IND320 part 4, Elhub, Spark, Cassandra, MongoDB

Imports and base config

In [1]:
```
import time, requests, pandas as pd, numpy as np
from datetime import datetime, timedelta, timezone
import matplotlib.pyplot as plt
from IPython.display import display
from zoneinfo import ZoneInfo

OSLO = ZoneInfo("Europe/Oslo")

def month_bounds_oslo(year: int, month: int):
    start = pd.Timestamp(year=year, month=month, day=1, tz=OSLO)
    end = (start + pd.offsets.MonthEnd(1)).replace(hour=23, minute=59, se
    return start, end

def iso_for_url(dt: pd.Timestamp) -> str:
    s = dt.strftime("%Y-%m-%dT%H:%M:%S%z")
    s = s[:-2] + ":" + s[-2:]
    return s.replace(":", "%3A").replace("+", "%2B")

BASE   = "https://api.elhub.no/energy-data/v0"
ENTITY = "price-areas"

DATASET_PROD = "PRODUCTION_PER_GROUP_MBA_HOUR"
DATASET_CONS = "CONSUMPTION_PER_GROUP_MBA_HOUR"

VALID_AREAS = {"NO1","NO2","NO3","NO4","NO5"}

print("Config OK")
```

```
Config OK
```

Fetch helpers with retry, monthly first, weekly fallback

In [2]:
```
def fetch_month(year: int, month: int, dataset: str):
    s, e = month_bounds_oslo(year, month)
    url = f"{BASE}/{ENTITY}?dataset={dataset}&startDate={iso_for_url(s)}&
    for attempt in range(4):
        try:
            r = requests.get(url, timeout=60, headers={"Accept": "applica
            if r.status_code == 429:
                time.sleep(1.5 * (attempt + 1)); continue
            r.raise_for_status()
            js = r.json()
            return js if isinstance(js, dict) and js.get("data") else Non
        except Exception:
            time.sleep(1.5 * (attempt + 1))
    return None
```

```
def fetch_weekly(year: int, month: int, dataset: str):
    s_month, e_month = month_bounds_oslo(year, month)
    s = s_month
    parts = []
    while s <= e_month:
        e = min(s + pd.Timedelta(days=6, hours=23, minutes=59, seconds=59
        url = f"{BASE}/{ENTITY}?dataset={dataset}&startDate={iso_for_url(
        ok = False
        for attempt in range(3):
            try:
                r = requests.get(url, timeout=60, headers={"Accept": "app
                if r.status_code == 429:
                    time.sleep(1.5 * (attempt + 1)); continue
                r.raise_for_status()
                js = r.json()
                if isinstance(js, dict) and js.get("data"):
                    parts.append(js)
                ok = True; break
            except Exception:
                time.sleep(1.5 * (attempt + 1))
        if not ok:
            print(f"Weekly fetch failed for window {s} to {e}")
        s = e + pd.Timedelta(seconds=1)
    return parts
```

Generic normalizer for production and consumption

In [3]:
```
def _norm_area(raw):
    if not raw: return None
    s = str(raw).strip().upper()
    if s in VALID_AREAS: return s
    name_map = {"OSLO":"NO1","KRISTIANSAND":"NO2","TRONDHEIM":"NO3","TROM
    if s in name_map: return name_map[s]
    import re
    m = re.search(r"NO\s*([1-5])", s)
    return f"NO{m.group(1)}" if m else None

def _extract_qty(rec: dict):
    for key in ("quantityKwh", "quantity", "kWh", "quantity_kwh"):
        if key in rec and rec[key] is not None:
            v = rec[key]
            return v.get("value") if isinstance(v, dict) and "value" in v

    return None

def _clean_group(g):
    if g is None: return None
    base = str(g).strip().lower()
    if base in {"hydro","hydropower","water"}:  return "hydro"
    if base in {"wind","windpower"}:            return "wind"
    if base in {"solar","pv"}:                  return "solar"
    if base in {"thermal","fossil","gas","coal"}: return "thermal"
    if base in {"other","misc"}:                return "other"
    return base

def normalize_payload_generic(js: dict, year_from: int, year_to_excl: int
    array_keys = ["productionPerGroupMbaHour","consumptionPerGroupMbaHour
    group_keys = ["productionGroup","consumptionGroup","group"]
    rows = []
    data_list = js.get("data", []) if isinstance(js, dict) else []
    for item in data_list:
        attrs = item.get("attributes", {}) or {}
        parent_area = _norm_area(attrs.get("name") or attrs.get("eic") or

        seq = None
        for k in array_keys:
            if k in attrs and isinstance(attrs[k], list):
                seq = attrs[k] or []
                break
        if seq is None:
            continue

        for rec in seq:
            pa = _norm_area(rec.get("priceArea")) or parent_area
            grp = None
            for gk in group_keys:
                if gk in rec and rec[gk] is not None:
                    grp = _clean_group(rec[gk]); break
            st  = rec.get("startTime") or rec.get("start")
            qty = _extract_qty(rec)
            if not pa or pa not in VALID_AREAS or not grp or not st:
                continue
            rows.append((pa, grp, st, qty))

    df = pd.DataFrame(rows, columns=["priceArea","group","startTime","qua
    df["startTime"]   = pd.to_datetime(df["startTime"], utc=True, errors=
    df["quantityKwh"] = pd.to_numeric(df["quantityKwh"], errors="coerce")
    df = df.dropna(subset=["startTime"])
    df = df[(df["startTime"] >= f"{year_from}-01-01") & (df["startTime"]
    return df.reset_index(drop=True)
```

Merge utility, gather monthly and weekly parts into one JSON per dataset

In [4]:
```
def fetch_merge_year_span(year_from: int, year_to_incl: int, dataset: str
    parts = []
    for y in range(year_from, year_to_incl + 1):
        for m in range(1, 13):
            js = fetch_month(y, m, dataset)
            if js is not None:
                parts.append(js)
            else:
                parts.extend(fetch_weekly(y, m, dataset))

    merged = {"data": []}
    by_area = {}
    array_key = "productionPerGroupMbaHour" if dataset == DATASET_PROD el

    for payload in parts:
        for area in payload.get("data", []):
            attrs = area.get("attributes", {}) or {}
            key = attrs.get("name") or attrs.get("eic")
            if not key:
                continue

            seq = attrs.get(array_key)
            if not isinstance(seq, list):
                seq = attrs.get("productionPerGroupMbaHour") or attrs.get
            if not isinstance(seq, list):
                continue

            if key not in by_area:
                by_area[key] = {"attributes": {"name": key, array_key: li
            else:
                by_area[key]["attributes"][array_key].extend(seq)

    merged["data"] = [{"attributes": v["attributes"]} for v in by_area.va
    return merged
```

Production, fetch 2022 to 2024 and normalize

In [5]:
```
merged_prod_22_24 = fetch_merge_year_span(2022, 2024, DATASET_PROD)
prod_22_24 = normalize_payload_generic(merged_prod_22_24, 2022, 2025)

print(len(prod_22_24), "rows")
print("Areas:", sorted(prod_22_24.priceArea.unique()))
print("Groups:", sorted(prod_22_24["group"].unique()))
prod_22_24.head()
```

```
657575 rows
Areas: ['NO1', 'NO2', 'NO3', 'NO4', 'NO5']
Groups: ['hydro', 'other', 'solar', 'thermal', 'wind']
```

