Medallion Pipeline – Silver Layer (Curated Elhub Data)

Goal of this notebook (SILVER):

- Load raw-but-structured data from the **Bronze layer**:
  - Production: `Data_Assignment_4/bronze/production/*.csv`
  - Consumption: `Data_Assignment_4/bronze/consumption/*.csv`
- (Optionally) load **2021 production** from existing Cassandra table `energy.production_2021`
- Clean and standardize schema:
  - Consistent column names and dtypes
  - Remove duplicates / obvious junk
- Save **curated Silver data** to:
  - `Data_Assignment_4/silver/production/production_YYYY.csv` (or combined)
  - `Data_Assignment_4/silver/consumption/consumption_YYYY.csv`
- Optionally write Silver data into **Cassandra & MongoDB** as new curated tables.

In [ ]:
```
# Configuration and setup for the data medallion architecture
import os
import pandas as pd
from datetime import datetime

SPARK_AVAILABLE = False
try:
    from pyspark.sql import SparkSession
    from pyspark.sql.types import (
        StructType, StructField, StringType,
        TimestampType, DoubleType, IntegerType
    )
    SPARK_AVAILABLE = True
except Exception as e:
    print("Spark unavailable:", e)

print("Pandas:", pd.__version__)
print("Spark available:", SPARK_AVAILABLE)

# we define project directories
PROJECT_ROOT = ".."

BRONZE_ROOT = os.path.join(PROJECT_ROOT, "Data_Assignment_4", "bronze")
BRONZE_PROD_DIR = os.path.join(BRONZE_ROOT, "production")
BRONZE_CONS_DIR = os.path.join(BRONZE_ROOT, "consumption")

SILVER_ROOT = os.path.join(PROJECT_ROOT, "Data_Assignment_4", "silver")
SILVER_PROD_DIR = os.path.join(SILVER_ROOT, "production")
```

```
SILVER_CONS_DIR = os.path.join(SILVER_ROOT, "consumption")

os.makedirs(SILVER_PROD_DIR, exist_ok=True)
os.makedirs(SILVER_CONS_DIR, exist_ok=True)

print("Bronze production dir:", BRONZE_PROD_DIR)
print("Bronze consumption dir:", BRONZE_CONS_DIR)
print("Silver production dir:", SILVER_PROD_DIR)
print("Silver consumption dir:", SILVER_CONS_DIR)

# ---- Cassandra & Mongo config for SILVER tables ----
CASSANDRA_HOSTS = "127.0.0.1"
CASSANDRA_KEYSPACE = "energy"

CASSANDRA_TABLE_PROD_2021 = "production_2021"
CASSANDRA_TABLE_SILVER_PROD = "production_2021_2024"
CASSANDRA_TABLE_SILVER_CONS = "consumption_2021_2024"

MONGO_URI = "mongodb://localhost:27017"
MONGO_DATABASE = "ind320"
MONGO_COLL_SILVER_PROD = "production_2021_2024"
MONGO_COLL_SILVER_CONS = "consumption_2021_2024"
```

```
Spark unavailable: No module named 'pyspark.errors'
Pandas: 2.3.3
Spark available: False
Bronze production dir: ..\Data_Assignment_4\bronze\production
Bronze consumption dir: ..\Data_Assignment_4\bronze\consumption
Silver production dir: ..\Data_Assignment_4\silver\production
Silver consumption dir: ..\Data_Assignment_4\silver\consumption
```

In [ ]:

```
# Load the bronze CSV files into DataFrames
def load_bronze_folder(folder: str, parse_dates=None) -> pd.DataFrame:
    if not os.path.isdir(folder):
        print(f"⚠ Folder does not exist: {folder}")
        return pd.DataFrame()

    files = [f for f in os.listdir(folder) if f.endswith(".csv")]
    if not files:
        print(f"⚠ No CSV files in {folder}")
        return pd.DataFrame()

    frames = []
    for fname in sorted(files):
        path = os.path.join(folder, fname)
        print("Loading", path)
        df = pd.read_csv(path, parse_dates=parse_dates)
        frames.append(df)

    if not frames:
        return pd.DataFrame()

    return pd.concat(frames, ignore_index=True)


# Bronze → production & consumption DataFrames
```

```
bronze_prod_df = load_bronze_folder(
    BRONZE_PROD_DIR,
    parse_dates=["startTime", "endTime"]
)

bronze_cons_df = load_bronze_folder(
    BRONZE_CONS_DIR,
    parse_dates=["startTime", "endTime"]
)

print("\nBronze production rows:", len(bronze_prod_df))
print("Bronze consumption rows:", len(bronze_cons_df))

if not bronze_prod_df.empty:
    print("Production sample:")
    display(bronze_prod_df.head())

if not bronze_cons_df.empty:
    print("Consumption sample:")
    display(bronze_cons_df.head())
```

```
Loading ..\Data_Assignment_4\bronze\production\production_2025.csv
Loading ..\Data_Assignment_4\bronze\consumption\consumption_2025.csv

Bronze production rows: 678636
Bronze consumption rows: 868800
Production sample:
```

	priceArea	group	quantityKwh	countMeteringPoints	startTime	endTime
0	NO1	hydro	2332513.0	NaN	2025-10-22 19:00:00+00:00	2025-10-22 20:00:00+00:00
1	NO1	hydro	2281520.8	NaN	2025-10-22 20:00:00+00:00	2025-10-22 21:00:00+00:00
2	NO1	hydro	2308167.5	NaN	2025-10-22 21:00:00+00:00	2025-10-22 22:00:00+00:00
3	NO1	hydro	2253987.5	NaN	2025-10-22 22:00:00+00:00	2025-10-22 23:00:00+00:00
4	NO1	hydro	2241552.0	NaN	2025-10-22 23:00:00+00:00	2025-10-23 00:00:00+00:00

```
Consumption sample:
```

	priceArea	group	quantityKwh	countMeteringPoints	startTime	endTime
0	NO1	cabin	67069.060	NaN	2025-10-22 19:00:00+00:00	2025-10-22 20:00:00+00:00
1	NO1	cabin	66767.110	NaN	2025-10-22 20:00:00+00:00	2025-10-22 21:00:00+00:00
2	NO1	cabin	65964.305	NaN	2025-10-22 21:00:00+00:00	2025-10-22 22:00:00+00:00
3	NO1	cabin	65159.824	NaN	2025-10-22 22:00:00+00:00	2025-10-22 23:00:00+00:00
4	NO1	cabin	64141.660	NaN	2025-10-22 23:00:00+00:00	2025-10-23 00:00:00+00:00

In [ ]:

```
# Load 2021 production data from Cassandra (if available)
def build_silver_production(bronze_df: pd.DataFrame, prod2021_df: pd.DataFrame) ->
    # 1) Bronze 2022+ (from API CSVs)
    if bronze_df.empty:
        bronze_clean = pd.DataFrame(columns=["priceArea", "productionGroup", "start
    else:
        bronze_clean = bronze_df.copy()

        # Map generic 'group' -> productionGroup if present
        if "group" in bronze_clean.columns:
            bronze_clean["productionGroup"] = bronze_clean["group"]
        elif "productionGroup" not in bronze_clean.columns:
            bronze_clean["productionGroup"] = None

        # Ensure necessary columns exist
        for col in ["priceArea", "productionGroup", "startTime", "quantityKwh"]:
            if col not in bronze_clean.columns:
                bronze_clean[col] = None

        # Typing & cleaning
        bronze_clean["startTime"] = pd.to_datetime(bronze_clean["startTime"], utc=T
        bronze_clean["priceArea"] = bronze_clean["priceArea"].astype(str).str.upper
        bronze_clean["productionGroup"] = bronze_clean["productionGroup"].astype(st
        bronze_clean["quantityKwh"] = pd.to_numeric(bronze_clean["quantityKwh"], er

        bronze_clean = bronze_clean[["priceArea", "productionGroup", "startTime", "
        bronze_clean = bronze_clean.dropna(subset=["priceArea", "productionGroup",

    # 2) 2021 from Cassandra (if available)
    if prod2021_df is None or prod2021_df.empty:
        combined = bronze_clean
    else:
        # Align schema just in case
        p2021 = prod2021_df[["priceArea", "productionGroup", "startTime", "quantity
        p2021["startTime"] = pd.to_datetime(p2021["startTime"], utc=True, errors="c
        p2021["priceArea"] = p2021["priceArea"].astype(str).str.upper()
        p2021["productionGroup"] = p2021["productionGroup"].astype(str).str.upper()
        p2021["quantityKwh"] = pd.to_numeric(p2021["quantityKwh"], errors="coerce")
```

```
        combined = pd.concat([p2021, bronze_clean], ignore_index=True)

    # 3) Deduplicate
    if not combined.empty:
        combined = combined.drop_duplicates(
            subset=["priceArea", "productionGroup", "startTime"],
            keep="first"
        ).sort_values("startTime").reset_index(drop=True)

    return combined


silver_prod_df = build_silver_production(bronze_prod_df, prod_2021_df)
print("Silver production rows:", len(silver_prod_df))

if not silver_prod_df.empty:
    print("Silver production time span:",
          silver_prod_df["startTime"].min(), "→", silver_prod_df["startTime"].max()
    display(silver_prod_df.head())
```

```
Silver production rows: 18851
Silver production time span: 2025-10-22 19:00:00+00:00 → 2025-11-21 22:00:00+00:00
```

	priceArea	productionGroup	startTime	quantityKwh
0	NO1	HYDRO	2025-10-22 19:00:00+00:00	2332513.000
1	NO5	THERMAL	2025-10-22 19:00:00+00:00	20266.242
2	NO1	SOLAR	2025-10-22 19:00:00+00:00	326.164
3	NO5	SOLAR	2025-10-22 19:00:00+00:00	51.456
4	NO1	THERMAL	2025-10-22 19:00:00+00:00	10453.320

