Measuring Energy Consumption

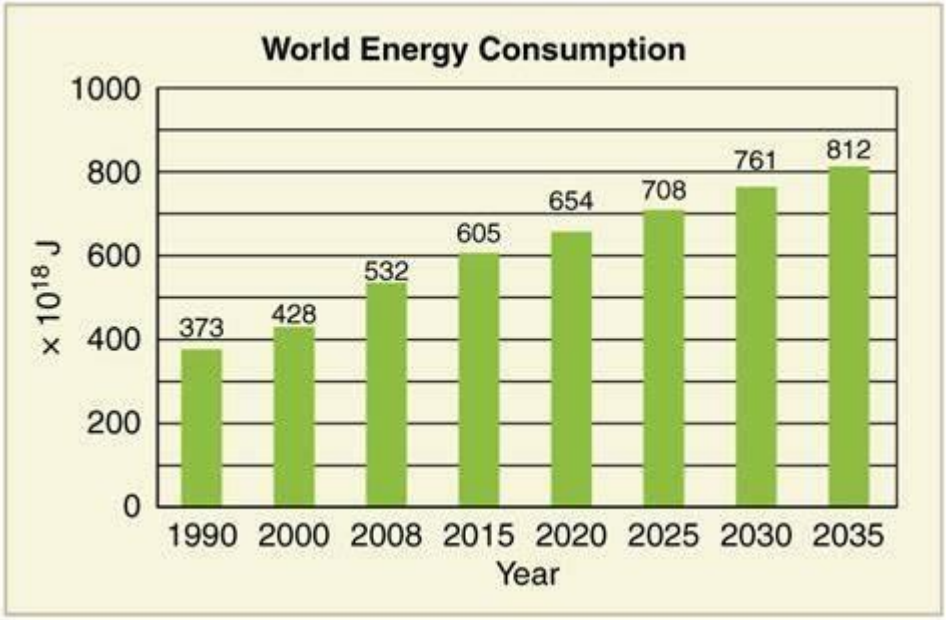
PHASE-2

Introduction

- PREDICTING FUTURE ENERGY CONSUMPTION PATTERNS IS CRUCIAL FOR EFFICIENT RESOURCE MANAGEMENT. TIME SERIES ANALYSIS AND MACHINE LEARNING OFER INNOVATIVE TECHNIQUES FOR THIS TASK:

Time series analysis

- SEASONAL DECOMPOSITION: DECOMPOSE HISTORICAL ENERGY CONSUMPTION DATA INTO ITS SEASONAL, TREND, AND RESIDUAL COMPONENTS TO UNDERSTAND RECURRING PATTERNS.
- EXPONENTIAL SMOOTHING: METHODS LIKE HOLT-WINTERS CAN CAPTURE TRENDS AND SEASONALITY IN TIME SERIES DATA.



- AUTOREGRESSIVE INTEGRATED MOVING AVERAGE (ARIMA): A WIDELY USED MODEL FOR FORECASTING BASED ON HISTORICAL VALUES, DIFFERENCING, AND LAGGED VALUES.
- FOURIER ANALYSIS: UTILIZE FOURIER TRANSFORMS TO IDENTIFY PERIODIC COMPONENTS IN ENERGY CONSUMPTION DATA.

Machine Learning Models:

- **Regression Models: Linear regression, polynomial regression, or ridge regression can be applied to capture relationships between energy consumption and various factors like temperature, day of the week, and more.**
- **Decision Trees and Random Forests: These models can handle non-linear relationships and interactions in the data.**
- **Neural Networks: Deep learning models, such as recurrent neural networks (RNNs) and long short-term memory networks (LSTMs), are effective for time series forecasting.**
- **XGBoost and Gradient Boosting: These ensemble methods excel in capturing complex patterns and are robust against overfitting.**

Feature Engineering:

