Links

- Github repository:https://github.com/Gianna-liu/Weathering_with_you_GegeLiu
- streamlit App: https://weatheringwithyou-gegeliu.streamlit.app/

For this CA3, I added two pages, `production-data quality` and `weather-data quality`

The structure of Github repository:

- scripts/: This folder contains the Jupyter Notebook, PDF, and HTML files used for data analysis and documentation.
- streamlitApp/: This folder contains all the files required to develop and run the Streamlit application.
- requirements.txt: This file lists all the Python package dependencies required to run the project, allowing easy installation via pip install -r requirements.txt.

Development Log

For this assignment, I performed implemented methods to detect noise, outliers, anomalies which are extremly useful when working with raw or unprocessed data.

Jupyter Notebook

- In the notebook part, I first created a dataframe containing geographical information for five cities and easily retrieved meteo data with the provided python code from the open-meteo api. For the main analytical tasks, I applied high-pass filtering with DCT to remove long-term seasonal effects and detected outliers using robust MAD-based SPC boundaries on this time-series dataset. I spent some time tuning the cutoff frequency of the high-pass filter for the temperature variable, since we wanteed to remove the long-term trends but retain short-term variations,I selected W_filter = 1/(10*24), roughly corresponding to a 10-day cycle, which aligns with common weather forecast horizons. I also developed another function that detects anomalies in a time series using the Local Outlier Factor algorithm, where parameters such as the contamination and n_neighbors need to be adjusted. When switching to the eletricity production data across these five areas, I used the LOESS-based STL decomposition method to separate the data into trend, seasonal, and residual components. Given that this data is hourly and covers a full year, I set the period parameter to 24 and experimented with various smoothing parameters, while setting robust=True to improve resistance to noise.

Streamlit Application

- For the streamlit part, Since the core functions could be directly reused from the notebook, most of the work involved structuring the interface and organizing multiple visualizations. However, I encounted one chanllege when I tried to pass variables between different pages using st.session_state. Although the session ID remained consistent, the state values were not preserved across pages. As a result, I had to re-create the selection widgets (e.g., city or variable dropdowns) on each page, which is not user-friendly.

AI usage

During this assignment, I used GitHub Copilot in my VS Code environment to assist with code review, debugging, and improving overall code structure.

I also consulted ChatGPT to learn how to visualize data using Plotly, such as creating spectrograms and STL decomposition plots. Additionally, I explored ChatGPT-generated solutions for passing variables between different Streamlit pages—although these attempts were not fully successful on my end.

Furthermore, I used ChatGPT to gain a deeper understanding of the underlying concepts and techniques, particularly when tuning parameters and interpreting results for anomaly detection and time-series decomposition.

Tasks

Library imports

In [1]:
```
# Basic library
import requests
import pandas as pd
import numpy as np
import json

# Loading data from API
import openmeteo_requests
import requests_cache
from retry_requests import retry

# Loading data from MongoDB
from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi

# Ploting
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.io as pio
pio.renderers.default = "notebook_connected"

# Detecting outlier or anomalies
from scipy.fft import dct, idct
import scipy.stats as stats
from scipy.signal import stft
from sklearn.neighbors import LocalOutlierFactor
from statsmodels.tsa.seasonal import STL
```

1. Create the dataframe for the five cities in Norway

In [2]:
```
basic_data = {
    "city":["Oslo", "Kristiansand", "Trondheim", "Tromsø", "Bergen"],
    "price_area_code":["NO1", "NO2", "NO3", "NO4", "NO5"],
    "latitude": [59.9127, 58.1467, 63.4305, 69.6489, 60.393],
    "longitude": [10.7461, 7.9956, 10.3951, 18.9551, 5.3242]
}

basic_info = pd.DataFrame(basic_data)
```

