Achyut Golakiya

March 18, 2024

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from xgboost import XGBRegressor
from datetime import timedelta
from sklearn.ensemble import RandomForestRegressor
```

[2]:
```
data1=pd.read_csv("energy_dataset.csv")
data1.head()
```

[2]:
```
                        time  generation biomass  \
0  2015-01-01 00:00:00+01:00               447.0
1  2015-01-01 01:00:00+01:00               449.0
2  2015-01-01 02:00:00+01:00               448.0
3  2015-01-01 03:00:00+01:00               438.0
4  2015-01-01 04:00:00+01:00               428.0

   generation fossil brown coal/lignite  generation fossil coal-derived gas  \
0                                 329.0                                 0.0
1                                 328.0                                 0.0
2                                 323.0                                 0.0
3                                 254.0                                 0.0
4                                 187.0                                 0.0

   generation fossil gas  generation fossil hard coal  generation fossil oil  \
0                 4844.0                       4821.0                  162.0
1                 5196.0                       4755.0                  158.0
2                 4857.0                       4581.0                  157.0
3                 4314.0                       4131.0                  160.0
4                 4130.0                       3840.0                  156.0

   generation fossil oil shale  generation fossil peat  generation geothermal  \
```

```
0                          0.0                     0.0                    0.0
1                          0.0                     0.0                    0.0
2                          0.0                     0.0                    0.0
3                          0.0                     0.0                    0.0
4                          0.0                     0.0                    0.0

   …  generation waste  generation wind offshore  generation wind onshore  \
0  …             196.0                       0.0                   6378.0
1  …             195.0                       0.0                   5890.0
2  …             196.0                       0.0                   5461.0
3  …             191.0                       0.0                   5238.0
4  …             189.0                       0.0                   4935.0

   forecast solar day ahead  forecast wind offshore eday ahead  \
0                      17.0                                NaN
1                      16.0                                NaN
2                       8.0                                NaN
3                       2.0                                NaN
4                       9.0                                NaN

   forecast wind onshore day ahead  total load forecast  total load actual  \
0                           6436.0              26118.0            25385.0
1                           5856.0              24934.0            24382.0
2                           5454.0              23515.0            22734.0
3                           5151.0              22642.0            21286.0
4                           4861.0              21785.0            20264.0

   price day ahead  price actual
0            50.10         65.41
1            48.10         64.92
2            47.33         64.48
3            42.27         59.32
4            38.41         56.04

[5 rows x 29 columns]
```

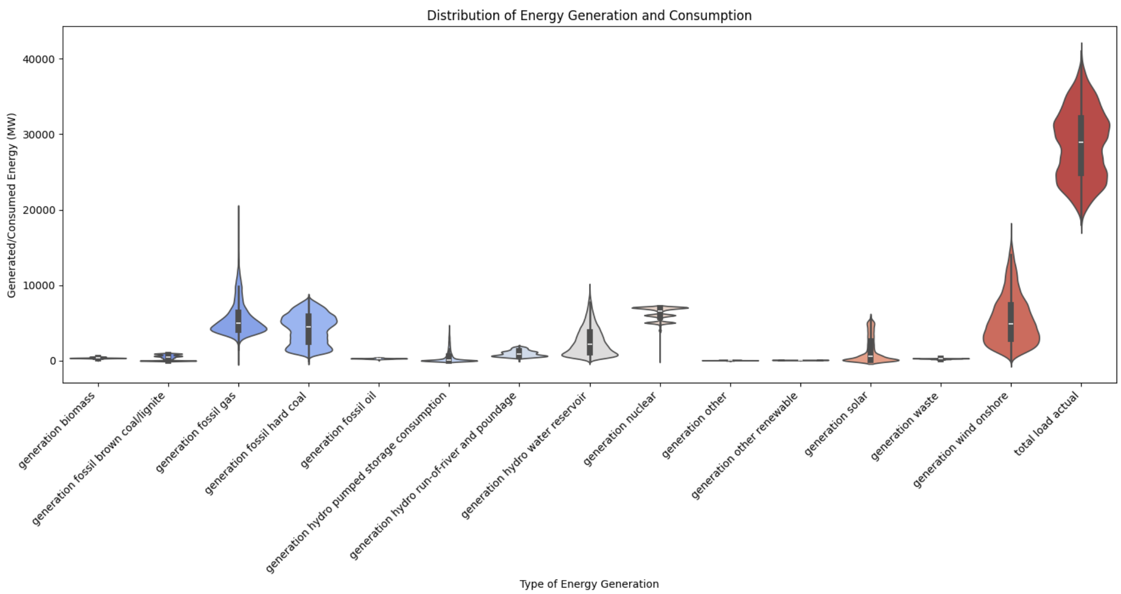
[3]:

```python
# Convert the 'time' column to datetime
data1['time'] = pd.to_datetime(data1['time'])

# Define the columns we want to plot
columns_to_plot = [
    'generation biomass',
    'generation fossil brown coal/lignite',
    'generation fossil gas',
    'generation fossil hard coal',
    'generation fossil oil',
    'generation hydro pumped storage consumption',
```

```
    'generation hydro run-of-river and poundage',
    'generation hydro water reservoir',
    'generation nuclear',
    'generation other',
    'generation other renewable',
    'generation solar',
    'generation waste',
    'generation wind onshore',
    'total load actual'
]

# Filter the dataframe to only include the desired columns
data_to_plot = data1[columns_to_plot]

# Melt the data so that we have one column for variable names and one for values
data_long = pd.melt(data_to_plot, var_name='Type of Generation',␣
↪value_name='Generation Output (MW)')

# Set a custom color palette
palette = sns.color_palette("coolwarm", len(columns_to_plot))

# Create the violin plot
plt.figure(figsize=(15, 8))
sns.violinplot(x='Type of Generation', y='Generation Output (MW)',␣
↪data=data_long, palette=palette)

# Improve the aesthetics
plt.xticks(rotation=45, ha='right')  # Rotate the x labels for better␣
↪readability
plt.title('Distribution of Energy Generation and Consumption')
plt.xlabel('Type of Energy Generation')
plt.ylabel('Generated/Consumed Energy (MW)')

# Show the plot
plt.tight_layout()  # This adjusts subplot params for the figure to fit into␣
↪the display area
plt.show()
```

