

# IND320 - Assignment 3

Name: Eskil Torp Skogsholm

GitHub link: [https://github.com/esksko/IND320-assignments-esksko/tree/assignment\\_3](https://github.com/esksko/IND320-assignments-esksko/tree/assignment_3)

Streamlit app: <https://temp-assignment3.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 debug and understand the API, DCT, LOF, LOESS and Spectrogram better.

## Log

First, I started by making a new GitHub branch called "assignment\_3" to ensure peer review was finished before pushing new updates to the main branch of the repository. After this, I began working on the Jupyter Notebook tasks. The open-meteo API was straightforward to work with. After exploring the documentation at open-meteo.com, I identified the relevant sections for ERA5 historical reanalysis data. I created a DataFrame containing the necessary data, getting their coordinates and area from Wikipedia. Creating the API function was just a matter of entering the latitude, longitude and year as parameters, and making sure that the API link was formatted correctly.

Using the Discrete Cosine Transform (DCT) was a little bit more difficult, but turned out to be quite simple once I understood how it worked by tinkering with it a bit. After that, I used the SATV to detect outliers by computing the median, mad and a robust estimate for standard deviation. From this, it was quite easy to define the SPC boundaries, adding the temperature to get them as curves and plotting them together with the temperature and outliers.

Using LOF (Local Outlier Factor) was also quite simple. It was just a matter of using the LocalOutlierFactor function from sklearn.neighbors on data with the correct shape, fitting the LOF, and using a mask to detect outliers. Using LOESS (STL) and creating the spectrogram followed pretty much the same process. For the streamlit part of the assignment, I first re-organized all the pages to adhere to the task description. Then, I updated the table-page to use API instead of downloaded csv data. For the new pages NewA and NewB, getting the variables to not reset every time I changed page was somewhat complicated as I had some minor issues with the st.session\_state function. Currently the functionality works, but there is room for improvement.

## Tasks

```
In [74]: import pandas as pd
import requests

# Using Oslo, Kristiansand, Trondheim, Tromsø and Bergen as representatives for the
cities_data = [
    {"area_code": "NO1", "city": "Oslo", "latitude": 59.9127, "longitude": 10.7461},
    {"area_code": "NO2", "city": "Kristiansand", "latitude": 58.1467, "longitude": 7.9956},
    {"area_code": "NO3", "city": "Trondheim", "latitude": 63.4305, "longitude": 10.3951},
    {"area_code": "NO4", "city": "Tromsø", "latitude": 69.6489, "longitude": 18.9551},
    {"area_code": "NO5", "city": "Bergen", "latitude": 60.393, "longitude": 5.3242}
]

# Converting to DataFrame
cities_df = pd.DataFrame(cities_data)

cities_df.head()
```

Out[74]:

	area_code	city	latitude	longitude
0	NO1	Oslo	59.9127	10.7461
1	NO2	Kristiansand	58.1467	7.9956
2	NO3	Trondheim	63.4305	10.3951
3	NO4	Tromsø	69.6489	18.9551
4	NO5	Bergen	60.3930	5.3242

```
In [ ]: def download_weather_data(lat, lon, year, variables=["temperature_2m", "precipitation_mm"]):
    url = f"https://archive-api.open-meteo.com/v1/era5?latitude={lat}&longitude={lon}&start_date={year}-01-01&end_date={year}-12-31&hourly={variables}"

    # Appending variables to the URL
    for var in variables:
        url += f"&{var},"
        if var != variables[-1] else f"&{var}"
    url += "&timezone=Europe%2FOslo"

    print(f"Downloading data from: {url}")
    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()

        # Extracting hourly data into dataframe
        hourly_data = data.get("hourly", {})
        df = pd.DataFrame(hourly_data)

        # Converting time column to datetime
        df["time"] = pd.to_datetime(df["time"])

    return df
```

```

    else:
        print(f"Failed to retrieve data: {response.status_code}")
        return None

```

In [76]: # Downloading weather data for Bergen for the year 2019

```
hourly_df = download_weather_data(60.3930, 5.3242, 2019)
```

