# Week 1 Report : Data Analysis & Feature Engineering

---

This document details the complete analysis and modeling process documented in the Analysis.ipynb Jupyter Notebook. This report covers data loading, preprocessing, exploratory data analysis, feature engineering, and the development of an LSTM (Long Short-Term Memory) model for time-series forecasting.

All code and analysis described here can be found in the Analysis.ipynb Jupyter Notebook.

## 1. About the Dataset

- The foundation of this project is the Dataset.csv file, which contains hourly data on electricity prices, load, and generation capacity from Spain.

**Source :** Sourced from Kaggle

**Download Link :** Hourly Energy Demand Generation and Weather

### Dataset Column Breakdown

- The dataset provides information on electricity price, demand, and generation capacity.

#### A. Core Identifiers

- **timestamp :** The date and time of the reading (hourly frequency).

#### B. Target Variable (What We Need to Predict)

- **price_actual :** (in MWh) The actual market price of electricity.

#### C. Feature Variables (Our Predictors)

- **total_load_actual :** (in MW) The total actual electricity consumed.
- **solar_capacity :** (in MW) The installed solar generation capacity.
- **wind_capacity :** (in MW) The installed wind generation capacity.

## 2. Methodology : Analysis & Feature Engineering

- This section provides a step-by-step walkthrough of the data analysis and preparation process from the Analysis.ipynb notebook.

## Step 2.1 : Library Imports

- We begin by importing the necessary libraries for analysis, preprocessing, and modeling :
    - **pandas, numpy, matplotlib.pyplot, seaborn :** For data loading, manipulation, and visualization.
    - **sklearn.preprocessing (MinMaxScaler) :** To normalize our data for the neural network.
    - **sklearn.model_selection (train_test_split) :** To split data into training and testing sets.
    - **sklearn.metrics (mean_absolute_error, etc.) :** To evaluate model performance.
    - **tensorflow, keras (Sequential, Dense, LSTM, Dropout) :** For building and training the LSTM model.

## Step 2.2 : Data Loading and Initial Inspection

- With our libraries ready, we load and inspect the dataset.

### A. Load Data :

- We use pd.read_csv('Dataset.csv') to load our data into a DataFrame named df.

### B. Examine Data Structure (df.info()) :

- This is our first and most important diagnostic step.
- **Key Finding (Data Types) :** It reveals that the timestamp column is of type object (a string). This is a critical problem. To perform any time-series analysis, we must convert this column to a proper datetime format.
- **Finding (Null Values) :** The inspection shows no missing (non-null) data, which means we can skip data imputation steps.

### C. Statistical Summary (df.describe()) :

- This command provides a statistical overview (mean, median, min, max) of all numerical columns.
- **Purpose :** This is a "sanity check" to spot anomalies or understand the general range of our data.

## Step 2.3 : Data Preparation

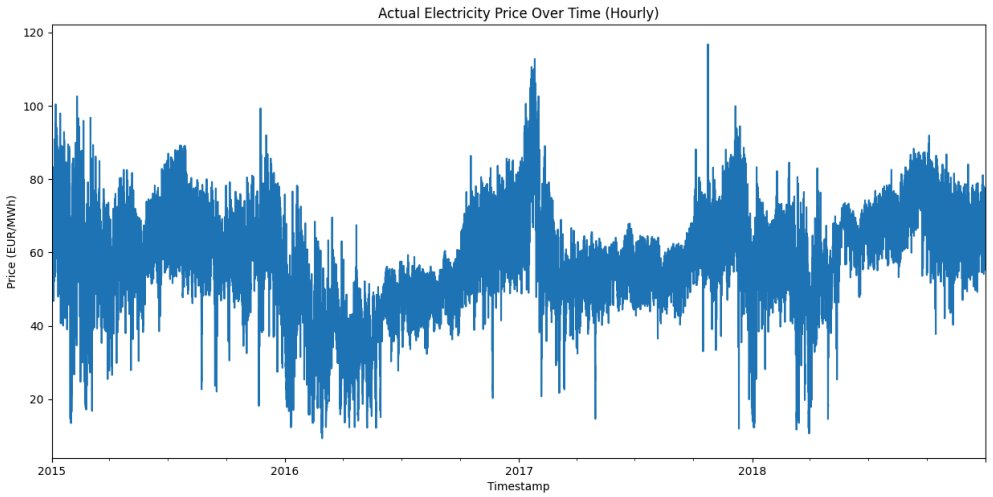
- Before analysis, the data is cleaned and prepared for time-series modeling.
- **Timestamp Conversion :** We convert the timestamp column using pd.to_datetime(df['timestamp']).
- **Set Index :** The timestamp column is set as the DataFrame's index, which is standard practice for time-series analysis.