```
C:\Users\achyu\AppData\Local\Temp\ipykernel_19492\1821487972.py:2:
FutureWarning: In a future version of pandas, parsing datetimes with mixed time
zones will raise an error unless `utc=True`. Please specify `utc=True` to opt in to
the new behaviour and silence this warning. To create a `Series` with mixed offsets
and `object` dtype, please use `apply` and `datetime.datetime.strptime`
 data1['time'] = pd.to_datetime(data1['time'])
C:\Users\achyu\AppData\Local\Temp\ipykernel_19492\1821487972.py:34:
FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.violinplot(x='Type of Generation', y='Generation Output (MW)',
data=data_long, palette=palette)
```



[4]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 29 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   time                                         35064 non-null  object
 1   generation biomass                           35045 non-null  float64
 2   generation fossil brown coal/lignite         35046 non-null  float64
 3   generation fossil coal-derived gas           35046 non-null  float64
 4   generation fossil gas                        35046 non-null  float64
 5   generation fossil hard coal                  35046 non-null  float64
 6   generation fossil oil                        35045 non-null  float64
 7   generation fossil oil shale                  35046 non-null  float64
 8   generation fossil peat                       35046 non-null  float64
 9   generation geothermal                        35046 non-null  float64
 10  generation hydro pumped storage aggregated   0 non-null      float64
 11  generation hydro pumped storage consumption  35045 non-null  float64
 12  generation hydro run-of-river and poundage   35045 non-null  float64
 13  generation hydro water reservoir             35046 non-null  float64
```

```
 14  generation marine                          35045 non-null  float64
 15  generation nuclear                         35047 non-null  float64
 16  generation other                           35046 non-null  float64
 17  generation other renewable                 35046 non-null  float64
 18  generation solar                           35046 non-null  float64
 19  generation waste                           35045 non-null  float64
 20  generation wind offshore                   35046 non-null  float64
 21  generation wind onshore                    35046 non-null  float64
 22  forecast solar day ahead                   35064 non-null  float64
 23  forecast wind offshore eday ahead          0 non-null      float64
 24  forecast wind onshore day ahead            35064 non-null  float64
 25  total load forecast                        35064 non-null  float64
 26  total load actual                          35028 non-null  float64
 27  price day ahead                            35064 non-null  float64
 28  price actual                               35064 non-null  float64
dtypes: float64(28), object(1)
memory usage: 7.8+ MB
```

[5]:
```python
data1.describe()
```

[5]:
```
       generation biomass  generation fossil brown coal/lignite  \
count        35045.000000                          35046.000000
mean           383.513540                            448.059208
std             85.353943                            354.568590
min              0.000000                              0.000000
25%            333.000000                              0.000000
50%            367.000000                            509.000000
75%            433.000000                            757.000000
max            592.000000                            999.000000

       generation fossil coal-derived gas  generation fossil gas  \
count                             35046.0           35046.000000
mean                                  0.0            5622.737488
std                                   0.0            2201.830478
min                                   0.0               0.000000
25%                                   0.0            4126.000000
50%                                   0.0            4969.000000
75%                                   0.0            6429.000000
max                                   0.0           20034.000000

       generation fossil hard coal  generation fossil oil  \
count                 35046.000000           35045.000000
mean                   4256.065742             298.319789
std                    1961.601013              52.520673
min                       0.000000               0.000000
25%                    2527.000000             263.000000
50%                    4474.000000             300.000000
```

```
75%                  5838.750000            330.000000
max                  8359.000000            449.000000

       generation fossil oil shale  generation fossil peat  \
count                      35046.0                 35046.0
mean                           0.0                     0.0
std                            0.0                     0.0
min                            0.0                     0.0
25%                            0.0                     0.0
50%                            0.0                     0.0
75%                            0.0                     0.0
max                            0.0                     0.0

       generation geothermal  generation hydro pumped storage aggregated  …  \
count                35046.0                                         0.0  …
mean                     0.0                                         NaN  …
std                      0.0                                         NaN  …
min                      0.0                                         NaN  …
25%                      0.0                                         NaN  …
50%                      0.0                                         NaN  …
75%                      0.0                                         NaN  …
max                      0.0                                         NaN  …

       generation waste  generation wind offshore  generation wind onshore  \
count      35045.000000                   35046.0             35046.000000
mean         269.452133                       0.0              5464.479769
std           50.195536                       0.0              3213.691587
min            0.000000                       0.0                 0.000000
25%          240.000000                       0.0              2933.000000
50%          279.000000                       0.0              4849.000000
75%          310.000000                       0.0              7398.000000
max          357.000000                       0.0             17436.000000

       forecast solar day ahead  forecast wind offshore eday ahead  \
count              35064.000000                                0.0
mean                1439.066735                                NaN
std                 1677.703355                                NaN
min                    0.000000                                NaN
25%                   69.000000                                NaN
50%                  576.000000                                NaN
75%                 2636.000000                                NaN
max                 5836.000000                                NaN

       forecast wind onshore day ahead  total load forecast  \
count                     35064.000000         35064.000000
mean                       5471.216689         28712.129962
std                        3176.312853          4594.100854
```

```
min                        237.000000         18105.000000
25%                       2979.000000         24793.750000
50%                       4855.000000         28906.000000
75%                       7353.000000         32263.250000
max                      17430.000000         41390.000000

       total load actual  price day ahead  price actual
count       35028.000000     35064.000000  35064.000000
mean        28696.939905        49.874341     57.884023
std          4574.987950        14.618900     14.204083
min         18041.000000         2.060000      9.330000
25%         24807.750000        41.490000     49.347500
50%         28901.000000        50.520000     58.020000
75%         32192.000000        60.530000     68.010000
max         41015.000000       101.990000    116.800000

[8 rows x 28 columns]
```

As we have a lot of data columns, so we will take only the relevant columns required for our model. Keep in mind 'price' is an important column, but we also require 'Appliances' data that can be used for forecasting model. As we don't have 'Appliances' data so we will not consider 'Price' column data for our model.

