

IND320 - Assignment 2

Name: Eskil Torp Skogsholm

GitHub link: <https://github.com/esksko/IND320-assignments-esksko>

Streamlit app: <https://ind320-esksko.streamlit.app/>

AI usage

During development of this assignment, I utilized AI tools to assist me. I used GitHub copilot for code completion, and ChatGPT/Claude for explanations and some code snippets. These tools helped me understand APIs better and debug issues. AI was especially useful when troubleshooting issues with installing Spark.

Log

First, I made sure that Cassandra, Spark and MongoDB were working as intended on my computer. Previously, I had some issues with compatibility with Python versions (Even within the range specified in the setup section of the book), but by using version 3.11.13 everything works correctly. In the first part of my Jupyter Notebook, I tested each of these tools separately using code/examples from the classes. Cassandra and Docker were easy to use, and MongoDB became intuitive after some tinkering. Fortunately, I didn't have too many issues with getting Spark to work with Cassandra as well. Even if it took some troubleshooting and AI usage to make sense of error messages. The API was also quite easy to work with. I struggled a little bit with finding out how to only get data from a specific time-range, but after I figured out how to do that, it was quite simple.

After testing that all these parts were working correctly, I moved on to using the Elhub API to retrieve the required data, transforming it into a DataFrame, and inserting it into Cassandra using Spark. The only minor issue I encountered here was that either Cassandra or Spark changed the column names to be fully lower-case. This was easy to fix by just making the DataFrame columns lower-case as well. Inserting the data into MongoDB and plotting the pie- and line-charts went smoothly.

With the experience I gained from working on assignment 1, this assignment felt easier in terms of working with Streamlit. With GitHub and Streamlit already set up, I just changed the dummy page (page 4) to include the required elements as described in the assignment description. Connecting to MongoDB was intuitive, although I had to add 0.0.0.0/0 to the IP Access List. There is probably a better solution, but for that, I would need to find what IP address Streamlit uses when the app is deployed.

During this process, I also experienced the importance of caching the data when loading it into Streamlit. By doing this, the loading times were significantly reduced, and the app became much more responsive.

```
In [ ]: import os

# Setting environment variables for Spark and Hadoop
os.environ["HADOOP_HOME"] = "C:/Hadoop/hadoop-3.3.1"

os.environ["JAVA_HOME"] = "C:/Program Files/Microsoft/jdk-11.0.28.6-hotspot"

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

os.environ["HADOOP_VERSION"] = "without"
```

Connecting to and testing Cassandra

```
In [ ]: from cassandra.cluster import Cluster

# Connecting to a Local Cassandra instance
cluster = Cluster(["localhost"], port=9042)
session = cluster.connect()
```

```
In [ ]: # Setting keyspace, and testing table creation
session.set_keyspace("my_first_keyspace")
session.execute("DROP TABLE IF EXISTS my_first_keyspace.planets;")
session.execute("CREATE TABLE IF NOT EXISTS planets (planet text PRIMARY KEY, dista
```

```
Out[ ]: <cassandra.cluster.ResultSet at 0x1952e3a0c90>
```

Activating Spark

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

# Configuring Spark session to connect to Cassandra
spark = SparkSession.builder.appName('SparkCassandraApp').\
    config('spark.jars.packages', 'com.datastax.spark:spark-cassandra-connector_2.1').\
    config('spark.cassandra.connection.host', 'localhost').\
    config('spark.sql.extensions', 'com.datastax.spark.connector.CassandraSparkExte').\
    config('spark.sql.catalog.mycatalog', 'com.datastax.spark.connector.datasourc').\
    config('spark.cassandra.connection.port', '9042').getOrCreate()
```

Testing MongoDB

```
In [ ]: from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
import tomllib
```

```

# Getting username and password for MongoDB connection

with open("../streamlit/secrets.toml", "rb") as f:
    cfg = tomllib.load(f)

USR = cfg["MongoDB"]["username"]
PWD = cfg["MongoDB"]["pwd"]

uri = f"mongodb+srv://esksko:{PWD}@ind320-esksko.5nbj7x0.mongodb.net/?retryWrites=t

