In [ ]:
```
from entsoe import EntsoePandasClient
from tqdm import tqdm
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
```

Scrape data from ENTSOE API

- https://github.com/EnergieID/entsoe-py
- Day ahead prices
- Load
- Load Forecast
- Load and forecast
- Generation Forecast
- Wind and Solar Forecast
- Scheduled Exchanges
- Net Transfer Capacity

In [ ]:
```
# Define the EntsoePandasClient object with the API key
client = EntsoePandasClient(api_key='a6160036-4d49-4c39-960f-99c3c690b6da

# Define the time period and the country codes
# start = pd.Timestamp('20220601', tz='Europe/Copenhagen')
# end = pd.Timestamp('20220630', tz='Europe/Copenhagen')
start_date = pd.Timestamp('20220601 00:00:00', tz='Europe/Brussels').tz_c
end_date = pd.Timestamp('20220630 23:00:00', tz='Europe/Brussels').tz_con
country_codes = ['DK_1', 'DK_2', 'SE_1', 'SE_2', 'SE_3', 'SE_4', 'NO_1',
```

In [ ]:
```
# Create empty dataframes for storing the data
day_ahead_prices = pd.DataFrame()
scheduled_exchanges = pd.DataFrame()
net_transfer_capacity = pd.DataFrame()
load = pd.DataFrame()
load_forecast = pd.DataFrame()
load_and_forecast = pd.DataFrame()
generation_forecast = pd.DataFrame()
wind_and_solar_forecast = pd.DataFrame()
```

In [ ]:
```
# Loop over the country codes and query the data
for country_code in country_codes:
    # day ahead prices
    day_ahead = pd.DataFrame(client.query_day_ahead_prices(country_code,
    day_ahead['country_code'] = country_code
    day_ahead_prices = pd.concat([day_ahead_prices, day_ahead])
    # load
    ld = pd.DataFrame(client.query_load(country_code, start=start_date, e
```

```
    ld['country_code'] = country_code
    load = pd.concat([load, ld])
    # load forecast
    ld_fc = pd.DataFrame(client.query_load_forecast(country_code, start=s
    ld_fc['country_code'] = country_code
    load_forecast = pd.concat([load_forecast, ld_fc])
    # load and forecast
    ld_fc = pd.DataFrame(client.query_load_and_forecast(country_code, sta
    ld_fc['country_code'] = country_code
    load_and_forecast = pd.concat([load_and_forecast, ld_fc])
    # generation forecast
    gen_fc = pd.DataFrame(client.query_generation_forecast(country_code,
    gen_fc['country_code'] = country_code
    generation_forecast = pd.concat([generation_forecast, gen_fc])
    # wind and solar forecast
    ws_fc = pd.DataFrame(client.query_wind_and_solar_forecast(country_cod
    ws_fc['country_code'] = country_code
    wind_and_solar_forecast = pd.concat([wind_and_solar_forecast, ws_fc])
```

```
Connection Error, retrying in 10 seconds
Connection Error, retrying in 10 seconds
```

In [ ]:
```
data_frame_lists = ['day_ahead_prices', 'load', 'load_forecast', 'load_an
# save dataframes to csv
for data_frame_list in data_frame_lists:
    data = eval(data_frame_list)
    # adjust hour for finnish data as it is in UTC+2
    data_finnish = data[data['country_code'] == 'FI']
    data = data[data['country_code'] != 'FI']
    # subtract 1 hour from the index
    data_finnish.index = data_finnish.index - pd.DateOffset(hours=1)
    # Concat to the original dataframe
    data = pd.concat([data, data_finnish])
    # change the index to be a column
    # data['Timestamp'] = data.index
    # data.index = range(len(data))
    # data['Timestamp'] = pd.to_datetime(data['Timestamp'])
    data.to_csv(f'data/{data_frame_list}.csv')
```

In [ ]:
```
da = pd.read_csv('data/day_ahead_prices.csv')
# rename Unnamed: 0 to Timestamp
da.rename(columns={'Unnamed: 0': 'Timestamp'}, inplace=True)
# convert Timestamp to datetime
da_finnish = da[da['country_code'] == 'FI']
da = da[da['country_code'] != 'FI']
da['Timestamp'] = pd.to_datetime(da['Timestamp'])
da['Timestamp'] = da['Timestamp'].dt.tz_localize(None)
da_finnish['Timestamp'] = pd.to_datetime(da_finnish['Timestamp'])
da_finnish['Timestamp'] = da_finnish['Timestamp'].dt.tz_localize(None)
da = pd.concat([da, da_finnish])
da[da['country_code'] == 'FI'].head()
```

Out[ ]:

	Timestamp	0	country_code
7920	2022-06-01 00:00:00	10.00	FI
7921	2022-06-01 01:00:00	9.50	FI
7922	2022-06-01 02:00:00	9.00	FI
7923	2022-06-01 03:00:00	9.99	FI
7924	2022-06-01 04:00:00	11.59	FI

Download of cross border data

In [ ]:
```python
# Function to query cross-border flows
def queryWeekAheadCapacities(mapping_table, start_date, end_date):

    """
    Query cross-border flows from the ENTSO-E API based on a mapping tabl

    Parameters:
    - mapping_table (dict): A mapping table with key-value pairs represen
    - start_date (str): Start date for the query.
    - end_date (str): End date for the query.

    Returns:
    pd.DataFrame: DataFrame containing cross-border flow data with column

    """

    df_list = []
    for key, values in tqdm(mapping_table.items(), desc='Processing NTCs'
        for value in tqdm(values, desc=f'Processing NTC from {key}'):
            try:
                data_temp = client.query_net_transfer_capacity_weekahead(
                data_temp = data_temp.reset_index()
                data_temp['From'] = key
                data_temp['To'] = value
                df_list.append(data_temp)
            except:
                tqdm.write(f'No data for {key} -> {value}')

    print('Done')
    df_queried = pd.concat(df_list, ignore_index=True)
    df_queried = df_queried.rename(columns={0:'WeekAhead_NTC', 'index':'M

    return df_queried
```

