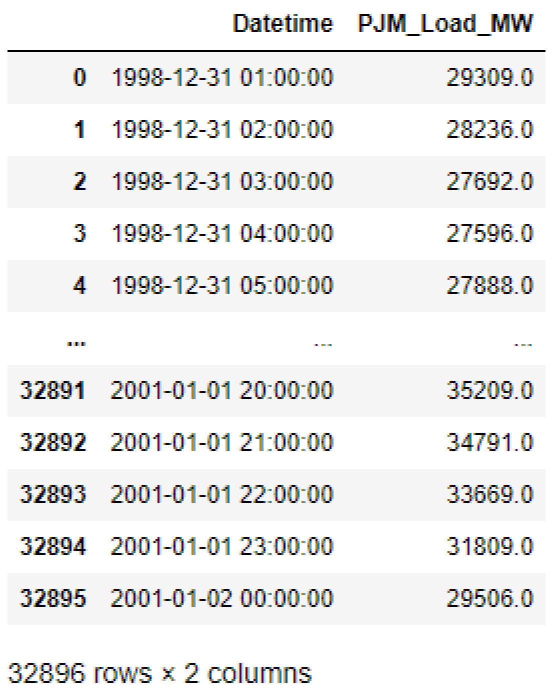
Project 4: Measure Energy Consumption

Objective:

The objective of this project is to create an automated system that measures energy consumption, analyzes the data, and provides visualizations for informed decision-making. This solution aims to enhance efficiency, accuracy, and ease of understanding in managing energy consumption across various sectors.

1.Data Source:

A good data source for measuring energy consumption in houses using artificial intelligence should be accurate and accessible.



	Datetime	PJM_Load_MW
0	1998-12-31 01:00:00	29309.0
1	1998-12-31 02:00:00	28236.0
2	1998-12-31 03:00:00	27692.0
3	1998-12-31 04:00:00	27596.0
4	1998-12-31 05:00:00	27888.0
...	...	...
32891	2001-01-01 20:00:00	35209.0
32892	2001-01-01 21:00:00	34791.0
32893	2001-01-01 22:00:00	33669.0
32894	2001-01-01 23:00:00	31809.0
32895	2001-01-02 00:00:00	29506.0

32896 rows × 2 columns

Link:(https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption/data)

2.Data Preprocessing:

1.Data Inspection:

Begin by loading your dataset and taking a close look at its structure. Use functions like head(), info(), and describe() in Python's Pandas library to get a preliminary understanding of the data.

2.Handling Missing Data:

Identify the missing rows.

Remove rows with missing values if they are relatively small in number and won't significantly impact the analysis.

Impute missing values using methods such as mean, median, mode, or more sophisticated imputation techniques, like k-nearest neighbors or regression.

3.Handling Duplicates:
Check for and remove duplicate records, if any, as they can distort your analysis.

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from pandas.plotting import lag_plot
from pylab import rcParams
from statsmodels.tsa.seasonal import seasonal_decompose
from pandas import DataFrame
from pandas import concat

df=pd.read_csv("C:\desktop\hourly-energy-consumption\AEP_hourly.csv",index_col='D
atetime',parse_dates=True)
df.head()

df.sort_values(by='Datetime', inplace=True)
print(df)

df.shape

df.info()

df.describe()

df.index = pd.to_datetime(df.index)

df["Month"] = df.index.month

df["Year"] = df.index.year

df["Date"] = df.index.date
```

```
df["Hour"] = df.index.hour
df["Week"] = df.index.week
df["Day"] = df.index.day_name()
df.head()
```

Output:

```
AEP_MW
Datetime
2004-12-31 01:00:00  13478.0
2004-12-31 02:00:00  12865.0
2004-12-31 03:00:00  12577.0
2004-12-31 04:00:00  12517.0
2004-12-31 05:00:00  12670.0

                     AEP_MW
Datetime
2004-10-01 01:00:00  12379.0
2004-10-01 02:00:00  11935.0
2004-10-01 03:00:00  11692.0
2004-10-01 04:00:00  11597.0
2004-10-01 05:00:00  11681.0
...                      ...
2018-08-02 20:00:00  17673.0
2018-08-02 21:00:00  17303.0
2018-08-02 22:00:00  17001.0
2018-08-02 23:00:00  15964.0
2018-08-03 00:00:00  14809.0