[6]:
```python
energy_df=data1.copy(deep=True)
energy_df = energy_df.drop(columns=['forecast solar day ahead','forecast wind offshore eday ahead','forecast wind onshore day ahead','total load forecast','price day ahead','generation hydro pumped storage aggregated','price actual'])
energy_df.isna().sum()
```

[6]:
```
time                                            0
generation biomass                             19
generation fossil brown coal/lignite           18
generation fossil coal-derived gas             18
generation fossil gas                          18
generation fossil hard coal                    18
generation fossil oil                          19
generation fossil oil shale                    18
generation fossil peat                         18
generation geothermal                          18
generation hydro pumped storage consumption    19
generation hydro run-of-river and poundage     19
generation hydro water reservoir               18
generation marine                              19
generation nuclear                             17
generation other                               18
generation other renewable                     18
```

```
generation solar                                 18
generation waste                                 19
generation wind offshore                         18
generation wind onshore                          18
total load actual                                36
dtype: int64
```

[7]:

```
#Changing the type of 'time' from string to date-time
energy_df['time']=pd.to_datetime(energy_df['time'],utc=True)
energy_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35064 entries, 0 to 35063
Data columns (total 22 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   time                                         35064 non-null  datetime64[ns,
UTC]
 1   generation biomass                           35045 non-null  float64
 2   generation fossil brown coal/lignite         35046 non-null  float64
 3   generation fossil coal-derived gas           35046 non-null  float64
 4   generation fossil gas                        35046 non-null  float64
 5   generation fossil hard coal                  35046 non-null  float64
 6   generation fossil oil                        35045 non-null  float64
 7   generation fossil oil shale                  35046 non-null  float64
 8   generation fossil peat                       35046 non-null  float64
 9   generation geothermal                        35046 non-null  float64
 10  generation hydro pumped storage consumption  35045 non-null  float64
 11  generation hydro run-of-river and poundage   35045 non-null  float64
 12  generation hydro water reservoir             35046 non-null  float64
 13  generation marine                            35045 non-null  float64
 14  generation nuclear                           35047 non-null  float64
 15  generation other                             35046 non-null  float64
 16  generation other renewable                   35046 non-null  float64
 17  generation solar                             35046 non-null  float64
 18  generation waste                             35045 non-null  float64
 19  generation wind offshore                     35046 non-null  float64
 20  generation wind onshore                      35046 non-null  float64
 21  total load actual                            35028 non-null  float64
dtypes: datetime64[ns, UTC](1), float64(21)
memory usage: 5.9 MB
```

As we can see, 'energy_df' dataframe has no duplicate values. Nevertheless, it has some NaNs and thus, we have to investigate further. Since this is a time-series forecasting task, we cannot simply drop the rows with the missing values and it would be a better idea to fill the missing values using interpolation.

[8]:
```
print('There are {} missing values or NaNs in energy_df.'
      .format(energy_df.isnull().values.sum()))
temp_energy = energy_df.duplicated(keep='first').sum()

print('There are {} duplicate rows in energy_df based on all columns.'
      .format(temp_energy))
energy_df.isnull().sum(axis=0)
```

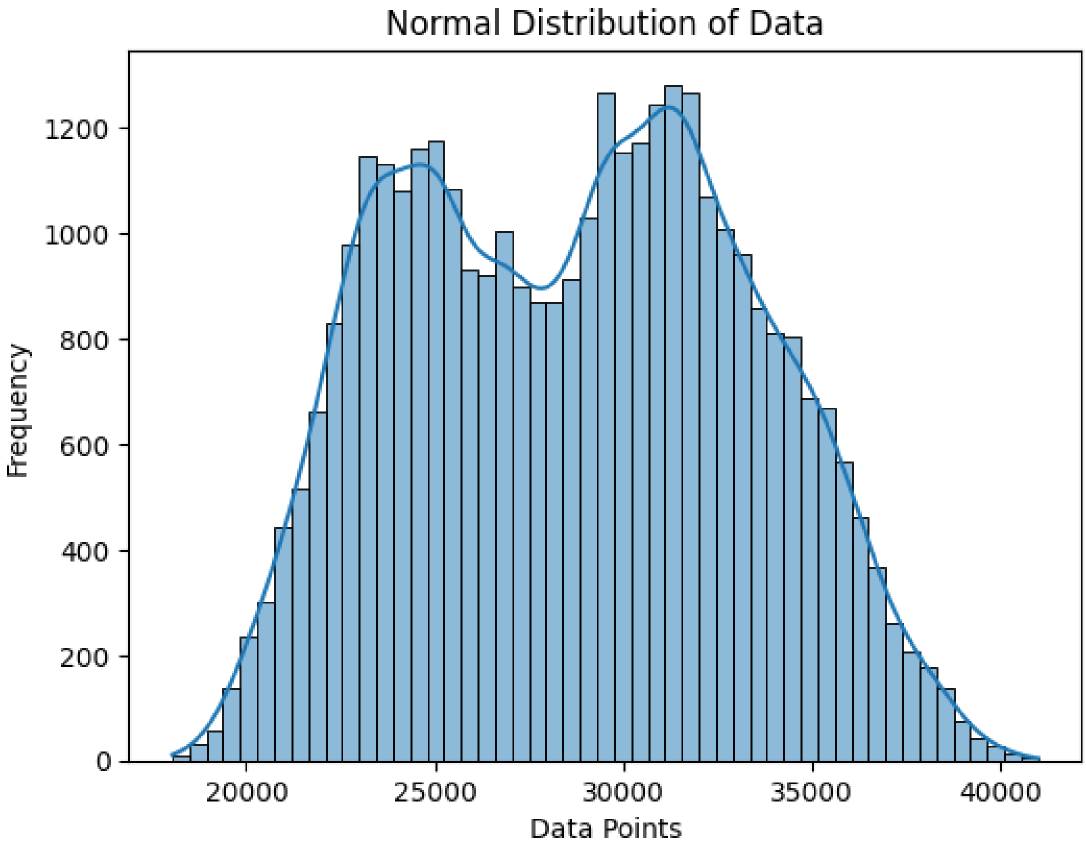
```
There are 401 missing values or NaNs in energy_df.
There are 0 duplicate rows in energy_df based on all columns.
```

[8]:
```
time                                            0
generation biomass                             19
generation fossil brown coal/lignite           18
generation fossil coal-derived gas             18
generation fossil gas                          18
generation fossil hard coal                    18
generation fossil oil                          19
generation fossil oil shale                    18
generation fossil peat                         18
generation geothermal                          18
generation hydro pumped storage consumption    19
generation hydro run-of-river and poundage     19
generation hydro water reservoir               18
generation marine                              19
generation nuclear                             17
generation other                               18
generation other renewable                     18
generation solar                               18
generation waste                               19
generation wind offshore                       18
generation wind onshore                        18
total load actual                              36
dtype: int64
```

Most null values can be found in the 'total load actual' column which represents the energy consumption. Therefore, it is a good idea to visualize it and see what we can do. The similar numbers in null values in the columns which have to do with the type of energy generation probably indicate that they will also appear in the same rows. Let us first define a normal distribution to see the irregualrities.