In [ ]:
```python
# Function to convert daily granularity to hourly granularity for cross-b
def fromDailyToHourlyGranularity(NTCs, start_date, end_date):

    # Assuming 'index' is the index of your DataFrame
    # Convert 'index' to datetime if it's not already
```

```
    NTCs['MTU'] = pd.to_datetime(NTCs['MTU'])

    # Set 'index' as the index of your DataFrame
    NTCs.set_index('MTU', inplace=True)

    # Create a secondary index by combining 'From' and 'To'
    NTCs['secondary_index'] = NTCs['From'] + '_' + NTCs['To']

    NTCs_hourly = pd.DataFrame()
    for secondary in NTCs['secondary_index'].unique():
        selection = NTCs.loc[NTCs['secondary_index'] == secondary]

        new_index = pd.date_range(start=start_date, end=end_date, freq='h
        selection = selection.reindex(new_index)

        # Reset the index to get 'dateTimeUtc' back as a column
        #selection.reset_index(inplace=True)

        # Forward fill to propagate values for new timestamps
        selection = selection.ffill()

        # concat the result to new_table
        NTCs_hourly = pd.concat([NTCs_hourly, selection])

    # drop the secondary index
    NTCs_hourly.drop('secondary_index', axis=1, inplace=True)

    # rename From and To to biddingZoneFrom and biddingZoneTo
    NTCs_hourly.rename(columns={'From':'biddingZoneFrom', 'To':'biddingZo
    NTCs_hourly.index.rename('MTU', inplace=True)

    return NTCs_hourly
```

In [ ]:
```
# Function to query scheduled exchanges
def query_scheduled_exchanges_ENTSOE(mapping_table, start_date, end_date)

    """
    Query cross-border flows from the ENTSO-E API based on a mapping tabl

    Parameters:
    - mapping_table (dict): A mapping table with key-value pairs represen
    - start_date (str): Start date for the query.
    - end_date (str): End date for the query.

    Returns:
    pd.DataFrame: DataFrame containing cross-border flow data with column

    """

    df_list = []
    for key, values in tqdm(mapping_table.items(), desc='Processing sched
        for value in tqdm(values, desc=f'Processing scheduled exchanges f
            try:
```

```
                data_temp = client.query_scheduled_exchanges(key, value,
                data_temp = data_temp.reset_index()
                data_temp['From'] = key
                data_temp['To'] = value
                df_list.append(data_temp)
            except:
                tqdm.write(f'No data for {key} -> {value}')

    print('Done')
    df_queried = pd.concat(df_list, ignore_index=True)
    df_queried = df_queried.rename(columns={0:'Sch_Exchange', 'index':'MT

    return df_queried
```

In [ ]:

```
# Function to achieve hourly granularity of scheduled exchanges
def achieve_hourly_granularity(df):

    """
    Process a DataFrame to achieve hourly granularity of flow data.

    Parameters:
    - df (pd.DataFrame): Input DataFrame containing flow data with column
                         Here, some MTU may be of 15 minutes granularity,

    Returns:
    pd.DataFrame: Processed DataFrame with hourly granularity, where the
                  for each hour between 'From' and 'To'.

    Notes:
    - The 'MTU' column is converted to UTC datetime.
    - New columns 'Hour' and 'Date' are created to store the hour and dat
    - The DataFrame is then grouped by 'Date', 'Hour', 'From', and 'To' t
    - The 'Date' column is adjusted to represent the midpoint of each hou
    - The 'Hour' column is dropped from the final result.

    """

    # Convert 'MTU' column to UTC datetime
    df['MTU'] = pd.to_datetime(df['MTU'], utc=True)

    # Create a new column 'Hour' to store the hour information
    df['Hour'] = df['MTU'].dt.hour

    # Create a new column 'Date' to store the date information
    df['Date'] = df['MTU'].dt.date

    # Group by 'Date', 'Hour', 'From', and 'To' and calculate the average
    df = df.groupby(['Date', 'Hour', 'From', 'To'])['Sch_Exchange'].mean(

    # Convert 'index' column to datetime
    df['Date'] = pd.to_datetime(df['Date'])

    df['Date'] = df['Date'] + pd.to_timedelta(df['Hour'].astype(str) + ':
```

```
    df.drop(columns=['Hour'], inplace=True)

    df.rename(columns={'Date': 'MTU'}, inplace=True)

    return df
```

In [ ]:
```
# Function to define cross border connections and run the queries
def main_border_queries(start_date, end_date):

    ######################
    ### MAPPING TABLE ###
    ######################

    NEIGHBOURS_Real_System = {
        'NL': ['NO_2', 'DK_1'],
        'DE_AT_LU': ['DK_1', 'DK_2', 'SE_4'],
        'GB': ['NO_2','DK_1'],
        'NO_2': ['DE_LU', 'DK_1', 'NL', 'NO_1', 'NO_5', 'GB'],
        'PL': ['SE_4'],
        'DK_1': ['DE_AT_LU', 'DE_LU', 'DK_2', 'NO_2', 'SE_3', 'NL','GB'],
        'LT': ['SE_4'],
        'SE_3': ['DK_1', 'FI', 'NO_1', 'SE_2', 'SE_4'],
        'NO_1': ['NO_2', 'NO_3', 'NO_5', 'SE_3'],
        'SE_4': ['DE_AT_LU', 'DE_LU', 'DK_2', 'LT', 'PL', 'SE_3'],
        'NO_5': ['NO_1', 'NO_2', 'NO_3'],
        'EE': ['FI'],
        'DK_2': ['DE_AT_LU', 'DE_LU', 'DK_1', 'SE_4'],
        'FI': ['EE', 'NO_4', 'RU', 'SE_1', 'SE_3'],
        'NO_4': ['SE_2', 'FI', 'NO_3', 'SE_1'],
        'SE_1': ['FI', 'NO_4', 'SE_2'],
        'SE_2': ['NO_3', 'NO_4', 'SE_1', 'SE_3'],
        'DE_LU': ['DK_1', 'DK_2', 'NO_2', 'SE_4'],
        'NO_3': ['NO_1', 'NO_4', 'NO_5', 'SE_2']
    }

    NTCs = queryWeekAheadCapacities(NEIGHBOURS_Real_System, start_date, e
    NTCs = fromDailyToHourlyGranularity(NTCs, start_date, end_date)

    # Query cross-border scheduled exchanges from ENTSO-E
    SchExch = query_scheduled_exchanges_ENTSOE(NEIGHBOURS_Real_System, st
    SchExch_H = achieve_hourly_granularity(SchExch)

    return NTCs, SchExch, SchExch_H
```

