Exploratory Data Analysis & Feature Definition

Hourly-level household power-consumption data (UCI dataset).
Goal: extract insights that directly motivate useful predictive features.

1. Project Setup

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from IPython.display import Markdown, display
import seaborn as sns
import sys
sys.path.append('../')
from src.data_preparation import download_and_extract_data
from statsmodels.tsa.seasonal import seasonal_decompose

plt.style.use('seaborn-v0_8-whitegrid')
pd.set_option('display.max_columns', 50)
```

2. Raw Data Inspection and handlling nulls

```
raw_path = r'C:\Users\AliRashaideh\OneDrive - Seagulls\Desktop\energy_forecasting_project\data\raw\household_power_consumption.csv'
DATA_URL = "https://archive.ics.uci.edu/ml/machine-learning-databases/00235/household_power_consumption.zip"
RAW_DIR = './data/raw'
raw_file_path = download_and_extract_data(DATA_URL, RAW_DIR)
```

```
Downloading dataset from https://archive.ics.uci.edu/ml/machine-learning-databases/00235/household_power_consumption.zip...
Downloading: 19.7MB [00:04, 4.65MB/s]
Extracting dataset...
```

```
print("Loading data...")
df_without_na = pd.read_csv(raw_file_path, sep=';')
print("Loading data...")
df = pd.read_csv(raw_file_path, sep=';', na_values=['?', 'nan'], parse_dates={'datetime': ['Date', 'Time']})
df.set_index('datetime', inplace=True)
```

```
Loading data...
C:\Users\AliRashaideh\AppData\Local\Temp\ipykernel_18184\2964537705.py:2: DtypeWarning: Columns (2,3,4,5,6,7) have mixed types. Specify
  df_without_na = pd.read_csv(raw_file_path, sep=';')
C:\Users\AliRashaideh\AppData\Local\Temp\ipykernel_18184\2964537705.py:4: FutureWarning: Support for nested sequences for 'parse_dates'
  df = pd.read_csv(raw_file_path, sep=';', na_values=['?', 'nan'], parse_dates={'datetime': ['Date', 'Time']})
Loading data...
C:\Users\AliRashaideh\AppData\Local\Temp\ipykernel_18184\2964537705.py:4: UserWarning: Parsing dates in %d/%m/%Y %H:%M:%S format when da
  df = pd.read_csv(raw_file_path, sep=';', na_values=['?', 'nan'], parse_dates={'datetime': ['Date', 'Time']})
```

```
print("Data without NA:")
print(df_without_na.info())
print(df_without_na.describe().T)
print(df_without_na.isnull().sum())
```

```
Data without NA:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2075259 entries, 0 to 2075258
Data columns (total 9 columns):
 #   Column                 Dtype
---  ------                 -----
 0   Date                   object
 1   Time                   object
 2   Global_active_power    object
 3   Global_reactive_power  object
 4   Voltage                object
 5   Global_intensity       object
 6   Sub_metering_1         object
 7   Sub_metering_2         object
 8   Sub_metering_3         float64
dtypes: float64(1), object(8)
memory usage: 142.5+ MB
```

```
None
                 count      mean       std  min  25%  50%   75%   max
Sub_metering_3  2049280.0  6.458447  8.437154  0.0  0.0  1.0  17.0  31.0
Date                        0
Time                        0
Global_active_power         0
Global_reactive_power       0
Voltage                     0
Global_intensity            0
Sub_metering_1              0
Sub_metering_2              0
Sub_metering_3          25979
dtype: int64
```

```
print("Data with NA detection:")
print(df.info())
print(df.describe().T)
print(df.isnull().sum())
```

