Big Data Analytics for Short Term Load Forecasting using Hybrid Artificial Neural Network

Import Libraries

In [92]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import datetime
from datetime import date
import holidays
import pickle
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
plt.style.use('bmh')
# Above is a special style template for matplotlib, highly useful for visualizing time series data
%matplotlib inline
from pylab import rcParams
import statsmodels.api as sm

from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf
from tensorflow.keras.layers import Conv1D, LSTM, Dense, Dropout, Bidirectional, TimeDistributed
from tensorflow.keras.layers import MaxPooling1D, Flatten
from tensorflow.keras.metrics import RootMeanSquaredError
from keras.metrics import MeanSquaredError, MeanAbsoluteError, MeanAbsolutePercentageError
from tensorflow.keras.utils import plot_model
from tensorflow.keras.regularizers import L1
from sklearn.metrics import r2_score, explained_variance_score

from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

# pip install holidays

warnings.filterwarnings("ignore")
```

Data Preprocessing

Dataset contains the load data from January 2017 to October 2022

Load Data Preprocessing

In [2]:
```
load_data = pd.read_excel("2017-2022.xlsx", sheet_name="Sheet1")
load_data
```

Out[2]:

	datetime	demand load	sldc load	scada load	od/ud sldc load	od/ud scada load
0	01-01-2017 00:00:00	635.0	710.0	756.0	75.0	121.0
1	01-01-2017 00:15:00	730.0	682.0	730.0	-48.0	0.0
2	01-01-2017 00:30:00	686.0	660.0	701.0	-26.0	15.0
3	01-01-2017 00:45:00	645.0	641.0	674.0	-4.0	29.0
4	01-01-2017 01:00:00	628.0	620.0	650.0	-8.0	22.0
...	...	...	...	...	...	...
204475	31-10-2022 22:45:00	1053.0	1032.0	1032.0	-21.0	-21.0
204476	31-10-2022 23:00:00	1039.0	1008.0	1007.0	-31.0	-32.0
204477	31-10-2022 23:15:00	1013.0	984.0	982.0	-29.0	-31.0
204478	31-10-2022 23:30:00	963.0	957.0	957.0	-6.0	-6.0
204479	31-10-2022 23:45:00	969.0	936.0	933.0	-33.0	-36.0

204480 rows × 6 columns

In [3]:
```
load_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204480 entries, 0 to 204479
Data columns (total 6 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   datetime          204480 non-null  object
```

```
 1   demand load       202738 non-null  float64
 2   sldc load         202738 non-null  float64
 3   scada load        202738 non-null  float64
 4   od/ud sldc load   204480 non-null  float64
 5   od/ud scada load  204480 non-null  float64
dtypes: float64(5), object(1)
memory usage: 9.4+ MB
```

In [4]:
```
load_data.describe()
```

Out[4]:

	demand load	sldc load	scada load	od/ud sldc load	od/ud scada load
count	202738.000000	202738.000000	202738.000000	204480.000000	204480.000000
mean	1543.773984	1534.804437	1517.551100	-8.893134	-25.999487
std	527.918990	525.863416	531.549262	55.338041	79.380589
min	399.000000	400.000000	449.000000	-298.000000	-299.000000
25%	1170.000000	1163.000000	1127.000000	-38.000000	-55.000000
50%	1496.000000	1483.000000	1473.000000	-7.500000	-13.000000
75%	1924.000000	1912.000000	1890.000000	21.000000	17.000000
max	3455.000000	3345.000000	3405.000000	299.000000	299.000000

In [5]:
```
load_data.isnull().sum()
```

Out[5]:
```
datetime              0
demand load        1742
sldc load          1742
scada load         1742
od/ud sldc load       0
od/ud scada load      0
dtype: int64
```

To handling the missing value, spline Interpolation is used to estimating unknown data points in a given range.

Interpolation of DEMAND, SLDC and SCADA load data

In [6]:
```
load_data['DEMAND LOAD'] = load_data['demand load'].interpolate(option='spline',
                                                                order=5,
                                                                limit_direction='both')

load_data['SLDC LOAD'] = load_data['sldc load'].interpolate(option='spline',
                                                            order=5,
                                                            limit_direction='both')

load_data['SCADA LOAD'] = load_data['scada load'].interpolate(option='spline',
                                                              order=5,
                                                              limit_direction='both')
```

In [7]:
```
load_data = pd.DataFrame([load_data['datetime'],
                          load_data['DEMAND LOAD'],
                          load_data['SLDC LOAD'],
                          load_data['SCADA LOAD']]).transpose()
load_data
```

Out[7]:

	datetime	DEMAND LOAD	SLDC LOAD	SCADA LOAD
0	01-01-2017 00:00:00	635.0	710.0	756.0
1	01-01-2017 00:15:00	730.0	682.0	730.0
2	01-01-2017 00:30:00	686.0	660.0	701.0
3	01-01-2017 00:45:00	645.0	641.0	674.0
4	01-01-2017 01:00:00	628.0	620.0	650.0
...	...	...	...	...
204475	31-10-2022 22:45:00	1053.0	1032.0	1032.0
204476	31-10-2022 23:00:00	1039.0	1008.0	1007.0
204477	31-10-2022 23:15:00	1013.0	984.0	982.0
204478	31-10-2022 23:30:00	963.0	957.0	957.0
204479	31-10-2022 23:45:00	969.0	936.0	933.0

204480 rows × 4 columns

In [8]:
```
load_data.isnull().sum()
```

Out[8]:
```
datetime       0
DEMAND LOAD    0
SLDC LOAD      0
SCADA LOAD     0
dtype: int64
```

Load Dataset

In [9]:
```
final_load_dataset = pd.DataFrame([load_data['DEMAND LOAD'],
                                   load_data['SLDC LOAD'],
                                   load_data['SCADA LOAD']]).transpose()

final_load_dataset.index = pd.to_datetime(load_data['datetime'].astype(str))
final_load_dataset = final_load_dataset.rename_axis('datetime').reset_index()
final_load_dataset
```

Out[9]:

	datetime	DEMAND LOAD	SLDC LOAD	SCADA LOAD
0	2017-01-01 00:00:00	635.0	710.0	756.0
1	2017-01-01 00:15:00	730.0	682.0	730.0
2	2017-01-01 00:30:00	686.0	660.0	701.0
3	2017-01-01 00:45:00	645.0	641.0	674.0
4	2017-01-01 01:00:00	628.0	620.0	650.0
...	...	...	...	...
204475	2022-10-31 22:45:00	1053.0	1032.0	1032.0
204476	2022-10-31 23:00:00	1039.0	1008.0	1007.0
204477	2022-10-31 23:15:00	1013.0	984.0	982.0
204478	2022-10-31 23:30:00	963.0	957.0	957.0
204479	2022-10-31 23:45:00	969.0	936.0	933.0

204480 rows × 4 columns

In [10]:
```
final_load_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204480 entries, 0 to 204479
Data columns (total 4 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   datetime     204480 non-null  datetime64[ns]
 1   DEMAND LOAD  204480 non-null  float64       
 2   SLDC LOAD    204480 non-null  float64       
 3   SCADA LOAD   204480 non-null  float64       
dtypes: datetime64[ns](1), float64(3)
memory usage: 6.2 MB
```

In [11]:
```
final_load_dataset.describe()
```

Out[11]:

	DEMAND LOAD	SLDC LOAD	SCADA LOAD
count	204480.000000	204480.000000	204480.000000
mean	1543.867527	1534.892163	1517.803775
std	527.497234	525.373783	531.071648
min	399.000000	400.000000	449.000000
25%	1170.000000	1163.000000	1127.000000
50%	1497.000000	1485.000000	1475.000000
75%	1924.000000	1912.000000	1890.000000
max	3455.000000	3345.000000	3405.000000

Weather Data Preprocessing

In [12]:
```
weather_data = pd.read_csv("2017-01-01 to 2022-10-31.csv")
weather_data.head()
```

Out[12]:

	name	datetime	temp	feelslike	dew	humidity	precip	precipprob	preciptype	snow	...	sealevelpressure	cloudcover	visibility	solarradiation	so
0	G7R6 P5G, Balaji Estate, Kalkaji, New	2017-01-01T00:00:00	13.0	13.0	12.0	93.65	0.0	0	NaN	0.0	...	NaN	0.0	1.0	0.0	

	name	datetime	temp	feelslike	dew	humidity	precip	precipprob	preciptype	snow	...	sealevelpressure	cloudcover	visibility	solarradiation	so
	Delhi, D...															
1	G7R6 P5G, Balaji Estate, Kalkaji, New Delhi, D...	2017-01-01T01:00:00	13.0	13.0	12.0	93.65	0.0	0	NaN	0.0	...	NaN	0.0	1.0	0.0	
2	G7R6 P5G, Balaji Estate, Kalkaji, New Delhi, D...	2017-01-01T02:00:00	12.2	12.2	11.7	97.01	0.0	0	NaN	0.0	...	1016.1	0.0	1.0	0.0	
3	G7R6 P5G, Balaji Estate, Kalkaji, New Delhi, D...	2017-01-01T03:00:00	12.0	12.0	11.0	93.60	0.0	0	NaN	0.0	...	NaN	0.0	1.0	0.0	
4	G7R6 P5G, Balaji Estate, Kalkaji, New Delhi, D...	2017-01-01T04:00:00	12.0	12.0	11.0	93.60	0.0	0	NaN	0.0	...	NaN	0.0	0.0	0.0	

5 rows × 24 columns

In [13]:
```
weather_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51121 entries, 0 to 51120
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   name              51121 non-null  object 
 1   datetime          51121 non-null  object 
 2   temp              51121 non-null  float64
 3   feelslike         51121 non-null  float64
 4   dew               51121 non-null  float64
 5   humidity          51121 non-null  float64
 6   precip            51121 non-null  float64
 7   precipprob        51121 non-null  int64  
 8   preciptype        1257 non-null   object 
 9   snow              50534 non-null  float64
 10  snowdepth         50534 non-null  float64
 11  windgust          9068 non-null   float64
 12  windspeed         51121 non-null  float64
 13  winddir           51121 non-null  float64
 14  sealevelpressure  26680 non-null  float64
 15  cloudcover        51121 non-null  float64
 16  visibility        50524 non-null  float64
 17  solarradiation    51095 non-null  float64
 18  solarenergy       27787 non-null  float64
 19  uvindex           51095 non-null  float64
 20  severerisk        7063 non-null   float64
 21  conditions        51121 non-null  object 
 22  icon              51121 non-null  object 
 23  stations          51121 non-null  object 
dtypes: float64(17), int64(1), object(6)
memory usage: 9.4+ MB
```

In [14]:
```
weather_data.describe()
```

Out[14]:

	temp	feelslike	dew	humidity	precip	precipprob	snow	snowdepth	windgust	windspeed	winddir
count	51121.000000	51121.000000	51121.000000	51121.000000	51121.000000	51121.000000	50534.0	50534.0	9068.000000	51121.000000	51121.000000
mean	25.533190	27.466933	16.607885	63.499998	0.105216	1.885722	0.0	0.0	20.493913	7.613284	163.750204
std	8.290339	10.285564	6.988801	23.399803	1.615397	13.602201	0.0	0.0	11.920981	5.812156	118.928169
min	0.800000	-1.500000	-11.700000	5.030000	0.000000	0.000000	0.0	0.0	0.700000	0.000000	0.000000
25%	19.000000	19.000000	11.000000	45.270000	0.000000	0.000000	0.0	0.0	10.100000	2.800000	55.000000
50%	27.000000	27.800000	15.900000	65.490000	0.000000	0.000000	0.0	0.0	18.700000	7.600000	150.000000

	temp	feelslike	dew	humidity	precip	precipprob	snow	snowdepth	windgust	windspeed	winddir
75%	31.700000	35.600000	23.700000	83.650000	0.000000	0.000000	0.0	0.0	31.700000	11.200000	272.000000
max	47.500000	57.300000	30.000000	100.000000	91.645000	100.000000	0.0	0.0	85.300000	203.800000	360.000000

Eliminate the unwanted columns from the weather data and create a dataframe by using temperature and humidity data only.

In [15]:
```
hourly_weather_data = pd.DataFrame([weather_data['temp'],
                                    weather_data['humidity']]).transpose()

hourly_weather_data.index = pd.to_datetime(weather_data['datetime'])
hourly_weather_data
```

Out[15]:

datetime	temp	humidity
2017-01-01 00:00:00	13.0	93.65
2017-01-01 01:00:00	13.0	93.65
2017-01-01 02:00:00	12.2	97.01
2017-01-01 03:00:00	12.0	93.60
2017-01-01 04:00:00	12.0	93.60
...	...	...
2022-10-31 20:00:00	22.3	81.54
2022-10-31 21:00:00	23.0	64.72
2022-10-31 22:00:00	22.0	73.29
2022-10-31 23:00:00	20.7	84.41
2022-11-01 00:00:00	20.0	82.51

51121 rows × 2 columns

Checking for null values in temperature and humidity data

In [16]:
```
hourly_weather_data.isnull().sum()
```

Out[16]:
```
temp        0
humidity    0
dtype: int64
```

In [17]:
```
hourly_weather_data.describe()
```

Out[17]:

	temp	humidity
count	51121.000000	51121.000000
mean	25.533190	63.499998
std	8.290339	23.399803
min	0.800000	5.030000
25%	19.000000	45.270000
50%	27.000000	65.490000
75%	31.700000	83.650000
max	47.500000	100.000000

Convert the hourly data into 15 minutes slots

In [18]:
```
temperature = pd.DataFrame(hourly_weather_data.temp.resample('15Min').mean()).interpolate(method= 'spline',
                                                                                          order= 5,
                                                                                          limit_direction= 'both')

humidity = pd.DataFrame(hourly_weather_data.humidity.resample('15Min').mean()).interpolate(method= 'spline',
                                                                                           order= 5,
                                                                                           limit_direction= 'both')
```

In [19]:
```
minutes_weather_data = pd.DataFrame([temperature['temp'], humidity['humidity']]).transpose()
minutes_weather_data.reset_index()
minutes_weather_data.head()
```

Out[19]:

	temp	humidity
datetime		
2017-01-01 00:00:00	13.000000	93.650000
2017-01-01 00:15:00	12.947809	93.897439
2017-01-01 00:30:00	13.016861	94.251461
2017-01-01 00:45:00	13.059741	94.523641
2017-01-01 01:00:00	13.000000	93.650000

In [20]:
```
minutes_weather_data.describe()
```

Out[20]:

	temp	humidity
count	204481.000000	204481.000000
mean	25.533311	63.499588
std	8.240901	23.338832
min	0.800000	-6.087674
25%	19.260748	45.370000
50%	27.089904	65.490000
75%	31.558436	83.511697
max	47.500000	116.265025

Weather Dataset

In [21]:
```
final_weather_dataset = pd.DataFrame([minutes_weather_data['temp'],
                                      minutes_weather_data['humidity']]).transpose().iloc[:-1]

