Analyzing the energy consumption data and creating visualizations

Data Visualization:

Use Matplotlib and Seaborn to create histograms and box plots. Here's an example code for creating these visualizations:

Exploratory Analysis

First, we will need to load the modules we will be using as well as our dataset:

```
# Load Modulesimport numpy as np
import pandas as pdimport matplotlib
import matplotlib.pyplot as plt
import seaborn as sns# Set Plotting Styles
plt.style.use('ggplot')# Load Data
duq_df = pd.read_csv('data/DUQ_hourly.csv', index_col=[0],
parse_dates=[0])# Sort Data
duq_df.sort_index(inplace=True)
```

When our data was originally stored in the CSV file, timestamps were converted to strings. We need to parse them back into datetime objects for our analysis and set them as the dataset index. This can be done automatically using the parse_dates and index_col parameters for the read_csv function included with Pandas. Once we have established a datetime index, we sort the values to make sure they are presented in

chronological order. Skipping this step will reveal that the timestamps may have been sorted by their timestamp strings previously, as some timestamps are out of order.

A quick evaluation of the data indicates that it has 119,008 rows and one column (DUQ_MW). We will now go through the data to identify duplicate timestamps, impute missing values and prepare some basic visualizations.

Duplicate Values

Some datasets may include duplicate readings that share a common timestamp. The reason for this duplication should be investigated for each unique dataset, as reasons for data duplication can vary widely depending on the data collection methodology.

There are several approaches towards the handling of duplicate values. We have the option of discarding both or one of the duplicate values, or imputing the timestamp value using each of the available measurements. For our example, we will compute the mean energy consumption for each of our duplicate value pairs and use that value moving forward.

Once we have removed duplicate values, we manually set the frequency of the DatetimeIndex to hourly ('H'). Normally, the date parser would be able to determine the frequency of the DatetimeIndex automatically. The presence of duplicate DatetimeIndex values prevents this from happening though, so the frequency must be set manually following removal of

duplicate values. Setting the frequency now will help us avoid problems with plotting and calculations down the line.

```
# Identify Duplicate Indices
duplicate_index = duq_df[duq_df.index.duplicated()]
print(duq_df.loc[duplicate_index.index.values, :])# Replace
Duplicates with Mean Value
duq_df = duq_df.groupby('Datetime').agg(np.mean)# Set DatetimeIndex
Frequency
duq_df = duq_df.asfreq('H')
```

Missing Values

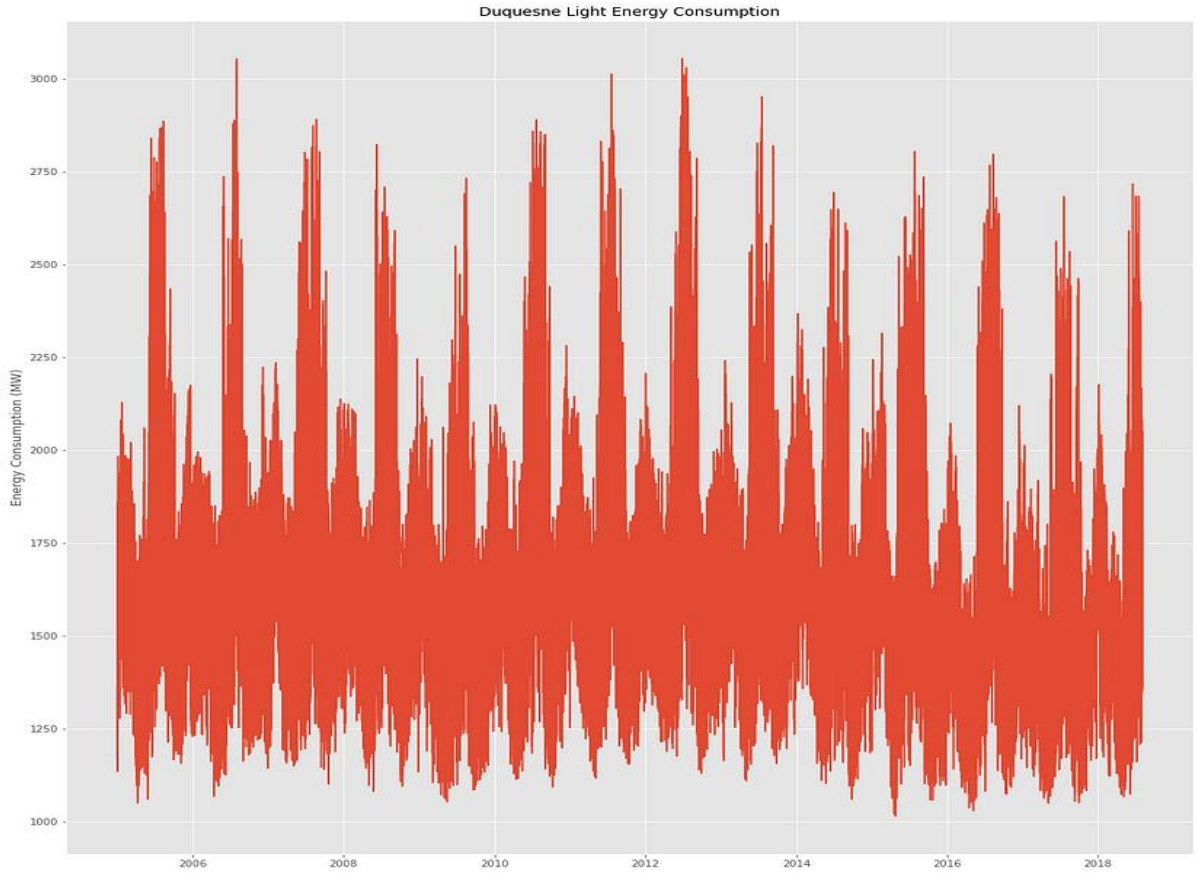
A quick search through the dataset indicates that there are 24 missing values across our date range. We will use mean interpolation to impute these missing values. We can do this using the interpolate() method on our dataframe object:

```
# Determine # of Missing Values
print('# of Missing DUQ_MW Values:
{}'.format(len(duq_df[duq_df['DUQ_MW'].isna()])))# Impute Missing
Values
duq_df['DUQ_MW'] =
duq_df['DUQ_MW'].interpolate(limit_area='inside', limit=None)
```

