Links

- Github repository:https://github.com/Gianna-liu/Weathering_with_you_GegeLiu

For this assignment, I added **production-plot** page for CA2

```
- scripts/: This folder contains the Jupyter Notebook, PDF, and HTML files used for data analysis and documentation.

- streamlitApp/: This folder contains all the files required to develop and run the Streamlit application.

- requirements.txt: This file lists all the Python package dependencies required to run the project, allowing easy installation via pip install -r
requirements.txt.
```

- streamlit App: https://weatheringwithyou-gegeliu.streamlit.app/

Development Log

In this assignment, I developed a complete workflow to retrieve data from an `API`, insert it into a local `Cassandra` database using `Spark`, clean it in `Jupyter Notebook`, and then import the cleaned data into a remote `MongoDB` database. I successfully connected `Streamlit` to MongoDB to visualize the data. Overall, the workflow was smooth, and following the lectures helped me understand and test each step in detail.

Using Spark for data operations provided a valuable opportunity to become familiar with its syntax and its capability to handle large datasets efficiently. During the data cleaning process, I performed basic descriptive statistics and found no missing or duplicate values, although potential outliers were not yet analyzed and could be addressed in future work. While working with MongoDB, I accidentally uploaded the secrets.toml file to GitHub and received a warning. Since this was only test data, I deleted it immediately and plan to manage secrets more securely in the future.

One challenge arose when fetching data from the API. The API limits retrieval to one month at a time, and October data initially failed due to the Daylight Saving Time transition, which added an extra hour. I resolved this by splitting October into two parts, and ChatGPT provided helpful guidance for handling DST-related time conversion issues. To avoid accidentally calling the API or database when generating HTML reports, I also added execution flags in the notebook, which allowed safe generation of static outputs.

Finally, creating visualizations took considerable effort, particularly for layout adjustments. Initially, I used Matplotlib, but very small slices in pie charts caused overlapping labels. Switching to Plotly resolved this issue and improved readability. Completing this entire workflow allowed me to gain hands-on experience with data extraction, processing, database management, and visualization, making the assignment both challenging and highly rewarding.

AI usage

During this assignment, I used GitHub Copilot in my VS Code environment to help review and debug code. When retrieving data from the API,

I referred to code examples to handle issues related to Daylight Saving Time (DST). While creating visualizations in Streamlit with Plotly, overlapping elements sometimes occurred, so I consulted ChatGPT examples to optimize chart layouts. Since this was also my first time using Docker,

I referred to ChatGPT-provided initialization code when creating the local database image. Additionally, for the requirements.txt file,

I consulted ChatGPT to identify unnecessary packages. Although my web app ran successfully locally, it failed to deploy on GitHub, and ChatGPT pointed out that some packages, such as Cassandra, were not needed.

Tasks

Library imports

In [1]:
```
EXECUTE_API = True  # Avoid execute api when convert into html file
EXECUTE_localDB = True   # Avoid execute database when convert into html file
EXECUTE_remoteDB = False

import json
import os
from pyjstat import pyjstat
import requests
from cassandra.cluster import Cluster

from pyspark.sql import SparkSession
from pyspark.sql.functions import sum, col, to_date, hour, lit, when, isnan, count

import pandas as pd
import pytz
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.io as pio
pio.renderers.default = "notebook_connected"

from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
```

Step1: Load data with API and insert the data to Cassandra with Spark

1.preparation in Cassandra

Connect to the Cassandra cluster from Python.

In [2]:
```
# Connecting to Cassandra
cluster = Cluster(['localhost'], port=9042)
session = cluster.connect()
```

Set up new keyspace

In [3]:
```
if EXECUTE_localDB:
    session.execute("CREATE KEYSPACE IF NOT EXISTS elnub WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };")
```

Create a table

- IF NOT EXISTS makes sure we do not overwrite existing tables

In [4]:
```
# Create a new table (first time only)
###!For a composite primary key like PRIMARY KEY ((priceArea, productionGroup), startTime), you don’t need to create a separate combined column.
if EXECUTE_localDB:
    session.set_keyspace('elnub')
    session.execute("DROP TABLE IF EXISTS elnub.production_data;")
    session.execute("CREATE TABLE IF NOT EXISTS elnub.production_data (\
        priceArea TEXT,\
        productionGroup TEXT,\
        startTime TIMESTAMP,\
        quantityKwh DOUBLE,\
```

```
        endTime TIMESTAMP,\
        lastUpdatedTime TIMESTAMP,\
        PRIMARY KEY ((priceArea, productionGroup), startTime));")
```