2. Create a function to load data using API

In [3]:
```
def load_data_fromAPI(longitude, latitude, selected_year):
        # Setup the Open-Meteo API client with cache and retry on error
        cache_session = requests_cache.CachedSession('.cache', expire_after = -1)
        retry_session = retry(cache_session, retries = 5, backoff_factor = 0.2)
        openmeteo = openmeteo_requests.Client(session = retry_session)

        # Make sure all required weather variables are listed here
        # The order of variables in hourly or daily is important to assign them correctly below
        url = "https://archive-api.open-meteo.com/v1/archive"
        params = {
                "latitude": latitude,
```

```
        "longitude": longitude,
"start_date": f"{selected_year}-01-01",
"end_date": f"{selected_year}-12-31",
        "hourly": ["temperature_2m", "wind_speed_10m", "wind_gusts_10m", "wind_direction_10m", "precipitation"],
        "models": "era5",
        "timezone": "auto",
}
responses = openmeteo.weather_api(url, params=params)

# Process first location. Add a for-loop for multiple locations or weather models
response = responses[0]
print(f"Coordinates: {response.Latitude()}°N {response.Longitude()}°E")
print(f"Date_range: {params['start_date']} - {params['end_date']}")
print(f"Variables: {params['hourly']}")
#print(f"Elevation: {response.Elevation()} m asl")
#print(f"Timezone difference to GMT+0: {response.UtcOffsetSeconds()}s")

# Process hourly data. The order of variables needs to be the same as requested.
hourly = response.Hourly()
hourly_temperature_2m = hourly.Variables(0).ValuesAsNumpy()
hourly_wind_speed_10m = hourly.Variables(1).ValuesAsNumpy()
hourly_wind_gusts_10m = hourly.Variables(2).ValuesAsNumpy()
hourly_wind_direction_10m = hourly.Variables(3).ValuesAsNumpy()
hourly_precipitation = hourly.Variables(4).ValuesAsNumpy()

hourly_data = {"date": pd.date_range(
        start = pd.to_datetime(hourly.Time(), unit = "s", utc = True),
        end =  pd.to_datetime(hourly.TimeEnd(), unit = "s", utc = True),
        freq = pd.Timedelta(seconds = hourly.Interval()),
        inclusive = "left"
)}

hourly_data["temperature_2m"] = hourly_temperature_2m
hourly_data["wind_speed_10m"] = hourly_wind_speed_10m
hourly_data["wind_gusts_10m"] = hourly_wind_gusts_10m
hourly_data["wind_direction_10m"] = hourly_wind_direction_10m
hourly_data["precipitation"] = hourly_precipitation

hourly_dataframe = pd.DataFrame(data = hourly_data)
# Change the time zone to Europe/Oslo
hourly_dataframe["date"] = hourly_dataframe["date"].dt.tz_convert("Europe/Oslo")
print(f"Sucessfully load the data")

return hourly_dataframe
```

In [4]:
```
# here, we use Bergen as a example to test our function
selected_year = 2019
selected_city = 'Bergen'
latitude = basic_info[basic_info["city"] == selected_city]['latitude']
longitude = basic_info[basic_info["city"] == selected_city]['longitude']
hourly_dataframe = load_data_fromAPI(longitude,latitude,selected_year)
```

```
Coordinates: 60.5°N 5.25°E
Date_range: 2019-01-01 - 2019-12-31
Variables: ['temperature_2m', 'wind_speed_10m', 'wind_gusts_10m', 'wind_direction_10m', 'precipitation']
Sucessfully load the data
```

Briefly explore the dataset: no missing data

In [5]:
```
hourly_dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   date                8760 non-null   datetime64[ns, Europe/Oslo]
 1   temperature_2m      8760 non-null   float32
 2   wind_speed_10m      8760 non-null   float32
 3   wind_gusts_10m      8760 non-null   float32
 4   wind_direction_10m  8760 non-null   float32
 5   precipitation       8760 non-null   float32
dtypes: datetime64[ns, Europe/Oslo](1), float32(5)
memory usage: 239.7 KB
```

In [6]:
```
hourly_dataframe.describe()
```

Out[6]:

	temperature_2m	wind_speed_10m	wind_gusts_10m	wind_direction_10m	precipitation
count	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000
mean	8.846621	14.565521	29.391081	186.588104	0.254053
std	5.089198	8.156624	15.241786	101.865685	0.541051
min	-5.450000	0.000000	4.680000	0.674022	0.000000
25%	4.750000	8.373386	17.639999	107.915705	0.000000
50%	8.050000	12.849528	26.280001	164.664124	0.000000
75%	12.700000	19.657119	38.519997	286.471169	0.300000
max	29.350000	55.753529	119.879997	360.000000	7.300000