[9]:
```
# Plotting the distribution
sns.histplot(energy_df['total load actual'], kde=True)  # 'kde=True' adds the␣
 ↪Kernel Density Estimate to smooth the histogram
plt.title('Normal Distribution of Data')
plt.xlabel('Data Points')
plt.ylabel('Frequency')
```

```
plt.show()
```

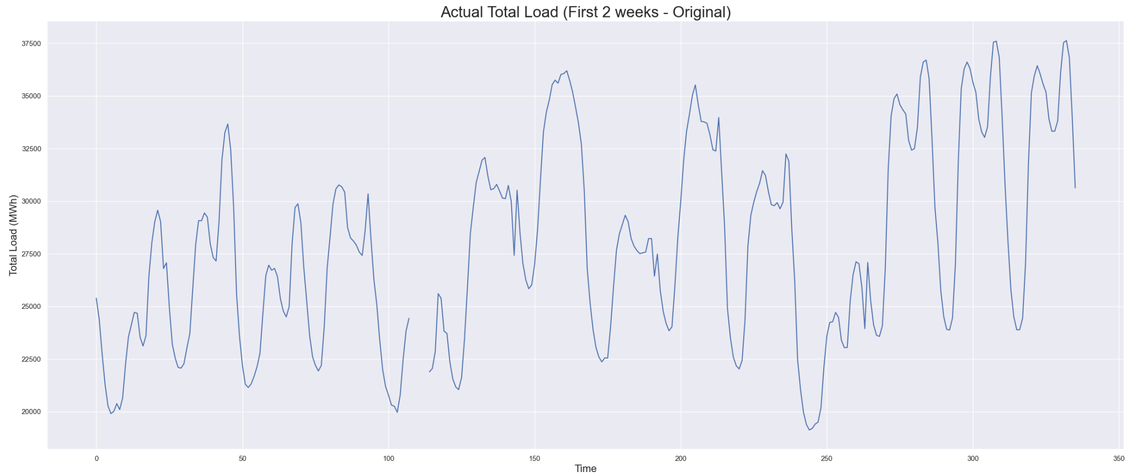


Now lets see using a line plot.

```
[10]: def plot_series(df=None, column=None, series=pd.Series([]),
                    label=None, ylabel=None, title=None, start=0, end=None):
          sns.set()
          fig, ax = plt.subplots(figsize=(30, 12))
          ax.set_xlabel('Time', fontsize=16)
          if column:
              ax.plot(df[column][start:end], label=label)
              ax.set_ylabel(ylabel, fontsize=16)
          if series.any():
              ax.plot(series, label=label)
              ax.set_ylabel(ylabel, fontsize=16)
          if label:
              ax.legend(fontsize=16)
          if title:
              ax.set_title(title, fontsize=24)
          ax.grid(True)
```

```
    return ax
```

[11]:
```
# Zoom into the plot of the hourly (actual) total load

ax = plot_series(df=energy_df, column='total load actual', ylabel='Total Load␣
↪(MWh)',
                 title='Actual Total Load (First 2 weeks - Original)',␣
↪end=24*7*2)
plt.show()
```



After zooming into the first 2 weeks of the 'total load actual' column, we can already see that there are null values for a few hours. However, the number of the missing values and the behavior of the series indicate that an interpolation would fill the NaNs quite well. Let us further investigate if the null values coincide across the different columns. Let us display the last five rows.

[12]:
```
# Display the rows with null values
energy_df[energy_df.isnull().any(axis=1)].tail()
```

[12]:
```
                            time  generation biomass  \
16612  2016-11-23 03:00:00+00:00                 NaN
25164  2017-11-14 11:00:00+00:00                 0.0
25171  2017-11-14 18:00:00+00:00                 0.0
30185  2018-06-11 16:00:00+00:00               331.0
30896  2018-07-11 07:00:00+00:00                 NaN

       generation fossil brown coal/lignite  \
16612                                 900.0
25164                                   0.0
25171                                   0.0
30185                                 506.0
30896                                   NaN
```

	generation fossil coal-derived gas	generation fossil gas	\
16612	0.0	4838.0	
25164	0.0	10064.0	
25171	0.0	12336.0	
30185	0.0	7538.0	
30896	NaN	NaN	

	generation fossil hard coal	generation fossil oil	\
16612	4547.0	269.0	
25164	0.0	0.0	
25171	0.0	0.0	
30185	5360.0	300.0	
30896	NaN	NaN	

	generation fossil oil shale	generation fossil peat	\
16612	0.0	0.0	
25164	0.0	0.0	
25171	0.0	0.0	
30185	0.0	0.0	
30896	NaN	NaN	

	generation geothermal	…	generation hydro water reservoir	\
16612	0.0	…	435.0	
25164	0.0	…	0.0	
25171	0.0	…	0.0	
30185	0.0	…	4258.0	
30896	NaN	…	NaN	

	generation marine	generation nuclear	generation other	\
16612	0.0	5040.0	60.0	
25164	0.0	0.0	0.0	
25171	0.0	0.0	0.0	
30185	0.0	5856.0	52.0	
30896	NaN	NaN	NaN	

	generation other renewable	generation solar	generation waste	\
16612	85.0	15.0	227.0	
25164	0.0	0.0	0.0	
25171	0.0	0.0	0.0	
30185	96.0	170.0	269.0	
30896	NaN	NaN	NaN	

	generation wind offshore	generation wind onshore	total load actual
16612	0.0	4598.0	23112.0
25164	0.0	0.0	NaN
25171	0.0	0.0	NaN

```
30185                       0.0                9165.0              NaN
30896                       NaN                   NaN              NaN

[5 rows x 22 columns]
```

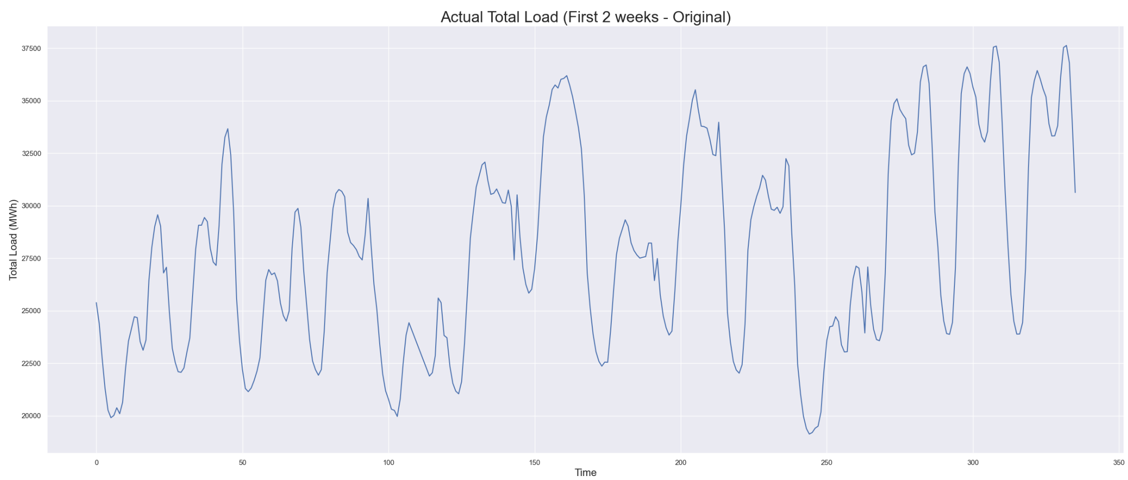
If we manually searched through all of them, we would confirm that the null values in the columns which have to do with the type of energy generation mostly coincide. The null values in 'actual total load' also coincide with the aforementioned columns, but also appear in other rows as well. In order to handle the null values in df_energy, we will use a linear interpolation with a forward direction. Perhaps other kinds of interpolation would be better; nevertheless, we prefer to use the simplest model possible. Only a small part of our input data will be noisy and it will not affect performance noticeably.

[13]:
```python
energy_df.replace(0, np.nan, inplace=True)
# Fill null values using interpolation
energy_df.interpolate(method='linear', limit_direction='forward', inplace=True,␣
 ↪axis=0)
# Display the number of non-zero values in each column

print('Non-zero values in each column:\n', energy_df.astype(bool).sum(axis=0),␣
 ↪sep='\n')
```

```
Non-zero values in each column:

time                                            35064
generation biomass                              35064
generation fossil brown coal/lignite            35064
generation fossil coal-derived gas              35064
generation fossil gas                           35064
generation fossil hard coal                     35064
generation fossil oil                           35064
generation fossil oil shale                     35064
generation fossil peat                          35064
generation geothermal                           35064
generation hydro pumped storage consumption     35064
generation hydro run-of-river and poundage      35064
generation hydro water reservoir                35064
generation marine                               35064
generation nuclear                              35064
generation other                                35064
generation other renewable                      35064
generation solar                                35064
generation waste                                35064
generation wind offshore                        35064
generation wind onshore                         35064
total load actual                               35064
dtype: int64
```

As we can see the count of values of all columns is similar, so now lets see through the line plot again

[14]:
```
ax = plot_series(df=energy_df, column='total load actual', ylabel='Total Load␣
↪(MWh)',
                  title='Actual Total Load (First 2 weeks - Original)',␣
↪end=24*7*2)
plt.show()
```



Now we filter out all the necessary column as per business requirements.

[15]:
```
energy_df=energy_df.drop(columns=['generation fossil coal-derived␣
↪gas','generation fossil oil shale','generation fossil peat','generation␣
↪geothermal','generation marine','generation wind offshore'])
energy_df.to_csv('energy_df.csv')
```

Now lets start the modelling process. We tried multiple models like ARIMA, Auto-ARIMA, SARIMAX, Prophet, XGBoost and RFR. Finally we proceeded with XGBoost & RFR.

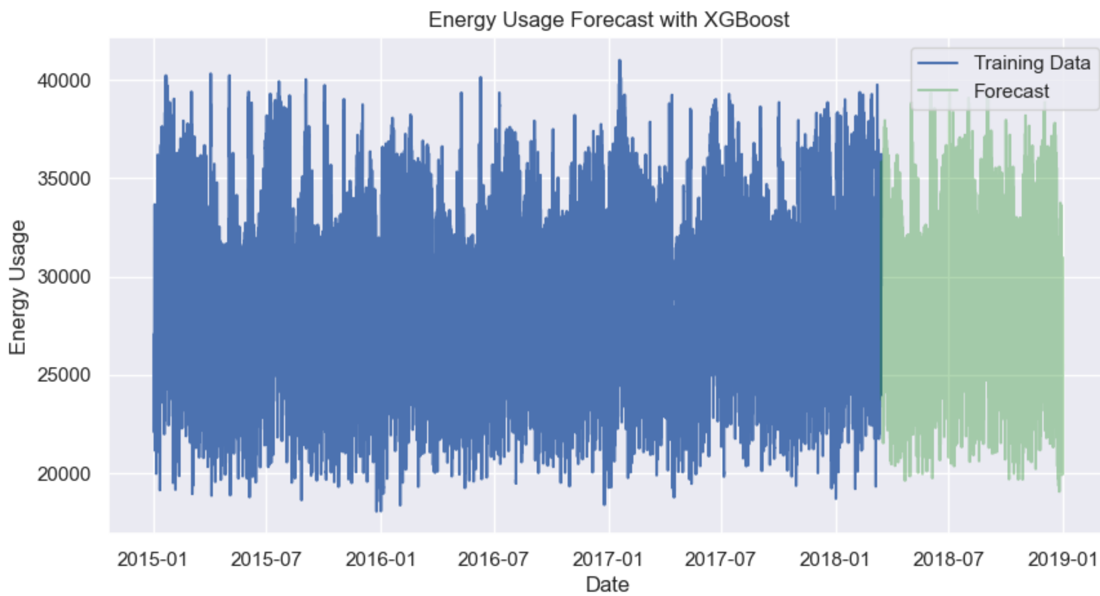
[16]:
```
# Load the dataset
df = pd.read_csv('energy_df.csv')

# Assume the date column is named 'time', change if different
df['time'] = pd.to_datetime(df['time'])
df.set_index('time', inplace=True)
df.sort_index(inplace=True)

# Create features: lag features and rolling means
for i in range(1, 25):  # extending lags to 24 hours
    df[f'lag_{i}'] = df['total load actual'].shift(i)
df['rolling_mean_6'] = df['total load actual'].rolling(window=6).mean()
df['rolling_mean_12'] = df['total load actual'].rolling(window=12).mean()
```

```python
# Drop NaN values that were created by lag features
df = df.dropna()

# Split features and target
X = df.drop('total load actual', axis=1)
y = df['total load actual']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, shuffle=False)

# Define the model
model = XGBRegressor(objective='reg:squarederror', random_state=42)

# Setup GridSearchCV
param_grid = {
    'n_estimators': [100, 500, 1000],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.7, 0.8, 0.9]
}

grid_search = GridSearchCV(model, param_grid, cv=3, scoring='r2', verbose=1)
grid_search.fit(X_train, y_train)

# Best model
best_model = grid_search.best_estimator_

# Make predictions
y_pred = best_model.predict(X_test)

# Calculate accuracy metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
```

