dimension
if date_cols:
    df_ts = df_ts.sort_values(by=date_cols)

X_ts = df_ts[feature_cols].values
y_ts = df_ts[target_reg].values

tscv = TimeSeriesSplit(n_splits=5)

fold_scores = []

for fold, (train_idx, test_idx) in enumerate(tscv.split(X_ts),
start=1):
    X_train, X_test = X_ts[train_idx], X_ts[test_idx]
    y_train, y_test = y_ts[train_idx], y_ts[test_idx]

    # Extra safety: skip degenerate folds with no variation in y_test
    if len(np.unique(y_test)) < 2:
        print(f"Fold {fold} skipped (not enough variation in
y_test).")
        continue

    model = RandomForestRegressor(
        n_estimators=300,
        random_state=RANDOM_STATE,
        n_jobs=-1
    )

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    score = r2_score(y_test, y_pred)
    fold_scores.append(score)
    print(f"Fold {fold} R^2: {score:.4f}")

if fold_scores:
    print("TS-CV mean R^2:", np.mean(fold_scores))
    print("TS-CV std:",  np.std(fold_scores))
else:
    print("TS-CV could not be computed: no valid folds.")
```

```
TS-CV data shape after dropping NaNs: (10690, 221)
Fold 1 R^2: 0.9671
Fold 2 R^2: 0.9553
Fold 3 R^2: 0.9619
Fold 4 R^2: 0.9498
Fold 5 R^2: 0.9532
TS-CV mean R^2: 0.9574782454878015
TS-CV std: 0.006221201963833789
```

5.10 Sensitivity Test – Remove avg_temp_c (Regression)

```
# Remove the dominant feature
features_wo_avg = [c for c in feature_cols if c != "avg_temp_c"]

X_reg_wo = master[features_wo_avg]
y_reg_wo = master[target_reg]

# Drop rows that contain NaN in X or y
valid_rows = ~(X_reg_wo.isna().any(axis=1) | y_reg_wo.isna())
X_reg_wo = X_reg_wo[valid_rows]
y_reg_wo = y_reg_wo[valid_rows]

print("Shape after removing avg_temp_c and dropping NaNs:",
X_reg_wo.shape)

# Train/test split
Xtr_wo, Xte_wo, ytr_wo, yte_wo = train_test_split(
    X_reg_wo, y_reg_wo,
    test_size=0.2,
    random_state=42
)

# Train RF
rf_wo = RandomForestRegressor(
    n_estimators=300,
    random_state=42,
    n_jobs=-1
)

rf_wo.fit(Xtr_wo, ytr_wo)
y_pred_wo = rf_wo.predict(Xte_wo)

# Evaluate
reg_metrics_wo = regression_metrics(yte_wo, y_pred_wo)

print("Regression WITHOUT avg_temp_c")
display(reg_metrics_wo)
```

```
Shape after removing avg_temp_c and dropping NaNs: (10690, 219)
Regression WITHOUT avg_temp_c

{'RMSE': np.float64(0.22351837130831007),
 'MAE': 0.0857272419706845,
 'R2': 0.9992170390273349}
```

Section 6: Unsupervised Learning & Spatial Analysis (Person D)

```
# ============================================================
# 6.0 Inspect Master Table (Before Clustering)
```

```
# ============================================================

import pandas as pd
import numpy as np

master_path = os.path.join(INTERIM_DIR, "cityday_master.parquet")
master = pd.read_parquet(master_path)

print("Master shape:", master.shape)
print("Columns:", len(master.columns))
display(master.head(9))
```

```
Master shape: (10695, 225)
Columns: 225
```

{"type":"dataframe"}

```
# ============================================================
# 6.0Filter by year: Only retain data for modern cities (after 2000).
# ============================================================

if "master" in globals():
    master_filtered = master.copy()
else:
    master_path = os.path.join(INTERIM_DIR, "cityday_master.parquet")
    master_filtered = pd.read_parquet(master_path)


master_filtered["dt"] = pd.to_datetime(master_filtered["dt"],
errors="coerce")
master_filtered["year"] = master_filtered["dt"].dt.year


master_filtered = master_filtered[ master_filtered["year"] >=
2000 ].copy()

print("After year filtering (>=2000):", master_filtered.shape)
print("Cities included:", master_filtered["city"].unique())


if "cluster_df" in globals() and "year" in cluster_df.columns:
    df_map = cluster_df[ cluster_df["year"] >= 2000 ].copy()
else:
    df_map = master_filtered.copy()
```

```
After year filtering (>=2000): (825, 225)
Cities included: ['Glendale' 'Mesa' 'Phoenix' 'Scottsdale' 'Tempe']
```

```
# ============================================================
# 6.1 Select Numeric Features for Clustering
```

```
# ============================================================

numeric_cols = [c for c in master.columns if
np.issubdtype(master[c].dtype, np.number)]

# Remove constant column
constant_cols = [c for c in numeric_cols if master[c].nunique() <= 1]

# Remove meaningless columns
remove_cols = ["lat", "lon", "lc_lat", "lc_lon", "year"]
remove_cols += [c for c in numeric_cols if ("geoid" in c.lower()) or
c.endswith("_id")]

drop_cols = set(constant_cols + remove_cols)

cluster_cols = [c for c in numeric_cols if c not in drop_cols]

print("Original numeric:", len(numeric_cols))
print("Removed:", drop_cols)
print("Final cluster features:", len(cluster_cols))