Visualizing Energy Consumption Data

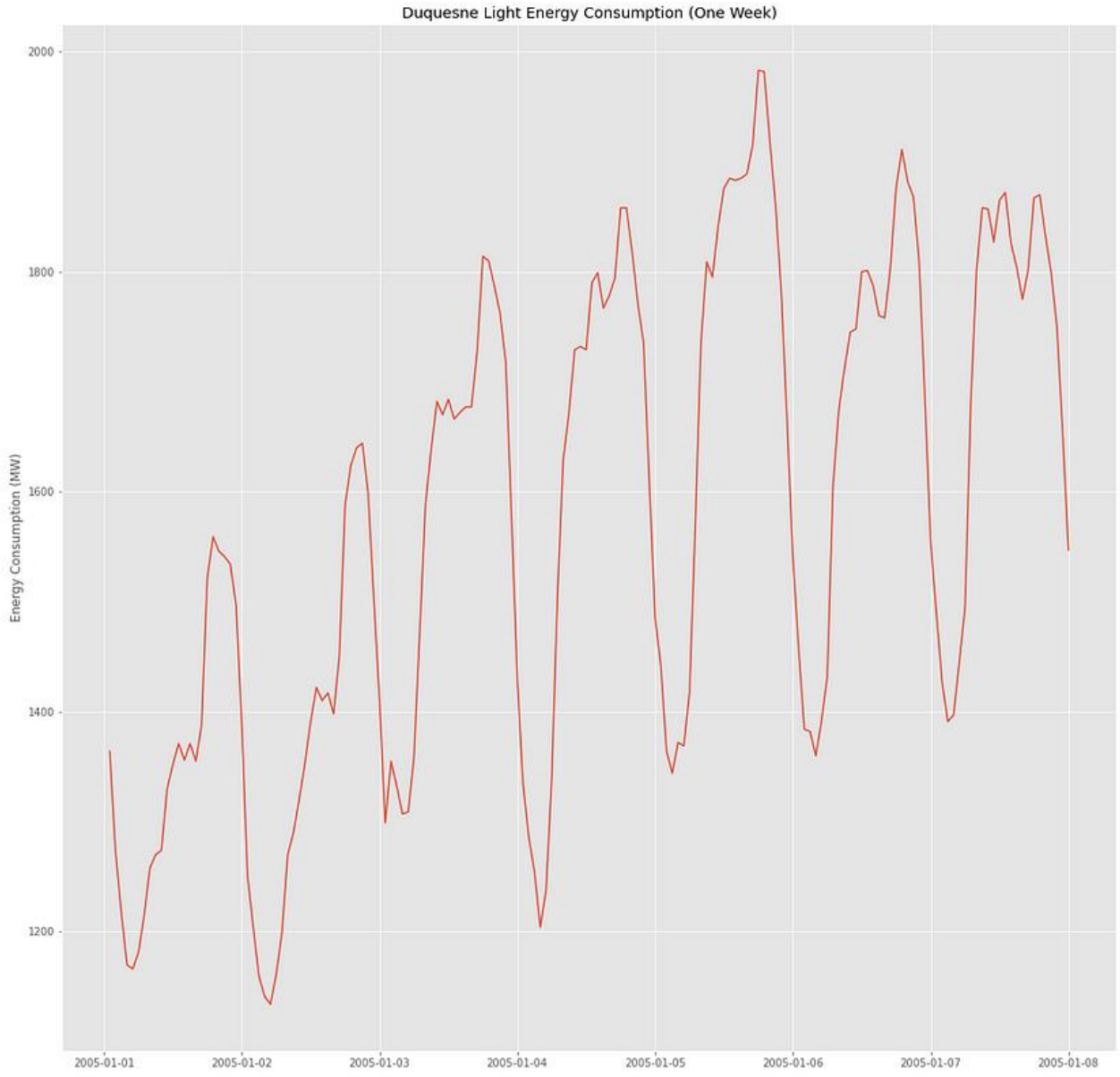
First, let's look at a simple time series plot of our data:

```
plt.plot(duq_df.index, duq_df['DUQ_MW'])
plt.title('Duquesne Light Energy Consumption')
plt.ylabel('Energy Consumption (MW)')
plt.show()
```



