Assignment 2

IND320

Sigvard Bratlie
https://github.com/sigvardbratlie/ind320
https://ind320-h63n5qj5uc26acyzlq3x39.streamlit.app/

AI-usage
Used for content refinement (spelling, syntax debugging) and structuring/drafting log entries.
Especially helpful for issues related to cassandra-spark connection errors.

LOG

I retrieved hourly production data for all price areas from Elhub (endpoint: `PRODUCTION_PER_GROUP_MBA_HOUR`) for the full year 2021. Because the API has limits on time windows per request, I looped over valid intervals and extracted only the `productionPerGroupMbaHour` list. All timestamps are handled in UTC to avoid confusion. The raw records are converted to a DataFrame for cleaning and pre-processesing before storage.

For local storage I use Cassandra with Spark as the read/write layer. I define the keyspace and table schema, write the hourly time series, and verify the load with a simple `select` / `show()`. For the analysis part, I use Spark to select exactly the required columns: `priceArea`, `productionGroup`, `startTime`, and `quantityKwh`. I then convert to pandas for plotting and quick transformations. The pie chart aggregates total annual production per group for a chosen price area. The line plot is explicitly filtered to January (the first month), with one line per production group. This corrects my earlier version that inadvertently summarized across the whole year.

Data is then pushed to a remote MongoDB. I perform a straightforward `insert_many`. On page 4 of the app, I connect to MongoDB via `st.secrets` (to avoid exposing credentials on GitHub). The layout uses two columns with `st.columns`. The left column contains a `st.radio` to select the price area and displays the same pie chart logic as in the notebook. The right column uses `st.pills` to include/exclude production groups and a single-month selector; the combination of price area, groups, and month drives the line plot. Below the columns, I add a short `st.expander` documenting the data source.

Git workflow: I develop on a separate branch until peer review/teacher feedback and then merge into main. The notebook has clear headings for navigation, and code cells are commented for reproducibility. Before exporting to PDF/HTML, I run all cells to ensure that tables and figures render properly.

In [6]:
```python
import requests
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Use the Elhub API to retrieve data

In [7]:
```python
def get_period(month : int):
    '''
    A function to fetch a single month's data from the Elhub API.

    Args:
        month (int): The month for which to fetch data (1-12).
    Returns:
        dict: The JSON response from the API.
    '''
    entity = "price-areas" #Fetched per price areas
    dataset = "PRODUCTION_PER_GROUP_MBA_HOUR" #Dataset to fetch
    base_url = f"https://api.elhub.no/energy-data/v0/{entity}" #Base URL for the API
    params = { #Parameters for the API request
        "dataset": dataset,
        "startDate": f"2021-{str(month).zfill(2)}-01", #using zfill to ensure two digits
        "endDate": f"2021-{str(month).zfill(2)}-31"
    }
    try:
        response = requests.get(base_url, params=params)
        response.raise_for_status()  # Raise an error for HTTP errors
        return response.json()
    except requests.RequestException as e:
        print(f"Error fetching data: {e}")

def ext_single(data):
    '''
    A function to extract the productionPerGroupMbaHour list from the API response.

    Args:
        data (dict): The JSON response from the API.
    Returns:
```

```
        list: A list of productionPerGroupMbaHour entries.
    '''
    ext_data = [] #tmp list to hold the data for a single month
    for price_area in data.get("data"):
        attrs = price_area.get("attributes")
        if attrs:
            ext_data.extend(attrs.get("productionPerGroupMbaHour"))
    return ext_data
```

In [8]:
```
#Loops trough all months, fetches and extracts the data, and combines it into a single DataFrame
all_data = []
for i in range(1, 13):
    data = get_period(i)
    if data:
        ext_data = ext_single(data)
        all_data.extend(ext_data)