```
Data with NA detection:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2075259 entries, 2006-12-16 17:24:00 to 2010-11-26 21:02:00
Data columns (total 7 columns):
 #   Column                 Dtype
---  ------                 -----
 0   Global_active_power    float64
 1   Global_reactive_power  float64
 2   Voltage                float64
 3   Global_intensity       float64
 4   Sub_metering_1         float64
 5   Sub_metering_2         float64
 6   Sub_metering_3         float64
dtypes: float64(7)
memory usage: 126.7 MB
None
                           count        mean       std      min      25%  \
Global_active_power    2049280.0    1.091615  1.057294    0.076    0.308
Global_reactive_power  2049280.0    0.123714  0.112722    0.000    0.048
Voltage                2049280.0  240.839858  3.239987  223.200  238.990
Global_intensity       2049280.0    4.627759  4.444396    0.200    1.400
Sub_metering_1         2049280.0    1.121923  6.153031    0.000    0.000
Sub_metering_2         2049280.0    1.298520  5.822026    0.000    0.000
Sub_metering_3         2049280.0    6.458447  8.437154    0.000    0.000

                           50%      75%      max
Global_active_power      0.602    1.528   11.122
Global_reactive_power    0.100    0.194    1.390
Voltage                241.010  242.890  254.150
Global_intensity         2.600    6.400   48.400
Sub_metering_1           0.000    0.000   88.000
Sub_metering_2           0.000    1.000   80.000
Sub_metering_3           1.000   17.000   31.000
Global_active_power      25979
Global_reactive_power    25979
Voltage                  25979
Global_intensity         25979
Sub_metering_1           25979
Sub_metering_2           25979
Sub_metering_3           25979
dtype: int64
```

```
df.fillna(method='ffill', inplace=True)
df.drop_duplicates(inplace=True)
print("\nMissing values after imputation:")
print(df.isnull().sum())
```

```
C:\Users\AliRashaideh\AppData\Local\Temp\ipykernel_18184\1624033518.py:1: FutureWarning: DataFrame.fillna with 'method' is deprecated ar
  df.fillna(method='ffill', inplace=True)

Missing values after imputation:
Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
Sub_metering_1           0
Sub_metering_2           0
Sub_metering_3           0
dtype: int64
```