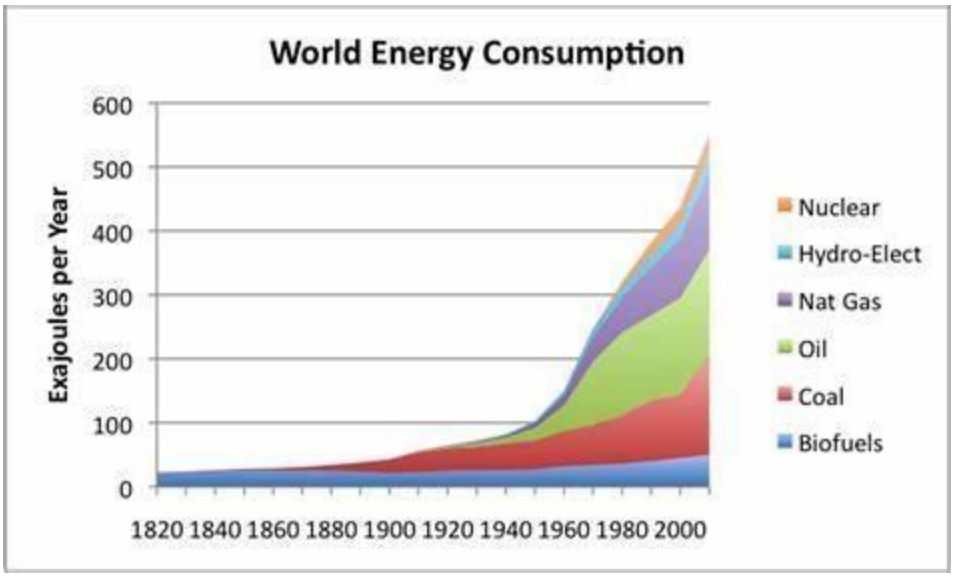
- **Create relevant features like time of day, weather conditions, holidays, and economic indicators to improve model accuracy**
- **In this section, the aforementioned feature engineering methods are applied to an illustrative dataset. Descriptions of this dataset are given in Section 3.1;**
- **the feature importance information with different feature engineering methods are discussed in Section 3.2;**
- **finally visualization of feature space is provided in Section 3.3 to give a better understanding about how the proposed feature engineering methods work**

Evaluation and Validation:

-
- **Use metrics like Mean Absolute Error (MAE), Root Mean Square Error (RMSE), or Mean Absolute Percentage Error (MAPE) to assess model performance.**
- **Employ cross-validation to validate model robustness.**

Data Preprocessing:

Clean and preprocess data to handle missing values and outliers.



- NORMALIZE OR SCALE FEATURES FOR BETTER MODEL CONVERGENCE. THE HEATING, VENTILATION AND AIR CONDITIONING (HVAC) SYSTEMS ACCOUNT FOR 40%-60% OF ENERGY CONSUMPTION IN BUILDINGS.
- OPTIMAL OPERATION CONTROL FOR HVAC SYSTEMS CAN HELP IMPROVE THE ENERGY PERFORMANCE OF THE HVAC SYSTEMS DRAMATICALLY
- ACCURATE SHORT-TERM ENERGY CONSUMPTION PREDICTION IS THE ESSENTIAL PREREQUISITE FOR DEVELOPING OPTIMAL CONTROL. WITH THE DEVELOPMENT OF COMPUTER SCIENCE AND THE INCREASING ACCESSIBILITY TO LARGE AMOUNTS OF BUILDING OPERATION DATA, DATA-DRIVEN MODELS ARE WIDELY USED TO DEVELOP ENERGY CONSUMPTION PREDICTION MODELS .
-
- DATA-DRIVEN MODELS USED FOR HVAC LOAD PREDICTION ARE GENERALLY DIVIDED INTO STATISTICAL MODELS AND MACHINE LEARNING MODELS [4].
- THE STATISTICAL MODELS CAN BE CONSTRUCTED BY REGRESSION AND TIME-SERIES ANALYSIS, WHICH USUALLY DERIVE THE INFLUENCE COEFFICIENTS OF INPUT VARIABLES FOR OUTPUTS FROM THE DATA

Ensemble Approaches:

- COMBINE MULTIPLE MODELS' PREDICTIONS THROUGH TECHNIQUES LIKE STACKING OR BLENDING FOR IMPROVED ACCURACY.

