Notebook_ca2

October 23, 2025

1 IND320 Course Project, Part 2

1.1 Code access and direct links

- The project is deployed here: ind320-henrikengdal-project
- The code is accessible at the repository: henrikengdal/ind320-henrikengdal-project

1.2 AI Usage

AI plays a multifaceted role throughout this project, primarily serving as an assistant and analytical tool. The project leverages AI in several areas:

Development and Code Generation: AI assists in writing and optimizing code for the application.

Data Analysis and Insights: AI helps analyze data patterns and identifying trends. It assists in generating meaningful statistical summaries and suggesting appropriate visualization techniques for the given data.

Documentation and Communication: AI supports the creation of clear documentation, such as code comments, and user interface text. It helps structure the project documentation and ensures technical concepts are communicated effectively.

Problem-Solving and Debugging: Throughout the development process, AI serves as a coding companion, helping troubleshoot issues, optimize data processing workflows, and suggesting best practices.

1.3 01_Loading the data from the Elhub API

[30]:
```
import requests
import json
from datetime import datetime, timedelta

# Function to fetch data for a specific month
def fetch_month_data(year, month):
    start_date = datetime(year, month, 1)

    # Calculate last day of month
    if month == 12:
        end_date = datetime(year + 1, 1, 1) - timedelta(days=1)
    else:
```

```
        end_date = datetime(year, month + 1, 1) - timedelta(days=1)

    # Format dates for API (URL encoded)
    start_str = start_date.strftime('%Y-%m-%dT00:00:00+02:00').replace(':',␣
↪'%3A').replace('+', '%2B')
    end_str = end_date.strftime('%Y-%m-%dT23:59:59+02:00').replace(':', '%3A').
↪replace('+', '%2B')

    # Build URL
    url = f"https://api.elhub.no/energy-data/v0/price-areas?
↪dataset=PRODUCTION_PER_GROUP_MBA_HOUR&startDate={start_str}&endDate={end_str}"

    print(f"Fetching data for {start_date.strftime('%B %Y')}...")

    try:
        response = requests.get(url)
        response.raise_for_status()
        return response.json()
    except Exception as e:
        print(f"Error fetching {start_date.strftime('%B %Y')}: {e}")
        return None

# Fetch data for all 12 months of 2021
all_data = []
combined_response = None

for month in range(1, 13):
    month_data = fetch_month_data(2021, month)

    if month_data and 'data' in month_data:
        if combined_response is None:
            # First month - use as template
            combined_response = month_data
        else:
            # Add data from subsequent months
            # Combine the productionPerGroupMbaHour arrays for each price area
            for new_area in month_data['data']:
                # Find matching area in combined data
                area_found = False
                for existing_area in combined_response['data']:
                    if existing_area['attributes']['name'] ==␣
↪new_area['attributes']['name']:
                        # Append production data
                        ␣
↪existing_area['attributes']['productionPerGroupMbaHour'].extend(
                            new_area['attributes']['productionPerGroupMbaHour']
                        )
```

```
                        area_found = True
                        break

                # If area not found, add it
                if not area_found:
                    combined_response['data'].append(new_area)

        print(f"{datetime(2021, month, 1).strftime('%B')} - Fetched␣
 ↪successfully")
    else:
        print(f"{datetime(2021, month, 1).strftime('%B')} - No data received")

# Save the combined response to a JSON file
if combined_response:
    with open('./assets/production_data.json', 'w') as f:
        json.dump(combined_response, f, indent=4)

    # Calculate total records
    total_records = sum(
        len(area['attributes']['productionPerGroupMbaHour'])
        for area in combined_response['data']
    )

    print(f"\n Successfully saved all 2021 data!")
    print(f"Total records: {total_records:,}")
    print(f"File saved to: ./assets/production_data.json")
else:
    print("  No data was fetched. Please check the API or your internet␣
 ↪connection.")
```

```
Fetching data for January 2021…
January - Fetched successfully
Fetching data for February 2021…
January - Fetched successfully
Fetching data for February 2021…
February - Fetched successfully
Fetching data for March 2021…
February - Fetched successfully
Fetching data for March 2021…
March - Fetched successfully
Fetching data for April 2021…
March - Fetched successfully
Fetching data for April 2021…
April - Fetched successfully
Fetching data for May 2021…
April - Fetched successfully
Fetching data for May 2021…
May - Fetched successfully
```

```
Fetching data for June 2021…
May - Fetched successfully
Fetching data for June 2021…
June - Fetched successfully
Fetching data for July 2021…
June - Fetched successfully
Fetching data for July 2021…
July - Fetched successfully
Fetching data for August 2021…
July - Fetched successfully
Fetching data for August 2021…
August - Fetched successfully
Fetching data for September 2021…
August - Fetched successfully
Fetching data for September 2021…
September - Fetched successfully
Fetching data for October 2021…
September - Fetched successfully
Fetching data for October 2021…
October - Fetched successfully
Fetching data for November 2021…
October - Fetched successfully
Fetching data for November 2021…
November - Fetched successfully
Fetching data for December 2021…
November - Fetched successfully
Fetching data for December 2021…
December - Fetched successfully
December - Fetched successfully

 Successfully saved all 2021 data!
Total records: 215,353
File saved to: ./assets/production_data.json

 Successfully saved all 2021 data!
Total records: 215,353
File saved to: ./assets/production_data.json
```

1.3.1 Converting the data to a Dataframe

[31]:
```python
# Convert the JSON data into a pandas DataFrame
import pandas as pd

# Load the JSON file
with open('./assets/production_data.json', 'r') as f:
    data = json.load(f)
```

```
records = []
for item in data['data']:
    for production in item['attributes']['productionPerGroupMbaHour']:
        records.append(production)

# Create DataFrame
df = pd.DataFrame(records)

# Convert time columns to datetime
df['startTime'] = pd.to_datetime(df['startTime'])
df['endTime'] = pd.to_datetime(df['endTime'])
df['lastUpdatedTime'] = pd.to_datetime(df['lastUpdatedTime'])

print(df.head())
```

```
/var/folders/cb/h4grq88s6sjgm0cnxyzjm7xw0000gn/T/ipykernel_91557/1732238392.py:1
8: FutureWarning: In a future version of pandas, parsing datetimes with mixed
time zones will raise an error unless `utc=True`. Please specify `utc=True` to
opt in to the new behaviour and silence this warning. To create a `Series` with
mixed offsets and `object` dtype, please use `apply` and
`datetime.datetime.strptime`
  df['startTime'] = pd.to_datetime(df['startTime'])
```

	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

```
/var/folders/cb/h4grq88s6sjgm0cnxyzjm7xw0000gn/T/ipykernel_91557/1732238392.py:1
9: FutureWarning: In a future version of pandas, parsing datetimes with mixed
time zones will raise an error unless `utc=True`. Please specify `utc=True` to
opt in to the new behaviour and silence this warning. To create a `Series` with
mixed offsets and `object` dtype, please use `apply` and
`datetime.datetime.strptime`
  df['endTime'] = pd.to_datetime(df['endTime'])
/var/folders/cb/h4grq88s6sjgm0cnxyzjm7xw0000gn/T/ipykernel_91557/1732238392.py:2
0: FutureWarning: In a future version of pandas, parsing datetimes with mixed
time zones will raise an error unless `utc=True`. Please specify `utc=True` to
opt in to the new behaviour and silence this warning. To create a `Series` with
```

```
mixed offsets and `object` dtype, please use `apply` and
`datetime.datetime.strptime`
  df['lastUpdatedTime'] = pd.to_datetime(df['lastUpdatedTime'])
```

1.3.2 Setting up Cassandra and Spark

[32]:
```
# Connecting to Cassandra
from cassandra.cluster import Cluster
cluster = Cluster(['localhost'], port=9042)
session = cluster.connect()
```

[33]:
```
# Create keyspace if it doesn't exist
session.execute("CREATE KEYSPACE IF NOT EXISTS ca2_production_data_keyspace␣
 ↪WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
 ↪")
```

