

```
In [2]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

```
In [3]: df = pd.read_csv("Electricity.csv") Code + Data :https://t.me/AlMLDeepThought/555
df.head().style.set_properties(
    **{
        'background-color': 'OliveDrab',
        'color': 'white',
        'border-color': 'darkblack'
    })
```

Out[3]:

|   | Datetime            | Consumption | Production | Nuclear | Wind | Hydroelectric | Oil and Gas | Coal | Solar | Biomass |
|---|---------------------|-------------|------------|---------|------|---------------|-------------|------|-------|---------|
| 0 | 2019-01-01 00:00:00 | 6352        | 6527       | 1395    | 79   | 1383          | 1896        | 1744 | 0     | 30      |
| 1 | 2019-01-01 01:00:00 | 6116        | 5701       | 1393    | 96   | 1112          | 1429        | 1641 | 0     | 30      |
| 2 | 2019-01-01 02:00:00 | 5873        | 5676       | 1393    | 142  | 1030          | 1465        | 1616 | 0     | 30      |
| 3 | 2019-01-01 03:00:00 | 5682        | 5603       | 1397    | 191  | 972           | 1455        | 1558 | 0     | 30      |
| 4 | 2019-01-01 04:00:00 | 5557        | 5454       | 1393    | 159  | 960           | 1454        | 1458 | 0     | 30      |

```
In [4]: df['DateTime'] = pd.to_datetime(df['DateTime'])
```

```
In [5]: df.describe().style.background_gradient(cmap='rainbow')
```

Out[5]:

|       | Datetime                      | Consumption  | Production   | Nuclear      | Wind         | Hydroelectric | Oil and Gas  | Coal         | Solar        |
|-------|-------------------------------|--------------|--------------|--------------|--------------|---------------|--------------|--------------|--------------|
| count | 46011                         | 46011.000000 | 46011.000000 | 46011.000000 | 46011.000000 | 46011.000000  | 46011.000000 | 46011.000000 | 46011.000000 |
| mean  | 2021-08-16 11:19:47.715981056 | 6587.616440  | 6518.645628  | 1291.177501  | 792.310882   | 1857.052444   | 1171.890418  | 1193.157332  | 156.688031   |
| min   | 2019-01-01 00:00:00           | 3889.000000  | 3315.000000  | 562.000000   | -26.000000   | 175.000000    | 198.000000   | 279.000000   | 0.000000     |
| 25%   | 2020-04-24 06:30:00           | 5773.000000  | 5814.000000  | 1347.000000  | 236.000000   | 1347.000000   | 858.000000   | 962.000000   | 0.000000     |
| 50%   | 2021-08-16 12:00:00           | 6552.000000  | 6462.000000  | 1383.000000  | 592.000000   | 1747.000000   | 1211.000000  | 1172.000000  | 2.000000     |
| 75%   | 2022-12-08 15:30:00           | 7321.000000  | 7176.000000  | 1405.000000  | 1205.000000  | 2265.000000   | 1511.000000  | 1406.000000  | 280.000000   |
| max   | 2024-03-31 23:00:00           | 9615.000000  | 9886.000000  | 1457.000000  | 2811.000000  | 4434.000000   | 2141.000000  | 2537.000000  | 1137.000000  |
| std   | nan                           | 1043.654923  | 986.805018   | 236.549637   | 675.812712   | 692.592157    | 434.748917   | 320.449368   | 229.502650   |

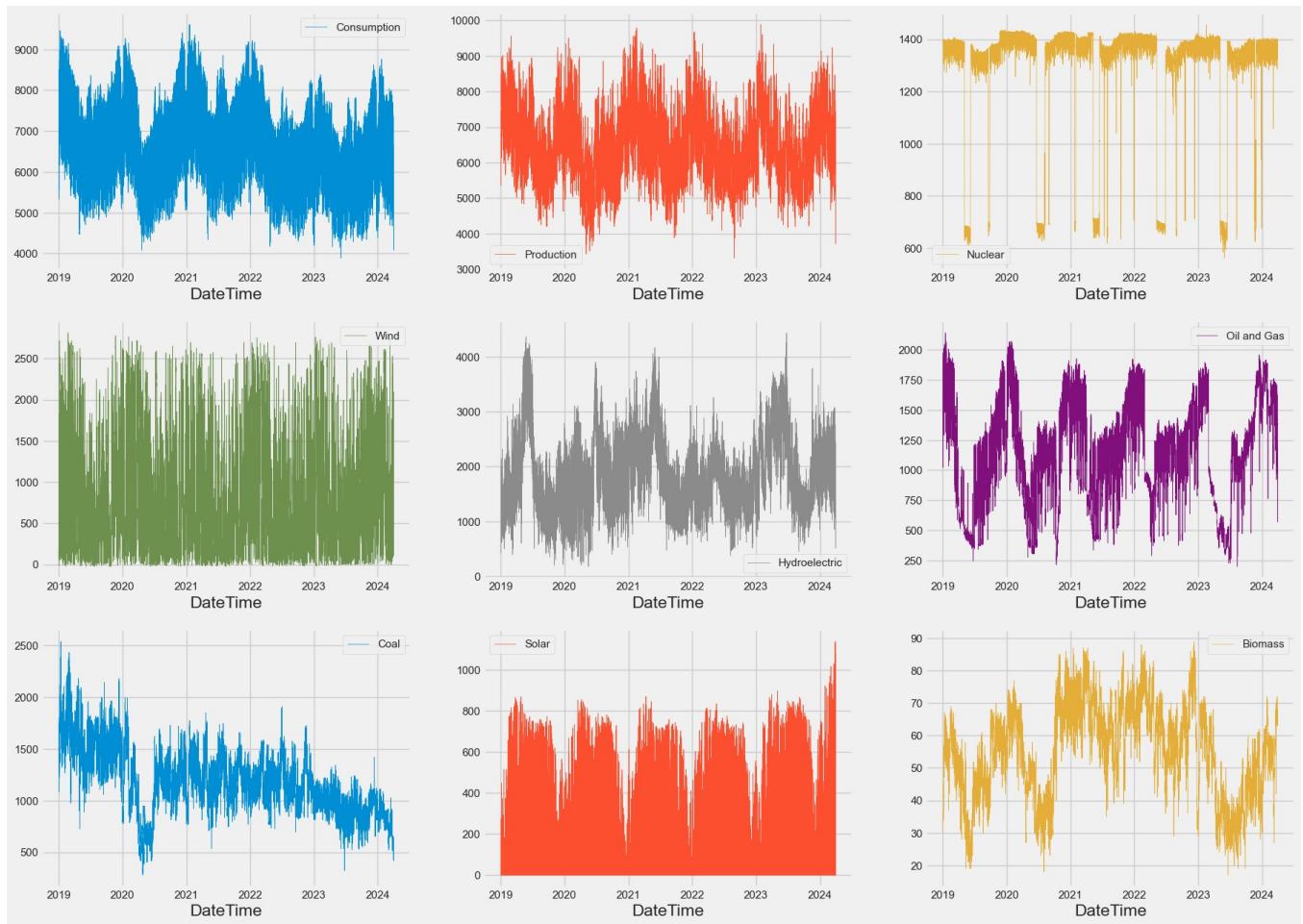
```
In [6]: print("Starting Date: ", df['DateTime'].min())
print("End Date      : ", df['DateTime'].max())
```

```
Starting Date: 2019-01-01 00:00:00
End Date      : 2024-03-31 23:00:00
```

```
In [100]: plt.style.use('fivethirtyeight')

df2 = df.copy('Deep')
df2 = df2.set_index('DateTime')

# Facet plots
df2.plot(subplots=True,
          linewidth=0.5,
          layout=(3, 3),
          figsize=(20, 15),
          sharex=False,
          sharey=False)
plt.show()
```

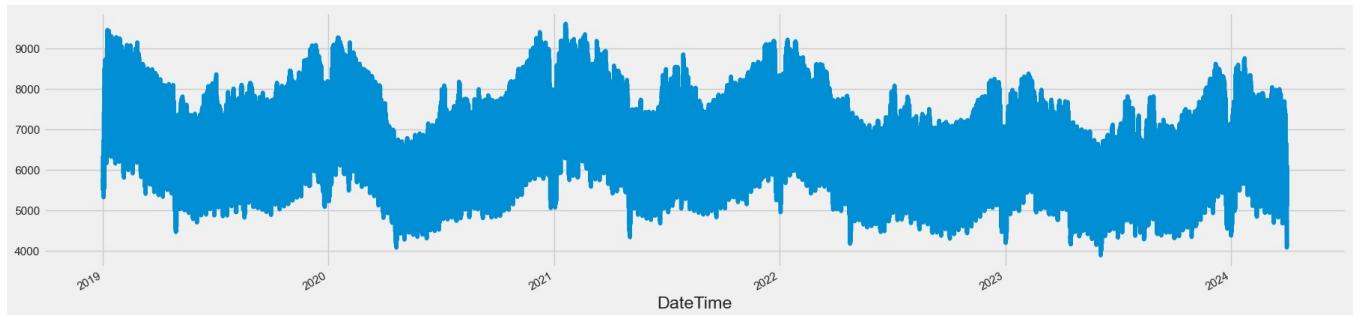


```
In [102]: df_consumption = df[['DateTime', 'Consumption']]
df_consumption.head()
```

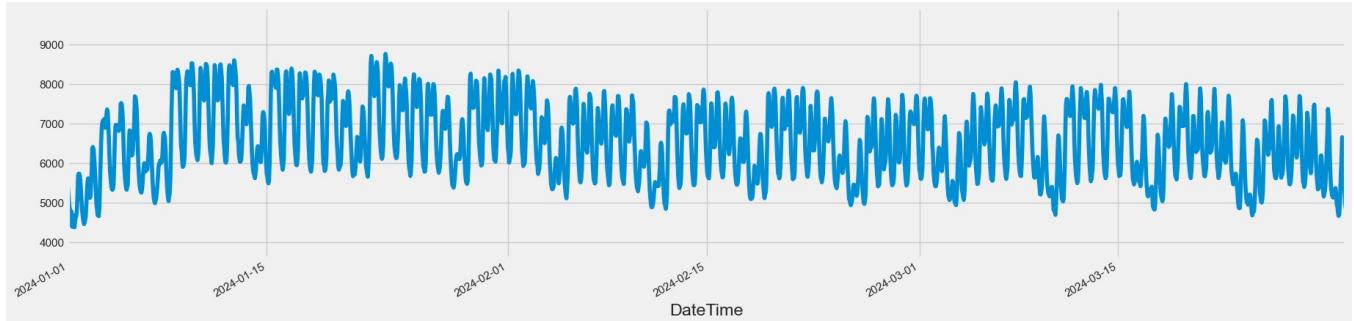
```
Out[102]:
```

|   | DateTime            | Consumption |
|---|---------------------|-------------|
| 0 | 2019-01-01 00:00:00 | 6352        |
| 1 | 2019-01-01 01:00:00 | 6116        |
| 2 | 2019-01-01 02:00:00 | 5873        |
| 3 | 2019-01-01 03:00:00 | 5682        |
| 4 | 2019-01-01 04:00:00 | 5557        |

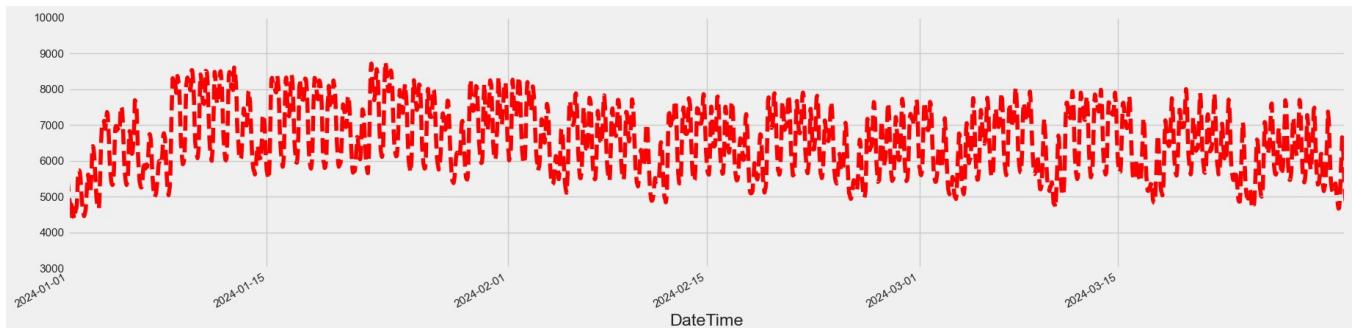
```
In [103]: df5 = df_consumption.copy('Deep')
df5 = df5.set_index('DateTime')
df5['Consumption'].plot(figsize=(20,5))
plt.show()
```



```
In [104]: ## xlim and y limit
df5['Consumption'].plot(xlim=['2024-01-01', '2024-03-31'], figsize=(20,5))
plt.show()
```



```
In [105]: ## xlim and ylim
df5['Consumption'].plot(xlim=['2024-01-01','2024-03-31'],figsize=(20,5),ylim=[3000,10000],ls='--',c='Red')
plt.show()
```



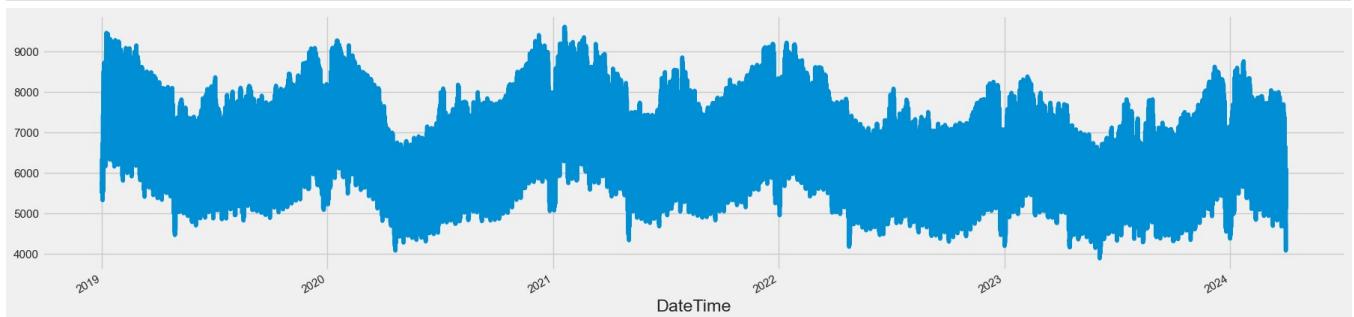
```
In [106]: index=df5.loc['2023-01-01':'2023-09-01'].index
Consumption=df5.loc['2023-01-01':'2023-09-01']['Consumption']
Consumption
```

```
Out[106]: DateTime
2023-01-01 00:00:00    4996
2023-01-01 01:00:00    4995
2023-01-01 02:00:00    4816
2023-01-01 03:00:00    4627
2023-01-01 04:00:00    4581
...
2023-09-01 19:00:00    6704
2023-09-01 20:00:00    6932
2023-09-01 21:00:00    6712
2023-09-01 22:00:00    6090
2023-09-01 23:00:00    5582
Name: Consumption, Length: 5855, dtype: int64
```

```
In [107]: print(plt.style.available)
```

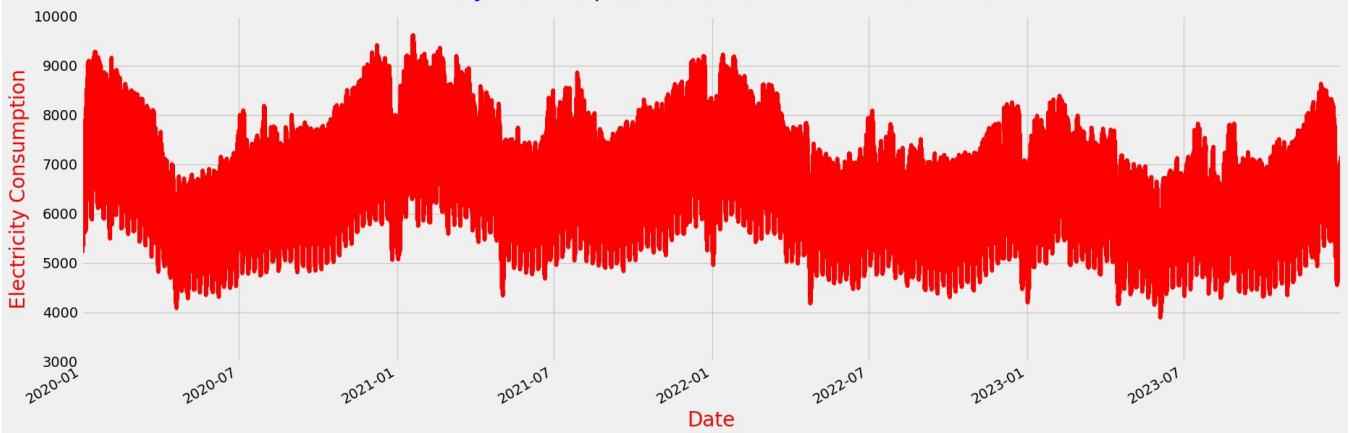
```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn-v0_8', 'seaborn-v0_8-bright', 'seaborn-v0_8-colorblind', 'seaborn-v0_8-dark', 'seaborn-v0_8-dark-palette', 'seaborn-v0_8-darkgrid', 'seaborn-v0_8-deep', 'seaborn-v0_8-muted', 'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn-v0_8-pastel', 'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-ticks', 'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

```
In [108]: plt.style.use('fivethirtyeight')
df5 = df_consumption.copy('Deep')
df5 = df5.set_index('DateTime')
df5['Consumption'].plot(figsize=(20,5))
plt.show()
```



```
In [16]: ax = df5['Consumption'].plot(xlim=['2020-01-01','2023-12-31'], ylim=[3000,10000] ,color ="Red", figsize=(20,7))
ax.set_xlabel('Date', color = "Red" ,fontsize= 20)
ax.set_ylabel('Electricity Consumption', color = "Red", fontsize=20)
ax.set_title('Electricity Consumption from 2020-01-01 to 2023-12-31 ', color = "Blue", fontsize=24 ,pad=20)
plt.show()
```

### Electricity Consumption from 2020-01-01 to 2023-12-31

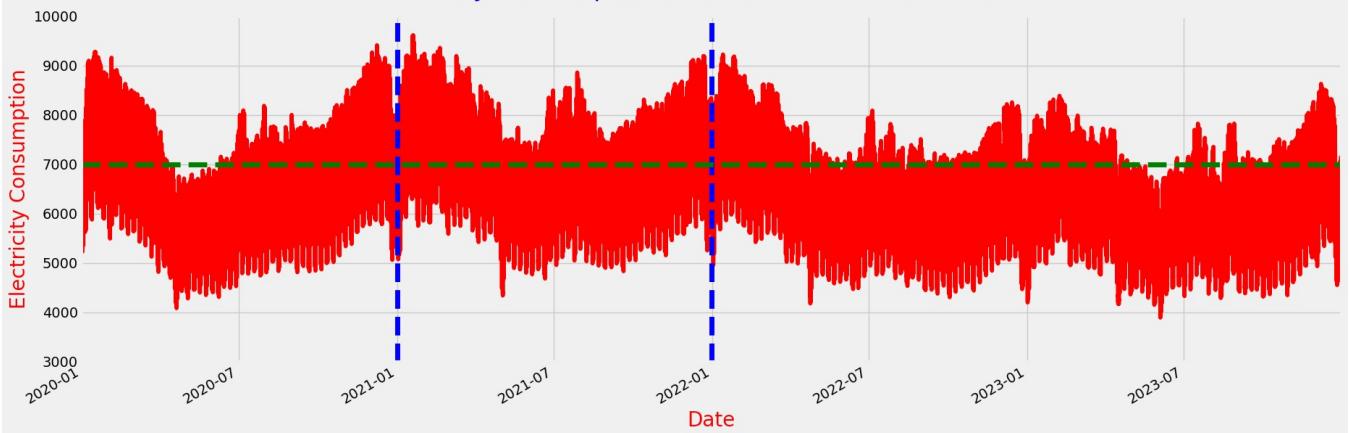


In [17]:

```
# adding markers
ax = df5['Consumption'].plot(xlim=['2020-01-01','2023-12-31'], ylim=[3000,10000] ,color ="Red", figsize=(20,7))
ax.set_xlabel('Date', color = "Red" ,fontsize= 20)
ax.set_ylabel('Electricity Consumption', color = "Red", fontsize=20)
ax.set_title('Electricity Consumption from 2020-01-01 to 2023-12-31 ', color = "Blue",  fontsize=24 ,pad=20)

ax.axvline('2021-01-01', color='Blue', linestyle='--', lw = 5)
ax.axvline('2021-12-31', color='Blue', linestyle='--', lw = 5)
ax.axhline(7000, color='green', linestyle='--', lw = 5)
plt.show()
```

