Lenker til prosjektet

- GitHub-repositorium: https://github.com/pialoschbrandt/streamlit#
- Streamlit-app: https://appgit-2khm3anafqsdgqrdfpx7vz.streamlit.app

LOGG 2

This notebook covers:

1. Retrieving production data from Elhub's API for all months of 2021.
2. Extracting JSON data and converting it into a combined Pandas DataFrame.
3. Preparing the data by selecting relevant columns, renaming them to snake_case, and converting timestamps to datetime with UTC.
4. Uploading the final dataset to MongoDB.
5. Verifying that the data was successfully stored by counting documents and displaying sample rows.
6. How this was done To achieve the goal of fetching data from an API → sending it to Cassandra → storing it in MongoDB → and finally displaying it in a Streamlit dashboard, several steps were required:
    A. Installed and configured Docker.
    B. Created accounts for both Cassandra and MongoDB.
    C. Connected Docker to the Cassandra database.
    D. Installed Apache Spark and set up the necessary Java connection for Spark integration.
    E. Learned how to fetch and process data from the Elhub API. This part of the project took the most time, as it involved many new concepts, tools, and dependencies that had to work together properly. Once the Extract and Load steps were functioning, it was relatively quick to create basic visualizations and integrate them into a Streamlit app.
7. Challenge The main challenge encountered was establishing communication between Spark and Cassandra. Despite several troubleshooting attempts (in collaboration with Kristian), the connection could not be established, and consequently, data could not be written to Cassandra. As a result, the data was successfully loaded only into MongoDB. For a more detailed explanation of this issue, see Section 1.9. It is also important to understand the type of data you are working with, as well as the formats of different structures such as Spark DataFrames, standard DataFrames, and Java objects. This knowledge is essential for determining how the data can best be processed and visualized afterward.
8. How AI was used AI was used extensively throughout the project — both as a conversation partner, a teacher, and a technical guide for setup, debugging, and code explanations. The combination of AI support and the course material in the D2D Book provided a strong foundation for completing the task. AI helped clarify complex setup steps (e.g., Docker, Spark, and Cassandra connections) and was also used to translate, format, and document the code clearly and consistently.

1.1 Import Libraries

In [65]:
```
#Import libraries
import requests
import pandas as pd
from pyspark.sql import SparkSession
from datetime import date, timedelta
import calendar
import os
import sys
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
import plotly.express as px
import calendar
```

1.2 Make Spark-Session

In [66]:
```
# ---- Java ----
os.environ["JAVA_HOME"] = "/opt/homebrew/Cellar/openjdk@17/17.0.16/libexec/openjdk.

# ---- Python (din conda-miljø sti) ----
os.environ["PYSPARK_PYTHON"] = "/Users/pialoschbrandt/anaconda3/envs/streamlit/bin/
os.environ["PYSPARK_DRIVER_PYTHON"] = "/Users/pialoschbrandt/anaconda3/envs/streaml

# ---- Start ny Spark-session ----
spark = (
    SparkSession.builder
    .master("local[*]")
    .appName("ReadElhubFromCassandra")
    # her legger du til connector-jar for Cassandra
    .config("spark.jars", "/Users/pialoschbrandt/spark_jars/spark-cassandra-connect
    # her legger du til host-informasjon (Cassandra kjører lokalt i Docker)
    .config("spark.cassandra.connection.host", "localhost")
    .config("spark.cassandra.connection.port", "9042")
    .getOrCreate()
)

print("✅ Spark kjører med versjon:", spark.version)
print("Python som brukes:", sys.executable)
```

```
✅ Spark kjører med versjon: 3.5.3
Python som brukes: /Users/pialoschbrandt/anaconda3/envs/streamlit/bin/python
```