df = pd.DataFrame(all_data)
```

In [9]:
```
df.iloc[len(df)//2] #+02 for summer time
```

Out[9]:
```
endTime            2021-07-22T13:00:00+02:00
lastUpdatedTime    2024-12-20T10:35:40+01:00
priceArea                                NO1
productionGroup                      thermal
quantityKwh                        15984.755
startTime          2021-07-22T12:00:00+02:00
Name: 104124, dtype: object
```

In [10]:
```
df.iloc[0] #+01 for winter time
```

Out[10]:
```
endTime            2021-01-01T01:00:00+01:00
lastUpdatedTime    2024-12-20T10:35:40+01:00
priceArea                                NO1
productionGroup                        hydro
quantityKwh                          2507716.8
startTime          2021-01-01T00:00:00+01:00
Name: 0, dtype: object
```

In [11]:
```
#Convert the time columns to datetime format
df["endTime"] = pd.to_datetime(df["endTime"], errors='coerce',utc= True) #utc = True to avoid timezone issues
df["startTime"] = pd.to_datetime(df["startTime"],errors='coerce',utc = True)
df["lastUpdatedTime"] = pd.to_datetime(df["lastUpdatedTime"],errors='coerce',utc = True)
```

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 208248 entries, 0 to 208247
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   endTime          208248 non-null  datetime64[ns, UTC]
 1   lastUpdatedTime  208248 non-null  datetime64[ns, UTC]
 2   priceArea        208248 non-null  object
 3   productionGroup  208248 non-null  object
 4   quantityKwh      208248 non-null  float64
 5   startTime        208248 non-null  datetime64[ns, UTC]
dtypes: datetime64[ns, UTC](3), float64(1), object(2)
memory usage: 9.5+ MB
```

In [13]:
```
df.tail()
```

Out[13]:

	endTime	lastUpdatedTime	priceArea	productionGroup	quantityKwh	startTime
208243	2021-12-30 19:00:00+00:00	2024-10-27 06:46:34+00:00	NO5	wind	0.0	2021-12-30 18:00:00+00:00
208244	2021-12-30 20:00:00+00:00	2024-10-27 06:46:34+00:00	NO5	wind	0.0	2021-12-30 19:00:00+00:00
208245	2021-12-30 21:00:00+00:00	2024-10-27 06:46:34+00:00	NO5	wind	0.0	2021-12-30 20:00:00+00:00
208246	2021-12-30 22:00:00+00:00	2024-10-27 06:46:34+00:00	NO5	wind	0.0	2021-12-30 21:00:00+00:00
208247	2021-12-30 23:00:00+00:00	2024-10-27 06:46:34+00:00	NO5	wind	0.0	2021-12-30 22:00:00+00:00

In [14]:
```
df.describe().T
```

Out[14]:

	count	mean	std	min	25%	50%	75%	max
quantityKwh	208248.0	731247.787576	1.552956e+06	0.0	7.38575	13201.6275	362502.965	9715193.0

In [15]:
```
df["productionGroup"].value_counts()
```

Out[15]:
```
productionGroup
hydro      42355
other      42355
solar      42355
thermal    42355
wind       38828
Name: count, dtype: int64
```

In [9]:
```
df["priceArea"].value_counts()
```

Out[9]:
```
priceArea
NO1    42355
NO2    42355
NO3    42355
NO4    42355
NO5    38828
Name: count, dtype: int64
```

In [10]:
```
df["startTime"].dt.month.value_counts().sort_index() #ensuring all month are present
```

Out[10]:
```
startTime
1     17280
2     15552
3     17280
4     16704
5     17280
6     17378
7     18000
8     18000
9     17400
10    17975
11    17400
12    17999
Name: count, dtype: int64
```

Insert the data into Cassandra using Spark

In [16]:
```
from cassandra.cluster import Cluster
from pyspark.sql import SparkSession
```

In [ ]:
```
#!docker start my_cassandra
#wait for cassandra container to start
```

```
my_cassandra
```

In [17]:
```
cluster = Cluster(["localhost"], port=9042) #defining the cassandra cluster
session = cluster.connect() #connection to local cassandra container
```