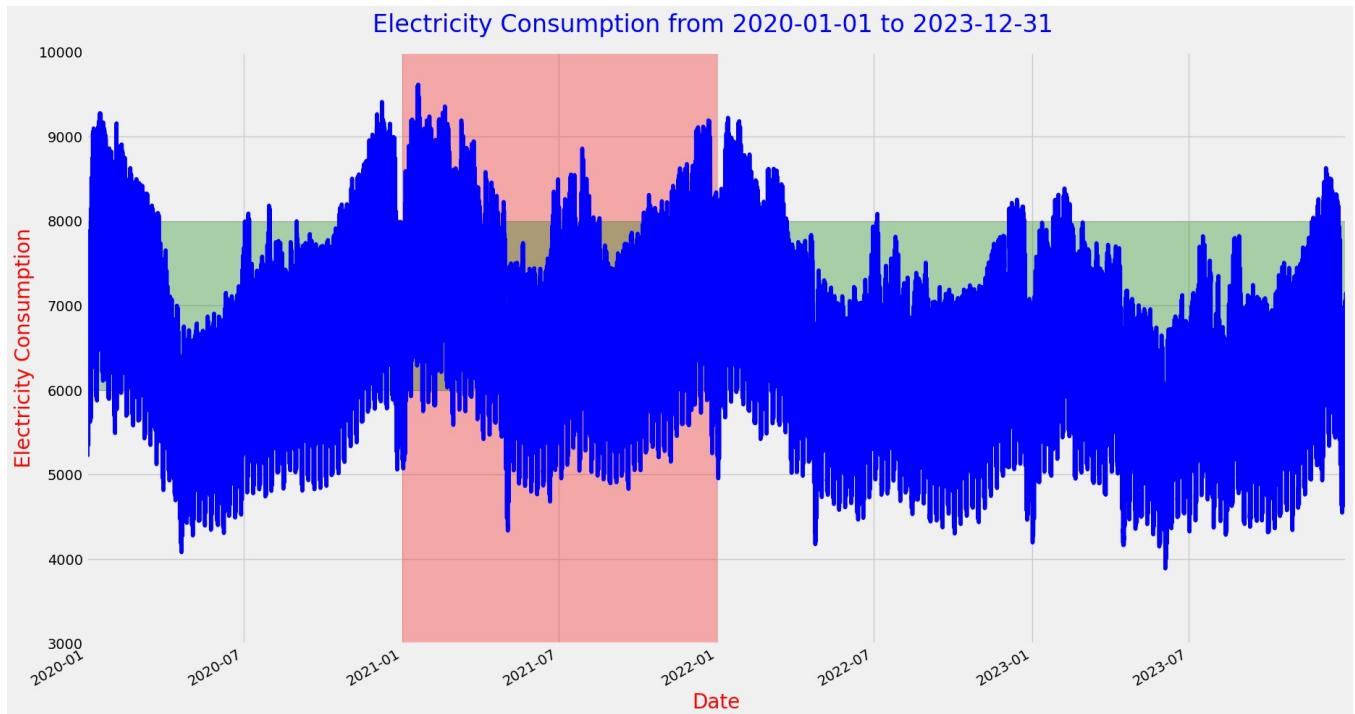
### Electricity Consumption from 2020-01-01 to 2023-12-31



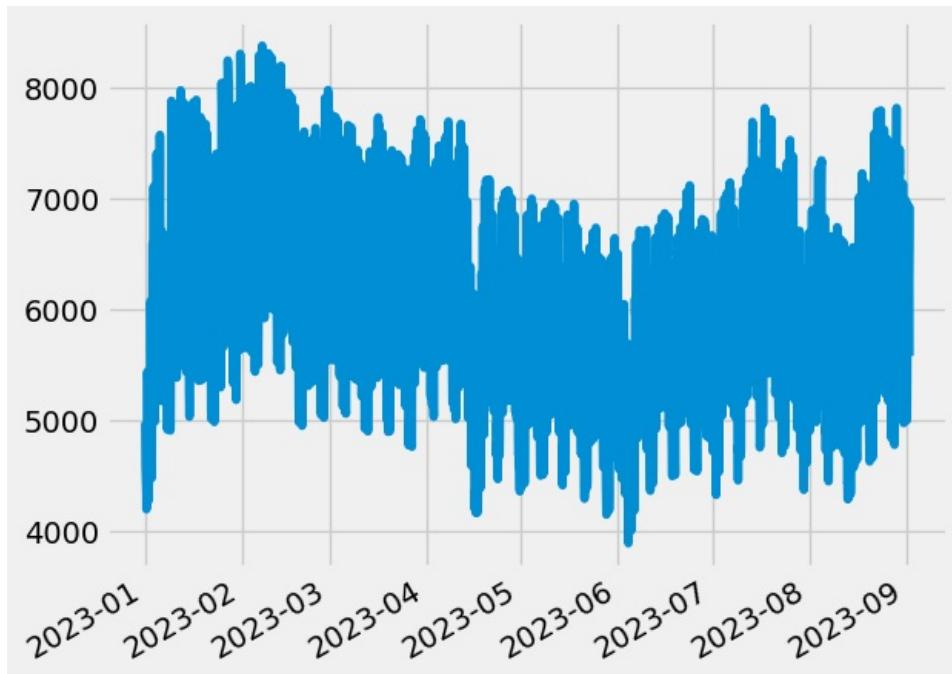
In [18]:

```
# Highlighting regions of interest
ax = df5['Consumption'].plot(xlim=['2020-01-01','2023-12-31'], ylim=[3000,10000] ,color ="Blue", figsize=(20,12)
ax.set_xlabel('Date', color = "Red" ,fontsize= 20)
ax.set_ylabel('Electricity Consumption', color = "Red", fontsize=20)
ax.set_title('Electricity Consumption from 2020-01-01 to 2023-12-31 ', color = "Blue",  fontsize=24 ,pad=20)

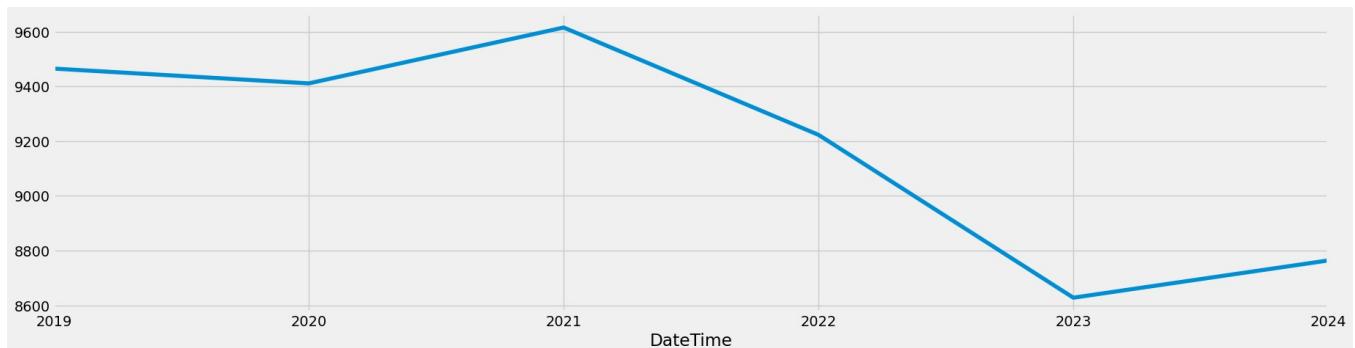
ax.axvspan('2021-01-01', '2021-12-31', color='Red', alpha=0.3)
ax.axhspan(8000, 6000, color='green', alpha=0.3)
plt.show()
```



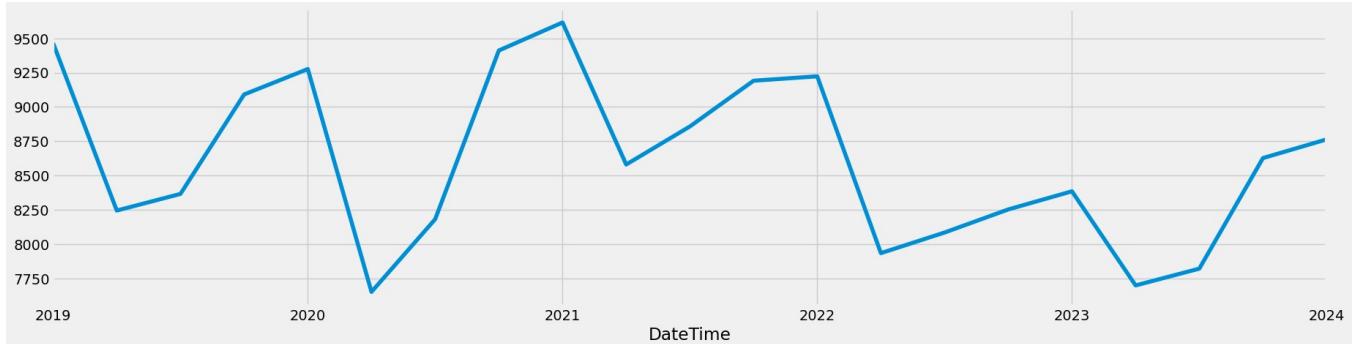
```
In [19]: figure, axis=plt.subplots()
plt.tight_layout()
## Preventing overlapping
figure.autofmt_xdate()
axis.plot(index, Consumption);
```



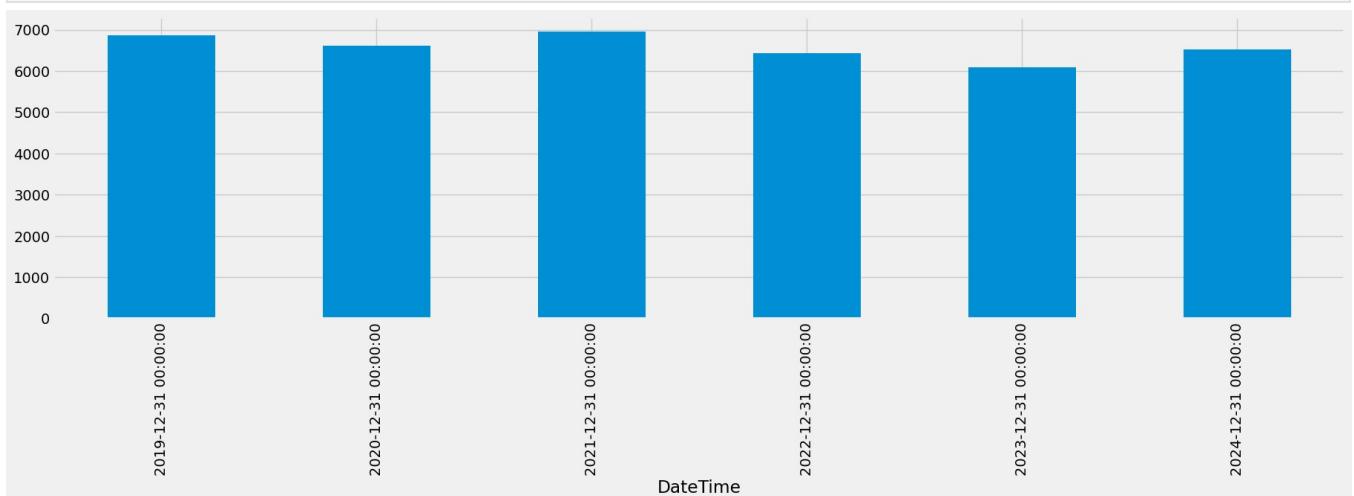
```
In [20]: ##year end frequency
df5.resample(rule='A').max()['Consumption'].plot(figsize=(20,5));
```



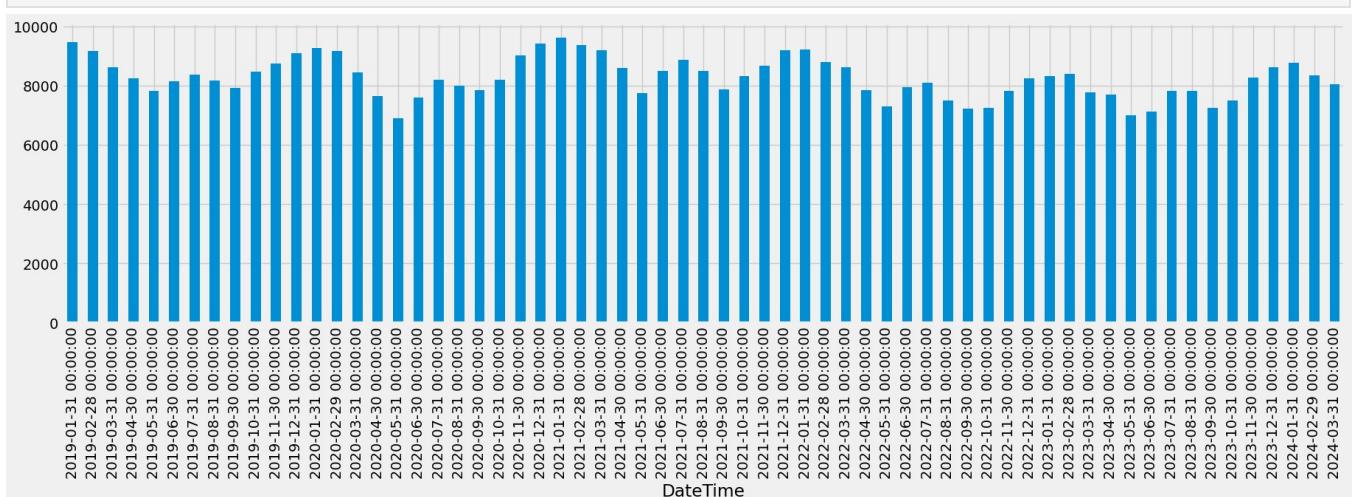
```
In [21]: df5.resample(rule='QS').max()['Consumption'].plot(figsize=(20,5));
```



```
In [22]: ##plotting
df5['Consumption'].resample(rule='A').mean().plot(kind='bar', figsize=(20,5));
```



```
In [23]: df5['Consumption'].resample(rule='M').max().plot(kind='bar', figsize=(20,5));
```



```
In [24]: df5['Consumption'].rolling(5).max().head(8)
```

```
Out[24]: DateTime
2019-01-01 00:00:00      NaN
2019-01-01 01:00:00      NaN
2019-01-01 02:00:00      NaN
2019-01-01 03:00:00      NaN
2019-01-01 04:00:00    6352.0
2019-01-01 05:00:00    6116.0
2019-01-01 06:00:00    5873.0
2019-01-01 07:00:00    5682.0
Name: Consumption, dtype: float64
```

```
In [25]: count_date = df5.groupby(df5.index.date)[['Consumption']].sum()

pw_clean = pd.DataFrame(count_date)
pw_clean['DateTime'] = pd.to_datetime(pw_clean.index)
pw_clean = pw_clean.set_index('DateTime')
pw_clean.head()
```

Out[25]:

**Consumption**

| DateTime   |        |
|------------|--------|
| 2019-01-01 | 142984 |
| 2019-01-02 | 151729 |
| 2019-01-03 | 174098 |
| 2019-01-04 | 183242 |
| 2019-01-05 | 177114 |

In [26]:

```
pw_clean['Consumption:30 days rolling']=pw_clean['Consumption'].rolling(30).mean()
pw_clean.head(3)
```

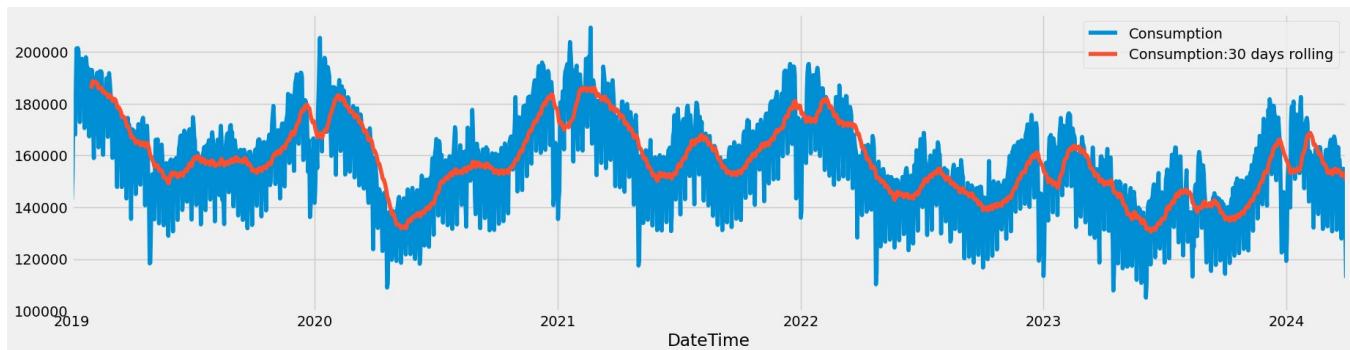
Out[26]:

**Consumption Consumption:30 days rolling**

| DateTime   |        |     |
|------------|--------|-----|
| 2019-01-01 | 142984 | NaN |
| 2019-01-02 | 151729 | NaN |
| 2019-01-03 | 174098 | NaN |

In [27]:

```
pw_clean[['Consumption', 'Consumption:30 days rolling']].plot(figsize=(20,5));
```



In [28]:

```
import matplotlib.cm as cm # Import colormap library

df1 = df[['DateTime','Consumption']].copy('Deep')
df1 = df1.set_index('DateTime')

years = [2019, 2020, 2021, 2022, 2023, 2024]

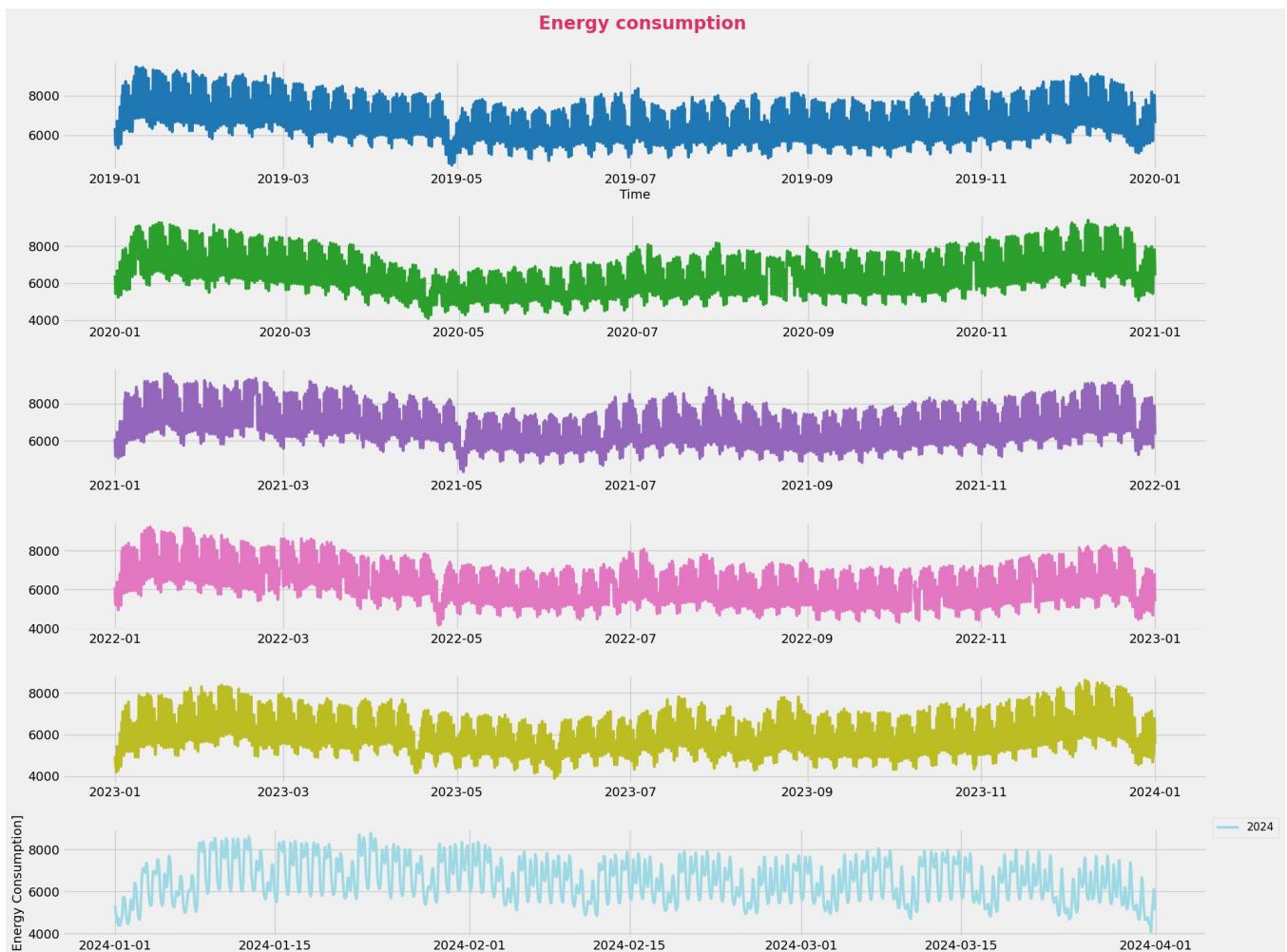
cmap = cm.get_cmap('tab20') # Choose a colormap with bright colors

fig, axs = plt.subplots(len(years), 1, figsize=(20, 15)) # Adjust figsize for layout
for i, year in enumerate(years):
    df_year = df1[(df1.index >= f"{year}-01-01") & (df1.index < f"{year+1}-01-01")]
    color = cmap(i / (len(years) - 1)) # Get color from colormap based on index
    axs[i].plot(df_year, lw=3, color=color, label=f'{year}') # Include year as label

# Set common labels and title
axs[0].set_xlabel('Time', fontsize=14) # Set on first subplot for all
axs[-1].set_ylabel('Energy Consumption', fontsize=14) # Set on last subplot for all
plt.suptitle('Energy consumption \n', weight='bold', fontsize=20, color="#DE3163")

# Add legend
handles, labels = axs[-1].get_legend_handles_labels() # Get labels from last subplot
plt.legend(handles, labels, loc='upper left', bbox_to_anchor=(1, 1.15), fontsize=12) # Add legend outside plot

plt.tight_layout()
plt.show()
```



