Data Understanding

Applying Moving Window Functions

Loading the Dataset

```
import sys
import os
import time

import numpy  as np
import pandas as pd
import xlrd

import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
```

```
elec = pd.read_csv('elx_big data.csv')

elec.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81769 entries, 0 to 81768
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Datetime  81769 non-null  object
 1   load_no1  81769 non-null  float64
 2   wind_no1  81769 non-null  float64
 3   temp_no1  81769 non-null  float64
 4   load_no2  81769 non-null  float64
 5   wind_no2  81769 non-null  float64
 6   temp_no2  81769 non-null  float64
 7   load_no3  81769 non-null  float64
 8   wind_no3  81769 non-null  float64
 9   temp_no3  81769 non-null  float64
 10  load_no4  81769 non-null  float64
```

```
 11  wind_no4  81769 non-null  float64
 12  temp_no4  81769 non-null  float64
 13  load_no5  81769 non-null  float64
 14  wind_no5  81769 non-null  float64
 15  temp_no5  81769 non-null  float64
dtypes: float64(15), object(1)
memory usage: 10.0+ MB
```

Data cleaning

```
elec.duplicated().sum()
# We have no duplicates or missing values
```

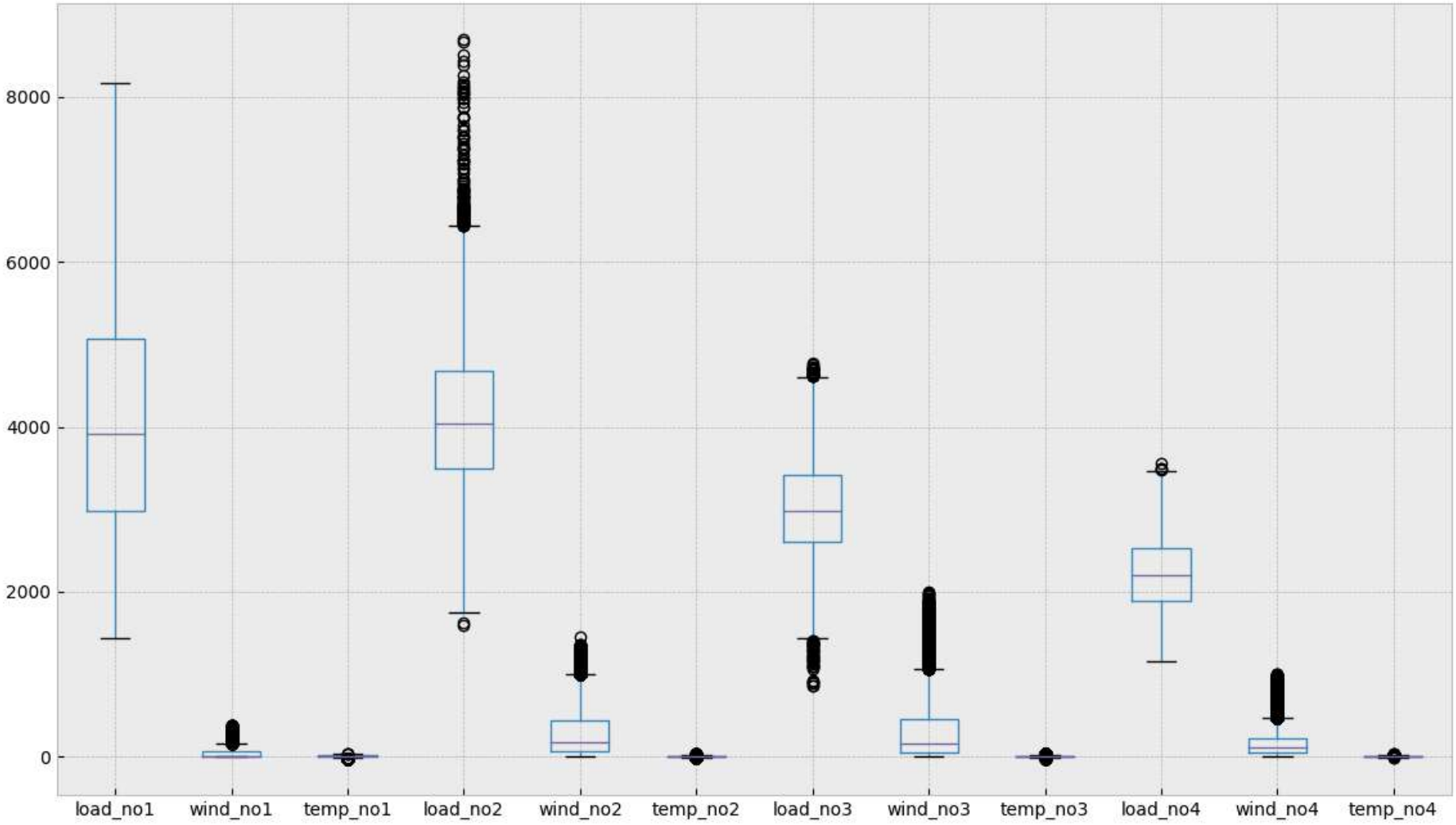
```
0
```

Checking for Outliers

```
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']

numeric = elec.select_dtypes(include=numerics)
```

```
plt.style.use('bmh')

# plotting outliers for the numeric columns
#
_t, cols = pd.DataFrame.boxplot(numeric.iloc[:, :-3], return_type='both', figsize=(14,8))

outliers = [flier.get_ydata() for flier in cols['fliers']]
out_list = [i.tolist() for i in outliers]

print(f" Outliers:\n {out_list}")
```

```
Outliers:
[[], [183.0, 189.0, 185.0, 186.0, 191.0, 195.0, 193.0, 187.0, 180.0, 183.0, 183.0, 171.0, 172.0, 174.0, 170.0, 172.0, 179.0, 173.0, 190.0, 178.0, 183.0, 183.0,
```



```
# Function for counting number of outliers in our data columns and cheking the percentage for each
# ----
#
def detect_outlier(data):
    outliers=[]
    threshold=3
    mean_1 = np.mean(data)
    std_1 =np.std(data)

    for y in data:
        z_score= (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            outliers.append(y)
    return outliers
```

```
# Counting number of outliers in our data columns and cheking the percentage for each column using z-score
#
#
for col in numeric:
    rows, columns = numeric.shape
    percent_coefficient = float(100 / rows)
    outliers = detect_outlier(numeric[col])
    outliers_count = len(outliers)
    outliers_percentage = outliers_count * percent_coefficient
    print(f"{col} has {outliers_count} outliers in total, which is {outliers_percentage:.2}% of data")
```

```
load_no1 has 15 outliers in total, which is 0.018% of data
wind_no1 has 1867 outliers in total, which is 2.3% of data
temp_no1 has 118 outliers in total, which is 0.14% of data
load_no2 has 157 outliers in total, which is 0.19% of data
wind_no2 has 484 outliers in total, which is 0.59% of data
temp_no2 has 182 outliers in total, which is 0.22% of data
load_no3 has 73 outliers in total, which is 0.089% of data
wind_no3 has 1545 outliers in total, which is 1.9% of data
temp_no3 has 260 outliers in total, which is 0.32% of data
load_no4 has 11 outliers in total, which is 0.013% of data
wind_no4 has 1647 outliers in total, which is 2.0% of data
temp_no4 has 121 outliers in total, which is 0.15% of data
load_no5 has 270 outliers in total, which is 0.33% of data
wind_no5 has 3429 outliers in total, which is 4.2% of data
temp_no5 has 182 outliers in total, which is 0.22% of data
```

```
# Getting ouliers from our dataframe using a z-test
#
from scipy import stats

z = np.abs(stats.zscore(numeric))


# Dropping and Confirming that our outliers have been dropped from the dataset.
#
df_o = numeric[(z < 3).all(axis=1)]

print(f"Previous dataframe size : {numeric.shape[0]}")
print(f"New dataframe size: {df_o.shape[0]}")
```

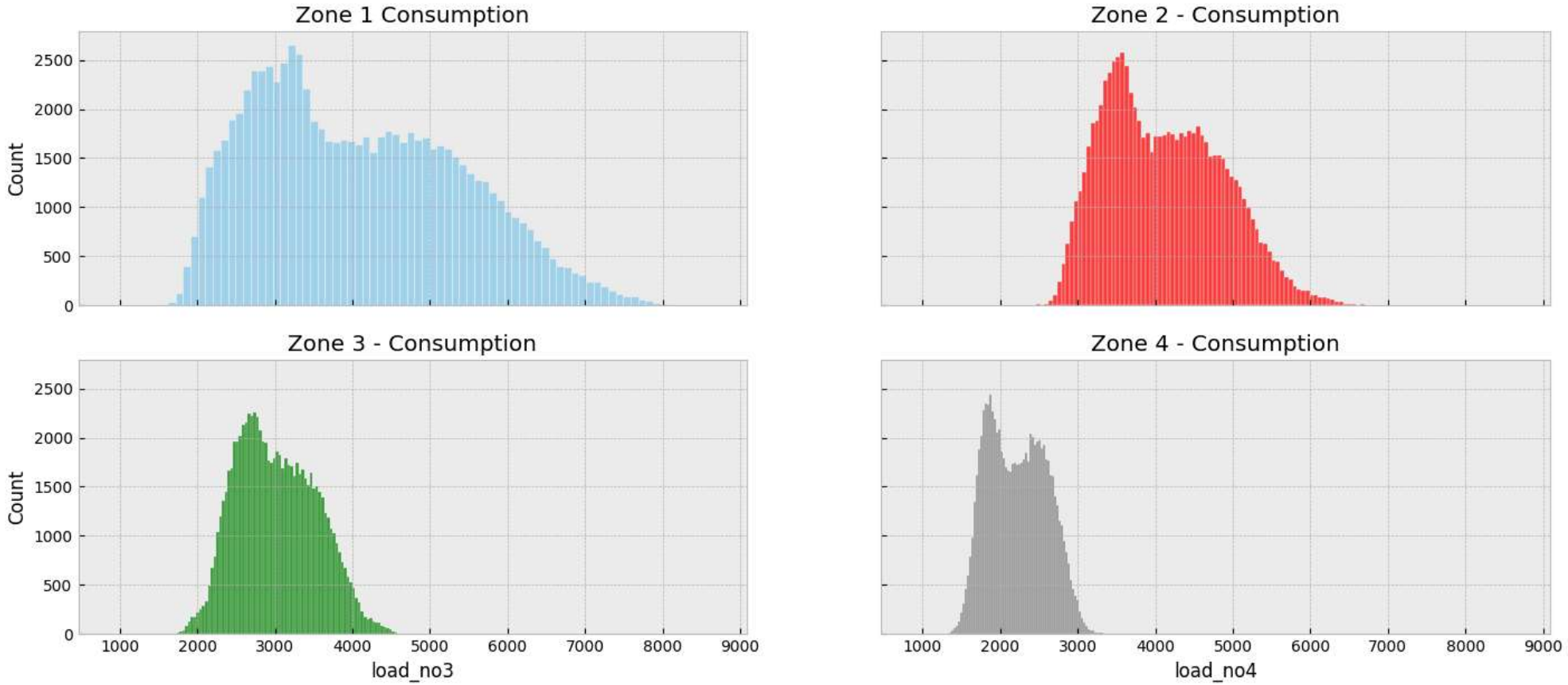
```
Previous dataframe size : 81769
New dataframe size: 73059
```

Exploratory Data Analysis

Vizualization

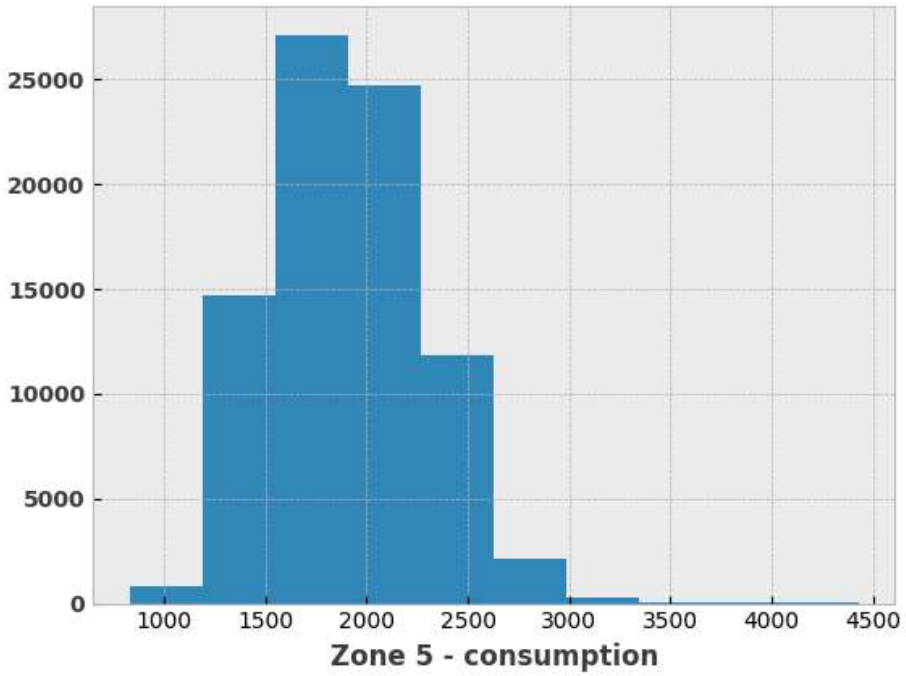
```
# plt.title("Consumption distribution", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
#Plot
fig,axes = plt.subplots(2,2,figsize=(17,7),sharex=True,sharey=True)

sns.histplot(elec["load_no1"],color="skyblue", ax=axes[0,0]).set_title("Zone 1 Consumption")
sns.histplot(elec["load_no2"],color="red", ax=axes[0,1]).set_title("Zone 2 - Consumption")
sns.histplot(elec["load_no3"],color="green", ax=axes[1,0]).set_title("Zone 3 - Consumption")
sns.histplot(elec["load_no4"],color="gray", ax=axes[1,1]).set_title("Zone 4 - Consumption")
```

```
Text(0.5, 1.0, 'Zone 4 - Consumption')
```




```
fig = plt.plot(figsize=(10,5))

plt.hist(elec["load_no5"])

plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Zone 5 - consumption",alpha=0.75, weight="bold")
plt.ylabel("",alpha=0.75, weight="bold")
```

```
Text(0, 0.5, '')
```




```
#print(elec['Datetime'].dtype)
elec['Datetime'] = pd.to_datetime(elec['Datetime'], format='%d/%m/%Y %H:%M')
```

We can now split the 'datetime' column into two columns.

These columns are:

- Date
- Time

The reason as to why we are splitting this column is due to the fact that we need to analyze how electricity consumption varies on an hourly basis.

```
# Separate date and time components
elec['Date'] = elec['Datetime'].dt.date
elec['Time'] = elec['Datetime'].dt.time
```

```
# Extract individual components
elec['year'] = elec['Datetime'].dt.year
elec['month'] = elec['Datetime'].dt.month
elec['Q'] = elec['Datetime'].dt.quarter
elec['Week_Number'] = elec['Datetime'].dt.strftime('%U')
elec["Dayofyear"] = elec['Datetime'].dt.dayofyear
elec['Dayofmonth'] = elec['Datetime'].dt.day
elec['Day'] = elec['Datetime'].dt.dayofweek
elec['hour'] = elec['Datetime'].dt.hour
elec['is_weekend'] = elec['Datetime'].dt.weekday.isin([5, 6])  # 5=Saturday, 6=Sunday
elec['is_weekday'] = ~elec['is_weekend']



# Display the updated DataFrame
print(elec)
```

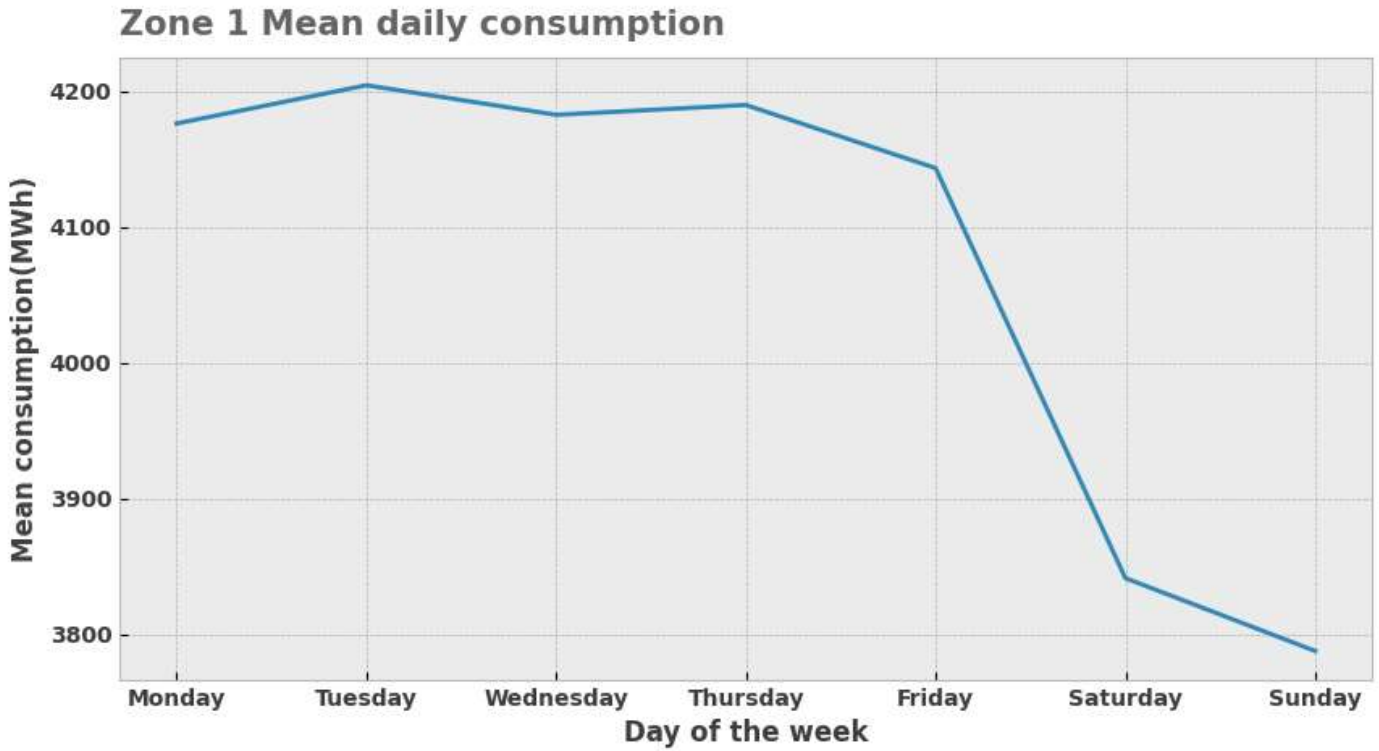
```
                 Datetime  load_no1  wind_no1  temp_no1  load_no2  wind_no2  \
0     2015-01-01 00:00:00    4659.0       0.0      -2.5    4139.0     156.0
1     2015-01-01 01:00:00    4552.0       0.0      -2.0    4039.0     159.0
2     2015-01-01 02:00:00    4469.0       0.0      -1.5    3956.0     149.0
3     2015-01-01 03:00:00    4442.0       0.0       0.0    3900.0     144.0
4     2015-01-01 04:00:00    4488.0       0.0       0.0    3915.0     149.0
...                   ...       ...       ...       ...       ...       ...
81764 2024-04-29 20:00:00    3715.0     206.0       7.0    4109.0     816.0
81765 2024-04-29 21:00:00    3464.0     201.0       7.0    3905.0     878.0
81766 2024-04-29 22:00:00    3464.0     201.0       7.0    3905.0     878.0
81767 2024-04-29 23:00:00    3464.0     201.0       7.0    3905.0     878.0
81768 2024-04-30 00:00:00    3464.0     201.0       7.0    3905.0     878.0

       temp_no2  load_no3  wind_no3  temp_no3  ...  year  month  Q  \
0           6.5    2370.0     259.0       3.0  ...  2015      1  1
1           7.0    2307.0     234.0       3.0  ...  2015      1  1
2           7.0    2273.0     200.0       3.0  ...  2015      1  1
3           7.0    2286.0     192.0       2.0  ...  2015      1  1
4           8.0    2333.0     192.0       2.0  ...  2015      1  1
...         ...       ...       ...       ...  ...   ...    ... ..
81764      10.0    2942.0     652.0       6.0  ...  2024      4  2
81765      10.0    2894.0     650.0       5.5  ...  2024      4  2
81766      10.0    2894.0     650.0       5.5  ...  2024      4  2
81767      10.0    2894.0     650.0       5.5  ...  2024      4  2
81768      10.0    2894.0     650.0       5.5  ...  2024      4  2

       Week_Number  Dayofyear  Dayofmonth Day hour  is_weekend  is_weekday
0               00          1           1   3    0       False        True
1               00          1           1   3    1       False        True
2               00          1           1   3    2       False        True
3               00          1           1   3    3       False        True
```

```
4              00          1           1   3    4       False        True
...           ...        ...         ...  ..  ...         ...         ...
81764          17        120          29   0   20       False        True
81765          17        120          29   0   21       False        True
81766          17        120          29   0   22       False        True
81767          17        120          29   0   23       False        True
81768          17        121          30   1    0       False        True

[81769 rows x 28 columns]
```

Mean of Daily Electricity Consumption

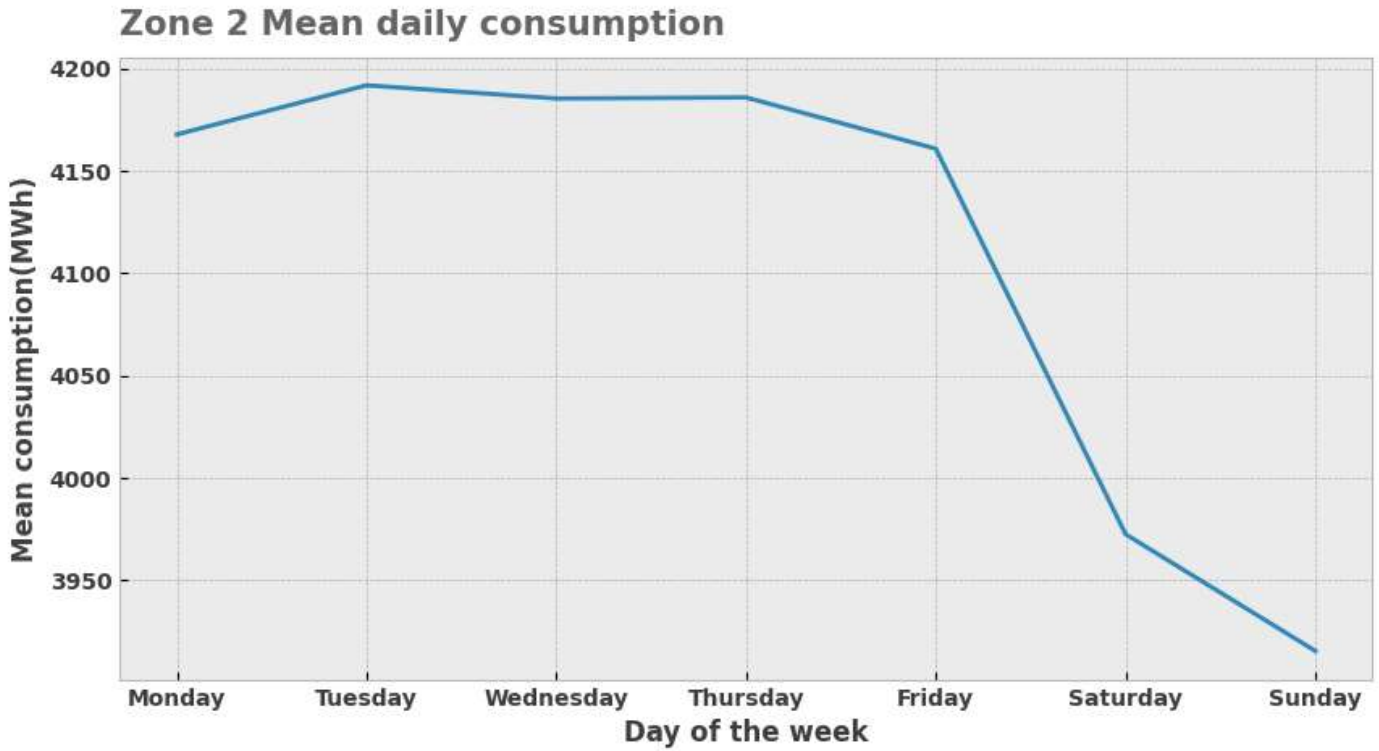
```
#Data prep
mean_per_day = elec.groupby("Day")["load_no1"].agg(["mean"])

#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_day.index,mean_per_day["mean"])

plt.xticks(mean_per_day.index, ["Monday","Tuesday", "Wednesday", "Thursday", "Friday", "Saturday","Sunday"], alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 1 Mean daily consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 1 Mean daily consumption')
```



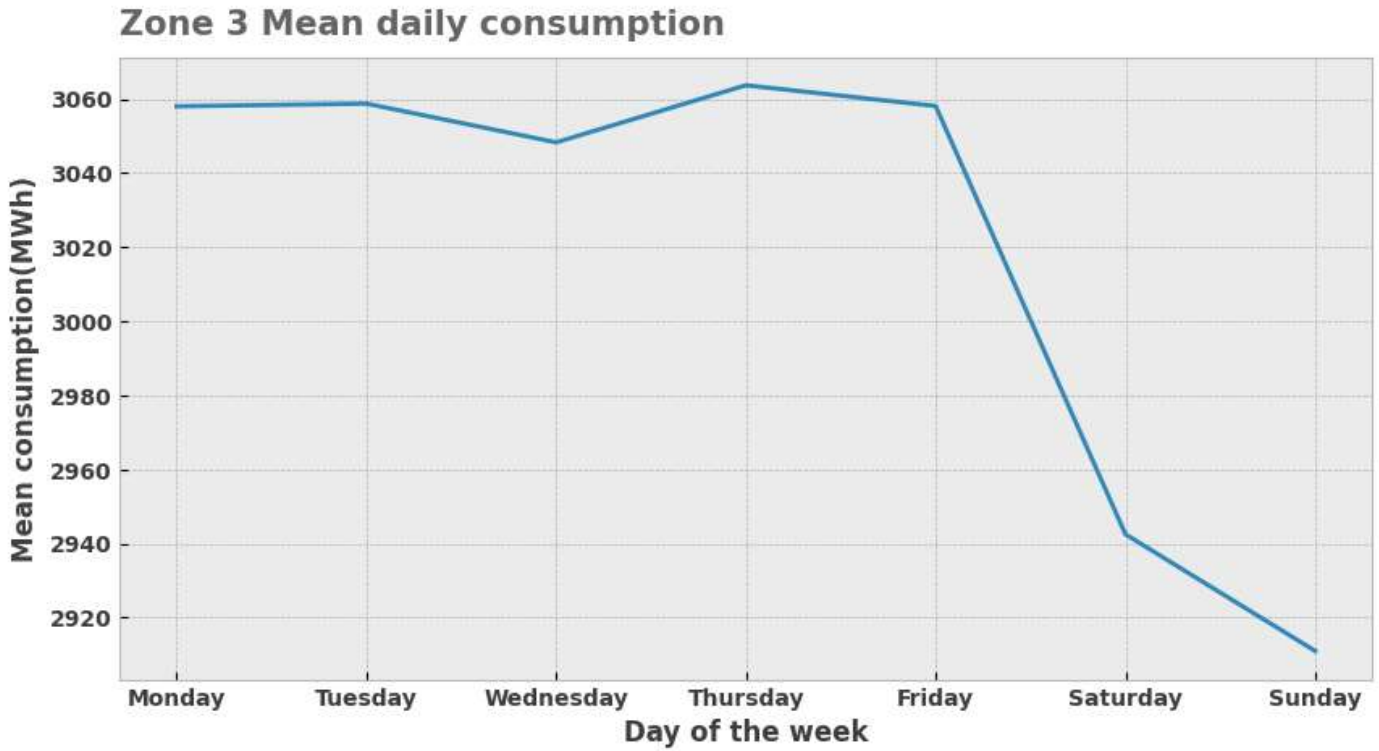

```
#Data prep
mean_per_day = elec.groupby("Day")["load_no2"].agg(["mean"])

#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_day.index,mean_per_day["mean"])

plt.xticks(mean_per_day.index, ["Monday","Tuesday", "Wednesday", "Thursday", "Friday", "Saturday","Sunday"], alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 2 Mean daily consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 2 Mean daily consumption')
```



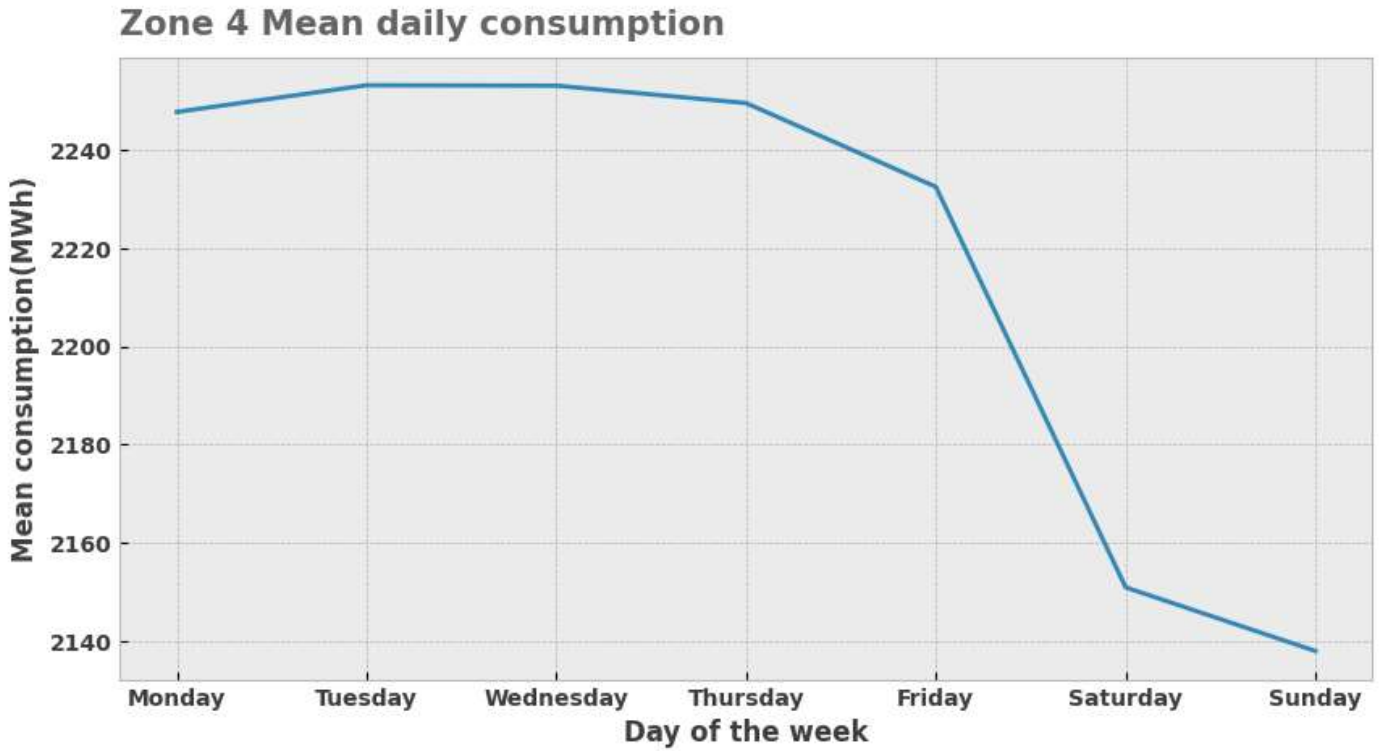

```
#Data prep
mean_per_day = elec.groupby("Day")["load_no3"].agg(["mean"])

#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_day.index,mean_per_day["mean"])

plt.xticks(mean_per_day.index, ["Monday","Tuesday", "Wednesday", "Thursday", "Friday", "Saturday","Sunday"], alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 3 Mean daily consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 3 Mean daily consumption')
```