```
df.shape
```

```
(1906698, 7)
```

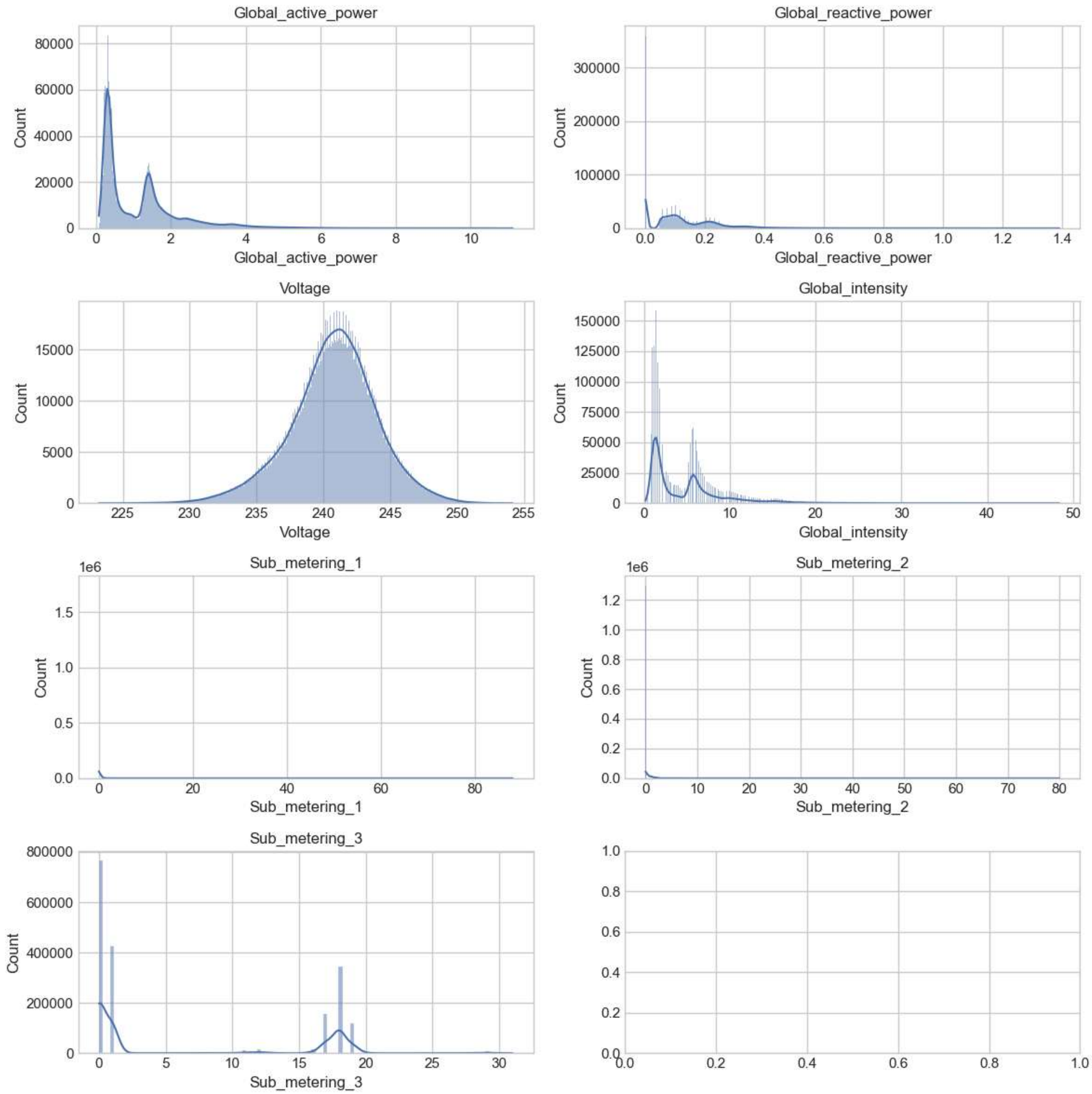
there is differnce when we dropped dublicate as we can see:

data shape before: (2075259, 7)

data shape after: (1906698, 7)

EDA

```
numeric_cols = df.select_dtypes(include=np.number).columns
rows = (len(numeric_cols) + 1) // 2
fig, axes = plt.subplots(rows, 2, figsize=(12, 3 * rows))
axes = axes.flatten()
for i, col in enumerate(numeric_cols):
    sns.histplot(df[col].dropna(), kde=True, ax=axes[i])
    axes[i].set_title(col)
plt.tight_layout()
plt.show()
```



1. histograms Active, reactive power & intensity Most readings are small; a few hours shoot up to very high values. Those tall spikes at the far right are the “outliers.”

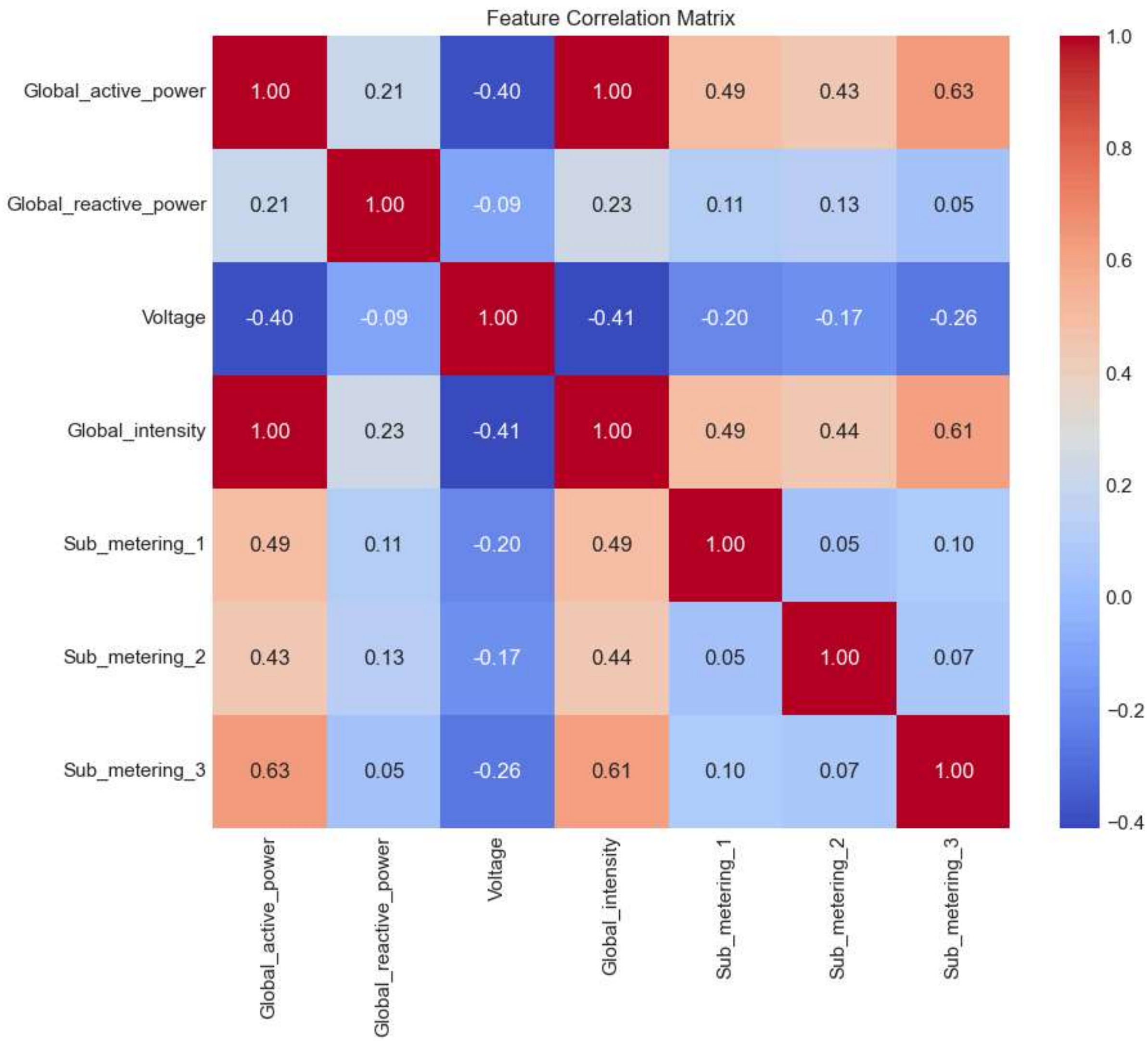
Voltage Looks like a neat bell curve centred around ~241 V. No obvious outliers.

Sub-metering 1 & 2 Almost always zero; they only spike when the specific appliance is on.

Sub-metering 3 Has three clear levels: off, medium (~~15 Wh) and high (~~30 Wh). The rare points above 30 Wh are the only extreme values worth flagging.