In [29]: `df.head(2)`

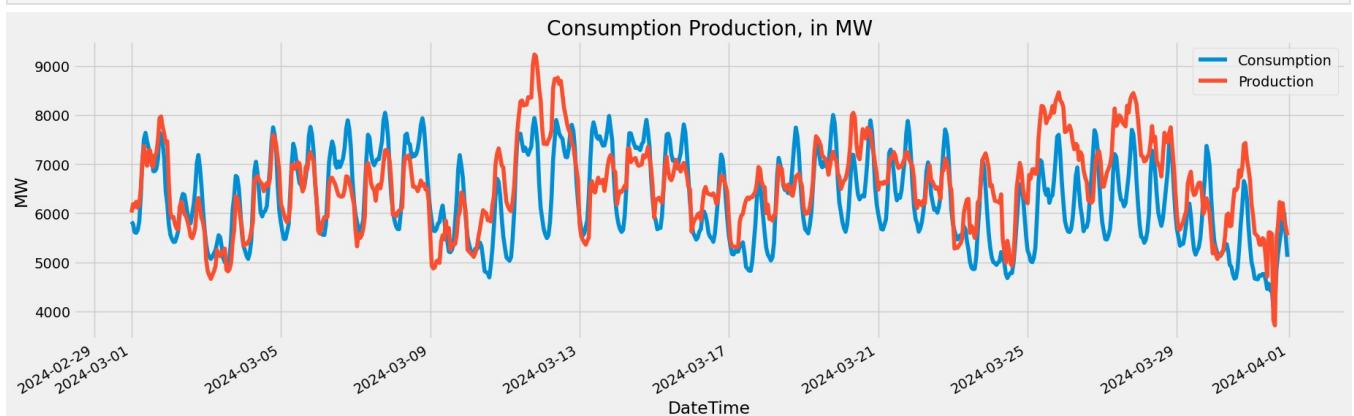
Out[29]:

|   | Datetime            | Consumption | Production | Nuclear | Wind | Hydroelectric | Oil and Gas | Coal | Solar | Biomass |
|---|---------------------|-------------|------------|---------|------|---------------|-------------|------|-------|---------|
| 0 | 2019-01-01 00:00:00 | 6352        | 6527       | 1395    | 79   | 1383          | 1896        | 1744 | 0     | 30      |
| 1 | 2019-01-01 01:00:00 | 6116        | 5701       | 1393    | 96   | 1112          | 1429        | 1641 | 0     | 30      |

In [30]:

```
df2 = df.copy('Deep')
df2 = df2.set_index('DateTime')

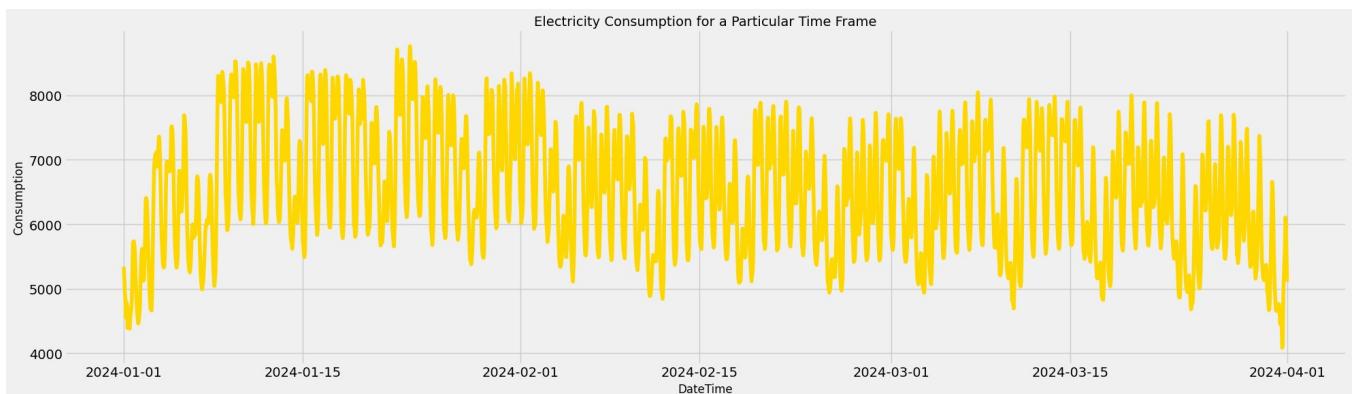
dfSummer = df2[['Consumption', 'Production']]["2024-03-01 00:00:00" : "2024-04-01 23:59:59"]
dfSummer.plot(style="-", figsize=(20, 6), title=f"Consumption Production, in MW")
plt.ylabel('MW')
plt.show()
```



In [31]:

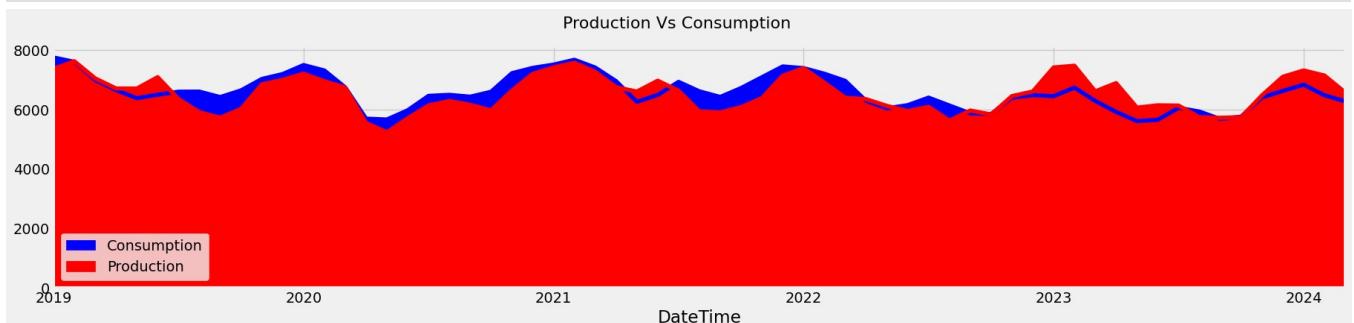
```
data1 = df[(df['DateTime'] >= '2024-01-01') & (df['DateTime'] < '2024-04-01')]
```

```
plt.figure(figsize=(20,6))
plt.plot(data1['DateTime'], data1['Consumption'], color='Gold')
plt.ylabel('Consumption', fontsize=12)
plt.xlabel('DateTime', fontsize=12)
plt.title('Electricity Consumption for a Particular Time Frame', fontsize=14)
plt.tight_layout()
plt.grid(True)
sns.despine(bottom=True, left=True)
plt.show()
```

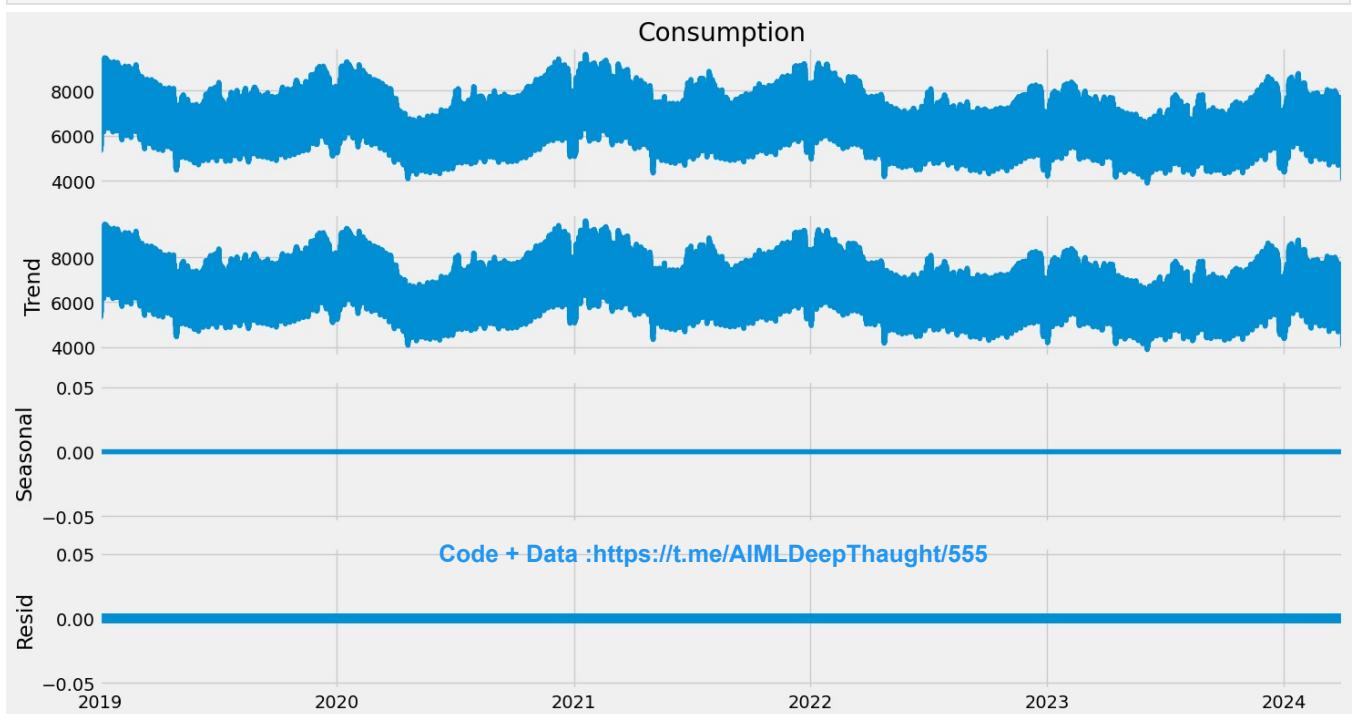


```
In [32]: df1 = df.copy('Deep')
df1.set_index('DateTime')

fig, ax = plt.subplots(figsize=(20,5)) # Create the figure and axes object
fig.suptitle('Production Vs Consumption', fontsize=16)
# Plot the first x and y axes:
df1.resample('M').mean().plot.area(use_index=True, y = 'Consumption', ax = ax,color='Blue')
df1.resample('M').mean().plot.area(use_index=True, y = 'Production', ax = ax, secondary_y = False,color='Red')
```



```
In [33]: plt.rcParams["figure.figsize"] = (15,8)
from statsmodels.tsa.seasonal import seasonal_decompose
decomp=seasonal_decompose(df1['Consumption'], model='additive', period=1)
decomp.plot()
plt.show()
```



```
In [34]: df_con = df.copy('deep')
df_con["DATE"] = pd.to_datetime(df_con["DateTime"]).dt.date
df_con["TIME"] = pd.to_datetime(df_con["DateTime"]).dt.time
df_con = df_con.set_index('DateTime')
df_con = df_con[["Consumption", "Production", "Solar", "DATE", "TIME"]][["2024-03-01 00:00:00" : "2024-03-30 23:59:59"]]
df_con.head()
```

Out[34]:

Consumption Production Solar DATE TIME

| Date                | Consumption | Production | Solar | DATE       | TIME     |
|---------------------|-------------|------------|-------|------------|----------|
| 2024-03-01 00:00:00 | 5830        | 6014       | 0     | 2024-03-01 | 00:00:00 |
| 2024-03-01 01:00:00 | 5780        | 6186       | 0     | 2024-03-01 | 01:00:00 |
| 2024-03-01 02:00:00 | 5617        | 6127       | 0     | 2024-03-01 | 02:00:00 |
| 2024-03-01 03:00:00 | 5600        | 6234       | 0     | 2024-03-01 | 03:00:00 |
| 2024-03-01 04:00:00 | 5656        | 6112       | 0     | 2024-03-01 | 04:00:00 |

In [35]:

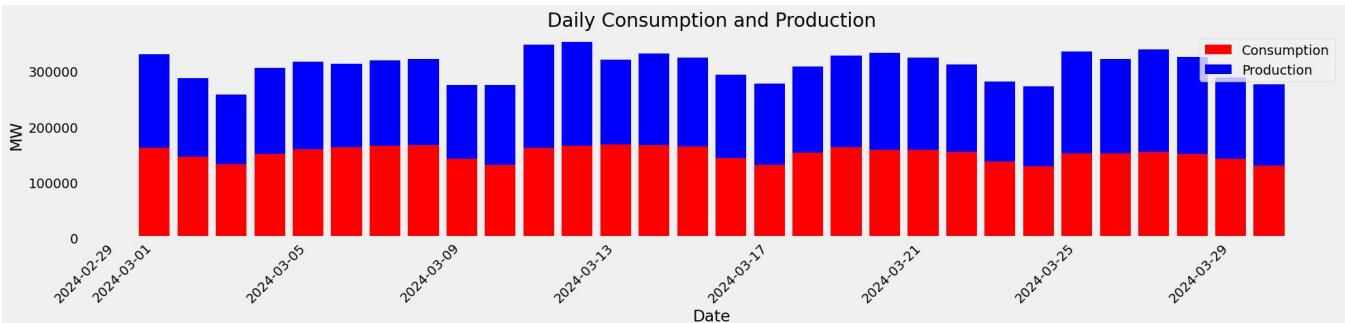
```
# Calculate daily consumption and production sums
daily_consumption = df_con.groupby('DATE')['Consumption'].agg('sum')
daily_production = df_con.groupby('DATE')['Production'].agg('sum')

# Create the combined plot
fig, ax = plt.subplots(figsize=(20, 5)) # Create figure and axis

# Plot consumption (red) and production (blue) on the same axis
ax.bar(daily_consumption.index, daily_consumption, color='red', label='Consumption')
ax.bar(daily_production.index, daily_production, color='blue', label='Production', bottom=daily_consumption)

# Set labels, title, and legend
ax.set_xlabel('Date')
ax.set_ylabel('MW')
plt.title('Daily Consumption and Production')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better readability (optional)
plt.legend()
plt.grid(False) # Add grid lines for better readability (optional)
plt.tight_layout() # Adjust spacing to prevent overlapping elements (optional)

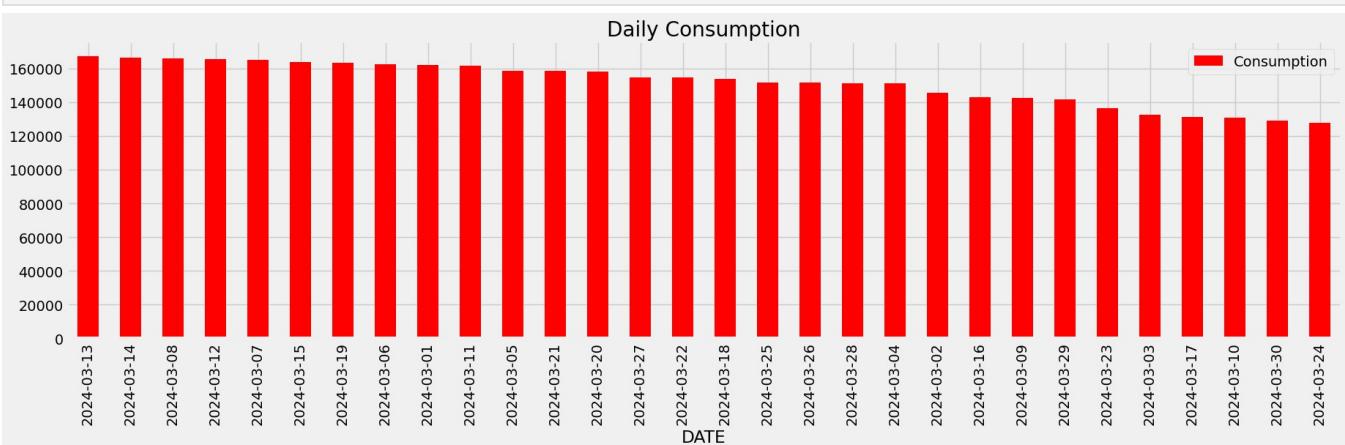
plt.show()
```



In [36]:

```
daily_Consumption = df_con.groupby('DATE')['Consumption'].agg('sum')

ax = daily_Consumption.sort_values(ascending=False).plot.bar(figsize=(20,5), legend=True, color='red')
plt.title('Daily Consumption')
plt.show()
```



In [37]:

```
df_solar = df.copy('deep')

df_solar["DATE"] = pd.to_datetime(df_solar["DateTime"]).dt.date
df_solar["TIME"] = pd.to_datetime(df_solar["DateTime"]).dt.time
df_solar["DATE"] = pd.to_datetime(df_solar["DateTime"]).dt.date
df_solar["DATE_STRING"] = df_solar["DATE"].astype(str) # add column with date as string

df_solar.head()
```

Out[37]:

|   | Datetime            | Consumption | Production | Nuclear | Wind | Hydroelectric | Oil and Gas | Coal | Solar | Biomass | DATE       | TIME     | DATE_STRING |
|---|---------------------|-------------|------------|---------|------|---------------|-------------|------|-------|---------|------------|----------|-------------|
| 0 | 2019-01-01 00:00:00 | 6352        | 6527       | 1395    | 79   | 1383          | 1896        | 1744 | 0     | 30      | 2019-01-01 | 00:00:00 | 2019-01-01  |
| 1 | 2019-01-01 01:00:00 | 6116        | 5701       | 1393    | 96   | 1112          | 1429        | 1641 | 0     | 30      | 2019-01-01 | 01:00:00 | 2019-01-01  |
| 2 | 2019-01-01 02:00:00 | 5873        | 5676       | 1393    | 142  | 1030          | 1465        | 1616 | 0     | 30      | 2019-01-01 | 02:00:00 | 2019-01-01  |
| 3 | 2019-01-01 03:00:00 | 5682        | 5603       | 1397    | 191  | 972           | 1455        | 1558 | 0     | 30      | 2019-01-01 | 03:00:00 | 2019-01-01  |
| 4 | 2019-01-01 04:00:00 | 5557        | 5454       | 1393    | 159  | 960           | 1454        | 1458 | 0     | 30      | 2019-01-01 | 04:00:00 | 2019-01-01  |