Downloading data from: https://archive-api.open-meteo.com/v1/era5?latitude=60.393&longitude=5.3242&start\_date=2019-01-01&end\_date=2019-12-31&hourly=temperature\_2m,precipitation,wind\_speed\_10m,wind\_gusts\_10m,wind\_direction\_10m&timezone=Europe%2Foslo

In [77]: hourly\_df.head()

Out[77]:

	time	temperature_2m	precipitation	wind_speed_10m	wind_gusts_10m	wind_direction_10m
0	2019-01-01 00:00:00	5.7	0.7	37.0	99.7	
1	2019-01-01 01:00:00	5.8	0.2	41.0	107.3	
2	2019-01-01 02:00:00	6.1	0.7	42.0	112.0	
3	2019-01-01 03:00:00	6.3	0.5	40.9	105.8	
4	2019-01-01 04:00:00	5.8	1.1	41.2	110.2	



## Outliers and anomalies

```

In [ ]: import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import dct, idct

# Using DCT to perform high-pass filtering on the temperature data

def dct_highpass_filter(signal, freq_cutoff):
    # Perform DCT
    signal_dct = dct(signal, norm="ortho")

    # Zero out low-frequency components
    filtered_dct = np.copy(signal_dct)

```

```

    filtered_dct[:freq_cutoff] = 0

    # Reconstruct high-frequency signal using inverse DCT
    filtered_signal = idct(filtered_dct, norm="ortho")

    return filtered_signal

```

```

In [79]: satv = dct_highpass_filter(hourly_df["temperature_2m"].values, freq_cutoff=100)

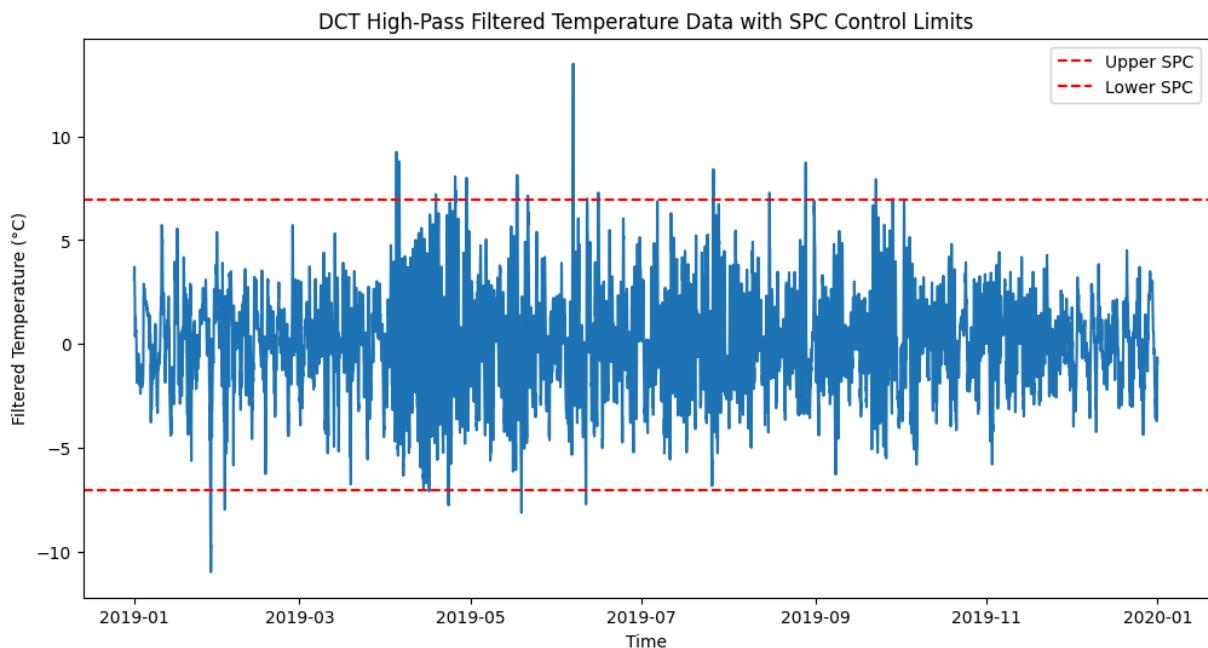
# Computing robust statistics
median = np.median(satv)
mad = np.median(np.abs(satv - median))
robust_std = mad * 1.4826

# Defining SPC and control limits
upper_bound = median + 3 * robust_std
lower_bound = median - 3 * robust_std

plt.figure(figsize=(12, 6))
plt.plot(hourly_df["time"], satv)
plt.axhline(upper_bound, color='r', linestyle='--', label='Upper SPC')
plt.axhline(lower_bound, color='r', linestyle='--', label='Lower SPC')
plt.title("DCT High-Pass Filtered Temperature Data with SPC Control Limits")
plt.xlabel("Time")
plt.ylabel("Filtered Temperature (°C)")
plt.legend()