There is a clear annual seasonal pattern visible in the data, but the details of each individual year are obscured by the density of the plot. Let's look at a single week:

```
WEEK_END_INDEX = 7*24
plt.plot(duq_df.index[:WEEK_END_INDEX],
duq_df['DUQ_MW'][:week_end_index])
plt.title('Duquesne Light Energy Consumption (One Week)')
plt.ylabel('Energy Consumption (MW)')
plt.show()
```

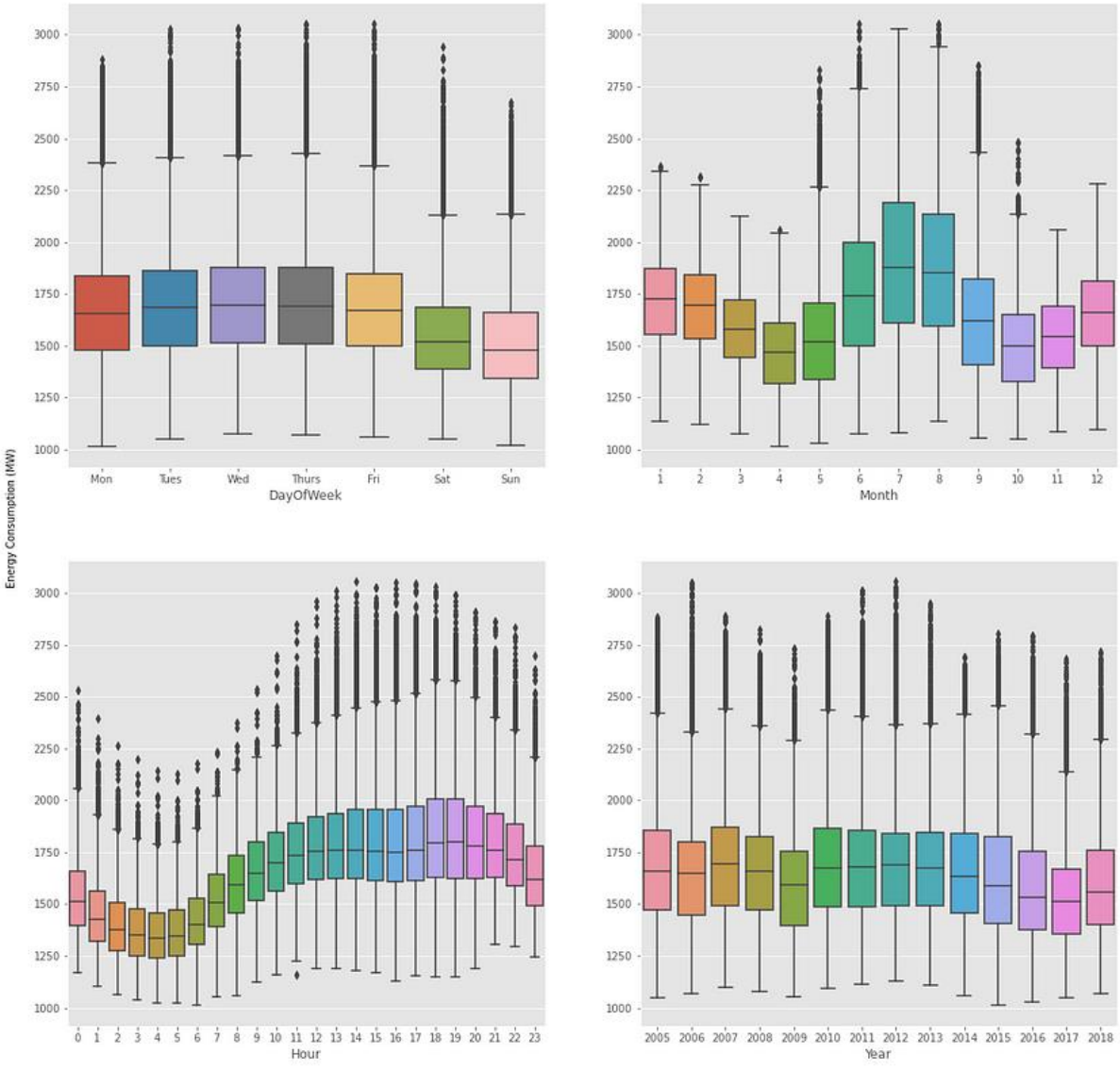


From this plot we can also see a clear daily seasonality, with a repeated pattern of hourly consumption rates occurring each day. We also observe an

upward trend within this week of data, but this trend is likely a part of the annual seasonality we saw previously.