In [18]:
```
# !which java #to ensure java is installed
# !java -version #to check java version
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.cassandra.connection.port', '9042').\
    config("spark.driver.extraJavaOptions", "-Dhadoop.security.subject.provider.backend=hadoop").\
    getOrCreate() #connecting spark and cassandra
print(spark.version) #checking spark version
```

```
25/10/23 09:15:33 WARN Utils: Your hostname, Sigvard-sin-MacBook-Pro.local resolves to a loopback address: 127.0.0.
1; using 10.42.89.105 instead (on interface en0)
25/10/23 09:15:33 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Ivy Default Cache set to: /Users/sigvardbratlie/.ivy2/cache
The jars for the packages stored in: /Users/sigvardbratlie/.ivy2/jars
com.datastax.spark#spark-cassandra-connector_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-2615f920-94b0-4ec8-978d-881092603705;1.0
        confs: [default]
:: loading settings :: url = jar:file:/Users/sigvardbratlie/miniconda3/envs/datsci/lib/python3.11/site-packages/pysp
ark/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 228ms :: artifacts dl 8ms
        :: 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-2615f920-94b0-4ec8-978d-881092603705
        confs: [default]
        0 artifacts copied, 16 already retrieved (0kB/6ms)
25/10/23 09:15:34 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-jav
a classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
3.5.1
```

In [15]:
```
session.execute("CREATE KEYSPACE IF NOT EXISTS elhub WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_
```

Out[15]:
```
<cassandra.cluster.ResultSet at 0x15ccf9c50>
```

In [16]:
```
session.set_keyspace("elhub") #setting keyspace to use
```

In [ ]:
```
df["ind"] = df.index #adding an index column to use as primary key
```

In [ ]:
```
sdf = spark.createDataFrame(df) #creating a spark dataframe from the pandas dataframe
```

```
sdf.printSchema() #printing the schema
```

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

In [ ]:
```
#Building the CQL query to create an empty table
def build_query(df):
    '''
    A function to build a CQL query to create a table based on the DataFrame schema.

    Args:
        df (pd.DataFrame): The DataFrame for which to build the CQL query.
    Returns:
        str: The CQL query string.
    '''
    sql_query = 'CREATE TABLE IF NOT EXISTS prod_data ('
    for name,dtype in df.dtypes.items():
        if name == "ind":
            sql_query += f"ind int PRIMARY KEY, "
        else:
            if dtype == 'object':
                dtype = 'text'
```

```
            elif dtype == 'int64':
                dtype = 'int'
            elif dtype == 'float64':
                dtype = 'float'
            elif dtype == 'datetime64[ns, UTC]':
                dtype = 'timestamp'
            else:
                dtype = 'text'  #default to text if unknown type
            sql_query += f"{name} {dtype}, "

    sql_query += ");"
    return sql_query
sql_query = build_query(df)
```

Out[ ]:
```
'CREATE TABLE IF NOT EXISTS prod_data (endTime timestamp, lastUpdatedTime timestamp, priceArea text, productionGro
up text, quantityKwh float, startTime timestamp, ind int PRIMARY KEY, );'
```

In [ ]:
```
session.execute("DROP TABLE IF EXISTS elhub.prod_data;") #dropping table if it exists
session.execute(sql_query) #creating the table
```

Out[ ]:
```
<cassandra.cluster.ResultSet at 0x1756f11d0>
```

In [ ]:
```
sdf = sdf.toDF(*[c.lower() for c in sdf.columns]) #lower column names

sdf.write.format("org.apache.spark.sql.cassandra")\
   .options(table="prod_data", keyspace="elhub")\
   .mode("append").save() #writing data to cassandra table
```

```
25/10/06 21:40:49 WARN TaskSetManager: Stage 2 contains a task of very large size (1135 KiB). The maximum recommende
d task size is 1000 KiB.
```

