Project work, part 2

Sara Hørte

Links

GitHub repo link: https://github.com/sarahorte/ind320project.git

Streamlit app link: https://ind320project.streamlit.app/

Log

During the lectures earlier in the semester, I made sure that my Cassandra connection worked and that I could successfully manipulate data from my MongoDB account using Python.

I spent quite a lot of time exploring the Elhub API to figure out the correct way to retrieve hourly production data. With help from ChatGPT and GitHub Copilot in VS Code, I eventually managed to get the correct data. I then inserted the data into Cassandra using Spark and extracted the relevant columns.

Initially, this part worked quite smoothly, but when I reopened the project later, it stopped working. I received numerous error messages and struggled a lot with the connection between Cassandra and Python. Eventually, I had to start over — fortunately, it worked again after some trial and error. During this process, I used AI extensively to try to identify the problems, although it didn't always manage to solve the connection issues.

I created the visualizations using Plotly and inserted the data into MongoDB.

Next, I continued with the Streamlit app. I established a connection with my MongoDB database — I wasn't entirely sure how to handle the password and credentials securely, but I believe I did it correctly. Once the connection was working, I built page four as described in the assignment. This part went quite smoothly once the MongoDB connection was stable.

Overall, I found this assignment quite challenging. The most difficult part was getting all the connections configured correctly. It was hard to identify the root causes of the problems, and to be honest, I'm still not completely sure why it sometimes worked and sometimes didn't.

AI usage

AI was used extensively throughout this project, mainly through ChatGPT and GitHub Copilot in VS Code. I copied parts of the assignment text into ChatGPT and received code suggestions in return. Then I adjusted the code myself, using examples from lectures and additional AI assistance, until I got it to work. For the Streamlit app, I mainly added comments and used suggestions from GitHub Copilot in VS Code.

I also pasted error messages into ChatGPT to understand what was going wrong and tested different proposed solutions until I found one that worked. I used AI to generate the plots and later fine-tuned them to achieve the desired appearance.

AI has been extremely useful for completing this project, especially for debugging and understanding unfamiliar technologies. However, it was also quite frustrating at times — particularly when it failed to resolve the connection issues.

I also used ChatGPT to refine and improve the wording of the Log, AI Usage, and Fetching Data from the Elhub API sections.

Fetching data from Elhub API

When retrieving data from the Elhub API, it is important to account for the transitions between winter time and summer time. In 2023, the switch to summer time occurred on March 26, when the clock moved from 2023-03-26T01:00:00.000+01:00 to 2023-03-26T03:00:00.000+02:00. The return to winter time took place on October 29, when the time shifted from 2023-10-29T02:00:00.000+02:00 back to 2023-10-29T02:00:00.000+01:00.

Despite these time changes, there are still 24 data points per day, as the data is adjusted relative to the base time.

Additionally, the first timestamp in the dataset appears as 2020-12-31 23:00:00+00:00, because all timestamps are converted to UTC. This corresponds to 2021-01-01 00:00:00+01:00 in local Norwegian time. The same time adjustment pattern applies for 2021 as well.