[17]:
```python
plt.figure(figsize=(10, 5))
plt.plot(y_train.index, y_train, label='Training Data')
#plt.plot(y_test.index, y_test, label='Test Data')
plt.plot(y_test.index, y_pred, label='Forecast', color='green', alpha=0.3)
plt.title('Energy Usage Forecast with XGBoost')
plt.xlabel('Date')
plt.ylabel('Energy Usage')
```

```
plt.legend()
plt.show()
```



[18]:
```
# Display the metrics
print("Best model params:", grid_search.best_params_)
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R^2 Score:", r2)
```

```
Best model params: {'learning_rate': 0.05, 'max_depth': 7, 'n_estimators': 1000,
'subsample': 0.7}
Mean Absolute Error (MAE): 273.023627294253
Mean Squared Error (MSE): 169249.43036922635
Root Mean Squared Error (RMSE): 411.3993563062859
R^2 Score: 0.9917100572564472
```

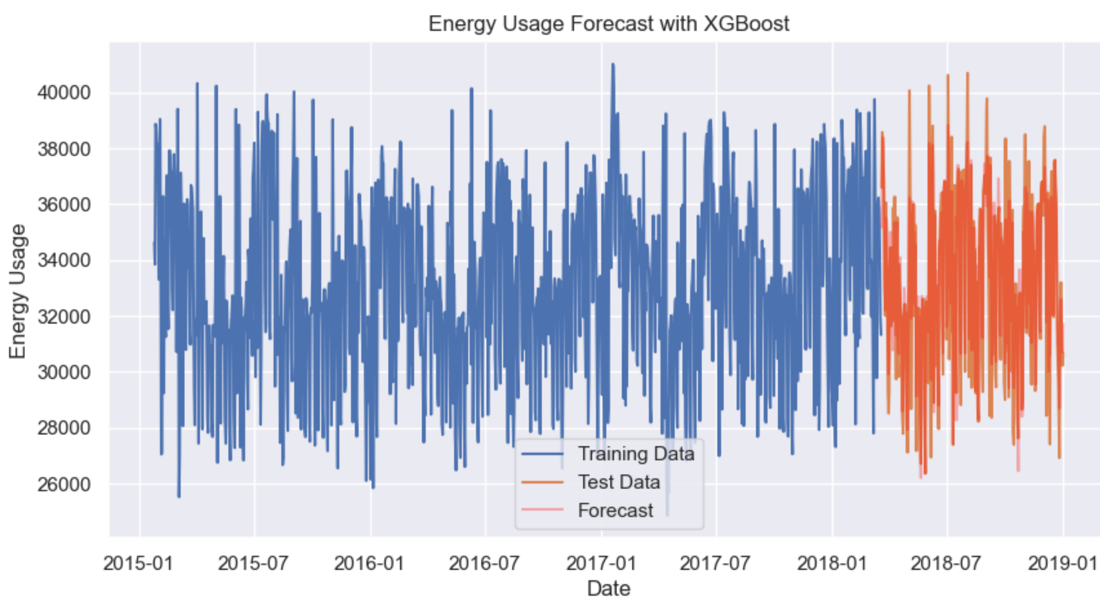
Looking into the R^2 value our model seems to be perfectly fit, but we want to add another additional requirement for our stakeholder in which we want to forecast the maximum energy consumption for each day according to the 'total load actual' column's maximum value for each day. For this we have filtered out data through the 'Excel' itself where we have taken the maximum values of 'total load actual' column for each day for the corresponding 'time' column.

[19]:
```
# Load the dataset
df = pd.read_csv("Maximum_Load_Per_Day_with_Timestamps.csv")
df['time'] = pd.to_datetime(df['time'], utc=True)
df.set_index('time', inplace=True)
df.sort_index(inplace=True)
```

```python
# Create features: lag features and rolling means
for i in range(1, 25):  # extending lags to 24 hours
    df[f'lag_{i}'] = df['total load actual'].shift(i)
df['rolling_mean_6'] = df['total load actual'].rolling(window=6).mean()
df['rolling_mean_12'] = df['total load actual'].rolling(window=12).mean()

# Drop NaN values that were created by lag features
df = df.dropna()

# Split features and target
X = df.drop('total load actual', axis=1)
y = df['total load actual']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, shuffle=False)

# Define the model
model = XGBRegressor(objective='reg:squarederror', random_state=42)

# Setup GridSearchCV
param_grid = {
    'n_estimators': [100, 500, 1000],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.7, 0.8, 0.9]
}

grid_search = GridSearchCV(model, param_grid, cv=3, scoring='r2', verbose=1)
grid_search.fit(X_train, y_train)

# Best model
best_model = grid_search.best_estimator_

# Make predictions
y_pred = best_model.predict(X_test)

# Calculate accuracy metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Display the metrics
print("Best model params:", grid_search.best_params_)
print("Mean Absolute Error (MAE):", mae)
```

```
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R^2 Score:", r2)

# Plotting the results
plt.figure(figsize=(10, 5))
plt.plot(y_train.index, y_train, label='Training Data')
plt.plot(y_test.index, y_test, label='Test Data')
plt.plot(y_test.index, y_pred, label='Forecast', color='red',alpha=0.3)
plt.title('Energy Usage Forecast with XGBoost')
plt.xlabel('Date')
plt.ylabel('Energy Usage')
plt.legend()
plt.show()
```