In [7]:
```
hourly_dataframe.head()
```

Out[7]:

	date	temperature_2m	wind_speed_10m	wind_gusts_10m	wind_direction_10m	precipitation
0	2019-01-01 00:00:00+01:00	6.70	42.671768	81.720001	260.776154	0.4
1	2019-01-01 01:00:00+01:00	6.55	47.959782	87.839996	277.765076	0.5
2	2019-01-01 02:00:00+01:00	6.80	48.621330	80.279999	296.375275	0.9
3	2019-01-01 03:00:00+01:00	6.85	52.638840	85.320000	310.006195	0.7
4	2019-01-01 04:00:00+01:00	6.55	55.753529	98.639999	314.215271	0.6

3. Outliers and anomalies:

Plot the temperature as a function of time.

The `plot_outlier_detection_dct()` functionPerforms high-pass filtering with DCT to remove seasonal effects and detects outliers using robust MAD-based SPC boundaries. Returns an interactive Plotly plot.

In [8]:
```
def plot_outlier_detection_dct(hourly_dataframe,selected_variable: str, W_filter: float = 1/(10*24), coef_k: float = 3):
    selected_data = hourly_dataframe[selected_variable]
```

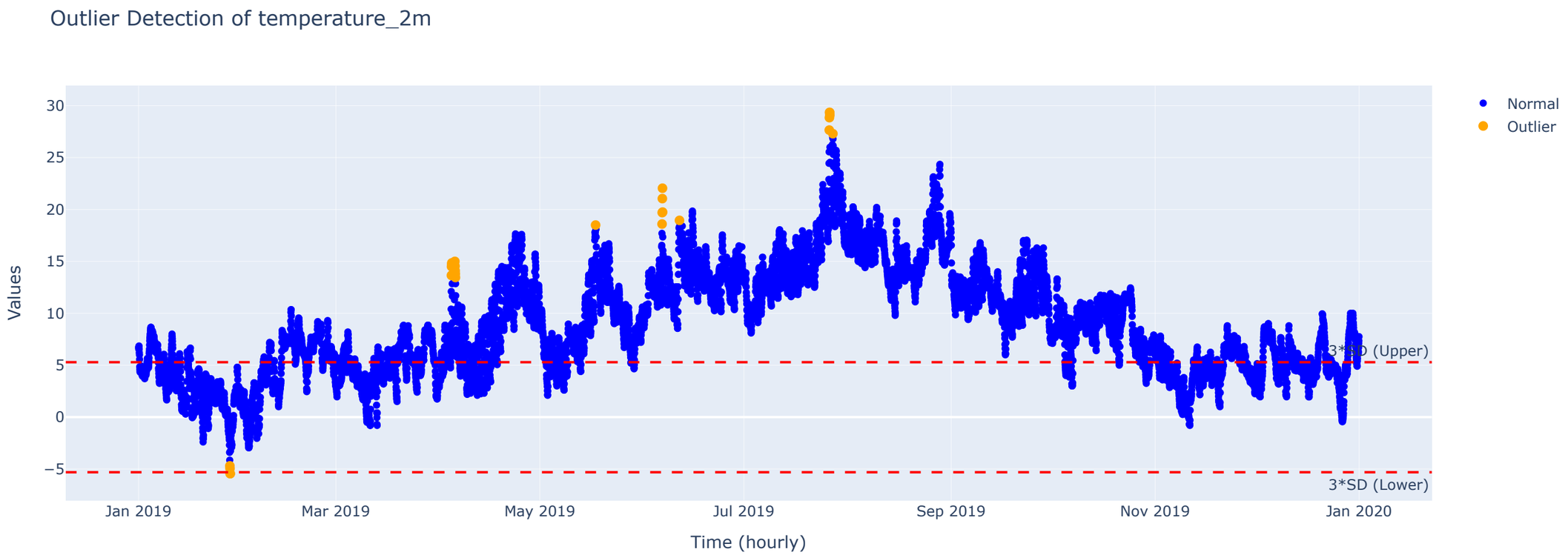
```
    N = hourly_dataframe.shape[0]
    dt = 1
    W = np.linspace(0, 1/(2*dt), N) # cycles/hour
    # Discrete Cosine transform
    fourier_signal = dct(selected_data.values, type=1, norm="forward")
    filtered_fourier_signal = fourier_signal.copy()
    filtered_fourier_signal[(W < W_filter)] = 0 # high-pass filter
    satv = idct(filtered_fourier_signal, type=1, norm="forward")

    # Median absolute deviation
    coef_k = coef_k
    trimmed_means = stats.trim_mean(satv, 0.05)
    mad =stats.median_abs_deviation(satv)
    sd = mad * 1.4826
    # Find the boundaries