Out[5]:

	priceArea	group	startTime	quantityKwh
0	NO1	hydro	2022-01-01 00:00:00+00:00	1246209.4
1	NO1	hydro	2022-01-01 01:00:00+00:00	1271757.0
2	NO1	hydro	2022-01-01 02:00:00+00:00	1204251.8
3	NO1	hydro	2022-01-01 03:00:00+00:00	1202086.9
4	NO1	hydro	2022-01-01 04:00:00+00:00	1235809.9

Sanity checks for production

In [6]:
```
def sanity(pdf: pd.DataFrame, label: str):
    assert set(sorted(pdf.priceArea.unique())) <= VALID_AREAS, f"Unexpect
    assert pdf["quantityKwh"].notna().all(), f"Found NaN in {label}.quant
    assert pdf["startTime"].notna().all(),  f"Found NaT in {label}.startT
    print(label, "OK, rows:", len(pdf))

sanity(prod_22_24, "production 2022 to 2024")

tmp = prod_22_24.copy()
tmp["year"] = tmp["startTime"].dt.year
display(pd.crosstab(tmp["priceArea"], tmp["year"]))
```

```
production 2022 to 2024 OK, rows: 657575
```

year	2022	2023	2024
priceArea			
NO1	43800	43800	43915
NO2	43800	43800	43915
NO3	43800	43800	43915
NO4	43800	43800	43915
NO5	43800	43800	43915

Spark setup for writing to Cassandra

In [7]:
```
# Spark setup for writing to Cassandra (throttled for single-node)

from pyspark.sql import SparkSession
import os

# Help Spark find the right Python
os.environ["PYSPARK_PYTHON"] = "python"
os.environ["PYSPARK_DRIVER_PYTHON"] = "python"

spark = (
    SparkSession.builder
    .appName("ind320-elhub-cassandra")
    # Cassandra connector for Spark 3.5.x and Scala 2.12
    .config("spark.jars.packages", "com.datastax.spark:spark-cassandra-co
    # Connection to local Cassandra
    .config("spark.cassandra.connection.host", "127.0.0.1")
    .config("spark.cassandra.connection.port", "9042")
    # Throttle writes so the single node does not die
    .config("spark.cassandra.output.consistency.level", "LOCAL_ONE")
    .config("spark.cassandra.output.batch.size.rows", "256")      # small
    .config("spark.cassandra.output.concurrent.writes", "2")      # few c
    .config("spark.cassandra.output.throughput_mb_per_sec", "2")  # cap t
    .getOrCreate()
)

print("Spark session ready:", spark.version)
```

```
25/11/28 16:57:56 WARN Utils: Your hostname, Youness-MacBook-Air.local res
olves to a loopback address: 127.0.0.1; using 192.168.0.6 instead (on inte
rface en0)
25/11/28 16:57:56 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to an
other address
Ivy Default Cache set to: /Users/youneshansen/.ivy2/cache
The jars for the packages stored in: /Users/youneshansen/.ivy2/jars
com.datastax.spark#spark-cassandra-connector_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-036a14b0
-9f53-4c6c-bf2f-c937e3f3366f;1.0
        confs: [default]
:: loading settings :: url = jar:file:/Users/youneshansen/Documents/ind32
0/ind320-yohan3351/.venv312/lib/python3.12/site-packages/pyspark/jars/ivy-
2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
```

```
        found com.datastax.spark#spark-cassandra-connector_2.12;3.5.1 in c
entral
        found com.datastax.spark#spark-cassandra-connector-driver_2.12;3.
5.1 in central
        found org.scala-lang.modules#scala-collection-compat_2.12;2.11.0 i
n central
        found org.apache.cassandra#java-driver-core-shaded;4.18.1 in centr
al
        found com.datastax.oss#native-protocol;1.5.1 in central
        found com.datastax.oss#java-driver-shaded-guava;25.1-jre-graal-sub
-1 in central
        found com.typesafe#config;1.4.1 in central
        found org.slf4j#slf4j-api;1.7.26 in central
        found io.dropwizard.metrics#metrics-core;4.1.18 in central
        found org.hdrhistogram#HdrHistogram;2.1.12 in central
        found org.reactivestreams#reactive-streams;1.0.3 in central
        found org.apache.cassandra#java-driver-mapper-runtime;4.18.1 in ce
ntral
        found org.apache.cassandra#java-driver-query-builder;4.18.1 in cen
tral
        found org.apache.commons#commons-lang3;3.10 in central
        found com.thoughtworks.paranamer#paranamer;2.8 in central
        found org.scala-lang#scala-reflect;2.12.19 in central
:: resolution report :: resolve 334ms :: artifacts dl 10ms
        :: modules in use:
        com.datastax.oss#java-driver-shaded-guava;25.1-jre-graal-sub-1 fro
m central in [default]
        com.datastax.oss#native-protocol;1.5.1 from central in [default]
        com.datastax.spark#spark-cassandra-connector-driver_2.12;3.5.1 fro
m central in [default]
        com.datastax.spark#spark-cassandra-connector_2.12;3.5.1 from centr
al in [default]
        com.thoughtworks.paranamer#paranamer;2.8 from central in [default]
        com.typesafe#config;1.4.1 from central in [default]
        io.dropwizard.metrics#metrics-core;4.1.18 from central in [defaul
t]
        org.apache.cassandra#java-driver-core-shaded;4.18.1 from central i
n [default]
        org.apache.cassandra#java-driver-mapper-runtime;4.18.1 from centra
l in [default]
        org.apache.cassandra#java-driver-query-builder;4.18.1 from central
in [default]
        org.apache.commons#commons-lang3;3.10 from central in [default]
        org.hdrhistogram#HdrHistogram;2.1.12 from central in [default]
        org.reactivestreams#reactive-streams;1.0.3 from central in [defaul
t]
        org.scala-lang#scala-reflect;2.12.19 from central in [default]
        org.scala-lang.modules#scala-collection-compat_2.12;2.11.0 from ce
ntral in [default]
        org.slf4j#slf4j-api;1.7.26 from central in [default]
        ------------------------------------------------------------------
---
        |                  |            modules            ||   artifacts
|
        |       conf       | number| search|dwnlded|evicted|| number|dwnld
ed|
        ------------------------------------------------------------------
---
        |      default     |   16  |   0   |   0   |   0   ||   16  |   0
|
        ----------------------------------------------------------------------
---
:: retrieving :: org.apache.spark#spark-submit-parent-036a14b0-9f53-4c6c-b
f2f-c937e3f3366f
        confs: [default]
        0 artifacts copied, 16 already retrieved (0kB/7ms)
25/11/28 16:57:57 WARN NativeCodeLoader: Unable to load native-hadoop libr
ary for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setL
ogLevel(newLevel).
Spark session ready: 3.5.1
```