1.3 Get data from API

In [67]:
```
# URL for Elhub's energy data API
base_url = "https://api.elhub.no/energy-data/v0/price-areas"

# Parameters for the API request – defines which dataset to fetch
params = {'dataset': 'PRODUCTION_PER_GROUP_MBA_HOUR'}

# Empty list to collect DataFrames for each month
all_data = []

# Loop through all 12 months of the year 2021
for month in range(1, 13):
    start = date(2021, month, 1)                                    # Start date of
    end = date(2022, 1, 1) if month == 12 else date(2021, month + 1, 1)  # End date

    params['startDate'] = start.isoformat()   # Add start date as an ISO-formatted
    params['endDate'] = end.isoformat()       # Add end date as an ISO-formatted st

    # Send a request to the Elhub API with the chosen parameters
    r = requests.get(base_url, params=params)
```

```
    # Extract the "data" field from the JSON response
    data = r.json()['data']

    # Temporary list to store all rows for this month
    # Each row is manually built from the JSON field 'productionPerGroupMbaHour'
    rows = []

    # Iterate through all price areas (NO1, NO2, NO3, etc.) in the JSON response
    for d in data:
        attr = d['attributes']                                  # Get the "attributes
        for p in attr['productionPerGroupMbaHour']:             # Iterate through al
            # This block UNPACKS each data entry from the JSON and builds a Python
            rows.append({
                'country': attr['country'],                     # Country (e.g., NO)
                'priceArea': p.get('priceArea'),                # Price area (e.g.,
                'productionGroup': p.get('productionGroup'),    # Type of production
                'quantityKwh': p.get('quantityKwh'),            # Amount of produced
                'startTime': p.get('startTime'),                # Start time of the
                'endTime': p.get('endTime'),                    # End time of the pe
                'lastUpdatedTime': p.get('lastUpdatedTime')     # When the data was
            })

    # This line CONVERTS the 'rows' list (which now contains all unpacked JSON rows
    # into a Pandas DataFrame – this is where the data is turned into tabular form
    df = pd.DataFrame(rows)

    # Add the monthly DataFrame to the overall list (for merging later)
    all_data.append(df)

# Combine all monthly DataFrames into one consolidated DataFrame containing all row
df_all = pd.concat(all_data, ignore_index=True)

# Print the first rows to verify that everything looks correct
print(df_all.head())

# Print total number of rows in the DataFrame to confirm that all data has been fet
print("Number of rows:", len(df_all))

# Print the first rows again to preview the data
print(df_all.head())

# Print total number of rows in the DataFrame
print("Number of rows:", len(df_all))
```

```
  country priceArea productionGroup  quantityKwh                  startTime  \
0      NO       NO1           hydro    2507716.8  2021-01-01T00:00:00+01:00
1      NO       NO1           hydro    2494728.0  2021-01-01T01:00:00+01:00
2      NO       NO1           hydro    2486777.5  2021-01-01T02:00:00+01:00
3      NO       NO1           hydro    2461176.0  2021-01-01T03:00:00+01:00
4      NO       NO1           hydro    2466969.2  2021-01-01T04:00:00+01:00

                     endTime            lastUpdatedTime
0  2021-01-01T01:00:00+01:00  2024-12-20T10:35:40+01:00
1  2021-01-01T02:00:00+01:00  2024-12-20T10:35:40+01:00
2  2021-01-01T03:00:00+01:00  2024-12-20T10:35:40+01:00
3  2021-01-01T04:00:00+01:00  2024-12-20T10:35:40+01:00
4  2021-01-01T05:00:00+01:00  2024-12-20T10:35:40+01:00
Number of rows: 215353
  country priceArea productionGroup  quantityKwh                  startTime  \
0      NO       NO1           hydro    2507716.8  2021-01-01T00:00:00+01:00
1      NO       NO1           hydro    2494728.0  2021-01-01T01:00:00+01:00
2      NO       NO1           hydro    2486777.5  2021-01-01T02:00:00+01:00
3      NO       NO1           hydro    2461176.0  2021-01-01T03:00:00+01:00
4      NO       NO1           hydro    2466969.2  2021-01-01T04:00:00+01:00

                     endTime            lastUpdatedTime
0  2021-01-01T01:00:00+01:00  2024-12-20T10:35:40+01:00
1  2021-01-01T02:00:00+01:00  2024-12-20T10:35:40+01:00
2  2021-01-01T03:00:00+01:00  2024-12-20T10:35:40+01:00
3  2021-01-01T04:00:00+01:00  2024-12-20T10:35:40+01:00
4  2021-01-01T05:00:00+01:00  2024-12-20T10:35:40+01:00
Number of rows: 215353
```