In [38]:

```

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df_solar' has columns 'DateTime', 'Consumption', 'Production', 'Solar', and 'Wind'

date = ["2024-03-24"] # List of dates to filter

# Filter data for the specified date(s)
filtered_data = df_solar[df_solar["DATE_STRING"].isin(date)]

# Create the figure with desired size
plt.figure(figsize=(20, 16))

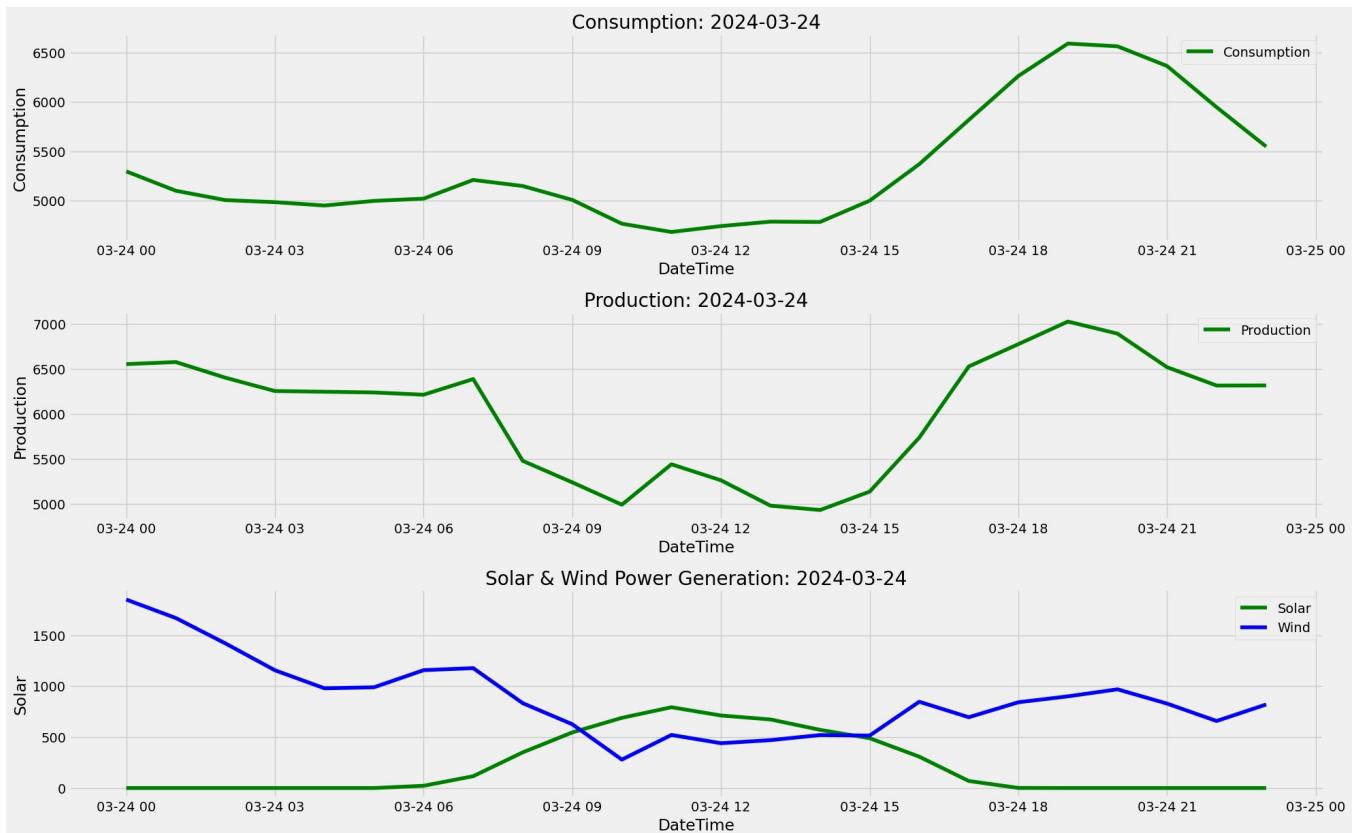
# Subplot 1: Consumption
plt.subplot(411)
sns.lineplot(data=filtered_data, x="DateTime", y="Consumption", label="Consumption", color='green')
plt.title("Consumption: {}".format(date[0]))

# Subplot 2: Production
plt.subplot(412)
sns.lineplot(data=filtered_data, x="DateTime", y="Production", label="Production", color='green')
plt.title("Production: {}".format(date[0]))

# Subplot 3: Solar & Wind
plt.subplot(413)
sns.lineplot(data=filtered_data, x="DateTime", y="Solar", label="Solar", color='green')
sns.lineplot(data=filtered_data, x="DateTime", y="Wind", label="Wind", color='blue')
plt.title("Solar & Wind Power Generation: {}".format(date[0]))

plt.tight_layout()
plt.show()

```



In [39]:

```

df_con_pro = df_con.pivot_table(values=['Consumption', 'Production'], index='TIME', columns='DATE')
df_con_pro.head()

```

Out[39]:

| DATE     | 2024-03-01      | 2024-03-02 | 2024-03-03 | 2024-03-04 | 2024-03-05 | 2024-03-06 | 2024-03-07 | 2024-03-08 | 2024-03-09 | 2024-03-10 | ... | 2024-03-21 | 2024-03-22 | 2024-03-23 | 2024-03-24 | 2024-03-25 | 2024-03-26 | ...  |
|----------|-----------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|-----|------------|------------|------------|------------|------------|------------|------|
| TIME     | Consumption ... |            |            |            |            |            |            |            |            |            |     |            |            |            |            |            |            |      |
| 00:00:00 | 5830.0          | 5774.0     | 5375.0     | 5331.0     | 5813.0     | 5889.0     | 5973.0     | 6064.0     | 6079.0     | 5516.0     | ... | 6479.0     | 6215.0     | 5907.0     | 6551.0     | 6202.0     | 7654.0     | 6... |
| 01:00:00 | 5780.0          | 5560.0     | 5186.0     | 5236.0     | 5625.0     | 5659.0     | 5821.0     | 5863.0     | 5834.0     | 5366.0     | ... | 6624.0     | 6106.0     | 5281.0     | 6574.0     | 6780.0     | 7670.0     | 6... |
| 02:00:00 | 5617.0          | 5475.0     | 5115.0     | 5126.0     | 5473.0     | 5582.0     | 5664.0     | 5743.0     | 5646.0     | 5244.0     | ... | 6622.0     | 6250.0     | 5301.0     | 6400.0     | 6706.0     | 7724.0     | 6... |
| 03:00:00 | 5600.0          | 5415.0     | 5069.0     | 5070.0     | 5478.0     | 5561.0     | 5600.0     | 5680.0     | 5634.0     | 5211.0     | ... | 6599.0     | 6202.0     | 5295.0     | 6252.0     | 6893.0     | 7786.0     | 6... |
| 04:00:00 | 5656.0          | 5418.0     | 5113.0     | 5189.0     | 5615.0     | 5559.0     | 5736.0     | 5675.0     | 5677.0     | 5161.0     | ... | 6644.0     | 6162.0     | 5348.0     | 6244.0     | 6900.0     | 7688.0     | 6... |

5 rows × 60 columns

```
In [40]: sns.set_theme(style="white")
df_con_pro = df_con.pivot_table(values=['Consumption', 'Production'], index='TIME', columns='DATE')

def Daywise_plot(data=None, row=None, col=None, title='Electricity Consumption and Production', ylim_top=8000):
    """
    Plots timeseries data for each column in a grid layout.

    Args:
        data: A pandas DataFrame containing the timeseries data.
        row: Number of rows in the grid layout (optional).
        col: Number of columns in the grid layout (optional).
        title: Title for the overall plot (optional).
        ylim_top: Upper limit for the y-axis (optional, default: 8000).
    """
    cols = data.columns.levels[1] # Get all column names

    # Create the figure with desired size
    gp = plt.figure(figsize=(20, 40))

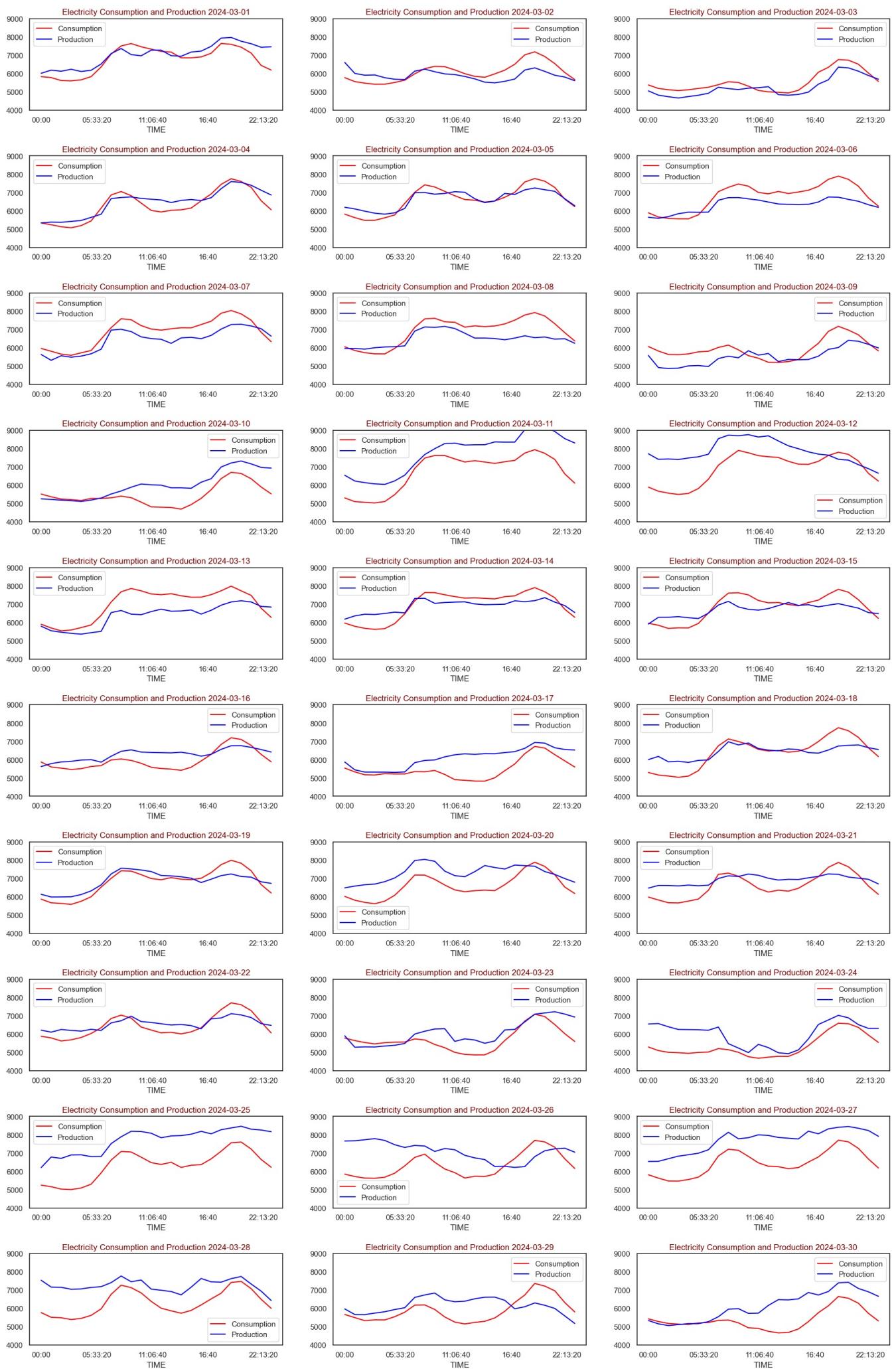
    # Adjust spacing between subplots
    gp.subplots_adjust(wspace=0.2, hspace=0.5)

    for i in range(1, len(cols) + 1):
        ax = gp.add_subplot(row, col, i)
        data[['Consumption', cols[i - 1]]].plot(ax=ax, color='red', label='Consumption')
        data[['Production', cols[i - 1]]].plot(ax=ax, color='blue', label='Production')

        # Set title and y-axis limit for each subplot
        ax.set_title('{0} {1}'.format(title, cols[i - 1]), color="#800000")
        ax.set_ylim(4000, 9000) # Update y-axis limit to 4000-8000
        ax.legend() # Show legend

    # Display the plot
    plt.show()

# Call the function with desired parameters and new y-axis limit
Daywise_plot(data=df_con_pro, row=12, col=3, ylim_top=8000)
plt.tight_layout()
plt.show()
```



<Figure size 1500x800 with 0 Axes>

Timeseries visualization of electricity consumption and generation for the entire month of March

2024, with the units on the y-axis labeled as MW (Megawatts).

1. Red Line: This line represents the electricity consumption trend over the month. It appears to fluctuate, potentially indicating higher consumption during daytime hours and lower consumption at night.

2. Blue Line: This line represents the electricity generation trend over the month. It also seems to fluctuate, and it's possible that the generation may not always keep up with consumption (indicated by the red line going above the blue line in some areas).

```
In [41]: plt.style.use('fivethirtyeight')
df_con_pro = df_con.pivot_table(values=['Solar'], index='TIME', columns='DATE')

def Daywise_plot(data=None, row=None, col=None, title='Solar Power Generation', ylim_top=1000):
    """
    Plots timeseries data for each column in a grid layout.

    Args:
        data: A pandas DataFrame containing the timeseries data.
        row: Number of rows in the grid layout (optional).
        col: Number of columns in the grid layout (optional).
        title: Title for the overall plot (optional).
        ylim_top: Upper limit for the y-axis (optional, default: 8000).
    """
    cols = data.columns.levels[1] # Get all column names

    # Create the figure with desired size
    gp = plt.figure(figsize=(20, 40))

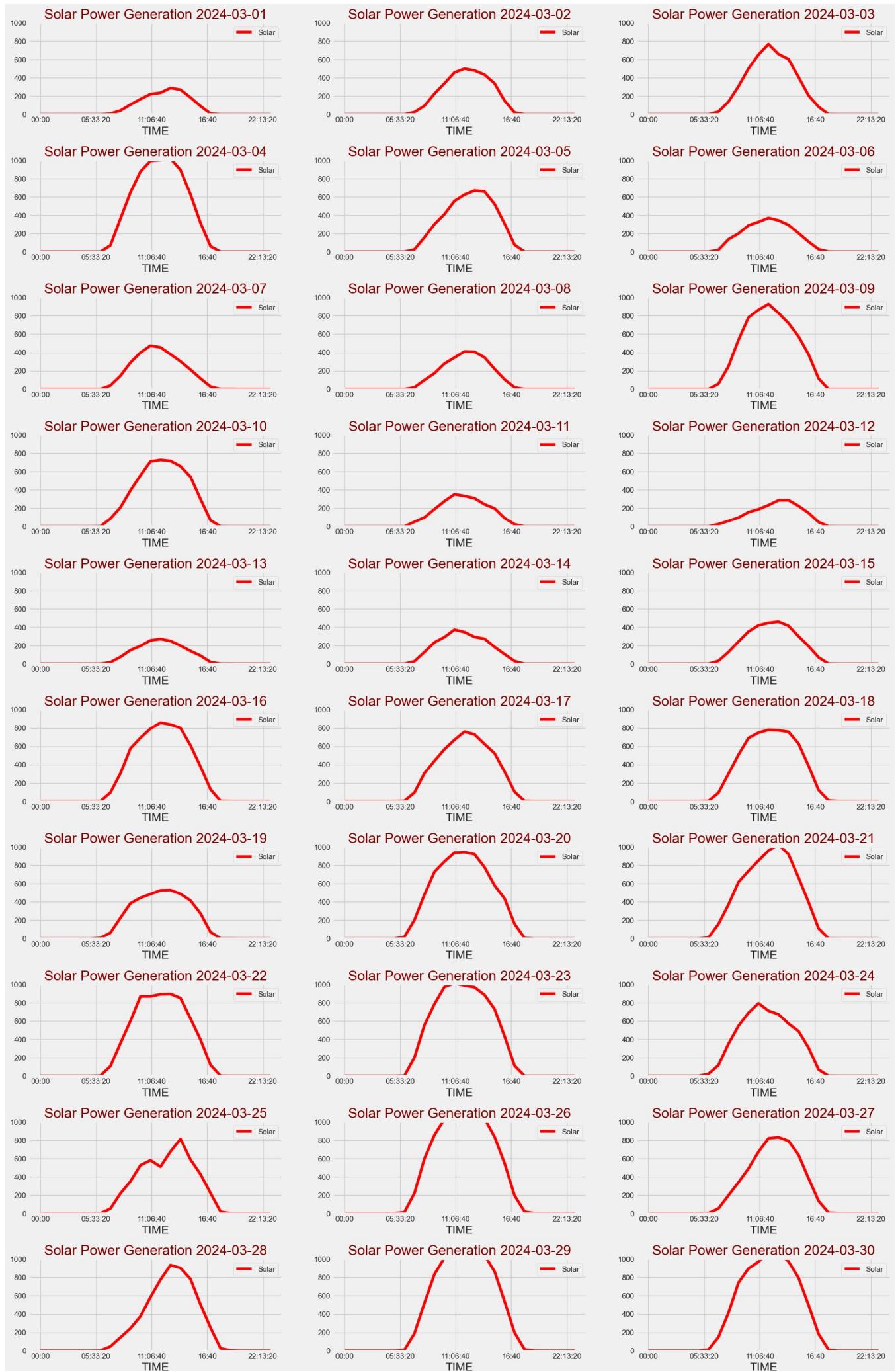
    # Adjust spacing between subplots
    gp.subplots_adjust(wspace=0.2, hspace=0.5)

    for i in range(1, len(cols) + 1):
        ax = gp.add_subplot(row, col, i)
        data['Solar', cols[i - 1]].plot(ax=ax, color='red', label='Solar')

        # Set title and y-axis limit for each subplot
        ax.set_title('{0} {1}'.format(title, cols[i - 1]), color='#800000')
        ax.set_ylim(0, 1000) # Update y-axis limit to 4000-8000
        ax.legend() # Show legend

    # Display the plot
    plt.show()

# Call the function with desired parameters and new y-axis limit
Daywise_plot(data=df_con_pro, row=12, col=3, ylim_top=1000)
```



```
In [42]: from statsmodels.tsa.seasonal import seasonal_decompose
```

```
def seasonal_decompose_plotter(df: pd.DataFrame, period=12, title='', figsize=(20, 12)):
```

```

"""
Perform and plot seasonal decomposition of a time series.

Parameters:
    df: DataFrame with time series data.
    col: Column name for data to decompose. Default is 'sqrt(03 AQI)'.
    date_col: Column name for datetime values. Default is 'Date'.
    period: Seasonality period. Default is 12.

Returns:
    A DecomposeResult object with seasonal, trend, and residual components.
"""

# Decomposition
decomposition = seasonal_decompose(df.values, period=period)
de_season = decomposition.seasonal
de_resid = decomposition.resid
de_trend = decomposition.trend

fig, ax = plt.subplots(4, sharex=True, figsize=figsize)

ax[0].set_title(title)
ax[0].plot(df.index, df.values, color='C3')
ax[0].set_ylabel(df.keys()[0])
ax[0].grid(alpha=0.25)

ax[1].plot(df.index, de_trend, color='C1')
ax[1].set_ylabel('Trend')
ax[1].grid(alpha=0.25)

ax[2].plot(df.index, de_season, color='C2')
ax[2].set_ylabel('Seasonal')
ax[2].grid(alpha=0.25)

ax[3].axhline(y=0, color='k', linewidth=1)
ax[3].scatter(df.index, de_resid, color='C0', s=10)
ax[3].set_ylabel('Resid')
ax[3].grid(alpha=0.25)

plt.tight_layout(h_pad=0)
plt.show()

return decomposition

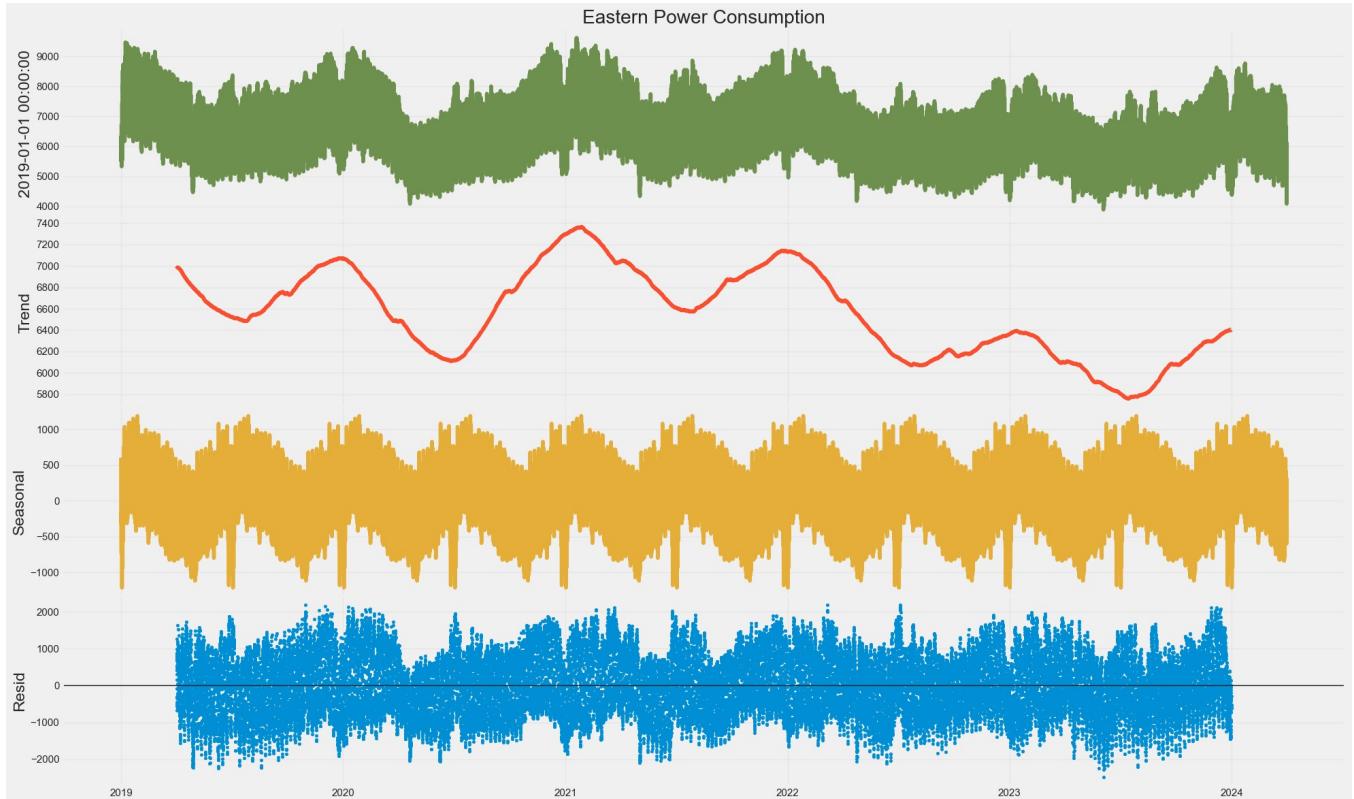
```

```

data1 = df[['DateTime', 'Consumption']].copy("deep")
data1 = data1.set_index('DateTime')