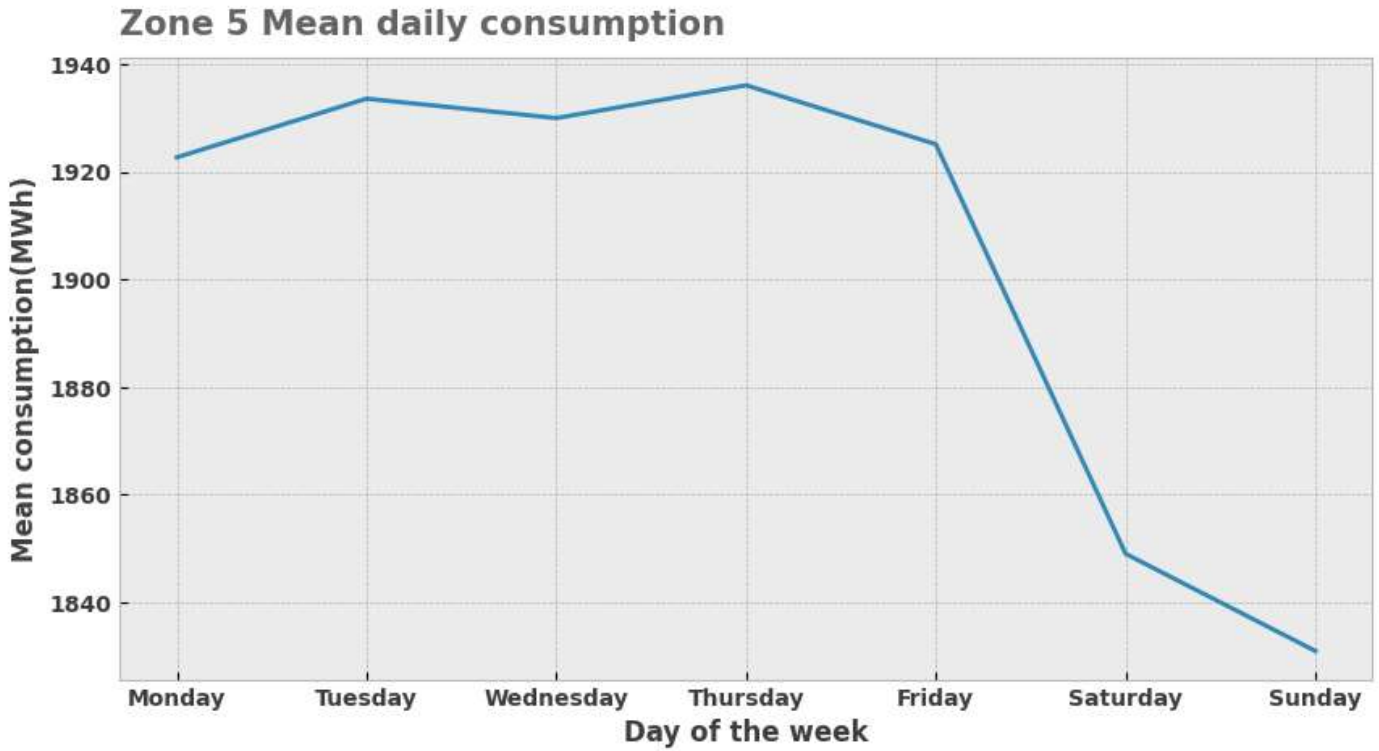

```
#Data prep
mean_per_day = elec.groupby("Day")["load_no4"].agg(["mean"])

#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_day.index,mean_per_day["mean"])

plt.xticks(mean_per_day.index, ["Monday","Tuesday", "Wednesday", "Thursday", "Friday", "Saturday","Sunday"], alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 4 Mean daily consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 4 Mean daily consumption')
```




```
#Data prep
mean_per_day = elec.groupby("Day")["load_no5"].agg(["mean"])

#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_day.index,mean_per_day["mean"])

plt.xticks(mean_per_day.index, ["Monday","Tuesday", "Wednesday", "Thursday", "Friday", "Saturday","Sunday"], alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 5 Mean daily consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

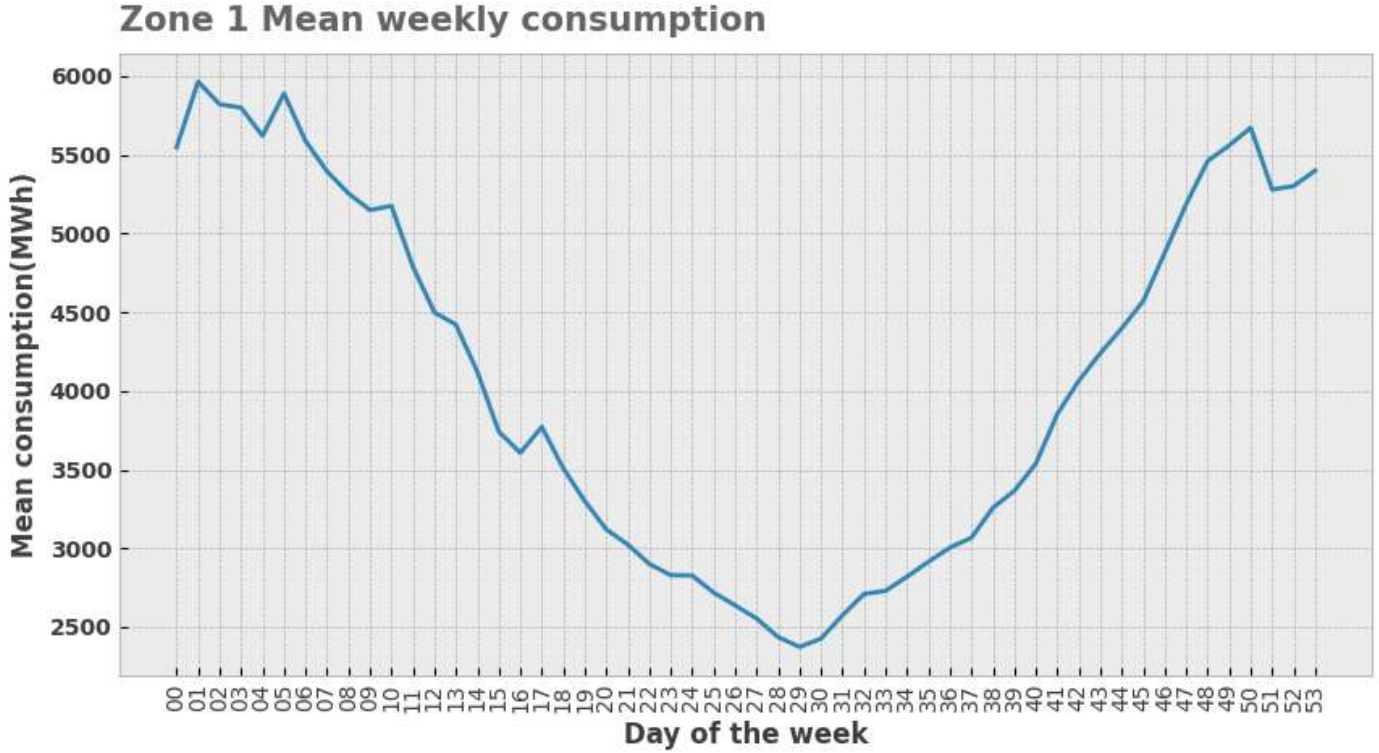
```
Text(0.0, 1.0, 'Zone 5 Mean daily consumption')
```



Mean of Week Electricity Consumption

```
#Data prep
mean_per_week = elec.groupby("Week_Number")["load_no1"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_week.index,mean_per_week["mean"])

plt.xticks(alpha=0.75, rotation = 90)
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 1 Mean weekly consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

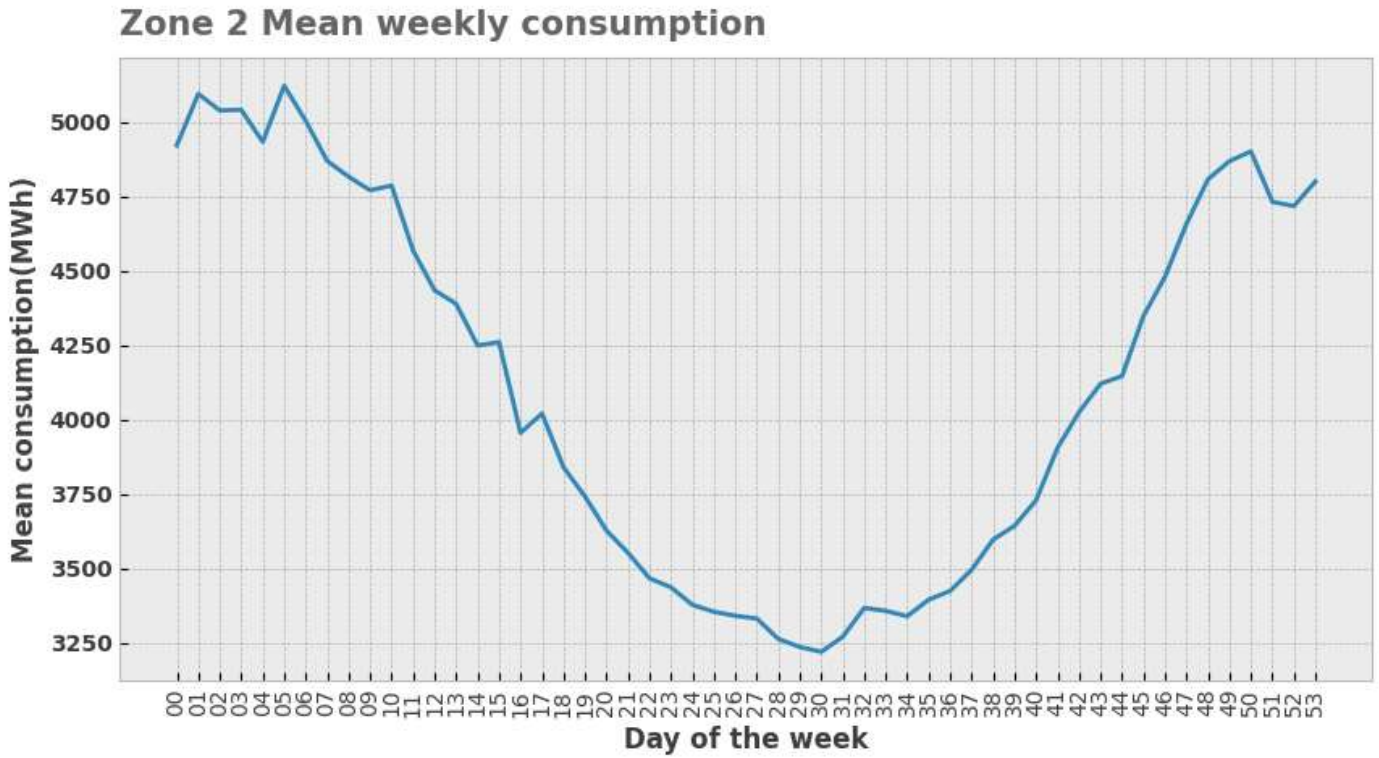
```
Text(0.0, 1.0, 'Zone 1 Mean weekly consumption')
```




```
#Data prep
mean_per_week = elec.groupby("Week_Number")["load_no2"].agg(["mean"])
```

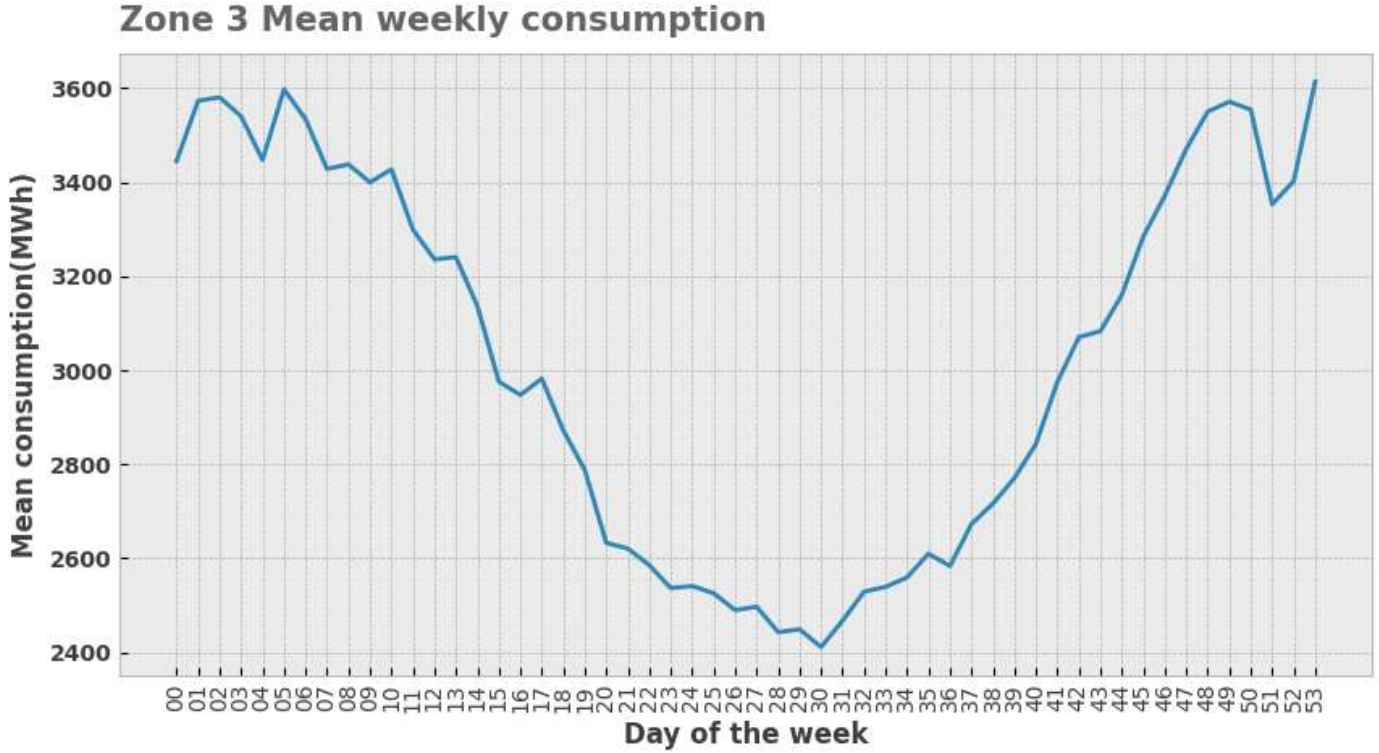
```
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_week.index,mean_per_week["mean"])

plt.xticks(alpha=0.75, rotation = 90)
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 2 Mean weekly consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 2 Mean weekly consumption')
```



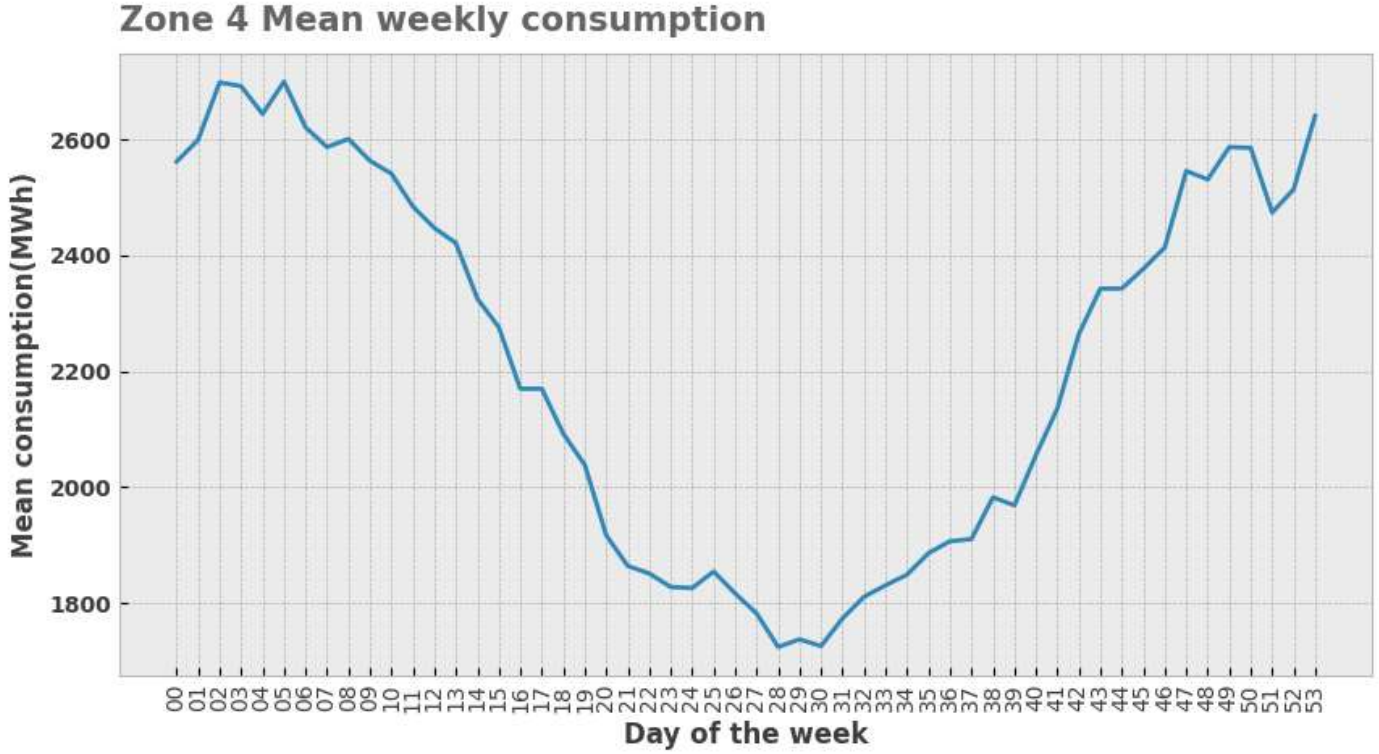
```
#Data prep
mean_per_week = elec.groupby("Week_Number")["load_no3"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_week.index,mean_per_week["mean"])

plt.xticks(alpha=0.75, rotation = 90)
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 3 Mean weekly consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 3 Mean weekly consumption')
```



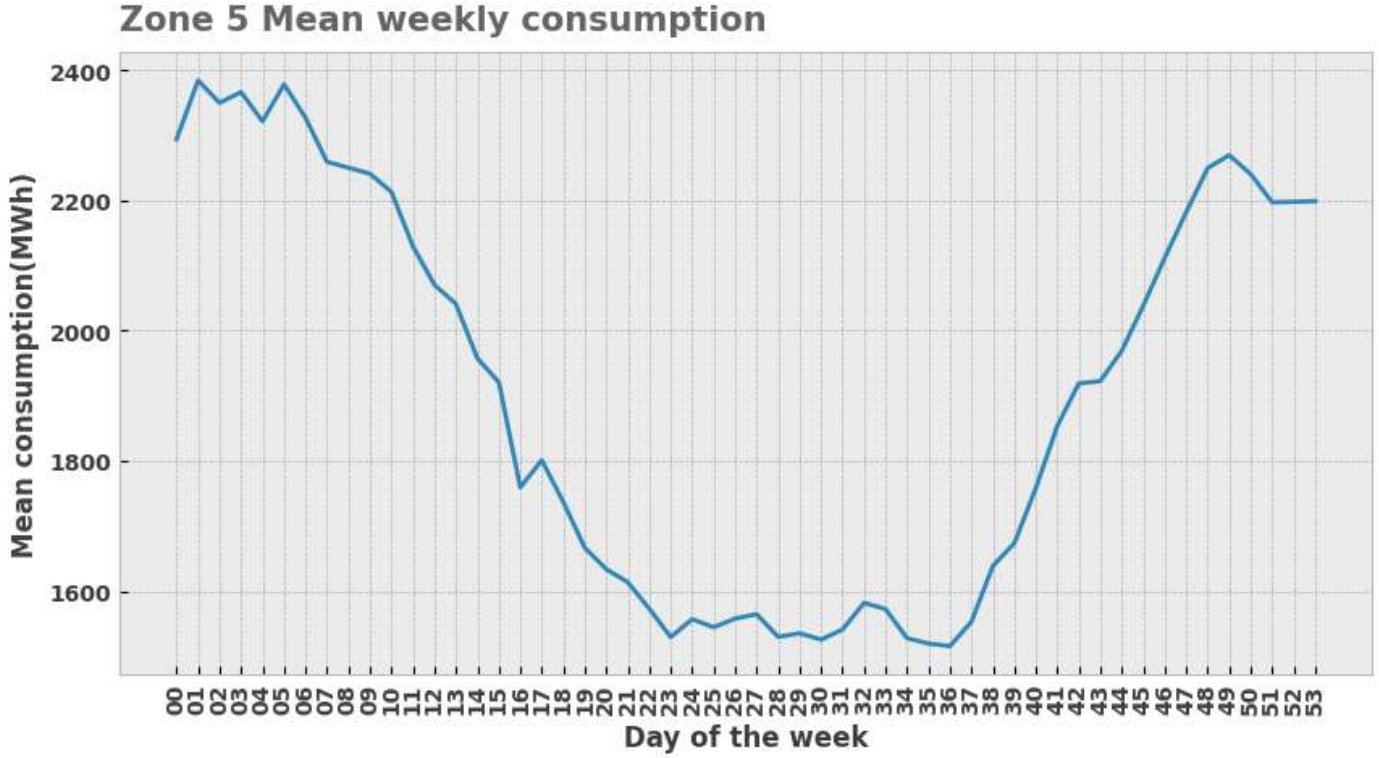
```
#Data prep
mean_per_week = elec.groupby("Week_Number")["load_no4"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_week.index,mean_per_week["mean"])

plt.xticks(alpha=0.75, rotation = 90)
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 4 Mean weekly consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 4 Mean weekly consumption')
```



```
#Data prep
mean_per_week = elec.groupby("Week_Number")["load_no5"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_week.index,mean_per_week["mean"])

plt.xticks(alpha=0.75, rotation = 90, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 5 Mean weekly consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

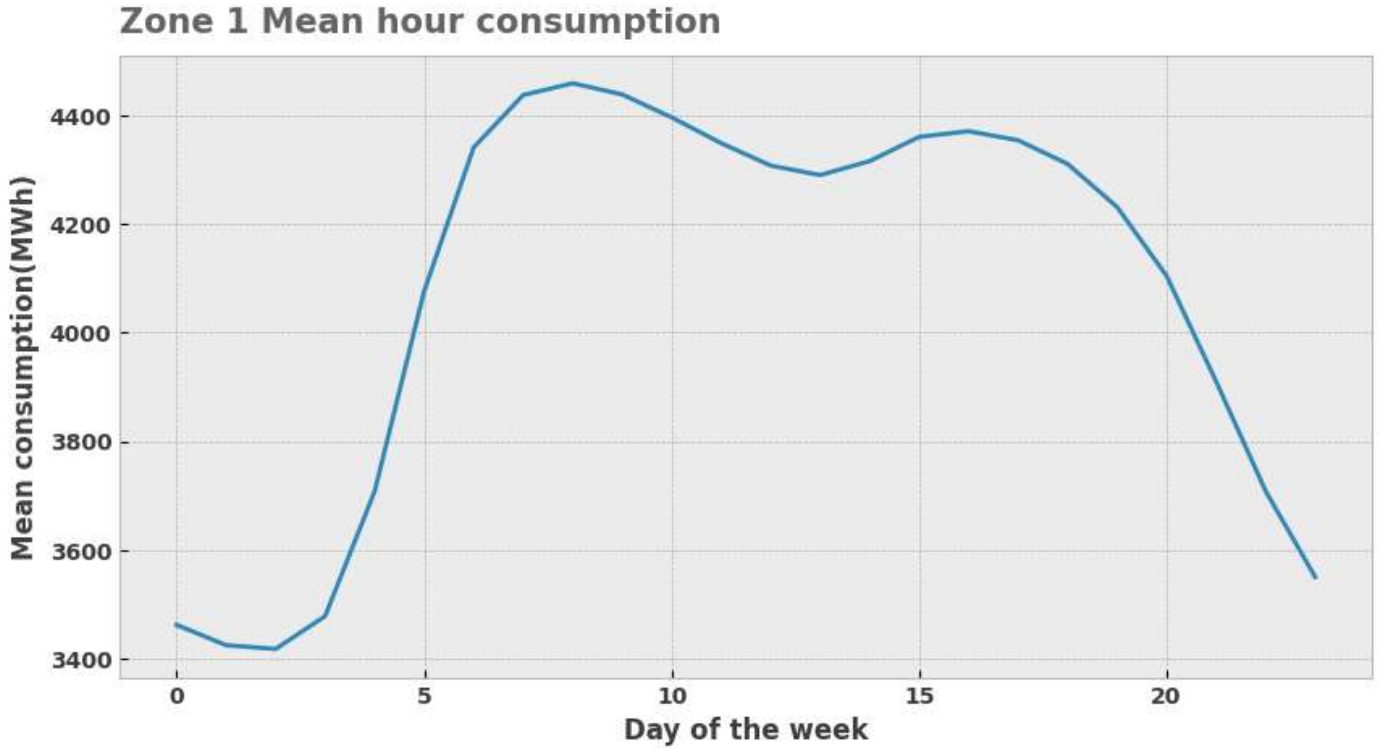
```
Text(0.0, 1.0, 'Zone 5 Mean weekly consumption')
```



Mean of Hourly Electricity Consumption

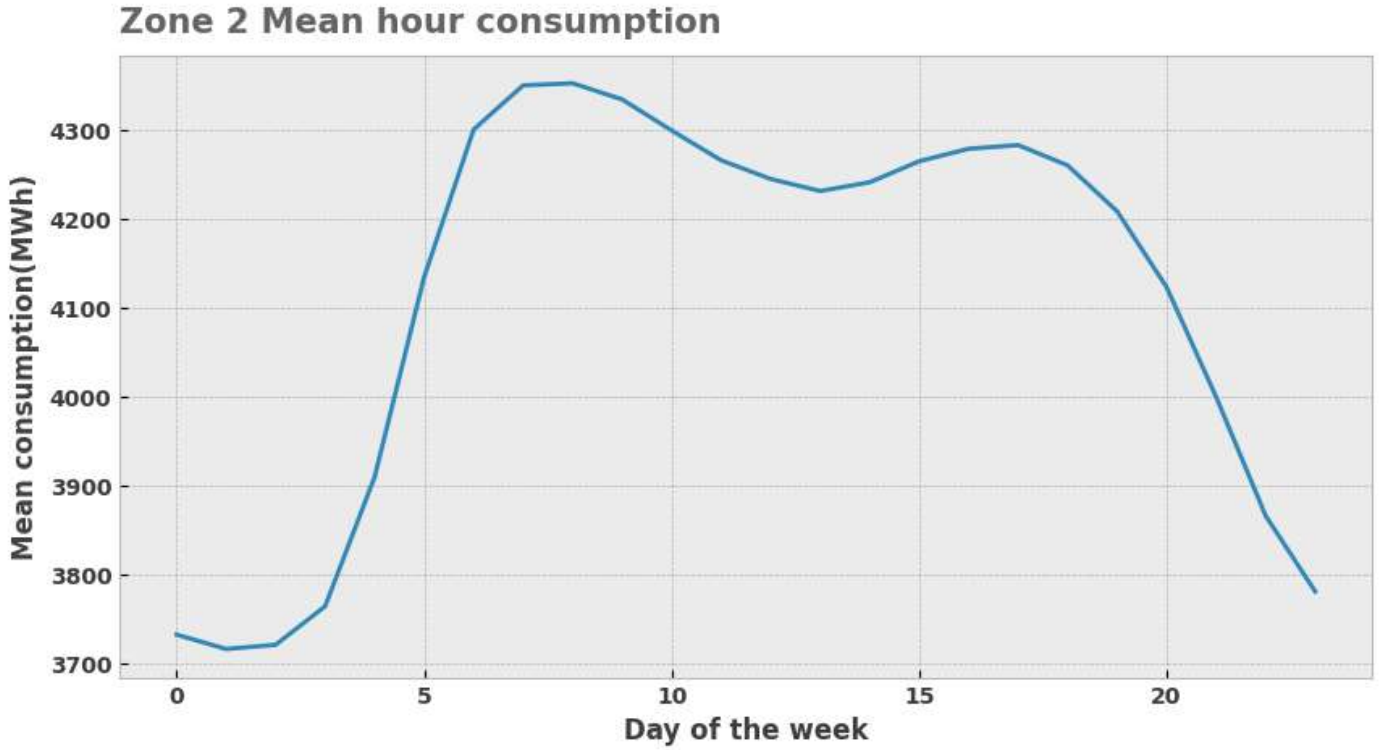
```
#Data prep
mean_per_week = elec.groupby("hour")["load_no1"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_week.index,mean_per_week["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 1 Mean hour consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 1 Mean hour consumption')
```



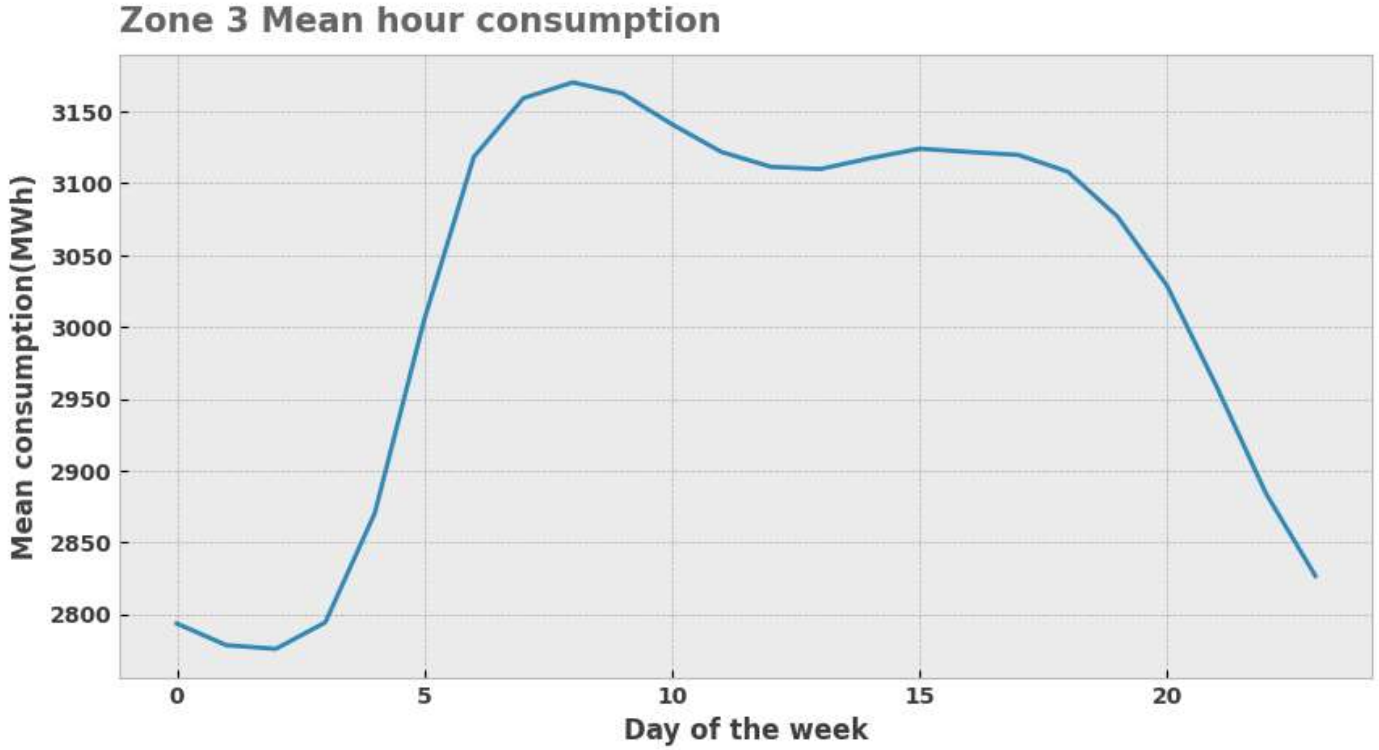
```
#Data prep
mean_per_week = elec.groupby("hour")["load_no2"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_week.index,mean_per_week["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 2 Mean hour consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 2 Mean hour consumption')
```



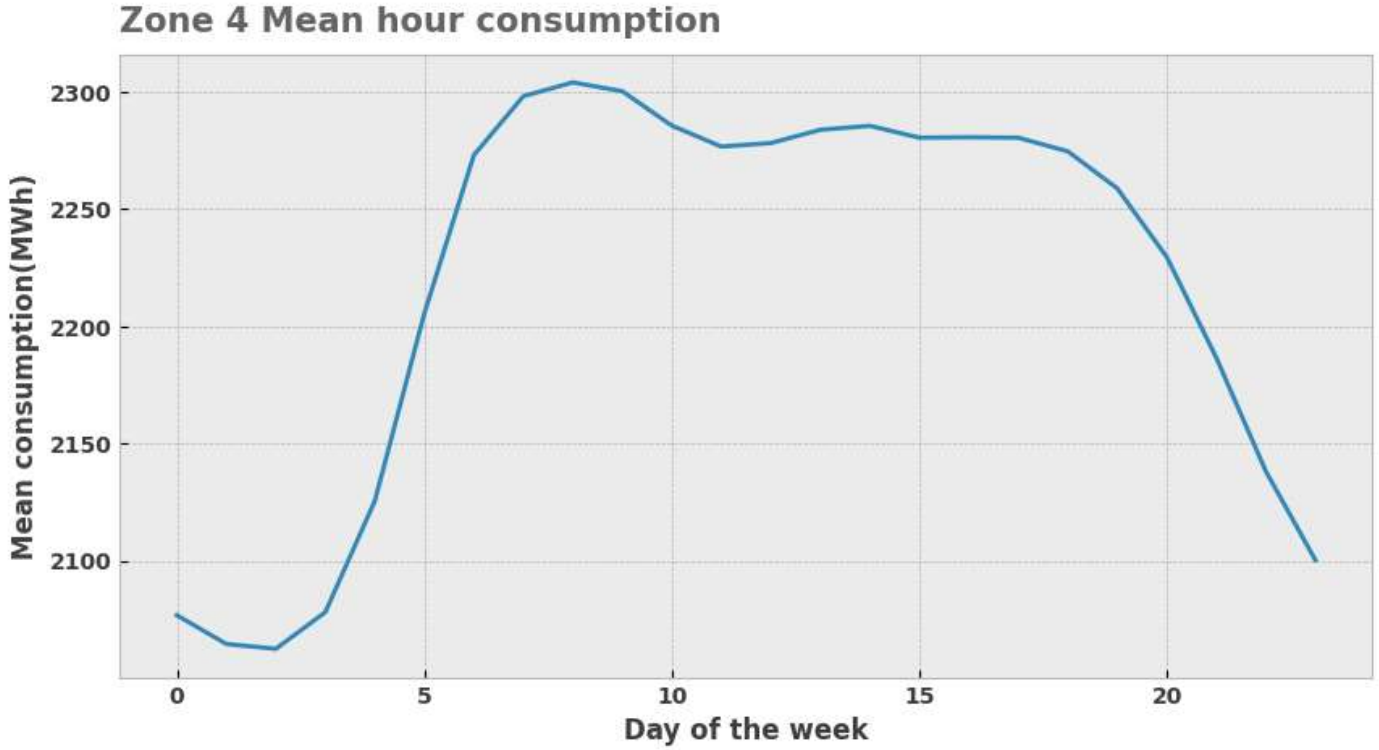
```
#Data prep
mean_per_week = elec.groupby("hour")["load_no3"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_week.index,mean_per_week["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 3 Mean hour consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 3 Mean hour consumption')
```



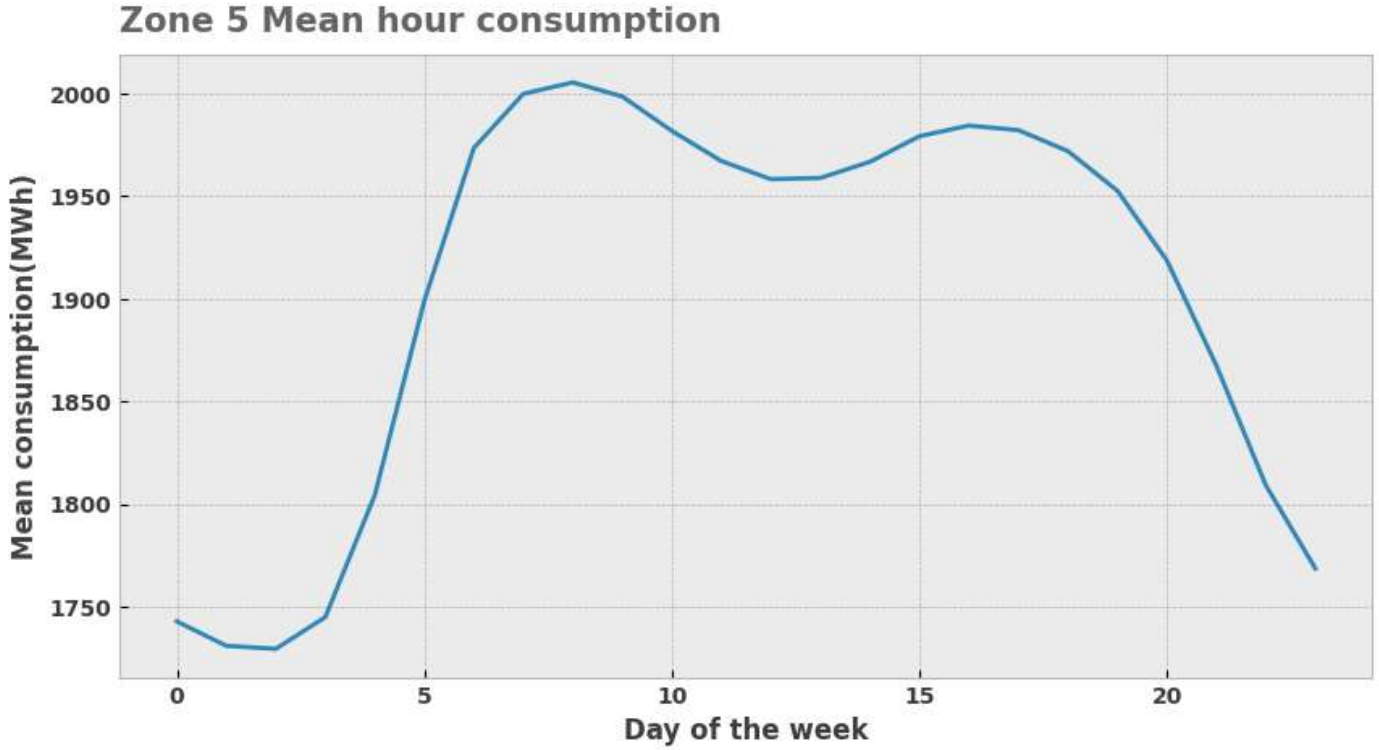
```
#Data prep
mean_per_week = elec.groupby("hour")["load_no4"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_week.index,mean_per_week["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 4 Mean hour consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 4 Mean hour consumption')
```



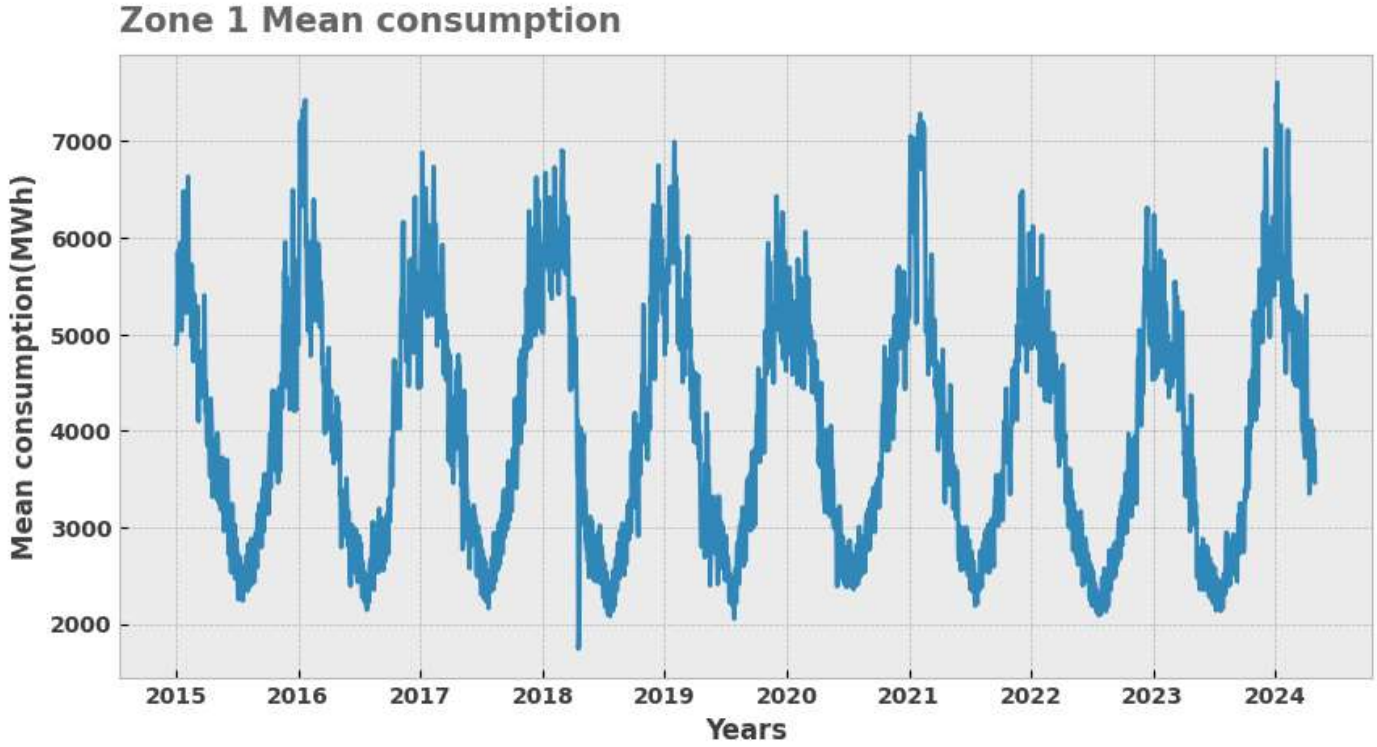
```
#Data prep
mean_per_week = elec.groupby("hour")["load_no5"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_week.index,mean_per_week["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Day of the week",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 5 Mean hour consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 5 Mean hour consumption')
```