```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='coolwarm')
```

```
plt.title('Feature Correlation Matrix')
plt.show()
```



2. Correlation heat-map

Active power ↔ Intensity: basically the same thing (correlation ≈ 1).

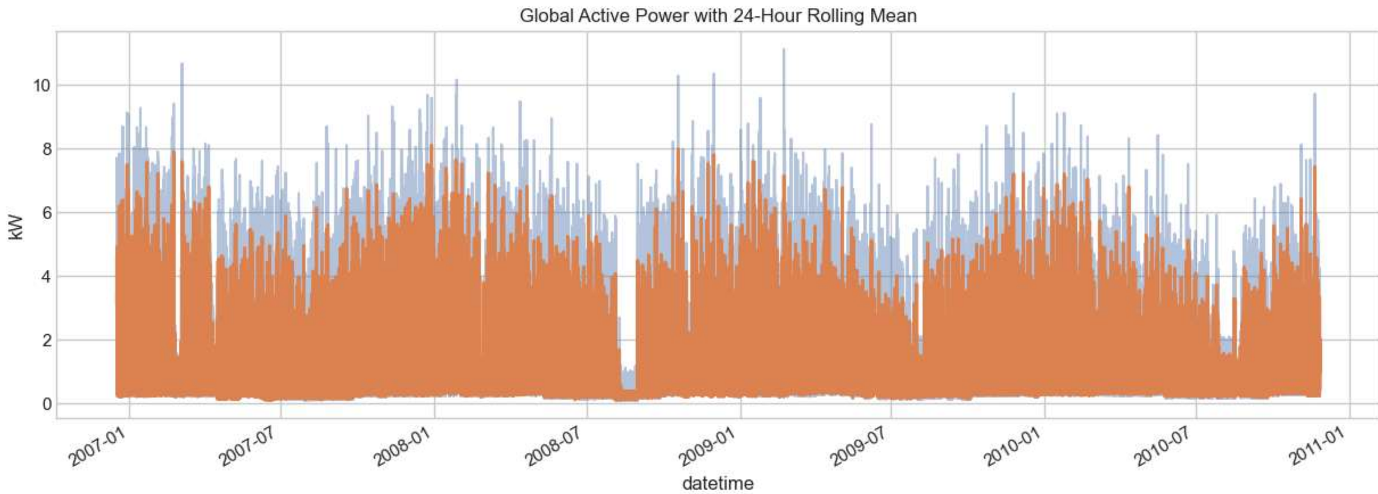
Active power ↔ Sub-metering 3: strong link—channel 3 drives big loads.

Voltage ↔ Load: light negative link—voltage dips slightly when load rises.

Sub-metering 1 & 2: almost independent of total load.

Sub-metering 3 and voltage are useful extra predictors; sub-metering 1/2 add little.

```
plt.figure(figsize=(15, 5))
df['Global_active_power'].plot(alpha=0.4)
df['Global_active_power'].rolling(24).mean().plot(linewidth=2)
plt.title('Global Active Power with 24-Hour Rolling Mean')
plt.ylabel('kW')
plt.show()
```



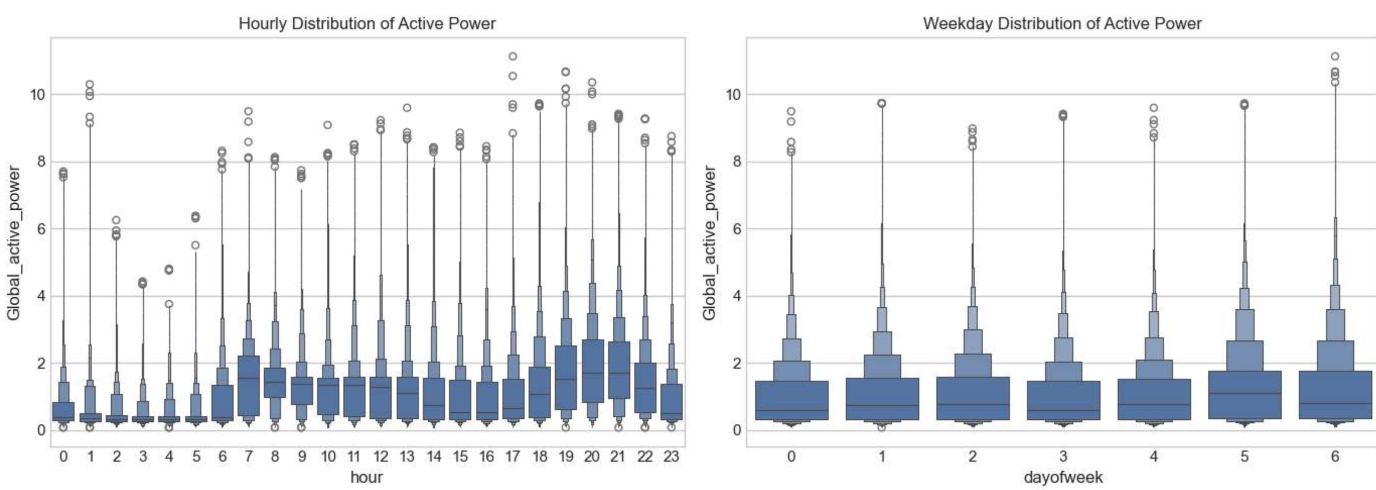
3. Active power with 24 h rolling mean

Clear winter peaks and summer dips → yearly seasonality.

A long flat strip around mid-2008 is missing data.

Isolated spikes above the orange 24-h average confirm the outliers seen earlier.