```

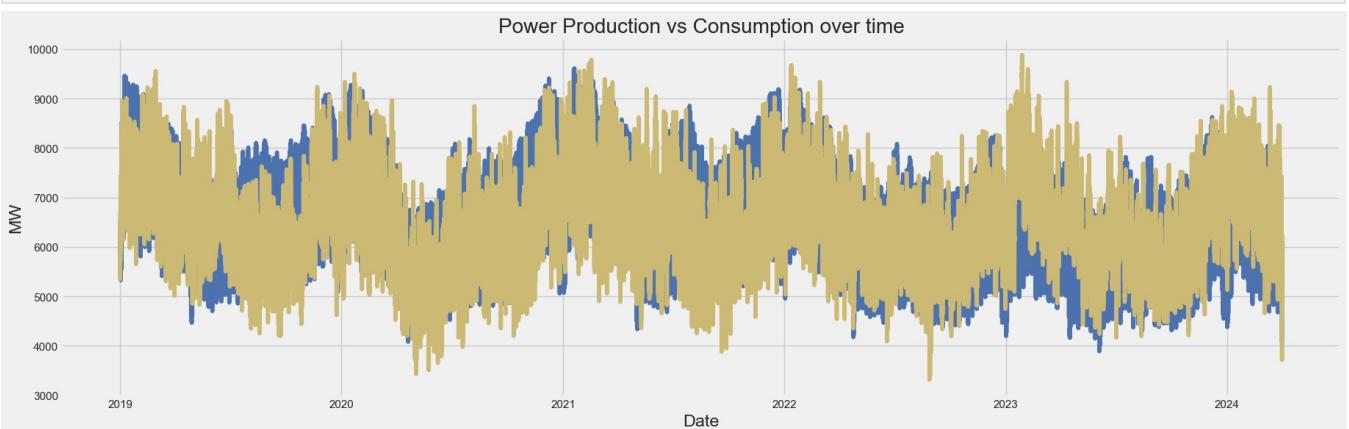
```
_ = seasonal_decompose_plotter(data1['Consumption'], period=365*12, title='Eastern Power Consumption', figsize=
```



```
In [43]: data = df.copy("deep")
data = data.set_index('DateTime')
```

```
In [44]: plt.figure(figsize=(20,6))
plt.plot(data.index,data['Consumption'], 'b')
plt.plot(data.index,data['Production'], 'y')
plt.title('Power Production vs Consumption over time')
```

```
plt.xlabel('Date')
plt.ylabel('MW')
plt.show()
```



```
In [45]: import matplotlib.pyplot as plt
from matplotlib.dates import MonthLocator, DateFormatter

# Assuming your data is in a pandas DataFrame named 'data'

# Plot for Consumption
ax = data.plot(figsize=(20, 5), use_index=True, grid=True,
               y='Consumption',
               title='Energy Consumption in MWs Unit',
               ylim=[4000, 10000],
               xlim=[data.index[0], data.index[-1]],
               style='b-')

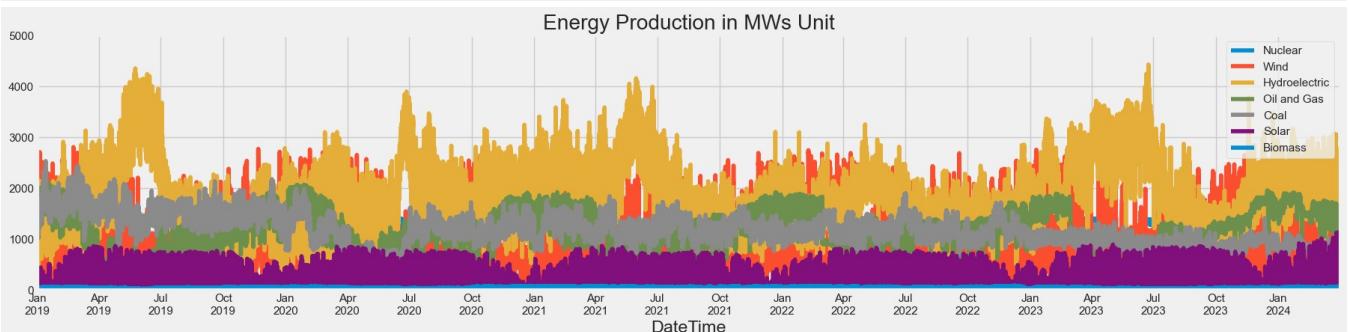
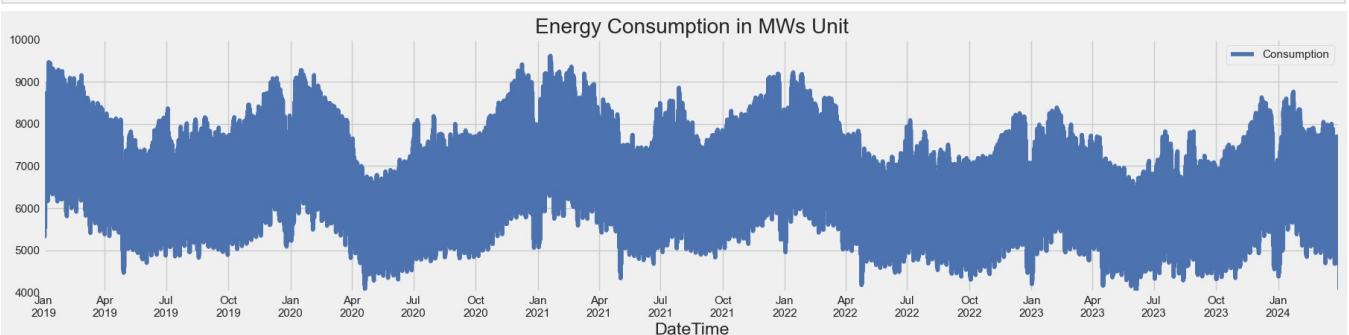
# Set x-axis labels with monthly formatting
ax.set_xticklabels(data.index, rotation=0, ha='center')
ax.xaxis.set_major_locator(MonthLocator(bymonth=(1, 4, 7, 10)))
ax.xaxis.set_major_formatter(DateFormatter('%b\n%Y'))

plt.show()

# Plot for Production (assuming 'Nuclear' etc. are columns in data)
ay = data.plot(figsize=(20, 5), use_index=True, grid=True,
               y=['Nuclear', 'Wind', 'Hydroelectric', 'Oil and Gas', 'Coal', 'Solar', 'Biomass'],
               title='Energy Production in MWs Unit',
               ylim=[0, 5000],
               xlim=[data.index[0], data.index[-1]])

# Set x-axis labels with monthly formatting
ay.set_xticklabels(data.index, rotation=0, ha='center')
ay.xaxis.set_major_locator(MonthLocator(bymonth=(1, 4, 7, 10)))
ay.xaxis.set_major_formatter(DateFormatter('%b\n%Y'))

plt.show()
```



```
In [46]: ax = data.plot(figsize=(20,5),use_index=True,grid=False,
                   y='Consumption',
                   title='2023 Energy Consumption in MWs Unit',
                   ylim=[4000,10000]),
```

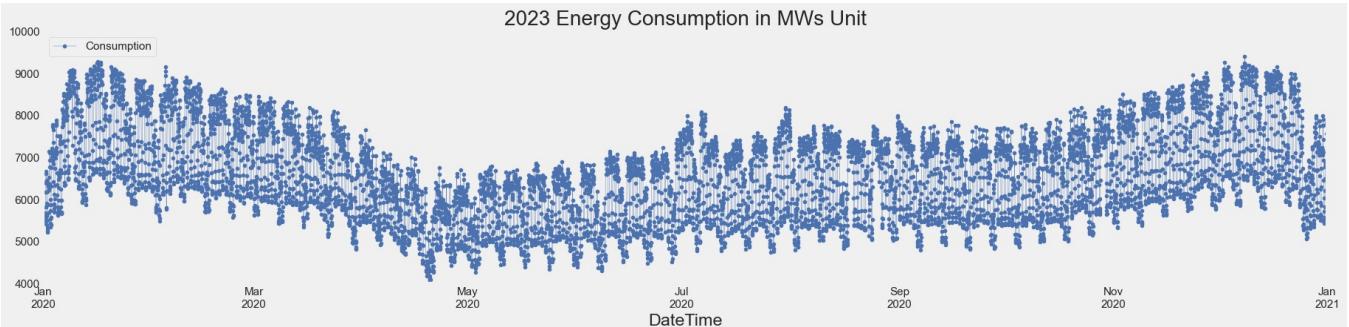
```

        xlim=[pd.Timestamp('2020-01-01'),pd.Timestamp('2021-01-01')]),
        style='b.-',lw=0.3)

ax.set_xticklabels(data.index, rotation=0, ha='center')
ax.xaxis.set_major_locator(MonthLocator(interval=2))
ax.xaxis.set_major_formatter(DateFormatter('%b\n%Y'))

plt.show()

```



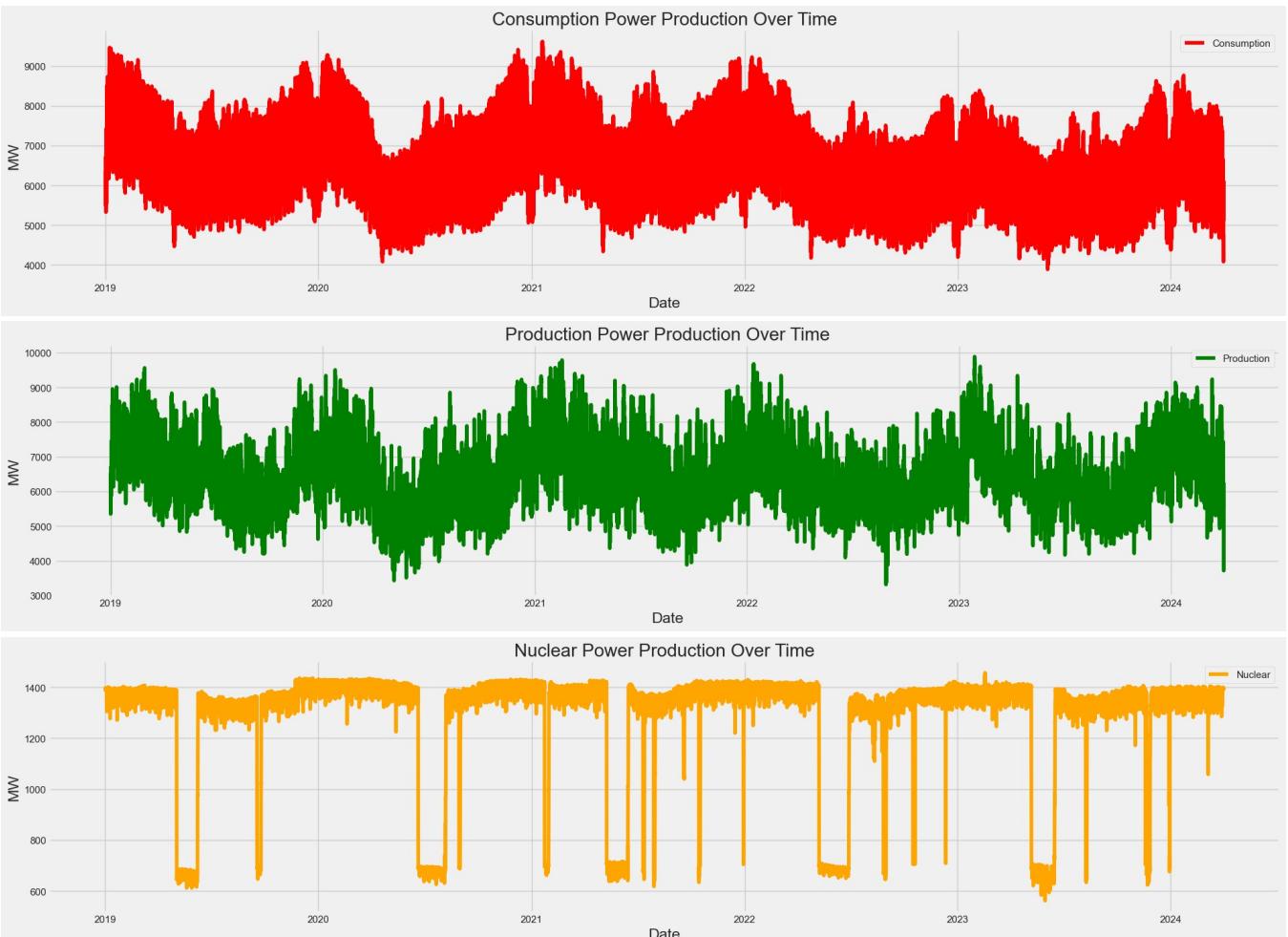
```

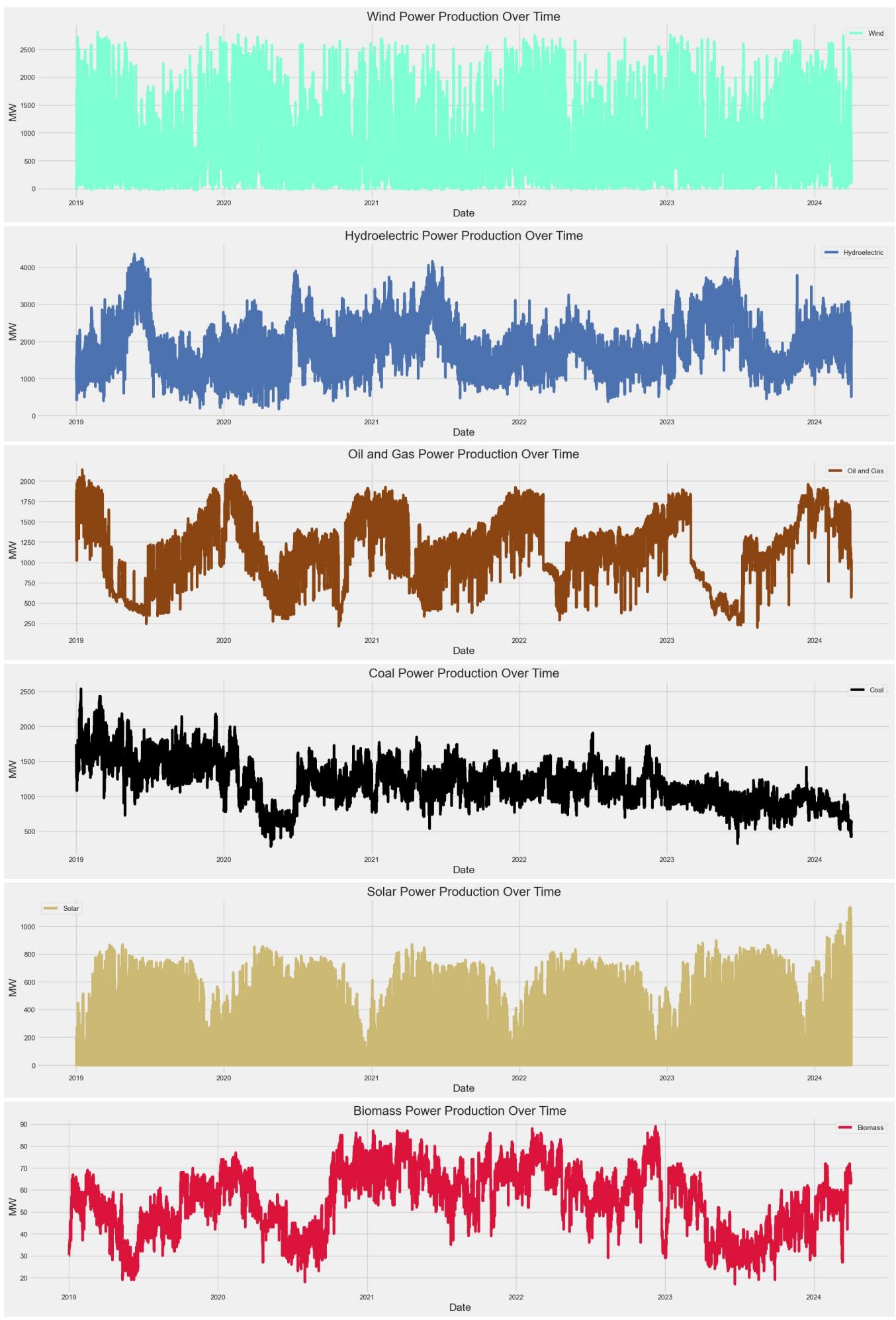
In [47]: production_types = list(data.columns) # Get all columns from index 2 onwards (skipping the first two)

# Dictionary of production type and color mapping
colors = {'Consumption' : 'Red', 'Production' : 'Green', 'Nuclear': 'orange', 'Wind': 'aquamarine', 'Hydroelectric': 'darkblue', 'Oil and Gas': 'saddlebrown', 'Coal': 'black', 'Solar': 'y', 'Biomass': 'crimson'}

# Loop through production types and create separate plots
for prType in production_types:
    plt.figure(figsize=(20, 5)) # Adjust figure size as desired
    plt.plot(data.index, data[prType], color=colors[prType], label=prType) # Add label
    plt.title(f"{prType} Power Production Over Time")
    plt.xlabel("Date")
    plt.ylabel("MW")
    plt.legend() # Add legend to identify each line
    plt.grid(True) # Add grid lines for better readability (optional)
    plt.tight_layout() # Adjust spacing to prevent overlapping elements (optional)
    plt.show()

```



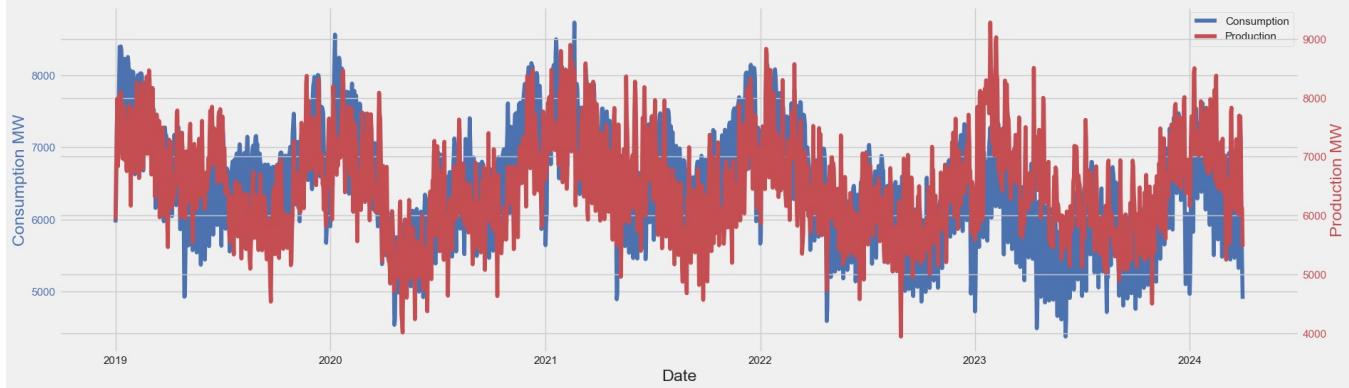


```
In [48]: df3 = df_solar[['DateTime','Consumption','Production']]  
  
df3 = df3.set_index('DateTime')  
daily_data = df3.resample('24h').mean()  
display(daily_data.head(2))
```

Consumption Production

| DateTime   | Consumption | Production  |
|------------|-------------|-------------|
| 2019-01-01 | 5957.666667 | 5924.416667 |
| 2019-01-02 | 6322.041667 | 6795.708333 |

```
In [49]: fig, ax1 = plt.subplots(figsize=(20,6))  
ax1.plot(daily_data.index, daily_data['Consumption'], 'b', label='Consumption')  
ax1.set_xlabel('Date')  
ax1.set_ylabel('Consumption MW', color='b')  
ax1.tick_params('y', colors='b')  
  
ax2 = ax1.twinx()  
  
ax2.plot(daily_data.index, daily_data['Production'], 'r', label='Production')  
ax2.set_ylabel('Production MW', color='r')  
ax2.tick_params('y', colors='r')  
  
lines1, labels1 = ax1.get_legend_handles_labels()  
lines2, labels2 = ax2.get_legend_handles_labels()  
lines = lines1 + lines2  
labels = labels1 + labels2  
ax1.legend(lines, labels, loc='upper right')  
plt.show()
```



```
In [50]: #group data by week  
groups = df_con['Consumption'].groupby(pd.Grouper(freq='W'))  
  
#set figure and axis  
fig, axs = plt.subplots(len(groups), 1, figsize=(20,30))  
for ax, (name, group) in zip(axs, groups):  
  
    #plot the data  
    ax.plot(pd.Series(group.values))  
  
    ax.set_xlabel('Hour of Year')  
    ax.set_ylabel('Consumption')  
    ax.set_title(name.week)  
    plt.subplots_adjust(hspace=0.5)
```