    upper_boundary = trimmed_means + coef_k * sd
    lower_boundary = trimmed_means - coef_k * sd
    # Detect the outliers
    outliers_index = np.where(np.abs(satv - trimmed_means) > coef_k * sd)[0]
    df_temp = hourly_dataframe.reset_index(drop=True)

    # Plot
    fig = go.Figure()
    fig.add_hline(y=upper_boundary, line_color='red', line_dash='dash',
                  annotation_text=f'{coef_k}*SD (Upper)', annotation_position='top right')
    fig.add_hline(y=lower_boundary, line_color='red', line_dash='dash',
                  annotation_text=f'{coef_k}*SD (Lower)', annotation_position='bottom right')

    fig.add_trace(go.Scatter(x=df_temp['date'], y=selected_data, mode='markers', marker=dict(color='blue'), name='Normal'))
    fig.add_trace(go.Scatter(x=df_temp.loc[outliers_index, 'date'],
                             y=df_temp.loc[outliers_index, selected_variable],
                             mode='markers', marker=dict(color='orange', size=8),
                             name='Outlier'))
    fig.update_layout(
        title=f"Outlier Detection of {selected_variable}",
        xaxis_title="Time (hourly)",
        yaxis_title="Values"
    )
    # add some basic info about the outliers
    summary = {
        "variable": selected_variable,
        "num_sample": N,
        "Sigma Multiplier (k)":coef_k,
        "High-pass Filter W_cutoff":W_filter,
        "upper_boundary": upper_boundary,
        "lower_boundary": lower_boundary,
        "num_outliers": len(outliers_index),
        "ratio_outlier": round(len(outliers_index) / N * 100, 2)
    }
    return fig,summary
```