In [ ]:
```
NTCs, SchExch, SchExch_H = main_border_queries(start_date, end_date)

# Putting the cross border exchange data in the right format
NTCs_pivot = NTCs.copy(deep=True).reset_index()
NTCs_pivot = NTCs_pivot.pivot_table(index=['MTU', 'biddingZoneFrom'], col
NTCs_pivot = NTCs_pivot.rename(columns={col: f'Cap_to_{col}' if col != 'M
NTCs_pivot = NTCs_pivot.fillna(0)
NTCs_pivot.rename(columns={'biddingZoneFrom': 'From'}, inplace=True)
```

```
# Scheduled Hourly Export
SchExch_H_Ex = SchExch_H.copy(deep=True)
SchExch_H_Ex = SchExch_H_Ex.pivot_table(index=['MTU', 'From'], columns='T
SchExch_H_Ex = SchExch_H_Ex.rename(columns={col: f'Ex_to_{col}' if col !=
SchExch_H_Ex = SchExch_H_Ex.fillna(0)

# Scheduled Hourly Import
SchExch_H_Imp = SchExch_H.copy(deep=True)
SchExch_H_Imp['From'], SchExch_H_Imp['To'] = SchExch_H_Imp['To'], SchExch
SchExch_H_Imp = SchExch_H_Imp.pivot_table(index=['MTU', 'From'], columns=
SchExch_H_Imp = SchExch_H_Imp.rename(columns={col: f'Imp_from_{col}' if c
SchExch_H_Imp = SchExch_H_Imp.fillna(0)

# Scheduled exchanges merge
Border_data = SchExch_H_Ex.merge(SchExch_H_Imp, on=['MTU', 'From'], how='
Border_data['MTU'] = pd.to_datetime(Border_data['MTU']).dt.tz_localize('U
Border_data = Border_data.merge(NTCs_pivot, on=['MTU', 'From'], how='left
```

```
Processing NTC from NL: 100%|██████████| 2/2 [00:04<00:00,  2.28s/it]
Processing NTCs:   5%|▌         | 1/19 [00:04<01:22,  4.61s/it]
Processing NTCs:   5%|▌         | 1/19 [00:05<01:22,  4.61s/it]
No data for DE_AT_LU -> DK_1
Processing NTCs:   5%|▌         | 1/19 [00:08<01:22,  4.61s/it]
Processing NTC from DE_AT_LU: 100%|██████████| 3/3 [00:04<00:00,  1.49s/i
t]
Processing NTCs:  11%|█         | 2/19 [00:09<01:17,  4.54s/it]
No data for DE_AT_LU -> DK_2
No data for DE_AT_LU -> SE_4
Processing NTC from GB: 100%|██████████| 2/2 [00:00<00:00,  2.49it/s]
Processing NTCs:  16%|█▌        | 3/19 [00:09<00:45,  2.84s/it]
No data for GB -> DK_1
Connection Error, retrying in 10 seconds
Connection Error, retrying in 10 seconds
Processing NTC from NO_2: 100%|██████████| 6/6 [01:53<00:00, 18.93s/it]
Processing NTCs:  21%|██        | 4/19 [02:03<11:38, 46.55s/it]Connection
Error, retrying in 10 seconds
Processing NTC from PL: 100%|██████████| 1/1 [00:11<00:00, 11.18s/it]
Processing NTCs:  26%|██▋       | 5/19 [02:14<07:53, 33.80s/it]
Processing NTCs:  26%|██▋       | 5/19 [02:15<07:53, 33.80s/it]
No data for DK_1 -> DE_AT_LU
Processing NTC from DK_1: 100%|██████████| 7/7 [00:07<00:00,  1.01s/it]
Processing NTCs:  32%|███▏      | 6/19 [02:21<05:21, 24.71s/it]
No data for DK_1 -> GB
Processing NTC from LT: 100%|██████████| 1/1 [00:00<00:00,  1.43it/s]
Processing NTC from SE_3: 100%|██████████| 5/5 [00:05<00:00,  1.06s/it]
Processing NTC from NO_1: 100%|██████████| 4/4 [00:04<00:00,  1.01s/it]
Processing NTCs:  47%|████▋     | 9/19 [02:31<01:43, 10.32s/it]
Processing NTCs:  47%|████▋     | 9/19 [02:32<01:43, 10.32s/it]
No data for SE_4 -> DE_AT_LU
```

```
Processing NTC from SE_4: 100%|██████████| 6/6 [00:12<00:00,  2.02s/it]
Processing NTC from NO_5: 100%|██████████| 3/3 [00:03<00:00,  1.29s/it]
Processing NTC from EE: 100%|██████████| 1/1 [00:02<00:00,  2.35s/it]
Processing NTCs:  63%|██████    | 12/19 [02:50<00:47,  6.79s/it]
Processing NTCs:  63%|██████    | 12/19 [02:51<00:47,  6.79s/it]
No data for DK_2 -> DE_AT_LU
Processing NTC from DK_2: 100%|██████████| 4/4 [00:06<00:00,  1.59s/it]
Processing NTCs:  68%|███████   | 13/19 [02:56<00:39,  6.66s/it]
Processing NTCs:  68%|███████   | 13/19 [02:59<00:39,  6.66s/it]
No data for FI -> NO_4
Processing NTC from FI: 100%|██████████| 5/5 [00:17<00:00,  3.44s/it]
Processing NTCs:  74%|███████   | 14/19 [03:13<00:49,  9.84s/it]
Processing NTCs:  74%|███████   | 14/19 [03:17<00:49,  9.84s/it]
No data for NO_4 -> FI
Processing NTC from NO_4: 100%|██████████| 4/4 [00:06<00:00,  1.56s/it]
Processing NTC from SE_1: 100%|██████████| 3/3 [00:07<00:00,  2.66s/it]
Processing NTC from SE_2: 100%|██████████| 4/4 [00:10<00:00,  2.61s/it]
Processing NTC from DE_LU: 100%|██████████| 4/4 [00:07<00:00,  1.81s/it]
Processing NTC from NO_3: 100%|██████████| 4/4 [00:06<00:00,  1.74s/it]
Processing NTCs: 100%|██████████| 19/19 [03:52<00:00, 12.24s/it]
Done
Processing scheduled exchanges from NL: 100%|██████████| 2/2 [00:04<00:00,
2.01s/it]
Processing scheduled exchanges:   5%|▌         | 1/19 [00:04<01:12,  4.03
s/it]
Processing scheduled exchanges:   5%|▌         | 1/19 [00:05<01:12,  4.03
s/it]
No data for DE_AT_LU -> DK_1
Processing scheduled exchanges:   5%|▌         | 1/19 [00:06<01:12,  4.03
s/it]
No data for DE_AT_LU -> DK_2
Processing scheduled exchanges from DE_AT_LU: 100%|██████████| 3/3 [00:02<
00:00,  1.04it/s]
Processing scheduled exchanges:  11%|█         | 2/19 [00:06<00:57,  3.37
s/it]
No data for DE_AT_LU -> SE_4
Processing scheduled exchanges from GB: 100%|██████████| 2/2 [00:09<00:00,
4.79s/it]
Processing scheduled exchanges:  16%|█▌        | 3/19 [00:16<01:39,  6.22
s/it]
No data for GB -> DK_1
Processing scheduled exchanges from NO_2: 100%|██████████| 6/6 [00:20<00:0
0,  3.38s/it]
Processing scheduled exchanges from PL: 100%|██████████| 1/1 [00:02<00:00,
2.08s/it]
Processing scheduled exchanges:  26%|██▋       | 5/19 [00:38<01:56,  8.29
s/it]
Processing scheduled exchanges:  26%|██▋       | 5/19 [00:41<01:56,  8.29
s/it]
No data for DK_1 -> DE_AT_LU
```

```
Processing scheduled exchanges from DK_1: 100%|██████████| 7/7 [00:13<00:0
0,  1.98s/it]
Processing scheduled exchanges:  32%|███       | 6/19 [00:52<02:12, 10.18
s/it]
```

No data for DK_1 -> GB

```
Processing scheduled exchanges from LT: 100%|██████████| 1/1 [00:02<00:00,
2.37s/it]
Processing scheduled exchanges from SE_3: 100%|██████████| 5/5 [00:09<00:0
0,  1.85s/it]
Processing scheduled exchanges from NO_1: 100%|██████████| 4/4 [00:09<00:0
0,  2.47s/it]
Processing scheduled exchanges:  47%|████      | 9/19 [01:14<01:26,  8.68
s/it]
Processing scheduled exchanges:  47%|████      | 9/19 [01:14<01:26,  8.68
s/it]
```

No data for SE_4 -> DE_AT_LU

```
Processing scheduled exchanges from SE_4: 100%|██████████| 6/6 [00:14<00:0
0,  2.40s/it]
Processing scheduled exchanges from NO_5: 100%|██████████| 3/3 [00:14<00:0
0,  4.70s/it]
Processing scheduled exchanges from EE: 100%|██████████| 1/1 [00:04<00:00,
4.61s/it]
Processing scheduled exchanges:  63%|██████    | 12/19 [01:47<01:06,  9.45
s/it]
Processing scheduled exchanges:  63%|██████    | 12/19 [01:48<01:06,  9.45
s/it]
```

No data for DK_2 -> DE_AT_LU

```
Processing scheduled exchanges from DK_2: 100%|██████████| 4/4 [00:09<00:0
0,  2.34s/it]
Processing scheduled exchanges:  68%|██████    | 13/19 [01:56<00:56,  9.43
s/it]
Processing scheduled exchanges:  68%|██████    | 13/19 [02:02<00:56,  9.43
s/it]
```

No data for FI -> RU

```
Processing scheduled exchanges from FI: 100%|██████████| 5/5 [00:16<00:00,
3.31s/it]
Processing scheduled exchanges from NO_4: 100%|██████████| 4/4 [00:08<00:0
0,  2.19s/it]
Processing scheduled exchanges from SE_1: 100%|██████████| 3/3 [00:13<00:0
0,  4.51s/it]
Processing scheduled exchanges:  84%|████████  | 16/19 [02:35<00:34, 11.57
s/it]Connection Error, retrying in 10 seconds
Processing scheduled exchanges from SE_2: 100%|██████████| 4/4 [00:28<00:0
0,  7.01s/it]
Processing scheduled exchanges:  89%|████████▉ | 17/19 [03:03<00:33, 16.53
s/it]Connection Error, retrying in 10 seconds
Processing scheduled exchanges from DE_LU: 100%|██████████| 4/4 [00:18<00:
00,  4.69s/it]
Processing scheduled exchanges from NO_3: 100%|██████████| 4/4 [00:18<00:0
0,  4.63s/it]
Processing scheduled exchanges: 100%|██████████| 19/19 [03:40<00:00, 11.63
s/it]
```

Done

In [ ]:
```
# convert MTU to timezone "Europe/Brussels" for all From, besides From =
Border_data['MTU'] = pd.to_datetime(Border_data['MTU']).dt.tz_convert('Eu
# save as csv
Border_data.to_csv('data/Border_data.csv')
```

In [ ]:
```
da = pd.read_csv('data/day_ahead_prices.csv')
```

Out[ ]:

	Unnamed: 0	0	country_code	Timestamp
0	0	220.00	DK_1	2022-06-01 00:00:00+02:00
1	1	207.45	DK_1	2022-06-01 01:00:00+02:00
2	2	199.09	DK_1	2022-06-01 02:00:00+02:00
3	3	182.53	DK_1	2022-06-01 03:00:00+02:00
4	4	182.28	DK_1	2022-06-01 04:00:00+02:00