## Power Consumption by Day of the Week

```
In [51]: df6 = df[['DateTime', 'Consumption', 'Production']]
df6 = df6.set_index('DateTime')
df6.head()
```

Out[51]:

## Consumption Production

## DateTime

| 2019-01-01 00:00:00 | 6352 | 6527 |
|---------------------|------|------|
| 2019-01-01 01:00:00 | 6116 | 5701 |
| 2019-01-01 02:00:00 | 5873 | 5676 |
| 2019-01-01 03:00:00 | 5682 | 5603 |
| 2019-01-01 04:00:00 | 5557 | 5454 |

In [52]:

```
df6['Day'] = df6.index.day  
df6['Year'] = df6.index.year  
df6['Month'] = df6.index.month_name()  
df6['WeekDay'] = df6.index.day_name()  
df6.head()
```

Out[52]:

Consumption Production Day Year Month WeekDay

## DateTime

|                            |      |      |   |      |         |         |
|----------------------------|------|------|---|------|---------|---------|
| <b>2019-01-01 00:00:00</b> | 6352 | 6527 | 1 | 2019 | January | Tuesday |
| <b>2019-01-01 01:00:00</b> | 6116 | 5701 | 1 | 2019 | January | Tuesday |
| <b>2019-01-01 02:00:00</b> | 5873 | 5676 | 1 | 2019 | January | Tuesday |
| <b>2019-01-01 03:00:00</b> | 5682 | 5603 | 1 | 2019 | January | Tuesday |
| <b>2019-01-01 04:00:00</b> | 5557 | 5454 | 1 | 2019 | January | Tuesday |

In [53]:

```
dfbyDay = df6.drop('Month', axis=1).groupby('WeekDay').mean()
dfbyDay.head()
```

Out[53]:

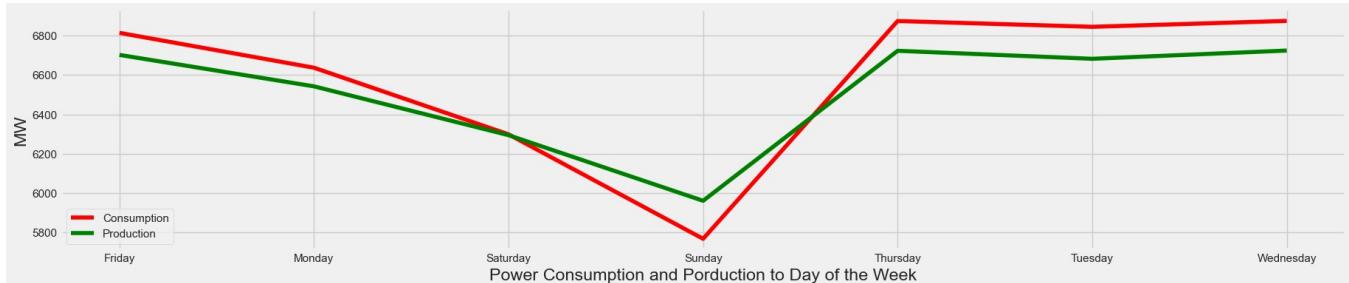
## Consumption      Production      Day      Year

## WeekDay

|                 |             |             |           |             |
|-----------------|-------------|-------------|-----------|-------------|
| <b>Friday</b>   | 6814.832725 | 6702.620894 | 15.580292 | 2021.142338 |
| <b>Monday</b>   | 6637.362332 | 6542.623474 | 15.739927 | 2021.142857 |
| <b>Saturday</b> | 6298.403437 | 6294.190389 | 15.686131 | 2021.145985 |
| <b>Sunday</b>   | 5766.934793 | 5960.282110 | 15.804682 | 2021.148807 |
| <b>Thursday</b> | 6875.098844 | 6723.519009 | 15.791971 | 2021.138686 |

In [54]:

```
plt.figure(figsize=(20,4))
plt.plot(dfbyDay.index, dfbyDay['Consumption'], 'Red')
plt.plot(dfbyDay.index, dfbyDay['Production'], 'Green')
plt.xlabel('Power Consumption and Production to Day of the Week')
plt.ylabel('MW')
plt.legend(['Consumption','Production'])
plt.show()
```



## Power Consumption by Month

In [55]:

```
dfbyMonth = df6.drop('WeekDay', axis=1).groupby('Month').mean()
dfbyMonth.head()
```

Out[55]:

## Consumption      Production      Day      Year

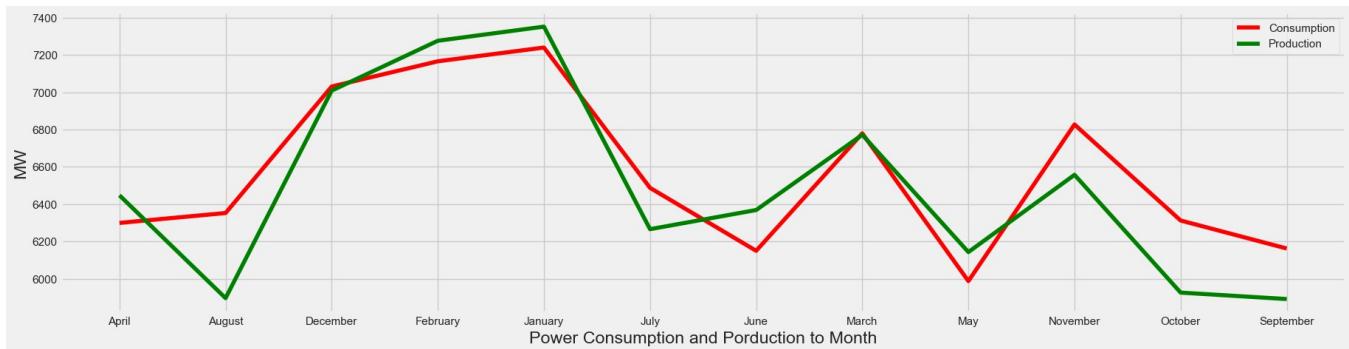
### Month

|                 |             |             |           |             |
|-----------------|-------------|-------------|-----------|-------------|
| <b>April</b>    | 6300.157500 | 6447.891389 | 15.500000 | 2021.000000 |
| <b>August</b>   | 6353.122581 | 5897.455376 | 16.000000 | 2021.000000 |
| <b>December</b> | 7030.345161 | 7008.483065 | 16.000000 | 2021.000000 |
| <b>February</b> | 7165.422794 | 7275.392402 | 14.670588 | 2021.505882 |
| <b>January</b>  | 7239.078629 | 7351.026434 | 16.000000 | 2021.500000 |

```

plt.figure(figsize=(12, 5))
plt.plot(dfbyMonth.index, dfbyMonth['Consumption'], 'Red')
plt.plot(dfbyMonth.index, dfbyMonth['Production'], 'Green')
plt.xlabel('Power Consumption and Production to Month')
plt.ylabel('MW')
plt.legend(['Consumption', 'Production'])
plt.show()

```



## Electricity Consumption and Production

```

In [57]: %%time
# add a new column for the delta
df6["delta"] = df6["Production"] - df6["Consumption"]

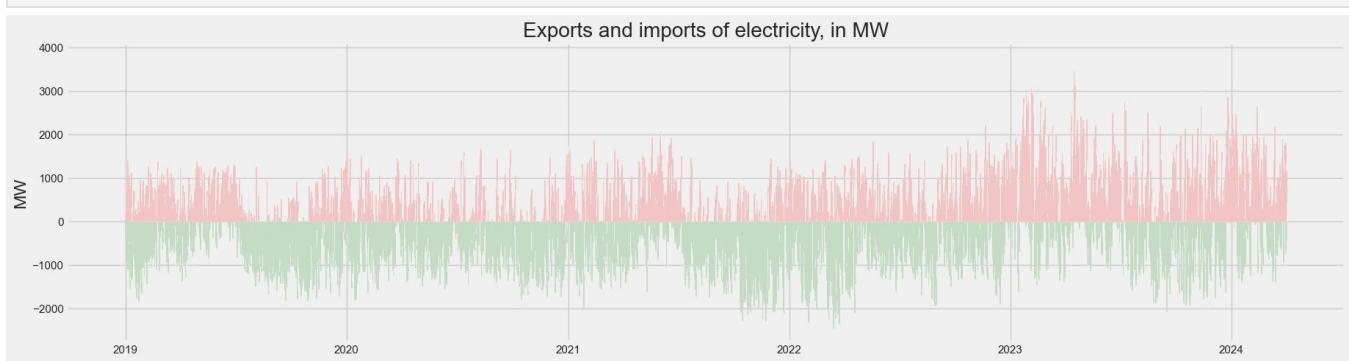
plt.rcParams["figure.figsize"] = (20, 5)

plt.bar(
    df6.index, df6['delta'],
    color=np.where(df6['delta'] > 0, 'Red', 'Green')
)

plt.title("Exports and imports of electricity, in MW")
plt.ylabel('MW')
plt.show()

# drop the column we have used for the delta
df6.drop(["delta"], axis=1, inplace=True)

```



CPU times: total: 1min 28s  
Wall time: 1min 33s

## Specific time Electricity Production and Consumption

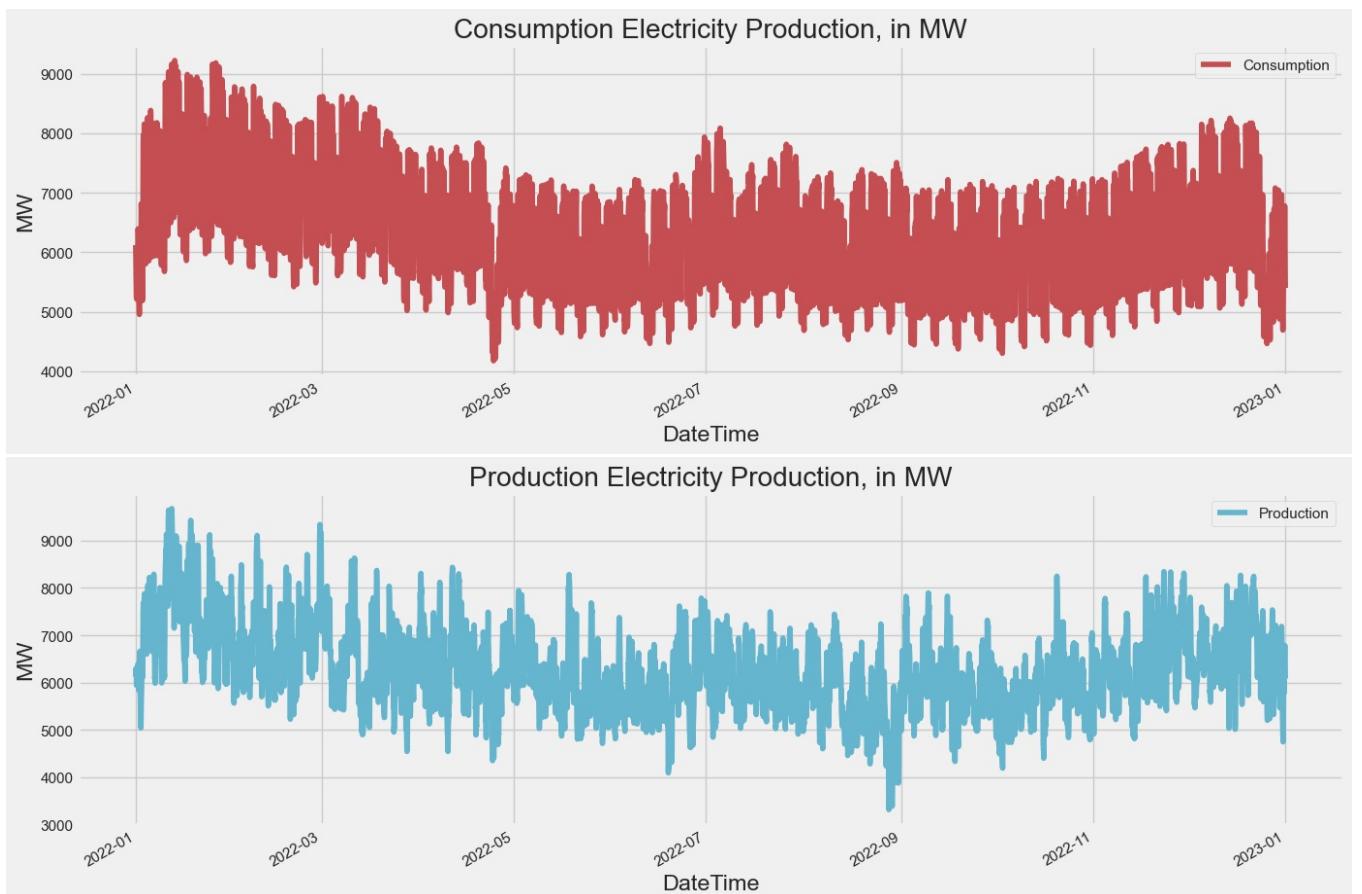
```

In [58]: dfYear = df6["2022-01-01 00:00:00" : "2022-12-31 23:59:59"]
productionTypes = ['Consumption', 'Production']

colorMap = {"Consumption": "r", "Production": "c"}

for productionType in productionTypes:
    dfYear[[productionType]].plot(style="-", figsize=(15, 5), title=f"{productionType} Electricity Production",
    plt.ylabel('MW')
    plt.show()

```



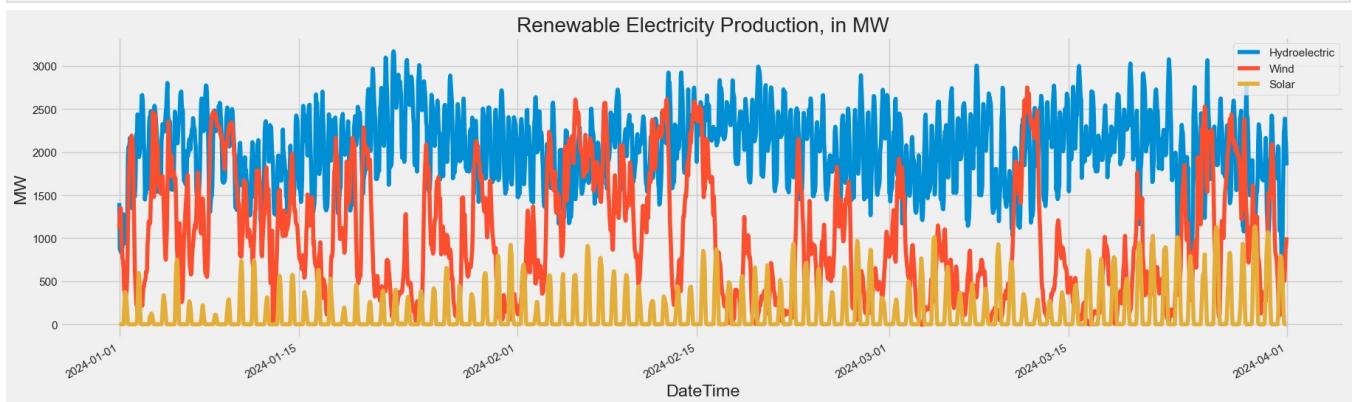
## Specific features and Specific date plot

```
In [59]: df7 = df[['DateTime','Consumption','Production',"Hydroelectric", "Wind", "Solar"]]
df7 = df7.set_index('DateTime')
df7.head()
```

```
Out[59]:
```

|                     | Consumption | Production | Hydroelectric | Wind | Solar |
|---------------------|-------------|------------|---------------|------|-------|
| DateTime            |             |            |               |      |       |
| 2019-01-01 00:00:00 | 6352        | 6527       | 1383          | 79   | 0     |
| 2019-01-01 01:00:00 | 6116        | 5701       | 1112          | 96   | 0     |
| 2019-01-01 02:00:00 | 5873        | 5676       | 1030          | 142  | 0     |
| 2019-01-01 03:00:00 | 5682        | 5603       | 972           | 191  | 0     |
| 2019-01-01 04:00:00 | 5557        | 5454       | 960           | 159  | 0     |

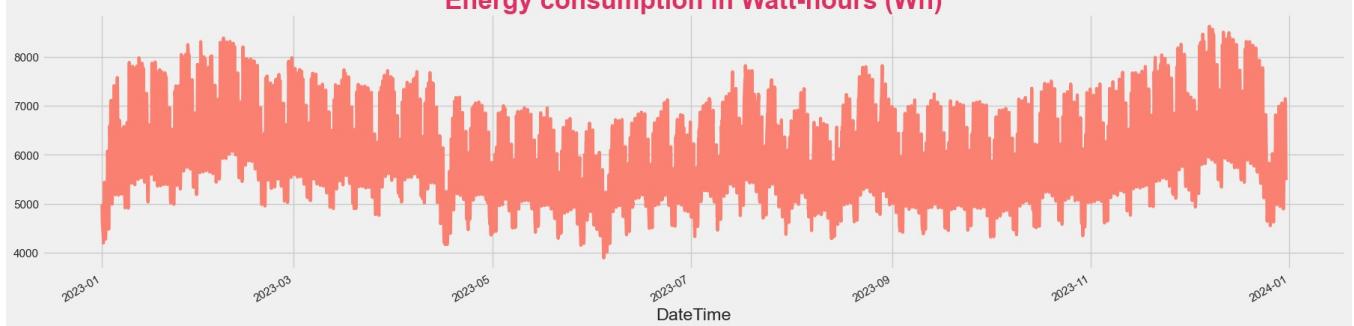
```
In [60]: df_winter = df7[["Hydroelectric", "Wind", "Solar"]][["2024-01-01" : "2024-03-31"]]
df_winter.plot(style="-", figsize=(20, 6), title=f"Renewable Electricity Production, in MW")
plt.ylabel('MW')
plt.show()
```



## Specific Date Plot

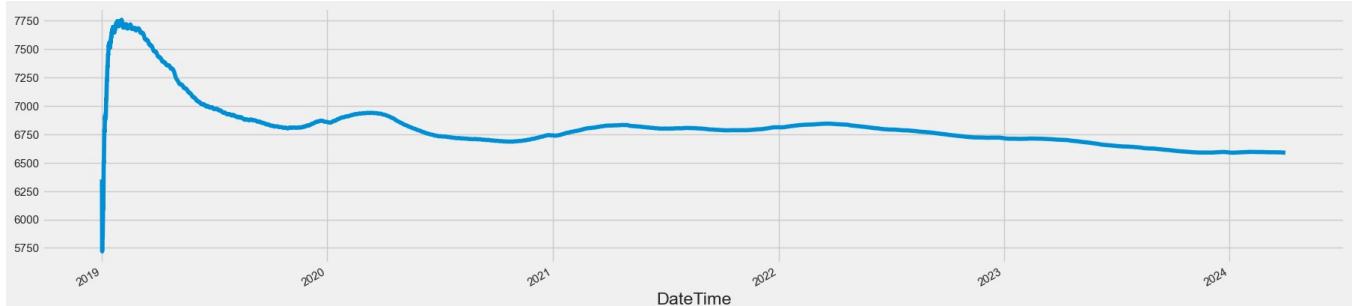
```
In [61]: df_2023 = df7["Consumption"][(df7["Consumption"].index >= '2023-01-01') & (df7["Consumption"].index < '2023-12-31')]
df_2023.plot(figsize=(20,5), lw=3,color = '#FA8072')
plt.title('Energy consumption in Watt-hours (Wh)', weight='bold', fontsize=25,color = '#DE3163');
```