[33]:
```
<cassandra.cluster.ResultSet at 0x324c477d0>
```

[34]:
```
# Create a new table (first time only)
session.set_keyspace('ca2_production_data_keyspace')
session.execute("DROP TABLE IF EXISTS ca2_production_data_keyspace.
 ↪my_first_table;") # Starting from scratch every time
session.execute("CREATE TABLE IF NOT EXISTS ca2_production_data_keyspace.
 ↪my_first_table (ind int PRIMARY KEY, company text, model text);")
```

[34]:
```
<cassandra.cluster.ResultSet at 0x324ce9390>
```

[35]:
```
import os
os.environ["JAVA_HOME"] = "/opt/homebrew/opt/openjdk@17/libexec/openjdk.jdk/
 ↪Contents/Home"
os.environ["PYSPARK_PYTHON"] = "python"
print(f"JAVA_HOME set to: {os.environ['JAVA_HOME']}")

# Verify Java is accessible
import subprocess
try:
    result = subprocess.run([f"{os.environ['JAVA_HOME']}/bin/java", "-version"],
                            capture_output=True, text=True, timeout=5)
    print(f"Java version: {result.stderr.split(chr(10))[0]}")
except Exception as e:
    print(f"Error checking Java: {e}")
```

```
JAVA_HOME set to: /opt/homebrew/opt/openjdk@17/libexec/openjdk.jdk/Contents/Home
Java version: openjdk version "17.0.16" 2025-07-15
```

[36]:
```
from pyspark.sql import SparkSession

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()
```

