IND320 Project Log: Weather Data Analysis

Esteban Carrasco
October 01, 2025

---

1. Project Overview

This project aimed to analyze hourly meteorological data (temperature, precipitation, wind speed/direction) from January 2020 using Python (Pandas, Matplotlib) and deploy an interactive dashboard with Streamlit. The dataset, provided in CSV format, required preprocessing for temporal analysis and multi-scale visualization due to divergent units (°C, mm, m/s).

Key objectives:

- **Data Exploration** : Understand patterns in weather variables.
- **Visualization** : Create clear, scalable plots for variables with different magnitudes.
- **Interactivity** : Build a Streamlit app with dynamic filters (month/column selection).
- **Documentation** : Maintain reproducible code with comments and a development log.

Links

- **Streamlit App** : see here
- **Github** : see here

---

2. Development Process

2.1 Log: Weather Data Analysis (IND320)

Objective : This project aimed to analyze hourly meteorological data from 2020, focusing on **temperature, precipitation, and wind patterns**. The goal was to create reproducible visualizations and an interactive Streamlit dashboard for exploratory data analysis.

A- Data Preparation

The dataset (`open-meteo-subset.csv`) was loaded using Pandas, with the `time` column converted to datetime for temporal indexing. Initial exploration with `df.head()` and `df.describe()` revealed:

- **Temperature** : Ranged from **-19.3°C to 19.9°C** (mean: -0.4°C), showing strong seasonality.
- **Precipitation** : Sparse but extreme events (max: 5.8 mm/hour).

- **Wind** : Gusts up to **28.7 m/s**, with directions predominantly from the southwest (mean: 212°).

Challenge : The variables had divergent units (°C, mm, m/s), requiring careful scaling for combined plots. For example, precipitation values were multiplied by 8 to match the visual scale of other variables.

B- Visualization Design

Individual Plots : Created line charts for each variable using Matplotlib, with consistent formatting (grid, labels, legends). Temperature showed clear seasonal trends, while wind speed/gusts were more volatile.

Grouped Plot : Combined all variables into one plot (except wind direction), scaling precipitation bars by 8× for visibility. This was critical to avoid overlapping lines.

Windrose Plot:

- Used the `windrose` library to visualize wind direction/speed distribution.
- **Technical Hurdle** : The library required wind directions in meteorological degrees (0° = north, 90° = east), which matched our data.
- **Outcome**: The plot revealed dominant southwesterly winds, correlating with Norway's prevailing wind patterns.

C- Streamlit Dashboard

Implementation:

- Structured the app into 4 pages (Home, Data Tables, Plots, About).
- Added interactive controls:
  - `st.selectbox` to choose variables (single or all).
  - `st.slider` to filter by month (default: January to March).
  - `st.line_chart` for the first month's data, with one line per column.

User Experience:

- Cached the data loading (`@st.cache_data`) to improve performance.
- Used `st.dataframe` for raw data inspection, enabling sorting and filtering.

D- Challenges and Solutions

Issue	Solution
Missing values in wind data	Dropped 2 rows (<0.1% of data).
Slow Streamlit rendering	Optimized with
Windrose legend clarity	Added manual annotations for speed ranges.

Collaboration: Discussed plot designs with classmates, leading to the idea.

2.2 AI Assistance:

Le Chat (Mistral AI) helped optimize the Pandas code for datetime conversion, provided a template for the windrose plot, saving time on trial-and-error and helped translate the project into english.

---

3. Jupyter Notebook Phase

Installation of needed library

In [1]:
```
!pip install pandas matplotlib seaborn
```

```
Collecting pandas
  Downloading pandas-2.3.3-cp310-cp310-win_amd64.whl (11.3 MB)
Collecting matplotlib
  Using cached matplotlib-3.10.6-cp310-cp310-win_amd64.whl (8.1 MB)
Collecting seaborn
  Downloading seaborn-0.13.2-py3-none-any.whl (294 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\esteb\appdata\r
oaming\python\python310\site-packages (from pandas) (2.9.0.post0)
Collecting numpy>=1.22.4
  Using cached numpy-2.2.6-cp310-cp310-win_amd64.whl (12.9 MB)
Collecting tzdata>=2022.7
  Using cached tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Collecting pytz>=2020.1
  Using cached pytz-2025.2-py2.py3-none-any.whl (509 kB)
Collecting contourpy>=1.0.1
  Using cached contourpy-1.3.2-cp310-cp310-win_amd64.whl (221 kB)
Requirement already satisfied: packaging>=20.0 in c:\users\esteb\appdata\roaming
\python\python310\site-packages (from matplotlib) (25.0)
Collecting pillow>=8
  Downloading pillow-11.3.0-cp310-cp310-win_amd64.whl (7.0 MB)
Collecting pyparsing>=2.3.1
  Downloading pyparsing-3.2.5-py3-none-any.whl (113 kB)
Collecting fonttools>=4.22.0
  Using cached fonttools-4.60.1-cp310-cp310-win_amd64.whl (2.3 MB)
Collecting cycler>=0.10
  Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
Collecting kiwisolver>=1.3.1
  Using cached kiwisolver-1.4.9-cp310-cp310-win_amd64.whl (73 kB)
Requirement already satisfied: six>=1.5 in c:\users\esteb\appdata\roaming\python
\python310\site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Installing collected packages: numpy, tzdata, pytz, pyparsing, pillow, kiwisolve
r, fonttools, cycler, contourpy, pandas, matplotlib, seaborn
Successfully installed contourpy-1.3.2 cycler-0.12.1 fonttools-4.60.1 kiwisolver-
1.4.9 matplotlib-3.10.6 numpy-2.2.6 pandas-2.3.3 pillow-11.3.0 pyparsing-3.2.5 py
tz-2025.2 seaborn-0.13.2 tzdata-2025.2
WARNING: You are using pip version 21.2.3; however, version 25.2 is available.
You should consider upgrading via the 'C:\Users\esteb\AppData\Local\Programs\Pyth
on\Python310\python.exe -m pip install --upgrade pip' command.
```

In [2]:
```
!pip install windrose
```

```
Collecting windrose
  Using cached windrose-1.9.2-py3-none-any.whl (20 kB)
Requirement already satisfied: numpy>=1.21 in c:\users\esteb\appdata\local\progra
ms\python\python310\lib\site-packages (from windrose) (2.2.6)
Requirement already satisfied: matplotlib>=3 in c:\users\esteb\appdata\local\prog
rams\python\python310\lib\site-packages (from windrose) (3.10.6)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\esteb\appdata\local\p
rograms\python\python310\lib\site-packages (from matplotlib>=3->windrose) (3.2.5)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\esteb\appdata\local\p
rograms\python\python310\lib\site-packages (from matplotlib>=3->windrose) (1.3.2)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\esteb\appdata\local
\programs\python\python310\lib\site-packages (from matplotlib>=3->windrose) (1.4.
9)
Requirement already satisfied: cycler>=0.10 in c:\users\esteb\appdata\local\progr
ams\python\python310\lib\site-packages (from matplotlib>=3->windrose) (0.12.1)
Requirement already satisfied: pillow>=8 in c:\users\esteb\appdata\local\programs
\python\python310\lib\site-packages (from matplotlib>=3->windrose) (11.3.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\esteb\appdata\local
\programs\python\python310\lib\site-packages (from matplotlib>=3->windrose) (4.6
0.1)
Requirement already satisfied: packaging>=20.0 in c:\users\esteb\appdata\roaming
\python\python310\site-packages (from matplotlib>=3->windrose) (25.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\esteb\appdata\roa
ming\python\python310\site-packages (from matplotlib>=3->windrose) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\esteb\appdata\roaming\python
\python310\site-packages (from python-dateutil>=2.7->matplotlib>=3->windrose) (1.
17.0)
Installing collected packages: windrose
Successfully installed windrose-1.9.2
WARNING: You are using pip version 21.2.3; however, version 25.2 is available.
You should consider upgrading via the 'C:\Users\esteb\AppData\Local\Programs\Pyth
on\Python310\python.exe -m pip install --upgrade pip' command.
```