In [ ]:

```
# Load 2021 consumption data from Cassandra (if available)
def build_silver_consumption(bronze_df: pd.DataFrame) -> pd.DataFrame:
    if bronze_df.empty:
        return pd.DataFrame(
            columns=["priceArea", "consumptionGroup", "startTime", "quantityKwh", "
        )

    df = bronze_df.copy()

    # Map generic 'group' -> consumptionGroup if present
    if "group" in df.columns:
        df["consumptionGroup"] = df["group"]
    elif "consumptionGroup" not in df.columns:
        df["consumptionGroup"] = None

    for col in ["priceArea", "consumptionGroup", "startTime", "quantityKwh", "count
        if col not in df.columns:
            df[col] = None

    df["startTime"] = pd.to_datetime(df["startTime"], utc=True, errors="coerce")
    df["priceArea"] = df["priceArea"].astype(str).str.upper()
    df["consumptionGroup"] = df["consumptionGroup"].astype(str).str.upper()
```

```
    df["quantityKwh"] = pd.to_numeric(df["quantityKwh"], errors="coerce")
    df["countMeteringPoints"] = pd.to_numeric(df["countMeteringPoints"], errors="co

    df = df[["priceArea", "consumptionGroup", "startTime", "quantityKwh", "countMet
    df = df.dropna(subset=["priceArea", "consumptionGroup", "startTime", "quantityK

    # Deduplicate
    df = df.drop_duplicates(
        subset=["priceArea", "consumptionGroup", "startTime"],
        keep="first"
    ).sort_values("startTime").reset_index(drop=True)

    return df


silver_cons_df = build_silver_consumption(bronze_cons_df)
print("Silver consumption rows:", len(silver_cons_df))

if not silver_cons_df.empty:
    print("Silver consumption time span:",
          silver_cons_df["startTime"].min(), "→", silver_cons_df["startTime"].max()
    display(silver_cons_df.head())
```

```
Silver consumption rows: 18100
Silver consumption time span: 2025-10-22 19:00:00+00:00 → 2025-11-21 22:00:00+00:00
```

	priceArea	consumptionGroup	startTime	quantityKwh	countMeteringPoints
0	NO1	CABIN	2025-10-22 19:00:00+00:00	67069.06	NaN
1	NO5	SECONDARY	2025-10-22 19:00:00+00:00	1085012.40	NaN
2	NO1	TERTIARY	2025-10-22 19:00:00+00:00	1086670.60	NaN
3	NO4	CABIN	2025-10-22 19:00:00+00:00	46269.73	NaN
4	NO1	HOUSEHOLD	2025-10-22 19:00:00+00:00	2279264.80	NaN

In [ ]:
```
# Now we have silver_prod_df and silver_cons_df ready
if not silver_prod_df.empty:
    silver_prod_df = silver_prod_df.copy()
    silver_prod_df["year"] = silver_prod_df["startTime"].dt.year

    print("Production years in SILVER:", sorted(silver_prod_df["year"].dropna().uni

    for year, df_y in silver_prod_df.groupby("year"):
        out_path = os.path.join(SILVER_PROD_DIR, f"production_silver_{int(year)}.cs
        df_y.drop(columns=["year"]).to_csv(out_path, index=False)
        print(f"Saved {len(df_y):,} rows to {out_path}")
else:
    print("⚠ No silver production rows to save.")
```

```python
# we save silver consumption data
if not silver_cons_df.empty:
    silver_cons_df = silver_cons_df.copy()
    silver_cons_df["year"] = silver_cons_df["startTime"].dt.year

    print("Consumption years in SILVER:", sorted(silver_cons_df["year"].dropna().un

    for year, df_y in silver_cons_df.groupby("year"):
        out_path = os.path.join(SILVER_CONS_DIR, f"consumption_silver_{int(year)}.c
        df_y.drop(columns=["year"]).to_csv(out_path, index=False)
        print(f"Saved {len(df_y):,} rows to {out_path}")
else:
    print("⚠ No silver consumption rows to save.")
```

```
Production years in SILVER: [np.int32(2025)]
Saved 18,851 rows to ..\Data_Assignment_4\silver\production\production_silver_2025.c
sv
Consumption years in SILVER: [np.int32(2025)]
Saved 18,100 rows to ..\Data_Assignment_4\silver\consumption\consumption_silver_202
5.csv
```