1.4 02_Use Spark to extract columns

[37]:
```
# Convert pandas DataFrame to Spark DataFrame
spark_df = spark.createDataFrame(df)

# Show the schema
spark_df.printSchema()

# Display first few rows
spark_df.show(5)

# Select specific columns
selected_columns = spark_df.select('priceArea', 'productionGroup',␣
 ↪'quantityKwh', 'startTime')
selected_columns.show(20)

# SPARK TRANSFORMATIONS AND AGGREGATIONS
# For example, group by production type and sum the quantity
from pyspark.sql.functions import sum as spark_sum, col, avg as spark_avg,␣
 ↪count, max as spark_max

# Aggregation 1: Total production by group and price area
production_summary = spark_df.groupBy('productionGroup', 'priceArea') \
    .agg(spark_sum('quantityKwh').alias('total_production')) \
    .orderBy(col('total_production').desc())

production_summary.show()

# Aggregation 2: Additional statistics per production group
production_stats = spark_df.groupBy('productionGroup') \
    .agg(
        spark_sum('quantityKwh').alias('total_kwh'),
        spark_avg('quantityKwh').alias('avg_kwh'),
        spark_max('quantityKwh').alias('max_kwh'),
        count('*').alias('record_count')
    ) \
    .orderBy(col('total_kwh').desc())
```

```python
print("\nProduction Statistics by Group:")
production_stats.show()

# Convert Spark results to pandas for visualization
production_summary_pd = production_summary.toPandas()
production_stats_pd = production_stats.toPandas()

print("\n  Spark aggregations completed")
print(f"Summary records: {len(production_summary_pd)}")
print(f"Stats records: {len(production_stats_pd)}")
print("\nSample summary data:")
print(production_summary_pd.head(10))
```

```
root
 |-- endTime: struct (nullable = true)
 |-- lastUpdatedTime: struct (nullable = true)
 |-- priceArea: string (nullable = true)
 |-- productionGroup: string (nullable = true)
 |-- quantityKwh: double (nullable = true)
 |-- startTime: struct (nullable = true)

+-------+---------------+---------+---------------+-----------+---------+
|endTime|lastUpdatedTime|priceArea|productionGroup|quantityKwh|startTime|
+-------+---------------+---------+---------------+-----------+---------+
|     {}|             {}|      NO1|          hydro|  2507716.8|       {}|
|     {}|             {}|      NO1|          hydro|  2494728.0|       {}|
|     {}|             {}|      NO1|          hydro|  2486777.5|       {}|
|     {}|             {}|      NO1|          hydro|  2461176.0|       {}|
|     {}|             {}|      NO1|          hydro|  2466969.2|       {}|
+-------+---------------+---------+---------------+-----------+---------+
only showing top 5 rows
+---------+---------------+-----------+---------+
|priceArea|productionGroup|quantityKwh|startTime|
+---------+---------------+-----------+---------+
|      NO1|          hydro|  2507716.8|       {}|
|      NO1|          hydro|  2494728.0|       {}|
|      NO1|          hydro|  2486777.5|       {}|
|      NO1|          hydro|  2461176.0|       {}|
|      NO1|          hydro|  2466969.2|       {}|
|      NO1|          hydro|  2467460.0|       {}|
|      NO1|          hydro|  2482320.8|       {}|
|      NO1|          hydro|  2509533.0|       {}|
|      NO1|          hydro|  2550758.2|       {}|
|      NO1|          hydro|  2693111.0|       {}|
|      NO1|          hydro|  2762854.8|       {}|
|      NO1|          hydro|  2767902.2|       {}|
|      NO1|          hydro|  2777353.2|       {}|
|      NO1|          hydro|  2779136.0|       {}|
```

```
|      NO1|          hydro|  2804195.0|       {}|
|      NO1|          hydro|  2846295.5|       {}|
|      NO1|          hydro|  2864646.2|       {}|
|      NO1|          hydro|  2859661.0|       {}|
|      NO1|          hydro|  2854212.2|       {}|
|      NO1|          hydro|  2802948.2|       {}|
+---------+---------------+-----------+---------+
only showing top 20 rows
+---------+---------------+-----------+---------+
|priceArea|productionGroup|quantityKwh|startTime|
+---------+---------------+-----------+---------+
|      NO1|          hydro|  2507716.8|       {}|
|      NO1|          hydro|  2494728.0|       {}|
|      NO1|          hydro|  2486777.5|       {}|