1.4 Check data from API

In [68]:
```python
# Print the total number of rows in the DataFrame
print("Number of rows:", len(df_all))

# ---- Get column names ----
print("\nColumn names in df_all:")
print(list(df_all.columns))

# More readable with line breaks:
print("\nColumns:")
for col in df_all.columns:
    print("-", col)
```

```
Number of rows: 215353

Column names in df_all:
['country', 'priceArea', 'productionGroup', 'quantityKwh', 'startTime', 'endTime',
'lastUpdatedTime']

Columns:
- country
- priceArea
- productionGroup
- quantityKwh
- startTime
- endTime
- lastUpdatedTime
```

1.5 Change dataframe before converting it to a Spark-Frame

In [69]:
```python
# Prepare the DataFrame before converting it to a Spark DataFrame
# Keep only the relevant columns
df_ready = df_all[["priceArea", "productionGroup", "startTime", "endTime", "quantit

# Rename columns to snake_case
df_ready.rename(columns={
```

```
    "priceArea": "price_area",
    "productionGroup": "production_group",
    "startTime": "start_time",
    "endTime": "end_time",
    "quantityKwh": "quantity_kwh"
}, inplace=True)

# Remove any rows missing key values
df_ready.dropna(subset=["price_area", "start_time"], inplace=True)

# Convert time columns from string to datetime with UTC
df_ready["start_time"] = pd.to_datetime(df_ready["start_time"], utc=True, errors="c
df_ready["end_time"] = pd.to_datetime(df_ready["end_time"], utc=True, errors="coerc

print(df_ready.head())
print("Columns after rename:", list(df_ready.columns))
```

```
  price_area production_group                start_time  \
0        NO1            hydro 2020-12-31 23:00:00+00:00
1        NO1            hydro 2021-01-01 00:00:00+00:00
2        NO1            hydro 2021-01-01 01:00:00+00:00
3        NO1            hydro 2021-01-01 02:00:00+00:00
4        NO1            hydro 2021-01-01 03:00:00+00:00

                   end_time  quantity_kwh
0 2021-01-01 00:00:00+00:00     2507716.8
1 2021-01-01 01:00:00+00:00     2494728.0
2 2021-01-01 02:00:00+00:00     2486777.5
3 2021-01-01 03:00:00+00:00     2461176.0
4 2021-01-01 04:00:00+00:00     2466969.2
Columns after rename: ['price_area', 'production_group', 'start_time', 'end_time',
'quantity_kwh']
```

1.6 Converting to Spark-Frame

In [70]:
```
# ---- convert Pandas DataFrame to Spark DataFrame ----
from pyspark.sql import SparkSession
from pyspark.sql.functions import to_timestamp

# Create Spark DataFrame from Pandas DataFrame
spark_df = spark.createDataFrame(df_ready)
```