```

Out[79]: <matplotlib.legend.Legend at 0x169cefeb650>



```

In [80]: def detect_temperature_outliers(time, temperature, freq_cutoff=100, num_std=3):
    # Apply high-pass DCT filter
    satv = dct_highpass_filter(temperature, freq_cutoff)

    # Computing robust statistics
    median = np.median(satv)

```

```
mad = np.median(np.abs(satv - median))
robust_std = mad * 1.4826

# Defining SPC and control limits
upper_bound = median + num_std * robust_std
lower_bound = median - num_std * robust_std

upper_curve = temperature + (median + num_std * robust_std - satv)
lower_curve = temperature + (median - num_std * robust_std - satv)

# Detecting outliers
outlier_mask = (satv > upper_bound) | (satv < lower_bound)
outlier_indices = np.where(outlier_mask)[0]

# Plot temperature and highlight outliers
fig, ax = plt.subplots(figsize=(15, 6))
ax.plot(time, temperature, label="Temperature", alpha=0.7)
#ax.hlines([upper_bound, lower_bound], xmin=time.min(), xmax=time.max(), colors='black')
ax.scatter(time[outlier_mask], temperature[outlier_mask],
           color="red", label="Outliers", zorder=5)

plt.plot(time, upper_curve, color='orange', linestyle='--', label='Upper SPC')
plt.plot(time, lower_curve, color='orange', linestyle='--', label='Lower SPC')

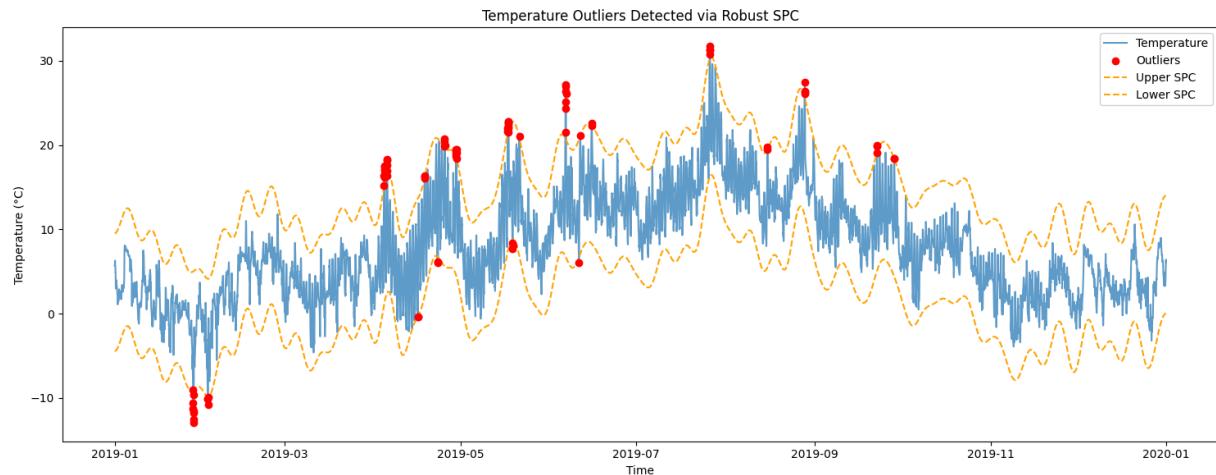
ax.set_xlabel("Time")
ax.set_ylabel("Temperature (°C)")
ax.set_title("Temperature Outliers Detected via Robust SPC")
ax.legend()
plt.tight_layout()
plt.show()

# Preparing summary

summary = {
    "outlier_indices": outlier_indices,
    "outlier_times": time[outlier_mask],
    "outlier_temperatures": temperature[outlier_mask],
    "num_outliers": len(outlier_indices),
    "upper_bound": upper_bound,
    "lower_bound": lower_bound,
    "robust_median": median,
    "robust_std": robust_std
}

return fig, summary

fig, summary = detect_temperature_outliers(hourly_df["time"].values, hourly_df["tem
```