final_weather_dataset = final_weather_dataset.reset_index()
final_weather_dataset.head()
```

Out[21]:

	datetime	temp	humidity
0	2017-01-01 00:00:00	13.000000	93.650000
1	2017-01-01 00:15:00	12.947809	93.897439
2	2017-01-01 00:30:00	13.016861	94.251461
3	2017-01-01 00:45:00	13.059741	94.523641
4	2017-01-01 01:00:00	13.000000	93.650000

In [22]:
```
final_weather_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204480 entries, 0 to 204479
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype         
---  ------    --------------   -----         
 0   datetime  204480 non-null  datetime64[ns]
 1   temp      204480 non-null  float64       
 2   humidity  204480 non-null  float64       
dtypes: datetime64[ns](1), float64(2)
memory usage: 4.7 MB
```

In [23]:
```
final_weather_dataset.describe()
```

Out[23]:

	temp	humidity
count	204480.000000	204480.000000
mean	25.533338	63.499495
std	8.240912	23.338851
min	0.800000	-6.087674
25%	19.260734	45.370000
50%	27.089993	65.490000
75%	31.558437	83.511709
max	47.500000	116.265025

In [24]:
```
final_weather_dataset.isnull().sum()
```

Out[24]:
```
datetime    0
temp        0
humidity    0
dtype: int64
```

Migration of Load and Weather dataset for feature engineering

In [25]:
```
final_dataset = pd.DataFrame([final_weather_dataset['temp'],
                              final_weather_dataset['humidity'],
                              final_load_dataset['DEMAND LOAD'],
                              final_load_dataset['SLDC LOAD'],
                              final_load_dataset['SCADA LOAD']]).transpose()

final_dataset.index = pd.to_datetime(load_data['datetime'].astype(str), format='%d-%m-%Y %H:%M:%S')

final_dataset = final_dataset.rename_axis('DATETIME').reset_index()
final_dataset.columns = ['DATETIME', 'TEMPERATURE', 'HUMIDITY', 'DEMAND LOAD', 'SLDC LOAD', 'SCADA LOAD']
final_dataset = final_dataset[['DATETIME', 'TEMPERATURE', 'HUMIDITY', 'DEMAND LOAD', 'SLDC LOAD', 'SCADA LOAD']]
```

In [26]:
```
final_dataset.head()
```

Out[26]:

	DATETIME	TEMPERATURE	HUMIDITY	DEMAND LOAD	SLDC LOAD	SCADA LOAD
0	2017-01-01 00:00:00	13.000000	93.650000	635.0	710.0	756.0
1	2017-01-01 00:15:00	12.947809	93.897439	730.0	682.0	730.0
2	2017-01-01 00:30:00	13.016861	94.251461	686.0	660.0	701.0
3	2017-01-01 00:45:00	13.059741	94.523641	645.0	641.0	674.0
4	2017-01-01 01:00:00	13.000000	93.650000	628.0	620.0	650.0

In [27]:
```
final_dataset.shape
```

Out[27]:
```
(204480, 6)
```

In [28]:
```
final_dataset.size
```

Out[28]:
```
1226880
```

In [29]:
```
final_dataset.isnull().sum()
```

Out[29]:
```
DATETIME       0
TEMPERATURE    0
HUMIDITY       0
DEMAND LOAD    0
SLDC LOAD      0
SCADA LOAD     0
dtype: int64
```

In [30]:
```
final_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204480 entries, 0 to 204479
Data columns (total 6 columns):
 #   Column       Non-Null Count   Dtype         
---  ------       --------------   -----         
 0   DATETIME     204480 non-null  datetime64[ns]
 1   TEMPERATURE  204480 non-null  float64       
 2   HUMIDITY     204480 non-null  float64       
 3   DEMAND LOAD  204480 non-null  float64       
 4   SLDC LOAD    204480 non-null  float64       
 5   SCADA LOAD   204480 non-null  float64       
dtypes: datetime64[ns](1), float64(5)
memory usage: 9.4 MB
```

In [31]:
```
final_dataset.describe()
```

Out[31]:

	TEMPERATURE	HUMIDITY	DEMAND LOAD	SLDC LOAD	SCADA LOAD
count	204480.000000	204480.000000	204480.000000	204480.000000	204480.000000
mean	25.533338	63.499495	1543.867527	1534.892163	1517.803775
std	8.240912	23.338851	527.497234	525.373783	531.071648
min	0.800000	-6.087674	399.000000	400.000000	449.000000
25%	19.260734	45.370000	1170.000000	1163.000000	1127.000000
50%	27.089993	65.490000	1497.000000	1485.000000	1475.000000
75%	31.558437	83.511709	1924.000000	1912.000000	1890.000000
max	47.500000	116.265025	3455.000000	3345.000000	3405.000000

In [32]:
```
final_dataset.to_excel('PreprocessingData.xlsx', index=False)
```

Feature Engineering

In [33]:
```
dataset = pd.read_excel('PreprocessingData.xlsx')
dataset.head()
```

Out[33]:

	DATETIME	TEMPERATURE	HUMIDITY	DEMAND LOAD	SLDC LOAD	SCADA LOAD
0	2017-01-01 00:00:00	13.000000	93.650000	635.0	710.0	756.0
1	2017-01-01 00:15:00	12.947809	93.897439	730.0	682.0	730.0
2	2017-01-01 00:30:00	13.016861	94.251461	686.0	660.0	701.0
3	2017-01-01 00:45:00	13.059741	94.523641	645.0	641.0	674.0
4	2017-01-01 01:00:00	13.000000	93.650000	628.0	620.0	650.0

Identify Holidays to use as a feature for a model

In [34]:
```
holiday_list, list_of_years=([] for i in range(2))
year = pd.DatetimeIndex(dataset['DATETIME']).year

for i in year:
    if i not in list_of_years:
        list_of_years.append(i)

for holiday in holidays.India(years = list_of_years).items():
    holiday_list.append(holiday)

holidays_df = pd.DataFrame(holiday_list, columns=["DATE", "HOLIDAY"])
holidays_df
```

Out[34]:

	DATE	HOLIDAY
0	2017-01-14	Makar Sankranti / Pongal
1	2017-01-26	Republic Day
2	2017-08-15	Independence Day
3	2017-10-02	Gandhi Jayanti
4	2017-05-01	Labour Day
5	2017-12-25	Christmas
6	2017-10-19	Diwali
7	2017-03-13	Holi
8	2018-01-14	Makar Sankranti / Pongal
9	2018-01-26	Republic Day
10	2018-08-15	Independence Day
11	2018-10-02	Gandhi Jayanti
12	2018-05-01	Labour Day
13	2018-12-25	Christmas
14	2018-11-07	Diwali
15	2018-03-02	Holi
16	2019-01-14	Makar Sankranti / Pongal
17	2019-01-26	Republic Day
18	2019-08-15	Independence Day
19	2019-10-02	Gandhi Jayanti

	DATE	HOLIDAY
20	2019-05-01	Labour Day
21	2019-12-25	Christmas
22	2019-10-27	Diwali
23	2019-03-21	Holi
24	2020-01-14	Makar Sankranti / Pongal
25	2020-01-26	Republic Day
26	2020-08-15	Independence Day
27	2020-10-02	Gandhi Jayanti
28	2020-05-01	Labour Day
29	2020-12-25	Christmas
30	2020-11-14	Diwali
31	2020-03-09	Holi
32	2021-01-14	Makar Sankranti / Pongal
33	2021-01-26	Republic Day
34	2021-08-15	Independence Day
35	2021-10-02	Gandhi Jayanti
36	2021-05-01	Labour Day
37	2021-12-25	Christmas
38	2021-11-04	Diwali
39	2021-03-28	Holi
40	2022-01-14	Makar Sankranti / Pongal
41	2022-01-26	Republic Day
42	2022-08-15	Independence Day
43	2022-10-02	Gandhi Jayanti
44	2022-05-01	Labour Day
45	2022-12-25	Christmas
46	2022-10-24	Diwali
47	2022-03-18	Holi

In [35]:

```
date_range = pd.to_datetime(dataset['DATETIME']).dt.date
is_holiday = []

for date in date_range:
    if date in  list(holidays_df['DATE']):
        is_holiday.append([date,1])
    else:
        is_holiday.append([date,0])

is_holiday = pd.DataFrame(is_holiday)
is_holiday.columns = ["DATE", "IS_HOLIDAY"]
is_holiday
```

Out[35]:

	DATE	IS_HOLIDAY
0	2017-01-01	0
1	2017-01-01	0
2	2017-01-01	0
3	2017-01-01	0
4	2017-01-01	0
...	...	...
204475	2022-10-31	0
204476	2022-10-31	0
204477	2022-10-31	0
204478	2022-10-31	0
204479	2022-10-31	0

204480 rows × 2 columns

Creating features like Day of year, Season, Week of year, Quarter from Datetime feature

In [36]:
```
dataset['YEAR'] = pd.DatetimeIndex(dataset['DATETIME']).year
dataset['MONTH'] = pd.DatetimeIndex(dataset['DATETIME']).month
dataset['DAY'] = pd.DatetimeIndex(dataset['DATETIME']).day
dataset['DAY_OF_YEAR'] = pd.DatetimeIndex(dataset['DATETIME']).dayofyear
dataset['WEEK_OF_YEAR'] = pd.DatetimeIndex(dataset['DATETIME']).weekofyear
dataset['QUARTER'] = pd.DatetimeIndex(dataset['DATETIME']).quarter
dataset['SEASON'] = dataset.MONTH%12 // 3 + 1
dataset['IS_HOLIDAY'] = is_holiday.IS_HOLIDAY

final_set = dataset[['DATETIME', 'YEAR', 'MONTH', 'DAY', 'DAY_OF_YEAR', 'WEEK_OF_YEAR', 'QUARTER', 'SEASON', 'IS_HOLIDAY',
                     'TEMPERATURE', 'HUMIDITY', 'DEMAND LOAD', 'SLDC LOAD', 'SCADA LOAD']]
```

In [37]:
```
final_set.head()
```

Out[37]:

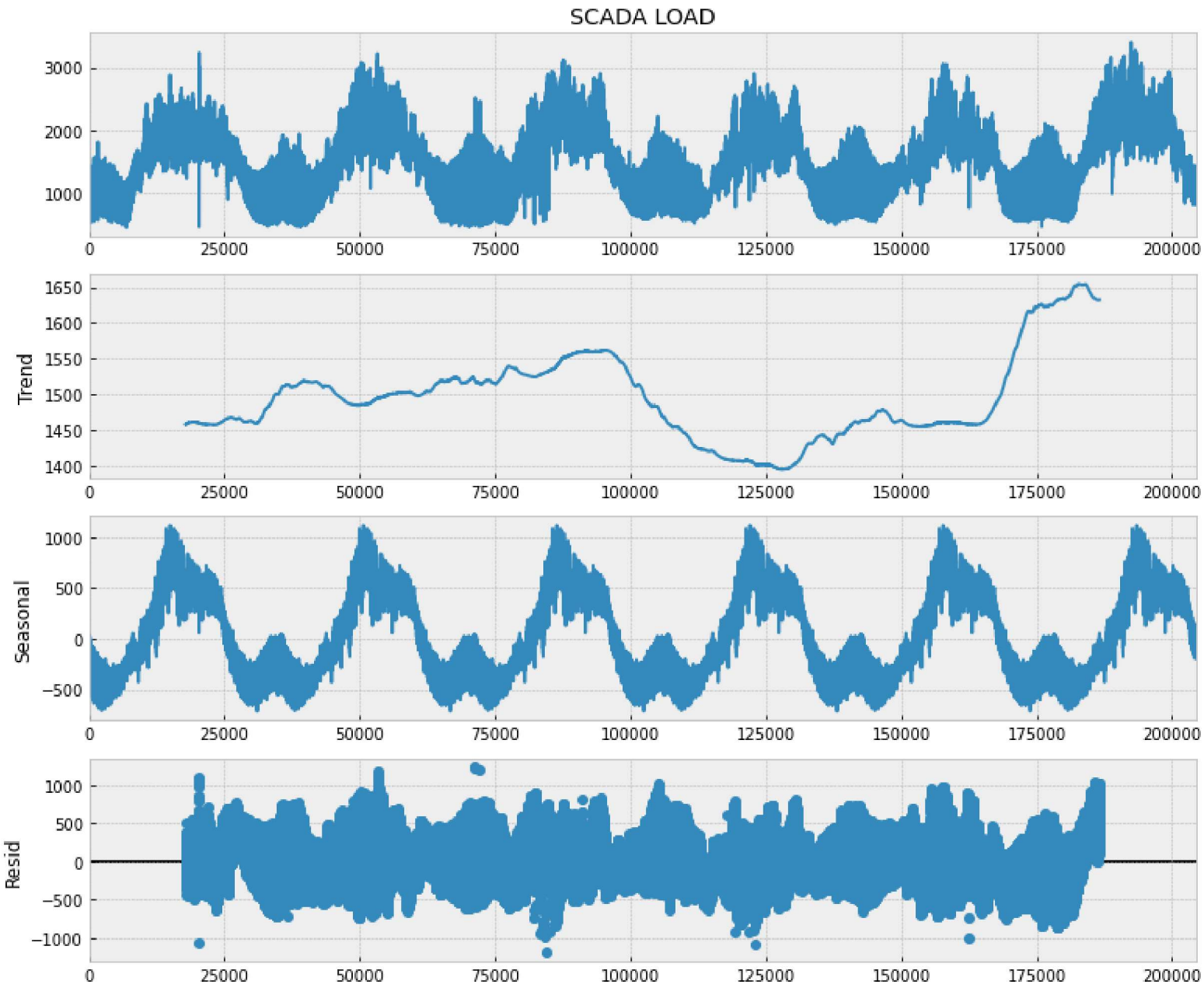
	DATETIME	YEAR	MONTH	DAY	DAY_OF_YEAR	WEEK_OF_YEAR	QUARTER	SEASON	IS_HOLIDAY	TEMPERATURE	HUMIDITY	DEMAND LOAD	SLDC LOAD	SCADA LOAD
0	2017-01-01 00:00:00	2017	1	1	1	52	1	1	0	13.000000	93.650000	635.0	710.0	756.(
1	2017-01-01 00:15:00	2017	1	1	1	52	1	1	0	12.947809	93.897439	730.0	682.0	730.(
2	2017-01-01 00:30:00	2017	1	1	1	52	1	1	0	13.016861	94.251461	686.0	660.0	701.(
3	2017-01-01 00:45:00	2017	1	1	1	52	1	1	0	13.059741	94.523641	645.0	641.0	674.(
4	2017-01-01 01:00:00	2017	1	1	1	52	1	1	0	13.000000	93.650000	628.0	620.0	650.(

In [38]:
```
final_set.to_excel('last_5_years.xlsx', index=False)
```

Exploratory Data Analysis

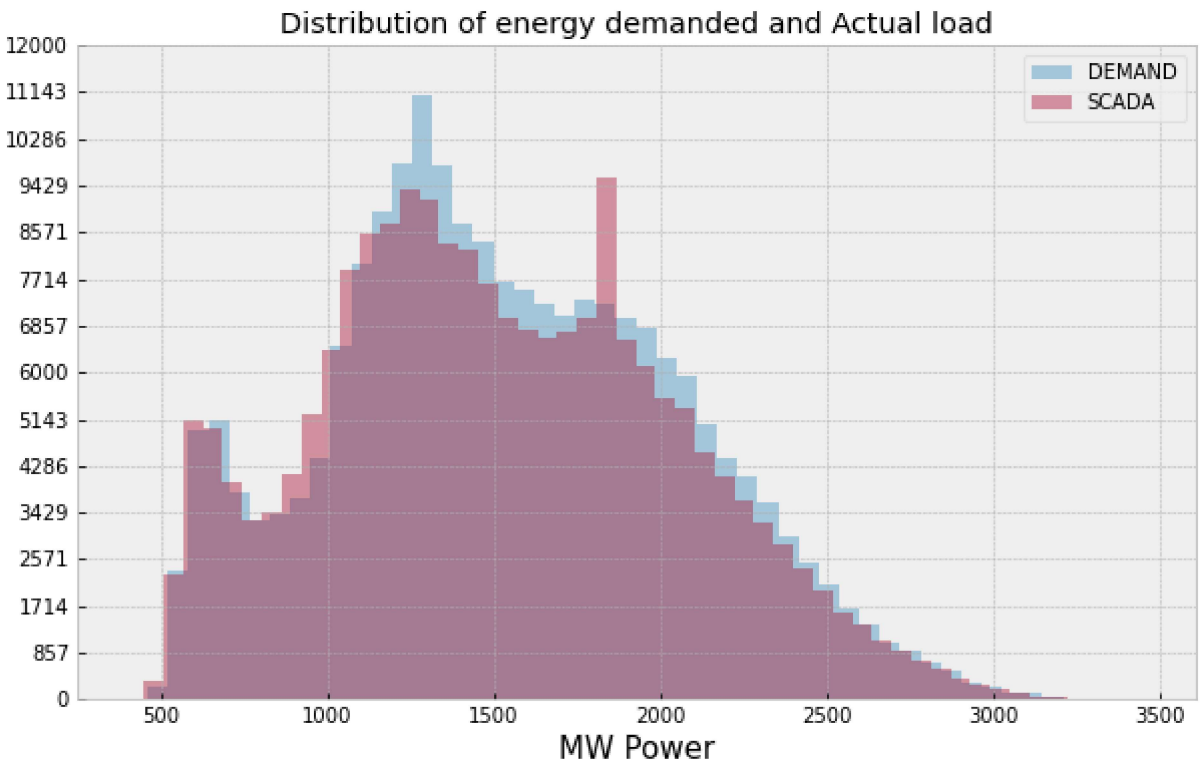
Seasonal Decompose for Load Data

In [39]:
```
rcParams['figure.figsize'] = 11, 9