```
df['hour'] = df.index.hour
df['dayofweek'] = df.index.dayofweek
fig, axes = plt.subplots(1, 2, figsize=(14, 5))
sns.boxenplot(x='hour', y='Global_active_power', data=df, ax=axes[0])
axes[0].set_title('Hourly Distribution of Active Power')
sns.boxenplot(x='dayofweek', y='Global_active_power', data=df, ax=axes[1])
axes[1].set_title('Weekday Distribution of Active Power')
plt.tight_layout()
plt.show()
```



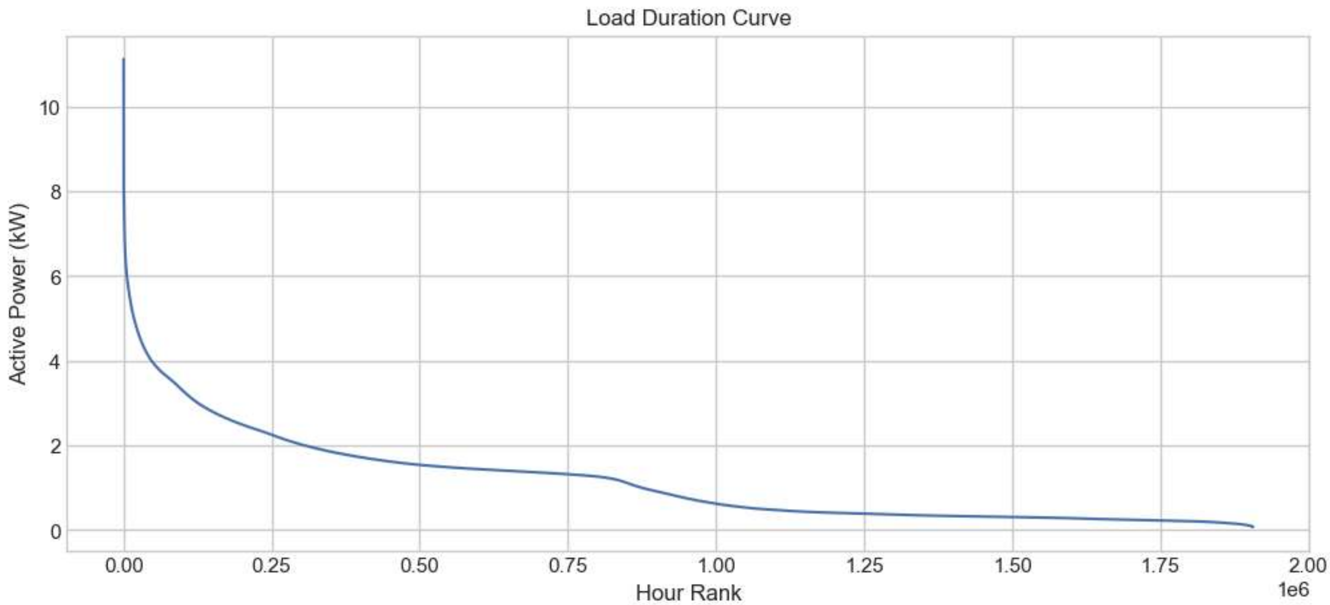
4. Hour-of-day & Day-of-week box-plots

Hourly: load climbs from sunrise, peaks at 17-22 h, drops overnight.

Weekdays vs. weekends: weekends are a touch higher and more spread out.

Dots above whiskers are the same high-load outliers.