2.Preparation with Spark

Set environment variables for PySpark

In [5]:
```
os.environ["JAVA_HOME"] = "/opt/homebrew/Cellar/openjdk@17/17.0.17/libexec/openjdk.jdk/Contents/Home"

os.environ["PYSPARK_PYTHON"] = "python"
os.environ["PYSPARK_DRIVER_PYTHON"] = "python"
```

Create a spark session to transfer data

In [6]:
```
spark = SparkSession.builder.appName('SparkCassandraApp').\
    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 15:34:31 WARN Utils: Your hostname, FlyNorth.local resolves to a loopback address: 127.0.0.1; using 192.168.68.110 instead (on interface en0)
25/10/24 15:34:31 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Ivy Default Cache set to: /Users/liugege/.ivy2/cache
The jars for the packages stored in: /Users/liugege/.ivy2/jars
com.datastax.spark#spark-cassandra-connector_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-47d28952-0fcb-4f1b-95f9-72f9e43eabb2;1.0
        confs: [default]
:: loading settings :: url = jar:file:/opt/miniconda3/envs/D2D_project/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 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
        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 357ms :: artifacts dl 18ms
        :: 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-47d28952-0fcb-4f1b-95f9-72f9e43eabb2
        confs: [default]
        0 artifacts copied, 16 already retrieved (0kB/13ms)
25/10/24 15:34:31 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 15:34:32 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
25/10/24 15:34:32 WARN Utils: Service 'SparkUI' could not bind on port 4041. Attempting port 4042.
```

3.Retrieve data with Elhub API

Create function to enhance code reusability

In [7]:
```python
def get_period_start_end(year, m):
    '''
    This function is used to generate a whole month time interval
    corresponding to the Norwegian time zone for the specified year and month.
    '''
    norway_tz = pytz.timezone("Europe/Oslo")

    start_date = norway_tz.localize(datetime(year, m, 1))
    if m == 12:
        end_date = norway_tz.localize(datetime(year + 1, 1, 1)) - timedelta(hours=1)
    else:
        end_date = norway_tz.localize(datetime(year, m + 1, 1)) - timedelta(hours=1)
    return start_date, end_date
```

In [8]:
```python
def load_parse_production_data(start_dt, end_dt):
    '''
    This function is used to obtain production data for a defined time period from the Elhub API
```

```
    and parse the data into a list.
    '''
    url = "https://api.elhub.no/energy-data/v0/price-areas"
    params = {
        "dataset": "PRODUCTION_PER_GROUP_MBA_HOUR",
        "startDate": start_dt.isoformat(),
        "endDate": end_dt.isoformat()
    }
    response = requests.get(url, params=params)
    if response.status_code != 200:
        print(f"Failed to get data for {start_dt} - {end_dt}: status {response.status_code}")
        return []

    data_json = response.json()
    parsed_data = []
    for data in data_json['data']:
        for item in data['attributes']['productionPerGroupMbaHour']:
            parsed_data.append(item)
    return parsed_data
```

In [9]:
```
def write_to_cassandra_ved_spark(data_list, table="production_data", keyspace="elnub"):
    '''
    This function writes a list of electricity production data to a local Cassandra database with spark.
    '''
    if not data_list:
        return

    df = pd.DataFrame(data_list)
    # define the low-case columns
    df.columns = ['endtime','lastupdatedtime','pricearea','productiongroup','quantitykwh','starttime']

    # Use spark to insert data into cassandra
    spark.createDataFrame(df)\
        .write\
        .format("org.apache.spark.sql.cassandra")\
        .options(table=table, keyspace=keyspace)\
        .mode("append")\
        .save()
```