Merge datasets to a complete final dataframe

In [ ]:
```
# add Border_data to data_frame_lists
data_frame_lists = ['day_ahead_prices', 'load_and_forecast', 'generation_
work_dir = os.getcwd()

# import data using os library
data = {}
for data_frame in data_frame_lists:
    data[data_frame] = pd.read_csv(os.path.join(work_dir + '/data', data_

# make dataframe data_df from the dict for each data_frame
day_ahead_prices = data['day_ahead_prices']
day_ahead_prices.rename(columns={'0': 'DA-price [EUR/MWh]'}, inplace=True

load_and_forecast = data['load_and_forecast']

generation_forecast = data['generation_forecast']
generation_forecast.rename(columns={'Actual Aggregated': 'Forecasted Gene

wind_and_solar_forecast = data['wind_and_solar_forecast']

cross_border_data = data['Border_data']
cross_border_data.rename(columns={'MTU': 'Timestamp', 'From': 'country_co
cross_border_data = cross_border_data.drop(columns=['Unnamed: 0'])
cross_border_data['Timestamp'] = pd.to_datetime(cross_border_data['Timest
cross_border_data['Timestamp'] = cross_border_data['Timestamp'].dt.tz_loc

#merge dataframes based on the Unnamed: 0 column and country_code column
data_df = pd.merge(day_ahead_prices, load_and_forecast, on=['Unnamed: 0',
data_df = pd.merge(data_df, generation_forecast, on=['Unnamed: 0', 'count
```

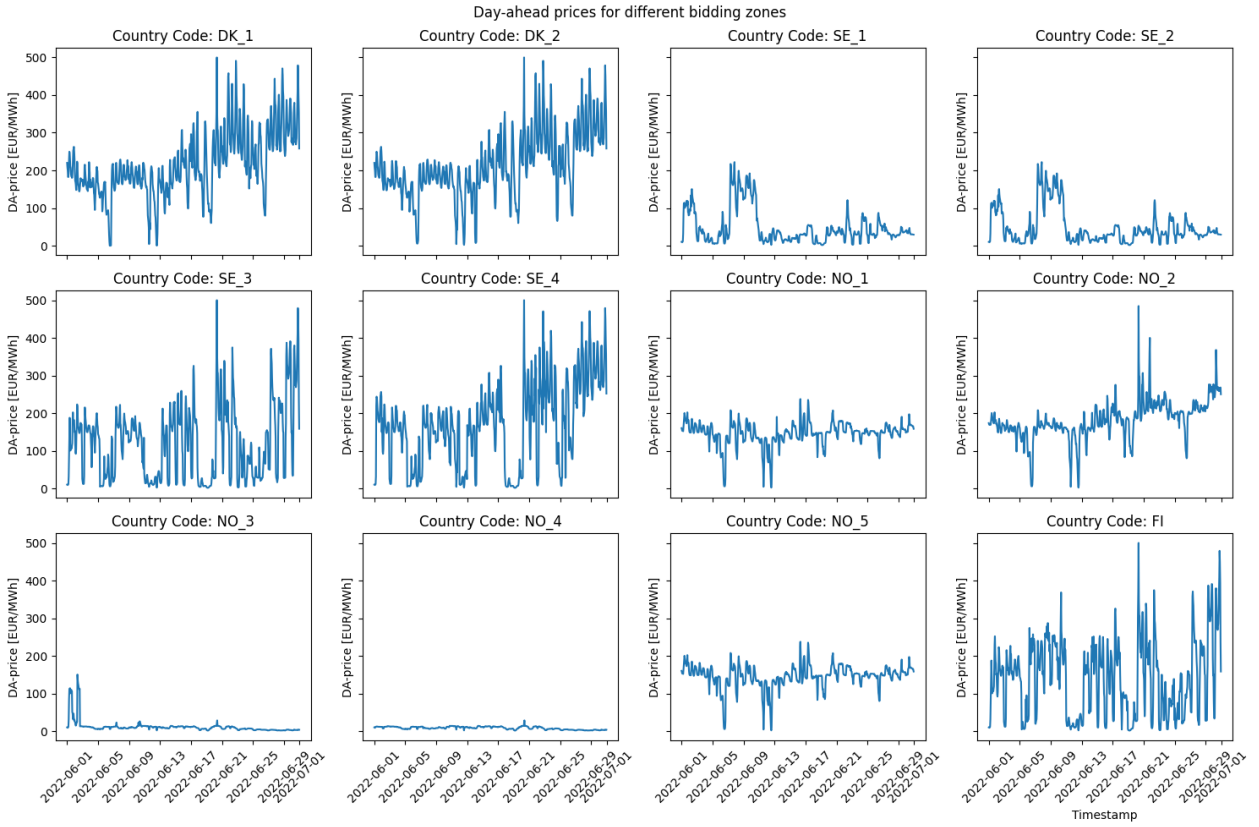
```
data_df = pd.merge(data_df, wind_and_solar_forecast, on=['Unnamed: 0', 'c

# Separate the finnish data and adjust the timezone
data_df.rename(columns={'Unnamed: 0': 'Timestamp'}, inplace=True)
data_finnish = data_df[data_df['country_code'] == 'FI']
data_df = data_df[data_df['country_code'] != 'FI']
data_df['Timestamp'] = pd.to_datetime(data_df['Timestamp'])
data_df['Timestamp'] = data_df['Timestamp'].dt.tz_localize(None)
data_finnish['Timestamp'] = pd.to_datetime(data_finnish['Timestamp'])
data_finnish['Timestamp'] = data_finnish['Timestamp'].dt.tz_localize(None
data_df = pd.concat([data_df, data_finnish])

# merge the cross border data
data_df = pd.merge(data_df, cross_border_data, on=['Timestamp', 'country_

#save the data to csv
data_df.to_csv('data/nordic_energy_data.csv')
```

```
(8640, 64)
```