Let's break our DatetimeIndex into separate features so we can look at some of these patterns a little more closely:

```
def create_features(df):
    df['Date'] = df.index
    df['Hour'] = df['Date'].dt.hour
    df['DayOfWeek'] = df['Date'].dt.dayofweek
    df['Quarter'] = df['Date'].dt.quarter
    df['Month'] = df['Date'].dt.month
    df['Year'] = df['Date'].dt.year
    df['DayOfYear'] = df['Date'].dt.dayofyear
    df['DayOfMonth'] = df['Date'].dt.day
    df['WeekOfYear'] = df['Date'].dt.weekofyear
    df['DayOfYearFloat'] = df['DayOfYear'] + df['Hour'] / 24
    df.drop('Date', axis=1, inplace=True)
    return dfduq_df = create_features(duq_df)fig, axes =
plt.subplots(2, 2, figsize=(16,16))# Day of Week
dow_labels = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun']
g = sns.boxplot(x=duq_df.DayOfWeek, y=duq_df.DUQ_MW, ax=axes[0][0])
g.set_xticklabels(dow_labels)
g.set_ylabel('')# Month of Year
g = sns.boxplot(x=duq_df.Month, y=duq_df.DUQ_MW, ax=axes[0][1])
g.set_ylabel('')# Hour of Day
g = sns.boxplot(x=duq_df.Hour, y=duq_df.DUQ_MW, ax=axes[1][0])
g.set_ylabel('')# Year
g = sns.boxplot(x=duq_df.Year, y=duq_df.DUQ_MW, ax=axes[1][1])
g.set_ylabel('')fig.text(0.08, 0.5, 'Energy Consumption (MW)',
va='center', rotation='vertical')
plt.show()
```

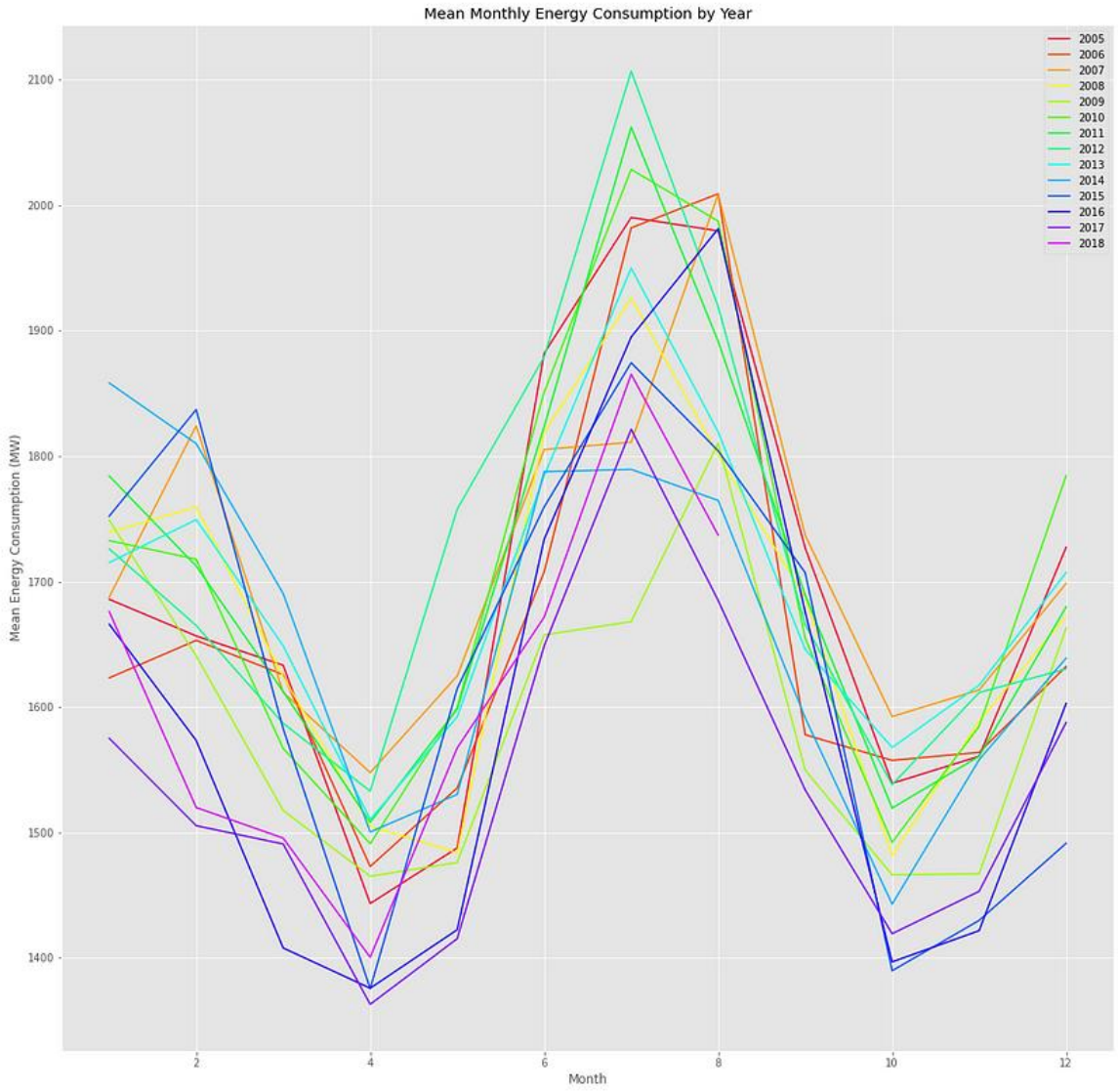


Here, we look at energy consumption in terms of the hour of day, day of the week, month of year, and year of our dataset range. We can see that energy consumption is slightly higher Monday-Friday than it is Saturday-Sunday,

which is to be expected since many businesses do not operate on weekends in the U.S. Energy consumption typically peaks around July/August, and there are several more outliers within the months preceding and following the peak period than are observed during the cooler months (November — April). Hourly energy consumption is at its lowest around 4 AM. After this time it gradually increases until it plateaus around 2 PM. It then peaks again around 7 PM before declining again until it reaches its early morning low point. Mean energy consumption rates and interquartile ranges are fairly consistent from year to year, although a slight downward trend is observable.

