# Medallion Pipeline – Gold Layer (Analytics-Ready Tables)

**Scope of this notebook (GOLD):**

- Load curated **Silver** data for production and consumption.
- Create analytics-ready tables:
  - Daily aggregates per price area and group.
  - Combined net balances (production - consumption).
  - Optional pivoted (wide) forms for easy plotting, correlation, and forecasting.
- Save results in `Data_Assignment_4/gold/...` for use in Streamlit.

In [ ]:
```python
# we define imports and configuration
import shutil
import os
import pandas as pd
import numpy as np

PROJECT_ROOT = ".."
STREAMLIT_DATA_DIR = r"C:\NMBU\IND320\streamlit\Data\gold"

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")

GOLD_ROOT = os.path.join(PROJECT_ROOT, "Data_Assignment_4", "gold")
GOLD_PROD_DIR = os.path.join(GOLD_ROOT, "production")
GOLD_CONS_DIR = os.path.join(GOLD_ROOT, "consumption")
GOLD_COMBINED_DIR = os.path.join(GOLD_ROOT, "combined")

os.makedirs(GOLD_PROD_DIR, exist_ok=True)
os.makedirs(GOLD_CONS_DIR, exist_ok=True)
os.makedirs(GOLD_COMBINED_DIR, exist_ok=True)

print("Silver production dir:", SILVER_PROD_DIR)
print("Silver consumption dir:", SILVER_CONS_DIR)
print("Gold root:", GOLD_ROOT)
```

```
Silver production dir: ..\Data_Assignment_4\silver\production
Silver consumption dir: ..\Data_Assignment_4\silver\consumption
Gold root: ..\Data_Assignment_4\gold
```

In [ ]:
```python
# we define function to load silver data
def load_silver_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)


silver_prod_df = load_silver_folder(
    SILVER_PROD_DIR,
    parse_dates=["startTime"]
)

silver_cons_df = load_silver_folder(
    SILVER_CONS_DIR,
    parse_dates=["startTime"]
)

print("\nSilver production rows:", len(silver_prod_df))
print("Silver consumption rows:", len(silver_cons_df))

if not silver_prod_df.empty:
    print("Silver production columns:", silver_prod_df.columns.tolist())
    display(silver_prod_df.head())

if not silver_cons_df.empty:
    print("Silver consumption columns:", silver_cons_df.columns.tolist())
    display(silver_cons_df.head())
```

```
Loading ..\Data_Assignment_4\silver\production\production_silver_2025.csv
Loading ..\Data_Assignment_4\silver\consumption\consumption_silver_2025.csv

Silver production rows: 18851
Silver consumption rows: 18100
Silver production columns: ['priceArea', 'productionGroup', 'startTime', 'quantityKw
h']
```

| | 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 |

```
Silver consumption columns: ['priceArea', 'consumptionGroup', 'startTime', 'quantity
Kwh', 'countMeteringPoints']
```

| | 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 [ ]:
```python
# we define function to prepare time features
def prep_time_features(df: pd.DataFrame, time_col: str = "startTime") -> pd.DataFra
    if df.empty:
        return df

    df = df.copy()
    df[time_col] = pd.to_datetime(df[time_col], utc=True, errors="coerce")
    df = df.dropna(subset=[time_col])

    df["date"] = df[time_col].dt.date
    df["year"] = df[time_col].dt.year
    df["month"] = df[time_col].dt.month
    df["day"] = df[time_col].dt.day
    df["hour"] = df[time_col].dt.hour

    return df


silver_prod_df = prep_time_features(silver_prod_df, "startTime")
silver_cons_df = prep_time_features(silver_cons_df, "startTime")

print("Production date range:",
      silver_prod_df["startTime"].min(), "→", silver_prod_df["startTime"].max()
      if not silver_prod_df.empty else "N/A")

print("Consumption date range:",
      silver_cons_df["startTime"].min(), "→", silver_cons_df["startTime"].max()
      if not silver_cons_df.empty else "N/A")
```

```
Production date range: 2025-10-22 19:00:00+00:00 → 2025-11-21 22:00:00+00:00
Consumption date range: 2025-10-22 19:00:00+00:00 → 2025-11-21 22:00:00+00:00
```

In [ ]:
```python
# we define function to prepare time features
def prep_time_features(df: pd.DataFrame, time_col: str = "startTime") -> pd.DataFra
    if df.empty:
        return df
    df = df.copy()
    df[time_col] = pd.to_datetime(df[time_col], utc=True, errors="coerce")
    df = df.dropna(subset=[time_col])

    df["date"] = df[time_col].dt.date
    df["year"] = df[time_col].dt.year
    df["month"] = df[time_col].dt.month
    df["day"] = df[time_col].dt.day
    df["hour"] = df[time_col].dt.hour

    return df


silver_prod_df = prep_time_features(silver_prod_df, "startTime")
silver_cons_df = prep_time_features(silver_cons_df, "startTime")

print("Production date range:",
      silver_prod_df["startTime"].min(), "→", silver_prod_df["startTime"].max()
      if not silver_prod_df.empty else "N/A")

print("Consumption date range:",
      silver_cons_df["startTime"].min(), "→", silver_cons_df["startTime"].max()
      if not silver_cons_df.empty else "N/A")
```

```
Production date range: 2025-10-22 19:00:00+00:00 → 2025-11-21 22:00:00+00:00
Consumption date range: 2025-10-22 19:00:00+00:00 → 2025-11-21 22:00:00+00:00
```

In [ ]:

```
# we define function to build daily production aggregates
def build_daily_production(df: pd.DataFrame):
    if df.empty:
        return pd.DataFrame(), pd.DataFrame()

    # 1) Daily per priceArea + productionGroup
    daily_group = (
        df.groupby(["date", "priceArea", "productionGroup"], as_index=False)
          .agg(
              quantityKwh_sum=("quantityKwh", "sum"),
              quantityKwh_mean=("quantityKwh", "mean")
          )
    )

    # 2) Daily per priceArea (summing groups)
    daily_area = (
        df.groupby(["date", "priceArea"], as_index=False)
          .agg(
              quantityKwh_sum=("quantityKwh", "sum"),
              quantityKwh_mean=("quantityKwh", "mean")
          )
    )

    return daily_group, daily_area


gold_prod_daily_group_df, gold_prod_daily_area_df = build_daily_production(silver_p

print("Gold production (daily, by group) rows:", len(gold_prod_daily_group_df))
print("Gold production (daily, by area) rows:", len(gold_prod_daily_area_df))

if not gold_prod_daily_group_df.empty:
    display(gold_prod_daily_group_df.head())

if not gold_prod_daily_area_df.empty:
    display(gold_prod_daily_area_df.head())
```

```
Gold production (daily, by group) rows: 808
Gold production (daily, by area) rows: 155
```

| | date | priceArea | productionGroup | quantityKwh_sum | quantityKwh_mean |
|---|---|---|---|---|---|
| **0** | 2025-10-22 | NO1 | HYDRO | 1.141774e+07 | 2.283548e+06 |
| **1** | 2025-10-22 | NO1 | OTHER | 1.170280e+02 | 2.340560e+01 |
| **2** | 2025-10-22 | NO1 | SOLAR | 1.950554e+03 | 3.901108e+02 |
| **3** | 2025-10-22 | NO1 | THERMAL | 5.137358e+04 | 1.027472e+04 |
| **4** | 2025-10-22 | NO1 | WIND | 4.134188e+05 | 8.268375e+04 |

| | date | priceArea | quantityKwh_sum | quantityKwh_mean |
|---|---|---|---|---|
| **0** | 2025-10-22 | NO1 | 1.188460e+07 | 4.753840e+05 |
| **1** | 2025-10-22 | NO2 | 3.457493e+07 | 1.152498e+06 |
| **2** | 2025-10-22 | NO3 | 1.619981e+07 | 6.479925e+05 |
| **3** | 2025-10-22 | NO4 | 1.763749e+07 | 7.054998e+05 |
| **4** | 2025-10-22 | NO5 | 1.699174e+07 | 5.663913e+05 |

```
In [ ]: # we define function to build daily consumption aggregates
        def build_daily_consumption(df: pd.DataFrame):
            if df.empty:
                return pd.DataFrame(), pd.DataFrame()

            # 1) Daily per priceArea + consumptionGroup
            daily_group = (
                df.groupby(["date", "priceArea", "consumptionGroup"], as_index=False)
                  .agg(
                      quantityKwh_sum=("quantityKwh", "sum"),
                      quantityKwh_mean=("quantityKwh", "mean"),
                      countMeteringPoints_sum=("countMeteringPoints", "sum")
                  )
            )

            # 2) Daily per priceArea (summing groups)
            daily_area = (
                df.groupby(["date", "priceArea"], as_index=False)
                  .agg(
                      quantityKwh_sum=("quantityKwh", "sum"),
                      quantityKwh_mean=("quantityKwh", "mean"),
                      countMeteringPoints_sum=("countMeteringPoints", "sum")
                  )
```

```
    )

    return daily_group, daily_area


gold_cons_daily_group_df, gold_cons_daily_area_df = build_daily_consumption(silver_

print("Gold consumption (daily, by group) rows:", len(gold_cons_daily_group_df))
print("Gold consumption (daily, by area) rows:", len(gold_cons_daily_area_df))

if not gold_cons_daily_group_df.empty:
    display(gold_cons_daily_group_df.head())

if not gold_cons_daily_area_df.empty:
    display(gold_cons_daily_area_df.head())
```

```
Gold consumption (daily, by group) rows: 775
Gold consumption (daily, by area) rows: 155
```

| | date | priceArea | consumptionGroup | quantityKwh_sum | quantityKwh_mean | countMeterin |
|---|---|---|---|---|---|---|
| **0** | 2025-10-22 | NO1 | CABIN | 329101.959 | 6.582039e+04 | |
| **1** | 2025-10-22 | NO1 | HOUSEHOLD | 9964986.100 | 1.992997e+06 | |
| **2** | 2025-10-22 | NO1 | PRIMARY | 184975.642 | 3.699513e+04 | |
| **3** | 2025-10-22 | NO1 | SECONDARY | 2886397.850 | 5.772796e+05 | |
| **4** | 2025-10-22 | NO1 | TERTIARY | 4841191.000 | 9.682382e+05 | |

| | date | priceArea | quantityKwh_sum | quantityKwh_mean | countMeteringPoints_sum |
|---|---|---|---|---|---|
| **0** | 2025-10-22 | NO1 | 1.820665e+07 | 728266.10204 | 0.0 |
| **1** | 2025-10-22 | NO2 | 1.880126e+07 | 752050.45416 | 0.0 |
| **2** | 2025-10-22 | NO3 | 1.564362e+07 | 625744.89880 | 0.0 |
| **3** | 2025-10-22 | NO4 | 1.144285e+07 | 457714.15460 | 0.0 |
| **4** | 2025-10-22 | NO5 | 8.958708e+06 | 358348.32200 | 0.0 |

In [ ]:
```
# we define function to build daily net balance
def build_daily_net_balance(prod_area_df: pd.DataFrame, cons_area_df: pd.DataFrame)
    if prod_area_df.empty or cons_area_df.empty:
        return pd.DataFrame()
    p = prod_area_df.rename(
        columns={
            "quantityKwh_sum": "prod_quantityKwh_sum",
            "quantityKwh_mean": "prod_quantityKwh_mean",
        }
    )

    c = cons_area_df.rename(
        columns={
            "quantityKwh_sum": "cons_quantityKwh_sum",
            "quantityKwh_mean": "cons_quantityKwh_mean",
            "countMeteringPoints_sum": "cons_countMeteringPoints_sum",
        }
    )

    merged = p.merge(c, on=["date", "priceArea"], how="outer")

    # Fill missing with zeros (no production or no consumption that day)
    for col in [
        "prod_quantityKwh_sum", "prod_quantityKwh_mean",
        "cons_quantityKwh_sum", "cons_quantityKwh_mean",
        "cons_countMeteringPoints_sum"
    ]:
        if col in merged.columns:
            merged[col] = merged[col].fillna(0.0)

    merged["net_quantityKwh_sum"] = merged["prod_quantityKwh_sum"] - merged["cons_q

    return merged


gold_daily_net_df = build_daily_net_balance(
    gold_prod_daily_area_df,
    gold_cons_daily_area_df
)

print("Gold net-balance rows:", len(gold_daily_net_df))
if not gold_daily_net_df.empty:
    display(gold_daily_net_df.head())
```

```
Gold net-balance rows: 155
```

| | date | priceArea | prod_quantityKwh_sum | prod_quantityKwh_mean | cons_quantityKwh_sum |
|---|---|---|---|---|---|
| **0** | 2025-10-22 | NO1 | 1.188460e+07 | 4.753840e+05 | 1.820665e+07 |
| **1** | 2025-10-22 | NO2 | 3.457493e+07 | 1.152498e+06 | 1.880126e+07 |
| **2** | 2025-10-22 | NO3 | 1.619981e+07 | 6.479925e+05 | 1.564362e+07 |
| **3** | 2025-10-22 | NO4 | 1.763749e+07 | 7.054998e+05 | 1.144285e+07 |
| **4** | 2025-10-22 | NO5 | 1.699174e+07 | 5.663913e+05 | 8.958708e+06 |

In [ ]:
```python
# we define function to pivot daily group data
def pivot_daily_group(df: pd.DataFrame, area_col: str, group_col: str,
                      value_col: str, price_area: str):
    """Return a wide dataframe: index=date, columns=group, values=value_col."""
    if df.empty:
        return pd.DataFrame()

    sub = df[df[area_col] == price_area].copy()
    if sub.empty:
        return pd.DataFrame()

    wide = (
        sub.pivot_table(
            index="date",
            columns=group_col,
            values=value_col,
            aggfunc="sum"
        )
        .sort_index()
    )

    wide.columns = [str(c) for c in wide.columns]
    wide = wide.rename_axis(None, axis=1)
    return wide


example_area = "NO1"

prod_pivot_no1 = pivot_daily_group(
    gold_prod_daily_group_df,
    area_col="priceArea",
    group_col="productionGroup",
    value_col="quantityKwh_sum",
    price_area=example_area
)

print("Pivoted production for", example_area, "shape:", prod_pivot_no1.shape)
display(prod_pivot_no1.head())
```

```
Pivoted production for NO1 shape: (31, 5)
```

| date | HYDRO | OTHER | SOLAR | THERMAL | WIND |
|---|---|---|---|---|---|
| **2025-10-22** | 11417740.8 | 117.028 | 1950.554 | 51373.580 | 413418.750 |
| **2025-10-23** | 55627515.4 | 883.822 | 54752.178 | 244615.910 | 1646541.483 |
| **2025-10-24** | 55710017.7 | 1116.824 | 13920.328 | 265221.438 | 5526525.870 |
| **2025-10-25** | 58500559.3 | 1479.485 | 58255.455 | 289919.669 | 1850983.570 |
| **2025-10-26** | 60837679.4 | 926.992 | 79604.672 | 279904.460 | 383813.980 |

```
In [ ]:
# finally, we save the GOLD data tables
if not gold_prod_daily_group_df.empty:
    path1 = os.path.join(GOLD_PROD_DIR, "production_daily_by_group.csv")
    gold_prod_daily_group_df.to_csv(path1, index=False)

    # also save to Streamlit
    path2 = os.path.join(STREAMLIT_DATA_DIR, "production_daily_by_group.csv")
    gold_prod_daily_group_df.to_csv(path2, index=False)

    print("Saved production_daily_by_group.csv to:")
    print(" -", path1)
    print(" -", path2)


if not gold_prod_daily_area_df.empty:
    path1 = os.path.join(GOLD_PROD_DIR, "production_daily_by_area.csv")
    gold_prod_daily_area_df.to_csv(path1, index=False)

    path2 = os.path.join(STREAMLIT_DATA_DIR, "production_daily_by_area.csv")
    gold_prod_daily_area_df.to_csv(path2, index=False)

    print("Saved production_daily_by_area.csv to:")
    print(" -", path1)
    print(" -", path2)


# and the consumption tables
if not gold_cons_daily_group_df.empty:
    path1 = os.path.join(GOLD_CONS_DIR, "consumption_daily_by_group.csv")
    gold_cons_daily_group_df.to_csv(path1, index=False)

    path2 = os.path.join(STREAMLIT_DATA_DIR, "consumption_daily_by_group.csv")
    gold_cons_daily_group_df.to_csv(path2, index=False)

    print("Saved consumption_daily_by_group.csv to:")
    print(" -", path1)
    print(" -", path2)


if not gold_cons_daily_area_df.empty:
    path1 = os.path.join(GOLD_CONS_DIR, "consumption_daily_by_area.csv")
```

```python
    gold_cons_daily_area_df.to_csv(path1, index=False)

    path2 = os.path.join(STREAMLIT_DATA_DIR, "consumption_daily_by_area.csv")
    gold_cons_daily_area_df.to_csv(path2, index=False)

    print("Saved consumption_daily_by_area.csv to:")
    print(" -", path1)
    print(" -", path2)


# and the net-balance table
if not gold_daily_net_df.empty:
    path1 = os.path.join(GOLD_COMBINED_DIR, "net_daily_by_area.csv")
    gold_daily_net_df.to_csv(path1, index=False)

    path2 = os.path.join(STREAMLIT_DATA_DIR, "net_daily_by_area.csv")
    gold_daily_net_df.to_csv(path2, index=False)

    print("Saved net_daily_by_area.csv to:")
    print(" -", path1)
    print(" -", path2)
```

```
Saved production_daily_by_group.csv to:
 - ..\Data_Assignment_4\gold\production\production_daily_by_group.csv
 - C:\NMBU\IND320\streamlit\Data\gold\production_daily_by_group.csv
Saved production_daily_by_area.csv to:
 - ..\Data_Assignment_4\gold\production\production_daily_by_area.csv
 - C:\NMBU\IND320\streamlit\Data\gold\production_daily_by_area.csv
Saved consumption_daily_by_group.csv to:
 - ..\Data_Assignment_4\gold\consumption\consumption_daily_by_group.csv
 - C:\NMBU\IND320\streamlit\Data\gold\consumption_daily_by_group.csv
Saved consumption_daily_by_area.csv to:
 - ..\Data_Assignment_4\gold\consumption\consumption_daily_by_area.csv
 - C:\NMBU\IND320\streamlit\Data\gold\consumption_daily_by_area.csv
Saved net_daily_by_area.csv to:
 - ..\Data_Assignment_4\gold\combined\net_daily_by_area.csv
 - C:\NMBU\IND320\streamlit\Data\gold\net_daily_by_area.csv
```