|      NO1|          hydro|  2461176.0|       {}|
|      NO1|          hydro|  2466969.2|       {}|
|      NO1|          hydro|  2467460.0|       {}|
|      NO1|          hydro|  2482320.8|       {}|
|      NO1|          hydro|  2509533.0|       {}|
|      NO1|          hydro|  2550758.2|       {}|
|      NO1|          hydro|  2693111.0|       {}|
|      NO1|          hydro|  2762854.8|       {}|
|      NO1|          hydro|  2767902.2|       {}|
|      NO1|          hydro|  2777353.2|       {}|
|      NO1|          hydro|  2779136.0|       {}|
|      NO1|          hydro|  2804195.0|       {}|
|      NO1|          hydro|  2846295.5|       {}|
|      NO1|          hydro|  2864646.2|       {}|
|      NO1|          hydro|  2859661.0|       {}|
|      NO1|          hydro|  2854212.2|       {}|
|      NO1|          hydro|  2802948.2|       {}|
+---------+---------------+-----------+---------+
only showing top 20 rows
+---------------+---------+--------------------+
|productionGroup|priceArea|    total_production|
+---------------+---------+--------------------+
|          hydro|      NO2|   5.240261793104E10|
|          hydro|      NO5|3.033686754729001E10|
|          hydro|      NO4|    2.36643770081E10|
|          hydro|      NO3|2.025394513040002E10|
|          hydro|      NO1|1.835678379994999…|
|           wind|      NO3| 5.149394634765007E9|
|           wind|      NO2| 3.221530055060005E9|
|           wind|      NO4| 1.815169179249999E9|
|        thermal|      NO5|        6.91427003E8|
|           wind|      NO1| 5.473603280449867E8|
|        thermal|      NO4| 2.483754983750002E8|
|        thermal|      NO1|2.3611798497100592E8|
```

```
|        thermal|      NO2| 1.563904761600004E8|
|          other|      NO4|1.5971439443999972E7|
|          solar|      NO1|1.4381936168000001E7|
|          solar|      NO2|1.3993474230000027E7|
|          solar|      NO3|   3966473.815000003|
|          solar|      NO5|  2223285.9049999537|
|          other|      NO2|   492903.0499999995|
|          solar|      NO4|   167653.6119999999|
+---------------+---------+--------------------+
only showing top 20 rows

Production Statistics by Group:
+---------------+---------+--------------------+
|productionGroup|priceArea|    total_production|
+---------------+---------+--------------------+
|          hydro|      NO2|   5.240261793104E10|
|          hydro|      NO5|3.033686754729001E10|
|          hydro|      NO4|    2.36643770081E10|
|          hydro|      NO3|2.025394513040002E10|
|          hydro|      NO1|1.835678379994999…|
|           wind|      NO3| 5.149394634765007E9|
|           wind|      NO2| 3.221530055060005E9|
|           wind|      NO4| 1.815169179249999E9|
|        thermal|      NO5|        6.91427003E8|
|           wind|      NO1| 5.473603280449867E8|
|        thermal|      NO4| 2.483754983750002E8|
|        thermal|      NO1|2.3611798497100592E8|
|        thermal|      NO2| 1.563904761600004E8|
|          other|      NO4|1.5971439443999972E7|
|          solar|      NO1|1.4381936168000001E7|
|          solar|      NO2|1.3993474230000027E7|
|          solar|      NO3|   3966473.815000003|
|          solar|      NO5|  2223285.9049999537|
|          other|      NO2|   492903.0499999995|
|          solar|      NO4|   167653.6119999999|
+---------------+---------+--------------------+
only showing top 20 rows

Production Statistics by Group:
+---------------+--------------------+------------------+---------+------------+
|productionGroup|           total_kwh|           avg_kwh|  max_kwh|record_count|
+---------------+--------------------+------------------+---------+------------+
|          hydro|  1.4501459141678E11| 3310835.420474429|9715193.0|       43800|
|           wind|  1.0733454198994E10| 267313.8793861978|1776178.1|       40153|
|        thermal|1.3323109825060067E9|30418.058961324354| 127152.0|       43800|
|          solar| 3.473282372999997E7| 792.9868431506842| 19737.61|       43800|
|          other| 1.666337722899995E7|380.44240248858335| 8658.902|       43800|
+---------------+--------------------+------------------+---------+------------+
```