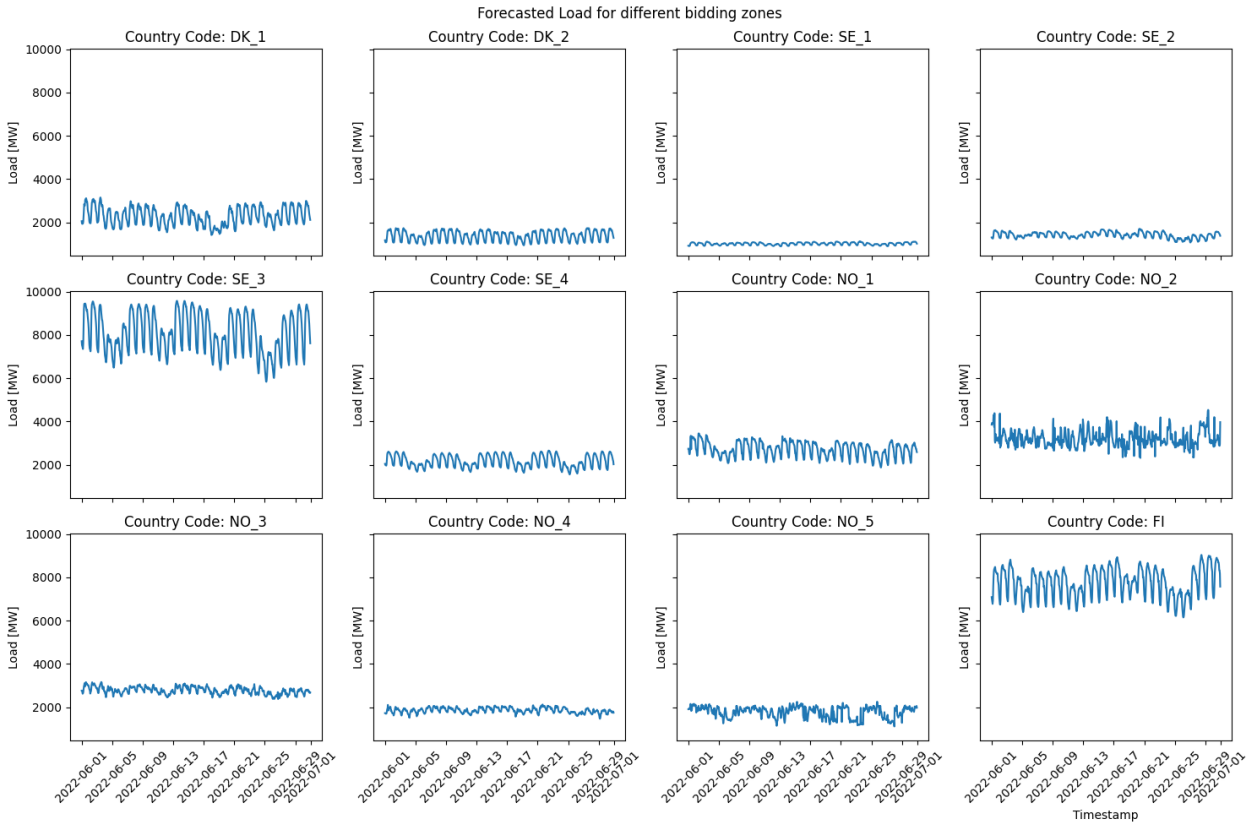
Exploratory Data analysis

In [ ]:

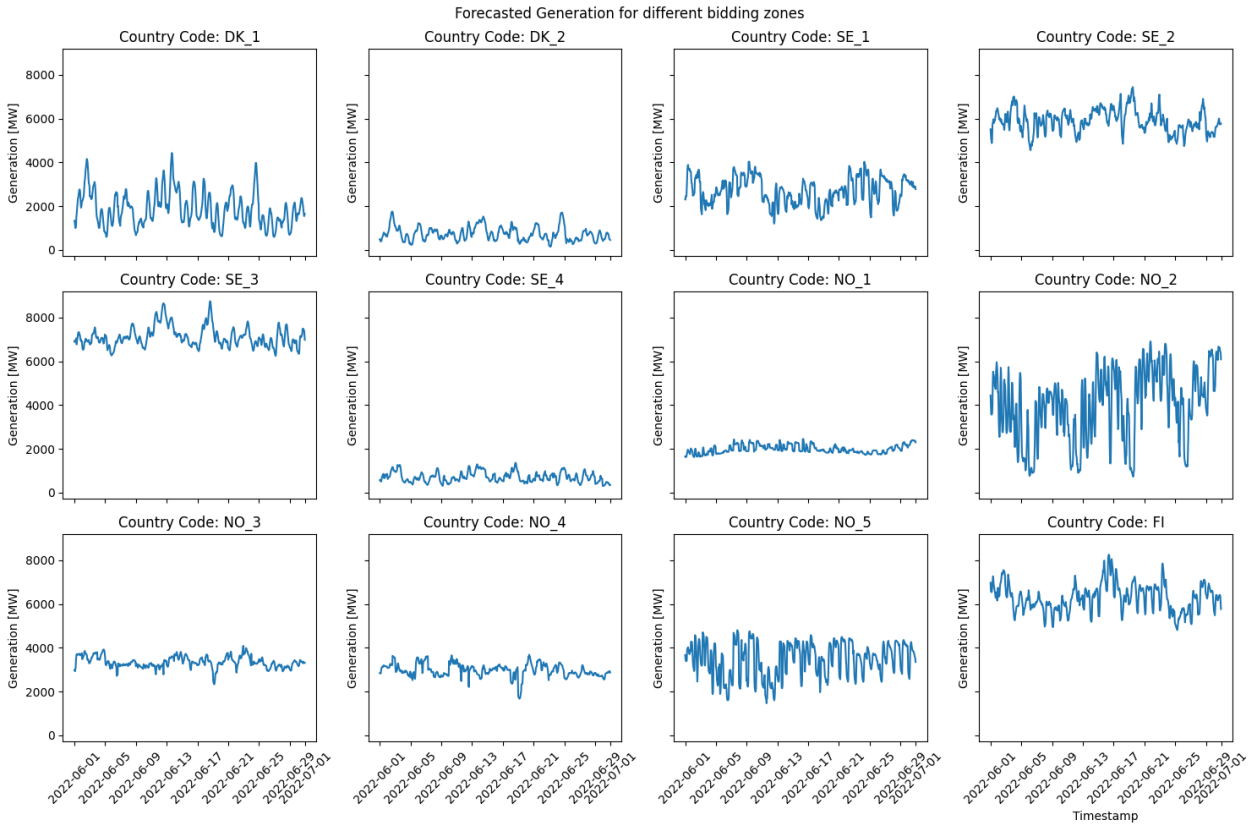
```
# Get unique country codes
unique_country_codes = data_df['country_code'].unique()

# Create subplots for each country code in a 3x4 grid
fig, axs = plt.subplots(3, 4, figsize=(15, 10), sharex=True, sharey=True)

# Flatten axs array for easier iteration
axs = axs.flatten()

# Plot DA-prices with the country_code as labels
for i, country_code in enumerate(unique_country_codes):
    data_df_country = data_df[data_df['country_code'] == country_code]
    axs[i].plot(data_df_country['Timestamp'], data_df_country['DA-price [
    axs[i].set_ylabel('DA-price [EUR/MWh]')
    axs[i].set_title(f'Country Code: {country_code}')
    axs[i].tick_params(axis='x', rotation=45)


# Set common x-label
plt.xlabel('Timestamp')
# make overall title
plt.suptitle('Day-ahead prices for different bidding zones')
#rotate x-labels for all subplots

# Adjust layout
plt.tight_layout()

# Show plot
plt.savefig('plots/DA-prices.png')
plt.show()
```