Continuous Monitoring:

- CONTINUOUSLY UPDATE MODELS WITH NEW DATA TO ADAPT TO CHANGING CONSUMPTION PATTERNS.

Domain Knowledge:

- INCORPORATE DOMAIN-SPECIFIC KNOWLEDGE TO REFINE MODELS FURTHER, SUCH AS UNDERSTANDING THE IMPACT OF GOVERNMENT POLICIES OR TECHNOLOGICAL ADVANCEMENTS.

THESE TECHNIQUES, WHEN APPLIED AND FINE-TUNED APPROPRIATELY, CAN PROVIDE VALUABLE INSIGHTS AND ACCURATE PREDICTIONS FOR FUTURE ENERGY CONSUMPTION PATTERNS, AIDING IN EFFECTIVE RESOURCE PLANNING AND SUSTAINABILITY EFFORTS.

- Energy consumption and industrial structure are synchronously adjusted and evolved, mutually related with each other.
- In the work, adjustment and evolution of energy consumption and industrial structure during 1978–2012 in China are verified and analyzed according to grey correlation and Granger causal theory.
- The result shows that industrial structure adjustment is the driving force of energy consumption structure evolution, giving one-way service to industrial structure evolution. Namely industrial structure adjustment is the driving force of energy consumption structure adjustment during 1978–2012 in China, thus making real energy economy unformed.

The objective of this article is to present the reader with a class in python that has a very intuitive and easy input to model and predict time series data using deep learning.

The data for this article can be found here:

https://www.kaggle.com/robikscube/hourly-energy-consumption

The packages that are used for deep modeling are TensorFlow and Keras.

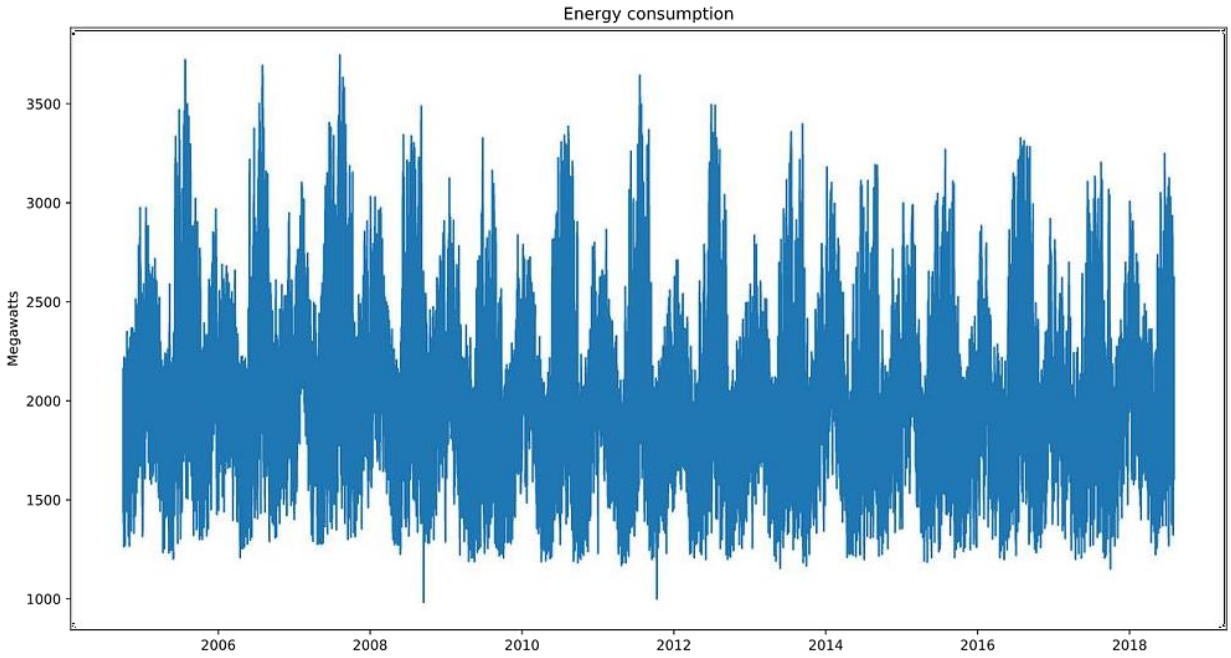
A time series is a sequence of numerical data points in successive order. These points are often measured at regular intervals (every month, every day, every hour, etc.). The data frequency used in this article is hourly and it was measured from 2004–10–01 to 2018– 08–03. The total number of raw data points is **121271**.