```
sorted_load = df['Global_active_power'].sort_values(ascending=False).reset_index(drop=True)
plt.figure(figsize=(12, 5))
plt.plot(sorted_load.values)
plt.title('Load Duration Curve')
plt.xlabel('Hour Rank')
plt.ylabel('Active Power (kW)')
plt.show()
```



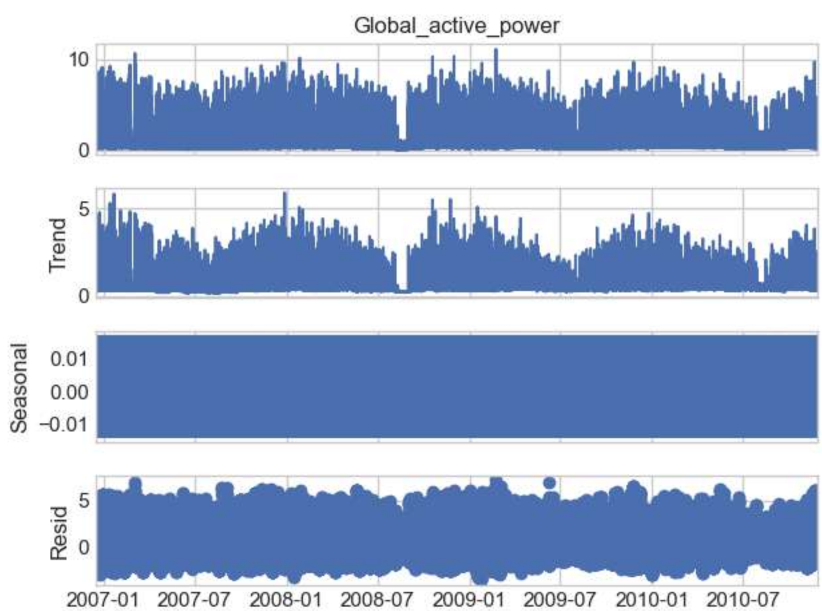
5. Load-duration curve

95 % of the time the house uses < 2 kW.

The top 1 % of hours jump to > 10 kW—rare but real heavy-use periods.

No sudden cliff, so extreme points look genuine.