# 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!

```
In [ ]: # Stopping Spark session
try:
    spark.stop()
except ConnectionRefusedError:
    print("Spark session already stopped.")
```

Testing API connection (Elhub API)

```
In [ ]: import requests

# Testing Elhub API connection

entity = "price-areas"
dataset = "CONSUMPTION_PER_GROUP_MBA_HOUR"

URL = f"https://api.elhub.no/energy-data/v0/{entity}?dataset={dataset}"

response = requests.get(URL)

print(response.status_code)
```

200

1: Elhub API

```
In [ ]: from datetime import datetime, timedelta
import calendar

import requests
import pandas as pd

# Example for one month
```

```

start_date = "2025-01-01T00:00:00%2B01"
end_date = "2025-01-31T23:59:59%2B01"

# Prepare month ranges for 2021 in correct format
monthly_ranges = []
for month in range(1, 13):
    # First day of the month
    start = datetime(2021, month, 1, 0, 0, 0)

    # Last day of the month
    last_day = calendar.monthrange(2021, month)[1]
    end = datetime(2021, month, last_day, 23, 59, 59)

    # Format as ISO8601 with URL-encoded +01
    start_str = start.strftime("%Y-%m-%dT%H:%M:%S") + "%2B01"
    end_str = end.strftime("%Y-%m-%dT%H:%M:%S") + "%2B01"

    monthly_ranges.append((start_str, end_str))

# Check result
print(monthly_ranges)

```

```

[('2021-01-01T00:00:00%2B01', '2021-01-31T23:59:59%2B01'), ('2021-02-01T00:00:00%2B01', '2021-02-28T23:59:59%2B01'), ('2021-03-01T00:00:00%2B01', '2021-03-31T23:59:59%2B01'), ('2021-04-01T00:00:00%2B01', '2021-04-30T23:59:59%2B01'), ('2021-05-01T00:00:00%2B01', '2021-05-31T23:59:59%2B01'), ('2021-06-01T00:00:00%2B01', '2021-06-30T23:59:59%2B01'), ('2021-07-01T00:00:00%2B01', '2021-07-31T23:59:59%2B01'), ('2021-08-01T00:00:00%2B01', '2021-08-31T23:59:59%2B01'), ('2021-09-01T00:00:00%2B01', '2021-09-30T23:59:59%2B01'), ('2021-10-01T00:00:00%2B01', '2021-10-31T23:59:59%2B01'), ('2021-11-01T00:00:00%2B01', '2021-11-30T23:59:59%2B01'), ('2021-12-01T00:00:00%2B01', '2021-12-31T23:59:59%2B01')]

```

```

In [ ]: entity = "price-areas"
dataset = "PRODUCTION_PER_GROUP_MBA_HOUR"

all_data = []

# Sending API requests for each month and collecting data
for start_date, end_date in monthly_ranges:
    url = f"https://api.elhub.no/energy-data/v0/{entity}?dataset={dataset}&startDat
    response = requests.get(url)

    # 200 == Good response
    if response.status_code == 200:
        data = response.json()
        # Iterate over all price areas
        for entry in data.get("data", []):
            monthly_list = entry.get("attributes", {}).get("productionPerGroupMbaHo
            all_data.extend(monthly_list)

        print(f"Retrieved {len(monthly_list)} records for {start_date} to {end_date}
    else:
        print(f"Failed to retrieve data for {start_date} to {end_date}: {response.s

    print(f"Total records collected: {len(all_data)}")

```

```
Retrieved 2976 records for 2021-01-01T00:00:00%2B01 to 2021-01-31T23:59:59%2B01.  
Retrieved 2688 records for 2021-02-01T00:00:00%2B01 to 2021-02-28T23:59:59%2B01.  
Retrieved 2972 records for 2021-03-01T00:00:00%2B01 to 2021-03-31T23:59:59%2B01.  
Retrieved 2880 records for 2021-04-01T00:00:00%2B01 to 2021-04-30T23:59:59%2B01.  
Retrieved 2976 records for 2021-05-01T00:00:00%2B01 to 2021-05-31T23:59:59%2B01.  
Retrieved 3576 records for 2021-06-01T00:00:00%2B01 to 2021-06-30T23:59:59%2B01.  
Retrieved 3720 records for 2021-07-01T00:00:00%2B01 to 2021-07-31T23:59:59%2B01.  
Retrieved 3720 records for 2021-08-01T00:00:00%2B01 to 2021-08-31T23:59:59%2B01.  
Retrieved 3600 records for 2021-09-01T00:00:00%2B01 to 2021-09-30T23:59:59%2B01.  
Retrieved 3725 records for 2021-10-01T00:00:00%2B01 to 2021-10-31T23:59:59%2B01.  
Retrieved 3600 records for 2021-11-01T00:00:00%2B01 to 2021-11-30T23:59:59%2B01.  
Retrieved 3720 records for 2021-12-01T00:00:00%2B01 to 2021-12-31T23:59:59%2B01.  
Total records collected: 215353
```

```
In [ ]: # Creating DataFrame from collected data  
df = pd.DataFrame(all_data)  
  
df.head()
```

```
Out[ ]:      endTime  lastUpdatedTime  priceArea  productionGroup  quantityKwh  
0  2021-01-  
  01T01:00:00+01:00  2024-12-  
                    20T10:35:40+01:00    NO1        hydro  2507716.8  01T00:00  
1  2021-01-  
  01T02:00:00+01:00  2024-12-  
                    20T10:35:40+01:00    NO1        hydro  2494728.0  01T01:00  
2  2021-01-  
  01T03:00:00+01:00  2024-12-  
                    20T10:35:40+01:00    NO1        hydro  2486777.5  01T02:00  
3  2021-01-  
  01T04:00:00+01:00  2024-12-  
                    20T10:35:40+01:00    NO1        hydro  2461176.0  01T03:00  
4  2021-01-  
  01T05:00:00+01:00  2024-12-  
                    20T10:35:40+01:00    NO1        hydro  2466969.2  01T04:00
```

```
In [71]: import os  
# Setting up environment variables for Spark and Hadoop  
os.environ["HADOOP_HOME"] = "C:/Hadoop/hadoop-3.3.1"  
  
os.environ["JAVA_HOME"] = "C:/Program Files/Microsoft/jdk-11.0.28.6-hotspot"  
  
os.environ["PYSPARK_PYTHON"] = "python"  
os.environ["PYSPARK_DRIVER_PYTHON"] = "python"  
  
os.environ["HADOOP_VERSION"] = "without"
```

```
In [72]: from cassandra.cluster import Cluster  
  
# Connecting to Cassandra  
cluster = Cluster(["localhost"], port=9042)  
session = cluster.connect()
```

```
In [73]: # Creating keyspace and setting it  
session.execute("CREATE KEYSPACE IF NOT EXISTS assignment_2_keyspace WITH REPLICATI
```

```

session.set_keyspace("assignment_2_keyspace")

# Creating table for production per group hour data
session.execute("DROP TABLE IF EXISTS production_per_group_hour;")
session.execute("""
CREATE TABLE IF NOT EXISTS production_per_group_hour (
    pricearea text,
    productiongroup text,
    starttime timestamp,
    endtime timestamp,
    quantitykwh double,
    lastupdatedtime timestamp,
    PRIMARY KEY ((pricearea, productiongroup), starttime)
);
""")

```

Out[73]: <cassandra.cluster.ResultSet at 0x19530b37d90>

```

In [ ]: from pyspark.sql import SparkSession

# Configuring Spark session to connect to Cassandra
spark = SparkSession.builder.appName('Assignment2SparkCassandraApp').\
    config('spark.jars.packages', 'com.datastax.spark:spark-cassandra-connector_2.1')
    config('spark.cassandra.connection.host', 'localhost').\
    config('spark.sql.extensions', 'com.datastax.spark.connector.CassandraSparkExte')
    config('spark.sql.catalog.mycatalog', 'com.datastax.spark.connector.datasourc')
    config('spark.cassandra.connection.port', '9042').getOrCreate()

```

```

In [ ]: # Converting pandas DataFrame into Spark DataFrame
sdf = spark.createDataFrame(df)
sdf.show(5)

```

	endTime	lastUpdatedTime	priceArea	productionGroup	quantityKwh
startTime					
2021-01-01T01:00:00	2024-12-20T10:35:00	N01	hydro	2507716.8	202
1-01-01T00:00:00					
2021-01-01T02:00:00	2024-12-20T10:35:00	N01	hydro	2494728.0	202
1-01-01T01:00:00					
2021-01-01T03:00:00	2024-12-20T10:35:00	N01	hydro	2486777.5	202
1-01-01T02:00:00					
2021-01-01T04:00:00	2024-12-20T10:35:00	N01	hydro	2461176.0	202
1-01-01T03:00:00					
2021-01-01T05:00:00	2024-12-20T10:35:00	N01	hydro	2466969.2	202
1-01-01T04:00:00					

only showing top 5 rows

```

In [ ]: # Displaying schema and column names of the Spark DataFrame
print("DataFrame columns:")

```

```
sdf.printSchema()
print("\nColumn names:", sdf.columns)
```

DataFrame columns:

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

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

```
In [ ]: from pyspark.sql.functions import to_timestamp

# Converting startTime, endTime, and LastUpdatedTime to timestamp type

sdf = (
    sdf
        .withColumn("starttime", to_timestamp("starttime"))
        .withColumn("endtime", to_timestamp("endtime"))
        .withColumn("lastupdatedtime", to_timestamp("lastupdatedtime")))
)

# Converting all column names to Lowercase
sdf = sdf.toDF(*[c.lower() for c in sdf.columns])
sdf.show(5)
```

```
+-----+-----+-----+-----+
|      endtime|  lastupdatedtime|pricearea|productiongroup|quantitykwh|
starttime|
+-----+-----+-----+-----+
|2021-01-01 01:00:00|2024-12-20 10:35:40|      N01|       hydro| 2507716.8|2021-
01-01 00:00:00|
|2021-01-01 02:00:00|2024-12-20 10:35:40|      N01|       hydro| 2494728.0|2021-
01-01 01:00:00|
|2021-01-01 03:00:00|2024-12-20 10:35:40|      N01|       hydro| 2486777.5|2021-
01-01 02:00:00|
|2021-01-01 04:00:00|2024-12-20 10:35:40|      N01|       hydro| 2461176.0|2021-
01-01 03:00:00|
|2021-01-01 05:00:00|2024-12-20 10:35:40|      N01|       hydro| 2466969.2|2021-
01-01 04:00:00|
+-----+-----+-----+-----+
-----+
only showing top 5 rows
```

```
In [78]: # Writing data into Cassandra
```

```
sdf.write\
    .format("org.apache.spark.sql.cassandra") \
    .options(keyspace="assignment_2_keyspace", table="production_per_group_hour") \
```

```
.mode("append") \
.save()
```

```
In [79]: # Extracting priceArea, productionGroup, startTime, and quantityKwh from Cassandra.

extracted_df = spark.read \
    .format("org.apache.spark.sql.cassandra") \
    .options(keyspace="assignment_2_keyspace", table="production_per_group_hour") \
    .load() \
    .select("pricearea", "productiongroup", "starttime", "quantitykwh")

extracted_df.show(5)

+-----+-----+-----+
|pricearea|productiongroup| starttime|quantitykwh|
+-----+-----+-----+
| NO4 | hydro | 2021-01-01 00:00:00 | 3740830.0 |
| NO4 | hydro | 2021-01-01 01:00:00 | 3746663.5 |
| NO4 | hydro | 2021-01-01 02:00:00 | 3712439.8 |
| NO4 | hydro | 2021-01-01 03:00:00 | 3699229.0 |
| NO4 | hydro | 2021-01-01 04:00:00 | 3685393.8 |
+-----+-----+-----+
only showing top 5 rows
```

```
In [88]: import matplotlib.pyplot as plt

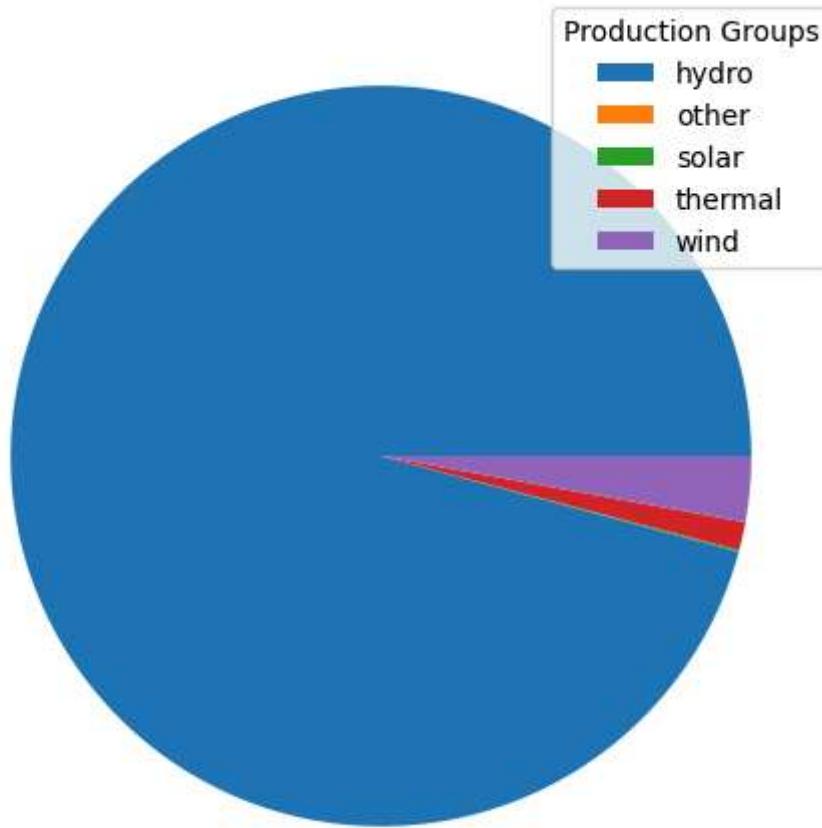
pdf = extracted_df.toPandas()

# Choosing one price area for visualization
chosen_area = "NO1"
area_data = pdf[pdf["pricearea"] == chosen_area]

# Group by production group and sum quantities
production_by_group = area_data.groupby("productiongroup")["quantitykwh"].sum()

# Creating pie chart
plt.figure(figsize=(10, 6))
plt.pie(production_by_group.values)
plt.title(f"Production Distribution by Group in {chosen_area} (2021)")
plt.legend(production_by_group.index, title="Production Groups", loc="best")
plt.show()
```

Production Distribution by Group in NO1 (2021)



```
In [ ]: january_data = area_data[area_data["starttime"].dt.month == 1]

# This creates a pivot table for better plotting
pivot_data = january_data.pivot_table(
    values="quantitykwh",
    index="starttime",
    columns="productiongroup",
    aggfunc="sum"
)

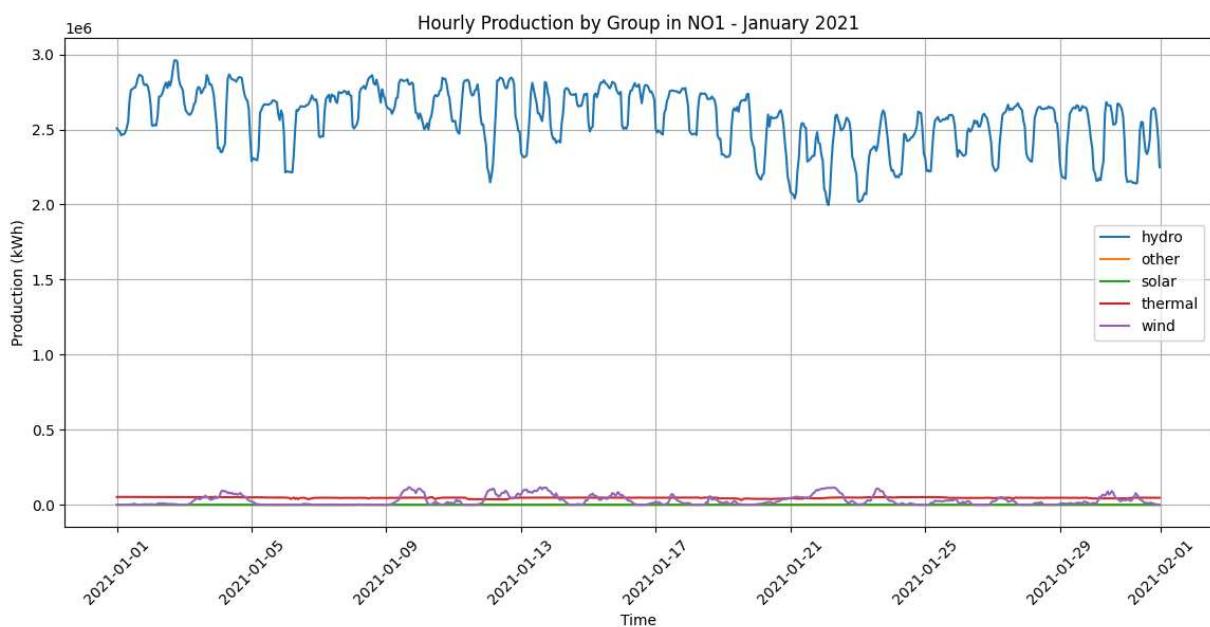
pivot_data.head()
```

```
Out[ ]:   productiongroup      hydro   other   solar   thermal   wind
           starttime
2021-01-01 00:00:00  2507716.8     0.0  6.106  51369.035  937.072
2021-01-01 01:00:00  2494728.0     0.0  4.030  51673.934  649.068
2021-01-01 02:00:00  2486777.5     0.0  3.982  51457.535  144.000
2021-01-01 03:00:00  2461176.0     0.0  8.146  51644.637  217.070
2021-01-01 04:00:00  2466969.2     0.0  8.616  51897.836  505.071
```

```
In [ ]: # Plotting Line for the first month of the year for a chosen price area

plt.figure(figsize=(14, 6))
for column in pivot_data.columns:
    plt.plot(pivot_data.index, pivot_data[column], label=column)

plt.xlabel("Time")
plt.ylabel("Production (kWh)")
plt.title(f"Hourly Production by Group in {chosen_area} - January 2021")
plt.legend()
plt.xticks(rotation=45)
plt.grid()
plt.show()
```



```
In [ ]: from pymongo import MongoClient
from pymongo.server_api import ServerApi
import tomllib

# Getting username and password for MongoDB connection
with open("../streamlit/secrets.toml", "rb") as f:
    cfg = tomllib.load(f)

USR = cfg["MongoDB"]["username"]
PWD = cfg["MongoDB"]["pwd"]

uri = f"mongodb+srv://esksko:{PWD}@ind320-esksko.5nbj7x0.mongodb.net/?retryWrites=true"

# Creating a new client and connecting to server
client = MongoClient(uri, server_api=ServerApi('1'))

# Selecting database and collection
db = client["IND320_assignment_2"] # Creates or connects to database
collection = db["production_data"] # Creates or connects to collection

# Converting spark DataFrame to pandas DataFrame, then to records for MongoDB inser
```

```
records = extracted_df.toPandas().to_dict("records")

# Clearing existing data
collection.delete_many({})

# Inserting new records
result = collection.insert_many(records)

print(f"Inserted {len(result.inserted_ids)} records into MongoDB.")

# Verifying insertion by checking count
count = collection.count_documents({})
print(f"Total documents in collection: {count}")

# Showing sample document
print("\nSample document:")
print(collection.find_one())
```

Inserted 215353 records into MongoDB.

Total documents in collection: 215353

Sample document:

```
{'_id': ObjectId('68efe05b7148e1cab990af9f'), 'pricearea': 'N04', 'productiongroup': 'solar', 'starttime': datetime.datetime(2021, 1, 1, 0, 0), 'quantitykwh': 0.0}
```