Loop every month to get the whole year data

In [10]:
```
start_year = 2021
end_year = 2021

if EXECUTE_API:
    for m in range(1, 13):
        start_dt, end_dt = get_period_start_end(start_year, m)
        # Special handling for October DST change
        if m == 10:
            norway_tz = pytz.timezone("Europe/Oslo")
            '''
            The API limits retrieval to one month at a time, and October data initially failed
            due to the Daylight Saving Time transition, which added an extra hour.
            I resolved this by splitting October into two parts.
            '''
            end_first_part = norway_tz.localize(datetime(start_year, 10, 30, 23))
            parts = [(start_dt, end_first_part),
                    (norway_tz.localize(datetime(start_year, 10, 31, 0)),
                    norway_tz.localize(datetime(start_year, 10, 31, 23)))]
```

```
        else:
            parts = [(start_dt, end_dt)]

        for start_part, end_part in parts:
            data_list = load_parse_production_data(start_part, end_part)
            write_to_cassandra_ved_spark(data_list)

        print(f"Month {m} finished.")
```

```
[Stage 0:>                                                          (0 + 8) / 8]
/opt/miniconda3/envs/D2D_project/lib/python3.12/site-packages/pyspark/python/lib/pyspark.zip/pyspark/daemon.py:154: DeprecationWarning: This process (pid=90389) is multi-threaded, use of fork() may lead to deadlocks in the child.

Month 1 finished.
Month 2 finished.
Month 3 finished.
Month 4 finished.
Month 5 finished.
Month 6 finished.
Month 7 finished.
Month 8 finished.
[Stage 8:======================>                                    (3 + 5) / 8]

Month 9 finished.
Month 10 finished.
Month 11 finished.
Month 12 finished.
```

Step2: Extract data with Spark and plot figures

load data from cassandra to notebook

In [11]:
```
df_from_cassandra = spark.read \
    .format("org.apache.spark.sql.cassandra") \
    .options(table="production_data", keyspace="elnub") \
    .load()

df_from_cassandra.show(5)
```

```
+---------+---------------+-------------------+-------------------+-------------------+-----------+
|pricearea|productiongroup|          starttime|            endtime|    lastupdatedtime|quantitykwh|
+---------+---------------+-------------------+-------------------+-------------------+-----------+
|      NO1|           wind|2021-01-01 00:00:00|2021-01-01 01:00:00|2024-12-20 10:35:40|    937.072|
|      NO1|           wind|2021-01-01 01:00:00|2021-01-01 02:00:00|2024-12-20 10:35:40|    649.068|
|      NO1|           wind|2021-01-01 02:00:00|2021-01-01 03:00:00|2024-12-20 10:35:40|      144.0|
|      NO1|           wind|2021-01-01 03:00:00|2021-01-01 04:00:00|2024-12-20 10:35:40|     217.07|
|      NO1|           wind|2021-01-01 04:00:00|2021-01-01 05:00:00|2024-12-20 10:35:40|    505.071|
+---------+---------------+-------------------+-------------------+-------------------+-----------+
only showing top 5 rows
```

extract the required columns

In [12]:
```
df_spark = df_from_cassandra.select("pricearea", "productiongroup", "starttime", "quantitykwh")
df_spark.columns
```

Out[12]:
```
['pricearea', 'productiongroup', 'starttime', 'quantitykwh']
```

Explore the dataset briefly

In [13]:
```
df_spark.describe().show()
```

```
[Stage 14:==========>                                              (3 + 8) / 17]
[Stage 14:=======================================>                (12 + 5) / 17]

[Stage 14:==================================================>     (15 + 2) / 17]
+-------+---------+---------------+-----------------+
|summary|pricearea|productiongroup|      quantitykwh|
+-------+---------+---------------+-----------------+
|  count|   215033|         215033|           215033|
|   mean|     NULL|           NULL|729742.5154550395|
| stddev|     NULL|           NULL|1549796.606412848|
|    min|      NO1|          hydro|              0.0|
|    max|      NO5|           wind|        9715193.0|
+-------+---------+---------------+-----------------+
```

In [14]:
```
df_spark.printSchema()
```

```
root
 |-- pricearea: string (nullable = false)
 |-- productiongroup: string (nullable = false)
 |-- starttime: timestamp (nullable = true)
 |-- quantitykwh: double (nullable = true)
```

In [15]:
```
total_rows = df_spark.count()
print(f"Total rows in Spark DataFrame: {total_rows}")
```

```
Total rows in Spark DataFrame: 215033
```

The wind group has less records than other four groups.