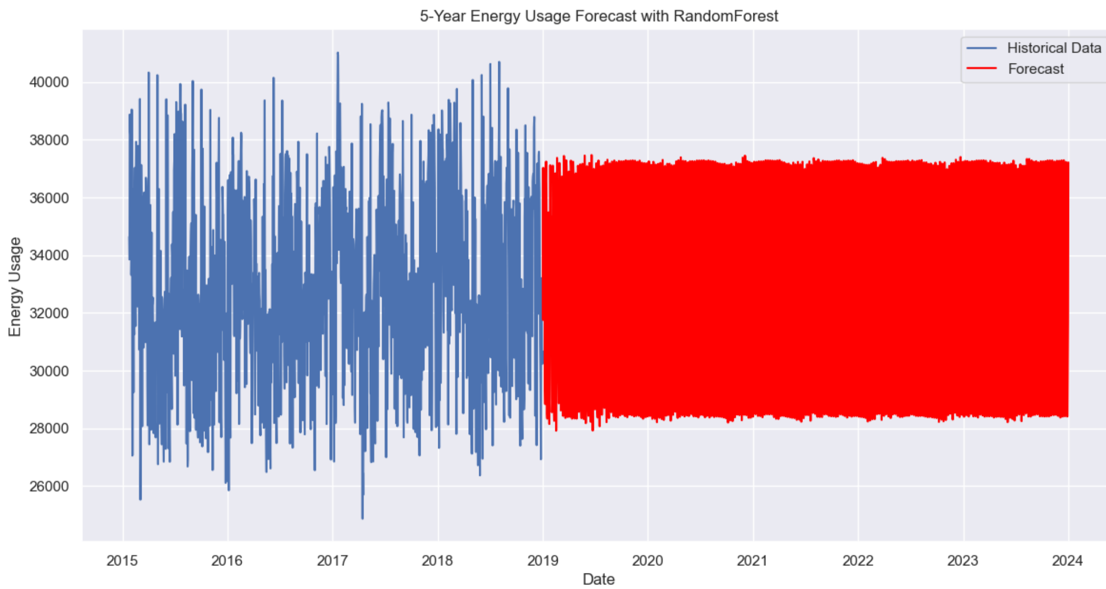
```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
Best model params: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 1000,
'subsample': 0.8}
Mean Absolute Error (MAE): 960.5805053710938
Mean Squared Error (MSE): 1627769.7854599026
Root Mean Squared Error (RMSE): 1275.8408150940706
R^2 Score: 0.8331241782707559
```



The above forecast was for 12 months for each 6 months . To forecast for next 5 years we have modified our parameters below.

[20]:
```python
start_date = df.index.max() + timedelta(days=1)
end_date = start_date + pd.DateOffset(years=5)

# Create a date range for the forecast period
future_dates = pd.date_range(start=start_date, end=end_date, freq='D')
# Initialize future DataFrame with necessary columns
future_df = pd.DataFrame(index=future_dates)
for i in range(1, 25):
    future_df[f'lag_{i}'] = np.nan

# Initially populate lag features using the last available values from `df`
last_values = df['total load actual'][-24:].values[::-1]
for i in range(1, 25):
    future_df.at[future_dates[0], f'lag_{i}'] = last_values[i-1]

# Now also calculate the rolling means for the first prediction point
future_df['rolling_mean_6'] = np.nan
future_df['rolling_mean_12'] = np.nan

# Start the recursive prediction and feature updating
predicted_values = []
for date in future_df.index:
    if len(predicted_values) >= 12:
        future_df.at[date, 'rolling_mean_12'] = np.mean(predicted_values[-12:])
    if len(predicted_values) >= 6:
        future_df.at[date, 'rolling_mean_6'] = np.mean(predicted_values[-6:])

    # Prepare the row for prediction, filling forward NaNs
    row = future_df.loc[date].fillna(method='ffill').to_frame().T
    prediction = best_model.predict(row)[0]
    predicted_values.append(prediction)

    # Update the lag features for the next day
    for i in range(1, 25):
        if date + timedelta(days=1) in future_df.index:
            future_df.at[date + timedelta(days=1), f'lag_{i}'] = prediction if␣
↪i == 1 else future_df.at[date, f'lag_{i-1}']

# Assign predictions back to future DataFrame
future_df['predicted_load'] = predicted_values

# Plot results
plt.figure(figsize=(14, 7))
plt.plot(df.index, df['total load actual'], label='Historical Data')
plt.plot(future_df.index, future_df['predicted_load'], label='Forecast',␣
↪color='red')
plt.title('5-Year Energy Usage Forecast')
```

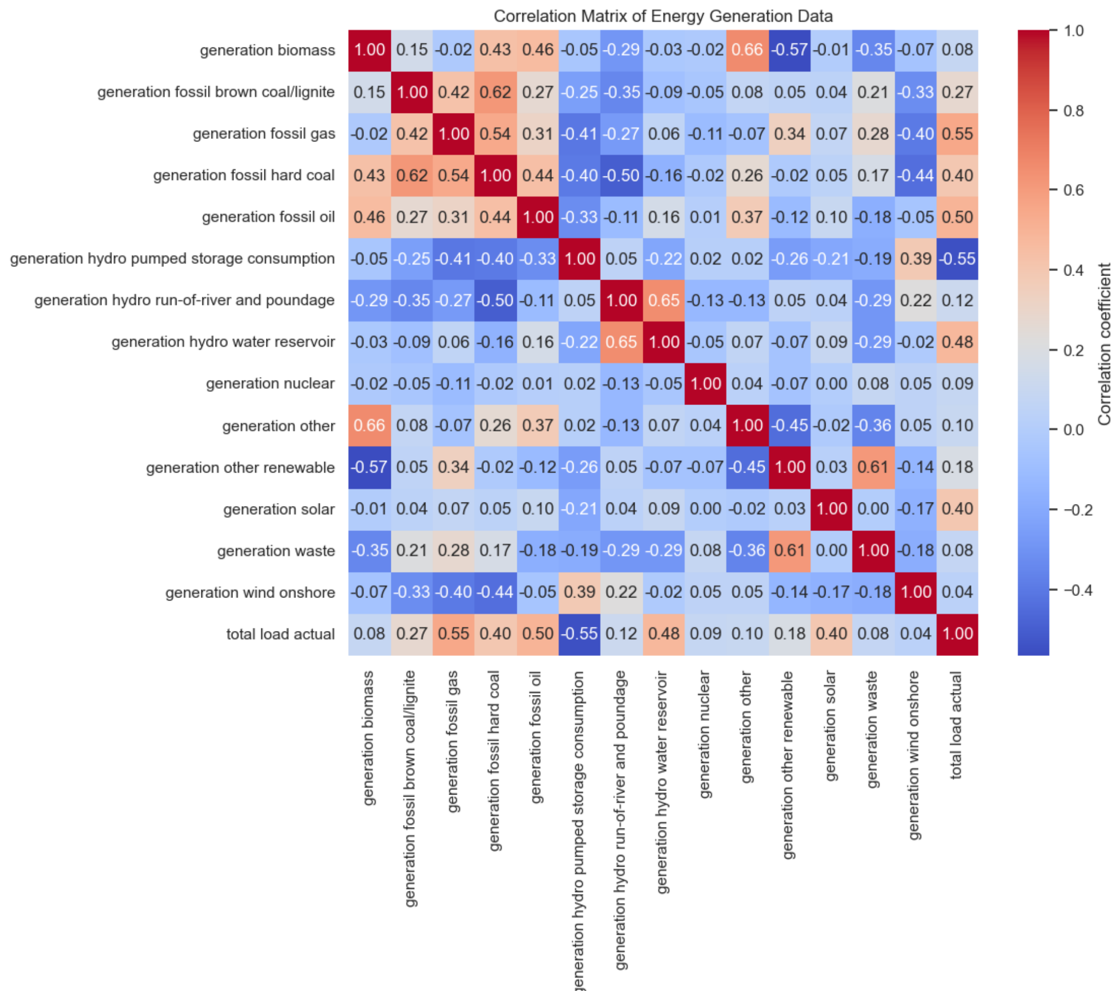
```
plt.xlabel('Date')
plt.ylabel('Energy Usage')
plt.legend()
plt.show()
```