In [9]:

```
# Test the plot_outlier_detection_dct
fig,dct_summary = plot_outlier_detection_dct(hourly_dataframe,"temperature_2m")
for k, v in dct_summary.items():
    print(f"{k}: {v}")
```

```
variable: temperature_2m
num_sample: 8760
Sigma Multiplier (k): 3
High-pass Filter W_cutoff: 0.004166666666666667
upper_boundary: 5.287843227386475
lower_boundary: -5.313112735748291
num_outliers: 29
ratio_outlier: 0.33
```

In [10]:
```
# show the plot within detected outliers
fig.show()
```



Plot the precipitation as a function of time.

The `plot_outlier_detection_lof()` function detects anomalies in a time series using the Local Outlier Factor (LOF) algorithm and visualizes normal and outlier points in an scatter plot

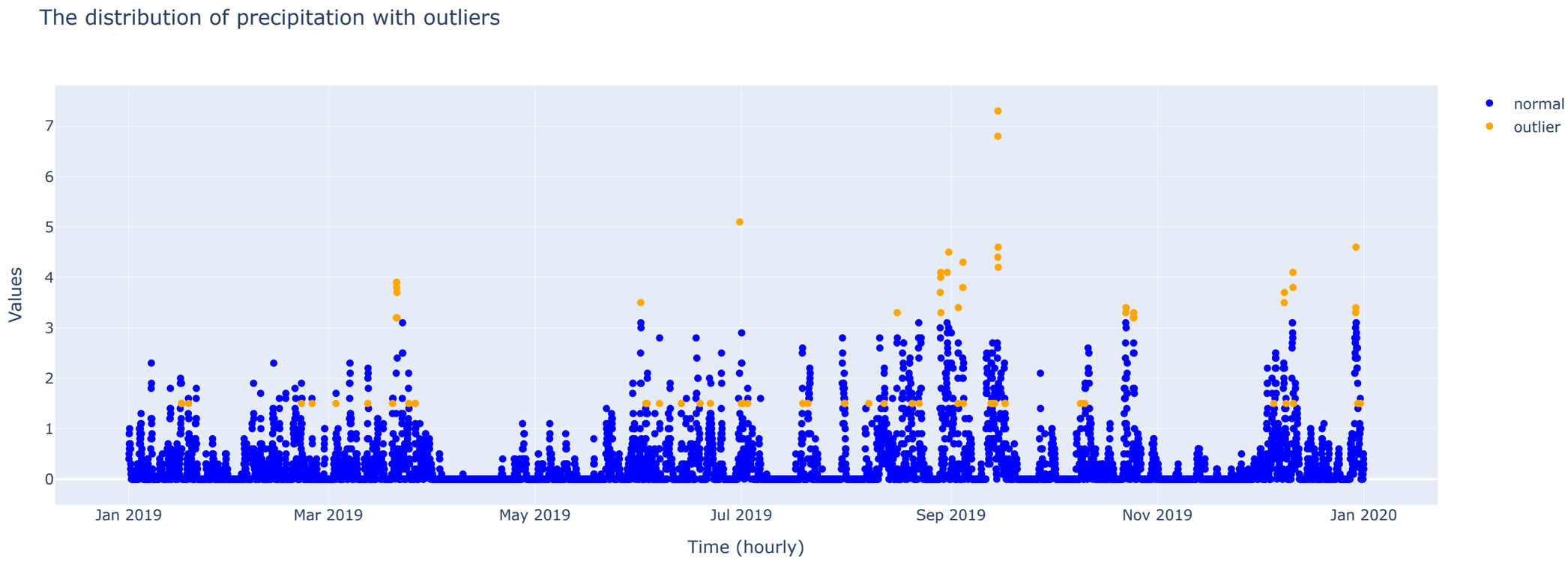
In [11]:
```
def plot_outlier_detection_lof(hourly_dataframe, selected_variable: str, contamination: float = 0.01, n_neighbors: int = 50):
    # Prepare the input data for LOF
    selected_data = hourly_dataframe[[selected_variable]].copy()

    # Fit LOF model
    lof = LocalOutlierFactor(n_neighbors=n_neighbors, contamination=contamination)
    pred_labels = lof.fit_predict(selected_data)

    # Separate normal and outliers
    outlier_mask = pred_labels == -1
    normal_mask = pred_labels == 1
```

```
    # Visualization
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=hourly_dataframe.loc[normal_mask, 'date'],y=selected_data.loc[normal_mask, selected_variable], mode='markers', marker=dict(color='blue'), name=
    fig.add_trace(go.Scatter(x=hourly_dataframe.loc[outlier_mask, 'date'],y=selected_data.loc[outlier_mask, selected_variable], mode='markers', marker=dict(color='orange'), na
    fig.update_layout(title=f'The distribution of {selected_variable} with outliers', xaxis_title='Time (hourly)', yaxis_title='Values')

    # Add the brief summary
    summary = {
        "variable": selected_variable,
        "num_sample": len(selected_data),
        "contamination_param":contamination,
        "n_neighbors":n_neighbors,
        "num_outliers": outlier_mask.sum(),
        "ratio_outlier (%)": round(outlier_mask.mean() * 100, 2),
        "mean_value": round(selected_data[selected_variable].mean(), 2)
    }

    return fig, summary
```

In [12]:
```
# Test the data
fig,lof_summary= plot_outlier_detection_lof(hourly_dataframe, "precipitation")
for k, v in lof_summary.items():
    print(f"{k}: {v}")
```

```
variable: precipitation
num_sample: 8760
contamination_param: 0.01
n_neighbors: 50
num_outliers: 80
ratio_outlier (%): 0.91
mean_value: 0.25
/opt/miniconda3/envs/D2D_project/lib/python3.12/site-packages/sklearn/neighbors/_lof.py:322: UserWarning:

Duplicate values are leading to incorrect results. Increase the number of neighbors for more accurate results.
```

In [13]:
```
# show the figure
fig.show()
```



4.Seasonal-Trend decomposition using LOESS (STL)