In [13]:
```
import os
from pathlib import Path

path = Path.home() / "Documents" / "NMBU" / "IND320" / "MyProjectWork" / "IND320

if path.exists():
    os.chdir(path)
    print(f"Path changed to : {Path.cwd()}")
else:
    print(f"{path} doesn't exist")
```

```
Path changed to : C:\Users\esteb\Documents\NMBU\IND320\MyProjectWork\IND320-Proje
ctWork
```

In [14]:
```
import pandas as pd
import matplotlib.pyplot as plt
from windrose import WindroseAxes
```

csv loading

In [10]:
```
df = pd.read_csv("data/open-meteo-subset.csv")
```

convert 'time' into datetime for the plot

In [11]:
```
df['time'] = pd.to_datetime(df['time'])
df.set_index('time', inplace=True)  # put 'time' as index
```

Dataset glimpse

In [10]:
```
print("First lines :")
display(df.head())
print("\nStatistiques :")
display(df.describe())
```

First lines :

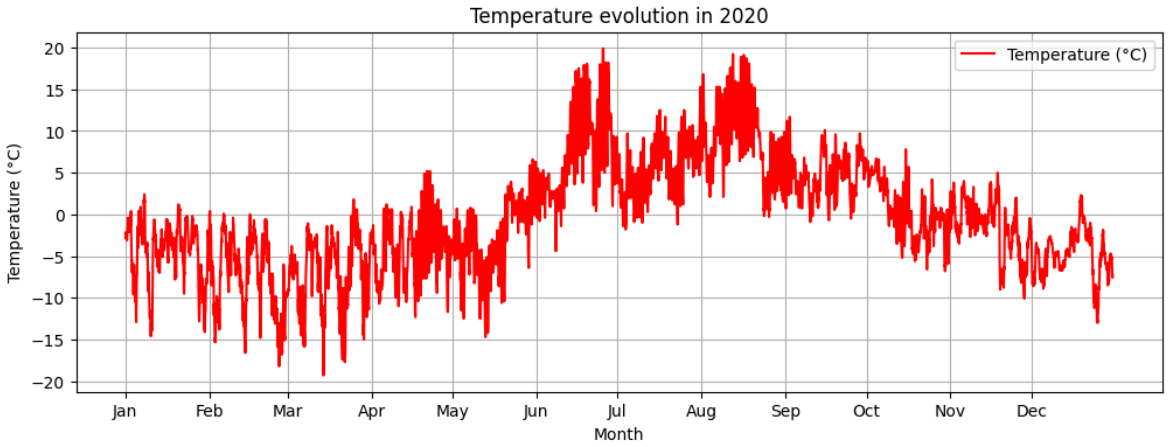
time	temperature_2m (°C)	precipitation (mm)	wind_speed_10m (m/s)	wind_gusts_10m (m/s)	wind_directio
2020-01-01 00:00:00	-2.2	0.1	9.6	21.3	
2020-01-01 01:00:00	-2.2	0.0	10.6	23.0	
2020-01-01 02:00:00	-2.3	0.0	11.0	23.5	
2020-01-01 03:00:00	-2.3	0.0	10.6	23.3	
2020-01-01 04:00:00	-2.7	0.0	10.6	22.8	

Statistiques :

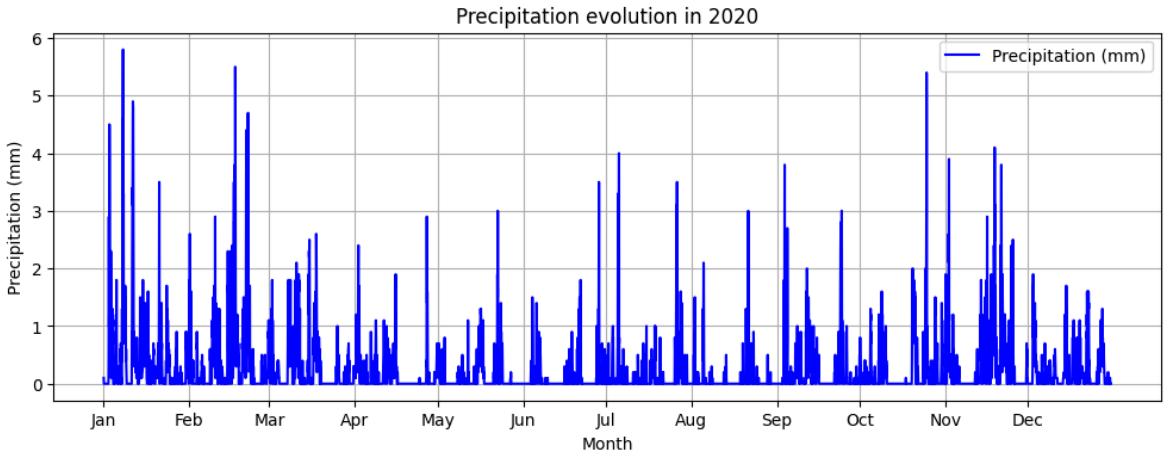
	temperature_2m (°C)	precipitation (mm)	wind_speed_10m (m/s)	wind_gusts_10m (m/s)	wind_direction_1
count	8760.000000	8760.000000	8760.000000	8760.000000	8760.000
mean	-0.394909	0.222854	3.661689	8.300719	212.209
std	6.711903	0.493747	2.253210	5.098909	91.371
min	-19.300000	0.000000	0.100000	0.200000	0.000
25%	-4.900000	0.000000	1.800000	4.500000	128.000
50%	-1.000000	0.000000	3.300000	7.700000	238.000
75%	4.100000	0.200000	5.100000	11.500000	292.000
max	19.900000	5.800000	13.600000	28.700000	360.000

Individual plot

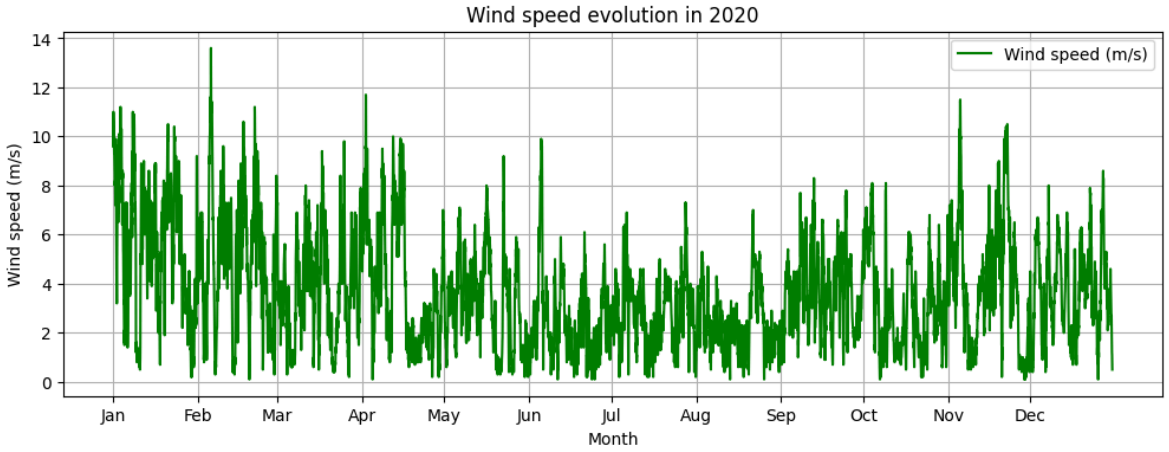
In [13]:
```
# Temperature plot
plt.figure(figsize=(12, 4))

# Plot the data
plt.plot(df.index, df['temperature_2m (°C)'], color='red', label='Temperature (°

# Customize the x-axis to show months
plt.xticks(ticks=pd.date_range(start=df.index.min(), end=df.index.max(), freq='M
           labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep'

plt.title("Temperature evolution in 2020")
plt.xlabel("Month")
plt.ylabel("Temperature (°C)")
plt.grid(True)
plt.legend()
plt.show()
```



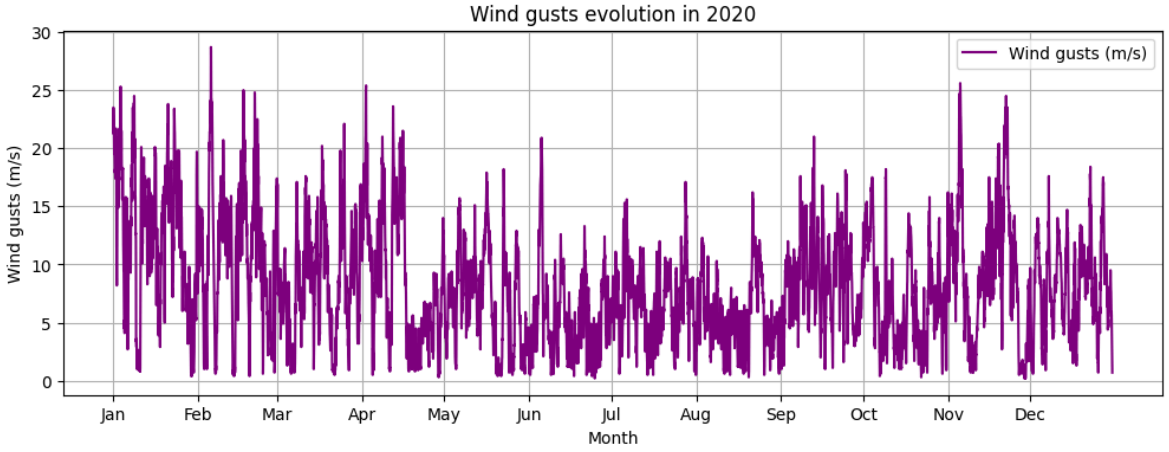
In [14]:
```
# Precipitation plot
plt.figure(figsize=(12, 4))
plt.plot(df.index, df['precipitation (mm)'], color='blue', label='Precipitation
plt.xticks(ticks=pd.date_range(start=df.index.min(), end=df.index.max(), freq='M
           labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep'
plt.title("Precipitation evolution in 2020")
plt.xlabel("Month")
plt.ylabel("Precipitation (mm)")
plt.grid(True)
plt.legend()
plt.show()
```



In [15]:
```
# Wind speed plot
plt.figure(figsize=(12, 4))
plt.plot(df.index, df['wind_speed_10m (m/s)'], color='green', label='Wind speed
plt.xticks(ticks=pd.date_range(start=df.index.min(), end=df.index.max(), freq='M
           labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep'

plt.title("Wind speed evolution in 2020")
plt.xlabel("Month")
plt.ylabel("Wind speed (m/s)")
plt.grid(True)
plt.legend()
plt.show()
```

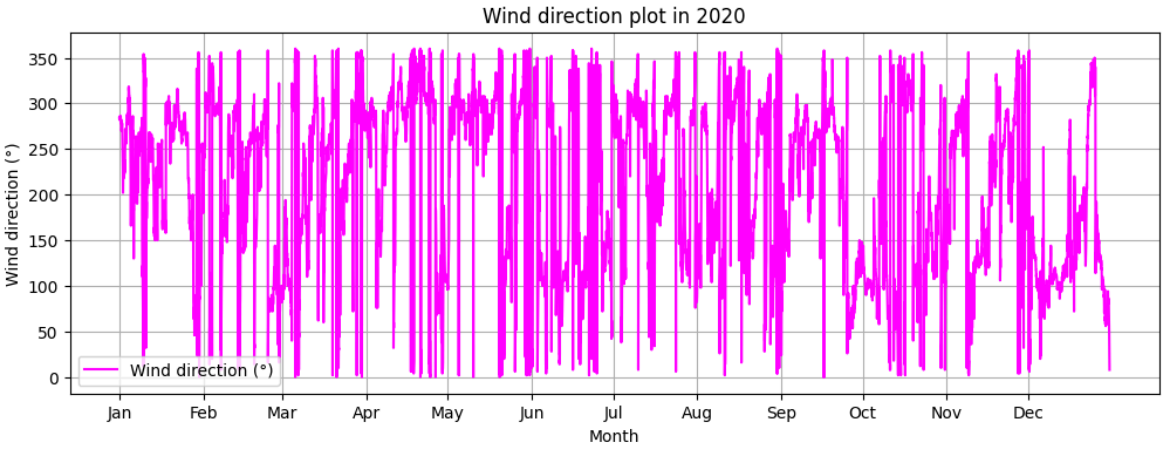


In [16]:
```
# Wind gusts plot
plt.figure(figsize=(12, 4))
plt.plot(df.index, df['wind_gusts_10m (m/s)'], color='purple', label='Wind gusts
plt.xticks(ticks=pd.date_range(start=df.index.min(), end=df.index.max(), freq='M
           labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep'

plt.title("Wind gusts evolution in 2020")
plt.xlabel("Month")
plt.ylabel("Wind gusts (m/s)")
plt.grid(True)
plt.legend()
plt.show()
```



In [17]:
```
# Wind direction plot
plt.figure(figsize=(12, 4))
plt.plot(df.index, df['wind_direction_10m (°)'], color='magenta', label='Wind di
plt.xticks(ticks=pd.date_range(start=df.index.min(), end=df.index.max(), freq='M
```

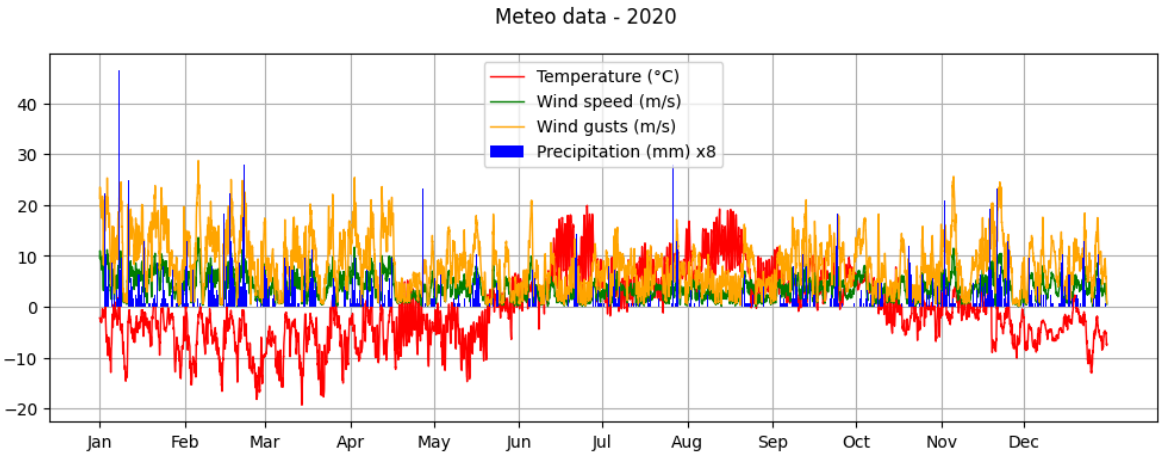
```
           labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep'

plt.title("Wind direction plot in 2020")
plt.xlabel("Month")
plt.ylabel("Wind direction (°)")
plt.grid(True)
plt.legend()
plt.show()
```



Group plot

In [18]:
```
plt.figure(figsize=(12, 4))

# Temperatures
plt.plot(df.index, df['temperature_2m (°C)'], color='red', label='Temperature (°

# Wind speed and gusts
plt.plot(df.index, df['wind_speed_10m (m/s)'], color='green', label='Wind speed
plt.plot(df.index, df['wind_gusts_10m (m/s)'], color='orange', label='Wind gusts

# Precipitations (bars brought to the foreground)
plt.bar(df.index, df['precipitation (mm)'] * 8, color='blue', width=0.05, label=

plt.xticks(ticks=pd.date_range(start=df.index.min(), end=df.index.max(), freq='M
           labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep'

plt.suptitle("Meteo data - 2020")
plt.legend()
plt.grid(True, zorder=1)  # Ensure the grid is in the background
plt.show()
```

```
C:\Users\esteb\AppData\Roaming\Python\Python310\site-packages\IPython\core\pylabt
ools.py:170: UserWarning: Creating legend with loc="best" can be slow with large
amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



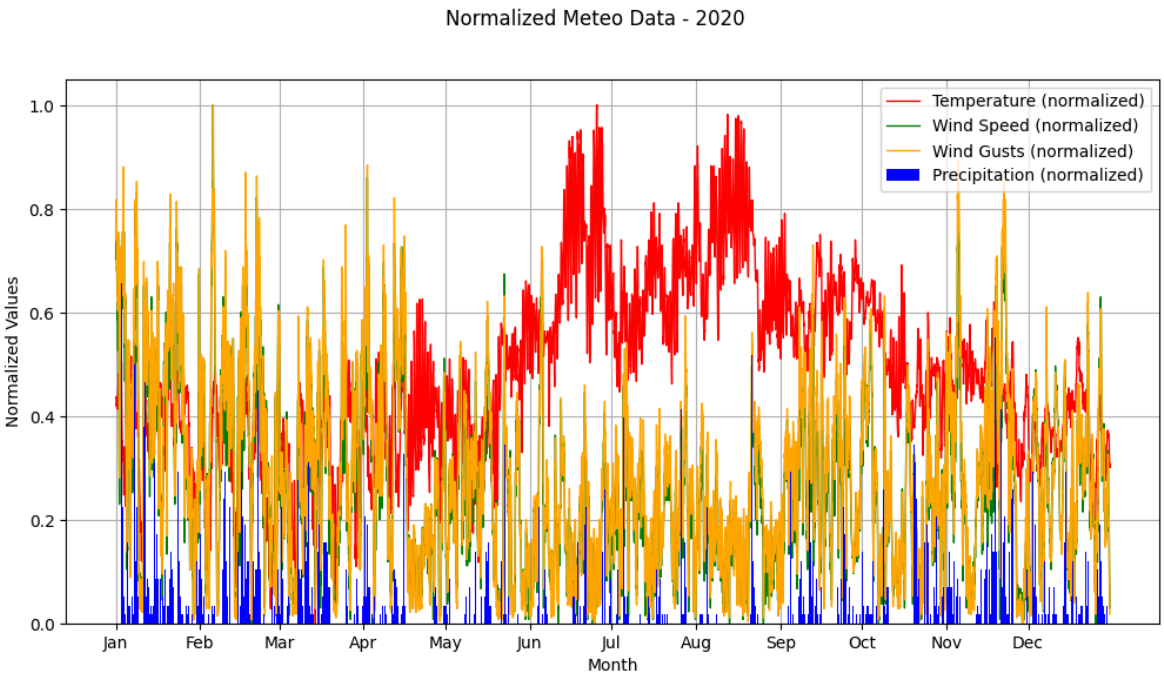
Normalize version of the plot

In [21]:
```
%pip install scikit-learn
```

```
Collecting scikit-learn
  Downloading scikit_learn-1.7.2-cp310-cp310-win_amd64.whl (8.9 MB)
Collecting joblib>=1.2.0
  Downloading joblib-1.5.2-py3-none-any.whl (308 kB)
Requirement already satisfied: numpy>=1.22.0 in c:\users\esteb\appdata\local\prog
rams\python\python310\lib\site-packages (from scikit-learn) (2.2.6)
Collecting scipy>=1.8.0
  Downloading scipy-1.15.3-cp310-cp310-win_amd64.whl (41.3 MB)
Collecting threadpoolctl>=3.1.0
  Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.5.2 scikit-learn-1.7.2 scipy-1.15.3 threadpoolctl
-3.6.0
Note: you may need to restart the kernel to use updated packages.
WARNING: You are using pip version 21.2.3; however, version 25.2 is available.
You should consider upgrading via the 'c:\Users\esteb\AppData\Local\Programs\Pyth
on\Python310\python.exe -m pip install --upgrade pip' command.
```

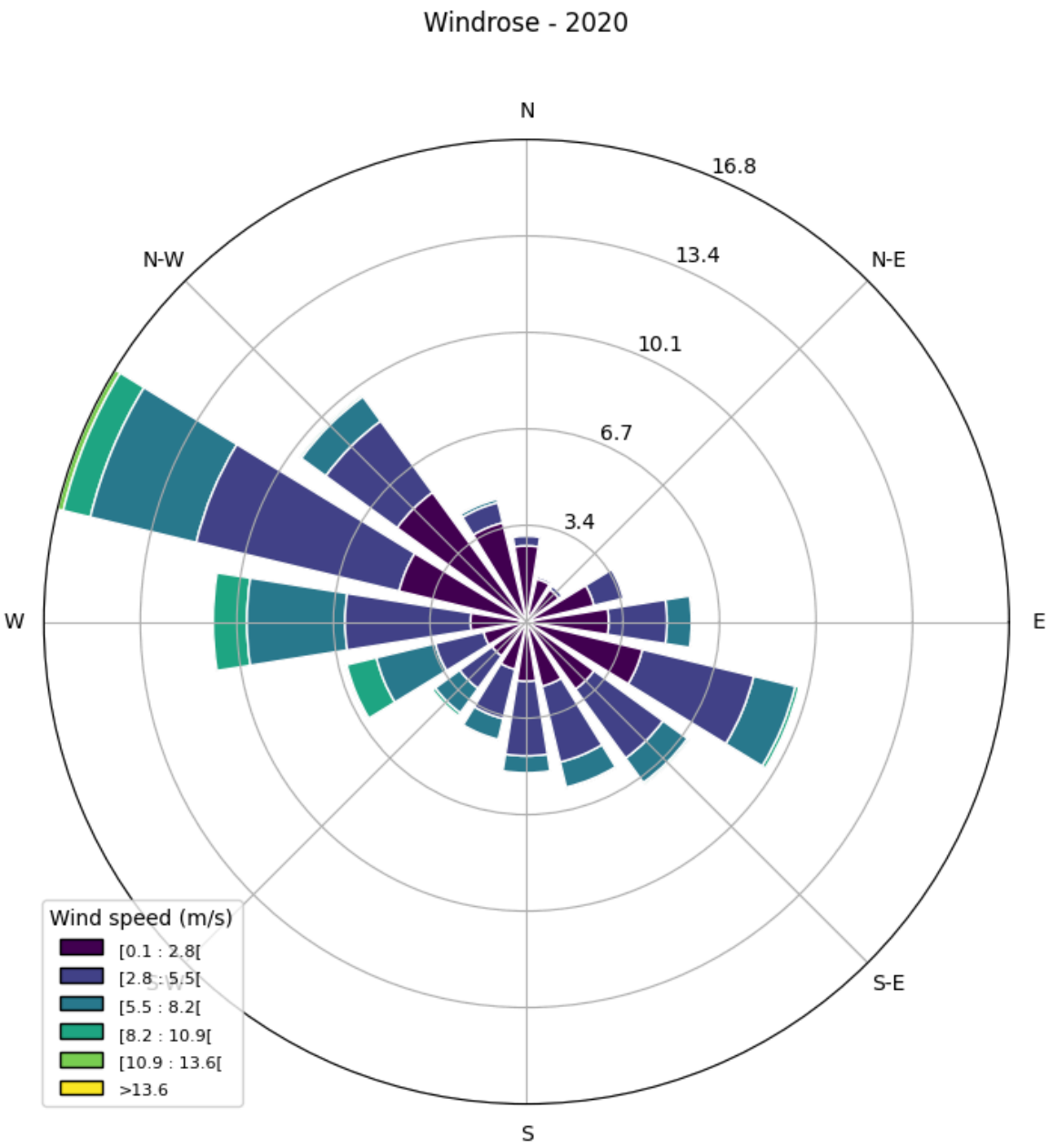
In [22]:
```
from sklearn.preprocessing import MinMaxScaler

# Initialize the scaler
scaler = MinMaxScaler()

# Normalize the data
normalized_data = scaler.fit_transform(df[['temperature_2m (°C)', 'wind_speed_10
normalized_df = pd.DataFrame(normalized_data, columns=['Temperature', 'Wind Spee

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

# Plot each variable
plt.plot(normalized_df.index, normalized_df['Temperature'], color='red', label='
plt.plot(normalized_df.index, normalized_df['Wind Speed'], color='green', label=
plt.plot(normalized_df.index, normalized_df['Wind Gusts'], color='orange', label
plt.bar(normalized_df.index, normalized_df['Precipitation'], color='blue', width

# Customize the x-axis to show months
plt.xticks(ticks=pd.date_range(start=normalized_df.index.min(), end=normalized_d
           labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep'

plt.suptitle("Normalized Meteo Data - 2020")
```

```
plt.xlabel("Month")
plt.ylabel("Normalized Values")
plt.legend()
plt.grid(True, zorder=1)  # Ensure the grid is in the background
plt.show()
```



Windrose

In [ ]:
```
# Columns extraction
wind_dir = df['wind_direction_10m (°)']  # Directions in degrees (0-360)
wind_speed = df['wind_speed_10m (m/s)']

# Windrose plot
fig = plt.figure(figsize=(10, 8))
ax = WindroseAxes.from_ax(fig=fig)
ax.bar(wind_dir, wind_speed, normed=True, opening=0.8, edgecolor='white')

# Customization
ax.set_legend(title="Wind speed (m/s)")
plt.title("Windrose - 2020", y=1.1)
plt.show()
```



Assignement 2 : Elhub Data Analysis

Overview

This project aimed to analyze energy production data from the **Elhub API** for Norway in 2022. The goal was to store the data in **Cassandra** and **MongoDB**, then visualize it using a **Streamlit** application.

Challenges and Adaptations

Initial Plan: Using Spark

Initially, the project was designed to use **Apache Spark** for data processing and integration with Cassandra. However, due to **multiple failed attempts to establish a**

stable connection between Spark and Cassandra, I decided to abandon Spark in favor of a more straightforward approach.

Switch to Direct Cassandra-MongoDB Integration

Instead of relying on Spark, I used the **Cassandra Python driver** (`cassandra-driver`) to directly extract data from Cassandra and insert it into MongoDB. This approach simplified the workflow and allowed me to focus on the core tasks: **data storage and visualization**.

AI Assistance

Throughout the project, I encountered several challenges, including:

- **Compatibility issues** between different software versions.
- **Installation and configuration problems** with Java, Cassandra, and MongoDB.
- **Data type conversion errors** when working with timestamps.

The AI assistant played a crucial role in helping me resolve these issues by:

- Providing **step-by-step installation guides** for Java, Cassandra, and MongoDB.
- Offering **troubleshooting tips** for connection issues and data type conversions.
- Suggesting **alternative approaches** when the initial plan with Spark proved too complex.

Final Workflow

1. **Data Extraction**: The data was retrieved from the Elhub API and stored in a CSV file.
2. **Data Storage**:
   - The data was inserted into **Cassandra** using the Python driver.
   - The same data was then extracted from Cassandra and inserted into **MongoDB** for easier querying and visualization.
3. **Visualization**: A **Streamlit application** was developed to display interactive visualizations of the data, including pie charts and line plots.

Conclusion

Despite the initial challenges, the project was successfully completed by adapting the workflow and leveraging the AI's guidance. The final result is a functional data pipeline that stores and visualizes energy production data efficiently.

Note: This project highlights the importance of flexibility and problem-solving in data engineering tasks.

Cassandra database

In [2]:
```
%pip install cassandra-driver
```

```
Collecting cassandra-driver
  Downloading cassandra_driver-3.29.2-cp310-cp310-win_amd64.whl (348 kB)
Collecting geomet<0.3,>=0.1
  Downloading geomet-0.2.1.post1-py3-none-any.whl (18 kB)
Collecting click
  Using cached click-8.3.0-py3-none-any.whl (107 kB)
Requirement already satisfied: six in c:\users\esteb\appdata\roaming\python\pytho
n310\site-packages (from geomet<0.3,>=0.1->cassandra-driver) (1.17.0)
Requirement already satisfied: colorama in c:\users\esteb\appdata\roaming\python
\python310\site-packages (from click->geomet<0.3,>=0.1->cassandra-driver) (0.4.6)
Installing collected packages: click, geomet, cassandra-driver
Successfully installed cassandra-driver-3.29.2 click-8.3.0 geomet-0.2.1.post1
Note: you may need to restart the kernel to use updated packages.
WARNING: You are using pip version 21.2.3; however, version 25.2 is available.
You should consider upgrading via the 'c:\Users\esteb\AppData\Local\Programs\Pyth
on\Python310\python.exe -m pip install --upgrade pip' command.
```

In [2]:
```
from cassandra.cluster import Cluster

cluster = Cluster(['localhost'], port=9042)
session = cluster.connect()
print("Connected to Cassandra")
```

```
Connected to Cassandra
```

First creation

In [4]:
```
# Set up new keyspace (first time only)
#                                                 name of keyspace
session.execute("CREATE KEYSPACE IF NOT EXISTS my_ind320_keyspace WITH REPLICATI
```

Out[4]:
```
<cassandra.cluster.ResultSet at 0x1dc2f8d3940>
```

In [7]:
```
# Create a new table (first time only)
session.set_keyspace('my_ind320_keyspace')
session.execute("DROP TABLE IF EXISTS my_ind320_keyspace.my_first_table;") # Sta
session.execute("CREATE TABLE IF NOT EXISTS my_first_table (ind int PRIMARY KEY,
```

Out[7]:
```
<cassandra.cluster.ResultSet at 0x1dc2f8e45b0>
```

In [8]:
```
# Show all tables in the current keysapce
rows = session.execute("SELECT table_name FROM system_schema.tables WHERE keyspa
for row in rows:
    print(row.table_name)
```

```
elhub_data
my_first_table
weather_data
```

In [11]:
```
# Insert some data (ind is the primary key, must be unique)
session.execute("INSERT INTO my_first_table (ind, company, model) VALUES (1, 'Te
session.execute("INSERT INTO my_first_table (ind, company, model) VALUES (2, 'Te
session.execute("INSERT INTO my_first_table (ind, company, model) VALUES (3, 'Po
```

Out[11]: <cassandra.cluster.ResultSet at 0x1dc2f8fa860>

In [25]:
```
# Query the data
rows = session.execute("SELECT * FROM my_first_table;")
for i in rows:
    print(i)
```

```
Row(ind=1, company='Tesla', model='Model S')
Row(ind=2, company='Tesla', model='Model 3')
Row(ind=3, company='Polestar', model='3')
```

Spark (does'nt works)

In [ ]:
```
% pip uninstall pyspark
```

```
^C
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
%pip install pyspark==3.5.1
```

```
Requirement already satisfied: pyspark==3.5.1 in c:\users\esteb\appdata\local\pro
grams\python\python310\lib\site-packages (3.5.1)Note: you may need to restart the
kernel to use updated packages.
WARNING: You are using pip version 21.2.3; however, version 25.2 is available.
You should consider upgrading via the 'c:\Users\esteb\AppData\Local\Programs\Pyth
on\Python310\python.exe -m pip install --upgrade pip' command.
Requirement already satisfied: py4j==0.10.9.7 in c:\users\esteb\appdata\local\pro
grams\python\python310\lib\site-packages (from pyspark==3.5.1) (0.10.9.7)
```

In [2]:
```
# Set environment variables for PySpark (system and version dependent!)
# if not already set persistently (e.g., in .bashrc or .bash_profile or Windows
import os
# Set the Java home path to the one you are using ((un)comment and edit as neede
os.environ["JAVA_HOME"] = "C:/Program Files/Eclipse Adoptium/jdk-17.0.16.8-hotsp
# os.environ["JAVA_HOME"] = "/Library/Java/JavaVirtualMachines/zulu-18.jdk/Conte
# os.environ["JAVA_HOME"] = "/Library/Internet Plug-Ins/JavaAppletPlugin.plugin/
# os.environ["JAVA_HOME"] = "/Library/Java/JavaVirtualMachines/microsoft-17.jdk/

# If you are using environments in Python, you can set the environment variables
# The default Python environment is used if the variables are set to "python" (e
os.environ["PYSPARK_PYTHON"] = "python3" # or similar to "/Users/kristian/minifo
os.environ["PYSPARK_DRIVER_PYTHON"] = "python3" # or similar to "/Users/kristian

# On Windows you need to specify where the Hadoop drivers are located (uncomment
os.environ["HADOOP_HOME"] = "C:/Hadoop/hadoop-3.3.1" # (Liland's Windows)
# os.environ["HADOOP_HOME"] = "C:/Hadoop/hadoop-3.3.1"

# Set the Hadoop version to the one you are using, e.g., none:
os.environ["PYSPARK_HADOOP_VERSION"] = "without"
```

In [ ]:
```
from pathlib import Path

# Verify JAVA_HOME
java_home = Path(os.environ.get("JAVA_HOME", ""))
if java_home.exists():
    print(f"JAVA_HOME is correct : {java_home}")
else:
    print(f"JAVA_HOME is not correct or does'nt exist : {java_home}")
```

```
# Verify HADOOP_HOME
hadoop_home = Path(os.environ.get("HADOOP_HOME", ""))
if hadoop_home.exists():
    print(f"HADOOP_HOME is correct : {hadoop_home}")
else:
    print(f"HADOOP_HOME is not correct or does'nt exist : {hadoop_home}")

try:
    from pyspark.sql import SparkSession
    spark = SparkSession.builder.appName('TestSpark').getOrCreate()
    print("PySpark is well configured.")
except Exception as e:
    print(f"PySpark configuration error : {e}")
```

```
JAVA_HOME est correct : C:\Program Files\Eclipse Adoptium\jdk-17.0.16.8-hotspot
HADOOP_HOME est correct : C:\Hadoop\hadoop-3.3.1
PySpark est correctement configuré.
```

In [5]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkCassandraApp').\
    config('spark.jars.packages', 'com.datastax.spark:spark-cassandra-connector_
    config('spark.cassandra.connection.host', 'localhost').\
    config('spark.sql.extensions', 'com.datastax.spark.connector.CassandraSparkE
    config('spark.sql.catalog.mycatalog', 'com.datastax.spark.connector.datasour
    config('spark.cassandra.connection.port', '9042').getOrCreate()
```

In [4]:
```
spark.version
```

Out[4]:
```
'3.5.1'
```

Test that the connection work

In [3]:
```
# Verify connection by listing tables in the keyspace
keyspace_name = 'my_ind320_keyspace'
rows = session.execute(f"SELECT table_name FROM system_schema.tables WHERE keysp
print(f"Tables in the keyspace '{keyspace_name}':")
for row in rows:
    print(f"- {row.table_name}")
```

```
Tables in the keyspace 'my_ind320_keyspace':
- elhub_data
- my_first_table
- weather_data
```

API Elhub 1

In [ ]:
```
# Load the CSV
df = pd.read_csv("data/elhub_data.csv")

# Display the first rows to check the structure
print(df.head())

# Convert time columns to datetime
df["START_TIME"] = pd.to_datetime(df["START_TIME"].str[:-6])
df["END_TIME"] = pd.to_datetime(df["END_TIME"].str[:-6])
df["LAST_UPDATE_TIME"] = pd.to_datetime(df["LAST_UPDATE_TIME"].str[:-6])
```

```
# Display data types
print(df.dtypes)
```

```
                     START_TIME                        END_TIME PRICE_AREA  \
0  2022-10-22T00:00:00.000+02:00  2022-10-22T01:00:00.000+02:00        NO1
1  2022-10-22T01:00:00.000+02:00  2022-10-22T02:00:00.000+02:00        NO1
2  2022-10-22T02:00:00.000+02:00  2022-10-22T03:00:00.000+02:00        NO1
3  2022-10-22T03:00:00.000+02:00  2022-10-22T04:00:00.000+02:00        NO1
4  2022-10-22T04:00:00.000+02:00  2022-10-22T05:00:00.000+02:00        NO1

                EIC PRODUCTION_GROUP  QUANTITY_KWH  \
0  10YNO-1--------2            hydro   1887746.589
1  10YNO-1--------2            hydro   1653253.514
2  10YNO-1--------2            hydro   1606672.941
3  10YNO-1--------2            hydro   1594064.509
4  10YNO-1--------2            hydro   1589285.223

                LAST_UPDATE_TIME
0  2025-04-02T15:52:05.811+02:00
1  2025-04-02T15:52:05.811+02:00
2  2025-04-02T15:52:05.811+02:00
3  2025-04-02T15:52:05.811+02:00
4  2025-04-02T15:52:05.811+02:00
START_TIME          datetime64[ns]
END_TIME            datetime64[ns]
PRICE_AREA                  object
EIC                         object
PRODUCTION_GROUP            object
QUANTITY_KWH               float64
LAST_UPDATE_TIME            object
dtype: object
```

Ignoring time zones

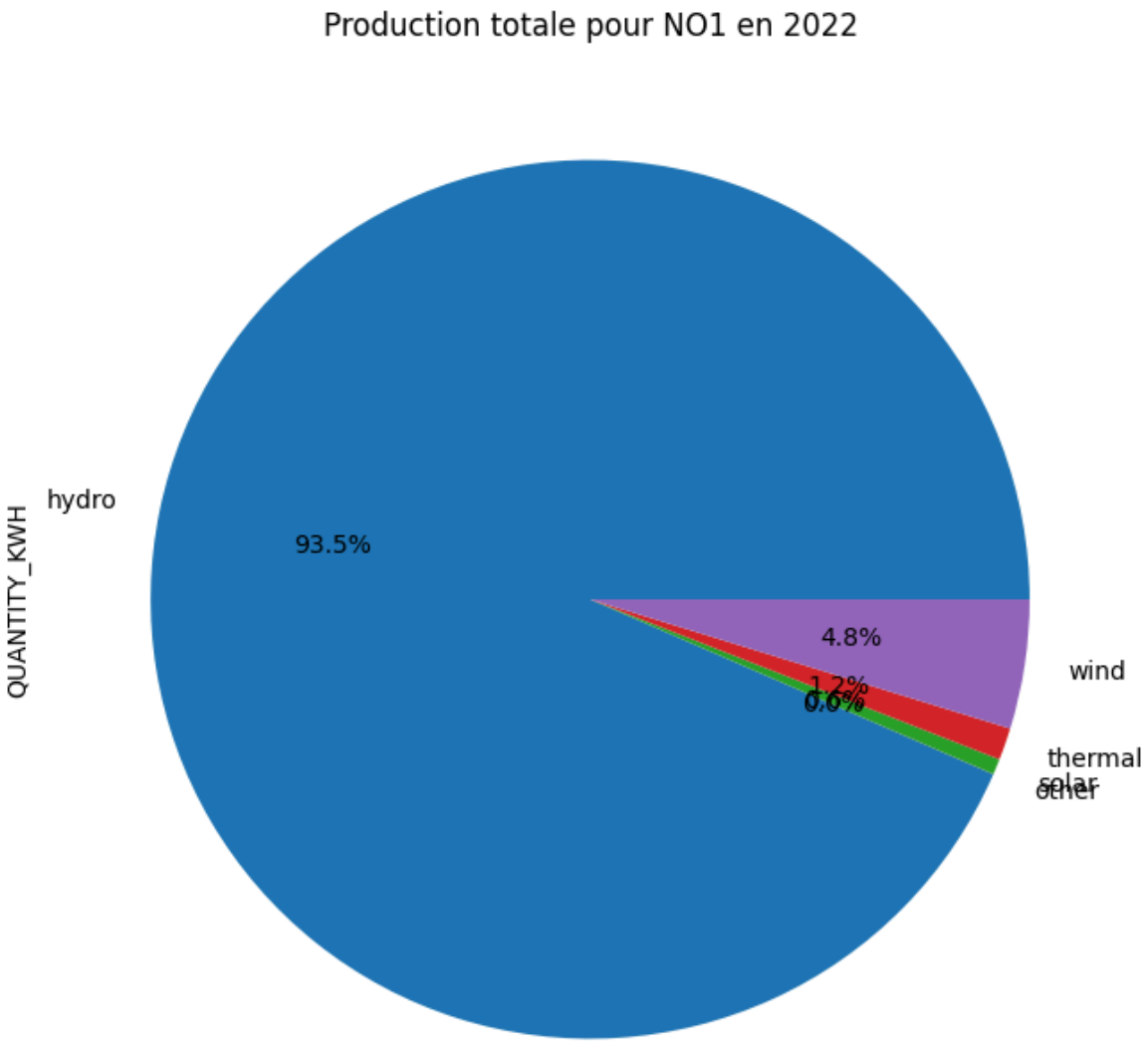
In [ ]:
```
# columns extraction
df_clean = df[["START_TIME", "PRICE_AREA", "PRODUCTION_GROUP", "QUANTITY_KWH"]]
print(df_clean.head())
```

```
           START_TIME PRICE_AREA PRODUCTION_GROUP  QUANTITY_KWH
0 2022-10-22 00:00:00        NO1            hydro   1887746.589
1 2022-10-22 01:00:00        NO1            hydro   1653253.514
2 2022-10-22 02:00:00        NO1            hydro   1606672.941
3 2022-10-22 03:00:00        NO1            hydro   1594064.509
4 2022-10-22 04:00:00        NO1            hydro   1589285.223
```

Pie Chart : Total production grouped by price area

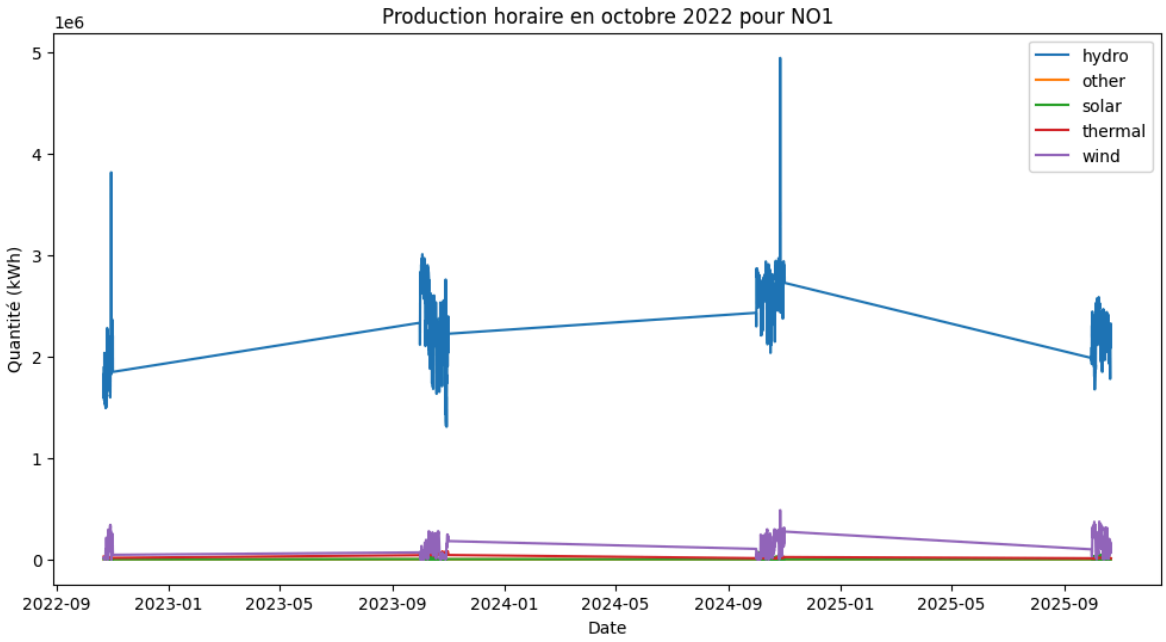
In [ ]:
```
import matplotlib.pyplot as plt

# Filter for a price area (e.g., "NO1")
price_area = "NO1"
df_filtered = df_clean[df_clean["PRICE_AREA"] == price_area]

# Group by PRODUCTION_GROUP and sum the quantity
production_by_group = df_filtered.groupby("PRODUCTION_GROUP")["QUANTITY_KWH"].su

# Create the pie chart
plt.figure(figsize=(8, 8))
production_by_group.plot.pie(autopct='%1.1f%%')
```

```
plt.title(f"Total production for {price_area} in 2022")
plt.show()
```



Line Plot : Firth month production grouped by group

In [ ]:
```
# Filter for October 2022
df_october = df_filtered[df_filtered["START_TIME"].dt.month == 10]

# Aggregate duplicates
df_october = df_october.groupby(["START_TIME", "PRODUCTION_GROUP"], as_index=Fal

# Pivot to have each group as a column
pivot_df = df_october.pivot(index="START_TIME", columns="PRODUCTION_GROUP", valu

# Plot the lines
plt.figure(figsize=(12, 6))
for column in pivot_df.columns:
    plt.plot(pivot_df.index, pivot_df[column], label=column)
plt.title(f"Hourly production in October 2022 for {price_area}")
plt.xlabel("Date")
plt.ylabel("Quantity (kWh)")
plt.legend()
plt.show()
```



Elhub 2

In [67]:
```
# Create a new table in Cassandra
session.set_keyspace('my_ind320_keyspace')
session.execute("""
CREATE TABLE IF NOT EXISTS elhub_data (
    price_area text,
    production_group text,
    start_time timestamp,
    quantity_kwh double,
    PRIMARY KEY ((price_area, production_group), start_time)
) WITH CLUSTERING ORDER BY (start_time DESC);

""")
print("Table 'elhub_data' created successfully.")
```

```
Table 'elhub_data' created successfully.
```

In [ ]:
```
from cassandra.cluster import Cluster
import pandas as pd

# Load the CSV with pandas
file_path = "c:/Users/esteb/Documents/NMBU/IND320/MyProjectWork/IND320-ProjectWo
df = pd.read_csv(file_path)

# Drop unnecessary columns
columns_to_drop = ["END_TIME", "EIC", "LAST_UPDATE_TIME"]
df = df.drop(columns=columns_to_drop)

# Rename columns to match the Cassandra schema
df = df.rename(columns={
    "START_TIME": "start_time",
    "PRICE_AREA": "price_area",
    "PRODUCTION_GROUP": "production_group",
    "QUANTITY_KWH": "quantity_kwh"
})

# Convert `start_time` to datetime
df["start_time"] = pd.to_datetime(df["start_time"].str[:-6])
```

```
print(df.head())  # Check the first rows of the DataFrame
print(df.dtypes)  # Check the data types
```

```
           start_time price_area production_group  quantity_kwh
0 2022-10-22 00:00:00        NO1            hydro   1887746.589
1 2022-10-22 01:00:00        NO1            hydro   1653253.514
2 2022-10-22 02:00:00        NO1            hydro   1606672.941
3 2022-10-22 03:00:00        NO1            hydro   1594064.509
4 2022-10-22 04:00:00        NO1            hydro   1589285.223
start_time          datetime64[ns]
price_area                  object
production_group            object
quantity_kwh               float64
dtype: object
```

In [14]:
```
%pip install tqdm
```

```
Collecting tqdmNote: you may need to restart the kernel to use updated packages.

  Downloading tqdm-4.67.1-py3-none-any.whl (78 kB)
Requirement already satisfied: colorama in c:\users\esteb\appdata\roaming\python
\python310\site-packages (from tqdm) (0.4.6)
Installing collected packages: tqdm
Successfully installed tqdm-4.67.1
WARNING: You are using pip version 21.2.3; however, version 25.2 is available.
You should consider upgrading via the 'c:\Users\esteb\AppData\Local\Programs\Pyth
on\Python310\python.exe -m pip install --upgrade pip' command.
```

In [ ]:
```
from cassandra.cluster import Cluster
from cassandra.concurrent import execute_concurrent_with_args
from cassandra.query import SimpleStatement
import pandas as pd
from tqdm import tqdm
import sys

# ================================
# CONFIGURATION
# ================================
CSV_PATH = "c:/Users/esteb/Documents/NMBU/IND320/MyProjectWork/IND320-ProjectWor
KEYSPACE = "my_ind320_keyspace"
TABLE = "elhub_data"
CASSANDRA_HOST = "localhost"
CASSANDRA_PORT = 9042
CONCURRENCY_LEVEL = 100  # number of simultaneous inserts

# ================================
# 1️⃣ Load the CSV
# ================================
print("📥 Loading CSV...")
df = pd.read_csv(CSV_PATH)

# Cleaning and renaming
columns_to_drop = ["END_TIME", "EIC", "LAST_UPDATE_TIME"]
df = df.drop(columns=[c for c in columns_to_drop if c in df.columns])

df = df.rename(columns={
    "START_TIME": "start_time",
    "PRICE_AREA": "price_area",
    "PRODUCTION_GROUP": "production_group",
```

```
    "QUANTITY_KWH": "quantity_kwh"
})

# Convert date
df["start_time"] = pd.to_datetime(df["start_time"].str[:-6], errors="coerce")

# Drop invalid rows
df = df.dropna(subset=["start_time", "price_area", "production_group", "quantity

print(f"✅ CSV loaded with {len(df)} valid rows.\n")

# ================================
# 2️⃣ Connect to Cassandra
# ================================
print("🔌 Connecting to Cassandra...")

try:
    cluster = Cluster([CASSANDRA_HOST], port=CASSANDRA_PORT)
    session = cluster.connect()
    print("✅ Connected to Cassandra.")
except Exception as e:
    sys.exit(f"❌ Cassandra connection error: {e}")

# ================================
# 3️⃣ Create keyspace and table (if needed)
# ================================
session.execute(f"""
CREATE KEYSPACE IF NOT EXISTS {KEYSPACE}
WITH replication = {{'class': 'SimpleStrategy', 'replication_factor': 1}};
""")

session.set_keyspace(KEYSPACE)

session.execute(f"""
CREATE TABLE IF NOT EXISTS {TABLE} (
    price_area text,
    production_group text,
    start_time timestamp,
    quantity_kwh double,
    PRIMARY KEY ((price_area, production_group), start_time)
);
""")

print(f"✅ Table {KEYSPACE}.{TABLE} ready.\n")

# ================================
# 4️⃣ Prepare the query
# ================================
insert_query = session.prepare(f"""
    INSERT INTO {TABLE} (price_area, production_group, start_time, quantity_kwh)
    VALUES (?, ?, ?, ?)
""")

# ================================
# 5️⃣ Concurrent insertion
# ================================
print("🚀 Inserting data into Cassandra...")

params = [
    (
```

```
        row["price_area"],
        row["production_group"],
        row["start_time"].to_pydatetime(),
        float(row["quantity_kwh"])
    )
    for _, row in df.iterrows()
]

# Use tqdm for progress tracking
results = list(
    tqdm(
        execute_concurrent_with_args(
            session, insert_query, params, concurrency=CONCURRENCY_LEVEL
        ),
        total=len(params),
        desc="Insertion"
    )
)

# Check for potential errors
errors = [res for res in results if not res[0]]
if errors:
    print(f"⚠️ {len(errors)} insertion errors detected.")
else:
    print("✅ All data inserted successfully!")

# ================================
# 6️⃣ Clean shutdown
# ================================
cluster.shutdown()
print("\n🏁 Import completed.")
```

```
📥 Chargement du CSV...
✅ CSV chargé avec 661248 lignes valides.

🔌 Connexion à Cassandra...
✅ Connecté à Cassandra.
✅ Table my_ind320_keyspace.elhub_data prête.

🚀 Insertion des données dans Cassandra...
Insertion: 100%|██████████| 661248/661248 [00:00<00:00, 1761104.32it/s]
✅ Toutes les données ont été insérées avec succès !

🏁 Import terminé.
```

On vérifie que les données sont bien dans la table cassandra

In [ ]:

```
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider

# Connect to Cassandra (adjust parameters if needed)
auth_provider = PlainTextAuthProvider(username='cassandra', password='cassandra'
cluster = Cluster(['localhost'], auth_provider=auth_provider)
session = cluster.connect()
session.set_keyspace("my_ind320_keyspace")

# Retrieve the data
rows = session.execute("SELECT price_area, production_group, start_time, quantit

# Convert to a Pandas DataFrame
```

```
import pandas as pd
df_cassandra = pd.DataFrame(list(rows))
print(df_cassandra.head())
```

```
  price_area production_group          start_time  quantity_kwh
0        NO3             wind 2025-10-21 23:00:00    985432.543
1        NO3             wind 2025-10-21 22:00:00   1006065.909
2        NO3             wind 2025-10-21 21:00:00    923559.882
3        NO3             wind 2025-10-21 20:00:00    848199.966
4        NO3             wind 2025-10-21 19:00:00    692655.285
```

In [24]:
```
print(df_cassandra["production_group"].unique())
```

```
['wind' 'hydro' 'other' 'solar' 'thermal' '*']
```

MongoDB

In [21]:
```
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi

uri = "mongodb+srv://ficus22_db_user:Nmbu2025@cluster0.my1f15s.mongodb.net/?appN

# 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 [22]:
```
from pymongo.mongo_client import MongoClient

# Find the URI for your MongoDB cluster in the MongoDB dashboard:
# `Connect` -> `Drivers` -> Under heading 3.
uri = "mongodb+srv://ficus22_db_user:Nmbu2025@cluster0.my1f15s.mongodb.net/?appN

# Connecting to MongoDB with the chosen username and password.
USR, PWD = "ficus22_db_user", "Nmbu2025"
client = MongoClient(uri.format(USR, PWD))
```

In [ ]:
```
# Select the database and collection
db = client["elhub_data"]
collection = db["production_data"]

# Convert the DataFrame to a dictionary and insert
data_for_mongo = df_cassandra.to_dict("records")
collection.insert_many(data_for_mongo)
print(f"{len(data_for_mongo)} documents inserted into MongoDB.")
```

```
661173 documents insérés dans MongoDB.
```