In [81]: `summary`

```
Out[81]: {'outlier_indices': array([ 652,  653,  654,  655,  656,  657,  658,  659,  775,
    776,  777,
        778, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2267, 2268, 2269,
    2270, 2271, 2525, 2526, 2582, 2583, 2584, 2689, 2693, 2747, 2748,
    2749, 2751, 2842, 2843, 2844, 2845, 2846, 2847, 2848, 3274, 3275,
    3276, 3277, 3278, 3279, 3280, 3281, 3282, 3315, 3316, 3317, 3371,
    3754, 3755, 3756, 3757, 3758, 3759, 3760, 3868, 3878, 3973, 3974,
    4957, 4958, 4959, 4960, 5438, 5439, 5747, 5748, 5749, 5750, 6349,
    6350, 6351, 6493, 6494]),
'outlier_times': array(['2019-01-28T04:00:00.000000000', '2019-01-28T05:00:00.000
000000',
'2019-01-28T06:00:00.000000000', '2019-01-28T07:00:00.000000000',
'2019-01-28T08:00:00.000000000', '2019-01-28T09:00:00.000000000',
'2019-01-28T10:00:00.000000000', '2019-01-28T11:00:00.000000000',
'2019-02-02T07:00:00.000000000', '2019-02-02T08:00:00.000000000',
'2019-02-02T09:00:00.000000000', '2019-02-02T10:00:00.000000000',
'2019-04-04T11:00:00.000000000', '2019-04-04T12:00:00.000000000',
'2019-04-04T13:00:00.000000000', '2019-04-04T14:00:00.000000000',
'2019-04-04T15:00:00.000000000', '2019-04-04T16:00:00.000000000',
'2019-04-04T17:00:00.000000000', '2019-04-05T11:00:00.000000000',
'2019-04-05T12:00:00.000000000', '2019-04-05T13:00:00.000000000',
'2019-04-05T14:00:00.000000000', '2019-04-05T15:00:00.000000000',
'2019-04-16T05:00:00.000000000', '2019-04-16T06:00:00.000000000',
'2019-04-18T14:00:00.000000000', '2019-04-18T15:00:00.000000000',
'2019-04-18T16:00:00.000000000', '2019-04-23T01:00:00.000000000',
'2019-04-23T05:00:00.000000000', '2019-04-25T11:00:00.000000000',
'2019-04-25T12:00:00.000000000', '2019-04-25T13:00:00.000000000',
'2019-04-25T15:00:00.000000000', '2019-04-29T10:00:00.000000000',
'2019-04-29T11:00:00.000000000', '2019-04-29T12:00:00.000000000',
'2019-04-29T13:00:00.000000000', '2019-04-29T14:00:00.000000000',
'2019-04-29T15:00:00.000000000', '2019-04-29T16:00:00.000000000',
'2019-05-17T10:00:00.000000000', '2019-05-17T11:00:00.000000000',