```
#Data prep
mean_per_year = elec.groupby("Date")["load_no1"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_year.index,mean_per_year["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Years",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 1 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

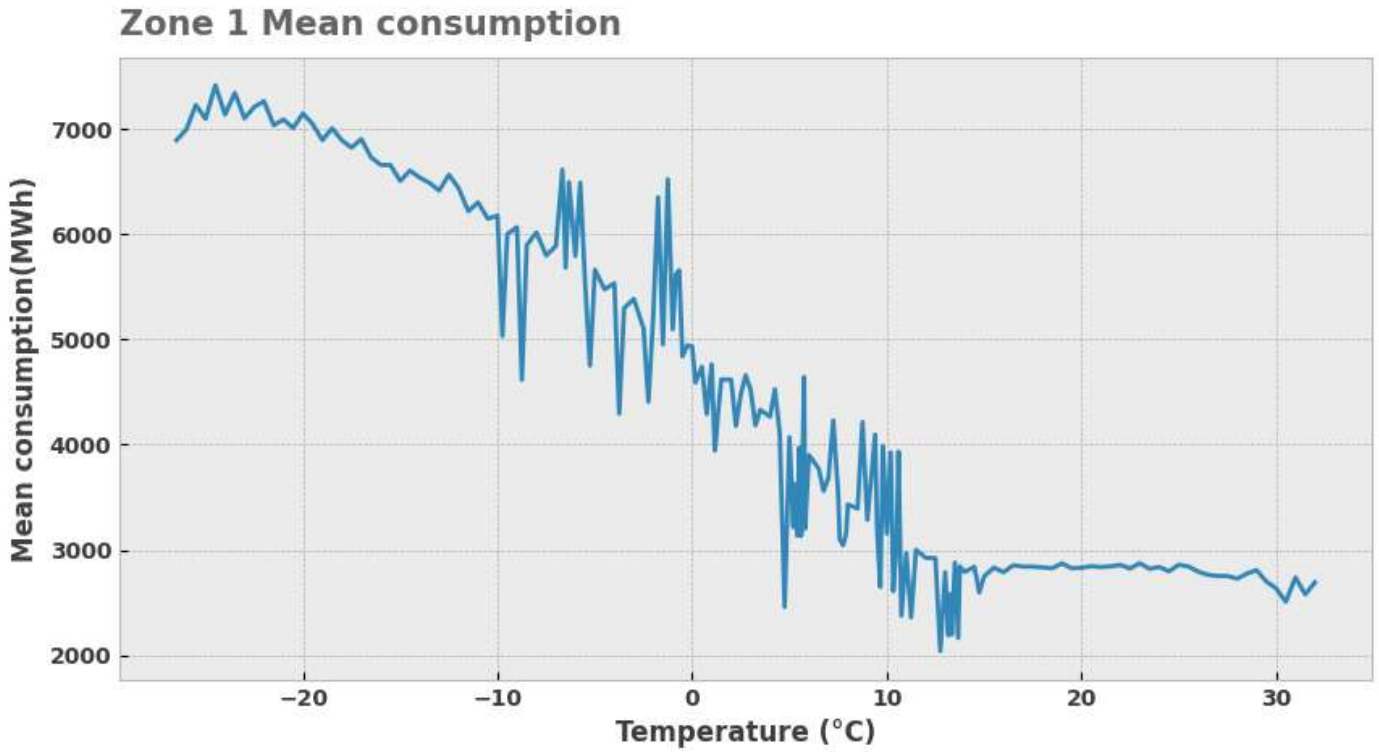
```
Text(0.0, 1.0, 'Zone 1 Mean consumption')
```



Effect of Tempereture

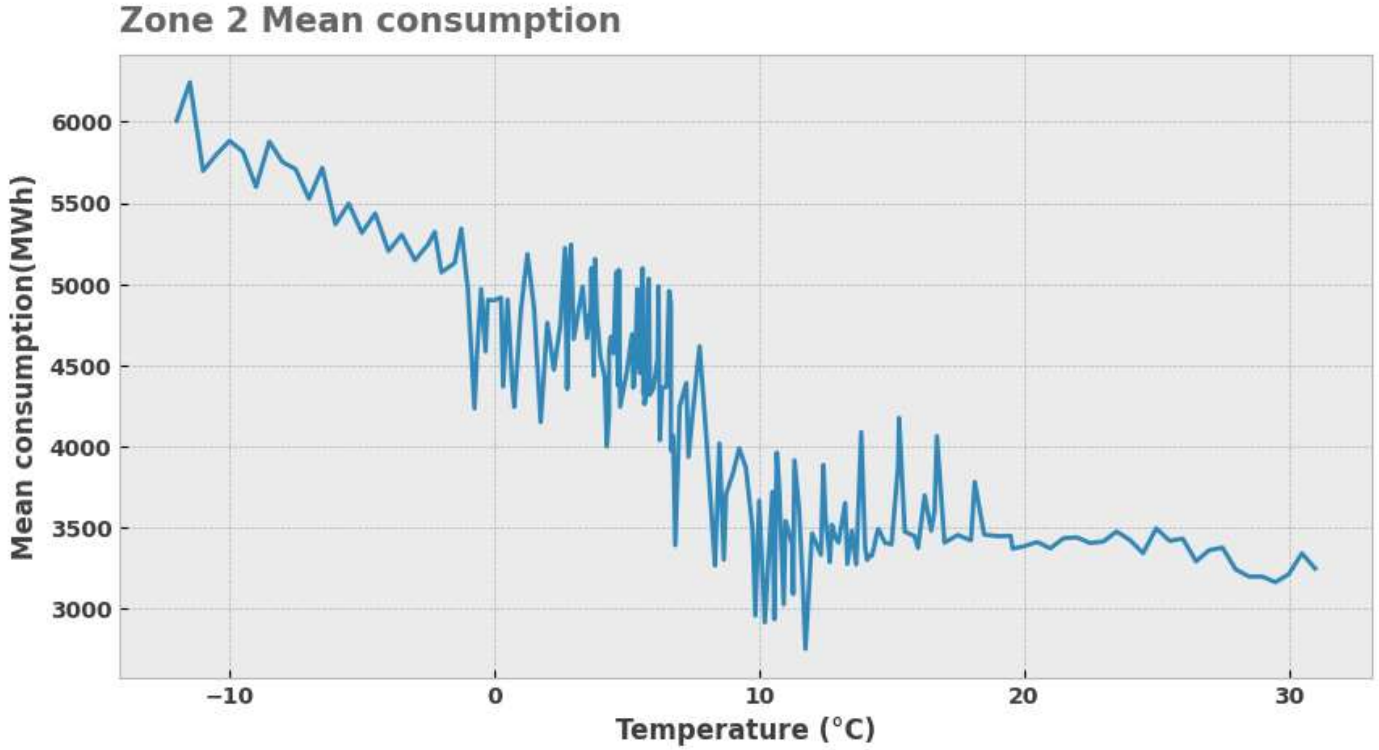
```
#Data prep
mean_per_temp = elec.groupby("temp_no1")["load_no1"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_temp.index,mean_per_temp["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Temperature (°C)",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 1 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 1 Mean consumption')
```



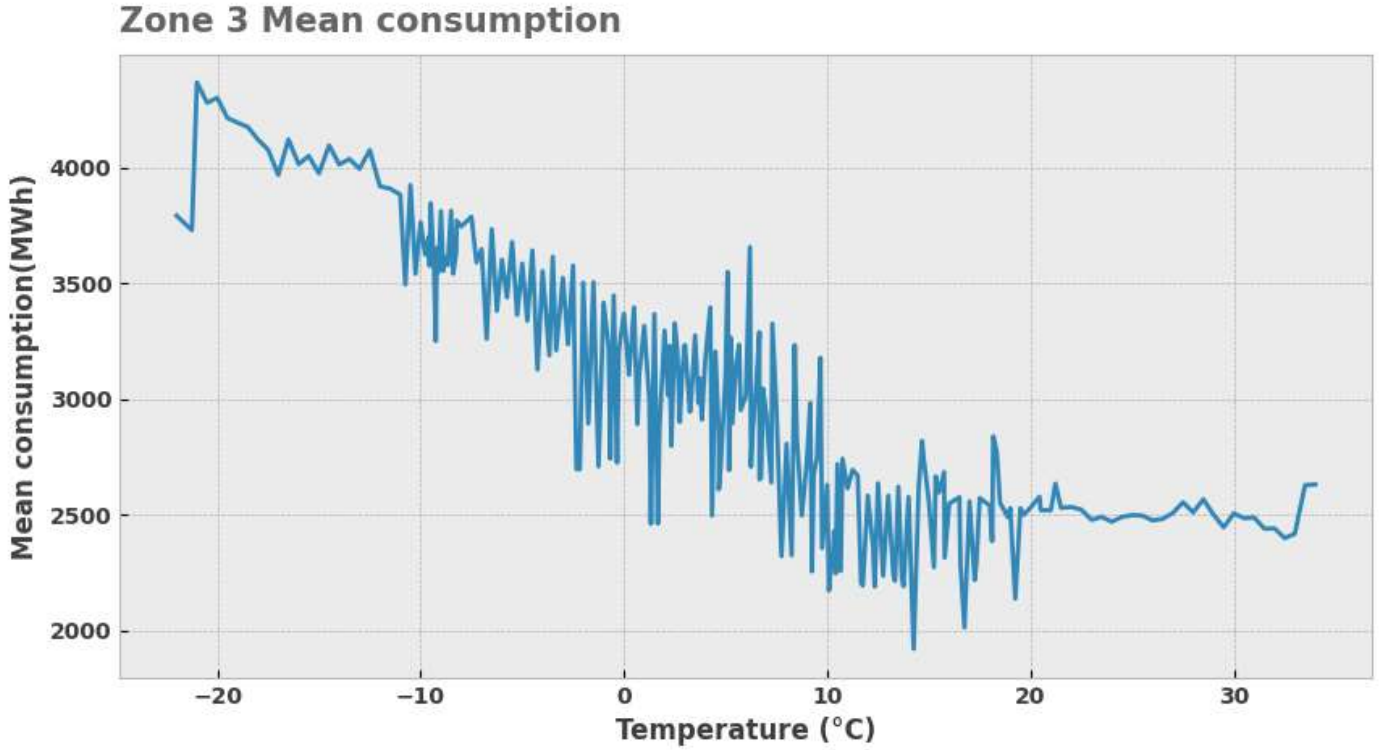
```
#Data prep
mean_per_temp = elec.groupby("temp_no2")["load_no2"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_temp.index,mean_per_temp["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Temperature (°C)",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 2 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 2 Mean consumption')
```



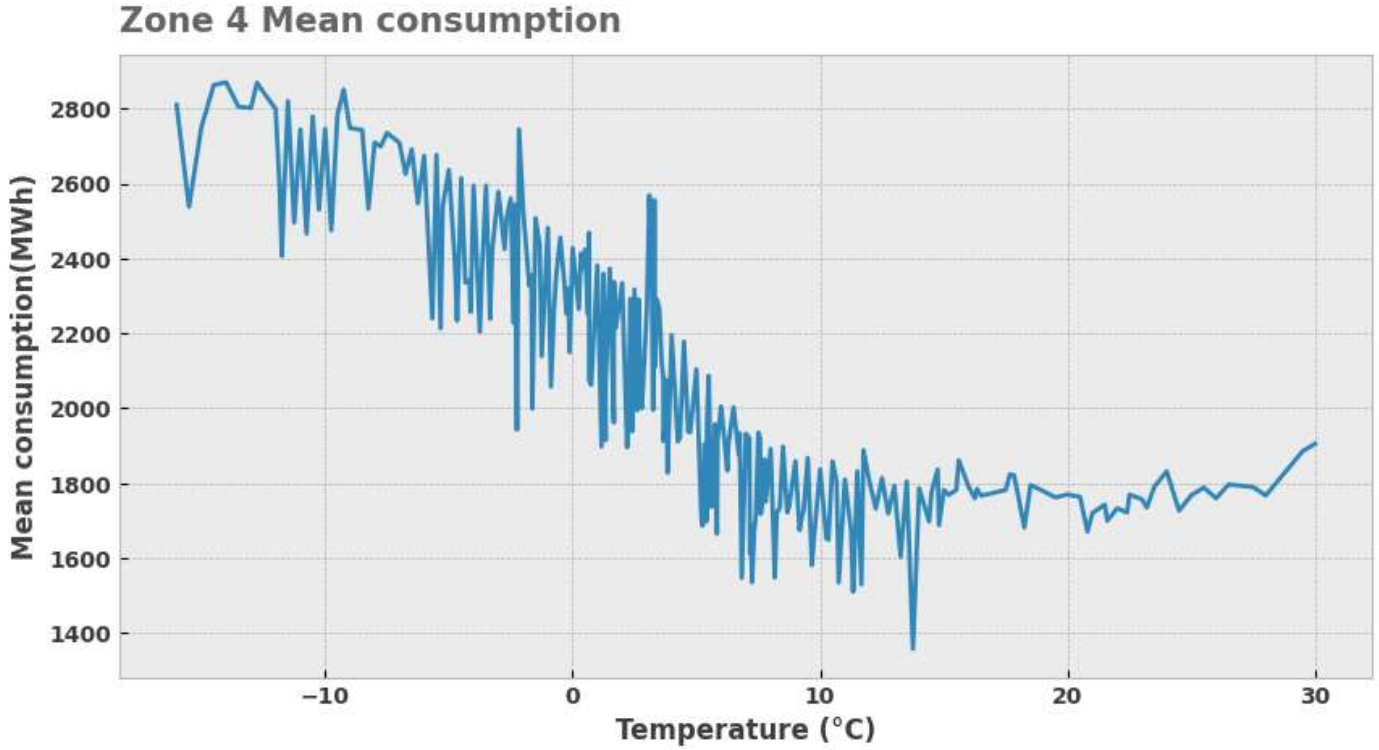
```
#Data prep
mean_per_temp = elec.groupby("temp_no3")["load_no3"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_temp.index,mean_per_temp["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Temperature (°C)",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 3 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 3 Mean consumption')
```



```
#Data prep
mean_per_temp = elec.groupby("temp_no4")["load_no4"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_temp.index,mean_per_temp["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Temperature (°C)",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 4 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

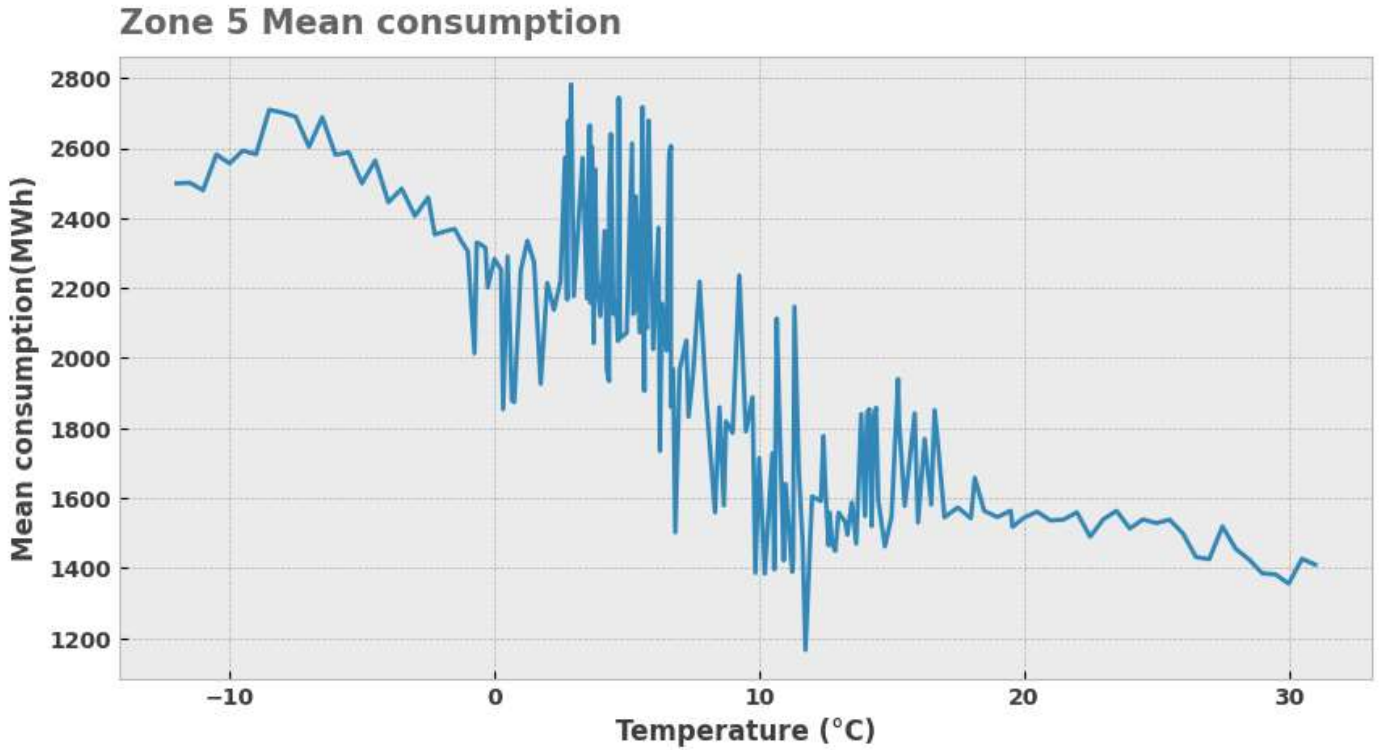
```
Text(0.0, 1.0, 'Zone 4 Mean consumption')
```




```
#Data prep
mean_per_temp = elec.groupby("temp_no5")["load_no5"].agg(["mean"])
#Plot
```

```
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_temp.index,mean_per_temp["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Temperature (°C)",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 5 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

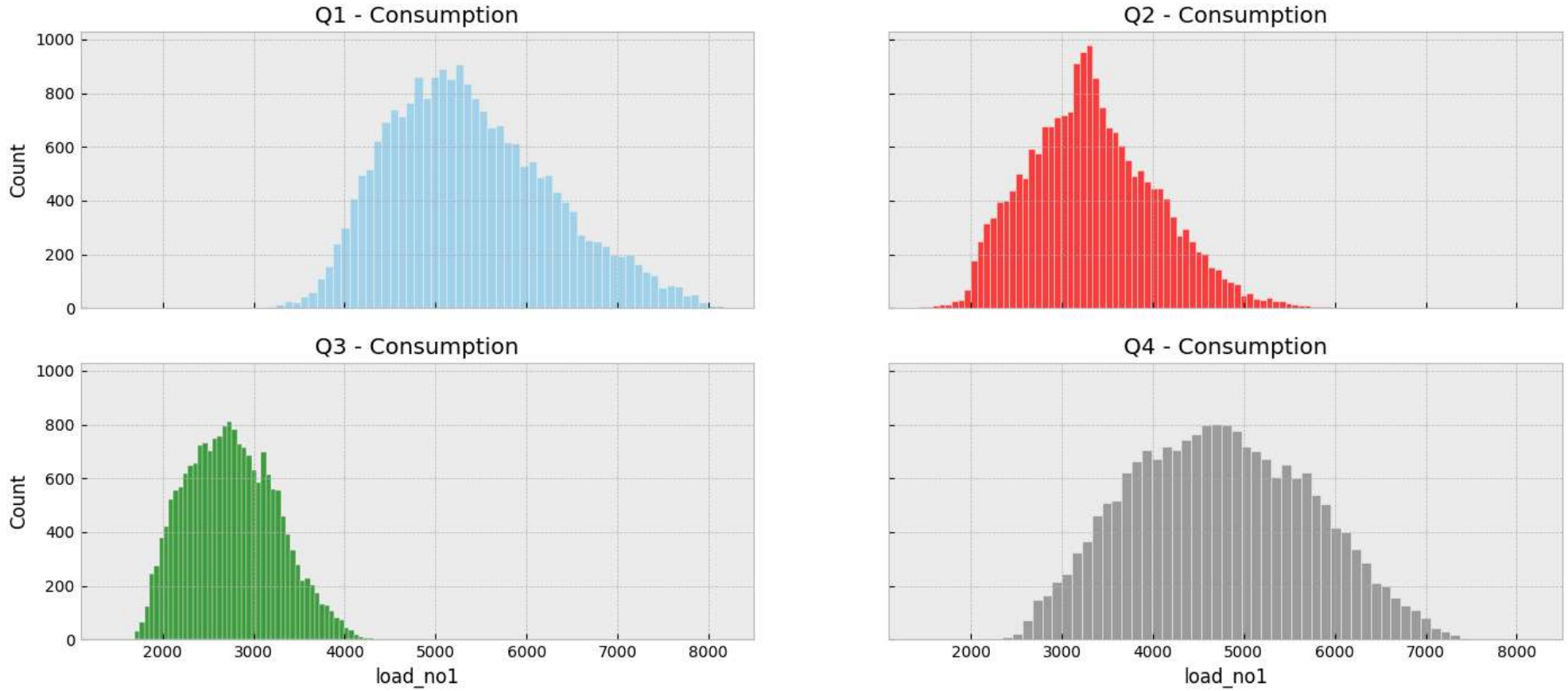
```
Text(0.0, 1.0, 'Zone 5 Mean consumption')
```