In [19]:
```
spark.read.format("org.apache.spark.sql.cassandra")\
    .options(table="prod_data",keyspace="elhub").load().show() #verifying data is written
```

```
+------+-------------------+-------------------+---------+---------------+-----------+-------------------+
|   ind|            endtime|    lastupdatedtime|pricearea|productiongroup|quantitykwh|          starttime|
+------+-------------------+-------------------+---------+---------------+-----------+-------------------+
| 52907|2021-04-02 12:00:00|2024-12-20 10:35:40|      NO1|           wind|  120816.92|2021-04-02 11:00:00|
|181542|2021-11-15 07:00:00|2024-10-27 02:04:14|      NO3|          solar|     36.486|2021-11-15 06:00:00|
|132343|2021-08-28 08:00:00|2024-12-20 10:35:40|      NO4|          solar|      8.912|2021-08-28 07:00:00|
|154666|2021-09-22 11:00:00|2024-12-20 10:35:40|      NO5|           wind|        0.0|2021-09-22 10:00:00|
| 32946|2021-03-05 19:00:00|2024-12-20 10:35:40|      NO1|          hydro|  2373319.0|2021-03-05 18:00:00|
|137212|2021-08-21 05:00:00|2024-12-20 10:35:40|      NO5|           wind|        0.0|2021-08-21 04:00:00|
|177968|2021-11-11 09:00:00|2024-10-27 01:19:06|      NO2|          solar|     42.121|2021-11-11 08:00:00|
| 81702|2021-05-22 07:00:00|2024-12-20 10:35:40|      NO5|          hydro|  1220537.4|2021-05-22 06:00:00|
|183529|2021-11-11 02:00:00|2024-10-27 01:19:06|      NO4|          hydro|  1069807.2|2021-11-11 01:00:00|
|183489|2021-11-09 10:00:00|2024-10-27 00:59:23|      NO4|          hydro|  2509049.8|2021-11-09 09:00:00|
|125783|2021-08-25 00:00:00|2024-12-20 10:35:40|      NO2|        thermal|  12021.626|2021-08-24 23:00:00|
| 91689|2021-06-28 10:00:00|2024-12-20 10:35:40|      NO3|          hydro|  2128446.2|2021-06-28 09:00:00|
| 45623|2021-03-24 17:00:00|2024-12-20 10:35:40|      NO4|          solar|      0.341|2021-03-24 16:00:00|
| 47419|2021-03-09 16:00:00|2024-12-20 10:35:40|      NO5|          hydro|  5495359.5|2021-03-09 15:00:00|
|207715|2021-12-08 20:00:00|2024-10-27 04:28:48|      NO5|           wind|        0.0|2021-12-08 19:00:00|
|156985|2021-10-30 02:00:00|2024-12-20 10:35:40|      NO1|          solar|     21.411|2021-10-30 01:00:00|
|107081|2021-07-25 18:00:00|2024-12-20 10:35:40|      NO2|          solar|   4468.718|2021-07-25 17:00:00|
|152625|2021-09-24 10:00:00|2024-12-20 10:35:40|      NO5|          other|        0.0|2021-09-24 09:00:00|
| 28691|2021-02-17 12:00:00|2024-12-20 10:35:40|      NO4|          solar|       7.42|2021-02-17 11:00:00|
|125630|2021-08-18 15:00:00|2024-12-20 10:35:40|      NO2|        thermal|   8083.639|2021-08-18 14:00:00|
+------+-------------------+-------------------+---------+---------------+-----------+-------------------+
only showing top 20 rows
```