Write production 2022 to 2024 to Cassandra

In [8]:
```
## Write production 2022 to 2024 to Cassandra

# Cassandra, ensure table exists, then write with Spark

from cassandra.cluster import Cluster
import pandas as pd

CASS_HOST = "127.0.0.1"
CASS_PORT = 9042
KEYSPACE  = "power"
TABLE_PROD = "production_2022_2024"

print("Connecting to Cassandra and preparing production table...")
cluster = Cluster([CASS_HOST], port=CASS_PORT, protocol_version=5)
session = cluster.connect()

# Keyspace
session.execute(f"""
CREATE KEYSPACE IF NOT EXISTS {KEYSPACE}
WITH replication = {{'class': 'SimpleStrategy', 'replication_factor': 1}}
""")
session.set_keyspace(KEYSPACE)

# Base table, including year column
session.execute(f"""
CREATE TABLE IF NOT EXISTS {TABLE_PROD} (
  pricearea text,
  productiongroup text,
  starttime timestamp,
  quantitykwh double,
  year int,
  PRIMARY KEY ((pricearea, productiongroup), starttime)
) WITH CLUSTERING ORDER BY (starttime ASC)
""")

# If table existed earlier without year, try to add it, ignore error if i
try:
    session.execute(f"ALTER TABLE {TABLE_PROD} ADD year int")
except Exception:
    pass

# Prepare pandas DataFrame for Spark
prod = prod_22_24.copy()
prod["year"] = prod["startTime"].dt.year.astype(int)
```

```
prod = prod.rename(columns={
    "priceArea": "pricearea",
    "group": "productiongroup",
    "startTime": "starttime",
    "quantityKwh": "quantitykwh"
})
# Remove timezone, Cassandra timestamp is naive
prod["starttime"] = pd.to_datetime(prod["starttime"], utc=True, errors="c

# Create Spark DataFrame and write to Cassandra (throttled)
prod_sdf = spark.createDataFrame(
    prod[["pricearea", "productiongroup", "starttime", "quantitykwh", "ye
)

# Reduce the number of Spark partitions to avoid hammering Cassandra
prod_sdf = prod_sdf.repartition(8, "pricearea", "productiongroup")

(
    prod_sdf.write
    .format("org.apache.spark.sql.cassandra")
    .mode("append")
    .options(table=TABLE_PROD, keyspace=KEYSPACE)
    .save()
)

# Quick validation with the driver
row_count = session.execute(f"SELECT count(*) FROM {TABLE_PROD}").one().c
print(f"Spark wrote {row_count:,} rows to {KEYSPACE}.{TABLE_PROD}")

cluster.shutdown()
print("Cassandra cluster connection closed for production.")
```

```
Connecting to Cassandra and preparing production table...
```

```
25/11/28 16:58:22 WARN DeprecatedConfigParameter: spark.cassandra.output.t
hroughput_mb_per_sec is deprecated (DSE 6.0.0) and has been automatically
replaced with parameter spark.cassandra.output.throughputMBPerSec.
25/11/28 16:58:22 WARN DeprecatedConfigParameter: spark.cassandra.output.t
hroughput_mb_per_sec is deprecated (DSE 6.0.0) and has been automatically
replaced with parameter spark.cassandra.output.throughputMBPerSec.
25/11/28 16:58:23 WARN DeprecatedConfigParameter: spark.cassandra.output.t
hroughput_mb_per_sec is deprecated (DSE 6.0.0) and has been automatically
replaced with parameter spark.cassandra.output.throughputMBPerSec.
25/11/28 16:58:24 WARN TaskSetManager: Stage 0 contains a task of very lar
ge size (2696 KiB). The maximum recommended task size is 1000 KiB.
```

```
Spark wrote 657,600 rows to power.production_2022_2024
Cassandra cluster connection closed for production.
```

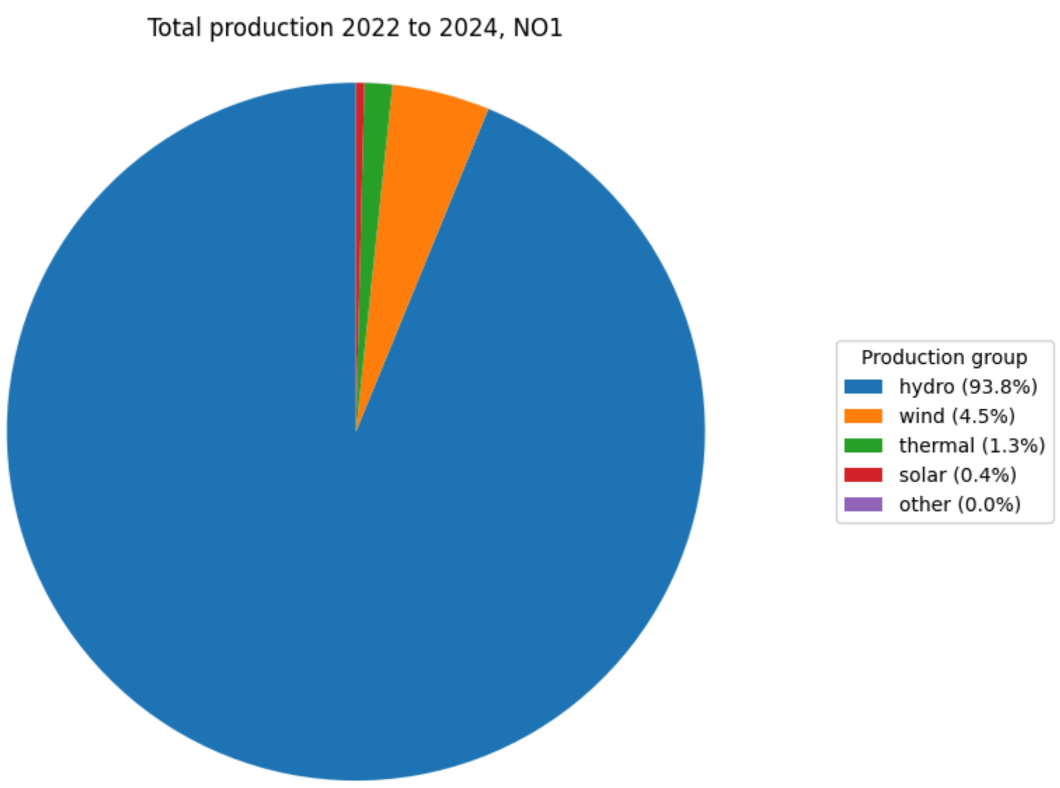
Read back for a quick audit and one plot

