



Geo-X training course

Hands-on 5

Please use the following citation for:

- **This exercise**

Miguel Sanchez-Lopez, Michaela Rychetska, 2026, Hands-on 5: Interpreting the Results – Geo-X Training Course (Version 1.0.). Climate Compatible Growth. DOI: [to be assigned].

- **Geo-X software**

Sanchez-Lopez, M., Naqvi, S., Leonard, A., Schirren, L., & Hirmer, S. (2026). Geo-X: Generalised geospatial software for production of green commodities. *SoftwareX*, 34, 102712. <https://doi.org/10.1016/j.softx.2026.102712>

Learning outcomes

By the end of this exercise, you will be able to:

- Load and inspect Geo-X GeoJSON outputs using Python and geopandas.
- Read and interpret heatmaps: spatial gradients, production zones, and dominant cost drivers.
- Decompose the levelised cost into its components and identify what drives costs in different locations.
- Construct a cumulative supply cost curve from hexagon-level outputs, and understand its assumptions and limitations.

This exercise is hydrogen only. The same code reruns for ammonia by changing one line (TYPE = 'ammonia'), which we leave for you to try afterwards. A companion notebook, `Geo-X Hands-on 5 - Ghana Analysis Starter.ipynb`, is provided alongside this document with all the cells below ready to run against your own results.

Activity 1 – Get a full-country run

In Hands-on 4, you set `restrict_hexagons: True` to keep your first run fast. That means your results file only has rows for hexagons containing Accra. This exercise needs spatial variation across all of Ghana's hexagons, so before anything else, you need a full-country run.

Try it:

- On your `ghana-h2-nh3` branch, set `restrict_hexagons: False` in `config.yaml`.
- Rerun: `snakemake -j 4 map_all`

Note: Ghana's small hexagon count (177) should keep this manageable, but it will still take longer than the restricted run from Hands-on 4. Commit the new results when the run finishes, before moving on.

Activity 2 – Set up your analysis notebook

This exercise uses Python and `geopandas`. The same GeoJSON outputs are fully compatible with R, QGIS, or any GIS tool, so feel free to follow along in your tool of choice; the code below is for Jupyter.

Try it:

- Activate your environment and open Jupyter: `mamba activate geox`, then `jupyter notebook`
- Open the companion notebook, or create a new one and copy the cells below as you go.

```
# Cell 1: imports
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt

# Cell 2: set your scenario
ISO = "GA"
YEAR = "2023"
TYPE = "hydrogen"
DEMAND_CENTRE = "Accra"
```

```
# Cell 3: load output files
df_cost = gpd.read_file(
    f"results/hex_total_cost_{ISO}_{YEAR}_{TYPE}.geojson"
)
df_comp = gpd.read_file(
    f"results/hex_cost_components_{ISO}_{YEAR}_{TYPE}.geojson"
)
```

Activity 3 – Sanity-check your outputs

Make this a habit before any visualisation. It catches most problems early.

```
print(df_cost.shape)
print(df_cost["demand_centre"].unique())
print(df_cost.columns.tolist())
print(df_comp.columns.tolist())
print(df_cost[["total_cost"]].describe())
print("Nulls:", df_cost["total_cost"].isnull().sum())
```

- **Very few rows:** restrict_hexagons may still be True in config.yaml. Go back to Activity 1.
- **Wrong demand centre name:** check demand_parameters.xlsx.
- **Many nulls in total_cost:** infeasible hexagons, check exclusion layers.

Activity 4 – Plot the heatmap

```
df = df_cost[df_cost["demand_centre"] == DEMAND_CENTRE]

fig, ax = plt.subplots(figsize=(10, 6))
df.plot(
    column="total_cost",
    cmap="viridis",
    legend=True,
    ax=ax,
    missing_kwds={"color": "lightgrey"},
)
ax.set_title(f"{DEMAND_CENTRE} – Total LCOH ({YEAR})")
```

```
ax.axis("off")
plt.show()
```

Note: heatmaps usually track solar, wind, and battery capacity, and solar and wind tend to be anti-correlated across a country. Ghana has strong solar irradiation but weaker wind, which should show up in the pattern you see. Solar-heavy hexagons typically need more battery storage to smooth intermittent output, which feeds into what you will see in Activity 5.

What the heatmap cannot tell you: it shows unit cost, not how much a hexagon could produce at that cost. It is also a single snapshot, sensitive to where you placed the demand centre, to your parameter assumptions, and to the one weather year you used. A low-cost hexagon is not necessarily a high-potential one; the supply curve in Activity 6 addresses that.

Activity 5 – Plot cost components

A single map answers “where is it cheap”. Comparing components tells you “why it is cheap”, which is the more useful question.

```
components = [
    ("production_cost", "Production cost", "viridis"),
    ("transport_cost", "Transport cost", "plasma"),
    ("storage_cost", "Storage cost", "magma"),
    ("water_cost", "Water cost", "cividis"),
]

dc = DEMAND_CENTRE
df = df_comp[df_comp["demand_centre"] == dc]

fig, axes = plt.subplots(2, 2, figsize=(14, 10))
axes = axes.flatten()

for ax, (col, title, cmap) in zip(axes, components):
    df.plot(
        column=col, cmap=cmap, legend=True, ax=ax,
        missing_kws={"color": "lightgrey"},
        vmax=df[col].quantile(0.95),
    )
    ax.set_title(title, fontsize=12, fontweight="bold")
    ax.axis("off")

fig.suptitle(f"Cost components – {dc} · {TYPE} · {YEAR}")
```

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

Then, to see the composition rather than the spatial pattern, a stacked bar of your lowest-cost hexagons:

```
cost_cols = ["storage_cost", "solar_cost", "wind_cost",
             "electrolyser_cost", "transport_cost"]
colours = {
    "storage_cost": "#F4A3A3",
    "solar_cost": "#F5C842",
    "wind_cost": "#A8C4E8",
    "electrolyser_cost": "#BFE3B4",
    "transport_cost": "#D3D3D3",
}

df = df_comp[df_comp["demand_centre"] == dc]
p05 = df["total_cost"].quantile(0.05)
low = df[df["total_cost"] <= p05]
means = low[cost_cols].mean()

fig, ax = plt.subplots(figsize=(6, 7))
bottom = 0
for col in cost_cols:
    ax.bar("Lowest-cost\n5%", means[col], bottom=bottom,
          color=colours[col], label=col.replace("_cost", "").title(),
          edgecolor="white", linewidth=0.5)
    bottom += means[col]

ax.set_ylabel("Average LCOH [EUR/kg]")
ax.legend(loc="upper right", fontsize=9, frameon=False)
plt.tight_layout()
plt.show()
```

Note: the colour palette above is borrowed from CCG's own SHIELD project publication figures, which use the same technique on Ukraine's case. If storage costs make up a large share of your lowest-cost hexagons, that is telling you those locations rely on battery capacity to bridge gaps in solar output, not that they are poor sites overall.

Activity 6 – Build a supply cost curve

A supply cost curve ranks hexagons by delivered cost, assigns each a production volume, and plots cumulative volume against unit cost. Where a heatmap shows where it is cheap,

a supply curve shows how much is available at each cost level, which is what a policy question like “how much could this country produce below X EUR/kg” actually needs.

```
df = df_cost[df_cost["demand_centre"] == DEMAND_CENTRE]
df = df.dropna(subset=["total_cost"])

plant_capacity = 1000 # t H2 / km2 / yr, adjust to your case
df["potential"] = df["eligible_area"] * plant_capacity

df_sorted = df.sort_values("total_cost").reset_index()
df_sorted["cum_potential"] = df_sorted["potential"].cumsum() / 1e6

fig, ax = plt.subplots(figsize=(8, 5))
ax.step(df_sorted["cum_potential"], df_sorted["total_cost"],
        where="post", color="#1E3A6E", linewidth=1.8,
        label="Supply curve")

ax.set_xlabel("Cumulative production potential [Mt H2/yr]")
ax.set_ylabel("Delivered cost [EUR/kg]")
ax.set_title(f"Supply cost curve – {DEMAND_CENTRE} · {TYPE} · {YEAR}")
ax.legend(frameon=False)
ax.grid(True, alpha=0.3)
plt.show()
```

Note on methodology and its limits: each hexagon is treated as an independent, standalone plant, sized to meet a fixed annual demand target, with no system-level interaction between hexagons. This is a screening assumption, not a dispatch-optimised system where demand could be met from several locations at once. The result is informative and aligned with other policy-oriented models, but more sophisticated approaches exist, such as higher spatial resolution, dynamic capacity scaling, or full system integration.

Wrap-up

You have completed this exercise. Before moving on, check that:

- You have a full-country run (restrict_hexagons: False) committed to your branch.
- You can load and sanity-check both output files before plotting anything.
- You can explain the spatial pattern in Ghana's LCOH heatmap in terms of solar and wind resource.



- You can identify which cost component dominates Ghana's lowest-cost hexagons, and why.
- You have built a supply cost curve, and can explain what it shows that the heatmap cannot.

Well done! You can now read and build the core outputs of a Geo-X run. In Class 6, we will cover advanced modelling options that extend what Geo-X can do, including new generation and storage technologies and demand-side flexibility.