Based on our model comparisons for forcasting, we are choosing XGBoost. Now we will try to find the correlation matrix between different energy sources of generation and 'total load actual' column.

[23]:
```
data = pd.read_csv('df_combined.csv')

# Exclude the non-relevant columns
columns_to_exclude = ['time', 'city_name']
data_filtered = data.drop(columns=columns_to_exclude)

# Calculate the correlation matrix
correlation_matrix = data_filtered.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm',␣
↪cbar_kws={'label': 'Correlation coefficient'})
plt.title('Correlation Matrix of Energy Generation Data')
plt.show()
```

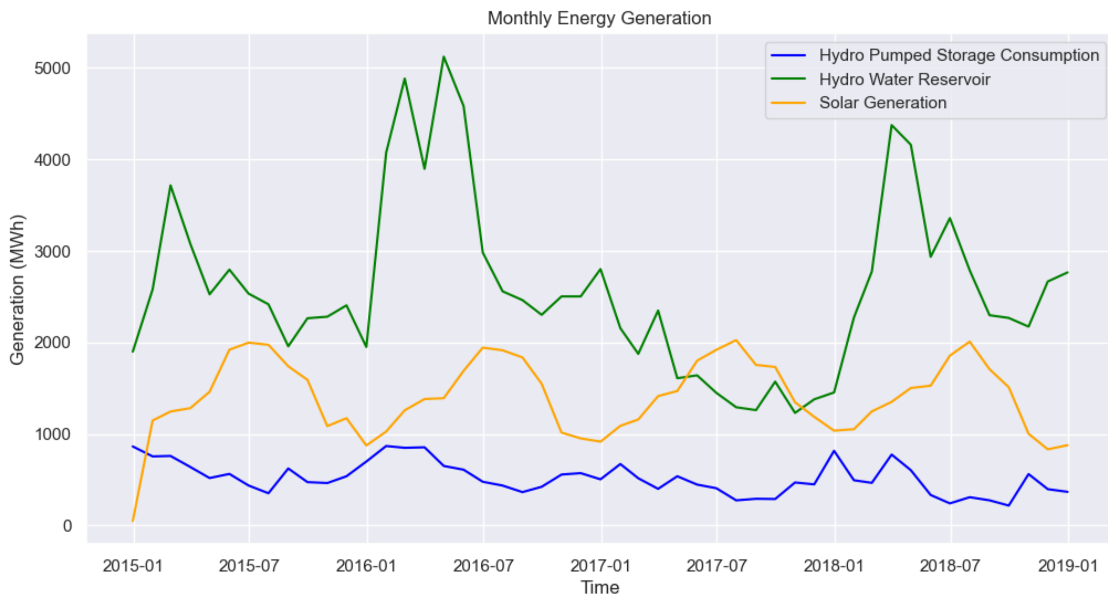


[24]:
```
correlations = correlation_matrix['total load actual'].drop('total load actual')
# Sort the correlations and get the top 5
top_6_parameters = correlations.abs().sort_values(ascending=False).head(6)

# Print the top 5 correlated parameters
print(top_6_parameters)
```

```
generation fossil gas                          0.549228
generation hydro pumped storage consumption    0.547674
generation fossil oil                          0.497627
generation hydro water reservoir               0.479624
```

```
generation fossil hard coal                      0.397552
generation solar                                 0.395592
Name: total load actual, dtype: float64
```

[25]:
```
data_2 = pd.read_csv('df_combined.csv')
data_3= data_2[['time','generation hydro pumped storage␣
↪consumption','generation hydro water reservoir','generation solar']]
#data_3.dtypes
# Ensure the 'time' column is treated as a datetime type
data_3['time'] = pd.to_datetime(data_3['time'])

# # Set 'time' as the index of the dataframe
data_3.set_index('time', inplace=True)

#data_2 .index
# # Convert generation columns to numeric, errors='coerce' will convert␣
↪non-numeric values to NaN
data_3['generation hydro pumped storage consumption'] = pd.
↪to_numeric(data_3['generation hydro pumped storage consumption'],␣
↪errors='coerce')
data_3['generation hydro water reservoir'] = pd.to_numeric(data_3['generation␣
↪hydro water reservoir'], errors='coerce')
data_3['generation solar'] = pd.to_numeric(data_3['generation solar'],␣
↪errors='coerce')

# Resample data monthly and calculate the mean for each month
monthly_data = data_3.resample('M').mean()

# Plot the resampled data
plt.figure(figsize=(12, 6))
plt.plot(monthly_data['generation hydro pumped storage consumption'],␣
↪label='Hydro Pumped Storage Consumption', color='blue')
plt.plot(monthly_data['generation hydro water reservoir'], label='Hydro Water␣
↪Reservoir', color='green')
plt.plot(monthly_data['generation solar'], label='Solar Generation',␣
↪color='orange')

# Adding title and labels
plt.title('Monthly Energy Generation')
plt.xlabel('Time')
plt.ylabel('Generation (MWh)')
plt.legend()

# Show the plot
plt.show()
# data_2.dtypes
```



As above visualisation highlights the energy generation for the sources mentioned in labels in Valencia,for our stakeholders we suggest to look into the “Hydro Pumped Storage”, “Hydro Water Reservoir” and “Solar Generation” renewable energy sources for investment in sustainable energy generation. If provided with similar data from utility companies and energy generation plants, our model could forecast the energy consumption and we could also suggest for different renewable source based on data analysis of provided data. For efficient forecasting we would ask stakeholders to provide additional following data: 1)Area Population Data. 2)Energy consumption history for appliances used in the region. 3)Utility Bills Pricing Data 4)Additional Data Points