In [1]:
```
# ==============================
# 1️⃣ FETCH DATA FROM ELHUB API
# ==============================
import requests
import pandas as pd
from datetime import datetime, timedelta
import time

# --- API SETTINGS ---
BASE_URL = "https://api.elhub.no/energy-data/v0/price-areas"
DATASET = "PRODUCTION_PER_GROUP_MBA_HOUR"

# --- FUNCTION TO FORMAT DATES ---
def format_date(dt_obj):
```

```
    """Formats datetime with timezone offset for Elhub (%2B02:00)."""
    return dt_obj.strftime("%Y-%m-%dT%H:%M:%S%%2B02:00")  # +02:00 is used f

all_records = []

# --- FETCH MONTHLY DATA FOR 2021 ---
for month in range(1, 13):
    start = datetime(2021, month, 1)
    next_month = (start + timedelta(days=32)).replace(day=1)
    end = next_month - timedelta(seconds=1)

    start_str = format_date(start)
    end_str = format_date(end)

    url = f"{BASE_URL}?dataset={DATASET}&startDate={start_str}&endDate={end_
    print(f"=== Fetching {start.date()} → {end.date()} ===")

    response = requests.get(url)
    if response.status_code != 200:
        print(f"❌ Error {response.status_code}")
        continue

    data = response.json()
    month_records = []

    for entry in data.get("data", []):
        attrs = entry.get("attributes", {})
        recs = attrs.get("productionPerGroupMbaHour", [])
        # Filter out placeholders
        recs = [r for r in recs if r.get("productionGroup") != "*"]
        month_records.extend(recs)

    all_records.extend(month_records)
    print(f"✅ {len(month_records)} records added")

    # Be kind to API
    time.sleep(1)

print(f"\nTotal records collected: {len(all_records)}")

# --- CONVERT TO DATAFRAME ---
df = pd.DataFrame(all_records)
df['startTime'] = pd.to_datetime(df['startTime'], utc=True)
df['endTime'] = pd.to_datetime(df['endTime'], utc=True)
df['quantityKwh'] = pd.to_numeric(df['quantityKwh'], errors='coerce')
df = df[['priceArea', 'productionGroup', 'startTime', 'quantityKwh']]
df.sort_values('startTime', inplace=True)
df.set_index('startTime', inplace=True)

print(df.info())
print(df.head(50))
print(f"DataFrame shape: {df.shape}")
```

```
=== Fetching 2021-01-01 → 2021-01-31 ===
✅ 17856 records added
=== Fetching 2021-02-01 → 2021-02-28 ===
✅ 16128 records added
=== Fetching 2021-03-01 → 2021-03-31 ===
✅ 17832 records added
=== Fetching 2021-04-01 → 2021-04-30 ===
✅ 17280 records added
=== Fetching 2021-05-01 → 2021-05-31 ===
✅ 17856 records added
=== Fetching 2021-06-01 → 2021-06-30 ===
✅ 17976 records added
=== Fetching 2021-07-01 → 2021-07-31 ===
✅ 18600 records added
=== Fetching 2021-08-01 → 2021-08-31 ===
✅ 18600 records added
=== Fetching 2021-09-01 → 2021-09-30 ===
✅ 18000 records added
=== Fetching 2021-10-01 → 2021-10-31 ===
✅ 18625 records added
=== Fetching 2021-11-01 → 2021-11-30 ===
✅ 18000 records added
=== Fetching 2021-12-01 → 2021-12-31 ===
✅ 18600 records added

Total records collected: 215353
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 215353 entries, 2020-12-31 23:00:00+00:00 to 2021-12-31 22:0
0:00+00:00
Data columns (total 3 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   priceArea        215353 non-null  object
 1   productionGroup  215353 non-null  object
 2   quantityKwh      215353 non-null  float64
dtypes: float64(1), object(2)
memory usage: 6.6+ MB
None
                          priceArea productionGroup  quantityKwh