[121273 rows x 1 columns]
```

```
(121273, 1)
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 121273 entries, 2004-10-01 01:00:00 to 2018-08-03
00:00:00
Data columns (total 1 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   AEP_MW  121273 non-null  float64
dtypes: float64(1)
memory usage: 1.9 MB
```

AEP_MW	
count	121273.000000
mean	15499.513717
std	2591.399065
min	9581.000000
25%	13630.000000
50%	15310.000000
75%	17200.000000
max	25695.000000

AEP_MW	Month	Year	Date	Hour	Week	Day	
Datetime							
2004-10-01 01:00:00	12379.0	10	2004	2004-10-01	1	40	Friday
2004-10-01 02:00:00	11935.0	10	2004	2004-10-01	2	40	Friday
2004-10-01 03:00:00	11692.0	10	2004	2004-10-01	3	40	Friday
2004-10-01 04:00:00	11597.0	10	2004	2004-10-01	4	40	Friday
2004-10-01 05:00:00	11681.0	10	2004	2004-10-01	5	40	Friday

3.Feature Extraction:

1. Data Representation:

Feature extraction begins with the raw data, which can be in various formats, such as text, images, time series, or numerical values. These data types require different techniques for feature extraction.

2. Dimensionality Reduction:

One primary objective of feature extraction is to reduce the number of dimensions (features) while preserving the most relevant information. High-dimensional data can lead to increased computational complexity and overfitting.

3. Feature Engineering vs. Feature Extraction:

Feature extraction differs from feature engineering, which involves creating new features from existing ones. Feature extraction, on the other hand, involves selecting or transforming existing features.

4. Common Feature Extraction Techniques:

Depending on the type of data, various techniques can be employed for feature extraction:
a. Statistical Features:
- Calculate statistical measures such as mean, median, standard deviation, skewness, and kurtosis from numerical data.

b. Frequency Domain Analysis:
- Apply techniques like Fourier Transform to analyze the frequency components of time-series data.

c. Text Feature Extraction:
- In natural language processing (NLP), techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings (e.g., Word2Vec or GloVe) are used to extract features from text data.

d. Image Feature Extraction:
- In computer vision, methods like edge detection, color histograms, texture analysis, and deep learning-based convolutional neural networks (CNNs) are used to extract features from images.

e. Dimensionality Reduction:
- Techniques such as Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) can be used to reduce dimensionality by creating new features that capture most of the variance in the data.

4.Model Development:

Linear regression: Linear regression is a simple but effective algorithm for house priceprediction. Linear regression models the relationship between the house price and thefeatures using a linear function.

Random forest regressor: Random forest regressor is a more complex algorithm thatbuilds a multitude of decision trees to predict the house price. Random forest regressorsare typically more accurate than linear regression models, but they can be morecomputationally expensive to train.

Gradient boosting regressor: Gradient boosting regressor is another complex algorithmthat builds a sequence of decision trees to predict the house price. Gradient boostingregressors are typically more accurate than random forest regressors, but they can beeven more computationally expensive to train.

Program:

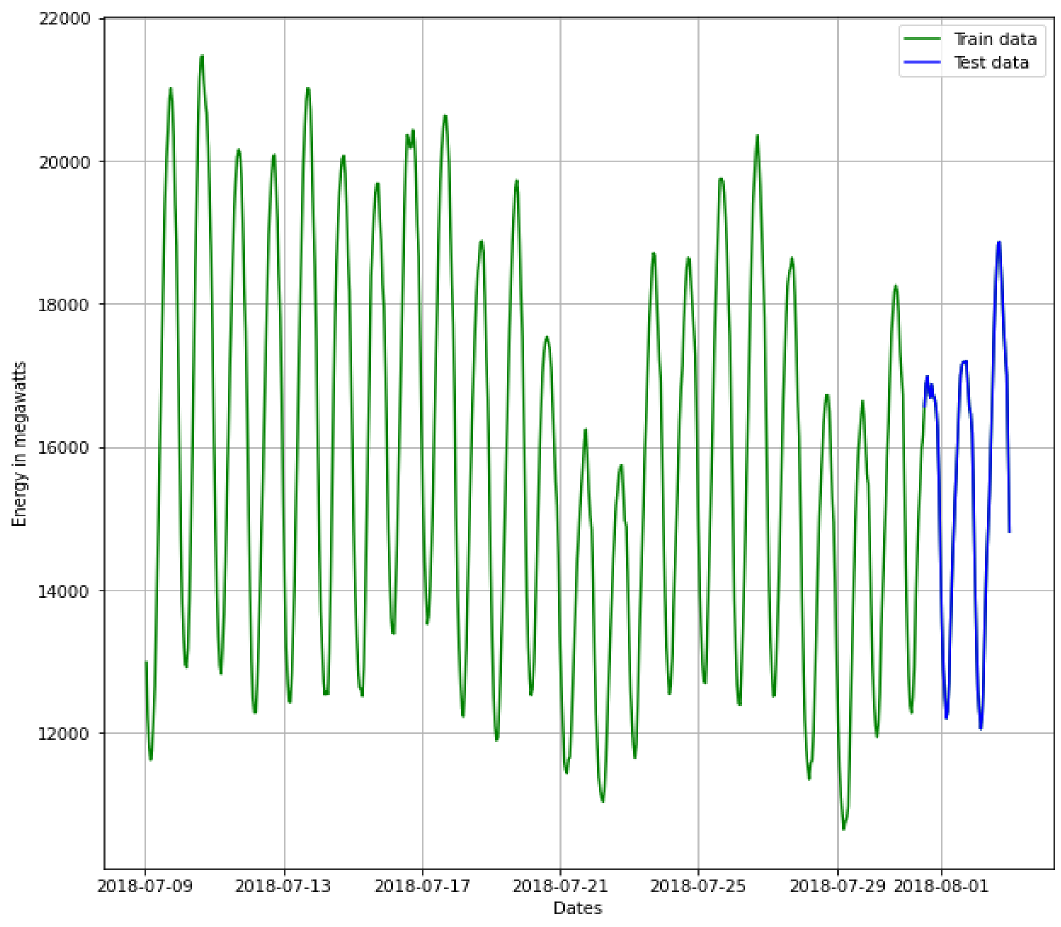
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
from math import sqrt
from sklearn.preprocessing import MinMaxScaler

# Analysis imports
from pandas.plotting import lag_plot
from pylab import rcParams
from statsmodels.tsa.seasonal import seasonal_decompose
from pandas import DataFrame
from pandas import concat

# Modelling imports
from statsmodels.tsa.ar_model import AR
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.arima_model import ARIMA
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM, GRU, RNN
from keras.layers import Dropout

values = DataFrame(df['AEP_MW'].values)
dataframe = concat([values.shift(1),
values.shift(5),values.shift(10),values.shift(30), values], axis=1)
dataframe.columns = ['t', 't+1', 't+5', 't+10', 't+30']
result = dataframe.corr()
print(result)

train_data, test_data = df[0:-60], df[-60:]
plt.figure(figsize=(10,10))
plt.grid(True)
plt.xlabel('Dates')
plt.ylabel('Energy in megawatts')
plt.plot(df['AEP_MW'].tail(600), 'green', label='Train data')
plt.plot(test_data['AEP_MW'], 'blue', label='Test data')
plt.legend()

mean_value = df['AEP_MW'].mean() # calculation of mean price
print('MSE: '+str(mean_squared_error(test_data['AEP_MW'],
np.full(len(test_data), mean_value))))
```

```
print('MAE: '+str(mean_absolute_error(test_data['AEP_MW'],
np.full(len(test_data), mean_value))))
print('RMSE: '+str(sqrt(mean_squared_error(test_data['AEP_MW'],
np.full(len(test_data), mean_value)))))
```

Output:

```
 t       t+1       t+5      t+10      t+30
t      1.000000  0.731161  0.345667  0.501972  0.976223
t+1    0.731161  1.000000  0.630009  0.847210  0.630007
t+5    0.345667  0.630009  1.000000  0.644479  0.317277
t+10   0.501972  0.847210  0.644479  1.000000  0.408315
t+30   0.976223  0.630007  0.317277  0.408315  1.000000
```

```
<matplotlib.legend.Legend at 0x78af47a94e50>
```



```
MSE: 3700885.0406027567
MAE: 1667.1805899362046
RMSE: 1923.768447761517
```

5.Visualization:

Visualization plays a crucial role in machine learning at various stages of the data analysis and model development process. Effective data visualization can help you understand your data, discover patterns, evaluate model performance, and communicate results. Here are some key aspects of visualization in machine learning:

Data Exploration:

Before diving into modeling, it's essential to explore and understand your data. Data visualization techniques like histograms, scatter plots, and box plots can help you identify data distributions, outliers, and potential relationships between variables.

Feature Analysis and Selection:

Visualization can assist in feature selection by visualizing the relationships between features and the target variable. Pair plots, correlation matrices, and feature importance plots can provide insights into which features are most informative for your model.

Data Preprocessing:

Visualization can help you make informed decisions during data preprocessing. For example, you can use missing data heatmaps to identify missing values, and you can visualize data transformations to ensure they are applied correctly.

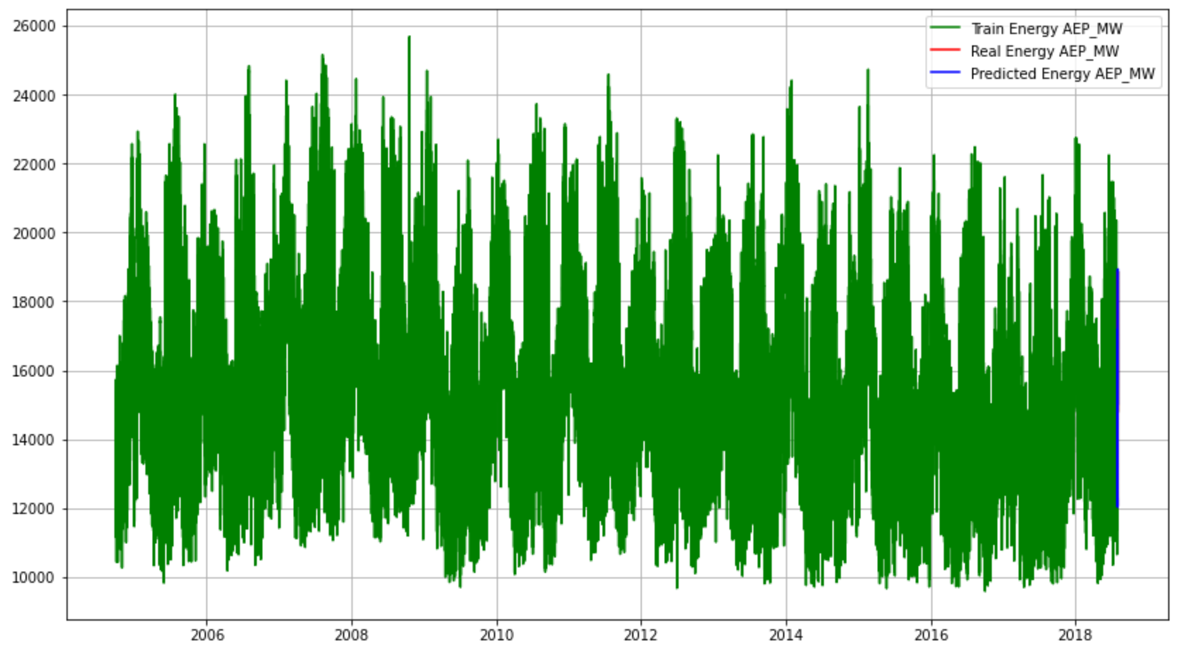
Model Understanding:

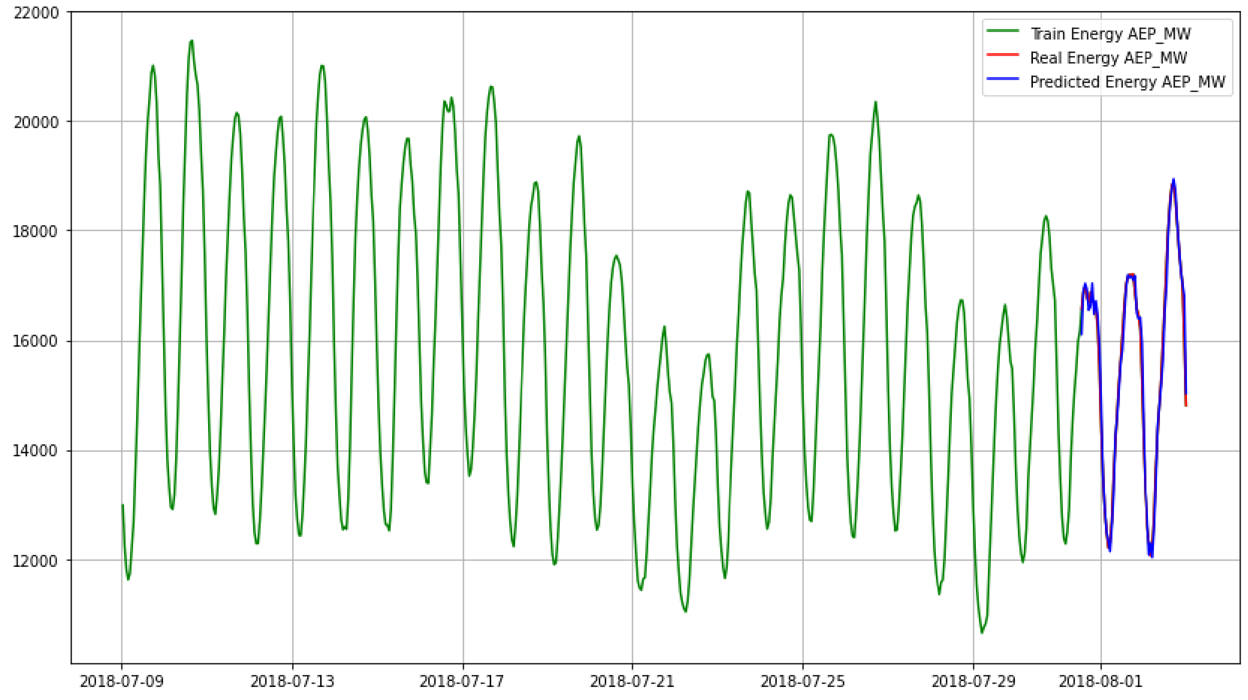
Understanding how a machine learning model works is essential for model interpretability and trust. Visualizations like decision trees, partial dependence plots, and feature importance plots can help you interpret and explain model predictions.

```
import statsmodels.api as sm
train_arima = train_data['AEP_MW']
test_arima = test_data['AEP_MW']

history = [x for x in train_arima]
y = test_arima
# make first prediction
predictions = list()
model = sm.tsa.arima.ARIMA(history, order=(5,1,0))
model_fit = model.fit()
yhat = model_fit.forecast()[0]
predictions.append(yhat)
history.append(y[0])
# rolling forecasts
for i in range(1, len(y)):
    # predict
    model = sm.tsa.arima.ARIMA(history, order=(5,1,0))
    model_fit = model.fit()
    yhat = model_fit.forecast()[0]
```

```
 predictions.append(yhat)
    # observation
    obs = y[i]
    history.append(obs)

plt.figure(figsize=(14,8))
plt.plot(df.index, df['AEP_MW'], color='green', label = 'Train Energy AEP_MW')
plt.plot(test_data.index, y, color = 'red', label = 'Real Energy AEP_MW')
plt.plot(test_data.index, predictions, color = 'blue', label = 'Predicted Energy AEP_MW')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(14,8))
plt.plot(df.index[-600:], df['AEP_MW'].tail(600), color='green', label = 'Train Energy
AEP_MW')
plt.plot(test_data.index, y, color = 'red', label = 'Real Energy AEP_MW')
plt.plot(test_data.index, predictions, color = 'blue', label = 'Predicted Energy AEP_MW')
plt.legend()
plt.grid(True)
plt.show()
```

Output:





6.Automation:

Automation in measuring energy consumption using AI involves leveraging artificial intelligence techniques to streamline and enhance the process of collecting, analyzing, and optimizing energy usage data. AI-driven automation can provide more accurate insights, predictive capabilities, and adaptive control over energy consumption. Here's how AI can be applied to automate energy measurement:

Data Collection:

Automated AI systems can collect data from various sources such as smart meters, sensors, IoT devices, and utility databases. These systems can ensure data integrity and real-time data retrieval.
Data Preprocessing:

AI can automate data preprocessing tasks such as data cleansing, outlier detection, and missing value imputation, ensuring that the collected data is clean and reliable.
Feature Extraction:

AI techniques, including deep learning, can automatically extract relevant features from raw energy consumption data, making it easier to identify patterns and anomalies.
Pattern Recognition:

AI models, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), can automate the recognition of consumption patterns, including daily and seasonal variations.

Predictive Analytics:

Machine learning models can forecast future energy consumption based on historical data, weather forecasts, and other relevant factors, enabling proactive energy management and optimization.
Optimization:

AI optimization algorithms can automatically adjust energy-consuming systems, such as HVAC, lighting, and industrial processes, to minimize energy usage while maintaining performance and comfort