## Energy consumption in Watt-hours (Wh)

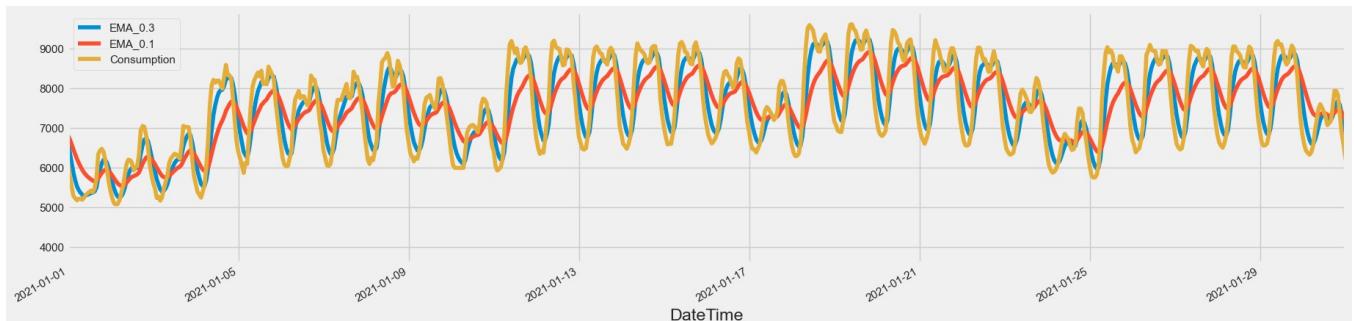


## Moving Average

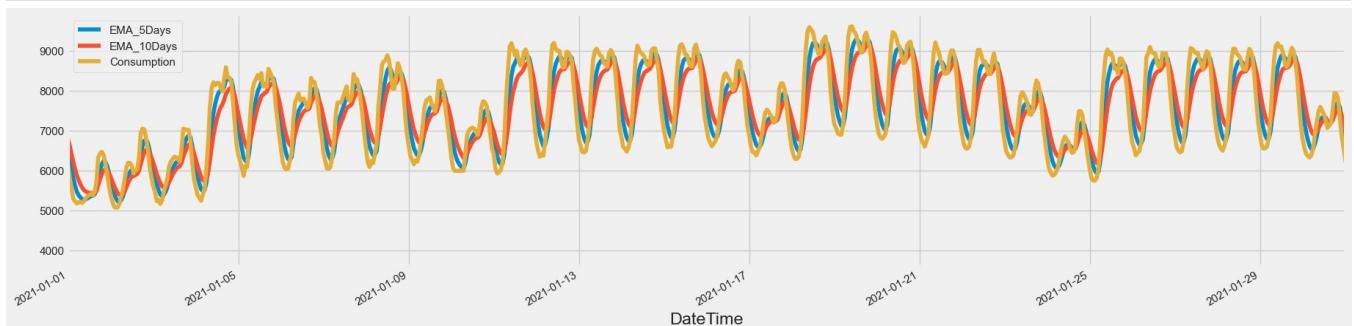
```
In [62]: # Simple Moving Average  
df7["Consumption"].expanding().mean().plot(figsize=(20,5));
```



```
In [63]: # Moving Average (Smoothing Factor 0.1)  
df7['EMA_0.1'] = df7['Consumption'].ewm(alpha=0.1, adjust=False).mean()  
df7['EMA_0.3'] = df7['Consumption'].ewm(alpha=0.3, adjust=False).mean()  
df7[['EMA_0.3', 'EMA_0.1', 'Consumption']].plot(figsize=(20,5), xlim=['2021-01-01', '2021-01-31']);
```



```
In [64]: # Span  
df7['EMA_5Days'] = df7['Consumption'].ewm(span=5, adjust=False).mean()  
df7['EMA_10Days'] = df7['Consumption'].ewm(span=10, adjust=False).mean()  
df7[['EMA_5Days', 'EMA_10Days', 'Consumption']].plot(figsize=(20,5), xlim=['2021-01-01', '2021-01-31']);
```



## adfuller Test

```
In [65]: from statsmodels.tsa.stattools import adfuller  
  
def adf_test(series):  
    result=adfuller(series)  
    print('ADF Statistics: {}'.format(result[0]))  
    print('p- value: {}'.format(result[1]))  
    if result[1] <= 0.05:  
        print("strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root a  
    else:  
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary")
```

```
In [66]: adf_test(df7['Consumption'])

ADF Statistics: -13.926470911140704
p- value: 5.18213926141328e-26
strong evidence against the null hypothesis, reject the null hypothesis. Data has no unit root and is stationary
```

```
In [67]: df2.head(2)
```

```
Out[67]:
```

|                     | Consumption | Production | Nuclear | Wind | Hydroelectric | Oil and Gas | Coal | Solar | Biomass |
|---------------------|-------------|------------|---------|------|---------------|-------------|------|-------|---------|
|                     | DateTime    |            |         |      |               |             |      |       |         |
| 2019-01-01 00:00:00 | 6352        | 6527       | 1395    | 79   | 1383          | 1896        | 1744 | 0     | 30      |
| 2019-01-01 01:00:00 | 6116        | 5701       | 1393    | 96   | 1112          | 1429        | 1641 | 0     | 30      |

## Correlation

```
In [68]: # Computing Correlation Matrices
from scipy.stats.stats import pearsonr
from scipy.stats.stats import spearmanr
from scipy.stats.stats import kendalltau

corr_p = df2[['Consumption', 'Nuclear', 'Coal', 'Solar']].corr(method='pearson')
print('Pearson correlation matrix')
print(corr_p)
print("\n")
corr_s = df2[['Consumption', 'Nuclear', 'Coal', 'Solar']].corr(method='spearman')
print('Spearman correlation matrix')
print(corr_s)
```

```
Pearson correlation matrix
      Consumption   Nuclear     Coal     Solar
Consumption  1.000000  0.127617  0.461859  0.126887
Nuclear      0.127617  1.000000 -0.008732 -0.128614
Coal         0.461859 -0.008732  1.000000 -0.066005
Solar        0.126887 -0.128614 -0.066005  1.000000
```

```
Spearman correlation matrix
      Consumption   Nuclear     Coal     Solar
Consumption  1.000000  0.010702  0.451529  0.254557
Nuclear      0.010702  1.000000 -0.084438 -0.316097
Coal         0.451529 -0.084438  1.000000 -0.025992
Solar        0.254557 -0.316097 -0.025992  1.000000
```

```
In [114]: df2.head()
```

```
Out[114]:
```

|                     | Consumption | Production | Nuclear | Wind | Hydroelectric | Oil and Gas | Coal | Solar | Biomass |
|---------------------|-------------|------------|---------|------|---------------|-------------|------|-------|---------|
|                     | DateTime    |            |         |      |               |             |      |       |         |
| 2019-01-01 00:00:00 | 6352        | 6527       | 1395    | 79   | 1383          | 1896        | 1744 | 0     | 30      |
| 2019-01-01 01:00:00 | 6116        | 5701       | 1393    | 96   | 1112          | 1429        | 1641 | 0     | 30      |
| 2019-01-01 02:00:00 | 5873        | 5676       | 1393    | 142  | 1030          | 1465        | 1616 | 0     | 30      |
| 2019-01-01 03:00:00 | 5682        | 5603       | 1397    | 191  | 972           | 1455        | 1558 | 0     | 30      |
| 2019-01-01 04:00:00 | 5557        | 5454       | 1393    | 159  | 960           | 1454        | 1458 | 0     | 30      |

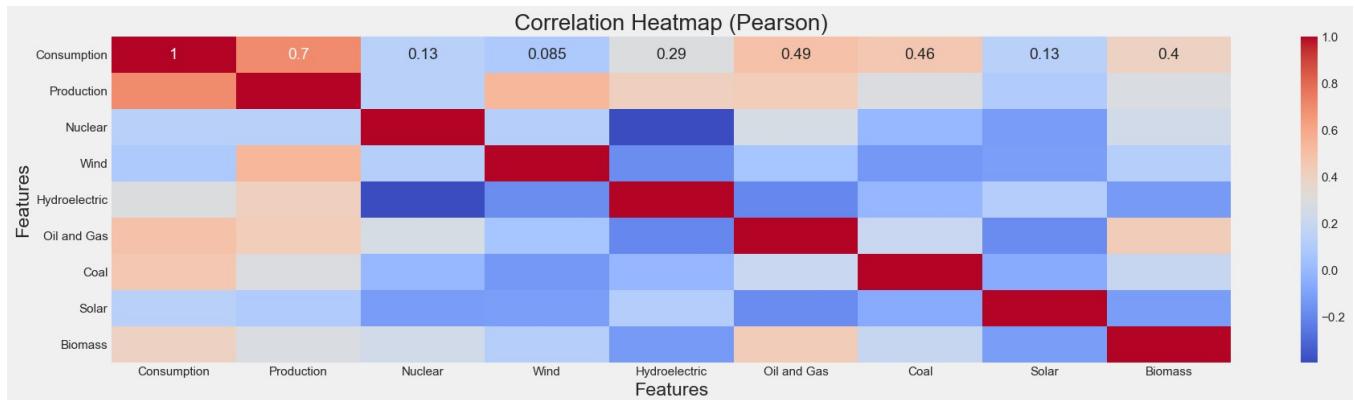
```
In [115]: import seaborn as sns
import matplotlib.pyplot as plt

corr_mat = df2.corr(method='pearson')

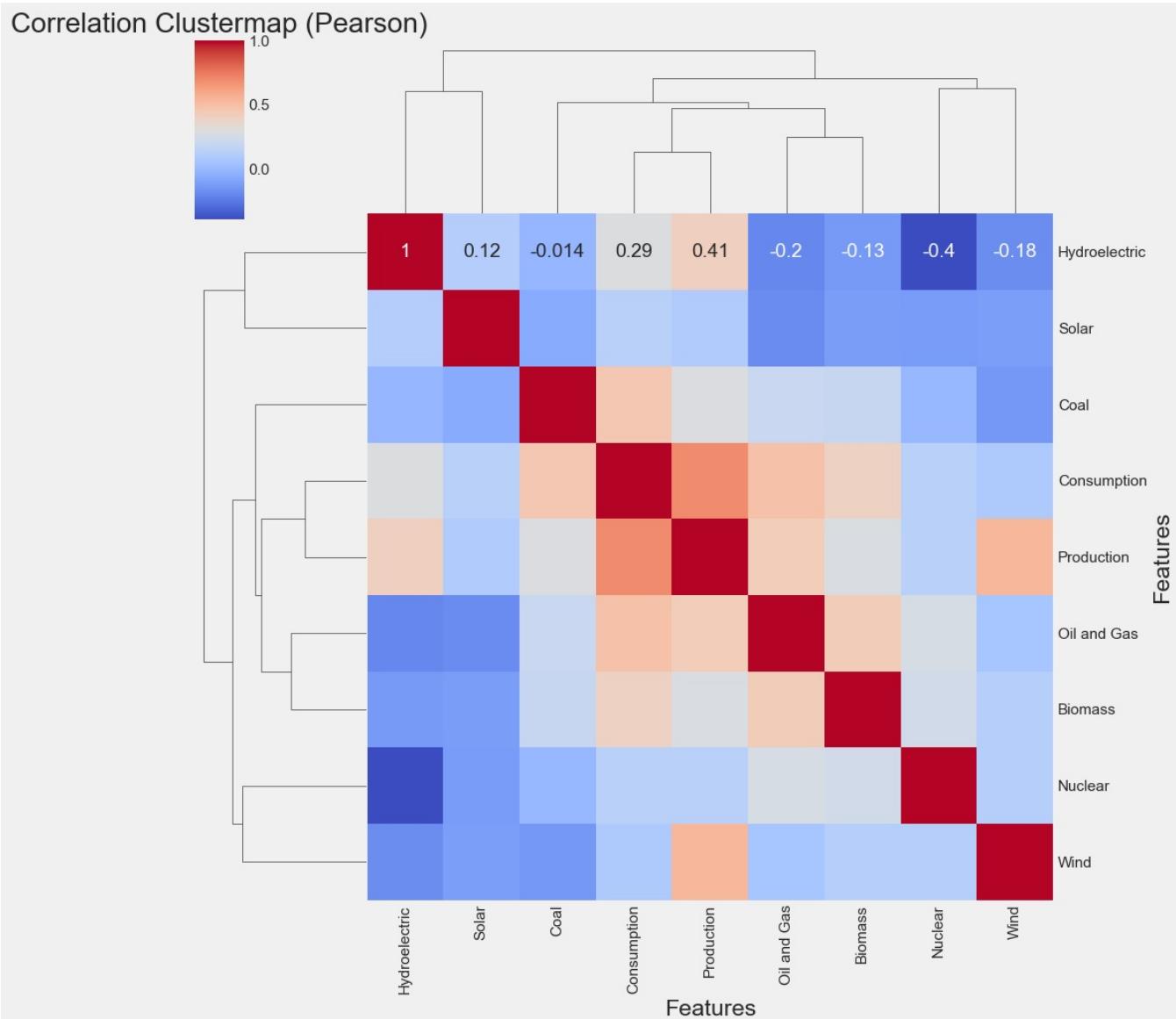
# Add annotation for correlation values
sns.heatmap(corr_mat, annot=True, cmap='coolwarm')

# Additional customizations (optional)
plt.title("Correlation Heatmap (Pearson)")
plt.xlabel("Features")
plt.ylabel("Features")

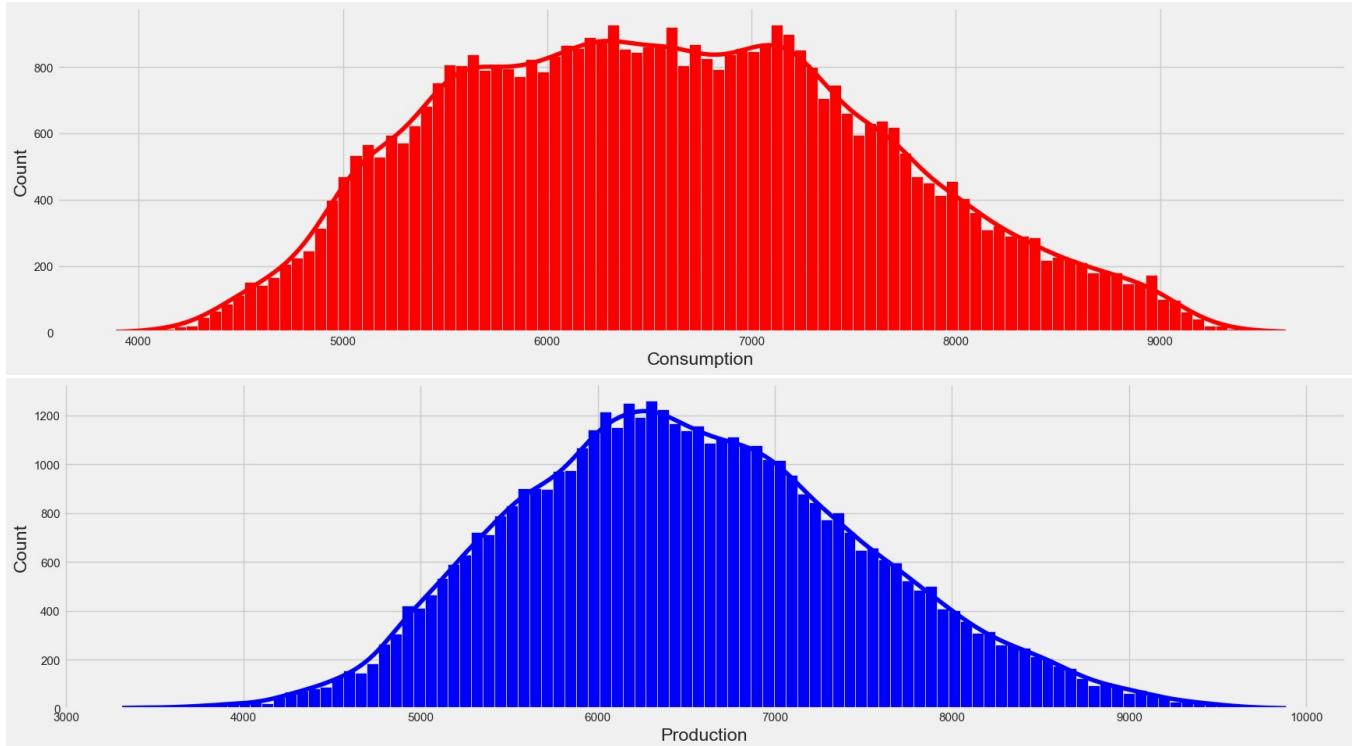
plt.show()
```



```
In [116]:  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
corr_mat = df2.corr(method='pearson')  
  
# Create a clustermap with annotations  
g = sns.clustermap(corr_mat, annot=True, cmap='coolwarm')  
  
# Add title and labels  
plt.title("Correlation Clustermap (Pearson)")  
g.ax_heatmap.set_xlabel("Features")  
g.ax_heatmap.set_ylabel("Features")  
  
plt.show()
```



```
In [109]:  
# Create the first subplot with red color  
ax1 = sns.displot(data=df, x="Consumption", kde=True, bins=100, color="red",  
                  facecolor="#FF0000", height=5, aspect=3.5)  
  
# Create the second subplot on the same figure with blue color  
ax2 = sns.displot(data=df, x="Production", kde=True, bins=100, color="blue",  
                  facecolor="#0000FF", height=5, aspect=3.5)
```



## Differencing

```
In [71]: monthly_data = df7.resample('M')['Consumption'].agg('mean')
monthly_data = monthly_data.reset_index()
df8 = monthly_data.set_index('DateTime')
df8.head()
```

```
Out[71]: Consumption
```

| Date       | Consumption |
|------------|-------------|
| 2019-01-31 | 7752.168011 |
| 2019-02-28 | 7605.071429 |
| 2019-03-31 | 6992.792732 |
| 2019-04-30 | 6652.491667 |
| 2019-05-31 | 6379.282258 |

```
In [72]: ## Use Techniques Differencing
df8['Consumption First Difference']= df8['Consumption'] - df8['Consumption'].shift(1)
df8.head()
```

```
Out[72]: Consumption Consumption First Difference
```

| Date       | Consumption | Consumption First Difference |
|------------|-------------|------------------------------|
| 2019-01-31 | 7752.168011 | NaN                          |
| 2019-02-28 | 7605.071429 | -147.096582                  |
| 2019-03-31 | 6992.792732 | -612.278696                  |
| 2019-04-30 | 6652.491667 | -340.301066                  |
| 2019-05-31 | 6379.282258 | -273.209409                  |

```
In [73]: adf_test(df8['Consumption First Difference'].dropna())
```

ADF Statistics: -2.1379721278691894  
p-value: 0.22952237850384044  
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary

```
In [74]: ### 12 months shift
## Use Techniques Differencing
df8['Consumption 12 Difference']=df8['Consumption']-df8['Consumption'].shift(12)
df8.head()
```

Out[74]:

Consumption Consumption First Difference Consumption 12 Difference

| DateTime   |             | Consumption | Consumption First Difference | Consumption 12 Difference |
|------------|-------------|-------------|------------------------------|---------------------------|
| 2019-01-31 | 7752.168011 |             | NaN                          | NaN                       |
| 2019-02-28 | 7605.071429 |             | -147.096582                  | NaN                       |
| 2019-03-31 | 6992.792732 |             | -612.278696                  | NaN                       |
| 2019-04-30 | 6652.491667 |             | -340.301066                  | NaN                       |
| 2019-05-31 | 6379.282258 |             | -273.209409                  | NaN                       |

In [75]:

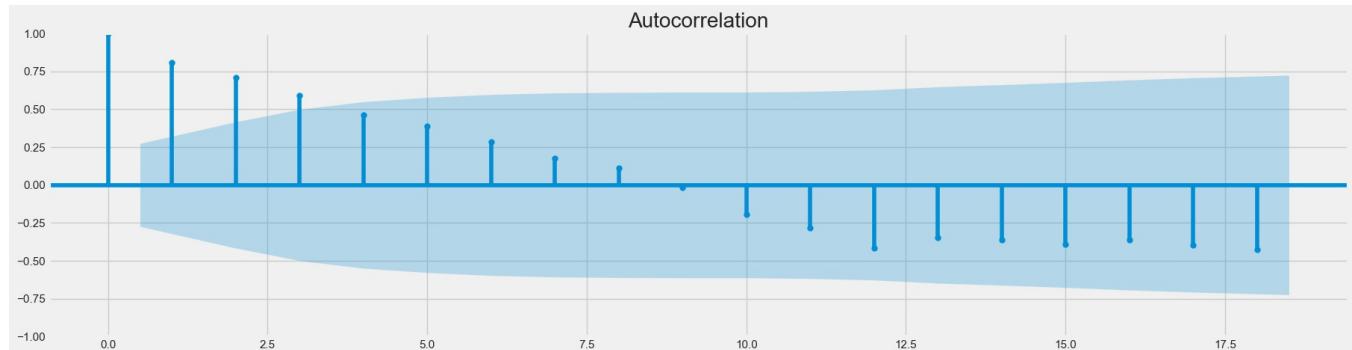
adf\_test(df8['Consumption 12 Difference'].dropna())