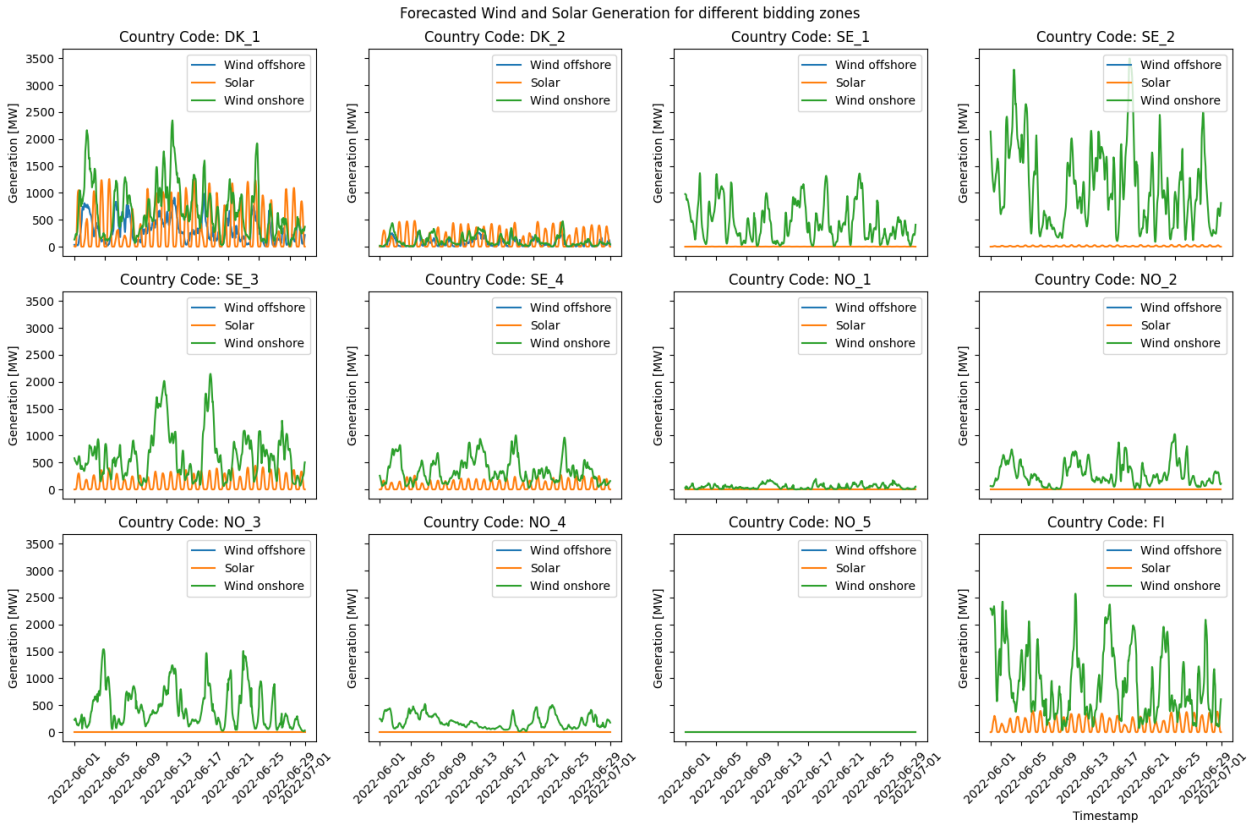
In [ ]:
```
#%%
#make similar plots for the other dataframes
#load
fig, axs = plt.subplots(3, 4, figsize=(15, 10), sharex=True, sharey=True)
axs = axs.flatten()
for i, country_code in enumerate(unique_country_codes):
    data_df_country = data_df[data_df['country_code'] == country_code]
    axs[i].plot(data_df_country['Timestamp'], data_df_country['Forecasted
    axs[i].set_ylabel('Load [MW]')
    axs[i].set_title(f'Country Code: {country_code}')
    axs[i].tick_params(axis='x', rotation=45)

plt.xlabel('Timestamp')
plt.suptitle('Forecasted Load for different bidding zones')

plt.tight_layout()
plt.savefig('plots/Forecasted_Load.png')
plt.show()
```



In [ ]:
```
#%%
#generation forecast
fig, axs = plt.subplots(3, 4, figsize=(15, 10), sharex=True, sharey=True)
axs = axs.flatten()
for i, country_code in enumerate(unique_country_codes):
    data_df_country = data_df[data_df['country_code'] == country_code]
    axs[i].plot(data_df_country['Timestamp'], data_df_country['Forecasted
    axs[i].set_ylabel('Generation [MW]')
    axs[i].set_title(f'Country Code: {country_code}')
    axs[i].tick_params(axis='x', rotation=45)

plt.xlabel('Timestamp')
plt.suptitle('Forecasted Generation for different bidding zones')

plt.tight_layout()
plt.savefig('plots/Forecasted_Generation.png')
plt.show()
```