	Datetime	DAYTON_MW
0	2004-10-01 01:00:00	1621.0
1	2004-10-01 02:00:00	1536.0
2	2004-10-01 03:00:00	1500.0
3	2004-10-01 04:00:00	1434.0
4	2004-10-01 05:00:00	1489.0
...	...	...
121266	2018-08-02 20:00:00	2554.0
121267	2018-08-02 21:00:00	2481.0
121268	2018-08-02 22:00:00	2405.0
121269	2018-08-02 23:00:00	2250.0
121270	2018-08-03 00:00:00	2042.0

121271 rows × 2 columns

Time series example in Python

Visualization of the time series:



A line plot for the energy consumption time series

The main objective of the deep learning algorithm for a given time series is to find a function **f** such that:

$$\mathbf{Y_t = f(Y_{t-1}, Y_{t-2}, \ldots, Y_{t-p})}$$

In other words, we want to estimate a function that explains the current values of energy consumption based on **p** lags of the same energy consumption.

Sample program for Measuring Energy Consumption :

```
import pandas as pd

# Loading date wrangling package

from datetime import datetime

# Reading the input data

d = pd.read_csv('input/DAYTON_hourly.csv')

# Formating to datetime

d['Datetime'] = [datetime.strptime(x, '%Y-%m-%d %H:%M:%S') for x
in d['Datetime']]

# Making sure there are no duplicated data

# If there are some duplicates we average the data during those
duplicated days

d = d.groupby('Datetime', as_index=False)['DAYTON_MW'].mean()

# Sorting the values

d.sort_values('Datetime', inplace=True)
```

We then need a function that converts the time series into an X and Y matrices for the deep learning model to start learning. Let us say that we want to create a function that explains current time series values using **3** lags:

$$Y = f(Y_{-1}, Y_{-2}, Y_{-3})$$

And we have this data:

ts = [1621.0, 1536.0, 1500.0, 1434.0, 1489.0, 1620.0]