Next, we'll look at a seasonal plot to get a better idea how energy consumption patterns vary from year to year. We'll use monthly mean energy consumption instead of hourly data to make it easier to distinguish between years when all of the data are plotted together.

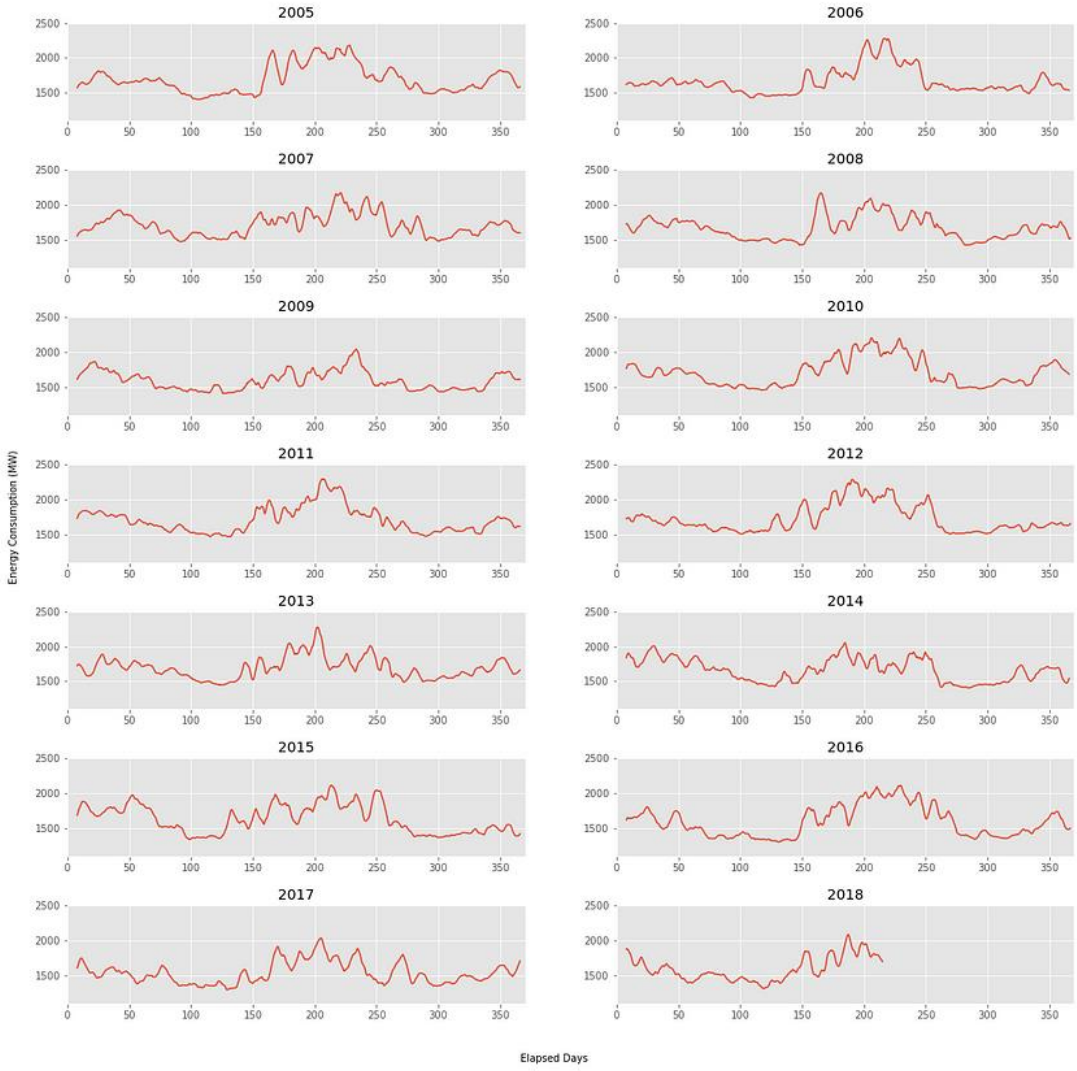
```
year_group = duq_df.groupby(['Year', 'Month']).mean().reset_index()
years = duq_df['Year'].unique()
NUM_COLORS = len(years)cm = plt.get_cmap('gist_rainbow')
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_prop_cycle(color=[cm(1.*i/NUM_COLORS) for i in
range(NUM_COLORS)])for i, y in enumerate(years):
    df = year_group[year_group['Year'] == y]
    #rolling_mean = df.DUQ_MW.rolling(window=7*24).mean()
    plt.plot(df['Month'], df['DUQ_MW'])
plt.title('Mean Monthly Energy Consumption by Year')
plt.xlabel('Month')
plt.ylabel('Mean Energy Consumption (MW)')
plt.legend(duq_df.Year.unique())
plt.show()
```