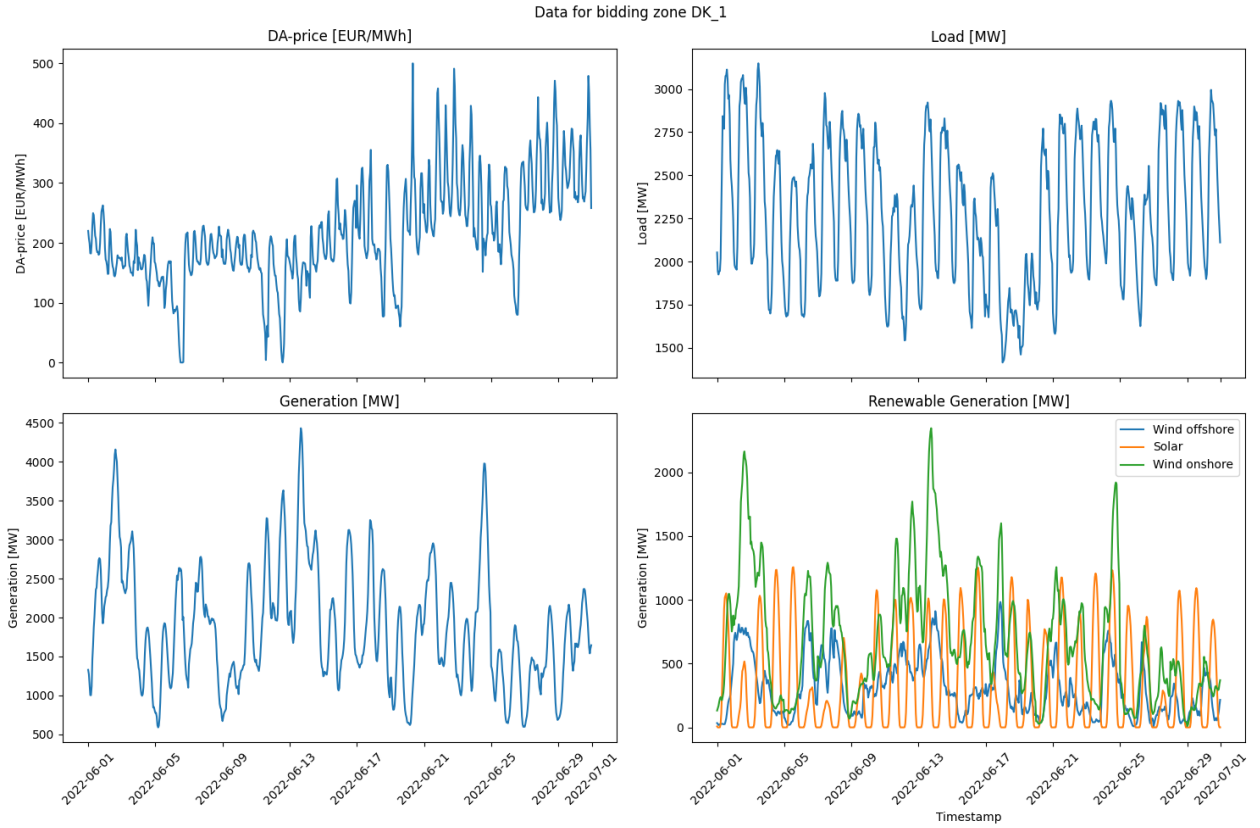


In [ ]:
```
#%%
#wind and solar forecast
fig, axs = plt.subplots(3, 4, figsize=(15, 10), sharex=True, sharey=True)
axs = axs.flatten()
for i, country_code in enumerate(unique_country_codes):
    data_df_country = data_df[data_df['country_code'] == country_code]
    axs[i].plot(data_df_country['Timestamp'], data_df_country['Wind Offsh
    axs[i].plot(data_df_country['Timestamp'], data_df_country['Solar'], l
    axs[i].plot(data_df_country['Timestamp'], data_df_country['Wind Onsho
    axs[i].set_ylabel('Generation [MW]')
    axs[i].set_title(f'Country Code: {country_code}')
    axs[i].tick_params(axis='x', rotation=45)
    axs[i].legend()

plt.xlabel('Timestamp')
plt.suptitle('Forecasted Wind and Solar Generation for different bidding

plt.tight_layout()
plt.savefig('plots/Forecasted_Wind_Solar.png')
plt.show()
```



In [ ]:
```
#%%
# for DK_1, plot day-ahead prices, load, forecasted generation, wind and
fig, axs = plt.subplots(2, 2, figsize=(15, 10), sharex=True, sharey=False
axs = axs.flatten()
country_code = 'DK_1'
data_df_country = data_df[data_df['country_code'] == country_code]
axs[0].plot(data_df_country['Timestamp'], data_df_country['DA-price [EUR/
axs[0].set_ylabel('DA-price [EUR/MWh]')
axs[0].set_title('DA-price [EUR/MWh]')
axs[0].tick_params(axis='x', rotation=45)

axs[1].plot(data_df_country['Timestamp'], data_df_country['Forecasted Loa
axs[1].set_ylabel('Load [MW]')
axs[1].set_title('Load [MW]')
axs[1].tick_params(axis='x', rotation=45)

axs[2].plot(data_df_country['Timestamp'], data_df_country['Forecasted Gen
axs[2].set_ylabel('Generation [MW]')
axs[2].set_title('Generation [MW]')
axs[2].tick_params(axis='x', rotation=45)

axs[3].plot(data_df_country['Timestamp'], data_df_country['Wind Offshore'
axs[3].plot(data_df_country['Timestamp'], data_df_country['Solar'], label
axs[3].plot(data_df_country['Timestamp'], data_df_country['Wind Onshore']
axs[3].set_ylabel('Generation [MW]')
axs[3].set_title('Renewable Generation [MW]')
axs[3].tick_params(axis='x', rotation=45)
axs[3].legend()
```

```
plt.xlabel('Timestamp')
plt.suptitle('Data for bidding zone DK_1')

plt.tight_layout()
plt.savefig('plots/DK_1_data.png')
plt.show()
```



In [ ]: