IND320 Project Work

- **Github Link :** https://github.com/Mobashra/M-Abeer-Project
- **Streamlit Link :** https://m-abeer-project.streamlit.app/

Project Log

For this project, I have extended my previous work by integrating **Cassandra**, **MongoDB**, **Spark**, and **Streamlit** to retrieve, process, and visualize **hourly electricity production data** from the Elhub API for all Norwegian price areas in **2021**.

Firstly, I have set up a local database 'Cassandra' and connected it to Spark using the Datastax Spark–Cassandra Connector. The `pyspark` version I used was *3.5.1* and the `Scala` version was *2.12.18*. Cassandra was used to store the raw API data, which allowed structured querying and aggregation through Spark SQL.

In the ***Data Retrieval*** step, I fetched data month by month using the API since there is time period limitations for each API requestand, and then extracted only the relevant production records. Time columns were converted to the **Europe/Oslo** timezone. All data of 2021 was combined into a pandas DataFrame and then converted into a Spark DataFrame for further analysis.

In the ***Data Processing & Visualization*** step, Spark was used to clean and prepare the data. I created interactive visualizations using `plotly` to explore production trends:

- A **pie chart** showing total annual production by energy group.
- A **line chart** showing hourly production for January.

For setting up ***Mongodb***, I created an account and configured a cluster using MongoDB Atlas-the cloud service. Then, I tried to use the **MongoDB Spark Connector** to write data directly from Spark, but the connection could not be established even after including the `.jar` file and trying multiple configurations. Lastly, I converted the *Spark DataFrame* to a pandas DataFrame and inserted the data into MongoDB using `PyMongo`, which worked reliably. The data was inserted correctly in the database and was ready to be used for the Streamlit part.

In the ***Streamlit App***, since I did not have anything important on page 4, I redesigned that page of the app to visualize the Mongodb data interactively. The layout consists of two columns:

- **Left column:** Allows users to select a price area and view a pie chart of total production.
- **Right column:** Allows filtering by production group and month to view a line chart of hourly production trends.

An `st.expander` briefly documents the data source. MongoDB credentials such as URI, database name and collection name are securely stored in **Streamlit Cloud Secrets Manager**.

I have also cleaned the repository by removing unnecessary .DS_Store files, added them to the `.gitignore`, and updated the `README.md`. This was my first time working with a complete data pipeline starting from API extraction to visualization. Overall, I really enjoyed the challenges and it taught me the importance of version control. and helped me understand big data integration, dependency management, and handling compatibility issues between PySpark, Java, Scala, and NumPy.

AI Usage

ChatGPT was mainly used as a guide when I got stuck or needed clarification. It helped me understand how to convert API timestamps to Oslo time, handle summer/winter time changes, and fix issues when converting API data to a pandas DataFrame due to an incompatible NumPy version.

I also used ChatGPT to learn how to set up a Spark session, extract data from Cassandra, and prepare it for MongoDB. While the Spark–MongoDB connection did not fully work, it helped me understand the process. ChatGPT also guided me on securely reading `.env` secrets to avoid exposing credentials.

For the Streamlit app, I mostly followed the official **Streamlit documentation**, and **IND320 Notebook** to pull data from Mongodb, implement interactive components like `st.columns`,

`st.radio` , `st.pills` , and `st.expander` , and to display plots dynamically based on user selections. AI helped clarify how to implement these UI components,filter and aggregate data for plotting, especially when creating the pie charts and line plots based on selected price areas, production groups, and months. Therefore, it was mainly for clarifying concepts and troubleshooting small issues when combining these features.

Importing all necessary libraries

In [1]:
```
import os
import time
import json
from datetime import datetime, timedelta
from typing import List, Dict
from zoneinfo import ZoneInfo

import requests
import pandas as pd
import plotly.express as px
from dotenv import load_dotenv
from pymongo import MongoClient
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum as spark_sum, month
```

```
/Users/mobashraabeer/miniconda3/envs/D2D_env/lib/python3.12/site-packages/requests/__init__.py:86: RequestsDependencyWarning: Unable to find acceptable character d
etection dependency (chardet or charset_normalizer).
  warnings.warn(
```

Part 1: Fetching Data from Elhub API

The code snippet below sends a `GET` request to the Elhub API to fetch hourly electricity consumption data between May 3, 2023, 20:00 and May 4, 2023, 00:00, and prints the result if the request succeeds. This code was adapted from here and was also used to check whether the **Elhub API endpoint** is working properly and returning data as expected. Some information about the **PRODUCTION_PER_GROUP_MBA_HOUR** dataset:

- Maximum allowed data range: **1 month**
- Access type: **Free** (no authentication tokens or content-type headers are required)
- Filter parameter: **productionGroup**
- Filter values: **solar, hydro, wind, thermal, other**

In [2]:
```
# Defining the base URL
base_url = "https://api.elhub.no/energy-data/v0/price-areas"

# Defining request parameters
params = {'dataset': 'PRODUCTION_PER_GROUP_MBA_HOUR', 'startDate': '2023-05-03T20:00:00+02:00', 'endDate': '2023-05-04T00:00:00+02:00'}

# Defining headers (empty since this API is free to access)
headers = {}

# GET request sent to Elhub API
response = requests.get(base_url, params=params, headers=headers)

# Checking if the request was successful
if response.status_code == 200:
    print(response.json()) # prints the actual data
```

```
else:
    print('Error:', response.status_code)
```


```
{'data': [{'attributes': {'country': 'NO', 'eic': '*', 'name': '*', 'productionPerGroupMbaHour': []}, 'id': '*', 'type': 'price-areas'}, {'attributes': {'country':
'NO', 'eic': '10YNO-1--------2', 'name': 'NO1', 'productionPerGroupMbaHour': [{'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:0
0', 'priceArea': 'NO1', 'productionGroup': 'hydro', 'quantityKwh': 1982521.1, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', '
lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'hydro', 'quantityKwh': 2035407.4, 'startTime': '2023-05-03T21:00:00+02:00'},
{'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'hydro', 'quantityKwh': 1971030.6,
'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'production
Group': 'hydro', 'quantityKwh': 1896238.0, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:4
5:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'other', 'quantityKwh': 32.141, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:
00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'other', 'quantityKwh': 31.101, 'startTime': '2023-05-03T21:00:00+02:0
0'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'other', 'quantityKwh': 28.95,
'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'production
Group': 'other', 'quantityKwh': 29.4, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+0
1:00', 'priceArea': 'NO1', 'productionGroup': 'solar', 'quantityKwh': 2701.154, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00',
'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'solar', 'quantityKwh': 2548.386, 'startTime': '2023-05-03T21:00:00+02:00'},
{'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'solar', 'quantityKwh': 2548.795, '
startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionG
roup': 'solar', 'quantityKwh': 2551.797, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:1
5+01:00', 'priceArea': 'NO1', 'productionGroup': 'thermal', 'quantityKwh': 36405.473, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+0
2:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'thermal', 'quantityKwh': 28021.0, 'startTime': '2023-05-03T21:00:00+
02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'thermal', 'quantityKwh':
41749.9, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'p
roductionGroup': 'thermal', 'quantityKwh': 63721.6, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03
-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'wind', 'quantityKwh': 107711.1, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:
00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'wind', 'quantityKwh': 101932.805, 'startTime': '2023-05-03T2
1:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO1', 'productionGroup': 'wind', 'quantityK
wh': 144433.1, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO
1', 'productionGroup': 'wind', 'quantityKwh': 173616.31, 'startTime': '2023-05-03T23:00:00+02:00'}]}, 'id': 'NO1', 'type': 'price-areas'}, {'attributes': {'countr
y': 'NO', 'eic': '10YNO-2--------T', 'name': 'NO2', 'productionPerGroupMbaHour': [{'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+
01:00', 'priceArea': 'NO2', 'productionGroup': 'hydro', 'quantityKwh': 7373191.5, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:0
0', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'hydro', 'quantityKwh': 7590527.5, 'startTime': '2023-05-03T21:00:00+02:
00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'hydro', 'quantityKwh': 71670
93.5, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'prod
uctionGroup': 'hydro', 'quantityKwh': 6544858.0, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29
T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'other', 'quantityKwh': 32.427, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:0
0+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'other', 'quantityKwh': 9.517, 'startTime': '2023-05-03T21:00:00+0
2:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'other', 'quantityKwh': 9.9
29, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'produc
tionGroup': 'other', 'quantityKwh': 8.215, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:4
5:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'solar', 'quantityKwh': 759.031, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+0
2:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'solar', 'quantityKwh': 172.929, 'startTime': '2023-05-03T21:00:00+0
2:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'solar', 'quantityKwh': 42.
438, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'produ
ctionGroup': 'solar', 'quantityKwh': 28.422, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:
45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'thermal', 'quantityKwh': 18972.123, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:
00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'thermal', 'quantityKwh': 19104.744, 'startTime': '2023-05-03T21:
00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'thermal', 'quantity
Kwh': 19045.812, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': '
NO2', 'productionGroup': 'thermal', 'quantityKwh': 19073.82, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime':
'2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'wind', 'quantityKwh': 218550.17, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-
05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'wind', 'quantityKwh': 206216.97, 'startTime': '2023-
05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO2', 'productionGroup': 'wind', 'qu
antityKwh': 207635.05, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceAr
ea': 'NO2', 'productionGroup': 'wind', 'quantityKwh': 184187.25, 'startTime': '2023-05-03T23:00:00+02:00'}]}, 'id': 'NO2', 'type': 'price-areas'}, {'attributes':
```