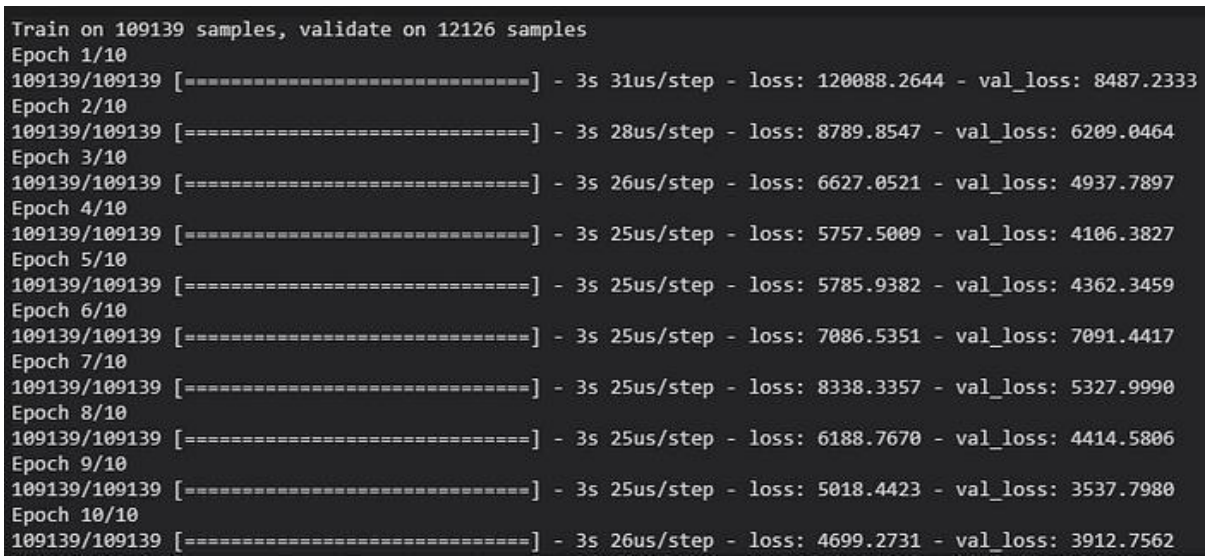
What we would like is to create two matrices:

```
X = [
[1621.0, 1536.0, 1500.0], # First three lags
[1536.0, 1500.0, 1434.0], # Second three lags
[1500.0, 1434.0, 1489.0], # Third three lags
]Y = [1434.0, 1489.0, 1620.0]
```

Fitting the model:

```
# Fitting the model
model = deep_learner.LSTModel()
```

After the above command, you can witness the fan-favorite training screen:



Training the model with more lags (hence, a larger X matrix) increases the training time:

```
deep_learner = DeepModelTS(
data = d,
Y_var = 'DAYTON_MW',
lag = 24, # 24 past hours are used
LSTM_layer_depth = 50,
epochs = 10,
batch_size = 256,
train_test_split = 0.15
)model = deep_learner.LSTModel()
```

```
Train on 109122 samples, validate on 12125 samples
Epoch 1/10
109122/109122 [==============================] - 13s 119us/step - loss: 198459.0185 - val_loss: 42519.4667
Epoch 2/10
109122/109122 [==============================] - 13s 116us/step - loss: 55858.1046 - val_loss: 48171.6780
Epoch 3/10
109122/109122 [==============================] - 13s 122us/step - loss: 56637.3061 - val_loss: 59201.6064
Epoch 4/10
109122/109122 [==============================] - 13s 123us/step - loss: 59519.4859 - val_loss: 50751.3374
Epoch 5/10
109122/109122 [==============================] - 13s 117us/step - loss: 30777.7760 - val_loss: 21393.2249
Epoch 6/10
109122/109122 [==============================] - 13s 116us/step - loss: 24113.7349 - val_loss: 19821.5885
Epoch 7/10
109122/109122 [==============================] - 13s 115us/step - loss: 21828.8772 - val_loss: 20949.0823
Epoch 8/10
109122/109122 [==============================] - 12s 114us/step - loss: 20531.1940 - val_loss: 16574.5223
Epoch 9/10
109122/109122 [==============================] - 13s 116us/step - loss: 18882.8906 - val_loss: 14898.6707
Epoch 10/10
109122/109122 [==============================] - 13s 117us/step - loss: 17565.6644 - val_loss: 15311.7685
```

Now that we have a created model we can start forecasting. The formula for the forecasts with a model trained with **p** lags:

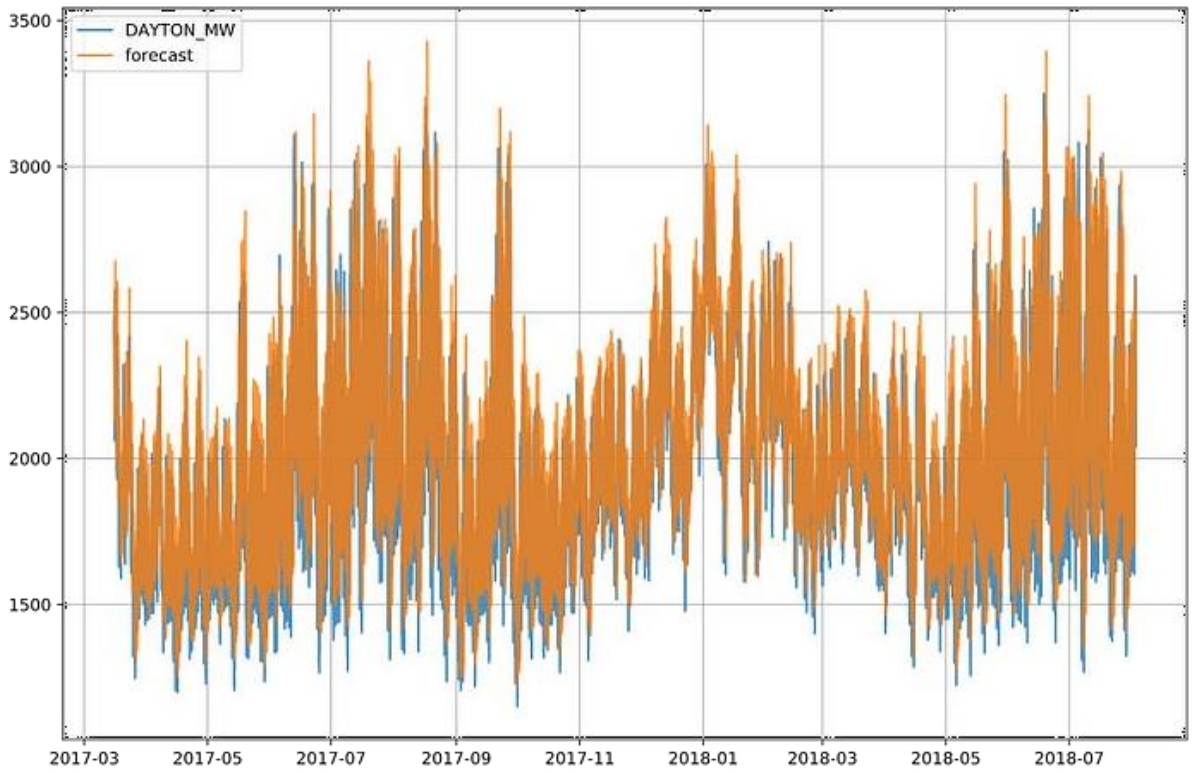
$$Y_{t+1} = f(Y_t, Y_{t-1}, \ldots, Y_{t-p})$$

```
# Defining the lag that we used for training of the model
lag_model = 24# Getting the last period
ts = d['DAYTON_MW'].tail(lag_model).values.tolist()# Creating
the X matrix for the model
X, _ = deep_learner.create_X_Y(ts, lag=lag_model)# Getting the
forecast
yhat = model.predict(X)
```

If the data was split into training and test sets then the **deep_learner.predict()** method will predict the points which are in the test set to see how our model performs out of sample.

```
yhat = deep_learner.predict()# Constructing the forecast
dataframe
fc = d.tail(len(yhat)).copy()
fc.reset_index(inplace=True)
fc['forecast'] = yhat# Ploting the forecasts
plt.figure(figsize=(12, 8))
for dtype in ['DAYTON_MW', 'forecast']:  plt.plot(
    'Datetime',
    dtype,
    data=fc,
    label=dtype,
```