Now, let's plot each year's worth of data on a separate subplot so individual years can be evaluated more closely. We will use a 7-day moving average to smooth the data, making general trends easier to identify for comparison.

```
num_rows = 7
num_cols = 2
year_index = 0fig, axes = plt.subplots(num_rows, num_cols,
figsize=(18,18))
years = duq_df['Year'].unique()for i in range(num_rows):
    for j in range(num_cols):
        df = duq_df[duq_df['Year'] == years[year_index]]
        rolling_mean = df['DUQ_MW'].rolling(window=7*24).mean()
        axes[i][j].plot(df['DayOfYearFloat'], rolling_mean.values)
        axes[i][j].set_title(str(years[year_index]))
        axes[i][j].set_ylim(1100, 2500)
        axes[i][j].set_xlim(0,370)
        year_index += 1

fig.text(0.5, 0.08, 'Elapsed Days', ha='center')
fig.text(0.08, 0.5, 'Energy Consumption (MW)', va='center',
rotation='vertical')
fig.subplots_adjust(hspace=0.5)
plt.show()
```

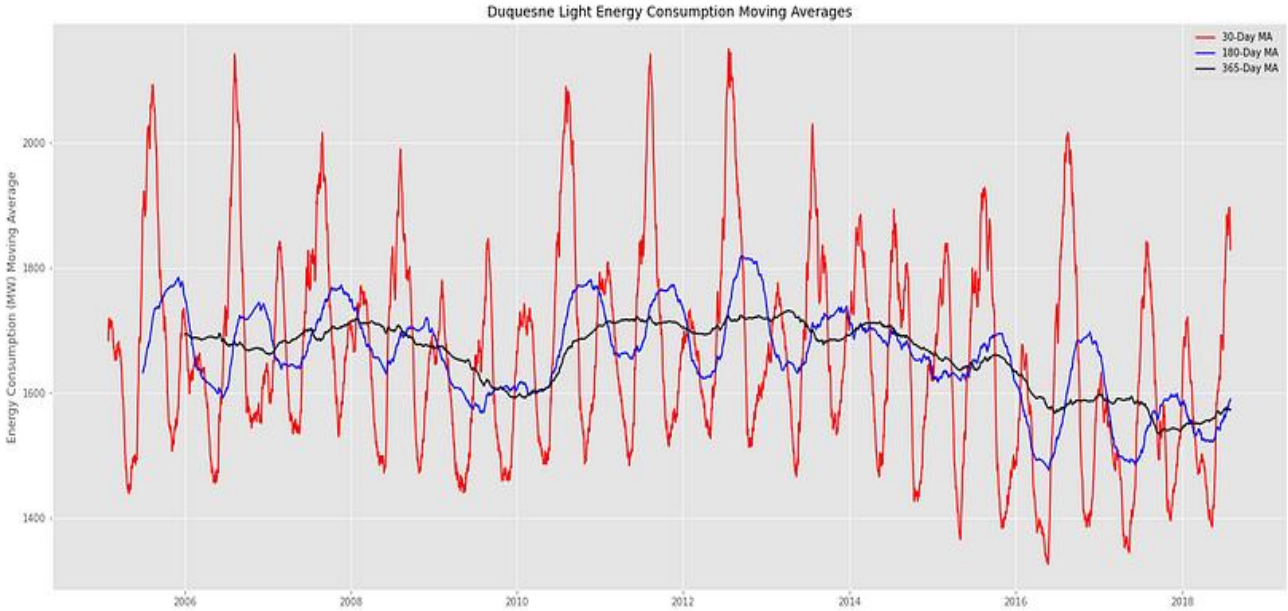


These plots help us identify the unique fluctuations of energy consumption rate across each year. For example, in 2008 the peak energy consumption rate occurs much earlier than in other years. Some years, such as 2012, demonstrate a clear peak summer season, while other years, such as 2015, also show local maxima in the winter season as well.

Moving Average

In the plots we just created, we used a smoothing method called a moving average. A **moving average** is an average value calculated over a fixed window of observations. The average value is calculated over the fixed window length, and the window is shifted one unit into the future. A new average value is calculated, and this process is repeated in a stepwise fashion across the entire time series.

```
MONTH_PERIOD = 24*30
MIDYEAR_PERIOD = 24*182
YEAR_PERIOD = 24*365month_roll =
duq_df.rolling(MONTH_PERIOD).mean()
midyear_roll = duq_df.rolling(MIDYEAR_PERIOD).mean()
year_roll = duq_df.rolling(YEAR_PERIOD).mean()fig, ax =
plt.subplots(figsize=(24, 10))
plt.plot(month_roll.index, month_roll['DUQ_MW'], color='red',
label='30-Day MA')
plt.plot(midyear_roll.index, midyear_roll['DUQ_MW'], color='blue',
label='180-Day MA')
plt.plot(year_roll.index, year_roll['DUQ_MW'], color='black',
label='365-Day MA')
plt.title('Duquesne Light Energy Consumption Moving Averages')
plt.ylabel('Energy Consumption (MW) Moving Average')
plt.legend()
plt.show()
```



Seasonality and Classical Seasonal Decomposition

Classical seasonal decomposition asserts that time series data can be broken apart into four separate components:

- Base Level / Average Value — The average, stationary value of the observations
- Trend — Increase / decrease of observations over time
- Seasonality — A pattern in the data that repeats over a fixed period
- Residuals — Error

For our analysis, the Base Level and Trend will be combined to form a single Trend component.