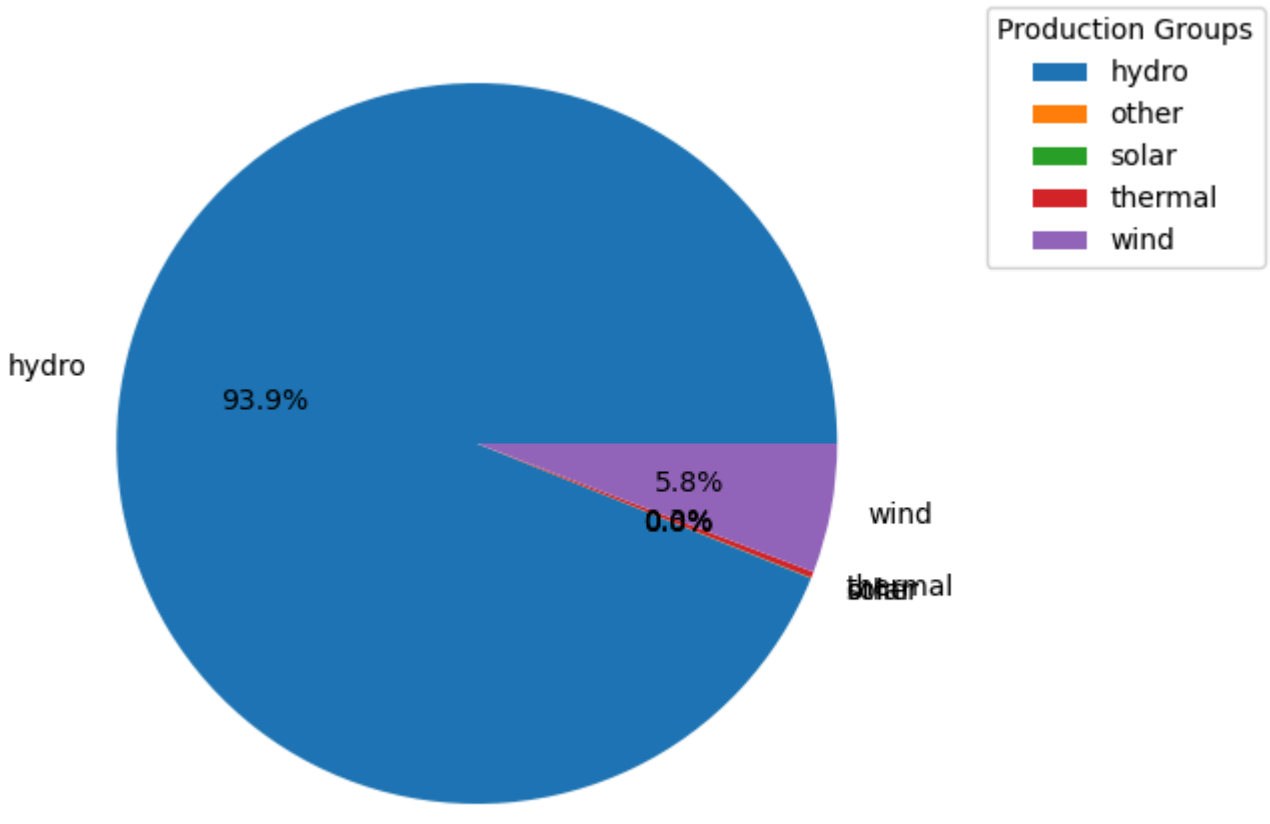
In [20]:
```
spark.read.format("org.apache.spark.sql.cassandra")\
    .options(table="prod_data",keyspace="elhub").load()\
    .createOrReplaceTempView("prod_data_view") #create view
```

In [21]:
```
query = "SELECT pricearea,productiongroup,starttime,quantitykwh FROM prod_data_view;"
sv = spark.sql(query).toPandas() #read view into pandas
```