{'country': 'NO', 'eic': '10YNO-3--------J', 'name': 'NO3', 'productionPerGroupMbaHour': [{'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'hydro', 'quantityKwh': 1569633.5, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'hydro', 'quantityKwh': 1801392.9, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'hydro', 'quantityKwh': 1724208.0, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'hydro', 'quantityKwh': 1669374.8, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'other', 'quantityKwh': 67.591, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'other', 'quantityKwh': 73.787, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'other', 'quantityKwh': 72.781, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'other', 'quantityKwh': 76.973, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'solar', 'quantityKwh': 191.29, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'solar', 'quantityKwh': 100.702, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'solar', 'quantityKwh': 96.468, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'solar', 'quantityKwh': 101.828, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'thermal', 'quantityKwh': 2.82, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'thermal', 'quantityKwh': 0.84, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'thermal', 'quantityKwh': 5.64, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'thermal', 'quantityKwh': 4.38, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'wind', 'quantityKwh': 232984.84, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'wind', 'quantityKwh': 204451.83, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'wind', 'quantityKwh': 165703.61, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO3', 'productionGroup': 'wind', 'quantityKwh': 148179.36, 'startTime': '2023-05-03T23:00:00+02:00'}]}, 'id': 'NO3', 'type': 'price-areas'}, {'attributes': {'country': 'NO', 'eic': '10YNO-4--------9', 'name': 'NO4', 'productionPerGroupMbaHour': [{'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'hydro', 'quantityKwh': 2978878.0, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'hydro', 'quantityKwh': 3107948.0, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'hydro', 'quantityKwh': 3080650.0, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'hydro', 'quantityKwh': 2860420.8, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'other', 'quantityKwh': 0.0, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'other', 'quantityKwh': 0.0, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'other', 'quantityKwh': 0.0, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'other', 'quantityKwh': 0.0, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'solar', 'quantityKwh': 2.144, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'solar', 'quantityKwh': 0.098, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'solar', 'quantityKwh': 0.006, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'solar', 'quantityKwh': 0.005, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'thermal', 'quantityKwh': 106610.2, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'thermal', 'quantityKwh': 96101.39, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'thermal', 'quantityKwh': 96904.25, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'thermal', 'quantityKwh': 94811.375, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'wind', 'quantityKwh': 300049.7, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'wind', 'quantityKwh': 237206.6, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'wind', 'quantityKwh': 235012.05, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO4', 'productionGroup': 'wind', 'quantityKwh': 227247.95, 'startTime': '2023-05-03T23:00:00+02:00'}]}, 'id': 'NO4', 'type': 'price-areas'}, {'attributes': {'country': 'NO', 'eic': '10Y1001A1001A48H', 'name': 'NO5', 'productionPerGroupMbaHour': [{'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'hydro', 'quantityKwh': 3119588.2, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'hydro', 'quantityKwh': 3007134.8, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'hydro', 'quantityKwh': 2741608.8, 'star

tTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'hydro', 'quantityKwh': 2314271.8, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'other', 'quantityKwh': 2.186, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'other', 'quantityKwh': 0.016, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'other', 'quantityKwh': 0.0, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'other', 'quantityKwh': 0.0, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'solar', 'quantityKwh': 322.071, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'solar', 'quantityKwh': 90.905, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'solar', 'quantityKwh': 85.844, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'solar', 'quantityKwh': 84.296, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'thermal', 'quantityKwh': 18311.0, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'thermal', 'quantityKwh': 17968.0, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'thermal', 'quantityKwh': 17052.0, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'thermal', 'quantityKwh': 17387.0, 'startTime': '2023-05-03T23:00:00+02:00'}, {'endTime': '2023-05-03T21:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'wind', 'quantityKwh': 0.476, 'startTime': '2023-05-03T20:00:00+02:00'}, {'endTime': '2023-05-03T22:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'wind', 'quantityKwh': 0.0, 'startTime': '2023-05-03T21:00:00+02:00'}, {'endTime': '2023-05-03T23:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'wind', 'quantityKwh': 0.0, 'startTime': '2023-05-03T22:00:00+02:00'}, {'endTime': '2023-05-04T00:00:00+02:00', 'lastUpdatedTime': '2025-03-29T01:45:15+01:00', 'priceArea': 'NO5', 'productionGroup': 'wind', 'quantityKwh': 0.0, 'startTime': '2023-05-03T23:00:00+02:00'}]}, 'id': 'NO5', 'type': 'price-areas'}], 'links': {'self': 'https://api.elhub.no/energy-data/v0/price-areas?dataset=PRODUCTION_PER_GROUP_MBA_HOUR&endDate=2023-05-04T00%3A00%3A00%2B02%3A00&startDate=2023-05-03T20%3A00%3A00%2B02%3A00'}, 'meta': {'created': '2025-10-24T13:23:01+02:00', 'lastUpdated': '2025-03-29T01:45:15+01:00'}}

Establishing Spark session with Cassandra connector

This creates a SparkSession configured to connect with a Cassandra database using the Spark–Cassandra Connector. The SparkSession is the entry point to using Apache Spark which allows me to create DataFrames, run SQL queries, and interact with external data sources such as Cassandra. The main coding idea was developed from **here**.

- `SparkSession.builder` is used to configure and create a Spark session.
- `spark.jars.packages", "com.datastax.spark:spark-cassandra-connector_2.12:3.5.1` : Adds the Cassandra connector package to Spark and the library (spark-cassandra-connector) enables Spark to communicate with Cassandra clusters. _2.12 specifies it's compiled for Scala 2.12, and 3.5.1 is the connector's version.

I have further tried to implement `.config("spark.jars", "mongo-spark-connector_2.12-10.1.1.jar")` to write data from Spark to Mongodb but it was not working.

In [3]:
```
# Initializing Spark session with Cassandra connector
spark = SparkSession.builder.appName("SparkCassandraWriteApp").\
     config("spark.jars.packages", "com.datastax.spark:spark-cassandra-connector_2.12:3.5.1").\
     config("spark.cassandra.connection.host", "localhost").\
     config("spark.sql.extensions", "com.datastax.spark.connector.CassandraSparkExtensions").\
     config("spark.sql.catalog.mycatalog", "com.datastax.spark.connector.datasource.CassandraCatalog").\
     config("spark.cassandra.connection.port", "9042").getOrCreate()
```

```
25/10/24 13:23:02 WARN Utils: Your hostname, Mobashras-MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 192.168.11.23 instead (on interface en0)
25/10/24 13:23:02 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Ivy Default Cache set to: /Users/mobashraabeer/.ivy2/cache
The jars for the packages stored in: /Users/mobashraabeer/.ivy2/jars
com.datastax.spark#spark-cassandra-connector_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-ebb9f7f3-ccd4-4705-bc4c-598c0160feff;1.0
        confs: [default]
        found com.datastax.spark#spark-cassandra-connector_2.12;3.5.1 in central
        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 in central
        found org.apache.cassandra#java-driver-core-shaded;4.18.1 in central
        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
:: loading settings :: url = jar:file:/Users/mobashraabeer/miniconda3/envs/D2D_env/lib/python3.12/site-packages/pyspark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
```

```
        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 central
        found org.apache.cassandra#java-driver-query-builder;4.18.1 in central
        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 174ms :: artifacts dl 7ms
        :: modules in use:
        com.datastax.oss#java-driver-shaded-guava;25.1-jre-graal-sub-1 from 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 from central in [default]
        com.datastax.spark#spark-cassandra-connector_2.12;3.5.1 from central 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 [default]
        org.apache.cassandra#java-driver-core-shaded;4.18.1 from central in [default]
        org.apache.cassandra#java-driver-mapper-runtime;4.18.1 from central 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 [default]
        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 central in [default]
        org.slf4j#slf4j-api;1.7.26 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   16  |   0   |   0   |   0   ||   16  |   0   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent-ebb9f7f3-ccd4-4705-bc4c-598c0160feff
        confs: [default]
        0 artifacts copied, 16 already retrieved (0kB/5ms)
25/10/24 13:23:02 WARN NativeCodeLoader: Unable to load native-hadoop library 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 setLogLevel(newLevel).
25/10/24 13:23:02 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
```

Retrieving Hourly Production Data for 2021 from the Elhub API

Here, I am fetching hourly electricity production data from the Elhub API for all price areas *[NO1,NO2, NO3, NO4, NO5]* and production groups *[hydro, other, solar, thermal, wind]* for the year 2021. It is iterating month by month to respect API time period limits. For each API response, it extracts only the **productionPerGroupMbaHour** list, combines all records into a single pandas DataFrame, and converts the date/time columns to Oslo timezone.

The resulting DataFrame is ready for inserting into Cassandra using Spark.

In [4]:
```
# Function to fetch production data from the Elhub API
# This function is fetching electricity production data from Elhub API
# for a given start and end date, and returning a list of production records.
# For this project, it will be the full year of 2021
```

```python
def fetch_elhub_production_data(start_date: str, end_date: str) -> List[Dict]:

    # Defining the base URL for the Elhub API
    base_url = "https://api.elhub.no/energy-data/v0/price-areas"

    # Setting query parameters including dataset name, start and end date
    params = {'dataset': 'PRODUCTION_PER_GROUP_MBA_HOUR', 'startDate': start_date, 'endDate': end_date}

    try:
        # Sending a GET request to the API with a timeout of 30 seconds
        response = requests.get(base_url, params=params, timeout=30)
        # Checking if the response status is okay, otherwise raising an exception
        response.raise_for_status()
        # Converting the response JSON into a Python dictionary
        data = response.json()

        # Preparing an empty list to store all production records
        all_production_records = []

        # Checking if 'data' key exists in the response
        if 'data' in data:
            # Iterating over each price area in the response
            for price_area_data in data['data']:
                # Checking if production records exist for this price area
                if 'attributes' in price_area_data and 'productionPerGroupMbaHour' in price_area_data['attributes']:
                    # Extracting production records for the current price area
                    production_records = price_area_data['attributes']['productionPerGroupMbaHour']
                    # Adding these records to the master list
                    all_production_records.extend(production_records)

        # Printing a warning if no data was found for the given date range
        if not all_production_records:
            print(f"Warning: No production data found for {start_date} to {end_date}")

        # Returning the list of all production records
        return all_production_records

    except requests.exceptions.RequestException as e:
        # Handling any request exceptions and printing an error message
        print(f"Error fetching data for {start_date} to {end_date}: {e}")
        return []

# Function to fetch all production data for 2021
# This function is fetching the electricity production data month by month for the full year of 2021
# since maximum allowed data range is 1 month, then
# combining all records into a single pandas DataFrame, and converting the date columns to Oslo timezone.
def fetch_full_year_2021() -> pd.DataFrame:

    # Preparing an empty list to store all records for the year
    all_records = []

    # Iterating over each month from January to December
    for month in range(1, 13):
        # Defining the start of the current month
        month_start = datetime(2021, month, 1, 0, 0, 0)
```

```
        # Defining the end of the current month
        if month == 12:
            month_end = datetime(2022, 1, 1, 0, 0, 0)  # December ends at start of next year
        else:
            month_end = datetime(2021, month + 1, 1, 0, 0, 0)

        # Converting datetime objects into ISO format strings with timezone offset
        start_str = month_start.strftime('%Y-%m-%dT%H:%M:%S+01:00')
        end_str = month_end.strftime('%Y-%m-%dT%H:%M:%S+01:00')

        # Printing a message indicating which month's data is being fetched
        print(f"Fetching data for {month_start.strftime('%B %Y')}...")

        # Fetching production data for the current month
        records = fetch_elhub_production_data(start_str, end_str)

        # Adding the monthly records to the master list
        all_records.extend(records)

        # Printing the number of records retrieved for the current month
        print(f"  Retrieved {len(records)} records")

        # Adding a small delay to avoid overwhelming the API
        time.sleep(0.5)

    # Converting the list of all records into a pandas DataFrame
    df = pd.DataFrame(all_records)

    # Printing the total number of records retrieved for the year
    print(f"\nTotal records retrieved: {len(df)}")

    # Checking if the DataFrame is not empty
    if not df.empty:
        # Converting the 'startTime', 'endTime', and 'lastUpdatedTime' columns to datetime in Oslo timezone
        if 'startTime' in df.columns:
            df['startTime'] = pd.to_datetime(df['startTime'], utc=True).dt.tz_convert("Europe/Oslo")
        if 'endTime' in df.columns:
            df['endTime'] = pd.to_datetime(df['endTime'], utc=True).dt.tz_convert("Europe/Oslo")
        if 'lastUpdatedTime' in df.columns:
            df['lastUpdatedTime'] = pd.to_datetime(df['lastUpdatedTime'], utc=True).dt.tz_convert("Europe/Oslo")

    # Returning the final DataFrame containing full year data
    return df


# Program Execution
if __name__ == "__main__":
    print("Fetching Elhub production data for all of 2021...")
    print("=" * 60)

    # Fetching full year 2021 data into a pandas DataFrame
    df = fetch_full_year_2021()

    # Checking if any data was retrieved
    if not df.empty:
```

```
        print("\n" + "=" * 60)
        print("Data retrieval complete!")
        # Displaying the shape of the DataFrame
        print(f"Shape: {df.shape}")
        # Displaying the column names
        print(f"\nColumns: {list(df.columns)}")
        # Showing the first few rows
        print(f"\nFirst few records:")
        print(df.head())
        # Displaying the data types of columns
        print(f"\nData types:")
        print(df.dtypes)
        # Showing unique price areas
        print(f"\nPrice areas: {df['priceArea'].unique()}")
        # Showing unique production groups
        print(f"Production groups: {df['productionGroup'].unique()}")

    else:
        print("\nNo data retrieved.")
```

```
Fetching Elhub production data for all of 2021...
============================================================
Fetching data for January 2021...
  Retrieved 17856 records
Fetching data for February 2021...
  Retrieved 16128 records
Fetching data for March 2021...
  Retrieved 17832 records
Fetching data for April 2021...
  Retrieved 17280 records
Fetching data for May 2021...
  Retrieved 17856 records
Fetching data for June 2021...
  Retrieved 17976 records
Fetching data for July 2021...
  Retrieved 18600 records
Fetching data for August 2021...
  Retrieved 18600 records
Fetching data for September 2021...
  Retrieved 18000 records
Fetching data for October 2021...
  Retrieved 18625 records
Fetching data for November 2021...
  Retrieved 18000 records
Fetching data for December 2021...
  Retrieved 18600 records

Total records retrieved: 215353

============================================================
Data retrieval complete!
Shape: (215353, 6)

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

First few records:
```

```
                   endTime           lastUpdatedTime priceArea  \
0 2021-01-01 01:00:00+01:00 2024-12-20 10:35:40+01:00       NO1
1 2021-01-01 02:00:00+01:00 2024-12-20 10:35:40+01:00       NO1
2 2021-01-01 03:00:00+01:00 2024-12-20 10:35:40+01:00       NO1
3 2021-01-01 04:00:00+01:00 2024-12-20 10:35:40+01:00       NO1
4 2021-01-01 05:00:00+01:00 2024-12-20 10:35:40+01:00       NO1

  productionGroup  quantityKwh                 startTime
0           hydro    2507716.8 2021-01-01 00:00:00+01:00
1           hydro    2494728.0 2021-01-01 01:00:00+01:00
2           hydro    2486777.5 2021-01-01 02:00:00+01:00
3           hydro    2461176.0 2021-01-01 03:00:00+01:00
4           hydro    2466969.2 2021-01-01 04:00:00+01:00

Data types:
endTime            datetime64[ns, Europe/Oslo]
lastUpdatedTime    datetime64[ns, Europe/Oslo]
priceArea                               object
productionGroup                         object
quantityKwh                            float64
startTime          datetime64[ns, Europe/Oslo]
dtype: object

Price areas: ['NO1' 'NO2' 'NO3' 'NO4' 'NO5']
Production groups: ['hydro' 'other' 'solar' 'thermal' 'wind']
```

Creating Spark dataframe

In [5]:
```python
# Converting to Spark DataFrame
spark_df = spark.createDataFrame(df)

# Checking if conversion worked
spark_df.show(5)
```

```
25/10/24 13:23:39 WARN TaskSetManager: Stage 0 contains a task of very large size (1395 KiB). The maximum recommended task size is 1000 KiB.

+-------------------+-------------------+---------+--------------+-----------+-------------------+
|            endTime|    lastUpdatedTime|priceArea|productionGroup|quantityKwh|          startTime|
+-------------------+-------------------+---------+--------------+-----------+-------------------+
|2021-01-01 01:00:00|2024-12-20 10:35:40|      NO1|          hydro|  2507716.8|2021-01-01 00:00:00|
|2021-01-01 02:00:00|2024-12-20 10:35:40|      NO1|          hydro|  2494728.0|2021-01-01 01:00:00|
|2021-01-01 03:00:00|2024-12-20 10:35:40|      NO1|          hydro|  2486777.5|2021-01-01 02:00:00|
|2021-01-01 04:00:00|2024-12-20 10:35:40|      NO1|          hydro|  2461176.0|2021-01-01 03:00:00|
|2021-01-01 05:00:00|2024-12-20 10:35:40|      NO1|          hydro|  2466969.2|2021-01-01 04:00:00|
+-------------------+-------------------+---------+--------------+-----------+-------------------+
only showing top 5 rows
```

Renaming the columns of the Spark DataFrame

In [6]:
```python
# Dictionary mapping old column names to new column names
rename_mapping = {
    "priceArea": "price_area",
    "productionGroup": "production_group",
```

```
    "startTime": "start_time",
    "endTime": "end_time",
    "lastUpdatedTime": "last_updated_time",
    "quantityKwh": "value"}

# Renaming columns using a loop
for old_name, new_name in rename_mapping.items():
    spark_df = spark_df.withColumnRenamed(old_name, new_name)
```

Inserting Spark DataFrame into Cassandra

Here, I am inserting the Spark DataFrame `spark_df` into the Cassandra table **production_2021**, which is located in the keyspace **energy_data**.
Before doing this step, I already created the keyspace and table in Cassandra with the following structure:

- **Table name:** `production_2021`
- **Keyspace:** `energy_data`
- **Columns:**
  - `price_area` → `text`
  - `production_group` → `text`
  - `start_time` → `timestamp`
  - `end_time` → `timestamp`
  - `last_updated_time` → `timestamp`
  - `value` → `double`

Explanation of the Code

- **spark_df.write** → starts the process of writing data from the Spark DataFrame.
- **.format("org.apache.spark.sql.cassandra")** → tells Spark that the destination is a Cassandra database.
- **.mode("append")** → ensures that new data is added to the existing table without deleting the previous data.
- **.option("keyspace", "energy_data")** → specifies the Cassandra keyspace where the table is located.
- **.option("table", "production_2021")** → specifies the table name where the data will be stored.
- **.save()** → executes the command and writes the data to Cassandra.

Checking the Data in Cassandra

After running this code, I checked whether the data has been successfully inserted by opening **CQLSH** and using the following command:

```
SELECT * FROM energy_data.production_2021 LIMIT 5;
```

In [7]:
```
spark_df.write.format("org.apache.spark.sql.cassandra").mode("append").\
     option("keyspace", "energy_data").option("table", "production_2021").save()

print("Data inserted into Cassandra!")
```

```
25/10/24 13:23:40 WARN TaskSetManager: Stage 1 contains a task of very large size (1395 KiB). The maximum recommended task size is 1000 KiB.
[Stage 1:>                                                          (0 + 8) / 8]
Data inserted into Cassandra!
```

Part 2 : Visualizing Data from Cassandra using Spark

Now, I am reading data from the Cassandra table `production_2021`, selecting only the relevant columns **price_area, production_group, start_time, value** and filtering for a specific price area *NO1*, and displaying a small sample to verify it.

In [8]:
```
# Defining the price area you want to analyze
# We can replace "NO1" with any other price area from the data

price_area = "NO1"


# Reading data from the Cassandra table 'production_2021' in the keyspace 'energy_data'
# Using Spark's Cassandra connector that we built previously
df = spark.read.format("org.apache.spark.sql.cassandra").\
     option("keyspace", "energy_data").\
     option("table", "production_2021").\
     load().\
     select("price_area", "production_group", "start_time", "value").filter(f"price_area = '{price_area}'")

# Displaying the first 5 rows of the DataFrame
df.show(5)
```

```
+----------+----------------+-------------------+-----+
|price_area|production_group|         start_time|value|
+----------+----------------+-------------------+-----+
|       NO1|           solar|2021-01-01 00:00:00|6.106|
|       NO1|           solar|2021-01-01 01:00:00| 4.03|
|       NO1|           solar|2021-01-01 02:00:00|3.982|
|       NO1|           solar|2021-01-01 03:00:00|8.146|
|       NO1|           solar|2021-01-01 04:00:00|8.616|
+----------+----------------+-------------------+-----+
only showing top 5 rows
```

Creating a Pie Chart for Total Production by Production Group

In this step, I am visualizing the total electricity production for a chosen price area using a pie chart.
As mentioned in the task, here I am

- **grouping the Spark DataFrame by `production_group`** and calculating the total production (`value`) for each group.
- **converting the aggregated Spark DataFrame into a pandas DataFrame** so that Plotly can work with it for plotting.
- **creating an interactive pie chart using Plotly**, where:
  - Each slice represents a different production group.
  - The size of each slice corresponds to the total electricity production of that group.
  - A pastel color palette is applied to make the chart visually appealing.
  - The chart is interactive, allowing us to **select production groups** on click to highlight or isolate them in the output.
- **applying optional styling tweaks**:
  - Showing both percentages and labels on the slices.
  - Pulling slices slightly for better emphasis.
  - Setting the chart width and height.

    - Centering the title at the top and adjusting the font size.
- Finally, I am **displaying the chart**, allowing us to visually inspect how electricity production is distributed among different production groups in the selected price area.

In [9]:
```
# Grouping the Spark DataFrame by 'production_group' and calculating the total production
# We are summing the 'value' column for each production group and aliasing it as 'total_quantity'
agg_df = df.groupBy("production_group").agg(spark_sum("value").alias("total_quantity"))

# Converting the aggregated Spark DataFrame into a pandas DataFrame
# We are doing this because Plotly works directly with pandas DataFrames for plotting
agg_pd = agg_df.toPandas()

# Creating an interactive pie chart using Plotly
# We are setting 'total_quantity' as the values and 'production_group' as the labels
# The chart is titled dynamically with the selected price area
fig = px.pie(agg_pd, values='total_quantity', names='production_group', title=f'Total Production in {price_area} (Year)',
      color_discrete_sequence=px.colors.qualitative.Pastel)

# Applying optional styling tweaks to make the chart more readable
# We are showing both percentages and labels on the chart slices and slightly pulling slices out
fig.update_traces(textinfo='percent+label', pull=[0.05]*len(agg_pd))

# Updating the overall layout of the plot
fig.update_layout(width=800, height=600,title=dict(text=f'Total Production in {price_area} (Year)',
                x=0.5, y=1.0, xanchor='center', yanchor='top'), font=dict(size=12))

# Displaying the interactive pie chart
fig.show()
```

Creating a Line Plot for Hourly Production in January

Now, I am implementing the line plot for **hourly electricity production for January 2021** for a chosen price area using a line chart. Here I have **filtered the Spark DataFrame for January**, selecting only rows where the month of `start_time` equals 1. Then, **converted the filtered Spark DataFrame into a pandas DataFrame** so that Plotly can plot it efficiently.

- For the layout, I am :
    - Setting the width and height of the chart.
    - Adding clear axis labels and a legend title.
    - Using a clean white template for better readability.

Lastly, the chart is displayed allowing interactive exploration of how hourly production varies across different production groups in January for the selected price area.

In [10]:
```
# Filtering the Spark DataFrame for January
# We are selecting only the rows where the month of 'start_time' is 1 (January)
jan_df = df.filter(month("start_time") == 1)

# Converting the filtered Spark DataFrame to a pandas DataFrame
# We are doing this so that Plotly can use it for plotting the line chart
jan_pd = jan_df.toPandas()

# Creating an interactive line chart using Plotly
# We are plotting 'start_time' on the x-axis and 'value' (production quantity) on the y-axis
# Each production group gets a separate line (color-coded)
```

```
fig_line = px.line(jan_pd, x='start_time', y='value',
    color='production_group',  # separate lines for each production group
    title=f'Hourly Production in {price_area} - January 2021',
    labels={'value': 'Quantity (kWh)', 'start_time': 'Date/Time', 'production_group': 'Production Group'})

# Improving the chart layout for readability
# We are setting width, height, axis titles, legend title, and using a clean white template
fig_line.update_layout(width=800, height=600, xaxis_title='Date', yaxis_title='Quantity (kWh)',legend_title='Production Group', template='plotly_white')

# Displaying the interactive line chart
fig_line.show()
```

Inserting Spark DataFrame into MongoDB

Here, I am converting the Spark DataFrame into a pandas DataFrame and then inserting it into MongoDB, since the `MongoDB Spark Connector` did not work for me.

In [11]:
```
# Loading environment variables from the .env file
# We are using load_dotenv() to make the secrets available in the environment
load_dotenv()

# Reading MongoDB credentials and configuration from environment variables
username = os.getenv("MONGO_USER")
password = os.getenv("MONGO_PASS")
cluster = os.getenv("MONGO_CLUSTER")
db_name = os.getenv("MONGO_DB")
collection_name = os.getenv("MONGO_COLLECTION")

# Creating the MongoDB connection URI
uri = f"mongodb+srv://{username}:{password}@{cluster}"

# Connecting to the MongoDB server using PyMongo
# We are creating a client object to interact with the database
client = MongoClient(uri)

# Selecting the database and collection where we want to insert data
collection = client[db_name][collection_name]

# Converting the Spark DataFrame to a pandas DataFrame
pandas_df = spark_df.toPandas()

# Converting the pandas DataFrame into a list of dictionaries
# We are using the 'records' orientation so each row becomes a dictionary
data_dict = json.loads(pandas_df.to_json(orient='records'))

# Inserting the data into the MongoDB collection
# We are using insert_many to add all the documents at once
collection.insert_many(data_dict)

# Printing a confirmation message after successful insertion
print("Data successfully inserted into MongoDB!")
```

```
25/10/24 13:23:47 WARN TaskSetManager: Stage 7 contains a task of very large size (1395 KiB). The maximum recommended task size is 1000 KiB.
Data successfully inserted into MongoDB!
```