1.7 Inspecting the Spark-Frame

In [71]:
```
# ---- Print schema (shows column names and data types) ----
print("\nSchema for spark_df:")
spark_df.printSchema()

# ---- Show the first 5 rows ----
print("\nContents of spark_df:")
spark_df.show(5, truncate=False)
```

```
Schema for spark_df:
root
 |-- price_area: string (nullable = true)
 |-- production_group: string (nullable = true)
 |-- start_time: timestamp (nullable = true)
 |-- end_time: timestamp (nullable = true)
 |-- quantity_kwh: double (nullable = true)


Contents of spark_df:
```

```
25/10/25 03:06:29 INFO SparkContext: Starting job: showString at NativeMethodAccess
orImpl.java:0
25/10/25 03:06:29 INFO DAGScheduler: Got job 2 (showString at NativeMethodAccessorI
mpl.java:0) with 1 output partitions
25/10/25 03:06:29 INFO DAGScheduler: Final stage: ResultStage 2 (showString at Nati
veMethodAccessorImpl.java:0)
25/10/25 03:06:29 INFO DAGScheduler: Parents of final stage: List()
25/10/25 03:06:29 INFO DAGScheduler: Missing parents: List()
25/10/25 03:06:29 INFO DAGScheduler: Submitting ResultStage 2 (MapPartitionsRDD[20]
at showString at NativeMethodAccessorImpl.java:0), which has no missing parents
25/10/25 03:06:29 INFO MemoryStore: Block broadcast_2 stored as values in memory (e
stimated size 15.0 KiB, free 434.4 MiB)
25/10/25 03:06:29 INFO MemoryStore: Block broadcast_2_piece0 stored as bytes in mem
ory (estimated size 7.6 KiB, free 434.4 MiB)
25/10/25 03:06:29 INFO BlockManagerInfo: Added broadcast_2_piece0 in memory on 10.2
0.11.250:65094 (size: 7.6 KiB, free: 434.4 MiB)
25/10/25 03:06:29 INFO SparkContext: Created broadcast 2 from broadcast at DAGSched
uler.scala:1585
25/10/25 03:06:29 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 2
(MapPartitionsRDD[20] at showString at NativeMethodAccessorImpl.java:0) (first 15 t
asks are for partitions Vector(0))
25/10/25 03:06:29 INFO TaskSchedulerImpl: Adding task set 2.0 with 1 tasks resource
profile 0
25/10/25 03:06:29 WARN TaskSetManager: Stage 2 contains a task of very large size
(1159 KiB). The maximum recommended task size is 1000 KiB.
25/10/25 03:06:29 INFO TaskSetManager: Starting task 0.0 in stage 2.0 (TID 2) (10.2
0.11.250, executor driver, partition 0, PROCESS_LOCAL, 1187035 bytes)
25/10/25 03:06:29 INFO Executor: Running task 0.0 in stage 2.0 (TID 2)
[Stage 2:>                                                          (0 + 1) / 1]
```

```
+----------+---------------+-------------------+-------------------+------------+
|price_area|production_group|start_time         |end_time           |quantity_kwh|
+----------+---------------+-------------------+-------------------+------------+
|NO1       |hydro          |2021-01-01 00:00:00|2021-01-01 01:00:00|2507716.8   |
|NO1       |hydro          |2021-01-01 01:00:00|2021-01-01 02:00:00|2494728.0   |
|NO1       |hydro          |2021-01-01 02:00:00|2021-01-01 03:00:00|2486777.5   |
|NO1       |hydro          |2021-01-01 03:00:00|2021-01-01 04:00:00|2461176.0   |
|NO1       |hydro          |2021-01-01 04:00:00|2021-01-01 05:00:00|2466969.2   |
+----------+---------------+-------------------+-------------------+------------+
only showing top 5 rows
```

```
25/10/25 03:06:30 INFO Executor: Finished task 0.0 in stage 2.0 (TID 2). 2021 bytes
result sent to driver
25/10/25 03:06:30 INFO TaskSetManager: Finished task 0.0 in stage 2.0 (TID 2) in 79
6 ms on 10.20.11.250 (executor driver) (1/1)
25/10/25 03:06:30 INFO TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all
completed, from pool
25/10/25 03:06:30 INFO DAGScheduler: ResultStage 2 (showString at NativeMethodAcces
sorImpl.java:0) finished in 0,923 s
25/10/25 03:06:30 INFO DAGScheduler: Job 2 is finished. Cancelling potential specul
ative or zombie tasks for this job
25/10/25 03:06:30 INFO TaskSchedulerImpl: Killing all running tasks in stage 2: Sta
ge finished
25/10/25 03:06:30 INFO DAGScheduler: Job 2 finished: showString at NativeMethodAcce
ssorImpl.java:0, took 0,942527 s
```

1.8 Checking the datatypes in Spark-Frame

In [72]:
```
spark_df.printSchema()
```

```
root
 |-- price_area: string (nullable = true)
 |-- production_group: string (nullable = true)
 |-- start_time: timestamp (nullable = true)
 |-- end_time: timestamp (nullable = true)
 |-- quantity_kwh: double (nullable = true)
```