Distribution of data in the quaters

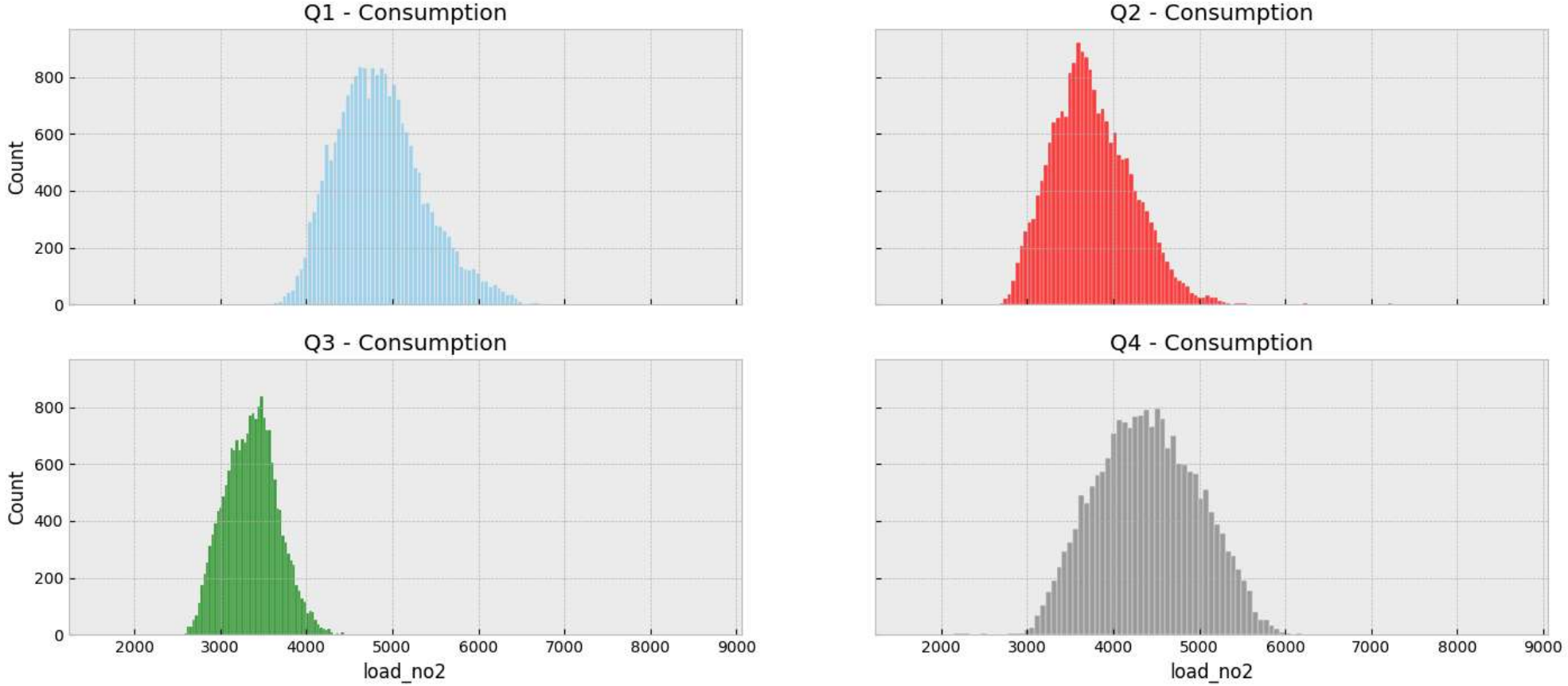
```
#Data prep
Q1 = elec[elec["Q"]==1]
Q2 = elec[elec["Q"]==2]
Q3 = elec[elec["Q"]==3]
Q4 = elec[elec["Q"]==4]

#Plot
fig,axes = plt.subplots(2,2,figsize=(17,7),sharex=True,sharey=True)

sns.histplot(Q1["load_no1"],color="skyblue", ax=axes[0,0]).set_title("Q1 - Consumption")
sns.histplot(Q2["load_no1"],color="red", ax=axes[0,1]).set_title("Q2 - Consumption")
sns.histplot(Q3["load_no1"],color="green", ax=axes[1,0]).set_title("Q3 - Consumption")
sns.histplot(Q4["load_no1"],color="gray", ax=axes[1,1]).set_title("Q4 - Consumption")
```

```
Text(0.5, 1.0, 'Q4 - Consumption')
```



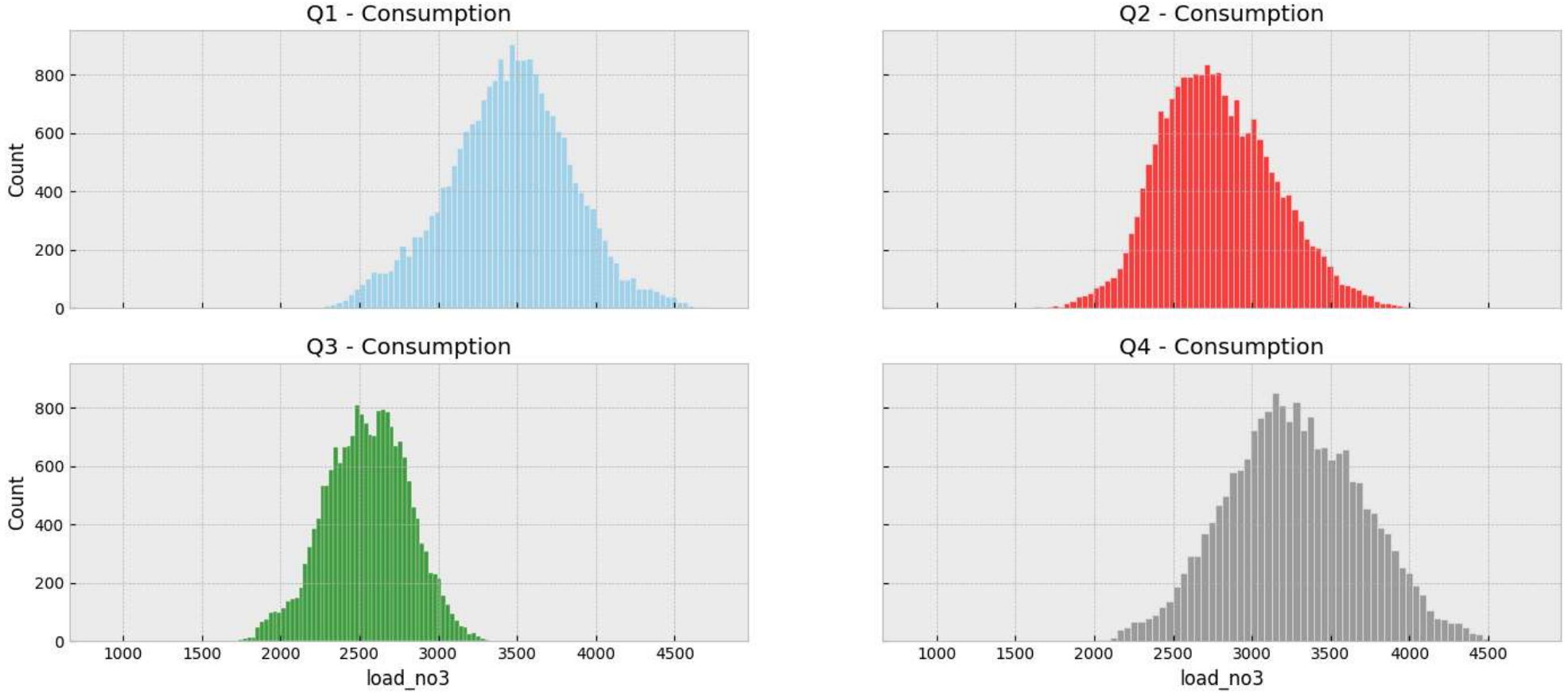
```
#Plot
fig,axes = plt.subplots(2,2,figsize=(17,7),sharex=True,sharey=True)

sns.histplot(Q1["load_no2"],color="skyblue", ax=axes[0,0]).set_title("Q1 - Consumption")
sns.histplot(Q2["load_no2"],color="red", ax=axes[0,1]).set_title("Q2 - Consumption")
sns.histplot(Q3["load_no2"],color="green", ax=axes[1,0]).set_title("Q3 - Consumption")
sns.histplot(Q4["load_no2"],color="gray", ax=axes[1,1]).set_title("Q4 - Consumption")
```

```
Text(0.5, 1.0, 'Q4 - Consumption')
```



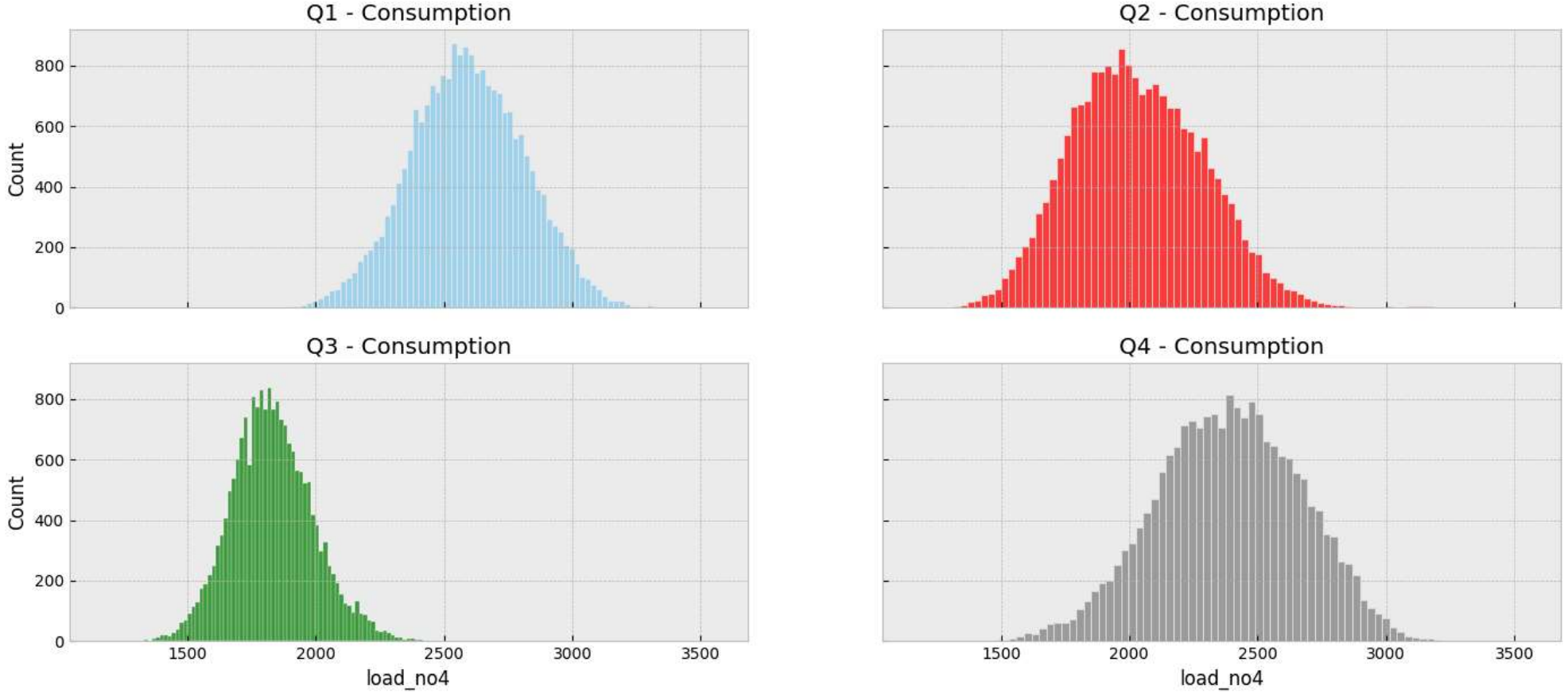
```
#Plot
fig,axes = plt.subplots(2,2,figsize=(17,7),sharex=True,sharey=True)

sns.histplot(Q1["load_no3"],color="skyblue", ax=axes[0,0]).set_title("Q1 - Consumption")
sns.histplot(Q2["load_no3"],color="red", ax=axes[0,1]).set_title("Q2 - Consumption")
sns.histplot(Q3["load_no3"],color="green", ax=axes[1,0]).set_title("Q3 - Consumption")
sns.histplot(Q4["load_no3"],color="gray", ax=axes[1,1]).set_title("Q4 - Consumption")
```

```
Text(0.5, 1.0, 'Q4 - Consumption')
```



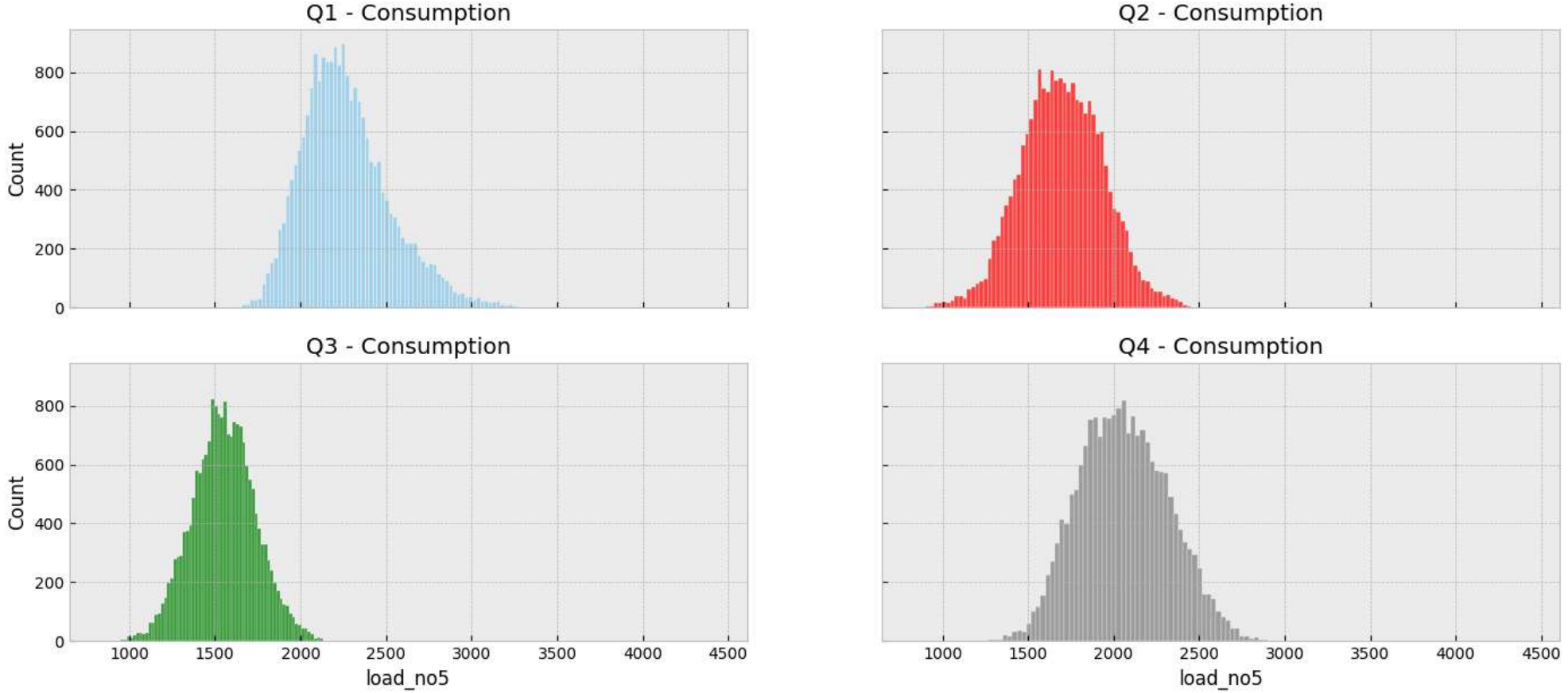
```
#Plot
fig,axes = plt.subplots(2,2,figsize=(17,7),sharex=True,sharey=True)

sns.histplot(Q1["load_no4"],color="skyblue", ax=axes[0,0]).set_title("Q1 - Consumption")
sns.histplot(Q2["load_no4"],color="red", ax=axes[0,1]).set_title("Q2 - Consumption")
sns.histplot(Q3["load_no4"],color="green", ax=axes[1,0]).set_title("Q3 - Consumption")
sns.histplot(Q4["load_no4"],color="gray", ax=axes[1,1]).set_title("Q4 - Consumption")
```

```
Text(0.5, 1.0, 'Q4 - Consumption')
```



```
#Plot
fig,axes = plt.subplots(2,2,figsize=(17,7),sharex=True,sharey=True)

sns.histplot(Q1["load_no5"],color="skyblue", ax=axes[0,0]).set_title("Q1 - Consumption")
sns.histplot(Q2["load_no5"],color="red", ax=axes[0,1]).set_title("Q2 - Consumption")
sns.histplot(Q3["load_no5"],color="green", ax=axes[1,0]).set_title("Q3 - Consumption")
sns.histplot(Q4["load_no5"],color="gray", ax=axes[1,1]).set_title("Q4 - Consumption")
```

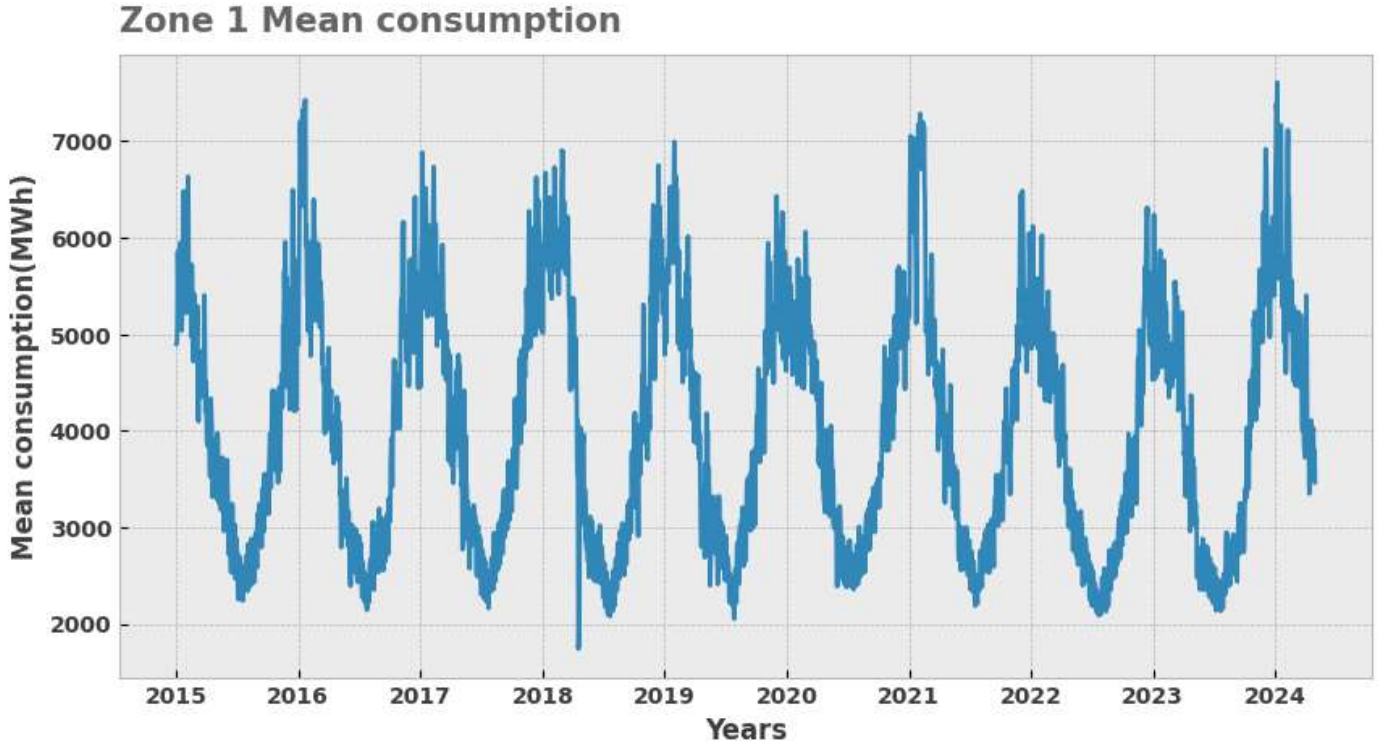
```
Text(0.5, 1.0, 'Q4 - Consumption')
```



Trend of Electricity Consumption

```
#Data prep
mean_per_year = elec.groupby("Date")["load_no1"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_year.index,mean_per_year["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Years",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 1 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

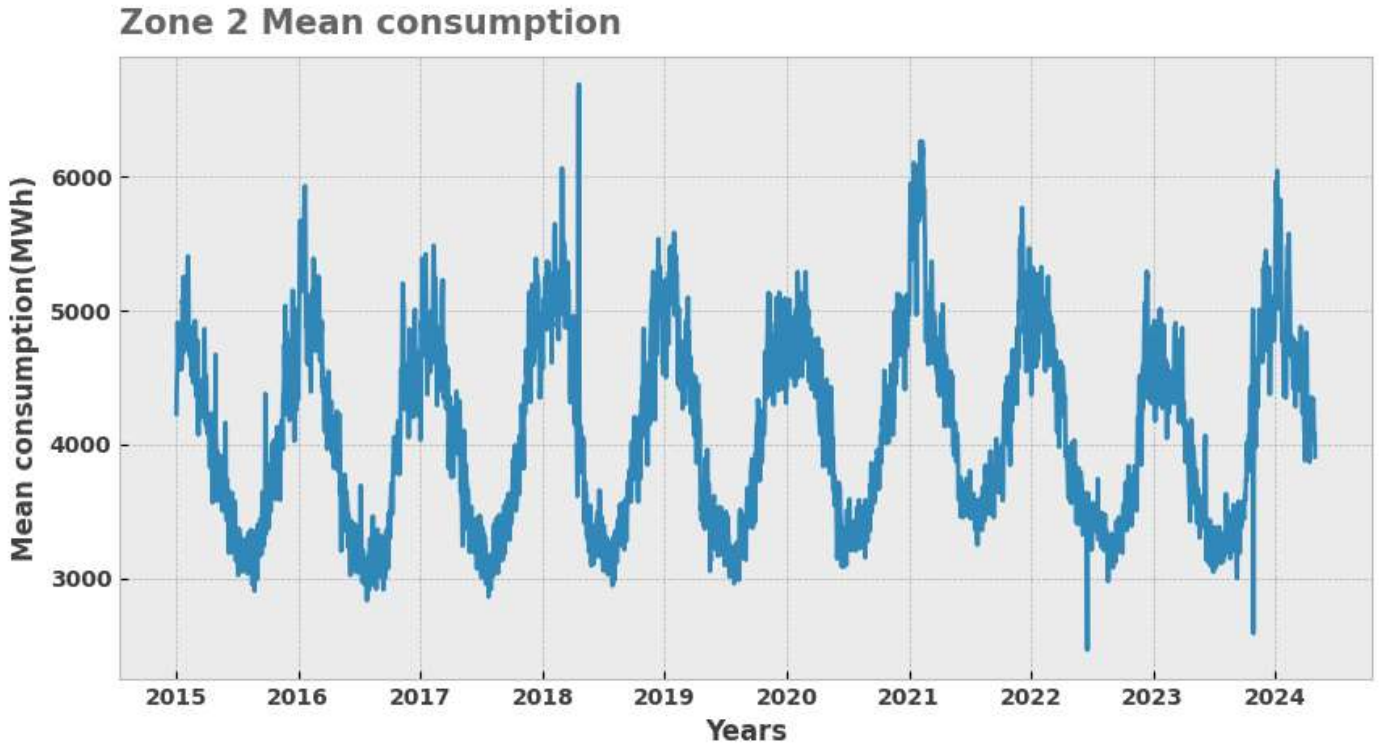
```
Text(0.0, 1.0, 'Zone 1 Mean consumption')
```




```
#Data prep
mean_per_year = elec.groupby("Date")["load_no2"].agg(["mean"])
#Plot
```

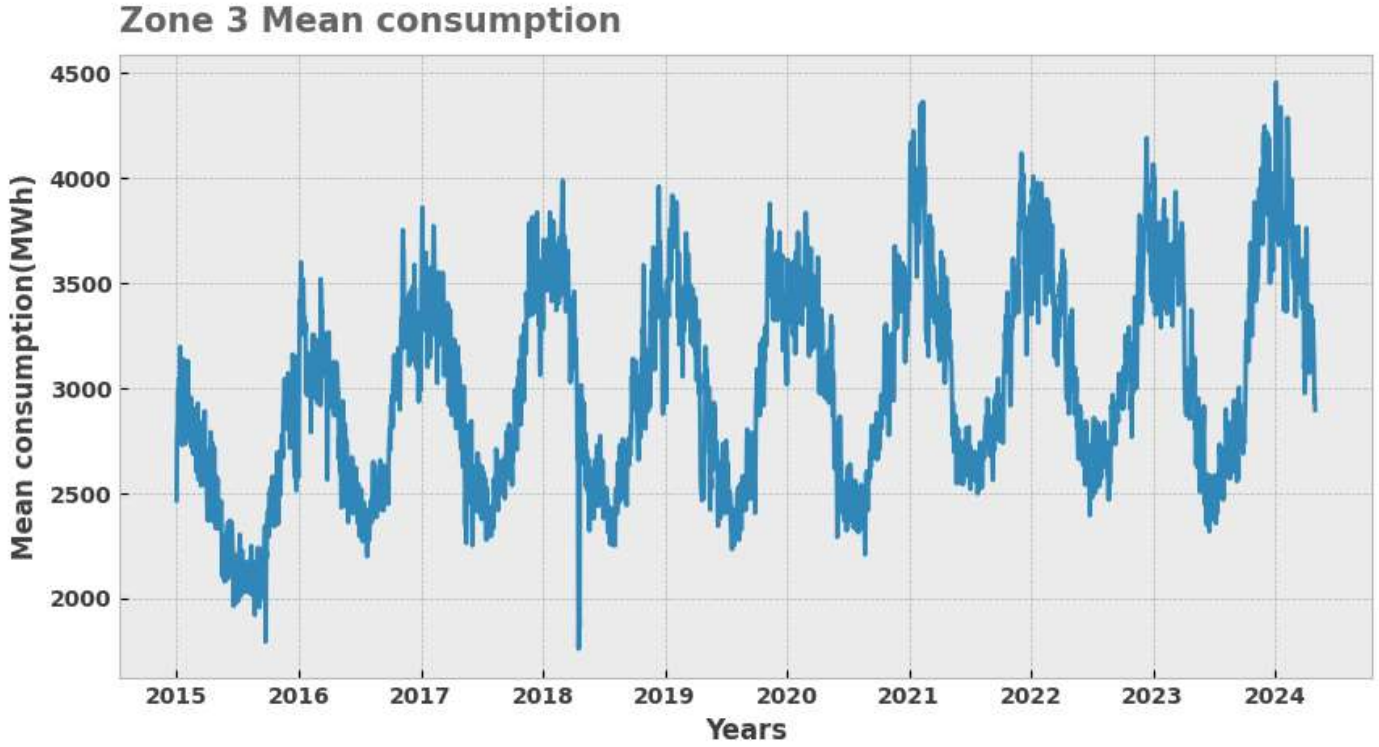
```
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_year.index,mean_per_year["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Years",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 2 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 2 Mean consumption')
```



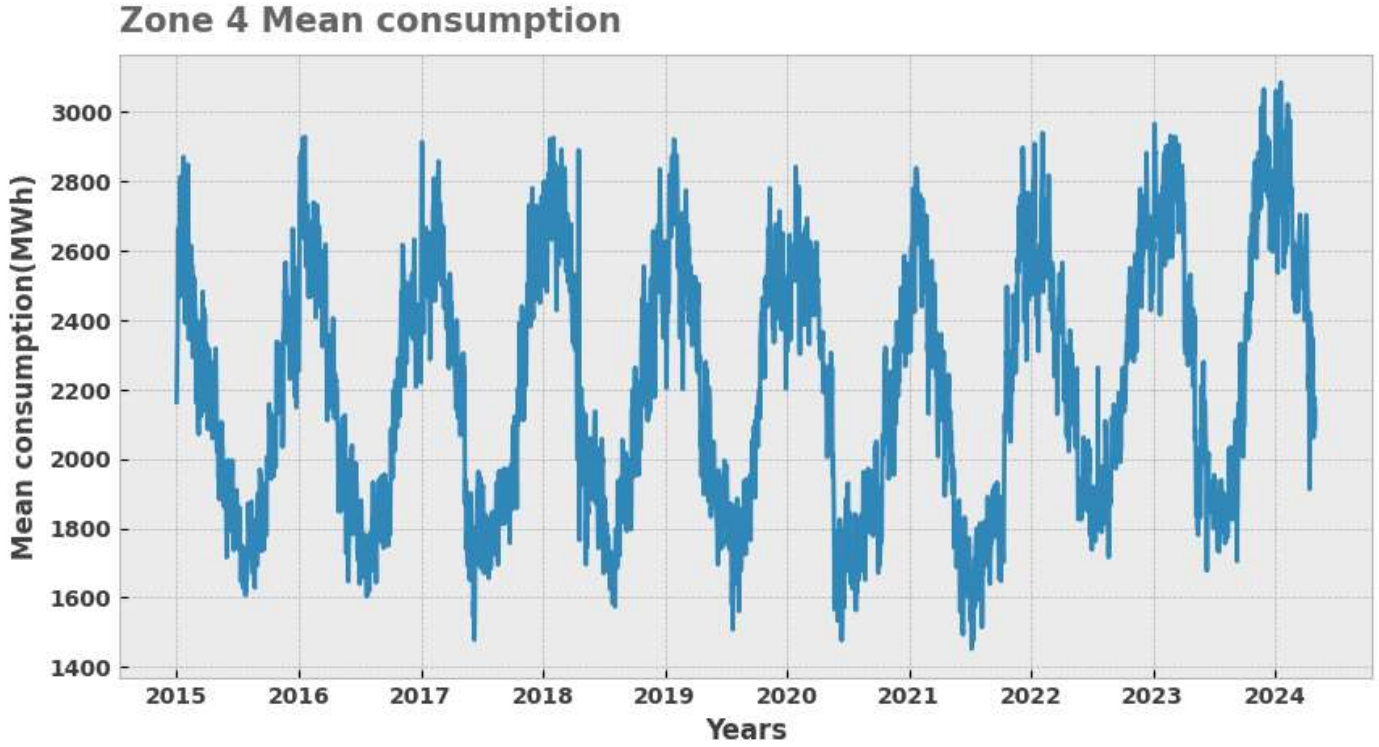
```
#Data prep
mean_per_year = elec.groupby("Date")["load_no3"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_year.index,mean_per_year["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Years",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 3 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 3 Mean consumption')
```



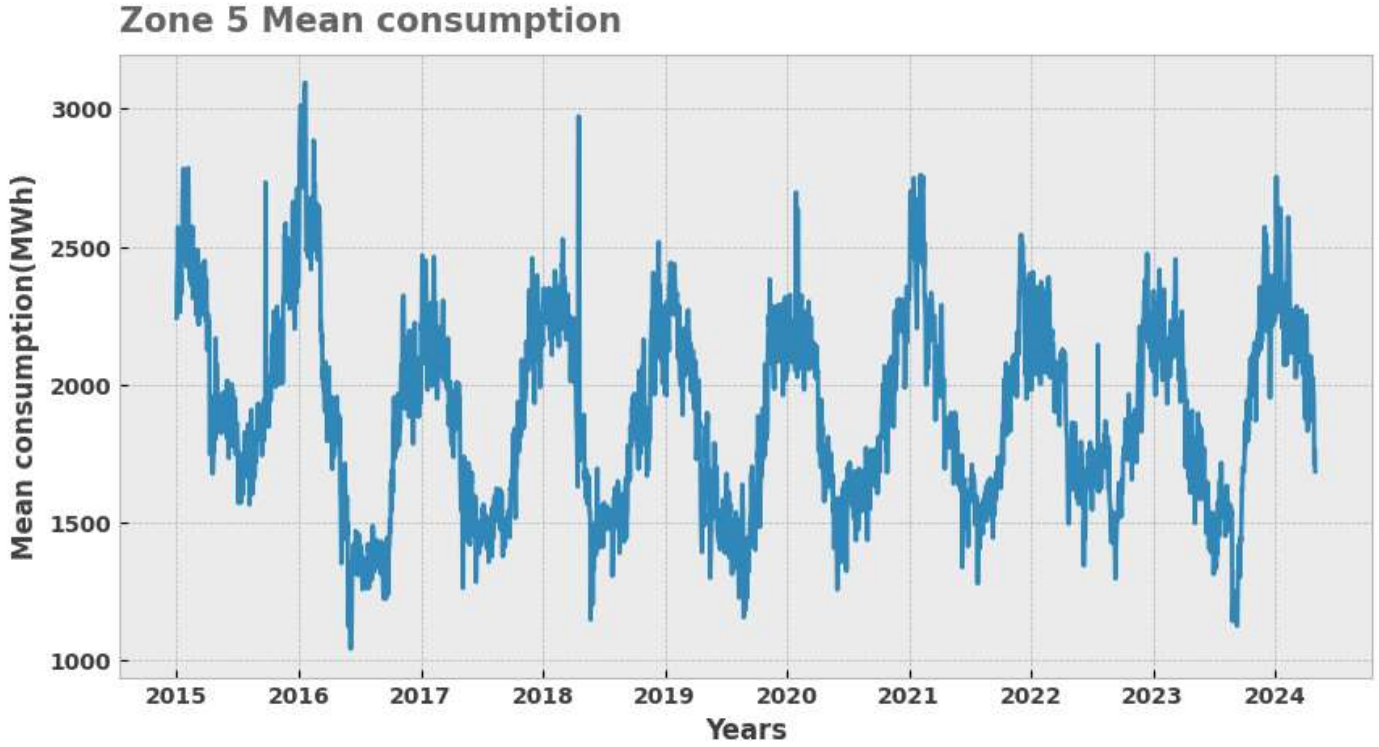
```
#Data prep
mean_per_year = elec.groupby("Date")["load_no4"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_year.index,mean_per_year["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Years",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 4 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 4 Mean consumption')
```



```
#Data prep
mean_per_year = elec.groupby("Date")["load_no5"].agg(["mean"])
#Plot
fig, ax = plt.subplots(figsize=(10,5))


plt.plot(mean_per_year.index,mean_per_year["mean"])

plt.xticks(alpha=0.75, weight="bold")
plt.yticks(alpha=0.75, weight="bold")

plt.xlabel("Years",alpha=0.75, weight="bold")
plt.ylabel("Mean consumption(MWh)",alpha=0.75, weight="bold")

plt.title("Zone 5 Mean consumption", alpha=0.60, weight="bold", fontsize=15, loc="left", pad=10)
```

```
Text(0.0, 1.0, 'Zone 5 Mean consumption')
```



Adding columns

```
# List of Norway public holidays from 2014 to 2024
norway_holidays = [
    '2024-01-01','2023-01-01','2022-01-01','2021-01-01','2020-01-01',
    '2019-01-01','2018-01-01','2017-01-01','2016-01-01','2015-01-01','2014-01-01', # New Year's Day
    '2024-05-01','2023-05-01','2022-05-01','2021-05-01','2020-05-01',
    '2019-05-01','2018-05-01','2017-05-01','2016-05-01','2015-05-01','2014-05-01', # Labour day
    '2024-12-25','2023-12-25','2022-12-25','2021-12-25','2020-12-25',
    '2019-12-25','2018-12-25','2017-12-25','2016-12-25','2015-12-25','2014-12-25', # christmas day
    '2024-12-24','2023-12-24','2022-12-24','2021-12-24','2020-12-24',
    '2019-12-24','2018-12-24','2017-12-24','2016-12-24','2015-12-24','2014-12-24', # christmas eve
    '2024-12-26','2023-12-26','2022-12-26','2021-12-26','2020-12-26',
    '2019-12-26','2018-12-26','2017-12-26','2016-12-26','2015-12-26','2014-12-26', # boxing day
    '2024-03-28','2023-04-06','2022-04-14','2021-04-01','2020-04-09',
    '2019-04-18','2018-03-29','2017-04-13','2016-03-24','2015-04-02','2014-04-17', # Maundy Thursday
    '2024-03-29','2023-04-07','2022-04-15','2021-04-02','2020-04-10',
    '2019-04-19','2018-03-30','2017-04-14','2016-03-25','2015-04-03','2014-04-18', # Good Friday
    '2024-03-31','2023-04-09','2022-04-17','2021-04-04','2020-04-12',
    '2019-04-21','2018-04-01','2017-04-16','2016-03-27','2015-04-05','2014-04-20', # Easter Monday
    '2024-05-09','2023-05-18','2022-05-26','2021-05-13','2020-05-21',
    '2019-05-30','2018-05-10','2017-05-25','2016-05-05','2015-05-14','2014-05-29', # Ascension day
    '2024-05-17','2023-05-17','2022-05-17','2021-05-17','2020-05-17',
    '2019-05-17','2018-05-17','2017-05-17','2016-05-17','2015-05-17','2014-05-17', # Constituition day
    '2024-05-20','2023-05-29','2022-06-06','2021-05-24','2020-06-01',
    '2019-06-10','2018-05-21','2017-06-05','2016-05-16','2015-05-25','2014-06-09' # Whit Monday
]


# Check if each datetime is a holiday
elec['is_holiday'] = elec['Datetime'].dt.date.astype(str).isin(norway_holidays)


# Display the DataFrame
print(elec[['Datetime', 'Date', 'Time', 'is_holiday']])
# Display the updated DataFrame
print(elec)
```

```
                 Datetime        Date      Time  is_holiday
0     2015-01-01 00:00:00  2015-01-01  00:00:00        True
1     2015-01-01 01:00:00  2015-01-01  01:00:00        True
2     2015-01-01 02:00:00  2015-01-01  02:00:00        True
3     2015-01-01 03:00:00  2015-01-01  03:00:00        True
4     2015-01-01 04:00:00  2015-01-01  04:00:00        True
...                   ...         ...       ...         ...
81764 2024-04-29 20:00:00  2024-04-29  20:00:00       False
81765 2024-04-29 21:00:00  2024-04-29  21:00:00       False
```

```
81766 2024-04-29 22:00:00  2024-04-29  22:00:00       False
81767 2024-04-29 23:00:00  2024-04-29  23:00:00       False
81768 2024-04-30 00:00:00  2024-04-30  00:00:00       False

[81769 rows x 4 columns]
                 Datetime  load_no1  wind_no1  temp_no1  load_no2  wind_no2  \
0     2015-01-01 00:00:00    4659.0       0.0      -2.5    4139.0     156.0
1     2015-01-01 01:00:00    4552.0       0.0      -2.0    4039.0     159.0
2     2015-01-01 02:00:00    4469.0       0.0      -1.5    3956.0     149.0
3     2015-01-01 03:00:00    4442.0       0.0       0.0    3900.0     144.0
4     2015-01-01 04:00:00    4488.0       0.0       0.0    3915.0     149.0
...                   ...       ...       ...       ...       ...       ...
81764 2024-04-29 20:00:00    3715.0     206.0       7.0    4109.0     816.0
81765 2024-04-29 21:00:00    3464.0     201.0       7.0    3905.0     878.0
81766 2024-04-29 22:00:00    3464.0     201.0       7.0    3905.0     878.0
81767 2024-04-29 23:00:00    3464.0     201.0       7.0    3905.0     878.0
81768 2024-04-30 00:00:00    3464.0     201.0       7.0    3905.0     878.0

       temp_no2  load_no3  wind_no3  temp_no3  ...  month  Q  Week_Number  \
0           6.5    2370.0     259.0       3.0  ...      1  1           00
1           7.0    2307.0     234.0       3.0  ...      1  1           00
2           7.0    2273.0     200.0       3.0  ...      1  1           00
3           7.0    2286.0     192.0       2.0  ...      1  1           00
4           8.0    2333.0     192.0       2.0  ...      1  1           00
...         ...       ...       ...       ...  ...    ... ..          ...
81764      10.0    2942.0     652.0       6.0  ...      4  2           17
81765      10.0    2894.0     650.0       5.5  ...      4  2           17
81766      10.0    2894.0     650.0       5.5  ...      4  2           17
81767      10.0    2894.0     650.0       5.5  ...      4  2           17
81768      10.0    2894.0     650.0       5.5  ...      4  2           17

       Dayofyear  Dayofmonth  Day hour is_weekend  is_weekday  is_holiday
0              1           1    3    0      False        True        True
1              1           1    3    1      False        True        True
2              1           1    3    2      False        True        True
3              1           1    3    3      False        True        True
4              1           1    3    4      False        True        True
...          ...         ...  ...  ...        ...         ...         ...
81764        120          29    0   20      False        True       False
81765        120          29    0   21      False        True       False
81766        120          29    0   22      False        True       False
81767        120          29    0   23      False        True       False
81768        121          30    1    0      False        True       False

[81769 rows x 29 columns]
```

```
def feature_worktime(row):
    if row["hour"] > 7 & row["hour"] <= 17:
        return "Worktime"
    return "NonWorkTime"
def feature_peak(row):
    if row["hour"] > 7 & row["hour"] <= 22:
        return "Peak"
    return "NonPeak"
```

```
elec["Work"] = elec.apply(lambda row: feature_worktime(row), axis=1)
elec["Peak"] = elec.apply(lambda row: feature_peak(row), axis=1)
```

```
# Display the DataFrame
print(elec[['Date', 'Time', 'is_holiday', 'Work']])
```

```
            Date      Time  is_holiday         Work
0      2015-01-01  00:00:00        True  NonWorkTime
1      2015-01-01  01:00:00        True  NonWorkTime
2      2015-01-01  02:00:00        True  NonWorkTime
3      2015-01-01  03:00:00        True  NonWorkTime
4      2015-01-01  04:00:00        True  NonWorkTime
...           ...       ...         ...          ...
81764  2024-04-29  20:00:00       False     Worktime
81765  2024-04-29  21:00:00       False     Worktime
81766  2024-04-29  22:00:00       False     Worktime
81767  2024-04-29  23:00:00       False     Worktime
81768  2024-04-30  00:00:00       False  NonWorkTime

[81769 rows x 4 columns]
```

```
elec.index = elec['Datetime']
elec = elec.drop(["Datetime"],axis=1)
```

```
elec.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 81769 entries, 2015-01-01 00:00:00 to 2024-04-30 00:00:00
Data columns (total 30 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
```

```
 0   load_no1     81769 non-null  float64
 1   wind_no1     81769 non-null  float64
 2   temp_no1     81769 non-null  float64
 3   load_no2     81769 non-null  float64
 4   wind_no2     81769 non-null  float64
 5   temp_no2     81769 non-null  float64
 6   load_no3     81769 non-null  float64
 7   wind_no3     81769 non-null  float64
 8   temp_no3     81769 non-null  float64
 9   load_no4     81769 non-null  float64
 10  wind_no4     81769 non-null  float64
 11  temp_no4     81769 non-null  float64
 12  load_no5     81769 non-null  float64
 13  wind_no5     81769 non-null  float64
 14  temp_no5     81769 non-null  float64
 15  Date         81769 non-null  object
 16  Time         81769 non-null  object
 17  year         81769 non-null  int32
 18  month        81769 non-null  int32
 19  Q            81769 non-null  int32
 20  Week_Number  81769 non-null  object
 21  Dayofyear    81769 non-null  int32
 22  Dayofmonth   81769 non-null  int32
 23  Day          81769 non-null  int32
 24  hour         81769 non-null  int32
 25  is_weekend   81769 non-null  bool
 26  is_weekday   81769 non-null  bool
 27  is_holiday   81769 non-null  bool
 28  Work         81769 non-null  object
 29  Peak         81769 non-null  object
dtypes: bool(3), float64(15), int32(7), object(5)
memory usage: 15.5+ MB
```

```
elec_data = elec.drop(["Date", "Time", "Week_Number"],axis=1)
elec_data.head()
```

Datetime	load_no1	wind_no1	temp_no1	load_no2	wind_no2	temp_no2	load_no3	wind_no3	temp_no3	load_no4	...	Q	Dayofyear	Dayofmonth	Day	hour	is_weekend
2015-01-01 00:00:00	4659.0	0.0	-2.5	4139.0	156.0	6.5	2370.0	259.0	3.0	2091.0	...	1	1	1	3	0	False
2015-01-01 01:00:00	4552.0	0.0	-2.0	4039.0	159.0	7.0	2307.0	234.0	3.0	2078.0	...	1	1	1	3	1	False
2015-01-01 02:00:00	4469.0	0.0	-1.5	3956.0	149.0	7.0	2273.0	200.0	3.0	2037.0	...	1	1	1	3	2	False
2015-01-01 03:00:00	4442.0	0.0	0.0	3900.0	144.0	7.0	2286.0	192.0	2.0	2013.0	...	1	1	1	3	3	False
2015-01-01 04:00:00	4488.0	0.0	0.0	3915.0	149.0	8.0	2333.0	192.0	2.0	2037.0	...	1	1	1	3	4	False

5 rows × 27 columns

Splitting Data for Training and Testing An important part of model building is splitting our data for training and testing, which ensures that you build a model that can generalize outside of the training data and that the performance and outputs are statistically meaningful.

We will split our data such that everything before November 2020 will serve as training data, with everything after 2020 becoming the testing data:

Get Dummies

```
# Get Dummies for categorical columns
categorical_columns = elec_data.select_dtypes(include='object')

# # Create dummy variables
cat_df = pd.get_dummies(categorical_columns, columns=categorical_columns.columns)

# # Assign back to the original DataFrame
elec_data = pd.concat([elec_data, cat_df], axis=1)

# # Describe the modified DataFrame
print(elec_data.describe().T)
print(elec_data.info())
```

```
load_no2    8700.0
wind_no2    1448.0
temp_no2      31.0
load_no3    4766.0
wind_no3    1997.0
temp_no3      34.0
load_no4    3562.0
wind_no4    1004.0
temp_no4      30.0
```

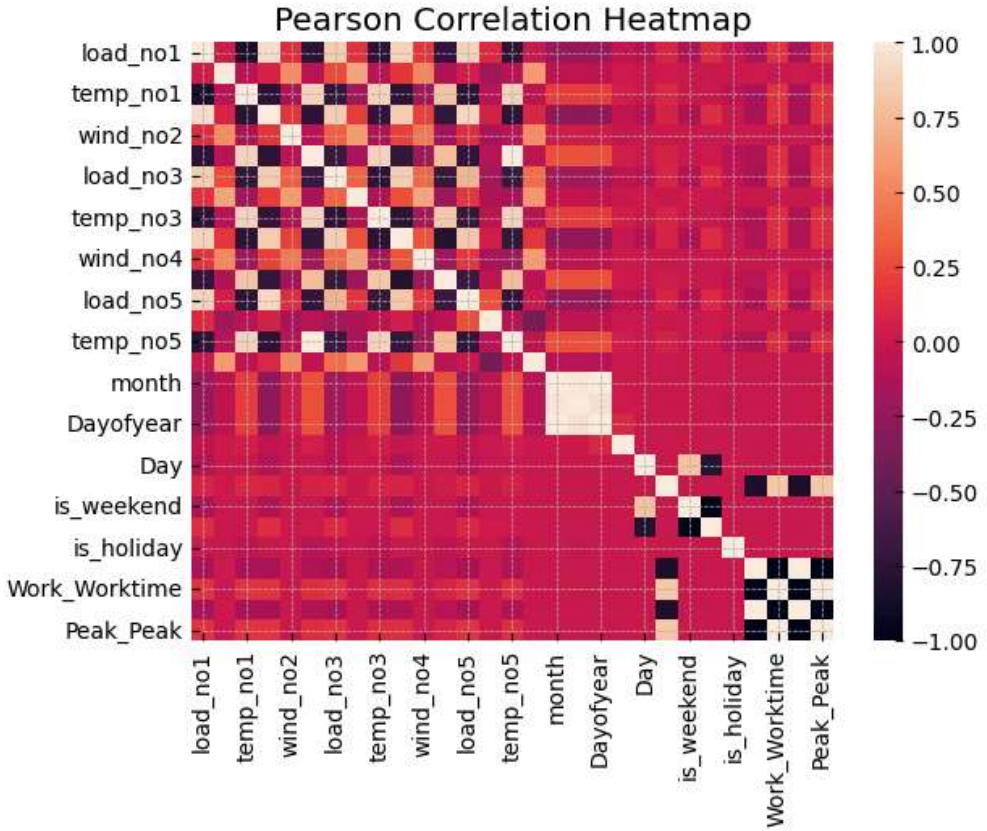
```
 13  wind_no5          81769 non-null  float64
 14  temp_no5          81769 non-null  float64
 15  year              81769 non-null  int32
 16  month             81769 non-null  int32
 17  Q                 81769 non-null  int32
 18  Dayofyear         81769 non-null  int32
 19  Dayofmonth        81769 non-null  int32
 20  Day               81769 non-null  int32
 21  hour              81769 non-null  int32
 22  is_weekend        81769 non-null  bool
 23  is_weekday        81769 non-null  bool
 24  is_holiday        81769 non-null  bool
 25  Work              81769 non-null  object
 26  Peak              81769 non-null  object
 27  Work_NonWorkTime  81769 non-null  bool
 28  Work_Worktime     81769 non-null  bool
 29  Peak_NonPeak      81769 non-null  bool
 30  Peak_Peak         81769 non-null  bool
dtypes: bool(7), float64(15), int32(7), object(2)
memory usage: 14.0+ MB
None
```

```
elec_data = elec_data.drop(["Work", "Peak"],axis=1)
elec_data.head()
```

Datetime	load_no1	wind_no1	temp_no1	load_no2	wind_no2	temp_no2	load_no3	wind_no3	temp_no3	load_no4	...	Dayofmonth	Day	hour	is_weekend	is_weekday
2015-01-01 00:00:00	4659.0	0.0	-2.5	4139.0	156.0	6.5	2370.0	259.0	3.0	2091.0	...	1	3	0	False	True
2015-01-01 01:00:00	4552.0	0.0	-2.0	4039.0	159.0	7.0	2307.0	234.0	3.0	2078.0	...	1	3	1	False	True
2015-01-01 02:00:00	4469.0	0.0	-1.5	3956.0	149.0	7.0	2273.0	200.0	3.0	2037.0	...	1	3	2	False	True
2015-01-01 03:00:00	4442.0	0.0	0.0	3900.0	144.0	7.0	2286.0	192.0	2.0	2013.0	...	1	3	3	False	True
2015-01-01 04:00:00	4488.0	0.0	0.0	3915.0	149.0	8.0	2333.0	192.0	2.0	2037.0	...	1	3	4	False	True

5 rows × 29 columns

```
pearsoncorr = elec_data.corr()
pearsoncorr
# Plot the heatmap
sns.heatmap(pearsoncorr, annot=False)
plt.title("Pearson Correlation Heatmap")
plt.show()
```



Implementing the solution

Stationarity

First, we need to check if a series is stationary or not.

A given time series is thought to consist of three systematic components including level, trend, seasonality, and one non-systematic component called noise.

These components are defined as follows:

Level: The average value in the series.

Trend: The increasing or decreasing value in the series.

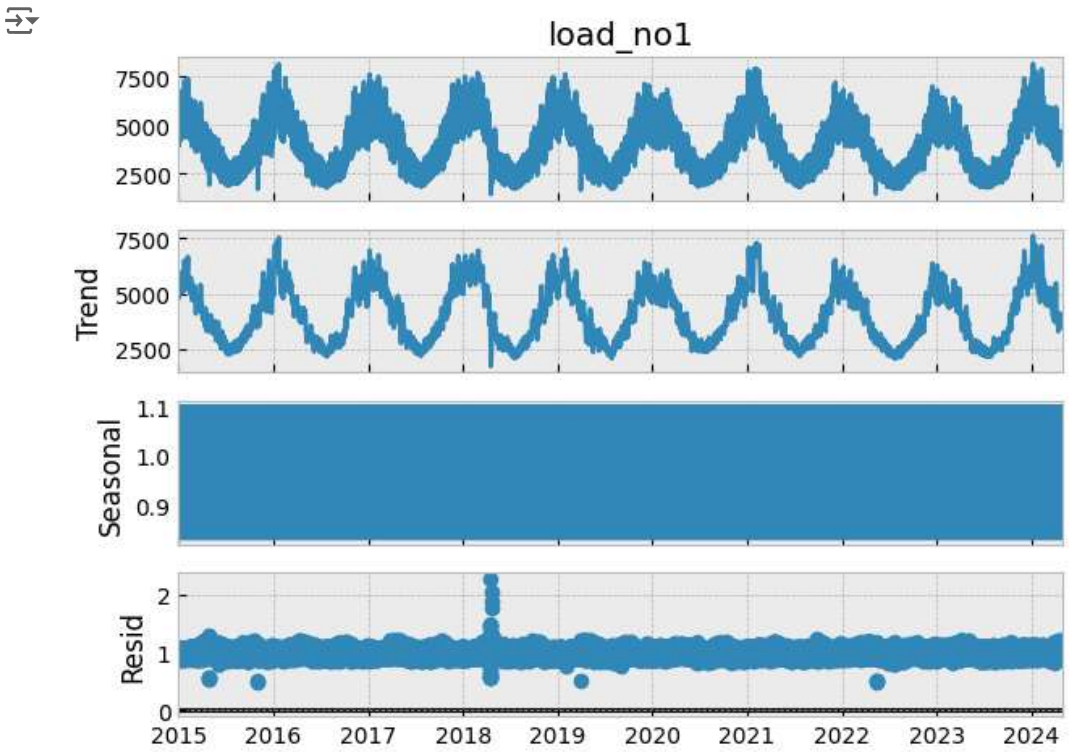
Seasonality: The repeating short-term cycle in the series.

Noise: The random variation in the series.

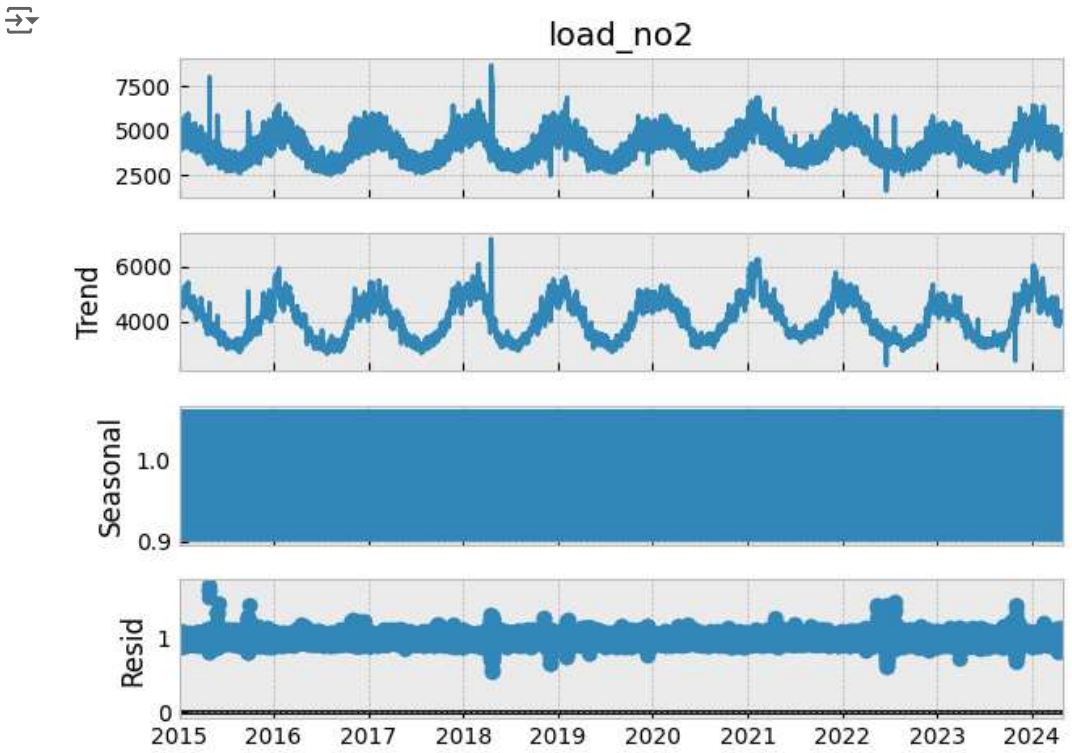
In order to perform a time series analysis, we may need to separate seasonality and trend from our series. The resultant series will become stationary through this process.

So let us separate Trend and Seasonality from the time series.

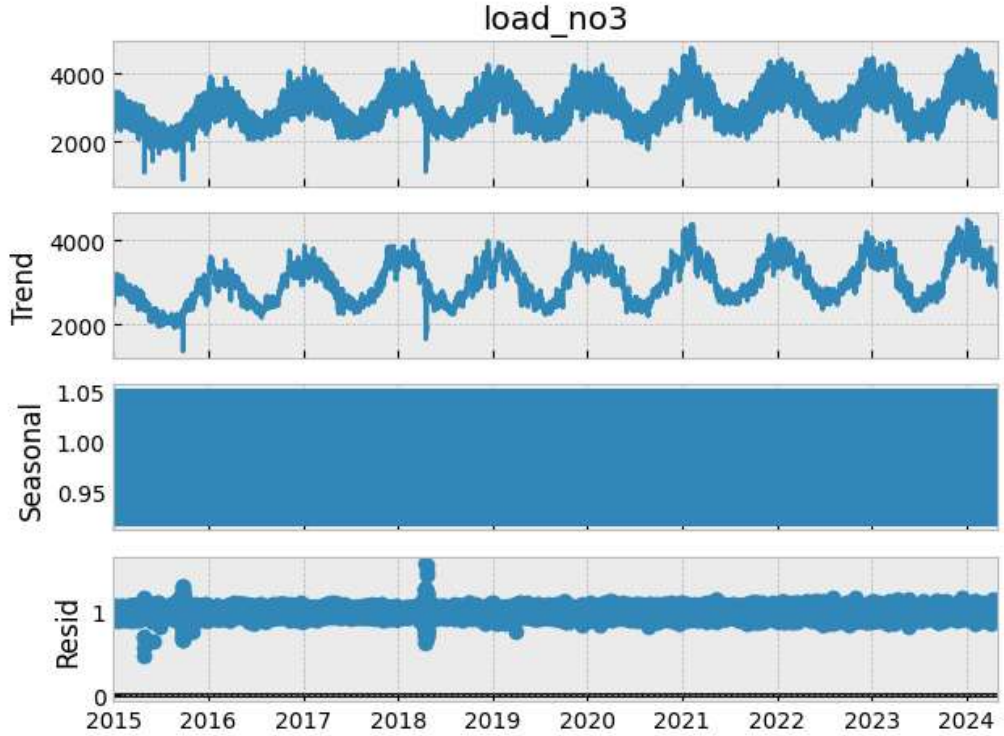
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix,accuracy_score
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.api as sm

s=sm.tsa.seasonal_decompose(elec_data.load_no1, model='multiplicative')
s.plot()
plt.show()
```



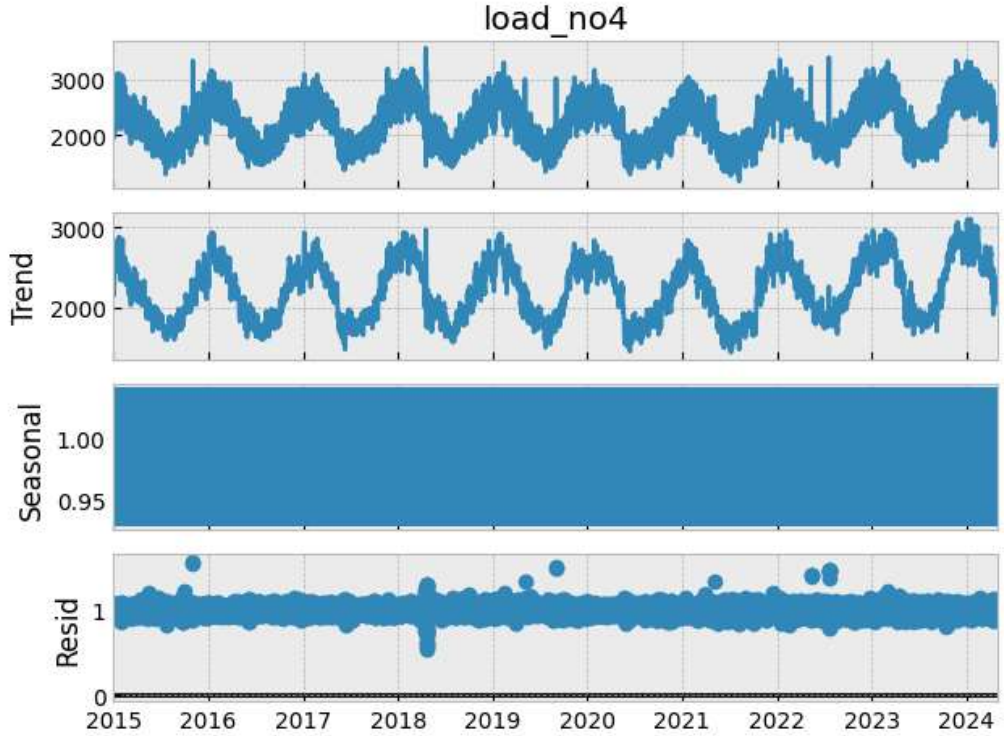
```
s=sm.tsa.seasonal_decompose(elec_data.load_no2, model='multiplicative')
s.plot()
plt.show()
```



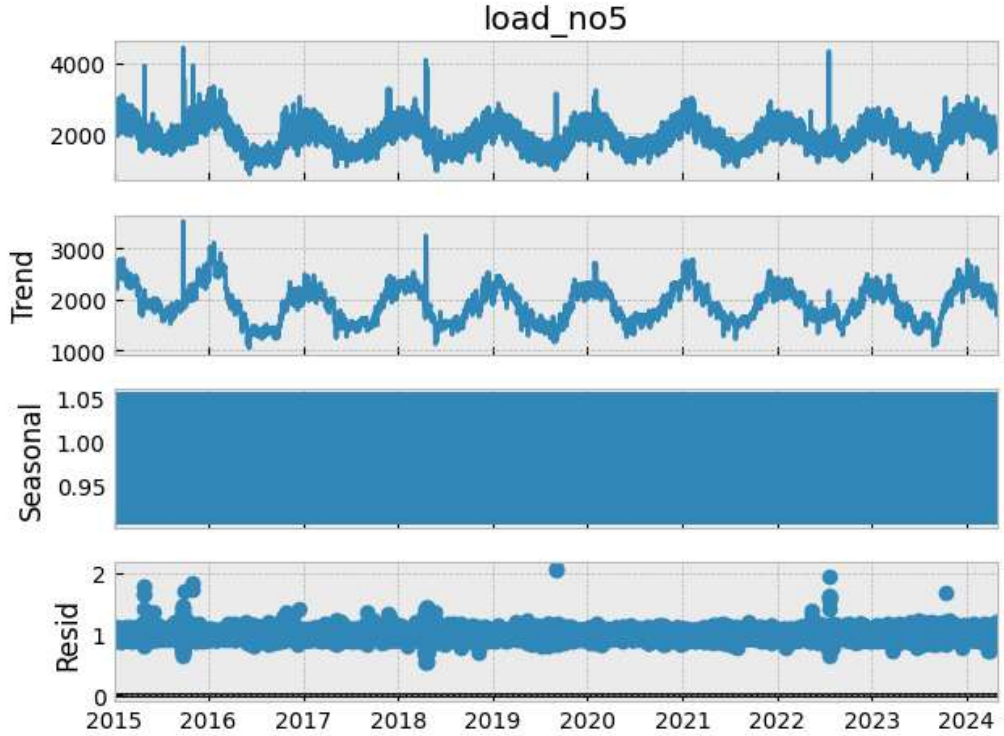

```
s=sm.tsa.seasonal_decompose(elec_data.load_no3, model='multiplicative')
s.plot()
plt.show()
```




```
s=sm.tsa.seasonal_decompose(elec_data.load_no4, model='multiplicative')
s.plot()
plt.show()
```




```
from statsmodels.tsa.seasonal import seasonal_decompose
s=sm.tsa.seasonal_decompose(elec_data.load_no5, model='multiplicative')
s.plot()
plt.show()
```



This gives us more insight into our data and real-world actions. Clearly, there is an upward and downward trend and a recurring event where electricity consumption shoots maximum and minimum every year.

ADF (Augmented Dickey-Fuller) Test

The Dickey-Fuller test is one of the most popular statistical tests. It can be used to determine the presence of unit root in the series, and hence help us understand if the series is stationary or not. The null and alternate hypothesis of this test is:

Null Hypothesis: The series has a unit root. (series is not stationary)

Alternate Hypothesis: The series has no unit root.

If we fail to reject the null hypothesis, we can say that the series is non-stationary. This means that the series can be linear or difference stationary (we will understand more about difference stationary in the next section).

If both mean and standard deviation are flat lines(constant mean and constant variance), the series becomes stationary.

The following function is one that can plot a series with its rolling mean and standard deviation.

```
from prettytable import PrettyTable
from statsmodels.tsa.stattools import adfuller

# Example ADF test results (replace with your actual results)
dftest = adfuller(elec['load_no1'], autolag = 'AIC')
dftest2 = adfuller(elec['load_no2'], autolag = 'AIC')
dftest3 = adfuller(elec['load_no3'], autolag = 'AIC')
dftest4 = adfuller(elec['load_no4'], autolag = 'AIC')
dftest5 = adfuller(elec['load_no5'], autolag = 'AIC')

adf_results = [
    ("load_no1", dftest[0], dftest[1], dftest[2], dftest[3], dftest[4]),
    ("load_no2", dftest2[0], dftest2[1], dftest2[2], dftest2[3],dftest2[4]),
    ("load_no3", dftest3[0], dftest3[1], dftest3[2], dftest3[3],dftest3[4]),
    ("load_no4", dftest4[0], dftest4[1], dftest4[2], dftest4[3],dftest4[4]),
    ("load_no5", dftest5[0], dftest5[1], dftest5[2], dftest5[3],dftest5[4])
]

# Initialize the table with column names
table = PrettyTable(["Series", "ADF", "P-Value", "Num Of Lags", "Num Of Observations", "Critical Values"])

# Add rows
for row in adf_results:
    table.add_row(row)

# Display the table
print(table)
```

Series	ADF	P-Value	Num Of Lags	Num Of Observations	Critical Values
load_no1	-5.331974935747544	4.6995159683839465e-06	65	81703	{'1%': -3.430430039967484, '5%': -2.861575376324344, '10%': -2.56
load_no2	-6.616094466873617	6.202030976603557e-09	65	81703	{'1%': -3.430430039967484, '5%': -2.861575376324344, '10%': -2.56
load_no3	-6.052269565406708	1.2692467479622716e-07	65	81703	{'1%': -3.430430039967484, '5%': -2.861575376324344, '10%': -2.56
load_no4	-6.500049520595319	1.167953335435998e-08	65	81703	{'1%': -3.430430039967484, '5%': -2.861575376324344, '10%': -2.56
load_no5	-6.582042144424071	7.472199494498441e-09	65	81703	{'1%': -3.430430039967484, '5%': -2.861575376324344, '10%': -2.56

We see that the p-value is greater than 0.05 so we cannot reject the Null hypothesis. Also, the test statistics is greater than the critical values. so the data is non-stationary.

To get a stationary series, we need to eliminate the trend and seasonality from the series.

We start by taking a log of the series to reduce the magnitude of the values and reduce the rising trend in the series. Then after getting the log of the series, we find the rolling average of the series. A rolling average is calculated by taking input for the past 12 months and giving a mean consumption value at every point further ahead in series.

```
df_log = np.log(elec['load_no1'])
moving_avg = df_log.rolling(12).mean()
std_dev = df_log.rolling(12).std()


df_log2 = np.log(elec['load_no2'])
moving_avg2 = df_log2.rolling(12).mean()
std_dev2 = df_log2.rolling(12).std()


df_log3 = np.log(elec['load_no3'])
moving_avg3 = df_log3.rolling(12).mean()
std_dev3 = df_log3.rolling(12).std()


df_log4 = np.log(elec['load_no4'])
moving_avg4 = df_log4.rolling(12).mean()
std_dev4 = df_log4.rolling(12).std()


df_log5 = np.log(elec['load_no5'])
moving_avg5 = df_log5.rolling(12).mean()
std_dev5 = df_log5.rolling(12).std()
```

After finding the mean, we take the difference of the series and the mean at every point in the series.

This way, we eliminate trends out of a series and obtain a more stationary series.

Perform the Dickey-Fuller test (ADFT) once again. We have to perform this function every time to check whether the data is stationary or not.

```
df_log_moving_avg_diff = df_log-moving_avg
df_log_moving_avg_diff.dropna(inplace=True)

df_log_moving_avg_diff2 = df_log2-moving_avg2
df_log_moving_avg_diff2.dropna(inplace=True)

df_log_moving_avg_diff3 = df_log3-moving_avg3
df_log_moving_avg_diff3.dropna(inplace=True)

df_log_moving_avg_diff4 = df_log4-moving_avg4
df_log_moving_avg_diff4.dropna(inplace=True)

df_log_moving_avg_diff5 = df_log5-moving_avg5
df_log_moving_avg_diff5.dropna(inplace=True)
```

```
adftest = adfuller(df_log_moving_avg_diff, autolag = 'AIC')
adftest2 = adfuller(df_log_moving_avg_diff2, autolag = 'AIC')
adftest3 = adfuller(df_log_moving_avg_diff3, autolag = 'AIC')
adftest4 = adfuller(df_log_moving_avg_diff4, autolag = 'AIC')
adftest5 = adfuller(df_log_moving_avg_diff5, autolag = 'AIC')
```

```
adf2_results = [
    ("load_no1", adftest[0], adftest[1], adftest[2], adftest[3],adftest[4] ),
    ("load_no2", adftest2[0], adftest2[1], adftest2[2], adftest2[3], adftest2[4]),
    ("load_no3", adftest3[0], adftest3[1], adftest3[2], adftest3[3], adftest3[4]),
    ("load_no4", adftest4[0], adftest4[1], adftest4[2], adftest4[3], adftest4[4]),
    ("load_no5", adftest5[0], adftest5[1], adftest5[2], adftest5[3], adftest5[4])
]
# for key,values in adft[4].items():
#         output['critical value (%s)'%key] =  values
#     print(output)

# Initialize the table with column names
table = PrettyTable(["Series", "ADF", "P-Value", "Num Of Lags", "Num Of Observations", "critical values"])

# Add rows
for row in adf2_results:
    table.add_row(row)

# Display the table
print(table)
```

```
+----------+--------------------+---------+-------------+---------------------+-----------------------------------------------------------------------------------------
|  Series  |         ADF        | P-Value | Num Of Lags | Num Of Observations |                                  critical values
+----------+--------------------+---------+-------------+---------------------+-----------------------------------------------------------------------------------------
| load_no1 |  -47.71983377684181 |   0.0   |      65     |        81692        | {'1%': -3.430430050745373, '5%': -2.861575381087926, '10%': -2.5667888321302605
| load_no2 |  -48.87347024321383 |   0.0   |      65     |        81692        | {'1%': -3.430430050745373, '5%': -2.861575381087926, '10%': -2.5667888321302605
| load_no3 | -49.419137570948024 |   0.0   |      65     |        81692        | {'1%': -3.430430050745373, '5%': -2.861575381087926, '10%': -2.5667888321302605
| load_no4 |  -48.78141447876949 |   0.0   |      65     |        81692        | {'1%': -3.430430050745373, '5%': -2.861575381087926, '10%': -2.5667888321302605
| load_no5 |  -47.96702583673493 |   0.0   |      65     |        81692        | {'1%': -3.430430050745373, '5%': -2.861575381087926, '10%': -2.5667888321302605
+----------+--------------------+---------+-------------+---------------------+-----------------------------------------------------------------------------------------
```

From the above graph, we observed that the data attained stationarity.

One of the modules is completed as we came to a conclusion. We need to check the weighted average, to understand the trend of the data in time series. Take the previous log data and to perform the following operation.

From the above graph, we observed that the data attained stationarity. We also see that the test statistics and critical value is relatively equal.

There can be cases when there is a high seasonality in the data. In those cases, just removing the trend will not help much. We need to also take care of the seasonality in the series. One such method for this task is differencing.

Differencing is a method of transforming a time series dataset. It can be used to remove the series dependence on time, so-called temporal dependence. This includes structures like trends and seasonality. Differencing can help stabilize the mean of the time series by removing changes in the level of a time series, and so eliminating (or reducing) trend and seasonality.

Differencing is performed by subtracting the previous observation from the current observation.

```
df_log_diff = df_log - df_log.shift()
df_log_diff.dropna(inplace=True)

df_log_diff2 = df_log2 - df_log2.shift()
df_log_diff2.dropna(inplace=True)

df_log_diff3 = df_log3 - df_log3.shift()
df_log_diff3.dropna(inplace=True)

df_log_diff4 = df_log4 - df_log4.shift()
df_log_diff4.dropna(inplace=True)

df_log_diff5 = df_log5 - df_log5.shift()
df_log_diff5.dropna(inplace=True)
```

Seasonality

Before we go on to build our forecasting model, we need to determine optimal parameters for our model. For those optimal parameters, we need ACF and PACF plots.

A nonseasonal ARIMA model is classified as an “ARIMA(p,d,q)” model, where:

p → Number of autoregressive terms, d → Number of nonseasonal differences needed for stationarity, and q → Number of lagged forecast errors in the prediction equation.

Values of p and q come through ACF and PACF plots. So let us understand both ACF and PACF!

Autocorrelation and Partial Autocorrelation

Autocorrelation Function(ACF) Statistical correlation summarizes the strength of the relationship between two variables. Pearson’s correlation coefficient is a number between -1 and 1 that describes a negative or positive correlation respectively. A value of zero indicates no correlation.

We can calculate the correlation for time series observations with previous time steps, called lags. Because the correlation of the time series observations is calculated with values of the same series at previous times, this is called a serial correlation, or an autocorrelation.

A plot of the autocorrelation of a time series by lag is called the AutoCorrelation Function, or the acronym ACF. This plot is sometimes called a correlogram or an autocorrelation plot.

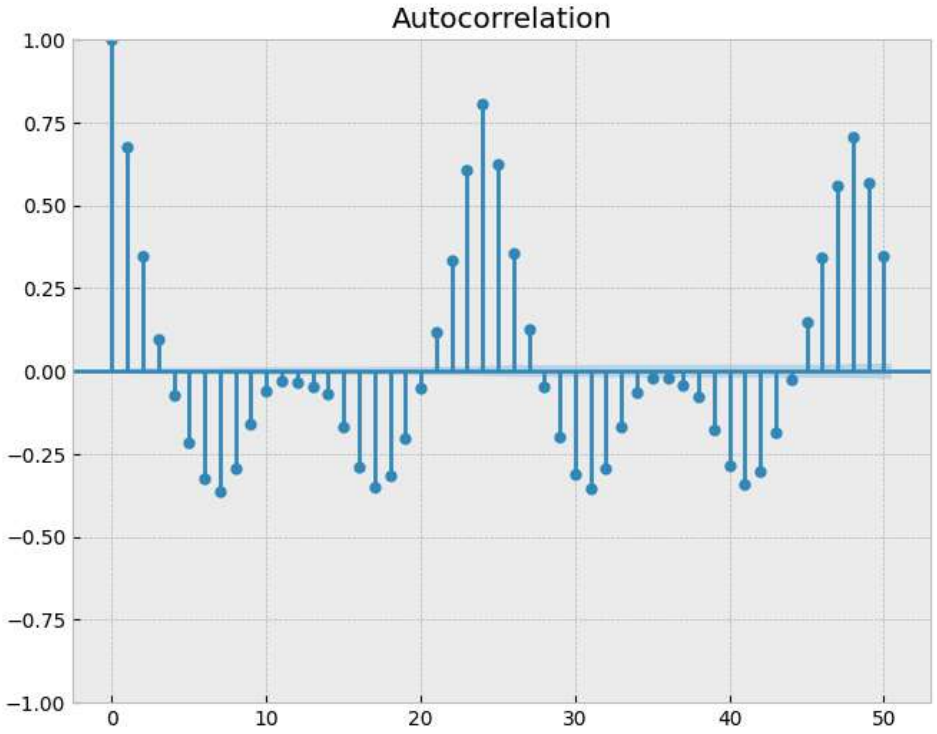
Partial Autocorrelation Function(PACF) A partial autocorrelation is a summary of the relationship between an observation in a time series with observations at prior time steps with the relationships of intervening observations removed.

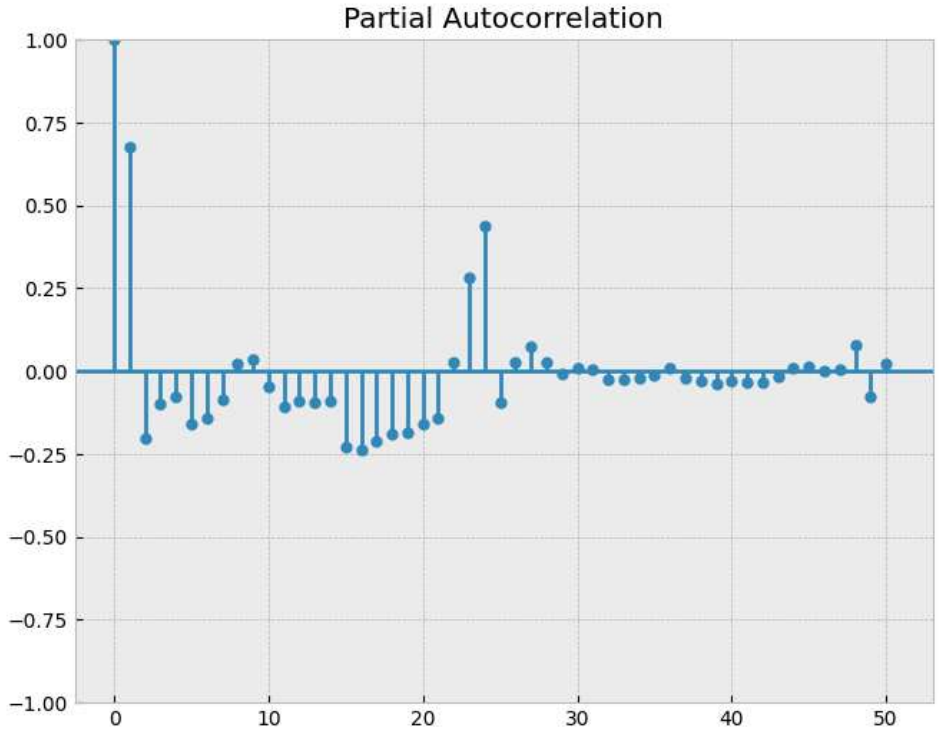
The partial autocorrelation at lag k is the correlation that results after removing the effect of any correlations due to the terms at shorter lags.

The autocorrelation for observation and observation at a prior time step is comprised of both the direct correlation and indirect correlations. It is these indirect correlations that the partial autocorrelation function seeks to remove.

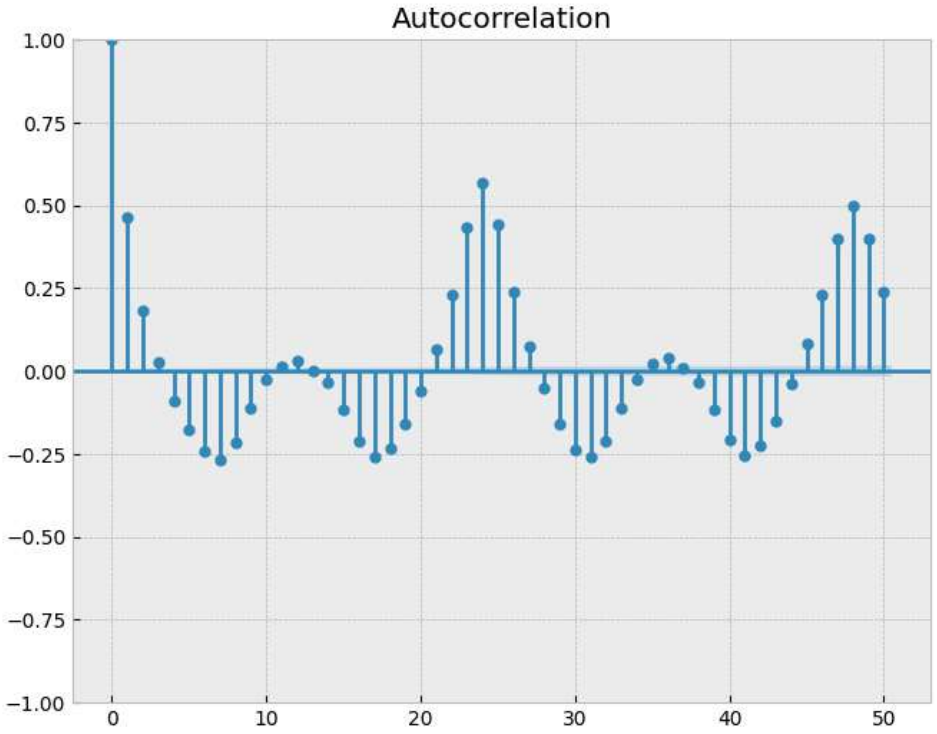
Below code plots, both ACF and PACF plots for us:

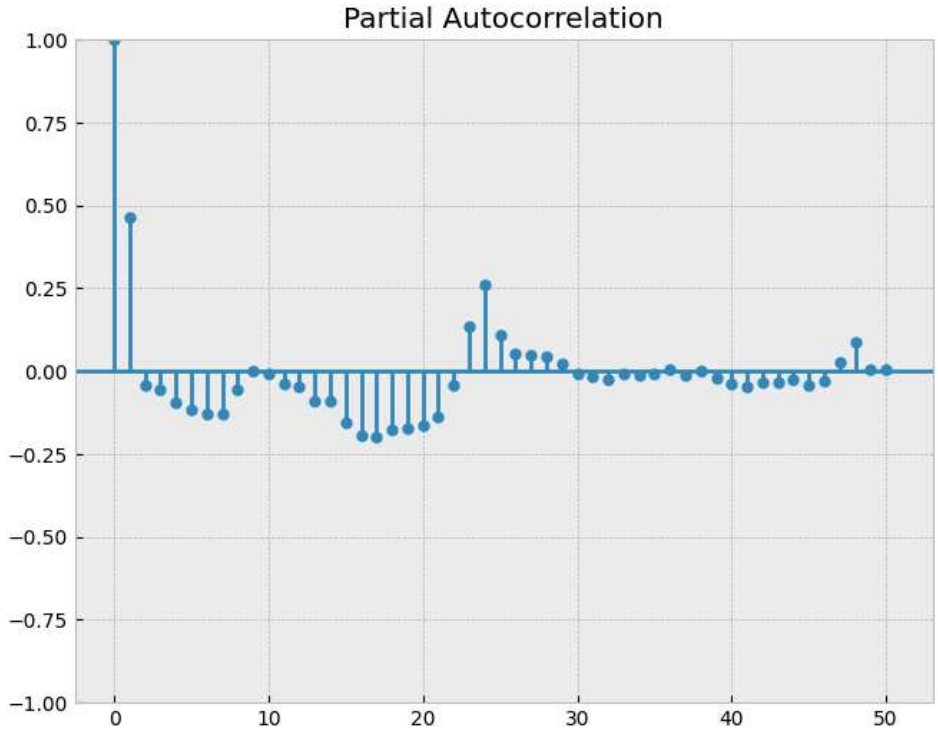
```
from statsmodels.tsa.stattools import acf,pacf
# we use d value here(data_log_shift)

# Graph data
fig, axes = plt.subplots(1, 2, figsize=(17,6))


fig = sm.graphics.tsa.plot_acf(df_log_diff, lags=50, ax=axes[0])

fig = sm.graphics.tsa.plot_pacf(df_log_diff, lags=50, ax=axes[1])
plt.show()
```



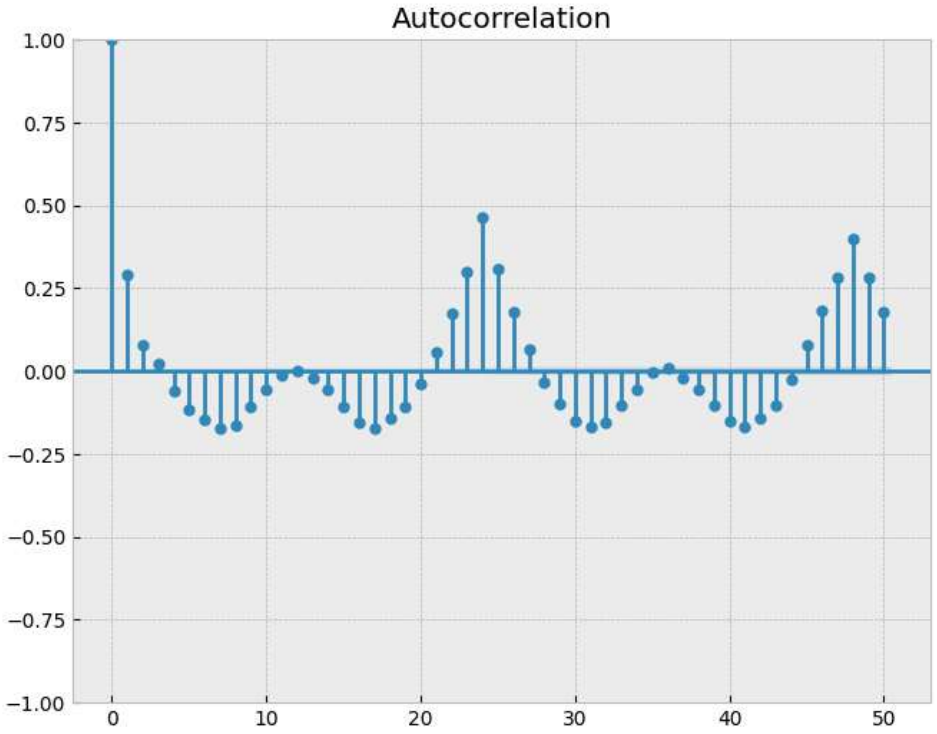


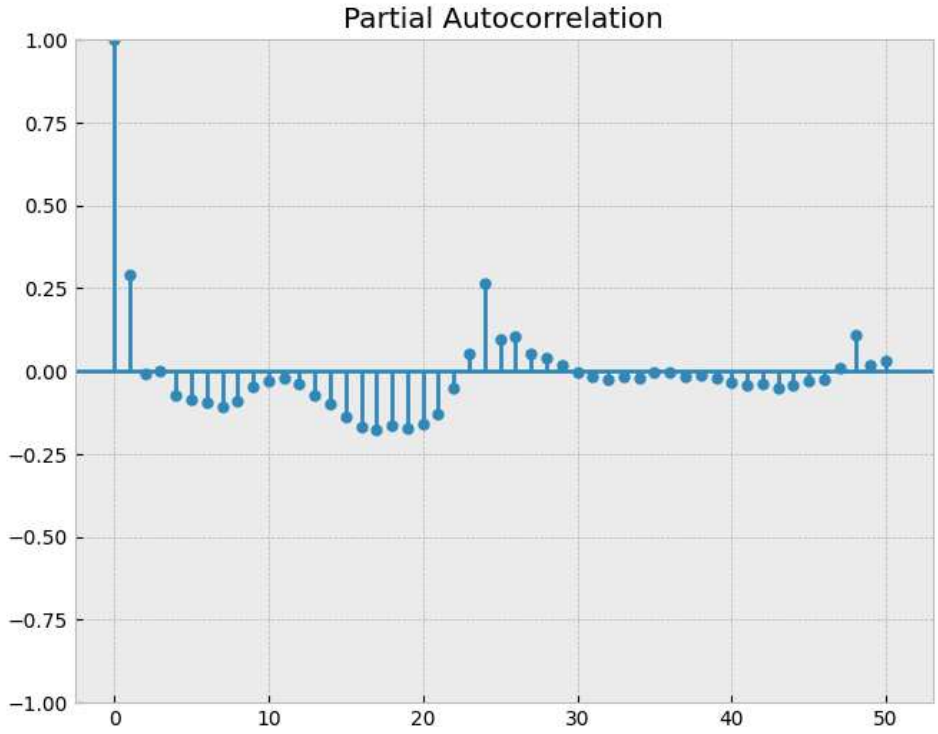

```
# Graph data
fig, axes = plt.subplots(1, 2, figsize=(17,6))


fig2 = sm.graphics.tsa.plot_acf(df_log_diff2, lags=50, ax=axes[0])

fig2 = sm.graphics.tsa.plot_pacf(df_log_diff2, lags=50, ax=axes[1])
plt.show()
```



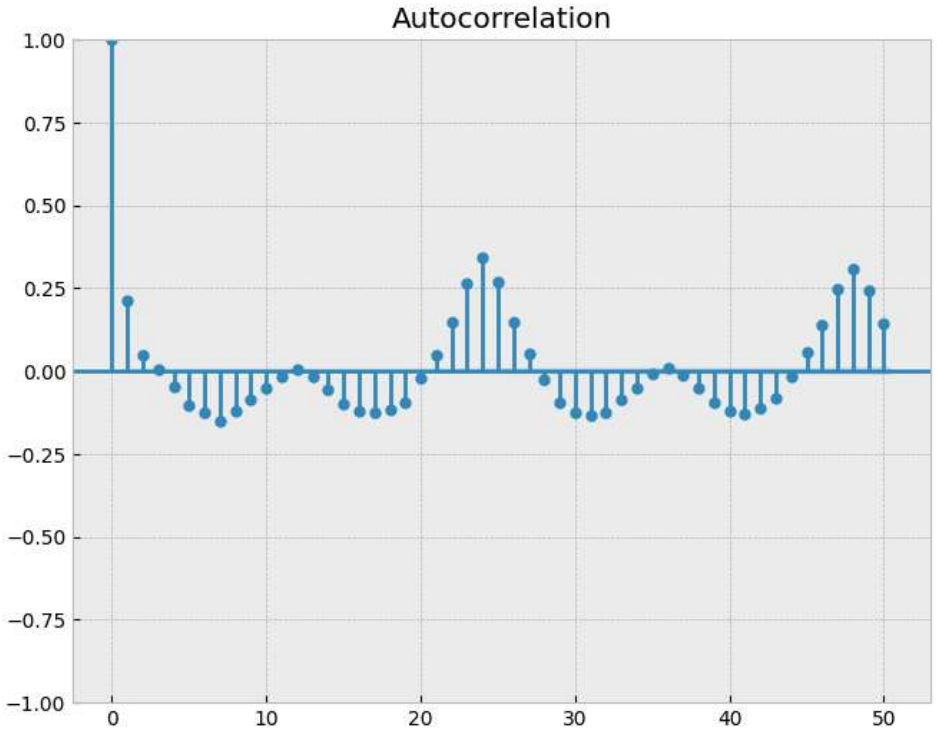


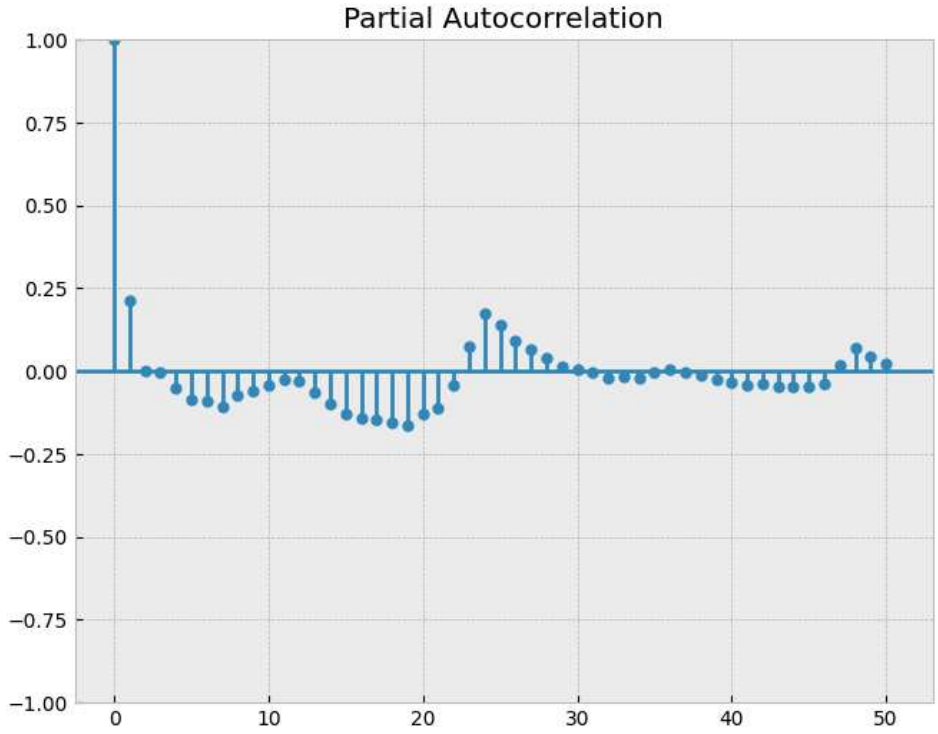

```
# Graph data
fig, axes = plt.subplots(1, 2, figsize=(17,6))


fig3 = sm.graphics.tsa.plot_acf(df_log_diff3, lags=50, ax=axes[0])

fig3 = sm.graphics.tsa.plot_pacf(df_log_diff3, lags=50, ax=axes[1])
plt.show()
```