In [14]:
```python
# Load the data from Mongodb
with open("config_local.json") as f:
        config = json.load(f)

mongo_uri = config["mongo_uri"]

# Create a new client and connect to the server
client = MongoClient(mongo_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)

# connect to the Mongodb
db = client["elhub_db"]
collection = db["production_data"]
```

```
Pinged your deployment. You successfully connected to MongoDB!
```

In [15]:
```python
# check the number of data
count = collection.count_documents({})
print(f"Document count: {count:,}")
```

```
Document count: 215,033
```

In [16]:
```
# process the starttime column and sort the values by starttime
cursor = collection.find({}, {"_id": 0}) # remove the id
df_production = pd.DataFrame(list(cursor))
df_production.head()
```

Out[16]:

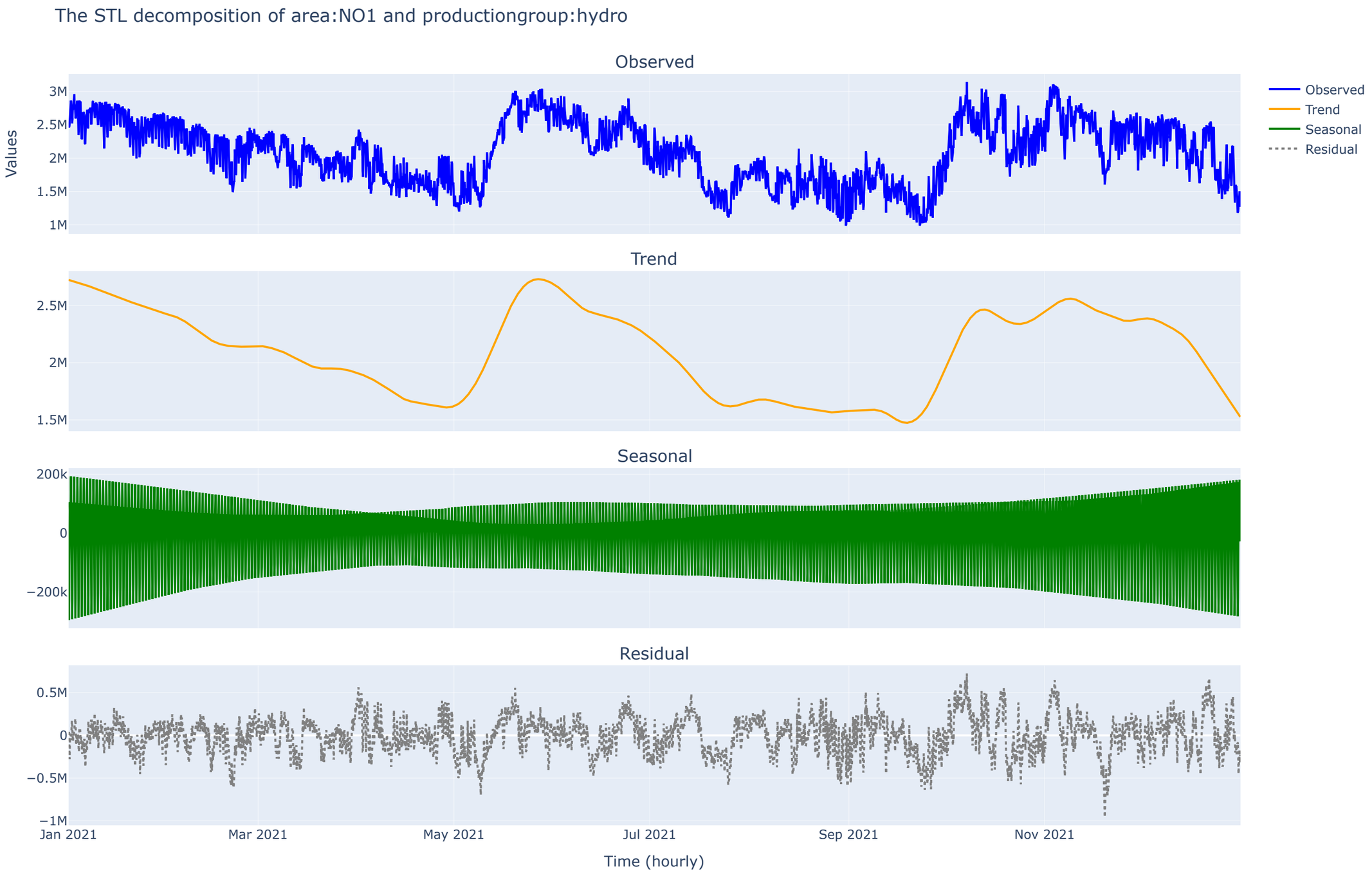
	pricearea	productiongroup	starttime	quantitykwh
0	NO5	thermal	2021-01-01 06:00:00	77913.0
1	NO5	thermal	2021-01-01 08:00:00	78222.0
2	NO5	thermal	2021-01-01 10:00:00	78141.0
3	NO5	thermal	2021-01-01 11:00:00	78399.0
4	NO5	thermal	2021-01-01 15:00:00	78157.0

In [17]:
```
df_production['starttime'] = pd.to_datetime(df_production['starttime'])
df_production = df_production.sort_values("starttime")
```

The plot_stl_decompostion() function performs a Seasonal-Trend decomposition using LOESS (STL) on production data for a selected area and production group, and visualizes the observed, trend, seasonal, and residual components in an interactive Plotly subplot.

In [18]:
```
def plot_stl_decompostion(df_production, area:str = 'NO1',group:str = 'hydro',period:int = 24,seasonal:int = 24*10+1,trend:int =24*30+1 ,robust:bool = True):
    # Prepare the data
    df_subset = df_production[(df_production['pricearea']==area) & (df_production['productiongroup']==group)]
    df_subset.reset_index(inplace=True,drop=True)
    # deploy the STL decompostion method
    stl = STL(df_subset["quantitykwh"], period=period,seasonal=seasonal,trend=trend,robust=robust)
    res = stl.fit() # Contains the components and a plot function
    ## Plot the results
    fig = go.Figure()
    fig = make_subplots(rows=4, cols=1, shared_xaxes=True,subplot_titles=["Observed", "Trend", "Seasonal", "Residual"],vertical_spacing=0.05)
    # Add original data
    fig.add_trace(go.Scatter(x=df_subset['starttime'],y=df_subset["quantitykwh"],mode='lines',name='Observed',line=dict(color='blue')),row=1, col=1)
    # Add Trend
    fig.add_trace(go.Scatter(x=df_subset['starttime'],y=res.trend,mode='lines',name='Trend',line=dict(color='orange')),row=2, col=1)
    # Add Seasonal
    fig.add_trace(go.Scatter(x=df_subset['starttime'],y=res.seasonal,mode='lines',name='Seasonal',line=dict(color='green')),row=3, col=1)
    # Add Residual
    fig.add_trace(go.Scatter(x=df_subset['starttime'],y=res.resid,mode='lines',name='Residual',line=dict(color='gray', dash='dot')),row=4, col=1)
    fig.update_layout(title=f'The STL decomposition of area:{area} and productiongroup:{group}', xaxis4_title='Time (hourly)', yaxis_title='Values',height=900)

    return fig
```

In [19]:
```
# Test the funciton
fig = plot_stl_decompostion(df_production)
fig.show()
```

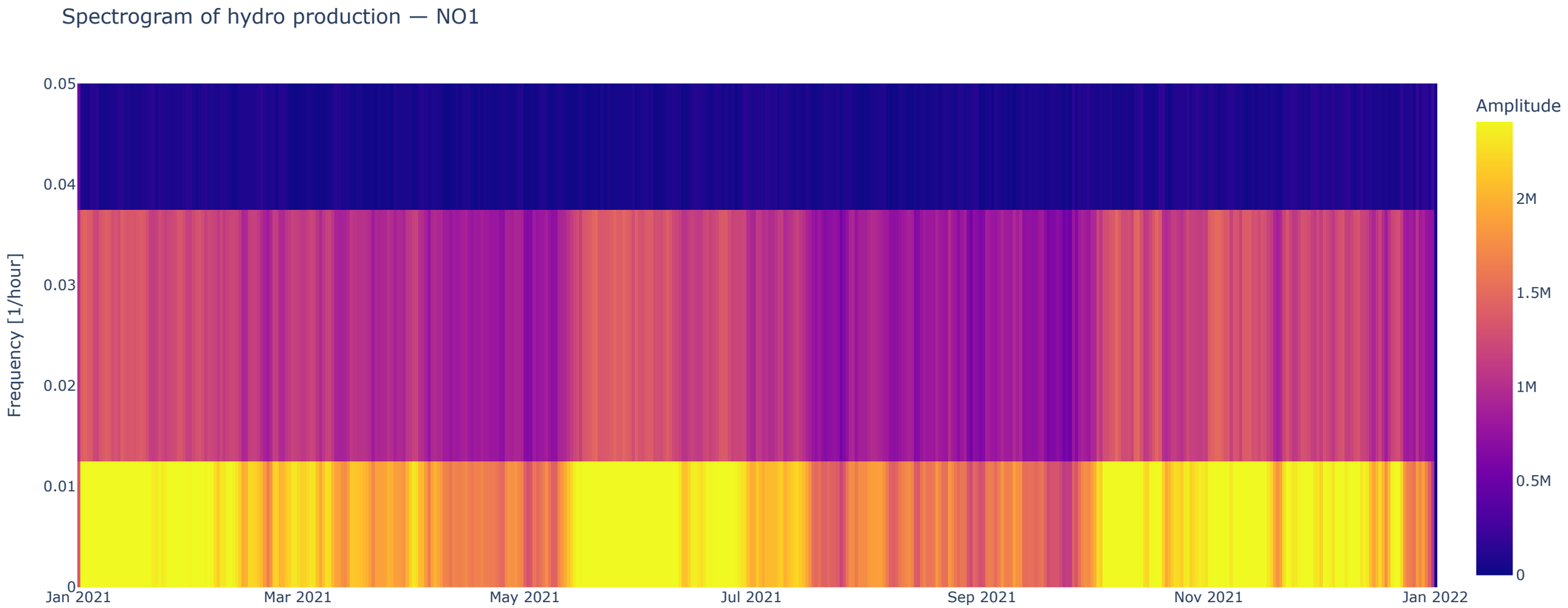


5.Spectrogram

The `plot_spectrogram_elhub()` function computes a Short-Time Fourier Transform of production data to visualize how signal energy varies across time and frequency, returning an spectrogram.

In [20]:
```python
def plot_spectrogram_elhub(df_production,area: str = "NO1",group: str = "hydro",nperseg: int = 40,noverlap: int = 20):
    # Prepare the dataset
    df_subset = df_production[(df_production["pricearea"] == area)& (df_production["productiongroup"] == group)].sort_values("starttime")
    y = df_subset["quantitykwh"].values # Signal
    fs = 1 # Sampling frequency
    # Deploy the short-time fourier transform
    f, t, Zxx = stft(y, fs=fs, nperseg=nperseg, noverlap=noverlap)

    # Visulazation
    fig = go.Figure()
    magnitude = np.abs(Zxx)
    start_time = df_subset["starttime"].iloc[0]
    t_datetime = [start_time + pd.Timedelta(hours=float(h)) for h in t]

    fig.add_trace(go.Heatmap(
        x=t_datetime,
        y=f,
        z=magnitude,
        colorbar=dict(title="Amplitude"),
        zmin=0,
        zmax=magnitude.max() * 0.8,
        hovertemplate="Date: %{x|%Y-%m-%d %H:%M}<br>Freq: %{y:.4f}/h<br>Amp: %{z:.2f}<extra></extra>"
    ))

    fig.update_layout(
        title=f"Spectrogram of {group} production - {area}",
        xaxis_title='Time (hourly)',
        yaxis_title="Frequency [1/hour]",
        template="plotly_white",
        height=600,
    )
    fig.update_yaxes(range=[0, 0.05]) # focus on the low-frequency
    return fig
```

In [21]:
```python
# Test the function
fig = plot_spectrogram_elhub(df_production)
fig.show()
```


Spectrogram of hydro production — NO1
Frequency [1/hour]
0.05
0.04
0.03
0.02
0.01
0
Jan 2021
Mar 2021
May 2021
Jul 2021
Sep 2021
Nov 2021
Jan 2022
Time (hourly)
Amplitude
2M
1.5M
1M
0.5M
0