ADF Statistics: -2.2278541415816235

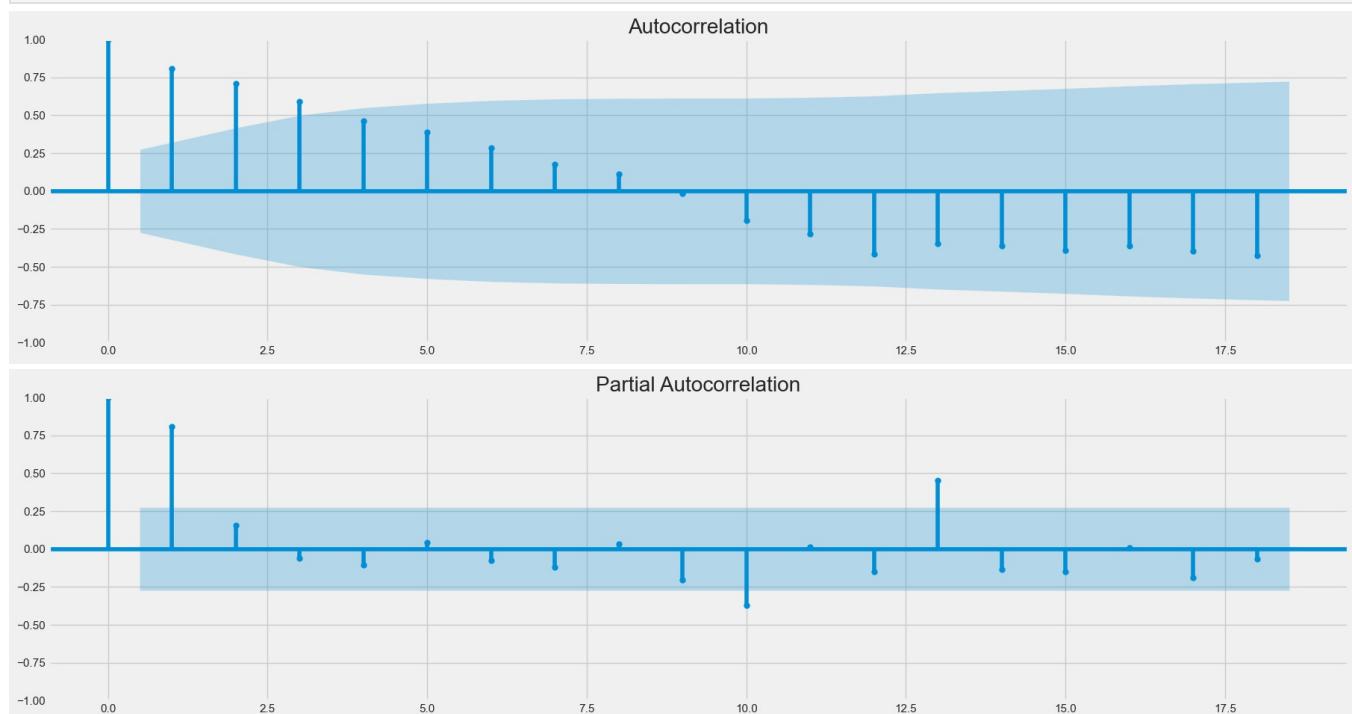
p- value: 0.19629175142304744

weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary

In [76]:

from statsmodels.graphics.tsaplots import plot\_acf,plot\_pacf  
acf = plot\_acf(df8['Consumption 12 Difference'].dropna())

In [77]:

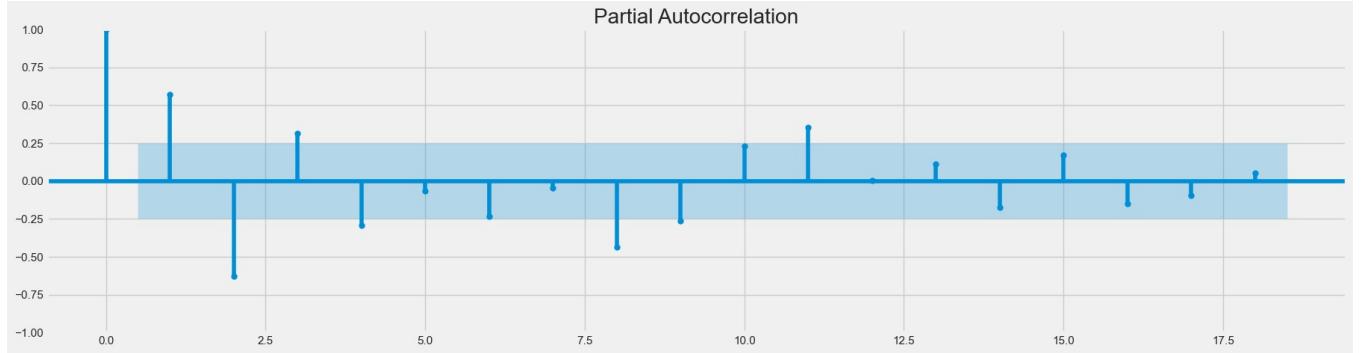
acf12 = plot\_acf(df8['Consumption 12 Difference'].dropna())  
pacf12 = plot\_pacf(df8['Consumption 12 Difference'].dropna())

In [78]:

## Use Techniques Differencing  
df8['Consumption 2 Difference']=df8['Consumption']-df8['Consumption'].shift(2)  
display(df8.head(3))  
  
result = plot\_pacf(df8["Consumption 2 Difference"].dropna())

Consumption Consumption First Difference Consumption 12 Difference Consumption 2 Difference

| DateTime   |             | Consumption | Consumption First Difference | Consumption 12 Difference | Consumption 2 Difference |
|------------|-------------|-------------|------------------------------|---------------------------|--------------------------|
| 2019-01-31 | 7752.168011 |             | NaN                          | NaN                       | NaN                      |
| 2019-02-28 | 7605.071429 |             | -147.096582                  | NaN                       | NaN                      |
| 2019-03-31 | 6992.792732 |             | -612.278696                  | NaN                       | -759.375279              |



```
In [79]: df9 = df8[['Consumption']]
df9.head()
```

Out[79]: Consumption

|            | Date | Consumption |
|------------|------|-------------|
| 2019-01-31 |      | 7752.168011 |
| 2019-02-28 |      | 7605.071429 |
| 2019-03-31 |      | 6992.792732 |
| 2019-04-30 |      | 6652.491667 |
| 2019-05-31 |      | 6379.282258 |

```
In [80]: from datetime import datetime, timedelta
train_dataset_end=datetime(2023,6,1)
test_dataset_end=datetime(2024,4,1)
```

```
In [81]: train_data=df9[:train_dataset_end]
test_data=df9[train_dataset_end+timedelta(days=1):test_dataset_end]
##prediction
pred_start_date=test_data.index[0]
pred_end_date=test_data.index[-1]
```

```
In [82]: train_data.head()
```

Out[82]: Consumption

|            | Date | Consumption |
|------------|------|-------------|
| 2019-01-31 |      | 7752.168011 |
| 2019-02-28 |      | 7605.071429 |
| 2019-03-31 |      | 6992.792732 |
| 2019-04-30 |      | 6652.491667 |
| 2019-05-31 |      | 6379.282258 |

```
In [83]: test_data.head()
```

Out[83]: Consumption

|            | Date | Consumption |
|------------|------|-------------|
| 2023-06-30 |      | 5651.966667 |
| 2023-07-31 |      | 6053.303763 |
| 2023-08-31 |      | 5929.065860 |
| 2023-09-30 |      | 5678.083333 |
| 2023-10-31 |      | 5758.722148 |

```
In [84]: import warnings
warnings.filterwarnings("ignore")

from statsmodels.tsa.arima.model import ARIMA
model_ARIMA=ARIMA(train_data['Consumption'],order=(10,2,0))
```

```
In [85]: model_Arima_fit=model_ARIMA.fit()
model_Arima_fit.summary()
```

Out[85]:

SARIMAX Results

| <b>Dep. Variable:</b>          | Consumption             | <b>No. Observations:</b> | 53       |       |          |          |
|--------------------------------|-------------------------|--------------------------|----------|-------|----------|----------|
| <b>Model:</b>                  | ARIMA(10, 2, 0)         | <b>Log Likelihood</b>    | -354.120 |       |          |          |
| <b>Date:</b>                   | Thu, 22 Aug 2024        | <b>AIC</b>               | 730.241  |       |          |          |
| <b>Time:</b>                   | 16:48:41                | <b>BIC</b>               | 751.491  |       |          |          |
| <b>Sample:</b>                 | 01-31-2019 - 05-31-2023 | <b>HQIC</b>              | 738.361  |       |          |          |
| <b>Covariance Type:</b>        | opg                     |                          |          |       |          |          |
|                                | coef                    | std err                  | z        | P> z  | [0.025   | 0.975]   |
| ar.L1                          | -0.4217                 | 0.103                    | -4.111   | 0.000 | -0.623   | -0.221   |
| ar.L2                          | -0.4829                 | 0.119                    | -4.045   | 0.000 | -0.717   | -0.249   |
| ar.L3                          | -0.6508                 | 0.110                    | -5.899   | 0.000 | -0.867   | -0.435   |
| ar.L4                          | -0.5284                 | 0.158                    | -3.339   | 0.001 | -0.839   | -0.218   |
| ar.L5                          | -0.4235                 | 0.132                    | -3.205   | 0.001 | -0.683   | -0.164   |
| ar.L6                          | -0.5286                 | 0.154                    | -3.438   | 0.001 | -0.830   | -0.227   |
| ar.L7                          | -0.5325                 | 0.154                    | -3.463   | 0.001 | -0.834   | -0.231   |
| ar.L8                          | -0.3378                 | 0.167                    | -2.026   | 0.043 | -0.665   | -0.011   |
| ar.L9                          | -0.4171                 | 0.111                    | -3.748   | 0.000 | -0.635   | -0.199   |
| ar.L10                         | -0.6336                 | 0.140                    | -4.526   | 0.000 | -0.908   | -0.359   |
| <b>sigma2</b>                  | 5.081e+04               | 1.36e+04                 | 3.742    | 0.000 | 2.42e+04 | 7.74e+04 |
| <b>Ljung-Box (L1) (Q):</b>     | 3.19                    | <b>Jarque-Bera (JB):</b> | 0.39     |       |          |          |
| <b>Prob(Q):</b>                | 0.07                    | <b>Prob(JB):</b>         | 0.82     |       |          |          |
| <b>Heteroskedasticity (H):</b> | 0.47                    | <b>Skew:</b>             | -0.15    |       |          |          |
| <b>Prob(H) (two-sided):</b>    | 0.13                    | <b>Kurtosis:</b>         | 2.70     |       |          |          |

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [86]:

```
##prediction
pred_start_date=test_data.index[0]
pred_end_date=test_data.index[-1]
print(pred_start_date)
print(pred_end_date)
```

2023-06-30 00:00:00  
2024-03-31 00:00:00

In [87]:

```
pred=model_Arima_fit.predict(start=pred_start_date,end=pred_end_date)
residuals=test_data['Consumption']-pred
```

In [88]:

```
pred.to_frame()
```

Out[88]:

|            | predicted_mean |
|------------|----------------|
| 2023-06-30 | 5782.510258    |
| 2023-07-31 | 5941.097195    |
| 2023-08-31 | 5762.945329    |
| 2023-09-30 | 5441.840938    |
| 2023-10-31 | 5659.123037    |
| 2023-11-30 | 5858.474618    |
| 2023-12-31 | 5870.687607    |
| 2024-01-31 | 6003.153652    |
| 2024-02-29 | 5911.563521    |
| 2024-03-31 | 5778.196544    |

In [89]:

```
residuals.to_frame()
```

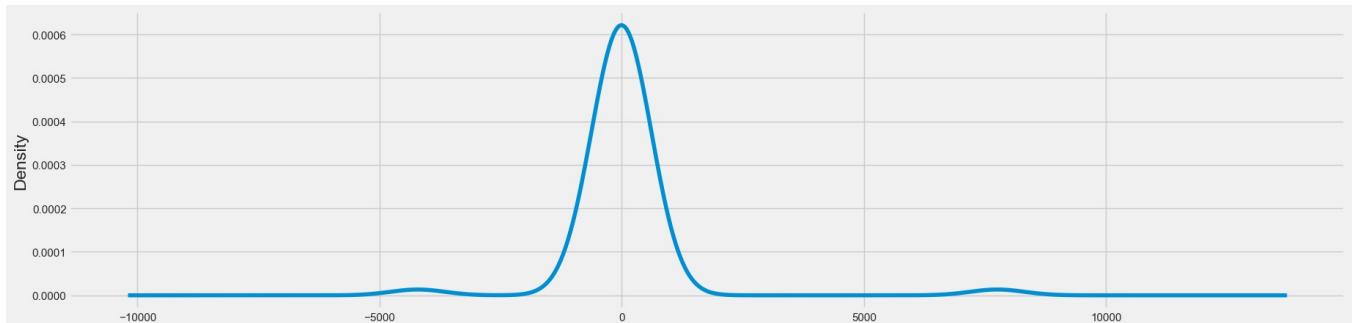
Out[89]:

0

| DateTime   |             |
|------------|-------------|
| 2023-06-30 | -130.543592 |
| 2023-07-31 | 112.206569  |
| 2023-08-31 | 166.120531  |
| 2023-09-30 | 236.242395  |
| 2023-10-31 | 99.599110   |
| 2023-11-30 | 546.747604  |
| 2023-12-31 | 752.983092  |
| 2024-01-31 | 832.097692  |
| 2024-02-29 | 566.848835  |
| 2024-03-31 | 496.099553  |

In [90]:

```
model_Arima_fit.resid.plot(kind='kde', figsize=(20,5));
```

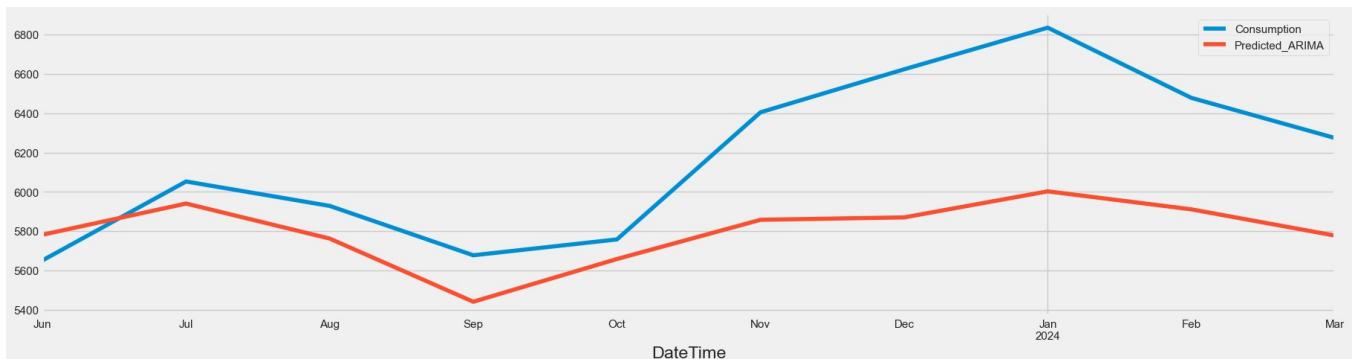


In [91]:

```
test_data['Predicted_ARIMA']=pred
```

In [92]:

```
test_data[['Consumption','Predicted_ARIMA']].plot(figsize=(20,5));
```



In [93]:

```
## create a SARIMA model
from statsmodels.tsa.statespace.sarimax import SARIMAX
model_SARIMA=SARIMAX(train_data['Consumption'],order=(3,0,5),seasonal_order=(0,1,0,12))
model_SARIMA_fit=model_SARIMA.fit()
model_SARIMA_fit.summary()
```

Out[93]:

## SARIMAX Results

| Dep. Variable:          | Consumption                     | No. Observations: | 53        |       |           |          |
|-------------------------|---------------------------------|-------------------|-----------|-------|-----------|----------|
| Model:                  | SARIMAX(3, 0, 5)x(0, 1, [], 12) | Log Likelihood    | -290.962  |       |           |          |
| Date:                   | Thu, 22 Aug 2024                | AIC               | 599.923   |       |           |          |
| Time:                   | 16:48:43                        | BIC               | 615.345   |       |           |          |
| Sample:                 | 01-31-2019 - 05-31-2023         | HQIC              | 605.539   |       |           |          |
| Covariance Type:        | opg                             |                   |           |       |           |          |
|                         | coef                            | std err           | z         | P> z  | [0.025    | 0.975]   |
| ar.L1                   | 0.5045                          | 0.482             | 1.047     | 0.295 | -0.440    | 1.449    |
| ar.L2                   | 0.5952                          | 0.577             | 1.032     | 0.302 | -0.535    | 1.725    |
| ar.L3                   | -0.3886                         | 0.225             | -1.728    | 0.084 | -0.829    | 0.052    |
| ma.L1                   | 0.0356                          | 3644.131          | 9.77e-06  | 1.000 | -7142.331 | 7142.402 |
| ma.L2                   | -0.3636                         | 4242.153          | -8.57e-05 | 1.000 | -8314.831 | 8314.103 |
| ma.L3                   | 0.8613                          | 1335.581          | 0.001     | 0.999 | -2616.830 | 2618.553 |
| ma.L4                   | -0.2701                         | 1933.450          | -0.000    | 1.000 | -3789.763 | 3789.223 |
| ma.L5                   | -0.5306                         | 0.533             | -0.995    | 0.320 | -1.575    | 0.514    |
| sigma2                  | 5.707e+04                       | 0.130             | 4.38e+05  | 0.000 | 5.71e+04  | 5.71e+04 |
| Ljung-Box (L1) (Q):     | 2.06                            | Jarque-Bera (JB): | 3.61      |       |           |          |
| Prob(Q):                | 0.15                            | Prob(JB):         | 0.16      |       |           |          |
| Heteroskedasticity (H): | 1.35                            | Skew:             | 0.62      |       |           |          |
| Prob(H) (two-sided):    | 0.58                            | Kurtosis:         | 3.77      |       |           |          |

Warnings:

- [1] Covariance matrix calculated using the outer product of gradients (complex-step).
- [2] Covariance matrix is singular or near-singular, with condition number 6.26e+25. Standard errors may be unstable.

In [94]:

```
##prediction
pred_start_date=test_data.index[0]
pred_end_date=test_data.index[-1]
print(pred_start_date)
print(pred_end_date)
```

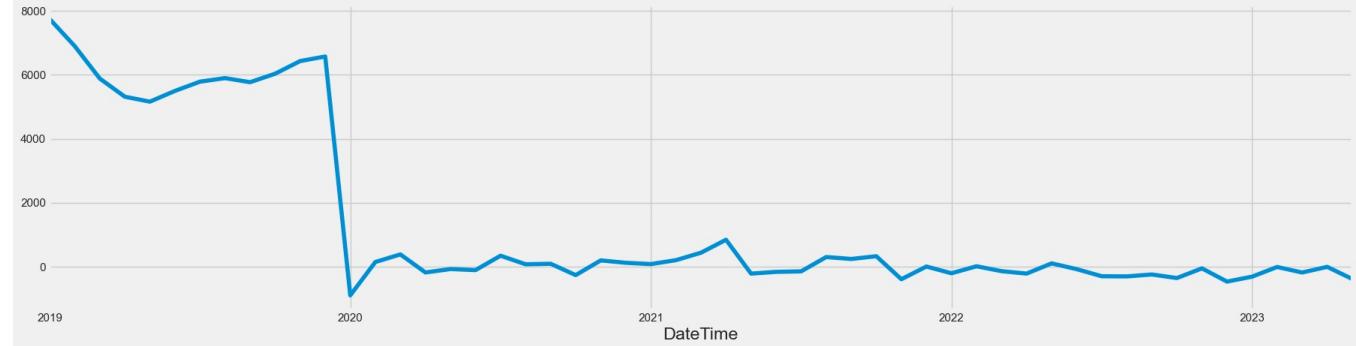
2023-06-30 00:00:00  
2024-03-31 00:00:00

In [95]:

```
pred_Sarima=model_SARIMA_fit.predict(start=datetime(2023,6,1),end=datetime(2024,4,1))
residuals=test_data['Consumption']-pred_Sarima
```

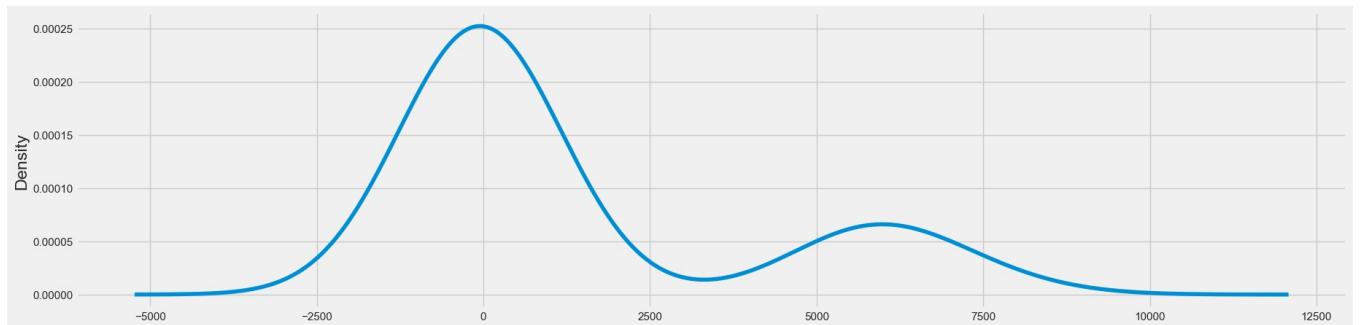
In [96]:

```
model_SARIMA_fit.resid.plot(figsize=(20,5));
```



In [97]:

```
model_SARIMA_fit.resid.plot(kind='kde', figsize=(20,5));
```



```
In [98]: test_data['Predicted_SARIMA']=pred_Sarima
```

```
In [99]: test_data[['Consumption','Predicted_SARIMA','Predicted_ARIMA']].plot(figsize=(20,5));
```