Create plots

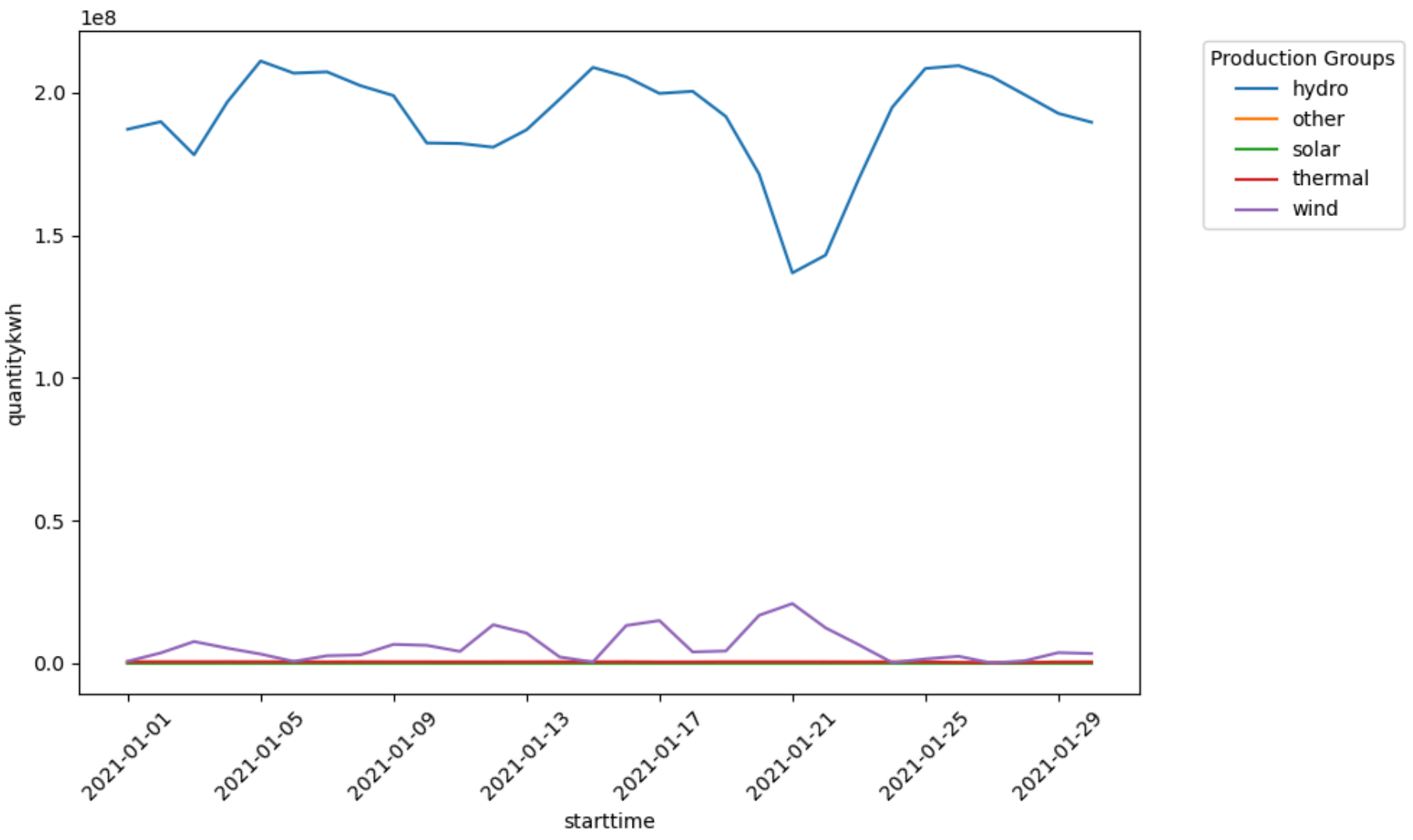
In [22]:
```
#A pie chart for the total production of the year from a chosen price area, where each piece of the pie is one of t
no2 = sv[(sv["pricearea"] == "NO2")] #select price area NO2
data = no2.groupby("productiongroup")["quantitykwh"].sum().reset_index() #create data

plt.pie(data["quantitykwh"], labels=data["productiongroup"], autopct='%1.1f%%'); #plotting pie chart
plt.tight_layout() #ensuring tight layout
plt.legend(title="Production Groups", bbox_to_anchor=(1.05, 1), loc='upper left') #adding legend
plt.show()
```



In [23]:
```
#A line plot for the first month of the year for a chosen price area. Make separate lines for each production group
data = no2[no2["starttime"].dt.month == no2["starttime"].min().month] #selecting data for January
data = data.groupby(["productiongroup",
                     pd.Grouper(key="starttime",
                                freq="D")])["quantitykwh"].sum().reset_index()

plt.figure(figsize=(10,6))
sns.lineplot(data = data, x = "starttime", y ="quantitykwh",hue = "productiongroup") #plotting line plot
plt.xticks(rotation=45)
plt.legend(title="Production Groups", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
```



Insert the Spark-extracted data into your MongoDB

In [24]:
```
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
import os
from dotenv import load_dotenv
load_dotenv() #loading the mongo db password as an environment variable
```

Out[24]: True

In [25]:
```
def get_database():
```

```
    # Connection string from MongoDB
    CONNECTION_STRING = f"mongodb+srv://sigvardbratlie:{os.getenv('MONGODB_PASSWORD')}@cluster0.y7mplij.mongodb.net/

    # Create a connection using MongoClient.
    client = MongoClient(CONNECTION_STRING)
    try:
        client.admin.command("ping")
        print(f'Everything Okay') #print if connection is successful
        return client
    except Exception as e:
      print(e)

client = get_database()
```

```
Everything Okay
```

In [ ]:
```
db = client["elhub"] #create database
collection = db["prod_data"] #create collection
```

In [ ]:
```
collection.insert_many(sv.to_dict('records')) #Insert the Spark-extracted data into your MongoDB.
```