startTime
2020-12-31 23:00:00+00:00       NO1           hydro  2507716.800
2020-12-31 23:00:00+00:00       NO2           other        4.346
2020-12-31 23:00:00+00:00       NO5           solar        3.720
2020-12-31 23:00:00+00:00       NO2            wind      706.206
2020-12-31 23:00:00+00:00       NO3           hydro  2836774.000
2020-12-31 23:00:00+00:00       NO4            wind   381065.000
2020-12-31 23:00:00+00:00       NO2           hydro  7245923.500
2020-12-31 23:00:00+00:00       NO3           other        0.000
2020-12-31 23:00:00+00:00       NO5         thermal    77742.000
2020-12-31 23:00:00+00:00       NO4         thermal    21349.000
2020-12-31 23:00:00+00:00       NO2         thermal    24171.203
2020-12-31 23:00:00+00:00       NO3           solar       19.722
2020-12-31 23:00:00+00:00       NO1            wind      937.072
2020-12-31 23:00:00+00:00       NO5           other        0.000
2020-12-31 23:00:00+00:00       NO3         thermal        0.000
2020-12-31 23:00:00+00:00       NO3            wind   259312.200
```

```
2020-12-31 23:00:00+00:00       NO4           solar        0.000
2020-12-31 23:00:00+00:00       NO5           hydro  4068096.500
2020-12-31 23:00:00+00:00       NO1         thermal    51369.035
2020-12-31 23:00:00+00:00       NO4           hydro  3740830.000
2020-12-31 23:00:00+00:00       NO4           other        0.161
2020-12-31 23:00:00+00:00       NO1           other        0.000
2020-12-31 23:00:00+00:00       NO2           solar      876.556
2020-12-31 23:00:00+00:00       NO1           solar        6.106
2021-01-01 00:00:00+00:00       NO1         thermal    51673.934
2021-01-01 00:00:00+00:00       NO3            wind   225762.900
2021-01-01 00:00:00+00:00       NO3           solar       25.433
2021-01-01 00:00:00+00:00       NO5           other        0.000
2021-01-01 00:00:00+00:00       NO3         thermal        0.000
2021-01-01 00:00:00+00:00       NO3           other        0.000
2021-01-01 00:00:00+00:00       NO4            wind   369910.000
2021-01-01 00:00:00+00:00       NO1           solar        4.030
2021-01-01 00:00:00+00:00       NO5           solar        3.600
2021-01-01 00:00:00+00:00       NO2            wind     3431.889
2021-01-01 00:00:00+00:00       NO2         thermal    24195.646
2021-01-01 00:00:00+00:00       NO1           other        0.000
2021-01-01 00:00:00+00:00       NO2           other        3.642
2021-01-01 00:00:00+00:00       NO5         thermal    77575.000
2021-01-01 00:00:00+00:00       NO4           hydro  3746663.500
2021-01-01 00:00:00+00:00       NO4           solar        0.000
2021-01-01 00:00:00+00:00       NO1            wind      649.068
2021-01-01 00:00:00+00:00       NO5           hydro  4104306.000
2021-01-01 00:00:00+00:00       NO4           other        0.161
2021-01-01 00:00:00+00:00       NO3           hydro  2836189.800
2021-01-01 00:00:00+00:00       NO4         thermal    22554.000
2021-01-01 00:00:00+00:00       NO2           solar      876.398
2021-01-01 00:00:00+00:00       NO1           hydro  2494728.000
2021-01-01 00:00:00+00:00       NO2           hydro  6750958.000
2021-01-01 01:00:00+00:00       NO2           other        3.562
2021-01-01 01:00:00+00:00       NO2         thermal    23558.420
DataFrame shape: (215353, 3)
```

In [2]:

```python
from cassandra.cluster import Cluster

# Connect to local Cassandra
cluster = Cluster(['127.0.0.1'])
session = cluster.connect()

# --- Create keyspace ---
session.execute("""
    CREATE KEYSPACE IF NOT EXISTS energy
    WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
""")

# --- Create table ---
session.execute("""
    CREATE TABLE IF NOT EXISTS energy.production_per_group (
        priceArea text,
        startTime timestamp,
        productionGroup text,
        quantityKwh double,
        PRIMARY KEY ((priceArea), startTime, productionGroup)
```

```
    );
""")

print("✅ Keyspace and table ready")
```

```
✅ Keyspace and table ready
```

In [3]:

```
# ========================================
# 2️⃣ SAVE DATA INTO CASSANDRA USING SPARK
# ========================================
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, Timestamp

# --- CREATE SPARK SESSION ---
spark = SparkSession.builder \
    .appName("ElhubProduction2021") \
    .config("spark.cassandra.connection.host", "127.0.0.1") \
    .config("spark.jars.packages", "com.datastax.spark:spark-cassandra-conne
    .getOrCreate()


print(spark.version)  # Should print 3.5.1


# --- DEFINE SCHEMA ---
schema = StructType([
    StructField("priceArea", StringType(), True),
    StructField("productionGroup", StringType(), True),
    StructField("startTime", TimestampType(), True),
    StructField("quantityKwh", DoubleType(), True)
])

# Select only the needed columns and reset index
spark_df = spark.createDataFrame(df.reset_index()[['startTime', 'priceArea',

# Rename columns to lowercase to match Cassandra table
spark_df_cassandra = spark_df.selectExpr(
    "priceArea as pricearea",
    "startTime as starttime",
    "productionGroup as productiongroup",
    "quantityKwh as quantitykwh"
)

spark_df_cassandra.printSchema()

# Write to Cassandra
spark_df_cassandra.write \
    .format("org.apache.spark.sql.cassandra") \
    .options(keyspace="energy", table="production_per_group") \
    .mode("append") \
    .save()

print("✅ Data written to Cassandra successfully")
```

```
:: loading settings :: url = jar:file:/opt/anaconda3/envs/D2D_env/lib/python
3.11/site-packages/pyspark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/
ivysettings.xml
```

```
Ivy Default Cache set to: /Users/sarahorte/.ivy2/cache
The jars for the packages stored in: /Users/sarahorte/.ivy2/jars
com.datastax.spark#spark-cassandra-connector_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-3ee0a1ce-1
707-488a-929b-d088d9d5ff6c;1.0
        confs: [default]
        found com.datastax.spark#spark-cassandra-connector_2.12;3.5.0 in cen
tral
        found com.datastax.spark#spark-cassandra-connector-driver_2.12;3.5.0
in central
        found org.scala-lang.modules#scala-collection-compat_2.12;2.11.0 in
central
        found com.datastax.oss#java-driver-core-shaded;4.13.0 in central
        found com.datastax.oss#native-protocol;1.5.0 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 com.github.stephenc.jcip#jcip-annotations;1.0-1 in central
        found com.github.spotbugs#spotbugs-annotations;3.1.12 in central
        found com.google.code.findbugs#jsr305;3.0.2 in central
        found com.datastax.oss#java-driver-mapper-runtime;4.13.0 in central
        found com.datastax.oss#java-driver-query-builder;4.13.0 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;2.12.11 in central
:: resolution report :: resolve 219ms :: artifacts dl 7ms
        :: modules in use:
        com.datastax.oss#java-driver-core-shaded;4.13.0 from central in [def
ault]
        com.datastax.oss#java-driver-mapper-runtime;4.13.0 from central in
[default]
        com.datastax.oss#java-driver-query-builder;4.13.0 from central in [d
efault]
        com.datastax.oss#java-driver-shaded-guava;25.1-jre-graal-sub-1 from
central in [default]
        com.datastax.oss#native-protocol;1.5.0 from central in [default]
        com.datastax.spark#spark-cassandra-connector-driver_2.12;3.5.0 from
central in [default]
        com.datastax.spark#spark-cassandra-connector_2.12;3.5.0 from central
in [default]
        com.github.spotbugs#spotbugs-annotations;3.1.12 from central in [def
ault]
        com.github.stephenc.jcip#jcip-annotations;1.0-1 from central in [def
ault]
        com.google.code.findbugs#jsr305;3.0.2 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.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;2.12.11 from central in [default]
```

```
        org.scala-lang.modules#scala-collection-compat_2.12;2.11.0 from cent
ral in [default]
        org.slf4j#slf4j-api;1.7.26 from central in [default]
        ---------------------------------------------------------------------
-
        |                  |            modules            ||   artifacts
|
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded
|
        ---------------------------------------------------------------------
-
        |      default     |   19  |   0   |   0   |   0   ||   19  |   0
|
        ---------------------------------------------------------------------
-
:: retrieving :: org.apache.spark#spark-submit-parent-3ee0a1ce-1707-488a-929
b-d088d9d5ff6c
        confs: [default]
        0 artifacts copied, 19 already retrieved (0kB/4ms)
25/10/22 10:07:00 WARN NativeCodeLoader: Unable to load native-hadoop librar
y 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 setLog
Level(newLevel).
3.5.1
root
 |-- pricearea: string (nullable = true)
 |-- starttime: timestamp (nullable = true)
 |-- productiongroup: string (nullable = true)
 |-- quantitykwh: double (nullable = true)

[Stage 0:>                                                          (0 + 10) /
10]
✅ Data written to Cassandra successfully
```

In [4]:

```python
# ========================================
# 3️⃣ READ DATA BACK FROM CASSANDRA
# ========================================
cass_df = spark.read \
    .format("org.apache.spark.sql.cassandra") \
    .options(keyspace="energy", table="production_per_group") \
    .load()

cass_df.show(5)

# Check how many rows were written to Cassandra
row_count = cass_df.count()
print(f"Total rows written to Cassandra: {row_count}")
```

```
+---------+-------------------+--------------+-----------+
|pricearea|          starttime|productiongroup|quantitykwh|
+---------+-------------------+--------------+-----------+
|      NO4|2021-01-01 00:00:00|         hydro|  3740830.0|
|      NO4|2021-01-01 00:00:00|         other|      0.161|
|      NO4|2021-01-01 00:00:00|         solar|        0.0|
|      NO4|2021-01-01 00:00:00|       thermal|    21349.0|
|      NO4|2021-01-01 00:00:00|          wind|   381065.0|
+---------+-------------------+--------------+-----------+
only showing top 5 rows

Total rows written to Cassandra: 215328
```

In [5]:
```
# Check for duplicates based on Cassandra primary key
duplicates = df.reset_index().duplicated(subset=['priceArea','startTime','pr
print(f"Duplicate rows: {duplicates.sum()}")
```

```
Duplicate rows: 0
```

In [6]:
```
# Reset index so startTime becomes a column again
df_reset = df.reset_index()

# Check for nulls in primary key columns
print(df_reset[['startTime', 'priceArea', 'productionGroup']].isnull().sum()
```

```
startTime          0
priceArea          0
productionGroup    0
dtype: int64
```

Noticing "DataFrame shape: (215353, 3)" in the beginning, while "Total rows written to Cassandra: 215328". There are 25 rows missing, but I could not figure out why.

Plots

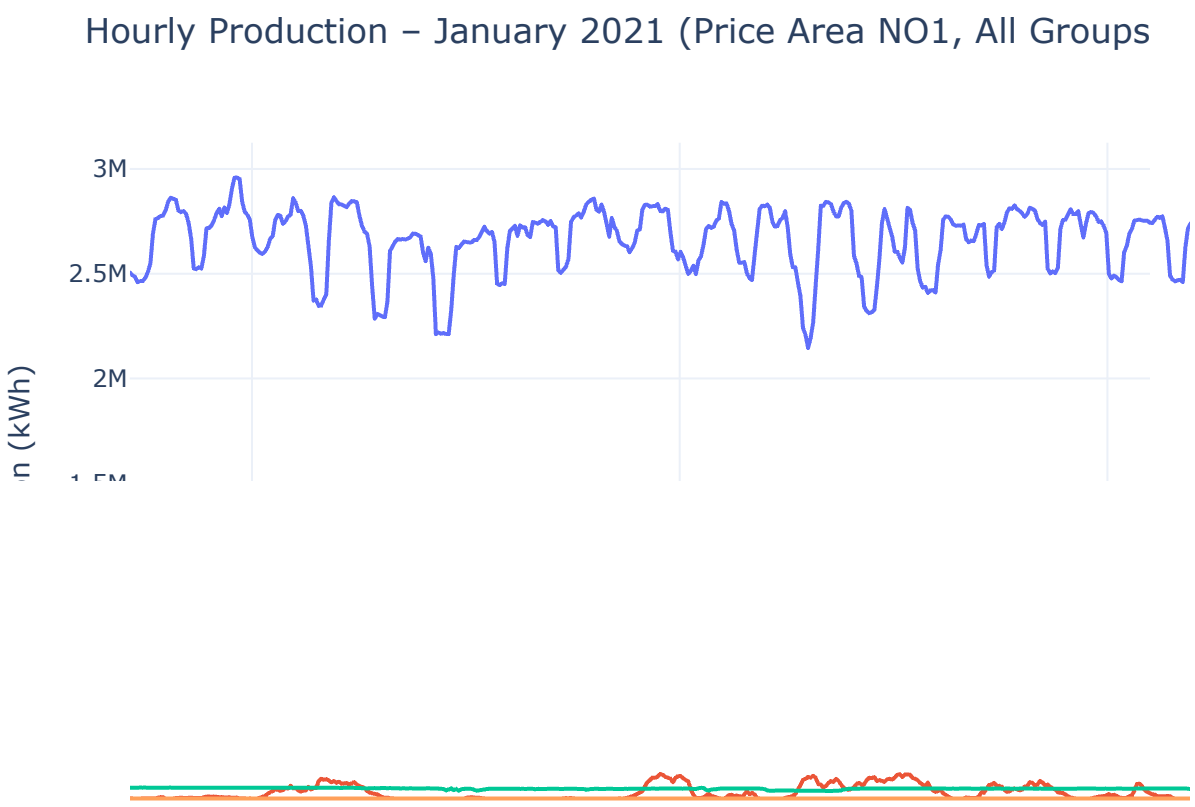
In [7]:
```
# A pie chart for the total production of the year from a chosen price area
import plotly.express as px

# --- Choose your price area ---
chosen_pricearea = "NO1"

# --- Filter Spark DataFrame for the chosen price area ---
df_selected = spark_df_cassandra.filter(spark_df_cassandra.pricearea == chos

# --- Aggregate total production per production group for the entire year --
df_pie = (
    df_selected.groupBy("productiongroup")
    .sum("quantitykwh")
    .toPandas()
)

# --- Rename column for readability ---
df_pie.rename(columns={"sum(quantitykwh)": "total_quantitykwh"}, inplace=Tru

# --- Create interactive pie chart ---
```

```
fig = px.pie(
    df_pie,
    names="productiongroup",
    values="total_quantitykwh",
    title=f"Total Production by Group – Price Area {chosen_pricearea} (2021)
)

fig.show()
```

In [8]:

```
# A line plot for the first month of the year for a chosen price area

import plotly.io as pio

# --- Set renderer to notebook for inline display ---
pio.renderers.default = "notebook"

# === 1️⃣ Choose price area ===
chosen_pricearea = "NO1"

# === 2️⃣ Filter Spark DataFrame for chosen area and January 2021 ===
df_january = (
    spark_df_cassandra
    .filter(spark_df_cassandra.pricearea == chosen_pricearea)
    .filter((spark_df_cassandra.starttime >= '2021-01-01') & (spark_df_cassa
)

# === 3️⃣ Convert to Pandas for Plotly ===
df_january_pd = df_january.toPandas()

# === 4️⃣ Plot 1 – All production groups ===
fig_all = px.line(
    df_january_pd,
    x="starttime",
    y="quantitykwh",
    color="productiongroup",
    title=f"Hourly Production – January 2021 (Price Area {chosen_pricearea},
    labels={"starttime": "Time", "quantitykwh": "Production (kWh)", "product
)
fig_all.update_layout(template="plotly_white")
fig_all.show()  # Inline in notebook

# === 5️⃣ Extract color mapping used by Plotly in the first figure ===
color_map = {trace.name: trace.line.color for trace in fig_all.data}

# === 6️⃣ Filter out hydro ===
df_no_hydro = df_january_pd[df_january_pd["productiongroup"].str.lower() !=

# === 7️⃣ Create consistent color map for remaining groups ===
consistent_colors = {
    group: color_map[group]
    for group in df_no_hydro["productiongroup"].unique()
    if group in color_map
}

# === 8️⃣ Plot 2 – Without hydro, using same colors, for consistency and to e
```

```
fig_no_hydro = px.line(
    df_no_hydro,
    x="starttime",
    y="quantitykwh",
    color="productiongroup",
    title=f"Hourly Production – January 2021 (Price Area {chosen_pricearea},
    labels={"starttime": "Time", "quantitykwh": "Production (kWh)", "product
    color_discrete_map=consistent_colors
)
fig_no_hydro.update_layout(template="plotly_white")
fig_no_hydro.show()  # Inline in notebook
```



Insert the Spark-extracted data into MongoDB.

In [9]:

```
from pymongo import MongoClient

# === MongoDB Atlas connection ===
uri = "mongodb+srv://{}:{}@cluster0.qwrlccf.mongodb.net/?retryWrites=true&w=

# --- Read credentials from file ---
USR, PWD = open('/Users/sarahorte/Documents/IND320/Personlig/No_sync/MongoDB

# --- Connect to MongoDB ---
client = MongoClient(uri.format(USR, PWD))

# --- Create database and collection ---
database = client['elhub']
collection = database['production_data']

# --- Convert Spark DataFrame to Pandas for MongoDB ---
df_mongo = spark_df_cassandra.toPandas()  # Use Cassandra-safe DataFrame

# --- Convert DataFrame to list of dictionaries ---
records = df_mongo.to_dict(orient="records")
```

```python
# --- Clear existing data ---
collection.delete_many({})

# --- Insert into MongoDB ---
collection.insert_many(records)

# --- Verify insertion ---
print("✅ Documents inserted:", collection.count_documents({}))
print("Example document:", collection.find_one())
print("Distinct price areas:", collection.distinct("pricearea"))
```

```
✅ Documents inserted: 215353
Example document: {'_id': ObjectId('68f890cd020b89d4489da210'), 'pricearea':
'NO1', 'starttime': datetime.datetime(2021, 1, 1, 0, 0), 'productiongroup':
'hydro', 'quantitykwh': 2507716.8}
Distinct price areas: ['NO1', 'NO2', 'NO3', 'NO4', 'NO5']
```