'2019-05-17T12:00:00.000000000', '2019-05-17T13:00:00.000000000',
'2019-05-17T14:00:00.000000000', '2019-05-17T15:00:00.000000000',
'2019-05-17T16:00:00.000000000', '2019-05-17T17:00:00.000000000',
'2019-05-17T18:00:00.000000000', '2019-05-19T03:00:00.000000000',
'2019-05-19T04:00:00.000000000', '2019-05-19T05:00:00.000000000',
'2019-05-21T11:00:00.000000000', '2019-06-06T10:00:00.000000000',
'2019-06-06T11:00:00.000000000', '2019-06-06T12:00:00.000000000',
'2019-06-06T13:00:00.000000000', '2019-06-06T14:00:00.000000000',
'2019-06-06T15:00:00.000000000', '2019-06-06T16:00:00.000000000',
'2019-06-11T04:00:00.000000000', '2019-06-11T14:00:00.000000000',
'2019-06-15T13:00:00.000000000', '2019-06-15T14:00:00.000000000',
'2019-07-26T13:00:00.000000000', '2019-07-26T14:00:00.000000000',
'2019-07-26T15:00:00.000000000', '2019-07-26T16:00:00.000000000',
'2019-08-15T14:00:00.000000000', '2019-08-15T15:00:00.000000000',
'2019-08-28T11:00:00.000000000', '2019-08-28T12:00:00.000000000',
'2019-08-28T13:00:00.000000000', '2019-08-28T14:00:00.000000000',
'2019-09-22T13:00:00.000000000', '2019-09-22T14:00:00.000000000',
'2019-09-22T15:00:00.000000000', '2019-09-28T13:00:00.000000000',
'2019-09-28T14:00:00.000000000'], dtype='datetime64[ns']'),
'outlier_temperatures': array([-9. , -10.6, -11.3, -11.8, -11.6, -12.9, -12.5,
    -9.6, -10.1,
        -10.8, -9.9, 15.2, 16.4, 16.9, 17.5, 17.4, 17.1,
    16.2, 16.4, 17.6, 18.2, 18.3, 16.9, -0.4, -0.4, 16.1,
    16.4, 16.3, 6.1, 6. , 19.9, 20.7, 20.4, 20. , 18.7,
```

```

    19. , 19.5, 19.3, 19.5, 19.1, 18.4, 21.6, 22. , 22.2,
    21.5, 22.5, 22.8, 22.8, 22.6, 22. , 8.4, 7.7, 8.1,
    21. , 21.5, 24.3, 25.1, 26.4, 27. , 27.2, 26.1, 6. ,
    21.1, 22.3, 22.6, 30.8, 31.3, 31.7, 31.2, 19.5, 19.8,
    26.1, 26.4, 27.4, 26.3, 19.1, 20. , 19.9, 18.4, 18.4]),
'num_outliers': 81,
'upper_bound': np.float64(6.951417523227211),
'lower_bound': np.float64(-7.015236795395043),
'robust_median': np.float64(-0.03190963608391589),
'robust_std': np.float64(2.3277757197703757)}