It is important to note the difference between seasonal and cyclical patterns, as both may be observed in time series data. **Seasonal patterns** occur with a regular, fixed period. Seasonality may be used to describe regular patterns that repeat daily, weekly, annually, etc. **Cyclic patterns**, on the other hand, are observed when data rises and falls with an irregular period.

When trying to determine if an observed pattern is seasonal or cyclic, first determine whether or not the pattern recurs at regular intervals. Patterns that occur at regular intervals can usually be identified by period lengths that correspond to our calendar/time-keeping system. For example, patterns that repeat daily, weekly, or annually would be considered seasonal.

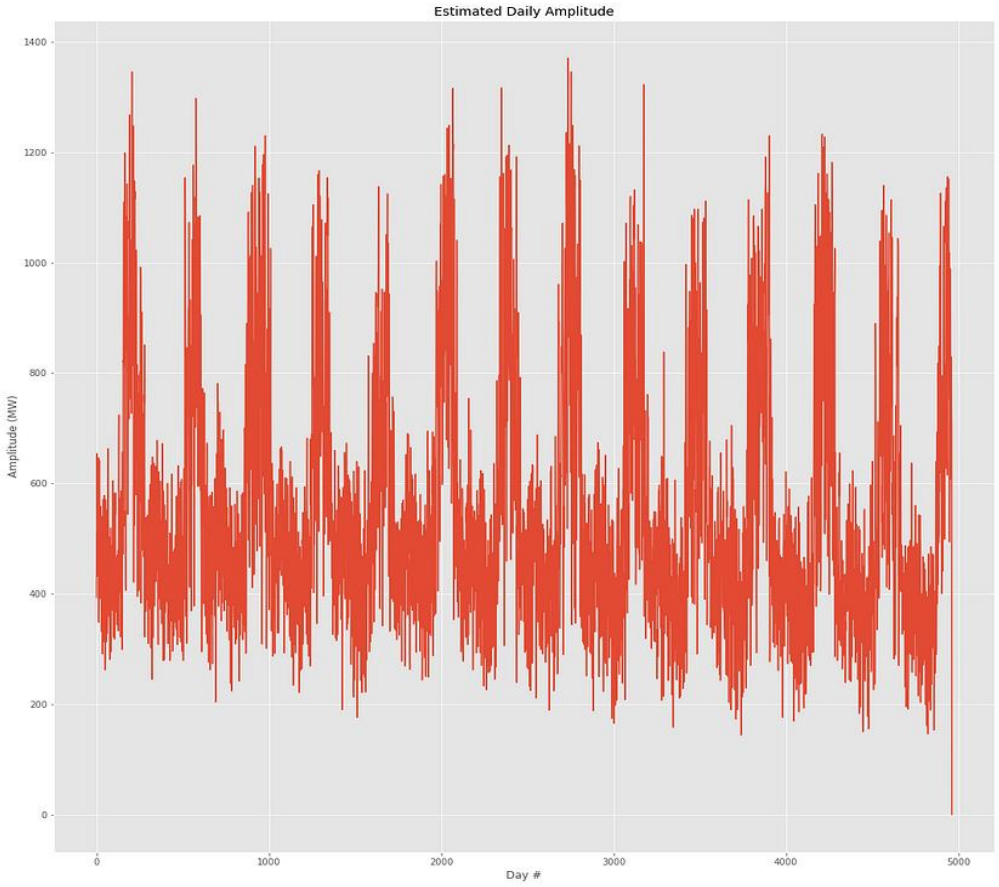
Let's break our energy consumption data into three components (trend, seasonality, and residuals). To do this, we will use the seasonal_decompose function that is provided as part of the statsmodels module.

Additive vs. Multiplicative Decomposition

When using the seasonal_decompose function, we will need to specify whether the decomposition should assume an additive or multiplicative model. An **additive** model assumes that the sum of the individual time series components (Trend + Seasonality + Residuals) is equal to the observed data. A **multiplicative** model assumes that the product of the individual time series components (Trend * Seasonality * Residuals) is equal to the observed data.

Typically, additive decomposition is most appropriate if there is little to no variation in the seasonality over time. Multiplicative decomposition is most appropriate when the magnitude of seasonality varies over time. Before we perform seasonal decomposition, we will look at the magnitude of some of our seasonalities by measuring the peak-to-peak amplitude of each seasonal waveform. The peak-to-peak amplitude is equal to the difference between the maximum value (peak) and minimum value (trough) within each seasonal cycle.