In [9]:

```
# Read back from Cassandra using Python driver and produce the same plot

from cassandra.cluster import Cluster
import pandas as pd
import matplotlib.pyplot as plt

cluster = Cluster(["127.0.0.1"], port=9042)
session = cluster.connect("power")
```

```
price_area = "NO1"

# Pull all rows for the chosen area 2022–2024
rows = session.execute("""
SELECT pricearea, productiongroup, year, starttime, quantitykwh
FROM production_2022_2024
WHERE pricearea = %s ALLOW FILTERING
""", (price_area,))

pdf = pd.DataFrame(
    [(r.pricearea, r.productiongroup, r.year, pd.to_datetime(r.starttime)
     for r in rows],
    columns=["pricearea","productiongroup","year","starttime","quantitykw
)

cluster.shutdown()

print("Rows read from Cassandra:", len(pdf))

totals_pdf = (
    pdf.groupby("productiongroup", as_index=False)["quantitykwh"]
       .sum()
       .rename(columns={"quantitykwh":"totalKwh"})
       .sort_values("totalKwh", ascending=False)
       .reset_index(drop=True)
)

grand = float(totals_pdf["totalKwh"].sum()) if not totals_pdf.empty else
totals_pdf["pct"] = (100.0 * totals_pdf["totalKwh"] / grand) if grand > 0

if totals_pdf.empty or grand == 0.0:
    print(f"No data to plot for {price_area}")
else:
    fig, ax = plt.subplots(figsize=(8.5, 6))
    sizes = totals_pdf["totalKwh"].to_numpy(dtype=float)
    wedges = ax.pie(sizes, labels=None, autopct=None, startangle=90, pctd
    ax.axis("equal")
    legend_labels = [f"{g} ({p:.1f}%)" for g, p in zip(totals_pdf["produc
    ax.set_title(f"Total production 2022 to 2024, {price_area}")
    ax.legend(wedges, legend_labels, title="Production group", loc="cente
    plt.tight_layout(); plt.show()
```

```
Rows read from Cassandra: 131520
```