1.9 Pushing data to Cassandra

In [42]:
```
# Push data to Cassandra table
spark_df.write \
    .format("org.apache.spark.sql.cassandra") \
    .mode("append") \
    .options(keyspace="elhub", table="production_per_group_hour") \
    .save()
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
Cell In[42], line 6
      1 # Push data to Cassandra table
      2 spark_df.write \
      3     .format(                                ) \
      4     .mode(        ) \
      5     .options(keyspace=       , table=                            ) \
----> 6     .save()

File ~/anaconda3/envs/streamlit/lib/python3.11/site-packages/pyspark/sql/readwrite
r.py:1461, in DataFrameWriter.save(self, path, format, mode, partitionBy, **option
s)
   1459     self.format(format)
   1460 if path is None:
-> 1461     self._jwrite.save()
   1462 else:
   1463     self._jwrite.save(path)

File ~/anaconda3/envs/streamlit/lib/python3.11/site-packages/py4j/java_gateway.py:1
322, in JavaMember.__call__(self, *args)
   1316 command = proto.CALL_COMMAND_NAME +\
   1317     self.command_header +\
   1318     args_command +\
   1319     proto.END_COMMAND_PART
   1321 answer = self.gateway_client.send_command(command)
-> 1322 return_value = get_return_value(
   1323     answer, self.gateway_client, self.target_id, self.name)
   1325 for temp_arg in temp_args:
   1326     if hasattr(temp_arg, "_detach"):

File ~/anaconda3/envs/streamlit/lib/python3.11/site-packages/pyspark/errors/excepti
ons/captured.py:179, in capture_sql_exception.<locals>.deco(*a, **kw)
    177 def deco(*a: Any, **kw: Any) -> Any:
    178     try:
--> 179         return f(*a, **kw)
    180     except Py4JJavaError as e:
    181         converted = convert_exception(e.java_exception)

File ~/anaconda3/envs/streamlit/lib/python3.11/site-packages/py4j/protocol.py:326,
in get_return_value(answer, gateway_client, target_id, name)
    324 value = OUTPUT_CONVERTER[type](answer[2:], gateway_client)
    325 if answer[1] == REFERENCE_TYPE:
--> 326     raise Py4JJavaError(
    327         "An error occurred while calling {0}{1}{2}.\n".
    328         format(target_id, ".", name), value)
    329 else:
    330     raise Py4JError(
    331         "An error occurred while calling {0}{1}{2}. Trace:\n{3}\n".
    332         format(target_id, ".", name, value))

Py4JJavaError: An error occurred while calling o96.save.
: org.apache.spark.SparkClassNotFoundException: [DATA_SOURCE_NOT_FOUND] Failed to f
ind the data source: org.apache.spark.sql.cassandra. Please find packages at `http
s://spark.apache.org/third-party-projects.html`.
        at org.apache.spark.sql.errors.QueryExecutionErrors$.dataSourceNotFoundErro
r(QueryExecutionErrors.scala:725)
        at org.apache.spark.sql.execution.datasources.DataSource$.lookupDataSource
(DataSource.scala:647)
        at org.apache.spark.sql.execution.datasources.DataSource$.lookupDataSourceV
2(DataSource.scala:697)
        at org.apache.spark.sql.DataFrameWriter.lookupV2Provider(DataFrameWriter.sc
ala:873)
        at org.apache.spark.sql.DataFrameWriter.saveInternal(DataFrameWriter.scala:
260)
        at org.apache.spark.sql.DataFrameWriter.save(DataFrameWriter.scala:251)
```

```
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native M
ethod)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMet
hodAccessorImpl.java:77)
        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(Deleg
atingMethodAccessorImpl.java:43)
        at java.base/java.lang.reflect.Method.invoke(Method.java:569)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:374)
        at py4j.Gateway.invoke(Gateway.java:282)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.ClientServerConnection.waitForCommands(ClientServerConnection.java:
182)
        at py4j.ClientServerConnection.run(ClientServerConnection.java:106)
        at java.base/java.lang.Thread.run(Thread.java:840)
Caused by: java.lang.ClassNotFoundException: org.apache.spark.sql.cassandra.Default
Source
        at java.base/java.net.URLClassLoader.findClass(URLClassLoader.java:445)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:592)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:525)
        at org.apache.spark.sql.execution.datasources.DataSource$.$anonfun$lookupDa
taSource$5(DataSource.scala:633)
        at scala.util.Try$.apply(Try.scala:213)
        at org.apache.spark.sql.execution.datasources.DataSource$.$anonfun$lookupDa
taSource$4(DataSource.scala:633)
        at scala.util.Failure.orElse(Try.scala:224)
        at org.apache.spark.sql.execution.datasources.DataSource$.lookupDataSource
(DataSource.scala:633)
        ... 16 more
```

Problem writing data to Cassandra!

I'm unable to push data from Spark to Cassandra. I've spoken with Kristian during class, and we looked into the issue together, but we couldn't find the cause.

I've gone through the following steps to troubleshoot:

1. Verified that the creation of the keyspace "elhub" matches the configuration in the Notebook code.
2. Confirmed that the table "production_per_group_hour" has been created in Cassandra.
3. Checked that the column names in the table match the columns in the DataFrame (see attached image).

Despite this, I'm still unable to write data from Spark to Cassandra.

```
zsh: command not found: DESCRIBE
pialoschbrandt@Pia-sin-MacBook-Air ~ % docker ps

CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS          PORTS                                                NAMES
702a83aca98c   cassandra:4.1   "docker-entrypoint.s…"   47 hours ago   Up 26 minutes   0.0.0.0:9042->9042/tcp, [::]:9042->9042/tcp   cassandra
pialoschbrandt@Pia-sin-MacBook-Air ~ % docker exec -it cassandra bash

root@702a83aca98c:/# cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.1.0 | Cassandra 4.1.10 | CQL spec 3.4.6 | Native protocol v5]
Use HELP for help.
cqlsh> USE elhub;
cqlsh:elhub> DESCRIBE TABLES;

production_per_group_hour

cqlsh:elhub> DESCRIBE TABLE production_per_group_hour;

CREATE TABLE elhub.production_per_group_hour (
    price_area text,
    start_time text,
    end_time text,
    production_group text,
    quantity_kwh double,
    PRIMARY KEY (price_area, start_time)
) WITH CLUSTERING ORDER BY (start_time ASC)
    AND additional_write_policy = '99p'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND cdc = false
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND memtable = 'default'
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND extensions = {}
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';
cqlsh:elhub> SELECT * FROM production_per_group_hour LIMIT 5;

 price_area | start_time | end_time | production_group | quantity_kwh
------------+------------+----------+------------------+--------------

(0 rows)
cqlsh:elhub>
cqlsh:elhub>
```

2. Push data to Mongodb

2.1 Import Libraries

In [73]:
```python
# Importing necessary libraries
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
```

2.2 Creating a new client and connect to the server

In [74]:
```python
USR, PWD = open('no_sync/mongo_db.txt').read().splitlines()

uri = f"mongodb+srv://{USR}:{PWD}@cluster0.12mozyp.mongodb.net/"

# 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!
```

2.3 Test the connection

In [75]:
```python
    # Velg database og collection
    database = client['elhub_data']
    collection = database['production_per_group_hour']

    # ❗ Slett alle dokumenter i collection (men behold strukturen)
    delete_result = collection.delete_many({})
    print(f"🗑️  Slettet {delete_result.deleted_count} dokumenter fra collectionen."
```

```
🗑️  Slettet 1 dokumenter fra collectionen.
```

In [76]:
```
# Testing the connection
try:
    client.admin.command('ping')
    print("✅ Connected to MongoDB!")

    # Selects database and collection
    database = client['elhub_data']
    collection = database['production_per_group_hour']

    # Test to check that the collection works:
    collection.insert_one({"test": "ok"})

    print("Success! Database and collection are working.")
except Exception as e:
    print(" Error:", e)
```

```
✅ Connected to MongoDB!
Success! Database and collection are working.
```

2.4 Load the Dataframe

In [ ]:
```
# ------------------------------
# 3️⃣ Upload DataFrame to MongoDB (only if not exists)
# ------------------------------

# Sjekk om collection allerede inneholder data
existing_count = collection.count_documents({})

if existing_count <= 1:
    data = df_all.to_dict("records")
    collection.insert_many(data)
    print("✅ Elhub-data er lastet opp til MongoDB (første gang).")
else:
    print(f"ℹ️ Collection inneholder allerede {existing_count} dokumenter – hopper
```

```
✅ Elhub-data er lastet opp til MongoDB (første gang).
```

2.5 Checks that the data is loaded

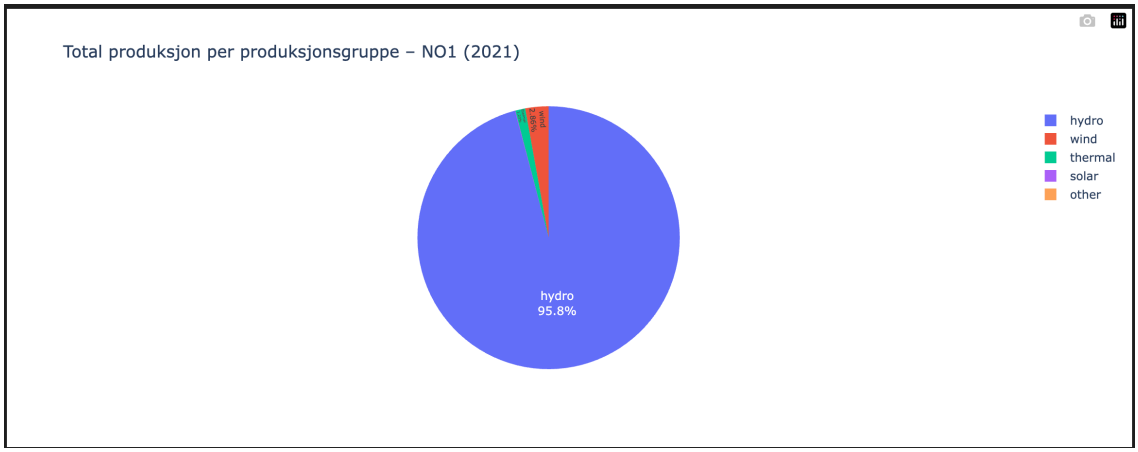
In [80]:
```
# Select the database and collection
database = client['elhub_data']
collection = database['production_per_group_hour']

# Count the number of documents (rows)
num_docs = collection.count_documents({})
print(f"Number of documents in 'production_per_group_hour': {num_docs}")
```

```
Number of documents in 'production_per_group_hour': 215354
```

2.6 Make some plots of the data

In [ ]:
```
# Sørg for at tid er datetime
df_ready["start_time"] = pd.to_datetime(df_ready["start_time"])

# Legg til måned
df_ready["month"] = df_ready["start_time"].dt.month

# Velg område og måned
chosen_area = "NO1"
chosen_month = 1  # Januar

# Filtrer data
df_january = df_ready[
    (df_ready["price_area"] == chosen_area) &
    (df_ready["month"] == chosen_month)
```

```
]

# Gruppér og sorter
line_data = (
    df_january.groupby(["start_time", "production_group"], as_index=False)["quantit
    .sort_values("start_time")
)

# Plot
fig = px.line(
    line_data,
    x="start_time",
    y="quantity_kwh",
    color="production_group",
    title=f"Produksjon per time – {calendar.month_name[chosen_month]} ({chosen_area
    labels={
        "start_time": "Tid",
        "quantity_kwh": "Produksjon (kWh)",
        "production_group": "Produksjonsgruppe"
    },
    template="plotly_white"
)

fig.show()
```



In [50]:

```
# Sørg for at start_time er datetime
df_ready["start_time"] = pd.to_datetime(df_ready["start_time"])
df_ready["year"] = df_ready["start_time"].dt.year

# Velg område og år
chosen_area = "NO1"
chosen_year = 2021

# Filtrer data
df_year = df_ready[
    (df_ready["price_area"] == chosen_area) &
    (df_ready["year"] == chosen_year)
]

# Summer total produksjon per produksjonsgruppe
pie_data = (
    df_year.groupby("production_group", as_index=False)["quantity_kwh"].sum()
)

# Lag kakediagram
fig = px.pie(
    pie_data,
    values="quantity_kwh",
    names="production_group",
    title=f"Total produksjon per produksjonsgruppe – {chosen_area} ({chosen_year})"
)
```

```
fig.update_traces(textposition="inside", textinfo="percent+label")
fig.update_layout(template="plotly_white")

fig.show()
```