decomposed_demand_load = sm.tsa.seasonal_decompose(final_set['SCADA LOAD'],freq=35700)

# THE FREQUENCY IS ANNUAL.
# 1 Days = 96 Slots
# 1 Month = 96 * 31 Days = 2976 slots approx.
# 1 Year = 2976 * 12 Months = 35712 slots approx.

figure = decomposed_demand_load.plot()
plt.show()
```



Load data distribution Between Demand and SCADA

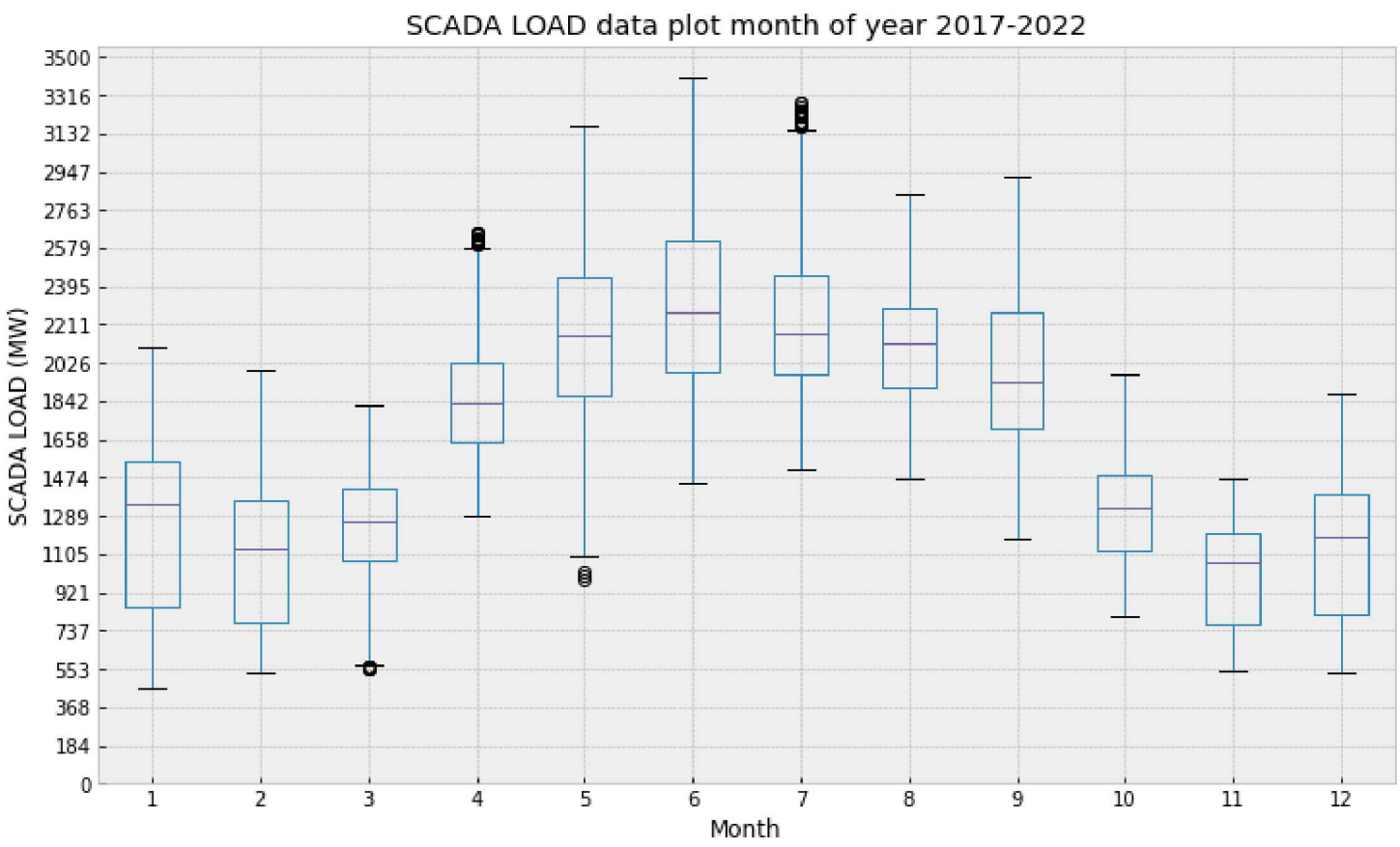
In [40]:
```
fig, ax = plt.subplots(figsize=(10, 6))
sns.distplot(final_set['DEMAND LOAD'].dropna(), ax=ax, kde=False).set_title('DEMAND LOAD', fontsize=16)
sns.distplot(final_set['SCADA LOAD'].dropna(), ax=ax, kde=False).set_title('SCADA LOAD', fontsize=16)
plt.xlabel('MW Power', fontsize=15)
plt.legend(['DEMAND', 'SCADA'])
ax.set_yticks(np.linspace(0, 12000, 15))
plt.title('Distribution of energy demanded and Actual load')
plt.show()
```



Load Data Distribution of Month of years (2017-2022)

In [41]:
```
fig, axs = plt.subplots(1, 1,  figsize=(12,7))

final_set.set_index('DATETIME', inplace=True)

groups = final_set['SCADA LOAD'].groupby(pd.Grouper(freq='M'))

df = pd.DataFrame()

for name, group in groups:
    df[name.month] = pd.Series(group.values)
```

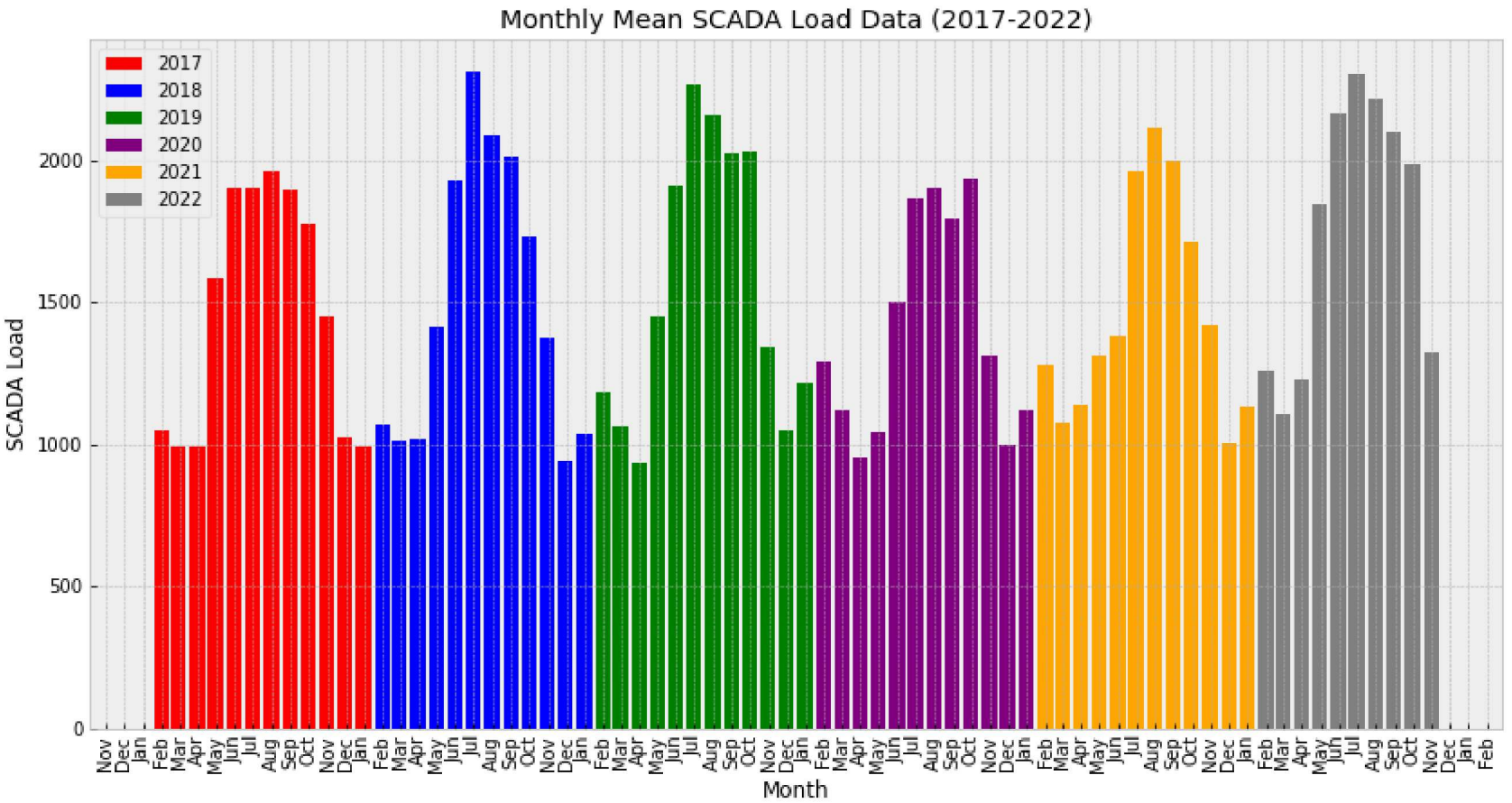
```
df.boxplot(ax=axs)
axs.set_xlabel('Month')
axs.set_ylabel('SCADA LOAD (MW)')
axs.set_title('SCADA LOAD data plot month of year 2017-2022')
axs.set_yticks(np.linspace(0,3500,20))
plt.subplots_adjust(hspace=0.5)

plt.show()
```



Monthly Mean SCADA Load Data (2017-2022)

In [42]:

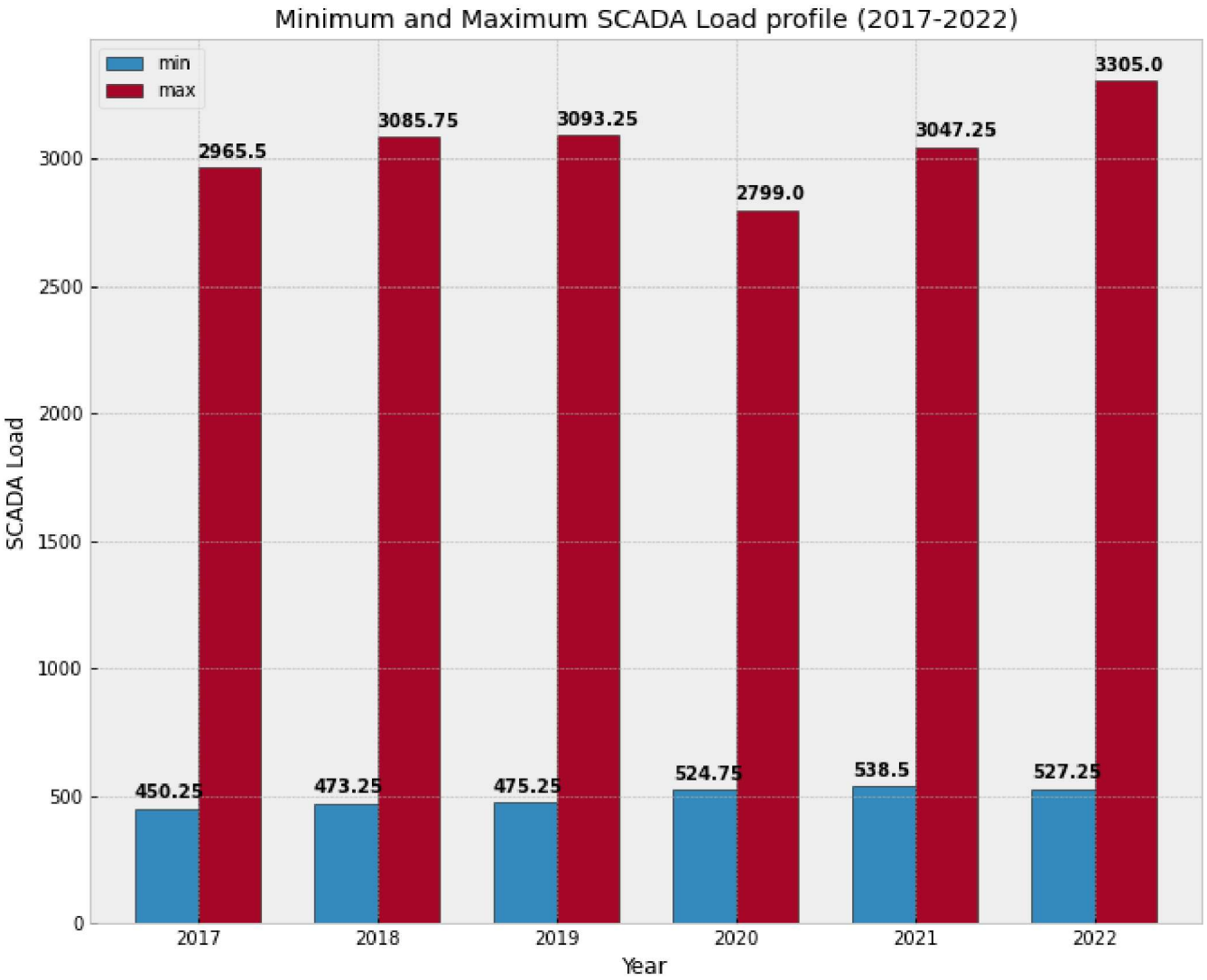
```
df_month = final_set.resample("M").mean()

fig, ax = plt.subplots(figsize=(14, 7))