Original numeric: 222
Removed: {'lc_buildings', 'tree_flood_buddy_index_cnty_rank',
'tree_in_flood_egress_index_st_pctl',
'tree_flood_buddy_index_nat_pctl', 'tree_vul_pop_index_nat_quan',
'tree_cooling_center_index_nat_pctl',
'tree_reduce_imp_surf_index_cnty_rank', 'tree_trees_index_cnty_pctl',
'tree_b01001_027e', 'tree_trees_index_st_rank',
'tree_b25002_calc_pcttotalrente', 'tree_vul_pop_index_cnty_rank',
'lulc_tree', 'tree_intervention_score', 'tree_pct_hh_lives_alone',
'tree_spart_index_cnty_rank', 'tree_heat_buddy_index',
'tree_spart_index_cnty_pctl', 'tree_in_flood_aware_index_cnty_rank',
'lc_lat', 'tree_mean_annual_est_pm2_5_µg_m3', 'lon',
'lulc_tair_diff_nt', 'tree_cnt_rd_inter_per_sqkm_box_cox',
'tree_trees_index_nat_pctl', 'tree_restore_builtup_index_nat_pctl',
'tree_pct_area_restore', 'tree_pres_open_space_index_cnty_rank',
'tree_air_filtration_index', 'tree_in_flood_aware_index_nat_pctl',
'tree_reduce_imp_surf_index', 'tree_air_filtration_index_st_pctl',
'tree_in_flood_aware_index', 'tree_pct_treecanopy_box_cox',
'tree_aland', 'tree_pct_tract_blw_sl_2050',
'tree_trees_index_cnty_rank', 'lulc_grass',
'tree_in_flood_aware_index_nat_quan',
'lc_mean_landsat_land_surface_temperature',
'tree_spart_index_nat_quan', 'tree_cnt_rd_inter_per_sqkm',
'tree_flood_buddy_index_st_pctl',
'tree_restore_builtup_index_nat_quan',
'tree_cooling_center_index_st_rank', 'tree_high_summer_mean_lst_f',
'tree_restore_builtup_index_st_rank', 'tree_spart_index_nat_pctl',
'tree_flood_buddy_index_nat_rank',
'tree_home_hardening_index_st_rank', 'tree_b28002_calc_pctnointe',
```

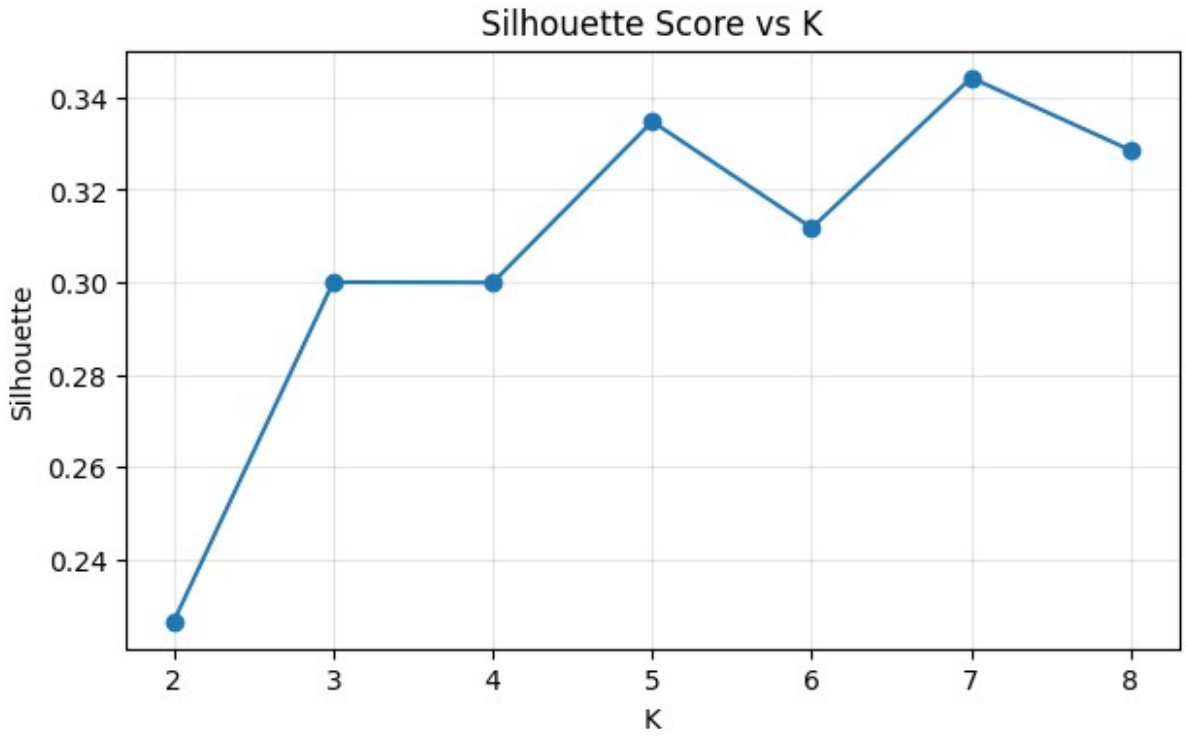
```
'tree_count_property', 'tree_pop_density_ppl_sqkm_box_cox',
'tree_pct_treecanopy', 'tree_cooling_center_index_cnty_rank',
'tree_reduce_imp_surf_index_st_pctl', 'lulc_tair_pred25_75',
'tree_awater', 'tree_cooling_center_index_cnty_pctl',
'tree_wf_risktohome_mean_box_cox', 'tree_man_index_nat_pctl',
'tree_in_flood_egress_index_nat_rank', 'tree_pct_hu_built_prior_1970',
'lulc_lst', 'tree_coastal_tract', 'tree_spartina_length',
'lulc_tair_pred95', 'tree_in_flood_aware_index_st_pctl',
'tree_pct_area_restore_box_cox', 'tree_improved_egress_index_st_pctl',
'tree_casthma_crudeprev', 'tree_flood_buddy_index',
'tree_improved_egress_index_cnty_pctl', 'tree_man_index_st_pctl',
'tree_single_tract', 'tree_wf_housingdensity_mean',
'tree_spart_index_st_pctl', 'lat',
'tree_pres_open_space_index_nat_pctl',
'tree_restore_builtup_index_cnty_pctl',
'tree_reduce_imp_surf_index_st_rank', 'tree_vul_pop_index',
'tree_wf_hazardpotential_mean', 'tree_wf_risktohome_mean',
'tree_vul_pop_index_nat_rank', 'tree_home_hardening_index_cnty_rank',
'lc_impervious_surface', 'tree_in_flood_egress_index_cnty_pctl',
'tree_area_sqkm', 'tree_shape__length', 'lc_regional_wind_speed',
'lc_sensor_recorded_air_temperature', 'tree_b25002_calc_pcttotalowne',
'tree_air_filtration_index_cnty_pctl',
'tree_restore_builtup_index_st_pctl',
'tree_pres_open_space_index_cnty_pctl', 'tree_pct_imperviousurfaces',
'tree_wf_hazardpotential_mean_box_cox', 'tree_objectid',
'tree_wf_exp_type_mean', 'tree_cooling_center_index_nat_quan',
'tree_air_filtration_index_nat_pctl', 'tree_man_index_nat_quan',
'tree_cooling_center_index_nat_rank', 'tree_spart_index_st_rank',
'lulc_lon', 'tree_home_hardening_index_st_pctl', 'year',
'tree_reduce_imp_surf_index_nat_pctl',
'tree_home_hardening_index_cnty_pctl',
'tree_improved_egress_index_nat_quan',
'tree_restore_builtup_index_cnty_rank',
'tree_air_filtration_index_nat_quan', 'tree_pct_riparian_square_root',
'tree_air_filtration_index_st_rank', 'tree_trees_index_st_pctl',
'tree_trees_index_nat_rank', 'tree_flood_buddy_index_cnty_pctl',
'tree_home_hardening_index', 'tree_improved_egress_index_nat_pctl',
'tree_pres_open_space_index_nat_rank', 'tree_max_cc',
'tree_heat_buddy_index_nat_rank', 'lc_lon',
'tree_spart_index_nat_rank', 'tree_b16004_calc_pctge18leae',
'tree_b03002_calc_pctnhwhitee', 'tree_in_flood_egress_index',
'tree_shape__area', 'tree_b17020_calc_pctpove', 'tree_pct_riparian',
'tree_pres_open_space_index_st_pctl',
'tree_in_flood_egress_index_cnty_rank', 'tree_b01001_calc_pctdepende',
'lc_turf/grass', 'tree_spart_index',
'tree_wf_housingdensity_mean_box_cox', 'tree_vul_pop_index_nat_pctl',
'tree_pct_young_people', 'lulc_tair_pred95_nt',
'tree_flood_buddy_index_nat_quan', 'tree_vul_pop_index_cnty_pctl',
'tree_b01001_001e', 'tree_geoid', 'tree_vul_pop_index_st_rank',
```

```
'lc_regional_air_temperature', 'tree_reduce_imp_surf_index_nat_rank',
'tree_vul_pop_index_st_pctl', 'tree_reduce_imp_surf_index_cnty_pctl',
'tree_pct_pop_minority', 'day', 'tree_b25002_001e',
'tree_man_index_st_rank', 'tree_cooling_center_index',
'tree_b01001_003e', 'tree_heat_buddy_index_nat_quan', 'lulc_building',
'tree_total_young_people', 'tree_heat_buddy_index_st_rank',
'tree_cooling_center_index_st_pctl',
'tree_improved_egress_index_nat_rank',
'tree_improved_egress_index_st_rank', 'tree_pres_open_space_index',
'tree_restore_builtup_index', 'tree_pop_density_ppl_sqkm',
'tree_man_index_cnty_pctl', 'tree_heat_buddy_index_st_pctl',
'tree_improved_egress_index', 'tree_heat_buddy_index_nat_pctl',
'tree_b18101_calc_pctde', 'lulc_tair_pred25_75_nt',
'tree_b08201_calc_pctnovehe', 'tree_cnt_rd_inter', 'tree_trees_index',
'lulc_impervious', 'tree_b01001_calc_pctge65e_box_cox',
'tree_pct_tract_undev', 'tree_man_index_cnty_rank',
'tree_pct_area_protect_box_cox', 'tree_heat_buddy_index_cnty_rank',
'tree_b25002_calc_pctvace', 'tree_reduce_imp_surf_index_nat_quan',
'tree_air_filtration_index_nat_rank', 'tree_pct_area_protect',
'tree_flood_buddy_index_st_rank', 'tree_heat_buddy_index_cnty_pctl',
'tree_in_flood_egress_index_nat_pctl',
'tree_home_hardening_index_nat_pctl', 'tree_avg_vul_2050',
'tree_improved_egress_index_cnty_rank',
'tree_b08201_calc_pctnovehe_box_cox',
'tree_in_flood_aware_index_cnty_pctl',
'lc_regional_relative_humidity', 'tree_man_index_nat_rank',
'tree_b15002_calc_pctlthse', 'tree_in_flood_aware_index_nat_rank',
'tree_mangrove_length', 'tree_in_flood_egress_index_st_rank',
'tree_b01001_calc_pctge65e', 'tree_in_flood_aware_index_st_rank',
'tree_pres_open_space_index_nat_quan',
'tree_pres_open_space_index_st_rank',
'tree_home_hardening_index_nat_rank', 'tree_trees_index_nat_quan',
'tree_pct_lackingcanopy', 'tree_man_index', 'lulc_lat',
'tree_home_hardening_index_nat_quan',
'tree_restore_builtup_index_nat_rank',
'tree_in_flood_egress_index_nat_quan',
'tree_air_filtration_index_cnty_rank', 'lc_tree_canopy'}
Final cluster features: 10

# ============================
# 6.2 KMeans + Silhouette
# ============================
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

# First, only take numerical features
df_cluster_full = master[cluster_cols]

#Record rows with absolutely no missing values.
```

```
valid_idx = df_cluster_full.dropna().index
df_cluster = df_cluster_full.loc[valid_idx]

print("Original row number:", master.shape[0])
print("Number of rows used for clustering (without missing rows):",
df_cluster.shape[0])

# standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_cluster)

# Silhouette
sil_scores = []
k_range = range(2, 9)

for k in k_range:
    km = KMeans(n_clusters=k, n_init=10, random_state=42)
    labels = km.fit_predict(X_scaled)
    sil = silhouette_score(X_scaled, labels)
    sil_scores.append(sil)
    print(f"k={k}, silhouette={sil:.4f}")

plt.figure(figsize=(7,4))
plt.plot(k_range, sil_scores, marker='o')
plt.title("Silhouette Score vs K")
plt.xlabel("K")
plt.ylabel("Silhouette")
plt.grid(alpha=0.3)
plt.show()

best_k = k_range[np.argmax(sil_scores)]
print("Best k =", best_k)

# Refit using the optimal k
kmeans_final = KMeans(n_clusters=best_k, n_init=10, random_state=42)
cluster_labels = kmeans_final.fit_predict(X_scaled)

# Create a new cluster column in the master module: first set all of
them to NaN.
master["cluster"] = np.nan
# Only fill in the labels for the lines in valid_idx.
master.loc[valid_idx, "cluster"] = cluster_labels
```

```
Original row number: 10695
Number of rows used for clustering (without missing rows): 10690
k=2, silhouette=0.2267
k=3, silhouette=0.3000
k=4, silhouette=0.2999
k=5, silhouette=0.3348
k=6, silhouette=0.3117
```

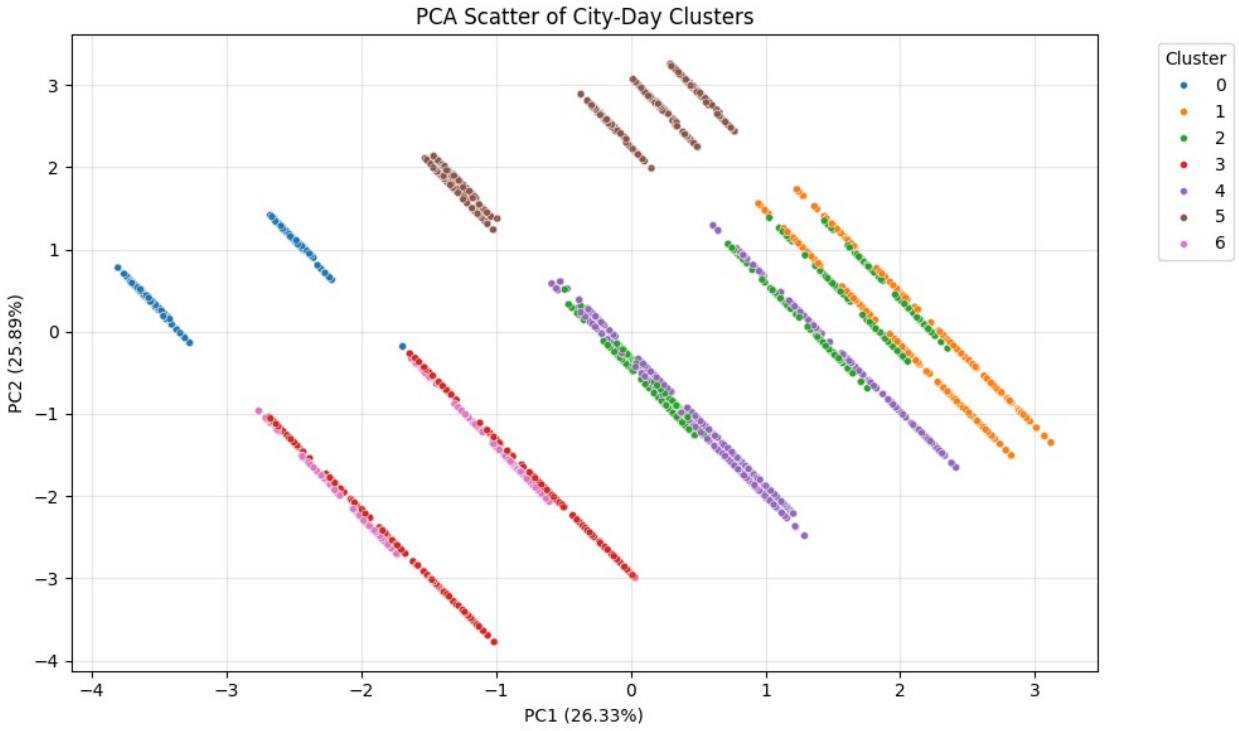
```
k=7, silhouette=0.3442
k=8, silhouette=0.3285
```




```
Best k = 7
```

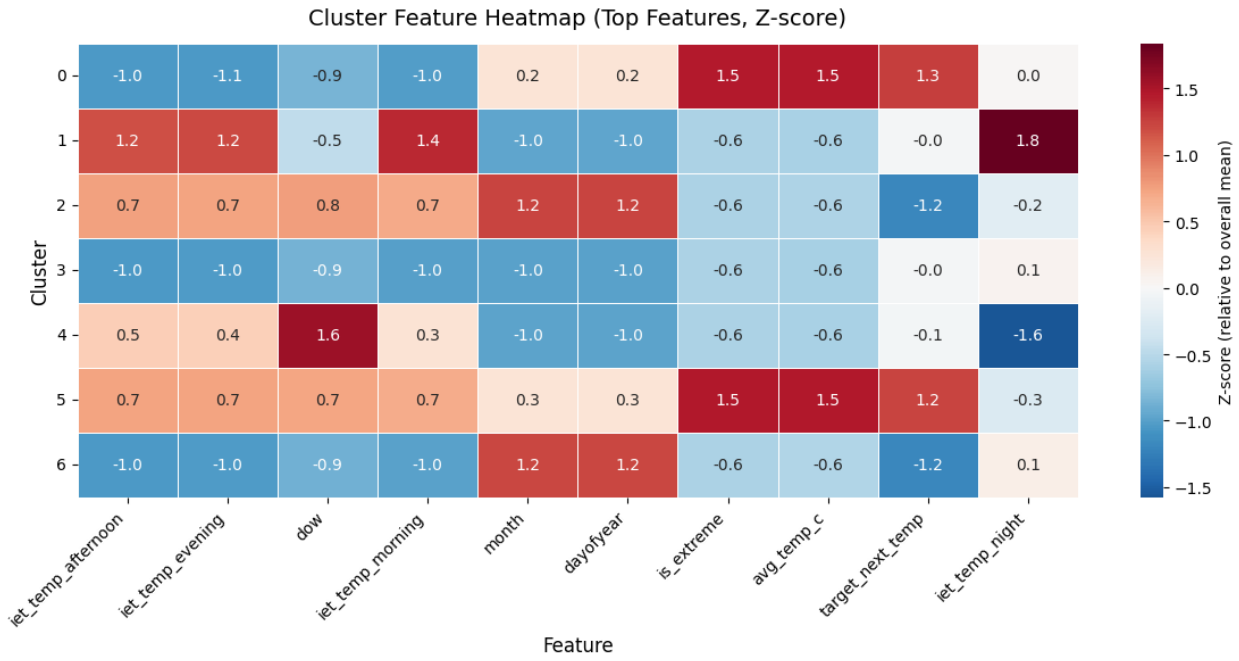
```python
# ============================================================
# 6.3 PCA Scatter Plot of City-Day Clusters
# ============================================================

from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Only take the labels that participated in the clustering.
cluster_labels = master.loc[valid_idx, "cluster"].astype(int).values

# PCA reduces the standardized features to 2 dimensions.
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)   # X_scaled comes from 6.2

print("Explained variance ratio:", pca.explained_variance_ratio_)

plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=X_pca[:, 0],
    y=X_pca[:, 1],
```

```
    hue=cluster_labels,
    palette="tab10",
    s=18
)
plt.title("PCA Scatter of City-Day Clusters")
plt.xlabel(f"PC1 ({pca.explained_variance_ratio_[0]:.2%})")
plt.ylabel(f"PC2 ({pca.explained_variance_ratio_[1]:.2%})")
plt.grid(alpha=0.3)
plt.legend(title="Cluster", bbox_to_anchor=(1.05, 1), loc="upper
left")
plt.tight_layout()

plt.savefig(os.path.join(REPORTS_DIR, "pca_clusters.png"), dpi=300)
plt.show()


Explained variance ratio: [0.2632559  0.25893066]
```

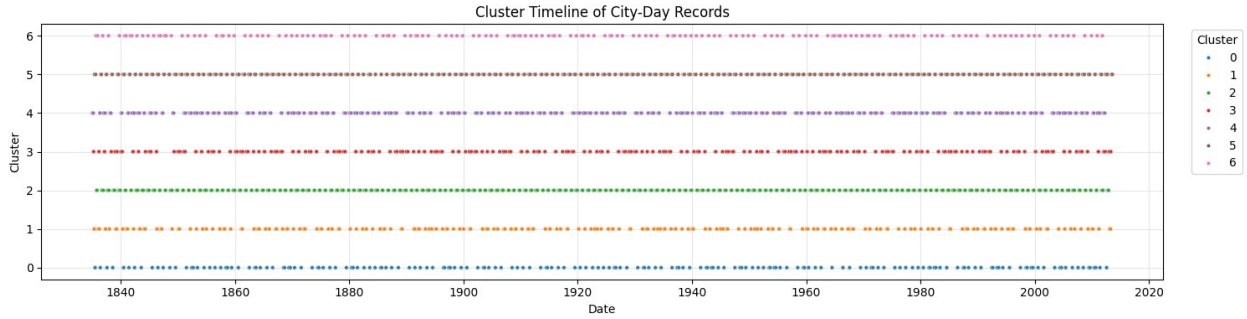



```
# ============================================================
# 6.4(a) Prettier Cluster Feature Heatmap
# ============================================================

import matplotlib.pyplot as plt
import seaborn as sns

# Use only the rows with the cluster tag.
master_valid = master.dropna(subset=["cluster"]).copy()
```

```
master_valid["cluster"] = master_valid["cluster"].astype(int)

# Eigenmean of each cluster
cluster_means = master_valid.groupby("cluster")[cluster_cols].mean()

# z-score
cluster_means_z = (cluster_means - cluster_means.mean()) /
cluster_means.std()

# 1)First, calculate "which features differ most between the
clusters".
feature_importance =
cluster_means_z.abs().mean(axis=0).sort_values(ascending=False)
topN = 12   # If you want more, change it to 15 or 20.
top_features = feature_importance.head(topN).index

cm = cluster_means_z[top_features]

cm.columns = [c.replace("_index_nat_quan", "") for c in cm.columns]
cm.columns = [c.replace("_pct_", "%_") for c in cm.columns]

plt.figure(figsize=(1.0*topN, 6))  # Width is linked to the number of
features, making it look more symmetrical.
sns.heatmap(
    cm,
    cmap="RdBu_r",
    center=0,
    annot=True,             # Write the numbers in the grid.
    fmt=".1f",
    linewidths=0.5,
    linecolor="white",
    cbar_kws={"label": "Z-score (relative to overall mean)"}
)

plt.title("Cluster Feature Heatmap (Top Features, Z-score)",
fontsize=14, pad=12)
plt.ylabel("Cluster", fontsize=12)
plt.xlabel("Feature", fontsize=12)

plt.xticks(rotation=45, ha="right", fontsize=10)
plt.yticks(rotation=0, fontsize=10)

plt.tight_layout()
plt.savefig(os.path.join(REPORTS_DIR,
"cluster_feature_heatmap_pretty.png"), dpi=300)
plt.show()
```




```
# ============================================================
# 6.4(b) Cluster Timeline Over Date
# ============================================================

plt.figure(figsize=(15, 4))
sns.scatterplot(
    data=master_valid,
    x="dt",
    y="cluster",
    hue="cluster",
    palette="tab10",
    s=10
)
plt.title("Cluster Timeline of City-Day Records")
plt.xlabel("Date")
plt.ylabel("Cluster")
plt.grid(alpha=0.3)
plt.legend(title="Cluster", bbox_to_anchor=(1.02, 1), loc="upper
left")
plt.tight_layout()

plt.savefig(os.path.join(REPORTS_DIR, "cluster_timeline.png"),
dpi=300)
plt.show()
```




```
# Initialize df_map_city from df_map, which is available from previous
cells
df_map_city = df_map.copy()

print("df_map_city dt range:", df_map_city["dt"].min(), "=>",
df_map_city["dt"].max())
if "year" in df_map_city.columns:
    print("df_map_city year range:", df_map_city["year"].min(), "=>",
df_map_city["year"].max())
else:
    print("df_map_city no year list")
```

```
df_map_city dt range: 2000-01-01 00:00:00 => 2013-09-01 00:00:00
df_map_city year range: 2000 => 2013
```

```
# ============================================================
# 6.5 Interactive Map A — City-Day Cluster / Temperature Summary
# ============================================================
!pip -q install folium branca

import folium
from folium.plugins import MarkerCluster
import branca.colormap as cm
import numpy as np
import pandas as pd
import os

# ---------- 0) From the previous steps, we obtained... df_map
----------
df_map_city = df_map.copy()

print("df_map_city dt range:", df_map_city["dt"].min(), "=>",
df_map_city["dt"].max())
if "year" in df_map_city.columns:
    print("df_map_city year range:", df_map_city["year"].min(), "=>",
df_map_city["year"].max())
else:
    print("df_map_city 没有 year 列")
```

```
if "cluster" not in df_map_city.columns:
    df_map_city["cluster"] = 0


df_map_city = df_map_city.dropna(subset=["lat", "lon"]).copy()

# ---------- 1)Aggregate by city / lat / lon and calculate
statistics.----------
city_summary = (
    df_map_city
    .groupby(["city", "lat", "lon"], as_index=False)
    .agg(
        n_days        = ("dt", "count"),
        avg_temp      = ("avg_temp_c", "mean"),
        extreme_rate  = ("is_extreme", "mean"),
        median_cluster= ("cluster", "median"),
    )
)

print(" city_summary (lat/lon (Not yet corrected):")
display(city_summary[["city", "lat", "lon"]])

# ---------- 2) Manually specify the actual coordinates of 5 cities
----------

manual_coords = {
    "Glendale":   (33.5386, -112.1850),
    "Phoenix":    (33.4484, -112.0740),
    "Mesa":       (33.4152, -111.8315),
    "Scottsdale": (33.4942, -111.9261),
    "Tempe":      (33.4255, -111.9400),
}

# Create new columns `plot_lat` and `plot_lon`: use manual coordinates
first, then fall back to the original `lat`/`lon` coordinates for
other cities.
city_summary["plot_lat"] = city_summary["city"].map(lambda c:
manual_coords.get(c, (np.nan, np.nan))[0])
city_summary["plot_lon"] = city_summary["city"].map(lambda c:
manual_coords.get(c, (np.nan, np.nan))[1])

mask = city_summary["plot_lat"].isna()
city_summary.loc[mask, "plot_lat"] = city_summary.loc[mask, "lat"]
city_summary.loc[mask, "plot_lon"] = city_summary.loc[mask, "lon"]

print("Revised city_summary：")
display(city_summary[["city", "lat", "lon", "plot_lat", "plot_lon"]])

# ---------- 3) Use the corrected plot_lat / plot_lon to draw Folium
maps.----------
```

```
center_lat = city_summary["plot_lat"].mean()
center_lon = city_summary["plot_lon"].mean()

m_city = folium.Map(
    location=[center_lat, center_lon],
    zoom_start=10,
    tiles="CartoDB positron"
)

vmin = city_summary["avg_temp"].min()
vmax = city_summary["avg_temp"].max()
cmap = cm.LinearColormap(
    colors=["blue", "yellow", "red"],
    vmin=vmin,
    vmax=vmax,
    caption="Mean daily temperature (°C)"
)

marker_cluster = MarkerCluster().add_to(m_city)

for _, row in city_summary.iterrows():
    color = cmap(row["avg_temp"])
    popup_html = f"""
    <b>{row['city']}</b><br>
    Days in dataset: {int(row['n_days'])}<br>
    Mean temp: {row['avg_temp']:.1f} °C<br>
    Extreme-day fraction: {row['extreme_rate']:.2f}<br>
    Median cluster: {int(row['median_cluster'])}
    """
    folium.CircleMarker(
        location=[row["plot_lat"], row["plot_lon"]],
        radius=8,
        color=color,
        fill=True,
        fill_color=color,
        fill_opacity=0.85,
        popup=folium.Popup(popup_html, max_width=260),
    ).add_to(marker_cluster)

cmap.add_to(m_city)

# ---------- 4) Save + Show ----------
map_city_path = os.path.join(REPORTS_DIR, "map_cityday_clusters.html")
m_city.save(map_city_path)
print("□ Saved interactive city-day map to:", map_city_path)

from IPython.display import HTML
HTML(m_city._repr_html_())
```

df_map_city dt range: 2000-01-01 00:00:00 => 2013-09-01 00:00:00
df_map_city year range: 2000 => 2013
 city_summary（lat/lon (Not yet corrected):


```
{"summary":"{\n  \"name\": \"HTML(m_city\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"city\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"Mesa\",\n          \"Tempe\",\n          \"Phoenix\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lat\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\":
32.95,\n        \"max\": 32.95,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          32.95\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lon\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.5888218580782548e-14,\n
\"min\": -112.02,\n        \"max\": -112.02,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          -112.02\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


Revised city_summary：


```
{"summary":"{\n  \"name\": \"HTML(m_city\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"city\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"Mesa\",\n          \"Tempe\",\n          \"Phoenix\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lat\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\":
32.95,\n        \"max\": 32.95,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          32.95\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"lon\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.5888218580782548e-14,\n
\"min\": -112.02,\n        \"max\": -112.02,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          -112.02\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"plot_lat\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.05143813760236781,\n        \"min\": 33.4152,\n        \"max\":
33.5386,\n        \"num_unique_values\": 5,\n        \"samples\": [\n
33.4152\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"plot_lon\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.13854398940408635,\n        \"min\": -
112.185,\n        \"max\": -111.8315,\n        \"num_unique_values\":
5,\n        \"samples\": [\n          -111.8315\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}

 Saved interactive city-day map to: /content/drive/MyDrive/CSE475 -
Final Project/reports_figures/map_cityday_clusters.html

<IPython.core.display.HTML object>

# ============================================================
# 6.5 AZ LULC Priority Points Map (using LULC_temp_prediction)
# ============================================================
import folium
from folium.plugins import MarkerCluster
import branca.colormap as cm
import numpy as np
import pandas as pd
import os

# 1) Ensure Lulc is already loaded

# If Lulc already exists, there's no need to load it again.
if "lulc" not in globals():
    lulc_path = os.path.join(RAW_DIR, "LULC_temp_prediction.csv")
    print("Reloading LULC from:", lulc_path)
    lulc = pd.read_csv(lulc_path)

print("LULC shape:", lulc.shape)

# 2)First, convert lat/lon to a numerical value (as a precaution).
lulc["lat"] = pd.to_numeric(lulc["lat"], errors="coerce")
lulc["lon"] = pd.to_numeric(lulc["lon"], errors="coerce")

# 3) Filter points within the AZ range by latitude and longitude
(rough bounding box)
az_lulc = lulc[
    lulc["lat"].between(31, 37) &    # Arizona approximate latitude
    lulc["lon"].between(-115, -109)  # Arizona approximate longitude
].copy()

print("AZ subset shape:", az_lulc.shape)
print("AZ cities:", az_lulc["city"].value_counts().head())

if az_lulc.empty:
    print("⚠ The absence of points within the AZ area suggests that
this LULC dataset may only include other cities such as Baltimore.。 ")
else:
    # 4)Construct a "priority index"

    for col in ["tree", "impervious", "building"]:
        if col not in az_lulc.columns:
            print(f"List {col} If it's not in LULC, please change it
```

```
to the actual column name in your data.。")
    az_lulc["priority"] = (
        0.5 * (1 - az_lulc["tree"]) +     # Fewer trees → higher score
        0.3 * az_lulc["impervious"] +     #Hard paving is more common
→ High score
        0.2 * az_lulc["building"]         # More buildings → higher
score
    )

    # 5) To prevent map lag, a maximum of 2000 points will be sampled.
    max_points = 2000
    if len(az_lulc) > max_points:
        az_lulc = az_lulc.sample(max_points,
random_state=RANDOM_STATE)
        print(f"Sampled {max_points} AZ points for mapping.")
    else:
        print(f"Using all {len(az_lulc)} AZ points for mapping.")

    # 6) Create a Folium map
    center_lat = az_lulc["lat"].mean()
    center_lon = az_lulc["lon"].mean()

    m_az = folium.Map(
        location=[center_lat, center_lon],
        zoom_start=10,
        tiles="CartoDB positron"
    )

    # Color range (using quantiles to prevent extreme values)
    vmin = az_lulc["priority"].quantile(0.05)
    vmax = az_lulc["priority"].quantile(0.95)

    cmap = cm.LinearColormap(
        colors=["green", "yellow", "red"],
        vmin=vmin,
        vmax=vmax,
        caption="Tree-planting priority (low → high)"
    )

    marker_cluster = MarkerCluster().add_to(m_az)

    val_min = az_lulc["priority"].min()
    val_max = az_lulc["priority"].max()
    denom = (val_max - val_min) if (val_max - val_min) != 0 else 1.0

    for _, row in az_lulc.iterrows():
        v = row["priority"]
        color = cmap(v)
        radius = 4 + 6 * (v - val_min) / denom  # Radius 4~10
```

```
        popup_html = f"""
        <b>{row.get('city', 'Unknown city')}</b><br>
        Priority score: {v:.2f}<br>
        Tree cover: {row['tree']:.2f}<br>
        Impervious: {row['impervious']:.2f}<br>
        (lat, lon) = ({row['lat']:.4f}, {row['lon']:.4f})
        """

        folium.CircleMarker(
            location=[row["lat"], row["lon"]],
            radius=float(radius),
            color=color,
            fill=True,
            fill_color=color,
            fill_opacity=0.8,
            popup=folium.Popup(popup_html, max_width=260),
        ).add_to(marker_cluster)

    cmap.add_to(m_az)

    # 7) Save + Show
    map_az_path = os.path.join(REPORTS_DIR,
"map_lulc_priority_az_points.html")
    m_az.save(map_az_path)
    print("Saved AZ LULC priority map to:", map_az_path)

    m_az
from IPython.display import HTML

HTML(m_az._repr_html_())
```

```
LULC shape: (693932, 14)
AZ subset shape: (251462, 14)
AZ cities: city
Phoenix    147190
Tucson     104272
Name: count, dtype: int64
Sampled 2000 AZ points for mapping.
Saved AZ LULC priority map to: /content/drive/MyDrive/CSE475 - Final
Project/reports_figures/map_lulc_priority_az_points.html

<IPython.core.display.HTML object>
```