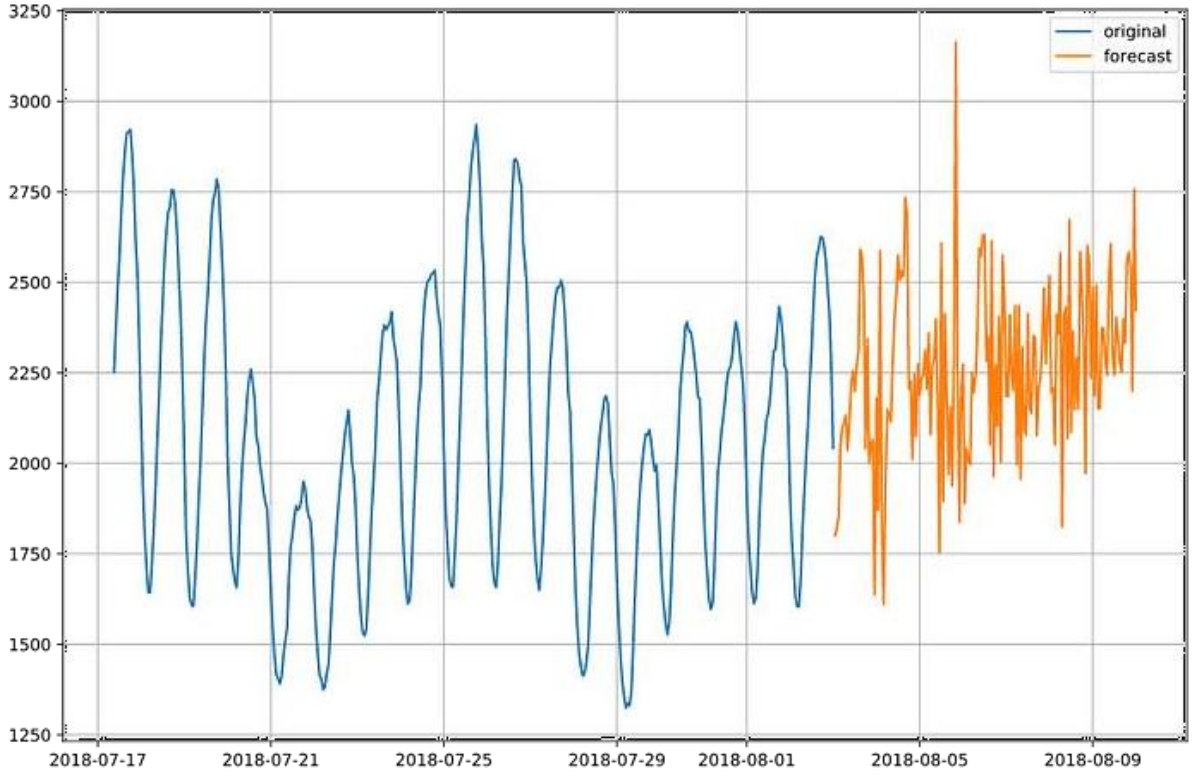
```
    alpha=0.8
  )plt.legend()
plt.grid()
plt.show()
```



We usually want to forecast ahead of the last original time series data. The class **DeepModelTS** has the method **predict_n_ahead(n_ahead)** which forecasts **n_ahead** time steps.

```
# Creating the model using full data and forecasting n steps
aheaddeep_learner = DeepModelTS(
data=d,
Y_var='DAYTON_MW',
lag=48,
LSTM_layer_depth=64,
epochs=10,
train_test_split=0
)# Fitting the model
deep_learner.LSTModel()# Forecasting n steps ahead
n_ahead = 168yhat = deep_learner.predict_n_ahead(n_ahead)
yhat = [y[0][0] for y in yhat]
```

The above code forecasts one week's worth of steps ahead (168 hours). Comparing it with the previous 400 hours:



In conclusion, this article presented a simple pipeline example when working with modeling and forecasting of the time series data:

Reading and cleaning the data (1 row for 1 time step)

Selecting the number of lags and model depth

Initiating the DeepModelTS() class

Fitting the model

Forecasting n_steps ahead

I hope that the reader can use the code showcased in this article in his/her professional and academic work.