```

## LOF - Local Outlier Factor on precipitation

```
In [82]: from sklearn.neighbors import LocalOutlierFactor

def detect_precipitation_outliers(time, precipitation, contamination=0.01, n_neighbors=20):

    # Reshaping data for LOF
    X = np.array(precipitation).reshape(-1, 1)

    # LOF
    lof = LocalOutlierFactor(n_neighbors=n_neighbors, contamination=contamination)
    pred_labels = lof.fit_predict(X)
    scores = -lof.negative_outlier_factor_ # Higher scores = More anomalous

    # Identifying outliers (LOF labels outliers as -1)
    outlier_mask = pred_labels == -1
    outlier_indices = np.where(outlier_mask)[0]

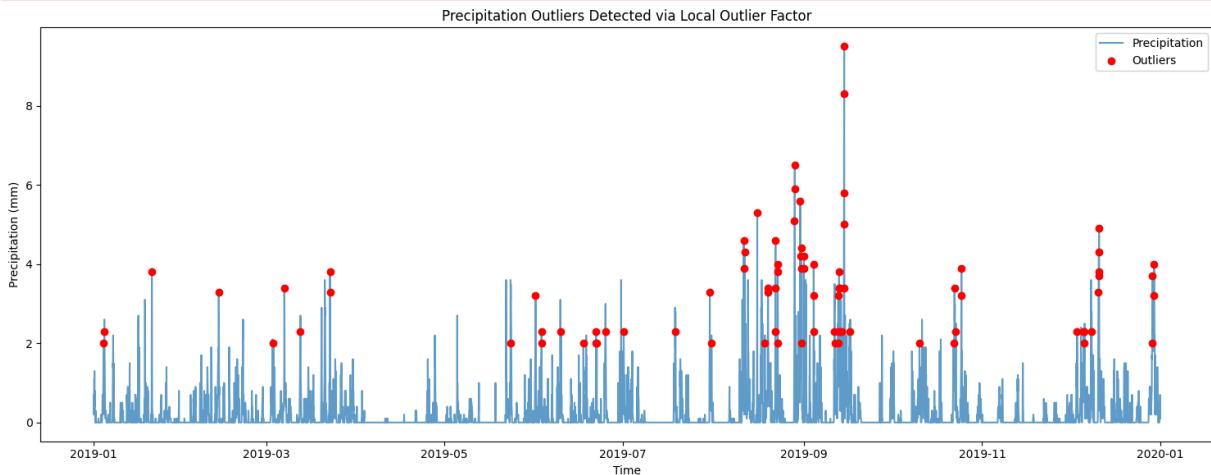
    # Plot precipitation and highlight outliers
    fig, ax = plt.subplots(figsize=(15, 6))
    ax.plot(time, precipitation, label="Precipitation", alpha=0.7)
    ax.scatter(time[outlier_mask], precipitation[outlier_mask],
               color="red", label="Outliers", zorder=5)

    ax.set_xlabel("Time")
    ax.set_ylabel("Precipitation (mm)")
    ax.set_title("Precipitation Outliers Detected via Local Outlier Factor")
    ax.legend()
    plt.tight_layout()
    plt.show()

    # Summary
    summary = {
        "num_outliers": len(outlier_indices),
        "outlier_indices": outlier_indices,
        "outlier_times": time[outlier_mask],
        "outlier_precipitations": precipitation[outlier_mask],
        "lof_scores": scores[outlier_mask]
    }

    return fig, summary
```

```
fig, summary = detect_precipitation_outliers(hourly_df["time"].values, hourly_df["p
c:\Users\esksk\miniconda3\envs\IND320_env\Lib\site-packages\sklearn\neighbors\_lof.p
y:322: UserWarning: Duplicate values are leading to incorrect results. Increase the
number of neighbors for more accurate results.
warnings.warn(
```



## LOESS (STL) - Seasonal-Trend decomposition

```
In [83]: from statsmodels.tsa.seasonal import STL

def stl_decomposition(df, price_area="N01", production_group="Solar", period=24, se
data = df[(df["pricearea"] == price_area) & (df["productiongroup"] == productio
ts = data["quantitykwh"]
ts.index = pd.to_datetime(data["starttime"])
ts.sort_index(inplace=True) # Ensure time series is sorted by datetime

stl = STL(ts, period=period, seasonal=seasonal, trend=trend, robust=robust)
result = stl.fit()

# Plot
fig, ax = plt.subplots(4, 1, figsize=(15, 10), sharex=True)
ax[0].plot(ts, label='Original')
ax[0].set_title(f"{production_group} Production ({price_area}) - Original")
ax[0].legend()

ax[1].plot(result.trend, label='Trend', color='orange')
ax[1].set_title("Trend Component")
ax[1].legend()

ax[2].plot(result.seasonal, label='Seasonal', color='green')
ax[2].set_title("Seasonal Component")
ax[2].legend()

ax[3].plot(result.resid, label='Residual', color='red')
ax[3].set_title("Residual Component")
ax[3].legend()

plt.tight_layout()
```

```
plt.show()
```

```
return fig
```

```
In [84]: from pymongo.mongo_client import MongoClient
from pymongo.server_api import ServerApi
import tomllib

# Importing Elhub data from MongoDB

# 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'))
db = client["IND320_assignment_2"] # Creates or connects to database
collection = db["production_data"] # Creates or connects to collection

data = list(collection.find())

elhub_df = pd.DataFrame(data)
```