MongoDB settings for task 4

```
In [10]: ## MongoDB settings for task 4 - production upsert only

import os, time
import pandas as pd
from pymongo import MongoClient, UpdateOne
import certifi

MONGODB_URI = os.getenv(
    "MONGODB_URI",
    "mongodb+srv://younes_user:Younes2002@oblig.okycarf.mongodb.net/?retr
)
DB_NAME = os.getenv("MONGODB_DB", "power")
COLL_PROD_21_24 = os.getenv("MONGODB_COLL_PROD", "production_2021_2024")

def bulk_upsert(df: pd.DataFrame, coll_name: str, key_fields: list[str],
    cli = MongoClient(MONGODB_URI, tlsCAFile=certifi.where(), serverSelec
    coll = cli[DB_NAME][coll_name]
    coll.create_index([(k, 1) for k in key_fields], unique=True, name="un
    ops, done = [], 0
    t0 = time.perf_counter()

    for doc in df.to_dict(orient="records"):
        key = {k: doc[k] for k in key_fields}
        ops.append(UpdateOne(key, {"$set": doc}, upsert=True))
        if len(ops) >= batch_size:
            coll.bulk_write(ops, ordered=False)
            done += len(ops); ops = []
            rate = done / max(1e-6, (time.perf_counter() - t0))
            print(f"Upserted {done:,}/{len(df):,} docs, ~{rate:,.0f}/s")

    if ops:
        coll.bulk_write(ops, ordered=False)
        done += len(ops)

    cli.close()
    print(f"Finished {coll_name}: {done:,} upserts")

# 2021 from Cassandra using Python driver
from cassandra.cluster import Cluster
cluster = Cluster(["127.0.0.1"], port=9042, protocol_version=5)
sess = cluster.connect("power")
rows_21 = sess.execute("SELECT pricearea, productiongroup, starttime, qua
prod_21 = pd.DataFrame(
    [(r.pricearea, r.productiongroup, pd.to_datetime(r.starttime), float(
     for r in rows_21],
    columns=["priceArea","group","startTime","quantityKwh"]
)
cluster.shutdown()

# merge 2021 with 2022–2024 you already built above
prod_21_24 = pd.concat([prod_21, prod_22_24], ignore_index=True)

# upsert to Mongo
bulk_upsert(
    prod_21_24[["priceArea","group","startTime","quantityKwh"]],
    COLL_PROD_21_24,
    ["priceArea","group","startTime"]
)
```

```
Upserted 5,000/872,903 docs, ~584/s
Upserted 10,000/872,903 docs, ~839/s
Upserted 15,000/872,903 docs, ~1,066/s
Upserted 20,000/872,903 docs, ~1,256/s
Upserted 25,000/872,903 docs, ~1,390/s
Upserted 30,000/872,903 docs, ~1,507/s
Upserted 35,000/872,903 docs, ~1,596/s
Upserted 40,000/872,903 docs, ~1,673/s
Upserted 45,000/872,903 docs, ~1,727/s
Upserted 50,000/872,903 docs, ~1,518/s
Upserted 55,000/872,903 docs, ~1,435/s
Upserted 60,000/872,903 docs, ~1,355/s
Upserted 65,000/872,903 docs, ~1,283/s
Upserted 70,000/872,903 docs, ~1,219/s
Upserted 75,000/872,903 docs, ~1,253/s
Upserted 80,000/872,903 docs, ~1,295/s
Upserted 85,000/872,903 docs, ~1,331/s
Upserted 90,000/872,903 docs, ~1,368/s
Upserted 95,000/872,903 docs, ~1,400/s
Upserted 100,000/872,903 docs, ~1,337/s
Upserted 105,000/872,903 docs, ~1,292/s
Upserted 110,000/872,903 docs, ~1,254/s
Upserted 115,000/872,903 docs, ~1,247/s
Upserted 120,000/872,903 docs, ~1,274/s
Upserted 125,000/872,903 docs, ~1,248/s
Upserted 130,000/872,903 docs, ~1,222/s
Upserted 135,000/872,903 docs, ~1,198/s
Upserted 140,000/872,903 docs, ~1,176/s
Upserted 145,000/872,903 docs, ~1,183/s
Upserted 150,000/872,903 docs, ~1,205/s
Upserted 155,000/872,903 docs, ~1,226/s
Upserted 160,000/872,903 docs, ~1,198/s
Upserted 165,000/872,903 docs, ~1,179/s
Upserted 170,000/872,903 docs, ~1,161/s
Upserted 175,000/872,903 docs, ~1,145/s
Upserted 180,000/872,903 docs, ~1,130/s
Upserted 185,000/872,903 docs, ~1,113/s
Upserted 190,000/872,903 docs, ~1,121/s
Upserted 195,000/872,903 docs, ~1,136/s
Upserted 200,000/872,903 docs, ~1,152/s
Upserted 205,000/872,903 docs, ~1,168/s
Upserted 210,000/872,903 docs, ~1,182/s
Upserted 215,000/872,903 docs, ~1,197/s
Upserted 220,000/872,903 docs, ~1,211/s
Upserted 225,000/872,903 docs, ~1,226/s
Upserted 230,000/872,903 docs, ~1,207/s
Upserted 235,000/872,903 docs, ~1,193/s
Upserted 240,000/872,903 docs, ~1,180/s
Upserted 245,000/872,903 docs, ~1,167/s
Upserted 250,000/872,903 docs, ~1,171/s
Upserted 255,000/872,903 docs, ~1,183/s
Upserted 260,000/872,903 docs, ~1,196/s
Upserted 265,000/872,903 docs, ~1,207/s
Upserted 270,000/872,903 docs, ~1,219/s
Upserted 275,000/872,903 docs, ~1,231/s
Upserted 280,000/872,903 docs, ~1,241/s
Upserted 285,000/872,903 docs, ~1,253/s
Upserted 290,000/872,903 docs, ~1,237/s
Upserted 295,000/872,903 docs, ~1,235/s
Upserted 300,000/872,903 docs, ~1,245/s
Upserted 305,000/872,903 docs, ~1,235/s
Upserted 310,000/872,903 docs, ~1,224/s
Upserted 315,000/872,903 docs, ~1,213/s
Upserted 320,000/872,903 docs, ~1,202/s
Upserted 325,000/872,903 docs, ~1,191/s
Upserted 330,000/872,903 docs, ~1,181/s
Upserted 335,000/872,903 docs, ~1,185/s
Upserted 340,000/872,903 docs, ~1,194/s
Upserted 345,000/872,903 docs, ~1,203/s
Upserted 350,000/872,903 docs, ~1,212/s
Upserted 355,000/872,903 docs, ~1,221/s
Upserted 360,000/872,903 docs, ~1,229/s
Upserted 365,000/872,903 docs, ~1,239/s
Upserted 370,000/872,903 docs, ~1,247/s
Upserted 375,000/872,903 docs, ~1,255/s
Upserted 380,000/872,903 docs, ~1,243/s
Upserted 385,000/872,903 docs, ~1,233/s
Upserted 390,000/872,903 docs, ~1,228/s
Upserted 395,000/872,903 docs, ~1,228/s
Upserted 400,000/872,903 docs, ~1,236/s
Upserted 405,000/872,903 docs, ~1,244/s
Upserted 410,000/872,903 docs, ~1,251/s
Upserted 415,000/872,903 docs, ~1,259/s
Upserted 420,000/872,903 docs, ~1,266/s
Upserted 425,000/872,903 docs, ~1,274/s
Upserted 430,000/872,903 docs, ~1,281/s
Upserted 435,000/872,903 docs, ~1,289/s
Upserted 440,000/872,903 docs, ~1,295/s
Upserted 445,000/872,903 docs, ~1,302/s
Upserted 450,000/872,903 docs, ~1,310/s
Upserted 455,000/872,903 docs, ~1,316/s
Upserted 460,000/872,903 docs, ~1,304/s
Upserted 465,000/872,903 docs, ~1,299/s
Upserted 470,000/872,903 docs, ~1,291/s
Upserted 475,000/872,903 docs, ~1,285/s
Upserted 480,000/872,903 docs, ~1,278/s
Upserted 485,000/872,903 docs, ~1,273/s
Upserted 490,000/872,903 docs, ~1,265/s
Upserted 495,000/872,903 docs, ~1,257/s
Upserted 500,000/872,903 docs, ~1,249/s
Upserted 505,000/872,903 docs, ~1,242/s
Upserted 510,000/872,903 docs, ~1,235/s
Upserted 515,000/872,903 docs, ~1,228/s
Upserted 520,000/872,903 docs, ~1,220/s
Upserted 525,000/872,903 docs, ~1,214/s
Upserted 530,000/872,903 docs, ~1,207/s
Upserted 535,000/872,903 docs, ~1,201/s
Upserted 540,000/872,903 docs, ~1,206/s
Upserted 545,000/872,903 docs, ~1,211/s
Upserted 550,000/872,903 docs, ~1,217/s
Upserted 555,000/872,903 docs, ~1,223/s
Upserted 560,000/872,903 docs, ~1,228/s
Upserted 565,000/872,903 docs, ~1,220/s
Upserted 570,000/872,903 docs, ~1,215/s
Upserted 575,000/872,903 docs, ~1,208/s
Upserted 580,000/872,903 docs, ~1,202/s
Upserted 585,000/872,903 docs, ~1,197/s
Upserted 590,000/872,903 docs, ~1,191/s
Upserted 595,000/872,903 docs, ~1,186/s
Upserted 600,000/872,903 docs, ~1,181/s
Upserted 605,000/872,903 docs, ~1,185/s
Upserted 610,000/872,903 docs, ~1,190/s
Upserted 615,000/872,903 docs, ~1,184/s
Upserted 620,000/872,903 docs, ~1,181/s
Upserted 625,000/872,903 docs, ~1,177/s
Upserted 630,000/872,903 docs, ~1,175/s
Upserted 635,000/872,903 docs, ~1,171/s
Upserted 640,000/872,903 docs, ~1,168/s
Upserted 645,000/872,903 docs, ~1,164/s
Upserted 650,000/872,903 docs, ~1,158/s
Upserted 655,000/872,903 docs, ~1,153/s
Upserted 660,000/872,903 docs, ~1,154/s
Upserted 665,000/872,903 docs, ~1,157/s
Upserted 670,000/872,903 docs, ~1,154/s
Upserted 675,000/872,903 docs, ~1,150/s
Upserted 680,000/872,903 docs, ~1,146/s
Upserted 685,000/872,903 docs, ~1,146/s
Upserted 690,000/872,903 docs, ~1,150/s
Upserted 695,000/872,903 docs, ~1,147/s
Upserted 700,000/872,903 docs, ~1,143/s
Upserted 705,000/872,903 docs, ~1,139/s
Upserted 710,000/872,903 docs, ~1,136/s
Upserted 715,000/872,903 docs, ~1,132/s
Upserted 720,000/872,903 docs, ~1,136/s
Upserted 725,000/872,903 docs, ~1,140/s
Upserted 730,000/872,903 docs, ~1,135/s
Upserted 735,000/872,903 docs, ~1,135/s
Upserted 740,000/872,903 docs, ~1,139/s
Upserted 745,000/872,903 docs, ~1,144/s
Upserted 750,000/872,903 docs, ~1,148/s
Upserted 755,000/872,903 docs, ~1,143/s
Upserted 760,000/872,903 docs, ~1,139/s
Upserted 765,000/872,903 docs, ~1,139/s
Upserted 770,000/872,903 docs, ~1,143/s
Upserted 775,000/872,903 docs, ~1,140/s
Upserted 780,000/872,903 docs, ~1,137/s
Upserted 785,000/872,903 docs, ~1,133/s
Upserted 790,000/872,903 docs, ~1,130/s
Upserted 795,000/872,903 docs, ~1,127/s
Upserted 800,000/872,903 docs, ~1,130/s
Upserted 805,000/872,903 docs, ~1,134/s
Upserted 810,000/872,903 docs, ~1,137/s
Upserted 815,000/872,903 docs, ~1,141/s
Upserted 820,000/872,903 docs, ~1,145/s
Upserted 825,000/872,903 docs, ~1,149/s
Upserted 830,000/872,903 docs, ~1,152/s
Upserted 835,000/872,903 docs, ~1,148/s
Upserted 840,000/872,903 docs, ~1,151/s
Upserted 845,000/872,903 docs, ~1,149/s
Upserted 850,000/872,903 docs, ~1,145/s
Upserted 855,000/872,903 docs, ~1,142/s
Upserted 860,000/872,903 docs, ~1,139/s
Upserted 865,000/872,903 docs, ~1,136/s
Upserted 870,000/872,903 docs, ~1,133/s
Finished production_2021_2024: 872,903 upserts
```