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

colors = ['red', 'blue', 'green', 'purple', 'orange', 'gray']

# loop through the years and plot the bars with the corresponding color
for i, year in enumerate(range(2017, 2022+1)):
    ax.bar(df_month[str(year)].index,
           df_month.loc[str(year), "SCADA LOAD"],
           width=25, align='center',
           color=colors[i], label=str(year))

# set the X-axis tick locator to show one tick per month
ax.xaxis.set_major_locator(mdates.MonthLocator())

# set the X-axis tick formatter to show the abbreviated month name
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b'))

ax.set_xlabel('Month')
plt.xticks(rotation = 90)
ax.set_ylabel('SCADA Load')
ax.set_title('Monthly Mean SCADA Load Data (2017-2022)')
ax.legend()
plt.show()
```



Minimum and Maximum SCADA Load profile (2017-2022)

In [43]:

```
df_hourly = final_set.resample("H").mean()
df_hourly.reset_index(inplace=True)

df_hourly['DATETIME'] = df_hourly['DATETIME'].dt.year
min_max_load = df_hourly.groupby('DATETIME')['SCADA LOAD'].agg(['min', 'max'])

ax = min_max_load.plot.bar(rot=0,  width=0.7, edgecolor='black')

for i, val in enumerate(min_max_load['min']):
    ax.text(i-0.35, val+40, str(val), color='black', fontweight='bold')
for i, val in enumerate(min_max_load['max']):
    ax.text(i+0.00, val+40, str(val), color='black', fontweight='bold')


ax.set_xlabel('Year')
ax.set_title('Minimum and Maximum SCADA Load profile (2017-2022)')
ax.set_ylabel('SCADA Load')
plt.show()
```

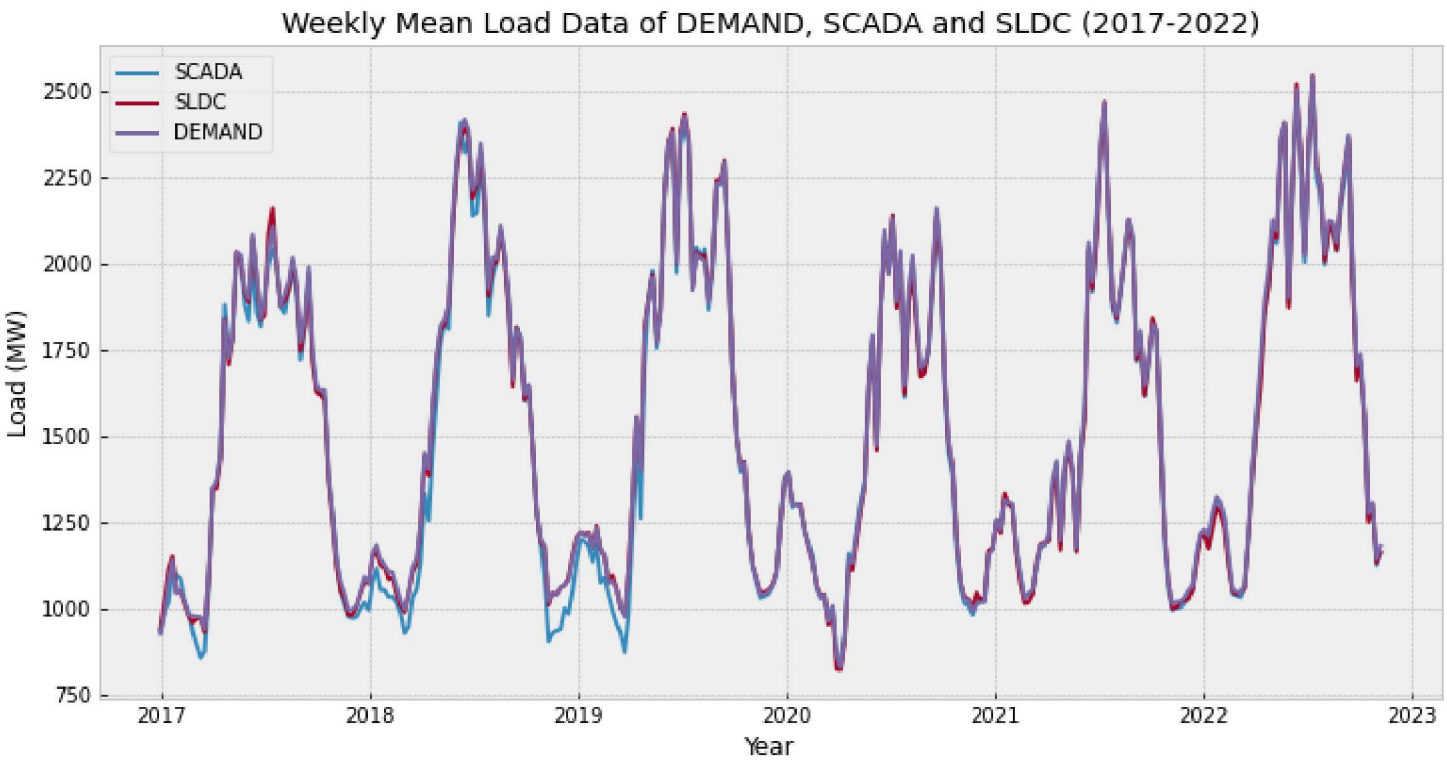


Weekly Mean Load Data of DEMAND, SCADA and SLDC (2017-2022)

In [44]:

```
SCADA_weekly = final_set['SCADA LOAD'].resample('W').mean()
SLDC_weekly = final_set['SLDC LOAD'].resample('W').mean()
```

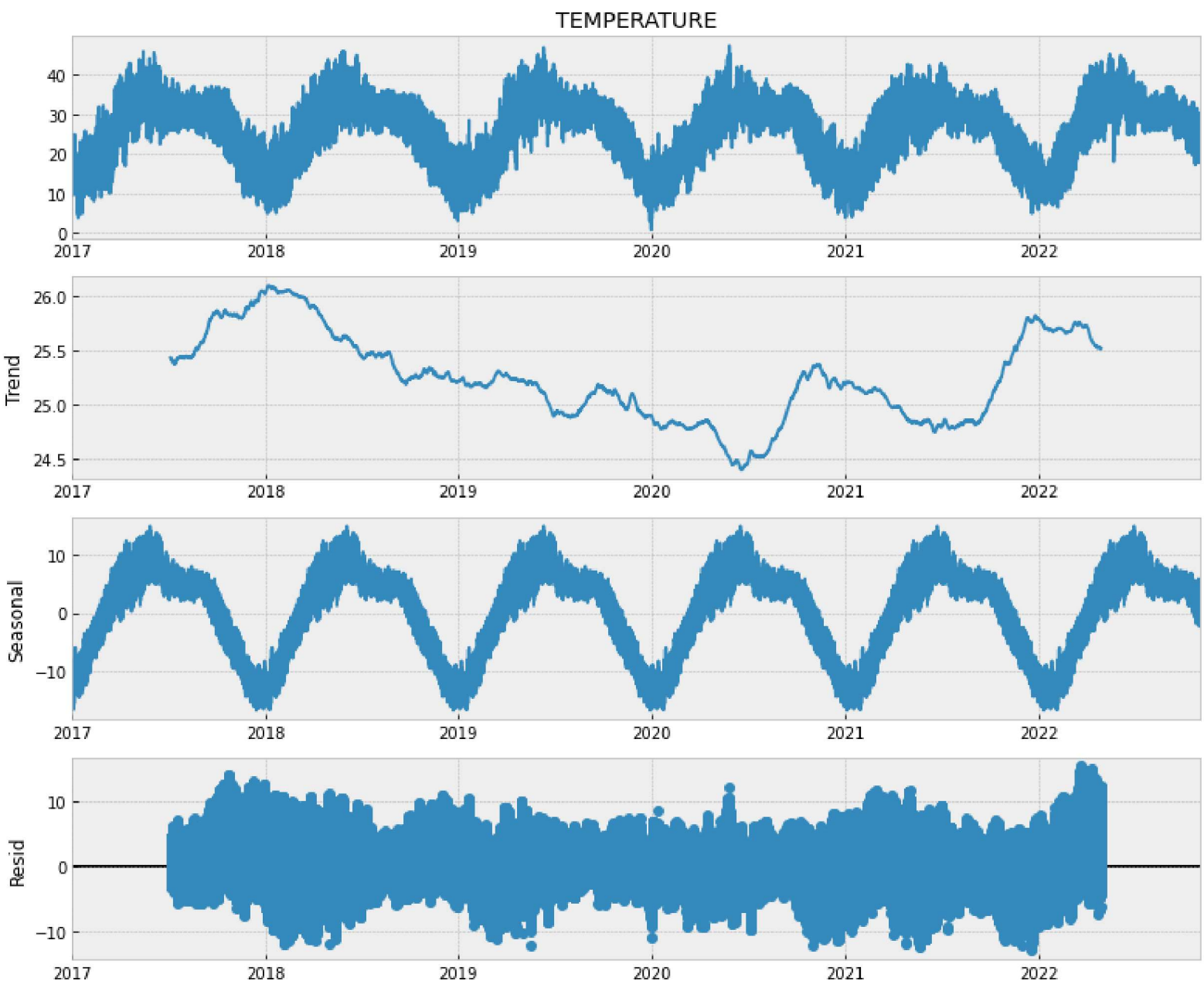
```
Demand_weekly = final_set['DEMAND LOAD'].resample('W').mean()

plt.figure(figsize=(12, 6))
sns.lineplot(data=SCADA_weekly, label='SCADA')
sns.lineplot(data=SLDC_weekly, label='SLDC')
sns.lineplot(data=Demand_weekly, label='DEMAND')
plt.xlabel('Year')
plt.ylabel('Load (MW)')
plt.title('Weekly Mean Load Data of DEMAND, SCADA and SLDC (2017-2022)')
plt.legend()
plt.show()
```



Seasonal Decompose for Weather Data

In [45]:

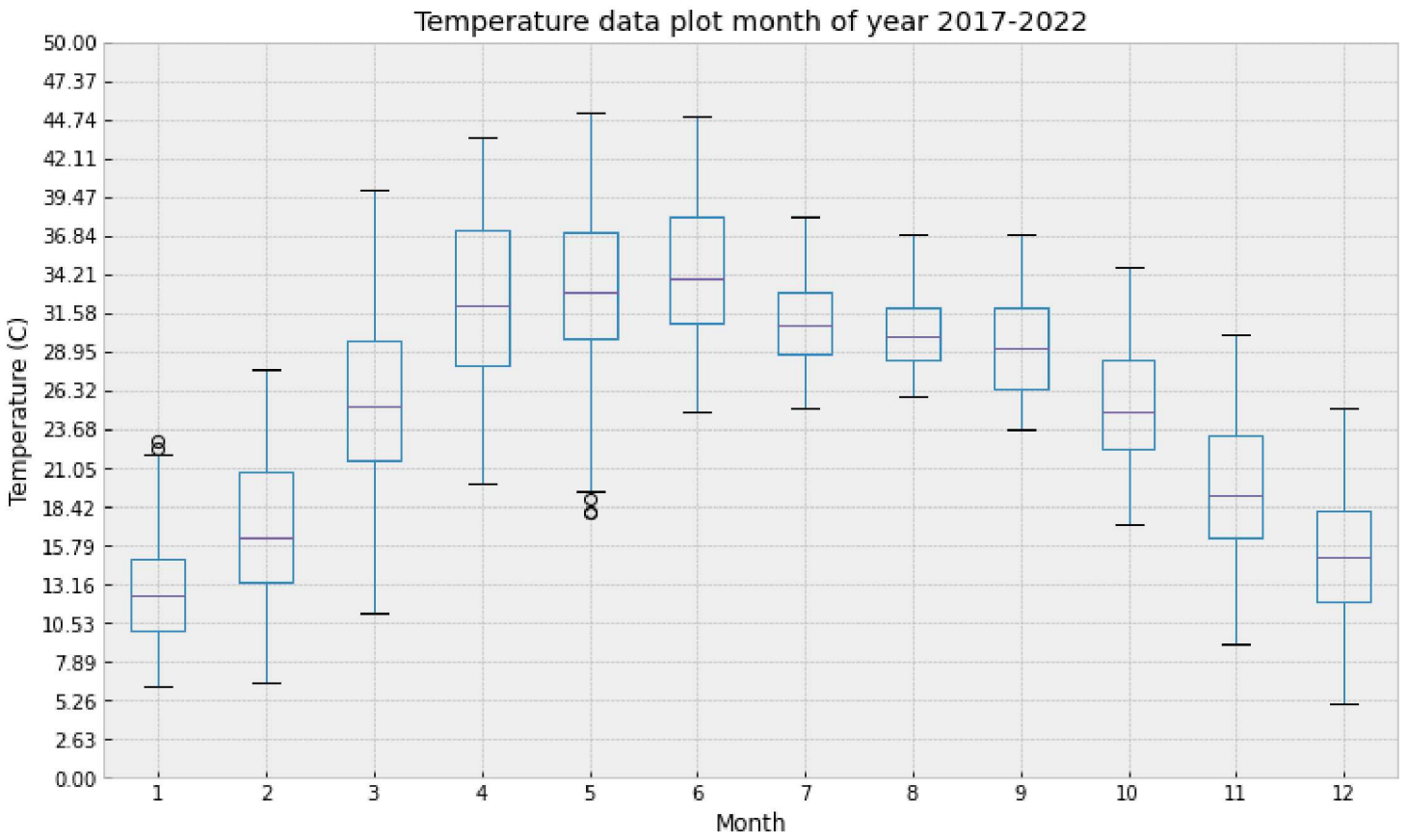
```
rcParams['figure.figsize'] = 11, 9

decomposed_demand_load = sm.tsa.seasonal_decompose(final_set['TEMPERATURE'],freq=35700)

# THE FREQUENCY IS ANNUAL.
# 1 Days = 96 Slots
# 1 Month = 96 * 31 Days = 2976 slots approx.
# 1 Year = 2976 * 12 Months = 35712 slots approx.