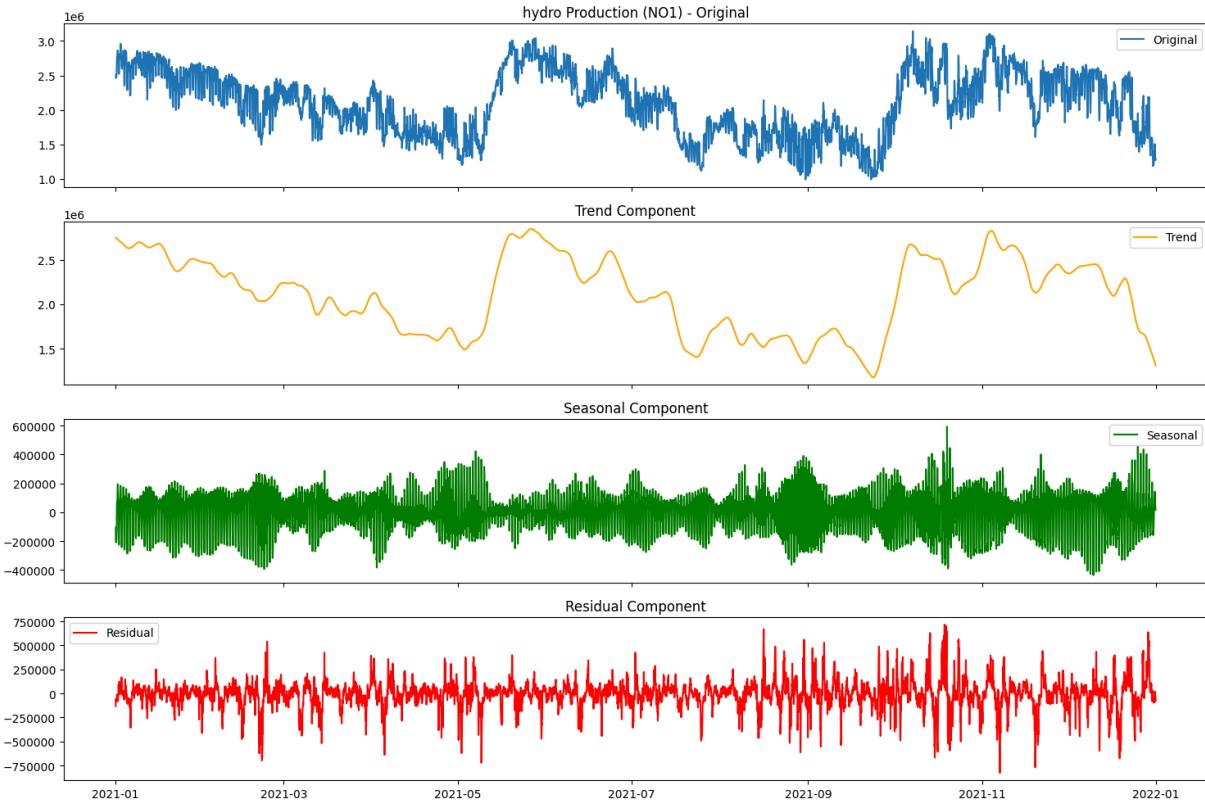
```
In [85]: elhub_df.head()
```

	_id	pricearea	productiongroup	starttime	quantitykwh
<b>0</b>	68efe05b7148e1cab990afa4	NO4	solar	2021-01-01 05:00:00	0.0
<b>1</b>	68efe05b7148e1cab990afa5	NO4	solar	2021-01-01 06:00:00	0.0
<b>2</b>	68efe05b7148e1cab990afab	NO4	solar	2021-01-01 12:00:00	0.0
<b>3</b>	68efe05b7148e1cab990afb7	NO4	solar	2021-01-02 00:00:00	0.0
<b>4</b>	68efe05b7148e1cab990afb8	NO4	solar	2021-01-02 01:00:00	0.0

```
In [86]: fig = stl_decomposition(elhub_df, price_area="NO1", production_group="hydro", perio
```

```
C:\Users\esksk\AppData\Local\Temp\ipykernel_6644\1964918440.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    ts.sort_index(inplace=True) # Ensure time series is sorted by datetime
```



## Spectrogram

```
In [98]: from scipy.signal import spectrogram

def plot_spectrogram(df, price_area="NO1", production_group="Solar", window_length=600,
                     overlap=300):
    data = df[(df["pricearea"] == price_area) & (df["productiongroup"] == production_group)]
    ts = data["quantitykwh"]
    ts.index = pd.to_datetime(data["starttime"])
    ts.sort_index(inplace=True) # Ensure time series is sorted by datetime

    # Compute spectrogram
    f, t, Sxx = spectrogram(ts, nperseg=window_length, noverlap=overlap)

    # Plotting
    fig, ax = plt.subplots(figsize=(12, 6))
    im = ax.pcolormesh(t, f, 10 * np.log10(Sxx), shading="gouraud", cmap="viridis")
    ax.set_ylabel("Frequency [1/hour]")
    ax.set_xlabel("Time [hours]")
    ax.set_title(f"Spectrogram of {production_group} Production ({price_area})")
    fig.colorbar(im, ax=ax, label="Power [dB]")
    plt.tight_layout()
```

```
return fig
```

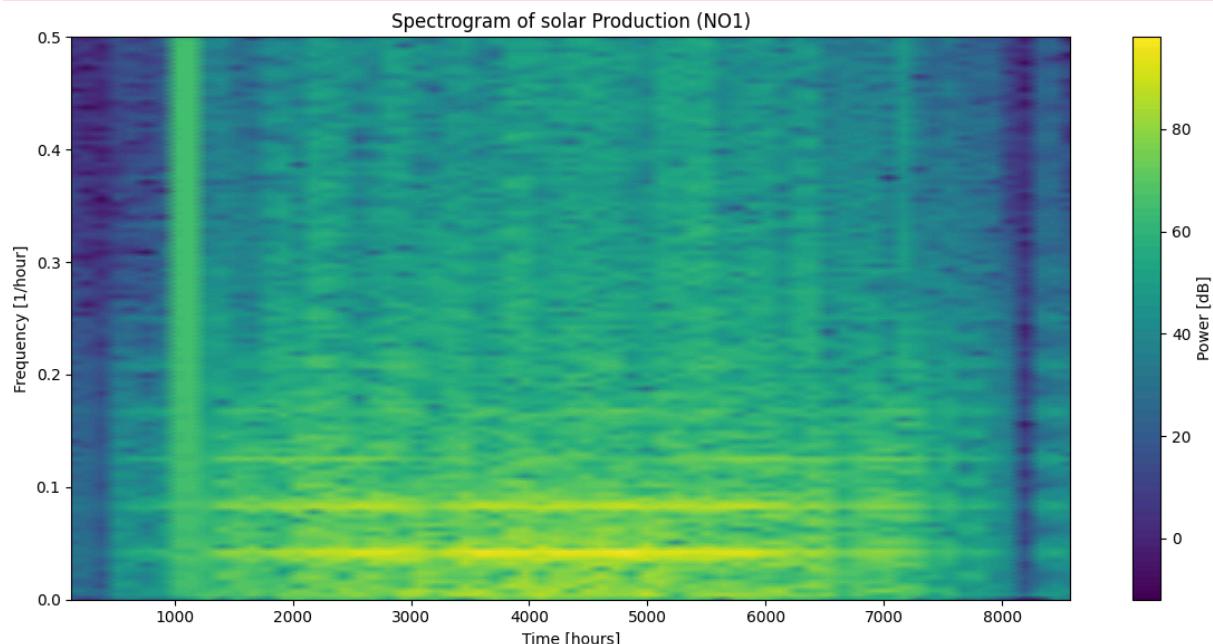
```
In [100]: fig = plot_spectrogram(elhub_df, price_area="NO1", production_group="solar", window
```

C:\Users\esksk\AppData\Local\Temp\ipykernel\_6644\3868019935.py:8: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
ts.sort_index(inplace=True) # Ensure time series is sorted by datetime
```



```
In [ ]:
```