

IND320 — Project Work, Part 2: Data Sources

Links

- **Live updated app:** <https://ind320-project-work-nonewthing.streamlit.app/>
- **Repo:** <https://github.com/TaoM29/IND320-dashboard-basics>

Plan for Part 2

1. Fetch Elhub **PRODUCTION_PER_GROUP_MBA_HOUR** for **2021** (chunked by month).
2. Parse `productionPerGroupMbaHour` → tidy DataFrame (`priceArea`, `productionGroup`, `startTime`, `endTime`, `quantityKwh`).
3. Write DataFrame to **Cassandra** (`ind320.elhub_production_raw`) via **Spark**.
4. Use **Spark** to select: `priceArea`, `productionGroup`, `startTime`, `quantityKwh`.
5. Plots:
 - Pie: total 2021 by **production group** for a chosen **price area**.
 - Line: **January 2021** for a chosen **price area**, separate lines per **production group**.
6. Insert curated data into **MongoDB** (for Streamlit).
7. Log (300–500 words) + brief AI usage note.

AI usage

I used an ChatGPT 5 as a coding and troubleshooting partner. It helped me:

- (1) interpret stack traces from Spark/Cassandra (e.g., classloader and Scala ABI errors).
- (2) propose alternative ingestion paths when the connector failed (CSV → cqlsh COPY → Cassandra, and later a MongoDB Atlas pipeline).
- (3) draft concise, idiomatic code snippets for pandas, PyMongo, and Streamlit,
- (4) suggest security and deployment practices (e.g., `st.secrets`, rotating credentials, and `.gitignore`). I reviewed and adapted all suggestions, verified them locally, and made final design decisions about schemas, indexes, and UI. No AI was used to fabricate data, it only assisted with code, debugging, and documentation.

Work Log

This part of the project focused on acquiring hourly electricity production data, curating it, storing it in a database suitable for a web UI, and building an interactive Streamlit page.

Jupyter Notebook (data work). I called the Elhub v0 API for the dataset `PRODUCTION_PER_GROUP_MBA_HOUR`. The payload nests the measurements inside `productionPerGroupMbaHour`, so I extracted only the fields I needed: `priceArea`, `productionGroup`, `startTime` (UTC), and `quantityKwh`. I implemented a helper to emit ISO-8601 with an explicit offset, iterated month-by-month across all price areas and groups for 2021,

normalized column names, parsed timestamps as timezone-aware UTC, and converted quantities to numeric. The result was a clean pandas DataFrame for 2021 that I also persisted to CSV.

I first attempted to write directly from Spark to Cassandra using the Spark-Cassandra connector. Despite having the assembly JAR on the classpath, I repeatedly hit classloader and Scala ABI issues (e.g., `NoClassDefFoundError: scala/lesscolon$less`). After several mitigations (forcing V1 datasource, assembly vs. slim, explicit provider names), I pivoted to a more robust approach for the course timeline: export CSV and use `cqlsh COPY` to validate the table design, then choose MongoDB Atlas as the primary store for the Streamlit app.

MongoDB pipeline. In Atlas I created database `ind320` with two collections: `prod_hour` (hourly facts) and `prod_year_totals` (pre-aggregated sums by area and group). I upserted the hourly rows in batches and added a unique index on (`price_area`, `production_group`, `start_time`) plus a supporting query index on (`price_area`, `start_time`). On the free tier, upserts took several minutes, which is expected because each write hits the index. I could have tested bulk insert + then indexing is faster if deduplication is guaranteed.

Streamlit (web UI). I refactored the app into a clean multipage structure and implemented a new page `energy_production` according to the brief: two columns via `st.columns`. The left column has a `st.radio` for price area and renders a pie chart (from the pre-aggregated totals, with an on-the-fly aggregation fallback). The right column uses `st.pills` (falling back to `st.multiselect` if unavailable) to choose production groups and a month selector to plot group-wise hourly lines. Secrets are read via `st.secrets` locally and on Streamlit Cloud to avoid leaking credentials. I also documented the data source in an `st.expander`.

Along the way I addressed a sporadic UI glitch by ensuring unique widget keys and creating fresh Matplotlib figures per rerun.

```
import requests
from cassandra.cluster import Cluster

cluster = Cluster(['localhost'], port=9042)
session = cluster.connect("elhub") # my keyspace

# Output verbosity control
VERBOSE = False

def debug(*args, **kwargs):
    """Lightweight logger to keep the notebook output clean.
    Toggle VERBOSE=True above to see messages."""
    if VERBOSE:
        print(*args, **kwargs)

# Setup + probes (Elhub)
import os, requests, pandas as pd
from datetime import datetime, timezone

BASE_V0 = "https://api.elhub.no/energy-data/v0"
DATASET = "PRODUCTION_PER_GROUP_MBA_HOUR"
ELHUB_API_TOKEN = os.getenv("ELHUB_API_TOKEN")
```

```

def headers_jsonapi():
    # v0 uses JSON:API; this content-type is required
    h = {"Accept": "application/vnd.api+json"}
    if ELHUB_API_TOKEN:
        h["Authorization"] = f"Bearer {ELHUB_API_TOKEN}"
    return h

def iso_utc_offset(dt: datetime) -> str:
    """
    Return ISO-8601 with offset like '+00:00' (v0 requires this
    format).
    """
    if dt.tzinfo is None:
        dt = dt.replace(tzinfo=timezone.utc)
    dt = dt.astimezone(timezone.utc)
    off = dt.strftime("%z")
    off = off[:-2] + ":" + off[-2:]
    return dt.strftime("%Y-%m-%dT%H:%M:%S") + off

# production-groups (lists valid group ids)
r = requests.get(f"{BASE_V0}/production-groups",
headers=headers_jsonapi(), timeout=30)
print("production-groups → HTTP", r.status_code, "| Content-Type:",
r.headers.get("Content-Type"))
r.raise_for_status()
pg_payload = r.json()

pg_rows = []
for item in pg_payload.get("data", []):
    attrs = item.get("attributes", {}) or {}
    pg_rows.append({"id": item.get("id"), "name": attrs.get("name"),
"description": attrs.get("description")})
production_groups_df = pd.DataFrame(pg_rows)
display(production_groups_df)

# show the entity we'll query (price-areas) just to confirm it
responds
r2 = requests.get(f"{BASE_V0}/price-areas", headers=headers_jsonapi(),
timeout=30, params={"dataset": DATASET, "pageSize": 1})
print("price-areas (dataset ping) → HTTP", r2.status_code, "| Content-
Type:", r2.headers.get("Content-Type"))

production-groups → HTTP 200 | Content-Type: application/json;
charset=utf-8

```

	id	name	description
0	solar	Solar	Unit in which solar energy is converted to ele...
1	hydro	Hydro	Unit in which moving water energy is converted...
2	wind	Wind	Unit in which wind energy is converted to elec...
3	thermal	Thermal	Unit in which heat energy is converted to elec...
4	nuclear	Nuclear	Unit in which the heat source is a nuclear rea...
5	other	Other	Other unspecified technology.
6	*	*	*

price-areas (dataset ping) → HTTP 200 | Content-Type:
application/json; charset=utf-8

Fetch one month (N01 + hydro) with flexible key mapping

```
import pandas as pd
import requests
from datetime import datetime, timezone

def fetch_month(price_area: str, production_group: str, year: int,
month: int) -> pd.DataFrame:

    # month window (inclusive start, exclusive end)
    start = datetime(year, month, 1, tzinfo=timezone.utc)
    end = datetime(year + (month==12), (month % 12) + 1, 1,
tzinfo=timezone.utc)

    params = {
        "dataset": DATASET,
        "priceArea": price_area,
        "productionGroup": production_group,
        "startDate": iso_utc_offset(start),
        "endDate": iso_utc_offset(end),
        "pageSize": 10000
    }

    url = f"{BASE_V0}/price-areas"
    r = requests.get(url, headers=headers_jsonapi(), params=params,
timeout=90)
    print("HTTP", r.status_code, "|", r.headers.get("Content-Type"))
    print("URL:", r.url)

    if r.status_code != 200:
        print("Body preview:", r.text[:600])
        return pd.DataFrame()

    j = r.json()
    data = [rec.get("attributes", {}) for rec in j.get("data", [])]
    if not data:
        print("No data returned.")
        return
    pd.DataFrame(columns=["priceArea", "productionGroup", "startTime", "quant
```

```

ityKwh"]])

df_raw = pd.DataFrame(data)
print("Raw columns from API:", list(df_raw.columns))

# Flexible mapping: try multiple variants for each required field
variants = {
    "priceArea": ["priceArea", "PRISOMRÅDE", "price_area"],
    "productionGroup": ["productionGroup", "PRODUKSJONSGRUPPE",
"production_group"],
    "startTime": ["startTime", "START_TID", "start_time",
"startDateTime", "start_date_time"],
    "quantityKwh": ["quantityKwh", "VOLUM_KWH",
"quantity_kwh", "quantityKWh", "kwh", "volumeKwh"],
}

colmap = {}
for target, opts in variants.items():
    for c in opts:
        if c in df_raw.columns:
            colmap[c] = target
            break

missing = [t for t in variants if t not in colmap.values()]
if missing:
    print("Could not find required fields:", missing)
    print("Please show these API columns to me so I can add
mappings.")
    display(df_raw.head())
    return
pd.DataFrame(columns=["priceArea", "productionGroup", "startTime", "quant
ityKwh"])

df = df_raw.rename(columns=colmap)
[["priceArea", "productionGroup", "startTime", "quantityKwh"]]
df["startTime"] = pd.to_datetime(df["startTime"], utc=True,
errors="coerce")
df["quantityKwh"] = pd.to_numeric(df["quantityKwh"],
errors="coerce")
df =
df.dropna(subset=["startTime", "quantityKwh"]).reset_index(drop=True)

print(f"Fetchd rows: {len(df)}")
return df

# test for one month: N01 + hydro, Jan 2021
df_test = fetch_month("N01", "hydro", 2021, 1)
display(df_test.head(10))
print("Columns:", df_test.columns.tolist())
print("Date range:", (df_test["startTime"].min() if not df_test.empty

```

```
else None),
    "→", (df_test["startTime"].max() if not df_test.empty else
None))
```

HTTP 200 | application/json; charset=utf-8

URL: https://api.elhub.no/energy-data/v0/price-areas?

dataset=PRODUCTION_PER_GROUP_MBA_HOUR&priceArea=N01&productionGroup=hydro&startDate=2021-01-01T00%3A00%3A00%2B00%3A00&endDate=2021-02-01T00%3A00%3A00%2B00%3A00&pageSize=10000

Raw columns from API: ['country', 'eic', 'name', 'productionPerGroupMbaHour']

Could not find required fields: ['priceArea', 'productionGroup', 'startTime', 'quantityKwh']

Please show these API columns to me so I can add mappings.

	country	eic	name	\
0	NO	*	*	
1	NO	10YNO-1-----2	N01	
2	NO	10YNO-2-----T	N02	
3	NO	10YNO-3-----J	N03	
4	NO	10YNO-4-----9	N04	

	productionPerGroupMbaHour
0	[]
1	[{'endTime': '2021-01-01T01:00:00+01:00', 'las...
2	[{'endTime': '2021-01-01T01:00:00+01:00', 'las...
3	[{'endTime': '2021-01-01T01:00:00+01:00', 'las...
4	[{'endTime': '2021-01-01T01:00:00+01:00', 'las...

Empty DataFrame

Columns: [priceArea, productionGroup, startTime, quantityKwh]

Index: []

Columns: ['priceArea', 'productionGroup', 'startTime', 'quantityKwh']

Date range: None → None

Expand nested productionPerGroupMbaHour

```
def fetch_month(price_area: str, production_group: str, year: int,
month: int, verbose: bool = False) -> pd.DataFrame:
    """
```

Fetch one month of hourly production data for a given price area and production group.

Extracts the nested list in 'productionPerGroupMbaHour'.

Set verbose=True to print HTTP/URL/row-count diagnostics.

"""

```
    start = datetime(year, month, 1, tzinfo=timezone.utc)
    end = datetime(year + (month == 12), (month % 12) + 1, 1,
tzinfo=timezone.utc)
```

```
    params = {
```

```

        "dataset": DATASET,
        "priceArea": price_area,
        "productionGroup": production_group,
        "startDate": iso_utc_offset(start),
        "endDate": iso_utc_offset(end),
        "pageSize": 10000,
    }

    url = f"{BASE_V0}/price-areas"
    r = requests.get(url, headers=headers_jsonapi(), params=params,
                    timeout=90)

    if verbose:
        print("HTTP", r.status_code, "|", r.headers.get("Content-
Type"))
        print("URL:", r.url)

    if r.status_code != 200:
        if verbose:
            print("Body preview:", r.text[:400])
        return pd.DataFrame()

    j = r.json()
    data = j.get("data", [])
    if not data:
        if verbose:
            print("No data rows in 'data'.")
        return pd.DataFrame(columns=["priceArea", "productionGroup",
"startTime", "quantityKwh"])

    rows = []
    for rec in data:
        attrs = rec.get("attributes", {})
        area = attrs.get("name") or rec.get("id")
        inner = attrs.get("productionPerGroupMbaHour", [])
        if not inner:
            continue
        for item in inner:
            rows.append({
                "priceArea": area,
                "productionGroup": item.get("productionGroup"),
                "startTime": item.get("startTime"),
                "quantityKwh": item.get("quantityKwh")
            })

    df = pd.DataFrame(rows)
    if df.empty:
        if verbose:
            print("No inner rows found (productionPerGroupMbaHour
empty for this month).")

```

```

        return
pd.DataFrame(columns=["priceArea", "productionGroup", "startTime", "quantityKwh"])

    df["startTime"] = pd.to_datetime(df["startTime"], utc=True,
errors="coerce")
    df["quantityKwh"] = pd.to_numeric(df["quantityKwh"],
errors="coerce")
    df = df.dropna(subset=["startTime",
"quantityKwh"]).reset_index(drop=True)

    if verbose:
        print(f"Fetchd rows: {len(df)}")

    return df

# Test: N01 + hydro, Jan 2021
df_test = fetch_month("N01", "hydro", 2021, 1)
display(df_test.head(10))
print("Columns:", df_test.columns.tolist())
print("Date range:", (df_test["startTime"].min() if not df_test.empty
else None),
      "→", (df_test["startTime"].max() if not df_test.empty else
None))

```

	priceArea	productionGroup	startTime	quantityKwh
0	N01	hydro	2020-12-31 23:00:00+00:00	2507716.8
1	N01	hydro	2021-01-01 00:00:00+00:00	2494728.0
2	N01	hydro	2021-01-01 01:00:00+00:00	2486777.5
3	N01	hydro	2021-01-01 02:00:00+00:00	2461176.0
4	N01	hydro	2021-01-01 03:00:00+00:00	2466969.2
5	N01	hydro	2021-01-01 04:00:00+00:00	2467460.0
6	N01	hydro	2021-01-01 05:00:00+00:00	2482320.8
7	N01	hydro	2021-01-01 06:00:00+00:00	2509533.0
8	N01	hydro	2021-01-01 07:00:00+00:00	2550758.2
9	N01	hydro	2021-01-01 08:00:00+00:00	2693111.0

```

Columns: ['priceArea', 'productionGroup', 'startTime', 'quantityKwh']
Date range: 2020-12-31 23:00:00+00:00 → 2021-01-31 22:00:00+00:00

```

```

# Build full-year (2021) DataFrame
import pandas as pd

PRICE_AREAS = ["N01", "N02", "N03", "N04", "N05"]
valid_groups = [g for g in production_groups_df["id"].tolist() if g !=
"*"]
print("Using production groups:", valid_groups)

year = 2021
parts, runs, non_empty = [], 0, 0

```



```

for area in PRICE_AREAS:
    for group in valid_groups:
        for m in range(1, 13):
            df_m = fetch_month(area, group, year, m)
            if not df_m.empty:
                parts.append(df_m)
                non_empty += 1
            runs += 1
            # keep this as debug so it's silent unless you set
VERBOSE=True elsewhere
            debug(f"{area} {group} {year}-{m:02d}: {len(df_m)} rows")

df_2021 = pd.concat(parts, ignore_index=True) if parts else
pd.DataFrame(
    columns=["priceArea", "productionGroup", "startTime", "quantityKwh"]
)

print("\n=== FULL YEAR SUMMARY ===")
print(f"Year: {year}")
print(f"Requests run: {runs} | Non-empty months: {non_empty}")
print("Total rows:", len(df_2021))
print("Areas:", sorted(df_2021['priceArea'].unique().tolist()) if not
df_2021.empty else [])
print("Groups:", sorted(df_2021['productionGroup'].unique().tolist())
if not df_2021.empty else [])
print("Time span:",
      (df_2021['startTime'].min() if not df_2021.empty else None),
      "→",
      (df_2021['startTime'].max() if not df_2021.empty else None))

display(df_2021.head())

```

Using production groups: ['solar', 'hydro', 'wind', 'thermal', 'nuclear', 'other']

=== FULL YEAR SUMMARY ===

Year: 2021

Requests run: 360 | Non-empty months: 300

Total rows: 1076765

Areas: ['N01', 'N02', 'N03', 'N04', 'N05']

Groups: ['hydro', 'other', 'solar', 'thermal', 'wind']

Time span: 2020-12-31 23:00:00+00:00 → 2021-12-31 22:00:00+00:00

	priceArea	productionGroup	startTime	quantityKwh
0	N01	solar	2020-12-31 23:00:00+00:00	6.106
1	N01	solar	2021-01-01 00:00:00+00:00	4.030
2	N01	solar	2021-01-01 01:00:00+00:00	3.982

3	N01	solar 2021-01-01 02:00:00+00:00	8.146
4	N01	solar 2021-01-01 03:00:00+00:00	8.616

```

from pathlib import Path
import pandas as pd

# Resolve paths relative to the notebook's folder
notebooks_dir = Path.cwd().resolve()
project_root = notebooks_dir.parent
data_dir = project_root / "data"
data_dir.mkdir(parents=True, exist_ok=True)

csv_path = data_dir / "elhub_prod_by_group_hour_2021.csv"
print("Will write CSV to:", csv_path)

# Normalize columns + format for cqlsh COPY
colmap = {
    "priceArea": "price_area", "productionGroup": "production_group",
    "startTime": "start_time", "quantityKwh": "quantity_kwh"
}
df = df_2021.rename(columns={c: colmap.get(c, c) for c in df_2021.columns}).copy()

df["start_time"] = pd.to_datetime(df["start_time"], utc=True,
errors="coerce")
df["start_time"] = df["start_time"].dt.strftime("%Y-%m-%dT%H:%M:%SZ")
df["quantity_kwh"] = pd.to_numeric(df["quantity_kwh"],
errors="coerce")

df = df.dropna(subset=["price_area", "production_group", "start_time", "quantity_kwh"])\
    .drop_duplicates(subset=["price_area", "production_group", "start_time"])

df.to_csv(csv_path, index=False)
print("Wrote rows:", len(df))

Will write CSV to:
/Users/taom/Documents/IND320/IND320-project-work/data/elhub_prod_by_group_hour_2021.csv
Wrote rows: 215353

from datetime import datetime, timezone

# Mock table to test insert
insert_ps = session.prepare("""
INSERT INTO production_per_group_mba_hour
(price_area, production_group, start_time, quantity_kwh)
VALUES (?, ?, ?, ?)

```

```

"""
price_area = "N01"
production_group = "hydro"
start_time = datetime(2021, 1, 1, 0, 0, 0, tzinfo=timezone.utc)
quantity_kwh = 123456.0

session.execute(insert_ps, (price_area, production_group, start_time,
quantity_kwh))

# Quick verify
rows = session.execute("""
SELECT price_area, production_group, start_time, quantity_kwh
FROM production_per_group_mba_hour
WHERE price_area='N01' AND production_group='hydro'
LIMIT 5
""")
for r in rows:
    print(r)

cluster.shutdown()

Row(price_area='N01', production_group='hydro',
start_time=datetime.datetime(2020, 12, 31, 1, 0),
quantity_kwh=2507716.8)
Row(price_area='N01', production_group='hydro',
start_time=datetime.datetime(2021, 1, 1, 0, 0), quantity_kwh=123456.0)
Row(price_area='N01', production_group='hydro',
start_time=datetime.datetime(2021, 1, 1, 1, 0),
quantity_kwh=2864646.2)
Row(price_area='N01', production_group='hydro',
start_time=datetime.datetime(2021, 1, 2, 1, 0),
quantity_kwh=2960820.2)
Row(price_area='N01', production_group='hydro',
start_time=datetime.datetime(2021, 1, 3, 1, 0),
quantity_kwh=2862310.5)

from pathlib import Path
import pandas as pd

# Resolve project paths
notebooks_dir = Path.cwd().resolve()
project_root = notebooks_dir.parent
data_dir = project_root / "data"

csv_path = data_dir / "elhub_prod_by_group_hour_2021.csv"
print("Reading:", csv_path)

# Read & normalize
df = pd.read_csv(csv_path)

```

```

colmap =
{"priceArea": "price_area", "productionGroup": "production_group",
 "startTime": "start_time", "quantityKwh": "quantity_kwh"}
df.rename(columns={c: colmap.get(c, c) for c in df.columns},
inplace=True)

# Parse types
df["start_time"] = pd.to_datetime(df["start_time"], utc=True,
errors="coerce")
df["quantity_kwh"] = pd.to_numeric(df["quantity_kwh"],
errors="coerce")

# Clean
df =
df.dropna(subset=["price_area", "production_group", "start_time", "quantity_kwh"])
df =
df.sort_values(["price_area", "production_group", "start_time"]).reset_index(drop=True)

print(df.head(10))
print("Rows:", len(df), "| Areas:", df.price_area.nunique(), "| Groups:", df.production_group.unique())

```

Reading:

/Users/taom/Documents/IND320/IND320-project-work/data/elhub_prod_by_group_hour_2021.csv

	price_area	production_group	start_time	quantity_kwh
0	N01	hydro	2020-12-31 23:00:00+00:00	2507716.8
1	N01	hydro	2021-01-01 00:00:00+00:00	2494728.0
2	N01	hydro	2021-01-01 01:00:00+00:00	2486777.5
3	N01	hydro	2021-01-01 02:00:00+00:00	2461176.0
4	N01	hydro	2021-01-01 03:00:00+00:00	2466969.2
5	N01	hydro	2021-01-01 04:00:00+00:00	2467460.0
6	N01	hydro	2021-01-01 05:00:00+00:00	2482320.8
7	N01	hydro	2021-01-01 06:00:00+00:00	2509533.0
8	N01	hydro	2021-01-01 07:00:00+00:00	2550758.2
9	N01	hydro	2021-01-01 08:00:00+00:00	2693111.0

Rows: 215353 | Areas: 5 | Groups: ['hydro' 'other' 'solar' 'thermal' 'wind']

```
import matplotlib.pyplot as plt
```

fixed colors per group (simple dict)

```

GROUP_COLORS = {
    "hydro": "#4E79A7",
    "wind": "#59A14F",
    "solar": "#EDC948",
    "thermal": "#E15759",

```

```

    "nuclear": "#B07AA1",
    "other":    "#BAB0AC",
}

def plot_pie_year(price_area: str, year_label="2021"):
    # sum by group
    totals = (df[df["price_area"] == price_area]
              .groupby("production_group", as_index=False)
              ["quantity_kwh"].sum()
              .sort_values("quantity_kwh", ascending=False))
    if totals.empty:
        print(f"No data for price_area={price_area}")
        return

    labels = totals["production_group"].tolist()
    sizes  = totals["quantity_kwh"].tolist()
    colors = [GROUP_COLORS.get(g, "#999999") for g in labels]

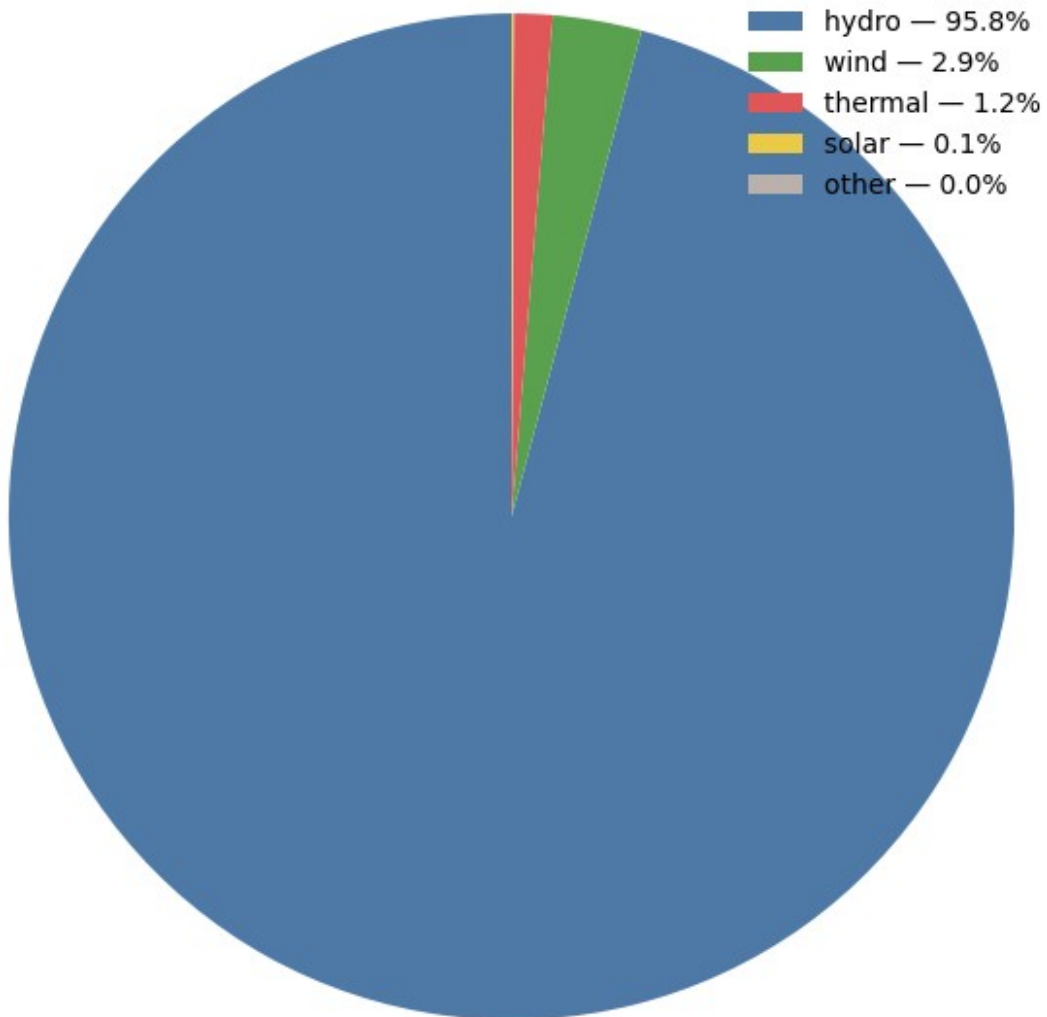
    # simple pie + legend (no cluttered slice labels)
    plt.figure(figsize=(6,6))
    plt.pie(sizes, colors=colors, startangle=90)
    plt.title(f"Total production in {year_label} - {price_area}")
    plt.axis("equal")

    # simple legend showing group + percent
    total = sum(sizes)
    legend_labels = [f"{g} - {100*s/total:.1f}%" for g, s in
zip(labels, sizes)]
    plt.legend(legend_labels, loc="best", frameon=False)
    plt.tight_layout()
    plt.show()

plot_pie_year("N01")

```

Total production in 2021 - NO1



```
import matplotlib.pyplot as plt

# reuse or define once (same colors as your pies/bars)
GROUP_COLORS = {
    "hydro": "#4E79A7",
    "wind": "#59A14F",
    "solar": "#EDC948",
    "thermal": "#E15759",
    "nuclear": "#B07AA1",
    "other": "#BAB0AC",
}

def plot_lines_month(price_area: str, year: int = 2021, month: int = 1):
    # filter
```

```

mdf = df[(df["price_area"] == price_area) &
          (df["start_time"].dt.year == year) &
          (df["start_time"].dt.month == month)].copy()
if mdf.empty:
    print(f"No data for {price_area} {year}-{month:02d}")
    return

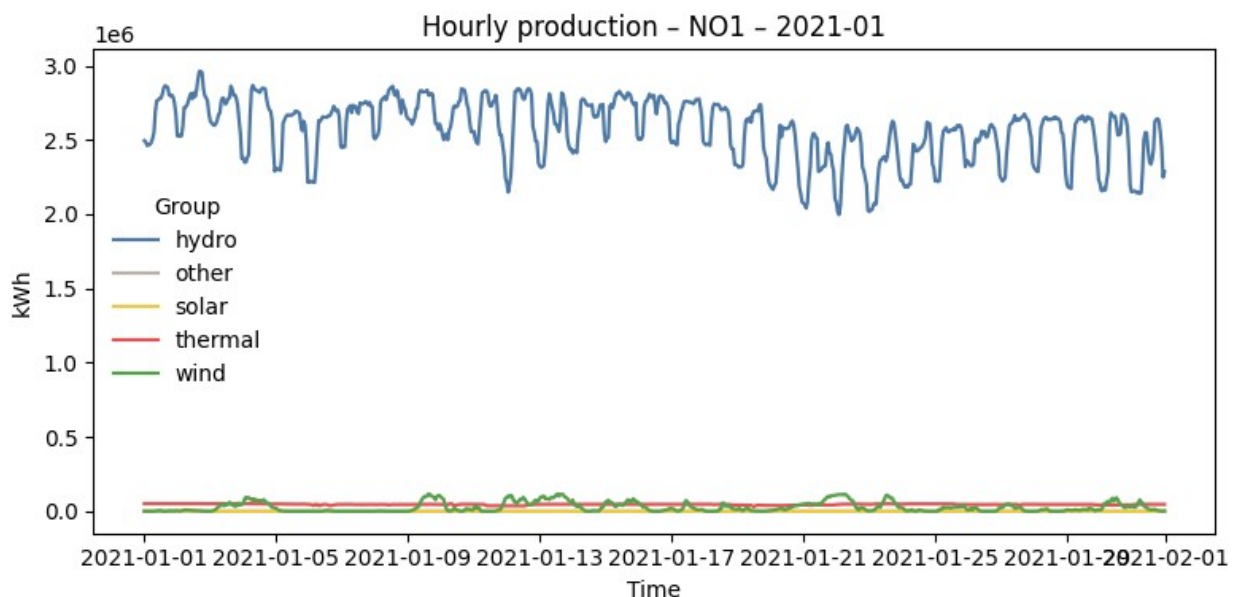
# one line per group
pivot = (mdf.pivot_table(index="start_time",
                          columns="production_group",
                          values="quantity_kwh",
                          aggfunc="sum")
          .sort_index())

plt.figure(figsize=(8, 4))
for col in pivot.columns:
    plt.plot(pivot.index, pivot[col], label=col,
             linewidth=1.6, color=GROUP_COLORS.get(col,
"#999999"))

plt.title(f"Hourly production - {price_area} - {year}-{month:02d}")
plt.xlabel("Time")
plt.ylabel("kWh")
plt.legend(title="Group", frameon=False)
plt.tight_layout()
plt.show()

plot_lines_month("N01", 2021, 1)

```



```

from pymongo import MongoClient
from pymongo.errors import PyMongoError

# Password will be rotated, so hardcoding it here for now
MONGO_URI =
"mongodb+srv://Taofik29:bcR3sF4Cs48ucLSx@cluster007.kkmkf1u.mongodb.net/?retryWrites=true&w=majority&appName=Cluster007&authSource=admin"
DB_NAME = "ind320"
COLL_HOURLY = "prod_hour"
COLL_YEARLY = "prod_year_totals"

try:
    client = MongoClient(MONGO_URI, serverSelectionTimeoutMS=10000)
    print("server_info:", client.server_info().get("version", "?")) #
    forces connect
    client.admin.command("ping")
    print("[] Connected & authenticated")
except PyMongoError as e:
    print("[] Mongo error:", e)
finally:
    try: client.close()
    except: pass

server_info: 8.0.15
[] Connected & authenticated

from pathlib import Path
import pandas as pd
from pymongo import MongoClient, ASCENDING, UpdateOne

# Load CSV from project data/
data_dir = Path.cwd().resolve().parent / "data"
csv_path = data_dir / "elhub_prod_by_group_hour_2021.csv"
df = pd.read_csv(csv_path)

# Normalize + types
colmap =
{"priceArea": "price_area", "productionGroup": "production_group",
 "startTime": "start_time", "quantityKwh": "quantity_kwh"}
df.rename(columns={c: colmap.get(c, c) for c in df.columns},
inplace=True)
df["start_time"] = pd.to_datetime(df["start_time"], utc=True,
errors="coerce")
df["quantity_kwh"] = pd.to_numeric(df["quantity_kwh"],
errors="coerce")
df =
df.dropna(subset=["price_area", "production_group", "start_time", "quantity_kwh"])

```



```

# Connect & quick ping
client = MongoClient(MONGO_URI)
client.admin.command("ping")
db = client[DB_NAME]
hour_col = db[COLL_HOURLY]
year_col = db[COLL_YEARLY]

# Indexes
hour_col.create_index(
    [("price_area", ASCENDING), ("production_group", ASCENDING),
    ("start_time", ASCENDING)],
    unique=True, name="ux_area_group_time"
)
hour_col.create_index([("price_area", ASCENDING), ("start_time",
ASCENDING)], name="ix_area_time")

# Upsert hourly in batches
ops, BATCH = [], 5000
for r in df.itertuples(index=False):
    key = {
        "price_area": r.price_area,
        "production_group": r.production_group,
        "start_time": pd.to_datetime(r.start_time).to_pydatetime(),
    }
    doc = {**key, "quantity_kwh": float(r.quantity_kwh)}
    ops.append(UpdateOne(key, {"$set": doc}, upsert=True))
    if len(ops) >= BATCH:
        hour_col.bulk_write(ops, ordered=False); ops = []
if ops: hour_col.bulk_write(ops, ordered=False)
print("□ Hourly upserts complete")

# Yearly totals for fast pie charts
totals = (df.groupby(["price_area", "production_group"],
as_index=False)["quantity_kwh"].sum()
        .rename(columns={"quantity_kwh": "total_kwh_2021"}))
year_col.drop()
if len(totals):
    year_col.insert_many(totals.to_dict(orient="records"))
    year_col.create_index([("price_area", ASCENDING)], name="ix_area")
    print("□ Yearly totals inserted:", len(totals))

client.close()

□ Hourly upserts complete
□ Yearly totals inserted: 25

```