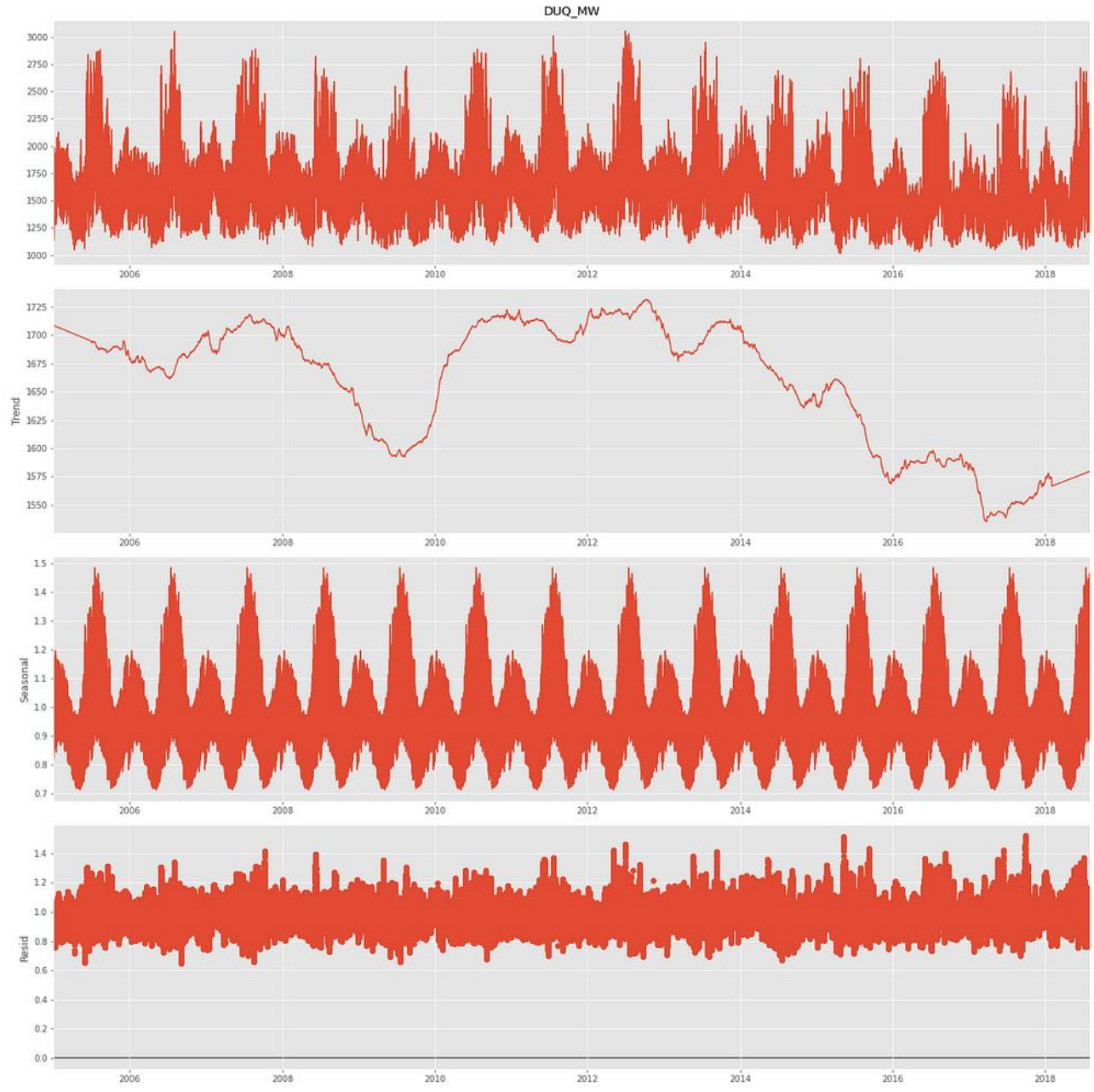
```
max_daily_vals = duq_df.groupby(['Year',
'DayOfYear']).max()['DUQ_MW'].values
min_daily_vals = duq_df.groupby(['Year',
'DayOfYear']).min()['DUQ_MW'].values
daily_amp = max_daily_vals - min_daily_valsplt.plot(daily_amp)
plt.xlabel('Day #')
plt.ylabel('Amplitude (MW)')
plt.title('Estimated Daily Amplitude')
plt.show()
```



We can see that the daily amplitude also follows a seasonal pattern, increasing or decreasing depending on the day of the year. In an additive model, we would expect the amplitude to remain relatively constant. Since the amplitude changes over time, we will assume that a multiplicative model best applies to the data.

```
from statsmodels.tsa.seasonal import seasonal_decompose
ANNUAL_PERIOD = 365*24mult_decomp =
```

```
seasonal_decompose(duq_df['DUQ_MW'], model='multiplicative',
extrapolate_trend='freq', period=ANNUAL_PERIOD)
mult_decomp.plot()
plt.show()
```



Conclusion:

Analyzing an energy consumption dataset is essential for understanding and managing energy usage effectively. Your well-structured conclusion should provide a clear and actionable roadmap for making informed decisions and taking steps to optimize energy consumption, reduce costs, and minimize environmental impact.