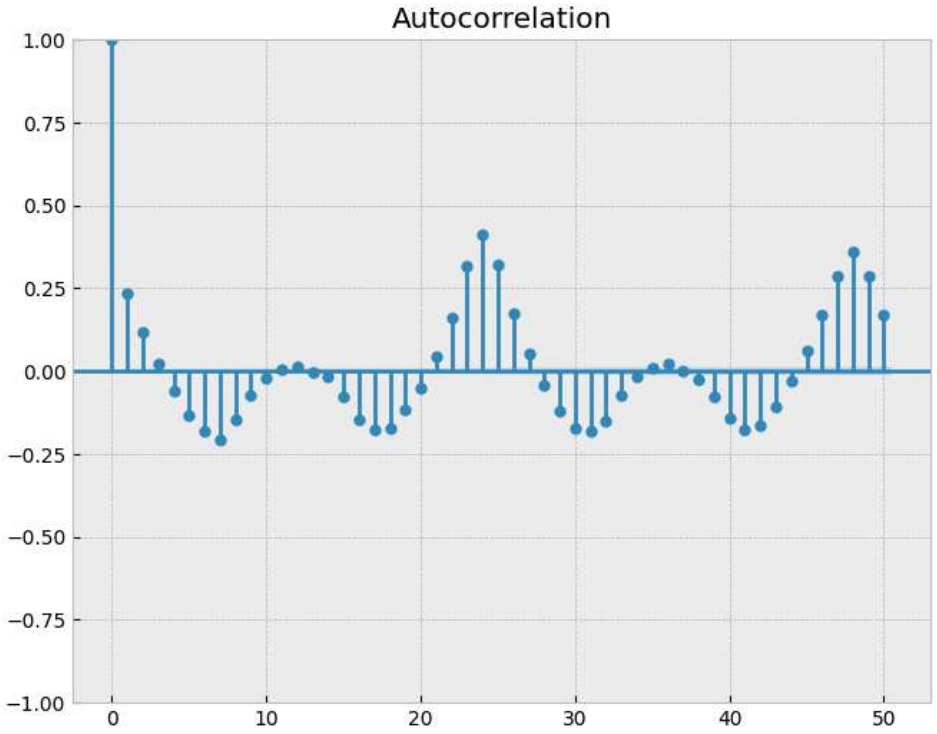


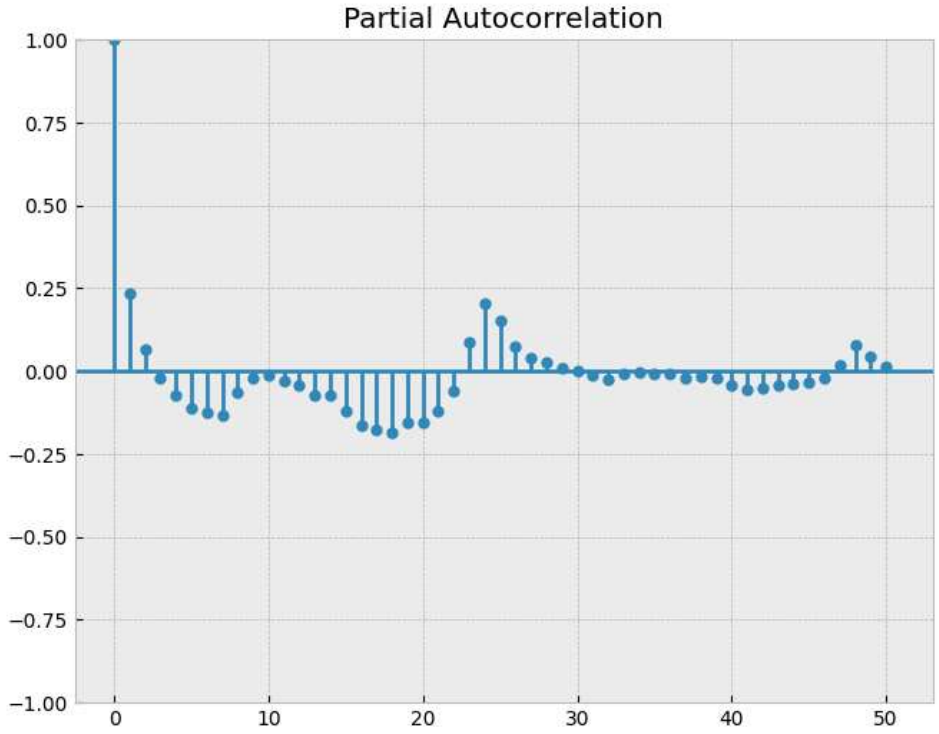

```
# Graph data
fig, axes = plt.subplots(1, 2, figsize=(17,6))


fig4 = sm.graphics.tsa.plot_acf(df_log_diff4, lags=50, ax=axes[0])

fig4 = sm.graphics.tsa.plot_pacf(df_log_diff4, lags=50, ax=axes[1])
plt.show()
```






```
# Graph data
fig, axes = plt.subplots(1, 2, figsize=(17,6))


fig5 = sm.graphics.tsa.plot_acf(df_log_diff5, lags=50, ax=axes[0])

fig5 = sm.graphics.tsa.plot_pacf(df_log_diff5, lags=50, ax=axes[1])
plt.show()
```





Prepping the data

```
loadno1 = elec_data.drop(columns=['load_no2','temp_no2', 'wind_no2',
                            'load_no3','temp_no3', 'wind_no3',
                            'load_no4','temp_no4', 'wind_no4',
                            'load_no5','temp_no5', 'wind_no5'])
# loadno1.index = pd.to_datetime(loadno1['Datetime'], format='%Y-%m-%d')
# del loadno1['Datetime']

loadno2 = elec_data.drop(columns=['load_no1','temp_no1', 'wind_no1',
                            'load_no3','temp_no3', 'wind_no3',
                            'load_no4','temp_no4', 'wind_no4',
                            'load_no5','temp_no5', 'wind_no5'])
# loadno2.index = pd.to_datetime(loadno2['Datetime'], format='%Y-%m-%d')
# del loadno2['Datetime']

loadno3 = elec_data.drop(columns=['load_no1','temp_no1', 'wind_no1',
                            'load_no2','temp_no2', 'wind_no2',
                            'load_no4','temp_no4', 'wind_no4',
                            'load_no5','temp_no5', 'wind_no5'])
# loadno3.index = pd.to_datetime(loadno3['Datetime'], format='%Y-%m-%d')
# del loadno3['Datetime']

loadno4 = elec_data.drop(columns=['load_no1','temp_no1', 'wind_no1',
                            'load_no2','temp_no2', 'wind_no2',
                            'load_no3','temp_no3', 'wind_no3',
                            'load_no5','temp_no5', 'wind_no5'])
# loadno4.index = pd.to_datetime(loadno4['Datetime'], format='%Y-%m-%d')
# del loadno4['Datetime']

loadno5 = elec_data.drop(columns=['load_no1','temp_no1', 'wind_no1',
                            'load_no2','temp_no2', 'wind_no2',
                            'load_no3','temp_no3', 'wind_no3',
                            'load_no4','temp_no4', 'wind_no4'])
# loadno5.index = pd.to_datetime(loadno5['Datetime'], format='%Y-%m-%d')
# del loadno5['Datetime']

loadno1.info(1)
# print(loadno2.head(1))
# print(loadno3.head(1))
# print(loadno4.head(1))
# print(loadno5.head(1))
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 81769 entries, 2015-01-01 00:00:00 to 2024-04-30 00:00:00
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   load_no1          81769 non-null  float64
```

```
 1   wind_no1          81769 non-null  float64
 2   temp_no1          81769 non-null  float64
 3   year              81769 non-null  int32
 4   month             81769 non-null  int32
 5   Q                 81769 non-null  int32
 6   Dayofyear         81769 non-null  int32
 7   Dayofmonth        81769 non-null  int32
 8   Day               81769 non-null  int32
 9   hour              81769 non-null  int32
 10  is_weekend        81769 non-null  bool
 11  is_weekday        81769 non-null  bool
 12  is_holiday        81769 non-null  bool
 13  Work_NonWorkTime  81769 non-null  bool
 14  Work_Worktime     81769 non-null  bool
 15  Peak_NonPeak      81769 non-null  bool
 16  Peak_Peak         81769 non-null  bool
dtypes: bool(7), float64(3), int32(7)
memory usage: 5.2 MB
```

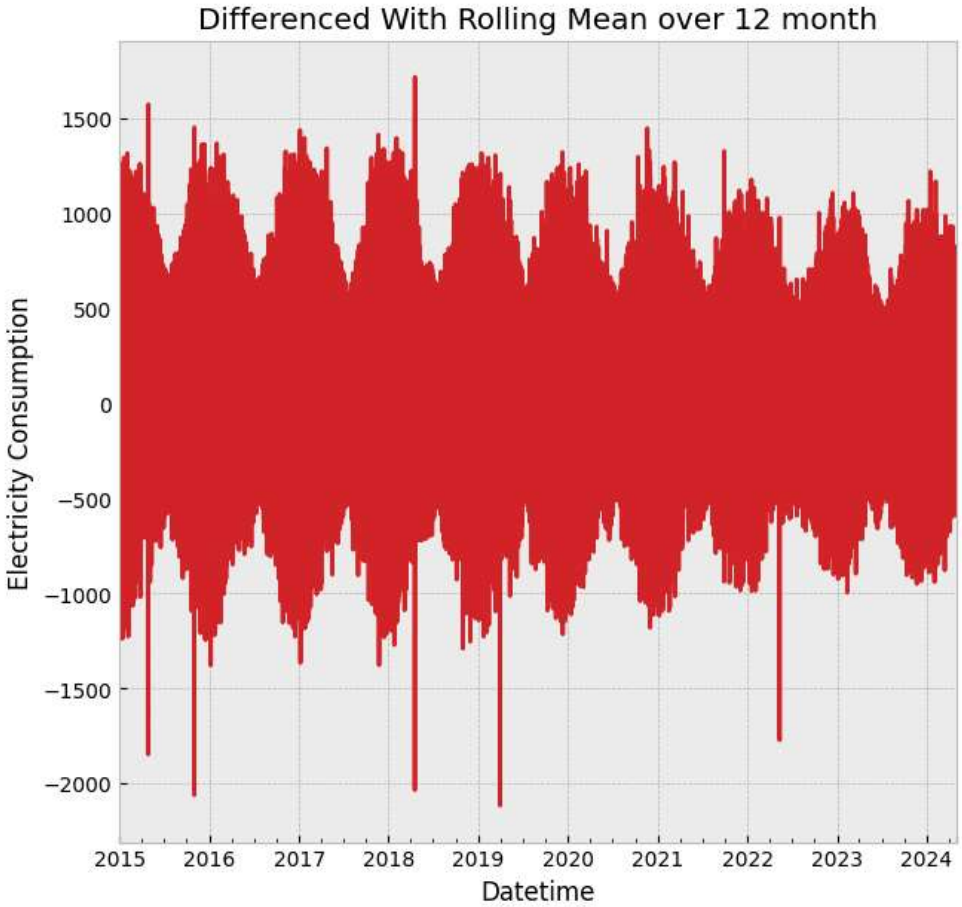
Fitting the Models

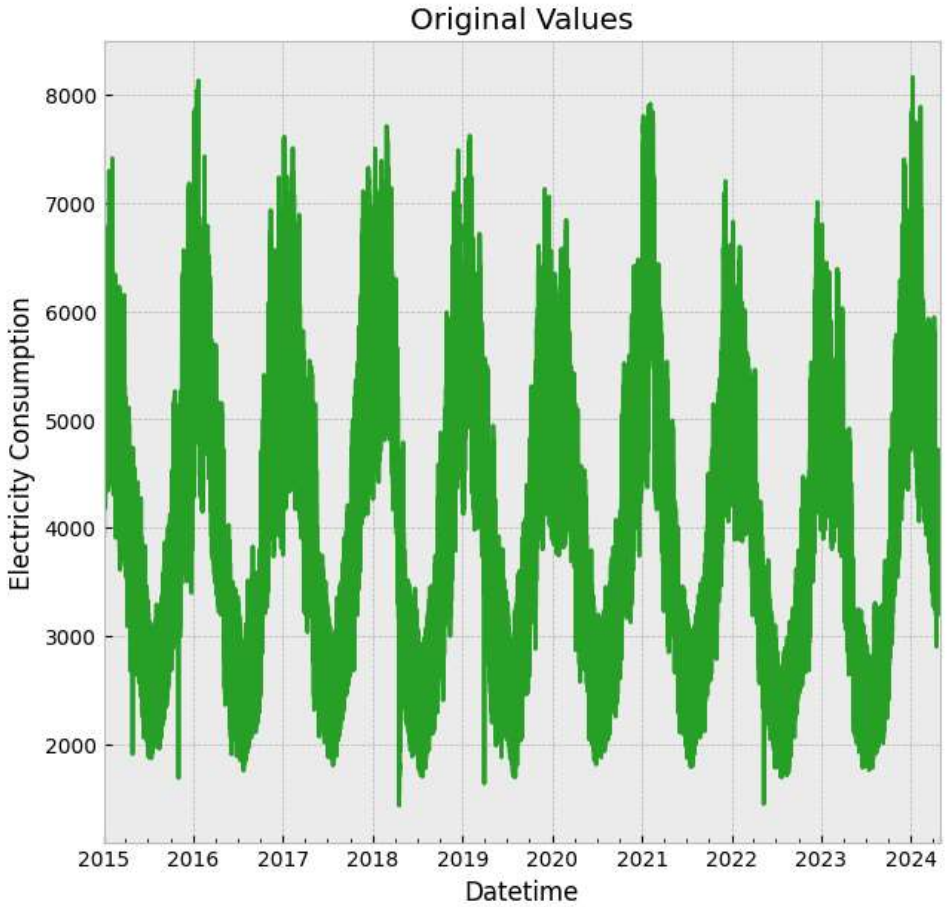
ARIMA Model

In order to find the p and q values from the above graphs, we need to check, where the graph cuts off the origin or drops to zero for the first time from the above graphs the p and q values are merely close to 3 where the graph cuts off the origin (draw the line to x-axis) now we have p,d,q values. So now we can substitute in the ARIMA model and let's see the output.

ZONE 1

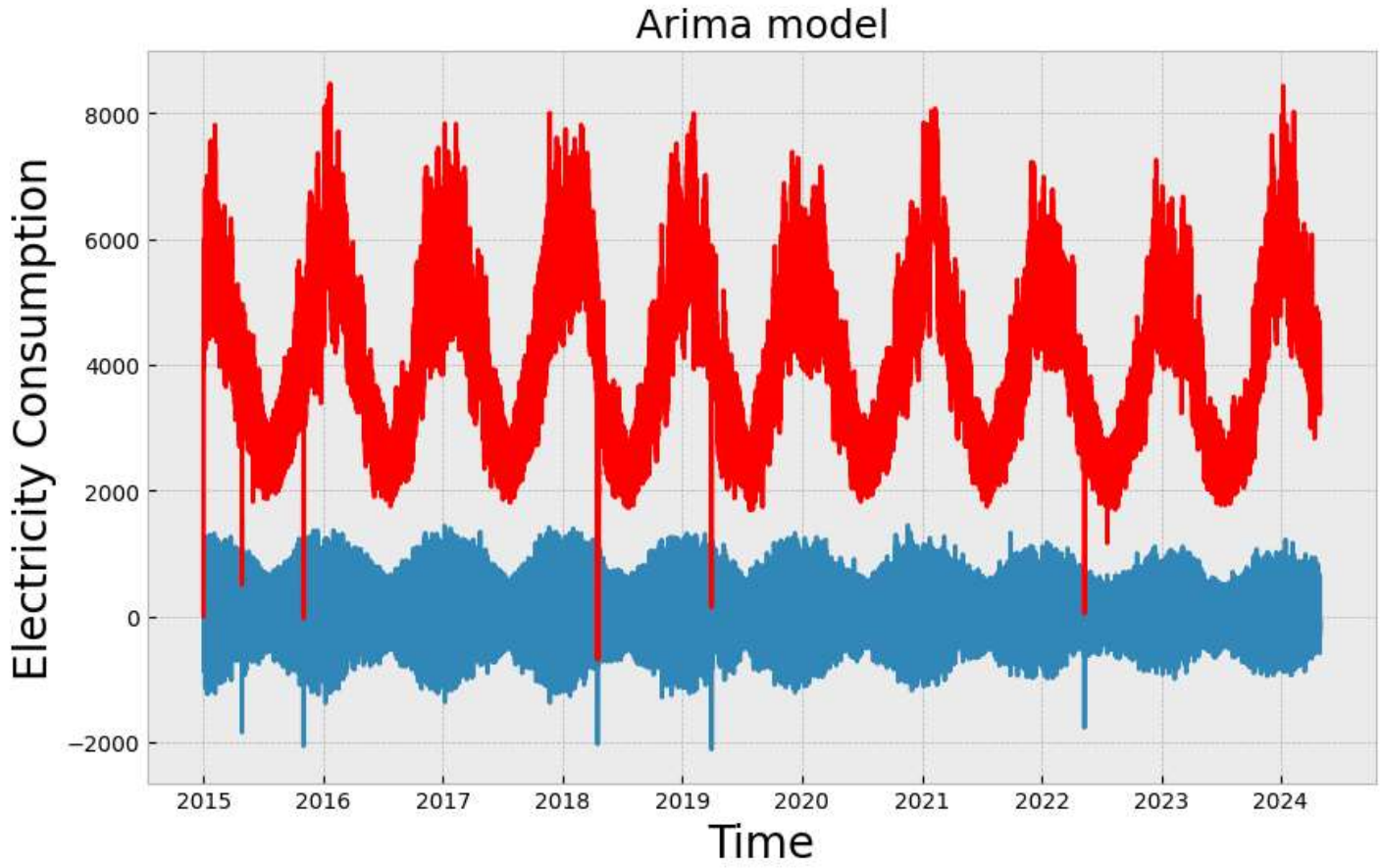
```
rolling_mean = loadno1['load_no1'].rolling(window = 12).mean()
consumption_rolled_detrended = loadno1['load_no1'] - rolling_mean

ax1 = plt.subplot(121)
consumption_rolled_detrended.plot(figsize=(16,7),color="tab:red", title="Differenced With Rolling Mean over 12 month", ax=ax1);
plt.ylabel("Electricity Consumption")
ax2 = plt.subplot(122)
loadno1['load_no1'].plot(figsize=(16,7), color="tab:green", title="Original Values", ax=ax2);
plt.ylabel("Electricity Consumption")
plt.show()
```


Differenced With Rolling Mean over 12 month
Electricity Consumption
1500
1000
500
0
−500
−1000
−1500
−2000
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Datetime


Original Values
Electricity Consumption
8000
7000
6000
5000
4000
3000
2000
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Datetime

```
from statsmodels.tsa.arima.model import ARIMA

# Moving Average Model.
#
figure, ax = plt.subplots(figsize=(10,6))
model = ARIMA(loadno1['load_no1'], order=(3,1,2))
results_ARIMA = model.fit()
plt.plot(consumption_rolled_detrended)
plt.plot(results_ARIMA.fittedvalues, color="red")
plt.xlabel("Time", fontsize="20")
plt.ylabel("Electricity Consumption", fontsize="20")
plt.title("Arima model", fontsize="18")
plt.show()
```

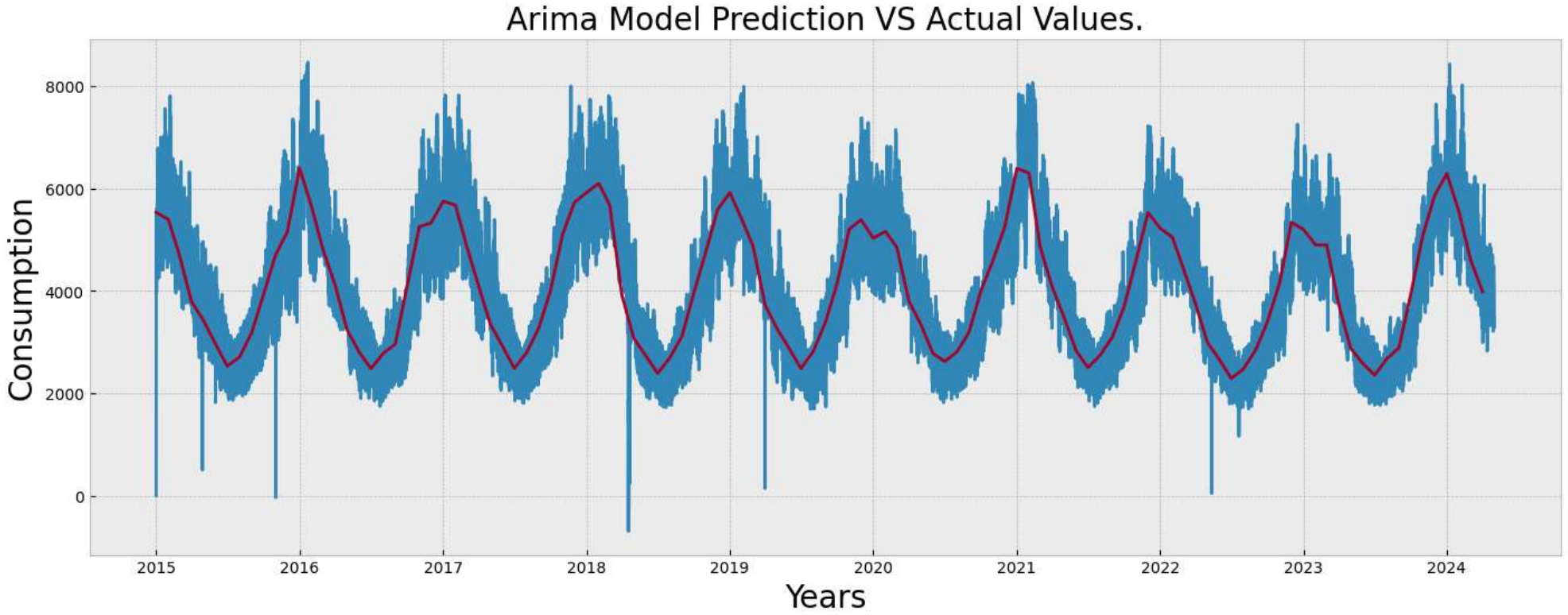
```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters found. Using z
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using zeros as star
  warn('Non-invertible starting MA parameters found.'
```



```
consumption = loadno1['load_no1'].resample('MS').mean()
consumption2 = loadno2['load_no2'].resample('MS').mean()
consumption3 = loadno3['load_no3'].resample('MS').mean()
consumption4 = loadno4['load_no4'].resample('MS').mean()
consumption5 = loadno5['load_no5'].resample('MS').mean()
```

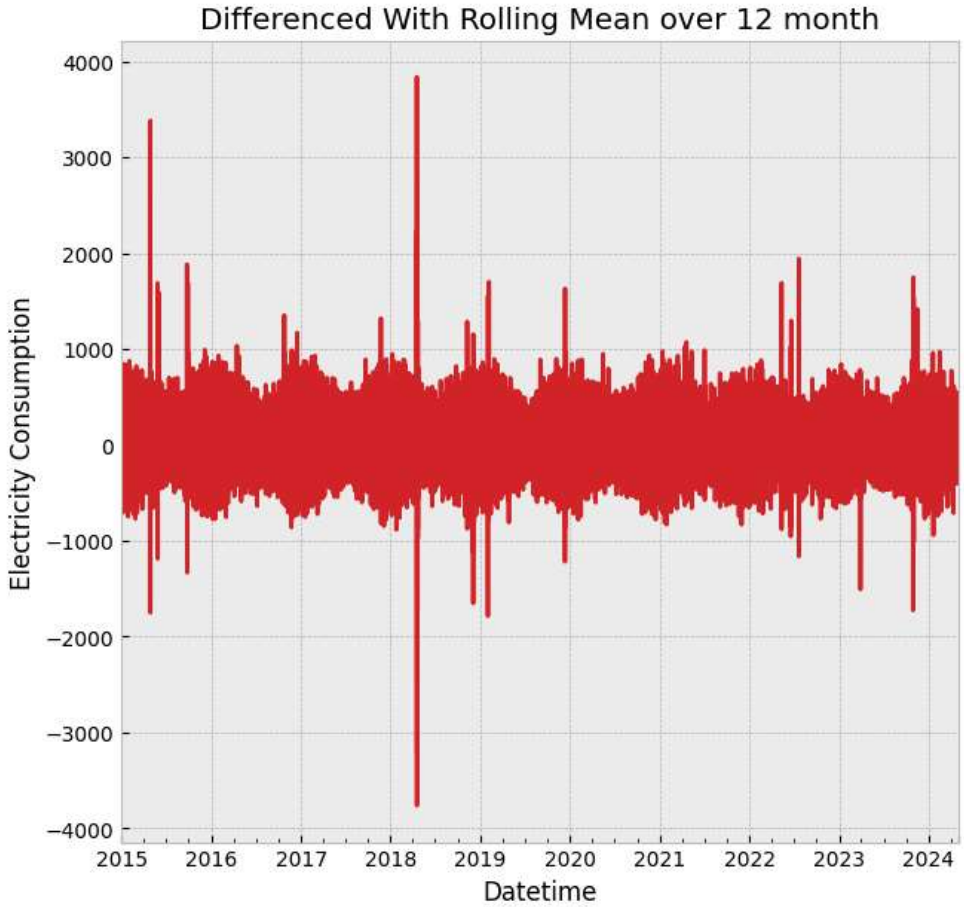
```
pred_ARIMA = pd.Series(results_ARIMA.fittedvalues, copy=True)

figure, ax = plt.subplots(figsize=(17,6))
plt.plot(pred_ARIMA)
plt.plot(consumption)
print(f"RMSE: {np.sqrt(sum((pred_ARIMA-consumption)**2)/len(consumption))}")
plt.xlabel("Years", fontsize="20")
plt.ylabel("Consumption", fontsize="20")
plt.title("Arima Model Prediction VS Actual Values.", fontsize="20")
plt.xticks(rotation="horizontal")
plt.show()
```

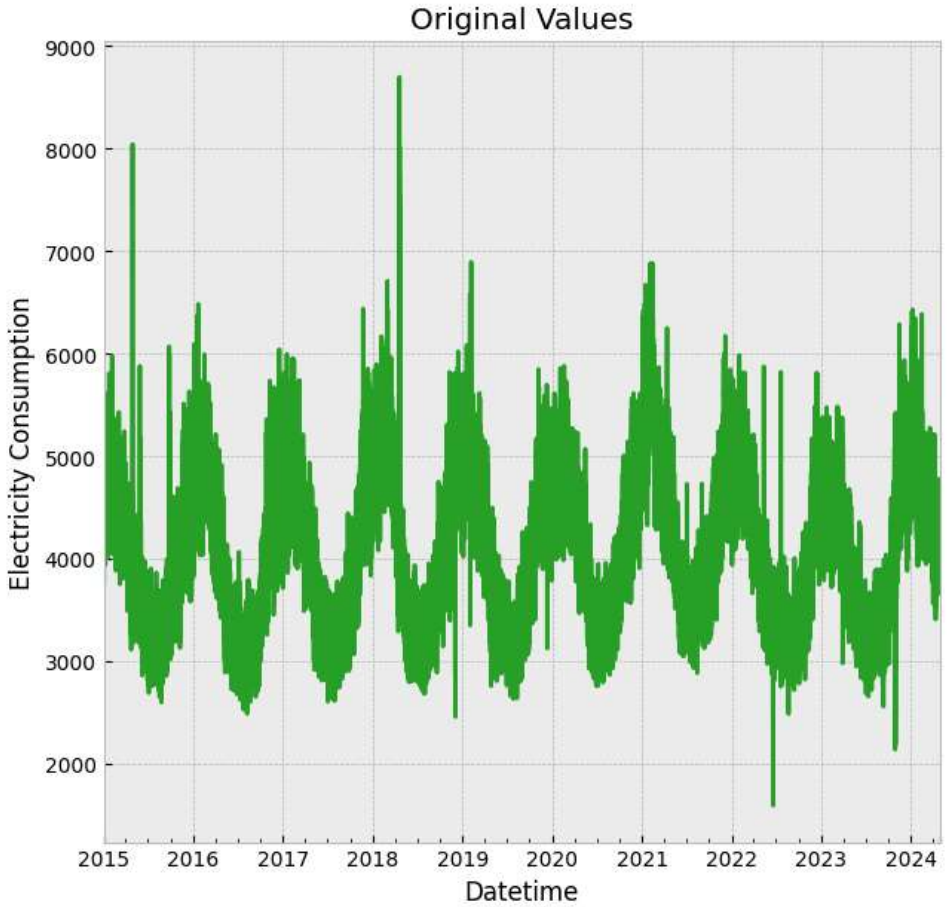
```
RMSE: nan
```



ZONE 2

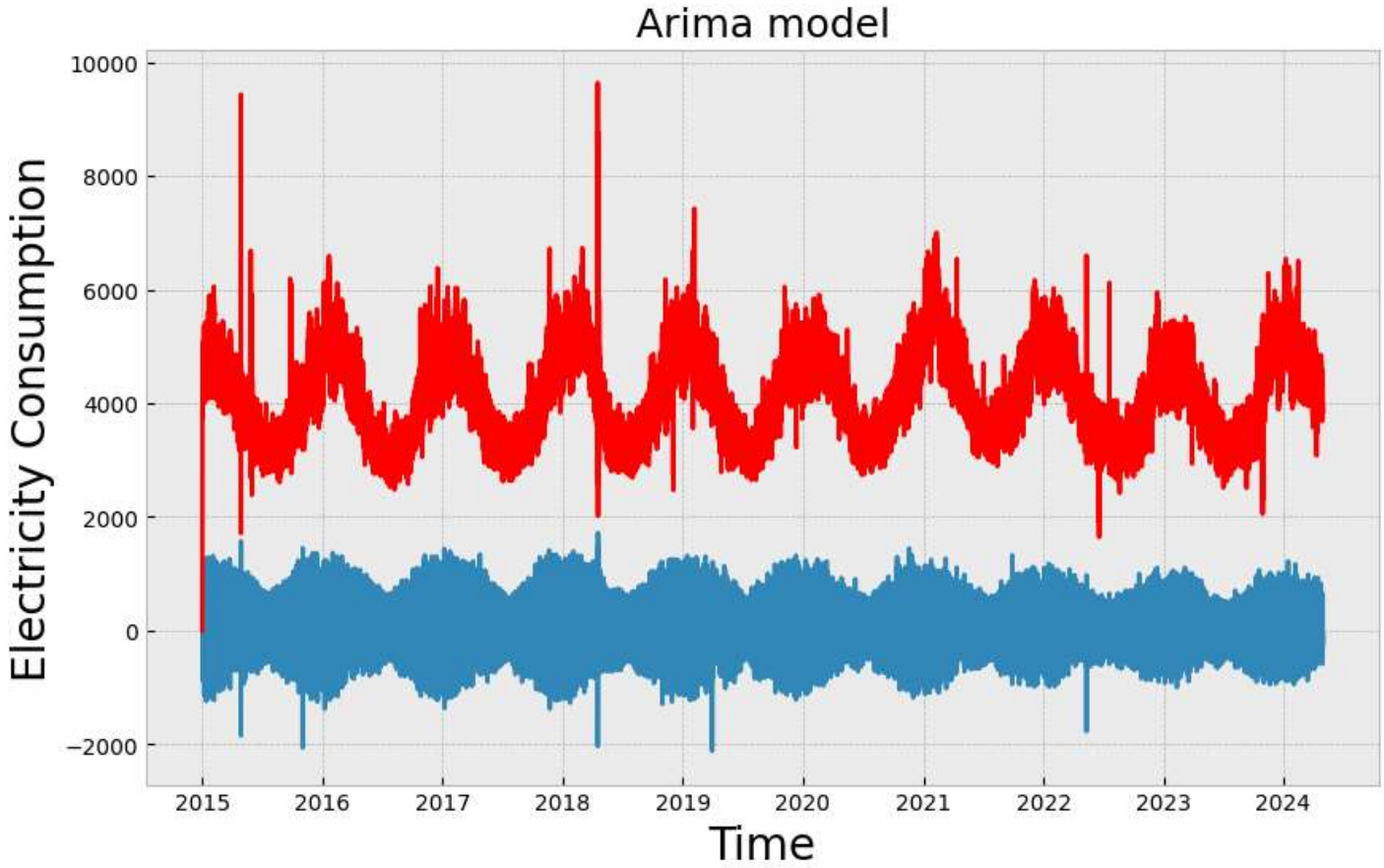
```
rolling_mean2 = loadno2['load_no2'].rolling(window = 12).mean()
consumption_rolled_detrended2 = loadno2['load_no2'] - rolling_mean2

ax1 = plt.subplot(121)
consumption_rolled_detrended2.plot(figsize=(16,7),color="tab:red", title="Differenced With Rolling Mean over 12 month", ax=ax1);
plt.ylabel("Electricity Consumption")
ax2 = plt.subplot(122)
loadno2['load_no2'].plot(figsize=(16,7), color="tab:green", title="Original Values", ax=ax2);
plt.ylabel("Electricity Consumption")
plt.show()
```





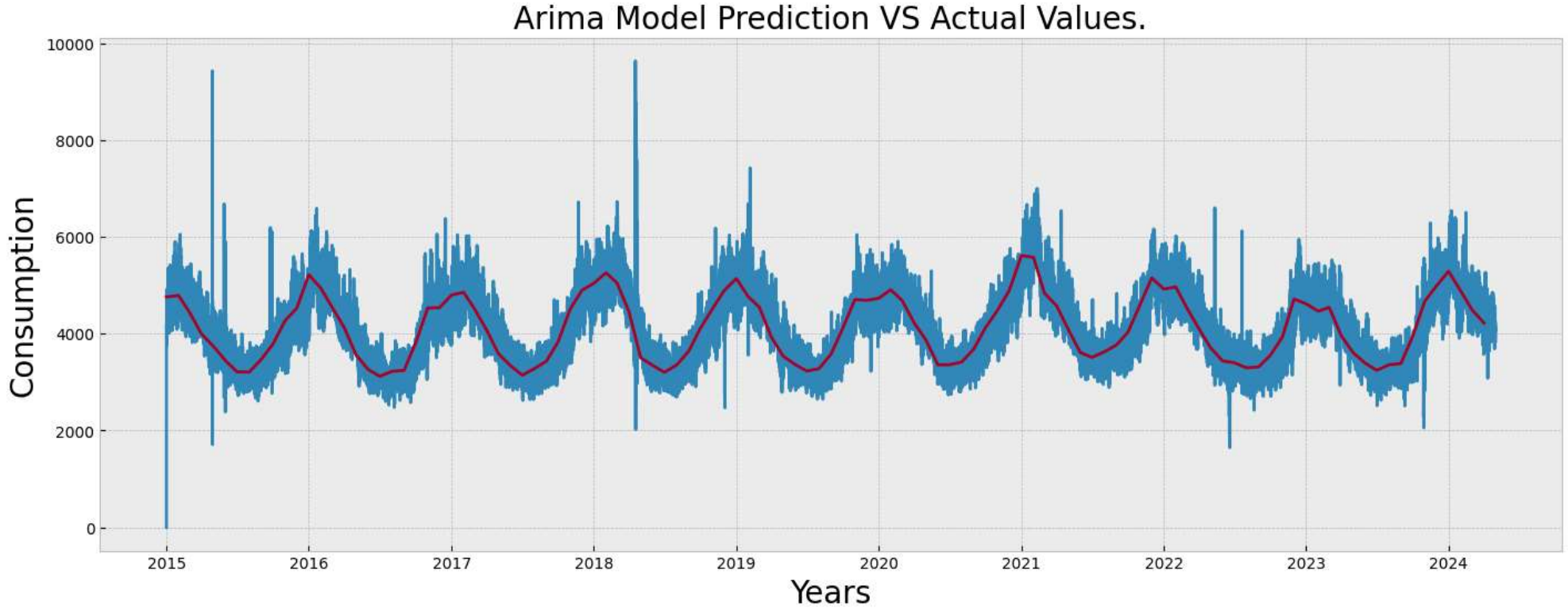

```
# Moving Average Model.
#
figure, ax = plt.subplots(figsize=(10,6))
model = ARIMA(loadno2['load_no2'], order=(3,1,2))
results_ARIMA2 = model.fit()
plt.plot(consumption_rolled_detrended)
plt.plot(results_ARIMA2.fittedvalues, color="red")
plt.xlabel("Time", fontsize="20")
plt.ylabel("Electricity Consumption", fontsize="20")
plt.title("Arima model", fontsize="18")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters found. Using z
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using zeros as star
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```



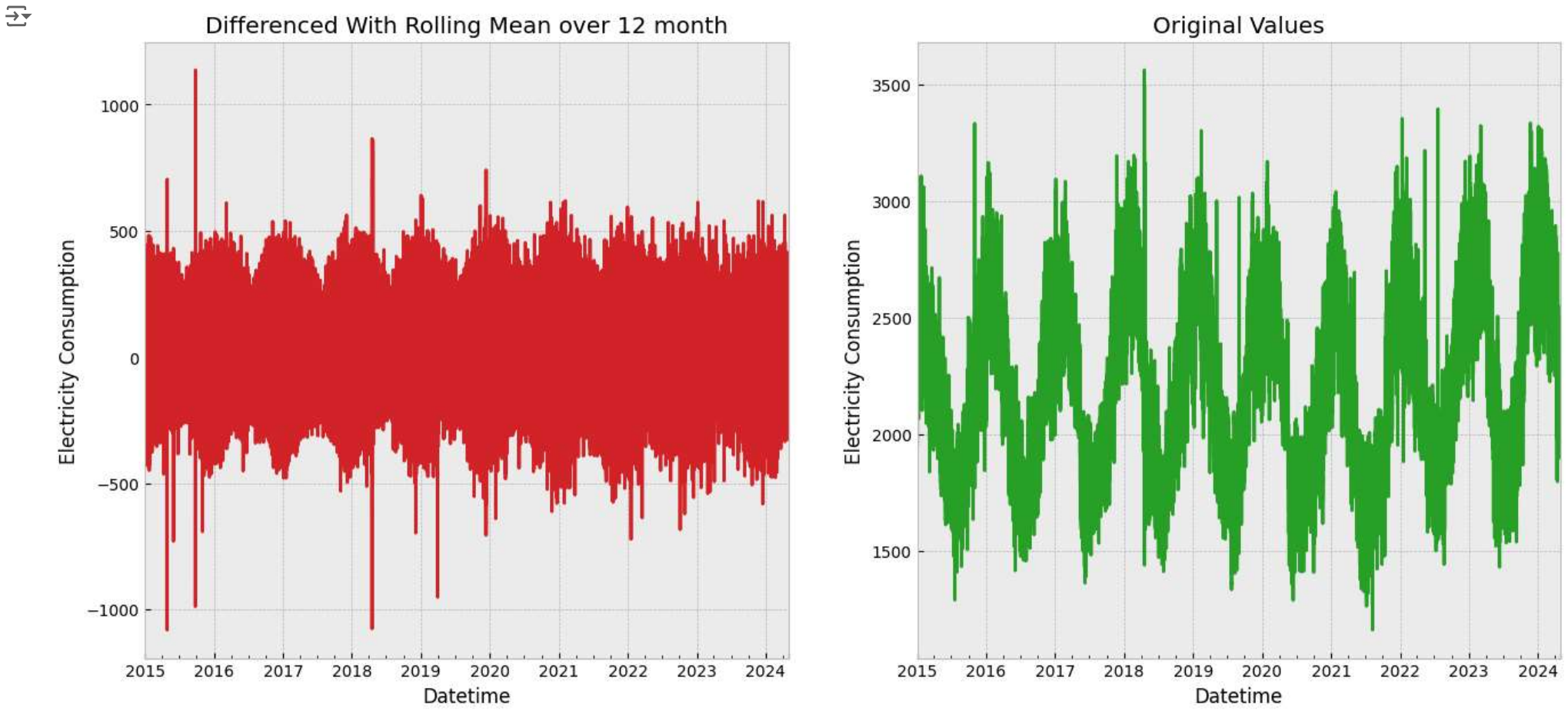
```
pred_ARIMA2 = pd.Series(results_ARIMA2.fittedvalues, copy=True)

figure, ax = plt.subplots(figsize=(17,6))
plt.plot(pred_ARIMA2)
plt.plot(consumption2)
print(f"RMSE: {np.sqrt(sum((pred_ARIMA2-consumption2)**2)/len(consumption2))}")
plt.xlabel("Years", fontsize="20")
plt.ylabel("Consumption", fontsize="20")
plt.title("Arima Model Prediction VS Actual Values.", fontsize="20")
plt.xticks(rotation="horizontal")
plt.show()
```

```
RMSE: nan
```



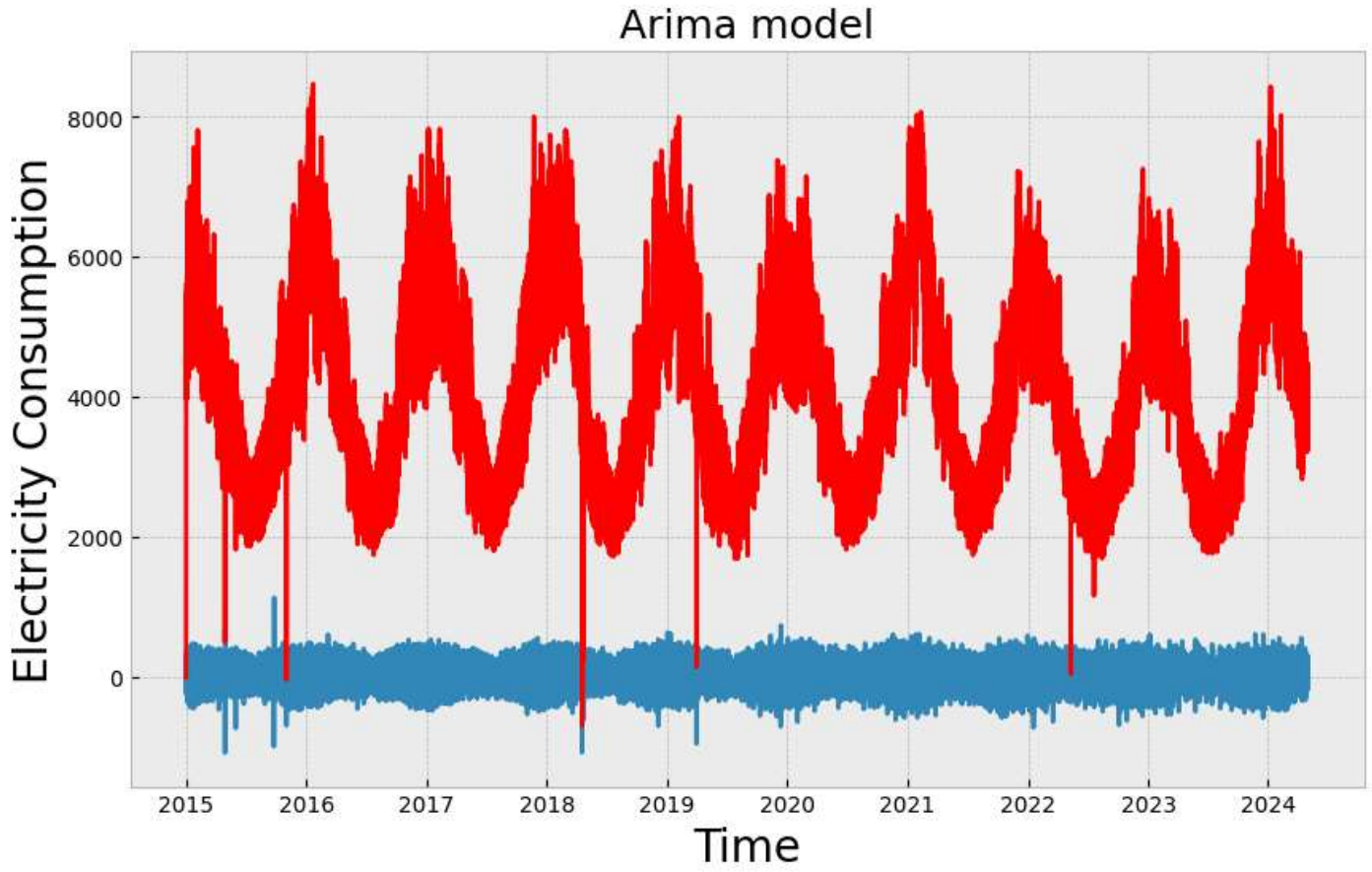
ZONE 3

```
rolling_mean3 = loadno3['load_no3'].rolling(window = 12).mean()
consumption_rolled_detrended3 = loadno3['load_no3'] - rolling_mean3

ax1 = plt.subplot(121)
consumption_rolled_detrended3.plot(figsize=(16,7),color="tab:red", title="Differenced With Rolling Mean over 12 month", ax=ax1);
plt.ylabel("Electricity Consumption")
ax2 = plt.subplot(122)
loadno4['load_no4'].plot(figsize=(16,7), color="tab:green", title="Original Values", ax=ax2);
plt.ylabel("Electricity Consumption")
plt.show()
```



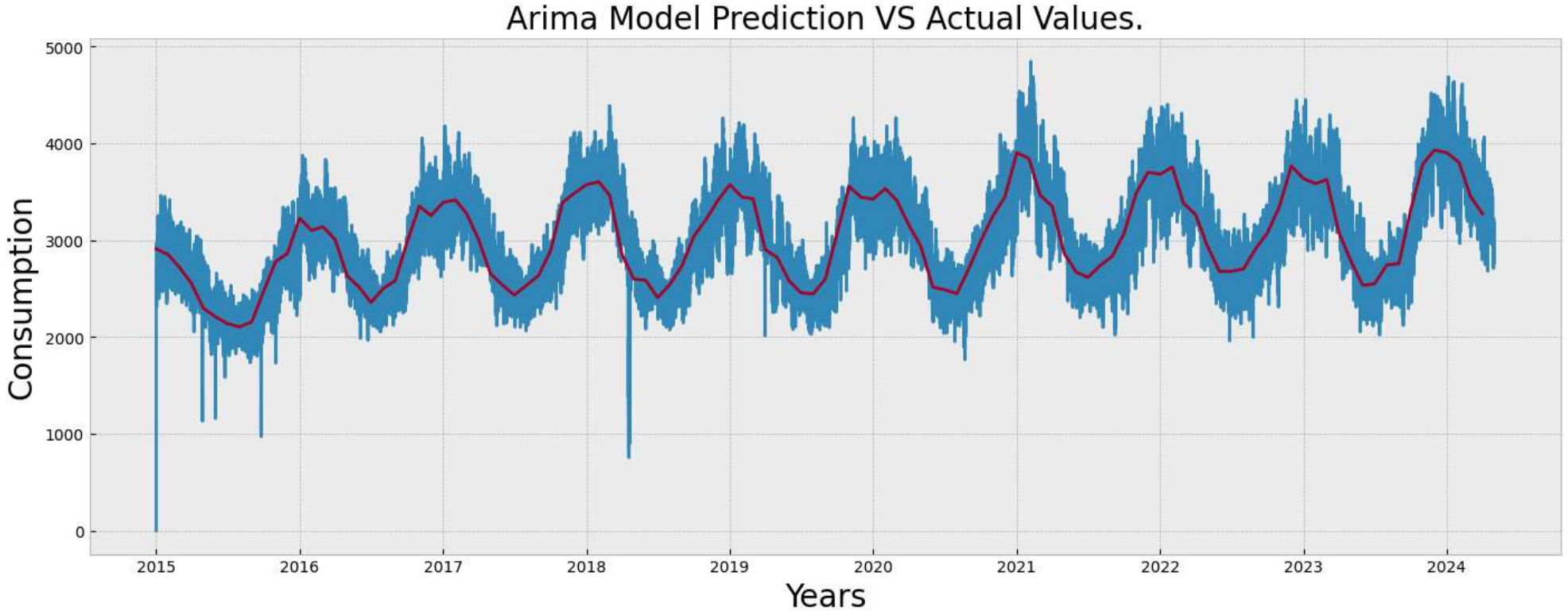
```
# Moving Average Model.
#
figure, ax = plt.subplots(figsize=(10,6))
model = ARIMA(loadno3['load_no3'], order=(3,1,2))
results_ARIMA3 = model.fit()
plt.plot(consumption_rolled_detrended3)
plt.plot(results_ARIMA.fittedvalues, color="red")
plt.xlabel("Time", fontsize="20")
plt.ylabel("Electricity Consumption", fontsize="20")
plt.title("Arima model", fontsize="18")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters found. Using z
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using zeros as star
  warn('Non-invertible starting MA parameters found.'
```



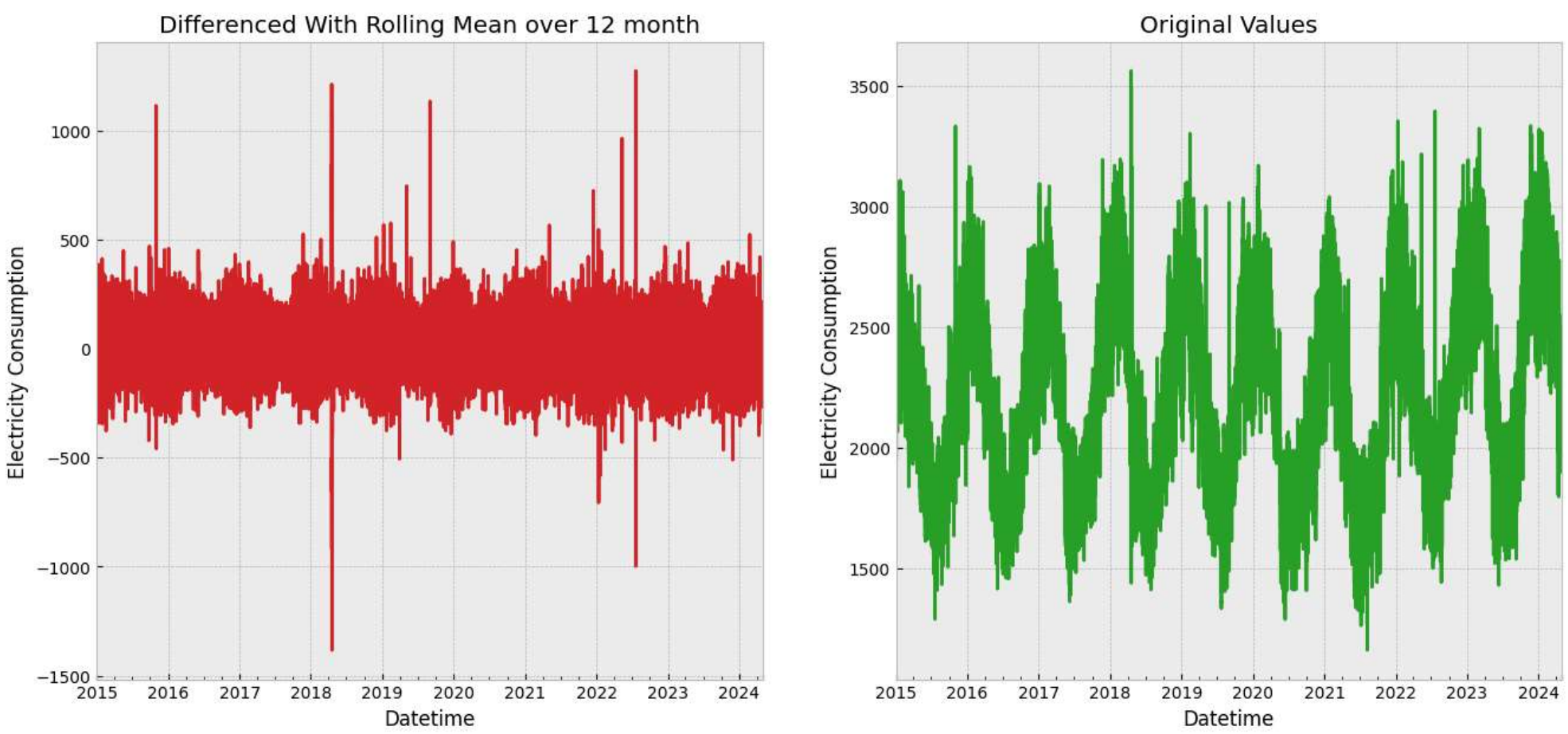
```
pred_ARIMA3 = pd.Series(results_ARIMA3.fittedvalues, copy=True)

figure, ax = plt.subplots(figsize=(17,6))
plt.plot(pred_ARIMA3)
plt.plot(consumption3)
print(f"RMSE: {np.sqrt(sum((pred_ARIMA3-consumption3)**2)/len(consumption3))}")
plt.xlabel("Years", fontsize="20")
plt.ylabel("Consumption", fontsize="20")
plt.title("Arima Model Prediction VS Actual Values.", fontsize="20")
plt.xticks(rotation="horizontal")
plt.show()
```

```
RMSE: nan
```



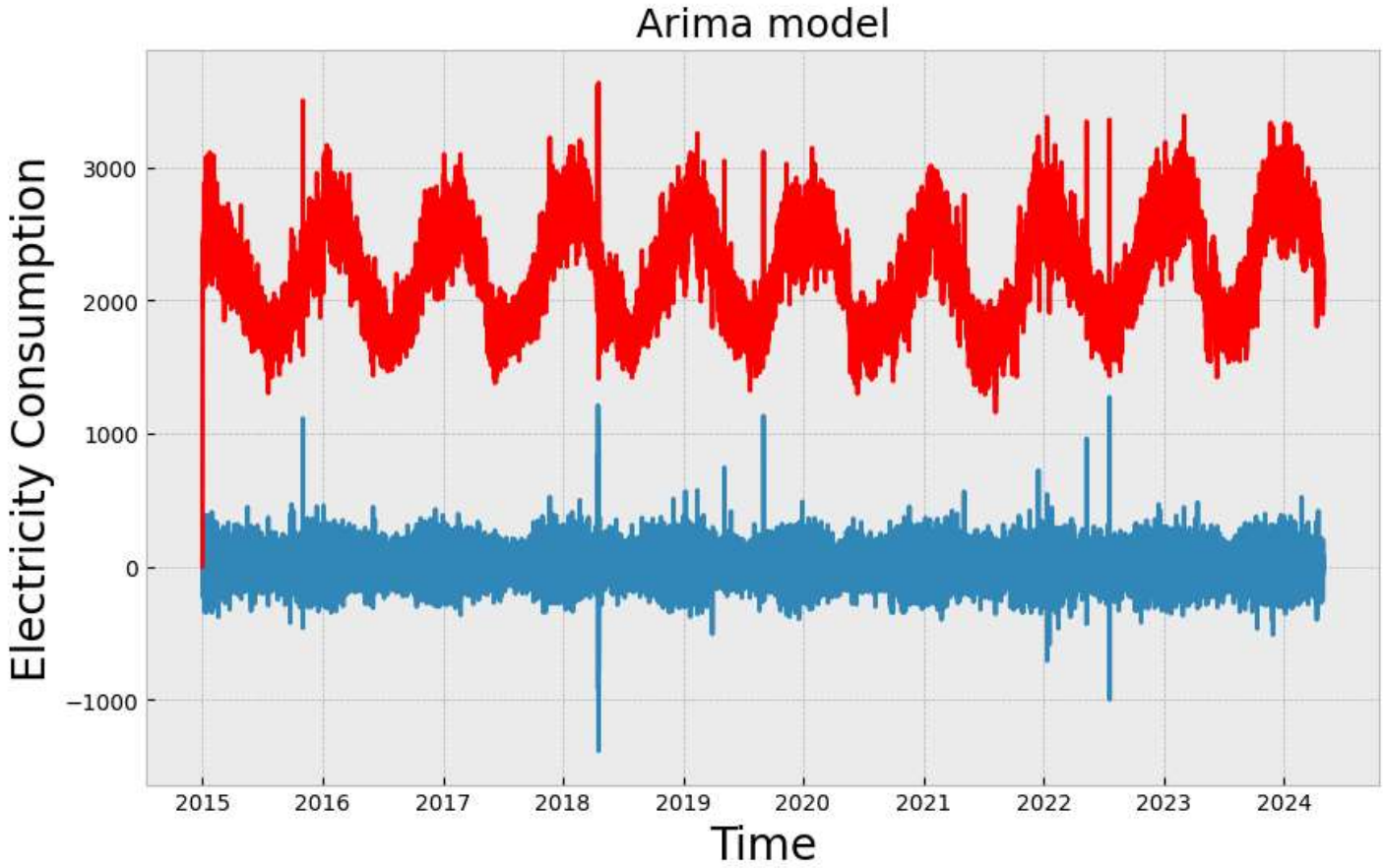
ZONE 4

```
rolling_mean4 = loadno4['load_no4'].rolling(window = 12).mean()
consumption_rolled_detrended4 = loadno4['load_no4'] - rolling_mean4

ax1 = plt.subplot(121)
consumption_rolled_detrended4.plot(figsize=(16,7),color="tab:red", title="Differenced With Rolling Mean over 12 month", ax=ax1);
plt.ylabel("Electricity Consumption")
ax2 = plt.subplot(122)
loadno4['load_no4'].plot(figsize=(16,7), color="tab:green", title="Original Values", ax=ax2);
plt.ylabel("Electricity Consumption")
plt.show()
```



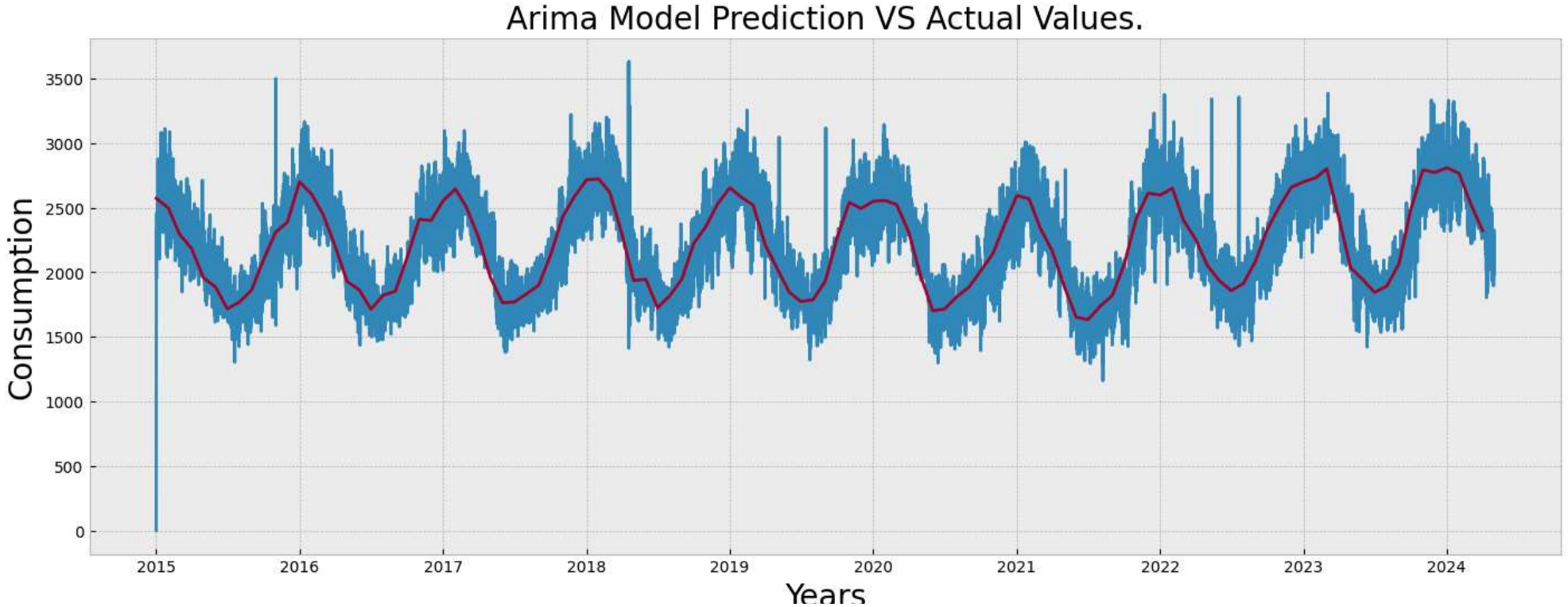
```
# Moving Average Model.
#
figure, ax = plt.subplots(figsize=(10,6))
model = ARIMA(loadno4['load_no4'], order=(3,1,2))
results_ARIMA4 = model.fit()
plt.plot(consumption_rolled_detrended4)
plt.plot(results_ARIMA4.fittedvalues, color="red")
plt.xlabel("Time", fontsize="20")
plt.ylabel("Electricity Consumption", fontsize="20")
plt.title("Arima model", fontsize="18")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provided, so inferred frequency H will
  self._init_dates(dates, freq)
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary starting autoregressive parameters found. Using z
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible starting MA parameters found. Using zeros as star
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```



```
pred_ARIMA4 = pd.Series(results_ARIMA4.fittedvalues, copy=True)

figure, ax = plt.subplots(figsize=(17,6))
plt.plot(pred_ARIMA4)
plt.plot(consumption4)
print(f"RMSE: {np.sqrt(sum((pred_ARIMA4-consumption4)**2)/len(consumption4))}")
plt.xlabel("Years", fontsize="20")
plt.ylabel("Consumption", fontsize="20")
plt.title("Arima Model Prediction VS Actual Values.", fontsize="20")
plt.xticks(rotation="horizontal")
plt.show()
```

```
RMSE: nan
```



ZONE 5