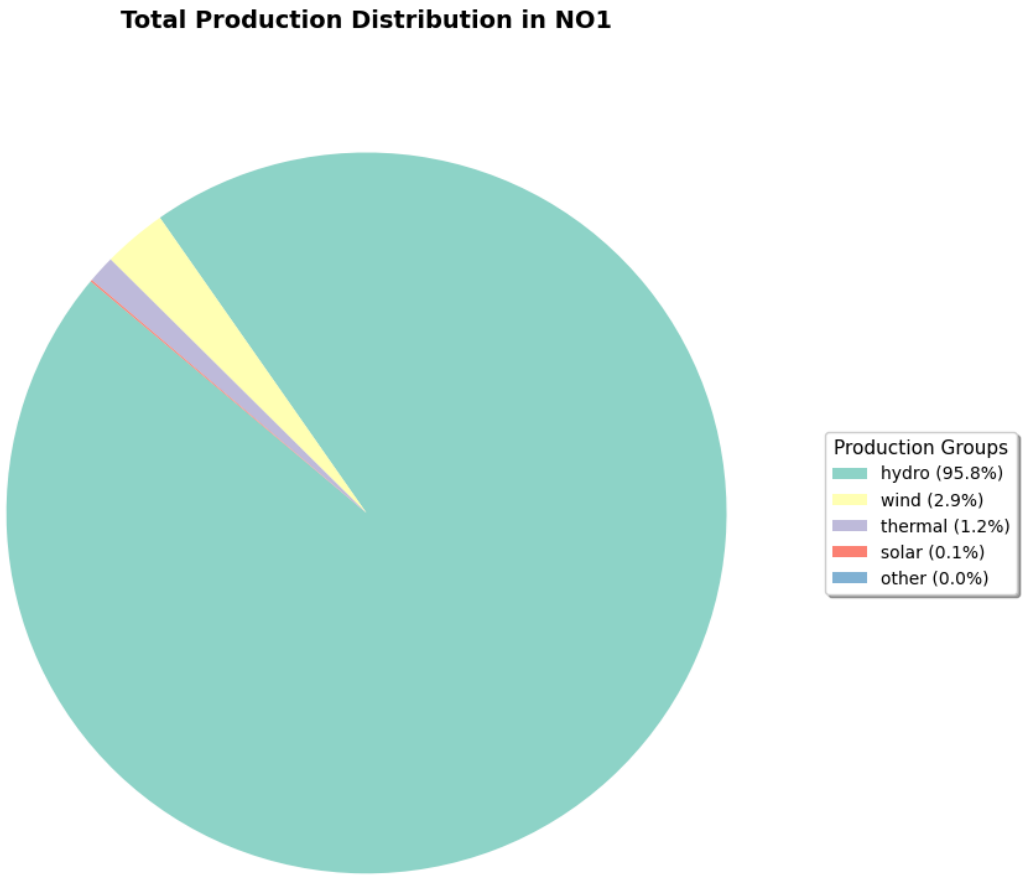
```
+---------------+--------------------+------------------+---------+------------+
|productionGroup|           total_kwh|           avg_kwh|  max_kwh|record_count|
+---------------+--------------------+------------------+---------+------------+
|          hydro|  1.4501459141678E11| 3310835.420474429|9715193.0|       43800|
|           wind|  1.0733454198994E10| 267313.8793861978|1776178.1|       40153|
|        thermal|1.3323109825060067E9|30418.058961324354| 127152.0|       43800|
|          solar| 3.473282372999997E7| 792.9868431506842| 19737.61|       43800|
|          other| 1.666337722899995E7|380.44240248858335| 8658.902|       43800|
+---------------+--------------------+------------------+---------+------------+


  Spark aggregations completed
Summary records: 25
Stats records: 5

Sample summary data:
  productionGroup priceArea  total_production
0           hydro       NO2      5.240262e+10
1           hydro       NO5      3.033687e+10
2           hydro       NO4      2.366438e+10
3           hydro       NO3      2.025395e+10
4           hydro       NO1      1.835678e+10
5            wind       NO3      5.149395e+09
6            wind       NO2      3.221530e+09
7            wind       NO4      1.815169e+09
8         thermal       NO5      6.914270e+08
9            wind       NO1      5.473603e+08

  Spark aggregations completed
Summary records: 25
Stats records: 5

Sample summary data:
  productionGroup priceArea  total_production
0           hydro       NO2      5.240262e+10
1           hydro       NO5      3.033687e+10
2           hydro       NO4      2.366438e+10
3           hydro       NO3      2.025395e+10
4           hydro       NO1      1.835678e+10
5            wind       NO3      5.149395e+09
6            wind       NO2      3.221530e+09
7            wind       NO4      1.815169e+09
8         thermal       NO5      6.914270e+08
9            wind       NO1      5.473603e+08
```

1.5 03_Create plots

1.5.1 Pie Chart

[38]:
```
import matplotlib.pyplot as plt

# Filter data for a chosen price area
selected_area = 'NO1'

# USE SPARK RESULT: Filter the Spark-generated production_summary_pd
# This demonstrates using Spark aggregation results instead of re-calculating␣
 ↪with pandas
filtered_data = production_summary_pd[production_summary_pd['priceArea'] ==␣
 ↪selected_area]
filtered_data = filtered_data.sort_values('total_production', ascending=False)

# Rename columns for consistency
production_summary_chart = filtered_data.copy()
production_summary_chart.columns = ['productionGroup', 'priceArea',␣
 ↪'total_production']

fig1, ax1 = plt.subplots(figsize=(10, 8))
colors = plt.cm.Set3(range(len(production_summary_chart)))

wedges, texts, autotexts = ax1.pie(
    production_summary_chart['total_production'],
    labels=None,
    autopct='',
    startangle=140,
    colors=colors
)

# Calculate percentages
total = production_summary_chart['total_production'].sum()
percentages = (production_summary_chart['total_production'] / total * 100).
 ↪values

# Create legend labels with percentages
legend_labels = [
    f"{name} ({pct:.1f}%)"
    for name, pct in zip(production_summary_chart['productionGroup'],␣
 ↪percentages)
]

# Add legend with fixed position and black text
legend = ax1.legend(
    wedges,
    legend_labels,
```

```
    title="Production Groups",
    loc="center left",
    bbox_to_anchor=(1, 0, 0.5, 1),
    fontsize=10,
    title_fontsize=11,
    frameon=True,
    fancybox=True,
    shadow=True
)

# Set legend text color to black
for text in legend.get_texts():
    text.set_color('black')
legend.get_title().set_color('black')

ax1.set_title(f"Total Production Distribution in {selected_area}", fontsize=14,␣
 ↪fontweight='bold', pad=20)
plt.tight_layout()
plt.show(fig1)
plt.close()

print(f"  Pie chart using Spark-aggregated data from production_summary_pd")
```

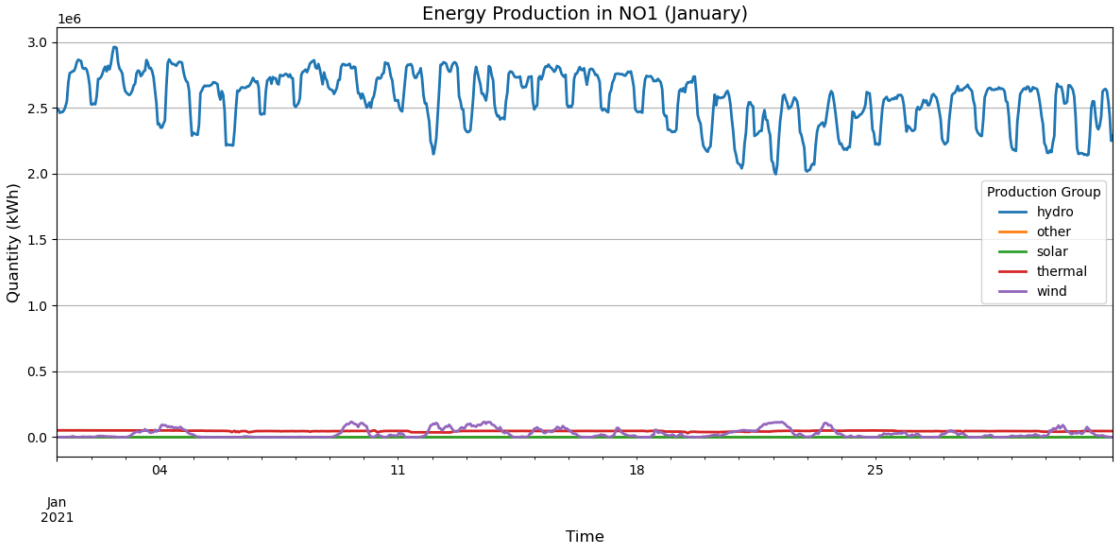


Pie chart using Spark-aggregated data from production_summary_pd

1.5.2 Line plot

[39]:
```
# Ensure the 'startTime' column is in datetime format and convert to UTC
df['startTime'] = pd.to_datetime(df['startTime'], utc=True)

# Filter data for the chosen price area and the first month of the year
filtered_df = df[(df['priceArea'] == selected_area) & (df['startTime'].dt.month␣
 ↪== 1)]

# Pivot the data to have production groups as columns
pivot_df = filtered_df.pivot(index='startTime', columns='productionGroup',␣
 ↪values='quantityKwh')

# Plot the data
plt.figure(figsize=(12, 6))
```

```
pivot_df.plot(ax=plt.gca(), linewidth=2)
plt.title(f"Energy Production in {selected_area} (January)", fontsize=14)
plt.xlabel("Time", fontsize=12)
plt.ylabel("Quantity (kWh)", fontsize=12)
plt.legend(title="Production Group", fontsize=10)
plt.grid(True)
plt.tight_layout()
plt.show()
```



1.6 04_Inserting Spark-extraced data into MongoDB

1.6.1 Checking if DB is reachable

[40]:
```
import streamlit as st
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi

db_user = st.secrets["database"]["db_user"]
secret = st.secrets["database"]["secret"]

uri = f"mongodb+srv://{db_user}:{secret}@cluster0.xxdbouc.mongodb.net/?
 ↪retryWrites=true&w=majority&appName=Cluster0"

# 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)
```

```
Pinged your deployment. You successfully connected to MongoDB!
```

1.6.2 Connecting

[41]:
```
# Selecting a database and a collection.
database = client['ca2_database']
collection = database['data']
```

1.6.3 Inserting

[42]:
```
# First, clear the existing collection to avoid duplicates
collection.delete_many({})
print("Cleared existing data from collection")

# Make a copy of the original pandas DataFrame
pandas_df = df.copy()

# Simplify datetime conversion using lambda
datetime_columns = ['startTime', 'endTime', 'lastUpdatedTime']
pandas_df[datetime_columns] = pandas_df[datetime_columns].applymap(
    lambda x: pd.to_datetime(x, errors='coerce').isoformat() if pd.notna(x)␣
 ↪else None
)

# Convert the Pandas DataFrame to a list of dictionaries
data_to_insert = pandas_df.to_dict(orient='records')

print(f"\nPreparing to insert {len(data_to_insert)} documents...")
print(f"Sample document: {data_to_insert[0]}")

# Insert the data into the MongoDB collection
collection.insert_many(data_to_insert)

print(f"\n  Successfully inserted {len(data_to_insert)} documents into the␣
 ↪MongoDB collection.")
print(f"Database: {database.name}")
print(f"Collection: {collection.name}")
```

```
Cleared existing data from collection
```

```
/var/folders/cb/h4grq88s6sjgm0cnxyzjm7xw0000gn/T/ipykernel_91557/2043375402.py:1
0: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map
instead.
  pandas_df[datetime_columns] = pandas_df[datetime_columns].applymap(
```

```
Preparing to insert 215353 documents…
Sample document: {'endTime': '2021-01-01T01:00:00+01:00', 'lastUpdatedTime':
'2024-12-20T10:35:40+01:00', 'priceArea': 'NO1', 'productionGroup': 'hydro',
'quantityKwh': 2507716.8, 'startTime': '2020-12-31T23:00:00+00:00'}

  Successfully inserted 215353 documents into the MongoDB collection.
Database: ca2_database
Collection: data

  Successfully inserted 215353 documents into the MongoDB collection.
Database: ca2_database
Collection: data
```

1.7 05_Word log

Started off fetching the data from the Elhub API. Had some troubles getting the correct timeperiod, but after checking some discussions in the class padlet i found a url that worked. Then for the next task "Storing it to a Dataframe" i found it simplest to just make it into a JSON file first, then read it to a dataframe. Since the API only allowed for 1 month of data at the time i had to combine 12 fetch calls into a single JSON file, then read that into a dataframe.

Setup Cassandra and Spark according to the documentation in the Course Book. Had some issues with jdk versions, but had it sorted after a while when Github Copilot picked up on the fact that my path was not complete.

Setting up the db connection i added the username and secret to a private file. Made sure to include it in the .gitignore.

For plots i had some issues. for the Pie Chart it was kinda hard getting a good understanding of the contribution of other areas than hydro since it made up the majority of the energy production. I therefore moved labels and values to a legend on the side for more readability. With the interactive chart solution in the streamlit app i think its easier to play around and get a grasp of the impact of other areas. Same goes for the line plots when hydro is involved, but then selecting other groups and looking at them isolated is much more valuable than one single static plot.

All in all i found this assignment to be fairly straight forward. My biggest issues was related to jdk / spark versions, where AI was super helpfull in indicating the problems, helping me understand error messages and how to resolve it.