In [16]:
```
df_spark.groupBy("productiongroup", "pricearea")\
    .count()\
    .orderBy("productiongroup", "pricearea", ascending=True)\
    .show(50)
```

```
+---------------+---------+-----+
|productiongroup|pricearea|count|
+---------------+---------+-----+
|          hydro|      NO1| 8747|
|          hydro|      NO2| 8747|
|          hydro|      NO3| 8747|
|          hydro|      NO4| 8747|
|          hydro|      NO5| 8747|
|          other|      NO1| 8747|
|          other|      NO2| 8747|
|          other|      NO3| 8747|
|          other|      NO4| 8747|
|          other|      NO5| 8747|
|          solar|      NO1| 8747|
|          solar|      NO2| 8747|
|          solar|      NO3| 8747|
|          solar|      NO4| 8747|
|          solar|      NO5| 8747|
|        thermal|      NO1| 8747|
|        thermal|      NO2| 8747|
|        thermal|      NO3| 8747|
|        thermal|      NO4| 8747|
|        thermal|      NO5| 8747|
|           wind|      NO1| 8747|
|           wind|      NO2| 8747|
|           wind|      NO3| 8747|
|           wind|      NO4| 8747|
|           wind|      NO5| 5105|
+---------------+---------+-----+
```

```
[Stage 20:========================================>                (12 + 5) / 17]
```

THere are not any null or duplicate value

In [17]:
```
df_clean = df_spark.dropna(subset=['starttime', 'quantitykwh', 'pricearea', 'productiongroup'])
df_clean = df_clean.dropDuplicates(['pricearea', 'productiongroup', 'starttime'])

df_clean.count()
```

```
[Stage 23:======>                                                   (2 + 8) / 17]
[Stage 23:===========================>                              (8 + 8) / 17]

[Stage 23:========================================>                (12 + 5) / 17]
[Stage 23:===============================================>         (14 + 3) / 17]
```

Out[17]: 215033

Plot1: A pie chart

Use the `plotly` module to get a clearer pie chart, especially for categories that account for a small proportion in the pei chart

In [18]:
```
# Filter and aggregate data
df_sub1 = df_spark.filter(df_spark.pricearea == 'NO1') \
    .groupBy('productiongroup') \
    .agg(sum('quantitykwh') \
```

```
    .alias('total_kwh'))

df_sub1_pd = df_sub1.toPandas()

# Sort the values
df_sub1_pd.sort_values(by='total_kwh', ascending=False)
```

```
[Stage 29:=====================================================>   (16 + 1) / 17]
```

Out[18]:

	productiongroup	total_kwh
3	hydro	1.833124e+10
1	wind	5.464368e+08
2	thermal	2.357448e+08
0	solar	1.438160e+07
4	other	5.255678e+04

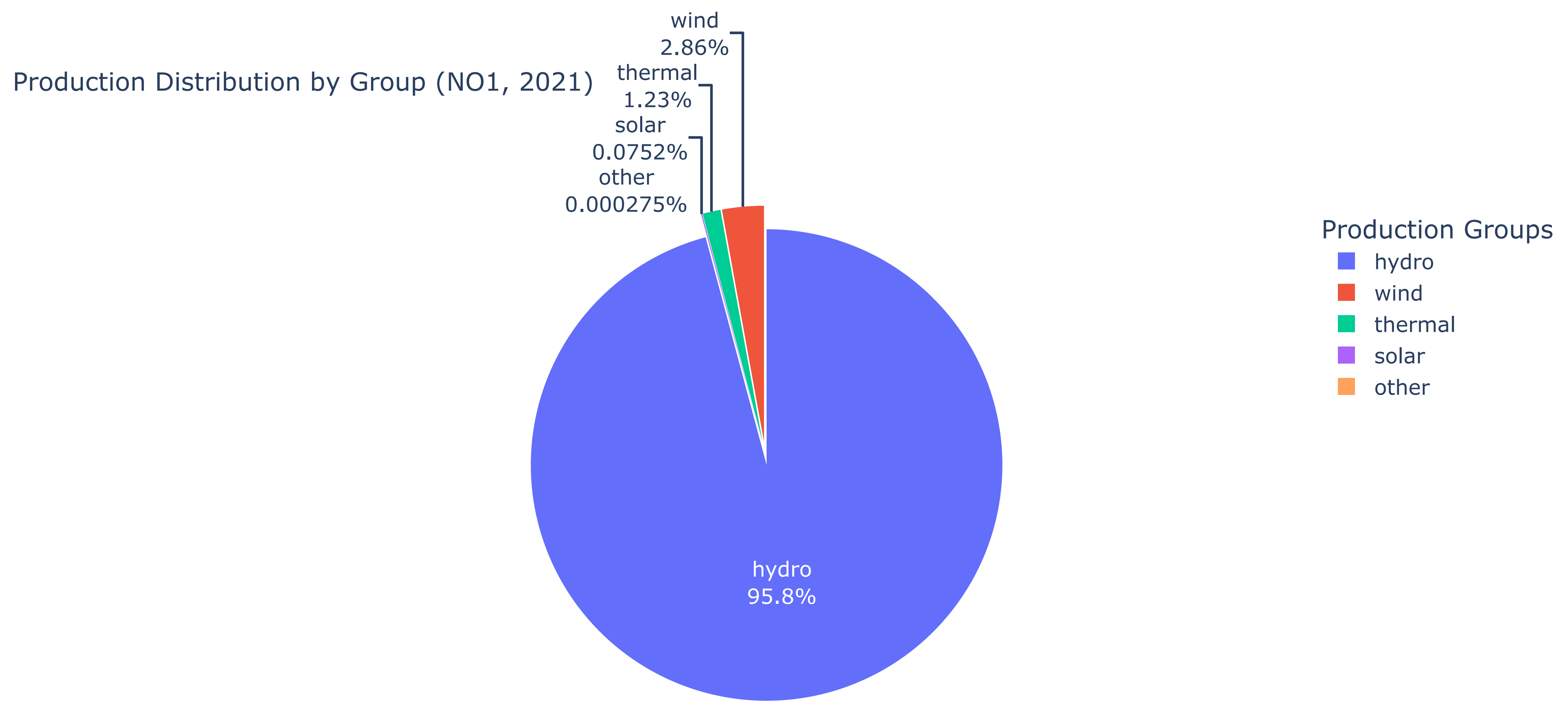
In [19]:

```
# Use the pie function to plot

fig = px.pie(
    df_sub1_pd,
    values='total_kwh',
    names='productiongroup',
    title=f"Production Distribution by Group (NO1, 2021)",
)

# Modify the structure of the plot
fig.update_traces(
    textinfo='percent+label', # display the percent and label for each pie
    pull=[0.05]*len(df_sub1_pd), # seperate each pie slightly
    textfont_size=15, # define the size of text
)

# Define the layout of the whole plot, like the layout of the legend, title
fig.update_layout(
    width=1200,
    height=600,
    legend_title_text='Production Groups',
    legend=dict(
        font=dict(size=15)
    ),
    title=dict(
        font=dict(size=18)
    )
)
# show the plot
fig.show()
```



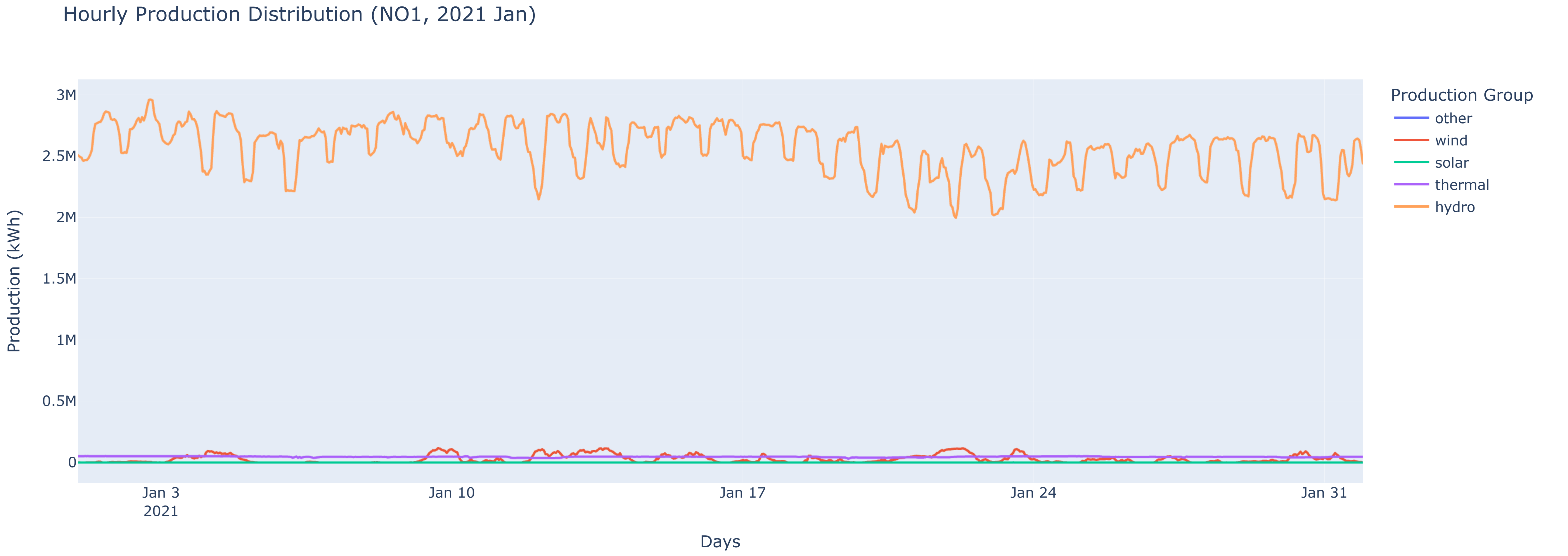
Most of the electricity in the selected price area comes from hydropower, producing about 18.3 billion kWh (95.8%). Wind and thermal power contribute much less, around 0.55 billion kWh (2.86%) and 0.24 billion kWh (1.26%), respectively. Solar and other sources make only a tiny contribution, with solar at 14.4 million kWh (0.08%) and other sources at 52.6 thousand kWh (0.003%).

Plot2: A line plot

In [20]:
```
from pyspark.sql.functions import col, month
# Filter data
df_sub2 = (
    df_spark
    .filter((col("pricearea") == "NO1") & (month(col("starttime")) == 1))
)
# Convert to Pandas for plotting
df_sub2_pd = df_sub2.toPandas()
```

In [21]:
```
# use the line function to plot and each line represent each production group
fig = px.line(
    df_sub2_pd,
    x="starttime",
    y="quantitykwh",
    color="productiongroup",
    title="Hourly Production Distribution (NO1, 2021 Jan)",
    labels={ # rename the lanels
```

```
        "starttime": "Days",
        "quantitykwh": "Production (kWh)",
        "productiongroup": "Production Group"
    }
)
fig.show()
```



For the line chart, hydropower production shows a fairly regular daily pattern, with lower output in the early morning and higher output in the afternoon. Solar production is irregular, with large peaks and troughs, which is normal given its strong dependence on weather conditions. Thermal production also fluctuates, but not dramatically. Wind production exhibits frequent and significant fluctuations. Other sources are extremely unstable, producing only a very small amount of electricity for a single hour.

Step3: Push data into Mongodb

In [22]:
```
if EXECUTE_remoteDB:
    with open("config_local.json") as f:
        config = json.load(f)

    mongo_uri = config["mongo_uri"]

    # Create a new client and connect to the server
    client = MongoClient(uri, server_api=ServerApi('1'))
    # Send a ping to confirm a successful connection
    try:
        client.admin.command('ping')
        print("Pinged your deployment. You successfully connected to MongoDB!")
    except Exception as e:
```

```
        print(e)

    # connect to the Mongodb
    db = client["elhub_db"]
    collection = db["production_data"]

    df = df_spark.toPandas()

    data_dict = df.to_dict(orient="records")

    # insert the data
    collection.insert_many(data_dict)

    print(f"{len(data_dict)} records inserted into MongoDB.")
```

Close the Spark session

In [23]:
```
spark.stop()
```