## Step 2.4 : Exploratory Data Analysis (EDA)

- We visualize the data to discover patterns and identify relationships between variables.

### A. Target Variable Behavior Over Time :

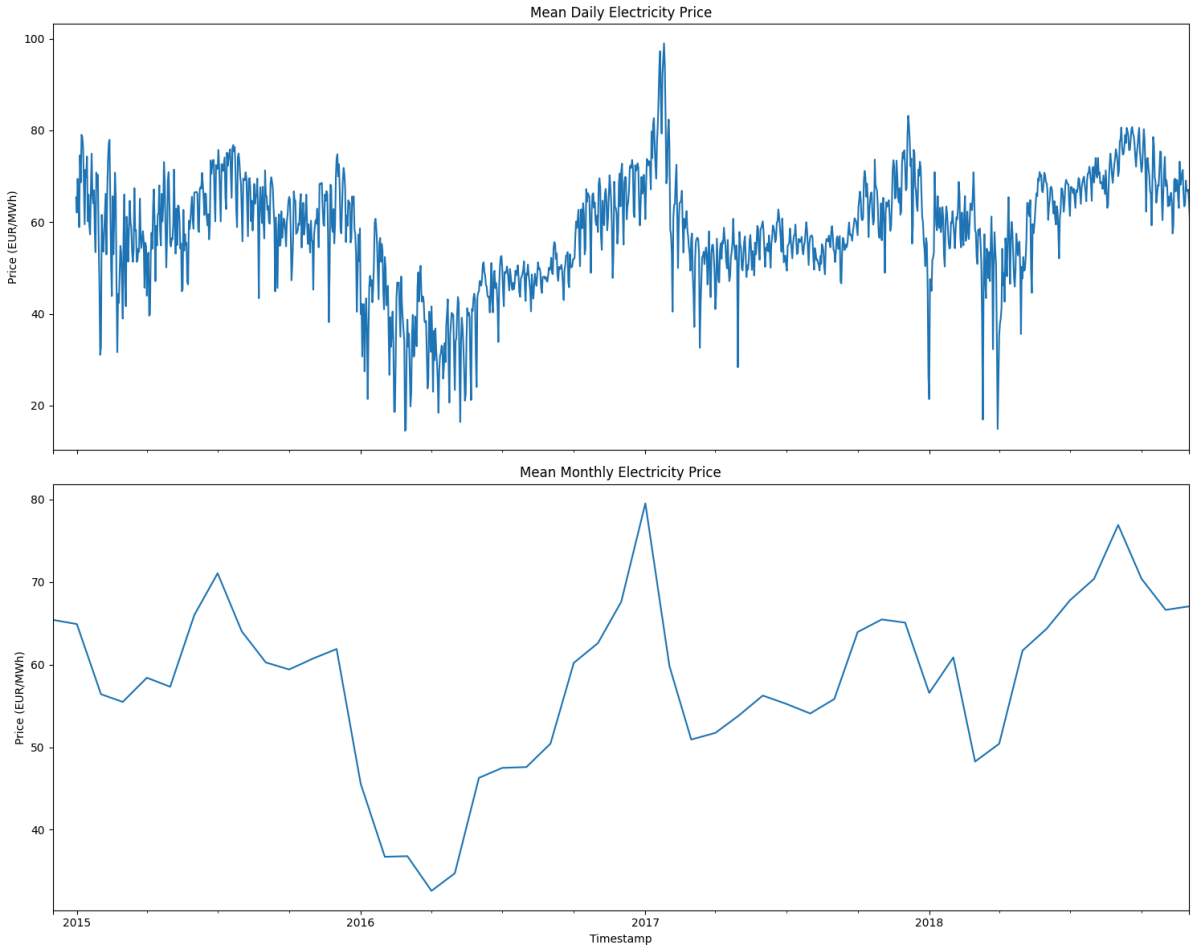
- We plot our target variable, price_actual, against time.
- **Insights :** The graph shows significant volatility. While some cyclical patterns may exist, the price is characterized by frequent, sharp spikes, indicating a complex and potentially difficult-to-model time-series.




Actual Electricity Price Over Time

### B. Cyclical Price Behavior :

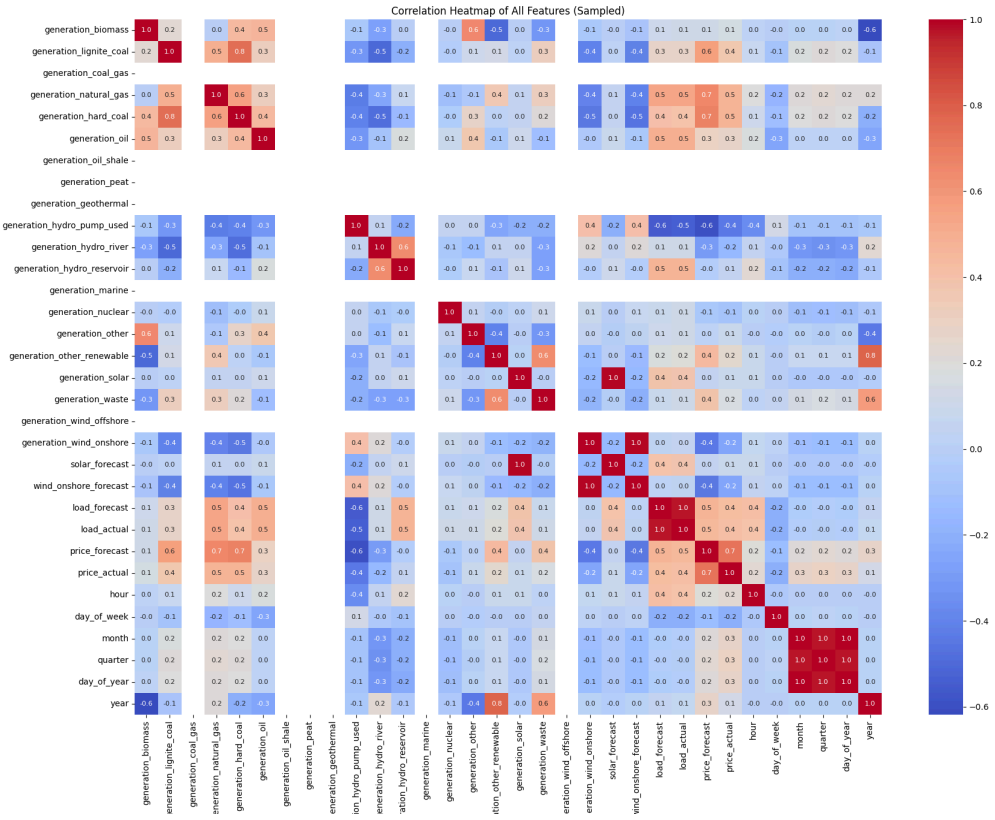
- We create boxplots to analyze the average price by the hour of the day and by the month of the year.
- **Insights :** The plots confirm clear cyclical patterns.
    - **Daily :** Prices tend to be lower during overnight hours (approx. 1-4 AM) and peak during high-demand periods in the morning (approx. 8-10 AM) and evening (approx. 6-8 PM).
    - **Monthly :** Prices show a seasonal trend, being generally higher in the winter months (like December and January) and lower in the spring.




Daily and Monthly Average Price Cycles

## C. Feature Correlation Heatmap :

- We compute the Pearson correlation matrix for all variables and visualize it as a heatmap.
- **Purpose :** This helps identify which features are most strongly related to our target, price_actual.
- **Insights :** The heatmap reveals a strong positive correlation between price_actual and total_load_actual. This confirms our hypothesis that as electricity demand (load) increases, the price also tends to increase. Other features like solar and wind capacity show a weaker correlation.