```
res = seasonal_decompose(df['Global_active_power'], model='additive', period=168)
res.plot()
plt.show()
```



6. Weekly seasonal decomposition (168 h period)

Trend: echoes the yearly up-and-down pattern.

Seasonal (weekly) part: almost flat—weekly cycle is weak.

Residuals: bursts line up with the high-load spikes.

Conclusions

Outliers exist only in power/intensity variables—real peaks rather than sensor errors.

Mid-2008 gap must be filled or removed.

Useful time features: hour, weekday/weekend, month, plus 24-h & 168-h rolling stats.

Key drivers: sub-metering 3 and slight voltage dips during high load.

Feature shortlist:

Hour, day-of-week, month, weekend flag

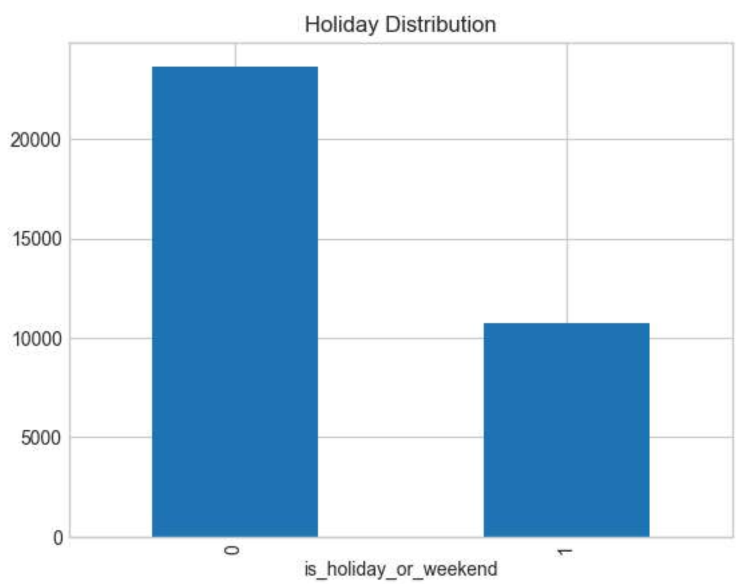
Lag-24, lag-48, lag-168 for active power & sub-metering 3

Rolling mean & std (24 h, 168 h)

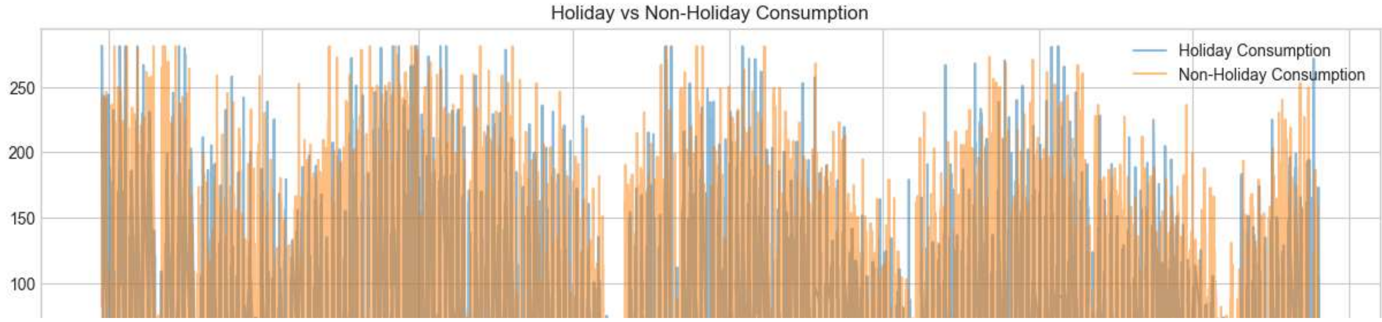
Holiday

```
import holidays
engineered_data= pd.read_csv(r'../data/processed/hourly_featured.csv', parse_dates=['datetime'])
engineered_data.set_index('datetime', inplace=True)


print("Engineered Data:")

engineered_data['is_holiday_or_weekend'].value_counts().plot(kind='bar', title='Holiday Distribution')
#display data where is holiday
```

```
Engineered Data:
<Axes: title={'center': 'Holiday Distribution'}, xlabel='is_holiday_or_weekend'>
```




```
holiday_data = engineered_data[engineered_data['is_holiday_or_weekend'] == 1]
non_holiday_data = engineered_data[engineered_data['is_holiday_or_weekend'] == 0]
plt.figure(figsize=(15, 5))
holiday_data['Global_active_power'].plot(label='Holiday Consumption', alpha=0.5)
non_holiday_data['Global_active_power'].plot(label='Non-Holiday Consumption', alpha=0.5)
plt.title('Holiday vs Non-Holiday Consumption')
plt.legend()
plt.show()
```


Holiday vs Non-Holiday Consumption
Holiday Consumption
Non-Holiday Consumption
250
200
150
100