Consumption, fetch 2021 to 2024 and normalize

In [11]:
```
merged_cons_21_24 = fetch_merge_year_span(2021, 2024, DATASET_CONS)
cons_21_24 = normalize_payload_generic(merged_cons_21_24, 2021, 2025)

print(len(cons_21_24), "rows")
print("Areas:", sorted(cons_21_24.priceArea.unique()))
print("Groups:", sorted(cons_21_24["group"].unique()))
cons_21_24.head()
```

```
876575 rows
Areas: ['NO1', 'NO2', 'NO3', 'NO4', 'NO5']
Groups: ['cabin', 'household', 'primary', 'secondary', 'tertiary']
```

Out[11]:

	priceArea	group	startTime	quantityKwh
0	NO1	cabin	2021-01-01 00:00:00+00:00	171335.12
1	NO1	cabin	2021-01-01 01:00:00+00:00	164912.02
2	NO1	cabin	2021-01-01 02:00:00+00:00	160265.77
3	NO1	cabin	2021-01-01 03:00:00+00:00	159828.69
4	NO1	cabin	2021-01-01 04:00:00+00:00	160388.17

Sanity checks for consumption

In [12]:
```
sanity(cons_21_24, "consumption 2021 to 2024")

tmpc = cons_21_24.copy()
tmpc["year"] = tmpc["startTime"].dt.year
display(pd.crosstab(tmpc["priceArea"], tmpc["year"]))
```

```
consumption 2021 to 2024 OK, rows: 876575
```

year	2021	2022	2023	2024
priceArea				
NO1	43800	43800	43800	43915
NO2	43800	43800	43800	43915
NO3	43800	43800	43800	43915
NO4	43800	43800	43800	43915
NO5	43800	43800	43800	43915

Write consumption 2021 to 2024 to Cassandra

In [13]:
```
## Write consumption 2021 to 2024 to Cassandra

# Cassandra bulk upsert CONSUMPTION 2021 to 2024, now using Spark for the

from cassandra.cluster import Cluster
from cassandra.policies import TokenAwarePolicy, DCAwareRoundRobinPolicy
import pandas as pd

# Connect
```

```
cluster = Cluster(
    ["127.0.0.1"],
    port=9042,
    protocol_version=5,
    load_balancing_policy=TokenAwarePolicy(DCAwareRoundRobinPolicy(local_
)
session = cluster.connect()

# Keyspace
session.execute("""
CREATE KEYSPACE IF NOT EXISTS power
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1}
""")
session.set_keyspace("power")

# If table exists without 'year', drop and recreate with correct primary
cols = session.execute("""
SELECT column_name FROM system_schema.columns
WHERE keyspace_name = 'power' AND table_name = 'consumption_2021_2024'
""")
have_table = False
have_year = False
for r in cols:
    have_table = True
    if r.column_name == 'year':
        have_year = True

if have_table and not have_year:
    session.execute("DROP TABLE power.consumption_2021_2024")

# Create with desired schema (year in the partition key, same strategy as
session.execute("""
CREATE TABLE IF NOT EXISTS consumption_2021_2024 (
  pricearea text,
  consumptiongroup text,
  year int,
  starttime timestamp,
  quantitykwh double,
  PRIMARY KEY ((pricearea, consumptiongroup, year), starttime)
) WITH CLUSTERING ORDER BY (starttime ASC)
""")

# Normalize to match table
cons = cons_21_24.copy()
cons["year"] = cons["startTime"].dt.year.astype(int)
cons = cons.rename(columns={
    "priceArea":"pricearea",
    "group":"consumptiongroup",
    "startTime":"starttime",
    "quantityKwh":"quantitykwh"
})[["pricearea","consumptiongroup","year","starttime","quantitykwh"]]

# Types to native for Spark and Cassandra
cons["starttime"] = pd.to_datetime(cons["starttime"], utc=True, errors="c
cons["quantitykwh"] = pd.to_numeric(cons["quantitykwh"], errors="coerce")

# Create Spark DataFrame and write to Cassandra
cons_sdf = spark.createDataFrame(cons)

(
    cons_sdf.write
    .format("org.apache.spark.sql.cassandra")
    .mode("append")
    .options(table="consumption_2021_2024", keyspace="power")
    .save()
)

# Give Cassandra some time to flush the heavy Spark write before validati
import time
print("Waiting 20 seconds before validation query so Cassandra can finish
time.sleep(20)

# Increase client timeout for the validation query
session.default_timeout = 60.0

# Simple validation
row_count = session.execute("SELECT count(*) FROM power.consumption_2021_
print(f"Spark wrote {row_count:,} rows into power.consumption_2021_2024")

cluster.shutdown()
print("Cassandra cluster connection closed for consumption.")
```

```
25/11/28 17:14:10 WARN DeprecatedConfigParameter: spark.cassandra.output.t
hroughput_mb_per_sec is deprecated (DSE 6.0.0) and has been automatically
replaced with parameter spark.cassandra.output.throughputMBPerSec.
25/11/28 17:14:10 WARN DeprecatedConfigParameter: spark.cassandra.output.t
hroughput_mb_per_sec is deprecated (DSE 6.0.0) and has been automatically
replaced with parameter spark.cassandra.output.throughputMBPerSec.
25/11/28 17:14:10 WARN DeprecatedConfigParameter: spark.cassandra.output.t
hroughput_mb_per_sec is deprecated (DSE 6.0.0) and has been automatically
replaced with parameter spark.cassandra.output.throughputMBPerSec.
25/11/28 17:14:11 WARN TaskSetManager: Stage 3 contains a task of very lar
ge size (4357 KiB). The maximum recommended task size is 1000 KiB.
```

```
Waiting 20 seconds before validation query so Cassandra can finish write
s...
Spark wrote 876,675 rows into power.consumption_2021_2024
Cassandra cluster connection closed for consumption.
```

MongoDB - consumption upsert

In [14]:

```
## MongoDB – consumption upsert

import os, time
import pandas as pd
from pymongo import MongoClient, UpdateOne
import certifi

MONGODB_URI = os.getenv(
    "MONGODB_URI",
    "mongodb+srv://younes_user:Younes2002@oblig.okycarf.mongodb.net/?retr
)
DB_NAME = os.getenv("MONGODB_DB", "power")
COLL_CONS_21_24 = os.getenv("MONGODB_COLL_CONS", "consumption_2021_2024")

def bulk_upsert(df: pd.DataFrame, coll_name: str, key_fields: list[str],
    cli = MongoClient(MONGODB_URI, tlsCAFile=certifi.where(), serverSelec
    coll = cli[DB_NAME][coll_name]
    coll.create_index([(k, 1) for k in key_fields], unique=True, name="un
    ops, done = [], 0
    t0 = time.perf_counter()

    for doc in df.to_dict(orient="records"):
        key = {k: doc[k] for k in key_fields}
        ops.append(UpdateOne(key, {"$set": doc}, upsert=True))
        if len(ops) >= batch_size:
            coll.bulk_write(ops, ordered=False)
            done += len(ops); ops = []
            rate = done / max(1e-6, (time.perf_counter() - t0))
            print(f"Upserted {done:,}/{len(df):,} docs, ~{rate:,.0f}/s")

    if ops:
        coll.bulk_write(ops, ordered=False)
        done += len(ops)

    cli.close()
    print(f"Finished {coll_name}: {done:,} upserts")

# upsert consumption
bulk_upsert(
    cons_21_24[["priceArea","group","startTime","quantityKwh"]],
    COLL_CONS_21_24,
    ["priceArea","group","startTime"]
)
```

```
Upserted 5,000/876,575 docs, ~756/s
Upserted 10,000/876,575 docs, ~1,170/s
Upserted 15,000/876,575 docs, ~1,025/s
Upserted 20,000/876,575 docs, ~949/s
Upserted 25,000/876,575 docs, ~908/s
Upserted 30,000/876,575 docs, ~938/s
Upserted 35,000/876,575 docs, ~921/s
Upserted 40,000/876,575 docs, ~900/s
Upserted 45,000/876,575 docs, ~885/s
Upserted 50,000/876,575 docs, ~938/s
Upserted 55,000/876,575 docs, ~994/s
Upserted 60,000/876,575 docs, ~1,048/s
Upserted 65,000/876,575 docs, ~1,096/s
Upserted 70,000/876,575 docs, ~1,056/s
Upserted 75,000/876,575 docs, ~1,031/s
Upserted 80,000/876,575 docs, ~1,005/s
Upserted 85,000/876,575 docs, ~978/s
Upserted 90,000/876,575 docs, ~973/s
Upserted 95,000/876,575 docs, ~955/s
Upserted 100,000/876,575 docs, ~930/s
Upserted 105,000/876,575 docs, ~909/s
Upserted 110,000/876,575 docs, ~885/s
Upserted 115,000/876,575 docs, ~864/s
Upserted 120,000/876,575 docs, ~851/s
Upserted 125,000/876,575 docs, ~853/s
Upserted 130,000/876,575 docs, ~852/s
Upserted 135,000/876,575 docs, ~849/s
Upserted 140,000/876,575 docs, ~867/s
Upserted 145,000/876,575 docs, ~865/s
Upserted 150,000/876,575 docs, ~853/s
Upserted 155,000/876,575 docs, ~855/s
Upserted 160,000/876,575 docs, ~849/s
Upserted 165,000/876,575 docs, ~837/s
Upserted 170,000/876,575 docs, ~824/s
Upserted 175,000/876,575 docs, ~827/s
Upserted 180,000/876,575 docs, ~840/s
Upserted 185,000/876,575 docs, ~856/s
Upserted 190,000/876,575 docs, ~852/s
Upserted 195,000/876,575 docs, ~865/s
Upserted 200,000/876,575 docs, ~879/s
Upserted 205,000/876,575 docs, ~874/s
Upserted 210,000/876,575 docs, ~875/s
Upserted 215,000/876,575 docs, ~874/s
Upserted 220,000/876,575 docs, ~871/s
Upserted 225,000/876,575 docs, ~869/s
Upserted 230,000/876,575 docs, ~866/s
Upserted 235,000/876,575 docs, ~870/s
Upserted 240,000/876,575 docs, ~879/s
Upserted 245,000/876,575 docs, ~891/s
Upserted 250,000/876,575 docs, ~886/s
Upserted 255,000/876,575 docs, ~886/s
Upserted 260,000/876,575 docs, ~883/s
Upserted 265,000/876,575 docs, ~880/s
Upserted 270,000/876,575 docs, ~888/s
Upserted 275,000/876,575 docs, ~880/s
Upserted 280,000/876,575 docs, ~873/s
Upserted 285,000/876,575 docs, ~864/s
Upserted 290,000/876,575 docs, ~857/s
Upserted 295,000/876,575 docs, ~861/s
Upserted 300,000/876,575 docs, ~872/s
Upserted 305,000/876,575 docs, ~881/s
Upserted 310,000/876,575 docs, ~891/s
Upserted 315,000/876,575 docs, ~900/s
Upserted 320,000/876,575 docs, ~896/s
Upserted 325,000/876,575 docs, ~891/s
Upserted 330,000/876,575 docs, ~893/s
Upserted 335,000/876,575 docs, ~884/s
Upserted 340,000/876,575 docs, ~876/s
Upserted 345,000/876,575 docs, ~870/s
Upserted 350,000/876,575 docs, ~864/s
Upserted 355,000/876,575 docs, ~862/s
Upserted 360,000/876,575 docs, ~869/s
Upserted 365,000/876,575 docs, ~877/s
Upserted 370,000/876,575 docs, ~885/s
Upserted 375,000/876,575 docs, ~882/s
Upserted 380,000/876,575 docs, ~880/s
Upserted 385,000/876,575 docs, ~878/s
Upserted 390,000/876,575 docs, ~877/s
Upserted 395,000/876,575 docs, ~875/s
Upserted 400,000/876,575 docs, ~873/s
Upserted 405,000/876,575 docs, ~876/s
Upserted 410,000/876,575 docs, ~883/s
Upserted 415,000/876,575 docs, ~890/s
Upserted 420,000/876,575 docs, ~898/s
Upserted 425,000/876,575 docs, ~904/s
Upserted 430,000/876,575 docs, ~910/s
Upserted 435,000/876,575 docs, ~906/s
Upserted 440,000/876,575 docs, ~905/s
Upserted 445,000/876,575 docs, ~902/s
Upserted 450,000/876,575 docs, ~897/s
Upserted 455,000/876,575 docs, ~894/s
Upserted 460,000/876,575 docs, ~896/s
Upserted 465,000/876,575 docs, ~898/s
Upserted 470,000/876,575 docs, ~899/s
Upserted 475,000/876,575 docs, ~906/s
Upserted 480,000/876,575 docs, ~912/s
Upserted 485,000/876,575 docs, ~910/s
Upserted 490,000/876,575 docs, ~908/s
Upserted 495,000/876,575 docs, ~906/s
Upserted 500,000/876,575 docs, ~902/s
Upserted 505,000/876,575 docs, ~899/s
Upserted 510,000/876,575 docs, ~894/s
Upserted 515,000/876,575 docs, ~888/s
Upserted 520,000/876,575 docs, ~890/s
Upserted 525,000/876,575 docs, ~891/s
Upserted 530,000/876,575 docs, ~888/s
Upserted 535,000/876,575 docs, ~887/s
Upserted 540,000/876,575 docs, ~885/s
Upserted 545,000/876,575 docs, ~883/s
Upserted 550,000/876,575 docs, ~882/s
Upserted 555,000/876,575 docs, ~881/s
Upserted 560,000/876,575 docs, ~883/s
Upserted 565,000/876,575 docs, ~888/s
Upserted 570,000/876,575 docs, ~893/s
Upserted 575,000/876,575 docs, ~898/s
Upserted 580,000/876,575 docs, ~903/s
Upserted 585,000/876,575 docs, ~908/s
Upserted 590,000/876,575 docs, ~913/s
Upserted 595,000/876,575 docs, ~917/s
Upserted 600,000/876,575 docs, ~921/s
Upserted 605,000/876,575 docs, ~926/s
Upserted 610,000/876,575 docs, ~923/s
Upserted 615,000/876,575 docs, ~921/s
Upserted 620,000/876,575 docs, ~919/s
Upserted 625,000/876,575 docs, ~920/s
Upserted 630,000/876,575 docs, ~922/s
Upserted 635,000/876,575 docs, ~924/s
Upserted 640,000/876,575 docs, ~924/s
Upserted 645,000/876,575 docs, ~920/s
Upserted 650,000/876,575 docs, ~918/s
Upserted 655,000/876,575 docs, ~922/s
Upserted 660,000/876,575 docs, ~921/s
Upserted 665,000/876,575 docs, ~919/s
Upserted 670,000/876,575 docs, ~918/s
Upserted 675,000/876,575 docs, ~915/s
Upserted 680,000/876,575 docs, ~912/s
Upserted 685,000/876,575 docs, ~908/s
Upserted 690,000/876,575 docs, ~904/s
Upserted 695,000/876,575 docs, ~905/s
Upserted 700,000/876,575 docs, ~901/s
Upserted 705,000/876,575 docs, ~903/s
Upserted 710,000/876,575 docs, ~908/s
Upserted 715,000/876,575 docs, ~912/s
Upserted 720,000/876,575 docs, ~910/s
Upserted 725,000/876,575 docs, ~909/s
Upserted 730,000/876,575 docs, ~908/s
Upserted 735,000/876,575 docs, ~910/s
Upserted 740,000/876,575 docs, ~914/s
Upserted 745,000/876,575 docs, ~918/s
Upserted 750,000/876,575 docs, ~922/s
Upserted 755,000/876,575 docs, ~925/s
Upserted 760,000/876,575 docs, ~929/s
Upserted 765,000/876,575 docs, ~933/s
Upserted 770,000/876,575 docs, ~937/s
Upserted 775,000/876,575 docs, ~940/s
Upserted 780,000/876,575 docs, ~943/s
Upserted 785,000/876,575 docs, ~946/s
Upserted 790,000/876,575 docs, ~950/s
Upserted 795,000/876,575 docs, ~953/s
Upserted 800,000/876,575 docs, ~955/s
Upserted 805,000/876,575 docs, ~956/s
Upserted 810,000/876,575 docs, ~958/s
Upserted 815,000/876,575 docs, ~958/s
Upserted 820,000/876,575 docs, ~959/s
Upserted 825,000/876,575 docs, ~962/s
Upserted 830,000/876,575 docs, ~965/s
Upserted 835,000/876,575 docs, ~963/s
Upserted 840,000/876,575 docs, ~965/s
Upserted 845,000/876,575 docs, ~964/s
Upserted 850,000/876,575 docs, ~962/s
Upserted 855,000/876,575 docs, ~958/s
Upserted 860,000/876,575 docs, ~955/s
Upserted 865,000/876,575 docs, ~951/s
Upserted 870,000/876,575 docs, ~948/s
Upserted 875,000/876,575 docs, ~944/s
Finished consumption_2021_2024: 876,575 upserts
```

Mongo validation

```
In [15]: # Mongo validation, correct field names for both collections
import certifi
from pymongo import MongoClient

cli = MongoClient(MONGODB_URI, tlsCAFile=certifi.where(), serverSelection

colp = cli[DB_NAME][COLL_PROD_21_24]
colc = cli[DB_NAME][COLL_CONS_21_24]

print("Production areas:", sorted(colp.distinct("priceArea")))
print("Production groups:", sorted(colp.distinct("group"))[:10])

print("Consumption areas:", sorted(colc.distinct("priceArea")))
print("Consumption groups:", sorted(colc.distinct("group"))[:10])

cli.close()
```

```
Production areas: ['NO1', 'NO2', 'NO3', 'NO4', 'NO5']
Production groups: ['hydro', 'other', 'solar', 'thermal', 'wind']
Consumption areas: ['NO1', 'NO2', 'NO3', 'NO4', 'NO5']
Consumption groups: ['cabin', 'household', 'primary', 'secondary', 'tertia
ry']
```

Links

- GitHub repository: https://github.com/youneshansen/ind320-yohan3351
- Streamlit app: https://ind320-yohan3351.streamlit.app/

Info

I chose the Snow drift bonus task for the Streamlit app and plotted the monthly and yearly data in one plot

Work log

For the task 4 notebook I extended the project with both production and consumption data for 2021 to 2024. I implemented monthly and weekly fallback fetching from the Elhub API, normalized the payloads into a consistent dataframe structure, and performed sanity checks for areas, groups, timestamps, and missing values. I wrote the full dataset for 2022 to 2024 production and 2021 to 2024 consumption into Cassandra using Spark with throttled settings and a corrected schema including the year column I also merged the 2021 production data with the new data and upserted everything into MongoDB. Finally, I verified the stored values with validation queries and a summary plot

For the strealit part of the task I started by converting all static plots to interactive Plotly visualizations, which made the app much more user friendly and cleaner.

I grouped the pages into four sections: Explorative Analysis (map, elhub production , data overview, snow drift), Signal Analysis (STL & spectrogram, Sliding window correlation), Anomaly Detection (SPC and LOF), and Predictive Analysis (SARIMAX forecasting). I felt this structure made sense based on the analysis types.

The biggest challenge was implementing the map page with GeoJSON overlays. The AI first suggested i use streamlit-plotly-events, but it did not work well at all. But then I tried Folium and was able to display the price areas and added choropleth coloring based on production values. Getting the coordinates to persist on the snow drift page using session state took some debugging.

I chose the Snow drift bonus task and plotted monthly and yearly data together on one chart, by way of columns with the yearly values in a more see through color and the monthly data in bright color. It required several attempts to get the visualizations right and i tried with lines for the yearly data, but felt it didnt look too good

I also implemented some error handling, caching, and progress indicators throughout to improve user experience, but did not dive very deep into those bonus tasks.

Ai log

I used AI (chatgpt 5) mainly for debugging and refactoring throughout notebook 4. It helped me identify why the Cassandra schema failed and guided me through adding the year column correctly. I also used AI to fix problems with my Python driver setup, handle timezone normalization, and ensure consistent field naming between production and consumption datasets. It also helped me rewrite the Cassandra and MongoDB upsert logic .

I used GitHub Copilot (Claude Sonnet 4.5) for the Streamlit app development. The copilot helped me with for example converting plots to Plotly, implementing the GeoJSON map with Folium, and integrating the snow drift calculations. It was especially usefull for analyzing the snow_drift.py file. When I encountered bugs, I described the problem and the copilot identified and fixed the issues. The copilot also assisted with adding caching, error handling, and cleaning up code comments.