Feature Correlation Heatmap

## Step 2.5 : Feature Engineering

- We create new, informative features from the datetime index to help the model understand the cyclical patterns identified in the EDA.
- Temporal Features (Extracting Cycles) :
    - df['hour'] = df.index.hour
    - df['day_of_week'] = df.index.dayofweek
    - df['month'] = df.index.month
    - df['quarter'] = df.index.quarter
    - df['year'] = df.index.year
    - df['day_of_year'] = df.index.dayofyear
- **Why :** This explicitly provides the model with numerical features representing the daily, weekly, and seasonal cycles.

# 3. Methodology : LSTM Model Development

- This section details the construction and training of the LSTM neural network.

## Step 3.1 : Define Features (X) and Target (y)

- We separate our data into features and the target variable.

- **X (Features) :** All columns except price_actual.
- **y (Target) :** The price_actual column.

## Step 3.2 : Train-Test Split

- We split the data into training and testing sets. For time-series data, it is crucial not to shuffle the data.
- **Action :** We use train_test_split with test_size=0.2 (80% train, 20% test) and set shuffle=False to ensure the model trains in the past and is tested on the most recent data.

## Step 3.3 : Data Scaling

- Neural networks perform best when input data is normalized.
- **Action :** We use MinMaxScaler to scale all features (X_train, X_test) and the target (y_train, y_test) to a range between 0 and 1. We create two separate scalers (scaler_X and scaler_y) so we can later "inverse transform" our predicted values back to their original MWh price.

## Step 3.4 : Create Time-Series Sequences

- LSTMs require data to be structured in sequences (e.g., "use the last 24 hours of data to predict the next hour").
- **Action :** A custom function create_sequences is defined to reshape the data.
- **look_back = 24 :** We set our "look back" period to 24 hours. The model will learn by looking at a sequence of 24 data points (X) to predict the 25th data point (y).

## Step 3.5 : Building the LSTM Model

- We define the neural network architecture using Keras.
- Model Architecture :
    - **LSTM(50, activation='relu', ...) :** The first LSTM layer with 50 neurons.
    - **Dropout(0.2) :** A dropout layer to prevent overfitting by randomly dropping 20% of connections.
    - **Dense(1) :** The final output layer with a single neuron to predict the one target value (price).
- **Compilation :** We compile the model using the adam optimizer and mean_squared_error as the loss function.

## Step 3.6 : Model Training

- We fit the compiled model to our prepared training sequences.
- **Action :** The model is trained for 50 epochs with a batch size of 32. We also use a validation_split=0.1 to monitor the model's performance on unseen training data during the training process.

## Step 3.7 : Model Evaluation (Loss)

- After training, we plot the model's training loss versus its validation loss.
- **Insights :** The plot shows both training and validation loss decreasing steadily over the 50 epochs and converging. This indicates that the model learned the patterns in the data effectively without significant overfitting.

## Step 3.8 : Generating Predictions

- We use the trained model to predict prices on the unseen test set.
- **Process :**
    - The X_test_seq data is fed into model.predict().
    - The scaled predictions are then inverse-transformed using scaler_y.inverse_transform() to convert them from the 0-1 scale back to their original electricity price values.
- **Result Visualization :** A plot is generated comparing the actual prices from the test set against the predicted prices from the model. The plot shows that the model successfully captures the general trend and cyclical behavior of the price, though it struggles to predict the exact magnitude of the most extreme price spikes (as expected, given their volatility).

## Step 3.9 : Performance Metrics

- Finally, we quantify the model's accuracy using standard regression metrics.
- Results :
    - **Mean Absolute Error (MAE) :** 4.75
    - **Mean Squared Error (MSE) :** 49.34
    - **Root Mean Squared Error (RMSE) :** 7.02
- **Conclusion :** These metrics provide a baseline for the model's performance, with the RMSE indicating the typical error of the model's prediction in MWh.