figure = decomposed_demand_load.plot()
plt.show()
```



Weather Data Distribution of Month of years (2017-2022)

In [87]:

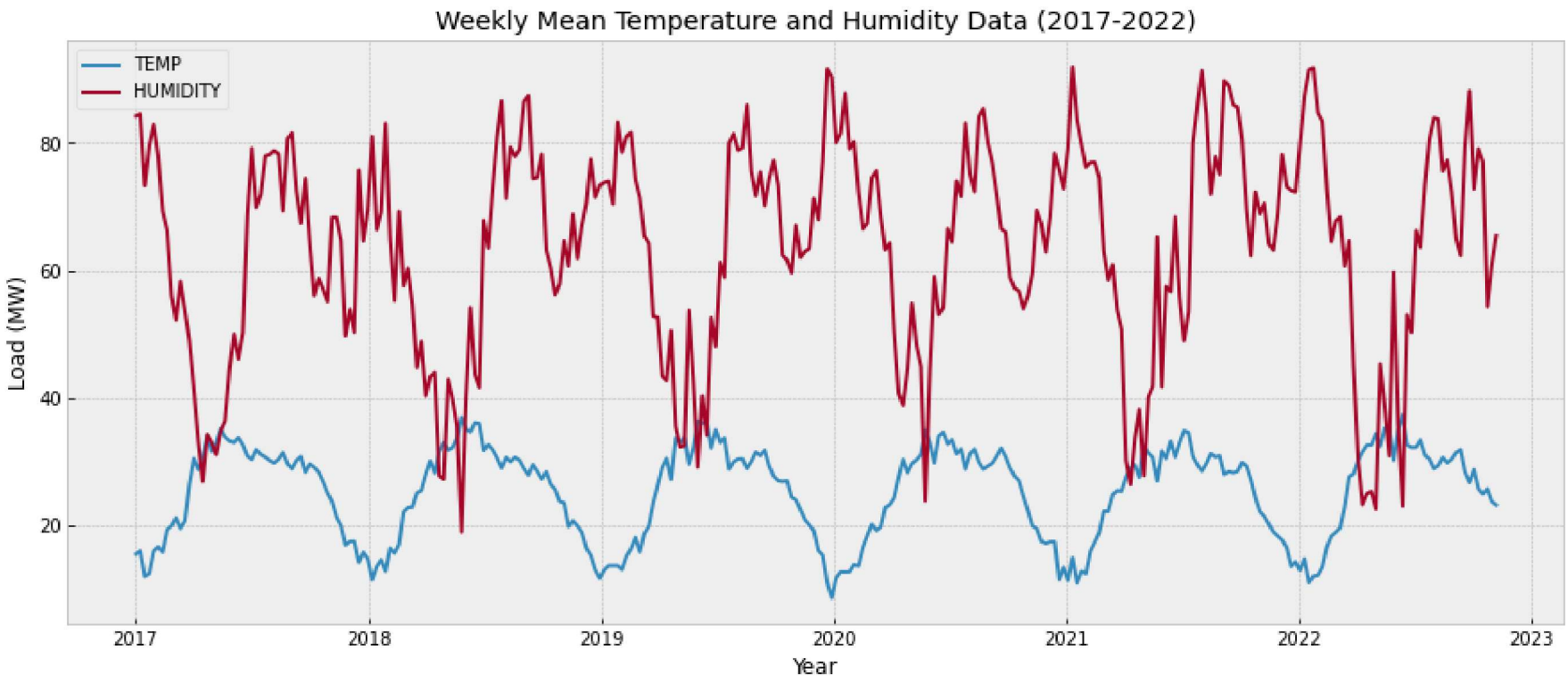
```
fig, axs = plt.subplots(1, 1,  figsize=(12,7))

groups = final_set['TEMPERATURE'].groupby(pd.Grouper(freq='M'))

df = pd.DataFrame()

for name, group in groups:
    df[name.month] = pd.Series(group.values)

df.boxplot(ax=axs)
axs.set_xlabel('Month')
axs.set_ylabel('Temperature (C)')
axs.set_title('Temperature data plot month of year 2017-2022')
axs.set_yticks(np.linspace(0,50,20))
plt.subplots_adjust(hspace=0.5)

plt.show()
```



Weekly Mean Temperature and Humidity Data (2017-2022)

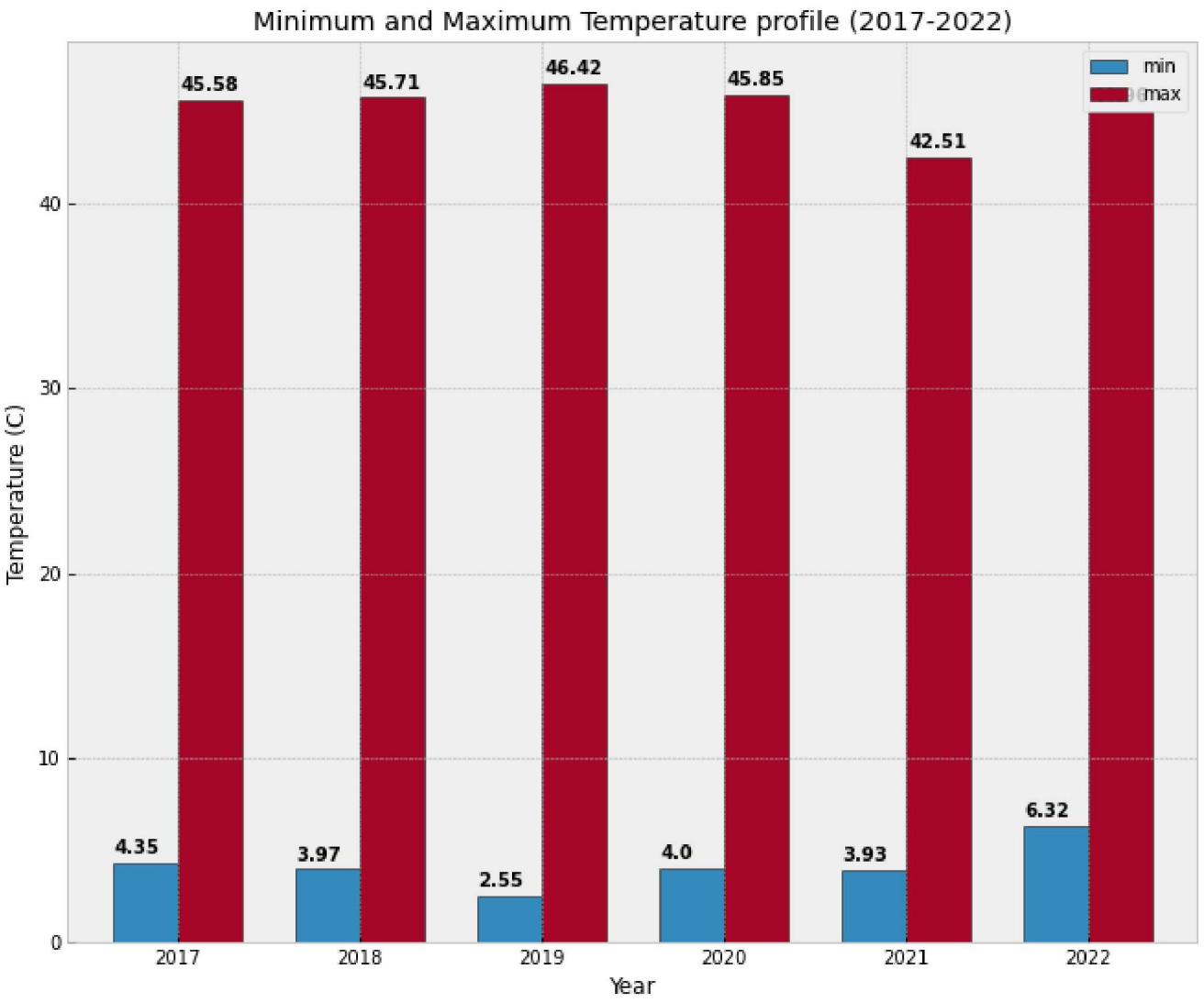
In [47]:

```
Temp_weekly = final_set.resample('W').mean()
Humd_weekly = final_set.resample('W').mean()

plt.figure(figsize=(15, 6))
sns.lineplot(data=Temp_weekly['TEMPERATURE'], label='TEMP')
sns.lineplot(data=Humd_weekly['HUMIDITY'], label='HUMIDITY')
plt.xlabel('Year')
plt.ylabel('Load (MW)')
plt.title('Weekly Mean Temperature and Humidity Data (2017-2022)')
plt.legend()
plt.show()
```



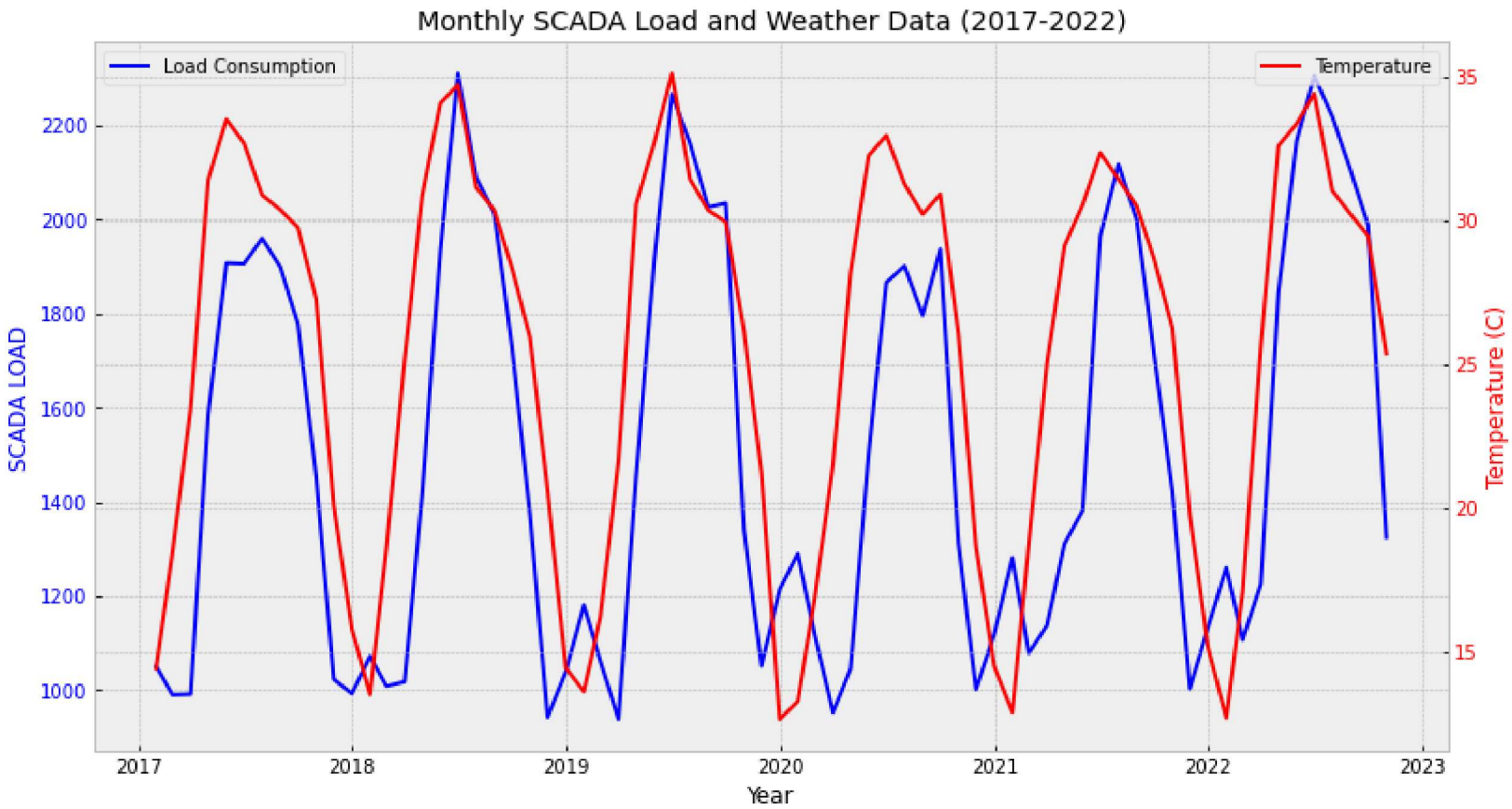
Minimum and Maximum Temperature profile (2017-2022)

In [48]:
```
df_hourly = final_set.resample("H").mean()
df_hourly.reset_index(inplace=True)

df_hourly['DATETIME'] = df_hourly['DATETIME'].dt.year
min_max_load = df_hourly.groupby('DATETIME')['TEMPERATURE'].agg(['min', 'max'])

ax = min_max_load.plot.bar(rot=0,  width=0.7, edgecolor='black')

for i, val in enumerate(min_max_load['min']):
    ax.text(i-0.35, val+0.5, str(round(val ,2)), color='black', fontweight='bold')
for i, val in enumerate(min_max_load['max']):
    ax.text(i+0.02, val+0.5, str(round(val ,2)), color='black', fontweight='bold')

ax.set_xlabel('Year')
ax.set_title('Minimum and Maximum Temperature profile (2017-2022)')
ax.set_ylabel('Temperature (C)')
plt.show()
```



Monthly SCADA Load and Weather Data (2017-2022)

In [49]:
```
load_monthly = final_set.resample('M').mean()
Temp_monthly = final_set.resample('M').mean()

fig, ax1 = plt.subplots(figsize=(13, 7))

ax1.plot(load_monthly["SCADA LOAD"], color="blue")
ax1.set_xlabel("Year")
ax1.set_ylabel("SCADA LOAD", color="blue")
ax1.tick_params(axis="y", labelcolor="blue")

ax2 = ax1.twinx()

ax2.plot(Temp_monthly["TEMPERATURE"], color="red")
ax2.set_ylabel("Temperature (C)", color="red")
ax2.tick_params(axis="y", labelcolor="red")

ax1.legend(["Load Consumption"], loc="upper left")
ax2.legend(["Temperature"], loc="upper right")

# Set the title and display the plot
plt.title("Monthly SCADA Load and Weather Data (2017-2022)")
plt.show()
```



Spliting of dataset into Train, Validation and Test Set

In [50]:
```
final_set.reset_index(inplace=True)
min_date = final_set.DATETIME.dt.date.min()
max_date = final_set.DATETIME.dt.date.max()

print('Start Date:', min_date)
print('End Date:', max_date)
```

```
Start Date: 2017-01-01
End Date: 2022-10-31
```

In [51]:
```
final_set.set_index('DATETIME', inplace=True)
```

Split data into 70% of Training set, 10% of Validation Set and 20% of Test set

Train set

In [52]:
```
train_percent = 0.7

time_between = max_date - min_date

train_start_date = min_date + pd.DateOffset(hours=0, minutes=0)
print('Start Date : ', train_start_date)

train_cutoff_date = min_date + train_percent * time_between
train_end_date = train_cutoff_date + pd.DateOffset(hours=23, minutes=45)
print('End Date   : ', train_end_date)

tsp_train_set = final_set.loc[train_start_date:train_end_date]
tsp_train_set.reset_index(inplace=True)
```

```
Start Date :  2017-01-01 00:00:00
End Date   :  2021-01-30 23:45:00
```

Validation Set

In [53]:
```
val_percent = 0.1

val_start_date = train_cutoff_date + pd.DateOffset(days=1)
print('Start Date : ', val_start_date)

val_cutoff_date = val_start_date.date() + val_percent * time_between
val_end_date = val_cutoff_date + pd.DateOffset(hours=23, minutes=45)
print('End Date   : ', val_end_date)

tsp_val_set = final_set.loc[val_start_date:val_end_date]
tsp_val_set.reset_index(inplace=True)
```

```
Start Date :  2021-01-31 00:00:00
End Date   :  2021-08-31 23:45:00
```

Test Set

In [54]:
```
test_percent = 0.2
```

```
test_start_date = val_cutoff_date + pd.DateOffset(days=1)
print('Start Date : ', test_start_date)

test_end_date = max_date + pd.DateOffset(hours=23, minutes=45)
print('End Date   : ', test_end_date)

tsp_test_set = final_set.loc[test_start_date:test_end_date]
tsp_test_set.reset_index(inplace=True)
```

```
Start Date :  2021-09-01 00:00:00
End Date   :  2022-10-31 23:45:00
```

Data Normalization

In [55]:
```
scaler = MinMaxScaler(feature_range = (0, 1))
```

In [56]:
```
tsp_scaled_train_set = tsp_train_set.drop(labels=['DATETIME'],axis=1).copy()
tsp_scaled_val_set = tsp_val_set.drop(labels=['DATETIME'],axis=1).copy()
tsp_scaled_test_set = tsp_test_set.drop(labels=['DATETIME'],axis=1).copy()
```

In [57]:
```
tsp_scaled_train_set = scaler.fit_transform(tsp_scaled_train_set)
tsp_scaled_val_set = scaler.fit_transform(tsp_scaled_val_set)
tsp_scaled_test_set = scaler.fit_transform(tsp_scaled_test_set)
```

Reshape data for input of CNN-LSTM Model

In [58]:
```
def create_sequences(ds, lb, look_back):
    windowed_dataset = []
    labels = []
    for i in range(look_back, ds.shape[0] + 1):
        windowed_dataset.append(ds[i - look_back:i])
        labels.append(lb[i - 1])

    return np.array(windowed_dataset), np.array(labels)
```

The input data can then be reshaped to have the required structure: [samples, features, timesteps]

In [59]:
```
look_back = 96
X_train, y_train = create_sequences(tsp_scaled_train_set[:,0:-1], tsp_scaled_train_set[:, -1], look_back)
X_val, y_val = create_sequences(tsp_scaled_val_set[:,0:-1], tsp_scaled_val_set[:, -1], look_back)
X_test, y_test = create_sequences(tsp_scaled_test_set[:,0:-1], tsp_scaled_test_set[:, -1], look_back)

print(X_train.shape, y_train.shape, X_val.shape, y_val.shape, X_test.shape, y_test.shape)
```

```
(143041, 96, 12) (143041,) (20353, 96, 12) (20353,) (40801, 96, 12) (40801,)
```

CNN-LSTM Model Implementation

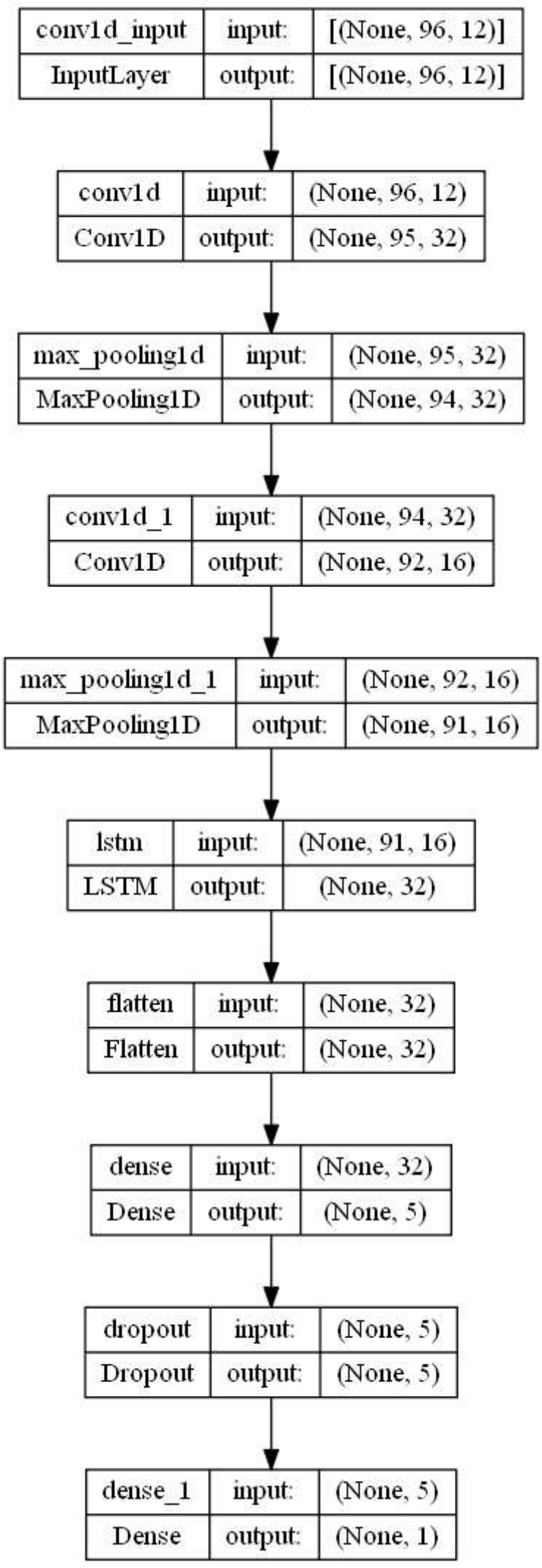
In [93]:
```
tf.keras.backend.clear_session()

# CNN Implementations
model2 = tf.keras.models.Sequential([
    tf.keras.layers.Conv1D(filters=32,
                           kernel_size=2,
                           strides=1,
                           activation="relu",
                           input_shape=(X_train.shape[1], X_train.shape[2])),
    tf.keras.layers.MaxPooling1D(pool_size=2, strides=1),
    tf.keras.layers.Conv1D(filters=16, kernel_size=3, strides=1, activation="relu"),
    tf.keras.layers.MaxPooling1D(pool_size=2, strides=1),

# LSTM Implementations
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(5, activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1, kernel_regularizer=L1(0.03))])

model2.compile(loss=tf.keras.losses.MeanSquaredError(),
               optimizer='adam',
               metrics=['mae'])

model2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 95, 32)            800

 max_pooling1d (MaxPooling1D  (None, 94, 32)           0
 )
```

```
conv1d_1 (Conv1D)           (None, 92, 16)            1552

max_pooling1d_1 (MaxPooling  (None, 91, 16)           0
1D)

lstm (LSTM)                 (None, 32)                6272

flatten (Flatten)           (None, 32)                0

dense (Dense)               (None, 5)                 165

dropout (Dropout)           (None, 5)                 0

dense_1 (Dense)             (None, 1)                 6

=================================================================
Total params: 8,795
Trainable params: 8,795
Non-trainable params: 0
_________________________________________________________________
```

In [94]:
```
plot_model(model2, to_file='model2.png', show_shapes=True, show_layer_names=True)
```

Out[94]:



In [95]:
```
log_dir2 = "logs\\" + datetime.datetime.now().strftime("%d-%m-%Y_%H%M%S")
tensorboard_callback2 = tf.keras.callbacks.TensorBoard(log_dir=log_dir2, histogram_freq=1)

tf.config.run_functions_eagerly(True)
history = model2.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=40, batch_size=100)
```

```
Epoch 1/40
1431/1431 [==============================] - 756s 524ms/step - loss: 0.0400 - mae: 0.0711 - val_loss: 0.0082 - val_mae: 0.0547
Epoch 2/40
1431/1431 [==============================] - 648s 453ms/step - loss: 0.0098 - mae: 0.0525 - val_loss: 0.0066 - val_mae: 0.0540
Epoch 3/40
1431/1431 [==============================] - 678s 474ms/step - loss: 0.0080 - mae: 0.0496 - val_loss: 0.0046 - val_mae: 0.0419
Epoch 4/40
1431/1431 [==============================] - 658s 460ms/step - loss: 0.0061 - mae: 0.0475 - val_loss: 0.0039 - val_mae: 0.0412
Epoch 5/40
1431/1431 [==============================] - 663s 463ms/step - loss: 0.0056 - mae: 0.0462 - val_loss: 0.0032 - val_mae: 0.0350
Epoch 6/40
1431/1431 [==============================] - 685s 479ms/step - loss: 0.0052 - mae: 0.0454 - val_loss: 0.0045 - val_mae: 0.0449
Epoch 7/40
1431/1431 [==============================] - 676s 472ms/step - loss: 0.0050 - mae: 0.0451 - val_loss: 0.0031 - val_mae: 0.0376
Epoch 8/40
1431/1431 [==============================] - 618s 432ms/step - loss: 0.0049 - mae: 0.0447 - val_loss: 0.0035 - val_mae: 0.0405
Epoch 9/40
1431/1431 [==============================] - 633s 442ms/step - loss: 0.0049 - mae: 0.0450 - val_loss: 0.0031 - val_mae: 0.0366
Epoch 10/40
1431/1431 [==============================] - 640s 447ms/step - loss: 0.0048 - mae: 0.0446 - val_loss: 0.0032 - val_mae: 0.0382
Epoch 11/40
1431/1431 [==============================] - 585s 409ms/step - loss: 0.0048 - mae: 0.0447 - val_loss: 0.0020 - val_mae: 0.0288
Epoch 12/40
1431/1431 [==============================] - 677s 473ms/step - loss: 0.0047 - mae: 0.0446 - val_loss: 0.0033 - val_mae: 0.0399
Epoch 13/40
1431/1431 [==============================] - 635s 444ms/step - loss: 0.0047 - mae: 0.0443 - val_loss: 0.0033 - val_mae: 0.0393
Epoch 14/40
1431/1431 [==============================] - 628s 439ms/step - loss: 0.0047 - mae: 0.0444 - val_loss: 0.0033 - val_mae: 0.0385
Epoch 15/40
1431/1431 [==============================] - 608s 425ms/step - loss: 0.0046 - mae: 0.0442 - val_loss: 0.0034 - val_mae: 0.0402
Epoch 16/40
1431/1431 [==============================] - 597s 417ms/step - loss: 0.0046 - mae: 0.0441 - val_loss: 0.0037 - val_mae: 0.0415
Epoch 17/40
1431/1431 [==============================] - 606s 423ms/step - loss: 0.0046 - mae: 0.0442 - val_loss: 0.0018 - val_mae: 0.0277
Epoch 18/40
1431/1431 [==============================] - 600s 419ms/step - loss: 0.0046 - mae: 0.0445 - val_loss: 0.0049 - val_mae: 0.0478
Epoch 19/40
1431/1431 [==============================] - 600s 419ms/step - loss: 0.0046 - mae: 0.0445 - val_loss: 0.0031 - val_mae: 0.0385
Epoch 20/40
1431/1431 [==============================] - 597s 417ms/step - loss: 0.0045 - mae: 0.0440 - val_loss: 0.0031 - val_mae: 0.0388
Epoch 21/40
1431/1431 [==============================] - 574s 401ms/step - loss: 0.0046 - mae: 0.0443 - val_loss: 0.0025 - val_mae: 0.0346
Epoch 22/40
1431/1431 [==============================] - 483s 337ms/step - loss: 0.0046 - mae: 0.0442 - val_loss: 0.0039 - val_mae: 0.0451
Epoch 23/40
1431/1431 [==============================] - 482s 337ms/step - loss: 0.0046 - mae: 0.0441 - val_loss: 0.0033 - val_mae: 0.0403
Epoch 24/40
1431/1431 [==============================] - 489s 342ms/step - loss: 0.0045 - mae: 0.0438 - val_loss: 0.0038 - val_mae: 0.0433
Epoch 25/40
1431/1431 [==============================] - 478s 334ms/step - loss: 0.0045 - mae: 0.0441 - val_loss: 0.0041 - val_mae: 0.0452
Epoch 26/40
1431/1431 [==============================] - 475s 332ms/step - loss: 0.0045 - mae: 0.0441 - val_loss: 0.0040 - val_mae: 0.0434
Epoch 27/40
1431/1431 [==============================] - 487s 340ms/step - loss: 0.0045 - mae: 0.0442 - val_loss: 0.0037 - val_mae: 0.0435
Epoch 28/40
1431/1431 [==============================] - 483s 337ms/step - loss: 0.0045 - mae: 0.0439 - val_loss: 0.0029 - val_mae: 0.0378
Epoch 29/40
1431/1431 [==============================] - 478s 334ms/step - loss: 0.0045 - mae: 0.0441 - val_loss: 0.0038 - val_mae: 0.0463
Epoch 30/40
1431/1431 [==============================] - 463s 324ms/step - loss: 0.0045 - mae: 0.0440 - val_loss: 0.0032 - val_mae: 0.0405
Epoch 31/40
1431/1431 [==============================] - 464s 324ms/step - loss: 0.0045 - mae: 0.0438 - val_loss: 0.0022 - val_mae: 0.0325
Epoch 32/40
1431/1431 [==============================] - 483s 337ms/step - loss: 0.0045 - mae: 0.0440 - val_loss: 0.0020 - val_mae: 0.0303
Epoch 33/40
1431/1431 [==============================] - 489s 342ms/step - loss: 0.0045 - mae: 0.0438 - val_loss: 0.0025 - val_mae: 0.0354
Epoch 34/40
1431/1431 [==============================] - 528s 369ms/step - loss: 0.0044 - mae: 0.0436 - val_loss: 0.0032 - val_mae: 0.0414
Epoch 35/40
1431/1431 [==============================] - 566s 395ms/step - loss: 0.0045 - mae: 0.0441 - val_loss: 0.0025 - val_mae: 0.0361
Epoch 36/40
1431/1431 [==============================] - 495s 346ms/step - loss: 0.0045 - mae: 0.0438 - val_loss: 0.0035 - val_mae: 0.0419
Epoch 37/40
1431/1431 [==============================] - 478s 334ms/step - loss: 0.0044 - mae: 0.0436 - val_loss: 0.0024 - val_mae: 0.0347
Epoch 38/40
1431/1431 [==============================] - 466s 326ms/step - loss: 0.0045 - mae: 0.0439 - val_loss: 0.0020 - val_mae: 0.0320
Epoch 39/40
1431/1431 [==============================] - 518s 362ms/step - loss: 0.0045 - mae: 0.0440 - val_loss: 0.0046 - val_mae: 0.0484
Epoch 40/40
1431/1431 [==============================] - 507s 354ms/step - loss: 0.0045 - mae: 0.0437 - val_loss: 0.0021 - val_mae: 0.0322
```

Save the model

In [96]:
```
saved_model2 = pickle.dumps(model2)
model_for_5year_data_pickle = pickle.loads(saved_model2)
```

```
Keras weights file (<HDF5 file "variables.h5" (mode r+)>) saving:
...layers\conv1d
......vars
........0
........1
...layers\conv1d_1
......vars
........0
........1
...layers\dense
......vars
........0
........1
...layers\dense_1
......vars
........0
........1
...layers\dropout
......vars
...layers\flatten
......vars
...layers\lstm
......vars
...layers\lstm\cell
......vars
........0
........1
........2
...layers\max_pooling1d
......vars
...layers\max_pooling1d_1
......vars
...metrics\mean
......vars
........0
........1
...metrics\mean_metric_wrapper
......vars
........0
........1
...optimizer
......vars
........0
........1
........10
........11
........12
........13
........14
........15
........16
........17
........18
........19
........2
........20
........21
........22
........3
........4
........5
........6
........7
........8
........9
...vars
Keras model archive saving:
File Name                                             Modified             Size
config.json                                     2023-03-12 22:32:36         4210
metadata.json                                   2023-03-12 22:32:36           64
variables.h5                                    2023-03-12 22:32:37       147728
Keras model archive loading:
File Name                                             Modified             Size
config.json                                     2023-03-12 22:32:36         4210
metadata.json                                   2023-03-12 22:32:36           64
variables.h5                                    2023-03-12 22:32:36       147728
Keras weights file (<HDF5 file "variables.h5" (mode r)>) loading:
...layers\conv1d
......vars
........0
........1
...layers\conv1d_1
......vars
........0
........1
...layers\dense
......vars
........0
```

```
.........1
...layers\dense_1
......vars
.........0
.........1
...layers\dropout
......vars
...layers\flatten
......vars
...layers\lstm
......vars
...layers\lstm\cell
......vars
.........0
.........1
.........2
...layers\max_pooling1d
......vars
...layers\max_pooling1d_1
......vars
...metrics\mean
......vars
.........0
.........1
...metrics\mean_metric_wrapper
......vars
.........0
.........1
...optimizer
......vars
.........0
.........1
.........10
.........11
.........12
.........13
.........14
.........15
.........16
.........17
.........18
.........19
.........2
.........20
.........21
.........22
.........3
.........4
.........5
.........6
.........7
.........8
.........9
...vars
```

Evaluate and Predict

In [97]:
```
loss, mse = model2.evaluate(X_test, y_test)
```

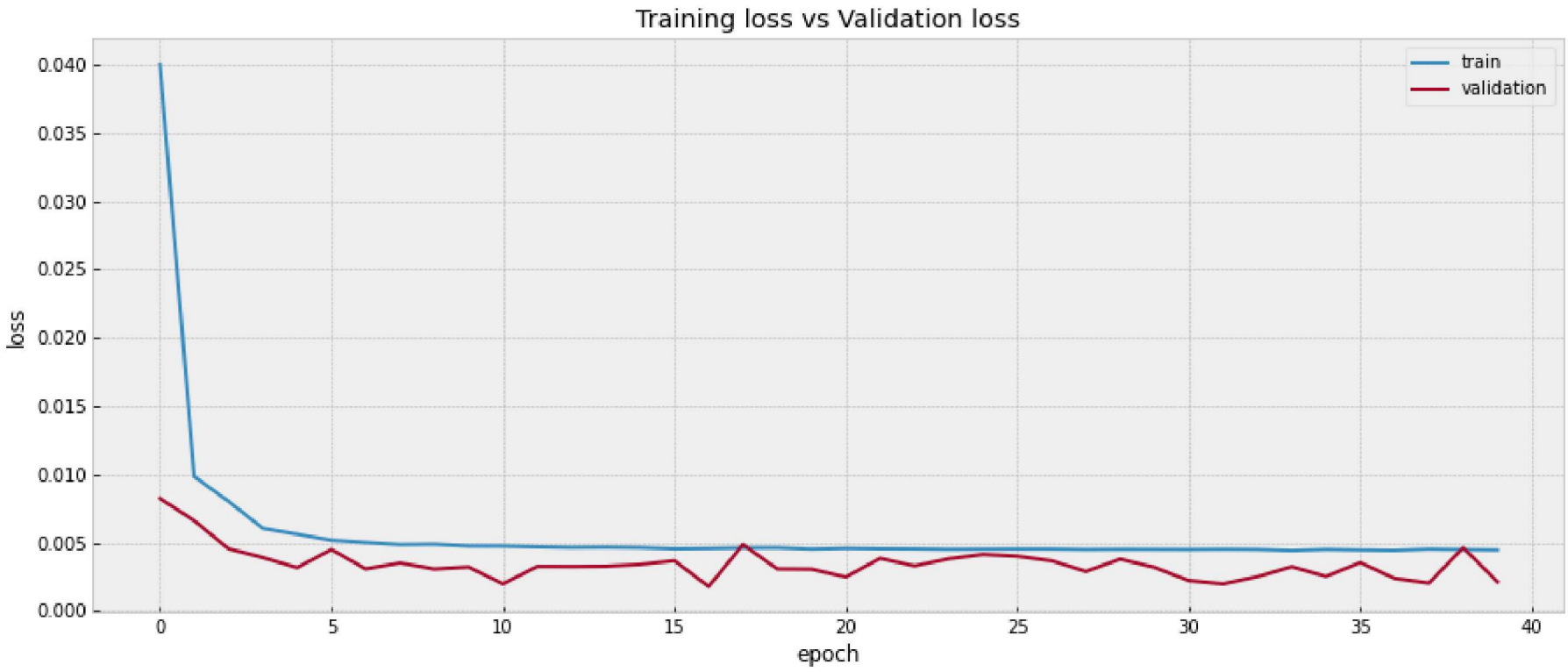
```
1276/1276 [==============================] - 179s 141ms/step - loss: 0.0011 - mae: 0.0209
```

In [98]:
```
y_pred = model2.predict(X_test)
```

```
1276/1276 [==============================] - 179s 140ms/step
```

Training Loss v/s Validation Loss

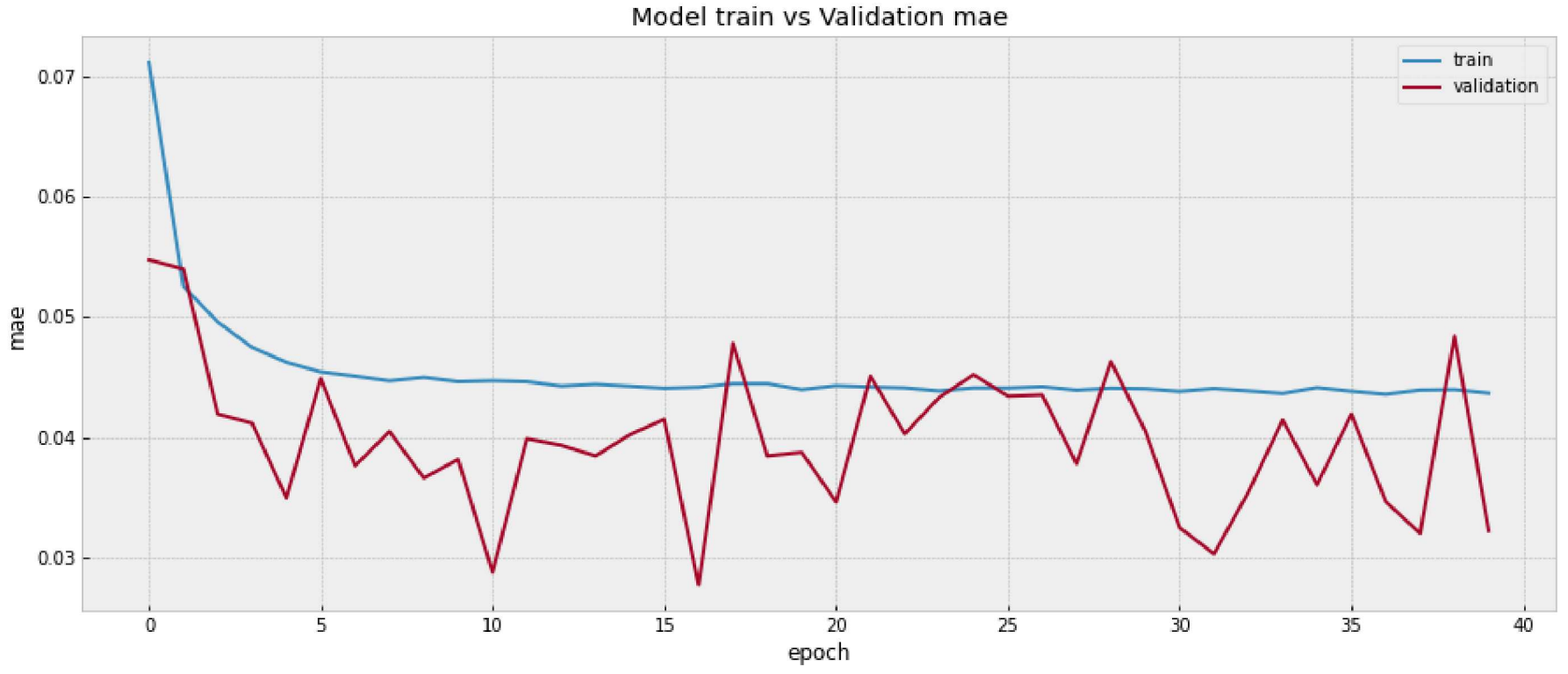
In [99]:
```
plt.figure(figsize=(15, 6))

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training loss vs Validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```



Training MAE v/s Validation MAE

In [100…

```
plt.figure(figsize=(15, 6))
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('Model train vs Validation mae')
plt.ylabel('mae')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```



Inverse Transform on test dataset to get the actual value

In [101…

```
y_test_array = np.repeat(y_test, 13, axis=-1)
y_test_array_new = y_test_array.reshape(-1, 13)
y_test_actual = scaler.inverse_transform(y_test_array_new)[:, -1]
y_test_actual
```

Out[101…

```
array([1582., 1558., 1531., ...,  982.,  957.,  933.])
```

In [102…

```
y_test_pred_array = np.repeat(y_pred, 13, axis=-1)
y_test_predict = scaler.inverse_transform(y_test_pred_array)[:, -1]
y_test_predict
```

Out[102…

```
array([1497.791 , 1519.0995, 1550.3806, ...,  908.2366,  885.0267,
        869.2805], dtype=float32)
```

Creating a dataframe to compare the actual SCADA LOAD with predicted SCADA LOAD and also check for OD/UD of predicted value

In [103…

```
actual_load_df = pd.DataFrame(y_test_actual, columns=['ACTUAL SCADA LOAD'])
pred_load_df = pd.DataFrame(y_test_predict, columns=['PRED. SCADA LOAD'])
```

In [104…

```
new_date = test_start_date + pd.DateOffset(hours=23, minutes=45)
date_index = pd.date_range(start=new_date, end=test_end_date, freq='15T')
```

```
datetime_df = pd.DataFrame({'DATETIME': date_index + pd.DateOffset(hours=0, minutes=0)})
```

In [105…

```
comp_df = pd.concat([datetime_df, actual_load_df, pred_load_df], axis=1)

comp_df['difference'] = comp_df['PRED. SCADA LOAD'].astype(int) - comp_df['ACTUAL SCADA LOAD'].astype(int)

comp_df['OD/UD PRED. SCADA LOAD'] = comp_df['difference'].apply(lambda x: '{0}{1}'.format('-' if x < 0 else '+', abs(x)))

comp_df = pd.concat([datetime_df, actual_load_df.astype(int), pred_load_df.astype(int), comp_df['OD/UD PRED. SCADA LOAD']], axis=1
comp_df
```

Out[105…

	DATETIME	ACTUAL SCADA LOAD	PRED. SCADA LOAD	OD/UD PRED. SCADA LOAD
0	2021-09-01 23:45:00	1582	1497	-85
1	2021-09-02 00:00:00	1558	1519	-39
2	2021-09-02 00:15:00	1531	1550	+19
3	2021-09-02 00:30:00	1508	1527	+19
4	2021-09-02 00:45:00	1491	1454	-37
...	...	...	...	...
40796	2022-10-31 22:45:00	1032	946	-86
40797	2022-10-31 23:00:00	1007	927	-80
40798	2022-10-31 23:15:00	982	908	-74
40799	2022-10-31 23:30:00	957	885	-72
40800	2022-10-31 23:45:00	933	869	-64

40801 rows × 4 columns

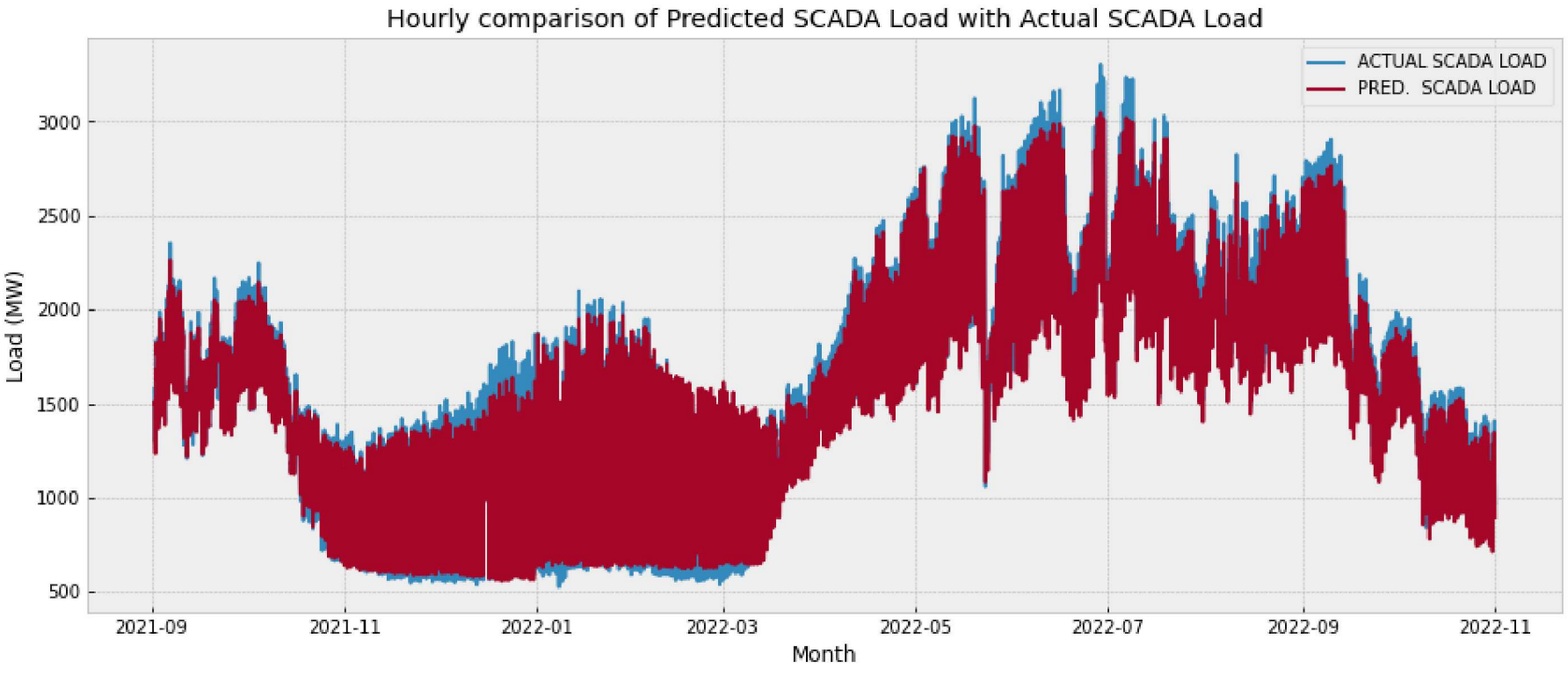
In [106…

```
comp_df.set_index('DATETIME', inplace=True)
```

Hourly data comparsion of Predicted SCADA Load with Actual SCADA Load

In [107…

```
comp_hourly = comp_df.resample("H").mean()

plt.figure(figsize=(15, 6))
sns.lineplot(data=comp_hourly['ACTUAL SCADA LOAD'], label='ACTUAL SCADA LOAD')
sns.lineplot(data=comp_hourly['PRED. SCADA LOAD'], label='PRED.  SCADA LOAD')
plt.xlabel('Month')
plt.ylabel('Load (MW)')
plt.title('Hourly comparison of Predicted SCADA Load with Actual SCADA Load')
plt.legend()
plt.show()
```

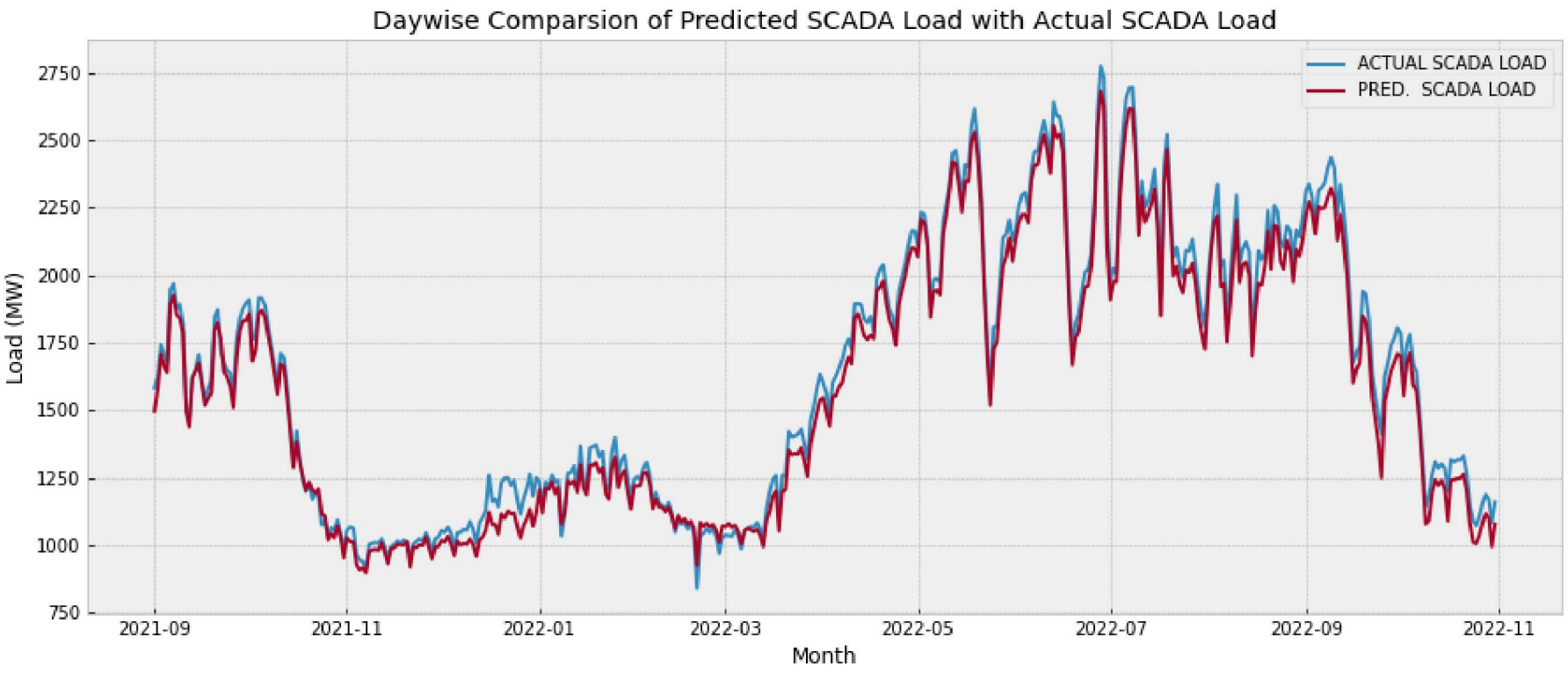


Daywise comparsion of Predicted SCADA Load with Actual SCADA Load on test data

In [108…

```
comp_hourly = comp_df.resample("D").mean()

plt.figure(figsize=(15, 6))
sns.lineplot(data=comp_hourly['ACTUAL SCADA LOAD'], label='ACTUAL SCADA LOAD')
sns.lineplot(data=comp_hourly['PRED. SCADA LOAD'], label='PRED.  SCADA LOAD')
plt.xlabel('Month')
plt.ylabel('Load (MW)')
plt.title('Daywise Comparsion of Predicted SCADA Load with Actual SCADA Load')
```

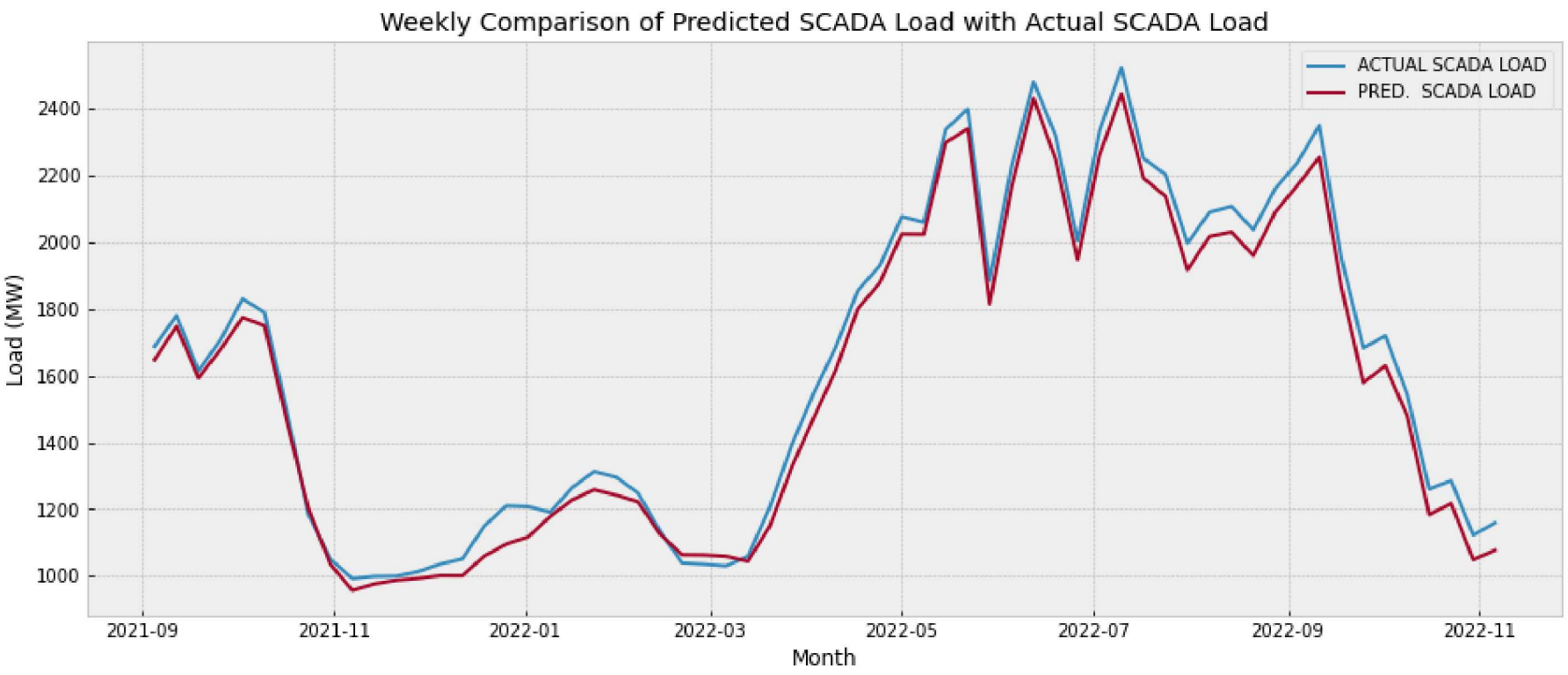
```
plt.legend()
```



Weekly data comparsion of Predicted SCADA Load with Actual SCADA Load

In [109…

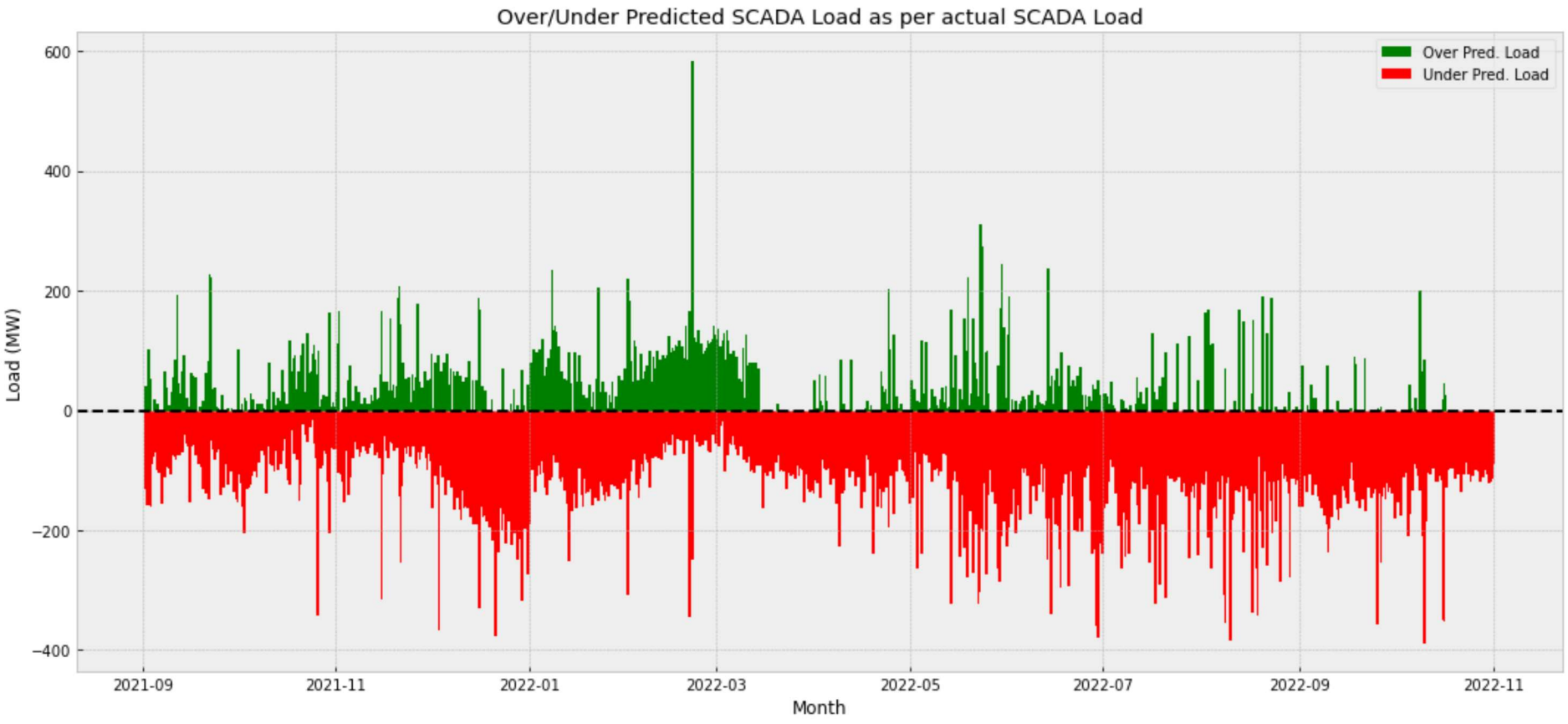
```
comp_hourly = comp_df.resample("W").mean()

plt.figure(figsize=(15, 6))
sns.lineplot(data=comp_hourly['ACTUAL SCADA LOAD'], label='ACTUAL SCADA LOAD')
sns.lineplot(data=comp_hourly['PRED. SCADA LOAD'], label='PRED.  SCADA LOAD')
plt.xlabel('Month')
plt.ylabel('Load (MW)')
plt.title('Weekly Comparison of Predicted SCADA Load with Actual SCADA Load')
plt.show()
```



Over and Under Predicted SCADA Load as per actual SCADA Load

In [110…

```
fig, ax = plt.subplots(figsize=(15, 7))

od_df = comp_df[comp_df['OD/UD PRED. SCADA LOAD'].astype(int) >= 0]
ud_df = comp_df[comp_df['OD/UD PRED. SCADA LOAD'].astype(int) < 0]

green_colors = ['g' for _ in od_df.index]
red_colors = ['r' for _ in ud_df.index]

ax.bar(od_df.index, od_df['OD/UD PRED. SCADA LOAD'].astype(int), color=green_colors, label='Over Pred. Load')
ax.bar(ud_df.index, ud_df['OD/UD PRED. SCADA LOAD'].astype(int), color=red_colors, label='Under Pred. Load')

ax.set_title('Over/Under Predicted SCADA Load as per actual SCADA Load')
ax.set_xlabel('Month')
ax.set_ylabel('Load (MW)')
ax.axhline(y=0, color='black', linestyle='--')
ax.legend(loc='upper right')

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



Evaluation Metrics

In [111…
```
def mean_absolute_percentage_error(test, pred, epsilon=1e-8):
    return np.mean(np.abs((test - pred) / (test + epsilon))) * 100

def forecasting_absolute_deviation_index(dataset):
    length_of_dataset = len(dataset)
    sum_of_value = sum(dataset)
    mean = sum_of_value / length_of_dataset
    deviations = [abs(num - mean) for num in dataset]
    return sum(deviations) / len(deviations)
```

1. Mean Squared Error

In [112…
```
mse = MeanSquaredError()
mse.update_state(y_test, y_pred)
mse_result = mse.result().numpy()
```

2. Mean Absolute Error

In [113…
```
mae = MeanAbsoluteError()
mae.update_state(y_test, y_pred)
mae_result = mae.result().numpy()
```

3. Root Mean Squared Error

In [114…
```
rmse = RootMeanSquaredError()
rmse.update_state(y_test, y_pred)
rmse_result = rmse.result().numpy()
```

4. R2 Score

In [115…
```
r2_result = r2_score(y_test, y_pred) * 100
```

5. Mean Absolute Precentage Error

In [116…
```
mape_result = mean_absolute_percentage_error(y_test_actual, y_test_predict)
```

6. Forecasting Absolute Deviation Index (FADI)

In [117…
```
comp_df = comp_df.astype(int)
dataset = list(comp_df['OD/UD PRED. SCADA LOAD'])
```

In [118…
```
fadi_result = forecasting_absolute_deviation_index(dataset)
```

Summary

In [119…
```
col_name = ['', 'MSE', 'MAE', 'RMSE', 'R2', 'MAPE', 'FADI']
metrics = ['', mse_result, mae_result, rmse_result, r2_result, mape_result, fadi_result]
```

```
metrics_table = pd.DataFrame([metrics], columns=col_name)
metrics_table = metrics_table.set_index([''])
metrics_table.style.set_caption('Model Performance on 5 Years BRPL Load Data')
```

Out[119...

Model Performance on 5 Years BRPL Load Data

MSE	MAE	RMSE	R2	MAPE	FADI
0.000640	0.020931	0.025294	98.228360	4.065958	39.493120

In [ ]: